**PENGEMBANGAN KAKAS VISUALISASI**

**DARI GRAF KODE PROGRAM UNTUK**

**MEMAHAMI EKSEKUSI KODE PROGRAM**

**SEMINAR TESIS**

**Karya tulis sebagai salah satu syarat**

**untuk memperoleh gelar Magister dari**

**Institut Teknologi Bandung**

**Oleh**

**HABIBIE ED DIEN**

**NIM: 23515043**

**(Program Studi Magister Informatika)**

****

**INSTITUT TEKNOLOGI BANDUNG**

**Mei 2018**

# ABSTRAK

**PENGEMBANGAN KAKAS VISUALISASI**

**DARI GRAF KODE PROGRAM UNTUK**

**MEMAHAMI EKSEKUSI KODE PROGRAM**

Oleh

**HABIBIE ED DIEN**

**NIM: 23515043**

**(Program Studi Magister Informatika)**

Tujuan penelitian ini adalah menghasilkan kakas visualisasi sebagai pendekatan untuk memahami eksekusi graf kode program. Penelitian ini adalah pengembangan kakas dari *pythontutor.com* (OPT). Metode pengukuran kinerja yang digunakan adalah evaluasi visualisasi. Tekniknya menggunakan kuesioner *online* dengan empat tahap, yaitu tahap pertama mengisi biodata responden, tahap kedua menyelesaikan pretes, tahap ketiga simulasi menggunakan kakas OPT dan kakas hasil pengembangan. Tahap terakhir responden menyelesaikan post-tes. Subyek penelitian ini adalah mahasiswa informatika sarjana (S1) dan pascasarjana (S2) di Institut Teknologi Bandung. Penelitian ini menghasilkan: 1) Pendekatan visualisasi dapat menjadi sarana yang efektif dan efisien untuk memahami eksekusi graf kode program; 2) Usability adalah aspek penting dalam pengembangan kakas visualisasi; 3) ...

Kata kunci: kakas visualisasi, memahami, eksekusi graf kode program

**PENGEMBANGAN KAKAS VISUALISASI**

**DARI GRAF KODE PROGRAM UNTUK**

**MEMAHAMI EKSEKUSI KODE PROGRAM**

# HALAMAN PENGESAHAN

Oleh

**Habibie Ed Dien**

**NIM: 23515043**

**(Program Studi Magister Informatika)**

Institut Teknologi Bandung

Menyetujui

Seminar Tesis

Tanggal 21 Mei 2018

|  |
| --- |
| Pembimbing  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Yudistira Dwi W. Asnar, S.T., Ph.D.) |

# PEDOMAN PENGGUNAAN TESIS

Tesis S2 yang tidak dipublikasikan terdaftar dan tersedia di Perpustakaan Institut Teknologi Bandung, dan terbuka untuk umum dengan ketentuan bahwa hak cipta ada pada pengarang dengan mengikuti aturan HaKI yang berlaku di Institut Teknologi Bandung. Referensi kepustakaan diperkenankan dicatat, tetapi pengutipan atau peringkasan hanya dapat dilakukan seizin pengarang dan harus disertai dengan kaidah ilmiah untuk menyebutkan sumbernya.

Sitasi hasil penelitian Tesis ini dapat ditulis dalam bahasa Indonesia sebagai berikut:

Habibie Ed Dien. (2018): *Pengembangan kakas visualisasi dari graf kode program untuk memahami eksekusi kode program*, Tesis Program Magister, Institut Teknologi Bandung.

dan dalam bahasa Inggris sebagai berikut:

Habibie Ed Dien. (2018): *Development of program visualization from code graph to understand code execution*, Master’s Program Thesis, Institut Teknologi Bandung.

Memperbanyak atau menerbitkan sebagian atau seluruh tesis haruslah seizin Dekan Sekolah Pascasarjana, Institut Teknologi Bandung.

# HALAMAN PERUNTUKAN

# KATA PENGANTAR

# DAFTAR ISI

HALAMAN PENGESAHAN i

[ABSTRAK i](#_Toc514380841)

[HALAMAN PENGESAHAN ii](#_Toc514380842)

[PEDOMAN PENGGUNAAN TESIS iii](#_Toc514380843)

[HALAMAN PERUNTUKAN iv](#_Toc514380844)

[KATA PENGANTAR v](#_Toc514380845)

[DAFTAR ISI vi](#_Toc514380846)

[DAFTAR GAMBAR DAN ILUSTRASI viii](#_Toc514380847)

[DAFTAR TABEL x](#_Toc514380848)

[Bab I Pendahuluan 1](#_Toc514380849)

[I.1 Latar Belakang 1](#_Toc514380850)

[I.2 Rumusan Masalah 3](#_Toc514380851)

[I.3 Tujuan Penelitian 3](#_Toc514380852)

[I.4 Batasan Masalah 3](#_Toc514380853)

[I.5 Metodologi Penelitian 3](#_Toc514380854)

[I.6 Sistematika Penulisan 5](#_Toc514380855)

[Bab II Tinjauan Pustaka dan Eksplorasi 6](#_Toc514380856)

[II.1 Terminologi Visualisasi Perangkat Lunak 6](#_Toc514380857)

[II.2 Metodologi dan Prinsip Visualisasi Data 7](#_Toc514380858)

[II.3 Teori Desain Interaksi 9](#_Toc514380859)

[II.4 Eksplorasi Kakas Visualisasi Program untuk Graf 11](#_Toc514380860)

[II.4.1 *Swan* 11](#_Toc514380861)

[II.4.2 *VisMod* 13](#_Toc514380862)

[II.4.3 *jGRASP* 14](#_Toc514380863)

[II.4.4 *Jype* 17](#_Toc514380864)

[II.4.5 *Online Python Tutor* 18](#_Toc514380865)

[II.5 Kesimpulan Awal Berdasarkan Tinjauan Pustaka dan Eksplorasi 25](#_Toc514380866)

[Bab III Analisis Masalah 27](#_Toc514380867)

[III.1 Analisis Tujuan Visualisasi Graf 27](#_Toc514380868)

[III.2 Analisis Desain Visualisasi Graf 29](#_Toc514380869)

[III.3 Analisis Deteksi Graf dalam Kode Program 33](#_Toc514380870)

[III.3.1 Representasi Data Graf dalam Kode Program 34](#_Toc514380871)

[III.3.2 Analisis Kakas Pendukung untuk Deteksi Data Graf 35](#_Toc514380872)

[III.3.3 Analisis Metode Deteksi Graf 38](#_Toc514380873)

[III.3.4 Analisis Kakas Pendukung untuk Visualisasi Graf 41](#_Toc514380874)

[III.4 Analisis Kebutuhan Perangkat 43](#_Toc514380875)

[III.4.1 Kebutuhan Perangkat Lunak 43](#_Toc514380876)

[III.4.2 Kebutuhan Perangkat Keras 44](#_Toc514380877)

[Bab IV Perancangan dan Implementasi Kakas 45](#_Toc514380878)

[IV.1 Perancangan Kakas 45](#_Toc514380879)

[IV.1.1 Gambaran Umum 45](#_Toc514380880)

[IV.1.2 Perancangan Antarmuka Pengguna 46](#_Toc514380881)

[IV.1.3 Perancangan Diagram Kelas 47](#_Toc514380882)

[IV.1.4 Proses Konstruksi Visualisasi Data 48](#_Toc514380883)

[IV.2 Implementasi Kakas 56](#_Toc514380884)

[IV.2.1 Lingkungan Implementasi 56](#_Toc514380885)

[IV.2.2 Implementasi Modul Visualisasi Graf 57](#_Toc514380886)

[IV.2.3 Implementasi Antarmuka Pengguna 57](#_Toc514380887)

[IV.2.4 Batasan Implementasi 60](#_Toc514380888)

[Bab V Evaluasi Visualisasi 62](#_Toc514380889)

[V.1 Survei Pengguna dengan Kuesioner *Online* 62](#_Toc514380890)

[V.2 Hasil Kuesioner *Online* 66](#_Toc514380891)

[V.3 Analisis Hasil Kuesioner 66](#_Toc514380892)

[Bab VI Kesimpulan dan Saran 67](#_Toc514380893)

[VI.1 Kesimpulan 67](#_Toc514380894)

[VI.2 Saran 67](#_Toc514380895)

[DAFTAR PUSTAKA 68](#_Toc514380896)

[LAMPIRAN 71](#_Toc514380897)

# DAFTAR GAMBAR DAN ILUSTRASI

[Gambar II.1 Keterhubungan antar tujuh tahapan (Fry, 2008) 8](#_Toc514381213)

[Gambar II.2 Model konseptual komponen interaksi (Fry, 2008) 10](#_Toc514381214)

[Gambar II.3 *Swan*: Dua visualisasi graf dengan algoritma *minimum spanning tree* (Shaffer dkk., 1996) 12](#_Toc514381215)

[Gambar II.4 *Swan: Binary search tree (Shaffer dkk., 1996)* 13](#_Toc514381216)

[Gambar II.5 *VisMod*: Visualisasi menghapus elemen dari *list* (Jiménez-Peris dkk., 1999) 14](#_Toc514381217)

[Gambar II.6 *jGRASP*: Tampilan ketika *node* dibuat dan ditambahan ke *root binary tree* (Cross II dkk., 2007) 15](#_Toc514381218)

[Gambar II.7 *jGRASP*: *Visualisasi* pohon dari struktur data *TreeMap* (Hendrix dkk., 2004) 16](#_Toc514381219)

[Gambar II.8 *Jype*: visualisasi *binary search tree* dari tipe data *array* (Helminen dan Malmi, 2010) 17](#_Toc514381220)

[Gambar II.9 Turunan kelas visualisasi *Matrix Framework* (Korhonen dkk., 2004) 18](#_Toc514381221)

[Gambar II.10 Tampilan antarmuka OPT untuk bahasa pemrograman C 19](#_Toc514381222)

[Gambar II.11 Tampilan antarmuka OPT untuk kolaborasi pemrograman 19](#_Toc514381223)

[Gambar II.12 Tampilan simulasi dan visualisasi kode program *C pointer* 20](#_Toc514381224)

[Gambar II.13 Arsitektur OPT untuk visualisasi kode program C dan C++ 21](#_Toc514381225)

[Gambar II.14 Ilustrasi sederhana proses kakas OPT 22](#_Toc514381226)

[Gambar II.15 Ilustrasi format *Trace* eksekusi kode program 23](#_Toc514381227)

[Gambar II.16 Peran Utama *D3.js* sebagai pendukung Visualisasi 24](#_Toc514381228)

[Gambar III.1 Visual graf berarah dengan dua *node* dan *edge* 31](#_Toc514381229)

[Gambar III.2 OPT: Visualisasi graf dengan *array* dimensi-2 (matriks) 33](#_Toc514381230)

[Gambar III.3 OPT: Visualisasi graf dengan *struct* dan *pointer* 33](#_Toc514381231)

[Gambar III.4 Contoh data matriks sebagai representasi graf 34](#_Toc514381232)

[Gambar III.5 Contoh data *array* atau *edge list* 34](#_Toc514381233)

[Gambar III.6 Contoh data *adjacency list* dengan indeks *node* 35](#_Toc514381234)

[Gambar III.7 Hasil *compile* kode program *C* dengan *GCC* 36](#_Toc514381235)

[Gambar III.8 Hasil keluaran dari *Valgrind* 3.11 versi asli 37](#_Toc514381236)

[Gambar III.9 Hasil keluaran dari *Valgrind* versi modifikasi 37](#_Toc514381237)

[Gambar III.10 Isi berkas *vgtrace* 38](#_Toc514381238)

[Gambar III.11 Hasil keluaran berupa *JSON* dari *Valgrind* versi modifikasi 38](#_Toc514381239)

[Gambar III.12 OPT: diagram alir proses perolehan data *JSON* 39](#_Toc514381240)

[Gambar III.13 OPT: Format data eksekusi *trace JSON* 39](#_Toc514381241)

[Gambar III.14 Contoh data *JSON pointer* pada atribut *heap* 40](#_Toc514381242)

[Gambar IV.1 Skema umum hasil pengembangan kakas 45](#_Toc514381243)

[Gambar IV.2 Skema umum modul visualisasi graf (*GraphVisualizer*) 45](#_Toc514381244)

[Gambar IV.3 Rancangan antarmuka pengguna 46](#_Toc514381245)

[Gambar IV.4 Rancangan diagram kelas 47](#_Toc514381246)

[Gambar IV.5 Klasifikasi data *JSON* menjadi dua bagian 51](#_Toc514381247)

[Gambar IV.6 Klasifikasi dari data matriks atau *pointer* 51](#_Toc514381248)

[Gambar IV.7 Contoh data *JSON pointer* yang telah di-*filter* 52](#_Toc514381249)

[Gambar IV.8 Diagram alir *level-1* pencocokan model 53](#_Toc514381250)

[Gambar IV.9 Diagram alir *level-0* proses visualisasi data 54](#_Toc514381251)

[Gambar IV.10 Diagram alir *level-1* proses *rendering* visualisasi graf 54](#_Toc514381252)

[Gambar IV.11 Visual graf: (a) visual dasar graf; (b) setelah perbaikan 55](#_Toc514381253)

[Gambar IV.12 (a) Perbaikan desain interaksi visual; (b) Fitur animasi 56](#_Toc514381254)

[Gambar IV.13 Implementasi antarmuka pengguna untuk *input* kode program 57](#_Toc514381255)

[Gambar IV.14 Implementasi antarmuka pengguna visualisasi graf kode program; (a) Navigasi kontrol; (b) *Slider*; (c) Panel visualisasi 58](#_Toc514381256)

[Gambar IV.15 Implementasi visualisasi graf berbobot tak-berarah 58](#_Toc514381257)

[Gambar IV.16 Implementasi visualisasi graf berbobot berarah 59](#_Toc514381258)

[Gambar IV.17 Implementasi panel “*Primitif Visualization*” 59](#_Toc514381259)

[Gambar IV.18 Implementasi panel “*Print Output*” 60](#_Toc514381260)

[Gambar IV.19 Implementasi fitur animasi pencarian rute antar *node* 60](#_Toc514381261)

# DAFTAR TABEL

[Tabel III.1 Daftar properti visual graf 31](#_Toc514381262)

[Tabel III.2 Variabel *‘a’* berupa matriks berdimensi 7 x 7 40](#_Toc514381263)

[Tabel III.3 Contoh isi alamat memori pada objek *heap* 41](#_Toc514381264)

[Tabel III.4 Daftar *library* *JavaScript* untuk visualisasi graf 43](#_Toc514381265)

[Tabel III.5 Kakas dan *library* pendukung pengembangan OPT 44](#_Toc514381266)

[Tabel IV.1 Contoh data eksekusi *trace* *JSON* berupa matriks 49](#_Toc514381267)

[Tabel IV.2 Daftar label untuk klasifikasi data *JSON* 50](#_Toc514381268)

[Tabel IV.3 Format dasar yang digunakan untuk visualisasi graf dengan *D3.js* 53](#_Toc514381269)

[Tabel IV.4 Format untuk *edge* dengan bobot 54](#_Toc514381270)

[Tabel V.1 Daftar anggota sampel yang akan mengisi kuesioner 65](#_Toc514381271)

# Bab I Pendahuluan

Pada bab ini dijelaskan mengenai latar belakang, rumusan masalah, tujuan penelitian, batasan implementasi, metodologi penelitian, dan sistematika penulisan laporan tesis.

## I.1 Latar Belakang

Proses belajar pemrograman tidak lepas dari melakukan eksekusi kode program. Belajar pemrograman bagi sebagian peserta didik tidak mudah. Selain pemahaman tentang algoritma, kegiatan mengonstruksi kode program adalah bagian penting untuk implementasi dari algoritma yang telah dirancang.

Bagi pengajar untuk menjelaskan proses eksekusi kode program di kelas terkadang menggunakan papan tulis atau *slide PowerPoint*. Hal ini membutuhkan persiapan ekstra seperti gambar, alur diagram, atau bagian-bagian kode program terkait materi yang akan dijelaskan. Terutama jika materi pemrograman itu masuk ke tingkat yang lebih rumit, seperti struktur data. Untuk dapat menjelaskan proses eksekusi kode program tersebut diperlukan media belajar khusus, sehingga peserta didik dapat memahami proses yang sebenarnya terjadi di dalam program komputer.

Sebuah penelitian yang telah dilakukan (Piteira dan Costa, 2013) di Institut Politeknik Setubal, Portugal, menemukan bahwa konsep pemrograman struktur data memiliki tingkat kesulitan yang tinggi bagi sebagian besar peserta didik. Hal itu disebabkan konsep pemrograman untuk struktur data merupakan konsep data abstrak, yang kurang dipahami oleh peserta didik jika ditulis dalam bentuk kode program. Penelitian tersebut juga memberikan perhatian khusus terhadap konsep pemrograman struktur data seperti graf, *pointer*, parameter, dan *abstract data types* (ADT).

Visualisasi sebagai media belajar pemrograman bukan suatu hal yang baru. Para peneliti telah banyak mengembangkan kakas visualisasi untuk membantu mempelajari algoritma dan pemrograman (Cetin dan Andrews-Larson, 2016; Sorva, 2012; Sorva dkk., 2013; Gračanin dkk., 2005; Guo, 2013). Karena melalui interaksi visual, manusia lebih cenderung menangkap lebih banyak informasi yang diterima dibandingkan melalui indera lainnya (Ware, 2004). Visualisasi dapat mendukung interaksi yang efisien dan efektif untuk beragam pekerjaan kognitif seperti menganalisis, meringkas, dan menarik kesimpulan atas informasi yang diperoleh.

Saat ini teknologi internet dan web memberikan kemudahan akses untuk berbagi informasi (Bonk, 2009). Aplikasi web terus berkembang dari sekadar penyampaian informasi kontekstual statis, sehingga berdampak terhadap kemutakhiran penyajian visualisasi informasi secara dinamis. Teknologi seperti *Java3D*, *VRML*, *X3D*, dan *SVG* memiliki kemampuan *rendering* yang *powerful*, tetapi sulit untuk berinteraksi dengan sumber data mentah (Holmberg dkk., 2006).

Perkembangan kakas visualisasi eksekusi kode program atau dengan istilah visualisasi program (VP) berbasis web untuk graf masih sangat langka (Sorva, 2012; Sorva dkk., 2013). Sorva (2012) menelaah secara komprehensif perkembangan 40 kakas VP. Sebagian besar kakas menggunakan *Java Applet* untuk dapat beroperasi di web. Sedangkan *Java Applet* masih perlu di-*install* dan konfigurasi pada *browser*, sehingga ini tidak memberikan kemudahan akses untuk menggunakannya.

Philip Guo (2010) telah mengembangkan kakas VP bernama *Online Python Tutor* (OPT). Kakas berbasis web ini memiliki fitur *embeddable* yang mudah digunakan. Fitur tersebut digunakan untuk melampirkan visualisasi di halaman web lain. Kakas ini menggunakan *library* *D3.js* (Bostock dkk., 2011) sebagai teknologi utama untuk mendukung visualisasi. Namun, OPT belum dikembangkan untuk visualisasi struktur data graf.

Pada penelitian tesis ini, dilakukan pengembangan kakas berdasar pada kode sumber dari OPT. Selain telah mendukung teknologi web, kakas ini bersifat bebas (*free*) dan bersumber terbuka (*open source*) (Guo, 2013). Hasil pengembangan kakas ini diharapkan dapat memudahkan peserta didik untuk memahami eksekusi kode program yang terdapat algoritma graf.

## I.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan tersebut, diperoleh rumusan masalah sebagai berikut:

1. Apakah yang dimaksud dengan memahami eksekusi graf kode program ?
2. Apakah dengan visualisasi graf dapat efektif untuk memahami eksekusi graf kode program ?
3. Bagaimana teknik untuk mendeteksi graf pada eksekusi graf kode program sehingga dapat divisualisasi ?

## I.3 Tujuan Penelitian

Tujuan dari penelitian tesis ini adalah menghasilkan kakas visualisasi graf yang dapat digunakan untuk memahami eksekusi graf kode program.

## I.4 Batasan Masalah

Batasan masalah pada tesis ini adalah sebagai berikut:

1. Permasalahan yang dikaji fokus terhadap keefektifan visualisasi untuk memahami eksekusi graf kode program. Contohnya, dengan menggunakan visualisasi menjadi lebih efektif dalam memahami eksekusi graf kode program dibandingkan tanpa visualisasi.
2. Tidak melakukan analisis mendalam untuk sistem *backend* terkait kinerja manajemen memori dan penggunaan kakas pendukungnya.
3. Hasil pengembangan kakas tidak diperuntukan *debugging* kode program berskala besar.
4. Permasalahan yang dikaji tidak mencangkup algoritma pembangkit model graf, seperti model Barabási–Albert.

## I.5 Metodologi Penelitian

Uraian langkah-langkah penelitian dijelaskan pada bagian ini yang dilaksanakan melalui beberapa tahapan sebagai berikut:

1. Tinjauan Pustaka dan Eksplorasi

Tahap ini menghasilkan kesimpulan awal yang dapat dijadikan landasan dalam perumusan masalah dan pengembangan kakas.

1. Merumuskan Hipotesis

Merumuskan hipotesis solusi yang dapat diterapkan berdasarkan hasil tinjauan pustaka dan eksplorasi kakas VP.

1. Pengumpulan dan Analisis Graf Kode Program

Tahap ini mengumpulkan kode program, kemudian menganalisisnya dengan berbagai jenis algoritma graf. Kode program ini sebagai *input* utama untuk visualisasi.

1. Analisis Tujuan Visualisasi Graf

Menganalisis tujuan dan spesifikasi kebutuhan untuk visualisasi agar proses memahami eksekusi kode program graf dapat tercapai.

1. Analisis Deteksi Graf dalam Kode Program

Menganalisis kode program yang berisi algoritma graf agar dapat divisualisasi sehingga diperoleh suatu teknik yang relevan.

1. Perancangan Kakas

Proses membuat rancangan berdasarkan hipotesis dan hasil analisis untuk dijadikan dasar implementasi kakas.

1. Eksperimen Visualisasi Graf

Tahap ini melakukan simulasi visual graf menggunakan data *dummy* dan *library* pendukung agar sesuai dengan hasil analisis desain visualisasi dan perancangan kakas.

1. Implementasi Kakas

Implementasi kakas berdasarkan rancangan dan hasil eksperimen visualisasi graf.

1. Evaluasi Visualisasi

Evaluasi akan dilakukan dengan survei pengguna melalui kuesioner *online*. Pengguna akan simulasi menggunakan kakas sebelum pengembangan, kemudian dibandingkan dengan menggunakan kakas yang telah dikembangkan ini.

## I.6 Sistematika Penulisan

Penulisan laporan hasil penelitian tesis ini akan dibagi menjadi enam bab, yaitu:

1. Bab I Pendahuluan

Bab ini berisi latar belakang, rumusan masalah, tujuan, batasan, metodologi penelitian, dan sistematika penulisan laporan tesis.

1. Bab II Tinjauan Pustaka dan Eksplorasi

Bab ini berisi uraian tentang terminologi visualisasi perangkat lunak, metodologi dan prinsip visualisasi data serta teori desain interaksi. Bab ini juga membahas perkembangan kakas dari hasil penelitian sebelumnya yang berkaitan dengan masalah yang dikaji, sehingga memberikan gambaran perkembangan terhadap masalah yang akan diteliti.

1. Bab III Analisis Masalah

Bab ini berisi analisis terhadap masalah yang akan diteliti, kemudian dirumuskan hipotesis untuk disusun solusi yang mungkin dapat diterapkan.

1. Bab IV Perancangan dan Implementasi Kakas

Bab ini menjelaskan perancangan dan implementasi kakas VP.

1. Bab V Evaluasi Visualisasi

Bab ini menguraikan proses evaluasi dengan survei pengguna dari visualisasi yang telah dikembangkan.

1. Bab VI Penutup

Bab ini berisi kesimpulan dan saran untuk pengembangan kakas lebih lanjut.

# Bab II Tinjauan Pustaka dan Eksplorasi

Bab ini berisi uraian hasil penelitian terdahulu yang berkaitan dengan masalah yang akan dikaji, sehingga memberikan gambaran perkembangan pengetahuan yang mendasari penulisan tesis.

## II.1 Terminologi Visualisasi Perangkat Lunak

Visualisasi perangkat lunak (*Software Visualization*) adalah salah satu bidang aktif dalam riset dan pengembangan sistem. Berbagai sistem visualisasi perangkat lunak bermunculan untuk digunakan dengan tujuan tertentu dan terus berkembang setiap tahunnya (Sorva, 2012). Gračanin dkk., (2005) mendefinisikan visualisasi perangkat lunak (*Software Visualization*) sebagai suatu bidang untuk menginvestigasi dengan pendekatan dan teknik tertentu yang bertujuan dalam merepresentasikan grafis algoritma secara statis atau dinamis, program (*code*), dan data yang diproses. Visualisasi perangkat lunak memiliki tujuan utama untuk menganalisis program dan pengembangan; untuk meningkatkan pemahaman terhadap konsep yang tak nampak dan cara kerja perangkat lunak. Tantangan utamanya adalah mencari langkah efektif dalam pemetaan berbagai aspek perangkat lunak untuk direpresentasikan secara grafis menggunakan metafora visual. Dengan kata lain, visualisasi perangkat lunak tidak fokus terhadap proses konstruksi program, akan tetapi lebih kepada analisis program dan proses pengembangan perangkat lunak.

Istilah “visualisasi perangkat lunak” didefinisikan sebagai penggunaan sebuah seni tipografi, desain grafis, animasi, dan sinematografi melalui interaksi modern antar manusia-komputer dan teknologi grafis komputer sebagai sarana untuk memahami dan keefektivan penggunaan perangkat lunak komputer (Cetin dan Andrews-Larson, 2016). Visualisasi perangkat lunak dibagi menjadi dua, yaitu visualisasi algoritma (VA) dan visualisasi program (VP) (Cetin dan Andrews-Larson, 2016; Sorva, 2012; Sorva dkk., 2013). VA berkaitan dengan abstraksi terhadap algoritma, konsep dan langkah kerja suatu perangkat lunak, sedangkan VP berkaitan dengan cara kerja eksekusi kode program dan proses struktur data.

Visualisasi program (VP) terdiri dari pembuatan animasi eksekusi program. Animasi ini membuat hasil visual yang sangat mengesankan. Hal ini tidak terbatas pada algoritma atau aktivitas dari eksekusi kode program. Beberapa aktivitas yang terjadi saat kode program dikompilasi adalah sistem *run-time*, proses data, dan bahkan perangkat keras yang mendasarinya. Sebagian besar sistem VP terbatas pada satu bahasa pemrograman, yang biasanya sama dengan bahasa yang digunakan untuk menulis sistem visualisasi tersebut. Namun ini bukan hal yang mutlak untuk diimplementasikan.

Kakas VP memiliki beberapa manfaat, yaitu (Sorva dkk., 2013):

1. *Debugging* atau proses mencari kesalahan dalam kode program;
2. Mengevaluasi dan memperbaiki performa suatu kode program;
3. Mengevaluasi dan mengurangi pemanfaatan sumber daya;
4. Mengevaluasi algoritma dalam konteks kode program lengkap dan data riil;
5. Memahami perilaku kode program;
6. Sebagai media belajar dan mengajar.

## II.2 Metodologi dan Prinsip Visualisasi Data

Kompleksitas data tidak akan bermakna menjadi sebuah informasi yang bermanfaat jika memahaminya dengan cara yang kurang tepat. Proses visualisasi dilakukan dengan mengombinasikan berbagai bidang disiplin ilmu, seperti statistik, *data mining*, desain grafis, dan visualisasi informasi. Desain grafis tidak menangani jutaan data, akan tetapi *data mining* yang memiliki kapasitas itu. Visualisasi perangkat lunak membuat interaksi dan berbagai jenis representasi data abstrak, namun metode yang digunakan untuk visual dinilai kurang aestetik karena tidak adanya keterlibatan dari desain grafis. Oleh karena itu, ketiga bidang disiplin ilmu tersebut harus saling melengkapi menjadi sebuah proses yang dapat mengomunikasikan data lebih efektif untuk direpresentasikan (Fry, 2008).

Kombinasi tahapan dari ketiga bidang ilmu tersebut dibutuhkan untuk visualisasi data. Meskipun ketiga bidang itu memiliki prinsipnya sendiri, tetapi harus disesuaikan setiap bagian menjadi kesatuan proses. Prosesnya dimulai dengan memahami seperangkat data mentah dan mengajukan sebuah pertanyaan yang berkaitan dengan data tersebut. Secara rinci ada tujuh tahapan untuk konstruksi visualisasi data (Fry, 2008):

1. *Acquire* (sumber data); cara perolehan sebuah data, apakah dari sebuah berkas di dalam *harddisk* komputer atau bersumber dari jaringan internet.
2. *Parse* (pengelompokan); mengurai struktur data menjadi makna tertentu dan mengategorikannya.
3. *Filter* (menyaring); menghapus data yang tidak penting sehingga tersisa data yang menjadi fitur.
4. *Mine* (menggali informasi); terapkan metode dari ilmu statistik atau *data mining* sebagai cara untuk memahami data sesuai konteksnya.
5. *Represent* (merepresentasikan); pilih visual model dasar seperti diagram batang, diagram garis, atau graf.
6. *Refine* (perbaikan visual); lakukan perbaikan terhadap representasi visual dasar untuk membuat lebih jelas dan menarik.
7. *Interact* (interaksi); tambahkan interaksi untuk memanipulasi data atau sebuah kontrol terhadap fitur data yang disajikan.

Pada Gambar II.1 menunjukkan urutan tahap dan keputusan akhir berdasarkan tahapan yang telah dilakukan sebelumnya. Setiap tahap pada proses ini saling terhubung karena saling memengaruhi satu dengan yang lain.



1. Keterhubungan antar tujuh tahapan (Fry, 2008)

Hubungan setiap tahapan menggambarkan pentingnya kerja sama antar individu atau kerja tim dalam kesatuan proyek. Biasanya seorang *programmer* menangani bagian teknis, seperti cara perolehan dan pengolahan data, dan *desainer* fokus terhadap pemilihan warna dan tampilan antarmuka. Ketika proses pengambilan data, harus ditentukan kapan data itu harus tersedia, setiap waktu atau sekali dalam sebulan. Hal ini juga menentukan desain grafis yang cocok untuk masalah ketersediaan data berikutnya.

Ada tiga prinsip utama untuk membuat visualisasi data (Fry, 2008), yaitu :

1. Sadari bahwa setiap proyek visualisasi memiliki spesifikasi kebutuhan yang unik, karena tidak setiap data dapat direpresentasikan ke bentuk visual yang sama;
2. Hindari visual data yang terlalu rumit atau kurangi rincian data yang dapat mengganggu fitur data tersebut, sehingga pengguna tidak dapat melihat apa yang menjadi fokus dalam visualisasi tersebut; dan
3. Pahami konsumen atau pengguna dari visualisasi tersebut. Visualisasi yang ditampilkan menggunakan komputer meja dibandingkan dengan gawai, tentu berbeda dalam hal interaksi visual dan tampilan kerumitan datanya.

## II.3 Teori Desain Interaksi

Desain interaksi secara khusus memiliki makna untuk menciptakan pengalaman pengguna (*user experience* / UX) dalam meningkatkan fungsi suatu produk sehingga dapat saling berkomunikasi dan berinteraksi dengan manusia. Winograd (Preece, 2002) mendeskripsikan bahwa desain interaksi adalah ruang untuk berkomunikasi dan berinteraksi dengan manusia yang bertujuan untuk mendukung aktivitas manusia. Komunikasi dan interaksi merupakan pencerminan dari aksi dan reaksi terhadap sesuatu. Media interaktif dikatakan efektif dan aestetik ketika dapat memberikan reaksi terhadap setiap aksi yang dilakukan oleh pengguna sehingga tercipta komunikasi.

Untuk dapat memahami konsep interaksi, dapat melihat pada Gambar II.2. Pada gambar tersebut, ada tiga esensi utama untuk bisa dikatakan sebagai sebuah interaksi pada desain produk (baik itu perangkat lunak atau keras), yaitu *designer*, pengguna, dan sistem. Secara tidak langsung, ketiga komponen ini saling berkaitan membentuk model konseptual (Fry, 2008) sebagai berikut:

1. Model desain—model yang dibuat oleh *designer* bagaimana sistem itu seharusnya bekerja;
2. Sistem—bagaimana sistem itu sebenarnya bekerja;
3. Model pengguna—bagaimana pengguna dapat memahami sistem itu bekerja.



1. Model konseptual komponen interaksi (Fry, 2008)

Idealnya dalam dunia nyata, ketiga komponen tersebut saling memetakan dengan komponen lainnya. Pengguna harus dapat melakukan aktivitas atau tugasnya sesuai yang didesain untuk dapat berinteraksi dengan sistem. Jika pengguna tidak dapat memahami model desain, maka tidak akan dapat memahami kerja sistem sehingga sistem akan berjalan dengan tidak efektif dan muncul kesalahan sistem (*error*).

Aspek kognisi (Fry, 2008) adalah salah satu kemampuan yang dimiliki oleh pengguna seperti berpikir, memperhatikan, mempelajari, mengingat, persepsi, membuat keputusan, merencanakan, membaca, berbicara, dan mendengar. Aspek ini sangat penting dan memiliki relevansi terhadap model desain dan proses interaksi. Sebuah tampilan antarmuka dapat dikatakan baik jika pengguna dapat mempersepsikan, mempelajari, dan mengingatnya untuk menyelesaikan tugas atau aktivitasnya. Kognisi memiliki tiga perspektif atau pendekatan (Fry, 2008), yaitu:

1. Mental model—didefinisikan sebagai pemahaman dan pengalaman dasar yang dimiliki pengguna terhadap kebiasaan menggunakan suatu produk;
2. Memproses informasi—pendekatan ini memberikan informasi yang mudah dipersepsikan bagi pengguna untuk menggunakan produk;
3. Kognisi eksternal—merupakan proses kognisi untuk berinteraksi dengan berbagai bentuk representasi atau media eksternal dalam membantu pengguna untuk mendukung aktivitasnya.

Secara esensi proses pengembangan desain interaksi memiliki empat tahap utama (Preece dkk., 2015), yaitu:

1. Mengidentifikasi kebutuhan pengguna dan menetapkan persyaratan khusus;
2. Mengembangkan alternatif desain yang sesuai dengan persyaratan tersebut;
3. Membangun purwarupa versi desain interaktif sehingga dapat saling berkomunikasi dan dinilai kelayakannya;
4. Mengevaluasi desain selama proses pengembangan.

Keempat tahap tersebut saling berkaitan dan harus selalu diulang untuk memastikan desain yang dibangun telah memenuhi syarat dan kebutuhan. Sebagai contoh, keefektifan penggunaan suatu produk yang telah dibangun dapat diukur dengan memperhatikan setiap perbaikan yang telah dilakukan atau dengan melihat persyaratan yang belum terpenuhi.

Pengguna mengharapkan sebuah sistem dan produk yang mudah untuk dipelajari dan digunakan dengan efektif, efisien, aman, dan memuaskan. Tampilan yang menarik dan atraktif juga merupakan esensi dari beberapa produk. Namun, semua itu tidak dapat secara langsung terwujud, akan tetapi diperlukan evaluasi secara terus-menerus dan berulang melalui pengujian dan survei kepada pengguna. Terdapat tiga prinsip yang direkomendasikan dalam hal ini (Fry, 2008), yaitu (1) fokus terhadap pengguna dan aktivitasnya terhadap produk tersebut; (2) amati, ukur, dan analisis interaksi pengguna dengan produk atau sistem; dan (3) lakukan desain secara berulang dengan versi berbeda. Evaluasi ini dibutuhkan untuk memastikan pengguna dapat menggunakan produk atau sistem dengan nyaman.

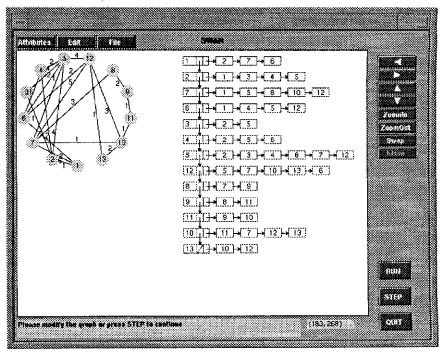
## II.4 Eksplorasi Kakas Visualisasi Program untuk Graf

Ada lima kakas VP yang telah dikembangkan untuk graf, yaitu:

### II.4.1 *Swan*

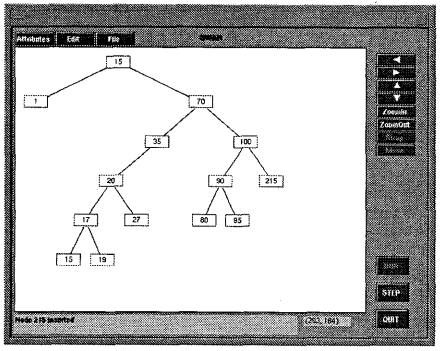
Swan (Shaffer dkk., 1996) adalah kakas VP untuk struktur data dan eksekusi kode C/C++. Kakas ini dapat memvisualisasikan graf, pohon, *list*, dan *array*. Metodenya dengan menggunakan anotasi pada kode program yang disebut SAIL (*Swan Annotation Interface Library*). Tujuan utama kakas Swan adalah untuk membuat *library* anotasi yang mudah digunakan dalam visualisasi.

*Swan* memiliki tiga komponen utama: (1) *Swan Annotation Interface Library* (SAIL), (2) *kernel Swan*, dan (3) *Swan Viewer Interface* (SVI). SAIL adalah *library* yang berisi fungsi-fungsi untuk menganotasi kode program terhadap visualisasi yang akan direpresentasikan. SVI adalah hasil visualisasi yang dapat dieksplor dari anotasi kode program. *Kernel Swan* adalah modul utama yang ada di dalam kakas. Modul ini berfungsi untuk mengonstruksi, memanajemen, dan merepresentasikan visualisasi yang dibangkitkan berdasar SAIL. Modul ini juga dapat berfungsi sebagai komunikasi antara SVI dengan SAIL untuk memanipulasi visualisasi.



1. *Swan*: Dua visualisasi graf dengan algoritma *minimum spanning tree* (Shaffer dkk., 1996)

Pada Gambar II.3 menunjukkan representasi fisik berupa *adjacency list* dari sebuah graf di sebelah kanan, yang diimplementasi menggunakan anotasi program. Visualisasi sebelah kiri merupakan representasi logik untuk graf, yang merepresentasikan secara abstrak dari *adjacency list*. Pada gambar tersebut menunjukkan komunikasi antara hasil proses anotasi program dengan SVI. Ada sebuah panel kontrol dan pembagian tiga *window* dalam *window* SVI: *display window*, *I/O window*, dan *location window*. *Display window* menampilkan graf sebagai *output* dari kakas Swan. *I/O window* digunakan sebagai *annotator* dan menampilkan pesan per baris atau menerima input dari visualisasi. Kemudian *location window* menampilkan nilai koordinat posisi kursor di *display window*.



1. *Swan: Binary search tree (Shaffer dkk., 1996)*

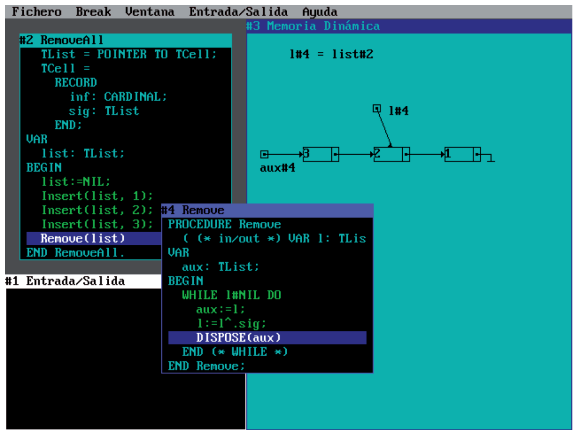
Sebuah *node* atau *edge* yang tampil di *display window* dapat dieksplor. Aksi seperti *pan* dan *zoom* dapat dilakukan pada visualisasi graf; dapat mengubah keadaan *RUN* atau *Step*; dan dapat memodifikasi atribut visual graf. Modifikasi yang dapat dilakukan untuk atribut *node* terdiri dari bentuk, warna, ukuran, dan ketebalan garis. Sedangkan atribut *edge* terdiri dari warna dan ketebalan garis.

### II.4.2 *VisMod*

*VisMod* (Jiménez-Peris dkk., 1999) dapat memvisualisasikan eksekusi kode program dengan bahasa pemrograman *Modula-2*. Kakas ini mendukung visualisasi struktur data linier dan pohon. Metodenya dengan membaca data variabel *pointer* dan referensinya. Selain itu, kakas ini dapat mengecek kesalahan sintaks dan memastikan penggunaan semua variabel yang telah dideklarasikan sebelumnya.

*Pointer* direpresentasikan dengan kotak dan alamat yang dituju ditandai dengan anak panah (lihat Gambar II.5). Kotak diberi label dengan nama variabel atau parameter yang telah didefinisikan. Untuk membedakan antara parameter *pointer* dan variabel (biasanya dalam proses pemanggilan rekursif), pada *window* diberi label angka berurut yang berisi proses eksekusi program.

*VisMod* belum terdapat fitur *reverse execution*, yaitu kembali ke langkah eksekusi program sebelumnya. *VisMod* hanya memiliki tiga cara dalam mengeksekusi program: pertama *step-by-step*, kedua dengan *breakpoint*, dan ketiga proses eksekusi secara lengkap. Mode *step-by-step* memberikan akses kepada pengguna untuk mengontrol jalannya eksekusi pada setiap baris kode program. Ketika beberapa bagian program telah paham, pengguna dapat melewati proses eksekusi dengan *breakpoint*. Pengguna dapat fokus ke bagian kode program yang menjadi perhatian khusus.

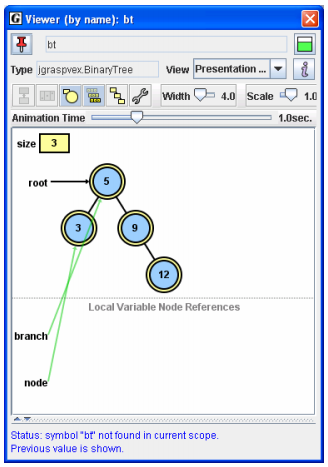
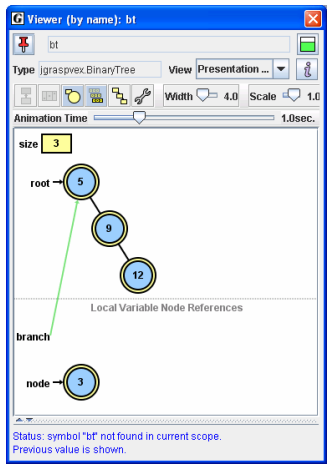


1. *VisMod*: Visualisasi menghapus elemen dari *list* (Jiménez-Peris dkk., 1999)

### II.4.3 *jGRASP*

jGRASP (Cross II dkk., 2007) adalah kakas visualisasi eksekusi kode program yang mendukung bahasa *Java*. Kakas ini dapat memvisualisasi *binary tree* dan *linked list*. Visualisasi berdasarkan pada tipe data *pointer* yang direpresentasikan sebagai *node* dan *reference* sebagai *edge*.

Identifikasi otomatis dilakukan dengan memeriksa struktur kelas dan keterhubungan antar kelas. Sebagai contoh, *linked list* diimplementasi dengan kelas yang berisi dua *field*, yaitu *head* sebagai *node* dan *link* sebagai *edge*. Metode seperti ini dapat menyebabkan visualisasi gagal ditampilkan, karena hanya memperkirakan berdasar nama variabel. Misalnya, sebuah kelas bernama *MyTree* sebagai *root*, memiliki dua *field* yaitu *left* dan *right*. Kelas ini sangat mendekati sebagai visual *binary tree*. Jika visualisasi yang dimaksud memang *binary tree*, maka dapat ditampilkan. Namun, ini bisa saja terjadi bahwa visualisasi bukan yang dimaksud. Kelemahan teknik ini adalah hanya akan bekerja jika bahasa yang digunakan untuk nama kelas dan *field* telah diketahui. Identifikasi penamaan variabel saat ini hanya untuk bahasa inggris. Penggunaan nama variabel yang tidak biasa atau tidak memiliki makna akan membuat identifikasi gagal. Jika identifikasi otomatis gagal, maka visualisasi dapat dikonfigurasi secara manual seperti pada Gambar II.6 berikut ini.



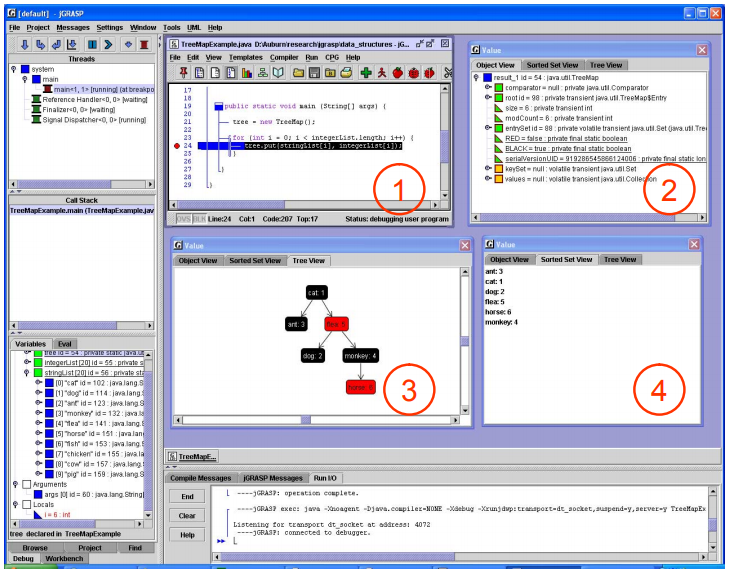
1. *jGRASP*: Tampilan ketika *node* dibuat dan ditambahan ke *root binary tree* (Cross II dkk., 2007)

Pada bahasa pemrograman *Java* terdapat struktur data seperti *linked list*, tumpukan, antrian, dan pohon yang merupakan sekumpulan objek yang memiliki beragam derajat kompleksitas (Hendrix dkk., 2004). *Java debugger* dan *workbench* menyediakan informasi mengenai rincian objek atau variabel yang sedang dieksekusi oleh kode program. Pemahaman dengan representasi abstrak akan berguna untuk memahami program dan *debugging*.

Sebagai contoh, kelas *TreeMap* dalam kode program *Java* di Gambar II.7. Pada label (1) menunjukkan kode program yang sedang dieksekusi, (2) representasi objek dengan tingkat rendah, (3) representasi tingkat tinggi dalam bentuk pohon, dan (4) representasi tekstual.

Kakas *jGRASP* dapat diintegrasikan dengan *Java debugger* agar dapat menampilkan visualisasi struktur data. Kode program dapat dijalankan dalam *debugger* atau langsung dari kakas *jGRASP* untuk memvisualisasikan struktur data. Pengguna dapat membuat visualisasi dari kelas yang telah dibuat sendiri, tidak hanya struktur data yang telah didefinisikan dari *Java library*.

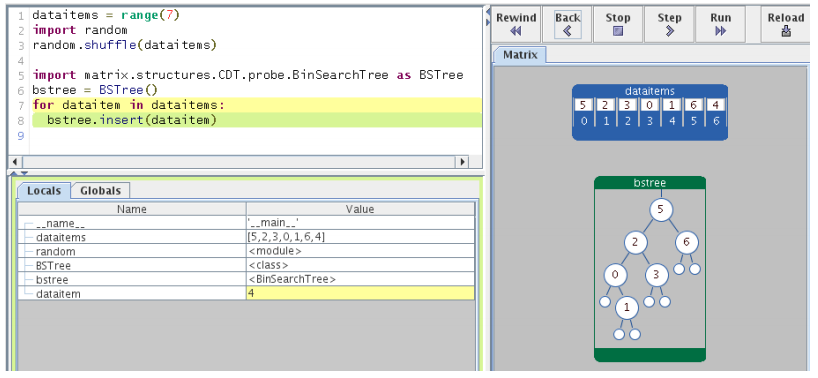
Kakas *jGRASP* tidak mendukung fitur *reverse execution*, yaitu untuk kembali ke langkah eksekusi sebelumnya (Gestwicki dan Jayaraman, 2005), sehingga tidak dapat dilakukan analisis dengan membandingkan kode program sebelum dan sesudah eksekusi pada baris kode program selanjutnya.



1. *jGRASP*: *Visualisasi* pohon dari struktur data *TreeMap* (Hendrix dkk., 2004)

### II.4.4 *Jype*

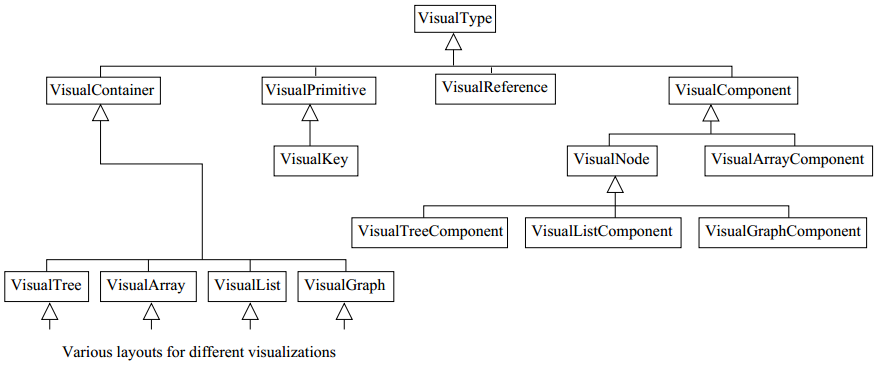
*Jype* (Helminen dan Malmi, 2010) menggunakan *Matrix Framework* (Korhonen dkk., 2004) untuk memvisualisasikan struktur data seperti *array* dan pohon secara otomatis ketika terdeteksi di dalam kode program *Python* (lihat Gambar II.8). *Jype* dikembangkan dengan *Java* yang dapat beroperasi di web sebagai *Java Applet*. Untuk mengaktifkannya, seluruh komponen dalam aplikasi harus diunduh, kemudian aplikasi dapat beroperasi di komputer.



1. *Jype*: visualisasi *binary search tree* dari tipe data *array* (Helminen dan Malmi, 2010)

*Matrix framework* (Korhonen dkk., 2004) adalah sebuah kerangka kerja yang berisi empat konsep visual utama, yaitu *visual container*, *visual component*, *visual reference*, dan *visual data*. *Visual container* berisi struktur yang kompleks, yang memiliki beberapa variabel seperti *node*, indeks, yang saling terhubung. Setiap variabel di dalam *visual container* terdapat *visual component* yang dapat memvisualisasikan variabel tersebut. Keterhubungan variabel dalam *visual component* dapat divisualisasikan dengan *visual reference*. Setiap variabel memiliki atribut kunci yang terhubung dengan *visual data*. Untuk lebih jelas dapat melihat Gambar II.9.

Kode sumber atau proyek *Jype* sudah tidak ditemukan (Guo, 2013). Kakas ini sudah tidak dikembangkan lebih lanjut.



1. Turunan kelas visualisasi *Matrix Framework* (Korhonen dkk., 2004)

### II.4.5 *Online Python Tutor*

*Online Python Tutor* atau disingkat OPT (Guo, 2013) adalah kakas VP yang bersifat bebas dan bersumber terbuka. Kakas ini telah memiliki banyak fitur yang mendukung berbagai macam bahasa pemrograman, seperti *Python, Java, C, C++, Ruby, JavaScript, TypeScript* dan masih terus dikembangkan[[1]](#footnote-1). Ketika penelitian tesis ini dibuat, OPT telah berkembang pada versi 5 yang dirilis pada tanggal 27 Juli 2016. Penelitian aktif dilakukan pada versi 5 dengan menggunakan bahasa pemrograman *TypeScript* sebagai dasar pembangunan kakas OPT.

Pada Gambar II.10 ditunjukkan tampilan antarmuka kakas OPT untuk bahasa pemrograman C. Fungsi nomor 1 adalah untuk berbagi antarmuka bagi pengguna lain yang ingin belajar pemrograman secara berkolaborasi. Ketika tombol tersebut diklik, maka tampilan berubah seperti pada Gambar II.11. Fungsi ini menggunakan *library* *TogetherJS*[[2]](#footnote-2) yang dikembangkan oleh *Mozilla Labs* secara bebas dan terbuka. Pengguna lain dapat ikut bergabung dengan mengunjungi pranala yang dibagikan oleh pengguna pertama yang mengaktifkan fitur ini.



1. Tampilan antarmuka OPT untuk bahasa pemrograman C



1. Tampilan antarmuka OPT untuk kolaborasi pemrograman

Fungsi nomor dua pada Gambar II.10 adalah untuk memasukkan kode program. Pengguna dapat mulai belajar bahasa pemrograman dengan mengetikkan kode program di *form editor* tersebut. Selanjutnya dapat menekan tombol pada nomor tiga untuk melihat simulasi dan visualisasi dari kode program yang telah dibuatnya, seperti pada Gambar II.12 berikut ini.



1. Tampilan simulasi dan visualisasi kode program *C pointer*

Pada beberapa subbab berikut dijelaskan susunan arsitektur dasar pembangunan kakas OPT yang mendukung proses VP khusus dalam bahasa pemrograman C dan C++.

#### II.5.5.1 Arsitektur Kakas OPT

Penjelasan arsitektur ini berguna untuk memberikan gambaran umum tentang cara kerja atau alur sistem didalamnya, sehingga dapat menentukan metode yang optimal. Selain itu, dapat dijadikan dasar dalam mengembangkan sistem dan menunjang modularisasi pada jangka waktu yang panjang.

Pada Gambar II.13 ditampilkan diagram alur komunikasi antara *browser* milik pengguna dengan *server*. Pertukaran informasi kode program dilakukan dengan fitur permintaan *AJAX* (*Asynchronous JavaScript And XML*).



1. Arsitektur OPT untuk visualisasi kode program C dan C++

Untuk lebih jelas tahapan proses pada Gambar II.13 di atas, berikut langkah-langkah yang terjadi ketika pengguna melakukan “Eksekusi Visualisasi” pada kode program.

1. Pada antarmuka *browser*, sebenarnya kode program tersebut dikirim sebagai tipe data *string* ke *server* dengan metode *AJAX request*.
2. Kemudian *server* mengeksekusi kode program tersebut dengan *NodeJS* yang dikombinasikan dengan *server* *Python* untuk menghasilkan *trace* v*algrind*.
3. *Server Python* mengubah format *trace* eksekusi sebagai *JSON* (*JavaScript Object Notation*) sebelum dikembalikan ke *server* *NodeJS*.
4. *JSONP* digunakan untuk memperoleh data dengan permintaan *AJAX* yang berbeda domain atau alamat *IP* (*Internet Protocol*). Proses ini dilakukan dalam *server NodeJS*.
5. Respon *AJAX* diterima oleh *visualizer* kemudian dibaca sesuai format yang ditentukan. Maka diperoleh visualisasi kode program yang tampil di *browser*.
6. Ketika pengguna menekan tombol “*Forward*” atau “*Back*” (lihat Gambar II.12) proses visualisasi akan membaca poin indeks yang berada pada *trace* eksekusi sesuai dengan banyak langkah yang telah terbentuk.

Alur sistem kakas OPT dapat disederhanakan menjadi beberapa inti komponen dengan representasi abstrak seperti terlihat pada Gambar II.14 berikut ini.



1. Ilustrasi sederhana proses kakas OPT

*OPT backend* berfungsi untuk merespon terhadap terbentuknya *trace* eksekusi kode program. Kemudian *trace* eksekusi diubah menjadi standar format yang ada pada subbab II.5.5.3 berbentuk *JSON*. *OPT frontend* (antarmuka *browser*) menerjemahkan *JSON* menjadi visualisasi yang dikombinasikan dengan *library* *D3.js*. Maka visualisasi dapat tampil kepada pengguna seperti pada Gambar II.12.

#### II.5.5.2 *Capturing Execution Trace*

Ketika kode program dikirim ke *OPT backend*, kode program C/C++ dieksekusi dengan *valgrind framework*. Kemudian dari *valgrind* terbentuk *trace* yang sudah dimodifikasi oleh pengembangnya. Versi *valgrind* yang digunakan adalah 3.11.0 dengan modifikasi beberapa baris kode program agar dapat menghasilkan *trace* eksekusi program. Beberapa berkas yang dimodifikasi adalah sebagai berikut:

|  |  |
| --- | --- |
| 1 | valgrind-3.11.0\memcheck\**mc\_main.c**  (Baris ke 54, 55, 3789, 5693, 5715-5718, 7394-7406, 7646-7647, 7761-7763) |
| 2 | valgrind-3.11.0\include\**pub\_tool\_debuginfo.h**  (Baris ke 36-37, 124-132, 191, 198, 214, 285-408) |
| 3 | valgrind-3.11.0\coregrind\m\_debuginfo\**debuginfo.c**  (Baris ke 74-1458, 5043-5159, 5472, 5594-5719, 5846) |
| 4 | valgrind-3.11.0\memcheck\**mc\_translate.c**  (Baris ke 6261-6481, 6665, 6708-6716, 6795-6811) |
| 5 | valgrind-3.11.0\coregrind\m\_debuginfo\**tytypes.c**  (Baris ke 47-51, 328-864) |
| 6 | valgrind-3.11.0\coregrind\m\_debuginfo\**priv\_tytypes.h**  (Baris ke 160-165) |
| 7 | valgrind-3.11.0\**config.h** (Baris ke 41) |
| 8 | valgrind-3.11.0\memcheck\**mc\_include.h** (Baris 142-156) |

Berkas-berkas *valgrind* tersebut dianalisis perbedaannya dengan menggunakan kakas *WinMerge* (*winmerge.org*). Tekniknya dengan membandingkan *valgrind* versi asli yang diunduh dari *www.valgrind.org*.

Untuk dapat menghasilkan *trace* eksekusi program, kode program dikirim ke *server* yang diolah oleh *ExpressJS*. Kemudian dengan program *Python* dihubungkan dengan *Pipe* ke *terminal* untuk di-*compile* dalam *valgrind debugger*. Dari sini menghasilkan *trace* khusus dari *valgrind* disebut *vgtrace*. Selanjutnya dikonversi ke bentuk *JSON* atau *JSONP* (*JSON with Padding*) oleh *ExpressJS* dan dikembalikan sebagai respon permintaan *AJAX*. Ilustrasi lengkap dapat dilihat pada Gambar II.13.

#### II.5.5.3 *Execution Trace Format*

*OPT back-end* menyimpan informasi *trace* eksekusi dalam standar format *JSON* yang kemudian dapat digunakan untuk berinteraksi dengan *visualizer* di *OPT front-end.* Sebagai ilustrasi format *trace* eksekusi program yang dimaksud dapat dilihat pada Gambar II.15 berikut ini.



1. Ilustrasi format *Trace* eksekusi kode program

Pada Gambar II.15 tersebut mengilustrasikan posisi eksekusi kode program pada saat masih kosong (*empty state*). *Trace* eksekusi kode program disimpan dalam bentuk *JSON* yang didefinisikan dalam dua bagian besar, yaitu:

1. “***code***” berbentuk string, yang berisi kode program dari pengguna.
2. “***trace***” berbentuk objek-objek larik, yang setiap objek merepresentasikan posisi eksekusi kode program sebagai berikut:

|  |  |  |
| --- | --- | --- |
| 1 | ordered\_globals: [], | // sebuah larik yang merepresen-tasikan urutan visual untuk atribut global |
| 2 | stdout: “”, | // total standar output dari kode program pada posisi eksekusi saat ini |
| 3 | func\_name: “<module>”, | // fungsi saat ini yang sedang diekseskusi |
| 4 | stack\_to\_render: [], | // merupakan *list* dari objek-objek, setiap objek direpresentasikan sebagai sebuah *stack frame* |
| 5 | globals: {}, | // sebuah *dictionary global* ‘*stack frame*’ |
| 6 | heap: {}, | // sebuah *dictionary* objek-objek *heap* |
| 7 | line: 1, | // indikasi baris kode program yang sedang dieksekusi |
| 8 | event: “step\_line” | // sebuah parameter *event* yang dapat berisi: *user\_call*, *user\_return*, *user\_exception* atau *user\_line* |

#### II.5.5.4 Fitur *Library* *Data-Driven Documents* (D3)

Teknologi visualisasi utama yang digunakan oleh OPT adalah *Data-Driven Documents* (D3) (Bostock dkk., 2011) yang merupakan salah satu *library* terpopuler untuk implementasi visualisasi berbasis web. Format *trace* eksekusi program dibaca oleh *D3.js* untuk mendeklarasikan pemetaan data dan atribut yang diperlukan oleh elemen *Document Object Model* (DOM) pada laman *Hyper Text Markup Language* (HTML). Peran *D3.js* dapat dilihat pada Gambar II.16 berikut ini.



1. Peran Utama *D3.js* sebagai pendukung Visualisasi

*Library* *D3JS* sangat fleksibel terhadap abstraksi data tingkat tinggi. *Library* ini juga mampu memvisualisasikan berbagai macam data, bahkan cocok untuk visualisasi dengan tingkat kerumitan yang tinggi.

## II.5 Kesimpulan Awal Berdasarkan Tinjauan Pustaka dan Eksplorasi

Berdasarkan tinjauan pustaka dan eksplorasi yang telah dilakukan, sebagian besar kakas VP dibuat untuk komputer *desktop* atau *Java Virtual Machine* (JVM). Aplikasi *Java Applet* dapat beroperasi di web, namun konfigurasi *Java* pada *browser* harus sesuai dengan yang dibutuhkan. Ini tentu berbeda dengan aplikasi web *HTML*, *CSS*, dan *JavaScript* yang dapat langsung digunakan di *browser*. Pengguna tidak perlu melakukan konfigurasi *Java* untuk mengaksesnya.

1. Ringkasan hasil eksplorasi kakas VP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Swan** | **VisMod** | **jGRASP** | **Jype** | **OPT** |
| **Tahun** | 1996 | 1999 | 2004 | 2009 | 2010 |
| **Metode Visualisasi** | Anotasi | *Pointer* dan *reference* | *Pointer*, *reference*, Identifikasi nama variabel dan struktur kelas | *Matrix Framework* | Eksekusi *trace JSON* |
| **Fitur** | *Custom* visualisasi, *pan*, dan *zoom* | Pengecekan kesalahan sintaks, *pretty printer* | *Extensible framework* | *Automatic assessment tool* | *Embeddable*, kolaborasi, *chat* |
| ***Platform*** | Komputer *desktop* | Komputer *desktop* | JVM | JVM, web | web |
| **Eksekusi Kode Program** | C/C++ | Modula-2 | Java | Python | Python, C/C++, JavaScript, Typescript, Java, Ruby |
| **Visual Struktur Data** | Graf, pohon, *list*, dan *array* | *List*, pohon | Tumpukan, antrian, *Binary Tree*, *linked list*, *hash table* | *Binary search tree* | - |

Pada Tabel II.1 menunjukkan data perbandingan kakas VP yang telah dieksplor. Hasil eksplorasi diperoleh bahwa tidak ada satu pun kakas VP berbasis web dengan bahasa C atau C++ yang dapat menampilkan visualisasi graf. Sehingga ini menjadi peluang untuk penelitian lebih lanjut dalam pengembangannya. Terutama penggunaan *Typescript* sebagai dasar pengembangan kakas yang masih baru dan memiliki prospek ke depannya dalam teknologi web terkini.

OPT dipilih menjadi dasar pengembangan kakas, karena selain telah mendukung basis web, kakas ini juga bersifat bebas dan bersumber kode terbuka. Kakas ini juga masih dalam proses pengembangan dan penelitian oleh pengembangnya[[3]](#footnote-3). Jadi, masih terdapat banyak peluang dan celah untuk diteliti lebih lanjut agar dapat dimanfaatkan dengan baik. Mengingat teknologi web dan aplikasi piranti bergerak terus berkembang pesat hingga dekade tahun terakhir ini.

Untuk membangun sebuah kakas VP, ada beberapa yang perlu dipertimbangkan. Salah satu aspek tersulit dari VP adalah memilah data mana yang dapat direpresentasikan dengan baik dan benar. Secara umum, ini lebih sulit dari pada animasi algoritma. Aspek perilaku suatu kode program juga harus diperhatikan untuk visualisasi. Pilihan representasi visualisasi untuk lingkungan program juga harus dipertimbangkan. Selain itu, dampak dari visualisasi terhadap perilaku program juga harus dipertimbangkan, karena bisa terjadi salah paham terhadap suatu proses eksekusi kode program. Keterbatasan ruang layar komputer juga sering menjadi permasalahan yang perlu diperhatikan dalam desain visualisasi.

# Bab III Analisis Masalah

Bab ini menganalisis permasalahan kakas VP yang akan dikembangkan. Beberapa masalah telah dirumuskan pada subbab I.2. Berangkat dari kesulitan yang dialami oleh peserta didik dalam memahami eksekusi graf kode program, visualisasi menjadi salah satu solusi efektif yang diusulkan.

## III.1 Analisis Tujuan Visualisasi Graf

Visualisasi[[4]](#footnote-4) adalah representasi dari sebuah objek, situasi, atau seperangkat informasi berupa grafik atau gambar. Merujuk pada pengertian tersebut, dapat dikatakan visualisasi merupakan perwakilan berupa grafik atau gambar dari sebuah objek. Visualisasi diusulkan sebagai salah satu metode untuk menyampaikan substansi dari seperangkat informasi. Karena manusia menerima dan memahami informasi lebih dominan melalui indera visual dibanding indera lainnya (Ware, 2004).

Informasi yang ingin disampaikan adalah proses eksekusi graf kode program. Kode program memiliki dua bentuk, yaitu statis dan dinamis (Sorva, 2012). Bentuk statis berarti tertulis jelas pada kode program, sedangkan bentuk dinamis masih implisit dalam kode program. Contohnya, variabel X tertulis di kode program memiliki tipe data *integer* (disebut bentuk statis), sedangkan ketika terjadi perubahan nilai pada variabel X selama proses eksekusi kode program disebut bentuk dinamis. Pada kasus eksekusi graf kode program terdapat proses operasi struktur data, seperti pendefinisian tipe data abstrak, pengaturan, dan penyimpanan data di memori komputer yang tidak dapat dilihat oleh indera visual manusia. Oleh karena itu, diharapkan melalui visualisasi dapat digunakan untuk membantu memahami proses eksekusi graf kode program.

Makna ‘memahami’ eksekusi graf kode program memiliki persepsi berbeda antar setiap orang. Untuk menghindari perbedaan persepsi tersebut, perlu dijelaskan definisi ‘memahami’ dalam penelitian tesis ini. Memahami merupakan salah satu tingkatan dalam proses kognitif manusia untuk memperoleh pengetahuan. Terkait proses kognitif, ada sebuah teori yang dapat menjelaskan tingkatan proses kognitif manusia, yaitu taksonomi Bloom (Sorva, 2012).

Pada taksonomi Bloom terdapat tingkatan kedua yang disebut sebagai ‘memahami’ dari enam tingkatan. Taksonomi ini diusulkan oleh Anderson pada tahun 1956 yang kemudian direvisi pada tahun 2001 dengan perubahan beberapa istilah dari tingkatan tersebut (Sorva, 2012). ‘Memahami’ (Thompson dkk., 2008) didefinisikan sebagai kemampuan dalam menyusun arti dari materi pelajaran untuk dikomunikasikan melalui oral, tulisan, dan gambar. Taksonomi yang telah direvisi, menyebutkan definisi ‘memahami’ (Thompson dkk., 2008) adalah kemampuan dalam menginterpretasi, memberikan contoh, mengklasifikasi, meringkas, menyimpulkan, membandingkan, dan menjelaskan. Merujuk pada definisi tersebut, yang dimaksud dengan ‘memahami’ eksekusi graf kode program adalah:

1. Mampu menelusuri dan menjelaskan alur eksekusi graf kode program;
2. Mampu memperbaiki kesalahan graf kode program;
3. Mampu menjelaskan algoritma graf yang digunakan dalam kode program;
4. Mampu menjelaskan setiap prosedur, operasi, langkah-langkah, dan fungsi yang terdapat di dalam graf kode program.

Tujuan visualisasi adalah untuk menyampaikan informasi dengan efisien dan efektif. Efisien berarti informasi dapat dipahami dengan mudah dan cepat oleh penerimanya. Efektif berarti makna yang terkandung di dalam informasi dapat dipersepsi dengan benar, sehingga tujuan dari penyampaian informasi tersebut dapat tercapai.

Keefektifan menjadi dasar pengukuran untuk mencapai tujuan visualisasi (Urquiza-Fuentes dan Velázquez-Iturbide, 2009). Selain itu, *usability* juga merupakan aspek penting yang perlu dinilai (Urquiza-Fuentes and Velázquez-Iturbide, 2009). Karena keefektifan sebuah visualisasi sangat didukung dengan *usability*. Metode pengukurannya menggunakan teknik evaluasi yang akan dijelaskan pada Bab V.

## III.2 Analisis Desain Visualisasi Graf

Tahap ini menganalisis desain visualisasi graf yang dibutuhkan sehingga tujuan visualisasi dapat tercapai. Tujuan yang ingin dicapai adalah pengguna dapat memahami eksekusi graf kode program. Untuk mendukung tujuan tersebut, diasumsikan pengguna telah:

1. memahami teori graf;
2. memiliki dasar pemrograman struktur data dengan bahasa C/C++; dan
3. biasa menggunakan aplikasi web.

Berdasarkan studi literatur dan eksplorasi kakas yang telah dilakukan, diperoleh beberapa spesifikasi kebutuhan pengguna untuk membangun visualisasi graf, yaitu:

1. Visualisasi eksekusi graf kode program dapat menampilkan graf;
2. Visualisasi untuk operasi tambah dan hapus *node* dapat ditampilkan sesuai eksekusi graf kode program;
3. Animasi pencarian *node* dapat divisualisasi sesuai dari eksekusi kode program.

Seperti yang telah dijelaskan pada subbab sebelumnya bahwa *usability* memiliki peran utama untuk mencapai keefektifan visualisasi. Ada sepuluh aturan *usability* yang didefinisikan oleh Jakob Nielsen (Nielsen, n.d.; Preece dkk., 2015) dapat menjadi dasar pengukuran efektifitas sebuah desain kakas visualisasi graf, yaitu:

1. Status visibilitas sistem. Aturan ini sesuai yang telah dijelaskan pada Gambar II.2. Sistem harus memberikan status yang sesuai dan tepat waktu kepada pengguna. Contohnya, ketika sistem sedang proses eksekusi data, tampilkan informasi berupa *progress bar* atau estimasi waktu, sehingga pengguna mengetahui kapan proses akan selesai.
2. Sistem harus menyesuaikan dengan keadaan seperti di dunia nyata. Contohnya, ketika merancang antarmuka web untuk anak-anak, gunakan istilah yang dikenal atau akrab oleh anak, format informasi mudah untuk dikenali.
3. Kebebasan berinteraksi dan kemudahan dalam kontrol pengguna. Pengguna mudah mengenali komponen kontrol untuk berinteraksi dengan sistem. Contohnya, menyediakan fungsi *copy*, *paste*, dan keluar dari sistem.
4. Memiliki standar dan konsistensi. Kontrol pengguna, ikon, penggunaan istilah, pesan *error* harus konsisten dan memiliki standar umum.
5. Penanganan kesalahan pengguna yang dapat menyebabkan *error*. Memberikan informasi kepada pengguna ketika berada pada area kontrol yang sensitif pada sistem. Contohnya, memberikan pesan peringatan kepada pengguna bahwa data dapat terhapus dengan melakukan aksi tersebut.
6. Komponen mudah dikenali. Kurangi beban berpikir pengguna ketika berinteraksi dengan sistem, misalnya dengan menampilkan ikon yang biasa dikenal, aksi, dan pilihan dalam sistem.
7. Fleksibel dan efisien dalam penggunaan. Bagi pengguna pemula dan yang sudah mahir mempunyai kebiasaan berbeda dalam menggunakan suatu sistem. Sistem harus mudah digunakan dan efisien bagi semua tingkatan pengguna. Bagi pengguna mahir, dapat disediakan fitur khusus untuk mempercepat proses pekerjaan dengan sistem. Contohnya, menyediakan *shortcut* seperti kombinasi kunci *Windows* + E untuk membuka aplikasi *Explorer*.
8. Desain estetis dan minimalis. Hindari menampilkan informasi dan elemen yang berlebihan, sehingga antarmuka menjadi tidak relevan bagi pengguna. Contohnya, warna latar hampir sama dengan warna teks yang membuat sulit untuk dibaca.
9. Beri kemudahan kepada pengguna untuk mengenali, mendiagnosa, dan pemulihan jika terjadi error atau kesalahan sistem. Tampilkan pesan *error* kepada pengguna dengan instruksi yang jelas untuk menangani kesalahan, dari pada menggunakan kode yang sulit untuk dipahami oleh pengguna.
10. Menyediakan dokumentasi dan bantuan alternatif kepada pengguna. Desain antarmuka terbaik adalah yang simpel digunakan tanpa membutuhkan dokumenasi dan bantuan alternatif. Namun, keadaan tertentu terkadang membutuhkan bantuan dan dokumentasi tersebut. Dokumentasi dan bantuan harus mudah ditemukan, serta instruksinya mudah untuk diikuti oleh semua pengguna.

Dari sepuluh aturan *usability* tersebut, maka dapat disusun spesifikasi kebutuhan untuk rancangan antarmuka pengguna adalah sebagai berikut:

1. Menampilkan status proses sistem yang sedang berjalan, seperti saat proses kompilasi kode program, proses ekstraksi data, dan proses animasi graf.
2. Menggunakan istilah dan bahasa yang mudah dipahami oleh pengguna, misalnya pada navigasi kontrol, menu, editor kode, dan proses yang terjadi di dalam sistem.
3. Meletakkan navigasi kontrol dan *slider* yang mudah terlihat untuk kontrol visualisasi.
4. Pesan *error* karena kesalahan kode program harus diposisikan sedekat mungkin dengan kode program yang terindikasi kesalahan dan memberikan instruksi yang jelas untuk diperbaiki oleh pengguna. Kesalahan sistem yang terjadi bukan karena aktivitas pengguna, harus disampaikan dengan jelas dan simpel serta memberikan solusi sederhana yang dapat dilakukan oleh pengguna.
5. Menampilkan animasi pada visualisasi graf yang menandakan terjadinya proses penambahan node, penghapusan, atau perubahan nilai variabel.

Sebuah graf digambarkan dengan *node* dan *edge*. *Node* dan *edge* memiliki atribut masing-masing. Atribut *node* terdiri dari bentuk, warna, ukuran, dan label. Sedangkan atribut *edge* terdiri dari ketebalan garis, label, warna, dan anak panah (untuk graf berarah). Gambar dasar visual graf dengan *node* dan *edge* dapat dilihat pada Gambar III.1 berikut ini.



1. Visual graf berarah dengan dua *node* dan *edge*

Visualisasi graf memiliki beberapa properti yang dibutuhkan. Pada Tabel III.1 menunjukkan properti visual graf yang dapat direpresentasikan. Dari daftar itu tidak semua properti visual digunakan, harus dapat menyesuaikan data yang sedang diproses dari eksekusi kode program.

1. Daftar properti visual graf

| **No** | **Nama Properti** | **Visualisasi** |
| --- | --- | --- |
| 1 | Label untuk *node* | x |
| 2 | Bentuk *node* |  |
| 3 | Bentuk *edge* | atau |
| 4 | Label *edge* | x  x  atau |
| 5 | Warna *node* dan *edge* | atau  atau |
| 6 | Ketebalan *edge* | atau |

Pada Gambar III.2 menunjukkan representasi graf yang dilakukan oleh kakas OPT berupa tabel data larik atau matriks. Gambar tersebut merupakan salah satu contoh graf kode program dengan algoritma *Breadth First Search*. Representasi berupa data matriks dapat divisualisasi menjadi bentuk graf.

Selain itu, pada Gambar III.3 menunjukkan bahwa *pointer* direpresentasikan dengan garis panah untuk relasi antar simpul. Representasi tersebut menjadi lebih rumit ketika data semakin bertambah dalam visualisasi, sehingga membuat pengguna kesulitan dalam menelusurinya. Desain alternatif yang dapat dilakukan adalah menyembunyikan data yang tidak diperlukan. Jika data berisi *node* dan *edge*, maka dapat divisualisasi menjadi graf.

Perubahan visual antar setiap langkah pada baris kode program harus memberikan informasi berdasarkan data yang berubah, bertambah, atau berkurang. Proses ini akan memudahkan pengguna dalam memahami informasi yang tampil pada visualisasi tersebut.



1. OPT: Visualisasi graf dengan *array* dimensi-2 (matriks)



1. OPT: Visualisasi graf dengan *struct* dan *pointer*

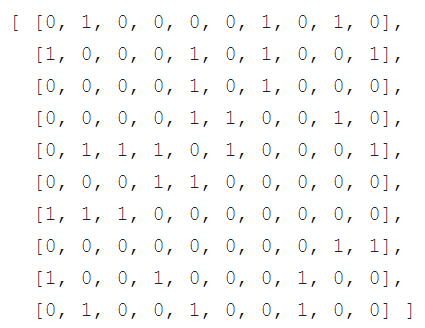
## III.3 Analisis Deteksi Graf dalam Kode Program

Permasalahan lain dalam melakukan visualisasi graf adalah mendeteksi adanya sumber data. Sebelum menganalisis metode yang dapat digunakan, perlu diketahui terlebih dahulu jenis representasi data graf yang ada dalam kode program yang akan dijelaskan pada subbab III.3.1 berikut ini.

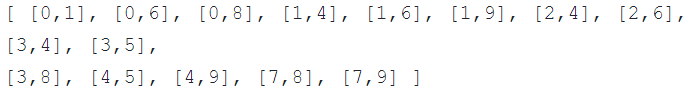
### III.3.1 Representasi Data Graf dalam Kode Program

Untuk menganalisis teknik pendeteksian graf kode program, perlu diketahui terlebih dahulu cara representasi data graf dalam kode program sehingga sumber data untuk visualisasi dapat ditentukan. Representasi data graf dalam kode program dapat dilakukan dengan tiga cara, yaitu sebagai berikut (Sedgewick dan Wayne, 2011):

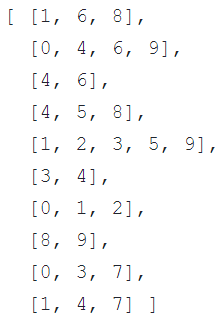
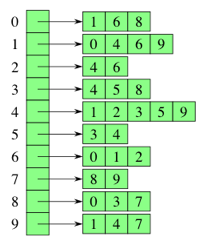
1. Representasi *adjacency matrix* atau berupa data matriks yang simetris dengan 2-dimensi *array* (lihat Gambar III.4), yaitu baris dan kolom sebagai *node*. Baris dan kolom yang ada *edge*, diberi nilai 1 atau *true*, jika kosong maka diberi nilai 0 atau *false*. Jika matriks tidak simetris, maka direpresentasikan sebagai graf berarah begitu sebaliknya. Akan tetapi representasi ini tidak efektif karena akan membutuhkan ruang sebesar ;



1. Contoh data matriks sebagai representasi graf
2. Representasi *array of edges*, yaitu menggunakan *edge class* dengan dua variabel bertipe *int* (lihat Gambar III.5). Namun *class* tersebut dalam implementasinya akan melibatkan semua data *edge* yang ada dalam graf. Representasi ini juga tidak efektif, karena proses pencarian akan melibatkan semua data *edge*;



1. Contoh data *array* atau *edge list*
2. Representasi *array of adjacency lists*, ini menggunakan indeks *node* yang menyimpan *list* dari simpul yang saling berdekatan atau bertetangga. Representasi ini (lihat Gambar III.6) cukup efektif dibanding kedua jenis representasi di atas. Adanya indeks *node* memudahkan penelusuran *edge*, karena untuk mengakses *node* yang berdekatan cukup dari indeks *array* tersebut.



1. Contoh data *adjacency list* dengan indeks *node*

Secara umum representasi data graf menggunakan *adjacency list* lebih disarankan, karena hal ini lebih efektif untuk graf yang kepadatannya rendah, yaitu nilai kedekatan antar *node* berbeda jauh. Untuk *adjacency matrix* lebih disarankan ketika graf yang dibentuk memiliki kepadatan yang tinggi.

### III.3.2 Analisis Kakas Pendukung untuk Deteksi Data Graf

Setelah mengetahui ketiga cara representasi data graf, proses selanjutnya adalah mencari ciri data yang cocok dengan ketiga representasi tersebut. OPT memanfaatkan teknik *tracking* *memory* untuk membaca data fisik di dalam memori komputer. OPT menggunakan kakas *Valgrind* untuk menghasilkan *trace* eksekusi dari kode program C/C++. Sebenarnya kakas *Valgrind* digunakan untuk mencari celah atau kebocoran (*leak memory debugger*) data memori dalam kode program. Namun, beberapa fungsi *Valgrind* dimodifikasi agar dapat menghasilkan *trace* eksekusi yang dimanfaatkan untuk visualisasi.

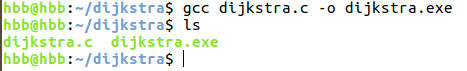
Selain kakas *Valgrind*, ada beberapa kakas sejenis yang memiliki fungsi sama. Contohnya (Bruening dan Zhao, 2011) seperti *Dr. memory*, *Purify* (komersial), *Intel Parallel Inspector* (komersial), *BoundsChecker* (komersial), *LeakTracer*, *mprof*, *Third Degree* (*Alpha platform*), dan *Insure*++ (komersial). Namun, *Valgrind* memiliki beberapa keunggulan dibanding kakas lainnya. *Memcheck* adalah salah satu sub kakas yang terdapat di dalam *Valgrind*. *Memcheck* adalah kakas yang menggunakan metode analisis dinamis dengan empat teknik untuk melakukan pengecekan kesalahan memori (Bruening dan Zhao, 2011; *www.valgrind.org*). Empat teknik itu adalah sebagai berikut:

1. *Memcheck* dapat melacak posisi alamat memori pada setiap *bit*. Teknik ini bermanfaat untuk mendeteksi semua alamat memori termasuk yang tidak dapat diakses.
2. *Memcheck* dapat melacak semua blok *heap* yang dialokasikan dengan fungsi *malloc()*, objek *new*, dan objek *array*. Teknik ini bermanfaat untuk mendeteksi blok *heap* yang rusak atau tidak terisi dengan nilai, dan dapat mendeteksi kebocoran memori saat terminasi program.
3. *Memcheck* dapat mengecek memori yang *overlap*, seperti penggunaan fungsi *strcpy()* dan *memcpy()*.
4. *Memcheck* dapat melakukan pengecekan secara detail pada setiap *bit* data dengan menelusurinya di register *CPU* dan memori.

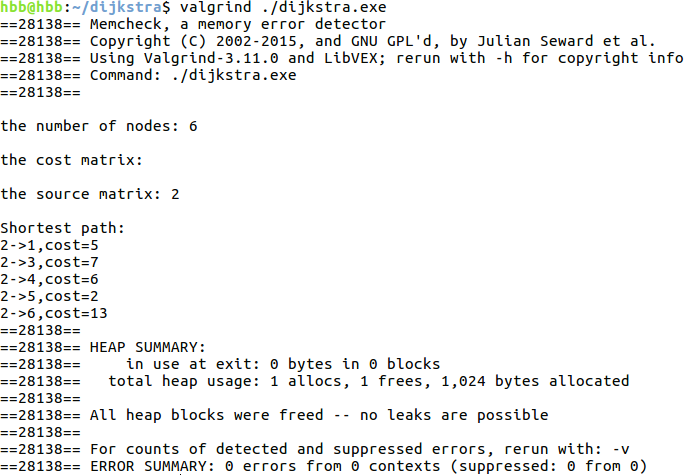
*Memcheck* adalah kakas pertama yang melakukan pelacakan memori secara detail sampai pada tingkat *bit*. Kakas lain melakukan pelacakan memori hanya sampai pada tingkat *byte* (kakas *Purify*) dan tingkat *word* (kakas *Third Degree*) (Bruening dan Zhao, 2011). Hal ini disimpulkan bahwa *Memcheck* dapat menangani kode program dengan baik sampai tingkat *bit*, dibandingkan kakas lain yang hanya mampu melacak alamat memori sampai tingkat *byte* atau *word*. Dampaknya adalah ketika melakukan pengecekan kesalahan memori, kakas lain memiliki peluang kesalahan lebih besar dalam pelaporannya dibandingkan dengan kakas *Valgrind* (*Memcheck*).

Contoh penggunaan *Memcheck* dari *Valgrind* sebagai berikut:

1. Sebelum menggunakan *Valgrind*, kode program di *compile* terlebih dahulu, contohnya pada Gambar III.7 berikut ini:



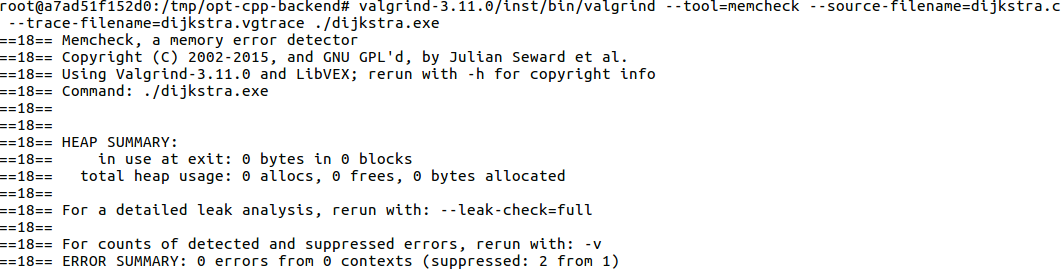
1. Hasil *compile* kode program *C* dengan *GCC*
2. Kemudian hasil *compile* berupa file eksekusi tersebut dijadikan sebagai *input* untuk *Valgrind*, contohnya pada Gambar III.8 berikut ini:



1. Hasil keluaran dari *Valgrind* 3.11 versi asli

Pada Gambar III.8 menunjukkan tidak ada kesalahan atau kebocoran penggunaan memori dalam kode program. Angka 28138 adalah *process ID* yang sedang digunakan dalam proses eksekusi *valgrind*.

Beberapa fungsi di dalam *Valgrind* dimodifikasi agar dapat mendukung visualisasi, seperti yang telah dijelaskan pada subbab II.4.5.2. Hasil keluaran dari *Valgrind* versi modifikasi dapat dilihat pada Gambar III.9. Tidak berbeda jauh dengan hasil keluaran dari *Valgrind* versi asli pada Gambar III.8.

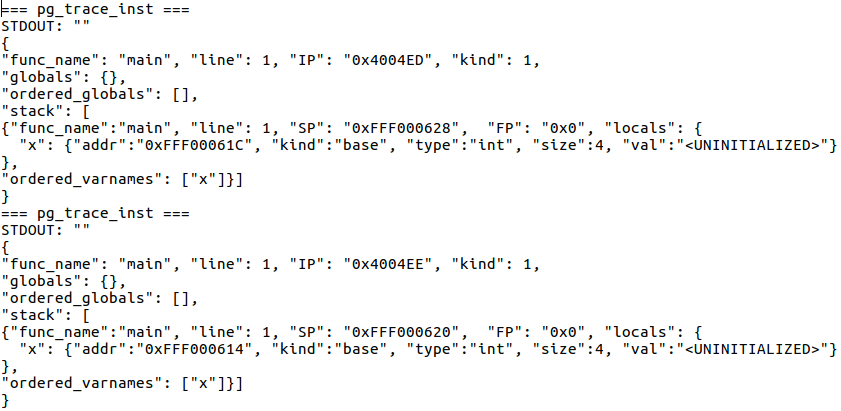


1. Hasil keluaran dari *Valgrind* versi modifikasi

Namun, pada Gambar III.9 menunjukkan ada perbedaan perintah, yaitu:

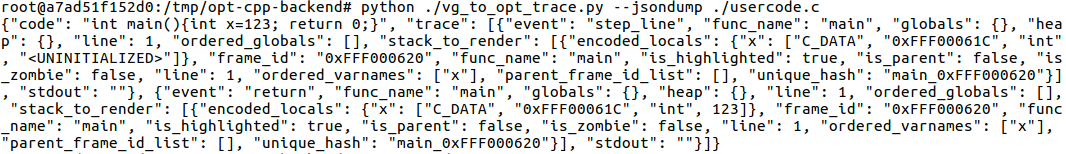
--trace-filename=dijkstra.vgtrace

Perintah ini berfungsi untuk menghasilkan *valgrind trace* yang disimpan ke berkas *dijkstra.vgtrace*. Ilustrasi isi berkas *vgtrace* dapat dilihat pada Gambar III.10.



1. Isi berkas *vgtrace*

Dari format *vgtrace* diubah ke format *trace* eksekusi OPT seperti yang telah dibahas pada subbab II.4.5.3. Format *vgtrace* tidak jauh berbeda dengan format *trace* eksekusi *JSON* yang ada di subbab II.4.5.3, hanya dibagi dua atribut utama yaitu *code* dan *trace*. Hasil data *trace* eksekusi *JSON* dapat dilihat pada Gambar III.11 berikut ini.



1. Hasil keluaran berupa *JSON* dari *Valgrind* versi modifikasi

### III.3.3 Analisis Teknik Deteksi Graf

Pada tahap proses *compile* dan menghasilkan *vgtrace* tidak ada intervensi yang dilakukan. Namun, keluaran berupa *JSON* yang dikirim ke *browser* pengguna, dilakukan ekstraksi untuk menyaring data representasi graf di dalamnya. Proses untuk menghasilkan data *JSON* secara lengkap dapat dilihat pada Gambar III.12. *Server* meng-*compile* kode program dari pengguna dengan kakas *Valgrind*. Pada bagian label “Ekstraksi data graf” terdapat proses untuk menyaring data graf. Jika ada yang cocok, maka proses ekstraksi sumber data graf dilakukan.



Ekstraksi

data graf

1. OPT: diagram alir proses perolehan data *JSON*



Ekstraksi

data graf

1. OPT: Format data eksekusi *trace JSON*

Sumber data yang digunakan untuk visualisasi adalah data *trace* eksekusi *JSON*. Salah satu contoh data *JSON* berupa matriks yang dihasilkan dari Gambar III.13 dapat dilihat pada Tabel III.2 berikut ini. Data ini adalah contoh variabel *‘a’* berupa matriks yang memiliki dimensi 7 × 7. Nilai berupa "C\_MULTIDIMENSIONAL\_ARRAY" adalah jenis data *n-array*; nilai baris kedua "0x601100" adalah posisi alamat memori yang digunakan; kemudian ukuran dimensi yang digunakan berada di posisi ketiga [7, 7]. Sisa baris berikutnya adalah isi data setiap elemen di dalam variabel matriks tersebut. Data ini berada di atribut globals (lihat Gambar III.13).

1. Variabel *‘a’* berupa matriks berdimensi 7 x 7

|  |
| --- |
| "a": [  "C\_MULTIDIMENSIONAL\_ARRAY",  "0x601100", [  7,  7  ],  [  "C\_DATA",  "0x601100",  "int",  0  ],  [  "C\_DATA",  "0x601104",  "int",  0  ],  [...]  ] |

Pada Gambar III.14 menunjukkan contoh data objek *heap* yang berisi alamat memori *pointer* dan *struct*. Proses pengolahan data *JSON* menggunakan bantuan kakas web yang dapat diakses melalui alamat http://www.json2table.com. Kakas ini membantu mengolah dan mencari pola visualisasi dalam data *JSON* dengan mengonversi ke bentuk tabel dan daftar poin.



1. Contoh data *JSON pointer* pada atribut *heap*

Pada Tabel III.3 berikut ini menunjukkan contoh isi alamat memori pada objek *heap*. Data ini akan dilakukan ekstraksi sesuai pada subbab II.2 tentang tujuh tahapan konstruksi visualisasi data.

1. Contoh isi alamat memori pada objek *heap*

|  |
| --- |
| "0x5402210": [  "C\_ARRAY",  "0x5402210", [  "C\_STRUCT",  "0x5402210",  "node", [  "vertex", [  "C\_DATA",  "0x5402210",  "int",  2  ]  ],  [  "next", [  "C\_DATA",  "0x5402218",  "pointer",  "0x5402170"  ]  ]  ]  ] |

Untuk dapat melakukan visualisasi secara otomatis, diperlukan fitur dari data-data tersebut. Ada beberapa fitur yang bisa menjadi acuan, yaitu :

1. "C\_MULTIDIMENSIONAL\_ARRAY" digunakan untuk menemukan variabel data matrik yang menyimpan nilai-nilai simpul dan relasinya serta bobot suatu graf (akan dibahas lebih lanjut pada Bab IV);
2. "C\_STRUCT" digunakan untuk menemukan variable *struct* yang masih harus dilanjutkan dengan pengecekan fitur ketiga berikut;
3. "pointer" di dalam "C\_DATA" yang mengindikasikan adanya relasi antar simpul dengan menggunakan *pointer* di dalam *struct*.

Secara garis besar, data *JSON* tersebut dapat dibagi menjadi dua kategori utama untuk representasi graf kode program, yaitu (1) bersumber dari tabel informasi berupa matriks atau *n-*dimensi *array*, dan (2) berupa *struct* yang berisi *pointer*.

### III.3.4 Analisis Kakas Pendukung untuk Visualisasi Graf

D3.js (Bostock dkk., 2011) adalah *library JavaScript* untuk visualisasi data dengan teknik memanipulasi *Documen Object Model* (DOM). DOM adalah standar model objek untuk HTML dan XML yang bersifat *platform* independen. Sebuah *web browser* tidak harus menggunakan DOM untuk menampilkan dokumen HTML. Namun DOM diperlukan oleh *JavaScript* yang akan mengubah tampilan sebuah *website* secara dinamis. Dengan kata lain, DOM adalah cara *JavaScript* melihat suatu halaman HTML. D3.js memanipulasi langsung terhadap HTML dan CSS untuk melakukan visualisasi data sehingga memiliki kontrol lebih bebas untuk menentukan posisi sebuah data, teks, dan komponen visualisasi lain.

Sigma.js dan Vis.js juga memanfaatkan DOM untuk membuat visualisasi. *Library* selain D3.js ini hanya fokus ke pengembangan algoritma untuk visualisasi graf. Dagre adalah *library* visualisasi graf lain yang dibuat berbasis D3.js yang hanya fokus pada algoritma *layout* graf, untuk *rendering* Dagre menggunakan *library* lain, yaitu dagre-D3, JointJS, atau Cytoscape.js. Springy juga hanya fokus pada algoritma *layout* graf, namun tidak memberikan kontrol leluasa untuk membuat visualisasi graf. Ngraph salah satu *library* visualisasi graf yang dapat digunakan di *browser* dan *server*. Untuk *rendering* ngraph menggunakan *library* VivaGraph. Vivagraph didesain untuk me-*render* graf dari ngraph.

*Library* lain untuk visualisasi graf yang dibangun dengan fitur web workers dan jQuery adalah Arbor. Library ini menjanjikan proses render visualisasi graf yang efisien dan cepat tanpa harus memuat ulang halaman web ketika ada perubahan data. Namun, Arbor sudah tidak dimutakhirkan lagi sejak tujuh tahun yang lalu.

Dracula adalah salah satu *library* visualisasi interaktif graf berbasis SVG yang memiliki berbagai jenis *layout* algoritma graf, seperti *Bellman Ford*, *Binary Heap Map*, *Dijkstra*, *Floyd Warshall*, dan *Topological Sort*. *Library* lain bernama JSNetworkX memiliki fitur untuk memproses dan menganalisis graf. *Library* ini merupakan versi JavaScript dari *library* Python NetworkX. JSNetworkX ini dapat digunakan bersama dengan D3.js di *browser* untuk membuat visualisasi graf. Sedangkan Alchemy.js adalah library visualisaasi graf yang dibangun berdasarkan library dari D3.js. Semua perkembangan *library* JavaScript untuk visualisasi graf dapat dilihat pada Tabel III.4 yang diurut berdasarkan jumlah pengguna favorit dari situs GitHub.com.

1. Daftar[[5]](#footnote-5) *library* *JavaScript* untuk visualisasi graf

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Nama *Library*** | **Jumlah Pengguna Favorit\*** | **Lisensi** |
| 1 | D3.js | 75.874 | BSD 3 |
| 2 | Sigma.js | 7.936 | MIT |
| 3 | Vis.js | 6.636 | Apache 2.0 |
| 4 | Cytoscape.js | 4.288 | LGPL |
| 5 | VivaGraphJS | 2.597 | BSD 3 |
| 6 | Arbor | 2.474 | MIT |
| 7 | Dagre | 1.608 | MIT |
| 8 | Springy | 1.565 | MIT |
| 9 | Ngraph | 753 | MIT |
| 10 | Dracula | 641 | MIT |
| 11 | JSNetworkX | 477 | BSD 3 |
| 12 | Alchemy | 419 | AGPLv3 |

\*Berdasarkan jumlah *stars* dari GitHub.com (diakses tanggal 20 Mei 2018)

D3.js dipilih sebagai kakas pendukung visualisasi graf, karena memiliki banyak fitur dan fungsi yang disediakan. Terdapat 31 modul yang dapat digunakan untuk keperluan visualisasi, seperti pewarnaan, bentuk grafik, *layout* hirarki, animasi, transisi, dan sebagainya. Kontrol untuk memanipulasi visualisasi data dengan komponen SVG, HTML, dan CSS dapat dilakukan dengan lebih detail. D3.js versi 3 dipilih pada pengembangan kakas ini dibanding versi 4 atau versi 5 yang terbaru, karena pada versi 3 lebih banyak sumber tutorial, buku referensi, dan contoh visualisasi yang tersedia. D3.js juga sudah digunakan sejak awal pengembangan kakas OPT sehingga tidak akan banyak mengubah struktur kode program kakas.

## III.4 Analisis Kebutuhan Perangkat

Untuk menunjang pengembangan kakas diperlukan seperangkat kebutuhan yang harus dipenuhi, baik dari sisi perangkat lunak yang mutakhir maupun perangkat kerasnya. Subbab berikut akan merinci spesifikasi kebutuhan sistem.

### III.4.1 Kebutuhan Perangkat Lunak

*TypeScript* merupakan bahasa pemrograman berbasis objek dari *JavaScript* yang cocok untuk pengembangan kakas OPT. Bahasa ini dibuat oleh Microsoft pada tahun 2012 untuk menunjang pengembangan aplikasi web *JavaScript* berskala besar dengan konsep pemrograman berorientasi objek. *TypeScript* masih membutuhkan *library* pendukung eksternal untuk keperluan pengembangan yang cukup kompleks. Rincian kakas dan *library* pendukung untuk pengembangan kakas OPT yang telah terintegrasi dapat dilihat pada Tabel III.5 berikut ini.

1. Kakas dan *library* pendukung pengembangan OPT

| **No.** | **Nama** | **Versi** | **Keterangan** |
| --- | --- | --- | --- |
| 1 | NodeJS | 4.5.0 | *JavaScript runtime* |
| 2 | Npm | 2.15.9 | *Node package manager*, memanajemen *library* dari *node* |
| 3 | Webpack | 3.5.5 | *Bundle js, ts, css, image* |
| 4 | Typescript | 1.8.10 | Bahasa pemrograman utama |
| 5 | Typings | 2.1.1 | *Typescript definitions* |
| 6 | Ace Code Editor | 1.2.8 | *Text editor* pada aplikasi web untuk *input* kode program |
| 7 | D3.js | 2.0 | *Library* *Data-Driven Documents* |
| 8 | jQuery | 3.0.0 | *Cross-platform JavaScript library* |
| 9 | jQuery.bbq | 1.3pre | *Back Button and Query Library* |
| 10 | jQueryUI | 1.11.4 | *JavaScript Library* untuk interaksi antarmuka pengguna |
| 11 | jQuery.qtip | 2.0.0 | *jQuery tooltip plugin* |
| 12 | RequireJS | 2.1.20 | *JavaScript module loader*. *Framework* untuk manajemen dependensi. |
| 13 | jQuery.jsPlumb | 1.3.10 | *Library* visual *connectivity* untukaplikasi web |
| 14 | jQuery.simplemodal | 1.4.4 | *Lightweight jQuery plugin* untuk antarmuka *modal* dialog |

### III.4.2 Kebutuhan Perangkat Keras

OPT dapat berjalan dengan baik pada komputer dengan spesifikasi yang direkomendasikan sebagai berikut:

* *Processor core i3*
* *RAM 2GB*
* Sistem operasi Ubuntu 16.04.

Kebutuhan perangkat itu untuk proses pengembangan kakas, namun untuk produksi versi publik yang dipasang pada komputer *server*, dibutuhkan spesifikasi *RAM* sekitar 4GB – 8GB. Hal ini karena sistem *sandbox docker engine* membutuhkan ruang memori lebih dan waktu hampir satu menit (60 detik) jika memproses sekitar 500 baris kode program untuk dapat menghasilkan data *trace* eksekusi *JSON*.

# Bab IV Perancangan dan Implementasi Kakas

Bab ini menjelaskan perancangan dan implementasi kakas. Perancangan kakas terdiri dari gambaran umum cara kerja kakas, desain antarmuka pengguna, perancangan diagram kelas, dan proses konstruksi visualisasi data. Sedangkan implementasi kakas berisi penjelasan tentang lingkungan implementasi, kode sumber kelas visualisasi graf, implementasi antarmuka pengguna, dan batasan implementasi yang dilakukan.

## IV.1 Perancangan Kakas

Kakas terdiri dari satu kelas utama, yaitu mengimplementasikan visualisasi graf. Kelas ini memiliki beberapa fungsi, mulai dari ekstrasi fitur data hingga menampilkan visual graf.

### IV.1.1 Gambaran Umum

Kakas yang dikembangkan fokus pada visualisasi graf eksekusi kode program. Tujuan pengembangan kakas adalah menambahkan fitur visualisasi graf pada kakas OPT, sehingga dapat digunakan untuk memahami eksekusi graf kode program. Gambar IV.1 menunjukkan skema umum hasil pengembangan kakas. *Input* berupa kode program C/C++, jika terdapat representasi graf di dalamnya, maka *output* dapat menyesuaikan untuk menampilkan visual graf.



1. Skema umum hasil pengembangan kakas



1. Skema umum modul visualisasi graf (*GraphVisualizer*)

Pada Gambar IV.2 menunjukkan skema umum khusus untuk kelas visualisasi graf. Sebenarnya yang diproses dari *input* kelas ini adalah berupa data JSON, bukan kode program. Data JSON yang terdapat representasi graf, akan diekstrak dan *output* bisa menyesuaikan dengan menampilkan visualisasi graf atau tidak.

### IV.1.2 Perancangan Antarmuka Pengguna

Pada Gambar IV.3 menunjukkan rancangan desain yang akan dikembangkan berdasarkan analisis kebutuhan antarmuka pengguna yang telah dilakukan pada subbab III.1.2. Secara garis besar, antarmuka hampir sama dengan kakas OPT. Perbedaannya adalah penambahan *tab* untuk membagi visualisasi menjadi dua, yaitu primitif dan visual graf. Pada *tab* primitif berisi visualisasi sebelumnya milik kakas OPT, sedangkan *tab Graph Visualization* berisi visualisasi graf yang sedang dikembangkan. Selain itu, *button control panel* akan tetap berada di bagian bawah tengah (posisi statis), walaupun pengguna melakukan aksi *scroll* pada kode program atau visualisasi. Posisi navigasi kontrol ini diharapkan dapat memudahkan pengguna ketika melakukan eksplorasi visualisasi. Pesan kesalahan akan ditampilkan dalam komponen *side panel* ketika terjadi *error* pada kode program. Jika kode program berjalan lancar tanpa ada pesan kesalahan, maka komponen ini akan menutup ke sebelah kiri.



1. Rancangan antarmuka pengguna

Pada *tab Graph Visualization* terdapat *panel* variabel, yaitu untuk menginformasikan variabel representasi graf kepada pengguna yang sedang ditampilkan. Proses animasi seperti pencarian rute, informasinya akan ditampilkan dalam *panel* variabel ini. Kemudian di bagian pojok kiri *tab*, ada status atau informasi tentang graf yang sedang tampil, seperti jumlah *node*, *edge*, statusnya berbobot atau tidak, berarah atau tidak.

### IV.1.3 Perancangan Diagram Kelas

Pada Gambar IV.4 terdapat kelas *GraphVisualizer* (kotak berwarna merah), yaitu modul visualisasi graf yang dikembangkan dalam tesis ini. Modul ini berisi beberapa fungsi, yang utama terdapat fungsi untuk pencocokan model dan proses *rendering* visualisasi graf. Kelas lain dimodifikasi seperlunya terkait perbaikan visual maupun yang ada hubungannya dengan proses di kelas *GraphVisualizer*.



1. Rancangan diagram kelas

Semua kelas memiliki ekstensi *.ts* (*TypeScript*). Kelas yang akan pertama kali dieksekusi adalah *Main*. Kelas *Main* dan *OptFrontendWithTestCases* memiliki relasi *composition*, sedangkan kelas *AbstractBaseFrontend* dengan *OptFrontend* berelasi *inheritance* (*extends*). Setiap kelas memiliki fungsi sebagai berikut.

1. Kelas *Main*: bagian utama yang akan pertama kali dieksekusi. Dimulai dengan deklarasi objek variabel:

let OptFrontend = new OptFrontendWithTestCases

1. Kelas *OptFrontendWithTestCases*: berfungsi untuk melakukan pengujian visualisasi dengan tampilan antarmuka pengguna.
2. Kelas *OptTestCases*: berfungsi untuk membuat butir uji.
3. Kelas *OptFrontendSharedSessions*: berfungsi untuk mendukung tampilan antarmuka ketika berkolaborasi dengan pengguna lain, bisa untuk *chatting* dan belajar bersama.
4. Kelas *OptFrontend*: berfungsi untuk mendukung komponen-komponen pada tampilan antarmuka pengguna.
5. Kelas *AbstractBaseFrontend*: berfungsi untuk model abstrak atau definisi dasar fungsi pada komponen antarmuka pengguna.
6. Kelas *ExecutionVisualizer*: berfungsi untuk menopang semua komponen visualisasi eksekusi kode program.
7. Kelas *DataVisualizer*: berfungsi untuk mengelola data visualisasi berupa *trace* eksekusi *JSON* dari *backend server*.
8. Kelas *ProgramOutputBox*: berfungsi untuk mengelola hasil *print* atau *standard ouput* (*stdout*) dari eksekusi kode program.
9. Kelas *CodeDisplay*: berfungsi untuk mengelola tampilan kode program dan perpindahan baris kode program pada setiap langkah *rendering*.
10. Kelas *NavigationController*: berfungsi untuk mendukung komponen *panel control button*.

### IV.1.4 Proses Konstruksi Visualisasi Data

Tujuh tahap berikut menjelaskan metode konstruksi visualisasi dari sumber perolehan data hingga perancangan desain interaksi visual graf. Teori tahapan ini telah dibahas pada subbab II.2 tentang metodologi dan prinsip visualisasi data. Ketujuh tahapan ini diperlukan untuk menganalisis data, sehingga dapat dibentuk visualisasi graf. Adapun langkah-langkahnya sebagai berikut:

1. *Acquire* (menentukan sumber data)

Tahap ini memastikan perolehan data yang dapat diproses lebih lanjut. Sumber data berupa *JSON* yang diperoleh dari proses *ajax*. Pada Tabel IV.1 merupakan contoh data eksekusi *trace JSON*. Data ini tersedia ketika ada permintaan dari pengguna melalui *browser*. Jadi, sumber data yang digunakan bersifat dinamis dan tergantung dari kode program yang sedang dieksekusi oleh pengguna.

1. Contoh data eksekusi *trace* *JSON* berupa matriks

|  |
| --- |
| **"code"**: "#include <stdlib.h>\n\nint a[10][10] ...”  **"trace"**: [{  **"event"**: "step\_line",  **"func\_name":** "main",  **"globals":** {  "a": ["**C\_MULTIDIMENSIONAL\_ARRAY**", "0x6010E0", [10, 10],  ["C\_DATA", "0x601264", "int", 0],  ["C\_DATA", "0x601268", "int", 0],  ["C\_DATA", "0x60126C", "int", 0], ...  ],  "f": ["C\_DATA", "0x601064", "int", 0],  "i": ["C\_DATA", "0x601270", "int", 0],  "j": ["C\_DATA", "0x601084", "int", 0],  "n": ["C\_DATA", "0x601080", "int", 0],  "q": ["C\_ARRAY", "0x601280", ["C\_DATA", "0x601280", "int", 0],  ["C\_DATA", "0x601284", "int", 0], ... ],  "r": ["C\_DATA", "0x601050", "int", -1],  "visited": ["C\_ARRAY", "0x6010A0", ["C\_DATA", "0x6010A0", "int", 0],  ["C\_DATA", "0x6010A4", "int", 0],  ["C\_DATA", "0x6010A8", "int", 0], ...  ]  },  **"heap":** {},  **"line":** 28,  **"ordered\_globals":** ["a", "q", "visited", "n", "i", "j", "f", "r"],  **"stack\_to\_render":** [{  **"encoded\_locals":** {  "v": ["C\_DATA", "0xFFF000BDC", "int", "<UNINITIALIZED>"]  },  **"frame\_id":** "0xFFF000BF0",  **"func\_name":** "main",  **"is\_highlighted":** true,  **"is\_parent":** false,  **"is\_zombie":** false,  **"line":** 28,  **"ordered\_varnames":** ["v"],  **"parent\_frame\_id\_list"**: [],  **"unique\_hash"**: "main\_0xFFF000BF0"  }],  **"stdout"**: ""  }, { ... }  ] |

1. *Parse* (mengurai dan klasifikasi data)

Data yang digunakan diklasifikasikan menjadi dua bagian besar (lihat Gambar IV.5), yaitu 2-dimensi *array* (berupa matriks) dan *pointer* (biasanya dalam bentuk *struct* atau *list*). Kedua tipe data ini yang digunakan sebagai dasar pembentukan visualisasi graf. Klasifikasi data ini dijelaskan dalam bentuk poin-poin sebagai berikut (lihat Gambar IV.6):

1. Data eksekusi *trace JSON* dibagi menjadi dua, yaitu multidimensi *array* (matriks) dan *pointer*. Karena representasi fisik yang terdapat dalam kode program untuk melakukan operasi struktur data graf secara umum adalah kedua tipe data tersebut. Atribut dan nilai yang akan diambil dari data *JSON* adalah C\_ARRAY, C\_MULTIDIMENSIONAL\_ARRAY, atau C\_DATA dengan berisi alamat memori dengan atribut bersarang berupa pointer.
2. Pendeklarasian variabel, yaitu *global* atau *local*. Definisi variabel global adalah sebuah variabel dengan tipe data tertentu yang dideklarasikan dengan *scope* global, maksudnya variabel dapat diakses bebas oleh semua fungsi atau prosedur dalam kode program. Sedangkan definisi variabel *local* lebih bersifat *private*, yaitu hanya dideklarasikan di dalam fungsi dan hanya dapat diakses di dalam fungsi itu sendiri.
3. Berbobot (*weighted*) atau tak-berbobot (*unweighted*). Klasifikasi ini digunakan untuk *edge* yang memiliki bobot (*cost*) atau tidak.
4. Berarah (*directed*) atau tak-berarah (*undirected*). Klasifikasi ini digunakan untuk *edge* yang memiliki anak panah atau tidak, yang berguna untuk menunjukkan graf berarah atau tak-berarah.

Untuk memudahkan dalam klasifikasi data, maka dibuat tabel pada setiap klasifikasi data yang memenuhi syarat. Daftar label untuk membantu klasifikasi data dapat dilihat pada Tabel IV.2 berikut ini:

1. Daftar label untuk klasifikasi data *JSON*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Matriks** | **Pointer** | **Global** | **Berbobot** | **Berarah** |
| Ya | ✓ | - | ✓ | - | - |
| Tidak | - | ✓ | - | ✓ | ✓ |

Hasil dari tabel tersebut adalah data berupa matriks dengan variabel global, tak-berbobot, dan tak-berarah. Penjelasan masing-masing label pada kolom tersebut adalah sebagai berikut:

1. Matriks adalah *array* berdimensi dua yang ditandai dengan atribut “C\_MULTIDIMENSIONAL\_ARRAY” di dalam data *JSON*.
2. *Pointer* adalah label yang digunakan untuk menandai adanya variabel yang didefinisikan sebagai *pointer* di dalam kode. Label ini digunakan ketika di dalam data *JSON* terdapat atribut ‘*pointer*’.
3. Label ‘*Global’* adalah variabel yang dideklarasikan di luar fungsi main(), sedangkan variabel *private* dideklarasikan di dalam fungsi main() atau dalam fungsi lain. Variabel *private* ditandai dengan nilai ‘tidak’ pada label ‘*global’* ini.
4. Label ‘berbobot’ digunakan ketika terdapat nilai rentang antara 0 – 999. Label ini digunakan untuk bobot *edge*.
5. Label ‘berarah’ digunakan ketika terdapat nilai antara *node* *i* (baris) ke *j* (kolom) tidak sama dengan *node* *j* (kolom) ke *i* (baris).



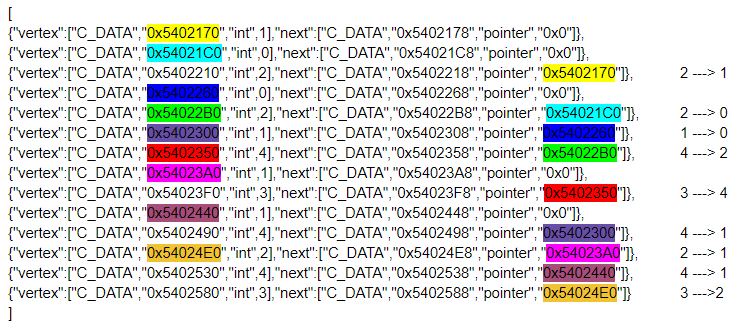
1. Klasifikasi data *JSON* menjadi dua bagian



1. Klasifikasi dari data matriks atau *pointer*
2. *Filter* (penyaringan data)

Proses ini mengambil data fitur yang dibutuhkan dan mengabaikan data lain. Karena dalam *JSON* tersebut memiliki banyak jenis data yang tidak berkaitan dengan terbentuknya visual graf. Untuk menguji keberadaan data fitur, atribut yang diuji mulai dari *global* (globals), kemudian *local* (stack\_to\_render [0].encoded\_locals). Sedangkan untuk tipe data *pointer*, atribut yang diuji adalah heap. Di dalam heap, terdapat alamat memori yang saling terhubung jika fitur data merepresentasikan node dan edge.

Contoh data *JSON* dengan tipe data *pointer* yang telah di-*filter* dapat dilihat pada Gambar IV.7. Dua warna yang sama mengindikasikan alamat memori yang saling terhubung. Terlihat ada data berupa [“C\_DATA”,”0x5402170”,”int”,1], artinya kolom pertama (C\_DATA) adalah tipe data C, kolom kedua adalah alamat memori, kolom ketiga adalah tipe data *integer*, dan kolom terakhir isi nilai dari memori tersebut. Jika kolom ketiga berisi “*pointer*” dan kolom keempat berisi alamat memori (misalnya “0x5402170”), ini mengindikasikan tipe data *pointer* yang menunjuk pada alamat memori yang ada di kolom keempat tersebut. Pada Gambar IV.7 di bagian kanan terdapat nilai 2 🡪 1 dan seterusnya, ini untuk menjelaskan bahwa nilai *node 2* terhubung dengan nilai *node 1*.



1. Contoh data *JSON pointer* yang telah di-*filter*
2. *Mine* (menggali informasi)

Data yang diproses selanjutnya dicari informasi yang dapat menunjukkan adanya graf di dalam kode program. Seperti yang telah dijelaskan pada subbab III.2 bahwa representasi graf bisa dilakukan dengan tiga cara berbeda. Tiga cara ini dibuat model, kemudian dicocokkan mana yang lebih mendekati data *JSON* tersebut dengan ketiga model ini. Pada Gambar IV.8 terlihat diagram alir *level-1* untuk proses pencocokan model. Untuk tahap pengecekan tipe data sesuai model atau tidak adalah untuk menguji tipe data yang tersedia dalam *JSON* seperti yang telah dijelaskan pada langkah tiga di atas. Lalu tahap pengecekan satu variabel bertujuan untuk memastikan visual graf hanya ada satu. Karena saat ini hanya dikembangkan untuk satu visual graf. Jika data ini cocok dengan model, maka selanjutnya disimpan ke dalam bentuk format data visual graf *D3.js* yang bisa digunakan untuk langkah selanjutnya. Jika tidak cocok dengan model saat ini, maka berlanjut ke model berikutnya hingga semua model telah dilakukan pengujian.



1. Diagram alir *level-1* pencocokan model
2. *Represent* (representasi)

Pada bagian ini proses representasi data menggunakan *library* *D3.js*. *Library* *D3.js* membantu dalam proses visualisasi data. Format data seperti pada Tabel IV.3 dapat digunakan oleh *D3.js* untuk visual graf adalah sebagai berikut:

1. Format dasar yang digunakan untuk visualisasi graf dengan *D3.js*

|  |
| --- |
| { **nodes**: [{ name: 1 }, { name: 2 }],  **edges**: [{ source: 0, target: 1}, { source: 2, target: 1}]} |

Jika *edge* memiliki bobot, maka akan ditambah atribut *weight*, seperti yang terlihat pada Tabel IV.4 berikut ini:

1. Format untuk *edge* dengan bobot

|  |
| --- |
| **edges**: [{ source: 0, target: 1, **weight: 18**}, ...] |

Representasi visual graf dapat melalui proses seperti pada Gambar IV.9. Pada proses “Pencocokan Model” telah dilakukan pada langkah sebelumnya (pada langkah 4 *Mine* atau menggali informasi). Proses selanjutnya adalah visualisasi menggunakan *library* *D3.js*. Proses *rendering* visualisasi dapat dilihat pada Gambar IV.10.



1. Diagram alir *level-0* proses visualisasi data



1. Diagram alir *level-1* proses *rendering* visualisasi graf
2. *Refine* (perbaikan visual)

Bagian ini merupakan perbaikan desain visual dari representasi dasar seperti pada Gambar IV.11 (a). Perbaikan visual (lihat Gambar IV.11 (b)) yang dilakukan diantaranya pemilihan warna komponen, bentuk dan ketebalan garis, pemilihan *font*, posisi label, dan warna latar.

|  |  |
| --- | --- |
| (a) | (b) |

1. Visual graf: (a) visual dasar graf; (b) setelah perbaikan
2. *Interact* (interaksi)

Menambahkan fitur interaksi untuk memudahkan pengguna dalam mengeksplorasi visualisasi data. Langkah ini menambahkan fitur *sticky*, yaitu *node* yang mendapatkan aksi *double-click* dan *drag* dari pengguna, akan mengikuti posisi *cursor*. *Node* akan berubah warna berbeda dari *node* lain yang tidak mendapatkan aksi *double-click* dari pengguna (lihat Gambar IV.12 (a)). Fitur ini membantu pengguna untuk melihat graf dengan model atau posisi sesuai yang diinginkan.

Selain fitur tersebut, graf dapat lentur bebas bergerak untuk memberikan kesan menarik dan sebaran data *node* yang merata apabila ada penambahan. Graf juga akan kembali berada di posisi tengah ketika pengguna melakukan salah satu *drag* pada *node* hingga ke tepi *canvas*.

Fitur animasi perubahan warna pada *node* dan *edge* yang sedang dilakukan proses algoritma, sehingga visualisasi terlihat sinkron dengan jalannya program. Contoh animasi dapat dilihat pada Gambar IV.12 (b). Operasi yang dilakukan dalam program biasanya seperti pencarian jalur terpendek.

|  |  |
| --- | --- |
| (a) | (b) |

1. (a) Perbaikan desain interaksi visual; (b) Fitur animasi

## IV.2 Implementasi Kakas

Implementasi kakas menggunakan bahasa pemrograman *TypeScript* versi 2.4 dan *code editor* Visual Studio Code versi 1.14 (https://code.visualstudio.com). Beberapa *library* perlu diperbarui versi pengembangannya seperti D3.js dari versi 2.0 ke versi 3.0, *TypeScript* dari versi 1.8 ke versi 2.4. Hal ini dilakukan untuk memutakhirkan fitur-fitur yang tersedia di dalam *library* tersebut.

### IV.2.1 Lingkungan Implementasi

Implementasi dilakukan pada sebuah laptop Lenovo G40 dengan spesifikasi perangkat keras sebagai berikut.

*Processor* : Intel Core i5-5200U @2.20GHz (4 CPU)

*RAM* : 4 GB (empat *giga bytes*)

*VGA* : AMD Radeon R5 M230 (4 GB)

Spesifikasi perangkat lunak yang digunakan adalah sebagai berikut.

Sistem Operasi : Ubuntu 16.04 LTS

*Backend server* : Docker engine (docker pull habibie/valgrind\_backend)

*Browser* : Chrome 64.0.3282, Mozzila Firefox 58.0

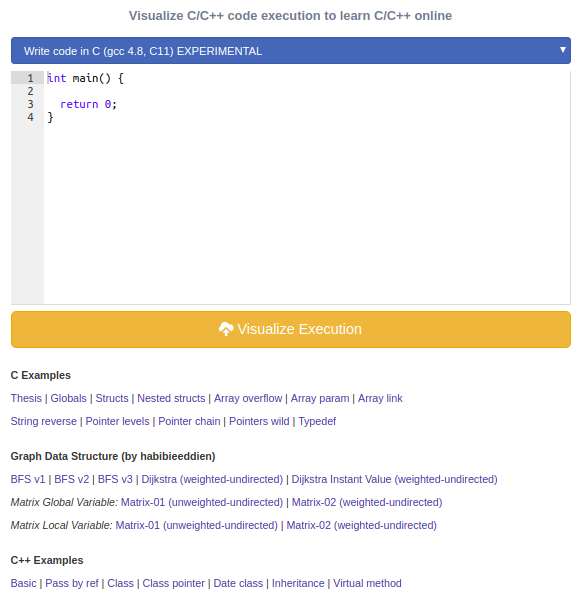
Konfigurasi yang harus dilakukan untuk menjalankan kakas ini dapat dilihat pada petunjuk teknis Lampiran A.

### IV.2.2 Implementasi Modul Visualisasi Graf

Implementasi dari kelas visualisasi graf dapat dilihat pada Lampiran B.

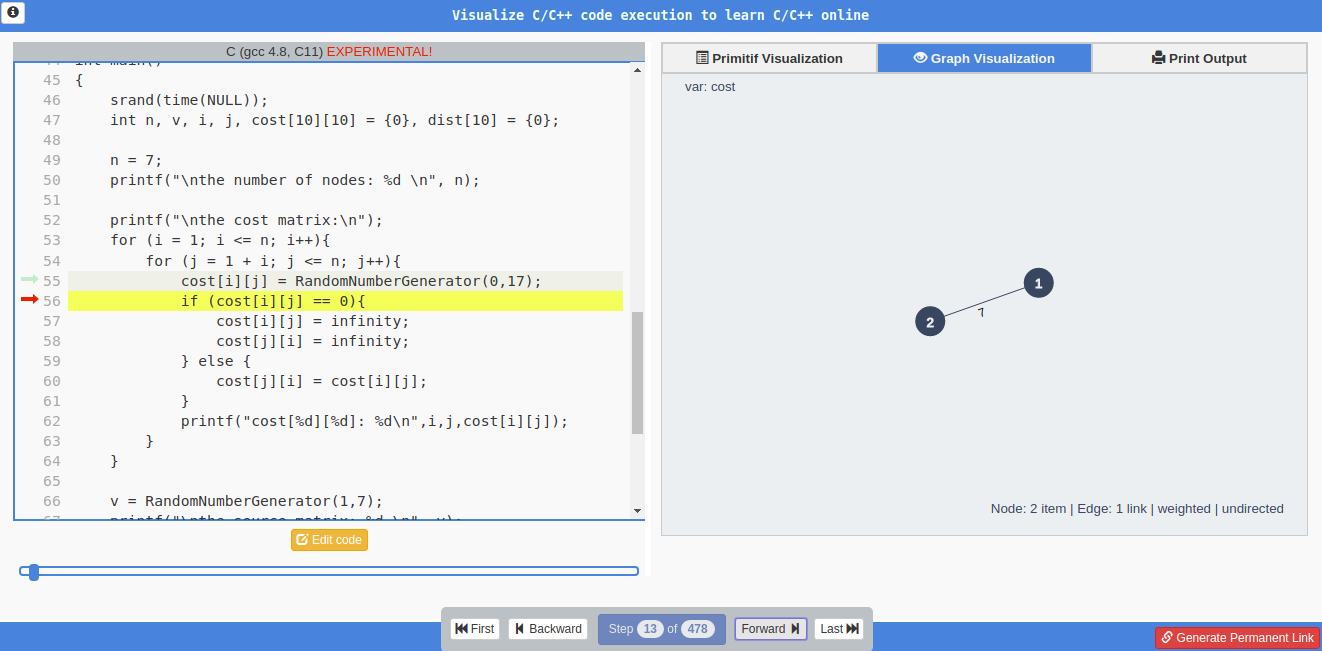
### IV.2.3 Implementasi Antarmuka Pengguna

Pada Gambar IV.13 menunjukkan implementasi antarmuka pengguna untuk *input* kode program C/C++ di *browser Chrome*. Pengguna dapat memilih contoh kode program di bagian bawah tombol ‘*Visualize Execution*’. Pengguna juga dapat mengetik sendiri kode program di *code editor* yang telah disediakan. Hasil visualisasi akan tampil seperti pada Gambar IV.14. Pengguna dapat eksplorasi eksekusi kode program dengan navigasi kontrol (Gambar IV.14 (a)) yang tersedia, atau dapat menggunakan *slider* (Gambar IV.14 (b)) untuk melihat langsung pada langkah yang diinginkan.



1. Implementasi antarmuka pengguna untuk *input* kode program

Gambar IV.14 (c) merupakan panel yang terdiri dari tiga visualisasi, yaitu “*Primitif Visualization*” adalah untuk menampilkan data variabel kode program secara sederhana berupa *stack* dan *heap*. Panel “*Graph Visualization*” adalah bagian utama visualisasi graf yang dikembangkan dalam penelitian ini. Panel “*Print Output*” adalah untuk menampilkan standar *output* yang berasal dari perintah printf di dalam kode program.



a

c

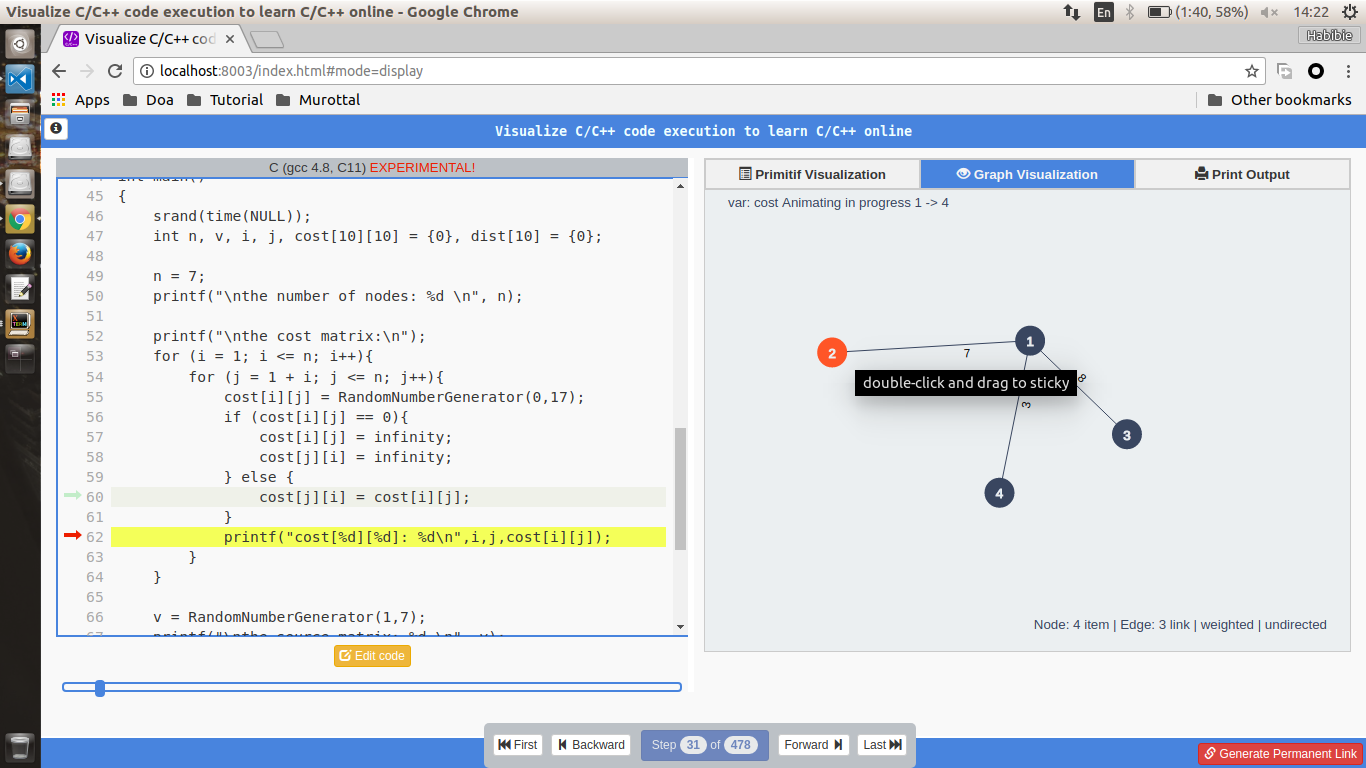
a

b

a

a

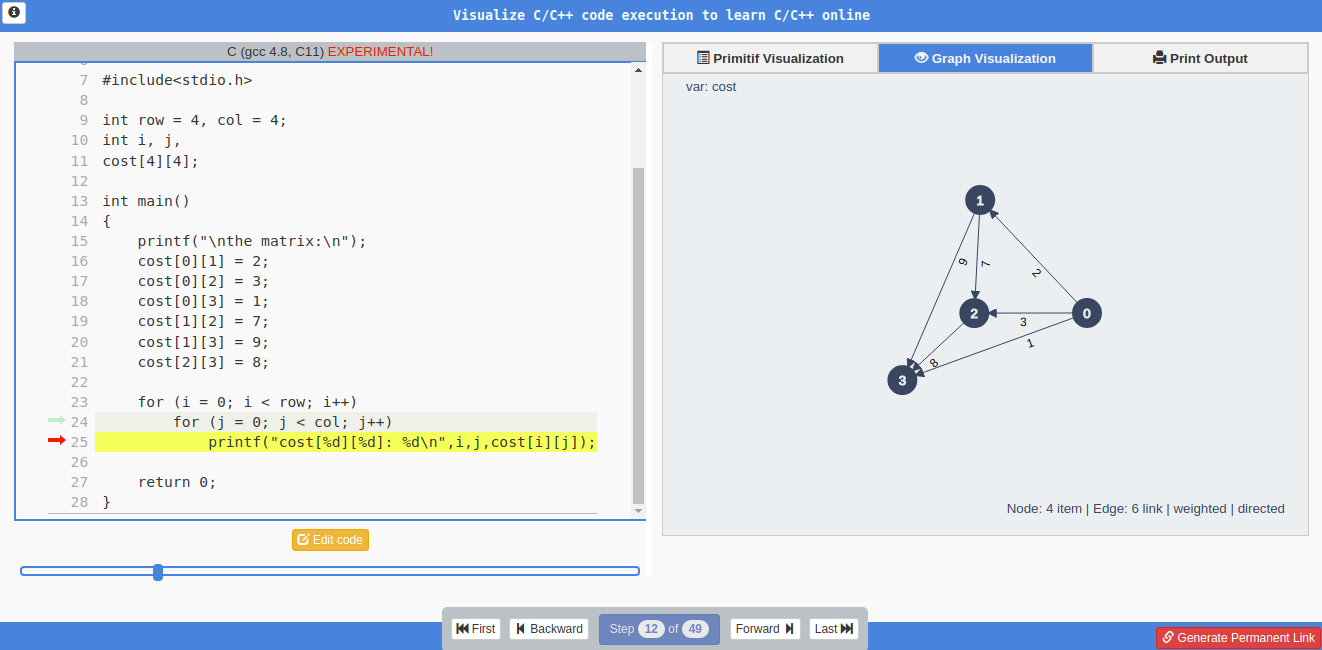
1. Implementasi antarmuka pengguna visualisasi graf kode program; (a) Navigasi kontrol; (b) *Slider*; (c) Panel visualisasi



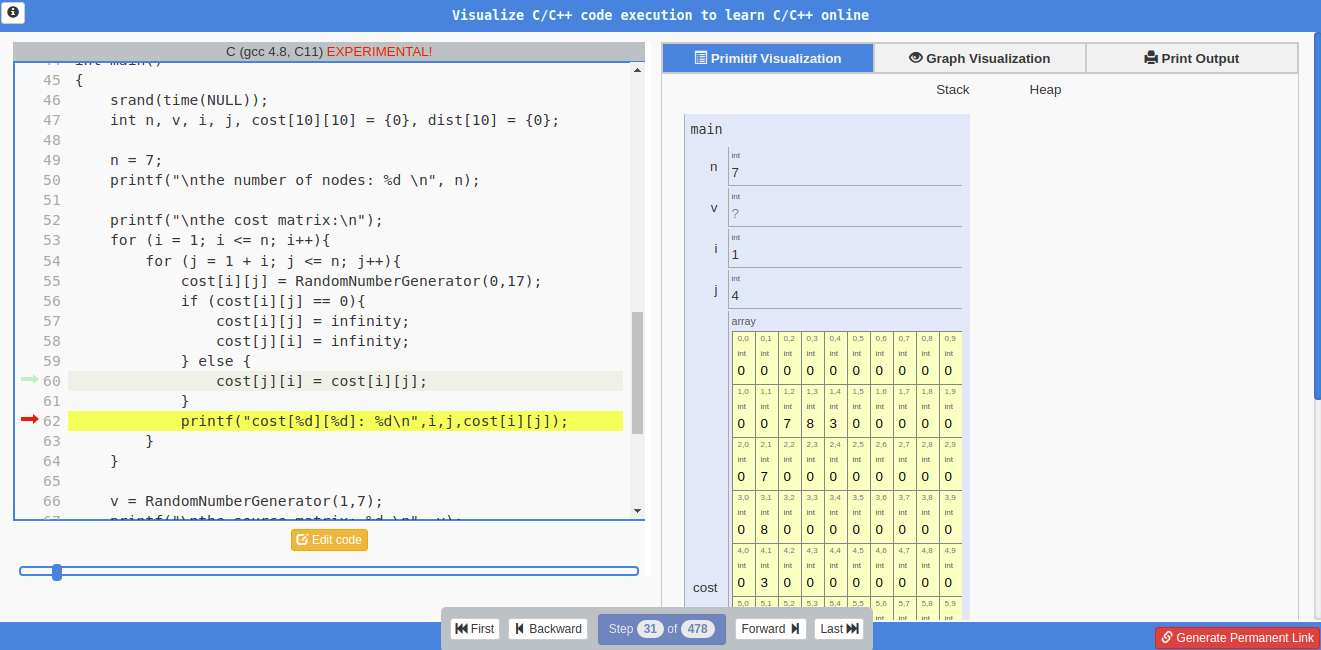
1. Implementasi visualisasi graf berbobot tak-berarah

Pada Gambar IV.15 menunjukkan implementasi kakas untuk visualisasi graf berbobot dan tak-berarah dengan representasi data matriks. *Node* dapat menerima aksi *drag* dan *drop* dari pengguna untuk diposisikan sesuai yang diinginkan oleh pengguna.

Pada Gambar IV.16 menunjukkan implementasi kakas untuk visualisasi graf berbobot dan berarah dengan representasi data matriks. *Edge* akan otomatis menampilkan anak panah jika terdeteksi data variabel yang berisi graf berarah. Label angka untuk bobot tiap *edge* juga otomatis tampil ketika terdeteksi variabel bobot antar *node*.

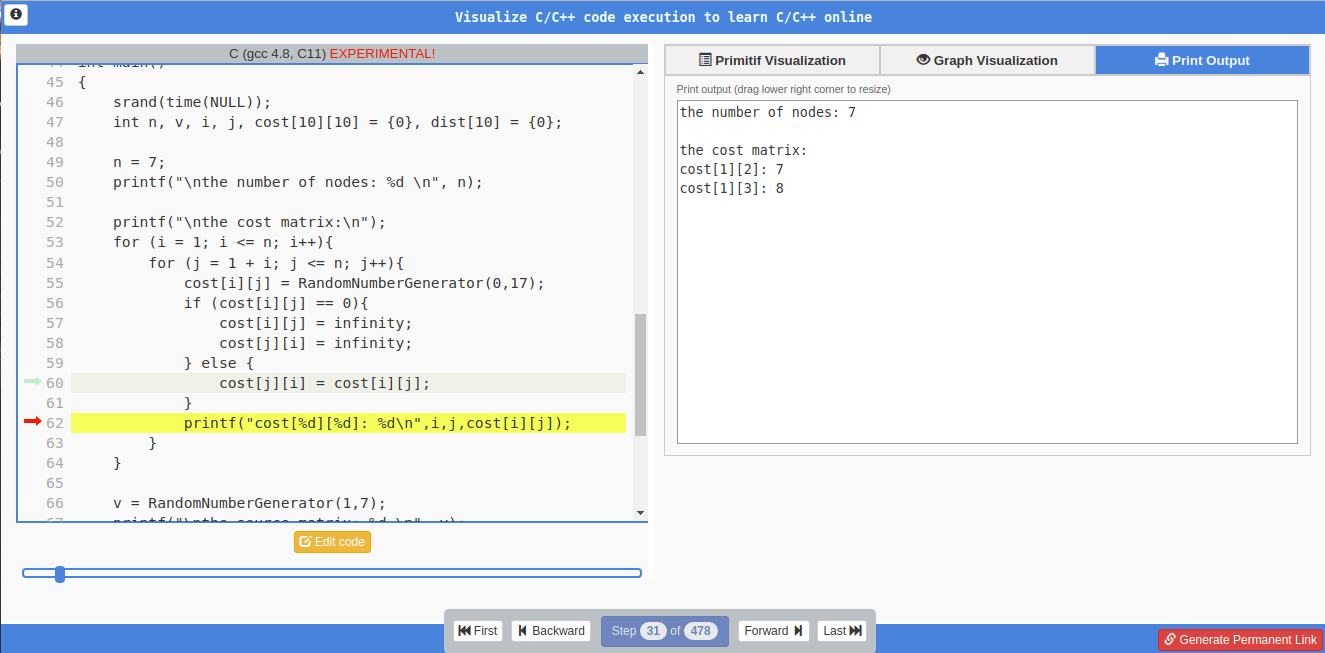


1. Implementasi visualisasi graf berbobot berarah

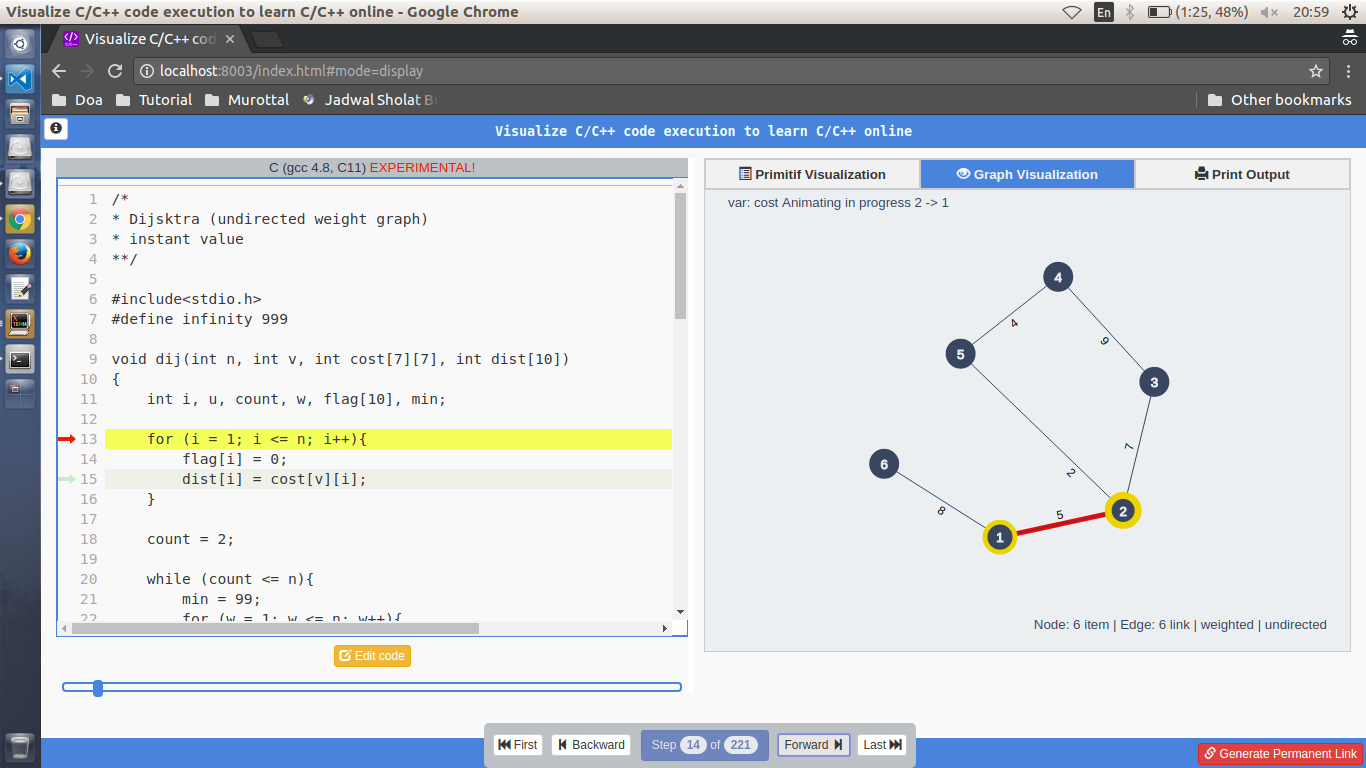
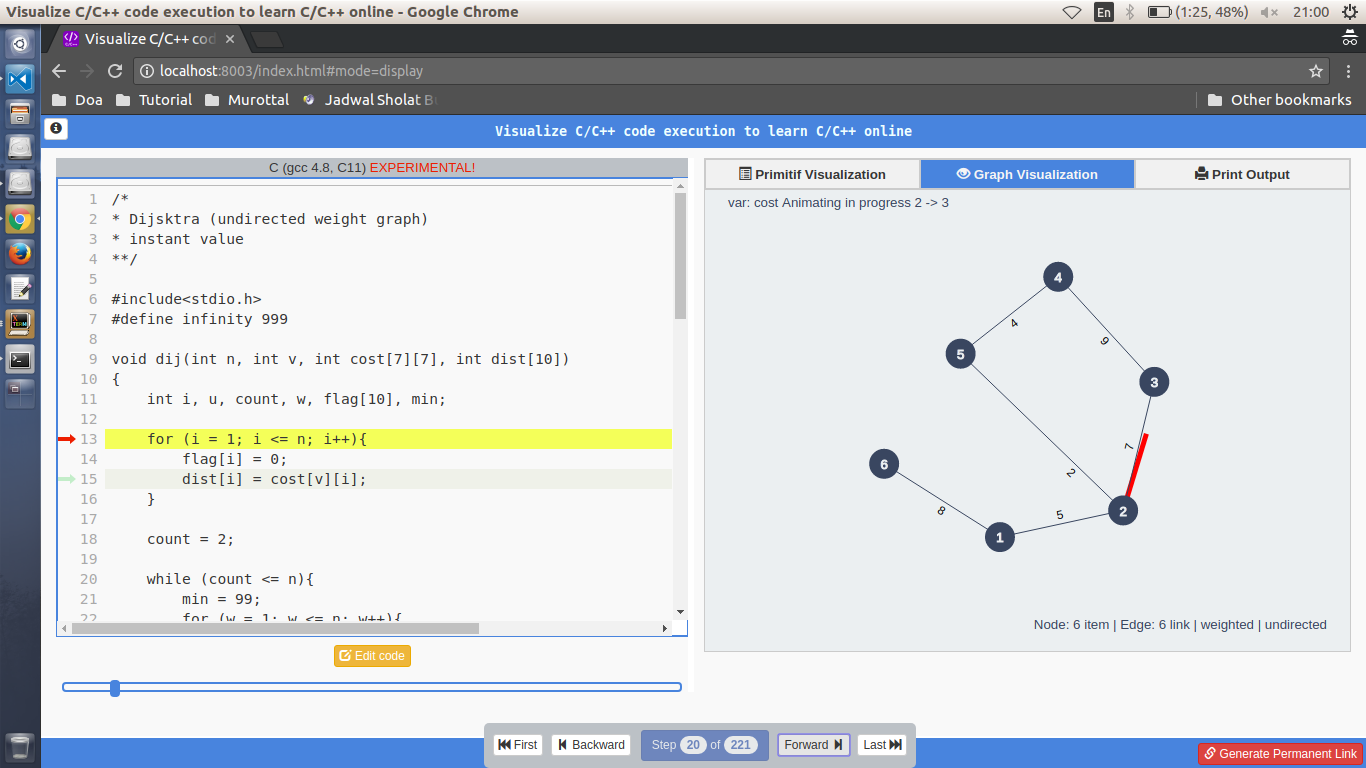


1. Implementasi panel “*Primitif Visualization*”

Pada Gambar IV.17 dan Gambar IV.18 merupakan implementasi untuk panel “*Primitif Visualization*” dan “*Print Output*” yang sebenarnya sudah tersedia pada kakas OPT sebelumnya. Hanya menempatkan pada panel untuk memudahkan pengguna dalam mengaksesnya sesuai yang telah dijelaskan pada subbab III.1.2 tentang rancangan antarmuka pengguna.



1. Implementasi panel “*Print Output*”

b

a

a

a

1. Implementasi fitur animasi pencarian rute antar *node*

Pada Gambar IV.19 menunjukkan implementasi fitur animasi untuk pencarian rute antar node. Huruf (a) pada Gambar IV.19 merupakan animasi pencarian rute dari *node* 2 ke *node* 1. Huruf (b) pada Gambar IV.19 merupakan kelanjutan animasi pencarian rute dari *node* 2 ke *node* 3.

### IV.2.4 Batasan Implementasi

Batasan implementasi pada tesis ini adalah sebagai berikut:

1. *Step rendering* yang dapat ditampilkan maksimal 1000 (seribu) langkah. Pembatasan ini dilakukan karena untuk menghindari adanya kode program yang berisi *infinite looping* sehingga menyebabkan *server backend* mengalami kerusakan.
2. Berdasarkan poin a, baris kode program yang dapat dijadikan *input* maksimal 1000 (seribu) baris, jika kondisi tiap baris kode program dibaca sekali dalam *step rendering*. Namun, jika terdapat pengulangan, maka visual *rendering* akan berhenti pada baris maksimal tersebut.
3. *Backend server* dapat melayani setiap permintaan (*Ajax Request*) dengan waktu tunggu kompilasi selama maksimal 1 (satu) menit. Jika melebihi batas waktu kompilasi, maka akan dikembalikan dengan pesan kesalahan untuk memperbaiki kode program dan mencoba kembali. Pesan kesalahan ini akan muncul ketika melebihi beban kompilasi baris kode program.
4. *Backend server* dibatasi penggunaan *RAM* di *docker engine* sebesar 1 GB (satu *giga bytes*).
5. Graf yang dapat divisualisasi maksimal 50 *node*. Mengingat kakas ini dikembangkan sebagai media belajar pemrograman, bukan untuk pengembangan perangkat lunak dengan skala besar.
6. Graf yang dapat divisualisasi masih terbatas yang berlabel angka, bukan huruf (*string*).
7. Representasi graf dalam kode program hanya dapat ditampilkan dari satu variabel saja. Jika dalam kode program terdapat lebih satu variabel representasi graf, maka visualisasi tidak akan dapat ditampilkan.
8. Platform yang digunakan adalah web adaptif dengan ukuran layar minimal 800 × 600 *pixels*.
9. *Input* bahasa pemrograman yang dapat divisualisasi adalah C dan C++.
10. Kakas tidak menerima *standard input* (*stdin*) dan tidak menerima deklarasi *header file* dari pengguna, artinya hanya *Standard Template Library* (STL) *C/C++* yang telah tersedia di *compiler server*.

# Bab V Evaluasi Visualisasi

Pada bab ini dijelaskan proses evaluasi visualisasi. Tujuannya adalah untuk mengukur keefektifan visualisasi dalam memahami eksekusi graf kode program.

## Survei Pengguna dengan Kuesioner *Online*

Hipotesis penelitian ini adalah pengguna dapat lebih efektif dan efisien dalam memahami eksekusi graf kode program melalui pendekatan visualisasi. Hipotesis tersebut perlu diuji dan dilakukan evaluasi visualisasi. Dalam hal ini, responden dapat dibagi menjadi dua jenis, yaitu pertama, responden yang pernah mempelajari konsep atau teori graf. Kedua, responden yang belum pernah mempelajari konsep atau teori graf. Kedua jenis responden ini akan nampak korelasinya terhadap hasil keefektifan visualisasi yang dimaksud.

Salah satu metode yang digunakan untuk mengevaluasi hasil pengembangan kakas ini adalah melalui survei pengguna dengan kuesioner *online* (teknik *query*) (Urquiza-Fuentes dan Velázquez-Iturbide, 2009). Metode kuesioner secara *online* dipilih karena memiliki kecepatan respon data yang cepat dan kualitas data yang dapat langsung disimpan ke basis data (Sue dan Ritter, 2007). Kuesioner *online* dapat merekam perilaku pengguna dalam menggunakan kakas secara akurat. Pertanyaan kontingensi dapat diprogram dengan efektif sehingga objektivitas permasalahan dapat tercapai.

Keefektifan visualisasi diukur berdasarkan tujuan yang ingin dicapai sesuai yang telah dijelaskan pada subbab III.1. Metodenya (Velázquez-Iturbide dkk., 2017) adalah dengan menghitung selisih hasil nilai responden dalam mengerjakan kuesioner melalui simulasi kakas OPT dan kakas hasil pengembangan yang selanjutnya disebut *CodeViz*. Jika hasil nilai tes simulasi menggunakan kakas *CodeViz* lebih tinggi dari kakas OPT, maka visualisasi yang dikembangkan ini dapat disimpulkan efektif atau tujuan visualisasi telah tercapai. Namun sebaliknya, jika hasil nilai tes simulasi menggunakan kakas *CodeViz* lebih rendah dari kakas OPT, maka dapat dikatakan visualisasi yang dikembangkan tidak berdampak signifikan terhadap keefektifan visualisasi. Selanjutnya kuesioner ini dibagi menjadi empat tahap, yaitu sebagai berikut:

1. **Tahap 1**: Responden mengisi data pribadi. Tujuan tahap ini untuk memastikan data tidak fiktif dan dapat diberikan insentif sebagai penghargaan telah mengisi kuesioner (Sue dan Ritter, 2007).
2. **Tahap 2**: Responden mengisi pertanyaan pre-tes. Tujuan tahap ini untuk menggali informasi atau pengetahuan yang telah dimiliki oleh responden sehingga dapat dianalisis korelasi antara data hasil tes simulasi dan *post*-tes.
3. **Tahap 3**: Responden melakukan simulasi dengan menggunakan kakas OPT dan *CodeViz*. Pada tahap ini terdapat tiga skenario, yaitu pertama, responden bebas eksplorasi terhadap kakas OPT dan kakas *CodeViz*. Tujuannya adalah untuk menilai tingkat *usability* kedua kakas yang berkorelasi erat dengan keefektifan visualisasi. Kedua, responden diminta menyelesaikan soal terkait graf kode program. Ketiga, responden ditugaskan untuk memperbaiki graf kode program. Skenario kedua dan ketiga dimaksudkan untuk menilai keefektifan tujuan visualisasi yang ingin dicapai.
4. **Tahap 4**: Responden mengisi pertanyaan *post-*tes. Pertanyaan ini dibagi dua, yaitu untuk kakas OPT dan CodeViz. Tahap ini bertujuan untuk mengumpulkan opini responden terhadap kedua kakas tersebut sebagai saran pengembangan.

Pengukuran kuantitatif dilakukan secara implisit dengan beberapa metode (Preece dkk., 2015), yaitu sebagai berikut:

1. Lama waktu pengguna dalam menyelesaikan sebuah tugas;
2. Lama waktu pengguna dalam menyelesaikan sebuah tugas ketika batas waktu ditentukan;
3. Jumlah dan jenis kesalahan yang terjadi setiap tugas yang diberikan;
4. Jumlah kesalahan per unit dalam satuan waktu;
5. Jumlah pengguna yang mengakses bantuan atau pedoman produk;
6. Jumlah pengguna yang melakukan kesalahan;
7. Jumlah pengguna yang menyelesaikan tugas.

Kegiatan survei dilakukan dengan perencanaan menyusun pertanyaan dan tujuan yang ingin dicapai sehingga pertanyaan tidak meluas atau dapat sesuai dengan objektivitas penelitian (Sue dan Ritter, 2007). Pertanyaan dan objektivitas penelitian disajikan pada Lampiran C.1.

Teknik sampling (Sugiono, 2015) yang digunakan dalam penelitian ini adalah *Proportionate Stratified Random Sampling,* yaitu teknik sampling yang digunakan bila populasi mempunyai anggota atau unsur yang tidak homogen dan berstrata secara proporsional. Subyek penelitian adalah mahasiswa informatika sarjana (S1) dan pascasarjana (S2) di Institut Teknologi Bandung. Setiap jenjang S1 dan S2 diambil tahun angkatan 2015, 2016, dan 2017. Daftar populasi diurutkan berdasarkan Nomor Induk Mahasiswa (NIM) yang terlampir pada Lampiran C.2. Uraian mengenai populasi dan sampel yang akan diambil adalah sebagai berikut.

Jumlah populasi jenjang S1 sebanyak 150 orang terdiri dari tiga angkatan, yaitu:

1. Angkatan tahun 2015 sebanyak 50 orang;
2. Angkatan tahun 2016 sebanyak 50 orang;
3. Angkatan tahun 2017 sebanyak 50 orang.

Jumlah populasi jenjang S2 sebanyak 95 orang terdiri dari tiga angkatan, yaitu:

1. Angkatan tahun 2015 sebanyak 35 orang;
2. Angkatan tahun 2016 sebanyak 30 orang;
3. Angkatan tahun 2017 sebanyak 30 orang.

Total jumlah populasi adalah 150 + 95 = 245 orang.

Jumlah sampel yang akan diambil secara acak sebanyak 23 orang.

Teknik penentuan ukuran sampel per jenjang strata adalah sebagai berikut:

S1: 150/245 X 23 = 14,081 = 14 orang

S2: 95/245 X 23 = 8,91 = 9 orang

Jumlah = 23 orang

Sedangkan untuk teknik penentuan ukuran sampel per angkatan adalah sebagai berikut:

1. Jenjang S1

Angkatan tahun 2015: 50/150 X 14 = 4,66 = 4 orang

Angkatan tahun 2016: 50/150 X 14 = 4,66 = 5 orang

Angkatan tahun 2017: 50/150 X 14 = 4,66 = 5 orang

Jumlah = 14 orang

1. Jenjang S2

Angkatan tahun 2015: 35/95 X 9 = 3,31 = 3 orang

Angkatan tahun 2016: 30/95 X 9 = 2,84 = 3 orang

Angkatan tahun 2017: 30/95 X 9 = 2,84 = 3 orang

Jumlah = 9 orang

Adapun cara penarikan sampel pada setiap angkatan dilakukan dengan cara undian menggunakan rumus *RANDBETWEEN* dalam aplikasi *Microsoft Excel*. Hasil dari rumus tersebut menunjukkan angka nomor akhir NIM yang terpilih. Khusus untuk jenjang S2 angkatan tahun 2015, hasil yang keluar dari rumus tersebut merupakan nomor urut dari tabel yang telah diurutkan berdasarkan angka terendah sampai angka tertinggi (lihat Tabel V.1), karena NIM yang tertera tidak urut dari 001 sampai dengan 050. Hasil pengundiannya adalah sebagai berikut:

1. Daftar anggota sampel yang akan mengisi kuesioner

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **S1** | | | **S2** | | |
| **Tahun 2015** | **Tahun 2016** | **Tahun 2017** | **Tahun 2015** | **Tahun 2016** | **Tahun 2017** |
| 1 | 23 | 24 | 46 | 19 | 19 | 15 |
| 2 | 19 | 17 | 8 | 3 | 30 | 13 |
| 3 | 22 | 29 | 11 | 5 | 5 | 4 |
| 4 | 14 | 41 | 30 |  |  |  |
| 5 |  | 27 | 9 |  |  |  |

## Hasil Kuesioner *Online*

## Analisis Hasil Kuesioner

# Bab VI Kesimpulan dan Saran

## VI.1 Kesimpulan

Beberapa kesimpulan yang telah diperoleh selama penelitian ini adalah sebagai berikut:

1. Visualisasi ...

## VI.2 Saran

Berdasarkan hasil penelitian ...

+ jika bisa compile di komputer client, akan lebih memudahkan

+ proses visualisasi graf mungkin bisa dipadukan dengan metode text mining / text pattern recognition

# DAFTAR PUSTAKA

Bonk, C.J. (2009): *The world is open: how Web technology is revolutionizing education*, 1st ed, San Francisco, Calif, Jossey-Bass.

Bostock, M., Ogievetsky, V. dan Heer, J. (2011): D3: Data-Driven Documents, *IEEE Trans. Vis. Comput. Graph.*, **17**, 2301–2309.

Bruening, D. dan Zhao, Q. (2011): Practical memory checking with Dr. Memory, *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, IEEE Computer Society, 213–223.

Cetin, I. dan Andrews-Larson, C. (2016): Learning sorting algorithms through visualization construction, *Comput. Sci. Educ.*, **26**, 27–43.

Cross II, J.H., Hendrix, T.D., Jain, J. dan Barowski, L.A. (2007): Dynamic object viewers for data structures, *ACM SIGCSE Bulletin*, ACM, 4–8.

Fry, B. (2008): *Visualizing Data*, First Edition, USA, O`Reilly Media, Inc.

Gestwicki, P. dan Jayaraman, B. (2005): Methodology and architecture of JIVE, *Proceedings of the 2005 ACM symposium on Software visualization*, ACM, 95–104.

Gračanin, D., Matković, K. dan Eltoweissy, M. (2005): Software visualization, Innov. Syst. Softw. Eng., 1, 221–230.

Guo, P.J. (2013): Online python tutor: embeddable web-based program visualization for cs education, Proceeding of the 44th ACM technical symposium on Computer science education, ACM, 579–584.

Helminen, J. dan Malmi, L. (2010): Jype-a program visualization and programming exercise tool for Python, *Proceedings of the 5th international symposium on Software visualization*, ACM, 153–162, diperoleh melalui situs internet: http://dl.acm.org/citation.cfm?id=1879234 (diakses 20 July 2016).

Hendrix, T.D., Cross II, J.H. dan Barowski, L.A. (2004): An extensible framework for providing dynamic data structure visualizations in a lightweight IDE, *ACM SIGCSE Bulletin*, ACM, 387–391.

Holmberg, N., Wünsche, B. dan Tempero, E. (2006): A framework for interactive web-based visualization, *Proceedings of the 7th Australasian User interface conference-Volume 50*, Australian Computer Society, Inc., 137–144, diperoleh melalui situs internet: http://dl.acm.org/citation.cfm?id=1151778 (diakses 15 September 2017).

Jiménez-Peris, R., Patiño-Martínez, M. dan Pacios-Martínez, J. (1999): VisMod: a beginner-friendly programming environment, ACM Press, 115–120, diperoleh melalui situs internet: http://portal.acm.org/citation.cfm?doid=298151.298218 (diakses 31 December 2017).

Korhonen, A., Malmi, L., Silvasti, P., Karavirta, V., Lönnberg, J., Nikander, J., Stålnacke, K. dan Tenhunen, P. (2004): *Matrix - a framework for interactive software visualization*, Research Report TKO-B 154/04, Department of Computer Science and Engineering, Helsinki University of Technology, 26–35.

Nielsen, J. (n.d.): Usability First - Methods - Heuristic Evaluation | Usability First, *Heuristic Evaluation*, diperoleh melalui situs internet: http://www.usabilityfirst.com/usability-methods/heuristic-evaluation (diakses 15 Februari 2018).

Piteira, M. dan Costa, C. (2013): Learning computer programming: study of difficulties in learning programming, *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, ACM, 75–80, diperoleh melalui situs internet: http://dl.acm.org/citation.cfm?id=2503871 (diakses 8 September 2016).

Preece, J. (2002): *Interaction Design: Beyond Human-Computer Interaction, First Edition*, First Edition, USA, John Wiley & Sons, Inc.

Preece, J., Sharp, H. dan Rogers, Y. (2015): *Interaction Design, beyond human-computer interaction Fourth Edition*, 4th edition, John Wiley & Sons, Ltd.

Sedgewick, R. dan Wayne, K. (2011): *Algorithms Fourth Edition*, Fourth Edition, USA, Pearson Education, Inc.

Shaffer, C.A., Heath, L.S. dan Yang, J. (1996): Using the Swan data structure visualization system for computer science education, *ACM SIGCSE Bull.*, **28**, 140–144.

Sorva, J. (2012): *Visual program simulation in introductory programming education*, Aalto University publication series Doctoral dissertations, Espoo, Aalto Univ. School of Science.

Sorva, J., Karavirta, V. dan Malmi, L. (2013): A review of generic program visualization systems for introductory programming education, ACM Trans. Comput. Educ. TOCE, 13, 15.

Sue, V.M. dan Ritter, L.A. (2007): *Conducting online surveys*, United States of America, Sage Publications.

Sugiono (2015): *Metode Penelitian Tindakan Komprehensif*, Bandung, Alfabeta.

Thompson, E., Luxton-Reilly, A., Whalley, J.L., Hu, M. dan Robbins, P. (2008): Bloom’s Taxonomy for CS Assessment, ACE ’08, Wollongong, NSW, Australia, Australian Computer Society, Inc., 155–161, diperoleh melalui situs internet: https://dl.acm.org/citation.cfm?id=1379265.

Urquiza-Fuentes, J. dan Velázquez-Iturbide, J.Á. (2009): A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems, *ACM Trans. Comput. Educ.*, **9**, 1–21.

Velázquez-Iturbide, J.Á., Hernán-Losada, I. dan Paredes-Velasco, M. (2017): Evaluating the Effect of Program Visualization on Student Motivation, *IEEE TRANSACTIONS ON EDUCATION*.

Ware, C. (2004): *Information visualization: perception for design, 2nd edition*, 2nd edition, San Francisco, Kanada, Elsevier Inc.

# LAMPIRAN

Lampiran A Petunjuk Teknis Konfigurasi Kakas

Untuk mengoperasikan hasil pengembangan kakas ini, dapat mengikuti langkah-langkah sebagai berikut:

1. **Sistem Operasi Linux / Ubuntu**
2. Pastikan *Git* telah di-*install* pada komputer dan jalankan perintah berikut di *terminal* untuk mengunduh proyek tesis ini:

|  |
| --- |
| git pull http://gitlab.informatika.org/habibieeddien/TESIS.git |

1. Untuk pengembangan file ada di folder src/dev langkah-langkahnya adalah sebagai berikut:
   1. Pastikan komputer telah ter-*install* *NodeJs* dan *npm*, kemudian lakukan perintah berikut untuk memasang semua dependensi *library*:

|  |
| --- |
| npm install |

* 1. *Install* aplikasi *docker* dengan perintah berikut:

|  |
| --- |
| sudo apt get install docker.io |

Untuk *pull images backend*, jalankan perintah berikut:

|  |
| --- |
| docker pull habibie/valgrind\_backend |

Jika terjadi *error* atau isu lebih lanjut, dapat membaca petunjuk di berkas *src/dev/README.txt*

* 1. Pastikan komputer telah ter-*install* *Python* dan *bottle* (bottlepy.org). Untuk menjalankan di *localhost*, cukup run file RUN.sh seperti berikut:

|  |
| --- |
| ./RUN.sh |

* 1. Buka *browser* dan ketikkan alamat *http://localhost:8003/index.html*.
  2. Kakas siap digunakan.

1. Untuk *deployment* di *server*, berkas-berkas yang dibutuhkan ada di direktori dist.
   1. Letakkan berkas-berkas berikut ini di *server* (misalnya di *Apache*):

|  |
| --- |
| build/main.min.js  example-code/\*  index.html  favicon.ico  doctest\_exec.py  doctest\_reader.py  doctest\_splitter.py  eureka\_survey.py  runtime\_err\_survey.py  syntax\_err\_survey.py  viz\_interaction.py  web\_exec\_c.py  web\_exec\_cpp.py |

* 1. Letakkan berkas backend.js di direktori *server*. *Install* modul *express-js* dengan perintah berikut:

|  |
| --- |
| npm install express |

Untuk *backend server* agar tetap dapat *live*, perlu *install library* *forever* menggunakan npm, caranya sebagai berikut:

|  |
| --- |
| npm install forever  sudo forever backend.js |

* 1. Pastikan *server* telah ter-*install* *docker* seperti pada langkah nomor 2 poin b di atas.
  2. Kakas siap digunakan.

1. **Sistem Operasi Windows**

Untuk konfigurasi di sistem operasi Windows hampir sama dengan Linux/Ubuntu, hanya perbeda di langkah nomor 2 poin c, yaitu harus menjalankan masing-masing *server* di *window* *command prompt* berbeda. Konfigurasi langkah selebihnya sama dengan di sistem operasi Linux/Ubuntu. Perintah yang berbeda untuk dilakukan adalah sebagai berikut:

|  |
| --- |
| python bottle\_server.py  node backend.js  npm run webpack |

Lampiran B Implementasi Kelas *GraphVisualizer*

**Lampiran C Survei Pengguna**

Kuesioner ini dibagi menjadi empat tahap, yaitu:

1. **Tahap 1**: Responden mengisi data pribadi.
2. **Tahap 2**: Responden mengisi pertanyaan atau soal *Pre-Test*. Tujuan tahap ini untuk menggali informasi atau pengetahuan responden sebelum simulasi sehingga hasil survei dapat dibandingkan dengan sebelum menggunakan kakas hasil pengembangan.
3. **Tahap 3**: Responden melakukan simulasi dengan menggunakan kakas OPT dan kakas hasil pengembangan. Responden diminta menyelesaikan soal terkait graf kode program.
4. **Tahap 4**: Responden mengisi pertanyaan atau soal *Post-Test*. Pertanyaan ini dibagi dua, yaitu untuk kakas OPT dan untuk kakas hasil pengembangan sehingga hasilnya dapat dibandingkan.

**C.1 Rancangan Kuesioner *Online***

Pertanyaan *Pre-Test*

|  |  |
| --- | --- |
| **Soal Nomor** | 1 |
| **Tujuan** | Untuk mengetahui perilaku atau kebiasaan pengguna dalam memprogram |
| **Objektivitas** | Proses mengetik kode program di editor kode |
| **Pertanyaan Utama** | Kapan terakhir Anda membuat kode program (*ngoding*) ? |
| **Pertanyaan Kontingensi** | 1. Bahasa Pemrograman apa yang terakhir kali Anda gunakan ? 2. Menurut penilaian Anda, secara keseluruhan berapa tingkat kemahiran Anda dalam memprogram C/C++ ? 3. Apakah Anda pernah belajar Pemrograman Struktur Data ? (seperti penggunaan *Array*, *Struct*, atau *Pointer*) 4. Kapan terakhir Anda membuat kode program struktur data ? 5. Kode program tentang apa yang Anda buat ? |

|  |  |
| --- | --- |
| **Soal Nomor** | 2 |
| **Tujuan** | Untuk menilai tingkat pemahaman pengguna terhadap teori graf yang telah dipelajarinya |
| **Objektivitas** | * Pengguna sudah biasa dengan teori graf yang disampaikan dalam kuesioner ini atau pengguna bingung dengan istilah-istilah graf * Pengguna merasa kesulitan atau tidak dalam menyelesaikan soal-soal terkait graf kode program |
| **Pertanyaan Utama** | Apakah Anda pernah belajar teori tentang Struktur Data Graf ? |
| **Pertanyaan Kontingensi** | 1. Kapan terakhir Anda belajar teori tentang Struktur Data Graf ? 2. Teori Graf apa saja yang pernah Anda pelajari ? 3. Di antara semua teori tersebut, mana yang paling Anda kuasai/pahami ? 4. Di antara semua teori tersebut, mana yang paling Anda tidak mengerti ? 5. Menurut penilaian Anda, berapa tingkat pemahaman Anda tentang teori graf secara keseluruhan ? 6. Mohon jelaskan dengan rinci mengapa Anda merasa kurang dalam memahami teori graf ? Bagian mana yang menjadi kesulitan ? |

**C.2 Daftar Populasi Responden**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Jenjang S1** | | | **Jenjang S2** | | |
| **Tahun 2015** | **Tahun 2016** | **Tahun 2017** | **Tahun 2015** | **Tahun 2016** | **Tahun 2017** |
| 1 | 13515001 | 13516001 | 13517001 | 23515001 | 23516001 | 23517001 |
| 2 | 13515002 | 13516002 | 13517002 | 23515002 | 23516002 | 23517002 |
| 3 | 13515003 | 13516003 | 13517003 | 23515006 | 23516003 | 23517003 |
| 4 | 13515004 | 13516004 | 13517004 | 23515007 | 23516004 | 23517004 |
| 5 | 13515005 | 13516005 | 13517005 | 23515009 | 23516005 | 23517005 |
| 6 | 13515006 | 13516006 | 13517006 | 23515011 | 23516006 | 23517006 |
| 7 | 13515007 | 13516007 | 13517007 | 23515016 | 23516007 | 23517007 |
| 8 | 13515008 | 13516008 | 13517008 | 23515018 | 23516008 | 23517008 |
| 9 | 13515009 | 13516009 | 13517009 | 23515019 | 23516009 | 23517009 |
| 10 | 13515010 | 13516010 | 13517010 | 23515020 | 23516010 | 23517010 |
| 11 | 13515011 | 13516011 | 13517011 | 23515021 | 23516011 | 23517011 |
| 12 | 13515012 | 13516012 | 13517012 | 23515023 | 23516012 | 23517012 |
| 13 | 13515013 | 13516013 | 13517013 | 23515024 | 23516013 | 23517013 |
| 14 | 13515014 | 13516014 | 13517014 | 23515030 | 23516014 | 23517014 |
| 15 | 13515015 | 13516015 | 13517015 | 23515034 | 23516015 | 23517015 |
| 16 | 13515016 | 13516016 | 13517016 | 23515036 | 23516016 | 23517016 |
| 17 | 13515017 | 13516017 | 13517017 | 23515039 | 23516017 | 23517017 |
| 18 | 13515018 | 13516018 | 13517018 | 23515040 | 23516018 | 23517018 |
| 19 | 13515019 | 13516019 | 13517019 | 23515041 | 23516019 | 23517019 |
| 20 | 13515020 | 13516020 | 13517020 | 23515042 | 23516020 | 23517020 |
| 21 | 13515021 | 13516021 | 13517021 | 23515044 | 23516021 | 23517021 |
| 22 | 13515022 | 13516022 | 13517022 | 23515047 | 23516022 | 23517022 |
| 23 | 13515023 | 13516023 | 13517023 | 23515048 | 23516023 | 23517023 |
| 24 | 13515024 | 13516024 | 13517024 | 23515049 | 23516024 | 23517024 |
| 25 | 13515025 | 13516025 | 13517025 | 23515050 | 23516025 | 23517025 |
| 26 | 13515026 | 13516026 | 13517026 | 23515051 | 23516026 | 23517026 |
| 27 | 13515027 | 13516027 | 13517027 | 23515055 | 23516027 | 23517027 |
| 28 | 13515028 | 13516028 | 13517028 | 23515056 | 23516028 | 23517028 |
| 29 | 13515029 | 13516029 | 13517029 | 23515057 | 23516029 | 23517029 |
| 30 | 13515030 | 13516030 | 13517030 | 23515058 | 23516030 | 23517030 |
| 31 | 13515031 | 13516031 | 13517031 | 23515061 |  |  |
| 32 | 13515032 | 13516032 | 13517032 | 23515062 |  |  |
| 33 | 13515033 | 13516033 | 13517033 | 23515063 |  |  |
| 34 | 13515034 | 13516034 | 13517034 | 23515067 |  |  |
| 35 | 13515035 | 13516035 | 13517035 | 23515068 |  |  |
| 36 | 13515036 | 13516036 | 13517036 |  |  |  |
| 37 | 13515037 | 13516037 | 13517037 |  |  |  |
| 38 | 13515038 | 13516038 | 13517038 |  |  |  |
| 39 | 13515039 | 13516039 | 13517039 |  |  |  |
| 40 | 13515040 | 13516040 | 13517040 |  |  |  |
| 41 | 13515041 | 13516041 | 13517041 |  |  |  |
| 42 | 13515042 | 13516042 | 13517042 |  |  |  |
| 43 | 13515043 | 13516043 | 13517043 |  |  |  |
| 44 | 13515044 | 13516044 | 13517044 |  |  |  |
| 45 | 13515045 | 13516045 | 13517045 |  |  |  |
| 46 | 13515046 | 13516046 | 13517046 |  |  |  |
| 47 | 13515047 | 13516047 | 13517047 |  |  |  |
| 48 | 13515048 | 13516048 | 13517048 |  |  |  |
| 49 | 13515049 | 13516049 | 13517049 |  |  |  |
| 50 | 13515050 | 13516050 | 13517050 |  |  |  |

**D.2 Data Hasil Survei Pengguna**

**Lampiran E Contoh Graf Kode Program**

**E.1 Contoh Graf Kode Program dengan Representasi Matriks**

**E.2 Contoh Graf Kode Program dengan Representasi *Array of Adjacency Lists***

**E.3 Contoh Graf Kode Program dengan Representasi *Array of Edges***

1. https://github.com/pgbovine/OnlinePythonTutor [↑](#footnote-ref-1)
2. https://togetherjs.com/ [↑](#footnote-ref-2)
3. https://github.com/pgbovine/OnlinePythonTutor [↑](#footnote-ref-3)
4. https://en.oxforddictionaries.com/definition/visualization [↑](#footnote-ref-4)
5. <http://anvaka.github.io/graph-drawing-libraries/> (diakses 20 Mei 2018) [↑](#footnote-ref-5)