**Implementasi VoG (*Vocabulary* based summarization of *Graph*) Pada *Web* *Graph***

***Implementation of VoG (Vocabulary based summarization of Graph) on the Web Graph***

**Tugas Akhir**

**Kelompok Keahlian : SIDE**

**Satrio Adityo Hartomo**

**1103120029**

****

**Program Studi Teknik Informatika**

**Fakultas Informatika**

**Universitas Telkom**

**Bandung**

**2016**

# LEMBAR PERNYATAAN

Dengan ini saya menyatakan bahwa tugas akhir dengan judul **“Implementasi VoG (*Vocabulary* based summarization of *Graph*) pada *Web* *Graph*”** dan seluruh isinya benar-benar merupakan karya saya sendiri dan saya tidak melakukan penjiplakan atau pengutipan dengan cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan. Atas pernyataan ini, saya siap menanggung resiko/sanksi yang dijatuhkan kepada saya apabila kemudian ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya ini, atau ada klaim dari pihak lain terhadap karya keaslian saya ini.

|  |
| --- |
| Bandung, Januari 2016  Penulis, |
|  |
| Satrio Adityo Hartomo |

# LEMBAR PENGESAHAN

**Implementasi VoG (*Vocabulary* based summarization of *Graph*) pada *Web* *Graph***

***Implementation of VoG (Vocabulary based summarization of Graph) on the Web Graph***

**Satrio Adityo Hartomo**

**1103120029**

Telah disetujui dan disahkan sebagai tugas akhir

Program Studi Teknik Informatika

Fakultas Informatika Universitas Telkom

Bandung, Januari 2016

Menyetujui

|  |  |  |
| --- | --- | --- |
| Pembimbing I |  | Pembimbing II |
|  |  |  |
|  |  |  |
| Kemas Rahmat Saleh W, S.T., M.Eng. NIP. 06830335-1 |  | Siti Sa'adah, S.T., M.T.  NIP. 13861143-1 |

# ABSTRAK

*Web* *graph* yang memiliki data semakin banyak membuat *graph* semakin sulit untuk diidentifikasi struktur dan informasinya. Untuk mengidentifikasi struktur dan informasi diperlukan suatu metode untuk meringkas *graph* yang besar menjadi *subgraph*-*subgraph* yang lebih kecil. Metode VoG dapat digunakan untuk meringkas *graph* yang besar dan mengidentifikasi struktur *subgraph*. Dalam proses pengidentifikasian *subgraph*, jumlah *node* maksimum dalam GCC berpengaruh terhadap struktur *subgraph* yang dihasilkan. Semakin banyak jumlah *node* maksimum dalam GCC semakin sedikit menghasilkan *subgraph* dengan struktur *clique* dan *star*, namum semakin banyak mengasilkan *subgraph* dengan struktur *chain*. Jumlah *node* maksimum dalam GCC juga berpengaruh terhadap waktu eksekusi VoG. Semakin banyak jumlah *node* dalam GCC semakin cepat pula waktu eksekusi VoG. Dari *subgraph*-*subgraph* yang dihasilkan dan struktur *subgraph* yang diidentifikasi, informasi yang dapat diperoleh adalah *subgraph* *clique* merepresentasikan adanya jaringan user yang saling berpendapat dalam suatu link pertanyaan, *subgraph* *star* merepresentasikan adanya satu user yang berkontribusi menjawab pertanyaan di beberapa link pertanyaan dalam *web* quora.com.

**Kata Kunci:** *web* *graph*, GCC, VoG, *subgraph*, struktur, informasi

# LEMBAR PERSEMBAHAN

Banyaknya pihak yang membantu penulis untuk menyelesaikan tugas akhir ini, baik itu secara langsung maupun tidak langsung, dalam lembar pengesahan ini penulis ingin menyampaikan terima kasih kepada:

1. Allah Subhanahu Wata’ala yang selalu memberikan kasih sayang dan petunjuk yang tak terkira bagi penulis untuk menyelesaikan tugas akhir ini.
2. Rasulullah Muhammad Shallallahu ‘alaihi wasalam yang selalu menjadi panutan bagi penulis.
3. Kedua orang tua yang selalu memberikan semangat, dukungan, dan dorongan bagi penulis.
4. Kakak-kakak yang memberikan arahan dan motivasi bagi penulis.
5. Kedua dosen pembimbing penulis yaitu Bapak Kemas Rahmat S.W, S.T, M.Eng dan Ibu Siti Sa’adah, S.T, M.T yang selalu memberikan bimbingan dan solusi ketika mengalami kesulitan dalam masa pengerjaan tugas akhir.
6. Dosen Wali penulis yaitu Ibu Siti Sa’adah yang membantu masa-masa perkuliahan dari awal pertama kali masuk hingga saat ini
7. Seluruh dosen yang pernah mengajar penulis, khususnya Bapak Soewono yang telah memberikan contoh sikap dan nilai-nilai kehidupan bagi penulis. “Kuliah harus kerja keras !!”
8. Teman-teman PAMTKLA yang selalu menjadi tempat suka dan duka bersama penulis.
9. Teman-teman IFLAB dan Basis Data Lab yang selalu dapat menjadi tempat sharing dan bermain.
10. Teman-teman Informatika khususnya teman-teman IF3602 yang telah memberikan semangat untuk terus berkarya bagi penulis.
11. Teman-teman basket yang selalu memberi motivasi, semangat kerja keras dan kerja sama bagi penulis.
12. Teman-teman kosan BJS102 yang selalu menemani ketika berada di kosan dalam keadaan senang maupun suntuk.
13. Teman-teman Tim Project Sideline, Pixma Gaming, dan Soulfy yang telah memberikan pengalaman yang luar biasa bagi penulis.
14. Teman-teman bisnis Ayam Geprek Mas Ragil yang selalu kuat bekerja sama membangun usaha.
15. Teman-teman Mbah Gatot Klaten yang memberikan persahabatan kelucuan dalam kehidupan penulis.
16. Teman spesial Dian Kharima beserta keluarga yang telah menemani dan membantu dalam perjuangan penulis hingga seperti saat ini, “Adanya kalian perjuanganku lebih ringan”.
17. Someone special AIPMNA yang selalu membuat penulis termotivasi untuk berkarya lebih baik dan lebih baik lagi.
18. Berbagai pihak yang tidak dapat disebutkan oleh penulis.

# KATA PENGANTAR

Puji syukur kepada Allah Subhanahu Wata’ala atas rahmat dan karunianya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul **“Implementasi VoG (*Vocabulary* based summarization of *Graph*) Pada *Web* *Graph*”.**

Tiada gading yang tak retak, penulis menyadari bahwa dalam tugas akhir ini terdapat kekurangan. Oleh karena itu penulis menerima segala kritik dan saran untuk pengembangan tugas akhir ini. Semoga tugas akhir ini dapat bermanfaat bagi semua pihak yang berkepentingan.

|  |
| --- |
| Bandung, Januari 2016  Penulis, |
|  |
| Satrio Adityo Hartomo |

# DAFTAR ISI

[LEMBAR PERNYATAAN ii](#_Toc440974297)

[LEMBAR PENGESAHAN iii](#_Toc440974298)

[ABSTRAK iv](#_Toc440974299)

[LEMBAR PERSEMBAHAN v](#_Toc440974300)

[KATA PENGANTAR vi](#_Toc440974301)

[DAFTAR ISI vii](#_Toc440974302)

[DAFTAR GAMBAR ix](#_Toc440974303)

[DAFTAR TABEL xi](#_Toc440974304)

[DAFTAR ISTILAH xii](#_Toc440974305)

[1. PENDAHULUAN 1](#_Toc440974306)

[1.1 Latar Belakang 1](#_Toc440974307)

[1.2 Perumusan Masalah 2](#_Toc440974308)

[1.3 Tujuan 2](#_Toc440974309)

[1.4 Batasan Masalah 2](#_Toc440974310)

[1.5 Metodologi Penyelesaian Masalah 2](#_Toc440974311)

[1.6 Sistematika Penulisan 3](#_Toc440974312)

[2. DASAR TEORI 4](#_Toc440974313)

[2.1 Teori *Graph* 4](#_Toc440974314)

[2.2 *Graph* *Database* 5](#_Toc440974315)

[2.3 *Web* *Graph* 6](#_Toc440974316)

[2.4 *Graph* *Compression* 7](#_Toc440974317)

[2.4.1 *Graph* *Shattering* 7](#_Toc440974318)

[2.4.2 Slash and Burn 7](#_Toc440974319)

[2.5 VoG 8](#_Toc440974320)

[2.5.1 *Perfect* *Clique* 9](#_Toc440974321)

[2.5.2 *Near* *Clique* 9](#_Toc440974322)

[2.5.3 *Perfect* *Bipartite* 9](#_Toc440974323)

[2.5.4 *Near* *Bipartite* 10](#_Toc440974324)

[2.5.5 *Perfect* *Star* 10](#_Toc440974325)

[2.5.6 *Near* *Star* 11](#_Toc440974326)

[2.5.7 *Perfect* *Chain* 11](#_Toc440974327)

[2.5.8 *Near* *Chain* 11](#_Toc440974328)

[2.5.9 Algoritma VoG 12](#_Toc440974329)

[3. PERANCANGAN SISTEM 13](#_Toc440974330)

[3.1 Deskripsi Sistem 13](#_Toc440974331)

[3.2 Perancangan Sistem 16](#_Toc440974332)

[3.2.1 *Web* *Crawling* 16](#_Toc440974333)

[3.2.2 Representasi *Graph* G dari *Dataset* 18](#_Toc440974334)

[3.2.3 Implementasi VoG 19](#_Toc440974335)

[4. PENGUJIAN DAN ANALISIS 21](#_Toc440974336)

[4.1 Tujuan Pengujian 21](#_Toc440974340)

[4.2 Skenario Pengujian 21](#_Toc440974341)

[4.2.1 Skenario pengujian algoritma *graph* *decomposition* (Slashburn) 21](#_Toc440974342)

[4.2.2 Skenario pengujian pengaruh jumlah *node* maksimum dalam GCC terhadap *subgraph* yang dihasilkan oleh VoG 21](#_Toc440974343)

[4.2.3 Skenario pengujian pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG 22](#_Toc440974344)

[4.3 Analisis Hasil Pengujian 22](#_Toc440974345)

[4.3.1 Analisis hasil pengujian algoritma *graph* *decomposition* 22](#_Toc440974346)

[4.3.2 Analisis pengaruh jumlah *node* maksimum dalam GCC terhadap *subgraph* yang dihasilkan oleh VoG 27](#_Toc440974347)

[4.3.3 Analisis pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG 32](#_Toc440974348)

[4.3.4 Analisis informasi struktur *subgraph* 34](#_Toc440974349)

[5. KESIMPULAN DAN SARAN 36](#_Toc440974350)

[5.1 Kesimpulan 36](#_Toc440974351)

[5.2 Saran 36](#_Toc440974352)

[DAFTAR PUSTAKA 37](#_Toc440974353)

[LAMPIRAN 1 : HASIL PENGUJIAN *SUBGRAPH* YANG DIHASILKAN OLEH VoG 38](#_Toc440974354)

[LAMPIRAN 2 : HASIL PENGUJIAN WAKTU EKSEKUSI VoG 41](#_Toc440974355)

# DAFTAR GAMBAR

[Gambar 2‑1: Directed Graph 4](#_Toc440974356)

[Gambar 2‑2: Undirected Graph 4](#_Toc440974357)

[Gambar 2‑3: Simple Graph 4](#_Toc440974358)

[Gambar 2‑4: Multigraph 4](#_Toc440974359)

[Gambar 2‑5: Pseudograph 4](#_Toc440974360)

[Gambar 2‑6: Graph Berlabel 4](#_Toc440974361)

[Gambar 2‑7: Graph Tak Berlabel 4](#_Toc440974362)

[Gambar 2‑8: Graph G 5](#_Toc440974363)

[Gambar 2‑9: Graph H (subgraph G) 5](#_Toc440974364)

[Gambar 2‑10 Representasi graph - adjacency list [7] 5](#_Toc440974365)

[Gambar 2‑11: Representasi graph - adjacency matrix 5](#_Toc440974366)

[Gambar 2‑12 Model graph database 6](#_Toc440974367)

[Gambar 2‑13: Contoh Web Graph [4] 7](#_Toc440974368)

[Gambar 2‑14: Algoritma Slashburn 8](#_Toc440974369)

[Gambar 2‑15: (a) adalah graph G, (b) adalah graph G' setelah satu kali eksekusi slashburn [13] 8](#_Toc440974370)

[Gambar 2‑16: Contoh full-clique subgraph 9](#_Toc440974371)

[Gambar 2‑17: Beberapa contoh near-clique subgraph 9](#_Toc440974372)

[Gambar 2‑18: full-bipartite subgraph 10](#_Toc440974373)

[Gambar 2‑19: near-bipartite subgraph 10](#_Toc440974374)

[Gambar 2‑20: Star subgraph 10](#_Toc440974375)

[Gambar 2‑21: Near Star 11](#_Toc440974376)

[Gambar 2‑22: Chain subgraph 11](#_Toc440974377)

[Gambar 2‑23: Near Chain subgraph 11](#_Toc440974378)

[Gambar 2‑24: Pseudocode Algoritma VoG [3]. 12](#_Toc440974379)

[Gambar 3‑1:Flowchart sistem secara umum 13](#_Toc440974380)

[Gambar 3‑2: Flowchart proses slashburn untuk decomposing graph menjadi sekumpulan subgraph 14](#_Toc440974381)

[Gambar 3‑3: Flowchart subgraph labeling untuk mengidentifikasi struktur subgraph 15](#_Toc440974382)

[Gambar 3‑4: link question dari quora.com 17](#_Toc440974383)

[Gambar 3‑5: satu link question hasil crawl untuk dijadikan dataset 17](file:///E:\VoG\Buku%20Implementasi%20VoG.docx#_Toc440974384)

[Gambar 3‑6: Flowchart parsing dataset untuk dibentuk graph 18](#_Toc440974385)

[Gambar 3‑7: Visualisasi graph G dari dataset 19](#_Toc440974386)

[Gambar 3‑8: Visualisasi subgraph dari Graph G setelah proses slashburn selesai 20](#_Toc440974387)

[Gambar 4‑1: Gambar dataset-1 yang digunakan untuk pengujian algoritma graph decomposition 22](#_Toc440974388)

[Gambar 4‑2: Gambar keluaran algoritma slashburn untuk graph decomposition 23](#_Toc440974389)

[Gambar 4‑3: Gambar keluaran proses SUBGRAPH LABELING 23](#_Toc440974390)

[Gambar 4‑4: Subgraph-subgraph dengan struktur perfect clique dari gambar 4-2 23](#_Toc440974391)

[Gambar 4‑5: Subgraph-subgraph dengan struktur perfect star dari gambar 4-2 24](#_Toc440974392)

[Gambar 4‑6: Subgraph-subgraph dengan struktur perfect bipartite dari gambar 4-2 24](#_Toc440974393)

[Gambar 4‑7: Subgraph-subgraph dengan struktur perfect chain dari gambar 4-2 24](#_Toc440974394)

[Gambar 4‑8: Gambar dataset-2 yang digunakan untuk pengujian algoritma graph decomposition 25](#_Toc440974395)

[Gambar 4‑9: Gambar keluaran algoritma slashburn untuk graph decomposition. Subgraph yang diberi tanda oval adalah subgraph non-perfect structure. 25](#_Toc440974396)

[Gambar 4‑10: Gambar keluaran proses SUBGRAPH LABELING, non-perfect structure teridentifikasi. 26](#_Toc440974397)

[Gambar 4‑11: Subgraph dengan struktur near clique dari gambar 4-9 26](#_Toc440974398)

[Gambar 4‑12: Subgraph dengan struktur near star dari gambar 4-9 26](#_Toc440974399)

[Gambar 4‑13: Subgraph dengan struktur near bipartite dari gambar 4-9 27](#_Toc440974400)

[Gambar 4‑14: Subgraph dengan struktur near chain dari gambar 4-9 27](#_Toc440974401)

[Gambar 4‑15: Dataset hasil crawling quora.com dengan jumlah node 2250 dan edge 5029 28](#_Toc440974402)

[Gambar 4‑16: Graph quora yang sudah dilakukan graph decomposition, terdapat 1742 node dan 1321 edge 28](#_Toc440974403)

[Gambar 4‑17: Zoom Graph dari gambar 4-8 29](#_Toc440974404)

[Gambar 4‑18: Hasil identifikasi struktur graph quora dengan jumlah node maksimum dalam GCC = 3 29](#_Toc440974405)

[Gambar 4‑19: Grafik hasil pengujian pola pertama 31](#_Toc440974406)

[Gambar 4‑20: Grafik hasil pengujian pola kedua 31](#_Toc440974407)

[Gambar 4‑21: Grafik hasil pengujian waktu eksekusi 34](#_Toc440974408)

[Gambar 4‑22: Subgraph clique hasil pengujian 35](#_Toc440974409)

[Gambar 4‑23: Subgraph star hasil pengujian 35](#_Toc440974410)

# DAFTAR TABEL

[Tabel 4‑1: Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 3 29](#_Toc440974411)

[Tabel 4‑2: Tabel summary hasil pengujian pola pertama 30](#_Toc440974412)

[Tabel 4‑3: Tabel summary hasil pengujian pola kedua 30](#_Toc440974413)

[Tabel 4‑4: Hasil pengujian waktu eksekusi VoG dengan jumlah node maksimum dalam GCC = 5 32](#_Toc440974414)

[Tabel 4‑5: Hasil pengujian waktu rata-rata eksekusi VoG 33](#_Toc440974415)

# DAFTAR ISTILAH

*Graph* Sekumpulan *node* dan *edge* yang terhubung

*Subgraph* Bagian kecil dari *graph*

*Real World Graph* Representasi *graph* dari kasus-kasus yang ada di dunia nyata

*Web* Suatu halaman yang berisikan data yang dapat diakses melalui *internet*

*Web Graph* Representasi *graph* dari suatu *web*

*GCC Giant Connected Component, subgraph* yang memiliki jumlah *node* terbanyak

*Summarize* Meringkas suatu *graph* agar menjadi lebih sedikit jumlah *node* dan *edge*nya

*Vocabulary* Sekumpulan struktur *subgraph* seperti *clique, star, chain* dan *bipartite*

*Manual Inspection* Mengamati dan menganalisis data secara manual.

# PENDAHULUAN

## Latar Belakang

Dengan adanya teknik penyimpanan data yang baru yaitu *graph database* [5], cara untuk menyimpan data tidak lagi terbatas pada *relational database* yang struktural. Data yang mempunyai perbedaan atribut (semi terstruktur) dapat disimpan ke *database* dengan representasi *graph* [5], seperti Facebook dan Twitter yang juga menggunakan *graph database* untuk menyimpan data-datanya [8,10]. *Website* yang lain pun memiliki data yang unik, berjumlah banyak dan semakin bertambah hingga saat ini [2]. *Website* yang dahulu hanya menampilkan data statis kini telah berevolusi menjadi *website* yang dapat menampilkan data secara dinamis dan dapat direpresentasikan sebagai *graph* (*web graph*) [5].

Pada referensi [2] disebutkan bahwa pada Juli 2000 terdapat dua milyar *web*, dan terus meningkat jumlahnya hingga empat milyar *web* pada tahun 2001. *Web graph* yang besar dengan ribuan *node* dan *edge* [9], akan sulit untuk diidentifikasi bagaimana bentuk strukturnya, apakah memiliki struktur *random* atau tidak. Dengan ukuran yang besar tersebut akan sulit juga untuk memperoleh informasi. Oleh karena itu diperlukan metode untuk meringkas *web graph* agar ukurannya menjadi lebih kecil dari *web graph* aslinya, sehingga dapat dilakukan pengidentifikasian struktur dan analisis untuk mendapatkan informasi pada hasil ringkasan *web graph*.

Hasil ringkasan yang dapat digunakan untuk pengidentifikasian struktur dan analisis informasi adalah subgraph. Subgraph yang dihasilkan mempunyai banyak kemungkinan struktur seperti pada hasil penelitian dalam referensi [3] terdapat struktur *clique, star, chain*, dan *bipartite*. Dalam metode VoG, terdapat *vocabulary* atau subgraph yang memiliki struktur *clique, star, chain*, dan *bipartite* yang sesuai dengan struktur yang dibutuhkan untuk analisis informasi. Karena VoG dapat menghasilkan subgraph untuk meringkas *web graph* maka dalam tugas akhir ini akan digunakan metode VoG untuk meringkas *web graph* dan mendapatkan analisis informasi dari *subgraph* yang dihasilkan menggunakan metode *manual inspection*.

VoG (*Vocabulary based summarization of Graph*) adalah metode untuk meringkas *graph* yang besar secara efisien [3]. Dalam metode VoG akan dilakukan *graph decomposition* menggunakan algoritma Slashburn. Algoritma Slashburn akan menghasilkan sekumpulan *subgraph* yang mana *subgraph* yang dihasilkan akan dilakukan proses identifikasi struktur. Hal ini sesuai dengan yang diharapkan dalam metode VoG yang harus menghasilkan subgraph. Dalam proses menghasilkan *subgraph*, jumlah *node* maksimum pada dalam GCC (*Giant Connected Component*) akan mempengaruhi jumlah dan struktur *subgraph*. Waktu eksekusi algoritma VoG juga akan dipengaruhi oleh hal tersebut. Dalam tugas akhir ini akan dilakukan implementasi *web graph* *summarization* menggunakan metode VoG untuk meringkas *web graph* menjadi *subgraph-subgraph* yang lebih kecil ukuran atau jumlah *node*nya, menganalisis pengaruh jumlah *node* maksimum dalam GCC terhadap *subgraph* yang dihasilkan, menganalisis pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG dan mendapatkan informasi dari *subgraph-subgraph* yang dihasilkan.

## Perumusan Masalah

Dari latar belakang yang telah diuraikan, permasalahan yang akan dibahas dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana mengimplementasikan VoG untuk *summarize* *web graph*?
2. Bagaimana pengaruh jumlah *node* maksimum dalam GCC (*k*) terhadap struktur yang dihasilkan oleh VoG?
3. Bagaimana pengaruh jumlah *node* maksimum dalam GCC (*k*) terhadap waktu eksekusi VoG?
4. Bagaimana menganalisis struktur *subgraph* yang dihasilkan oleh VoG untuk memperoleh minimum satu kalimat informasi?

## Tujuan

Adapun tujuan dalam tugas akhir ini adalah sebagai berikut:

1. Mengimplementasikan metode VoG untuk *summarize* *web* *graph*.
2. Menganalisis pengaruh jumlah *node* maksimum dalam GCC terhadap struktur yang dihasilkan oleh VoG untuk mengetahui pengaruh *k* dalam menghasilkan *subgraph,* yang mana *subgraph* ini dapat digunakan sebagai alat bantu analisis informasi dari struktur yang terdapat dalam *vocabulary*.
3. Menganalisis pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG untuk mengetahui pengaruh *k* dalam menghasilkan keluaran dengan waktu yang cepat.
4. Menganalisis struktur *subgraph* yang dihasilkan oleh VoG untuk memperoleh minimum satu kalimat informasi.

## Batasan Masalah

Mengenai batasan permasalahan dalam tugas akhir ini adalah sebagai berikut:

1. Struktur yang dapat diidentifikasi oleh metode VoG dalam tugas akhir ini terdiri dari *Perfect Clique, Perfect Star, Perfect Bipartite, Perfect Chain, Near Clique, Near Star, Near Bipartite,* dan *Near Chain.*
2. Metode VoG hanya dapat melakukan identifikasi struktur setelah memiliki keluaran dari algoritma *graph decomposition* yang dalam tugas akhir ini adalah algoritma Slashburn.
3. *Dataset* yang digunakan didapat dari hasil *crawling web* menggunakan aplikasi buatan sendiri.
4. Dua *node* yang terhubung hanya memiliki satu *edge*.
5. Untuk pengolahan *graph* menggunakan bantuan *library* GraphStream.
6. Untuk menganalisis informasi dalam *subgraph* digunakan metode *manual inspection*.

## Metodologi Penyelesaian Masalah

Metode penyelesaian masalah yang diterapkan dalam tugas akhir ini adalah sebagai berikut:

1. Identifikasi permasalahan

Pada tahap ini dilakukan studi literatur dan diskusi untuk mengidentifikasi permasalahan yang akan dibahas pada tugas akhir ini. Identifikasi meliputi observasi terkait fenomena yang terjadi di bidang *graph database*, metode yang digunakan untuk solusi permasalahan, dan batasan masalah yang ada.

1. Studi literatur

Pada tahap ini dilakukan pengumpulan dan pemahaman terkait materi-materi yang dibutuhkan untuk menyelesaikan permasalahan yang tertera di perumusan masalah. Referensi yang dicari antara lain materi mengenai metode VoG, metode *graph decomposition* untuk menghasilkan *graph* *summarization*.

1. Perancangan dan pembuatan sistem

Pada tahap ini dilakukan pemodelan *dataset* ke dalam bentuk *graph*. Kemudian membuat rancangan sistem untuk memberikan solusi dari permasalahan. Dari rancangan sistem yang sudah dibuat dilakukan pembuatan sistem dengan menggunakan bahasa pemrograman Java.

1. Pengujian dan analisis

Pada tahap ini dilakukan pengujian terhadap *dataset* yang sudah dimodelkan dalam bentuk *graph*. Pengujian dilakukan sesuai dengan skenario yang ada pada bagian strategi pengujian. Kemudian dilakukan analisis terhadap *subgraph-subgraph* hasil ringkasan *graph*.

1. Penyusunan laporan

Pada tahap ini dilakukan dokumentasi dan pelaporan hasil sesuai dengan aturan dan sistematika penulisan yang ditetapkan institusi.

## Sistematika Penulisan

Sistematika penulisan buku tugas akhir ini adalah sebagai berikut:

* 1. Bab I Pendahuluan

Bab ini berisi latar belakang, perumusan masalah, tujuan, batasan masalah, metodologi penyelesaian masalah dan sistematika penulisan buku tugas akhir.

* 1. Bab II Landasan Teori

Bab ini berisi landasan teori yang menjadi dasar dilakukannya penelitian pada tugas akhir ini.

* 1. Bab III Perancangan Sistem

Bab ini berisi perancangan sistem yang akan diimplementasikan pada tugas akhir ini.

* 1. Bab IV Pengujian dan Analisis

Bab ini berisi pengujian dan analisis hasil dari sistem yang telah dibuat.

* 1. Bab V Kesimpulan dan Saran

Bab ini berisi kesimpulan yang didapat dari permasalahan, pengujian dan analisis hasil, serta saran untuk penelitian selanjutnya.

# DASAR TEORI

## 2.1 Teori *Graph*

*Graph*, secara definisi adalah sekumpulan *nodes* () dan *edges* (). Sekumpulan *nodes* bisa saja tidak terbatas atau bisa disebut ***infinite graph***, sedangkan yang sekumpulan *nodes*nya terbatas disebut ***finite graph***. Dalam tugas akhir ini hanya akan fokus pada *finite graph*. *Graph* dapat dinotasikan sebagai . adalah sekumpulan *nodes* dan adalah sekumpulan *edges* yang menghubungkan *nodes* [6].

|  |  |  |  |
| --- | --- | --- | --- |
| Gambar 2‑1: Directed Graph | | Gambar 2‑2: Undirected Graph | |
| Gambar 2‑3: Simple Graph | Gambar 2‑4: Multigraph | | Gambar 2‑5: Pseudograph |
| Gambar 2‑6: Graph Berlabel | | Gambar 2‑7: Graph Tak Berlabel | |

Perbedaan jenis *graph* dapat dilihat berdasarkan ada atau tidaknya arah *edge*, *graph* jenis ini terdiri dari ***directed graph*** (*graph* yang mempunyai *edge* berarah) dan ***undirected graph* (***graph* yang mempunyai *edge* tidak berarah). Berdasarkan *edge* yang menghubungkan *nodes* terdiri dari ***simple graph*** (tidak memiliki *loop* dan *multiple edge*), ***multigraph*** (tidak memiliki *loop edge* tetapi memiliki *multiple edge*) dan ***pseudograph*** (memiliki *loop edge*). Berdasarkan ada atau tidaknya label terdiri dari ***graph* berlabel** dan ***graph* tak berlabel**. Contoh jenis-jenis *graph* dapat dilihat pada gambar 2-1 sampai 2-7 di bawah ini [6].

Sekumpulan *nodes* dan *edges* yang ada di *graph* dapat digunakan untuk memodelkan permasalahan yang ada pada dunia nyata, sebagai contoh *graph* jaringan komputer. Namun, dalam kondisi tertentu untuk menyelesaikan permasalahan yang ada, tidak seluruh *nodes* digunakan. Sehingga yang digunakan untuk menyelesaikan masalah adalah bagian kecil dari *graph*, atau bisa disebut dengan *subgraph*. Secara definisi, *subgraph* dari *graph* adalah *graph* dimana dan dan [6].

|  |  |
| --- | --- |
| Gambar 2‑8: Graph G | Gambar 2‑9: Graph H (subgraph G) |

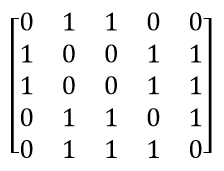
Cara merepresentasikan *graph* ada beberapa cara, diantaranya yaitu menggunakan *Adjacency Matrix* dan *Adjacency List*. Berikut contoh cara merepresentasikan *graph* [6]:

Dari gambar 2-8 dapat direpresentasikan *adjacency list* seperti gambar 2-10 di bawah ini :



Gambar 2‑10 Representasi graph - adjacency list [7]

Atau sebagai *adjacency matrix* seperti di bawah ini :



Gambar 2‑11: Representasi graph - adjacency matrix

Angka 1 merepresentasikan adanya *edge* antar*nodes* [6].

## 2.2 *Graph* *Database*

*Graph Database Management System* adalah *database management system* yang memiliki metode CRUD (*Create*, *Read*, *Update*, dan *Delete*) untuk memaparkan model data *graph*. Properti model *graph* dapat dideskripsikan sebagai berikut :

* Terdiri dari *nodes* dan *relationship/edges*.
* Setiap *nodes* terdiri dari pasangan *key-value*.
* *Edge* mempunyai nama dan arah, jika tidak memiliki arah disebut *undirected graph.*
* *Edge* dapat juga mempunyai properti.

Dengan abstraksi sederhana dari *nodes* dan *edges* kedalam struktur yang saling terhubung, *graph database* memungkinkan untuk dibangunnya model yang mirip dengan permasalahan yang ada di dunia nyata [5].



Gambar 2‑12 Model graph database

## 2.3 *Web* *Graph*

*Webpages* dan *hyperlinks* di *World Wide Web* dapat direpresentasikan sebagai *nodes* dan *edge* di *directed graph*. Saat ini, *graph* tersebut memiliki milyaran *nodes* dan *edge* dan bertambah banyak seiring berjalannya waktu. Ada beberapa alasan untuk mempelajari *web graph*, diantaranya adalah bagaimana algoritma yang bekerja pada *web graph*, bagaimana *web search*nya, dan bagaimana pengklasifikasiannya [12].

Tidak hanya itu, ada alasan lain untuk mengembangkan pemahaman dari *web graph* ini, diantaranya dapat :

1. Mendesain strategi *crawling* pada *web*.
2. Memahami sosiologi pembuatan konten pada *web*.
3. Menganalisis *behavior* dari algoritma *web*.
4. Memprediksi evolusi struktur *web*.
5. Memprediksi munculnya fenomena baru di *web* [1].

Berikut contoh *web graph*:



Gambar 2‑13: Contoh Web Graph [4]

## 2.4 *Graph* *Compression*

Jika dalam suatu *graph* dapat ditemukan *good community*, maka *graph* tersebut dapat dikompres yang mana dapat membantu untuk menyusutkan ukuran data [13]. Rincian proses *graph compression* dalam tugas akhir ini dijelaskan pada bagian 2.4.1 dan 2.4.2.

### *Graph* *Shattering*

Untuk menyelesaikan permasalahan dalam *graph compression*, diperlukan proses untuk mengurai *graph* dengan cara mengeksploitasi *hubs* (*node* dengan *degree* paling tinggi). Dengan hasil eksploitasi *hubs* didapatkan *alternative community* yang berbeda dari *clique community* [13].

Dari referensi [13] disebutkan bahwa berdasarkan observasi, *real world graph* dapat dengan mudah dipecah dengan menghapus *hub node* dari *graph*. Dengan penghapusan *hub node*, *graph* terurai menjadi banyak *subgraph-subgraph*.

### 2.4.2 Slash and Burn

Baris 1 dan 2 dari algoritma 1 menghapus *nodes* dengan *degree* paling tinggi dan *edge* yang berasosiasi, dengan demikian *graph* terurai dalam tiga kelompok:

* -*hub*set: *nodes* dengan *degree* paling tinggi.
* GCC: *node* yang dimiliki oleh *Giant Conncected Component* dari .
* Spokes: *node* yang dimiliki oleh *non-giant connected component* dari [13].



Gambar 2‑14: Algoritma Slashburn



Gambar 2‑15: (a) adalah graph G, (b) adalah graph G' setelah satu kali eksekusi slashburn [13]

## 2.5 VoG

*Vocabulary based summarization of Graph* (VoG) merupakan metode yang efektif dan efisien untuk meringkas *graph* yang besar. *Vocabulary* (Ω) adalah sekumpulan struktur *graph* yang terdiri dari struktur *full-clique, near-clique, full-bipartite, near-bipartite, perfect-star, near-star, perfect-chains* dan *near-chains*. Masih ada struktur *graph* yang lain, akan tetapi dari referensi [3]disebutkan bahwa struktur *graph* yang sering muncul dalam *real world* *graph* adalah struktur yang ada pada *Vocabulary* tersebut. Oleh karena itu, dalam tugas akhir ini dibangun sistem yang dapat mengidentifikasi struktur dalam *vocabulary*. Penjelasan mengenai struktur *graph* dapat dilihat di bagian 2.5.1 - 2.5.8 [3].

*Graph* yang ada di dunia sangatlah banyak dan besar, orang-orang sulit untuk memahami dengan mudah *graph* yang besar dan mungkin *random* strukturnya. Dengan adanya VoG, *graph* dapat diringkas dan orang-orang dapat lebih mudah memahaminya [3].

### 2.5.1 *Perfect* *Clique*

*Perfect Clique* adalah *complete subgraph* dari suatu *graph* atau semua *node* didalam *subgraph* saling terhubung dengan *node* yang lain di dalam *subgraph* yang sama. Berikut contoh struktur *perfect clique* *graph* seperti gambar 2-15 [11]:



Gambar 2‑16: Contoh full-clique subgraph

### 2.5.2 *Near* *Clique*

*Near Clique* adalah *subgraph* yang hampir *complete* dari suatu *graph*. Misalnya diberikan *graph* seperti gambar 2-15. *Near clique* yang terbentuk adalah [3]:



Gambar 2‑17: Beberapa contoh near-clique subgraph

### 2.5.3 *Perfect* *Bipartite*

*Perfect bipartite* adalah sekumpulan *nodes* dan yang mana semua *node* di terhubung ke semua *node* di , namun tidak ada *edge* dalam kumpulan *nodes* atau . Contoh *full bipartite* [3]:



Gambar 2‑18: full-bipartite subgraph

### 2.5.4 *Near* *Bipartite*

*Near bipartite* adalah *nodes* dan yang mana **hampir** semua *node* di terhubung ke semua *node* di , **ada** *node* di yang tidak terhubung ke salah satu *node* di , namun tidak ada *edge* dalam kumpulan *nodes* atau . Contoh *near bipartite* [3]:



Gambar 2‑19: near-bipartite subgraph

### 2.5.5 *Perfect* *Star*

*Star* adalah kasus spesifik dari *bipartite graph*, yang mana hanya ada *single node* terhubung ke semua *node* yang mana paling tidak terdiri minimal 2 *nodes* [3].



Gambar 2‑20: Star subgraph

### 2.5.6 *Near* *Star*

*Near Star* adalah *subgraph* yang mempunyai struktur nyaris *perfect* *star* [3]. Berikut contoh *near* *star* dari *star* pada gambar 2-19:



Gambar 2‑21: Near Star

### 2.5.7 *Perfect* *Chain*

*Perfect Chain* adalah *graph* yang setiap *node*nya memiliki *edge* ke *node* selanjutnya [3].



Gambar 2‑22: Chain subgraph

### 2.5.8 *Near* *Chain*

*Near Chain* adalah *subgraph* yang mempunyai struktur hampir *perfect* *chain* [3].



Gambar 2‑23: Near Chain subgraph

### 2.5.9 Algoritma VoG

Pseudocode dari VoG adalah sebagai berikut :

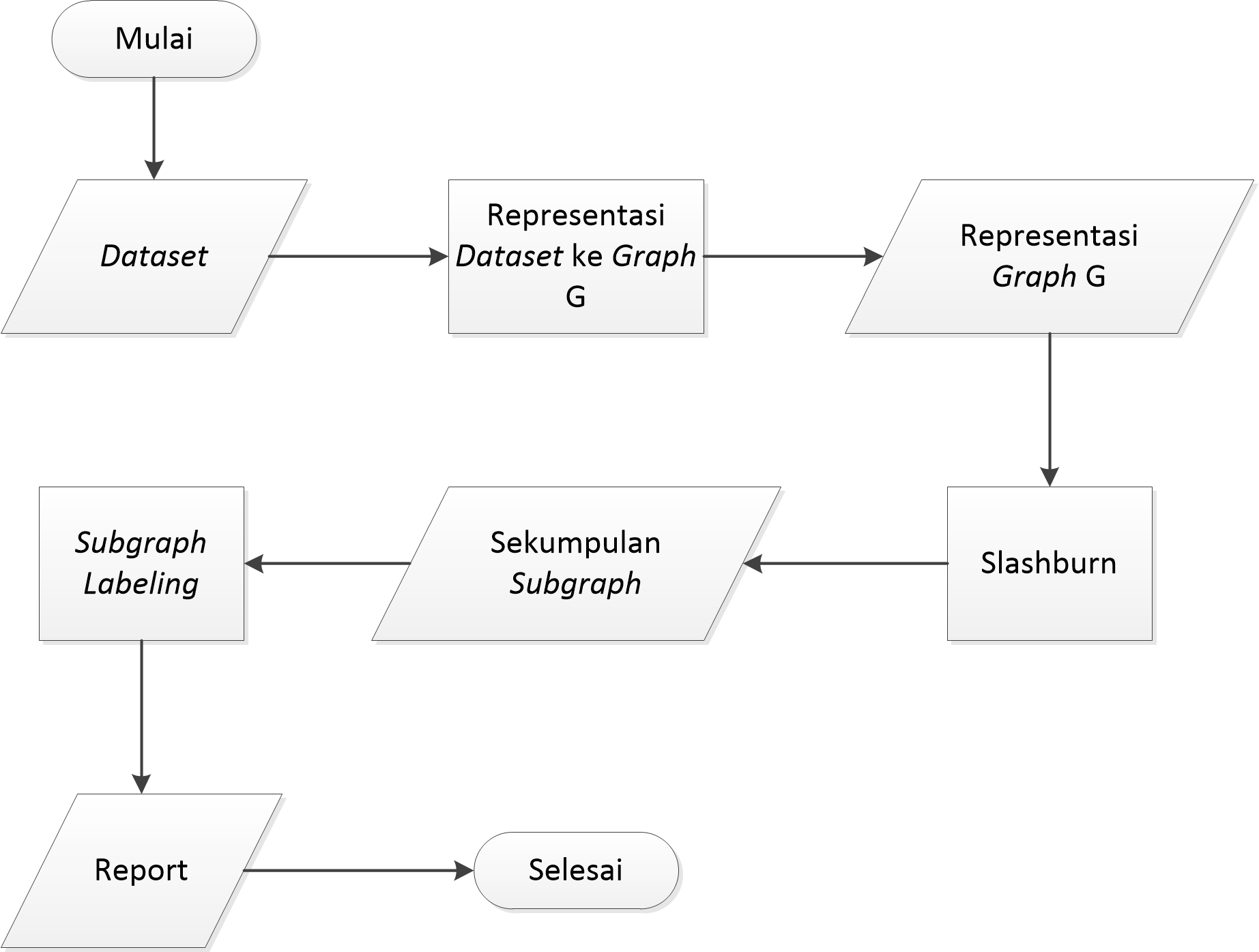


Gambar 2‑24: Pseudocode Algoritma VoG [3].

# PERANCANGAN SISTEM

## 3.1 Deskripsi Sistem

Sistem yang dibangun dalam tugas akhir ini adalah sistem yang dapat meringkas *graph* yang besar menjadi sekumpulan *subgraph* yang berukuran lebih kecil. Dari masing-masing *subgraph* akan diidentifikasi struktur dan dianalisis informasinya. *Dataset* yang digunakan dalam pengujian sistem adalah *dataset* yang didapat melalui *crawling* *web* quora.com. Data yang di-*crawl* adalah data *username* dalam suatu *link question*. Dari data yang sudah diperoleh kemudian direpresentasikan sebagai *graph*, yang mana *node* merepresentasikan *username*, dan *edge* merepresentasikan dua *username* menjawab pertanyaan di *link* yang sama. Berikut ini adalah *flow chart* dari sistem yang dibangun:

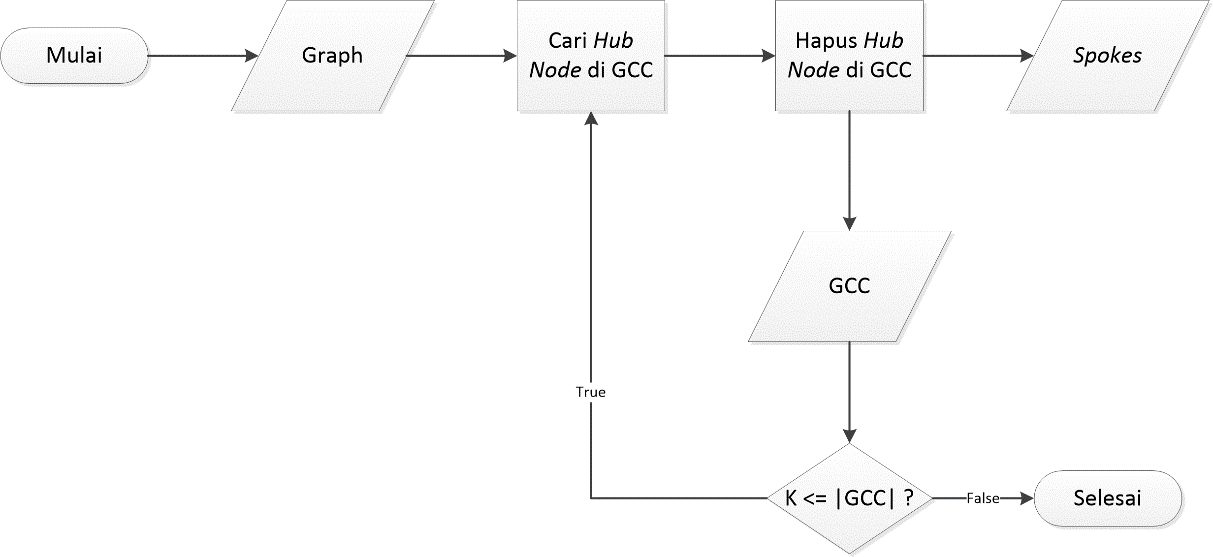


Gambar 3‑1:Flowchart sistem secara umum

*Flowchart* pada gambar 3-1 di atas terbagi menjadi beberapa bagian yang diimplementasikan, bagian tersebut adalah:

1. **Representasi *Dataset* ke *Graph* G**, yaitu penerimaan input *dataset* yang kemudian dilakukan pengubahan bentuk menjadi *graph*. *Graph* tersebutlah yang akan dilakukan proses *summarize*.
2. **Slashburn**, yaitu proses untuk mengurai *graph* menjadi sekumpulan *subgraph*-*subgraph* yang mana *subgraph* tersebut akan diidentifikasi strukturnya.
3. ***Subgraph* *Labeling***, yaitu proses untuk mengidentifikasi struktur dari *subgraph*-*subgraph* yang telah dihasilkan oleh Slashburn. Keluaran proses ini adalah laporan jumlah *vocabulary*.

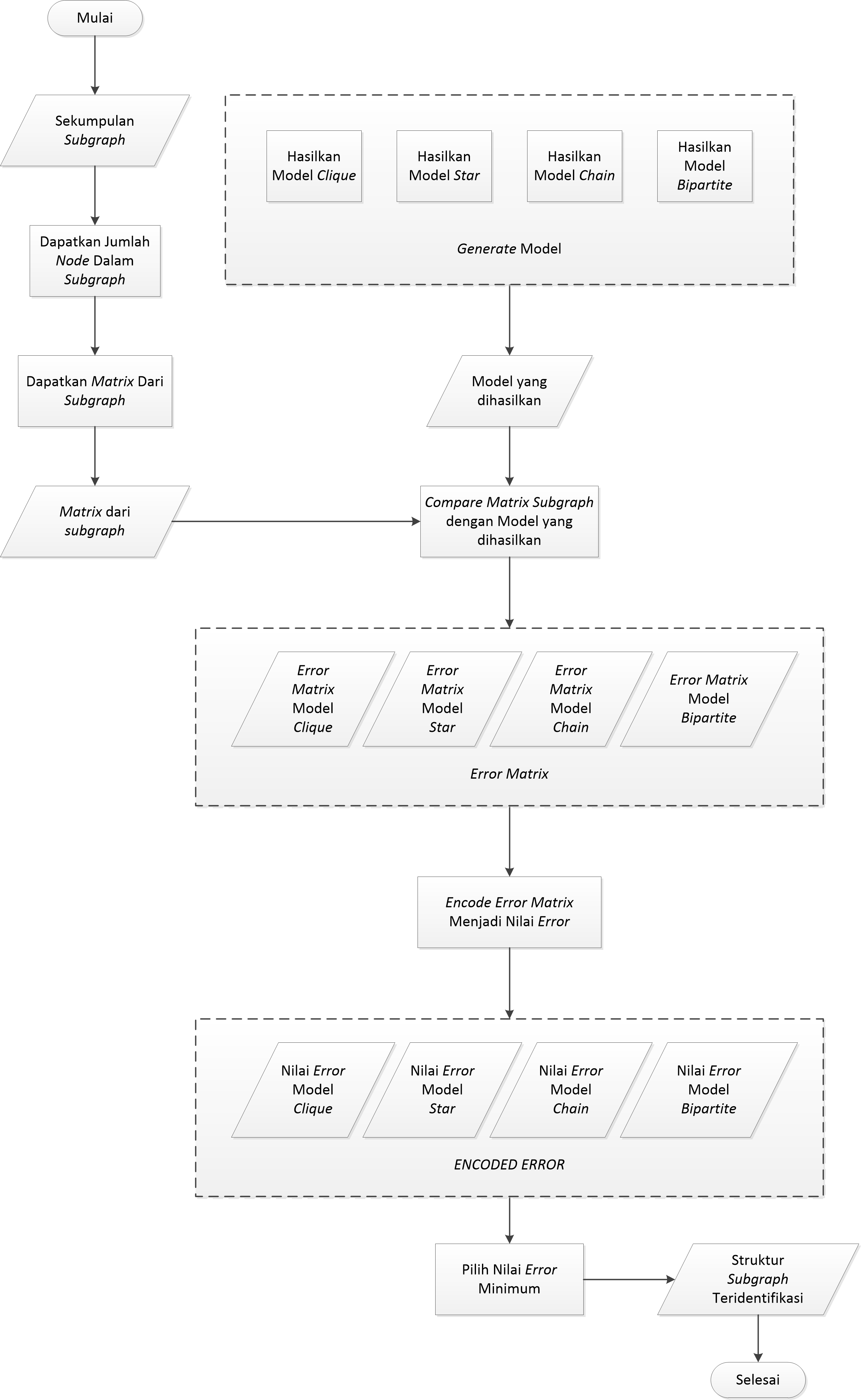
Gambar 3-1 adalah gambaran secara umum proses-proses yang ada dalam sistem yang dibangun. Proses yang lebih rinci mengenai proses *slashburn* dan *subgraph labeling*, dapat dilihat dalam gambar 3-2 dan 3-3 di bawah ini:



Gambar 3‑2: Flowchart proses slashburn untuk decomposing graph menjadi sekumpulan subgraph

*Flowchart* pada gambar 3-2 terdiri dari beberapa bagian yang dilakukan, diantaranya adalah:

1. **Cari *Hub* *Node* di GCC**, yaitu proses pencarian *hub* *node* atau *node* dengan *degree* paling tinggi yang berada pada *graph* yang menjadi masukan sistem.
2. **Hapus *Hub* *Node* di GCC**, yaitu proses penghapusan *hub* *node* yang berada di GCC. Proses penghapusan *hub* ini akan menghapus juga *edge*-*edge* yang berasosiasi dengan *hub*. Proses ini akan menghasilkan *subgraph*-*subgraph* baru yang terdiri atas spokes (*subgraph* non GCC) dan GCC. Proses ini akan terus dijalankan apabila jumlah *node* yang berada GCC lebih dari atau sama dengan batas .



Gambar 3‑3: Flowchart subgraph labeling untuk mengidentifikasi struktur subgraph

*Flowchart* *subgraph* *labeling* pada gambar 3-3 terdiri beberapa bagian yang dilakukan, diantaranya adalah:

1. **Dapatkan Jumlah *Node* Dalam *Subgraph***, yaitu proses untuk mendapatkan jumlah *node* dalam *subgraph*.
2. **Dapatkan *Matrix* Dari *Subgraph***, yaitu proses mendapatkan representasi adjacency *matrix* dari *subgraph*.
3. ***Generate* Model**, yaitu proses untuk menghasilkan adjacency *matrix* model *clique*, *star*, *chain* dan *bipartite*. Ukuran *matrix* yang dihasilkan sama dengan jumlah *node* yang sudah didapatkan pada langkah sebelumnya.
4. ***Compare* *Matrix* *Subgraph* dengan Model yang dihasilkan**, yaitu proses membandingkan *matrix* *subgraph* dan masing-masing *matrix* model yang sudah dihasilkan dengan operasi XOR. Hasil perbandingan ini akan menghasilkan *error* *matrix* dari masing-masing model.
5. **Encode *Error* *Matrix* Menjadi Nilai *Error***, yaitu proses pengubahan *error* *matrix* masing-masing model menjadi suatu nilai *error* masing-masing model.
6. **Pilih Nilai *Error* Minimum**, yaitu proses pemilihan nilai *error* paling kecil diantara masing-masing model. Dengan terpilihnya nilai *error* yang paling kecil ini, maka *subgraph* dapat teridentifikasi sesuai dengan model yang mempunyai nilai *error* paling kecil.

## 3.2 Perancangan Sistem

Alur dari sistem yang dibangun dalam tugas akhir ini terdiri dari tiga tahap:

* 1. *Crawling* *dataset* dari *web* quora.com, *dataset* disimpan dalam bentuk file dengan format .txt.
  2. Representasi *dataset* ke bentuk *graph* dengan bantuan *library* GraphStream Java.
  3. Penerapan VoG, yaitu *graph compression* untuk menghasilkan *subgraph-subgraph* dari *graph input*, lalu *subgraph labeling* untuk mengidentifikasi struktur masing-masing *subgraph*.

### 3.2.1 *Web* *Crawling*

Untuk mendapatkan *dataset* yang digunakan dalam tugas akhir ini, dilakukan *web crawling* pada *web* quora.com. Data yang diambil adalah *username* dan *link question*. Berikut ini contoh *web* yang di-*crawl*:



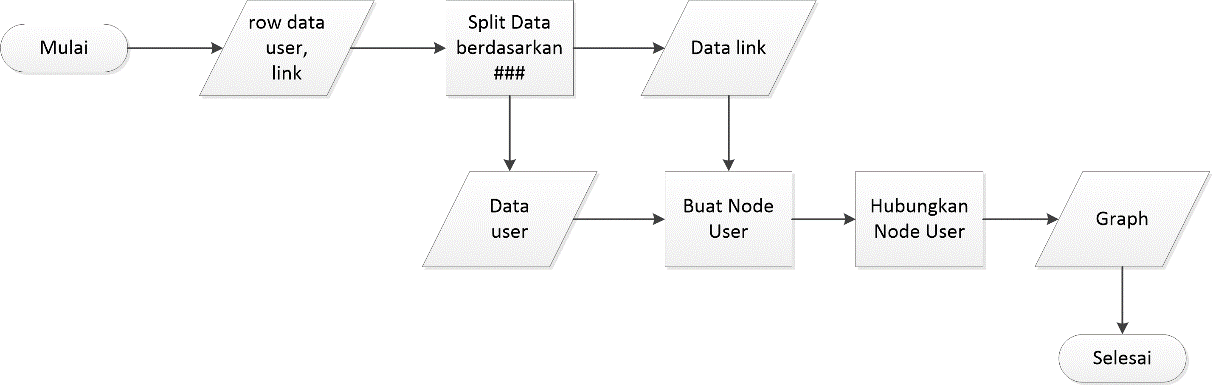
Gambar 3‑4: link question dari quora.com

Setelah *web crawler* selesai melakukan *crawling*, akan diperoleh *dataset*. Berikut ini adalah contoh *dataset* yang didapatkan:



Gambar 3‑5: satu link question hasil crawl untuk dijadikan dataset

Dari dataset yang sudah didapatkan seperti pada gambar 3-5 di atas, setiap baris akan dipisah menjadi *url* dan *username* untuk dibuat representasi *graph*nya. *Username* akan dibuat ke dalam *node* dalam sistem, kemudian dihubungkan dengan *node username* yang lain yang berada pada satu *link question* yang sama. Proses lebih detail dapat dilihat pada *flowchart* berikut ini:



Gambar 3‑6: Flowchart parsing dataset untuk dibentuk graph

Flowchat pada gambar 3-6 terdiri dari beberapa proses, diantaranya:

1. **Split data**, dari satu row dataset yang dimiliki, akan dipisahkan berdasarkan ###. Dari proses ini akan didapatkan data user dan link.
2. **Buat node user**, dari data user yang sudah didapat lalu dibuat node.
3. **Hubungkan node user**, dari node-node tersebut lalu user yang menjawab pertanyaan yang sama akan dihubungkan. Keluaran yang didapatkan adalah *graph*.

### 3.2.2 Representasi *Graph* G dari *Dataset*

Dari *dataset* yang sudah didapatkan, data setiap baris akan dipecah untuk mendapatkan *url*, dan *username*. Dari *url* dan *username* tersebut akan dibuat representasi *graph* menggunakan bantuan *library* GraphStream Java. Berikut ini adalah contoh visualisasi *graph* di aplikasi:



Gambar 3‑7: Visualisasi graph G dari dataset

### 3.2.3 Implementasi VoG

Untuk proses implementasi VoG, harus ditentukan terlebih dahulu berapa jumlah *node* maksimum dalam GCC (). Jumlah *node* maksimal ini akan menjadi batas untuk iterasi proses *graph compression* menggunakan algoritma Slashburn.

#### 3.2.3.1 Proses Graph Compression menggunakan Algoritma Slashburn

Algoritma Slashburn akan berhenti dieksekusi jika . Ketika proses slashburn berjalan, *graph* diproses secara iteratif dengan batas . *Graph* akan dicari *hub node* di dalam GCC untuk dihapus. Penghapusan *hub node* ini disertai dengan *edge*-nya dan menghasilkan *subgraph*. Jika proses penghapusan *node* ini menghasilkan satelit *node* yang *isolated*, maka *hub node* akan dibuat kembali dan satelit *node* akan dihubungkan dengan *hub node*. Hal ini dilakukan agar tidak terbentuk *subgraph* dengan satu *node*. Setelah proses *graph compression* selesai, akan dihasilkan *subgraph-subgraph*. Berikut ini adalah *subgraph* yang terbentuk dari pada gambar 3-7 dengan :



Gambar 3‑8: Visualisasi subgraph dari Graph G setelah proses slashburn selesai

#### 3.2.3.2 Proses Subgraph Labeling

Setelah proses *graph compression* menggunakan algoritma slashburn selesai, didapatkan *subgraph*. Dari *subgraph-subgraph* tersebut akan diidentifikasi strukturnya. Untuk proses identifikasi struktur, yang pertama dilakukan adalah mendapatkan jumlah *node* dalam *subgraph* dan *adjacency matrix*-nya. Setelah jumlah *node* didapat, maka dibuatlah empat model yaitu *clique, star, chain,* dan *bipartite* model. Model-model inilah yang akan digunakan untuk proses mendapatkan *error matrix*. Berikut ini adalah contoh proses untuk mendapatkan *error matrix*:

(3.1)

Contoh operasi matematika (3.1) diatas adalah operasi *adjacency matrix subgraph* XOR dengan *matrix* model *clique*, yang menghasilkan *error matrix* untuk model *clique*.

(3.2)

Contoh operasi matematika (3.2) diatas adalah operasi *adjacency matrix subgraph* XOR dengan *matrix* model *star*, yang menghasilkan *error matrix* untuk model *star*.

Dari dua contoh *error matrix* diatas, *error matrix* kemudian di-*encode* menjadi *error score* untuk masing-masing model. Dalam kasus di atas, *error score* untuk *error matrix* model *clique* = 2. Sedangkan *error score* untuk *error matrix star* adalah 0. Setelah didapatkan *error score* untuk masing-masing model, akan dipilih *error score* yang paling kecil nilainya, sehingga *subgraph* akan diidentifikasi sebagai ***Subgraph* dengan Struktur *Star***.

# PENGUJIAN DAN ANALISIS

Sistem dalam tugas akhir ini dibuat menggunakan bahasa pemrograman Java dengan bantuan *library* GraphStream untuk mengelola *graph*. *Dataset* yang digunakan dapat berbentuk file .txt yang dalam hal ini adalah *dataset* hasil *crawling web* quora.com. Dapat juga berbentuk file .dgs hasil penyimpanan *graph* yang mana file .dgs tersebut dapat dimuat kembali ke sistem.



## 4.1 Tujuan Pengujian

Dari pengujian yang dilakukan, tujuan yang ingin dicapai adalah sebagai berikut:

1. Menganalisis *subgraph* yang dihasilkan oleh algoritma *graph decomposition* apakah *subgraph* yang dihasilkan sesuai dengan *vocabulary* dalam VoG.
2. Menganalisis pengaruh jumlah *nodes* maksimum dalam GCC terhadap *subgraph* yang dihasilkan VoG.
3. Menganalisis pengaruh jumlah *nodes* maksimum dalam GCC terhadap waktu eksekusi VoG.
4. Menganalisis informasi yang terkandung dalam struktur *subgraph*.

## 4.2 Skenario Pengujian

Dalam pengujian sistem ini, terdapat tiga skenario pengujian. Skenario yang pertama adalah pengujian algoritma *graph decomposition* (Slashburn) untuk menghasilkan *subgraph* yang ada pada *vocabulary*. Skenario kedua adalah pengujian pengaruh jumlah *node* maksimum dalam GCC terhadap *subgraph* yang dihasilkan oleh VoG. Skenario ketiga adalah pengujian pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG.

### 4.2.1 Skenario pengujian algoritma *graph* *decomposition* (Slashburn)

Pada skenario pengujian ini, akan dibuktikan apakah algoritma untuk mengurai *graph* menjadi *subgraph* sudah sesuai dengan *vocabulary* yang ada pada VoG. Untuk membuktikannya, dibuatlah *dataset* yang memiliki seluruh struktur yang ada pada *vocabulary*. Kemudian *dataset* tersebut direpresentasikan sebagai *web graph* dalam sistem. *Graph* yang terbentuk kemudian dilakukan *graph decomposition* menggunakan algoritma Slashburn. Setelah proses *graph decomposition* selesai, *subgraph* yang dihasilkan akan diidentifikasi strukturnya.

### 4.2.2 Skenario pengujian pengaruh jumlah *node* maksimum dalam GCC terhadap *subgraph* yang dihasilkan oleh VoG

Pada skenario pengujian ini, akan dilakukan pengujian terhadap *dataset* hasil *crawling web* quora.com. Pengujian dilakukan delapan kali dengan jumlah *node* maksimum dalam GCC yang diubah-ubah. Karena setiap kali jumlah *node* maksimum dalam GCC diubah akan menghasilkan sekumpulan *subgraph* yang berbeda. Oleh karena itu, *subgraph* yang dihasilkan oleh algoritma Slashburn dapat dianalisis polanya. Dalam pengujian ini akan dilakukan dengan dua pola. Pola pertama yaitu dengan jumlah *node* maksimum dalam GCC sama dengan 5, 7, 9, dan 11. Pola yang kedua yaitu dengan jumlah *node* maksimum dalam GCC sama dengan 5, 10, 15, dan 20.

### 4.2.3 Skenario pengujian pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG

Pada skenario pengujian ini, dilakukan pengujian terhadap *dataset* untuk mengukur waktu eksekusi VoG. Karena tujuannya adalah menganalisis pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG, maka pengujian dilakukan dengan cara mengubah-ubah jumlah *node* maksimum dalam GCC. Kemudian dilakukan pengukuran terhadap waktu eksekusi VoG. Pengukuran waktu eksekusi VoG akan dilakukan sebanyak 30 kali untuk setiap jumlah *node* maksimum dalam GCC. Hal ini ditujukan untuk mendapatkan waktu rata-rata pengeksekusian VoG. Kemudian dari waktu eksekusi yang telah didapatkan akan dilakukan analisis pengaruh jumlah *node* dalam GCC terhadap waktu eksekusi VoG.

## 4.3 Analisis Hasil Pengujian

Dari skenario pengujian yang telah diuraikan pada bagian sebelumnya, dalam bagian ini akan diuraikan analisis hasil pengujian.

### 4.3.1 Analisis hasil pengujian algoritma *graph* *decomposition*

Pengujian *graph decomposition* menggunakan *sample dataset* yang dibuat sendiri. Terdapat dua *dataset* yang digunakan. Berikut adalah tampilan *graph* dari *dataset* pertama yang digunakan untuk menguji algoritma *graph decomposition*:



Gambar 4‑1: Gambar dataset-1 yang digunakan untuk pengujian algoritma graph decomposition

Setelah *graph* pada gambar 4-1 dilakukan proses *graph decomposition* menggunakan algoritma Slashburn, didapatlah keluaran sekumpulan *subgraph*.

Berikut adalah gambar keluaran algoritma Slashburn:



Gambar 4‑2: Gambar keluaran algoritma slashburn untuk graph decomposition

Setelah didapatkan sekumpulan *subgraph* hasil algoritma Slashburn, akan diidentifikasi masing-masing struktur *subgraph*nya. Untuk proses pengidentifikasian struktur, dilakukan proses *subgraph labeling* seperti yang tertera pada Perancangan Sistem. Berikut ini adalah gambar keluaran proses *subgraph labeling*:



Gambar 4‑3: Gambar keluaran proses SUBGRAPH LABELING

Dari gambar 4-3 diperoleh *subgraph* dengan struktur *perfect clique* berjumlah 3. *Subgraph* yang mempunyai struktur *perfect clique* dalam gambar 4-2 diantaranya adalah:



Gambar 4‑4: Subgraph-subgraph dengan struktur perfect clique dari gambar 4-2

Dari gambar 4-3 diperoleh *subgraph* dengan struktur *perfect star* berjumlah 4, diantaranya adalah:



Gambar 4‑5: Subgraph-subgraph dengan struktur perfect star dari gambar 4-2

Dari gambar 4-3 diperoleh *subgraph* dengan struktur *perfect bipartite* berjumlah 3, diantaranya adalah:



Gambar 4‑6: Subgraph-subgraph dengan struktur perfect bipartite dari gambar 4-2

Dari gambar 4-3 diperoleh *subgraph* dengan struktur *perfect chain* berjumlah 2, diantaranya adalah:



Gambar 4‑7: Subgraph-subgraph dengan struktur perfect chain dari gambar 4-2

Berikut adalah gambar *dataset* kedua yang digunakan:



Gambar 4‑8: Gambar dataset-2 yang digunakan untuk pengujian algoritma graph decomposition

Setelah *graph* *dataset*-2 pada gambar 4-8 dilakukan proses *graph decomposition* menggunakan algoritma Slashburn, didapatlah keluaran sekumpulan *subgraph*. Berikut adalah gambar keluaran algoritma Slashburn:



Gambar 4‑9: Gambar keluaran algoritma slashburn untuk graph decomposition. Subgraph yang diberi tanda oval adalah subgraph non-perfect structure.

Setelah didapatkan sekumpulan *subgraph* hasil algoritma Slashburn di gambar 4-9, akan diidentifikasi masing-masing strukturnya. Untuk proses pengidentifikasian struktur, dilakukan proses *subgraph labeling* seperti yang tertera pada Perancangan Sistem. Berikut ini adalah gambar keluaran proses *subgraph labeling*:



Gambar 4‑10: Gambar keluaran proses SUBGRAPH LABELING, non-perfect structure teridentifikasi.

Dari gambar 4-10 diperoleh struktur *subgraph near clique* berjumlah 1. Struktur tersebut dari gambar 4-9 adalah:



Gambar 4‑11: Subgraph dengan struktur near clique dari gambar 4-9

Dari gambar 4-10 diperoleh *subgraph* dengan struktur *near star* berjumlah 1, struktur tersebut dari gambar 4-9 adalah:



Gambar 4‑12: Subgraph dengan struktur near star dari gambar 4-9

Dari gambar 4-10 diperoleh *subgraph* dengan struktur *near bipartite* berjumlah 2, struktur tersebut diantaranya adalah:



Gambar 4‑13: Subgraph dengan struktur near bipartite dari gambar 4-9

Dari gambar 4-10 diperoleh *subgraph* dengan struktur *near chain* berjumlah 1, struktur tersebut adalah:



Gambar 4‑14: Subgraph dengan struktur near chain dari gambar 4-9

Dari gambar 4-2, algoritma Slashburn berhasil mengurai *graph* pada gambar 4-1 menjadi sekumpulan *subgraph* yang mana masing-masing *subgraph* berhasil diidentifikasi strukturnya. Struktur yang dapat teridentifikasi adalah *perfect clique, perfect bipartite, perfect star*, dan *perfect clique*.

Dari gambar 4-8, *dataset* kedua yang digunakan berhasil diurai oleh algoritma Slashburn menghasilkan sekumpulan *subgraph* seperti pada gambar 4-9. Terdapat beberapa *subgraph* yang kurang untuk menjadi *perfect structure*. Karena struktur *subgraph* tidak sempurna, maka akan diidentifikasi sebagai *near structure*. VoG dapat mengidentifikasi struktur *near clique, near star, near chain,* dan *near bipartite* seperti tertera pada gambar 4-10.

Dari kedua *sample dataset* yang digunakan, *graph* dapat diidentifikasi strukturnya sesuai dengan *vocabulary* dalam VoG.

### 4.3.2 Analisis pengaruh jumlah *node* maksimum dalam GCC terhadap *subgraph* yang dihasilkan oleh VoG

Pada pengujian VoG ini digunakan *dataset* yang didapat dari hasil *crawl web* quora.com. Hasil VoG akan ditampilkan dalam bentuk tabel. Berikut ini adalah gambar *dataset* dari quora.com yang digunakan:



Gambar 4‑15: Dataset hasil crawling quora.com dengan jumlah node 2250 dan edge 5029

Dari *graph* seperti pada gambar 4-15 yang sudah dimuat kedalam sistem, akan dilakukan *graph decomposition* untuk menghasilkan *subgraph* dengan variable jumlah *node* dalam GCC yang diubah-ubah. Berikut ini adalah gambar *graph* setelah *graph decomposition* dengan jumlah *node* maksimum dalam GCC sama dengan 3:



Gambar 4‑16: Graph quora yang sudah dilakukan graph decomposition, terdapat 1742 node dan 1321 edge

Untuk lebih jelas, dapat dilakukan *zoom* di sistem. Berikut ini adalah *graph* gambar 4-16 yang di-*zoom*:



Gambar 4‑17: Zoom Graph dari gambar 4-8

Dari hasil *graph docomposition*, masing-masing *subgraph* akan diidentifikasi strukturnya sesuai dengan *vocabulary*. Berikut ini adalah hasil identifikasi struktur *graph* gambar 4-16:



Gambar 4‑18: Hasil identifikasi struktur graph quora dengan jumlah node maksimum dalam GCC = 3

Untuk selanjutnya, hasil identifikasi struktur akan dibuat dalam bentuk tabel seperti tabel dibawah ini:

Tabel 4‑1: Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 3

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 349 |
| *Perfect* *Star* | 15 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 0 |
| *Near* *Clique* | 0 |
| *Near* *Star* | 0 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 0 |

Dari *graph* quora seperti pada gambar 4-15 dilakukan 2 pola *graph decomposition* dengan jumlah *node* maksimum dalam GCC sama dengan 5, 7, dan 9 untuk pola pertama. Untuk pola kedua akan dilakukan *graph decomposition* dengan jumlah *node* maksimum dalam GCC sama dengan 5, 10, 15 dan 20. Berikut ini adalah hasil pengujian untuk pola pertama:

Tabel 4‑2: Tabel summary hasil pengujian pola pertama

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Struktur | Jumlah *node* maksimum dalam GCC | | | |
| **5** | **7** | **9** | **11** |
| *Perfect* *Clique* | 337 | 293 | 273 | 253 |
| *Perfect* *Star* | 13 | 9 | 5 | 5 |
| *Perfect* *Bipartite* | 0 | 0 | 0 | 0 |
| *Perfect* *Chain* | 2 | 2 | 1 | 1 |
| *Near* *Clique* | 12 | 21 | 21 | 18 |
| *Near* *Star* | 21 | 47 | 47 | 50 |
| *Near* *Bipartite* | 0 | 0 | 0 | 0 |
| *Near* *Chain* | 2 | 9 | 19 | 27 |

Berikut ini adalah hasil pengujian untuk pola kedua:

Tabel 4‑3: Tabel summary hasil pengujian pola kedua

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Struktur | Jumlah *node* maksimum dalam GCC | | | |
| **5** | **10** | **15** | **20** |
| *Perfect* *Clique* | 337 | 264 | 230 | 225 |
| *Perfect* *Star* | 13 | 5 | 5 | 5 |
| *Perfect* *Bipartite* | 0 | 0 | 0 | 0 |
| *Perfect* *Chain* | 2 | 1 | 1 | 1 |
| *Near* *Clique* | 12 | 20 | 11 | 10 |
| *Near* *Star* | 21 | 49 | 38 | 34 |
| *Near* *Bipartite* | 0 | 0 | 0 | 0 |
| *Near* *Chain* | 2 | 22 | 43 | 43 |

Untuk hasil pengujian yang lebih rinci terlampir pada lampiran 1 dalam buku tugas akhir ini.

Gambar 4‑19: Grafik hasil pengujian pola pertama

Gambar 4‑20: Grafik hasil pengujian pola kedua

Dari grafik pada gambar 4-19 dan gambar 4-20 terlihat bahwa struktur *perfect clique* semakin berkurang jumlahnya seiring bertambahnya jumlah *node* maksimum dalam GCC. Struktur *near clique* dan *near star* juga semakin menurun pada jumlah *node* maksimum dalam GCC diatas 10. Penurunan ini dikarenakan bentuk GCC yang banyak tersusun dari *subgraph* *clique* dan *star* lalu ketika algoritma Slashburn beroperasi, Slashburn menghapus *hub node* dalam GCC, memotong *edge*nya dan membentuk *subgraph-subgraph near chain*. Terlihat dari grafik 4-11 dan 4-12 *subgraph near chain* bertambah jumlahnya pada jumlah *node* maksimum dalam GCC diatas 10.

### 4.3.3 Analisis pengaruh jumlah *node* maksimum dalam GCC terhadap waktu eksekusi VoG

Pada pengujian VoG ini digunakan *dataset* yang didapat dari hasil *crawl web* quora.com. Dari satu *dataset* tersebut akan dilakukan pengujian dengan mengubah-ubah jumlah *node* maksimum dalam GCC lalu mencatat waktu eksekusi VoG. Berikut ini adalah tabel hasil pengujian:

Tabel 4‑4: Hasil pengujian waktu eksekusi VoG dengan jumlah node maksimum dalam GCC = 5

| **Percobaan ke** | **Eksekusi Slashburn (ms)** | **Eksekusi *Subgraph* *Labeling* (ms)** | **Eksekusi VoG (ms)** |
| --- | --- | --- | --- |
| **1** | **10513** | **3498** | **14012** |
| **2** | **10850** | **2932** | **13782** |
| **3** | **10751** | **3166** | **13918** |
| **4** | **11254** | **3131** | **14386** |
| **5** | **11549** | **2988** | **14538** |
| **6** | **11192** | **2741** | **13934** |
| **7** | **11543** | **3402** | **14946** |
| **8** | **10665** | **3067** | **13733** |
| **9** | **12474** | **4199** | **16674** |
| **10** | **11422** | **3195** | **14617** |
| **11** | **12892** | **3741** | **16633** |
| **12** | **11973** | **3738** | **15712** |
| **13** | **11182** | **3388** | **14571** |
| **14** | **11579** | **3265** | **14844** |
| **15** | **12151** | **3218** | **15370** |
| **16** | **11773** | **3198** | **14972** |
| **17** | **11209** | **3708** | **14918** |
| **18** | **11494** | **2714** | **14208** |
| **19** | **11288** | **4006** | **15294** |
| **20** | **11265** | **3159** | **14424** |
| **21** | **12422** | **4916** | **17339** |
| **22** | **11861** | **3885** | **15747** |
| **23** | **12024** | **3335** | **15360** |
| **24** | **11738** | **3464** | **15202** |
| **25** | **11204** | **3128** | **14333** |
| **26** | **11400** | **3381** | **14782** |
| **27** | **11605** | **3649** | **15254** |
| **28** | **11169** | **3488** | **14658** |
| **29** | **11515** | **3201** | **14717** |
| **30** | **11938** | **3211** | **15149** |
| **Average** | **11529.83333** | **3403.733333** | **14934.23333** |

Untuk lebih rincinya, hasil pengujian terkait waktu eksekusi VoG terlampir pada lampiran 2 dalam buku tugas akhir ini. Berikut ini adalah tabel hasil pengujian yang diperoleh mengenai waktu rata-rata eksekusi VoG:

Tabel 4‑5: Hasil pengujian waktu rata-rata eksekusi VoG

|  |  |  |  |
| --- | --- | --- | --- |
| **Jumlah *node* maksimum dalam GCC** | **Eksekusi Slashburn (ms)** | **Eksekusi *Subgraph* *Labeling* (ms)** | **Eksekusi VoG (ms)** |
| **5** | **11529.83333** | **3403.733333** | **14934.23333** |
| **10** | **5085.966667** | **3755.133333** | **8841.6** |
| **15** | **3257.6** | **3323.433333** | **6581.466667** |
| **20** | **2375.033333** | **3472.633333** | **5848** |

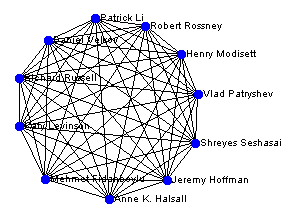
Gambar 4‑21: Grafik hasil pengujian waktu eksekusi

Dari grafik pada gambar 4-21 terlihat bahwa semakin banyak jumlah *node* maksimum dalam GCC semakin cepat juga waktu eksekusi VoG. Hal ini juga dipengaruhi oleh waktu eksekusi algoritma Slashburn. Semakin banyak jumlah *node* maksimum dalam GCC semakin cepat juga waktu eksekusi Slashburn. Hal ini dikarenakan semakin banyak jumlah *node* dalam GCC *subgraph* yang dihasilkan semakin sedikit, oleh karena itu proses untuk mengurai *graph* menjadi *subgraph* yang sedikit jumlahnya membutuhkan waktu yang lebih singkat.

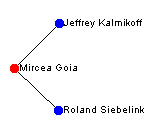
Untuk waktu eksekusi proses *subgraph labeling* dari *subgraph-subgraph* yang dihasilkan oleh Slashburn relatif stabil pada waktu 3 detik. Hal ini dikarenakan oleh jumlah banyaknya *subgraph* dan jumlah *node* yang ada pada *subgraph*. Pada jumlah *node* maksimum dalam GCC sama dengan 5, *subgraph* yang dihasilkan berjumlah lebih banyak daripada pada jumlah *node* maksimum dalam GCC sama dengan 20. Untuk memproses banyak *subgraph* dengan *node* yang sedikit waktu eksekusinya akan relatif sama dengan memproses sedikit *subgraph* dengan *node* yang banyak. Hal ini terbukti pada hasil pengujian dalam grafik di atas yang mana waktu eksekusi *subgraph* *labeling* relatif stabil pada waktu 3 detik.

### 4.3.4 Analisis informasi struktur *subgraph*

Dari hasil pengujian *graph* quora yang telah dilakukan sebelumnya, dapat diperoleh struktur *clique*, *star*, dan *chain*. Berikut ini adalah *sampling subgraph clique* dan *star* yang diambil dari hasil pengujian:



Gambar 4‑22: Subgraph clique hasil pengujian



Gambar 4‑23: Subgraph star hasil pengujian

Karena *graph* yang digunakan mempunyai representasi *node* sebagai *user* dan *edge*nya adalah relasi bahwa dua *user* menjawab pertanyaan yang sama, maka dengan metode *manual inspection* pada masing-masing *subgraph* yang dihasilkan, struktur *clique* dapat dianalisis informasinya sebagai *subgraph* yang banyak *user*nya melakukan jajak pendapat pada suatu pertanyaan tertentu. Sedangkan untuk struktur *star* dapat dianalisis informasinya bahwa terdapat satu *user* yang banyak berkontribusi menjawab banyak pertanyaan dalam *web* quora.

# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Berdasarkan hasil pengujian dapat diambil kesimpulan sebagai berikut:

1. Metode VoG dapat digunakan untuk *summarize* *graph* dan mengidentifikasi struktur *subgraph* yang dihasilkan. *Subgraph*-*subgraph* yang dihasilkan memiliki struktur *perfect* *clique*, *perfect* *star*, *perfect* *chain*, *perfect* *bipartite*, *near* *clique*, *near* *star*, *near* *chain* dan *near* *bipartite*. Struktur yang dihasilkan sesuai dengan *vocabulary* dalam VoG.
2. Jumlah *node* maksimum dalam GCC berbanding terbalik dengan jumlah struktur *perfect* *clique*, *near* *clique*, dan *near* *star* yang dihasilkan, namun berbanding lurus dengan jumlah struktur *near* *chain* yang dihasilkan. Semakin banyak *node* dalam GCC menghasilkan struktur *clique* dan *star* yang semakin sedikit namun semakin banyak menghasilkan struktur *chain*.
3. Jumlah *node* maksimum dalam GCC berbanding terbalik dengan waktu eksekusi VoG. Semakin banyak *node* dalam GCC semakin cepat waktu eksekusi VoG.
4. Struktur *subgraph* yang dihasilkan oleh VoG dapat dianalisis dengan metode *manual inspection* untuk mendapatkan informasi. Dalam tugas akhir ini, struktur *clique* menggambarkan bahwa banyak *user* di quora yang melakukan jajak pendapat dalam suatu pertanyaan. Struktur *star* menggambarkan bahwa ada satu *user* yang berkontribusi menjawab banyak pertanyaan di quora.com.

## 5.2 Saran

Berikut ini adalah saran yang untuk penelitian selanjutnya:

1. Sistem yang dikembangkan dalam tugas akhir ini tidak terbatas pada *web* *graph*. Representasi *graph* apapun dapat digunakan menjadi *dataset* dalam sistem ini. Jadi untuk selanjutnya dapat digunakan *dataset* Email *Graph* yang mungkin dapat terdapat struktur *bipartite* atau *chain* yang menggambarkan terdapat jaringan broadcast email dan email forwarding.
2. Metode Slashburn yang digunakan untuk mengurai *graph* menjadi *subgraph* masih dapat dioptimasi menggunakan algoritma Greedy untuk memilih *hub* yang terbaik untuk dihapus.
3. Metode *Subgraph* *Labeling* dapat ditingkatkan akurasi pemilihan *subgraph*nya dengan encoding *subgraph* menjadi nilai bits. Nilai hasil encode yang rendah dapat dipilih untuk menentukan struktur *subgraph*.
4. Untuk menganalisis informasi dalam *subgraph* dapat menggunakan metode lain yang lebih akurat dan objektif.

# DAFTAR PUSTAKA

|  |  |
| --- | --- |
| [1] | A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Thomkins dan J. Wiener, “Graph Structure in the Web,” *Computer Nerworks Elsevier,* vol. 33, pp. 309-320, 2000. |
| [2] | B. H. Murray dan A. Moore, “Sizing The Internet,” Cyveillance, 2000. |
| [3] | D. Koutra, U. Kang, J. Vreeken dan C. Faloutsos, “VoG : Summarizing and Understanding Large Graphs,” dalam *Proceedings of the 2014 SIAM International Conference on Data Mining*, 2014. |
| [4] | Dhia, “Finding the Patterns in a Mysterious New DGA,” [Online]. Available: https://labs.opendns.com/2013/10/24/mysterious-dga-lets-investigate-sgraph/. [Diakses 18 March 2015]. |
| [5] | I. Robinson, J. Webber dan E. Eifrem, Graph Databases, Sebastopol: O'Reilly Media, Inc, 2013. |
| [6] | K. H. Rosen, Discrete Mathematics and Its Applications, New York: The McGraw-Hill Companies, Inc., 2012. |
| [7] | M. A. Kolosovskiy, Data structure for representing a graph:, Altai State Technical University, Russia. |
| [8] | M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, G. Shen, G. Woss, C. Yang dan N. Zhang, “Unicorn: A System for Searching the Social Graph,” dalam *Proceedings of the VLDB Endowment*, Trento, 2013. |
| [9] | N. Shervashidze, S. Vishwanathan, T. H. Petri, K. Mehlhorn dan K. M. Borgwardt, “Efficient graphlet kernels for large graph comparison,” dalam *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Florida, 2009. |
| [10] | P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang dan R. Zadeh, “WTF: The Who to Follow Service at Twitter,” dalam *International World Wide Web Conference Committee (IW3C2)*, Rio de Janeiro, 2013. |
| [11] | R. Balakhisnan dan K. Ranganathan, A Textbook of Graph Theory, Springer Science & Business Media, 2012. |
| [12] | R. Kumar, P. Raghavan, S. Rajagopalan, D. Shivakumar, A. S. Thomkins dan E. Upfal, “The Web as a Graph,” dalam *ACM*, Texas, 2000. |
| [13] | Y. Lim, U. Kang dan C. Faloutsos, “SlashBurn: Graph Compression and Mining beyond Caveman Communities,” dalam *IEEE Transaction On Knowledge and Data Engineering*, 2014. |

# LAMPIRAN 1 : HASIL PENGUJIAN *SUBGRAPH* YANG DIHASILKAN OLEH VoG

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 5

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 337 |
| *Perfect* *Star* | 13 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 2 |
| *Near* *Clique* | 12 |
| *Near* *Star* | 21 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 2 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 7

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 293 |
| *Perfect* *Star* | 9 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 2 |
| *Near* *Clique* | 21 |
| *Near* *Star* | 47 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 9 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 9

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 273 |
| *Perfect* *Star* | 5 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 1 |
| *Near* *Clique* | 21 |
| *Near* *Star* | 47 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 19 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 11

| Struktur | Jumlah |
| --- | --- |
| *Perfect* *Clique* | 253 |
| *Perfect* *Star* | 5 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 1 |
| *Near* *Clique* | 18 |
| *Near* *Star* | 50 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 27 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 5

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 337 |
| *Perfect* *Star* | 13 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 2 |
| *Near* *Clique* | 12 |
| *Near* *Star* | 21 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 2 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 10

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 264 |
| *Perfect* *Star* | 5 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 1 |
| *Near* *Clique* | 20 |
| *Near* *Star* | 49 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 22 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 15

| Struktur | Jumlah |
| --- | --- |
| *Perfect* *Clique* | 230 |
| *Perfect* *Star* | 5 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 1 |
| *Near* *Clique* | 11 |
| *Near* *Star* | 38 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 43 |

#### Tabel Hasil Identifikasi Struktur dengan jumlah node maksimum dalam GCC sama dengan 20

|  |  |
| --- | --- |
| Struktur | Jumlah |
| *Perfect* *Clique* | 225 |
| *Perfect* *Star* | 5 |
| *Perfect* *Bipartite* | 0 |
| *Perfect* *Chain* | 1 |
| *Near* *Clique* | 10 |
| *Near* *Star* | 34 |
| *Near* *Bipartite* | 0 |
| *Near* *Chain* | 43 |

# LAMPIRAN 2 : HASIL PENGUJIAN WAKTU EKSEKUSI VoG

#### Tabel hasil pengujian waktu eksekusi VoG dengan jumlah node maksimum dalam GCC = 5

| **Percobaan ke** | **Eksekusi Slashburn (ms)** | **Eksekusi *Subgraph* *Labeling* (ms)** | **Eksekusi VoG (ms)** |
| --- | --- | --- | --- |
| **1** | **10513** | **3498** | **14012** |
| **2** | **10850** | **2932** | **13782** |
| **3** | **10751** | **3166** | **13918** |
| **4** | **11254** | **3131** | **14386** |
| **5** | **11549** | **2988** | **14538** |
| **6** | **11192** | **2741** | **13934** |
| **7** | **11543** | **3402** | **14946** |
| **8** | **10665** | **3067** | **13733** |
| **9** | **12474** | **4199** | **16674** |
| **10** | **11422** | **3195** | **14617** |
| **11** | **12892** | **3741** | **16633** |
| **12** | **11973** | **3738** | **15712** |
| **13** | **11182** | **3388** | **14571** |
| **14** | **11579** | **3265** | **14844** |
| **15** | **12151** | **3218** | **15370** |
| **16** | **11773** | **3198** | **14972** |
| **17** | **11209** | **3708** | **14918** |
| **18** | **11494** | **2714** | **14208** |
| **19** | **11288** | **4006** | **15294** |
| **20** | **11265** | **3159** | **14424** |
| **21** | **12422** | **4916** | **17339** |
| **22** | **11861** | **3885** | **15747** |
| **23** | **12024** | **3335** | **15360** |
| **24** | **11738** | **3464** | **15202** |
| **25** | **11204** | **3128** | **14333** |
| **26** | **11400** | **3381** | **14782** |
| **27** | **11605** | **3649** | **15254** |
| **28** | **11169** | **3488** | **14658** |
| **29** | **11515** | **3201** | **14717** |
| **30** | **11938** | **3211** | **15149** |
| **Average** | **11529.83333** | **3403.733333** | **14934.23333** |

#### Tabel hasil pengujian waktu eksekusi VoG dengan jumlah node maksimum dalam GCC = 10

| **Percobaan ke** | **Eksekusi Slashburn (ms)** | **Eksekusi *Subgraph* *Labeling* (ms)** | **Eksekusi VoG (ms)** |
| --- | --- | --- | --- |
| **1** | **4535** | **3332** | **7868** |
| **2** | **4434** | **3715** | **8150** |
| **3** | **4614** | **3747** | **8362** |
| **4** | **4665** | **3061** | **7727** |
| **5** | **5230** | **5128** | **10358** |
| **6** | **4469** | **3121** | **7591** |
| **7** | **5017** | **3128** | **8146** |
| **8** | **4967** | **4364** | **9332** |
| **9** | **5036** | **5142** | **10178** |
| **10** | **4912** | **3249** | **8162** |
| **11** | **5111** | **2855** | **7967** |
| **12** | **4931** | **5010** | **9941** |
| **13** | **4707** | **3400** | **8108** |
| **14** | **4959** | **3981** | **8940** |
| **15** | **6209** | **3800** | **10010** |
| **16** | **4945** | **3913** | **8859** |
| **17** | **4976** | **3907** | **8883** |
| **18** | **5233** | **3180** | **8414** |
| **19** | **5651** | **3976** | **9627** |
| **20** | **5708** | **3488** | **9196** |
| **21** | **5289** | **3259** | **8548** |
| **22** | **5129** | **3868** | **8997** |
| **23** | **5474** | **3644** | **9118** |
| **24** | **4299** | **3915** | **8214** |
| **25** | **5205** | **3661** | **8867** |
| **26** | **5690** | **3784** | **9474** |
| **27** | **5537** | **4021** | **9558** |
| **28** | **4461** | **3655** | **8116** |
| **29** | **5495** | **4096** | **9592** |
| **30** | **5691** | **3254** | **8945** |
| **Average** | **5085.966667** | **3755.133333** | **8841.6** |

#### Tabel hasil pengujian waktu eksekusi VoG dengan jumlah node maksimum dalam GCC = 15

| **Percobaan ke** | **Eksekusi Slashburn (ms)** | **Eksekusi *Subgraph* *Labeling* (ms)** | **Eksekusi VoG (ms)** |
| --- | --- | --- | --- |
| **1** | **3161** | **3852** | **7013** |
| **2** | **3013** | **3217** | **6230** |
| **3** | **3078** | **3446** | **6525** |
| **4** | **2477** | **2938** | **5415** |
| **5** | **3341** | **3542** | **6883** |
| **6** | **3328** | **3151** | **6480** |
| **7** | **3254** | **2895** | **6149** |
| **8** | **3431** | **2973** | **6405** |
| **9** | **3172** | **3252** | **6424** |
| **10** | **3261** | **3334** | **6595** |
| **11** | **3108** | **4558** | **7667** |
| **12** | **3544** | **2993** | **6538** |
| **13** | **3192** | **3578** | **6771** |
| **14** | **3486** | **3677** | **7163** |
| **15** | **3403** | **3030** | **6433** |
| **16** | **3004** | **3312** | **6317** |
| **17** | **2879** | **2724** | **5603** |
| **18** | **3264** | **2868** | **6132** |
| **19** | **3156** | **3630** | **6786** |
| **20** | **3552** | **2854** | **6406** |
| **21** | **3013** | **3148** | **6161** |
| **22** | **2813** | **3146** | **5960** |
| **23** | **3307** | **3379** | **6687** |
| **24** | **3208** | **2945** | **6154** |
| **25** | **3587** | **3244** | **6831** |
| **26** | **3406** | **3254** | **6660** |
| **27** | **3327** | **3600** | **6928** |
| **28** | **3537** | **3437** | **6975** |
| **29** | **3706** | **3542** | **7249** |
| **30** | **3720** | **4184** | **7904** |
| **Average** | **3257.6** | **3323.433333** | **6581.466667** |

#### Tabel hasil pengujian waktu eksekusi VoG dengan jumlah node maksimum dalam GCC = 20

| **Percobaan ke** | **Eksekusi Slashburn (ms)** | **Eksekusi *Subgraph* *Labeling* (ms)** | **Eksekusi VoG (ms)** |
| --- | --- | --- | --- |
| **1** | **2946** | **2867** | **5813** |
| **2** | **2703** | **3410** | **6113** |
| **3** | **3213** | **3370** | **6583** |
| **4** | **3061** | **3055** | **6116** |
| **5** | **1915** | **3354** | **5269** |
| **6** | **1871** | **2695** | **4566** |
| **7** | **1908** | **3147** | **5055** |
| **8** | **2519** | **3543** | **6063** |
| **9** | **2149** | **2850** | **4999** |
| **10** | **2311** | **2970** | **5282** |
| **11** | **2391** | **3403** | **5795** |
| **12** | **2113** | **3365** | **5478** |
| **13** | **2133** | **3534** | **5667** |
| **14** | **3245** | **3550** | **6796** |
| **15** | **2161** | **3587** | **5748** |
| **16** | **2289** | **4455** | **6745** |
| **17** | **2570** | **3561** | **6131** |
| **18** | **2083** | **3015** | **5098** |
| **19** | **2055** | **3363** | **5419** |
| **20** | **2156** | **3143** | **5299** |
| **21** | **2064** | **3345** | **5410** |
| **22** | **2220** | **3122** | **5342** |
| **23** | **2112** | **3507** | **5619** |
| **24** | **2256** | **3927** | **6184** |
| **25** | **2503** | **3999** | **6503** |
| **26** | **2512** | **3998** | **6510** |
| **27** | **2260** | **4538** | **6798** |
| **28** | **2046** | **3407** | **5454** |
| **29** | **2236** | **3798** | **6034** |
| **30** | **3250** | **4301** | **7551** |
| **Average** | **2375.033333** | **3472.633333** | **5848** |