**PENGEMBANGAN KAKAS VISUALISASI**

**DARI GRAF KODE PROGRAM UNTUK**

**MEMAHAMI EKSEKUSI KODE PROGRAM**

**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**

**Januari 2018**

**PENGEMBANGAN KAKAS VISUALISASI**

**DARI GRAF KODE PROGRAM UNTUK**

**MEMAHAMI EKSEKUSI KODE PROGRAM**

Oleh

**Habibie Ed Dien**

**NIM: 23515043**

**(Program Studi Magister Informatika)**

Institut Teknologi Bandung

Menyetujui

Tim Pembimbing

Tanggal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Pembimbing Pertama  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Dr. M.M. Inggriani Liem) | Pembimbing Kedua  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Yudistira Dwi W. Asnar, S.T., Ph.D.) |

# DAFTAR ISI

HALAMAN PENGESAHAN i

[DAFTAR ISI ii](#_Toc504114434)

[DAFTAR GAMBAR DAN ILUSTRASI iv](#_Toc504114435)

[DAFTAR TABEL v](#_Toc504114436)

[Bab I Pendahuluan 1](#_Toc504114437)

[I.1 Latar Belakang 1](#_Toc504114438)

[I.2 Rumusan Masalah 3](#_Toc504114439)

[I.3 Tujuan Penelitian 3](#_Toc504114440)

[I.4 Batasan Masalah 3](#_Toc504114441)

[I.5 Metodologi Penelitian 3](#_Toc504114442)

[I.6 Sistematika Penulisan 5](#_Toc504114443)

[Bab II Tinjauan Pustaka dan Eksplorasi 6](#_Toc504114444)

[II.1 Terminologi Visualisasi Perangkat Lunak 6](#_Toc504114445)

[II.1.2 Sistem dan Manfaat Visualisasi Program (VP) 7](#_Toc504114446)

[II.2 Metodologi dan Prinsip Visualisasi Data 7](#_Toc504114447)

[II.3 Teori Desain Interaksi 9](#_Toc504114448)

[II.4 Eksplorasi Kakas Visualisasi Program untuk Graf 11](#_Toc504114449)

[II.4.1 *Swan* 11](#_Toc504114450)

[II.4.2 *VisMod* 13](#_Toc504114451)

[II.4.3 *jGRASP* 14](#_Toc504114452)

[II.4.4 *Jype* 17](#_Toc504114453)

[II.4.5 *Online Python Tutor* 18](#_Toc504114454)

[II.5 Kesimpulan Awal Berdasarkan Tinjauan Pustaka dan Eksplorasi 25](#_Toc504114455)

[Bab III Analisis Masalah 27](#_Toc504114456)

[III.1 Analisis Desain Interaksi Visualisasi Graf 27](#_Toc504114457)

[III.1.1 Analisis Spesifikasi Kebutuhan Pengguna 27](#_Toc504114458)

[III.1.2 Analisis Pengembangan Desain 28](#_Toc504114459)

[III.1.3 Membangun Purwarupa 31](#_Toc504114460)

[III.1.4 Evaluasi Selama Proses Pengembangan Desain 31](#_Toc504114461)

[III.2 Analisis Deteksi Graf dalam Kode Program 33](#_Toc504114462)

[III.3 Analisis Kebutuhan Perangkat 37](#_Toc504114463)

[III.3.1 Kebutuhan Perangkat Lunak 38](#_Toc504114464)

[III.3.2 Kebutuhan Perangkat Keras 38](#_Toc504114465)

[Bab IV Perancangan dan Implementasi Kakas 40](#_Toc504114466)

[IV.1 Proses Konstruksi Visualisasi Data 40](#_Toc504114467)

[IV.2 Perbaikan Arsitektur Kakas 48](#_Toc504114468)

[IV.3 Implementasi Kakas 50](#_Toc504114469)

[Bab V Pengujian dan Evaluasi Kakas 51](#_Toc504114470)

[V.1 Pengujian Kakas 51](#_Toc504114471)

[V.1.1 Pengujian Fungsional Kakas 51](#_Toc504114472)

[V.1.2 Desain Eksperimen 51](#_Toc504114473)

[V.2 Evaluasi Kakas 51](#_Toc504114474)

[V.2.1 Tujuan Pengembangan Kakas 51](#_Toc504114475)

[V.2.2 Indikator Keberhasilan Kakas 51](#_Toc504114476)

[V.2.3 Perbandingan Kakas Lama dan Baru 51](#_Toc504114477)

[V.2.4 Kelebihan dan Kekurangan Kakas 51](#_Toc504114478)

[Bab VI Kesimpulan dan Saran 52](#_Toc504114479)

[VI.1 Kesimpulan 52](#_Toc504114480)

[VI.2 Saran 52](#_Toc504114481)

[DAFTAR PUSTAKA 53](#_Toc504114482)

# DAFTAR GAMBAR DAN ILUSTRASI

[Gambar II.1 Keterhubungan antar tujuh tahapan (Fry, 2008) 8](#_Toc504114523)

[Gambar II.2 Konseptual model komponen interaksi (Fry, 2008) 10](#_Toc504114524)

[Gambar II.3 *Swan*: Dua visualisasi graf dengan algoritma *minimum spanning tree* (Shaffer dkk., 1996) 12](#_Toc504114525)

[Gambar II.4 *Swan*: *Binary search tree* 13](#_Toc504114526)

[Gambar II.5 *VisMod*: Visualisasi menghapus elemen dari *list* 14](#_Toc504114527)

[Gambar II.6 *jGRASP*: Tampilan ketika *node* dibuat dan ditambahan ke *root binary tree* (Cross II dkk., 2007) 15](#_Toc504114528)

[Gambar II.7 *jGRASP*: *Visualisasi* pohon dari struktur data *TreeMap* (Hendrix dkk., 2004) 16](#_Toc504114529)

[Gambar II.8 *Jype*: visualisasi *binary search tree* dari tipe data *array* 17](#_Toc504114530)

[Gambar II.9 Turunan kelas visualisasi Matrix Framework (Korhonen dkk., 2004) 18](#_Toc504114531)

[Gambar II.10 Tampilan Antarmuka OPT untuk Bahasa Pemrograman C 19](#_Toc504114532)

[Gambar II.11 Tampilan Antarmuka OPT untuk Kolaborasi Pemrograman 19](#_Toc504114533)

[Gambar II.12 Tampilan simulasi dan visualisasi kode program *C pointer* 20](#_Toc504114534)

[Gambar II.13 Arsitektur OPT untuk visualisasi kode program C dan C++ 21](#_Toc504114535)

[Gambar II.14 Ilustrasi Sederhana Proses Kakas OPT 22](#_Toc504114536)

[Gambar II.15 Ilustrasi format *trace* eksekusi kode program 23](#_Toc504114537)

[Gambar II.16 Peran Utama *D3JS* sebagai *Framework* Visualisasi 24](#_Toc504114538)

[Gambar III.1 Visual graf berarah dengan dua *node* dan *edge* 28](#_Toc504114539)

[Gambar III.2 OPT: Visualisasi graf dengan *array* dimensi-2 (matriks) 30](#_Toc504114540)

[Gambar III.3 OPT: Visualisasi graf dengan *struct* dan *pointer* 30](#_Toc504114541)

[Gambar III.4 Purwarupa visualisasi graf dengan pustaka *D3JS* 31](#_Toc504114542)

[Gambar III.5 Contoh data matriks sebagai representasi graf 33](#_Toc504114543)

[Gambar III.6 Contoh data *array* atau *edge list* 33](#_Toc504114544)

[Gambar III.7 Contoh data *adjacency list* dengan indeks *node* 34](#_Toc504114545)

[Gambar III.8 OPT: diagram alir proses perolehan data *JSON* 35](#_Toc504114546)

[Gambar III.9 OPT: Format data eksekusi *trace JSON* 35](#_Toc504114547)

[Gambar III.10 Contoh data *JSON pointer* pada atribut *heap* 36](#_Toc504114548)

[Gambar IV.1 Klasifikasi data *JSON* menjadi dua bagian 43](#_Toc504114549)

[Gambar IV.2 Klasifikasi dari data matriks atau *pointer* 43](#_Toc504114550)

[Gambar IV.3 Contoh data *JSON pointer* yang telah di-*filter* 44](#_Toc504114551)

[Gambar IV.4 Diagram alur *level-1* pencocokan model secara umum 45](#_Toc504114552)

[Gambar IV.5 Diagram alur *level-0* proses visualisasi data 46](#_Toc504114553)

[Gambar IV.6 Diagram alur *level-1* proses *rendering* visualisasi graf 46](#_Toc504114554)

[Gambar IV.7 Visual graf: (a) visual dasar graf; (b) setelah perbaikan 47](#_Toc504114555)

[Gambar IV.8 (a) Perbaikan desain interaksi visual; (b) Fitur animasi 48](#_Toc504114556)

[Gambar IV.9 Arsitektur kode program kakas OPT 48](#_Toc504114557)

[Gambar IV.10 Arsitektur kakas OPT yang telah diperbaiki 49](#_Toc504114558)

# DAFTAR TABEL

[Tabel III.1 Daftar properti visual graf 28](#_Toc504114600)

[Tabel III.2 Format data graf *JSON* 31](#_Toc504114601)

[Tabel III.3 Variabel *‘a’* berupa matriks berdimensi 7 x 7 36](#_Toc504114602)

[Tabel III.4 Isi alamat memori pada objek *heap* 37](#_Toc504114603)

[Tabel III.5 Kakas dan *library* pendukung pengembangan OPT 38](#_Toc504114604)

[Tabel IV.1 Contoh data eksekusi *trace* *JSON* berupa matriks 40](#_Toc504114605)

[Tabel IV.2 Daftar label untuk klasifikasi data *JSON* 42](#_Toc504114606)

[Tabel IV.3 Format dasar yang digunakan untuk visualisasi graf dengan *D3JS* 45](#_Toc504114607)

[Tabel IV.4 Format untuk *edge* dengan bobot 45](#_Toc504114608)

# 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* 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 *D3JS* (Bostock dkk., 2011) sebagai teknologi utama untuk visualisasi. Namun, kakas ini 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, maka diperoleh rumusan masalah sebagai berikut:

1. Desain interaksi apa yang sesuai untuk visualisasi graf sehingga dapat digunakan untuk memahami eksekusi graf kode program ?
2. Apa strategi pengembangan kakas untuk mendeteksi graf di dalam eksekusi kode program sehingga dapat divisualisasikan ?

## I.3 Tujuan Penelitian

Tujuan dari penelitian tesis ini adalah untuk menghasilkan modifikasi kakas OPT yang dapat melakukan visualisasi graf dari eksekusi kode program.

## I.4 Batasan Masalah

Batasan masalah pada tesis ini adalah :

1. Permasalahan yang dikaji lebih fokus terhadap desain interaksi dan visualisasi untuk memahami eksekusi graf kode program, tidak terkait pedagogi pemrograman.
2. Ukuran kode program yang digunakan sebagai *input* maksimal 500 baris.
3. Platform yang digunakan adalah web adaptif.
4. Batasan ukuran layar minimal 800 × 600 *pixels*.
5. Bahasa pemrograman yang dapat divisualisasi adalah C dan C++.
6. Kakas tidak menerima *standard input* (*stdin*) dan tidak menerima deklarasi *header file* dari pengguna, artinya hanya *standard library C/C++* yang sudah tersedia di *compiler server*.

## 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 Graf Kode Program

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

1. Analisis Desain Interaksi Visualisasi

Menganalisis spesifikasi kebutuhan desain interaksi untuk visualisasi agar proses memahami eksekusi graf kode program dapat tercapai.

1. Analisis Deteksi Graf dalam Kode Program

Menganalisis kode program yang terdapat algoritma graf agar dapat divisualisasikan, sehingga diperoleh suatu metode yang relevan.

1. Perancangan Kakas

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

1. Eksperimen

Tahap ini melakukan simulasi visual graf menggunakan data *dummy* dan pustaka *D3JS* agar sesuai dengan hasil analisis desain interaksi.

1. Implementasi Kakas

Implementasi kakas berdasarkan rancangan yang digabung dengan hasil eksperimen visual graf.

1. Pengujian Kakas

Hasil pengembangan kakas diuji dengan memasukkan berbagai jenis kode program dan tipe data, kemudian menganalisis hasil dari visualisasi tersebut.

1. Evaluasi Kakas

Kakas perlu dievaluasi secara empiris dengan bantuan responden untuk menilai efektivitas kakas dalam representasi visual graf. Responden akan mencoba menggunakan kakas sebelum pengembangan, kemudian membandingkan 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.

1. Bab II Tinjauan Pustaka dan Eksplorasi

Bab ini berisi uraian tentang terminologi visualisasi perangkat lunak, 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 Pengujian dan Evaluasi Kakas

Bab ini menguraikan proses dan hasil pengujian serta evaluasi dari kakas yang 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 peneliti 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” telah lama berkembang dan didefinisikan sebagai sebuah seni tipografi, desain grafis, animasi, dan sinematografi melalui interaksi modern antar manusia-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 datanya.

### II.1.2 Sistem dan Manfaat Visualisasi Program (VP)

Visualisasi program (VP) terdiri dari pembuatan animasi eksekusi kode program. Animasi ini membuat hasil visual yang sangat mengesankan. Hal ini tidak terbatas pada algoritma atau aktivitas dari eksekusi kode program. Beberapa aktivitas yang mungkin terjadi saat kode program dikompilasi adalah seperti sistem *run-time*, 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 *programer* 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 konseptual model (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. ­­Konseptual model 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. Memroses 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, 2002), 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 apa yang sedang dibangun selama proses pengembangan desain berlangsung.

Keempat tahap tersebut saling berkaitan dan harus selalu diulang untuk memastikan desain yang dibangun telah memenuhi syarat dan kebutuhan. Sebagai contoh, kegunaan (*usability*) suatu produk yang telah dibangun dapat diukur dengan memperhatikan setiap perbaikan yang telah dilakukan atau dengan melihat persyaratan tertentu 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.

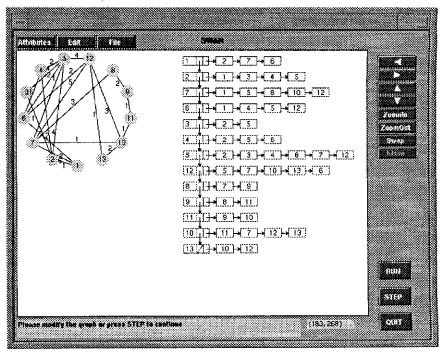
## II.4 Eksplorasi Kakas Visualisasi Program untuk Graf

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

### II.4.1 *Swan*

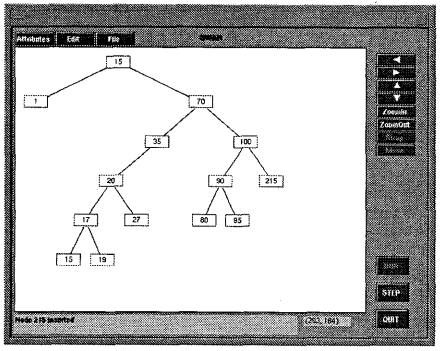
Swan (Shaffer dkk., 1996) adalah kakas VP untuk struktur data dan eksekusi kode program C/C++. Kakas ini dapat memvisualisasikan graf, pohon, *list*, dan *array*. Metodenya dengan menggunakan anotasi pada kode program yang disebut SAIL (*Swan Annotation Interface Library*). Tujuan utama kakas Swan adalah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

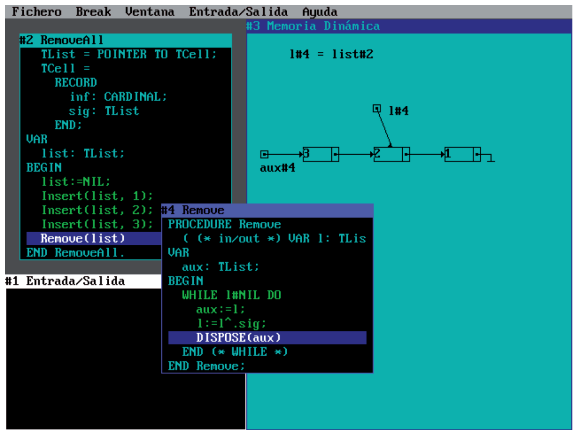
Sebuah *node* atau *edge* yang tampil di *display window* dapat dieksplor. Aksi seperti *pan* dan *zoom* dapat dilakukan pada visualisasi graf; dapat mengubah keadaan *RUN* atau *Step*; dan dapat memodifikasi atribut visual graf. Modifikasi yang dapat dilakukan untuk atribut *node* terdiri dari bentuk, 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*. Sehingga pengguna dapat fokus ke bagian kode program yang menjadi perhatian khusus.

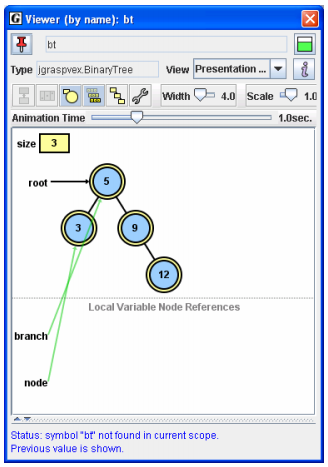
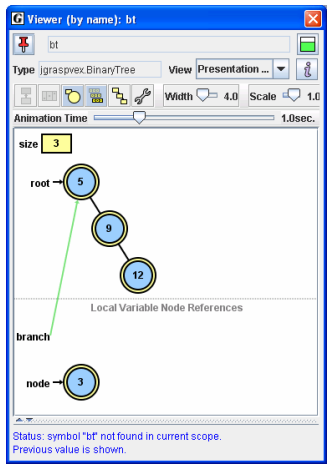


1. *VisMod*: Visualisasi menghapus elemen dari *list* (Jiménez-Peris dkk., 1999)

### 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.



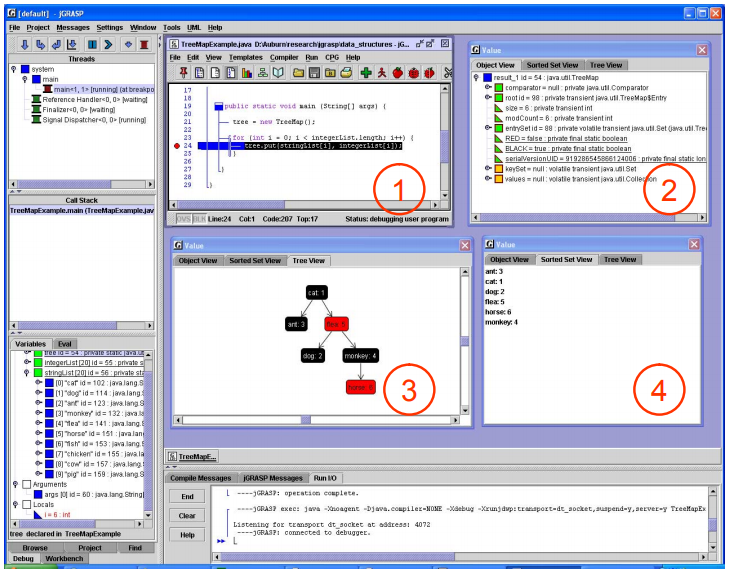
1. *jGRASP*: Tampilan ketika *node* dibuat dan ditambahan ke *root binary tree* (Cross II dkk., 2007)

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. Sehingga 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*.

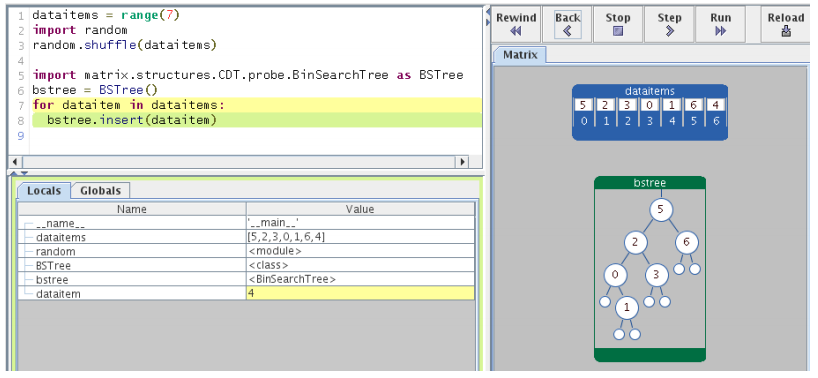
Kakas *jGRASP* tidak mendukung fitur *reverse execution*, yaitu untuk kembali ke langkah eksekusi sebelumnya (Gestwicki dan Jayaraman, 2005), sehingga tidak dapat dilakukan analisis dengan membandingkan kode program sebelum dan sesudah eksekusi pada baris kode program selanjutnya.



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

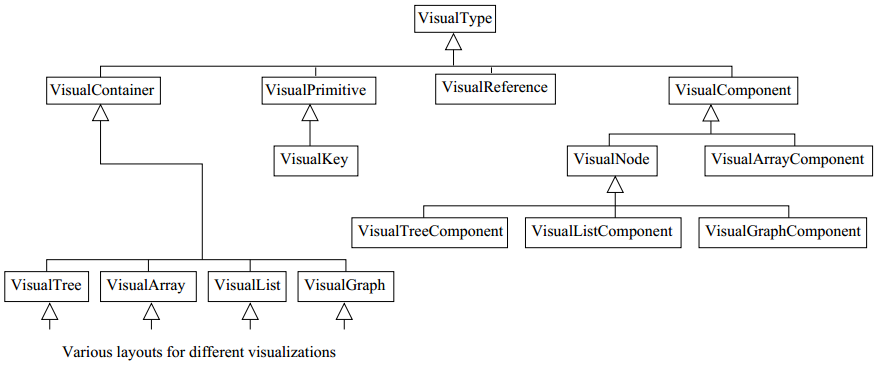
### II.4.4 *Jype*

*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 berikut ini.



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

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

### 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 kode pustaka dari *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 tertentu yang dipaketkan berbentuk *JSON*. *OPT frontend* (antarmuka *browser*) menerjemahkan *JSON* menjadi visualisasi yang dikombinasikan dengan pustaka *D3JS*. 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 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 tertentu 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 Pustaka *Data-Driven Documents* (D3)

Teknologi visualisasi utama yang digunakan oleh OPT adalah *Data-Driven Documents* (D3JS) (Bostock dkk., 2011) yang merupakan salah satu pustaka terpopuler untuk implementasi visualisasi berbasis web. Format *trace* eksekusi program dibaca oleh *D3JS* untuk mendeklarasikan pemetaan data dan atribut yang diperlukan oleh elemen *Document Object Model* (DOM) pada laman *Hyper Text Markup Language* (HTML). Peran *D3JS* dapat dilihat pada Gambar II.16 berikut ini.



1. Peran Utama *D3JS* sebagai *Framework* Visualisasi

Pustaka *D3JS* sangat fleksibel terhadap abstraksi data tingkat tinggi. Pustaka 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, mayoritas 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** | 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 visual 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 visual 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 interaksi visual.

# Bab III Analisis Masalah

Bab ini menganalisis masalah kakas VP yang akan dikembangkan, selanjutnya dilakukan penentuan ruang lingkup pengembangan kakas terhadap proses visualisasi graf dan peluang pemecahan masalah.

## III.1 Analisis Desain Interaksi Visualisasi Graf

Tahapan analisis spesifikasi kebutuhan desain interaksi untuk visualisasi graf terdiri dari empat tahap sesuai yang telah dibahas pada subbab II.3, yaitu:

### III.1.1 Analisis Spesifikasi Kebutuhan Pengguna

Tahap ini berguna untuk menganalisis kebutuhan pengguna sehingga desain dapat disesuaikan dengan tujuan yang ingin dicapai. Tujuan yang ingin dicapai oleh pengguna adalah dapat memahami proses 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.

Ada dua tujuan utama mengapa diperlukan identifikasi spesifikasi kebutuhan pengguna (Preece dkk., 2015). Pertama, untuk memahami tentang perilaku pekerjaan pengguna terhadap suatu sistem atau produk yang sedang dikembangkan sehingga bisa tercapai kebutuhan pengguna. Kedua, untuk menghasilkan seperangkat kebutuhan dasar yang akan menjadi acuan desain interaksi. Desain akan menyesuaikan spesifikasi kebutuhan yang telah disusun.

Berdasarkan studi literatur dan eksplorasi kakas, diperoleh beberapa spesifikasi kebutuhan pengguna untuk membuat desain interaksi visualisasi, yaitu:

1. Eksekusi graf kode program harus dapat tampil visual graf;
2. Operasi tambah dan hapus *node* dapat dilakukan dari eksekusi kode program;
3. Animasi pencarian *node* dapat divisualisasi sesuai dari eksekusi kode program.

### III.1.2 Analisis Pengembangan Desain

Pada tahap ini dibutuhkan untuk menganalisis desain apa saja yang diperlukan untuk memenuhi kebutuhan pengguna. Selain itu, tujuan visualisasi harus dapat tercapai dengan desain yang telah dirancang. Ada dua jenis desain, yaitu konseptual dan konkrit (Preece dkk., 2015). Desain konseptual adalah proses untuk mengubah spesifikasi kebutuhan pengguna menjadi model konseptual. Model konseptual merupakan ringkasan terhadap perilaku pengguna yang dapat dilakukan dengan suatu sistem atau produk, serta konsep apa saja yang diperlukan untuk dapat saling berinteraksi. Sedangkan desain konkrit lebih menekankan kepada implementasi yang sebenarnya. Desain konkrit adalah tahapan sebelum purwarupa suatu produk atau sistem dibangun. Salah satu aspek dalam desain konkrit adalah tampilan visual, seperti pewarnaan, desain *icon*, *button*, tata letak, dan interaksi yang dipilih.

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 berbagai jenis 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* | atau atau |
| 3 | Bentuk *edge* | atau atau |
| 4 | Label *edge* | x  x  atau |
| 5 | Warna *node* dan *edge* | atau  atau |
| 6 | Ketebalan *edge* | atau |

Eksplorasi kakas OPT dilakukan dengan meninjau proses kerja sistem, alur kode program dan membaca baris-baris kode pengembangnya. Versi terkahir yang dikembangkan telah bermigrasi ke bahasa pemrograman *Typescript*. Pengembang berargumen agar dapat dikembangkan lebih lanjut dengan teknik modularisasi[[4]](#footnote-4). Ini bermanfaat ketika peneliti atau pengembang lain yang ingin melanjutkan dengan penambahan fitur dapat dilakukan dengan mudah.

Pada Gambar III.2 terlihat bahwa untuk merepresentasikan graf dilakukan dengan berupa tabel data larik atau matriks. Gambar tersebut merupakan salah satu contoh graf kode program dengan algoritma *Breadth First Search*. Representasi berupa matriks dapat ditingkatkan dengan visualisasi graf.

Selain itu, pada Gambar III.3 *pointer* direpresentasikan dengan garis panah untuk relasi antar simpul. Representasi tersebut menjadi lebih rumit ketika data semakin bertambah dalam visualisasi. Hal ini membuat pengguna untuk menelusuri kode dengan kecocokan visualnya menjadi lebih rumit. Alternatif desain bisa menyembunyikan data yang tidak diperlukan, seperti data yang sudah diproses pada langkah sebelumnya. Jika data ini berisi *node* dan *edge*, maka dapat divisualisasikan sebagai graf.

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



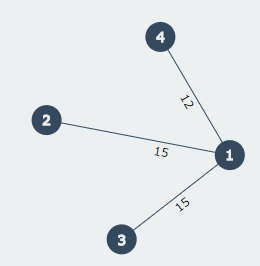
1. OPT: Visualisasi graf dengan *array* dimensi-2 (matriks)



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

### III.1.3 Membangun Purwarupa

Tahap ini dilakukan eksperimen visualisasi graf dengan pustaka *D3JS* menggunakan data *dummy*. Eksperimen ini dilakukan untuk menyesuaikan dengan kebutuhan desain interaksi (lihat Gambar III.4). Data *dummy* sudah sesuai dengan format *JSON* yang digunakan oleh *D3JS* untuk visualisasi. Sehingga data yang diperlukan untuk dapat memvisualisasikan graf harus dikonversi ke format ini terlebih dahulu.



1. Purwarupa visualisasi graf dengan pustaka *D3JS*

Format yang digunakan oleh pustaka *D3JS* untuk menampilkan visualisasi graf ada pada Tabel III.2. Atribut nodes mewakili data *node* dan teks labelnya. Sedangkan atribut edges mewakili data *edge* yang menghubungkan antar *node* dengan alamat indeks. Alamat indeks *node* dimulai dengan nilai nol, bukan satu. Jadi, untuk mengakses *node* “1” dalam *edge*, harus mengakses nilai 0.

1. Format data graf *JSON*

|  |
| --- |
| {  **nodes**:[{name: “1”}, {name: “2”}, {name: “3”}, ... ]  ,**edges**:[{source: 0, target: 1}, {source: 0, target: 2}, ...]  } |

### III.1.4 Evaluasi Selama Proses Pengembangan Desain

Evaluasi dikategorikan menjadi tiga bagian (Preece dkk., 2015), tergantung skenario, keterlibatan pengguna, dan tingkat kontrol. Kategori evaluasi yaitu:

1. Skenario yang dikontrol dan melibatkan pengguna. Contoh, laboratorium dan lingkungan di dalam. Aktivitas pengguna dapat dikontrol dengan menguji hipotesis dan diamati perilakunya. Metode utama untuk kategori ini adalah pengujian penggunaan dan eksperimen.
2. Skenario secara alami dan melibatkan pengguna. Contoh, komunitas online dan produk yang digunakan di tempat umum. Kontrol terhadap aktivitas pengguna hampir tidak ada, karena produk yang digunakan dalam dunia nyata bisa saja sembarang digunakan. Metode utama untuk kategori ini adalah studi lapangan.
3. Skenario tanpa melibatkan pengguna. Contoh, pekerjaan konsultan, prediksi peneliti, dan model yang dibuat dengan analisis masalah dalam penggunaan. Metode yang dapat digunakan untuk kategori ini adalah inspeksi, heuristik, membuat pedoman, model, dan analisis.

Analisis masalah awal menggunakan evaluasi kategori ketiga, yaitu skenario tanpa melibatkan pengguna. Tekniknya dengan membuat model dan menganalisis kesesuaian dengan kebutuhan pengguna. Selanjutnya, evaluasi akan dilakukan pada kategori pertama, yaitu melibatkan beberapa pengguna untuk dilakukan evaluasi dan prosedur pengujian. Evaluasi terakhir dengan skenario secara alami yang melibatkan banyak pengguna dengan menyebarkan situs aplikasi web.

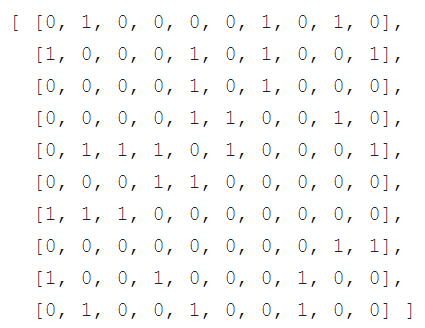
Ada beberapa pertanyaan yang perlu dipertimbangkan dalam pengembangan desain ini, yaitu:

1. Bagaimana tujuan visualisasi dapat tercapai?
2. Apa yang ingin dipengaruhi melalui visualisasi kepada pengguna?
3. Apakah pengguna akan lebih cepat memahami eksekusi graf kode program dengan visualisasi dibanding tanpa visualisasi?
4. Bagaimana dengan data yang tidak bisa divisualisasi? Misalnya karena ada duplikasi *node*.
5. Bagaimana dengan nama *node* dengan karakter yang panjang?
6. Bagaimana dengan graf berarah yang memiliki dua *node* yang tersusun melingkar (siklik)? Apakah garis *edge* harus menyesuaikan agar tidak saling menindih?

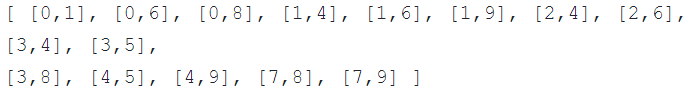
## III.2 Analisis Deteksi Graf dalam Kode Program

Analisis ini bertujuan untuk mendapatkan solusi yang mungkin dapat diimplementasi. Untuk menganalisis teknik pendeteksian graf dalam kode program, perlu diketahui cara representasinya dalam kode program. Representasi graf dalam kode program dapat dilakukan dengan tiga cara, yaitu sebagai berikut (Sedgewick dan Wayne, 2011):

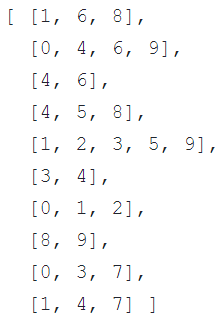
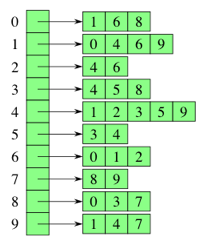
1. Representasi *adjacency matrix* atau berupa data matriks yang simetris dengan 2-dimensi *array* (lihat Gambar III.5), yaitu baris dan kolom sebagai *node*. Baris dan kolom yang ada *edge*, diberi nilai 1 atau *true*, jika kosong maka diberi nilai 0 atau *false*. Jika matriks tidak simetris, maka direpresentasikan 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.6). 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.7) 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.

Setelah mengetahui ketiga cara representasi graf tersebut, proses selanjutnya adalah mencari ciri data yang cocok dengan ketiga representasi tersebut. Pengembangan kakas ini berdasarkan kode dari OPT. Proses untuk menghasilkan data *JSON* dapat dilihat pada Gambar III.8. *Server* meng-*compile* kode program dari pengguna dengan kakas *Valgrind debugger*. Proses ini telah dibahas pada subbab II.5.5.2. Pada bagian label “Ekstraksi data graf” terdapat proses untuk mencari data graf. Jika ada yang cocok, maka proses ekstraksi dilakukan sehingga mendapatkan data yang sesuai dengan format pada Tabel III.2.



Ekstraksi

data graf

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



Ekstraksi

data graf

1. OPT: Format data eksekusi *trace JSON*

Data eksekusi *trace* *JSON* adalah sumber data yang digunakan untuk visualisasi. Salah satu contoh data *JSON* berupa matriks yang dihasilkan dari Gambar III.9 dapat dilihat pada Tabel III.3 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.9).

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  ],  [...]  ] |

Sedangkan untuk data *JSON* dari Gambar III.3 dapat dilihat pada Gambar III.10. Objek *heap* ini berisi alamat memori yang digunakan oleh *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-poin.



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

Pada Tabel III.4 dan Gambar III.10 menunjukkan hasil data *JSON trace* yang akan dilakukan proses visualisasi data sesuai pada subbab II.2 tentang tujuh tahapan proses konstruksi visualisasi data. Data *JSON* dapat dibagi menjadi dua kategori utama untuk representasi graf dalam kode, yaitu (1) bersumber dari tabel informasi berupa matriks atau *n-*dimensi *array*, dan (2) berupa *struct* yang berisi *pointer*.

1. 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 tertentu 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*.

Fitur-fitur tersebut merupakan kunci utama, selain harus dicari pola visual lain seperti variabel pendukung lain yang sedang digunakan dalam kode program.

## III.3 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.3.1 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.

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 | D3JS | 2.0 | Pustaka *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.3.2 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 jika memroses sekitar 500 baris kode untuk dapat menghasilkan data *trace* eksekusi *JSON*.

# Bab IV Perancangan dan Implementasi Kakas

Bab ini menjelaskan proses pengembangan kakas yang terdiri dari proses visualisasi data, perbaikan arsitektur kakas dan implementasi kakas. Salah satu tujuan pengembangan kakas ini selain untuk dukungan visualisasi graf adalah dapat dikembangkan dengan modularisasi. Bersifat *modular* akan bermanfaat untuk penelitian dan penambahan fitur-fitur selanjutnya.

## IV.1 Proses Konstruksi Visualisasi Data

Tujuh tahap berikut menjelaskan metode konstruksi visualisasi dari sumber perolehan data hingga perancangan desain interaksi. Teori tahapan ini telah dibahas pada subbab II.2 tentang metodologi dan prinsip visualisasi data. Sebelum mengembangkan kakas, diperlukan ketujuh tahapan ini 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 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.1), 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 atau lihat Gambar IV.2:

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 ceklist 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

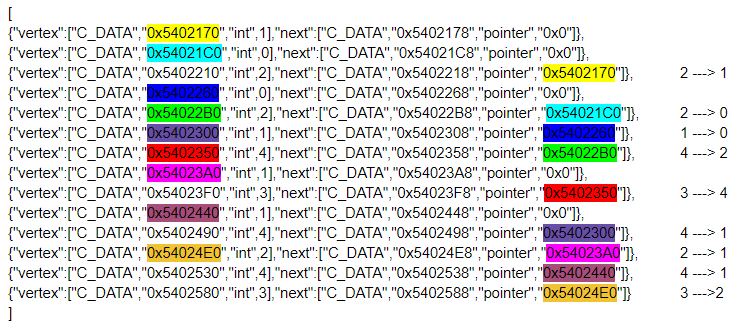


1. Klasifikasi dari data matriks atau *pointer*
2. *Filter* (penyaringan data)

Proses ini mengambil fitur data yang dibutuhkan dan mengabaikan data lain.

Karena dalam *JSON* tersebut memiliki banyak jenis data yang tidak berkaitan dengan terbentuknya visual graf. Untuk menguji keberadaan fitur data, 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.3. 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.3 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. Seperti yang telah dijelaskan pada subbab II.4 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.4 terlihat diagram alur *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 *D3JS* 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 alur *level-1* pencocokan model secara umum
2. *Represent* (representasi)

Pada bagian ini proses representasi data menggunakan pustaka *D3JS*. Pustaka *D3JS* membantu dalam proses visualisasi data. Format dasar data seperti pada Tabel IV.3 dapat digunakan oleh *D3JS* untuk visual graf adalah sebagai berikut:

1. Format dasar yang digunakan untuk visualisasi graf dengan *D3JS*

|  |
| --- |
| { **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.5. Pada proses “Pencocokan Model” telah dilakukan pada langkah sebelumnya (pada langkah 4 *Mine* atau menggali informasi). Selanjutnya proses ini adalah visualisasi menggunakan pustaka *D3JS*. Proses *rendering* visualisasi dapat dilihat pada Gambar IV.6.



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



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

Bagian ini merupakan perbaikan desain visual dari representasi dasar seperti pada Gambar IV.7 (a). Perbaikan visual (lihat Gambar IV.7 (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.8 (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.8 (b). Operasi yang dilakukan dalam program biasanya seperti pencarian jalur terdekat.

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

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

## IV.2 Perbaikan Arsitektur Kakas

Perbaikan arsitektur kakas dilakukan karena saat mulai pengembangan kakas mengalami kesulitan untuk membaca alur sistemnya. Pada Gambar IV.9 terlihat struktur berkas, kelas (fungsi dan variabel yang di-*import* atau *export* sengaja tidak digambar karena akan nampak semakin rumit) dan objek yang tidak rapi. Anak panah menunjukkan ada fungsi atau variabel yang di-*export*. Selain itu, berkas dokumentasi yang terbatas, penamaan berkas dan variabel dalam kode kurang efektif. Oleh karena itu, sebelum mengembangkan kakas, proses perbaikan ini diharapkan dapat memudahkan dalam penambahan fitur yang akan dikembangkan.



1. Arsitektur kode program kakas OPT

Modularisasi memudahkan pengembang untuk memanajemen fitur-fitur yang akan dikembangkan. Alur sistem yang terstruktur membantu proses analisis kebutuhan sistem. Pada Gambar IV.10 adalah arsitektur kakas OPT yang telah diperbaiki. Masing-masing penamaan berkas (ekstensi *.ts*) mewakili kelas yang ada didalamnya.



1. Arsitektur kakas OPT yang telah diperbaiki

Pada Gambar IV.10 tersebut terdapat kelas *GraphViz* (kotak berwarna merah), yaitu penambahan fitur untuk visual graf yang sedang dikembangkan dalam penelitian ini. Kelas ini berisi beberapa fungsi, yang utama terdapat fungsi untuk pencocokan model visualisasi dan proses *rendering* visual graf. Berkas lain dimodifikasi seperlunya terkait perbaikan visual maupun yang ada hubungannya dengan proses kelas *GraphViz*.

## IV.3 Implementasi Kakas

Secara garis besar, kakas OPT belum dapat memvisualisasikan graf. Pengembangan kakas ini diharapkan dapat menambah fitur untuk visualisasi graf saat pengguna mengeksekusi kode sumber yang terdapat algoritma graf. Tujuannya adalah agar membantu pengguna untuk memahami eksekusi kode graf. Metode pengembangan kakas yang dilakukan adalah sebagai berikut:

1. Lakukan eksperimen untuk merepresentasikan algoritma graf dan pohon yang ada di **Error! Reference source not found.** dan **Error! Reference source not found.** dalam bentuk visual dengan D3JS
2. Visual pada poin 1 dibangun animasi per tahap sesuai alur algoritma
3. Mencari pola pembentukan visualnya berdasarkan kode C, C++ dan *execution trace format*. Jika ada *array* atau *pointer* dalam kode program, perlu dipertimbangkan untuk menampilkan visualisasi graf
4. Untuk mengetahui cara kerja kakas terbentuknya visual OPT diperlukan sebuah abstraksi dengan membuat diagram kelas.
5. Integrasikan model visualisasi dengan OPT.

+ diagram class

+ interface dan implementation

+

# Bab V Pengujian dan Evaluasi Kakas

Bab V ini berisi pengujian dan evaluasi terhadap hasil pengembangan kakas. Pengujian selain fungsional kakas, juga akan dilakukan eksperimen terhadap pelajar sebagai penggunanya. Kemudian hasil pengujian dianalisis untuk dievaluasi terhadap tujuan pengembangan kakas yang diharapkan.

## V.1 Pengujian Kakas

### V.1.1 Pengujian Fungsional Kakas

### V.1.2 Desain Eksperimen

Desain eksperimen kakas dibagi menjadi dua bagian, yaitu melalui eksperimen simulasi mengajar atau studi kasus dan wawancara mendalam. Dalam eksperimen simulasi mengajar, dilakukan dua uji kasus yaitu pelajar menggunakan OPT dan pelajar menggunakan kakas hasil pengembangan visual graf.

## V.2 Evaluasi Kakas

### V.2.1 Tujuan Pengembangan Kakas

### V.2.2 Indikator Keberhasilan Kakas

### V.2.3 Perbandingan Kakas Lama dan Baru

### V.2.4 Kelebihan dan Kekurangan Kakas

# Bab VI Kesimpulan dan Saran

## VI.1 Kesimpulan

## VI.2 Saran

+ jika bisa compile di komputer client, akan lebih memudahkan

+ proses visualisasi graf mungkin bisa dipadukan dengan metode machine learning

# 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. and Heer, J. (2011): D$^3$ data-driven documents, *IEEE Trans. Vis. Comput. Graph.*, **17**, 2301–2309.

Cetin, I. and 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. and 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. and 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. and 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 (accessed 20 July 2016).

Hendrix, T.D., Cross II, J.H. and 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. and 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 (accessed 15 September 2017).

Jiménez-Peris, R., Patiño-Martínez, M. and 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 (accessed 31 December 2017).

Korhonen, A., Malmi, L., Silvasti, P., Karavirta, V., Lönnberg, J., Nikander, J., Stålnacke, K. and 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.

Piteira, M. and 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 (accessed 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. and Rogers, Y. (2015): *Interaction Design, beyond human-computer interaction Fourth Edition*, 4th edition, John Wiley & Sons, Ltd.

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

Shaffer, C.A., Heath, L.S. and 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.

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

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://github.com/pgbovine/OnlinePythonTutor [↑](#footnote-ref-4)