SKRIPSI

PEMBUAT SOAL UNTUK PERMAINAN MENGHUBUNGKAN TITIK

Albert

NPM: 2014730007

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2018

UNDERGRADUATE THESIS

PROBLEM GENERATOR FOR CONNECTING DOTS GAME

Albert

NPM: 2014730007

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2018

LEMBAR PENGESAHAN

PEMBUAT SOAL UNTUK PERMAINAN MENGHUBUNGKAN TITIK

Albert

NPM: 2014730007

Bandung, 25 Mei 2018

Menyetujui,

Pembimbing

Joanna Helga, M.Sc.

Ketua Tim Penguji

Dr.rer.nat. Cecilia Esti Nugraheni

Anggota Tim Penguji

Pascal Alfadian, M.Comp.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PEMBUAT SOAL UNTUK PERMAINAN MENGHUBUNGKAN TITIK

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 25 Mei 2018

Albert
NPM: 2014730007

ABSTRAK

Permainan menghubungkan titik umum digunakan untuk membantu anak dalam belajar berhitung. Pada permainan menghubungkan titik terdapat bagian soal dan garis bantuan. Bagian soal terdiri atas titik dan angka. Dalam permainan ini, seorang anak ditugaskan untuk menghubungkan titik-titik yang terdapat pada bagian soal secara berurutan sehingga membentuk sebuah gambar. Soal permainan menghubungkan titik umumnya dibuat secara manual. Pada skripsi ini akan dikembangkan sebuah perangkat lunak yang dapat membuat soal permainan menghubungkan titik secara otomatis.

Masukan perangkat lunak merupakan gambar SVG. *Scalable Vector Graphics* (SVG) merupakan gambar dua dimensi berbasis vektor yang didefinisikan dalam format *Extensible Markup Language* (XML). Gambar SVG dapat dimodelkan menjadi sebuah *graph* dengan mengubah elemen-elemen yang terdapat pada dokumen SVG menjadi bagian *graph*. Sebuah *vertex* pada *graph* merepresentasikan sebuah titik pada soal permainan menghubungkan titik.

Sebuah *vertex* yang termasuk bagian soal permainan diberikan nomor urut penelusuran. Algoritma Hierholzer berfungsi untuk menentukan *euler path* pada sebuah *euler graph* terhubung. Algoritma tersebut diterapkan pada *graph* hasil pemodelan gambar SVG masukan untuk menentukan nomor urut penelusuran setiap *vertex* yang termasuk bagian soal.

Perangkat lunak telah diuji fungsionalitasnya dengan menggunakan sejumlah kasus uji yang dibuat secara manual. Soal yang dihasilkan perangkat lunak tidak selalu memiliki garis bantuan yang minimal. Pengujian dengan cara mengerjakan soal permainan yang dihasilkan perangkat lunak secara manual juga telah dilakukan.

Kata-kata kunci: Permainan Menghubungkan Titik, *Scalable Vector Graphics*, *Extensible Markup Language*, *Graph*, *Euler Graph*, Hierholzer

ABSTRACT

Connecting dots game is commonly used to help children to learn how to count. The game is divided into two parts; the problem part and help lines. The problem part consists of dots and numbers. In this game, a child has to connect the dots contained in the problem part sequentially so that the problem forms a picture. Connecting dots game problems are usually made manually. This research is conducted to develop a software capable of generating a problem for connecting dots game automatically.

The software takes an SVG image as an input. The Scalable Vector Graphics (SVG) is a vector-based two-dimensional image that is defined using Extensible Markup Language (XML). SVG image can be modelled into a graph by transforming its elements into a graph. A vertex in a graph represents a dot in the connecting dots game.

A vertex that belongs to the part of the game problem is assigned a traversal number. Hierholzer's algorithm is used to find an euler path of a connected euler graph. The algorithm applied to the graph model parsed from the SVG image to determine the traversal numbers of all vertices that belongs to the part of the game problem.

The software's functionality has been tested by a number of manually created test-cases. The problems generated by the software do not always have the minimum amount of help lines. Testing by solving the generated problem manually has also been done.

Keywords: Connecting Dots Game, Scalable Vector Graphics, Extensible Markup Language, Graph, Euler Graph, Hierholzer

Dipersembahkan untuk orang tua

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan skripsi yang berjudul "Pembuat Soal untuk Permainan Menghubungkan Titik" dengan baik. Pada masa perkuliahan dan pembuatan skripsi ini penulis mendapatkan berbagai macam bantuan baik secara langsung maupun tidak langsung dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada pihak-pihak yang telah membantu penulis, yaitu:

- Ibu Joanna Helga, M.Sc. yang telah memberikan ilmu dan bimbingan bagi penulis dari awal masa perkuliahan hingga masa penulisan skripsi ini.
- Kedua orang tua penulis yang telah membesarkan, mendidik, dan memfasilitasi penulis serta anggota keluarga penulis lainnya yang telah mendukung penulis dalam menyelesaikan skripsi ini.
- Ibu Dr.rer.nat. Cecilia Esti Nugraheni dan Bapak Pascal Alfadian, M.Comp. yang telah memberikan saran dan kritik yang membangun untuk penulisan skripsi ini.
- Ibu Mariskha Tri Adithia, P.D.Eng selaku koordinator skripsi yang telah menanamkan kedisiplinan dan komitmen sehingga skripsi ini dapat diselesaikan dengan baik dan tepat waktu.
- Bapak Husnul Hakim, M.T., Bapak Claudio Franciscus, M.T., dan seluruh dosen Informatika UNPAR yang telah memberikan ilmu dan membimbing penulis pada masa perkuliahan.
- Keenan A. Leman dan Samuel Lusandi sebagai teman berdiskusi dan bertukar pikiran semasa penulisan skripsi ini.
- Marcell Trixie A., Riki Rusli, Reza Reynaldi H., dan teman-teman penulis lainnya yang menempuh masa perkuliahan bersama-sama dengan penulis

Bandung, Mei 2018

Penulis

DAFTAR ISI

KATA PENGANTAR xv

DAFTAR ISI xvii

DAFTAR GAMBAR xix

DAFTAR TABEL xxi

1 PENDAHULUAN 1
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 2
- 1.3 Tujuan . . . 2
- 1.4 Batasan Masalah . . . 2
- 1.5 Metodologi Penelitian . . . 3
- 1.6 Sistematika Pembahasan . . . 3

2 LANDASAN TEORI 5
- 2.1 Sistem Grafik [1] . . . 5
- 2.2 Dokumen SVG [2] . . . 6
  - 2.2.1 Bentuk Dasar pada SVG . . . 7
  - 2.2.2 *Path* pada SVG . . . 9
- 2.3 Algoritma de Casteljau [3] . . . 14
  - 2.3.1 Pembentukan Kurva Bezier Kuadratik . . . 14
  - 2.3.2 Pembentukan Kurva Bezier Kubik . . . 16
- 2.4 *Graph* [4] [5] [6] . . . 16
  - 2.4.1 *Euler Graph* . . . 18
  - 2.4.2 Algoritma Hierholzer . . . 19

3 ANALISIS MASALAH 21
- 3.1 Analisis Cara Mengubah Gambar SVG Menjadi Graph . . . 21
  - 3.1.1 Analisis Cara Mengubah Bentuk Dasar pada SVG Menjadi *Graph* . . . 21
  - 3.1.2 Analisis Cara Mengubah *Path* pada SVG Menjadi *Graph* . . . 23
  - 3.1.3 Analisis Cara Mengubah Kurva Menjadi Beberapa Ruas Garis . . . 24
  - 3.1.4 Analisis Cara Mengubah Busur Elips Menjadi *Graph* [2] . . . 25
  - 3.1.5 Analisis Cara Menangani Bagian Gambar yang Terlalu Kecil . . . 27
  - 3.1.6 Analisis Cara Menangani Bagian Gambar yang Bertumpukan . . . 28
- 3.2 Analisis Cara Mengubah *Graph* Menjadi *Euler Graph* . . . 29

4 PERANCANGAN 33
- 4.1 Perancangan Input dan Output . . . 33
- 4.2 Perancangan Antarmuka . . . 34
- 4.3 Perancangan Perangkat Lunak . . . 35
  - 4.3.1 Kelas SVGPARSER . . . 36

4.3.2 Kelas GRAPHMAKER 37
4.3.3 Kelas GRAPHDRAWER 40
4.4 Perancangan Pengujian 40

5 IMPLEMENTASI DAN PENGUJIAN **43**
5.1 Implementasi 43
5.2 Pengujian 44
5.2.1 Pengujian Bentuk Dasar 45
5.2.2 Pengujian SVG *Path* 46
5.2.3 Pengujian pada Kasus Gambar yang Bertumpukan 48
5.2.4 Pengujian pada Kasus *Graph* Tak Terhubung 50
5.2.5 Pengujian Algoritma Hierholzer 51

6 KESIMPULAN DAN SARAN **55**
6.1 Kesimpulan 55
6.2 Saran 55

DAFTAR REFERENSI **57**

A KODE PROGRAM **59**

DAFTAR GAMBAR

1.1 Contoh permainan menghubungkan titik 1

2.1 Perbandingan sistem grafik 5
2.2 Perbedaan skalabilitas grafik raster dan grafik vektor 6
2.3 Contoh dokumen SVG 6
2.4 Contoh implementasi pembentukan persegi panjang pada dokumen SVG 7
2.5 Contoh implementasi pembentukan lingkaran pada dokumen SVG 7
2.6 Contoh implementasi pembentukan elips pada dokumen SVG 8
2.7 Contoh implementasi pembentukan garis pada dokumen SVG 8
2.8 Contoh implementasi pembentukan *polyline* pada dokumen SVG 8
2.9 Contoh implementasi pembentukan poligon pada dokumen SVG 9
2.10 Contoh implementasi perintah *path moveto* pada dokumen SVG 9
2.11 Contoh implementasi perintah *path lineto* pada dokumen SVG 9
2.12 Contoh implementasi perintah *path horizontal lineto* pada dokumen SVG 10
2.13 Contoh implementasi perintah *path vertical lineto* pada dokumen SVG 10
2.14 Contoh implementasi perintah *path quadratic Bezier curveto* pada dokumen SVG . 10
2.15 Contoh implementasi perintah *path shorthand quadratic Bezier curveto* 11
2.16 Contoh implementasi perintah *path cubic Bezier curveto* pada dokumen SVG . . . 11
2.17 Contoh implementasi perintah *path shorthand cubic Bezier curveto* 12
2.18 Contoh implementasi perintah *path elliptical arc* pada dokumen SVG 13
2.19 Contoh implementasi *close path* pada dokumen SVG 13
2.20 Perbandingan implementasi *path* menggunakan *absolute* dan *relative positioning* . 13
2.21 Contoh titik pada ruas garis 14
2.22 Contoh *polyline* kontrol pada kurva Bezier kuadratik 14
2.23 Contoh pembentukan kurva Bezier kuadratik 15
2.24 Contoh *polyline* kontrol pada kurva Bezier kubik 16
2.25 Contoh sebuah *graph* 17
2.26 Contoh *graph* terhubung dan *graph* tidak terhubung 17
2.27 Contoh sebuah *bridge* 18
2.28 Pemodelan permasalahan *Seven Bridge of Königsberg* sebagai sebuah *graph* 18
2.29 Perbandingan penelusuran pada *vertex* berjumlah *degree* genap dan ganjil 19
2.30 Contoh pembentukan *euler circuit* algoritma Hierholzer 20

3.1 Contoh pengubahan bentuk dasar persegi panjang menjadi *graph* 22
3.2 Contoh titik pada lingkaran 22
3.3 Contoh pengubahan bentuk dasar lingkaran menjadi *graph* 22
3.4 Contoh pengubahan bentuk dasar elips menjadi *graph* 23
3.5 Contoh perubahan kurva menjadi empat buah ruas garis 24
3.6 Contoh hasil pengubahan kurva menjadi beberapa ruas garis 25
3.7 Contoh kombinasi nilai argumen yang tidak sesuai pada pembentukan busur elips . 25
3.8 Transformasi yang dilakukan pada busur elips 26
3.9 Contoh *bounding rectangle* sebuah segitiga 28
3.10 Contoh bagian gambar yang bertumpukan 28

3.11 Contoh perpotongan pada *graph* 29
3.12 Contoh penggunaan *bitstring* untuk mengubah *graph* menjadi *euler graph* 30
3.13 Contoh solusi optimal permasalahan mengubah *graph* menjadi *euler graph* 30
3.14 Contoh kasus heuristik tidak menghasilkan solusi optimal 31

4.1 Rancangan soal yang dihasilkan perangkat lunak 33
4.2 Rancangan antarmuka halaman beranda 34
4.3 Rancangan antarmuka halaman membuat soal 34
4.4 Diagram kelas yang disederhanakan 35
4.5 Diagram urutan proses pembuatan soal menghubungkan titik 36
4.6 Kelas SVGPARSER 36
4.7 Kelas Abstrak ELEMENT 37
4.8 Kelas GRAPHMAKER 37
4.9 Kelas PATHCOMMANDGROUP 38
4.10 Kelas GRAPH dan kelas-kelas yang berhubungan dengan kelas GRAPH 38
4.11 Kelas MATRIX dan kelas MATRIXMATH 39
4.12 Kelas GRAPHDRAWER 40

5.1 Versi bahasa pemrograman Java yang digunakan pada skripsi 43
5.2 Antarmuka perangkat lunak halaman beranda 44
5.3 Antarmuka perangkat lunak halaman membuat soal 44
5.4 Hasil pengujian pada kasus pertama gambar yang bertumpukan 48
5.5 Hasil pengujian pada kasus kedua gambar yang bertumpukan 49
5.6 Hasil pengujian pada kasus ketiga gambar yang bertumpukan 49
5.7 Hasil pengujian pada kasus koordinat titik potong yang memiliki banyak angka di belakang koma 50
5.8 Hasil pengujian pada kasus *graph* tak terhubung 50

DAFTAR TABEL

5.1 Hasil pengujian bentuk dasar 45
5.2 Hasil pengujian *path* 46
5.3 Hasil pengujian algoritma Hierholzer 51
5.4 Hasil pengujian algoritma Hierholzer pada kasus tidak optimal 53

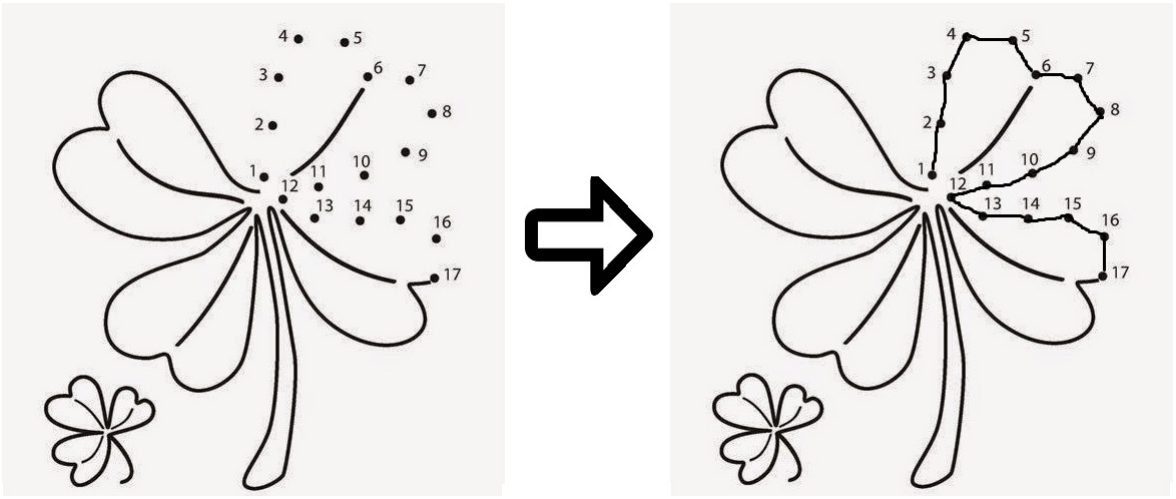
BAB 1

PENDAHULUAN

1.1 Latar Belakang

Belajar berhitung merupakan salah satu bagian dari perkembangan seorang anak. Permainan menghubungkan titik umum digunakan untuk membantu anak dalam belajar berhitung. Dalam memainkan permainan ini seorang anak harus menghubungkan titik-titik yang terdapat pada soal sehingga membentuk sebuah gambar. Seluruh titik yang terdapat pada soal memiliki angka sehingga dalam menghubungkan dua buah titik seorang anak harus memperhatikan keterurutan angka.

Permainan menghubungkan titik umumnya dibuat secara manual. Pembuat soal awalnya membuat sebuah gambar kemudian pembuat soal menentukan titik-titik pembentuk gambar tersebut. Titik-titik tersebut kemudian diberi angka secara berurutan dan setiap garis yang menghubungkan dua buah titik dihapus sehingga terbentuk soal permainan. Dalam soal biasanya terdapat bagian gambar yang tidak dihapus guna membantu pemain dalam memperkirakan bentuk gambar. Namun, garis bantuan yang terlalu banyak membuat soal permainan terlalu mudah.



Gambar 1.1: Contoh permainan menghubungkan titik

Pada skripsi ini, dikembangkan sebuah perangkat lunak yang dapat membuat soal permainan menghubungkan titik secara otomatis. Pembuat soal hanya perlu memasukkan gambar sebagai masukan perangkat lunak kemudian perangkat lunak akan menghasilkan keluaran berupa soal permainan menghubungkan titik. Dengan adanya perangkat lunak ini, pembuat soal tidak perlu menentukan titik pembentuk gambar, menghapus garis antar titik, dan memberi angka pada setiap titik secara manual.

Soal permainan yang dihasilkan perangkat lunak ini diharapkan memiliki garis bantuan yang minimal sehingga permainan menjadi lebih menarik. Soal permainan juga dibuat agar pemain dapat menghubungkan seluruh titik tanpa harus mengangkat pensil, tetapi tidak semua gambar dapat digambar tanpa mengangkat pensil. *Euler path* merupakan sebuah *path* pada *graph* yang

melewati seluruh *edge* pada *graph* tepat satu kali. Sebuah gambar dapat direpresentasikan sebagai sebuah *graph* sehingga perangkat lunak akan mencari *euler path* optimal pada gambar masukan dan setiap *edge* yang tidak termasuk dalam *path* tersebut akan menjadi garis bantuan.

Scalable Vector Graphics (SVG) merupakan sebuah gambar dua dimensi berbasis vektor yang didefinisikan dalam format *Extensible Markup Language* (XML). SVG tidak mengalami penurunan kualitas gambar ketika diperbesar sehingga SVG umum digunakan. Perangkat lunak yang dikembangkan pada skripsi ini menerima masukan berupa SVG sehingga perangkat lunak dapat memroses SVG dalam format XML.

Gambar masukan perangkat lunak ubah menjadi sebuah *graph*, dalam mengubah gambar menjadi *graph* terdapat beberapa hal yang perlu ditangani. Sebuah gambar dapat memuat beberapa bangun datar sehingga dua buah bangun datar mungkin bertumpukan dalam suatu gambar. Agar soal yang dihasilkan dapat digambar tanpa mengangkat pensil, titk potong dua buah bangun datar yang bertumpukan harus menjadi *vertex* pada *graph*. Pada permainan ini setiap titik dihubungkan oleh garis lurus sehingga setiap kurva yang terdapat pada gambar juga harus diubah menjadi beberapa ruas garis.

1.2 Rumusan Masalah

Berdasarkan deskripsi pada bagian sebelumnya, berikut adalah rumusah masalah yang ada:

1. Bagaimana format penyimpanan SVG dalam XML?
2. Bagaimana algoritma untuk mengubah SVG menjadi sebuah *graph*?
3. Bagaimana algoritma untuk mencari *euler path* yang optimal pada sebuah *graph* untuk membuat soal permainan menghubungkan titik yang memiliki garis bantuan minimal?
4. Bagaimana cara mengimplementasikan kedua algoritma pada poin sebelumnya dalam perangkat lunak?

1.3 Tujuan

Tujuan dari skripsi ini adalah sebagai berikut:

1. Mengetahui format penyimpanan SVG dalam XML.
2. Mengembangkan algoritma untuk mengubah SVG menjadi sebuah *graph*.
3. Mempelajari algoritma untuk mencari *euler path* yang optimal pada sebuah *graph* untuk membuat soal permainan menghubungkan titik yang memiliki garis bantuan minimal.
4. Mengimplementasikan kedua algoritma pada poin sebelumnya dalam perangkat lunak.

1.4 Batasan Masalah

Batasan masalah skripsi ini adalah sebagai berikut:

1. Gambar masukan perangkat lunak merupakan gambar SVG.
2. Gambar masukan perangkat lunak merupakan gambar yang dapat diubah menjadi sebuah *graph* terhubung. Pada sebuah *graph* terhubung, terdapat minimal sebuah *path* yang menghubungkan setiap pasang *vertex* yang ada pada *graph* tersebut.
3. Elemen SVG yang ditangani pada skripsi ini adalah elemen RECTANGLE, CIRCLE, ELLIPSE, LINE, POLYLINE, POLYGON, dan PATH.

1.5 Metodologi Penelitian

Metodologi penelitian yang digunakan pada skripsi ini adalah sebagai berikut:

1. Mempelajari format SVG dalam XML.
2. Mengembangkan algoritma untuk mengubah SVG menjadi *graph*.
3. Melakukan studi literatur tentang karakteristik *euler graph*.
4. Melakukan studi literatur tentang algoritma pencarian *euler path* optimal pada sebuah *graph*.
5. Merancang perangkat lunak serta mengimplementasikan algoritma untuk mengubah SVG menjadi sebuah *graph* dan algoritma pencarian *euler path* yang optimal untuk membuat soal permainan menghubungkan titik dengan garis bantuan yang minimal pada perangkat lunak tersebut.
6. Menguji fungsionalitas perangkat lunak pada berbagai kasus.
7. Menulis dokumen skripsi.

1.6 Sistematika Pembahasan

Dokumen skripsi ini terdiri atas enam bab dengan sistematika pembahasan sebagai berikut:

1. Bab 1 Pendahuluan
   Bab 1 membahas latar belakang dibuatnya perangkat lunak pembuat soal permainan menghubungkan titik. Pada bab ini dibahas juga mengenai rumusan masalah, tujuan skripsi, batasan masalah, dan metodologi penelitian yang digunakan pada skripsi.
2. Bab 2 Landasan Teori
   Bab 2 memuat teori yang mendasari skripsi ini. Bab ini akan berisi pembahasan mengenai sistem grafik, dokumen SVG, algoritma de Casteljau yang berperan dalam pembentukan kurva Bezier, dan algoritma Hierholzer yang berperan dalam pencarian *euler path* optimal. Penjelasan mengenai *graph* dan *euler graph* juga terdapat pada bab ini untuk mendasari pembahasan algoritma Hierholzer dan *euler path*.
3. Bab 3 Analisis Masalah
   Bab 3 berisi analisis masalah tentang cara mengubah gambar SVG menjadi *graph*, analisis masalah-masalah yang terdapat pada proses pengubahan gambar menjadi *graph*, serta analisis cara mengubah *graph* menjadi *euler graph* yang merupakan kebutuhan algoritma Hierholzer.
4. Bab 4 Perancangan
   Bab 4 membahas perancangan input dan output perangkat lunak, perancangan antarmuka perangkat lunak, perancangan perangkat lunak, dan perancangan pengujian perangkat lunak. Pada bagian perancangan perangkat lunak akan ditampilkan diagram kelas yang menunjukkan struktur perangkat lunak dan diagram urutan yang memodelkan cara kerja perangkat lunak.
5. Bab 5 Implementasi dan Pengujian
   Bab 5 pada skripsi ini membahas implementasi perangkat lunak dan pengujian yang dilakukan terhadap perangkat lunak tersebut. Bagian ini menjelaskan tentang spesifikasi perangkat lunak, implementasi antarmuka, dan pengujian yang dilakukan pada skripsi ini.
6. Bab 6 Kesimpulan dan Saran
   Bab 6 berisi kesimpulan dan saran dari skripsi ini.

BAB 2

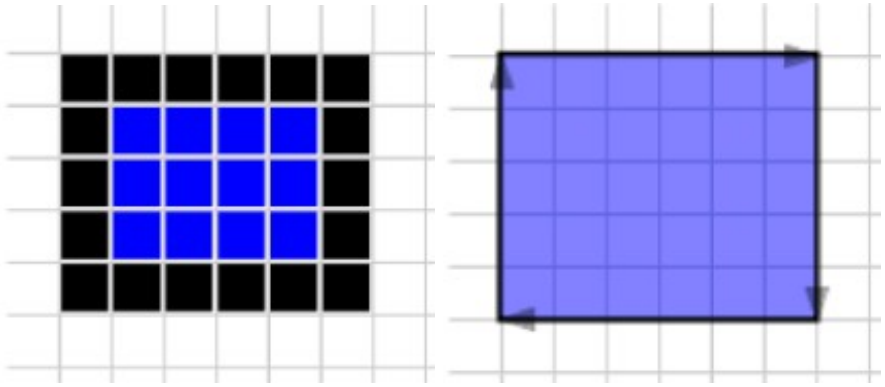
LANDASAN TEORI

Bab ini berisi landasan untuk memahami pembahasan pada skripsi ini. Skripsi ini berfokus pada proses pengubahan gambar bertipe SVG menjadi soal permainan menghubungkan titik. Sebuah gambar SVG terdiri dari berbagai bentuk termasuk kurva Bezier sehingga pada bagian ini dijelaskan mengenai algoritma de Casteljau yang berperan dalam pembentukan kurva Bezier. Bab ini juga menjelaskan SVG yang merupakan masukan perangkat lunak serta algoritma Hierholzer yang digunakan untuk mencari *euler path* optimal pada soal permainan menghubungkan titik. Penjelasan mengenai *graph* khususnya *euler graph* juga dibahas pada bab ini sebagai pengantar dalam membahas *euler path*.

2.1 Sistem Grafik [1]

Sistem grafik merupakan sebuah sistem yang mendefinisikan cara untuk merepresentasikan informasi grafik pada komputer. Terdapat dua buah sistem grafik utama, yaitu, *raster* dan *vector graphics*. Kedua sistem grafik tersebut memiliki cara yang berbeda dalam menampilkan informasi grafik sehingga kedua sistem grafik tersebut memiliki kegunaan serta keunggulannya masing-masing.

Grafik raster merepresentasikan gambar sebagai susunan *picture elements* (*pixels*) dan setiap *pixel* tersebut menyimpan nilai RGB yang merepresentasikan warna pada *pixel* tersebut seperti pada Gambar 2.1a. Berbeda dengan grafik raster, grafik vektor merepresentasikan sebuah gambar sebagai serangkaian bentuk geometris seperti titik, garis, dan kurva. Pada Gambar 2.1b, grafik vektor terbentuk dari serangkaian garis.

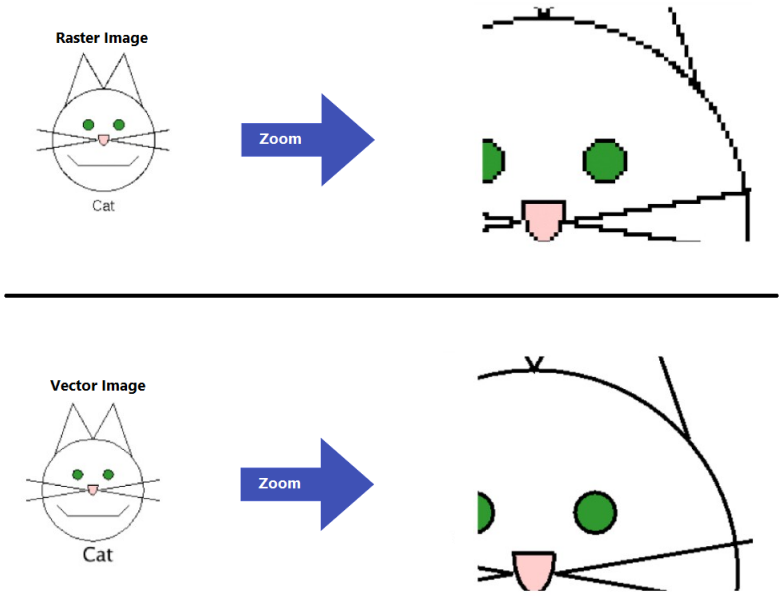


(a) Sistem grafik raster (b) Sistem grafik vektor

Gambar 2.1: Perbandingan sistem grafik

Keunggulan utama grafik vektor dibandingkan dengan grafik raster adalah skalabilitas grafik vektor yang ditunjukkan pada Gambar 2.2. Gambar yang direpresentasikan dengan grafik raster terlihat semakin tidak mulus ketika dilakukan *zoom* terhadap gambar tersebut. Hal tersebut dikarenakan *zoom* yang dilakukan pada gambar yang direpresentasikan dengan grafik raster memperbesar seluruh *pixel* pada gambar. Berbeda dengan grafik raster, grafik vektor melakukan *zoom* pada

sebuah gambar dengan cara menghitung ulang koordinat yang ada kemudian menggambar ulang gambar tersebut dengan resolusi penuh sehingga gambar tetap terlihat mulus.



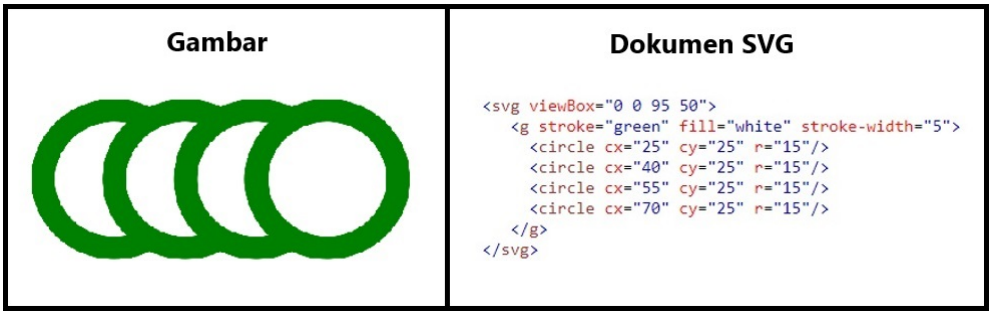
Gambar 2.2: Perbedaan skalabilitas grafik raster dan grafik vektor

2.2 Dokumen SVG [2]

Scalable Vector Graphics (SVG) merupakan aplikasi dari *Extensible Markup Language* (XML). SVG merepresentasikan informasi grafik dalam bentuk vektor. Dewasa ini SVG sangat umum digunakan, hampir seluruh *web browser* modern dapat menampilkan SVG.

Dokumen SVG diawali dengan *tag* pembuka <svg> dan diakhiri dengan *tag* penutup </svg>. Sebuah dokumen SVG dapat tidak berisi elemen apapun, hanya berisi satu elemen, hingga berisi banyak *container elements* dan *graphic elements* seperti pada Gambar 2.3. Setiap elemen, termasuk elemen SVG memiliki atribut-atribut yang mendefinisikan elemen tersebut. Atribut-atribut tersebut tentunya memiliki nilai yang berpengaruh terhadap elemen tersebut.

Pada elemen SVG nilai atribut *width* dan *height* dapat ditentukan untuk mendefinisikan luas bidang yang digunakan untuk menampilkan gambar SVG. Lokasi penampilan gambar SVG juga dapat ditentukan dengan mendefinisikan nilai atribut x dan y yang merepresentasikan koordinat titik kiri atas dari bidang yang menampilkan SVG tersebut. Jika atribut x dan y tidak didefinisikan, nilai *default* atribut tersebut adalah nol.



Gambar 2.3: Contoh dokumen SVG

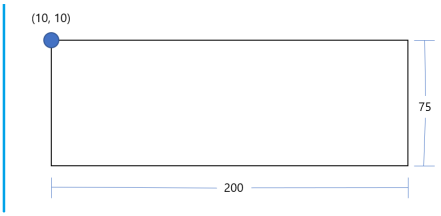
Elemen g merupakan *container element* yang berfungsi untuk mengelompokkan *graphic element* yang berhubungan. Dengan dikelompokkannya *graphic element* yang berhubungan, dokumen SVG menjadi lebih terstruktur serta dapat meningkatkan *reusability* dokumen. Elemen g dapat juga memuat elemen g di dalamnya.

2.2.1 Bentuk Dasar pada SVG

Terdapat enam buah bentuk dasar yang dapat digunakan untuk mendefinisikan gambar SVG pada dokumen SVG. Keenam bentuk dasar tersebut memiliki atribut-atribut berbeda yang berguna untuk mendefinisikan bentuk serta ukuran. Enam bentuk dasar tersebut adalah:

1. Persegi panjang
   Persegi panjang memiliki atribut x dan y. Atribut x dan y merepresentasikan koordinat titik kiri atas dari persegi panjang tersebut. Selain atribut x dan y, atribut *width* dan *height* juga mendefinisikan panjang dan lebar dari persegi panjang. Pada Gambar 2.4 sebuah persegi panjang memiliki lebar 200 *pixel* dan tinggi 75 *pixel*. Persegi panjang tersebut terletak pada koordinat (10, 10).

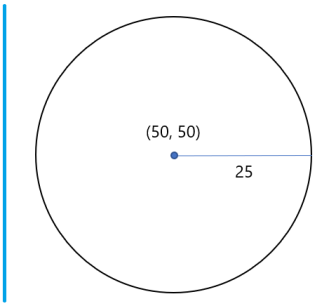
```
<svg height="1000" width="1000">
  <rect x="10" y="10" width="200" height="75" stroke="black" />
</svg>
```



Gambar 2.4: Contoh implementasi pembentukan persegi panjang pada dokumen SVG

2. Lingkaran
   Sebuah lingkaran didefinisikan dengan koordinat titik pusat dan radius. Atribut cx dan cy mendefinisikan koordinat titik pusat lingkaran dan atribut r mendefinisikan radius lingkaran. Sebuah lingkaran yang memiliki koordinat titik pusat (50, 50) dan radius sebesar 25 *pixel* diimplementasikan seperti pada Gambar 2.5.

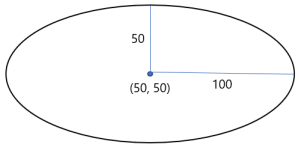
```
<svg height="1000" width="1000">
  <circle cx="50" cy="50" r="25" stroke="black" />
</svg>
```



Gambar 2.5: Contoh implementasi pembentukan lingkaran pada dokumen SVG

3. Elips
   Sama seperti lingkaran, sebuah elips juga didefinisikan dengan koordinat titik pusat dan radius. Pada elips terdapat dua buah atribut radius, yaitu, rx dan ry. Koordinat pusat elips didefinisikan oleh atribut cx dan cy. Gambar 2.6 merupakan contoh implementasi elips pada dokumen SVG. Elips tersebut bertitik pusat pada koordinat (50, 50) serta memiliki rx sebesar 100 *pixel* dan ry sebesar 50 *pixel*.

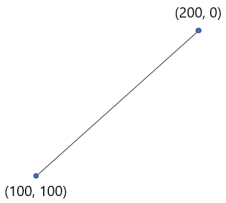
```
<svg height="1000" width="1000">
  <ellipse cx="50" cy="50" rx="100" ry="50" stroke="black" />
</svg>
```



Gambar 2.6: Contoh implementasi pembentukan elips pada dokumen SVG

4. Garis
   Sebuah garis didefinisikan oleh titik awal dan titik akhir. Pada dokumen SVG, sebuah garis dapat didefinisikan dengan menentukan atribut x1 dan y1 sebagai koordinat titik awal juga x2 dan y2 yang merepresentasikan koordinat titik akhir. Garis pada Gambar 2.7 memiliki koordinat titik awal (100, 100) dan koordinat titik akhir (200, 0).

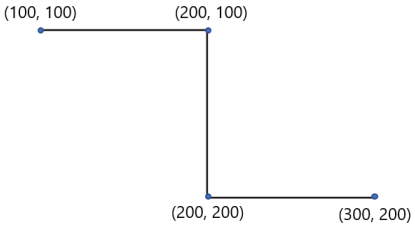
```
<svg height="1000" width="1000">
  <line x1="100" y1="100" x2="200" y2="0" stroke="black" />
</svg>
```



Gambar 2.7: Contoh implementasi pembentukan garis pada dokumen SVG

5. *Polyline*
   Polyline merupakan sekumpulan ruas garis yang terhubung. Bentuk *polyline* didefinisikan dengan menentukan koordinat titik-titik yang menghubungkan ruas-ruas garis pada *polyline*. Pada Gambar 2.8 *polyline* memiliki empat buah titik pada koordinat (100, 100), (200, 100), (200, 200), dan (300, 200) yang menghubungkan ruas-ruas garis pada *polyline* tersebut.

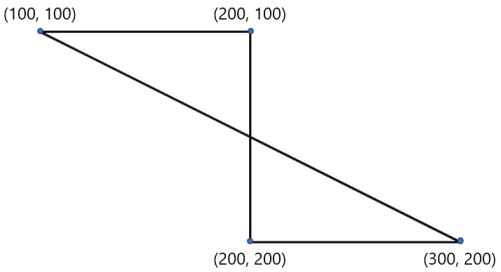
```
<svg height="1000" width="1000">

  <polyline points="100, 100
                    200, 100
                    200, 200
                    300, 200"
            stroke="black"
            fill="transparent" />

</svg>
```



Gambar 2.8: Contoh implementasi pembentukan *polyline* pada dokumen SVG

6. Poligon
   Sama seperti *polyline*, poligon didefinisikan dengan menentukan koordinat titik-titik yang menghubungkan ruas-ruas garis pada poligon. Namun, pada poligon terdapat ruas garis yang menghubungkan titik akhir dengan titik awal. Gambar 2.9 merupakan contoh implementasi poligon yang memiliki empat buah titik pada koordinat (100, 100), (200, 100), (200, 200), dan (300, 200).

```
<svg height="1000" width="1000">

  <polygon points="100, 100
                   200, 100
                   200, 200
                   300, 200"
           stroke="black"
           fill="transparent" />

</svg>
```



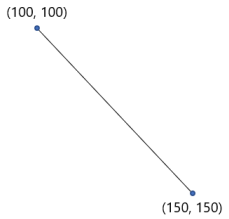
Gambar 2.9: Contoh implementasi pembentukan poligon pada dokumen SVG

2.2.2 *Path* pada SVG

Cara lain untuk mendefinisikan bentuk pada gambar SVG adalah menggunakan *path*. Sebuah *path* merepresentasikan garis besar dari sebuah bentuk. Analogi menggambar pada kertas dapat dihubungkan dengan mengimplementasikan *path* pada dokumen SVG. Sebuah *path* dapat didefinisikan sebagai perpindahan pena dari sebuah titik ke titik lain. Terdapat sepuluh perintah yang dapat digunakan untuk membentuk sebuah *path* pada dokumen SVG, yaitu:

1. M (*moveto*)
   Perintah *moveto* digunakan untuk mengawali pendefinisian sebuah *path*. Perintah ini membutuhkan sebuah koordinat yang dijadikan sebagai koordinat awal pembentukan *path*. Jika didefinisikan dua buah koordinat sebagai parameter, dua buah koordinat tersebut dijadikan titik awal dan akhir sebuah garis yang mengawali pembentukan *path* seperti pada Gambar 2.10.

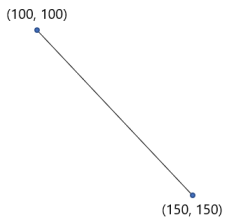
```
<svg height="210" width="400">
  <path d="M150 150 100 100" stroke="black" />
</svg>
```



Gambar 2.10: Contoh implementasi perintah *path moveto* pada dokumen SVG

2. L (*lineto*)
   Perintah *lineto* membutuhkan sebuah koordinat yang merepresentasikan titik akhir dari garis yang akan dibentuk. Koordinat titik awal garis merupakan koordinat titik terakhir pada perintah sebelumnya. Pada Gambar 2.11 sebuah ruas garis terbentuk dengan koordinat titik awal (150, 150) dan koordinat titik akhir (100, 100).

```
<svg height="210" width="400">
  <path d="M150 150 L100 100" stroke="black" />
</svg>
```



Gambar 2.11: Contoh implementasi perintah *path lineto* pada dokumen SVG

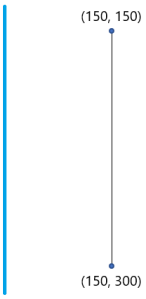
3. H (*horizontal lineto*)
   Perintah *horizontal lineto* merupakan sebuah perintah yang digunakan untuk membuat garis horizontal. Perintah tersebut didefinisikan oleh sebuah parameter yaitu koordinat sumbu x yang merepresentasikan akhir dari pembentukan garis horizontal tersebut. Pada Gambar 2.12 sebuah garis horizontal dengan panjang 150 *pixel* terbentuk dari koordinat (150, 150) hingga (300, 150).

```
<svg height="1000" width="1000">
  <path d="M150 150 H300" stroke="black" />
</svg>
```


Gambar 2.12: Contoh implementasi perintah *path horizontal lineto* pada dokumen SVG

4. V (*vertical lineto*)
   Perintah *vertical lineto* merupakan sebuah perintah yang digunakan untuk membuat garis vertikal. Perintah tersebut didefinisikan oleh sebuah parameter yaitu koordinat sumbu y yang merepresentasikan akhir dari pembentukan garis vertikal tersebut. Pada Gambar 2.13 sebuah garis vertikal dengan panjang 150 *pixel* terbentuk dari koordinat (150, 150) hingga (150, 300).

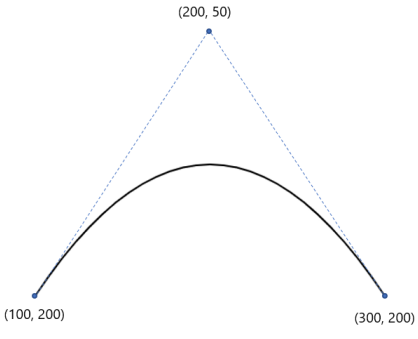
```
<svg height="1000" width="1000">
  <path d="M150 150 V300" stroke="black" />
</svg>
```



Gambar 2.13: Contoh implementasi perintah *path vertical lineto* pada dokumen SVG

5. Q (*quadratic Bezier curveto*)
   Perintah *quadratic Bezier curveto* merupakan perintah untuk membangun kurva Bezier kuadratik. Kurva Bezier kuadratik memiliki tiga buah titik kontrol. Dua pasang parameter diperlukan untuk membangun kurva bezier kuadratik. Dua pasang parameter tersebut merupakan koordinat dua titik kontrol terakhir dari kurva tersebut. Koordinat titik kontrol pertama merupakan koordinat terakhir pada perintah sebelumnya. Pada Gambar 2.14 sebuah kurva Bezier kuadratik memiliki tiga buah titik kontrol. Titik kontrol pertama terletak pada koordinat (100, 200), titik kontrol kedua terletak pada koordinat (200, 50), dan titik kontrol ketiga terletak pada koordinat (300, 200).

```
<svg height="210" width="400">

  <path d="M100 200 Q200 50 300 200"
        stroke="black"
        fill="transparent" />

</svg>
```

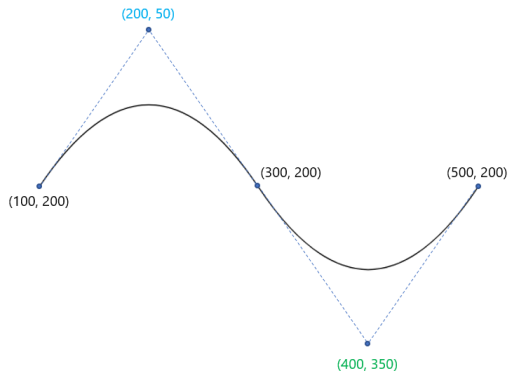


Gambar 2.14: Contoh implementasi perintah *path quadratic Bezier curveto* pada dokumen SVG

6. T (*shorthand/smooth quadratic Bezier curveto*)
   Perintah *shorthand/smooth quadratic Bezier curveto* merupakan penulisan singkat untuk membuat kurva Bezier kuadratik dengan memperhatikan perintah yang ada sebelumnya. Titik kontrol kedua dari kurva Bezier kuadratik yang akan dibangun merupakan pencerminan titik kontrol kedua dari perintah pembentukan kurva Bezier kuadratik sebelumnya. Jika perintah sebelumnya bukan merupakan pembentukan kurva Bezier kuadratik (perintah sebelumnya bukan Q atau T), sebuah garis lurus akan terbentuk.

   Pada Gambar 2.15 kurva Bezier kuadratik yang terbentuk dari penulisan singkat yang dilakukan memiliki tiga buah titik kontrol. Titik kontrol pertama terletak pada koordinat (300, 200), titik kontrol kedua terletak pada koordinat (400, 350), dan titik kontrol ketiga terletak pada koordinat (500, 200). Koordinat titik kontrol pertama merupakan koordinat terakhir perintah sebelumnya sedangkan koordinat titik kontrol kedua diperoleh dengan mencerminkan titik kontrol kedua kurva Bezier sebelumnya terhadap koordinat titik kontrol pertama kurva Bezier kuadratik yang akan dibangun.

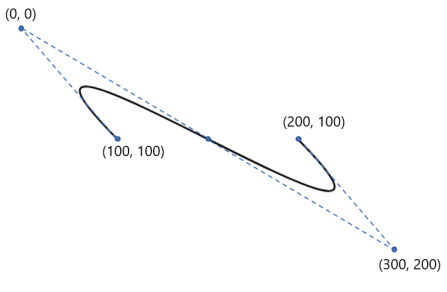
```
<svg height="1000" width="1000">

  <path d="M100 200
           Q200 50 300 200
           T500 200"
        stroke="black"
        fill="transparent" />

</svg>
```



Gambar 2.15: Contoh implementasi perintah *path shorthand quadratic Bezier curveto*

7. C (*cubic Bezier curveto*)
   Perintah *cubic Bezier curveto* merupakan perintah untuk membangun kurva Bezier kubik. Kurva Bezier kubik memiliki empat buah titik kontrol. Pembentukan kurva Bezier kubik membutuhkan tiga pasang parameter yang merupakan koordinat tiga titik kontrol terakhir dari kurva tersebut. Koordinat titik kontrol pertama merupakan koordinat terakhir perintah sebelumnya. Pada Gambar 2.16 kurva Bezier kubik memiliki empat buah titik kontrol. Titik kontrol pertama terletak pada koordinat (100, 100), titik kontrol kedua terletak pada koordinat (0, 0), titik kontrol ketiga terletak pada koordinat (300, 200), dan titik kontrol terakhir terletak pada koordinat (200, 100).

```
<svg height="1000" width="1000">

  <path d="M100 100
           C0 0 300 200 200 100"
        stroke="black"
        fill="transparent" />

</svg>
```

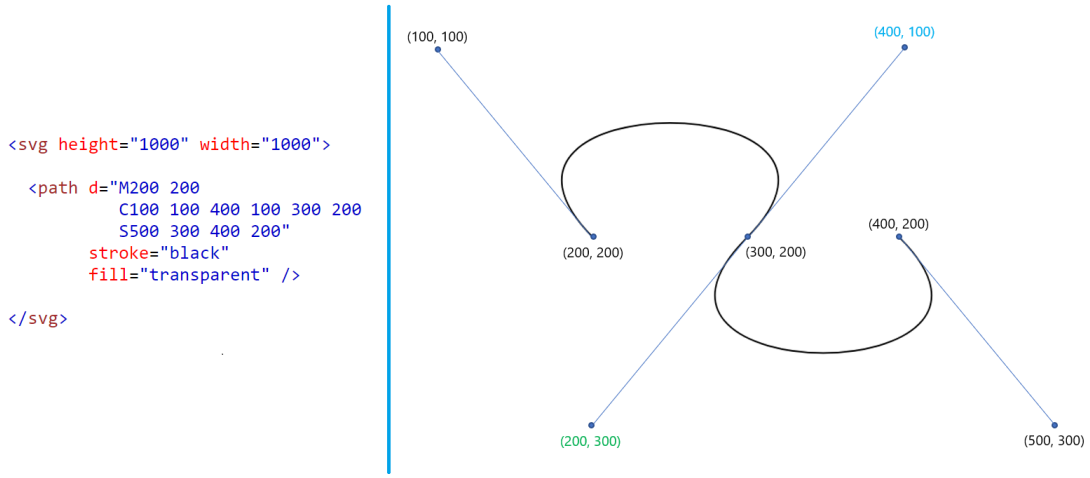


Gambar 2.16: Contoh implementasi perintah *path cubic Bezier curveto* pada dokumen SVG

8. S (*shorthand/smooth cubic Bezier curveto*)

Perintah *shorthand/smooth cubic Bezier curveto* merupakan penulisan singkat untuk membuat kurva Bezier kubik dengan memperhatikan perintah yang ada sebelumnya. Titik kontrol kedua dari kurva Bezier kubik yang akan dibangun merupakan pencerminan titik kontrol ketiga dari perintah pembentukan kurva Bezier kubik sebelumnya. Jika perintah sebelumnya bukan merupakan pembentukan kurva Bezier kubik (perintah sebelumnya bukan S atau C), kurva Bezier kuadratik akan terbentuk.

Pada Gambar 2.17 kurva Bezier kubik yang terbentuk dari penulisan singkat yang dilakukan memiliki empat buah titik kontrol. Titik kontrol pertama terletak pada koordinat (300, 200), titik kontrol kedua terletak pada koordinat (200, 300), titik kontrol ketiga terletak pada koordinat (500, 300), dan titik kontrol terakhir terletak pada koordinat (400, 200). Koordinat titik kontrol pertama merupakan koordinat terakhir perintah sebelumnya sedangkan koordinat titik kontrol kedua diperoleh dengan mencerminkan titik kontrol ketiga kurva Bezier kubik sebelumnya terhadap koordinat titik kontrol pertama kurva Bezier kubik yang akan dibangun.

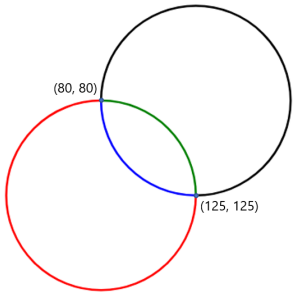


Gambar 2.17: Contoh implementasi perintah *path shorthand cubic Bezier curveto*

9. A (*elliptical arc*)
   Perintah *elliptical arc* digunakan untuk membentuk sebuah busur elips. Beberapa parameter diperlukan untuk mendefinisikan bentuk dan lokasi sebuah busur elips, yaitu, parameter rx dan ry yang merupakan radius dari elips, sudut rotasi terhadap sumbu x, *large-arc-flag*, *sweep-flag*, dan koordinat titik akhir dari busur tersebut. Parameter *sweep-flag* berfungsi untuk mendefinisikan penggambaran busur elips, jika bernilai satu, busur elips digambar searah jarum jam, jika bernilai nol, busur elips digambar berlawanan arah jarum jam. Pembentukan busur elips juga ditentukan oleh parameter *large-arc-flag*, jika parameter tersebut bernilai satu, yang dibentuk adalah busur besar, jika parameter bernilai nol, yang dibentuk adalah busur kecil.

   Terdapat empat buah kemungkinan untuk menggambar suatu busur yang merupakan bagian dari sebuah elips jika didefinisikan koordinat titik awal dan koordinat titik akhir busur tersebut ditunjukkan pada Gambar 2.18. Empat buah kemungkinan tersebut berasal dari dua buah elips yang dapat menyinggung kedua titik yang didefinisikan tersebut. Pada sebuah elips terdapat dua cara untuk menghubungkan dua buah titik, yaitu menggunakan busur besar yang memiliki sudut lebih besar atau sama dengan 180 derajat, atau menggunakan busur kecil yang memiliki sudut lebih kecil atau sama dengan 180 derajat. Jika kedua titik yang dihubungkan membagi elips menjadi dua buah busur simetris (kedua busur memiliki sudut 180 derajat), parameter *large-arc-flag* tidak berpengaruh.

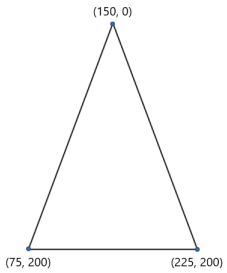
```
<svg width="325" height="325">

  <path d="M80 80 A 45 45, 0, 0, 0, 125 125"
        fill="transparent"
        stroke="blue" />

  <path d="M80 80 A 45 45, 0, 1, 0, 125 125"
        fill="transparent"
        stroke="red" />

  <path d="M80 80 A 45 45, 0, 0, 1, 125 125"
        fill="transparent"
        stroke="green" />

  <path d="M80 80 A 45 45, 0, 1, 1, 125 125"
        fill="transparent"
        stroke="black" />

</svg>
```



Gambar 2.18: Contoh implementasi perintah *path elliptical arc* pada dokumen SVG

10. Z (*close path*)
    Perintah *close path* berfungsi untuk menutup sebuah *path*. Penutupan *path* tersebut dilakukan dengan cara menambahkan sebuah ruas garis yang menghubungkan titik akhir dan titik awal *path*. Pada Gambar 2.19 terdapat ruas garis yang menghubungkan titik akhir *path* dengan titik awal *path* sehingga *path* membentuk sebuah segitiga.

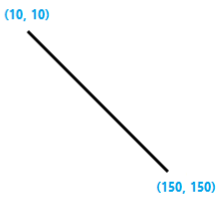
```
<svg width="1000" height="1000">
  <path d="M150 0 L75 200 L225 200 Z" stroke="black" fill="transparent"/>
</svg>
```



Gambar 2.19: Contoh implementasi *close path* pada dokumen SVG

Setiap perintah pada *path* kecuali *moveto* dapat diekspresikan menggunakan huruf kapital dan huruf non-kapital. Huruf kapital menandakan bahwa perintah tersebut diimplementasikan menggunakan *absolute positioning* dan huruf non-kapital menandakan perintah tersebut diimplementasikan menggunakan *relative positioning*. Pada Gambar 2.20 terlihat perbedaan pembentukan *path* menggunakan *absolute positioning* dan *relative positioning*. *Absolute positioning* menginterpretasikan koordinat yang mendefinisikan perintah sebagaimana adanya, sedangkan *relative positioning* menginterpretasikan koordinat masukan dengan cara menambahkannya dengan koordinat terakhir perintah *path* sebelumnya.

```
<!--absolute positioning-->
<svg height="210" width="400">
  <path d="M10 10 L150 150" stroke="black" stroke-width="3" />
</svg>


<!--relative positioning-->
<svg height="210" width="400">
  <path d="M10 10 l140 140" stroke="black" stroke-width="3" />
</svg>
```



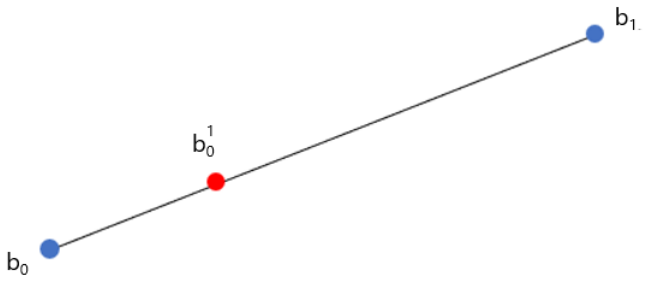
Gambar 2.20: Perbandingan implementasi *path* menggunakan *absolute* dan *relative positioning*

2.3 Algoritma de Casteljau [3]

Sebuah garis terdiri dari rangkaian sejumlah titik. Terkadang kita perlu mengetahui koordinat dari sebuah titik yang merupakan bagian dari sebuah ruas garis seperti pada Gambar 2.21. Kita dapat mendefinisikan sebuah parameter t ($0 \leq t \leq 1$) untuk menunjukkan letak titik yang dimaksud pada ruas garis tersebut. Untuk mendapatkan koordinat titik tersebut kita dapat menggunakan rumus:

$$b_0^1(t) = (1-t)\,b_0 + t\,b_1,\ 0 \leq t \leq 1 \tag{2.1}$$

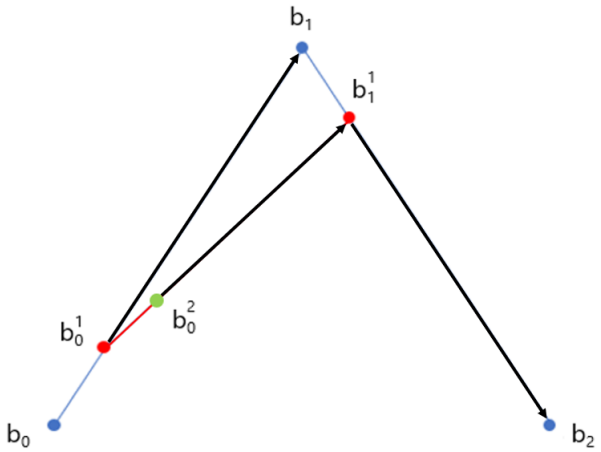
Rumus 2.1 digunakan untuk mencari koordinat titik pada sumbu x dan sumbu y sehingga perhitungan terhadap rumus tersebut dilakukan sebanyak dua kali. Jika parameter t bernilai 0, hasil perhitungan $b_0^1(t)$ menghasilkan koordinat b_0, sedangkan jika parameter t bernilai 1, hasil perhitungan $b_0^1(t)$ menghasilkan koordinat b_1. Rumus 2.1 mendasari rumus pembentukan kurva Bezier.



Gambar 2.21: Contoh titik pada ruas garis

2.3.1 Pembentukan Kurva Bezier Kuadratik

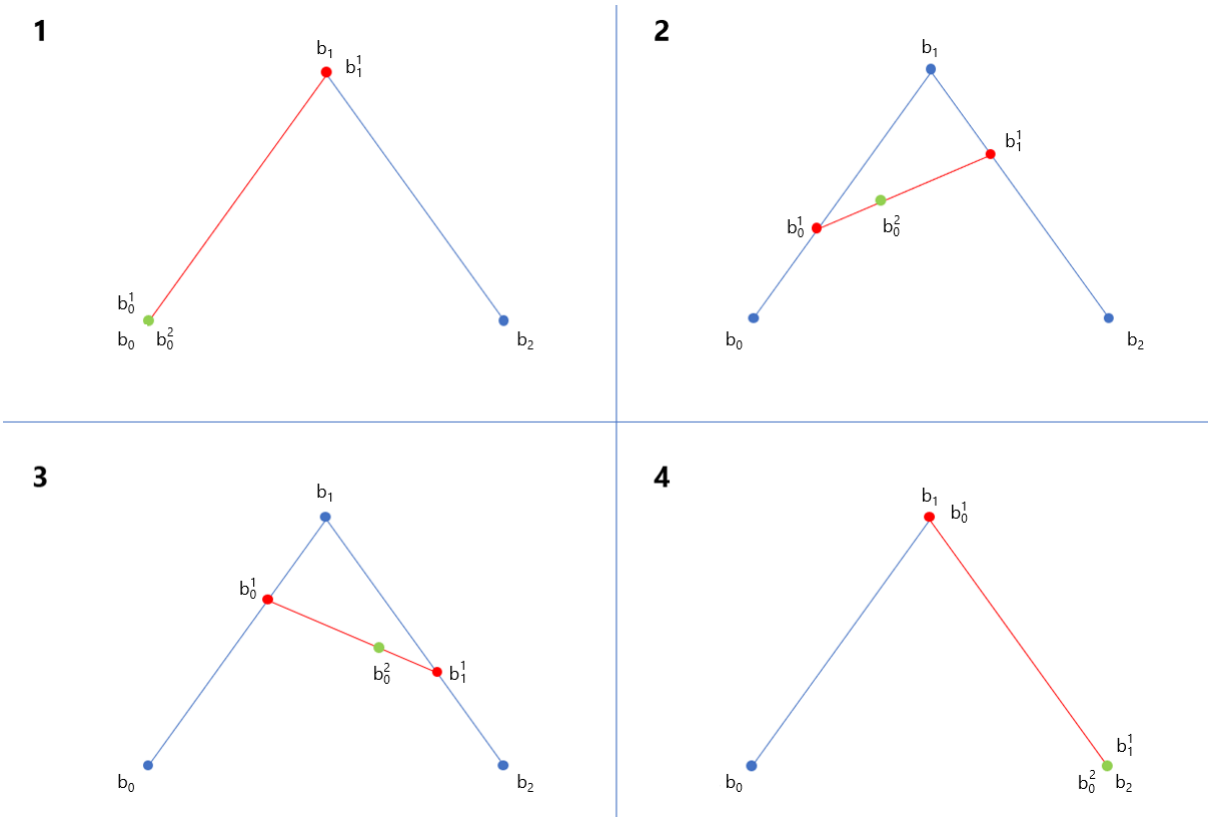
Sebuah kurva Bezier memiliki derajat atau *order* yang merupakan pangkat tertinggi pada persamaan kurva tersebut. Kurva Bezier kuadratik memiliki derajat dua dan tiga buah titik kontrol. Tiga buah titik kontrol tersebut jika dihubungkan akan membentuk *polyline* yang mengontrol pembentukan kurva Bezier kuadratik seperti pada Gambar 2.22. Dua buah ruas garis yang bersebelahan pada *polyline* tersebut dihubungkan oleh sebuah garis.



Gambar 2.22: Contoh *polyline* kontrol pada kurva Bezier kuadratik

Pada pembentukan kurva Bezier kuadratik awalnya titik b_0^1 berada pada titik kontrol b_0, titik b_1^1

berada pada titik kontrol b_1, dan titik b_0^2 berada pada titik b_0^1. Ketiga titik tersebut bergerak sesuai alur *polyline* kontrol, titik b_0^1 terus bergerak hingga berada pada titik kontrol b_1, titik b_1^1 bergerak hingga berada pada titik kontrol b_2, dan titik b_0^2 bergerak hingga berada pada titik b_1^1. Perpindahan titik b_0^2 membentuk sebuah kurva Bezier kuadratik seperti pada Gambar 2.23.



Gambar 2.23: Contoh pembentukan kurva Bezier kuadratik

Sesuai dengan pembentukan kurva yang telah dijelaskan sebelumnya, untuk mendapatkan sebuah koordinat pada kurva Bezier kuadratik kita dapat menurunkan Rumus 2.1 sehingga dapat diimplementasikan pada kurva Bezier kuadratik. Titik b_0^1 bergerak mulai dari titik kontrol b_0 hingga titik kontrol b_1 sehingga diperoleh persamaan $b_0^1(t) = (1-t)\,b_0 + t\,b_1$. Bersamaan dengan hal tersebut, titik b_1^1 bergerak mulai dari titik kontrol b_1 hingga titik kontrol b_2 dan diperoleh persamaan $b_1^1(t) = (1-t)\,b_1 + t\,b_2$. Kedua persamaan tersebut dapat disubstitusikan ke dalam persamaan b_0^2, yaitu, $b_0^2(t) = (1-t)\,b_0^1 + t\,b_1^1$ sehingga didapatkan rumus untuk mencari koordinat dengan parameter t pada kurva Bezier kuadratik sebagai berikut:

$$b_0^2(t) = (1-t)^2\,b_0 + 2\,t\,(1-t)\,b_1 + t^2\,b_2,\ 0 \leq t \leq 1 \tag{2.2}$$

Pembentukan kurva Bezier yang memiliki derajat n berhubungan dengan pembentukan kurva Bezier yang memiliki derajat n+1. Rumus pembentukan kurva Bezier yang memiliki derajat n dapat diturunkan untuk mendapatkan rumus pembentukan kurva Bezier yang memiliki derajat n+1. Algoritma de Casteljau menggeneralisasikan rumus pembentukan kurva Bezier sebagai berikut:

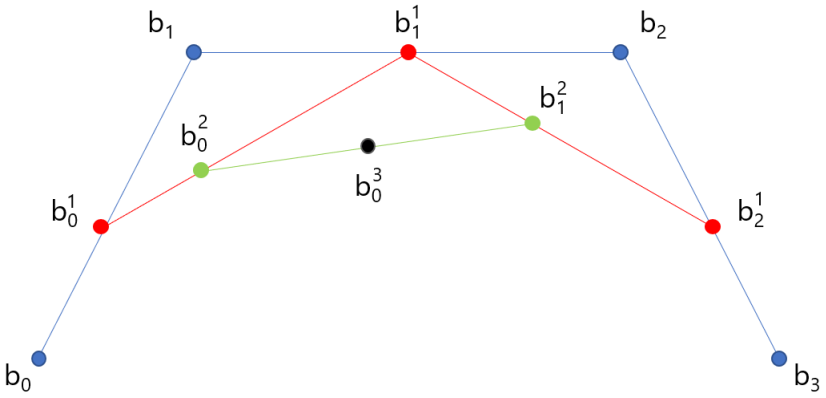
1. Diberikan titik kontrol kurva Bezier $b_0, b_1, ..., b_n$ dan parameter t $(0 \leq t \leq 1)$.
2. Persamaan-persamaan yang ada pada pembentukan kurva tersebut adalah:

$$b_i^r(t) = (1-t)\,b_i^{r-1}(t) + t\,b_{i+1}^{r-1}(t) \begin{cases} r = 1, ..., n \\ i = 0, ..., n-r \end{cases} \tag{2.3}$$

3. Koordinat titik pada kurva Bezier dengan derajat n berdasarkan parameter t adalah $b_0^n(t)$ dan $b_i^0 = b_i$.

2.3.2 Pembentukan Kurva Bezier Kubik

Kurva Bezier kubik memiliki derajat tiga dan empat buah titik kontrol (b_0, b_1, b_2, dan b_3). Sama seperti pembentukan kurva Bezier sebelumnya, keempat titik kontrol tersebut dihubungkan dengan ruas-ruas garis sehingga membentuk *polyline* yang mengontrol pembentukan kurva dan terdapat ruas garis yang menghubungkan dua buah ruas garis bersebelahan pada *polyline* tersebut. Dapat dilihat pada Gambar 2.24, pembentukan kurva Bezier kuadratik berhubungan dengan pembentukan kurva Bezier kubik.



Gambar 2.24: Contoh *polyline* kontrol pada kurva Bezier kubik

Berdasarkan algoritma de Casteljau, pembentukan kurva Bezier kubik ($n = 3$) menghasilkan rumus-rumus berikut:

$$\left.\begin{aligned} b_0^1(t) &= (1-t)\,b_0 + t\,b_1 \\ b_1^1(t) &= (1-t)\,b_1 + t\,b_2 \\ b_2^1(t) &= (1-t)\,b_2 + t\,b_3 \end{aligned}\right\} \; r = 1,\; i = 0,\,1,\,2 \tag{2.4}$$

$$\left.\begin{aligned} b_0^2(t) &= (1-t)\,b_0^1(t) + t\,b_1^1(t) \\ b_1^2(t) &= (1-t)\,b_1^1(t) + t\,b_2^1(t) \end{aligned}\right\} \; r = 2,\; i = 0,\,1 \tag{2.5}$$

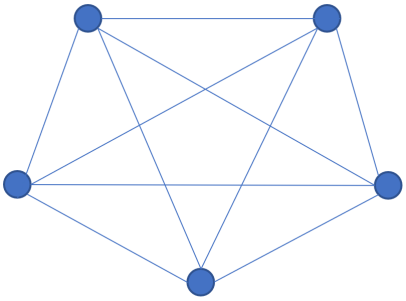
$$b_0^3(t) = (1-t)\,b_0^2(t) + t\,b_1^2(t),\; r = 3,\; i = 0 \tag{2.6}$$

Rumus 2.4 disubstitusikan ke dalam Rumus 2.5. Setelah itu, Rumus 2.5 disubstitusikan ke dalam Rumus 2.6. Proses tersebut menghasilkan persamaan pembentukan kurva Bezier kubik seperti pada Rumus 2.7.

$$b_0^3(t) = (1-t)^3\,b_0 + 3\,t\,(1-t)^2\,b_1 + 3\,t^2\,(1-t)\,b_2 + t^3\,b_3,\; 0 \le t \le 1 \tag{2.7}$$

2.4 *Graph* [4] [5] [6]

Graph merupakan sebuah struktur yang terdiri dari himpunan *vertex* dan himpunan *edge*. Sebuah *edge* pada *graph* menghubungkan dua buah *vertex* yang merupakan simpul pada *graph*. Umumnya *vertex* pada *graph* digambarkan sebagai sebuah titik dan *edge* pada *graph* digambarkan sebagai sebuah garis seperti pada Gambar 2.25.

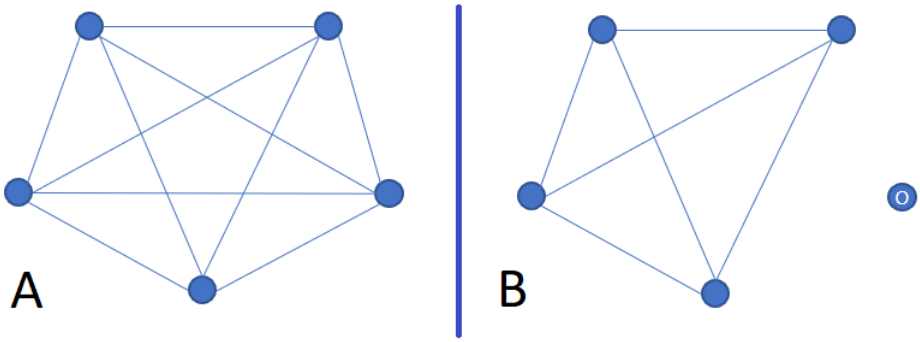


Gambar 2.25: Contoh sebuah *graph*

Setiap *vertex* pada sebuah *graph* memiliki *degree*. *Degree* merupakan nilai yang dimiliki setiap vertex untuk menyatakan jumlah *edge* yang berhubungan dengan *vertex* tersebut. Pada *self loop*, yaitu ketika sebuah *edge* menghubungkan sebuah *vertex* dengan *vertex* itu sendiri, *degree vertex* tersebut bertambah sebanyak dua.

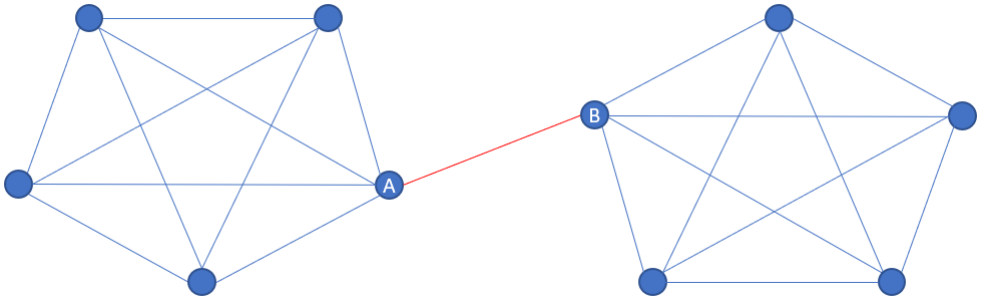
Pada sebuah *graph*, cara sebuah *vertex* mencapai suatu *vertex* lain direpresentasikan oleh sebuah *path*. *Path* pada *graph* merupakan sebuah deret yang terdiri dari *vertex* dan *edge* yang merepresentasikan sebuah jalur dari suatu *vertex* menuju *vertex* yang lain. Jika *vertex* awal dan *vertex* akhir sebuah *path* sama, *path* tersebut dapat disebut sebagai sebuah *circuit*.

Sebuah *graph* dikatakan terhubung jika terdapat paling sedikit sebuah *path* untuk setiap pasang *vertex* yang terdapat pada *graph* tersebut. Pada Gambar 2.26 *graph* A merupakan *graph* terhubung dan *graph* B merupakan *graph* tidak terhubung. *Graph* B disebut sebagai *graph* tidak terhubung karena tidak terdapat *path* yang menghubungkan *vertex* O dengan *vertex* yang lain pada *graph*.



Gambar 2.26: Contoh *graph* terhubung dan *graph* tidak terhubung

Pada teori *graph* dikenal istilah *bridge*. *Bridge* adalah sebuah *edge* pada *graph* yang jika dilepaskan dari *graph* tersebut akan menyebabkan *graph* menjadi tidak terhubung. Pada Gambar 2.27 *edge* yang menghubungkan *vertex* A dengan *vertex* B merupakan *bridge*. Jika *edge* yang menghubungkan *vertex* A dengan *vertex* B dilepaskan dari *graph*, *graph* tersebut menjadi tidak terhubung.



Gambar 2.27: Contoh sebuah *bridge*

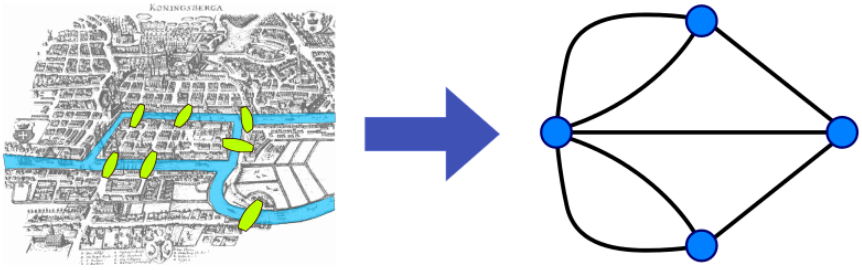
Sebuah *graph* dapat direpresentasikan menggunakan lis ketetanggaan. Lis ketetanggaan merupakan sebuah daftar yang memuat hubungan ketetanggaan setiap *vertex* pada sebuah *graph*. Selain lis ketetanggaan, *graph* juga dapat direpresentasikan sebagai *list of edges* dan *adjacency matrix*.

Penelusuran pada *graph* dilakukan untuk memperoleh informasi dari *graph* atau melakukan perubahan terhadap *graph* tersebut. Terdapat berbagai metode yang dapat digunakan untuk menelusuri sebuah *graph*. Salah satu metode penelusuran *graph* yang umum digunakan adalah *depth first search* (DFS).

Metode DFS menelusuri sebuah *graph* dengan mengutamakan penelusuran secara mendalam. Setiap kali penelusuran menemukan percabangan, yaitu ketika sebuah *vertex* yang sedang ditelusuri memiliki lebih dari satu tetangga, metode DFS mengutamakan penelusuran terhadap salah satu tetangga *vertex* tersebut dan mengunjungi tetangganya. Proses tersebut terus berulang hingga *vertex* yang ditelusuri tidak memiliki tetangga yang dapat ditelusuri. Setelah penelusuran tidak dapat menelusuri *graph* lebih dalam, proses penelusuran mundur dan menelusuri tetangga yang belum ditelusuri, jika terdapat tetangga suatu *vertex* yang belum ditelusuri.

2.4.1 *Euler Graph*

Euler graph pertama kali dipelajari oleh Leonhard Euler ketika menyelesaikan permasalahan *Seven Bridges of Königsberg* pada tahun 1736. Permasalahan *Seven Bridges of Königsberg* memodelkan tujuh jembatan *Königsberg* sebagai *edge* pada sebuah *graph* dan daratan yang dihubungkan oleh tujuh jembatan tersebut sebagai *vertex* pada *graph*. Masalah utama pada *Seven Bridges of Königsberg* adalah mencari sebuah *path* atau *circuit* pada *graph* yang menelusuri semua *edge* pada *graph* tersebut tepat satu kali. Gambar 2.28 adalah pemodelan permasalahan tujuh jembatan *Königsberg* sebagai sebuah *graph*.



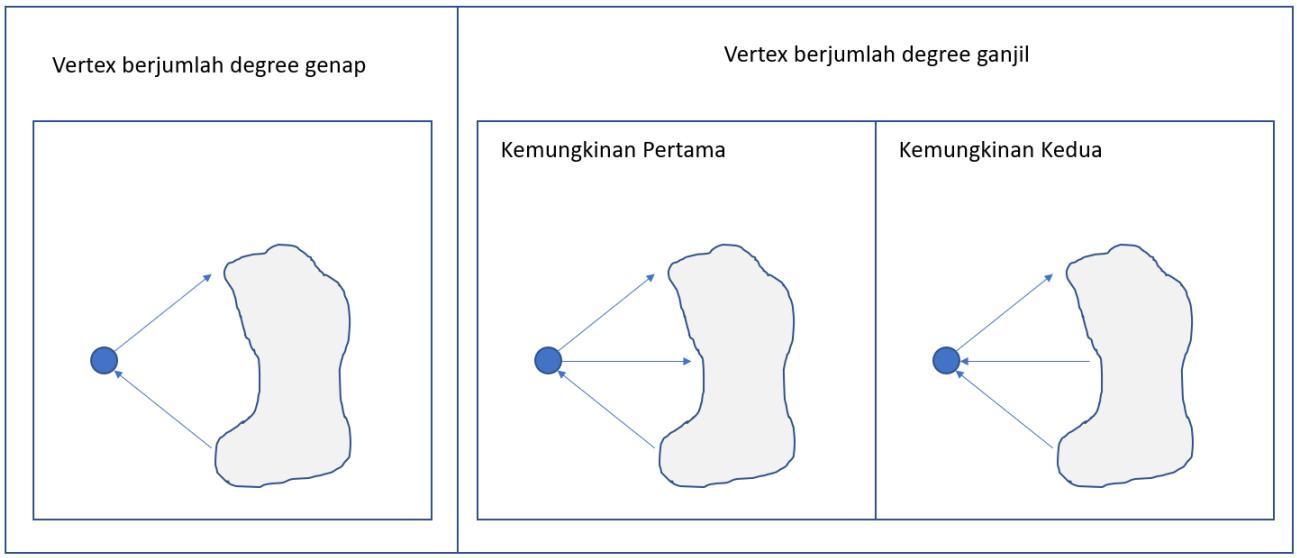
Gambar 2.28: Pemodelan permasalahan *Seven Bridge of Königsberg* sebagai sebuah *graph*

Pada permasalahan *Seven Bridge of Königsberg* tidak terdapat *path* atau *circuit* yang dapat menelusuri semua *edge* pada *graph* tepat satu kali. *Graph* hasil pemodelan permasalahan *Seven Bridge of Königsberg* bukan merupakan *euler graph*. Sebuah *graph* dapat disebut sebagai *euler graph* jika *graph* tersebut memiliki *euler path* atau *euler circuit*.

Euler path adalah sebuah *path* pada suatu *graph* yang menelusuri setiap *edge* pada *graph* tersebut tepat satu kali, sedangkan *euler circuit* adalah sebuah *euler path* yang dimulai dan berakhir pada

vertex yang sama. Sebuah *graph* memiliki *euler path* jika pada *graph* tersebut terdapat tepat dua buah *vertex* yang memiliki *degree* berjumlah ganjil. *Euler path* dimulai dari sebuah *vertex* yang memiliki *degree* berjumlah ganjil dan berakhir pada *vertex* yang memiliki jumlah *dgree* ganjil lainnya. Jika pada sebuah *graph* tidak terdapat *vertex* yang memiliki *degree* berjumlah ganjil, *graph* tersebut memiliki *euler circuit*.

Pada *vertex* yang memiliki *degree* berjumlah genap, penelusuran yang memasuki *vertex* sebanding dengan penelusuran yang meninggalkan *vertex*, sedangkan pada *vertex* yang memiliki *degree* berjumlah ganjil penelusuran yang memasuki *vertex* tidak sebanding dengan penelusuran yang meninggalkan *vertex*. Hal tersebut menyebabkan penelusuran yang melewati seluruh *edge* pada *graph* tepat satu kali hanya dapat dilakukan pada *graph* yang memiliki *vertex* berjumlah *degree* ganjil sebanyak tepat dua atau pada *graph* yang tidak memiliki *vertex* yang berjumlah *degree* ganjil. Gambar 2.29 merupakan perbandingan penelusuran yang melibatkan *vertex* dengan jumlah *degree* genap dan penelusuran yang melibatkan *vertex* dengan jumlah *degree* ganjil.

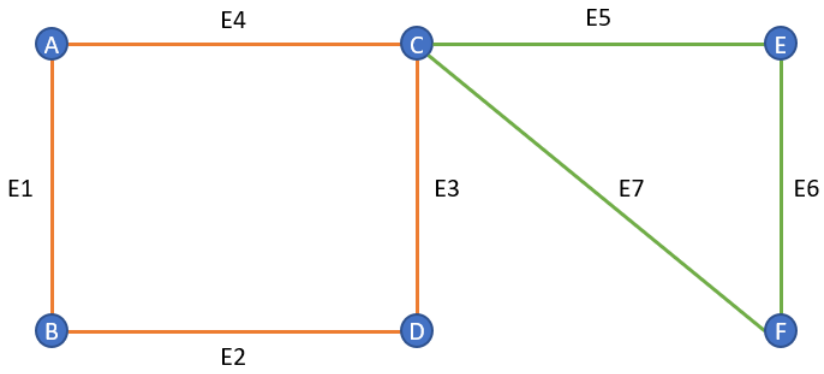


Gambar 2.29: Perbandingan penelusuran pada *vertex* berjumlah *degree* genap dan ganjil

2.4.2 Algoritma Hierholzer

Terdapat berbagai cara untuk menentukan *euler path* dan *euler circuit* pada sebuah *euler graph*. Algoritma Hierholzer mendefinisikan cara untuk memperoleh *euler circuit* pada sebuah *euler graph*. Kompleksitas waktu dari algoritma Hierholzer adalah $O(|E|)$, $|E|$ merupakan kardinalitas himpunan *edge* pada *graph*.

Ide dasar dari algoritma Hierholzer adalah membentuk *euler circuit* secara berkala. Pertama-tama algoritma ini memulai penelusuran dari sebuah *vertex* pada *graph*. Penelusuran tersebut berlanjut hingga membentuk sebuah *circuit*. Jika penelusuran belum mengunjungi semua *edge* pada *graph*, penelusuran dilanjutkan dari sebuah *vertex* anggota *circuit* tersebut yang masih memiliki *edge* yang belum ditelusuri. Setelah penelusuran tersebut membentuk sebuah *circuit* lain, *circuit* tersebut digabungkan dengan *circuit* utama yang pertama kali terbentuk.



Gambar 2.30: Contoh pembentukan *euler circuit* algoritma Hierholzer

Pada Gambar 2.30, pertama-tama algoritma Hierholzer membentuk sebuah *circuit*, yaitu, $(A - E1 - B - E2 - D - E3 - C - E4 - A)$. Setelah terbentuk sebuah *circuit*, penelusuran dilanjutkan dari *vertex* C dan membentuk sebuah *circuit* lain, yaitu, $(C - E5 - E - E6 - F - E7 - C)$. Kedua buah *circuit* tersebut kemudian digabungkan sehingga membentuk *euler circuit* $(A - E1 - B - E2 - D - E3 - C - E5 - E - E6 - F - E7 - C - E4 - A)$. Jika algoritma Hierholzer menelusuri *graph* yang hanya memiliki *euler path*, algoritma Hierholzer menghasilkan sebuah *euler path* yang dimulai dari sebuah *vertex* yang memiliki *degree* berjumlah ganjil dan berakhir pada *vertex* lain yang juga memiliki *degree* berjumlah ganjil. Algorithm 1 merupakan *pseudocode* dari algoritma Hierholzer.

Algorithm 1 Hierholzer

Require:
adjList, lis ketetanggaan yang merepresentasikan sebuah *graph*
u, vertex awal penelusuran
Ensure:
eulerPath, sebuah *euler path* atau *euler circuit* yang terdapat pada *graph*

```
function HIERHOLZER(adjList, v)
    for all v : neighbors(u) do
        if v has not been visited yet then
            Set v visited
            for all uu : neighbors(v) do
                if uu is u AND uu has not been visited yet then
                    Set uu visited
                    break
                end if
            end for
        end if
        HIERHOLZER(adjList, v)
        Add v to eulerPath
    end for
    return eulerPath
end function
```

BAB 3

ANALISIS MASALAH

Bab ini berisi dasar-dasar pemikiran pada skripsi ini. Terdapat beberapa hal yang harus ditangani ketika mengubah gambar menjadi sebuah *graph*. Pada bab ini dijelaskan tentang masalah-masalah tersebut beserta cara menanganinya. Agar algoritma Hierholzer yang digunakan untuk mencari *euler path* optimal dapat bekerja, *graph* hasil pemodelan gambar masukan harus diubah menjadi *euler graph*. Pada bab ini juga dibahas mengenai permasalahan mengubah *graph* menjadi *euler graph*.

3.1 Analisis Cara Mengubah Gambar SVG Menjadi Graph

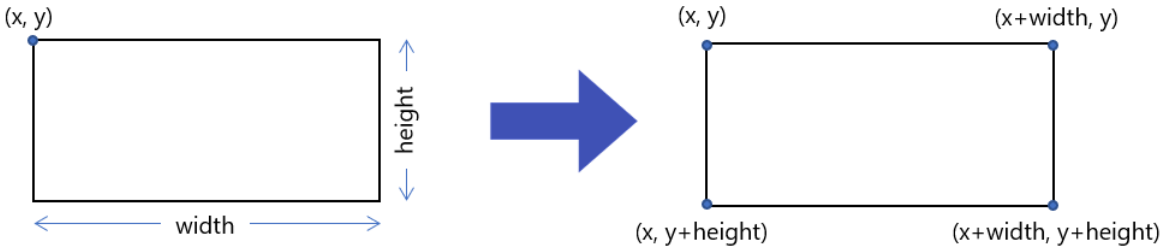
Sebuah gambar SVG terdiri dari berbagai elemen yang mendefinisikan gambar tersebut. Setiap elemen harus diubah menjadi sekumpulan *vertex* dan *edge* sehingga terbentuk sebuah bagian *graph* yang merepresentasikan elemen tersebut. Pada dokumen SVG terdapat dua cara untuk mendefinisikan bentuk, yaitu, menggunakan bentuk dasar dan *path*.

Terdapat beberapa hal yang harus ditangani ketika mengubah gambar menjadi sebuah *graph*. Beberapa permasalahan yang timbul pada proses pengubahan gambar menjadi sebuah *graph* adalah mengubah kurva menjadi beberapa ruas garis, menangani bagian gambar yang terlalu kecil untuk dijadikan bagian soal, dan menangani bagian gambar yang bertumpukan. Bagian ini menjelaskan masalah-masalah tersebut beserta cara menanganinya.

3.1.1 Analisis Cara Mengubah Bentuk Dasar pada SVG Menjadi *Graph*

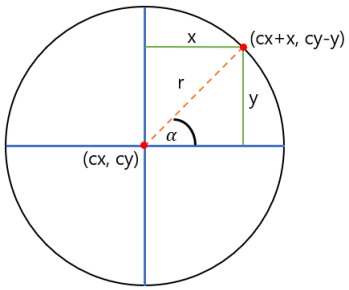
Terdapat enam buah bentuk dasar yang dapat digunakan untuk mendefinisikan gambar SVG pada dokumen SVG. Keenam bentuk dasar tersebut memiliki atribut yang berbeda-beda sehingga proses pengubahan suatu bentuk dasar menjadi *graph* berbeda dengan proses pengubahan bentuk dasar lainnya. Berikut adalah masing-masing cara atau proses pengubahan keenam bentuk dasar tersebut:

1. Persegi panjang
   Sebuah persegi panjang direpresentasikan sebagai sebuah *graph* dengan menjadikan keempat buah titik yang dimilikinya sebagai *vertex* dan keempat sisinya sebagai *edge*. Pendefinisian bentuk dasar persegi panjang pada dokumen SVG dilakukan dengan menentukan koordinat titik kiri atas, lebar, dan tinggi persegi panjang tersebut. Tiga titik lain yang belum didefinisikan dapat dicari dengan menambahkan lebar atau tinggi pada koordinat titik kiri atas tersebut sehingga terbentuk sebuah *graph* seperti pada Gambar 3.1.



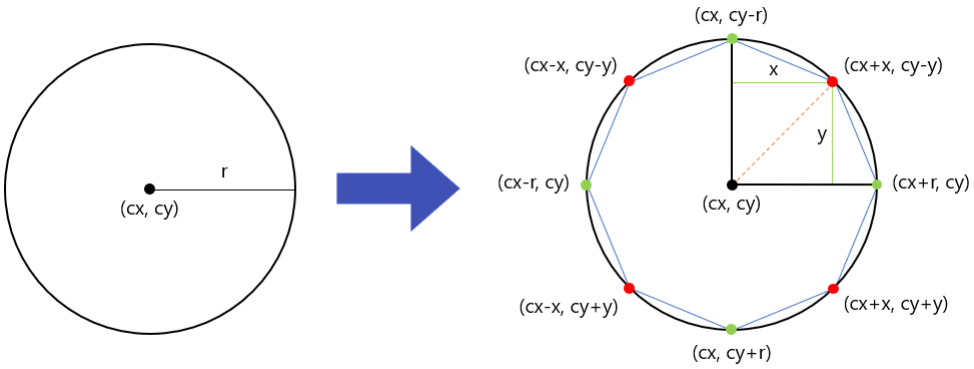
Gambar 3.1: Contoh pengubahan bentuk dasar persegi panjang menjadi *graph*

2. Lingkaran
   Sebuah lingkaran didefinisikan dengan koordinat titik pusat dan radius pada dokumen SVG. Dalam permainan menghubungkan titik, dua buah titik dihubungkan menggunakan garis lurus. Sebuah lingkaran diubah terlebih dahulu menjadi segi delapan untuk memenuhi hal tersebut. Segi delapan memiliki delapan buah titik dan delapan buah sisi. Empat dari delapan buah titik tersebut dapat diperoleh dengan menambahkan atau mengurangkan koordinat titik pusat dengan radius.



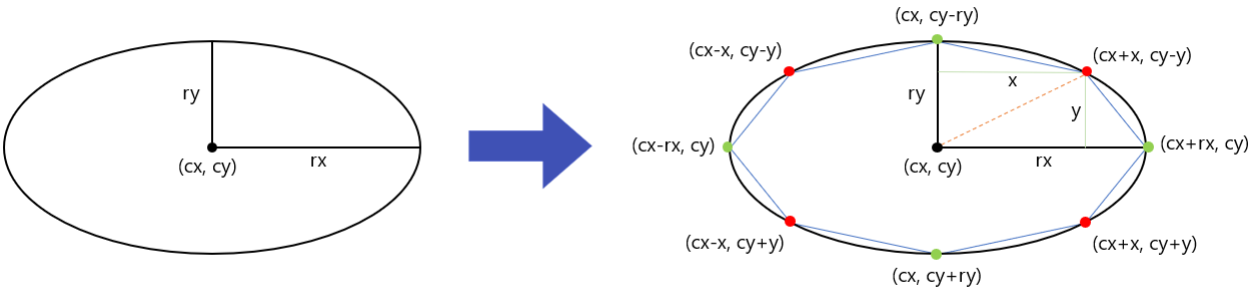
Gambar 3.2: Contoh titik pada lingkaran

Koordinat empat titik lainnya yang belum diperoleh dapat didefinisikan jika diketahui besar x dan y pada Gambar 3.2. Radius lingkaran, x, dan y membentuk segitiga dengan sudut α sehingga besar x dapat dicari dengan rumus $x = r \cos \alpha$ dan besar y dapat dicari dengan rumus $y = r \sin \alpha$ dengan sudut α bernilai 45°. Jika telah didapatkan sebuah titik menggunakan rumus tersebut, pencerminan titik tersebut terhadap garis x = cx atau garis y = cy dapat dilakukan untuk mencari tiga titik lainnya. Sebuah *graph* terbentuk seperti pada Gambar 3.3 setelah diperoleh delapan buah titik yang membentuk segi delapan.



Gambar 3.3: Contoh pengubahan bentuk dasar lingkaran menjadi *graph*

3. Elips
   Sama seperti lingkaran, sebuah elips diubah menjadi segi delapan terlebih dahulu kemudian setiap titik pada segi delapan tersebut dijadikan *vertex* pada *graph* dan setiap sisi dijadikan *edge* pada *graph*. Karena elips memiliki dua buah radius, tidak semua sisi pada segi delapan yang terbentuk memiliki panjang yang sama. Rumus yang digunakan untuk mencari besar x dan y juga berbeda dengan lingkaran karena elips memiliki dua buah radius sehingga rumus untuk mencari besar x adalah $x = rx \cos \alpha$ dan rumus untuk mencari besar y adalah $y = ry \sin \alpha$ dengan α bernilai 45°. Sebuah *graph* seperti pada Gambar 3.4 setelah diperoleh delapan buah titik yang membentuk segi delapan.



Gambar 3.4: Contoh pengubahan bentuk dasar elips menjadi *graph*

4. Garis
   Sebuah garis didefinisikan pada dokumen SVG dengan cara menentukan koordinat titik awal dan akhir garis tersebut. Dua buah titik tersebut diubah menjadi *vertex* dan segmen garis yang menghubungkan dua titik tersebut diubah menjadi *edge* pada *graph*.

5. *Polyline*
   Sebuah *polyline* memiliki banyak titik. Titik-titik pada *polyline* saling terhubung melalui segmen garis. Titik pada *polyline* menjadi *vertex* pada *graph* dan segmen garis yang menghubungkan titik-titik tersebut dijadikan *edge* pada *graph*.

6. Poligon
   Sama seperti *polyline*, setiap titik pada poligon dijadikan *vertex* pada *graph* dan setiap segmen garis yang menghubungkan titik-titik tersebut dijadikan *edge*, tetapi pada poligon titik akhir dan titik awal terhubung.

3.1.2 Analisis Cara Mengubah *Path* pada SVG Menjadi *Graph*

Cara lain untuk mendefinisikan bentuk pada gambar SVG adalah menggunakan *path*. Terdapat sepuluh perintah yang dapat diimplementasikan menggunakan *path* pada dokumen SVG. Perintah-perintah tersebut jika diimplementasikan menggunakan huruf kapital akan diinterpretasikan sebagai *absolute positioning* dan jika diimplementasikan menggunakan huruf non-kapital akan diinterpretasikan sebagai *relative positioning*.

Relative positioning menginterpretasikan parameter yang didefinisikan sebagai jarak dari koordinat terakhir perintah sebelumnya. Berbeda dengan *relative positioning*, *absolute positioning* menginterpretasikan parameter sebagai sebuah koordinat. Pada Gambar 2.20 dapat terlihat perbedaan implementasi menggunakan *absolute positioning* dan *relative positioning* untuk menghasilkan gambar yang sama.

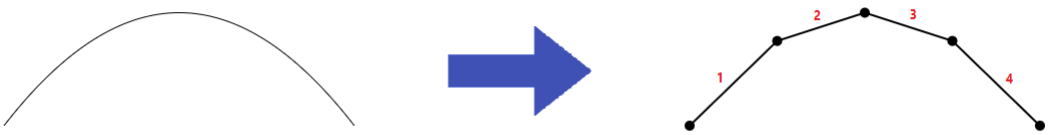
Pada skripsi ini seluruh implementasi perintah *path* menggunakan *relative positioning* diubah menjadi *absolute positioning* terlebih dahulu sebelum mengubah *path* tersebut menjadi sebuah

graph. Pengubahan implementasi perintah *path* yang menggunakan *relative positioning* menjadi *absolute positioning* dilakukan dengan memperhatikan perintah-perintah sebelumnya. Koordinat akhir setiap perintah yang ada perlu dicatat untuk mengubah parameter berupa jarak pada *relative positioning* menjadi sebuah koordinat seperti parameter pada *absolute positioning*. Koordinat hasil pencatatan tersebut ditambahkan dengan parameter perintah *relative positioning* yang berupa jarak untuk mendapatkan koordinat perintah tersebut.

Setelah implementasi perintah *path* diseragamkan menjadi *absolute positioning*, pengubahan *path* menjadi sebuah *graph* dapat lebih mudah dilakukan. Pada umumnya pengubahan *path* menjadi *graph* dilakukan dengan menjadikan koordinat parameter sebuah perintah sebagai *vertex* pada *graph* dan garis yang menghubungkan koordinat-koordinat parameter tersebut dijadikan *edge* pada *graph*. Penanganan khusus diperlukan untuk mengubah kurva dan busur elips yang dibentuk menggunakan *path*. Masalah tersebut dibahas pada bagian selanjutnya.

3.1.3 Analisis Cara Mengubah Kurva Menjadi Beberapa Ruas Garis

Sebuah gambar dapat memuat sejumlah kurva. Pada permainan menghubungkan titik, setiap titik dihubungkan menggunakan garis lurus sehingga jika pada gambar terdapat kurva, kurva tersebut harus diubah menjadi beberapa ruas garis. Sebuah kurva diubah menjadi empat buah ruas garis seperti pada Gambar 3.5 untuk mempertahankan bentuk kurva.



Gambar 3.5: Contoh perubahan kurva menjadi empat buah ruas garis

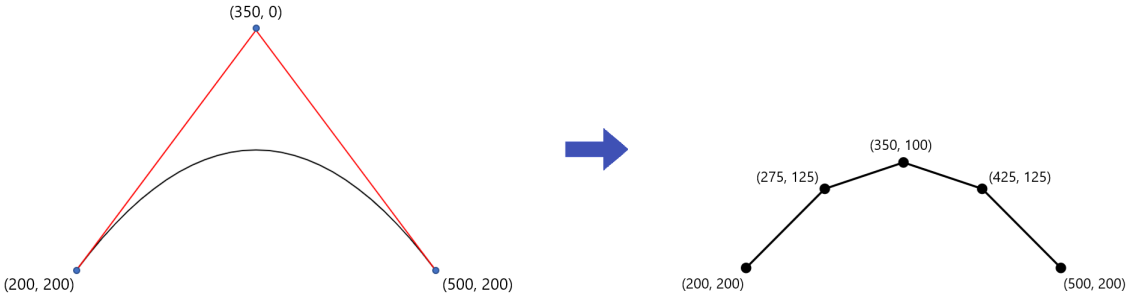
Path digunakan untuk membuat kurva pada dokumen SVG. Terdapat dua jenis kurva Bezier yang dapat dibentuk menggunakan *path*, yaitu, kurva Bezier kuadratik dan kubik. Kurva Bezier kuadratik memiliki derajat dua dan kurva Bezier kubik memiliki derajat tiga. Setiap kurva Bezier memiliki titik kontrol sejumlah derajatnya ditambah satu. Contohnya, kurva Bezier dengan derajat tiga (kurva Bezier kubik) memiliki empat buah titik kontrol. Koordinat lima buah titik yang menentukan ruas-ruas garis tersebut harus diperoleh terlebih dahulu untuk membagi kurva menjadi empat buah ruas garis.

Rumus 2.2 dapat digunakan untuk mendapatkan koordinat sebuah titik pada kurva Bezier kuadratik dan Rumus 2.7 dapat digunakan untuk mendapatkan koordinat sebuah titik pada kurva Bezier kubik. Perhitungan menggunakan rumus tersebut dilakukan sebanyak dua kali untuk mencari koordinat titik pada sumbu x dan sumbu y. Koordinat seluruh titik kontrol dan parameter t yang menunjukan letak titik pada kurva dibutuhkan dalam perhitungan rumus tersebut.

Pada dokumen SVG, pembentukan kurva menggunakan *path* memerlukan koordinat titik kontrol sebagai parameter yang menentukan bentuk kurva. Koordinat titik kontrol tersebut digunakan sebagai masukan perhitungan rumus sesuai dengan derajat kurva tersebut. Perhitungan rumus untuk sebuah kurva dilakukan dengan parameter t yang bernilai 0.25, 0.5, dan 0.75. Koordinat titik awal dan akhir kurva yang merupakan titik kontrol pertama dan titik kontrol terakhir kurva telah didapatkan sebelumnya.

Misalkan terdapat sebuah kurva Bezier kuadratik yang memiliki tiga buah titik kontrol pada koordinat (200, 200), (350, 0), dan (500 200). Rumus 2.2 digunakan untuk mendapatkan koordinat tiga buah titik yang membagi kurva menjadi empat buah ruas garis. Perhitungan menggunakan rumus tersebut dengan parameter t bernilai 0.25 menghasilkan koordinat (275, 125), dengan parameter t bernilai 0.5 menghasilkan koordinat (350, 100), dan dengan parameter t bernilai 0.75

menghasilkan koordinat (425, 125). Pengubahan kurva Bezier kuadratik tersebut menjadi ruas-ruas garis ditunjukkan pada Gambar 3.6.



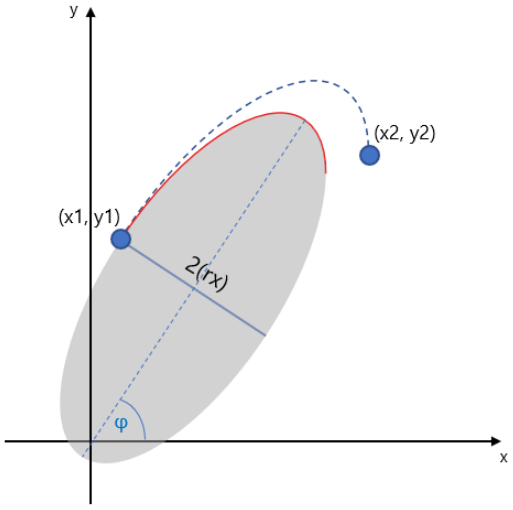
Gambar 3.6: Contoh hasil pengubahan kurva menjadi beberapa ruas garis

3.1.4 Analisis Cara Mengubah Busur Elips Menjadi *Graph* [2]

Busur elips pada dasarnya merupakan bagian dari elips. Sama seperti sebuah kurva, busur elips yang dibentuk oleh *path* SVG tidak dapat langsung diubah menjadi *graph*. Sebuah busur elips diubah terlebih dahulu menjadi sebuah ruas garis atau beberapa ruas garis sesuai dengan sudut yang dibentuk oleh busur elips tersebut.

Pada skripsi ini sebuah elips diubah menjadi *graph* yang berbentuk segi delapan. Terdapat sebuah *vertex* setiap 45 derajat pada sebuah *graph* yang terbentuk dari sebuah elips. Berdasarkan hal tersebut, banyaknya *vertex* pada *graph* yang terbentuk dari sebuah busur elips adalah sebanyak $\triangle\theta \div 45^o + 1$, $\triangle\theta$ merupakan besar sudut yang dibentuk busur elips. Jika sudut yang dibentuk busur elips lebih kecil dari 45 derajat, jumlah *vertex* pada *graph* yang terbentuk dari busur elips tersebut adalah dua.

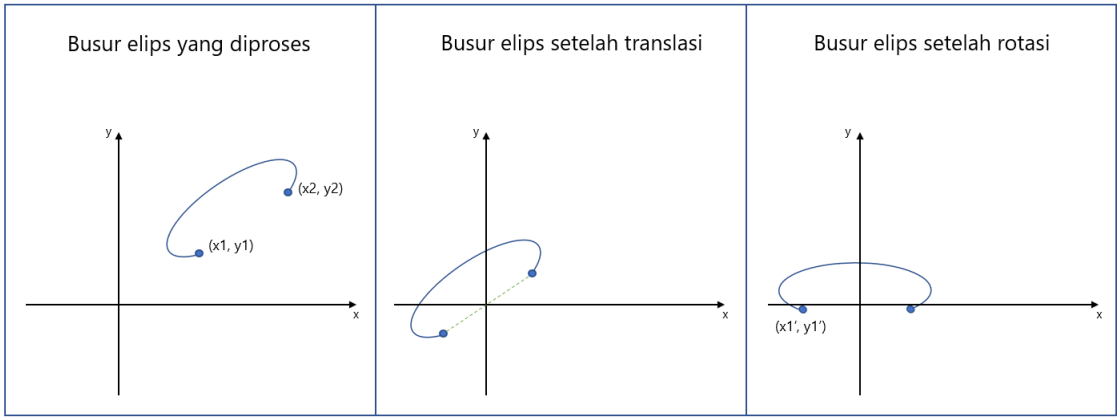
Parameter sebuah busur elips adalah rx dan ry yang merupakan radius dari elips, sudut rotasi terhadap sumbu x (ϕ), *large arc flag*, *sweep flag*, dan titik akhir busur elips. Titik awal busur elips dapat diketahui dengan melihat koordinat terakhir perintah *path* sebelumnya. Tidak semua argumen yang dimasukan oleh pengguna dapat membentuk busur elips. Kombinasi nilai radius, sudut rotasi terhadap sumbu x, titik awal, dan titik akhir yang tidak sesuai dapat menyebabkan busur elips tidak dapat dibentuk seperti pada Gambar 3.7. Pada gambar tersebut elips yang terbentuk dari kombinasi nilai radius elips dan sudut rotasi terhadap sumbu x tidak cukup besar untuk menyinggung titik awal dan titik akhir busur elips yang didefinisikan.



Gambar 3.7: Contoh kombinasi nilai argumen yang tidak sesuai pada pembentukan busur elips

Koordinat titik-titik yang membagi busur elips menjadi ruas-ruas garis perlu diketahui untuk mengubah busur elips menjadi ruas-ruas garis. Setelah koordinat titik-titik tersebut ditentukan, sebuah *graph* dibentuk dengan menjadikan titik-titik tersebut sebagai *vertex* dan ruas-ruas garis yang menghubungkan titik-titik tersebut sebagai *edge* pada *graph*. Beberapa langkah yang perlu dilakukan untuk menentukan titik-titik yang membagi busur elips menjadi ruas-ruas garis adalah sebagai berikut:

1. Melakukan translasi sehingga garis yang menghubungkan titik awal dan titik akhir busur elips berpusat pada titik origin, yaitu titik yang memiliki koordinat (0, 0). Kemudian, busur elips dirotasi sehingga sumbu elips sejajar dengan sumbu koordinat. Gambar 3.8 menunjukkan transformasi yang dilakukan pada busur elips.



Gambar 3.8: Transformasi yang dilakukan pada busur elips

2. Menggunakan Rumus 3.1 untuk menentukan koordinat titik awal busur elips setelah transformasi dilakukan.

$$\begin{pmatrix} x_1\,' \\ y_1\,' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} \frac{x_1 - x_2}{2} \\ \frac{y_1 - y_2}{2} \end{pmatrix} \tag{3.1}$$

3. Menggunakan Rumus 3.2 untuk menentukan titik pusat elips setelah transformasi dilakukan. Tanda $\pm$ pada Rumus 3.2 bernilai positif jika nilai *large arc flag* tidak sama dengan nilai *sweep flag* dan bernilai negatif jika nilai *large arc flag* sama dengan nilai *sweep flag*.

$$\begin{pmatrix} c_x\,' \\ c_y\,' \end{pmatrix} = \pm\sqrt{\frac{r_x^2\, r_y^2 \;-\; r_x^2\,(y_1\,')^2 \;-\; r_y^2\,(x_1\,')^2}{r_x^2\,(y_1\,')^2 \;+\; r_y^2\,(x_1\,')^2}} \begin{pmatrix} \frac{r_x\, y_1\,'}{r_y} \\ -\frac{r_y\, x_1\,'}{r_x} \end{pmatrix} \tag{3.2}$$

4. Sebelum melanjutkan perhitungan, radius elips perlu disesuaikan agar busur elips dapat dibentuk. Rumus 3.3 digunakan untuk menyesuaikan radius elips.

$$r_x = \sqrt{\frac{(x_1\,')^2}{r_x^2} + \frac{(y_1\,')^2}{r_y^2}}\, r_x \qquad r_y = \sqrt{\frac{(x_1\,')^2}{r_x^2} + \frac{(y_1\,')^2}{r_y^2}}\, r_y \tag{3.3}$$

5. Menggunakan Rumus 3.4 untuk menentukan titik pusat elips.

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} c_x\,' \\ c_y\,' \end{pmatrix} + \begin{pmatrix} \frac{x_1 + x_2}{2} \\ \frac{y_1 + y_2}{2} \end{pmatrix} \tag{3.4}$$

6. Menggunakan Rumus 3.5 untuk menentukan sudut yang dibentuk titik awal busur elips

dengan sumbu x.

$$\theta_1 = \angle\left(\begin{pmatrix}1\\0\end{pmatrix}, \quad \begin{pmatrix}\frac{{x_1}' - {c_x}'}{r_x}\\ \frac{{y_1}' - {c_y}'}{r_y}\end{pmatrix}\right) \tag{3.5}$$

7. Menggunakan Rumus 3.6 untuk menentukan besar sudut yang dibentuk oleh busur elips.

$$\triangle\theta = \angle\left(\begin{pmatrix}\frac{{x_1}' - {c_x}'}{r_x}\\ \frac{{y_1}' - {c_y}'}{r_y}\end{pmatrix}, \quad \begin{pmatrix}\frac{-{x_1}' - {c_x}'}{r_x}\\ \frac{-{y_1}' - {c_y}'}{r_y}\end{pmatrix}\right) mod\, 360^o \tag{3.6}$$

8. Rumus 3.7 digunakan untuk menghitung sudut antar vektor pada langkah 6 dan 7. Tanda $\pm$ pada Rumus 3.7 bernilai positif jika $(u_x\, v_y - u_y\, v_x)$ lebih besar atau sama dengan nol dan bernilai negatif jika $(u_x\, v_y - u_y\, v_x)$ lebih kecil dari nol.

$$\angle\left(\vec{u}, \quad \vec{v}\right) = \pm \arccos \frac{\vec{u}\,.\,\vec{v}}{\|\vec{u}\|\,\|\vec{v}\|} \tag{3.7}$$

9. Setelah sudut titik awal dan besar sudut busur elips diketahui, langkah selanjutnya adalah menentukan koordinat titik-titik yang membagi busur elips menjadi ruas-ruas garis. Setiap 45 derajat, dimulai dari sudut titik awal busur elips, Rumus 3.8 digunakan utnuk menentukan koordinat titik-titik yang membagi busur elips menjadi ruas-ruas garis.

$$\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}\cos\varphi & -\sin\varphi\\ \sin\varphi & \cos\varphi\end{pmatrix}\begin{pmatrix}r_x\cos\theta\\ r_y\sin\theta\end{pmatrix} + \begin{pmatrix}c_x\\c_y\end{pmatrix} \tag{3.8}$$

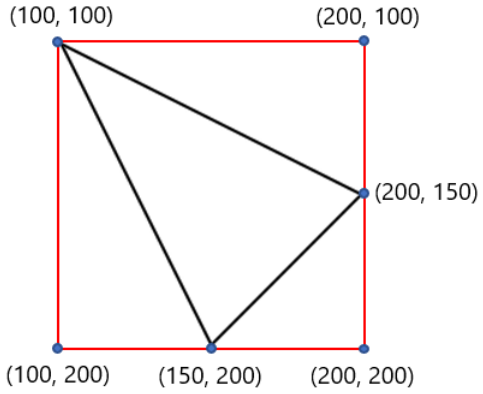
3.1.5 Analisis Cara Menangani Bagian Gambar yang Terlalu Kecil

Sebuah bagian gambar diimplementasikan pada dokumen SVG dengan mendefinisikan sebuah elemen. Elemen tersebut memiliki atribut yang berbeda-beda. Atribut-atribut pada elemen menentukan bentuk serta ukuran bagian gambar yang elemen tersebut representasikan.

Perangkat lunak yang dikembangkan pada skripsi ini menghasilkan keluaran berupa soal permainan menghubungkan titik sebagai gambar SVG. Sebuah titik pada pada gambar soal tersebut diimplementasikan menggunakan bentuk dasar lingkaran yang memiliki radius sebesar lima *pixel*. Sebuah bagian gambar yang terlalu kecil menyebabkan beberapa titik pada gambar soal bersinggungan sehingga membuat pengerjaan soal menjadi tidak memungkinkan. Bagian gambar yang terlalu kecil tidak dapat dijadikan bagian soal sehingga bagian gambar tersebut dijadikan garis bantuan.

Sebuah gambar SVG dapat terbentuk dari berbagai bangun datar. Luas sebuah bangun datar dapat dihitung untuk menentukan apakah bangun datar tersebut dapat menjadi bagian soal atau tidak. Namun, rumus perhitungan luas bangun datar berbeda-beda sehingga implementasi perhitungan luas setiap bangun datar yang ada pada gambar SVG menjadi rumit. Bangun datar sembarang juga dapat diimplementasikan menggunakan *path* pada dokumen SVG.

Bounding rectangle dapat digunakan untuk memperkirakan ukuran dari suatu bagian gambar. *Bounding rectangle* sebuah bangun datar merupakan persegi panjang yang memiliki tinggi dan lebar lebih besar dari bangun datar tersebut dan memiliki luas yang minimum seperti pada Gambar 3.9. Ukuran bagian gambar dapat diperkirakan dengan menghitung luas *bounding rectangle* bangun datar tersebut.



Gambar 3.9: Contoh *bounding rectangle* sebuah segitiga

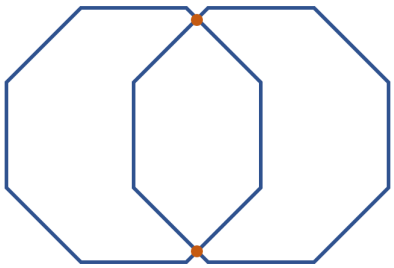
Sama seperti bentuk dasar persegi panjang, sebuah *bounding rectangle* didefinisikan dengan empat buah titik. Keempat titik tersebut dapat diperoleh dengan mencari nilai x minimal dan maksimal serta nilai y minimal dan maksimal dari sebuah bangun datar yang bersangkutan. Dapat diamati pada Gambar 3.9 koordinat titik kiri atas terbentuk dari nilai x minimal dan y minimal segitiga, titik kanan atas terbentuk dari nilai x maksimal dan nilai y minimal segitiga, titik kiri bawah terbentuk dari nilai x minimal dan nilai y maksimal segitiga, dan titik kanan bawah terbentuk dari nilai x maksimal dan nilai y maksimal segitiga.

Sebuah nilai perlu didefinisikan sebagai batas yang menentukan sebuah bangun datar dengan luas *bounding rectangle* tertentu dapat dijadikan bagian soal atau tidak. Pada skripsi ini nilai 2500 $pixel^2$ digunakan sebagai batas untuk menentukan sebuah bagian gambar dapat dijadikan bagian soal atau tidak. Jika luas *bounding rectangle* suatu bangun datar lebih kecil dari nilai yang didefinisikan, bangun datar tersebut tidak dijadikan bagian soal. Sebaliknya, jika suatu bangun datar memiliki luas *bounding rectangle* sama dengan atau lebih besar dari nilai yang didefinisikan, bangun datar tersebut dapat dijadikan bagian soal.

Sebuah garis vertikal dan horizontal hanya memiliki atribut panjang. Nilai 2500 $pixel^2$ terlalu besar untuk dijadikan sebagai batas penentu sebuah garis vertikal atau horizontal dapat dijadikan bagian soal atau tidak. Pada skripsi ini nilai 100 *pixel* digunakan untuk menentukan sebuah garis vertikal atau horizontal dapat dijadikan bagian soal atau tidak.

3.1.6 Analisis Cara Menangani Bagian Gambar yang Bertumpukan

Sebuah gambar dapat terdiri atas beberapa bangun datar. Dua atau lebih bangun datar dapat bertumpukan sehingga menghasilkan perpotongan seperti pada Gambar 3.10. Titik potong yang dihasilkan dari perpotongan tersebut dijadikan *vertex* pada *graph* untuk disertakan pada pencarian *euler path*.



Gambar 3.10: Contoh bagian gambar yang bertumpukan

Setelah sebuah gambar diubah menjadi *graph*, *edge* pada *graph* tersebut dapat dianggap sebagai ruas garis dengan kedua vertex yang dihubungkannya sebagai titik awal dan titik akhir ruas garis. Setiap *edge* pada *graph* dipasangkan dengan *edge* lain yang ada pada *graph* untuk diuji perpotongan antara dua buah *edge* tersebut. Jika terdapat dua buah *edge* atau ruas garis yang berpotongan, kemiringan masing-masing ruas garis perlu diketahui untuk mengetahui titik potong antara dua buah ruas garis. Kemiringan ruas garis dapat diperoleh menggunakan rumus kemiringan seperti pada Rumus 3.9.

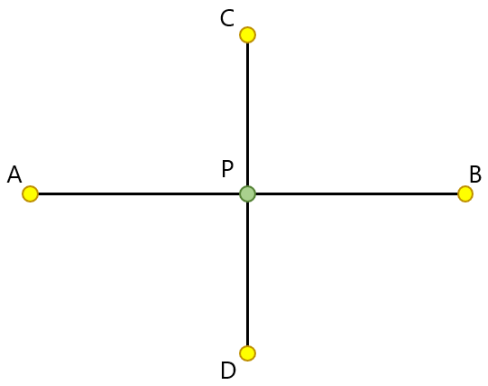
$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.9}$$

Setelah kemiringan ruas garis diketahui, hal selanjutnya yang dilakukan adalah mendefinisikan persamaan ruas garis tersebut. Persamaan sebuah ruas garis dapat diperoleh dengan mendefinisikan nilai x1, y1, dan m pada Rumus 3.10. Setelah hal tersebut dilakukan, terbentuk persamaan ruas garis $y = m\,x + b$.

$$y - y_1 = m\,(x - x_1) \tag{3.10}$$

Persamaan dua buah ruas garis, $y = m_1\,x + b_1$ dan $y = m_2\,x + b_2$, dibandingkan untuk mendapatkan koordinat x titik potong. Setelah koordinat x tersebut diperoleh, nilai tersebut dimasukkan pada salah satu persamaan ruas garis sehingga diperoleh koordinat y. Setelah sebuah titik potong ditemukan, proses pencarian perpotongan pada *graph* dilakukan dari awal untuk menemukan perpotongan lain yang ada.

Edge baru yang melibatkan titik potong yang dihasilkan dari sebuah perpotongan perlu ditambahkan pada *graph*. Pada Gambar 3.11 *edge* A-B berpotongan dengan *edge* C-D. Setelah ditemukannya titik potong dari perpotongan tersebut, yaitu titik P, *edge* baru A-P, P-B, C-P, dan P-D ditambahkan pada *graph*. Setelah keempat *edge* baru tersebut ditambahkan pada *graph*, *edge* A-B dan C-D dilepaskan dari *graph*.



Gambar 3.11: Contoh perpotongan pada *graph*

3.2 Analisis Cara Mengubah *Graph* Menjadi *Euler Graph*

Tidak semua gambar dapat dibuat tanpa mengangkat pensil. Beberapa gambar dibuat dengan mengangkat pensil satu kali atau lebih. Namun, pada sebuah gambar terdapat bagian gambar yang dapat dibuat tanpa mengangkat pensil.

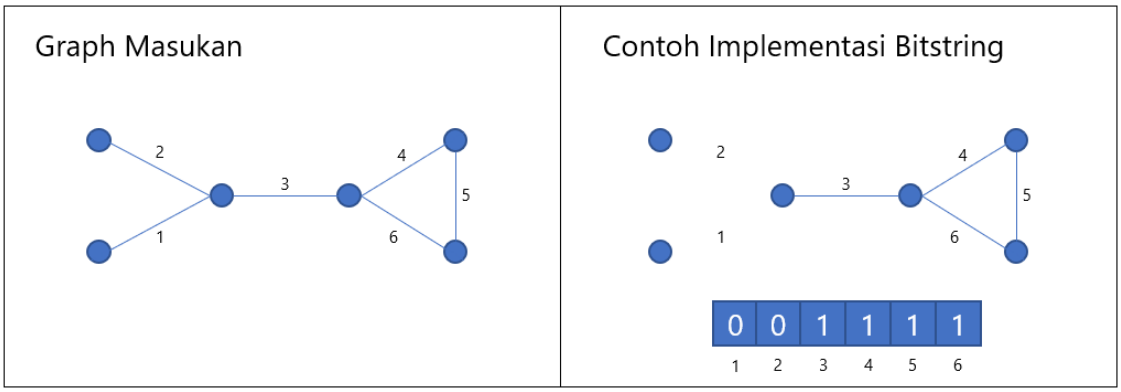
Pada perangkat lunak yang dikembangkan gambar masukan dimodelkan menjadi sebuah *graph*. Sebuah gambar yang dapat dibuat tanpa mengangkat pensil dapat dimodelkan menjadi *euler graph* yang memiliki *euler path* atau *euler circuit*. Tidak semua gambar dapat dimodelkan menjadi *euler graph* sehingga *graph* hasil pemodelan gambar masukan yang bukan merupakan *euler graph* harus

diubah menjadi *euler graph*.

Pada *Euler graph* semua *vertex* memiliki *degree* berjumlah genap atau dua buah *vertex* memiliki *degree* berjumlah ganjil. *Euler graph* yang tidak memiliki *vertex* dengan jumlah *degree* ganjil memiliki *euler circuit*, sedangkan *euler graph* yang memiliki dua buah *vertex* dengan jumlah *degree* ganjil memiliki *euler path*. Berdasarkan pernyataan sebelumnya, sebuah *graph* dapat diubah menjadi *euler graph* dengan cara melepaskan *edge* pada *graph* hingga jumlah *vertex* pada *graph* yang memiliki *degree* berjumlah ganjil adalah dua atau tidak ada sama sekali.

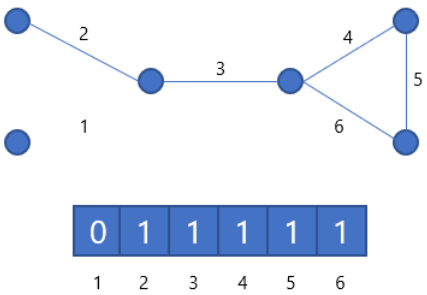
Hasil optimal yang diharapkan dari proses mengubah *graph* menjadi *euler graph* adalah sebuah *euler graph* dengan jumlah *edge* terbanyak. Jumlah *edge* yang dilepaskan dari *graph* untuk mengubah *graph* menjadi *euler graph* diharapkan minimal. Pendekatan *brute force* dapat menghasilkan hasil yang optimal untuk permasalahan ini. Pendekatan *brute force* mencoba semua kemungkinan cara untuk melepaskan *edge* pada *graph* untuk mengubah *graph* tersebut menjadi *euler graph*.

Bitstring adalah sebuah deret bilangan yang masing-masing bilangan pada deret tersebut bernilai nol atau satu. *Bitstring* merupakan implementasi dari pendekatan *brute force* yang dapat digunakan untuk menyelesaikan permasalahan mengubah *graph* menjadi *euler graph*. Panjang *bitstring* disesuaikan dengan banyaknya *edge* pada *graph*. Setiap *bit* pada *bitstring* menandakan sebuah *edge* merupakan anggota dari *euler graph* atau bukan. Pada Gambar 3.12 *edge* nomor 3 hingga *edge* nomor 6 memiliki *bit* bernilai satu sehingga *edge* nomor 3 hingga *edge* nomor 6 merupakan anggota dari *euler graph*.



Gambar 3.12: Contoh penggunaan *bitstring* untuk mengubah *graph* menjadi *euler graph*

Seluruh kemungkinan *bitstring* dihasilkan kemudian *bitstring* yang paling optimal dipilih sebagai solusi. *Bitstring* yang paling optimal merepresentasikan *euler graph* dengan jumlah *edge* terbanyak. Misalkan *graph* masukan seperti pada Gambar 3.12, solusi optimal yang didapatkan adalah seperti pada Gambar 3.13.



Gambar 3.13: Contoh solusi optimal permasalahan mengubah *graph* menjadi *euler graph*

Kompleksitas waktu pendekatan *brute force* pada permasalahan ini sangat besar sehingga pendekatan *brute force* tidak cocok untuk diimplementasikan pada perangkat lunak yang dikembangkan.

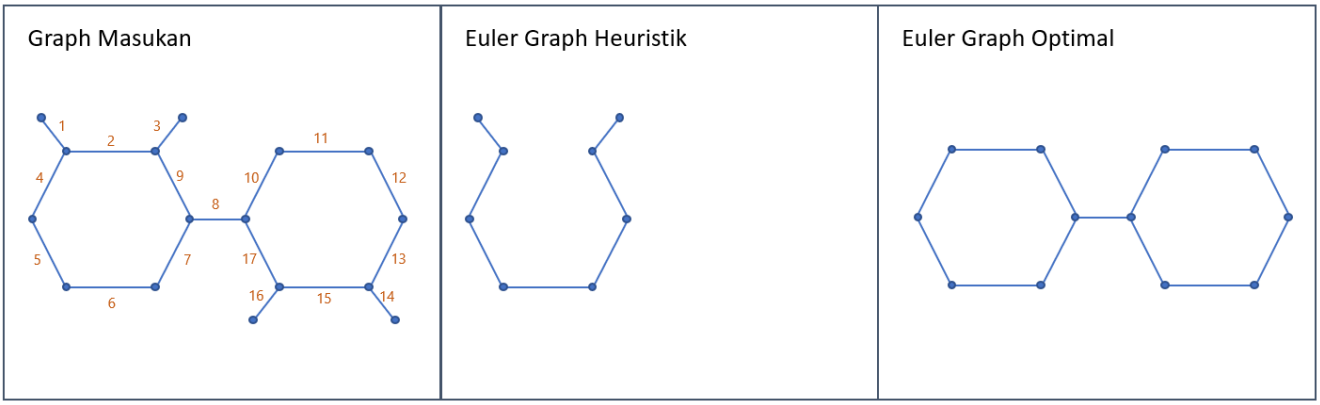
Pada skripsi ini digunakan sebuah heuristik yang dapat menghasilkan solusi optimal hampir di setiap kasus pada permasalahan ini. Heuristik yang digunakan dapat menyelesaikan permasalahan mengubah *graph* menjadi *euler graph* dengan kompleksitas waktu yang jauh lebih kecil dibandingkan pendekatan *brute force*.

Sebuah *edge* menghubungkan dua buah *vertex*. Pada heuristik ini prioritas melepaskan *edge* yang menghubungkan dua buah *vertex* dengan jumlah *degree* ganjil lebih besar dibandingkan melepaskan *edge* yang menghubungkan *vertex* dengan jumlah *degree* ganjil dan *vertex* dengan jumlah *degree* genap. Proses melepaskan *edge* dari *graph* terus dilakukan hingga tidak ada *vertex* yang memiliki *degree* berjumlah ganjil atau hanya dua buah *vertex* yang memiliki *degree* berjumlah ganjil pada *graph*.

Sebuah *edge* pada *graph* dapat berupa *bridge*. Jika *edge* yang merupakan *bridge* dilepaskan dari *graph*, *graph* tersebut akan terbagi menjadi beberapa bagian terhubung. Heuristik yang digunakan pada skripsi ini memprioritaskan proses melepaskan *edge* yang bukan merupakan *bridge* dibandingkan melepaskan *edge* yang merupakan *bridge*. Proses melepaskan *edge* yang merupakan *bridge* menyebabkan *graph* menjadi tidak terhubung sehingga setiap *edge* pada bagian terhubung yang lebih kecil juga ikut dilepaskan untuk menjaga *graph* agar tetap terhubung.

Edge yang dilepaskan dari *graph* dijadikan garis bantuan pada soal yang dihasilkan. Setiap *vertex* yang memiliki *degree* berjumlah nol dihilangkan dari soal. Setelah *graph* hasil memodelkan gambar masukan diubah menjadi *euler graph*, algoritma Hierholzer diterapkan pada *graph* untuk memberi nomor urut penelusuran pada setiap *vertex* pada soal.

Terdapat beberapa kasus yang menyebabkan heuristik tidak menghasilkan solusi optimal. Pada Gambar 3.14 solusi yang dihasilkan oleh heuristik tidak optimal. Hal tersebut dikarenakan prioritas melepaskan *edge* yang bukan merupakan *bridge* lebih besar dibandingkan melepaskan *edge* yang merupakan *bridge*.



Gambar 3.14: Contoh kasus heuristik tidak menghasilkan solusi optimal

Pertama-tama *edge* nomor 2 yang menghubungkan dua buah *vertex* berjumlah *degree* ganjil dilepaskan dari *graph*. *Edge* nomor 1, 3, 14, dan 16 belum dilepaskan dari *graph* karena *edge* tersebut merupakan *bridge*. *Edge* yang dilepaskan selanjutnya adalah *edge* nomor 15 kemudian *edge* nomor 8. Pada kasus ini, dua buah *graph* terhubung yang terbentuk dari proses melepaskan *edge* nomor 8 yang merupakan *bridge* pada *graph* sama besar sehingga *graph* terhubung yang dipilih menjadi bagian soal adalah *graph* yang pertama didefinisikan pada dokumen SVG.

BAB 4

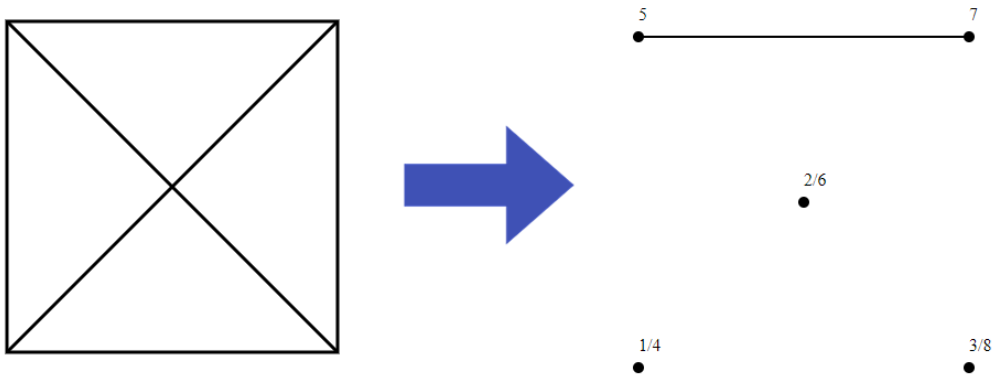
PERANCANGAN

Sebelum implementasi dilakukan, berbagai perancangan dibuat untuk menunjang proses implementasi. Perancangan yang dibuat pada skripsi ini adalah perancangan input, output, antarmuka, perangkat lunak, dan pengujian. Pada bagian ini akan dijelaskan secara detil mengenai perancangan-perancangan tersebut.

4.1 Perancangan Input dan Output

Masukan perangkat lunak yang dikembangkan adalah gambar SVG kemudian perangkat lunak akan menghasilkan keluaran berupa soal permainan menghubungkan titik. Gambar masukan merupakan gambar yang dapat diubah menjadi sebuah *graph* terhubung. Soal permainan menghubungkan titik yang dihasilkan perangkat lunak dapat dikerjakan tanpa mengangkat pensil dan memiliki garis bantuan minimal sehingga soal yang dihasilkan lebih menarik. Namun, terdapat beberapa kasus yang menyebabkan perangkat lunak tidak menghasilkan soal permainan menghubungkan titik yang memiliki garis bantuan minimal sesuai dengan analisis pada Bagian 3.2.

Keluaran program akan direpresentasikan dalam bentuk SVG. Pada soal permainan menghubungkan titik terdapat bagian soal dan garis bantuan. Garis bantuan pada soal yang dihasilkan perangkat lunak direpresentasikan menggunakan elemen garis, sedangkan pada bagian soal titik direpresentasikan menggunakan elemen lingkaran yang memiliki radius lima *pixel*. Setiap titik pada bagian soal memiliki angka yang menunjukan urutan penelusuran untuk menghubungkan titik-titik pada soal. Angka tersebut direpresentasikan menggunakan elemen teks. Gambar 4.1 merupakan contoh soal yang akan dihasilkan perangkat lunak.



Gambar 4.1: Rancangan soal yang dihasilkan perangkat lunak

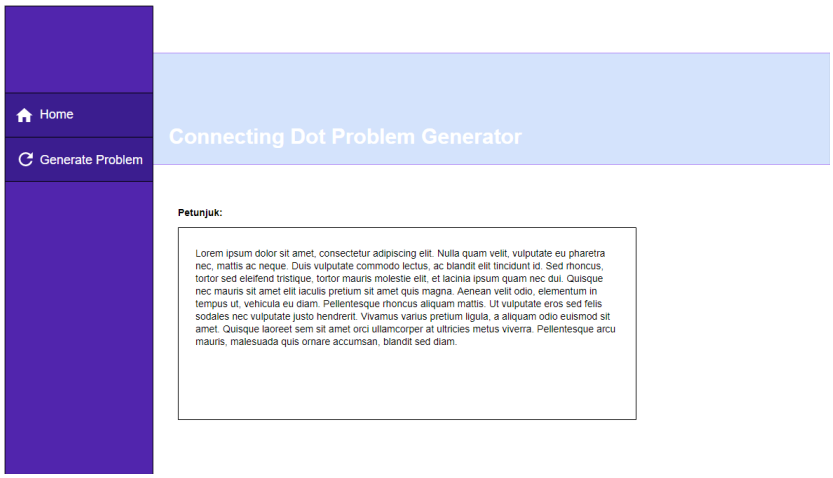
Elemen SVG memiliki atribut *width* dan *height* yang menentukan luas bidang untuk menampilkan gambar. Luas bidang yang digunakan untuk menampilkan gambar keluaran disesuaikan dengan luas bidang yang digunakan untuk menampilkan gambar masukan agar soal permainan yang dihasilkan memuat seluruh bagian gambar masukan. Elemen SVG yang dihasilkan perangkat lunak memiliki nilai *width* dan *height* yang sama dengan nilai *width* dan *height* elemen SVG masukan.

4.2 Perancangan Antarmuka

Perangkat lunak yang dikembangkan pada skripsi ini berfungsi untuk menghasilkan soal permainan menghubungkan titik yang berasal dari gambar SVG masukan. Pemilihan gambar SVG sebagai masukan perangkat lunak dilakukan melalui proses jelajah. Setelah masukan diterima oleh perangkat lunak, perangkat lunak akan meminta lokasi penyimpanan dan nama berkas soal permainan menghubungkan titik yang dihasilkan.

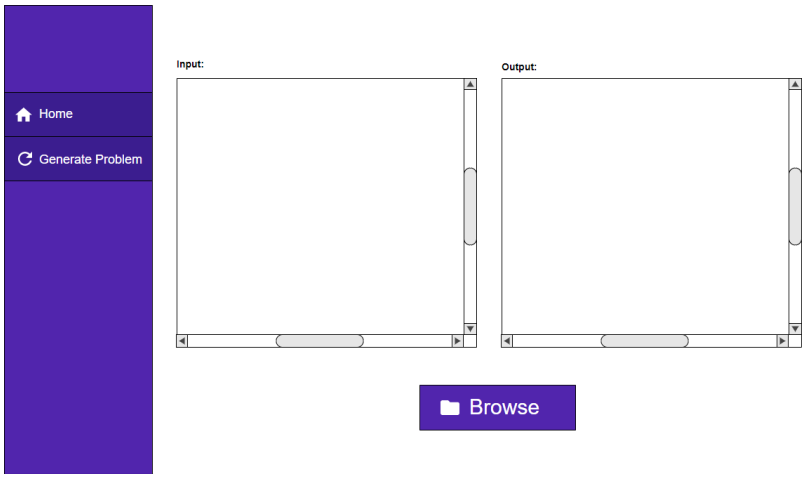
Pada antarmuka perangkat lunak terdapat bagian navigasi. Bagian navigasi memuat daftar halaman yang terdapat pada perangkat lunak. Pada perangkat lunak terdapat dua halaman, yaitu, halaman beranda dan halaman membuat soal.

Gambar 4.2 merupakan rancangan antarmuka halaman beranda. Pada halaman beranda terdapat kolom petunjuk yang berfungsi untuk menampilkan cara menggunakan perangkat lunak. Kolom petunjuk ini bertujuan membimbing pengguna dalam menggunakan perangkat lunak.



Gambar 4.2: Rancangan antarmuka halaman beranda

Pengguna berpindah ke halaman membuat soal untuk membuat soal permainan menghubungkan titik. Gambar 4.3 merupakan rancangan antarmuka halaman membuat soal. Pada halaman membuat soal terdapat tombol jelajah yang berfungsi untuk membuka *file chooser*. Pengguna kemudian dapat memilih berkas SVG yang ingin dijadikan sebagai masukan. Setelah perangkat lunak menerima berkas masukan, perangkat lunak akan meminta lokasi penyimpanan untuk soal yang dihasilkan.



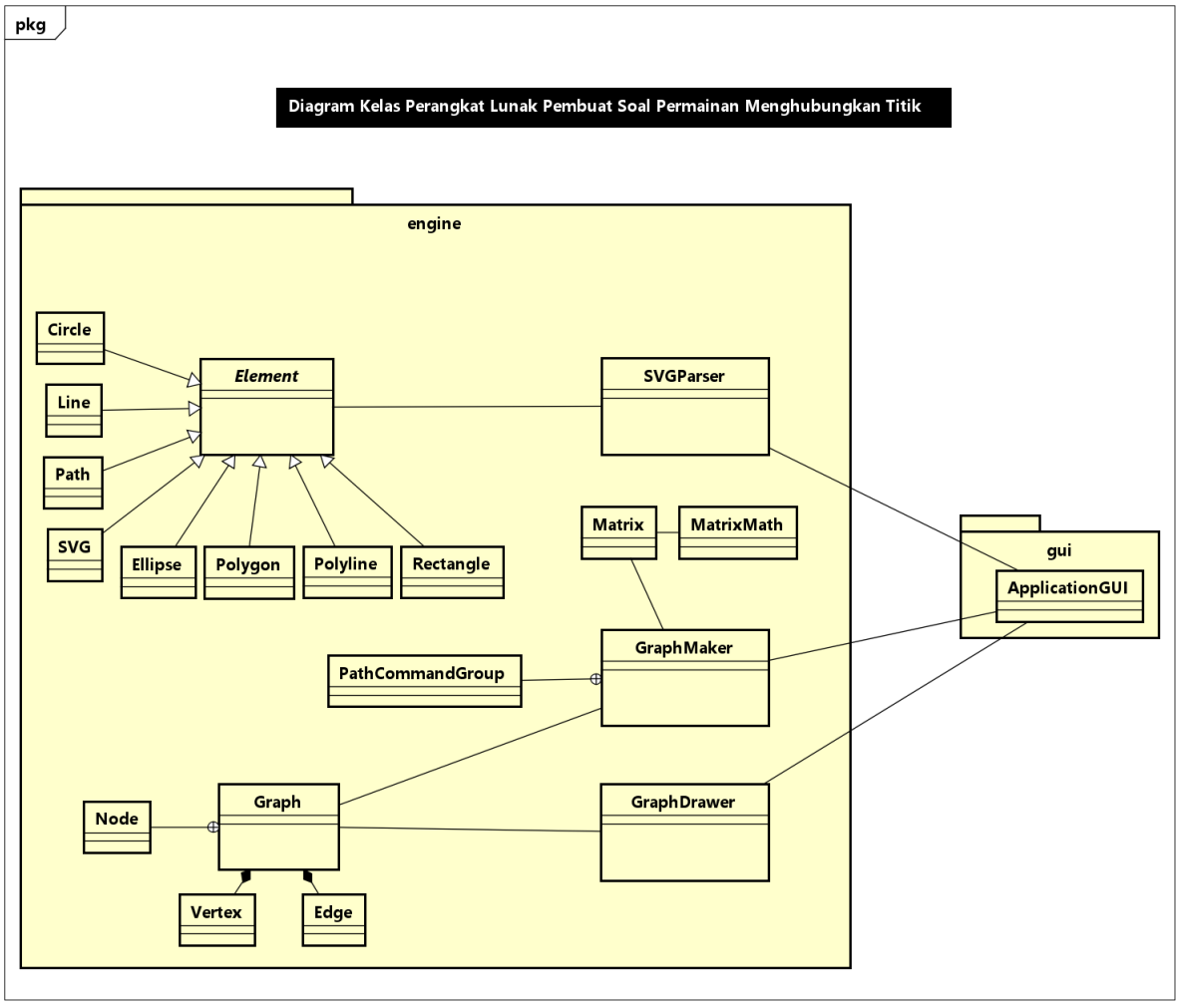
Gambar 4.3: Rancangan antarmuka halaman membuat soal

Pratinjau masukan dan keluaran akan ditampilkan pada halaman membuat soal setelah soal berhasil dibuat dan disimpan. Hal tersebut bertujuan untuk memberi gambaran pada pengguna mengenai perbandingan masukan dan keluaran yang dihasilkan oleh perangkat lunak. Pada pratinjau terdapat *scroll bar* sehingga pengguna dapat melihat gambar secara menyeluruh.

4.3 Perancangan Perangkat Lunak

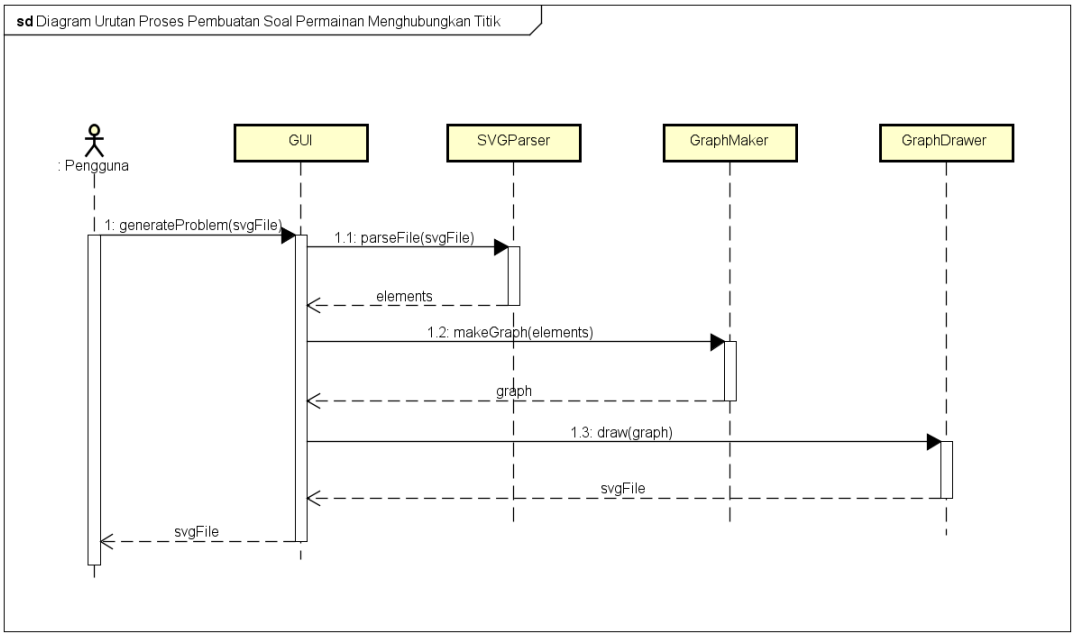
Sebelum implementasi pada perangkat lunak dilakukan, diagram kelas dirancang terlebih dahulu. Perancangan diagram kelas bertujuan untuk menentukan objek-objek yang terlibat dalam proses mengubah gambar SVG menjadi soal permainan menghubungkan titik. Hubungan antar objek juga didefinisikan dalam diagram kelas.

Gambar 4.4 merupakan diagram kelas perangkat lunak pembuat soal permainan menghubungkan titik yang telah disederhanakan. Pada diagram kelas tersebut terdapat dua buah *package*, yaitu, *package* ENGINE dan GUI. *Package* ENGINE berisi kelas-kelas yang berperan dalam komputasi mengubah gambar SVG menjadi soal permainan menghubungkan titik dan *package* GUI berisi sebuah kelas yang berperan dalam mengatur antarmuka perangkat lunak.



Gambar 4.4: Diagram kelas yang disederhanakan

Terdapat serangkaian proses dalam mengubah gambar SVG menjadi soal permainan menghubungkan titik. Gambar 4.5 merupakan diagram urutan proses pembuatan soal permainan menghubungkan titik. Pada diagram tersebut terdapat tiga buah proses, yaitu, proses *parsing* berkas SVG, membuat *graph*, dan menggambar soal permainan menghubungkan titik.

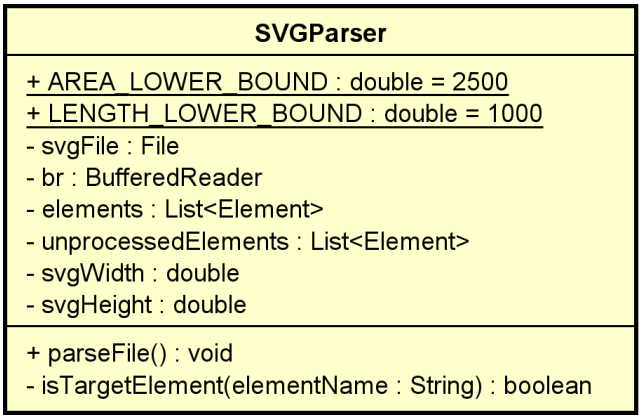


Gambar 4.5: Diagram urutan proses pembuatan soal menghubungkan titik

4.3.1 Kelas SVGPARSER

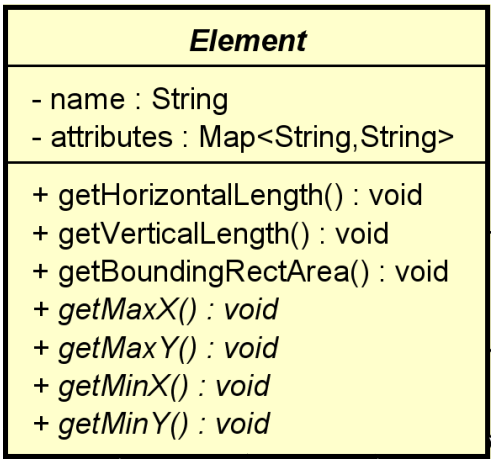
Kelas SVGPARSER berfungsi untuk melakukan proses *parsing* terhadap berkas SVG masukan. Hal yang pertama kali dilakukan oleh kelas SVGPARSER adalah membaca berkas SVG masukan. Pada kelas SVGPARSER terdapat atribut BR yang berfungsi untuk membaca berkas SVG.

Berkas SVG memuat berbagai elemen. Setelah membaca berkas, SVGPARSER akan menentukan elemen mana saja yang akan diproses lebih lanjut dan yang tidak. Elemen-elemen yang memiliki luas *bounding rectangle* lebih kecil dibandingkan batas yang ditentukan tidak akan diproses lebih lanjut. Gambar 4.6 memaparkan atribut dan metode yang dimiliki kelas SVGPARSER.



Gambar 4.6: Kelas SVGPARSER

Hasil dari proses *parsing* berkas SVG masukan adalah sejumlah elemen. Gambar 4.7 merupakan model dari kelas abstrak elemen. Kelas-kelas yang merepresentasikan bentuk-bentuk dasar pada SVG, *path*, dan elemen SVG merupakan generalisasi dari kelas abstrak elemen seperti ditunjukkan pada Gambar 4.4.

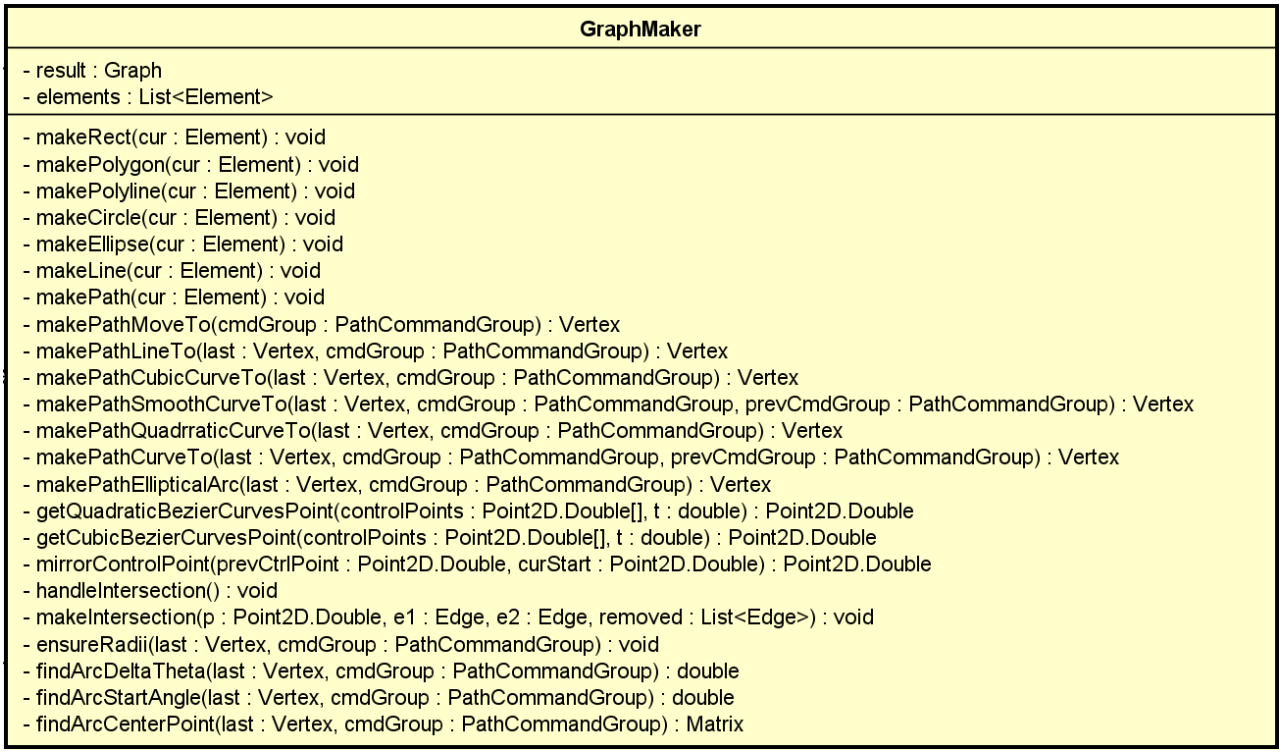


Gambar 4.7: Kelas Abstrak ELEMENT

Atribut dan nilai atribut dari setiap elemen disimpan dalam bentuk *hash map* dengan tujuan mempermudah pengaksesan nilai atribut suatu elemen. *Hash map* tersebut memiliki *key* berupa nama atribut dan *value* berupa nilai atribut. Setiap kelas yang merupakan generalisasi dari kelas abstrak elemen juga mengimplementasikan metode GETBOUNDINGRECTAREA dengan implementasi yang berbeda-beda.

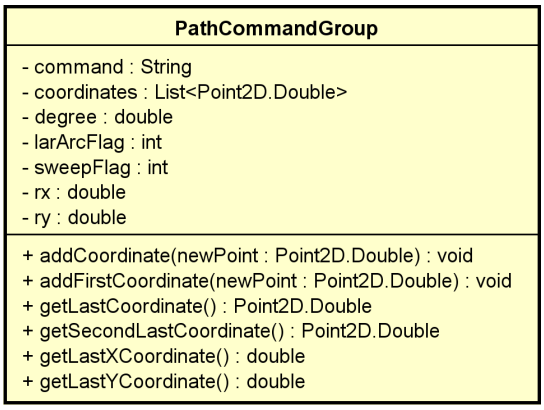
4.3.2 Kelas GRAPHMAKER

Kelas GRAPHMAKER merupakan sebuah kelas yang berperan dalam menghasilkan *graph* berdasarkan elemen-elemen yang dihasilkan oleh SVGPARSER. Gambar 4.8 adalah model dari kelas GRAPHMAKER. Pada kelas GRAPHMAKER terdapat berbagai operasi yang digunakan untuk mengubah berbagai jenis elemen menjadi bagian *graph*.



Gambar 4.8: Kelas GRAPHMAKER

Kelas PATHCOMMANDGROUP merupakan *inner class* dari kelas GRAPHMAKER. Kelas tersebut berfungsi untuk mengelompokkan sebuah perintah *path* menjadi sebuah objek. Koordinat-koordinat yang terdapat pada sebuah perintah *path* disimpan sebagai atribut pada kelas PATHCOMMANDGROUP. Gambar 4.9 merupakan model dari kelas PATHCOMMANDGROUP.

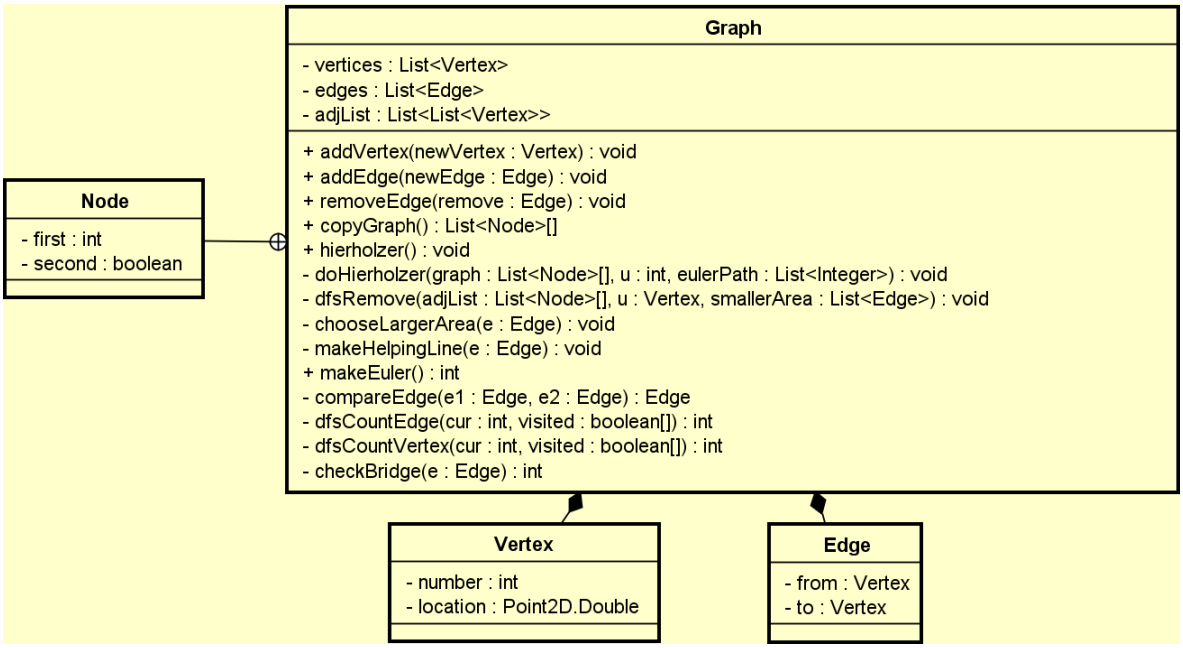


Gambar 4.9: Kelas PATHCOMMANDGROUP

Gambar yang bertumpukan ditangani oleh kelas GRAPHMAKER sesuai dengan analisis pada Bagian 3.1.6. Setiap titik potong pada gambar yang bertumpukan dijadikan *vertex* menggunakan metode MAKEINTERSECTION. Hasil dari proses mengubah elemen-elemen yang ada menjadi sebuah *graph* disimpan sebagai atribut.

Kelas GRAPH terdiri dari himpunan *vertex* dan himpunan *edge*. Kelas NODE merupakan *inner class* dari kelas GRAPH. Pada implementasi algoritma Hierholzer, kelas NODE digunakan sebagai representasi lain dari *vertex*. Gambar 4.10 merupakan model dari kelas GRAPH beserta kelas-kelas yang berhubungan dengan kelas GRAPH.

Algoritma 1 diimplementasikan dalam metode DOHIERHOLZER pada kelas GRAPH. Sebelum algoritma Hierholzer diterapkan pada *graph*, *graph* tersebut diubah menjadi *euler graph* terlebih dahulu menggunakan metode MAKEEULER sesuai dengan analisis pada Bagian 3.2. Metode HIERHOLZER berfungsi untuk memastikan metode MAKEEULER diterapkan sebelum mengoperasikan metode DOHIERHOLZER.



Gambar 4.10: Kelas GRAPH dan kelas-kelas yang berhubungan dengan kelas GRAPH

Pada kelas GRAPH terdapat metode CHECKBRIDGE. Sebuah *bridge* jika dilepaskan dari *graph* akan membagi *graph* tersebut menjadi dua buah *graph* terhubung. Metode CHECKBRIDGE mengembalikan nilai -1 jika *edge* argumen bukan merupakan *bridge* dan mengembalikan nilai sebesar jumlah *edge* pada *graph* terhubung yang memiliki jumlah *edge* terbanyak jika *edge* argumen merupakan *bridge*. Algoritma 2 merupakan pseudocode metode CHECKBRIDGE.

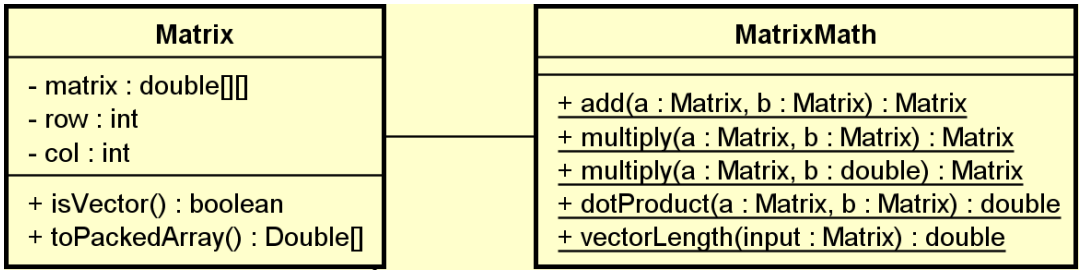
Algorithm 2 Check Bridge

Require:
adjList, lis ketetanggaan yang merepresentasikan sebuah *graph*
e, sebuah *edge* pada *graph*
visited, sebuah *array of boolean* yang memiliki ukuran sesuai dengan jumlah *vertex* pada *graph* untuk menandakan penelusuran

Ensure:
mengembalikan nilai -1 jika e bukan merupakan *bridge* dan jumlah *edge* pada *graph* terhubung yang memiliki jumlah *edge* terbanyak jika e merupakan *bridge*

```
function CHECKBRIDGE(adjList, e, visited)
    reset(visited)
    before ← DFSCOUNTVERTEX(e.firstVertex, adjList, visited)
    adjList[e.firstVertex].remove(e.secondVertex)
    adjList[e.secondVertex].remove(e.firstVertex)
    reset(visited)
    after ← DFSCOUNTVERTEX(e.firstVertex, adjList, visited)
    reset(visited)
    firstConnectedGraph ← DFSCOUNTEDGE(e.firstVertex, adjList, visited)
    reset(visited)
    secondConnectedGraph ← DFSCOUNTEDGE(e.secondVertex, adjList, visited)
    adjList[e.firstVertex].add(e.secondVertex)
    adjList[e.secondVertex].add(e.firstVertex)
    if before = after then
        return -1
    else
        return MAXIMUM(firstConnectedGraph, secondConnectedGraph)
    end if
end function
```

Kelas MATRIX dan MATRIXMATH membantu kelas GRAPHMAKER dalam proses mengubah busur elips menjadi bagian *graph*. Kelas MATRIX merepresentasikan sebuah *matrix* dan kelas MATRIXMATH menyediakan operasi-operasi yang dilakukan pada *matrix*. Gambar 4.11 merupakan model dari kelas MATRIX dan MATRIXMATH.

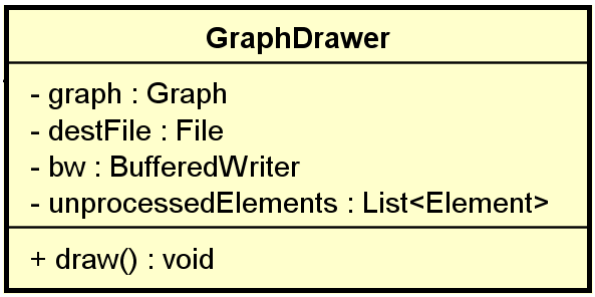


Gambar 4.11: Kelas MATRIX dan kelas MATRIXMATH

4.3.3 Kelas GRAPHDRAWER

Setelah *graph* yang merepresentasikan soal permainan menghubungkan titik terbentuk, kelas GRAPHDRAWER bertugas untuk menghasilkan sebuah berkas yang menjadi keluaran perangkat lunak. Kelas GRAPHDRAWER memiliki atribut BW yang berfungsi untuk menuliskan *graph* yang merepresentasikan soal permainan menghubungkan titik pada berkas keluaran dalam format SVG. Gambar 4.12 merupakan model dari kelas GRAPHDRAWER.

Pada proses *parsing* terdapat elemen-elemen yang tidak diproses lebih lanjut. Elemen-elemen tersebut akan langsung dituliskan pada berkas keluaran oleh GRAPHDRAWER. Metode DRAW pada kelas GRAPHDRAWER berfungsi untuk menuliskan *graph* yang merepresentasikan soal permainan menghubungkan titik dan elemen-elemen yang tidak diproses pada berkas keluaran dalam format SVG.



Gambar 4.12: Kelas GRAPHDRAWER

4.4 Perancangan Pengujian

Pengujian yang dilakukan terhadap perangkat lunak yang dikembangkan pada skripsi ini adalah pengujian fungsional. Pengujian fungsional bertujuan untuk memastikan perangkat lunak bekerja sesuai dengan yang diharapkan. Perancangan kasus masukan diperlukan untuk memastikan perangkat lunak teruji secara menyeluruh.

Terdapat dua buah cara yang dapat digunakan untuk mendefinisikan sebuah bentuk pada SVG, yaitu menggunakan bentuk dasar atau menggunakan *path*. Jika luas *bounding rectangle* suatu elemen lebih kecil dari batas yang telah ditentukan, elemen tersebut tidak dijadikan bagian soal. Pada skripsi ini pengujian dibagi menjadi empat bagian pengujian berdasarkan sasaran pengujian, yaitu:

1. Pengujian bentuk dasar
   Terdapat enam buah bentuk dasar yang dapat digunakan pada SVG, yaitu, persegi panjang, lingkaran, elips, garis, *polyline*, dan poligon. Enam buah bentuk dasar tersebut memiliki implementasi yang berbeda-beda. Pengujian setiap bentuk dasar dilakukan pada tiga kasus berbeda, yaitu, ketika bentuk dasar memiliki luas *bounding rectangle* lebih kecil, sama dengan, dan lebih besar dibandingkan dengan batas yang ditentukan.

2. Pengujian *path*
   Terdapat sepuluh perintah yang dapat digunakan untuk mendefinisikan sebuah *path*, yaitu, *moveto*, *lineto*, *horizontal lineto*, *vertical lineto*, *quadratic Bezier curveto*, *shorthand/smooth quadratic Bezier curveto*, *cubic Bezier curveto*, *shorthand/smooth cubic Bezier curveto*, *elliptical arc* dan *close path*. Sepuluh perintah tersebut memiliki implementasi yang berbeda-beda. Pengujian setiap perintah *path*, kecuali perintah *close path*, dilakukan pada tiga kasus berbeda, yaitu, ketika *path* yang terbentuk memiliki luas *bounding rectangle* lebih kecil, sama dengan, dan lebih besar dibandingkan dengan batas yang ditentukan. Perintah *close path* tidak mempengaruhi luas *bounding rectangle* sebuah *path*.

3. Pengujian pada kasus gambar yang bertumpukan
   Pengujian ini bertujuan untuk menguji kebenaran perangkat lunak dalam menangani bagian gambar yang bertumpukan. Setiap titik potong pada bagian gambar yang bertumpukan akan dijadikan *vertex* pada *graph*. Pengujian terhadap bagian gambar yang bertumpukan hanya dilakukan pada bagian gambar yang memiliki luas *bounding rectangle* lebih besar atau sama dengan batas yang telah ditentukan.

4. Pengujian pada kasus *graph* tak terhubung
   Masukan perangkat lunak adalah gambar SVG yang dapat diubah menjadi sebuah *graph* terhubung. Algoritma Hierholzer tidak berjalan dengan baik pada *graph* yang tidak terhubung. Tujuan pengujian ini adalah mengetahui keluaran perangkat lunak jika masukan berupa gambar yang tidak dapat diubah menjadi sebuah *graph* terhubung dan memahami cara kerja perangkat lunak dalam menanganinya.

5. Pengujian implementasi algoritma Hierholzer
   Algoritma Hierholzer berperan dalam menentukan bentuk soal permainan menghubungkan titik. Untuk menguji kebenaran implementasi algoritma Hierholzer, pengerjaan soal secara manual diperlukan. Kebenaran implementasi algoritma Hierholzer teruji jika soal permainan menghubungkan titik yang dihasilkan perangkat lunak berhasil dikerjakan. Pengerjaan soal secara manual akan dilakukan terhadap setiap kasus uji yang dijadikan sebagai masukan perangkat lunak.

BAB 5

IMPLEMENTASI DAN PENGUJIAN

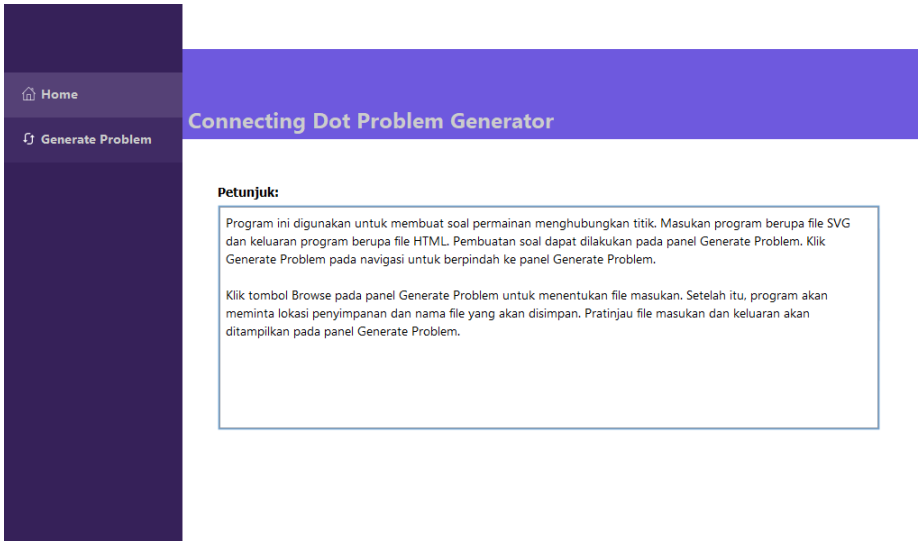
Bab ini membahas tentang implementasi perangkat lunak dan pengujian yang dilakukan terhadap perangkat lunak tersebut. Spesifikasi perangkat lunak dan implementasi antarmuka perangkat lunak akan dipaparkan pada bab ini. Pengujian yang dilakukan pada skripsi ini akan dibahas secara detil pada bab ini.

5.1 Implementasi

Perangkat lunak dikembangkan menggunakan bahasa pemrograman Java. Java merupakan bahasa pemrograman yang berorientasi objek sehingga dapat memudahkan proses implementasi permasalahan mengubah gambar SVG menjadi soal permainan menghubungkan titik. Aplikasi yang dikembangkan menggunakan bahasa pemrograman Java dapat berjalan di setiap komputer yang menjalankan *Java Virtual Machine* (JVM). Gambar 5.1 adalah versi bahasa pemrograman Java yang digunakan untuk mengembangkan perangkat lunak pada skripsi ini.

Gambar 5.1: Versi bahasa pemrograman Java yang digunakan pada skripsi

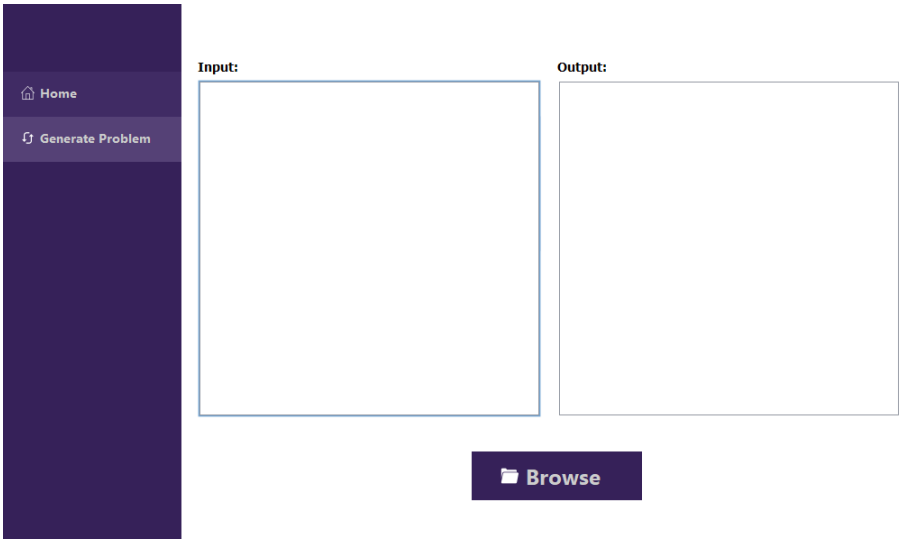
Pada antarmuka perangkat lunak terdapat dua halaman, yaitu halaman beranda dan membuat soal. Antarmuka terbagi menjadi dua bagian, yaitu bagian navigasi dan bagian konten. Bagian navigasi berfungsi untuk menampilkan daftar halaman yang terdapat pada perangkat lunak. Gambar 5.2 merupakan antarmuka perangkat lunak halaman beranda.



Gambar 5.2: Antarmuka perangkat lunak halaman beranda

Antarmuka perangkat lunak diimplementasikan menggunakan Java Swing. Java Swing adalah sebuah *application programming interface* yang disediakan oleh Java. Penggunaan Java Swing bertujuan mempermudah pengembangan *graphical user interface* perangkat lunak.

Gambar 5.2 merupakan antarmuka perangkat lunak halaman membuat soal. Pada halaman membuat soal terdapat pratinjau yang menampilkan masukan dan keluaran perangkat lunak. Pratinjau tersebut diimplementasikan menggunakan JavaFX WebView yang memiliki kemampuan untuk menampilkan berkas SVG.



Gambar 5.3: Antarmuka perangkat lunak halaman membuat soal

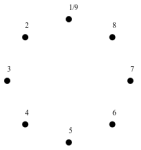
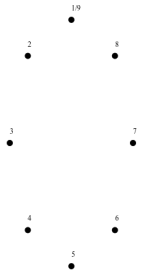
5.2 Pengujian

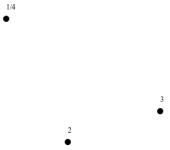
Pengujian yang dilakukan pada skripsi ini bertujuan untuk menguji fungsionalitas perangkat lunak. Pada skripsi ini pengujian dibagi menjadi lima bagian berdasarkan sasaran pengujian. Pengujian yang dilakukan adalah pengujian bentuk dasar, pengujian *path*, pengujian pada kasus gambar yang bertumpukan, pengujian pada kasus *graph* tak terhubung, dan pengujian algoritma Hierholzer.

5.2.1 Pengujian Bentuk Dasar

Terdapat enam buah bentuk dasar yang dapat digunakan pada SVG. Setiap bentuk dasar dilakukan pengujian sebanyak tiga kali, yaitu ketika bentuk dasar memiliki luas *bounding rectangle* lebih kecil, sama dengan, dan lebih besar dibandingkan batas yang ditentukan. Bentuk dasar akan diproses lebih lanjut jika memiliki luas *bounding rectangle* lebih besar atau sama dengan batas yang ditentukan. Tabel 5.1 merupakan hasil pengujian bentuk dasar yang dilakukan.

Tabel 5.1: Hasil pengujian bentuk dasar

Bentuk dasar	Lebih kecil dari batas	Sama dengan batas	Lebih besar dari batas
Persegi panjang			
Lingkaran			
Elips			
Garis			

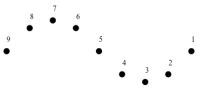
Polyline			
Poligon			

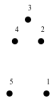
5.2.2 Pengujian SVG *Path*

Terdapat sepuluh perintah *path* yang dapat digunakan pada SVG untuk membentuk sebuah *path*. Pengujian pada *path* dilakukan pada setiap perintah *path* kecuali perintah *close path* yang tidak mempengaruhi luas *bounding rectangle*. Setiap perintah *path* yang diuji, diuji sebanyak tiga kali, yaitu ketika *path* yang terbentuk memiliki luas *bounding rectangle* lebih kecil, sama dengan, dan lebih besar dibandingkan batas yang ditentukan. *Path* yang memiliki luas *bounding rectangle* lebih besar atau sama dengan batas yang ditentukan akan diproses lebih lanjut. Tabel 5.2 merupakan hasil pengujian *path* yang dilakukan.

Tabel 5.2: Hasil pengujian *path*

Perintah *path*	Lebih kecil dari batas	Sama dengan batas	Lebih besar dari batas
M (*moveto*)			
L (*lineto*)			

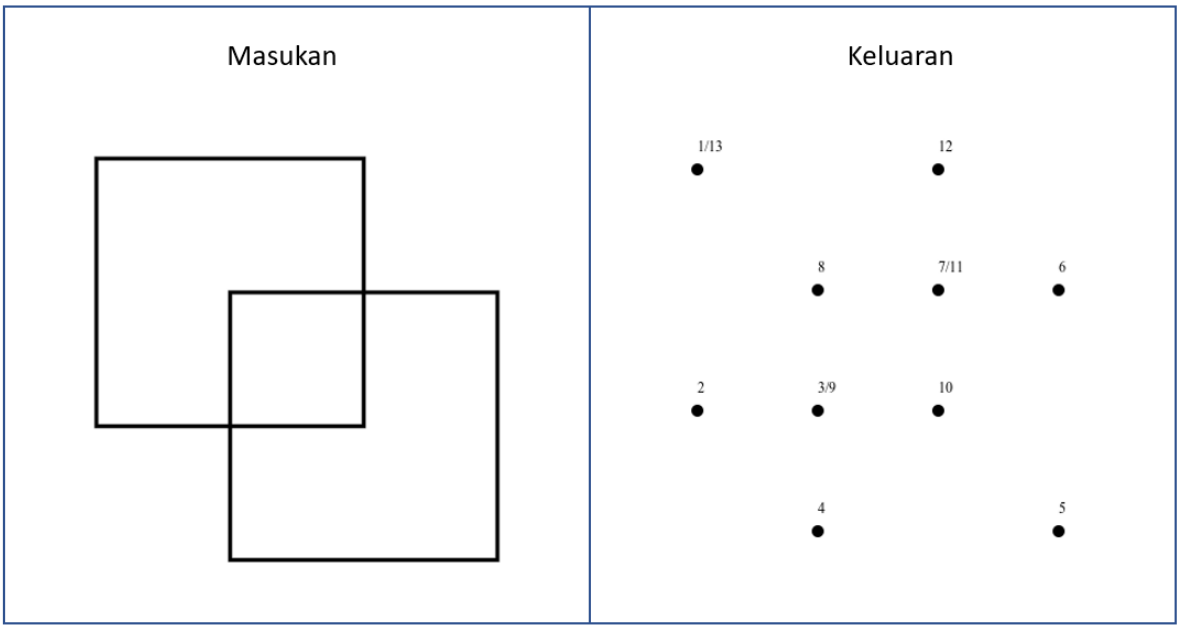
H (*horizontal lineto*)			
V (*vertical lineto*)			
Q (*quadratic Bezier curveto*)			
T (*shorthand/smooth quadratic Bezier curveto*)			
C (*cubic Bezier curveto*)			

S (*shorthand/smooth cubic Bezier curveto*)			
A (*elliptical arc*)			

5.2.3 Pengujian pada Kasus Gambar yang Bertumpukan

Sebuah gambar SVG dapat terdiri atas beberapa bagian gambar. Sebuah bagian gambar dapat bertumpukan dengan bagian gambar lainnya. Setiap titik potong pada bagian gambar yang bertumpukan dijadikan *vertex* pada *graph*.

Pengujian kasus gambar yang bertumpukan dilakukan dengan tiga kasus berbeda. Gambar 5.4 merupakan hasil pengujian pada kasus pertama gambar yang bertumpukan. Gambar masukan kasus pertama adalah dua buah bujur sangkar yang bertumpukan.

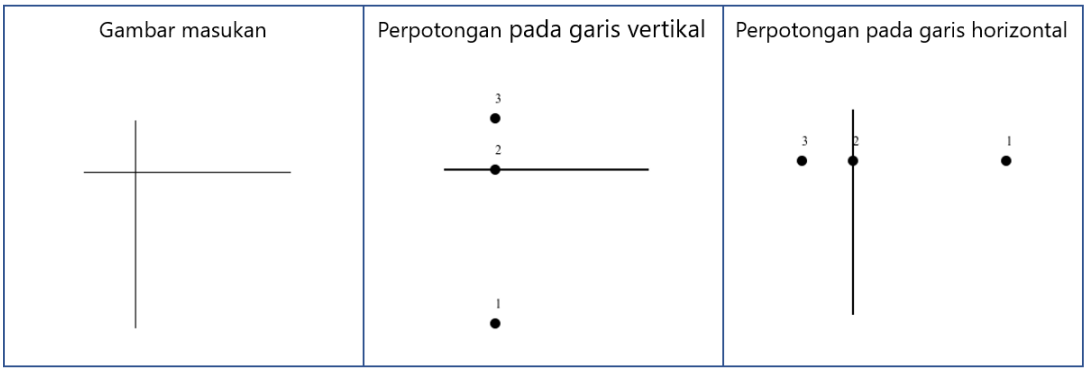


Gambar 5.4: Hasil pengujian pada kasus pertama gambar yang bertumpukan

Koordinat x titik awal dan titik akhir garis vertikal bernilai sama, sedangkan garis horizontal

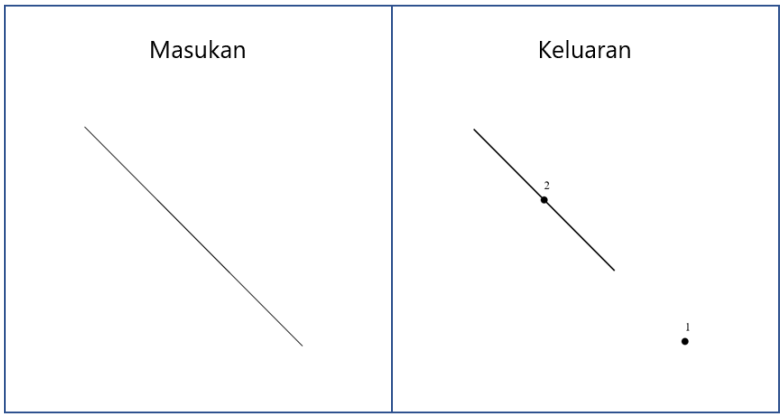
memiliki koordinat y titik awal dan koordinat y titik akhir yang sama. Berdasarkan Rumus 3.9 garis vertikal memiliki kemiringan tak terdefinisi dan garis horizontal memiliki kemiringan sebesar nol. Perpotongan bagian gambar yang melibatkan garis vertikal dan horizontal ditangani secara khusus oleh perangkat lunak.

Kasus pengujian kedua bertujuan untuk menguji perangkat lunak dalam menangani perpotongan yang melibatkan garis horizontal dan garis vertikal. Pengujian pada kasus kedua dilakukan sebanyak dua kali, pertama untuk menguji perpotongan yang melibatkan garis vertikal, dan kedua untuk menguji perpotongan yang melibatkan garis horizontal. Gambar 5.4 merupakan hasil pengujian pada kasus kedua gambar yang bertumpukan.



Gambar 5.5: Hasil pengujian pada kasus kedua gambar yang bertumpukan

Garis yang berimpitan memiliki titik potong lebih dari satu. Analisis pada Bagian 3.1.6 tidak dapat menangani perpotongan pada garis yang berimpitan. Gambar 5.6 merupakan hasil pengujian pada kasus garis yang berimpitan. Gambar masukan terdiri atas dua buah garis, garis pertama memiliki koordinat titik awal (100, 100) dan koordinat titik akhir (300, 300), sedangkan garis kedua memiliki koordinat titik awal (200, 200) dan koordinat titik akhir (400, 400). Perangkat lunak tidak menentukan titik potong antara dua buah garis yang berimpitan sehingga masukan pada Gambar 5.6 dianggap sebagai dua buah *graph* terpisah.

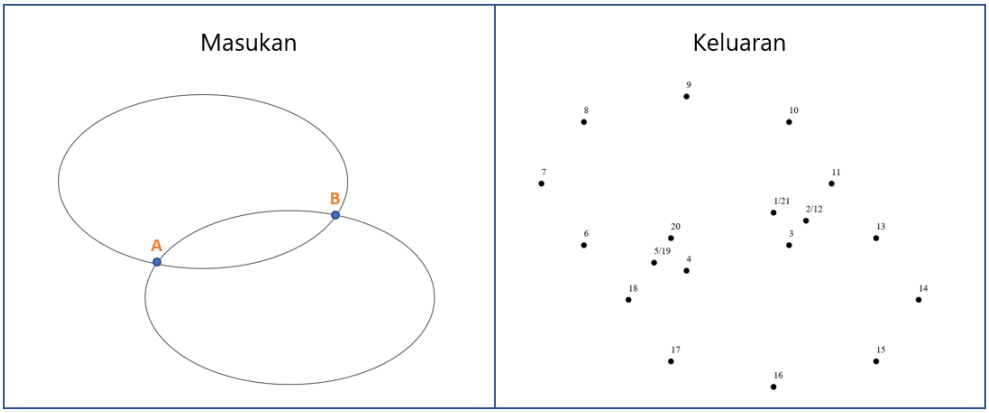


Gambar 5.6: Hasil pengujian pada kasus ketiga gambar yang bertumpukan

Koordinat titik potong disimpan dalam variabel bertipe data *double*. Hasil perhitungan yang disimpan dalam tipe data *double* memiliki presisi tertentu. Koordinat titik potong yang memiliki banyak angka di belakang koma hasil perhitungan rumus pada analisis Bagian 3.1.6 akan dibulatkan ketika disimpan pada sebuah variabel bertipe data *double* akibat keterbatasan presisi tipe data *double*.

Ketika terdapat sebuah perpotongan, beberapa *edge* baru ditambahkan pada *graph*. Titik potong yang dihasilkan perpotongan tersebut akan menjadi salah satu *vertex* yang dihubungkan oleh *edge* baru tersebut. Ketika terjadi pembulatan pada koordinat titik potong, rumus pada analisis Bagian 3.1.6 akan menghasilkan titik potong baru sehingga akan terus ditemukan titik potong baru pada perpotongan yang sama.

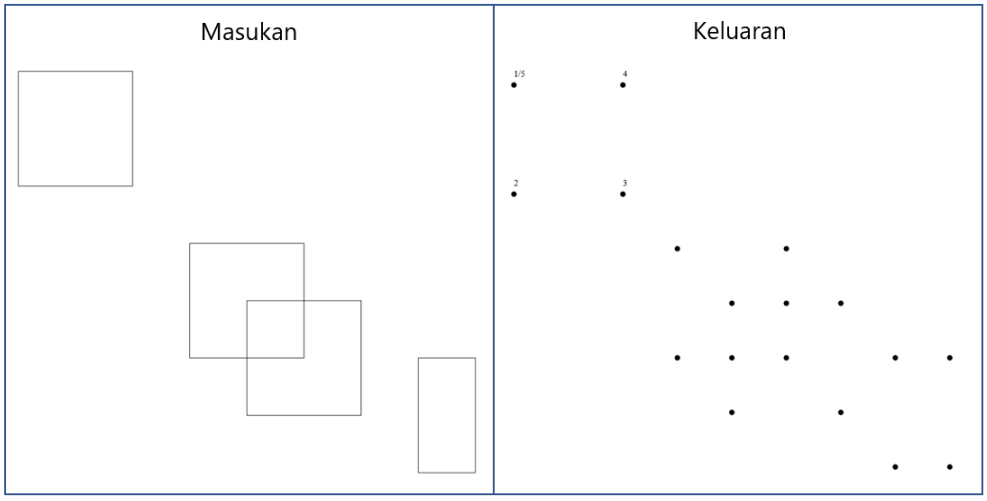
Presisi variabel yang menyimpan koordinat titik potong perlu ditentukan agar tidak ditemukan titik potong baru dari perpotongan yang sama. Pada skripsi ini koordinat titik potong yang dihasilkan sebuah perpotongan dibulatkan hingga dua angka di belakang koma. Gambar 5.7 merupakan pengujian pada kasus koordinat titik potong yang memiliki banyak angka di belakang koma. Koordinat titik potong A dibulatkan menjadi (244.11, 436.11) dan koordinat titik potong B dibulatkan menjadi (505.89, 363.89) sehingga tidak ditemukan titik potong baru dari perpotongan yang sama secara terus-menerus.



Gambar 5.7: Hasil pengujian pada kasus koordinat titik potong yang memiliki banyak angka di belakang koma

5.2.4 Pengujian pada Kasus *Graph* Tak Terhubung

Masukan yang diterima perangkat lunak pada skripsi ini adalah gambar SVG yang dapat diubah menjadi sebuah *graph* terhubung. Pengujian pada gambar yang tidak dapat diubah menjadi sebuah *graph* terhubung dilakukan untuk mengetahui cara perangkat lunak dalam menangani masukan yang berada di luar batasan masalah. Gambar 5.8 merupakan hasil pengujian pada kasus *graph* tak terhubung.



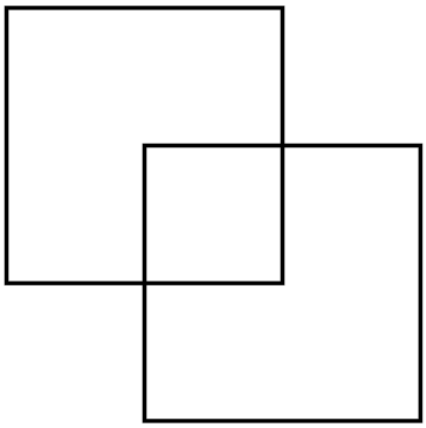
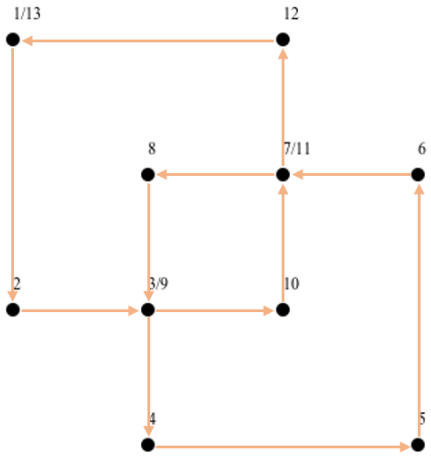
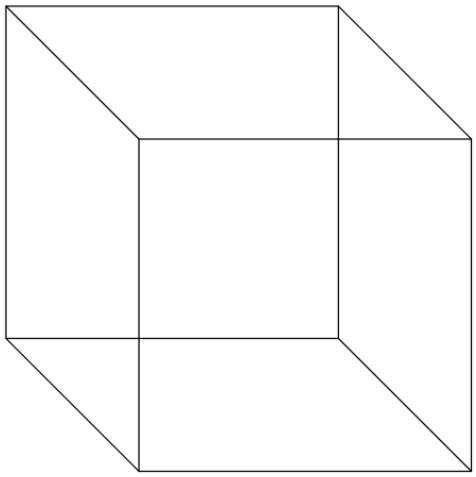
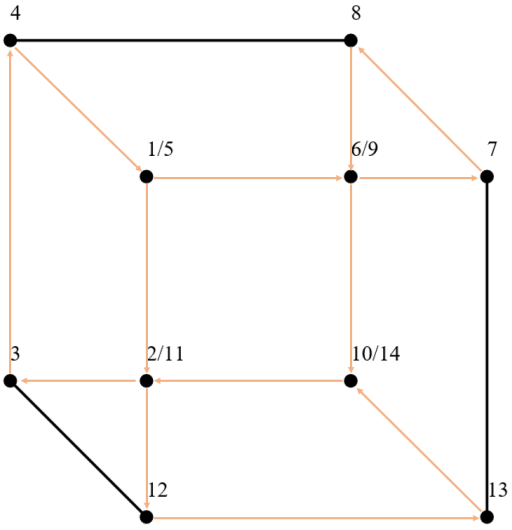
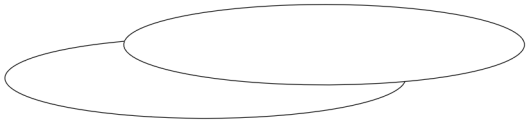
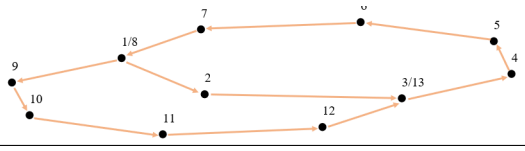
Gambar 5.8: Hasil pengujian pada kasus *graph* tak terhubung

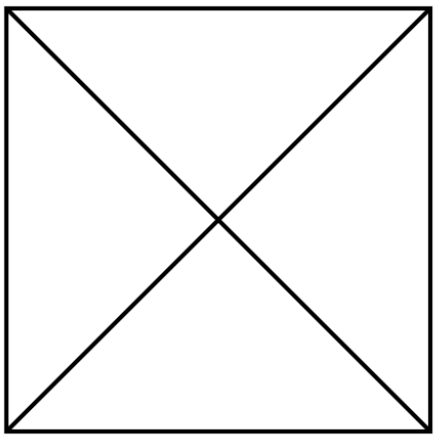
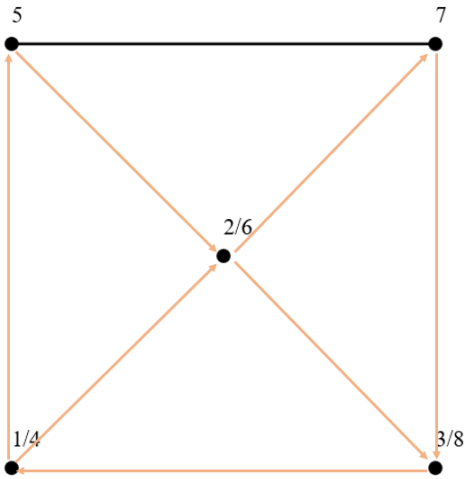
Pada Gambar 5.8 perangkat lunak hanya mampu mengubah sebuah *graph* terhubung menjadi bagian soal permainan menghubungkan titik. Dua buah *graph* terhubung yang lain tidak perangkat lunak ubah menjadi soal permainan. Hal tersebut dikarenakan algoritma Hierholzer hanya dapat menentukan *euler path* pada sebuah *graph* terhubung.

5.2.5 Pengujian Algoritma Hierholzer

Pengujian algoritma Hierholzer dilakukan dengan mengerjakan soal permainan menghubungkan titik yang dihasilkan perangkat lunak. Beberapa hasil pengujian yang dilakukan sebelumnya dikerjakan secara manual untuk menguji urutan penelusuran yang dihasilkan oleh algoritma Hierholzer yang diimplementasikan pada perangkat lunak. Keluaran perangkat lunak dikerjakan secara manual menggunakan kakas Microsoft Power Point sehingga hasil pengerjaan lebih terstruktur. Beberapa pengujian algoritma Hierholzer yang telah dilakukan ditunjukkan pada Tabel 5.3.

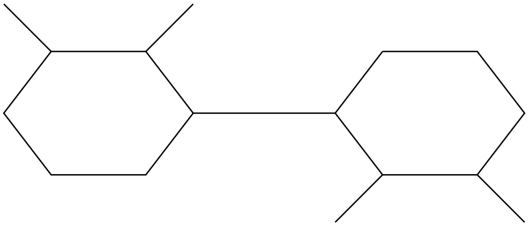
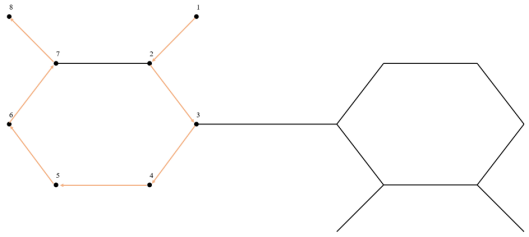
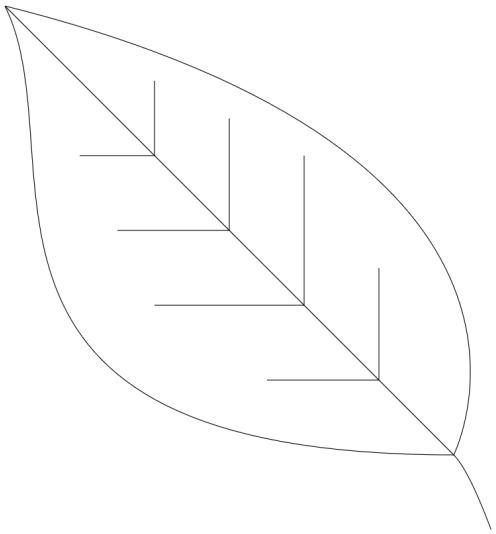
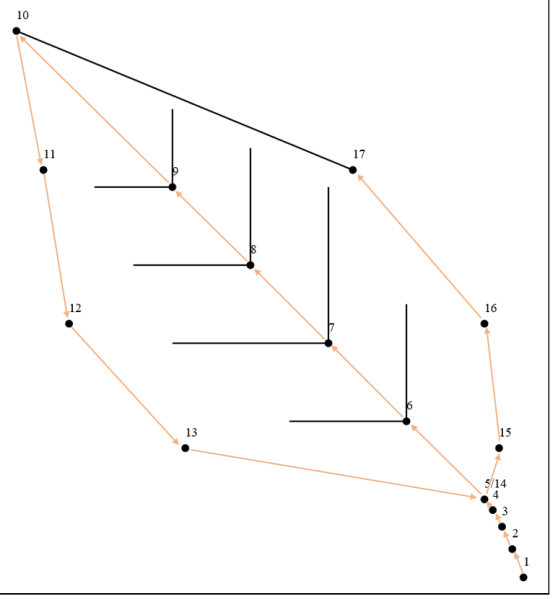
Tabel 5.3: Hasil pengujian algoritma Hierholzer

Kasus	Masukan	Hasil
1		
2		
3		

4		

Menurut analisis pada Bagian 3.2 terdapat beberapa kasus perangkat lunak tidak menghasilkan soal permainan titik dengan garis bantuan yang minimal. Pengujian terhadap kasus-kasus tersebut dilakukan untuk memastikan keluaran yang dihasilkan perangkat lunak sesuai dengan analisis pada bagian 3.2 dan algoritma Hierholzer bekerja dengan baik pada kasus-kasus tersebut. Tabel 5.4 merupakan pengujian algoritma hierholzer terhadap kasus-kasus yang tidak memiliki garis bantuan minimal. Kasus 1 pada Tabel 5.4 telah dibahas pada Bagian 3.2, sedangkan Kasus 2 pada Tabel 5.4 tidak memiliki garis bantuan minimal karena terdapat garis bantuan yang menghubungkan *vertex* dengan nomor urut penelusuran 10 dan 17.

Tabel 5.4: Hasil pengujian algoritma Hierholzer pada kasus tidak optimal

Kasus	Masukan	Hasil
1		
2		

BAB 6

KESIMPULAN DAN SARAN

Bagian ini berisi kesimpulan dan saran. Pada bagian ini disimpulkan hal-hal yang dilakukan pada skripsi ini. Beberapa saran dipaparkan pada bagian ini agar penelitian yang dilakukan pada skripsi ini dapat terus berkembang.

6.1 Kesimpulan

Scalable Vector Graphics (SVG) merupakan sebuah gambar dua dimensi berbasis vektor yang didefinisikan dalam format *Extensible Markup Language* (XML). Sebuah gambar SVG disimpan dalam bentuk dokumen. Dokumen SVG diawali dengan tag pembuka <svg> dan diakhiri dengan tag penutup </svg>. Pada dokumen SVG terdapat elemen-elemen yang merepresentasikan bagian gambar SVG. Setiap elemen memiliki atribut-atribut yang menentukan ukuran dan bentuk elemen tersebut.

Terdapat enam buah bentuk dasar yang dapat digunakan pada dokumen SVG untuk membentuk sebuah bagian gambar, yaitu, persegi panjang, lingkaran, elips, garis, *polyline*, dan poligon. Selain menggunakan bentuk dasar, *path* dapat digunakan untuk mendefinisikan bentuk pada dokumen SVG. Terdapat sepuluh perintah yang dapat digunakan untuk membentuk sebuah *path*, yaitu, *moveto*, *lineto*, *horizontal lineto*, *vertical lineto*, *quadratic Bezier curveto*, *shorthand quadratic Bezier curveto*, *cubic Bezier curveto*, *shorthand cubic Bezier curveto*, *elliptical arc*, dan *close path*.

Setiap elemen yang didefinisikan pada dokumen SVG diubah menjadi bagian *graph*. Algoritma untuk mengubah elemen menjadi bagian *graph* berbeda-beda sesuai dengan jenis elemen. Kurva dan busur elips perlu diubah menjadi ruas-ruas garis terlebih dahulu sebelum diubah menjadi bagian *graph*.

Luas *bounding rectangle* digunakan untuk memperkirakan ukuran suatu elemen. Bagian gambar yang terlalu kecil tidak dijadikan bagian soal permainan menghubungkan titik. Setelah gambar SVG diubah menjadi *graph*, setiap *edge* merepresentasikan ruas garis yang membentuk gambar tersebut. Setiap titik potong pada *edge* yang berpotongan dijadikan *vertex* untuk disertakan pada algoritma pencarian *euler path* Hierholzer.

Algoritma Hierholzer berfungsi untuk mencari *euler path* pada sebuah *euler graph* terhubung. *Graph* yang dihasilkan dari proses pengubahan gambar SVG belum tentu merupakan *euler graph* sehingga sebuah heuristik diimplementasikan untuk mengubah *graph* menjadi *euler graph*. Heuristik yang diimplementasikan tidak selalu menghasilkan solusi yang optimal pada setiap kasus sehingga tidak setiap soal yang dihasilkan perangkat lunak memiliki garis bantuan yang minimal.

6.2 Saran

Perangkat lunak yang dikembangkan pada skripsi ini hanya berfungsi untuk menghasilkan soal permainan menghubungkan titik berdasarkan gambar SVG masukan. Beberapa fitur seperti *print* soal dan pembuatan beberapa soal sekaligus dapat ditambahkan pada perangkat lunak. Penambahan fitur-fitur tersebut bertujuan untuk mempermudah pengguna dalam menghasilkan soal permainan menghubungkan titik.

Masukan perangkat lunak pada skripsi ini dibatasi, yaitu gambar SVG yang dapat diubah menjadi sebuah *graph* terhubung. Algoritma Hierholzer tidak berjalan dengan baik pada *graph* yang tidak terhubung. Penelitian lebih lanjut dapat dilakukan agar perangkat lunak dapat mengubah *graph* tidak terhubung menjadi soal permainan menghubungkan titik. Sebuah *graph* yang tidak terhubung terdiri atas beberapa *graph* terhubung sehingga algoritma Hierholzer dapat diterapkan pada setiap *graph* terhubung tersebut.

Terdapat dua jenis garis bantuan pada soal permainan menghubungkan titik yang dihasilkan oleh perangkat lunak yang dikembangkan pada skripsi ini. Sebuah bagian gambar yang terlalu kecil tidak diproses lebih lanjut oleh perangkat lunak sehingga bagian gambar tersebut dijadikan garis bantuan pada soal dengan cara memindahkan elemen yang merepresentasikan bagian gambar tersebut ke dokumen SVG keluaran. Berbeda dengan hal sebelumnya, garis bantuan yang dihasilkan pada proses pengubahan *graph* masukan menjadi *euler graph* direpresentasikan menggunakan sebuah elemen garis. Sebuah garis bantuan berbentuk kurva yang dihasilkan pada proses pengubahan *graph* menjadi *euler graph* akan direpresentasikan menjadi ruas-ruas garis, sedangkan sebuah kurva yang terlalu kecil akan dijadikan garis bantuan yang tetap berbentuk kurva.

Proses pembuatan garis bantuan yang dilakukan oleh perangkat lunak dapat dikembangkan sehingga setiap garis bantuan tetap menggunakan bagian gambar aslinya. Setiap garis bantuan yang berbentuk kurva tidak direpresentasikan sebagai ruas-ruas garis karena setiap garis bantuan menggunakan bagian gambar aslinya. Tidak semua garis bantuan yang dihasilkan pada proses pengubahan *graph* menjadi *euler graph* merupakan sebuah elemen utuh. Penelitian lebih lanjut dapat dilakukan untuk menangani hal tersebut.

DAFTAR REFERENSI

[1] Eisenberg, J. D. dan Bellamy-Royds, A. (2014) *SVG Essentials: Producing Scalable Vector Graphics with XML*. " O'Reilly Media, Inc.".

[2] REC-SVG11-20110816 (211) *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. World Wide Web Consortium. Cambridge, Massachusetts, U.S.

[3] Farin, G. (2014) *Curves and surfaces for computer-aided geometric design: a practical guide*. Elsevier.

[4] Gross, J. L. dan Yellen, J. (2004) *Handbook of graph theory*. CRC press.

[5] Even, S. (2011) *Graph algorithms*. Cambridge University Press.

[6] Halim, S. dan Halim, F. (2013) *Competitive Programming 3*. Lulu Independent Publish.

LAMPIRAN A

KODE PROGRAM

Listing A.1: Element.java

```java
package engine;

import java.util.HashMap;
import java.util.Map;

/**
 * @author Albert - 2014730007
 */
public abstract class Element {
    protected String name;
    protected Map<String, String> attributes;

    public Element(String name, String attributes) {
        this.name = name;
        this.attributes = new HashMap<String, String>();
        String att[] = attributes.split("\"");
        String attributeName = "";
        String attributeValue = "";
        if(!attributes.isEmpty()){
            for(int i=0;i<att.length;i+=2){
                attributeName = att[i].substring(0, att[i].length()-1);
                if(this.name.equals("path") && attributeName.equals("d")){
                    int attLen = att[i+1].length();
                    String newAtt = "";
                    Character cur = null;
                    for(int j=0;j<attLen;j++){
                        cur = att[i+1].charAt(j);
                        if(Character.isAlphabetic(cur)){
                            newAtt += "␣" + cur + "␣";
                        }
                        else{
                            if(cur == '-'){
                                newAtt += "␣" + cur;
                            }
                            else{
                                newAtt += cur;
                            }
                        }
                    }
                    newAtt = newAtt.trim();
                    attributeValue = newAtt;
                }
                else{
                    attributeValue = att[i+1];
                }
                this.attributes.put(attributeName, attributeValue);
            }
        }
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Map<String, String> getAttributes() {
        return attributes;
    }

    public double getHorizontalLength(){
        return this.getMaxX() - this.getMinX();
    }

    public double getVerticalLength(){
        return this.getMaxY() - this.getMinY();
    }

    public double getBoundingRectArea(){
        if(this.getHorizontalLength()==0)return this.getVerticalLength();
        if(this.getVerticalLength()==0)return this.getHorizontalLength();
        return this.getHorizontalLength() * this.getVerticalLength();
    }
```

```
76
77      public abstract double getMaxX();
78
79      public abstract double getMinX();
80
81      public abstract double getMaxY();
82
83      public abstract double getMinY();
84  }
```

Listing A.2: Rectangle.java

```
1   package engine;
2
3   /**
4    * @author Albert - 2014730007
5    */
6   public class Rectangle extends Element {
7
8       public Rectangle(String name, String attributes) {
9           super(name, attributes);
10      }
11
12      @Override
13      public double getMaxX() {
14          return this.getMinX() + Double.parseDouble(this.attributes.get("width"));
15      }
16
17      @Override
18      public double getMinX() {
19          if(!this.attributes.containsKey("x")){
20              return 0;
21          }
22          else return Double.parseDouble(this.attributes.get("x"));
23      }
24
25      @Override
26      public double getMaxY() {
27          return this.getMinY() + Double.parseDouble(this.attributes.get("height"));
28      }
29
30      @Override
31      public double getMinY() {
32          if(!this.attributes.containsKey("y")){
33              return 0;
34          }
35          else return Double.parseDouble(this.attributes.get("y"));
36      }
37  }
```

Listing A.3: Circle.java

```
1   package engine;
2
3   /**
4    * @author Albert - 2014730007
5    */
6   public class Circle extends Element {
7
8       public Circle(String name, String attributes) {
9           super(name, attributes);
10      }
11
12      @Override
13      public double getMaxX() {
14          return Double.parseDouble(this.attributes.get("cx")) + Double.parseDouble(this.attributes.get("r"));
15      }
16
17      @Override
18      public double getMinX() {
19          return Double.parseDouble(this.attributes.get("cx")) - Double.parseDouble(this.attributes.get("r"));
20      }
21
22      @Override
23      public double getMaxY() {
24          return Double.parseDouble(this.attributes.get("cy")) + Double.parseDouble(this.attributes.get("r"));
25      }
26
27      @Override
28      public double getMinY() {
29          return Double.parseDouble(this.attributes.get("cy")) - Double.parseDouble(this.attributes.get("r"));
30      }
31
32  }
```

Listing A.4: Ellipse.java

```
1   package engine;
2
3   /**
4    * @author Albert - 2014730007
5    */
6   public class Ellipse extends Element {
7
```

```
8       public Ellipse(String name, String attributes) {
9           super(name, attributes);
10      }
11
12      @Override
13      public double getMaxX() {
14          return Double.parseDouble(this.attributes.get("cx")) + Double.parseDouble(this.attributes.get("rx"));
15      }
16
17      @Override
18      public double getMinX() {
19          return Double.parseDouble(this.attributes.get("cx")) - Double.parseDouble(this.attributes.get("rx"));
20      }
21
22      @Override
23      public double getMaxY() {
24          return Double.parseDouble(this.attributes.get("cy")) + Double.parseDouble(this.attributes.get("ry"));
25      }
26
27      @Override
28      public double getMinY() {
29          return Double.parseDouble(this.attributes.get("cy")) - Double.parseDouble(this.attributes.get("ry"));
30      }
31
32  }
```

Listing A.5: Line.java

```
1   package engine;
2
3   /**
4    * @author Albert - 2014730007
5    */
6   public class Line extends Element {
7
8       public Line(String name, String attributes) {
9           super(name, attributes);
10      }
11
12      @Override
13      public double getMaxX() {
14          return Math.max(Double.parseDouble(this.attributes.get("x2")), Double.parseDouble(this.attributes.get("x1")));
15      }
16
17      @Override
18      public double getMinX() {
19          return Math.min(Double.parseDouble(this.attributes.get("x1")), Double.parseDouble(this.attributes.get("x2")));
20      }
21
22      @Override
23      public double getMaxY() {
24          return Math.max(Double.parseDouble(this.attributes.get("y2")), Double.parseDouble(this.attributes.get("y1")));
25      }
26
27      @Override
28      public double getMinY() {
29          return Math.min(Double.parseDouble(this.attributes.get("y1")), Double.parseDouble(this.attributes.get("y2")));
30      }
31
32  }
```

Listing A.6: Polyline.java

```
1   package engine;
2
3
4   /**
5    * @author Albert - 2014730007
6    */
7   public class Polyline extends Element {
8
9       public Polyline(String name, String attributes) {
10          super(name, attributes);
11      }
12
13      @Override
14      public double getMaxX() {
15          String points[] = this.attributes.get("points").split("\\s+");
16          int numPoints = points.length;
17          double maxX = Double.MIN_VALUE;
18          for(int i=0;i<numPoints;i+=2){
19              double cur = Double.parseDouble(points[i]);
20              if(cur>maxX)maxX = cur;
21          }
22          return maxX;
23      }
24
25      @Override
26      public double getMinX() {
27          String points[] = this.attributes.get("points").split("\\s+");
28          int numPoints = points.length;
29          double minX = Double.MAX_VALUE;
30          for(int i=0;i<numPoints;i+=2){
31              double cur = Double.parseDouble(points[i]);
32              if(cur<minX)minX = cur;
33          }
```

```
34          return minX;
35      }
36
37      @Override
38      public double getMaxY() {
39          String points[] = this.attributes.get("points").split("\\s+");
40          int numPoints = points.length;
41          double maxY = Double.MIN_VALUE;
42          for(int i=1;i<numPoints;i+=2){
43              double cur = Double.parseDouble(points[i]);
44              if(cur>maxY)maxY = cur;
45          }
46          return maxY;
47      }
48
49      @Override
50      public double getMinY() {
51          String points[] = this.attributes.get("points").split("\\s+");
52          int numPoints = points.length;
53          double minY = Double.MAX_VALUE;
54          for(int i=1;i<numPoints;i+=2){
55              double cur = Double.parseDouble(points[i]);
56              if(cur<minY)minY = cur;
57          }
58          return minY;
59      }
60  }
```

Listing A.7: Polygon.java

```
1   package engine;
2
3   /**
4    * @author Albert - 2014730007
5    */
6   public class Polygon extends Element {
7
8       public Polygon(String name, String attributes) {
9           super(name, attributes);
10      }
11
12      @Override
13      public double getMaxX() {
14          String points[] = this.attributes.get("points").split("\\s+");
15          int numPoints = points.length;
16          double maxX = Double.MIN_VALUE;
17          for(int i=0;i<numPoints;i+=2){
18              double cur = Double.parseDouble(points[i]);
19              if(cur>maxX)maxX = cur;
20          }
21          return maxX;
22      }
23
24      @Override
25      public double getMinX() {
26          String points[] = this.attributes.get("points").split("\\s+");
27          int numPoints = points.length;
28          double minX = Double.MAX_VALUE;
29          for(int i=0;i<numPoints;i+=2){
30              double cur = Double.parseDouble(points[i]);
31              if(cur<minX)minX = cur;
32          }
33          return minX;
34      }
35
36      @Override
37      public double getMaxY() {
38          String points[] = this.attributes.get("points").split("\\s+");
39          int numPoints = points.length;
40          double maxY = Double.MIN_VALUE;
41          for(int i=1;i<numPoints;i+=2){
42              double cur = Double.parseDouble(points[i]);
43              if(cur>maxY)maxY = cur;
44          }
45          return maxY;
46      }
47
48      @Override
49      public double getMinY() {
50          String points[] = this.attributes.get("points").split("\\s+");
51          int numPoints = points.length;
52          double minY = Double.MAX_VALUE;
53          for(int i=1;i<numPoints;i+=2){
54              double cur = Double.parseDouble(points[i]);
55              if(cur<minY)minY = cur;
56          }
57          return minY;
58      }
59  }
```

Listing A.8: Path.java

```
1   package engine;
2
3   import java.awt.geom.Point2D;
4   import java.util.ArrayList;
```

```
5  import java.util.List;
6
7  /**
8   * @author Albert - 2014730007
9   */
10 public class Path extends Element {
11     public Path(String name, String attributes) {
12         super(name, attributes);
13     }
14
15     @Override
16     public double getMaxX() {
17         String cmd[] = this.attributes.get("d").split("\\s+");
18         int cmdLen = cmd.length;
19         String curCommand = "";
20         String curProcessed = "";
21         double cumulativeX = 0;
22         double cumulativeY = 0;
23         int i = 0;
24         double maxX = Double.MIN_VALUE;
25         List<Point2D.Double> shorthandCurveToHelper = new ArrayList<Point2D.Double>();
26
27         while(i<cmdLen){
28             curProcessed = cmd[i];
29             if(Character.isAlphabetic(curProcessed.charAt(0))){
30                 curCommand = cmd[i];
31                 if(curCommand.equals("Q")){
32                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
33                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
34                 }
35                 else if(curCommand.equals("q")){
36                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
37                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
38                 }
39                 if(curCommand.equals("C")){
40                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
41                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
42                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5]), Double.parseDouble(cmd[i+6])));
43                 }
44                 else if(curCommand.equals("c")){
45                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
46                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
47                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5])+cumulativeX, Double.parseDouble(cmd
                            [i+6])+cumulativeY));
48                 }
49                 i++;
50             }
51             else{
52                 double currentProcessed = Double.parseDouble(curProcessed);
53                 if(curCommand.equals("H")){
54                     cumulativeX = currentProcessed;
55                     if(cumulativeX>maxX)maxX = cumulativeX;
56                     i++;
57                 }
58                 else if(curCommand.equals("h")){
59                     cumulativeX += currentProcessed;
60                     if(cumulativeX>maxX)maxX = cumulativeX;
61                     i++;
62                 }
63                 else if(curCommand.equals("V")){
64                     cumulativeY = currentProcessed;
65                     i++;
66                 }
67                 else if(curCommand.equals("v")){
68                     cumulativeY += currentProcessed;
69                     i++;
70                 }
71                 else if(curCommand.equals("A")){
72                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
73                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
74                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
75                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, currentProcessed,
                            currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i+3]), Integer.
                            parseInt(cmd[i+4]));
76                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
77                     if(cP.getMatrix()[0][0]+rads.getMatrix()[0][0]>maxX)maxX = cP.getMatrix()[0][0]+rads.getMatrix()[0][0];
78                     cumulativeX = currentProcessed3;
79                     cumulativeY = currentProcessed4;
80                     i+=7;
81                 }
82                 else if(curCommand.equals("a")){
83                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
84                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
85                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
86                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3+cumulativeX, currentProcessed4+
                            cumulativeY, currentProcessed, currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.
                            parseInt(cmd[i+3]), Integer.parseInt(cmd[i+4]));
87                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
88                     if(cP.getMatrix()[0][0]+rads.getMatrix()[0][0]>maxX)maxX = cP.getMatrix()[0][0]+rads.getMatrix()[0][0];
89                     cumulativeX += currentProcessed3;
90                     cumulativeY += currentProcessed4;
```

```
91                      i+=7;
92                  }
93                  else if(curCommand.equals("T")){
94                      double currentProcessed2 = Double.parseDouble(cmd[i+1]);
95                      if(shorthandCurveToHelper.size()>0){
96                          Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
97                          if(mCP.x>maxX){
98                              maxX = mCP.x;
99                          }
100                     }
101                     if(currentProcessed>maxX)maxX = currentProcessed;
102                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
103                         cumulativeX = currentProcessed;
104                         cumulativeY = currentProcessed2;
105                     }
106                     i+=2;
107                 }
108                 else if(curCommand.equals("t")){
109                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
110                     if(shorthandCurveToHelper.size()>0){
111                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
112                         if(mCP.x>maxX){
113                             maxX = mCP.x;
114                         }
115                     }
116                     if(currentProcessed+cumulativeX>maxX)maxX = currentProcessed+cumulativeX;
117                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
118                         cumulativeX += currentProcessed;
119                         cumulativeY += currentProcessed2;
120                     }
121                     i+=2;
122                 }
123                 else if(curCommand.equals("S")){
124                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
125                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
126                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
127                     if(shorthandCurveToHelper.size()>0){
128                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
129                         if(mCP.x>maxX){
130                             maxX = mCP.x;
131                         }
132                     }
133                     if(currentProcessed>maxX)maxX = currentProcessed;
134                     if(currentProcessed3>maxX)maxX = currentProcessed;
135                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
136                         cumulativeX = currentProcessed3;
137                         cumulativeY = currentProcessed4;
138                     }
139                     i+=4;
140                 }
141                 else if(curCommand.equals("s")){
142                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
143                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
144                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
145                     if(shorthandCurveToHelper.size()>0){
146                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
147                         if(mCP.x>maxX){
148                             maxX = mCP.x;
149                         }
150                     }
151                     if(currentProcessed+cumulativeX>maxX)maxX = currentProcessed+cumulativeX;
152                     if(currentProcessed3+cumulativeX>maxX)maxX = currentProcessed3+cumulativeX;
153                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
154                         cumulativeX += currentProcessed3;
155                         cumulativeY += currentProcessed4;
156                     }
157                     i+=4;
158                 }
159                 else{
160                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
161                     if(Character.isLowerCase(curCommand.charAt(0))){
162                         if(currentProcessed+cumulativeX>maxX)maxX = currentProcessed+cumulativeX;
163                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
164                             cumulativeX += currentProcessed;
165                             cumulativeY += currentProcessed2;
166                         }
167                     }
168                     else{
169                         if(currentProcessed>maxX)maxX = currentProcessed;
170                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
171                             cumulativeX = currentProcessed;
172                             cumulativeY = currentProcessed2;
173                         }
174                     }
175                     i+=2;
176                 }
177             }
178         }
179 
180         return maxX;
181     }
182 
183     @Override
184     public double getMinX() {
185         String cmd[] = this.attributes.get("d").split("\\s+");
```

```
186         int cmdLen = cmd.length;
187         String curCommand = "";
188         String curProcessed = "";
189         double cumulativeX = 0;
190         double cumulativeY = 0;
191         int i = 0;
192         double minX = Double.MAX_VALUE;
193         List<Point2D.Double> shorthandCurveToHelper = new ArrayList<Point2D.Double>();
194
195         while(i<cmdLen){
196             curProcessed = cmd[i];
197             if(Character.isAlphabetic(curProcessed.charAt(0))){
198                 curCommand = cmd[i];
199                 if(curCommand.equals("Q")){
200                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
201                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
202                 }
203                 else if(curCommand.equals("q")){
204                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
205                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
206                 }
207                 if(curCommand.equals("C")){
208                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
209                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
210                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5]), Double.parseDouble(cmd[i+6])));
211                 }
212                 else if(curCommand.equals("c")){
213                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
214                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
215                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5])+cumulativeX, Double.parseDouble(cmd
                            [i+6])+cumulativeY));
216                 }
217                 i++;
218             }
219             else{
220                 double currentProcessed = Double.parseDouble(curProcessed);
221                 if(curCommand.equals("H")){
222                     cumulativeX = currentProcessed;
223                     if(cumulativeX<minX)minX = cumulativeX;
224                     i++;
225                 }
226                 else if(curCommand.equals("h")){
227                     cumulativeX += currentProcessed;
228                     if(cumulativeX<minX)minX = cumulativeX;
229                     i++;
230                 }
231                 else if(curCommand.equals("V")){
232                     cumulativeY = currentProcessed;
233                     i++;
234                 }
235                 else if(curCommand.equals("v")){
236                     cumulativeY += currentProcessed;
237                     i++;
238                 }
239                 else if(curCommand.equals("A")){
240                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
241                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
242                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
243                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, currentProcessed,
                            currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i+3]), Integer.
                            parseInt(cmd[i+4]));
244                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
245                     if(cP.getMatrix()[0][0]-rads.getMatrix()[0][0]<minX)minX = cP.getMatrix()[0][0]-rads.getMatrix()[0][0];
246                     cumulativeX = currentProcessed3;
247                     cumulativeY = currentProcessed4;
248                     i+=7;
249                 }
250                 else if(curCommand.equals("a")){
251                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
252                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
253                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
254                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3+cumulativeX, currentProcessed4+
                            cumulativeY, currentProcessed, currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.
                            parseInt(cmd[i+3]), Integer.parseInt(cmd[i+4]));
255                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
256                     if(cP.getMatrix()[0][0]-rads.getMatrix()[0][0]<minX)minX = cP.getMatrix()[0][0]-rads.getMatrix()[0][0];
257                     cumulativeX += currentProcessed3;
258                     cumulativeY += currentProcessed4;
259                     i+=7;
260                 }
261                 else if(curCommand.equals("T")){
262                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
263                     if(shorthandCurveToHelper.size()>0){
264                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
265                         if(mCP.x<minX){
266                             minX = mCP.x;
267                         }
268                     }
269                     if(currentProcessed<minX)minX = currentProcessed;
270                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
```

```
271                         cumulativeX = currentProcessed;
272                         cumulativeY = currentProcessed2;
273                     }
274                     i+=2;
275                 }
276                 else if(curCommand.equals("t")){
277                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
278                     if(shorthandCurveToHelper.size()>0){
279                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
280                         if(mCP.x<minX){
281                             minX = mCP.x;
282                         }
283                     }
284                     if(currentProcessed+cumulativeX<minX)minX = currentProcessed+cumulativeX;
285                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
286                         cumulativeX += currentProcessed;
287                         cumulativeY += currentProcessed2;
288                     }
289                     i+=2;
290                 }
291                 else if(curCommand.equals("S")){
292                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
293                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
294                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
295                     if(shorthandCurveToHelper.size()>0){
296                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
297                         if(mCP.x<minX){
298                             minX = mCP.x;
299                         }
300                     }
301                     if(currentProcessed<minX)minX = currentProcessed;
302                     if(currentProcessed3<minX)minX = currentProcessed;
303                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
304                         cumulativeX = currentProcessed3;
305                         cumulativeY = currentProcessed4;
306                     }
307                     i+=4;
308                 }
309                 else if(curCommand.equals("s")){
310                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
311                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
312                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
313                     if(shorthandCurveToHelper.size()>0){
314                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
315                         if(mCP.x<minX){
316                             minX = mCP.x;
317                         }
318                     }
319                     if(currentProcessed+cumulativeX<minX)minX = currentProcessed+cumulativeX;
320                     if(currentProcessed3+cumulativeX<minX)minX = currentProcessed3+cumulativeX;
321                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
322                         cumulativeX += currentProcessed3;
323                         cumulativeY += currentProcessed4;
324                     }
325                     i+=4;
326                 }
327                 else{
328                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
329                     if(Character.isLowerCase(curCommand.charAt(0))){
330                         if(currentProcessed+cumulativeX<minX)minX = currentProcessed+cumulativeX;
331                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
332                             cumulativeX += currentProcessed;
333                             cumulativeY += currentProcessed2;
334                         }
335                     }
336                     else{
337                         if(currentProcessed<minX)minX = currentProcessed;
338                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
339                             cumulativeX = currentProcessed;
340                             cumulativeY = currentProcessed2;
341                         }
342                     }
343                     i+=2;
344                 }
345             }
346         }
347 
348         return minX;
349     }
350 
351     @Override
352     public double getMaxY() {
353         String cmd[] = this.attributes.get("d").split("\\s+");
354         int cmdLen = cmd.length;
355         String curCommand = "";
356         String curProcessed = "";
357         double cumulativeX = 0;
358         double cumulativeY = 0;
359         int i = 0;
360         double maxY = Double.MIN_VALUE;
361         List<Point2D.Double> shorthandCurveToHelper = new ArrayList<Point2D.Double>();
362 
363         while(i<cmdLen){
364             curProcessed = cmd[i];
365             if(Character.isAlphabetic(curProcessed.charAt(0))){
366                 curCommand = cmd[i];
```

```
367                 if(curCommand.equals("Q")){
368                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
369                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
370                 }
371                 else if(curCommand.equals("q")){
372                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
373                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
374                 }
375                 if(curCommand.equals("C")){
376                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
377                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
378                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5]), Double.parseDouble(cmd[i+6])));
379                 }
380                 else if(curCommand.equals("c")){
381                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
382                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
383                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5])+cumulativeX, Double.parseDouble(cmd
                            [i+6])+cumulativeY));
384                 }
385                 i++;
386             }
387             else{
388                 double currentProcessed = Double.parseDouble(curProcessed);
389                 if(curCommand.equals("H")){
390                     cumulativeX = currentProcessed;
391                     i++;
392                 }
393                 else if(curCommand.equals("h")){
394                     cumulativeX += currentProcessed;
395                     i++;
396                 }
397                 else if(curCommand.equals("V")){
398                     cumulativeY = currentProcessed;
399                     if(cumulativeY>maxY)maxY = cumulativeY;
400                     i++;
401                 }
402                 else if(curCommand.equals("v")){
403                     cumulativeY += currentProcessed;
404                     if(cumulativeY>maxY)maxY = cumulativeY;
405                     i++;
406                 }
407                 else if(curCommand.equals("A")){
408                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
409                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
410                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
411                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, currentProcessed,
                            currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i+3]), Integer.
                            parseInt(cmd[i+4]));
412                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
413                     if(cP.getMatrix()[1][0]+rads.getMatrix()[1][0]>maxY)maxY = cP.getMatrix()[1][0]+rads.getMatrix()[1][0];
414                     cumulativeX = currentProcessed3;
415                     cumulativeY = currentProcessed4;
416                     i+=7;
417                 }
418                 else if(curCommand.equals("a")){
419                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
420                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
421                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
422                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3+cumulativeX, currentProcessed4+
                            cumulativeY, currentProcessed, currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.
                            parseInt(cmd[i+3]), Integer.parseInt(cmd[i+4]));
423                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
424                     if(cP.getMatrix()[1][0]+rads.getMatrix()[1][0]>maxY)maxY = cP.getMatrix()[1][0]+rads.getMatrix()[1][0];
425                     cumulativeX += currentProcessed3;
426                     cumulativeY += currentProcessed4;
427                     i+=7;
428                 }
429                 else if(curCommand.equals("T")){
430                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
431                     if(shorthandCurveToHelper.size()>0){
432                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
433                         if(mCP.y>maxY){
434                             maxY = mCP.y;
435                         }
436                     }
437                     if(currentProcessed2>maxY)maxY = currentProcessed2;
438                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
439                         cumulativeX = currentProcessed;
440                         cumulativeY = currentProcessed2;
441                     }
442                     i+=2;
443                 }
444                 else if(curCommand.equals("t")){
445                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
446                     if(shorthandCurveToHelper.size()>0){
447                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
448                         if(mCP.y>maxY){
449                             maxY = mCP.y;
450                         }
```

```
451                     }
452                     if(currentProcessed2+cumulativeY>maxY)maxY = currentProcessed2+cumulativeY;
453                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
454                         cumulativeX += currentProcessed;
455                         cumulativeY += currentProcessed2;
456                     }
457                     i+=2;
458                 }
459                 else if(curCommand.equals("S")){
460                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
461                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
462                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
463                     if(shorthandCurveToHelper.size()>0){
464                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
465                         if(mCP.y>maxY){
466                             maxY = mCP.y;
467                         }
468                     }
469                     if(currentProcessed2>maxY)maxY = currentProcessed2;
470                     if(currentProcessed4>maxY)maxY = currentProcessed4;
471                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
472                         cumulativeX = currentProcessed3;
473                         cumulativeY = currentProcessed4;
474                     }
475                     i+=4;
476                 }
477                 else if(curCommand.equals("s")){
478                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
479                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
480                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
481                     if(shorthandCurveToHelper.size()>0){
482                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
483                         if(mCP.y>maxY){
484                             maxY = mCP.y;
485                         }
486                     }
487                     if(currentProcessed2+cumulativeY>maxY)maxY = currentProcessed2+cumulativeY;
488                     if(currentProcessed4+cumulativeY>maxY)maxY = currentProcessed4+cumulativeY;
489                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
490                         cumulativeX += currentProcessed3;
491                         cumulativeY += currentProcessed4;
492                     }
493                     i+=4;
494                 }
495                 else{
496                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
497                     if(Character.isLowerCase(curCommand.charAt(0))){
498                         if(currentProcessed2+cumulativeY>maxY)maxY = currentProcessed2+cumulativeY;
499                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
500                             cumulativeX += currentProcessed;
501                             cumulativeY += currentProcessed2;
502                         }
503                     }
504                     else{
505                         if(currentProcessed2>maxY)maxY = currentProcessed2;
506                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
507                             cumulativeX = currentProcessed;
508                             cumulativeY = currentProcessed2;
509                         }
510                     }
511                     i+=2;
512                 }
513             }
514         }
515 
516         return maxY;
517     }
518 
519     @Override
520     public double getMinY() {
521         String cmd[] = this.attributes.get("d").split("\\s+");
522         int cmdLen = cmd.length;
523         String curCommand = "";
524         String curProcessed = "";
525         double cumulativeX = 0;
526         double cumulativeY = 0;
527         int i = 0;
528         double minY = Double.MAX_VALUE;
529         List<Point2D.Double> shorthandCurveToHelper = new ArrayList<Point2D.Double>();
530 
531         while(i<cmdLen){
532             curProcessed = cmd[i];
533             if(Character.isAlphabetic(curProcessed.charAt(0))){
534                 curCommand = cmd[i];
535                 if(curCommand.equals("Q")){
536                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
537                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
538                 }
539                 else if(curCommand.equals("q")){
540                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
541                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
542                 }
543                 if(curCommand.equals("C")){
544                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1]), Double.parseDouble(cmd[i+2])));
545                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3]), Double.parseDouble(cmd[i+4])));
```

```
546                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5]), Double.parseDouble(cmd[i+6])));
547                 }
548                 else if(curCommand.equals("c")){
549                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+1])+cumulativeX, Double.parseDouble(cmd
                            [i+2])+cumulativeY));
550                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+3])+cumulativeX, Double.parseDouble(cmd
                            [i+4])+cumulativeY));
551                     shorthandCurveToHelper.add(new Point2D.Double(Double.parseDouble(cmd[i+5])+cumulativeX, Double.parseDouble(cmd
                            [i+6])+cumulativeY));
552                 }
553                 i++;
554             }
555             else{
556                 double currentProcessed = Double.parseDouble(curProcessed);
557                 if(curCommand.equals("H")){
558                     cumulativeX = currentProcessed;
559                     i++;
560                 }
561                 else if(curCommand.equals("h")){
562                     cumulativeX += currentProcessed;
563                     i++;
564                 }
565                 else if(curCommand.equals("V")){
566                     cumulativeY = currentProcessed;
567                     if(cumulativeY<minY)minY = cumulativeY;
568                     i++;
569                 }
570                 else if(curCommand.equals("v")){
571                     cumulativeY += currentProcessed;
572                     if(cumulativeY<minY)minY = cumulativeY;
573                     i++;
574                 }
575                 else if(curCommand.equals("A")){
576                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
577                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
578                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
579                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, currentProcessed,
                            currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i+3]), Integer.
                            parseInt(cmd[i+4]));
580                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
581                     if(cP.getMatrix()[1][0]-rads.getMatrix()[1][0]<minY)minY = cP.getMatrix()[1][0]-rads.getMatrix()[1][0];
582                     cumulativeX = currentProcessed3;
583                     cumulativeY = currentProcessed4;
584                     i+=7;
585                 }
586                 else if(curCommand.equals("a")){
587                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
588                     double currentProcessed3 = Double.parseDouble(cmd[i+5]);
589                     double currentProcessed4 = Double.parseDouble(cmd[i+6]);
590                     Matrix rads = ensureRadii(cumulativeX, cumulativeY, currentProcessed3+cumulativeX, currentProcessed4+
                            cumulativeY, currentProcessed, currentProcessed2, Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.
                            parseInt(cmd[i+3]), Integer.parseInt(cmd[i+4]));
591                     Matrix cP = findArcCenterPoint(cumulativeX, cumulativeY, currentProcessed3, currentProcessed4, rads.getMatrix
                            ()[0][0], rads.getMatrix()[1][0], Math.toRadians(Double.parseDouble(cmd[i+2])), Integer.parseInt(cmd[i
                            +3]), Integer.parseInt(cmd[i+4]));
592                     if(cP.getMatrix()[1][0]-rads.getMatrix()[1][0]<minY)minY = cP.getMatrix()[1][0]-rads.getMatrix()[1][0];
593                     cumulativeX += currentProcessed3;
594                     cumulativeY += currentProcessed4;
595                     i+=7;
596                 }
597                 else if(curCommand.equals("T")){
598                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
599                     if(shorthandCurveToHelper.size()>0){
600                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
601                         if(mCP.y<minY){
602                             minY = mCP.y;
603                         }
604                     }
605                     if(currentProcessed2<minY)minY = currentProcessed2;
606                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
607                         cumulativeX = currentProcessed;
608                         cumulativeY = currentProcessed2;
609                     }
610                     i+=2;
611                 }
612                 else if(curCommand.equals("t")){
613                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
614                     if(shorthandCurveToHelper.size()>0){
615                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(0), shorthandCurveToHelper.get(1))
                                ;
616                         if(mCP.y<minY){
617                             minY = mCP.y;
618                         }
619                     }
620                     if(currentProcessed2+cumulativeY<minY)minY = currentProcessed2+cumulativeY;
621                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
622                         cumulativeX += currentProcessed;
623                         cumulativeY += currentProcessed2;
624                     }
625                     i+=2;
626                 }
627                 else if(curCommand.equals("S")){
628                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
629                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
630                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
631                     if(shorthandCurveToHelper.size()>0){
```

```
632                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
633                         if(mCP.y<minY){
634                             minY = mCP.y;
635                         }
636                     }
637                     if(currentProcessed2<minY)minY = currentProcessed2;
638                     if(currentProcessed4<minY)minY = currentProcessed4;
639                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
640                         cumulativeX = currentProcessed3;
641                         cumulativeY = currentProcessed4;
642                     }
643                     i+=4;
644                 }
645                 else if(curCommand.equals("s")){
646                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
647                     double currentProcessed3 = Double.parseDouble(cmd[i+2]);
648                     double currentProcessed4 = Double.parseDouble(cmd[i+3]);
649                     if(shorthandCurveToHelper.size()>0){
650                         Point2D.Double mCP = this.mirrorControlPoint(shorthandCurveToHelper.get(1), shorthandCurveToHelper.get(2))
                                ;
651                         if(mCP.y<minY){
652                             minY = mCP.y;
653                         }
654                     }
655                     if(currentProcessed2+cumulativeY<minY)minY = currentProcessed2+cumulativeY;
656                     if(currentProcessed4+cumulativeY<minY)minY = currentProcessed4+cumulativeY;
657                     if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
658                         cumulativeX += currentProcessed3;
659                         cumulativeY += currentProcessed4;
660                     }
661                     i+=4;
662                 }
663                 else{
664                     double currentProcessed2 = Double.parseDouble(cmd[i+1]);
665                     if(Character.isLowerCase(curCommand.charAt(0))){
666                         if(currentProcessed2+cumulativeY<minY)minY = currentProcessed2+cumulativeY;
667                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
668                             cumulativeX += currentProcessed;
669                             cumulativeY += currentProcessed2;
670                         }
671                     }
672                     else{
673                         if(currentProcessed2<minY)minY = currentProcessed2;
674                         if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
675                             cumulativeX = currentProcessed;
676                             cumulativeY = currentProcessed2;
677                         }
678                     }
679                     i+=2;
680                 }
681             }
682         }
683 
684         return minY;
685     }
686 
687     private Matrix findArcCenterPoint(double x1, double y1, double x2, double y2, double rx, double ry, double varphi, int fA, int
          fS){
688         //find x1' and y1'
689         double matrix[][] = new double[2][2];
690         matrix[0][0] = Math.cos(varphi);
691         matrix[0][1] = Math.sin(varphi);
692         matrix[1][0] = Math.sin(varphi) * -1;
693         matrix[1][1] = Math.cos(varphi);
694         Matrix m1 = new Matrix(matrix);
695         matrix = new double[2][1];
696         matrix[0][0] = (x1-x2)/2.0;
697         matrix[1][0] = (y1-y2)/2.0;
698         Matrix m2 = new Matrix(matrix);
699         Matrix coordinateA = MatrixMath.multiply(m1, m2);
700         double x1A = coordinateA.getMatrix()[0][0];
701         double y1A = coordinateA.getMatrix()[1][0];
702 
703         //find cx' and cy'
704         double val = Math.sqrt( ((rx*rx*ry*ry) - (rx*rx*y1A*y1A) - (ry*ry*x1A*x1A)) / ((rx*rx*y1A*y1A) + (ry*ry*x1A*x1A)) );
705         if(fA==fS){
706             val *= -1;
707         }
708         matrix = new double[2][1];
709         matrix[0][0] = rx*y1A/ry;
710         matrix[1][0] = -1*(ry*x1A/rx);
711         Matrix m3 = new Matrix(matrix);
712         Matrix centerPointA = MatrixMath.multiply(m3, val);
713 
714         //find center points
715         matrix = new double[2][2];
716         matrix[0][0] = Math.cos(varphi);
717         matrix[0][1] = -1 * Math.sin(varphi);
718         matrix[1][0] = Math.sin(varphi);
719         matrix[1][1] = Math.cos(varphi);
720         Matrix m4 = new Matrix(matrix);
721         matrix = new double[2][1];
722         matrix[0][0] = (x1+x2)/2.0;
723         matrix[1][0] = (y1+y2)/2.0;
724         Matrix m5 = new Matrix(matrix);
725         Matrix centerPoint = MatrixMath.multiply(m4, centerPointA);
726         centerPoint = MatrixMath.add(centerPoint, m5);
727         return centerPoint;
```

```
728     }
729
730     private Matrix ensureRadii(double x1, double y1, double x2, double y2, double rx, double ry, double varphi, int fA, int fS){
731         double matrix[][] = new double[2][2];
732         matrix[0][0] = Math.cos(varphi);
733         matrix[0][1] = Math.sin(varphi);
734         matrix[1][0] = Math.sin(varphi) * -1;
735         matrix[1][1] = Math.cos(varphi);
736         Matrix m1 = new Matrix(matrix);
737         matrix = new double[2][1];
738         matrix[0][0] = (x1-x2)/2.0;
739         matrix[1][0] = (y1-y2)/2.0;
740         Matrix m2 = new Matrix(matrix);
741         Matrix coordinateA = MatrixMath.multiply(m1, m2);
742         double x1A = coordinateA.getMatrix()[0][0];
743         double y1A = coordinateA.getMatrix()[1][0];
744
745         rx = Math.abs(rx);
746         ry = Math.abs(ry);
747         double radii = Math.pow(x1A, 2)/Math.pow(rx, 2) + Math.pow(y1A, 2)/Math.pow(ry, 2);
748         if(radii>1){
749             rx = Math.sqrt(radii) * rx;
750             ry = Math.sqrt(radii) * ry;
751         }
752         double r[][] = new double[2][1];
753         r[0][0] = rx;
754         r[1][0] = ry;
755         return new Matrix(r);
756     }
757
758     private Point2D.Double mirrorControlPoint(Point2D.Double prevCtrlPoint, Point2D.Double curStart) {
759         Point2D.Double res = new Point2D.Double();
760         res.x = curStart.x - (prevCtrlPoint.x - curStart.x);
761         res.y = curStart.y - (prevCtrlPoint.y - curStart.y);
762         return res;
763     }
764 }
```

Listing A.9: SVG.java

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package engine;
7
8  /**
9   *
10  * @author Ermengarde
11  */
12 public class SVG extends Element{
13     public SVG(String name, String attributes) {
14         super(name, attributes);
15     }
16
17     @Override
18     public double getMaxX() {
19         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools |
               Templates.
20     }
21
22     @Override
23     public double getMinX() {
24         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools |
               Templates.
25     }
26
27     @Override
28     public double getMaxY() {
29         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools |
               Templates.
30     }
31
32     @Override
33     public double getMinY() {
34         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools |
               Templates.
35     }
36
37 }
```

Listing A.10: SVGParser.java

```
1  package engine;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileReader;
6  import java.io.IOException;
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * @author Albert - 2014730007
12  */
```

```java
public class SVGParser {
    public static final double AREA_LOWER_BOUND = 2500;
    public static final double LENGTH_LOWER_BOUND = 100;

    private File svgFile;
    private BufferedReader br;
    private List<Element> elements;
    private List<Element> unprocessedElements;
    private double svgWidth;
    private double svgHeight;

    public SVGParser(File svgFile) {
        this.elements = new ArrayList<Element>();
        this.unprocessedElements = new ArrayList<Element>();
        this.svgFile = svgFile;
        try{
            this.br = new BufferedReader(new FileReader(svgFile));
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }

    public void parseFile(){
        //memindahkan isi file SVG ke String fileContent
        String fileContent = "";
        String buffer;
        try{
            while((buffer = br.readLine()) != null){
                fileContent += buffer;
            }
        }
        catch(IOException e){
            e.printStackTrace();
        }

        //adding elements to arraylist
        String tags[] = fileContent.split("<");
        for(int i=0;i<tags.length;i++){
            String elementName = "";
            String elementAttributes = "";
            boolean flag = true;    //flag = true adalah fase mengambil elementName
            boolean inQuotes = false;
            char cur;
            for(int j=0;j<tags[i].length();j++){
                if(j!=0 && tags[i].charAt(j-1)=='/' && tags[i].charAt(j)=='>'){
                    break;
                }
                cur = tags[i].charAt(j);
                if(flag){
                    if(cur != '␣' && cur != '\t'){
                        elementName += cur;
                    }
                    else{
                       flag = false;
                    }
                }
                else{
                    if(inQuotes){
                        if(cur == ','){
                            elementAttributes += '␣';
                        }
                        else{
                            elementAttributes += cur;
                        }
                        if(cur == '\"'){
                            inQuotes = !inQuotes;
                        }
                    }
                    else{
                        if(cur != '␣' && cur != '\t'){
                            elementAttributes += cur;
                            if(cur == '\"'){
                                inQuotes = !inQuotes;
                            }
                        }
                    }
                }
            }
            if(isTargetElement(elementName)){
                if(elementName.equalsIgnoreCase("rect")){
                    Rectangle rect = new Rectangle(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
                    if(rect.getHorizontalLength()==0 || rect.getVerticalLength()==0){
                        if(rect.getBoundingRectArea()<LENGTH_LOWER_BOUND){
                            unprocessedElements.add(rect);
                        }
                        else{
                            elements.add(rect);
                        }
                    }
                    else{
                        if(rect.getBoundingRectArea()<AREA_LOWER_BOUND){
                            unprocessedElements.add(rect);
                        }
                        else{
                            elements.add(rect);
                        }
                    }
                }
```

```
112                 else if(elementName.equalsIgnoreCase("circle")){
113                     Circle circle = new Circle(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
114                     if(circle.getHorizontalLength()==0 || circle.getVerticalLength()==0){
115                         if(circle.getBoundingRectArea()<LENGTH_LOWER_BOUND){
116                             unprocessedElements.add(circle);
117                         }
118                         else{
119                             elements.add(circle);
120                         }
121                     }
122                     else{
123                         if(circle.getBoundingRectArea()<AREA_LOWER_BOUND){
124                             unprocessedElements.add(circle);
125                         }
126                         else{
127                             elements.add(circle);
128                         }
129                     }
130                 }
131                 else if(elementName.equalsIgnoreCase("ellipse")){
132                     Ellipse ellipse = new Ellipse(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
133                     if(ellipse.getHorizontalLength()==0 || ellipse.getVerticalLength()==0){
134                         if(ellipse.getBoundingRectArea()<LENGTH_LOWER_BOUND){
135                             unprocessedElements.add(ellipse);
136                         }
137                         else{
138                             elements.add(ellipse);
139                         }
140                     }
141                     else{
142                         if(ellipse.getBoundingRectArea()<AREA_LOWER_BOUND){
143                             unprocessedElements.add(ellipse);
144                         }
145                         else{
146                             elements.add(ellipse);
147                         }
148                     }
149                 }
150                 else if(elementName.equalsIgnoreCase("line")){
151                     Line line = new Line(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
152                     if(line.getHorizontalLength()==0 || line.getVerticalLength()==0){
153                         if(line.getBoundingRectArea()<LENGTH_LOWER_BOUND){
154                             unprocessedElements.add(line);
155                         }
156                         else{
157                             elements.add(line);
158                         }
159                     }
160                     else{
161                         if(line.getBoundingRectArea()<AREA_LOWER_BOUND){
162                             unprocessedElements.add(line);
163                         }
164                         else{
165                             elements.add(line);
166                         }
167                     }
168                 }
169                 else if(elementName.equalsIgnoreCase("polygon")){
170                     Polygon polygon = new Polygon(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
171                     if(polygon.getHorizontalLength()==0 || polygon.getVerticalLength()==0){
172                         if(polygon.getBoundingRectArea()<LENGTH_LOWER_BOUND){
173                             unprocessedElements.add(polygon);
174                         }
175                         else{
176                             elements.add(polygon);
177                         }
178                     }
179                     else{
180                         if(polygon.getBoundingRectArea()<AREA_LOWER_BOUND){
181                             unprocessedElements.add(polygon);
182                         }
183                         else{
184                             elements.add(polygon);
185                         }
186                     }
187                 }
188                 else if(elementName.equalsIgnoreCase("polyline")){
189                     Polyline polyline = new Polyline(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
190                     if(polyline.getHorizontalLength()==0 || polyline.getVerticalLength()==0){
191                         if(polyline.getBoundingRectArea()<LENGTH_LOWER_BOUND){
192                             unprocessedElements.add(polyline);
193                         }
194                         else{
195                             elements.add(polyline);
196                         }
197                     }
198                     else{
199                         if(polyline.getBoundingRectArea()<AREA_LOWER_BOUND){
200                             unprocessedElements.add(polyline);
201                         }
202                         else{
203                             elements.add(polyline);
204                         }
205                     }
206                 }
207                 else if(elementName.equalsIgnoreCase("path")){
208                     Path path = new Path(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
209                     if(path.getHorizontalLength()==0 || path.getVerticalLength()==0){
210                         if(path.getBoundingRectArea()<LENGTH_LOWER_BOUND){
```

```
                            unprocessedElements.add(path);
                        }
                        else{
                            elements.add(path);
                        }
                    }
                    else{
                        if(path.getBoundingRectArea()<AREA_LOWER_BOUND){
                            unprocessedElements.add(path);
                        }
                        else{
                            elements.add(path);
                        }
                    }
                }
            }
            else if(elementName.equalsIgnoreCase("svg")){
                SVG svg = new SVG(elementName, elementAttributes.substring(0, elementAttributes.length()-2));
                this.svgWidth = Double.parseDouble(svg.getAttributes().get("width"));
                this.svgHeight = Double.parseDouble(svg.getAttributes().get("height"));
            }
        }
    }

    public double getSvgWidth() {
        return svgWidth;
    }

    public void setSvgWidth(double svgWidth) {
        this.svgWidth = svgWidth;
    }

    public double getSvgHeight() {
        return svgHeight;
    }

    public void setSvgHeight(double svgHeight) {
        this.svgHeight = svgHeight;
    }

    public List<Element> getElements() {
        return elements;
    }

    public List<Element> getUnprocessedElements() {
        return unprocessedElements;
    }

    private boolean isTargetElement(String elementName) {
        if(elementName.equalsIgnoreCase("rect") || elementName.equalsIgnoreCase("circle")
                || elementName.equalsIgnoreCase("ellipse")|| elementName.equalsIgnoreCase("line")
                || elementName.equalsIgnoreCase("polygon") || elementName.equalsIgnoreCase("polyline")
                || elementName.equalsIgnoreCase("path")){
            return true;
        }
        else return false;
    }
}
```

Listing A.11: Vertex.java

```
package engine;

import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.List;

/**
 * @author Albert - 2014730007
 */
public class Vertex {
    public static final double EPSILON = 0.0000001;

    private int number;
    private Point2D.Double location;
    private int degree;
    private List<Integer> displayNumbers;

    public Vertex(int number, Point2D.Double location){
        this.number = number;
        this.location = location;
        this.degree = 0;
        this.displayNumbers = new ArrayList<>();
    }

    public Vertex(Point2D.Double location){
        this.location = location;
        this.displayNumbers = new ArrayList<>();
    }

    public List<Integer> getDisplayNumbers() {
        return displayNumbers;
    }

    public int getNumber() {
        return number;
    }

```

```
38      public void setNumber(int number) {
39          this.number = number;
40      }
41
42      public Point2D.Double getLocation() {
43          return location;
44      }
45
46      public void setLocation(Point2D.Double location) {
47          this.location = location;
48      }
49
50      public void setDegree(int newDegree){
51          this.degree = newDegree;
52      }
53
54      public int getDegree(){
55          return this.degree;
56      }
57
58      @Override
59      public boolean equals(Object o){
60          if(o instanceof Vertex){
61              Vertex v = (Vertex)o;
62              if(Math.abs(v.location.x-this.location.x)<EPSILON && Math.abs(v.location.y-this.location.y)<EPSILON)return true;
63              else return false;
64          }
65          else return false;
66      }
67  }
```

Listing A.12: Edge.java

```
1   package engine;
2
3   /**
4    * @author Albert - 2014730007
5    */
6   public class Edge {
7       private Vertex from;
8       private Vertex to;
9       private boolean helpLine;
10
11      public Edge(Vertex from, Vertex to) {
12          this.from = from;
13          this.to = to;
14          this.helpLine = false;
15      }
16
17      public Vertex getFrom() {
18          return from;
19      }
20
21      public void setFrom(Vertex from) {
22          this.from = from;
23      }
24
25      public Vertex getTo() {
26          return to;
27      }
28
29      public void setTo(Vertex to) {
30          this.to = to;
31      }
32
33      public boolean isHelpLine() {
34          return helpLine;
35      }
36
37      public void setHelpLine(boolean helpLine) {
38          this.helpLine = helpLine;
39      }
40
41      @Override
42      public boolean equals(Object o){
43          if(o instanceof Edge){
44              Edge e = (Edge)o;
45              if((this.from.equals(e.from) && this.to.equals(e.to)) || (this.from.equals(e.to) && this.to.equals(e.from))){
46                  return true;
47              }
48              else{
49                  return false;
50              }
51          }
52          else return false;
53      }
54
55      public String toString(){
56          return this.from.getLocation()+"␣"+this.to.getLocation();
57      }
58  }
```

Listing A.13: Graph.java

```
1   package engine;
2
```

```java
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
 * @author Albert - 2014730007
 */
public class Graph {
    private List<Vertex> vertices;
    private List<Edge> edges;
    private List<List<Vertex>> adjList;

    public Graph(){
        this.vertices = new ArrayList<Vertex>();
        this.edges = new ArrayList<Edge>();
        this.adjList = new ArrayList<List<Vertex>>();
    }

    /** Represents visited edge **/
    private class Node{
        private int first;
        private boolean second;
    }

    public List<Node>[] copyGraph(){
        List<Node> graph[] = new ArrayList[this.vertices.size()];
        for(int i=0;i<adjList.size();i++){
            graph[i] = new ArrayList<>();
            for(int j=0;j<adjList.get(i).size();j++){
                Vertex v = adjList.get(i).get(j);
                Node n = new Node();
                n.first = v.getNumber();
                n.second = false;
                graph[i].add(n);
            }
        }
        return graph;
    }

    /** Hierholzer's Algorithm Implementation**/
    public void hierholzer(){
        int res = this.makeEuler(); //res = 0 able to make euler circuit, res = 2 able to make euler path
        List<Integer> eulerPath = new ArrayList<>();
        int start = -1;
        if(res==2){
            for(int i=0;i<vertices.size();i++){
                if(vertices.get(i).getDegree()%2==1){
                    start = vertices.get(i).getNumber();
                    break;
                }
            }
        }
        else{
            start = 0;
        }
        List<Node>[] graph = copyGraph();
        doHierholzer(graph, start, eulerPath);
        eulerPath.add(start);
        if(eulerPath.size()>1){
            for(int i=0;i<eulerPath.size();i++){
                this.vertices.get(eulerPath.get(i)).getDisplayNumbers().add(i+1);
            }
        }
    }

    private void doHierholzer(List<Node>[] graph, int u, List<Integer> eulerPath) {
        if(graph.length==0)return ;
        for(int i=0;i<graph[u].size();i++){
            Node neigh = graph[u].get(i);
            if(!neigh.second){
                neigh.second = true;
                for(int j=0;j<graph[neigh.first].size();j++){
                    Node neighCur = graph[neigh.first].get(j);
                    if(neighCur.first==u && !neighCur.second){
                        neighCur.second = true;
                        break;
                    }
                }
                doHierholzer(graph, neigh.first, eulerPath);
                eulerPath.add(neigh.first);
            }
        }
    }

    public void addVertex(Vertex newVertex){
        if(!this.vertices.contains(newVertex)){
            newVertex.setNumber(this.vertices.size());
            this.vertices.add(newVertex);
            this.adjList.add(new ArrayList<Vertex>());
        }
        else{
            newVertex.setNumber(this.vertices.get(this.vertices.indexOf(newVertex)).getNumber());
        }
    }

    public boolean addEdge(Edge newEdge){
        if(!this.edges.contains(newEdge) && !newEdge.getFrom().equals(newEdge.getTo())){
            this.edges.add(newEdge);
```

```
            Vertex firstVertex = newEdge.getFrom();
            Vertex secondVertex = newEdge.getTo();
            if(this.vertices.contains(firstVertex)){
                firstVertex = this.vertices.get(this.vertices.indexOf(firstVertex));
            }
            if(this.vertices.contains(secondVertex)){
                secondVertex = this.vertices.get(this.vertices.indexOf(secondVertex));
            }
            firstVertex.setDegree(firstVertex.getDegree()+1);
            secondVertex.setDegree(secondVertex.getDegree()+1);
            this.adjList.get(firstVertex.getNumber()).add(secondVertex);
            this.adjList.get(secondVertex.getNumber()).add(firstVertex);
            return true;
        }
        else return false;
    }

    public void removeEdge(Edge e){
        if(this.edges.contains(e)){
            this.edges.remove(e);
            Vertex firstVertex = e.getFrom();
            Vertex secondVertex = e.getTo();
            if(this.vertices.contains(firstVertex)){
                firstVertex = this.vertices.get(this.vertices.indexOf(firstVertex));
            }
            if(this.vertices.contains(secondVertex)){
                secondVertex = this.vertices.get(this.vertices.indexOf(secondVertex));
            }
            firstVertex.setDegree(firstVertex.getDegree()-1);
            secondVertex.setDegree(secondVertex.getDegree()-1);
            this.adjList.get(firstVertex.getNumber()).remove(secondVertex);
            this.adjList.get(secondVertex.getNumber()).remove(firstVertex);
        }
    }

    public List<Vertex> getVertices() {
        return vertices;
    }

    public void setVertices(ArrayList<Vertex> vertices) {
        this.vertices = vertices;
    }

    public List<Edge> getEdges() {
        return edges;
    }

    public void setEdges(ArrayList<Edge> edges) {
        this.edges = edges;
    }

    public List<List<Vertex>> getAdjList(){
        return this.adjList;
    }

    private void dfsRemove(List<Node>[] adjList, Vertex u, List<Edge> smallerArea){
        int cur = u.getNumber();
        for(int i=0;i<adjList[cur].size();i++){
            Node v = adjList[cur].get(i);
            if(!v.second){
                v.second = true;
                Edge e = new Edge(u, this.vertices.get(v.first));
                e = edges.get(edges.indexOf(e));
                if(!smallerArea.contains(e)){
                    smallerArea.add(e);
                }
                dfsRemove(adjList, this.vertices.get(v.first), smallerArea);
            }
        }
    }

    private void chooseLargerArea(Edge e){
        boolean visited[] = new boolean[this.vertices.size()];
        int area1 = dfsCountEdge(e.getFrom().getNumber(), visited);
        visited = new boolean[this.vertices.size()];
        int area2 = dfsCountEdge(e.getTo().getNumber(), visited);
        List<Edge> smallerArea = new ArrayList<>();
        List<Node>[] graph = copyGraph();
        if(area1>=area2){
            dfsRemove(graph, e.getTo(), smallerArea);
        }
        else{
            dfsRemove(graph, e.getFrom(), smallerArea);
        }
        for(Edge edge: smallerArea){
            Vertex eFrom = this.vertices.get(this.vertices.indexOf(edge.getFrom()));
            Vertex eTo = this.vertices.get(this.vertices.indexOf(edge.getTo()));
            this.adjList.get(eFrom.getNumber()).remove(eTo);
            this.adjList.get(eTo.getNumber()).remove(eFrom);
            this.vertices.get(eFrom.getNumber()).setDegree(this.vertices.get(eFrom.getNumber()).getDegree()-1);
            this.vertices.get(eTo.getNumber()).setDegree(this.vertices.get(eTo.getNumber()).getDegree()-1);
            this.edges.get(this.edges.indexOf(edge)).setHelpLine(true);
        }
    }

    private void makeHelpingLine(Edge e) {
        boolean isBridge = false;
        if(checkBridge(e)!=-1){
            isBridge = true;
```

```
201         }
202         Vertex eFrom = e.getFrom();
203         Vertex eTo = e.getTo();
204         this.adjList.get(eFrom.getNumber()).remove(eTo);
205         this.adjList.get(eTo.getNumber()).remove(eFrom);
206         this.vertices.get(eFrom.getNumber()).setDegree(this.vertices.get(eFrom.getNumber()).getDegree()-1);
207         this.vertices.get(eTo.getNumber()).setDegree(this.vertices.get(eTo.getNumber()).getDegree()-1);
208         this.edges.get(this.edges.indexOf(e)).setHelpLine(true);
209         if(isBridge){
210             chooseLargerArea(e);
211         }
212     }
213 
214     public int makeEuler(){
215         int numOfOddVertices = 0;
216         for(int i=0;i<this.vertices.size();i++){
217             if(this.vertices.get(i).getDegree()%2==1){
218                 numOfOddVertices += 1;
219             }
220         }
221         while(numOfOddVertices!=0 && numOfOddVertices!=2){
222             Edge best = null;
223             boolean first = true;
224             for(int i=0;i<this.edges.size();i++){
225                 Edge e = this.edges.get(i);
226                 if(!e.isHelpLine() && (this.vertices.get(e.getFrom().getNumber()).getDegree()%2==1 || this.vertices.get(e.getTo().
                       getNumber()).getDegree()%2==1)){
227                     if(first){
228                         best = e;
229                         first = false;
230                     }
231                     else{
232                         best = compareEdge(best, e);
233                     }
234                 }
235             }
236             makeHelpingLine(best);
237             numOfOddVertices = 0;
238             for(int i=0;i<this.vertices.size();i++){
239                 if(this.vertices.get(i).getDegree()%2==1){
240                     numOfOddVertices += 1;
241                 }
242             }
243         }
244         return numOfOddVertices;
245     }
246 
247     private Edge compareEdge(Edge e1, Edge e2){
248         int isBridgeE1 = checkBridge(e1);
249         int isBridgeE2 = checkBridge(e2);
250         int numConnectedOddVertexE1 = 0;
251         int numConnectedOddVertexE2 = 0;
252 
253         if(this.vertices.get(e1.getFrom().getNumber()).getDegree()%2==1)numConnectedOddVertexE1++;
254         if(this.vertices.get(e1.getTo().getNumber()).getDegree()%2==1)numConnectedOddVertexE1++;
255         if(this.vertices.get(e2.getFrom().getNumber()).getDegree()%2==1)numConnectedOddVertexE2++;
256         if(this.vertices.get(e2.getTo().getNumber()).getDegree()%2==1)numConnectedOddVertexE2++;
257 
258         if(numConnectedOddVertexE1==2 && numConnectedOddVertexE2==2){
259             if(isBridgeE1==-1 && isBridgeE2==-1){
260                 return e1;
261             }
262             else if(isBridgeE1==-1 && isBridgeE2!=-1){
263                 return e1;
264             }
265             else if(isBridgeE1!=-1 && isBridgeE2==-1){
266                 return e2;
267             }
268             else{
269                 return (isBridgeE1>=isBridgeE2)? e1: e2;
270             }
271         }
272         else if(numConnectedOddVertexE1==2 && numConnectedOddVertexE2!=2){
273             return e1;
274         }
275         else if(numConnectedOddVertexE1!=2 && numConnectedOddVertexE2==2){
276             return e2;
277         }
278         else{
279             if(isBridgeE1==-1 && isBridgeE2==-1){
280                 return e1;
281             }
282             else if(isBridgeE1==-1 && isBridgeE2!=-1){
283                 return e1;
284             }
285             else if(isBridgeE1!=-1 && isBridgeE2==-1){
286                 return e2;
287             }
288             else{
289                 return (isBridgeE1>=isBridgeE2)? e1: e2;
290             }
291         }
292     }
293 
294     private int dfsCountEdge(int cur, boolean visited[]){
295         visited[cur] = true;
296         int count = adjList.get(cur).size();
297         for(int i=0;i<adjList.get(cur).size();i++){
298             Vertex v = adjList.get(cur).get(i);
```

```
            if(!visited[v.getNumber()]){
                count += dfsCountEdge(v.getNumber(), visited);
            }
        }
        return count/2;
    }

    private int dfsCountVertex(int cur, boolean visited[]){
        visited[cur] = true;
        int count = 1;
        for(int i=0;i<adjList.get(cur).size();i++){
            Vertex v = adjList.get(cur).get(i);
            if(!visited[v.getNumber()]){
                count += dfsCountVertex(v.getNumber(), visited);
            }
        }
        return count;
    }

    private int checkBridge(Edge e) {
        boolean visited[] = new boolean[this.vertices.size()];
        int before = dfsCountVertex(e.getFrom().getNumber(), visited);
        this.adjList.get(e.getFrom().getNumber()).remove(e.getTo());
        this.adjList.get(e.getTo().getNumber()).remove(e.getFrom());
        visited = new boolean[this.vertices.size()];
        int after = dfsCountVertex(e.getFrom().getNumber(), visited);
        visited = new boolean[this.vertices.size()];
        int countArea1 = dfsCountEdge(e.getFrom().getNumber(), visited);
        visited = new boolean[this.vertices.size()];
        int countArea2 = dfsCountEdge(e.getTo().getNumber(), visited);
        this.adjList.get(e.getFrom().getNumber()).add(e.getTo());
        this.adjList.get(e.getTo().getNumber()).add(e.getFrom());
        if(before==after){
            return -1;
        }
        else{
            return (countArea1>countArea2)? countArea1: countArea2;
        }
    }

    /** Method for development only **/
    private void printAdjList() {
        for(int i=0;i<adjList.size();i++){
            Vertex vi = this.vertices.get(i);
            if(vi.getDegree()==0)continue;
            System.out.println("Vertex␣"+vi.getNumber()+"␣Has␣Degree␣"+vi.getDegree()+"␣Located␣At␣"+vi.getLocation());
            System.out.println("Neighbour␣of␣Vertex␣"+vi.getNumber());
            for(int j=0;j<adjList.get(i).size();j++){
                Vertex vj = adjList.get(i).get(j);
                System.out.println("\t\t"+vj.getNumber()+"␣"+vj.getLocation());
            }
        }
    }
}
```

Listing A.14: Matrix.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package engine;

/**
 *
 * @author Ermengarde
 */
public class Matrix {
    private double matrix[][];
    private int rows;
    private int cols;

    public Matrix(double matrix[][]){
        this.matrix = matrix;
        this.rows = this.matrix.length;
        if(this.matrix.length>0){
            this.cols = matrix[0].length;
        }
    }

    public double[][] getMatrix() {
        return matrix;
    }

    public void setMatrix(double[][] matrix) {
        this.matrix = matrix;
    }

    public int getRows() {
        return rows;
    }

    public void setRows(int rows) {
        this.rows = rows;
    }

    public int getCols() {
```

```
42          return cols;
43      }
44
45      public void setCols(int cols) {
46          this.cols = cols;
47      }
48
49      public boolean isVector() {
50          if(getRows() == 1){
51              return true;
52          }
53          else{
54              return getCols()==1;
55          }
56      }
57
58      public Double[] toPackedArray() {
59          Double result[] = new Double[getRows() * getCols()];
60          int index = 0;
61          for (int r=0;r<getRows();r++) {
62              for (int c=0;c<getCols();c++) {
63                  result[index++] = this.matrix[r][c];
64              }
65          }
66          return result;
67      }
68  }
```

Listing A.15: MatrixMath.java

```
1   /*
2    * To change this license header, choose License Headers in Project Properties.
3    * To change this template file, choose Tools | Templates
4    * and open the template in the editor.
5    */
6   package engine;
7
8   /**
9    *
10   * @author Ermengarde
11   */
12  public class MatrixMath {
13      private MatrixMath(){
14
15      }
16
17      public static Matrix add(Matrix a, Matrix b){
18          if(a.getRows()==b.getRows() && a.getCols()==b.getCols()){
19              double result[][] = new double[a.getRows()][a.getCols()];
20              for(int i=0;i<result.length;i++){
21                  for(int j=0;j<result[i].length;j++){
22                      result[i][j] = a.getMatrix()[i][j] + b.getMatrix()[i][j];
23                  }
24              }
25              return new Matrix(result);
26          }
27          else return null;
28      }
29
30      public static Matrix multiply(Matrix a, double b) {
31          double result[][] = new double[a.getRows()][a.getCols()];
32          for(int row=0;row<a.getRows();row++){
33                  for(int col=0;col<a.getCols();col++){
34                          result[row][col] = a.getMatrix()[row][col] * b;
35                  }
36          }
37          return new Matrix(result);
38      }
39
40      public static Matrix multiply(Matrix a, Matrix b){
41          if(a.getCols()==b.getRows()){
42              double result[][] = new double[a.getRows()][b.getCols()];
43              for(int i=0;i<a.getRows();i++){
44                  for(int j=0;j<b.getCols();j++){
45                      double val = 0;
46                      for(int k=0;k<a.getCols();k++){
47                          val += a.getMatrix()[i][k] * b.getMatrix()[k][j];
48                      }
49                      result[i][j] = val;
50                  }
51              }
52              return new Matrix(result);
53          }
54          else{
55              return null;
56          }
57      }
58
59      public static double dotProduct(Matrix a, Matrix b){
60          if(a.isVector() && b.isVector()){
61              Double aArray[] = a.toPackedArray();
62              Double bArray[] = b.toPackedArray();
63              if(aArray.length==bArray.length){
64                  double result = 0;
65                  for(int i=0;i<aArray.length;i++) {
66              result += aArray[i] * bArray[i];
67          }
68                  return result;
```

```
            }
            else{
                return Double.NaN;
            }
        }
        else return Double.NaN;
    }

    public static double vectorLength(Matrix input) {
        if(input.isVector()){
            Double v[] = input.toPackedArray();
            double rtn = 0.0;
            for(int i=0;i<v.length;i++) {
                rtn += Math.pow(v[i], 2);
            }
            return Math.sqrt(rtn);
        }
        else{
            return Double.NaN;
        }
    }
}
```

Listing A.16: GraphMaker.java

```
package engine;

import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

/**
 * @author Albert - 2014730007
 */
public class GraphMaker {
    private Graph result;
    private List<Element> elements;

    public GraphMaker(List<Element> elements){
        this.elements = elements;
    }

    public Graph getResult() {
        this.result = new Graph();
        int elementsSize = this.elements.size();
        for(int i=0;i<elementsSize;i++){
            Element cur = elements.get(i);
            if(cur.getName().equals("rect")){
                makeRect(cur);
            }
            else if(cur.getName().equals("circle")){
                makeCircle(cur);
            }
            else if(cur.getName().equals("ellipse")){
                makeEllipse(cur);
            }
            else if(cur.getName().equals("line")){
                makeLine(cur);
            }
            else if(cur.getName().equals("polygon")){
                makePolygon(cur);
            }
            else if(cur.getName().equals("polyline")){
                makePolyline(cur);
            }
            else if(cur.getName().equals("path")){
                makePath(cur);
            }
        }
        handleIntersections();
        return result;
    }

    private void handleIntersections() {
        Set<String> pointIntersection = new HashSet<>();
        boolean intersectionFound = true;
        String stringIntersection = "";
        while(intersectionFound){
            intersectionFound = false;
            for(int i=0;i<result.getEdges().size();i++){
                Edge e1 = result.getEdges().get(i);
                Line2D.Double l1 = new Line2D.Double(e1.getFrom().getLocation(), e1.getTo().getLocation()); //vertical line has
                    undefined slope
                double x1E1 = e1.getFrom().getLocation().x;
                double y1E1 = e1.getFrom().getLocation().y;
                double x2E1 = e1.getTo().getLocation().x;
                double y2E1 = e1.getTo().getLocation().y;
                Double m1 = (y2E1 - y1E1)/(x2E1 - x1E1);
                double equationE1[] = {m1, (m1 * x1E1 - y1E1)*-1}; //[0] = x, [1] = constant
                boolean isVerticalLineE1 = false;
                if(x1E1==x2E1)isVerticalLineE1 = true;
                for(int j=i+1;j<result.getEdges().size();j++){
                    Edge e2 = result.getEdges().get(j);
```

```
72                  if(e1.getFrom().equals(e2.getFrom()) || e1.getFrom().equals(e2.getTo()) || e1.getTo().equals(e2.getFrom()) ||
                        e1.getTo().equals(e2.getTo())){
73                      continue;
74                  }
75                  Line2D.Double l2 = new Line2D.Double(e2.getFrom().getLocation(), e2.getTo().getLocation()); //vertical line
                        has undefined slope
76                  if(l1.intersectsLine(l2)){
77                      double x1E2 = e2.getFrom().getLocation().x;
78                      double y1E2 = e2.getFrom().getLocation().y;
79                      double x2E2 = e2.getTo().getLocation().x;
80                      double y2E2 = e2.getTo().getLocation().y;
81                      Double m2 = (y2E2 - y1E2)/(x2E2 - x1E2);
82                      double equationE2[] = {m2, (m2 * x1E2 - y1E2)*-1}; //[0] = x, [1] = constant
83                      boolean isVerticalLineE2 = false;
84                      if(x1E2==x2E2)isVerticalLineE2 = true;
85                      double x = 0;
86                      double y = 0;
87                      if(isVerticalLineE1){
88                          x = x1E1;
89                          y = equationE2[0] * x + (equationE2[1]);
90                      }
91                      else if(isVerticalLineE2){
92                          x = x1E2;
93                          y = equationE1[0] * x + (equationE1[1]);
94                      }
95                      else{
96                          x = (equationE1[1]-equationE2[1])/(equationE2[0]-equationE1[0]);
97                          y = equationE1[0] * x + (equationE1[1]);
98                      }
99                      if(m1.equals(m2)){
100                         continue;
101                     }
102                     x = Math.round(x*100.0)/100.0;
103                     y = Math.round(y*100.0)/100.0;
104                     Point2D.Double intersection = new Point2D.Double(x, y);
105                     stringIntersection = e1.toString()+"ـ-ـ"+e2.toString();
106                     if(!Double.isNaN(x) && !Double.isNaN(y) && !pointIntersection.contains(stringIntersection)){
107                         pointIntersection.add(stringIntersection);
108                         makeIntersection(intersection, e1, e2);
109                         intersectionFound = true;
110                         break;
111                     }
112                 }
113             }
114             if(intersectionFound)break;
115         }
116     }
117 }
118
119 private void makeIntersection(Point2D.Double p, Edge e1, Edge e2){
120     double x = p.x;
121     double y = p.y;
122     Vertex newVertex = new Vertex(new Point2D.Double(x, y));
123     if(result.getVertices().contains(newVertex)){
124         newVertex = result.getVertices().get(result.getVertices().indexOf(newVertex));
125     }
126     else{
127         result.addVertex(newVertex);
128     }
129     if(result.addEdge(new Edge(e1.getFrom(), newVertex)) && result.addEdge(new Edge(newVertex, e1.getTo()))){
130         result.removeEdge(e1);
131     }
132     if(result.addEdge(new Edge(e2.getFrom(), newVertex)) && result.addEdge(new Edge(newVertex, e2.getTo()))){
133         result.removeEdge(e2);
134     }
135 }
136
137 private void makeRect(Element cur) {
138     double x = 0;
139     double y = 0;
140     double width = 0;
141     double height = 0;
142     String found = cur.getAttributes().get("x");
143     if(found != null){
144         x = Double.parseDouble(found);
145     }
146     found = cur.getAttributes().get("y");
147     if(found != null){
148         y = Double.parseDouble(found);
149     }
150     found = cur.getAttributes().get("width");
151     if(found != null){
152         width = Double.parseDouble(found);
153     }
154     found = cur.getAttributes().get("height");
155     if(found != null){
156         height = Double.parseDouble(found);
157     }
158
159     Vertex v1, v2, v3, v4;
160     v1 = new Vertex(new Point2D.Double(x, y));
161     v2 = new Vertex(new Point2D.Double(x+width, y));
162     v3 = new Vertex(new Point2D.Double(x+width, y+height));
163     v4 = new Vertex(new Point2D.Double(x, y+height));
164
165     this.result.addVertex(v1);
166     this.result.addVertex(v2);
167     this.result.addVertex(v3);
168     this.result.addVertex(v4);
```

```
169         this.result.addEdge(new Edge(v1, v2));
170         this.result.addEdge(new Edge(v2, v3));
171         this.result.addEdge(new Edge(v3, v4));
172         this.result.addEdge(new Edge(v1, v4));
173     }
174 
175     private void makeEllipse(Element cur) {
176         double cx = 0;
177         double cy = 0;
178         double rx = 0;
179         double ry = 0;
180         String found = cur.getAttributes().get("cx");
181         if(found != null){
182             cx = Double.parseDouble(found);
183         }
184         found = cur.getAttributes().get("cy");
185         if(found != null){
186             cy = Double.parseDouble(found);
187         }
188         found = cur.getAttributes().get("rx");
189         if(found != null){
190             rx = Double.parseDouble(found);
191         }
192         found = cur.getAttributes().get("ry");
193         if(found != null){
194             ry = Double.parseDouble(found);
195         }
196 
197         Vertex v1, v2, v3, v4, v5, v6, v7, v8;
198         v1 = new Vertex(new Point2D.Double(cx, cy-ry));
199         v2 = new Vertex(new Point2D.Double(cx+(rx*Math.cos(Math.toRadians(45))), cy-(ry*Math.sin(Math.toRadians(45)))));
200         v3 = new Vertex(new Point2D.Double(cx+rx, cy));
201         v4 = new Vertex(new Point2D.Double(cx+(rx*Math.cos(Math.toRadians(45))), cy+(ry*Math.sin(Math.toRadians(45)))));
202         v5 = new Vertex(new Point2D.Double(cx, cy+ry));
203         v6 = new Vertex(new Point2D.Double(cx-(rx*Math.cos(Math.toRadians(45))), cy+(ry*Math.sin(Math.toRadians(45)))));
204         v7 = new Vertex(new Point2D.Double(cx-rx, cy));
205         v8 = new Vertex(new Point2D.Double(cx-(rx*Math.cos(Math.toRadians(45))), cy-(ry*Math.sin(Math.toRadians(45)))));
206         this.result.addVertex(v1);
207         this.result.addVertex(v2);
208         this.result.addVertex(v3);
209         this.result.addVertex(v4);
210         this.result.addVertex(v5);
211         this.result.addVertex(v6);
212         this.result.addVertex(v7);
213         this.result.addVertex(v8);
214         Edge e1, e2, e3, e4, e5, e6, e7, e8;
215         e1 = new Edge(v1, v2);
216         e2 = new Edge(v2, v3);
217         e3 = new Edge(v3, v4);
218         e4 = new Edge(v4, v5);
219         e5 = new Edge(v5, v6);
220         e6 = new Edge(v6, v7);
221         e7 = new Edge(v7, v8);
222         e8 = new Edge(v8, v1);
223         this.result.addEdge(e1);
224         this.result.addEdge(e2);
225         this.result.addEdge(e3);
226         this.result.addEdge(e4);
227         this.result.addEdge(e5);
228         this.result.addEdge(e6);
229         this.result.addEdge(e7);
230         this.result.addEdge(e8);
231     }
232 
233     private void makePolygon(Element cur) {
234         List<Point2D.Double> points = new ArrayList<Point2D.Double>();
235         String found = cur.getAttributes().get("points");
236         if(found!= null){
237             String inputs[] = found.split("\\s+");
238             int inputsLen = inputs.length;
239             for(int i=0;i<inputsLen;i+=2){
240                 points.add(new Point2D.Double(Double.parseDouble(inputs[i]), Double.parseDouble(inputs[i+1])));
241             }
242 
243             int numOfPoints = points.size();
244             Vertex current = null;
245             Vertex previous = null;
246             Vertex first = null;
247             for(int i=0;i<numOfPoints;i++){
248                 current = new Vertex(points.get(i));
249                 this.result.addVertex(current);
250                 if(i>0){
251                     Edge e = new Edge(previous, current);
252                     this.result.addEdge(e);
253                 }
254                 else{
255                     first = current;
256                 }
257                 previous = current;
258             }
259             this.result.addEdge(new Edge(first, current));
260         }
261     }
262 
263     private void makeLine(Element cur) {
264         double x1 = 0;
265         double y1 = 0;
266         double x2 = 0;
267         double y2 = 0;
```

```
268         String found = cur.getAttributes().get("x1");
269         if(found != null){
270             x1 = Double.parseDouble(found);
271         }
272         found = cur.getAttributes().get("y1");
273         if(found != null){
274             y1 = Double.parseDouble(found);
275         }
276         found = cur.getAttributes().get("x2");
277         if(found != null){
278             x2 = Double.parseDouble(found);
279         }
280         found = cur.getAttributes().get("y2");
281         if(found != null){
282             y2 = Double.parseDouble(found);
283         }
284 
285         Vertex v1, v2;
286         v1 = new Vertex(new Point2D.Double(x1, y1));
287         v2 = new Vertex(new Point2D.Double(x2, y2));
288         this.result.addVertex(v1);
289         this.result.addVertex(v2);
290         this.result.addEdge(new Edge(v1, v2));
291     }
292 
293     private void makePolyline(Element cur) {
294         List<Point2D.Double> points = new ArrayList<Point2D.Double>();
295         String found = cur.getAttributes().get("points");
296         if(found!= null){
297             String inputs[] = found.split("\\s+");
298             int inputsLen = inputs.length;
299             for(int i=0;i<inputsLen;i+=2){
300                 points.add(new Point2D.Double(Double.parseDouble(inputs[i]), Double.parseDouble(inputs[i+1])));
301             }
302 
303             int numOfPoints = points.size();
304             Vertex current = null;
305             Vertex previous = null;
306             for(int i=0;i<numOfPoints;i++){
307                 current = new Vertex(points.get(i));
308                 this.result.addVertex(current);
309                 if(i>0){
310                     Edge e = new Edge(previous, current);
311                     this.result.addEdge(e);
312                 }
313                 previous = current;
314             }
315         }
316     }
317 
318     private void makeCircle(Element cur) {
319         double cx = 0;
320         double cy = 0;
321         double r = 0;
322         String found = cur.getAttributes().get("cx");
323         if(found != null){
324             cx = Double.parseDouble(found);
325         }
326         found = cur.getAttributes().get("cy");
327         if(found != null){
328             cy = Double.parseDouble(found);
329         }
330         found = cur.getAttributes().get("r");
331         if(found != null){
332             r = Double.parseDouble(found);
333         }
334 
335         Vertex v1, v2, v3, v4, v5, v6, v7, v8;
336         v1 = new Vertex(new Point2D.Double(cx, cy-r));
337         v2 = new Vertex(new Point2D.Double(cx+(r*Math.cos(Math.toRadians(45))), cy-(r*Math.sin(Math.toRadians(45)))));
338         v3 = new Vertex(new Point2D.Double(cx+r, cy));
339         v4 = new Vertex(new Point2D.Double(cx+(r*Math.cos(Math.toRadians(45))), cy+(r*Math.sin(Math.toRadians(45)))));
340         v5 = new Vertex(new Point2D.Double(cx, cy+r));
341         v6 = new Vertex(new Point2D.Double(cx-(r*Math.cos(Math.toRadians(45))), cy+(r*Math.sin(Math.toRadians(45)))));
342         v7 = new Vertex(new Point2D.Double(cx-r, cy));
343         v8 = new Vertex(new Point2D.Double(cx-(r*Math.cos(Math.toRadians(45))), cy-(r*Math.sin(Math.toRadians(45)))));
344         this.result.addVertex(v1);
345         this.result.addVertex(v2);
346         this.result.addVertex(v3);
347         this.result.addVertex(v4);
348         this.result.addVertex(v5);
349         this.result.addVertex(v6);
350         this.result.addVertex(v7);
351         this.result.addVertex(v8);
352         Edge e1, e2, e3, e4, e5, e6, e7, e8;
353         e1 = new Edge(v1, v2);
354         e2 = new Edge(v2, v3);
355         e3 = new Edge(v3, v4);
356         e4 = new Edge(v4, v5);
357         e5 = new Edge(v5, v6);
358         e6 = new Edge(v6, v7);
359         e7 = new Edge(v7, v8);
360         e8 = new Edge(v8, v1);
361         this.result.addEdge(e1);
362         this.result.addEdge(e2);
363         this.result.addEdge(e3);
364         this.result.addEdge(e4);
365         this.result.addEdge(e5);
366         this.result.addEdge(e6);
```

```
367         this.result.addEdge(e7);
368         this.result.addEdge(e8);
369     }
370 
371     private void makePath(Element cur) {
372         String cmd[];
373         String found = cur.getAttributes().get("d");
374         if(found != null){
375             cmd = found.split("\\s+");
376             int cmdLen = cmd.length;
377             List<PathCommandGroup> groups = new ArrayList<PathCommandGroup>();
378             double cumulativeX = 0;
379             double cumulativeY = 0;
380             String curCommand = "";
381             String curProcessed = "";
382             int numOfGroups = 0;
383             int i = 0;
384             while(i<cmdLen){
385                 curProcessed = cmd[i];
386                 if(Character.isAlphabetic(curProcessed.charAt(0))){
387                     groups.add(new PathCommandGroup(cmd[i].toUpperCase()));
388                     curCommand = cmd[i];
389                     numOfGroups++;
390                     i++;
391                 }
392                 else{
393                     double currentProcessed = Double.parseDouble(curProcessed);
394                     if(curCommand.equals("H")){
395                         groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(currentProcessed, groups.get(numOfGroups-2).
                                getLastYCoordinate()));
396                         cumulativeX = currentProcessed;
397                         i++;
398                     }
399                     else if(curCommand.equals("h")){
400                         groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(currentProcessed + cumulativeX, cumulativeY));
401                         cumulativeX += currentProcessed;
402                         i++;
403                     }
404                     else if(curCommand.equals("V")){
405                         groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(groups.get(numOfGroups-2).getLastXCoordinate(),
                                currentProcessed));
406                         cumulativeY = currentProcessed;
407                         i++;
408                     }
409                     else if(curCommand.equals("v")){
410                         groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(cumulativeX, currentProcessed + cumulativeY));
411                         cumulativeY += currentProcessed;
412                         i++;
413                     }
414                     else if(curCommand.equals("A")){
415                         double currentProcessed2 = Double.parseDouble(cmd[i+1]);
416                         double currentProcessed3 = Double.parseDouble(cmd[i+5]);
417                         double currentProcessed4 = Double.parseDouble(cmd[i+6]);
418                         groups.get(numOfGroups-1).setRx(currentProcessed);
419                         groups.get(numOfGroups-1).setRy(currentProcessed2);
420                         groups.get(numOfGroups-1).setDegree(Double.parseDouble(cmd[i+2]));
421                         groups.get(numOfGroups-1).setLarArcFlag(Integer.parseInt(cmd[i+3]));
422                         groups.get(numOfGroups-1).setSweepFlag(Integer.parseInt(cmd[i+4]));
423                         groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(currentProcessed3, currentProcessed4));
424                         cumulativeX = currentProcessed3;
425                         cumulativeY = currentProcessed4;
426                         i+=7;
427                     }
428                     else if(curCommand.equals("a")){
429                         double currentProcessed2 = Double.parseDouble(cmd[i+1]);
430                         double currentProcessed3 = Double.parseDouble(cmd[i+5]);
431                         double currentProcessed4 = Double.parseDouble(cmd[i+6]);
432                         groups.get(numOfGroups-1).setRx(currentProcessed);
433                         groups.get(numOfGroups-1).setRy(currentProcessed2);
434                         groups.get(numOfGroups-1).setDegree(Double.parseDouble(cmd[i+2]));
435                         groups.get(numOfGroups-1).setLarArcFlag(Integer.parseInt(cmd[i+3]));
436                         groups.get(numOfGroups-1).setSweepFlag(Integer.parseInt(cmd[i+4]));
437                         groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(currentProcessed3 + cumulativeX,
                                currentProcessed4 + cumulativeY));
438                         cumulativeX += currentProcessed3;
439                         cumulativeY += currentProcessed4;
440                         i+=7;
441                     }
442                     else{
443                         double currentProcessed2 = Double.parseDouble(cmd[i+1]);
444                         if(Character.isLowerCase(curCommand.charAt(0))){
445                             groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(currentProcessed + cumulativeX,
                                    currentProcessed2 + cumulativeY));
446                             if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
447                                 cumulativeX += currentProcessed;
448                                 cumulativeY += currentProcessed2;
449                             }
450                         }
451                         else{
452                             groups.get(numOfGroups-1).addCoordinate(new Point2D.Double(currentProcessed, currentProcessed2));
453                             if(i >= cmdLen-2 || Character.isAlphabetic(cmd[i+2].charAt(0))){
454                                 cumulativeX = currentProcessed;
455                                 cumulativeY = currentProcessed2;
456                             }
457                         }
458                         i+=2;
459                     }
460                 }
461             }
```

```
462
463             Vertex last = null;
464             Vertex first = null;
465             for(i=0;i<numOfGroups;i++){
466                 PathCommandGroup cmdGroup = groups.get(i);
467                 if(i==0 && !cmdGroup.getCommand().equals("M")){
468                     break;
469                 }
470                 if(cmdGroup.getCommand().equals("M")){
471                     first = makePathMoveTo(cmdGroup);
472                     last = first;
473                 }
474                 else if(cmdGroup.getCommand().equals("L") || cmdGroup.getCommand().equals("H") || cmdGroup.getCommand().equals("V"
                        )){
475                     last = makePathLineTo(last, cmdGroup);
476                 }
477                 else if(cmdGroup.getCommand().equals("C")){
478                     last = makePathCubicCurveTo(last, cmdGroup);
479                 }
480                 else if(cmdGroup.getCommand().equals("S")){
481                     last = makePathSmoothCurveTo(last, cmdGroup, groups.get(i-1));
482                 }
483                 else if(cmdGroup.getCommand().equals("Q")){
484                     last = makePathQuadraticCurveTo(last, cmdGroup);
485                 }
486                 else if(cmdGroup.getCommand().equals("T")){
487                     last = makePathCurveTo(last, cmdGroup, groups.get(i-1));
488                 }
489                 else if(cmdGroup.getCommand().equals("A")){
490                     last = makePathEllipticalArc(last, cmdGroup);
491                 }
492                 else if(cmdGroup.getCommand().equals("Z")){
493                     this.result.addEdge(new Edge(last, first));
494                 }
495             }
496         }
497     }
498
499     private Vertex makePathMoveTo(PathCommandGroup cmdGroup) {
500         Vertex beginPath = new Vertex(cmdGroup.coordinates.get(0));
501         this.result.addVertex(beginPath);
502         if(cmdGroup.coordinates.size()==2){
503             Vertex endBeginPath = new Vertex(cmdGroup.coordinates.get(1));
504             this.result.addVertex(endBeginPath);
505             this.result.addEdge(new Edge(beginPath, endBeginPath));
506             return endBeginPath;
507         }
508         else{
509             return beginPath;
510         }
511     }
512
513     private Vertex makePathLineTo(Vertex last, PathCommandGroup cmdGroup) {
514         Vertex dest = new Vertex(cmdGroup.coordinates.get(0));
515         this.result.addVertex(dest);
516         this.result.addEdge(new Edge(last, dest));
517         return dest;
518     }
519
520     private Vertex makePathCubicCurveTo(Vertex last, PathCommandGroup cmdGroup) {
521         Point2D.Double controlPoints[] = {last.getLocation(), cmdGroup.coordinates.get(0), cmdGroup.coordinates.get(1), cmdGroup.
                coordinates.get(2)};
522         Vertex beginCurve, endCurve, middle, leftMiddle, rightMiddle;
523         beginCurve = last;
524         leftMiddle = new Vertex(getCubicBezierCurvesPoint(controlPoints, 0.25));
525         middle = new Vertex(getCubicBezierCurvesPoint(controlPoints, 0.5));
526         rightMiddle = new Vertex(getCubicBezierCurvesPoint(controlPoints, 0.75));
527         endCurve = new Vertex(cmdGroup.getLastCoordinate());
528         this.result.addVertex(leftMiddle);
529         this.result.addVertex(middle);
530         this.result.addVertex(rightMiddle);
531         this.result.addVertex(endCurve);
532         this.result.addEdge(new Edge(beginCurve, leftMiddle));
533         this.result.addEdge(new Edge(leftMiddle, middle));
534         this.result.addEdge(new Edge(middle, rightMiddle));
535         this.result.addEdge(new Edge(rightMiddle, endCurve));
536         return endCurve;
537     }
538
539     private Vertex makePathSmoothCurveTo(Vertex last, PathCommandGroup cmdGroup, PathCommandGroup prevCmdGroup) {
540         if(prevCmdGroup.getCommand().equals("C") || prevCmdGroup.getCommand().equals("S")){
541             Point2D.Double prevCtrlPoint = prevCmdGroup.getSecondLastCoordinate();
542             Point2D.Double curStart = prevCmdGroup.getLastCoordinate();
543             Point2D.Double curCtrlPoint = mirrorControlPoint(prevCtrlPoint, curStart);
544             cmdGroup.addFirstCoordinate(curCtrlPoint);
545             return makePathCubicCurveTo(last, cmdGroup);
546         }
547         else{
548             return makePathQuadraticCurveTo(last, cmdGroup);
549         }
550     }
551
552     private Vertex makePathQuadraticCurveTo(Vertex last, PathCommandGroup cmdGroup) {
553         Point2D.Double controlPoints[] = {last.getLocation(), cmdGroup.coordinates.get(0), cmdGroup.coordinates.get(1)};
554         Vertex beginCurve, endCurve, middle, leftMiddle, rightMiddle;
555         beginCurve = last;
556         leftMiddle = new Vertex(getQuadraticBezierCurvesPoint(controlPoints, 0.25));
557         middle = new Vertex(getQuadraticBezierCurvesPoint(controlPoints, 0.5));
558         rightMiddle = new Vertex(getQuadraticBezierCurvesPoint(controlPoints, 0.75));
```

```
559         endCurve = new Vertex(cmdGroup.getLastCoordinate());
560         this.result.addVertex(leftMiddle);
561         this.result.addVertex(middle);
562         this.result.addVertex(rightMiddle);
563         this.result.addVertex(endCurve);
564         this.result.addEdge(new Edge(beginCurve, leftMiddle));
565         this.result.addEdge(new Edge(leftMiddle, middle));
566         this.result.addEdge(new Edge(middle, rightMiddle));
567         this.result.addEdge(new Edge(rightMiddle, endCurve));
568         return endCurve;
569     }
570 
571     private Vertex makePathCurveTo(Vertex last, PathCommandGroup cmdGroup, PathCommandGroup prevCmdGroup) {
572         if(prevCmdGroup.getCommand().equals("Q") || prevCmdGroup.getCommand().equals("T")){
573             Point2D.Double prevCtrlPoint = prevCmdGroup.getSecondLastCoordinate();
574             Point2D.Double curStart = prevCmdGroup.getLastCoordinate();
575             Point2D.Double curCtrlPoint = mirrorControlPoint(prevCtrlPoint, curStart);
576             cmdGroup.addFirstCoordinate(curCtrlPoint);
577             return makePathQuadraticCurveTo(last, cmdGroup);
578         }
579         else{
580             return makePathLineTo(last, cmdGroup);
581         }
582     }
583 
584     private void ensureRadii(Vertex last, PathCommandGroup cmdGroup){
585         if(cmdGroup.getCommand().equals("A")){
586             double x1 = last.getLocation().x;
587             double y1 = last.getLocation().y;
588             double x2 = cmdGroup.getLastCoordinate().x;
589             double y2 = cmdGroup.getLastCoordinate().y;
590             double rx = cmdGroup.getRx();
591             double ry = cmdGroup.getRy();
592             double varphi = cmdGroup.getDegree();
593             int fA = cmdGroup.getLarArcFlag();
594             int fS = cmdGroup.getSweepFlag();
595 
596             double matrix[][] = new double[2][2];
597             matrix[0][0] = Math.cos(varphi);
598             matrix[0][1] = Math.sin(varphi);
599             matrix[1][0] = Math.sin(varphi) * -1;
600             matrix[1][1] = Math.cos(varphi);
601             Matrix m1 = new Matrix(matrix);
602             matrix = new double[2][1];
603             matrix[0][0] = (x1-x2)/2.0;
604             matrix[1][0] = (y1-y2)/2.0;
605             Matrix m2 = new Matrix(matrix);
606             Matrix coordinateA = MatrixMath.multiply(m1, m2);
607             double x1A = coordinateA.getMatrix()[0][0];
608             double y1A = coordinateA.getMatrix()[1][0];
609 
610             rx = Math.abs(rx);
611             ry = Math.abs(ry);
612             double radii = Math.pow(x1A, 2)/Math.pow(rx, 2) + Math.pow(y1A, 2)/Math.pow(ry, 2);
613             if(radii>1){
614                 rx = Math.sqrt(radii) * rx;
615                 ry = Math.sqrt(radii) * ry;
616             }
617             cmdGroup.setRx(rx);
618             cmdGroup.setRy(ry);
619         }
620     }
621 
622     private Vertex makePathEllipticalArc(Vertex last, PathCommandGroup cmdGroup) {
623         ensureRadii(last, cmdGroup);
624         Matrix centerPoint = findArcCenterPoint(last, cmdGroup);
625         double rx = cmdGroup.getRx();
626         double ry = cmdGroup.getRy();
627         double varphi = cmdGroup.getDegree();
628         int fS = cmdGroup.getSweepFlag();
629         int fA = cmdGroup.getLarArcFlag();
630         double theta1 = findArcStartAngle(last, cmdGroup);
631         double deltaTheta = findArcDeltaTheta(last, cmdGroup);
632         int parts = (int)Math.abs((Math.toDegrees(deltaTheta)/45));
633         double val = deltaTheta/parts;
634         double r[][] = new double[2][2];
635         r[0][0] = Math.cos(varphi);
636         r[0][1] = -1*Math.sin(varphi);
637         r[1][0] = Math.sin(varphi);
638         r[1][1] = Math.cos(varphi);
639         Matrix mR = new Matrix(r);
640 
641         Vertex u = last;
642         Vertex v = last;
643         double c[][];
644         for(int i=1;i<=parts-1;i++){
645             c = new double[2][1];
646             c[0][0] = rx*Math.cos(theta1+(i*val));
647             c[1][0] = ry*Math.sin(theta1+(i*val));
648             Matrix mC = new Matrix(c);
649             Matrix res = MatrixMath.multiply(mR, mC);
650             res = MatrixMath.add(res, centerPoint);
651             v = new Vertex(new Point2D.Double(res.getMatrix()[0][0], res.getMatrix()[1][0]));
652             this.result.addVertex(v);
653             this.result.addEdge(new Edge(u, v));
654             u = v;
655         }
656         Vertex endArc = new Vertex(cmdGroup.getLastCoordinate());
657         this.result.addVertex(endArc);
```

```
658         this.result.addEdge(new Edge(v, endArc));
659         return endArc;
660     }
661 
662     private double findArcDeltaTheta(Vertex last, PathCommandGroup cmdGroup){
663         double x1 = last.getLocation().x;
664         double y1 = last.getLocation().y;
665         double x2 = cmdGroup.getLastCoordinate().x;
666         double y2 = cmdGroup.getLastCoordinate().y;
667         double rx = cmdGroup.getRx();
668         double ry = cmdGroup.getRy();
669         double varphi = cmdGroup.getDegree();
670         int fA = cmdGroup.getLarArcFlag();
671         int fS = cmdGroup.getSweepFlag();
672 
673         double matrix[][] = new double[2][2];
674         matrix[0][0] = Math.cos(varphi);
675         matrix[0][1] = Math.sin(varphi);
676         matrix[1][0] = Math.sin(varphi) * -1;
677         matrix[1][1] = Math.cos(varphi);
678         Matrix m1 = new Matrix(matrix);
679         matrix = new double[2][1];
680         matrix[0][0] = (x1-x2)/2.0;
681         matrix[1][0] = (y1-y2)/2.0;
682         Matrix m2 = new Matrix(matrix);
683         Matrix coordinateA = MatrixMath.multiply(m1, m2);
684         double x1A = coordinateA.getMatrix()[0][0];
685         double y1A = coordinateA.getMatrix()[1][0];
686 
687         double val = Math.sqrt( ((rx*rx*ry*ry) - (rx*rx*y1A*y1A) - (ry*ry*x1A*x1A)) / ((rx*rx*y1A*y1A) + (ry*ry*x1A*x1A)) );
688         if(fA==fS){
689             val *= -1;
690         }
691         matrix = new double[2][1];
692         matrix[0][0] = rx*y1A/ry;
693         matrix[1][0] = -1*(ry*x1A/rx);
694         Matrix m3 = new Matrix(matrix);
695         Matrix centerPointA = MatrixMath.multiply(m3, val);
696         double cxA = centerPointA.getMatrix()[0][0];
697         double cyA = centerPointA.getMatrix()[1][0];
698 
699 
700 
701 
702         matrix = new double[2][1];
703         matrix[0][0] = (x1A - cxA)/rx;
704         matrix[1][0] = (y1A-cyA)/ry;
705         Matrix vU = new Matrix(matrix);
706         matrix = new double[2][1];
707         matrix[0][0] = ((-1*x1A)-cxA)/rx;
708         matrix[1][0] = ((-1*y1A)-cyA)/ry;
709         Matrix vV = new Matrix(matrix);
710 
711         double dotProduct = MatrixMath.dotProduct(vU, vV);
712         double vULength = MatrixMath.vectorLength(vU);
713         double vVLength = MatrixMath.vectorLength(vV);
714         double result = Math.acos(dotProduct / vULength * vVLength);
715 
716         if((vU.getMatrix()[0][0] * vV.getMatrix()[1][0] - vU.getMatrix()[1][0] * vV.getMatrix()[0][0])<0){
717             result *= -1;
718         }
719 
720         //mod 360
721         if(fS==0 && result>0){
722             result -= 2*Math.PI;
723         }
724         else if(fS==1 && result<0){
725             result += 2*Math.PI;
726         }
727         result %= 2*Math.PI;
728 
729         return result;
730     }
731 
732     private double findArcStartAngle(Vertex last, PathCommandGroup cmdGroup){
733         double x1 = last.getLocation().x;
734         double y1 = last.getLocation().y;
735         double x2 = cmdGroup.getLastCoordinate().x;
736         double y2 = cmdGroup.getLastCoordinate().y;
737         double rx = cmdGroup.getRx();
738         double ry = cmdGroup.getRy();
739         double varphi = cmdGroup.getDegree();
740         int fA = cmdGroup.getLarArcFlag();
741         int fS = cmdGroup.getSweepFlag();
742 
743         double matrix[][] = new double[2][2];
744         matrix[0][0] = Math.cos(varphi);
745         matrix[0][1] = Math.sin(varphi);
746         matrix[1][0] = Math.sin(varphi) * -1;
747         matrix[1][1] = Math.cos(varphi);
748         Matrix m1 = new Matrix(matrix);
749         matrix = new double[2][1];
750         matrix[0][0] = (x1-x2)/2.0;
751         matrix[1][0] = (y1-y2)/2.0;
752         Matrix m2 = new Matrix(matrix);
753         Matrix coordinateA = MatrixMath.multiply(m1, m2);
754         double x1A = coordinateA.getMatrix()[0][0];
755         double y1A = coordinateA.getMatrix()[1][0];
756 
```

```
757         double val = Math.sqrt( ((rx*rx*ry*ry) - (rx*rx*y1A*y1A) - (ry*ry*x1A*x1A)) / ((rx*rx*y1A*y1A) + (ry*ry*x1A*x1A)) );
758         if(fA==fS){
759             val *= -1;
760         }
761         matrix = new double[2][1];
762         matrix[0][0] = rx*y1A/ry;
763         matrix[1][0] = -1*(ry*x1A/rx);
764         Matrix m3 = new Matrix(matrix);
765         Matrix centerPointA = MatrixMath.multiply(m3, val);
766         double cxA = centerPointA.getMatrix()[0][0];
767         double cyA = centerPointA.getMatrix()[1][0];
768 
769         matrix = new double[2][1];
770         matrix[0][0] = 1;
771         matrix[1][0] = 0;
772         Matrix vU = new Matrix(matrix);
773         matrix = new double[2][1];
774         matrix[0][0] = (x1A-cxA)/rx;
775         matrix[1][0] = (y1A-cyA)/ry;
776         Matrix vV = new Matrix(matrix);
777 
778         double dotProduct = MatrixMath.dotProduct(vU, vV);
779         double vULength = MatrixMath.vectorLength(vU);
780         double vVLength = MatrixMath.vectorLength(vV);
781         double result = Math.acos(dotProduct / vULength * vVLength);
782 
783         if((vU.getMatrix()[0][0] * vV.getMatrix()[1][0] - vU.getMatrix()[1][0] * vV.getMatrix()[0][0])<0){
784             result *= -1;
785         }
786         return result;
787     }
788 
789     private Matrix findArcCenterPoint(Vertex last, PathCommandGroup cmdGroup){
790         double x1 = last.getLocation().x;
791         double y1 = last.getLocation().y;
792         double x2 = cmdGroup.getLastCoordinate().x;
793         double y2 = cmdGroup.getLastCoordinate().y;
794         double rx = cmdGroup.getRx();
795         double ry = cmdGroup.getRy();
796         double varphi = cmdGroup.getDegree();
797         int fA = cmdGroup.getLarArcFlag();
798         int fS = cmdGroup.getSweepFlag();
799 
800         //find x1' and y1'
801         double matrix[][] = new double[2][2];
802         matrix[0][0] = Math.cos(varphi);
803         matrix[0][1] = Math.sin(varphi);
804         matrix[1][0] = Math.sin(varphi) * -1;
805         matrix[1][1] = Math.cos(varphi);
806         Matrix m1 = new Matrix(matrix);
807         matrix = new double[2][1];
808         matrix[0][0] = (x1-x2)/2.0;
809         matrix[1][0] = (y1-y2)/2.0;
810         Matrix m2 = new Matrix(matrix);
811         Matrix coordinateA = MatrixMath.multiply(m1, m2);
812         double x1A = coordinateA.getMatrix()[0][0];
813         double y1A = coordinateA.getMatrix()[1][0];
814 
815         //find cx' and cy'
816         double val = Math.sqrt( ((rx*rx*ry*ry) - (rx*rx*y1A*y1A) - (ry*ry*x1A*x1A)) / ((rx*rx*y1A*y1A) + (ry*ry*x1A*x1A)) );
817         if(fA==fS){
818             val *= -1;
819         }
820         matrix = new double[2][1];
821         matrix[0][0] = rx*y1A/ry;
822         matrix[1][0] = -1*(ry*x1A/rx);
823         Matrix m3 = new Matrix(matrix);
824         Matrix centerPointA = MatrixMath.multiply(m3, val);
825 
826         //find center points
827         matrix = new double[2][2];
828         matrix[0][0] = Math.cos(varphi);
829         matrix[0][1] = -1 * Math.sin(varphi);
830         matrix[1][0] = Math.sin(varphi);
831         matrix[1][1] = Math.cos(varphi);
832         Matrix m4 = new Matrix(matrix);
833         matrix = new double[2][1];
834         matrix[0][0] = (x1+x2)/2.0;
835         matrix[1][0] = (y1+y2)/2.0;
836         Matrix m5 = new Matrix(matrix);
837         Matrix centerPoint = MatrixMath.multiply(m4, centerPointA);
838         centerPoint = MatrixMath.add(centerPoint, m5);
839         return centerPoint;
840     }
841 
842     private Point2D.Double getQuadraticBezierCurvesPoint(Point2D.Double controlPoints[], double t){
843         double resX = 0;
844         double resY = 0;
845         resX = Math.pow((1-t), 2) * controlPoints[0].x + 2 * t * (1-t) * controlPoints[1].x + Math.pow(t, 2) * controlPoints[2].x;
846         resY = Math.pow((1-t), 2) * controlPoints[0].y + 2 * t * (1-t) * controlPoints[1].y + Math.pow(t, 2) * controlPoints[2].y;
847         return new Point2D.Double(resX, resY);
848     }
849 
850     private Point2D.Double getCubicBezierCurvesPoint(Point2D.Double controlPoints[], double t){
851         double resX = 0;
852         double resY = 0;
853         resX = Math.pow((1-t), 3) * controlPoints[0].x + 3 * t * Math.pow((1-t), 2) * controlPoints[1].x + 3 * Math.pow(t, 2) * (1
                -t) * controlPoints[2].x + Math.pow(t, 3) * controlPoints[3].x;
854         resY = Math.pow((1-t), 3) * controlPoints[0].y + 3 * t * Math.pow((1-t), 2) * controlPoints[1].y + 3 * Math.pow(t, 2) * (1
```

```
            -t) * controlPoints[2].y + Math.pow(t, 3) * controlPoints[3].y;
        return new Point2D.Double(resX, resY);
    }

    private Point2D.Double mirrorControlPoint(Point2D.Double prevCtrlPoint, Point2D.Double curStart) {
        Point2D.Double res = new Point2D.Double();
        res.x = curStart.x - (prevCtrlPoint.x - curStart.x);
        res.y = curStart.y - (prevCtrlPoint.y - curStart.y);
        return res;
    }

    class PathCommandGroup{
        private String command;
        private List<Point2D.Double> coordinates;
        private double degree;
        private int larArcFlag;
        private int sweepFlag;
        private double rx;
        private double ry;


        public PathCommandGroup(String command){
            this.command = command;
            this.coordinates = new ArrayList<Point2D.Double>();
        }

        public double getRx() {
            return rx;
        }

        public void setRx(double rx) {
            this.rx = rx;
        }

        public double getRy() {
            return ry;
        }

        public void setRy(double ry) {
            this.ry = ry;
        }

        public void addCoordinate(Point2D.Double newPoint){
            this.coordinates.add(newPoint);
        }

        public void addFirstCoordinate(Point2D.Double newPoint){
            this.coordinates.add(0, newPoint);
        }

        public Point2D.Double getSecondLastCoordinate(){
            return this.coordinates.get(this.coordinates.size()-2);
        }

        public Point2D.Double getLastCoordinate(){
            return this.coordinates.get(this.coordinates.size()-1);
        }

        public double getLastXCoordinate(){
            return this.coordinates.get(this.coordinates.size()-1).x;
        }

        public double getLastYCoordinate(){
            return this.coordinates.get(this.coordinates.size()-1).y;
        }

        public String getCommand() {
            return command;
        }

        public List<Point2D.Double> getCoordinates() {
            return coordinates;
        }

        public void setCoordinates(List<Point2D.Double> coordinates) {
            this.coordinates = coordinates;
        }

        public double getDegree() {
            return degree;
        }

        public void setDegree(double degree) {
            this.degree = Math.toRadians(degree);
        }

        public int getLarArcFlag() {
            return larArcFlag;
        }

        public void setLarArcFlag(int larArcFlag) {
            this.larArcFlag = larArcFlag;
        }

        public int getSweepFlag() {
            return sweepFlag;
        }

        public void setSweepFlag(int sweepFlag) {
```

```
            this.sweepFlag = sweepFlag;
        }
    }
}
```

Listing A.17: GraphDrawer.java

```
package engine;

import java.awt.geom.Point2D;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

/**
 * @author Albert - 2014730007
 */
public class GraphDrawer {
    private Graph graph;
    private File destFile;
    private BufferedWriter bw;
    private List<Element> unprocessedElements;
    private double svgWidth;
    private double svgHeight;

    public GraphDrawer(Graph graph, List<Element> unprocessedElements, File destFile, double svgWidth, double svgHeight){
        this.unprocessedElements = unprocessedElements;
        this.graph = graph;
        this.destFile = destFile;
        this.svgWidth = svgWidth;
        this.svgHeight = svgHeight;
        try{
            this.bw = new BufferedWriter(new FileWriter(this.destFile.getPath()));
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }

    public void draw() throws IOException{
        if(this.graph != null){
            this.graph.hierholzer();
            bw.write("<svg height=\""+this.svgHeight+"\" width=\""+this.svgWidth+"\" style=\"padding:5;\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\">");
            bw.newLine();
            int numVertices = graph.getVertices().size();
            Set<String> hs = new HashSet<String>();
            for(Vertex cur: graph.getVertices()){
                if(cur.getDisplayNumbers().size()!=0){
                    String numbers = "";
                    for(int i=0;i<cur.getDisplayNumbers().size();i++){
                        if(i!=0)numbers+="/";
                        numbers += cur.getDisplayNumbers().get(i);
                    }
                    hs.add("<text x=\"" + cur.getLocation().x + "\" y=\"" + (cur.getLocation().y - 15) + "\">"+numbers+"</text>");
                }
                if(cur.getDegree()>0){
                    hs.add("<circle cx=\"" + cur.getLocation().x + "\" cy=\"" + cur.getLocation().y + "\" r=\"5\" fill=\"black\" />");
                }
            }
            int numEdges = graph.getEdges().size();
            for(int i=0;i<numEdges;i++){
                Edge cur = graph.getEdges().get(i);
                if(cur.isHelpLine()){
                    hs.add("<line x1=\"" + cur.getFrom().getLocation().x + "\" y1=\"" + cur.getFrom().getLocation().y + "\" "
                            + "x2=\"" + cur.getTo().getLocation().x + "\" y2=\"" + cur.getTo().getLocation().y + "\" style=\"stroke:rgb(0,0,0);stroke-width:2\" />");
                }
            }
            int numUnprocessed = unprocessedElements.size();
            for(int i=0;i<numUnprocessed;i++){
                Element cur = unprocessedElements.get(i);
                String str = "";
                str += ("<" + cur.getName() + " ");
                Map<String, String> attributes = cur.getAttributes();
                for(Map.Entry<String, String> entry: attributes.entrySet()){
                    str += (entry.getKey()+ "=" + "\"" + entry.getValue() +"\" ");
                }
                str += " />";
                hs.add(str);
            }
            for(String s: hs){
                bw.write(s);
                bw.newLine();
            }
            bw.write("</svg>");
            bw.close();
        }
    }
}
```

Listing A.18: ApplicationGUI.java

```
1  package gui;
2
3  import engine.GraphDrawer;
4  import engine.GraphMaker;
5  import engine.SVGParser;
6  import java.awt.Color;
7  import java.awt.Desktop;
8  import java.io.File;
9  import java.net.URI;
10 import javafx.application.Platform;
11 import javafx.embed.swing.JFXPanel;
12 import javafx.scene.Scene;
13 import javafx.scene.web.WebView;
14 import javax.swing.GroupLayout.Group;
15 import javax.swing.JFileChooser;
16 import javax.swing.JOptionPane;
17 import javax.swing.JPanel;
18 import javax.swing.filechooser.FileNameExtensionFilter;
19
20 /**
21  * @author Albert - 2014730007
22  */
23 public class ApplicationGUI extends javax.swing.JFrame {
24     private JFileChooser fileChooser;
25     private SVGParser svgParser;
26     private GraphMaker graphMaker;
27     private GraphDrawer graphDrawer;
28     WebView webViewInput;
29     WebView webViewOutput;
30
31     /**
32      * Creates new form ApplicationGUI
33      */
34     public ApplicationGUI() {
35         initComponents();
36     }
37
38     /**
39      * This method is called from within the constructor to initialize the form.
40      * WARNING: Do NOT modify this code. The content of this method is always
41      * regenerated by the Form Editor.
42      */
43     @SuppressWarnings("unchecked")
44     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
45     private void initComponents() {
46
47         jScrollPane4 = new javax.swing.JScrollPane();
48         jEditorPane3 = new javax.swing.JEditorPane();
49         navPanel = new javax.swing.JPanel();
50         homeNav = new javax.swing.JPanel();
51         jLabel4 = new javax.swing.JLabel();
52         jLabel5 = new javax.swing.JLabel();
53         generateProblemNav = new javax.swing.JPanel();
54         jLabel8 = new javax.swing.JLabel();
55         jLabel9 = new javax.swing.JLabel();
56         contentPanel = new javax.swing.JPanel();
57         homeContent = new javax.swing.JPanel();
58         titlePanel = new javax.swing.JPanel();
59         jLabel1 = new javax.swing.JLabel();
60         labelPetunjuk = new javax.swing.JLabel();
61         jScrollPane2 = new javax.swing.JScrollPane();
62         textAreaPetunjuk = new javax.swing.JTextArea();
63         generateProblemContent = new javax.swing.JPanel();
64         browseButton = new javax.swing.JPanel();
65         jLabel10 = new javax.swing.JLabel();
66         jLabel11 = new javax.swing.JLabel();
67         inputLabel = new javax.swing.JLabel();
68         outputLabel = new javax.swing.JLabel();
69         jScrollPane1 = new javax.swing.JScrollPane();
70         inputPane = new javax.swing.JEditorPane();
71         jScrollPane5 = new javax.swing.JScrollPane();
72         outputPane = new javax.swing.JEditorPane();
73
74         jScrollPane4.setViewportView(jEditorPane3);
75
76         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
77         setLocationByPlatform(true);
78         setResizable(false);
79
80         navPanel.setBackground(new java.awt.Color(54, 33, 89));
81         navPanel.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
82
83         homeNav.setBackground(new java.awt.Color(85, 65, 118));
84         homeNav.addMouseListener(new java.awt.event.MouseAdapter() {
85             public void mousePressed(java.awt.event.MouseEvent evt) {
86                 homeNavMousePressed(evt);
87             }
88         });
89
90         jLabel4.setForeground(new java.awt.Color(204, 204, 204));
91         jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
92         jLabel4.setIcon(new javax.swing.ImageIcon(getClass().getResource("/image/icons8_Home_15px.png"))); // NOI18N
93
94         jLabel5.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
95         jLabel5.setForeground(new java.awt.Color(204, 204, 204));
96         jLabel5.setText("Home");
97
```

```
98          javax.swing.GroupLayout homeNavLayout = new javax.swing.GroupLayout(homeNav);
99          homeNav.setLayout(homeNavLayout);
100         homeNavLayout.setHorizontalGroup(
101             homeNavLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
102             .addGroup(homeNavLayout.createSequentialGroup()
103                 .addGap(22, 22, 22)
104                 .addComponent(jLabel4)
105                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
106                 .addComponent(jLabel5)
107                 .addContainerGap(116, Short.MAX_VALUE))
108         );
109         homeNavLayout.setVerticalGroup(
110             homeNavLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
111             .addGroup(homeNavLayout.createSequentialGroup()
112                 .addGap(16, 16, 16)
113                 .addGroup(homeNavLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)
114                     .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
115                     .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                            MAX_VALUE))
116                 .addContainerGap(19, Short.MAX_VALUE))
117         );
118
119         navPanel.add(homeNav, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 80, -1, 50));
120
121         generateProblemNav.setBackground(new java.awt.Color(64, 43, 100));
122         generateProblemNav.addMouseListener(new java.awt.event.MouseAdapter() {
123             public void mousePressed(java.awt.event.MouseEvent evt) {
124                 generateProblemNavMousePressed(evt);
125             }
126         });
127
128         jLabel8.setForeground(new java.awt.Color(204, 204, 204));
129         jLabel8.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
130         jLabel8.setIcon(new javax.swing.ImageIcon(getClass().getResource("/image/icons8_Refresh_15px.png"))); // NOI18N
131
132         jLabel9.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
133         jLabel9.setForeground(new java.awt.Color(204, 204, 204));
134         jLabel9.setText("Generate Problem");
135
136         javax.swing.GroupLayout generateProblemNavLayout = new javax.swing.GroupLayout(generateProblemNav);
137         generateProblemNav.setLayout(generateProblemNavLayout);
138         generateProblemNavLayout.setHorizontalGroup(
139             generateProblemNavLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
140             .addGroup(generateProblemNavLayout.createSequentialGroup()
141                 .addGap(22, 22, 22)
142                 .addComponent(jLabel8)
143                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
144                 .addComponent(jLabel9)
145                 .addContainerGap(34, Short.MAX_VALUE))
146         );
147         generateProblemNavLayout.setVerticalGroup(
148             generateProblemNavLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
149             .addGroup(generateProblemNavLayout.createSequentialGroup()
150                 .addGap(16, 16, 16)
151                 .addGroup(generateProblemNavLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)
152                     .addComponent(jLabel9, javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
153                     .addComponent(jLabel8, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                            MAX_VALUE))
154                 .addContainerGap(19, Short.MAX_VALUE))
155         );
156
157         navPanel.add(generateProblemNav, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 130, 200, 50));
158
159         contentPanel.setLayout(new java.awt.CardLayout());
160
161         homeContent.setBackground(new java.awt.Color(255, 255, 255));
162
163         titlePanel.setBackground(new java.awt.Color(110, 89, 222));
164
165         jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 24)); // NOI18N
166         jLabel1.setForeground(new java.awt.Color(204, 204, 204));
167         jLabel1.setText("Connecting Dot Problem Generator");
168
169         javax.swing.GroupLayout titlePanelLayout = new javax.swing.GroupLayout(titlePanel);
170         titlePanel.setLayout(titlePanelLayout);
171         titlePanelLayout.setHorizontalGroup(
172             titlePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
173             .addGroup(titlePanelLayout.createSequentialGroup()
174                 .addContainerGap()
175                 .addComponent(jLabel1)
176                 .addContainerGap(400, Short.MAX_VALUE))
177         );
178         titlePanelLayout.setVerticalGroup(
179             titlePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
180             .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, titlePanelLayout.createSequentialGroup()
181                 .addContainerGap(62, Short.MAX_VALUE)
182                 .addComponent(jLabel1)
183                 .addContainerGap())
184         );
185
186         labelPetunjuk.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
187         labelPetunjuk.setText("Petunjuk:");
188
189         textAreaPetunjuk.setEditable(false);
190         textAreaPetunjuk.setColumns(20);
191         textAreaPetunjuk.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N
192         textAreaPetunjuk.setLineWrap(true);
193         textAreaPetunjuk.setRows(5);
194         textAreaPetunjuk.setText("Program ini digunakan untuk membuat soal permainan menghubungkan titik. Masukan program berupa
```

```
            file SVG dan keluaran program berupa file HTML. Pembuatan soal dapat dilakukan pada panel Generate Problem. Klik
            Generate Problem pada navigasi untuk berpindah ke panel Generate Problem.\n\nKlik tombol Browse pada panel Generate
            Problem untuk menentukan file masukan. Setelah itu, program akan meminta lokasi penyimpanan dan nama file yang akan
            disimpan. Pratinjau file masukan dan keluaran akan ditampilkan pada panel Generate Problem.");
195         textAreaPetunjuk.setToolTipText(null);
196         textAreaPetunjuk.setWrapStyleWord(true);
197         jScrollPane2.setViewportView(textAreaPetunjuk);
198
199         javax.swing.GroupLayout homeContentLayout = new javax.swing.GroupLayout(homeContent);
200         homeContent.setLayout(homeContentLayout);
201         homeContentLayout.setHorizontalGroup(
202             homeContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
203             .addComponent(titlePanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
204             .addGroup(homeContentLayout.createSequentialGroup()
205                 .addGap(39, 39, 39)
206                 .addGroup(homeContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
207                     .addComponent(labelPetunjuk)
208                     .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 735, javax.swing.GroupLayout.
                            PREFERRED_SIZE))
209                 .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
210         );
211         homeContentLayout.setVerticalGroup(
212             homeContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
213             .addGroup(homeContentLayout.createSequentialGroup()
214                 .addGap(54, 54, 54)
215                 .addComponent(titlePanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                        swing.GroupLayout.PREFERRED_SIZE)
216                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 45, Short.MAX_VALUE)
217                 .addComponent(labelPetunjuk)
218                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
219                 .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 250, javax.swing.GroupLayout.PREFERRED_SIZE)
220                 .addGap(123, 123, 123))
221         );
222
223         contentPanel.add(homeContent, "card2");
224
225         generateProblemContent.setBackground(new java.awt.Color(255, 255, 255));
226
227         browseButton.setBackground(new java.awt.Color(54, 33, 89));
228         browseButton.addMouseListener(new java.awt.event.MouseAdapter() {
229             public void mousePressed(java.awt.event.MouseEvent evt) {
230                 browseButtonMousePressed(evt);
231             }
232         });
233
234         jLabel10.setIcon(new javax.swing.ImageIcon(getClass().getResource("/image/icons8_Open_20px.png"))); // NOI18N
235
236         jLabel11.setFont(new java.awt.Font("Segoe UI", 1, 24)); // NOI18N
237         jLabel11.setForeground(new java.awt.Color(204, 204, 204));
238         jLabel11.setText("Browse");
239
240         javax.swing.GroupLayout browseButtonLayout = new javax.swing.GroupLayout(browseButton);
241         browseButton.setLayout(browseButtonLayout);
242         browseButtonLayout.setHorizontalGroup(
243             browseButtonLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
244             .addGroup(browseButtonLayout.createSequentialGroup()
245                 .addContainerGap(32, Short.MAX_VALUE)
246                 .addComponent(jLabel10, javax.swing.GroupLayout.PREFERRED_SIZE, 22, javax.swing.GroupLayout.PREFERRED_SIZE)
247                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
248                 .addComponent(jLabel11)
249                 .addGap(46, 46, 46))
250         );
251         browseButtonLayout.setVerticalGroup(
252             browseButtonLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
253             .addComponent(jLabel10, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
254             .addComponent(jLabel11, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE, 54, Short.
                    MAX_VALUE)
255         );
256
257         inputLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
258         inputLabel.setText("Input:");
259
260         outputLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
261         outputLabel.setText("Output:");
262
263         inputPane.setEditable(false);
264         jScrollPane1.setViewportView(inputPane);
265
266         outputPane.setEditable(false);
267         jScrollPane5.setViewportView(outputPane);
268
269         javax.swing.GroupLayout generateProblemContentLayout = new javax.swing.GroupLayout(generateProblemContent);
270         generateProblemContent.setLayout(generateProblemContentLayout);
271         generateProblemContentLayout.setHorizontalGroup(
272             generateProblemContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
273             .addGroup(generateProblemContentLayout.createSequentialGroup()
274                 .addGap(18, 18, 18)
275                 .addGroup(generateProblemContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
276                     .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 381, javax.swing.GroupLayout.
                            PREFERRED_SIZE)
277                     .addComponent(inputLabel))
278                 .addGap(18, 18, 18)
279                 .addGroup(generateProblemContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
280                     .addGroup(generateProblemContentLayout.createSequentialGroup()
281                         .addComponent(outputLabel)
282                         .addGap(0, 0, Short.MAX_VALUE))
283                     .addComponent(jScrollPane5, javax.swing.GroupLayout.DEFAULT_SIZE, 381, Short.MAX_VALUE))
284                 .addGap(18, 18, 18))
285             .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, generateProblemContentLayout.createSequentialGroup()
```

```
286                 .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
287                 .addComponent(browseButton, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                        swing.GroupLayout.PREFERRED_SIZE)
288                 .addGap(305, 305, 305))
289         );
290         generateProblemContentLayout.setVerticalGroup(
291             generateProblemContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
292             .addGroup(generateProblemContentLayout.createSequentialGroup()
293                 .addGap(66, 66, 66)
294                 .addGroup(generateProblemContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
295                     .addComponent(inputLabel, javax.swing.GroupLayout.Alignment.TRAILING)
296                     .addComponent(outputLabel, javax.swing.GroupLayout.Alignment.TRAILING))
297                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
298                 .addGroup(generateProblemContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
299                     .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 374, javax.swing.GroupLayout.
                            PREFERRED_SIZE)
300                     .addComponent(jScrollPane5, javax.swing.GroupLayout.PREFERRED_SIZE, 374, javax.swing.GroupLayout.
                            PREFERRED_SIZE))
301                 .addGap(38, 38, 38)
302                 .addComponent(browseButton, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                        swing.GroupLayout.PREFERRED_SIZE)
303                 .addContainerGap(45, Short.MAX_VALUE))
304         );
305 
306         contentPanel.add(generateProblemContent, "card2");
307 
308         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
309         getContentPane().setLayout(layout);
310         layout.setHorizontalGroup(
311             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
312             .addGroup(layout.createSequentialGroup()
313                 .addComponent(navPanel, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.
                        GroupLayout.PREFERRED_SIZE)
314                 .addGap(0, 0, 0)
315                 .addComponent(contentPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                        MAX_VALUE))
316         );
317         layout.setVerticalGroup(
318             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
319             .addComponent(navPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
320             .addComponent(contentPanel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
321         );
322 
323         pack();
324     }// </editor-fold>//GEN-END:initComponents
325 
326     private void homeNavMousePressed(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_homeNavMousePressed
327         setColor(homeNav);
328         resetColor(generateProblemNav);
329 
330         contentPanel.removeAll();
331         contentPanel.repaint();
332         contentPanel.revalidate();
333 
334         contentPanel.add(homeContent);
335         contentPanel.repaint();
336         contentPanel.revalidate();
337     }//GEN-LAST:event_homeNavMousePressed
338 
339     private void generateProblemNavMousePressed(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_generateProblemNavMousePressed
340         setColor(generateProblemNav);
341         resetColor(homeNav);
342 
343         contentPanel.removeAll();
344         contentPanel.repaint();
345         contentPanel.revalidate();
346 
347         contentPanel.add(generateProblemContent);
348         contentPanel.repaint();
349         contentPanel.revalidate();
350     }//GEN-LAST:event_generateProblemNavMousePressed
351 
352     private void browseButtonMousePressed(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_browseButtonMousePressed
353         fileChooser = new JFileChooser(new File("C:/"));
354         fileChooser.setFileFilter(new FileNameExtensionFilter("SVG_files", "svg"));
355         fileChooser.setDialogTitle("Choose_SVG_File");
356         int feedback = fileChooser.showOpenDialog(this);
357         final String inputURI = fileChooser.getSelectedFile().getAbsolutePath();
358         if(feedback == JFileChooser.APPROVE_OPTION){
359             svgParser = new SVGParser(fileChooser.getSelectedFile());
360             svgParser.parseFile();
361             graphMaker = new GraphMaker(svgParser.getElements());
362             fileChooser = new JFileChooser(new File("C:/"));
363             fileChooser.setFileFilter(new FileNameExtensionFilter("SVG_files", "svg"));
364             fileChooser.setDialogTitle("Save_Converted_File");
365             feedback = fileChooser.showSaveDialog(this);
366             final String outputURI = fileChooser.getSelectedFile().getAbsolutePath();
367             if(feedback == JFileChooser.APPROVE_OPTION){
368                 try{
369                     graphDrawer = new GraphDrawer(graphMaker.getResult(), svgParser.getUnprocessedElements(), fileChooser.
                            getSelectedFile(), svgParser.getSvgWidth(), svgParser.getSvgHeight());
370                     graphDrawer.draw();
371 
372                     JFXPanel jfxInput = new JFXPanel();
373                     JFXPanel jfxOutput = new JFXPanel();
374                     inputPane.add(jfxInput);
375                     outputPane.add(jfxOutput);
376                     Platform.runLater(new Runnable() {
```

```
377                         @Override
378                         public void run(){
379                             if(webViewInput==null){
380                                 webViewInput = new WebView();
381                                 jfxInput.setScene(new Scene(webViewInput));
382                             }
383                             webViewInput.getEngine().load("file:///"+inputURI);
384                             jfxInput.setVisible(true);
385                             jfxInput.setBounds(0, 0, inputPane.getWidth(), inputPane.getHeight());
386 
387                             if(webViewOutput==null){
388                                 webViewOutput = new WebView();
389                                 jfxOutput.setScene(new Scene(webViewOutput));
390                             }
391                             webViewOutput.getEngine().load("file:///"+outputURI);
392                             jfxOutput.setVisible(true);
393                             jfxOutput.setBounds(0, 0, outputPane.getWidth(), outputPane.getHeight());
394                         }
395                     });
396 
397                     JOptionPane.showMessageDialog(this, "Berhasil Membuat Soal");
398                 }
399                 catch(Exception e){
400                     e.printStackTrace();
401                 }
402             }
403         }
404     }//GEN-LAST:event_browseButtonMousePressed
405 
406     void setColor(JPanel panel){
407         panel.setBackground(new Color(85, 65, 118));
408     }
409 
410     void resetColor(JPanel panel){
411         panel.setBackground(new Color(64, 43, 100));
412     }
413 
414     /**
415      * @param args the command line arguments
416      */
417     public static void main(String args[]) {
418         /* Set the Nimbus look and feel */
419         //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
420         /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
421          * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
422          */
423         try {
424             for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
425                 if ("Nimbus".equals(info.getName())) {
426                     javax.swing.UIManager.setLookAndFeel(info.getClassName());
427                     break;
428                 }
429             }
430         } catch (ClassNotFoundException ex) {
431             java.util.logging.Logger.getLogger(ApplicationGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
432         } catch (InstantiationException ex) {
433             java.util.logging.Logger.getLogger(ApplicationGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
434         } catch (IllegalAccessException ex) {
435             java.util.logging.Logger.getLogger(ApplicationGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
436         } catch (javax.swing.UnsupportedLookAndFeelException ex) {
437             java.util.logging.Logger.getLogger(ApplicationGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
438         }
439         //</editor-fold>
440 
441         /* Create and display the form */
442         java.awt.EventQueue.invokeLater(new Runnable() {
443             public void run() {
444                 ApplicationGUI gui = new ApplicationGUI();
445                 gui.setVisible(true);
446             }
447         });
448     }
449 
450     // Variables declaration - do not modify//GEN-BEGIN:variables
451     private javax.swing.JPanel browseButton;
452     private javax.swing.JPanel contentPanel;
453     private javax.swing.JPanel generateProblemContent;
454     private javax.swing.JPanel generateProblemNav;
455     private javax.swing.JPanel homeContent;
456     private javax.swing.JPanel homeNav;
457     private javax.swing.JLabel inputLabel;
458     private javax.swing.JEditorPane inputPane;
459     private javax.swing.JEditorPane jEditorPane3;
460     private javax.swing.JLabel jLabel1;
461     private javax.swing.JLabel jLabel10;
462     private javax.swing.JLabel jLabel11;
463     private javax.swing.JLabel jLabel4;
464     private javax.swing.JLabel jLabel5;
465     private javax.swing.JLabel jLabel8;
466     private javax.swing.JLabel jLabel9;
467     private javax.swing.JScrollPane jScrollPane1;
468     private javax.swing.JScrollPane jScrollPane2;
469     private javax.swing.JScrollPane jScrollPane4;
470     private javax.swing.JScrollPane jScrollPane5;
471     private javax.swing.JLabel labelPetunjuk;
472     private javax.swing.JPanel navPanel;
473     private javax.swing.JLabel outputLabel;
474     private javax.swing.JEditorPane outputPane;
475     private javax.swing.JTextArea textAreaPetunjuk;
```

```
476     private javax.swing.JPanel titlePanel;
477     // End of variables declaration//GEN-END:variables
478 }
```