ALGORITMA UNTUK MEMECAHKAN MASALAH KETERHUBUNGAN (*CONNECTEDNESS*) PADA SUATU GRAF DAN IMPLEMENTASINYA DALAM BAHASA C

Muhammad Ghifary – NIM : 13505023

Sekolah Tinggi Elektro dan Informatika
Program Studi Informatika Institut Teknologi Bandung
Jl. Ganesha no. 10, Bandung
Email : if15023@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang implementasi pemecahan masalah yang berkaitan dengan teori graf dengan menggunakan program komputer. Salah satu masalah diskrit yang paling primitif pada graf adalah **keterhubungan**(*connectedness*), yaitu apakah suatu graf merupakan graf terhubung atau graf tak-terhubung. Informasi mengenai keterhubungan suatu graf merupakan suatu informasi yang sangat penting untuk memecahkan masalah-masalah graf yang lainnnya. Beberapa teori-teori pada graf mensyarakatkan keadaan awal, yaitu suatu graf haruslah terhubung, agar kebenaran teori tersebut terpenuhi. Dalam aplikasi program komputer dengan memanfaatkan teori-teori tersebut, hal ini akan meningkatkan efisiensi kinerja program karena keterhubungan suatu graf sudah diperiksa terlebih dahulu sebelum graf tersebut akan diperiksa dengan properti-properti lainnya (mis. keplanaran) sehingga sebelum sebuah graf masuk ke mekanisme yang lebih lanjut, program sudah dapat memberikan kesimpulan yang diinginkan oleh properti tersebut.

Ada banyak algoritma yang dapat menyelesaikan masalah keterhubungan ini. Pada makalah ini, penulis membuat suatu algoritma yang menggunakan operasi penyatuan(*fusion*) simpul-simpul graf yang bertetanggaan (*adjacent vertices*) untuk memeriksa keterhubungan suatu graf. Algoritma ini akan disajikan dalam bentuk *subroutine* yang dapat digunakan berkali-kali pada program utama. Penulis mengimplementasikan *subroutine* ini dalam bahasa C.

***Kata kunci**: connectedness*, keterhubungan, *fusion*, *adjacent*, *königsberg,* graf

1. Pendahuluan

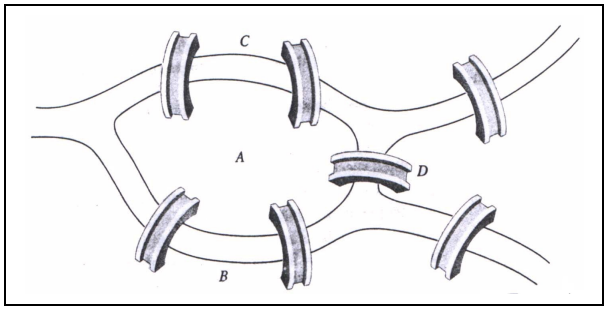
Teori graf termasuk ilmu yang usianya sudah tua namun memiliki banyak terapan hingga saat ini. Graf sendiri merupakan suatu metode untuk memecahkan masalah-masalah diskrit yang ditemui dalam kehidupan nyata. Graf digunakan untuk merepresentasi objek-objek diskrit dan hubungan antar objek tersebut. Oleh karena kesederhanaannya, graf menjadi metode yang paling sering dipakai untuk mencari solusi permasalahan dalam berbagai bidang seperti ilmu rekayasa, ilmu pengetahuan alam, sosial, linguistik, dan lain-lain.

Hampir semua objek-objek yang ada di kehidupan nyata dapat divisualisasikan dengan graf. Salah satu contohnya adalah peta. Banyak hal yang dapat dilakukan dari representasi tersebut, misalnya menentukan lintasan terpendek dari suatu kota dengan kota lain. Berikut ini 4 contoh penggunaan lain dari graf .

1. *Königsberg Bridge Problem* (Masalah Jembatan Königsberg)

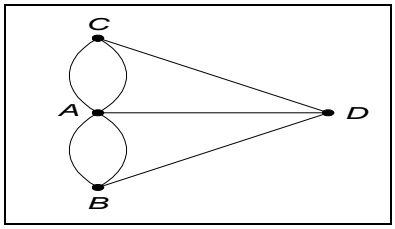
Masalah jembatan königsberg mungkin merupakan masalah yang paling populer dalam teori graf. Masalah ini tidak terpecahkan dalam waktu yang lama hingga terjawab oleh sebuah Leonhard Euler (1707-1783) pada tahun 1736. Solusi masalah tersebut ditulis Euler dalam bentuk makalah yang merupakan makalah pertama dalam sejarah teori graf. Berikut sketsa permasalahan jembatan königsberg ini

disertai representasinya dalam bentuk graf.



Gambar I-1. *Königsberg Bridge Problem*

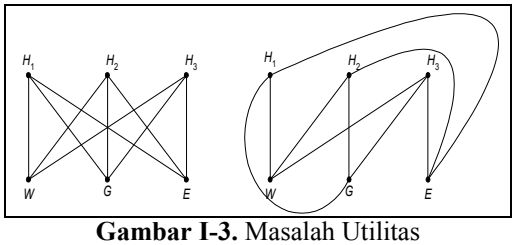
4 buah pulau, A,B,C, dan D yang dipisahkan dengan sungai, terhubung satu sama lain oleh jembatan-jembatan yang berjumlah total 7 buah. Problem yang muncul adalah bagaimana caranya bila seseorang berjalan yang dimulai dari sebarang pulau dan melewati semua pulau masing-masing tepat satu kali lalu kembali ke tempat semula (tanpa berenang melalui sungai). Euler merepresentasikan masalah ini dalam bentuk graf di bawah ini.



Gambar I-2. Graf dari *Königsberg Bridge Problem*

2. *Utilities Problem* (Masalah Utilitas)

Ada 3 rumah (Gambar I-3) yaitu H1, H2, dan H3, masing-masing dihubungkan dengan 3 utilitas, yaitu air(W), gas(G), dan Listrik(E). Permasalahan yang harus dipecahkan adalah bagaimana caranya agar ketiga saluran ini tidak berpotongan satu sama lain.



Gambar I-3. Masalah Utilitas

3. *Electrical Network Problem* (Masalah Jaringan Elektrik)

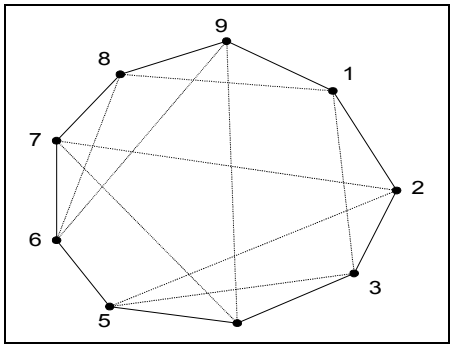
Sifat-sifat (seperti fungsi transfer dan impedansi masukan) dari jaringan listrik dipengaruhi oleh 2 faktor :

- Nilai besaran yang dimiliki oleh elemen-elemen pembentuk jaringan seperti resistor, induktor, transistor, dan lain-lain.
- Cara elemen-elemen tersebut dihubungkan (topologi dari jaringan).

Misalnya, agar impedansi masukan dari suatu jaringan listrik ingin diubah-ubah, maka kita dapat memanipulasi topologi jaringan listrik tersebut. Kita dapat menganalisis topologi dari jaringan tersebut dalam representasi graf.

4. *Seating Problem* (Masalah Tempat Duduk)

9 orang anggota dalam suatu klub melakukan pertemuan setiap hari untuk makan siang dan duduk melingkar dengan menggunakan meja lingkaran. Mereka memutuskan untuk mengatur cara duduk sedemikian rupa sehingga setiap anggota akan bertetangga dengan orang yang berbeda untuk setiap waktu makan siang. Permasalahannya adalah berapa hari lamanya susunan yang telah diputuskan tersebut bertahan.
Situasi masalah tempat duduk ini dapat direpresentasikan oleh sebuah graf (Gambar I-4) dengan 9 buah simpul yang merepresentasikan anggota dan sisi-sisi yang menghubungkan masing-masing simpul tersebut merepresentasikan hubungan yang duduk setelahnya.



Gambar I-4. Masalah Tempat Duduk

Dalam pengaplikasian graf pada program komputer, masalah keterhubungan (*connectedness*) merupakan salah satu masalah yang paling mendasar. Informasi mengenai keterhubungan suatu graf dibutuhkan untuk proses operasi-operasi lainnya yang dilakukan terhadap graf.

2. Jenis-Jenis Graf

Graf dapat dikelompokkan menjadi beberapa kategori (jenis) bergantung dari sudut pandang pengelompokannya. Pengelompokkan graf dapat dipandang berdasarkan ada tidaknya sisa ganda atau sisi kalang, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi.

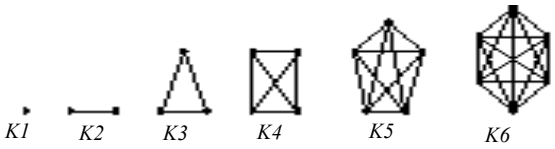
Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka graf dapat digolongkan menjadi :

2.1 Graf Sederhana (*simple graph*)

Graf sederhana merupakan graf yang tidak mengandung gelang maupun sisi ganda. Graf sederhana ini sangat dekat dengan situasi-situasi yang ada di sekitar kita. Oleh karena itu, untuk memudahkan persoalan graf sederhana dikelompokkan lagi menjadi beberapa jenis.

2.1.1 Graf Lengkap (*Complete Graph*)

Graf lengkap adalah graf sederhana yang tiap simpulnya mempunyai sisi ke semua simpul lainnya. Graf lengkap dengan *n* buah simpul dilambangkan dengan K_n. Setiap simpul pada K_n berderajat $n-1$. Berikut ini gambar enam buah buah graf lengkap.



Gambar II-1. Graf lengkap

Jumlah sisi pada graf lengkap yang terdiri dari n buah simpul adalah n(n-1)/2.

2.1.2 Graf Lingkaran

Graf lingkaran adalah graf sedehana yang setiap simpulnya berderajat dua. Graf lingkaran dengan n simpul dilambangkan dengan C_n. Jika simpul-simpul pada Cn adalah $v_1, v_2, v_3, \ldots, v_n$, maka sisi-sisinya adalah (v_1,v_2), (v_2,v_3), ... , (v_{n-1},v_n), dan (v_n,v_1). Dengan kata lain, ada sisi dari simpul terakhir, v_n, ke simpul pertama, v_1.

2.1.2 Graf Teratur (*Regular Graph*)

Graf yang setiap simpulnya mempunyai derajat yang sama disebut graf teratur. Apabila derajat setiap simpul adalah *r*, maka graf tersebut disebut sebagai graf teratur derajat *r*.

2.2. Graf Tak-sederhana (*unsimple graph*)

Graf tak-sederhana merupakan graf yang mengandung gelang maupun sisi ganda.

Berdasarkan jumlah simpul pada suatu graf, maka secara umum graf dapat digolongkan menjadi :

2.3 Graf Berhingga (*limited graph*)

Graf berhingga adalah graf yang jumlah simpulnya, n, berhingga.

2.4 Graf Tak Berhingga (*unlimited graph*)

Graf berhingga adalah graf yang jumlah simpulnya, n, tak-hingga.

Berdasarkan ada tidaknya orientasi arah pada sisi, maka graf dapat digolongkan menjadi :

2.5 Graf Tak-Berarah (*undirected graph*)

Graf tak-berarah merupakan graf yang sisinya tidak mempunyai orientasi arah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan.

2.6 Graf Berarah (*directed graph* atau *digraph*)

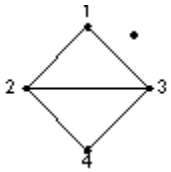
Graf berarah merupakan graf yang setiap sisinya diberikan orientasi arah. Sisi pada graf berarah biasanya disebut **busur (*arc*).**

3. Terminologi Dasar

Pada bab 3 ini kita akan membahas berbagai terminologi dasar yang berkaitan dengan graf. Berikut beberapa terminologi dasar yang sering digunakan.

3.1 Bertetangga

Definisi 3.1. Dua buah simpul pada graf tak-berarah G dikatakan bertetangga bila keduanya terhubung langsung dengan sebuah sisi. Dengan kata lain, v_i bertetangga dengan v_k jika (v_i, v_k) adalah sebuah sisi pada graf G.



Gambar III-1. G1

Pada gambar III-1, simpul 4 bertetangga dengan simpul 2 dan 3, tetapi tidak bertetangga dengan simpul 1.

3.2 Bersisian

Definisi 3.2. Untuk sembarang sisi $e=(v_i, v_k)$, sisi e dikatakan bersisian dengan simpul v_i dan simpul v_k.

Pada gambar III-1, sisi (2,3) bersisian dengan simpul 2 dan simpul 3, tetapi sisi (2,4) tidak bersisian dengan simpul 1.

3.3 Simpul Terpencil

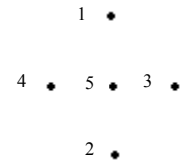
Definisi 3.3. Simpul terpencil adalah simpul yang tidak mempunyai sisi yang bersisian dengannya, atau dapat juga dinyatakan bahwa simpul terpencil adalah simpul yang tidak satupun bertetangga dengan simpul-simpul lainnya.

Pada gambar III-1, simpul 5 merupakan simpul terpencil.

3.4 Graf Kosong (*Null Graph* atau *Empty Graph*)

Definisi 3.4. Graf yang himpunan sisinya merupakan himpunan kosong disebut sebagai graf kosong dan ditulis sebagai N_n, yang dalam hal ini n adalah jumlah simpul.

Graf di bawah ini merupakan graf kosong yang disebut graf N_5.



Gambar III-2. Graf N_5

3.5 Derajat (*Degree*)

Definisi 3.5.1 Derajat suatu simpul pada graf tak-berarah adalah jumlah sisi yang bersisian dengan simpul tersebut.

Notasi: $d(v)$ menyatakan derajat simpul v

Pada gambar III-1, dapat dilihat bahwa,
$d(1) = d(4) = 2$ dan $d(2) = d(3) = 3$
$d(5) = 0$

Definisi 3.5.2 Pada graf berarah, derajat simpul v dinyatakan dengan $d_{in}(v)$ dan $d_{out}(v)$ yang dalam hal ini

- $d_{in}(v)$ = derajat-masuk = jumlah busur yang masuk ke simpul v
- $d_{out}(v)$ = derajat-keluar = jumlah busur yang keluar dari simpul v

dan

- $d(v) = d_{in}(v) + d_{out}(v)$

Pada graf berarah $G = (V,E)$ selalu berlaku hubungan

$$\sum_{v \in V} din(v) = \sum_{v \in V} dout(v) = |E|$$

Lemma Jabat Tangan. Jumlah derajat semua simpul pada suatu graf adalah genap, yaitu dua kali jumlah sisi pada graf tersebut. Dengan kata lain, jika G = (V,E) maka

3.6 Lintasan (*Path*)

Definisi 3.6. Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam graf G adalah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1,$

$e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, …, $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf.

Jika graf yang ditinjau adalah graf sederhana, maka kita cukup menuliskan lintasan sebagai barisan simpul-simpul saja :

$v_0, v_1, v_2, \ldots, v_{n-1}, v_n$

Tetapi, jika graf tersebut graf yang mengandung sisi ganda, kita harus menulis lintasan sebagai barisan berselang-seling antara simpul dan sisi menghindari kerancuan sisi mana dari sisi-sisi ganda yang dilalui.

1, e_1, 2, e_4, 3, e_5, 3

Sebuah lintasan dikatakan **lintasan sederhana** (*simple path*) jika semua simpulnya berbeda (setiap sisi hanya dilalui satu kali).

Lintasan yang berawal dan berakhir pada simpul yang sama disebut **lintasan tertutup** (*closed path*), sedangkan lintasan yang tidak berawal dan berakhir pada simpul yang sama disebut **lintasan terbuka** (*open path*).

Panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Pada gambar III-1, lintasan 2, 1, 3 memiliki panjang 2.

3.7 Siklus (*Cycle*) atau Sirkuit (*Circuit*)

Definisi 3.7. Lintasan yang berawal dan berakhir pada simpul yang sama disebut **sirkuit** atau **siklus**.

Pada gambar III-1, lintasan 2, 3, 1, 2 disebut sirkuit. **Panjang sirkuit** adalah jumlah sisi di dalam sirkuit tersebut. Sirkuit 2, 3, 1, 2 memiliki panjang 3.

3.8 Terhubung (*connected*)

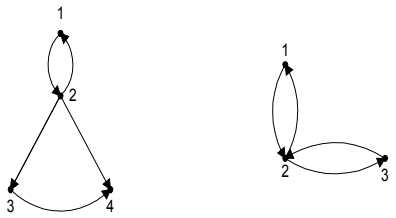
Dua buah simpul v_i dan v_j dikatakan terhubung jika terdapat lintasan dari v_i ke v_j. Jika dua buah simpul terhubung maka pasti suatu simpul dapat dicapai dari simpul lain.

Definisi 3.8.1 Graf tak-berarah G disebut **graf terhubung** (*connected graph*) jika untuk setiap pasang simpul v_i dan v_j di dalam himpunan V terdapat lintasan dari v_i ke v_j (yang juga harus berarti ada lintasan dari v_j ke v_i). Jika tidak, maka G disebut **graf tak-terhubung** (*disconnected graph*)

Pada gambar III-1, graf G1 merupakan graf tak-terhubung. Jika simpul 5 dihilangkan, maka graf G1 menjadi graf terhubung.

Definisi 3.8.2 Graf berarah G dikatakan terhubung jika graf tak-berarahnya terhubung (graf tak-berarah dari G diperoleh dengan menghilangkan arahnya)

Keterhubungan 2 buah simpul pada graf berarah dibedakan menjadi terhubung kuat dan terhubung lemah. Dua simpul, v_i dan v_j pada graf berarah G disebut **terhubung kuat** jika terdapat lintasan berarah dari v_i ke v_j dan juga sebaliknya lintasan berarah dari v_j ke v_i. Jika v_i dan v_j tidak terhubung kuat tetapi tetap terhubung pada graf tak-berarahnya, maka v_i dan v_j dikatakan **terhubung lemah**.



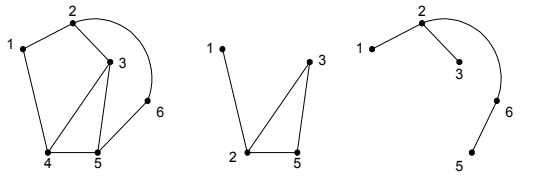
a) Graf terhubung lemah b) Graf terhubung kuat

Gambar III-3. Graf terhubung lemah dan kuat

3.9 Upagraf (*subgraf*) dan Komplemen Upagraf

Definisi 3.9.1 Misalkan G = (V,E) adalah sebuah graf. G1 = (V1,E1) adalah upagraf dari G jika $V1 \subseteq V$ dan $E1 \subseteq E$

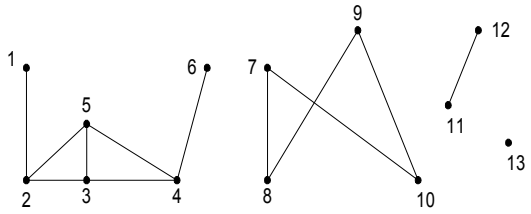
Definisi 3.9.2 Komplemen dari upagraf G1 terhadap *G* adalah graf *G2 = (V2,E2)* sedemikian sehingga *E2= E – E1* dan *V2* adalah himpunan simpul yang anggota-anggota *E2* bersisian dengannya.



a) Graf G b) Upagraf dari G c) Komplemen dari G

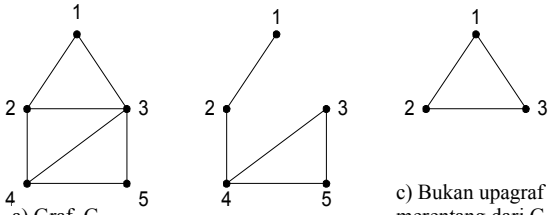
Gambar III-4

Jika graf tidak terhubung, maka graf tersebut terdiri dari beberapa komponen terhubung (*connected component*). **Komponen terhubung** adalah upagraf terhubung dari graf G yang tidak termuat di dalam upagraf terhubung dari G yang lebih besar. Ini berarti setiap komponen terhubung di dalam graf G saling lepas (*disjoint*). Pada gambar III-5 di bawah ini, graf G memiliki 4 buah komponen terhubung.



Gambar III-5. Graf tak-terhubung G

Definisi 3.9.3 Upagraf *G1= (V1,E1)* dari *G = (V,E)* dikatakan **upagraf merentang** jika *V1 = V* (yaitu mengandung semua simpul dari G)



a) Graf G b) Upagraf merentang dari G c) Bukan upagraf merentang dari G

Gambar III-6.

4. Representasi Graf

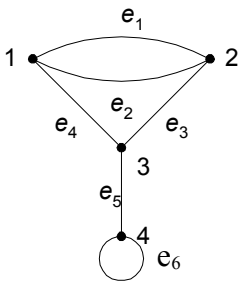
Walaupun graf dalam bentuk gambar (seperti gambar-gambar yang telah ditampilkan di atas) sangat cocok untuk studi secara visual, namun untuk aplikasi pada komputer bentuk tersebut sulit untuk diterapkan, karena graf harus direpresentasikan di dalam memori. Representasi yang sering digunakan untuk pemrosesan graf pada komputer adalah matriks. Pada makalah ini, penulis hanya menjelaskan 2 jenis dari sekian banyak representasi dengan matriks yaitu sebagai berikut.

4.1 Matriks Bersisian (*Incidence Matrix*)

Misalkan *G* merupakan graf sederhana dengan *n* buah simpul dan *e* buah sisi. Kita definisikan sebuah matriks A = [a_{ij}], dimana baris menunjukkan label simpul, dan kolom menunjukkan label sisi.

$$a_{ij}\begin{cases}1 & \text{, jika simpul } i \text{ berisian dengan simpul } j\\ 0 & \text{, jika simpul } i \text{ tidak berisian dengan simpul } j\end{cases}$$

Matriks bersisian dapat digunakan untuk merepresentasikan graf yang mengandung sisi ganda atau sisi gelang.
Derajat setiap simpul *i* dapat dihitung dengan menghitung jumlah seluruh elemen pada baris *i* (kecuali pada graf yang mengandung gelang). Jumlah elemen matriks bersisian adalah *nm*. Jika tiap elemen membutuhkan ruang memori sebesar *p*, maka ruang memori yang diperlukan seluruhnya adalah sebanyak *pnm*.



Gambar IV-1.

Pada gambar IV-1 diatas, graf tersebut dapat dinyatakan dalam matriks bersisian sebagai berikut.

$$e_1 \;\; e_2 \;\; e_3 \;\; e_4 \;\; e_5 \;\; e_6$$

Matriks di atas berbentuk matriks biner, dimana 1 menyatakan bahwa simpul *n* dihubungkan ke simpul lainnya oleh sisi e_i, sedangkan 0 menyatakan bahwa simpul n tidak dihubungkan oleh sisi e_i.

4.2 Matriks Ketetanggaan (*Adjacency Matrix*)

Matriks ketetanggaan merupakan representasi graf yang paling umum. Misalkan *G = (V,E)*

adalah graf dengan *n* simpul, $n \geq 1$. Matriks ketetanggaan adalah matriks berdimensi 2 yang berukuran *n* x *n*. Bila matriks tersebut dinamakan A = [a_{ij}], maka

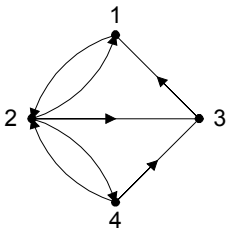
$$a_{ij} \begin{cases} 1 & \text{, jika simpul } i \text{ dan } j \text{ bertetanggaan} \\ 0 & \text{, jika simpul } i \text{ tidak bertanggaan dengan simpul } j \end{cases}$$

Karena matriks ketetanggaan hanya berisi nilai 0 dan satu, maka matriks tersebut disebut juga **matriks nol-satu**.
Jika simpul *i* memiliki sisi kalang, maka dianggap bertetanggaan dengan dirinya sendiri. Dengan demikian, $a_{ii} = 1$, untuk *i* merupakan simpul yang memiliki sisi kalang.
Untuk graf berarah, simpul *i* dan *j* dikatakan bertetanggaan bila ada satu busur yang memiliki arah dari *i* ke *j*. Sedangkan simpul *k* dikatakan tidak bertetanggaan dengan simpul *j* walaupun ada busur yang menghubungkan simpul *k* dengan *j*, tetapi tidak ada yang memiliki arah dari *k* ke *j*.

Perhatikanlah bahwa matriks ketetanggaan didasarkan pada pengurutan nomor simpul. Jadi, terdapat *n*! cara pengurutan nomor simpul, yang berarti ada *n*! matriks ketetanggaan untuk suatu graf dengan simpul *n*.

Graf tak-berarah pada gambar III-1 dapat juga dinyatakan sebagai matriks ketetanggaan berikut ini.

Untuk graf berarah di bawah ini,



Ga h G_1

matriks ketetanggaannya dapat dinyatakan sebagai berikut.

1 2 3 4

Sayangnya, matriks ketetanggaan nol-satu memiliki kelemahan yaitu tidak dapat digunakan untuk merepresentasikan graf yang mempunyai sisi ganda. Pemecahannya, kita dapat melakukan manipulasi terhadap matriks ketetanggaan tersebut. Nilai a_{ij} tidak lagi hanya terdiri dari 0 dan 1, melainkan jumlah sisi yang berasosiasi dengan (v_i, v_j).
Misalnya, graf yang terdapat pada gambar IV-1 ng memiliki sisi ganda dapat dinyatakan dalam matriks ketetanggaan sebagai berikut.

1 2 3 4

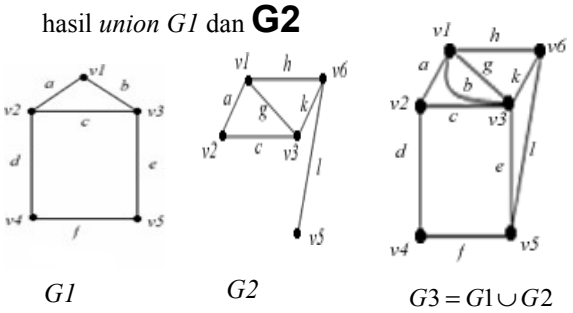
5. Operasi-Operasi pada Graf

Banyak cara yang dilakukan untuk melakukan operasi atau manipulasi terhadap graf, diantaranya adalah *union*, *decomposition*, *intersection*, *sum ring*, *deletion*, dan *fusion*.

5.1 Gabungan (*Union*)

Union merupakan penggabungan 2 buah graf menjadi 1 bentuk graf baru. Misalkan *G1 = (V1, E1)* dan *G2 = (V2,E2), G3= (V3,E3)* merupakan *union* dari *G1* dan *G2* () jika simpul dan sisi

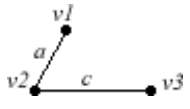
Pada gambar IV-2di bawah ini akan ditunjukkan bagaimana terbentuknya *G3* yang didapat dari hasil *union G1* dan **G2**



Gambar IV-2. Operasi *union* terhadap 2 buah graf

5.2 Irisan (*intersection*)

Sebuah graf G3 dikatakan hasil irisan dari graf G1 dan G2 () jika G3 hanya mengandung simpul-simpul dan sisi-sisi yang terdapat pada graf G1 dan G2. Graf di bawah ini merupakan hasil operasi irisan yang dilakukan terhadap graf G1 dan G2 pada gambar IV-2.



Gambar IV-3. Operasi *intersection* terhadap 2 buah graf

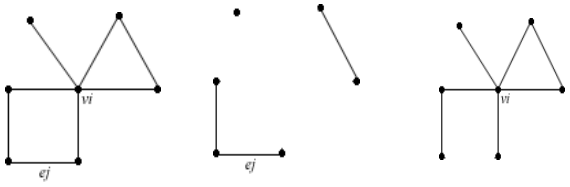
5.3 Pemecahan (*Decomposition*)

Sebuah graf G dapat dipecah menjadi 2 graf yaitu g1 dan g2 dimana,

$$g1 \cup g2 = G$$

5.4 Penghapusan (*Deletion*)

Penghapusan dapat dilakukan terhadap simpul maupun sisi. Penghapusan terhadap simpul juga akan menghapus sisi-sisi yang bersisian dengan simpul tersebut, sedangkan penghapusan terhadap sisi tidak menghapus simpul yang bersisian terhadapa sisi tersebut.



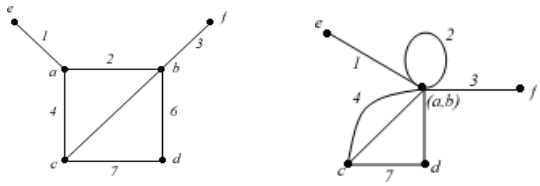
G1

G1 - v_i *G1* - v_i

Gambar IV-4. Operasi *deletion* terhadap 2 buah graf

5.5 Penyatuan (*Fusion*)

Sepasang simpul, *a* dan *b*, dikatakan bersatu jika *a* dan *b* digantikan dengan sebuah simpul baru *c*, dimana setiap sisi yang bersisian dengan simpul *a* atau *b* atau keduanya, juga bersisian dengan simpul *c*. Oleh karena itu, operasi penyatuan tidak mengubah jumlah sisi pada graf, tetapi hanya mengurangi jumlah simpul sebanyak 1 buah. Lihat gambar IV – 5 di bawah ini.



Gambar IV-5. Penyatuan simpul a dan b

6. Algoritma Status Keterhubungan

Dalam hal aplikasi graf pada program komputer, masalah keterhubungan (*connectedness*) merupakan masalah yang paling mendasar. Masalah ini merupakan pertanyaan pertama yang harus dicari solusinya pada saat akan memanipulasi sebuah graf baru. Tujuan pemecahan masalah keterhubungan ini adalah untuk meningkatkan efisiensi pemrosesan program. Sebelum graf akan diproses dengan suatu algoritma lainnya, misalnya memeriksa apakah graf tersebut planar atau tidak, maka mekanisme program akan menjadi lebih efisien bila graf tersebut sudah diketahui status keterhubungannya dan berapa banyaknya komponen yang terdapat pada graf tersebut.

Pada algoritma ini, graf akan direpresentasikan dalam bentuk matriks ketetanggaan (mis. matriks X) untuk memudahkan manipulasi. Ada banyak algoritma yang mampu memeriksa apakah suatu graf memiliki status terhubung atau tak-terhubung diantaranya, dengan menggunakan berbagai permutasi baris dan kolom yang bersangkutan terhadap matriks X, lalu memeriksa apakah matriks tersebut membentuk *block-diagonal* (tidak dijelaskan dalam makalah ini). Namun demikian, algoritma ini tidak cukup efisien karena kita harus menghitung permutasi *n!*. Dan masih banyak lagi algoritma yang \lainnya.

Penulis akan menggunakan algoritma dengan memanfaatkan operasi penyatuan (*fusion*) terhadap graf. Ide dasarnya adalah dengan mengambil satu simpul sebarang v_i, lalu disatukan (*fused*) terhadap simpul lainnnya yang bertetanggaan dengan v_i, hingga terbentuk satu simpul. Sebagai ilustrasi, lihat gambar IV-5, namun operasi penyatuan dilakukan berkali-kali hingga terbentuk satu simpul bernama *(a,b,c,d,e,f)* tanpa menghilangkan sisi-sisi yang bersisian dengan simpul-simpul graf pada waktu sebelum disatukan. Jika tidak ada lagi simpul lainnya, maka graf terhubung dengan jumlah komponen sebanyak 1 buah. Sedangkan, jika masih ada simpul yang tidak bertetanggaan dengan simpul *(a,b,c,d,e,f),* maka graf tidak

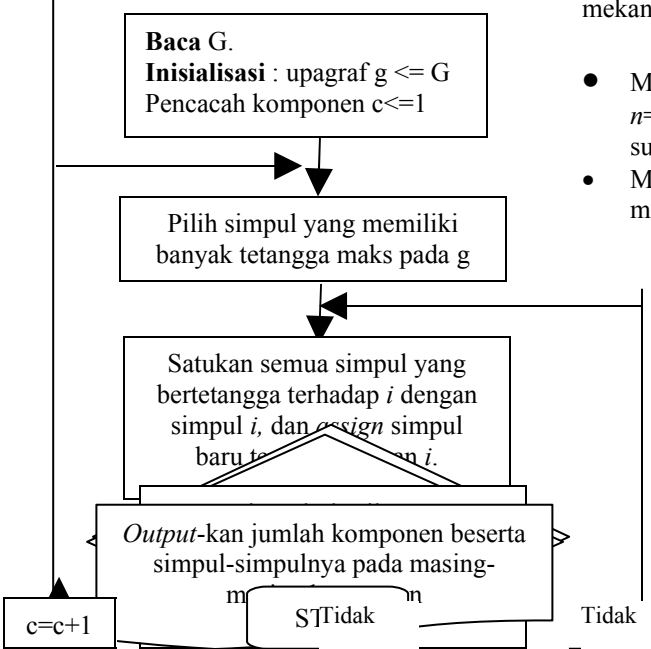
terhubung dengan jumlah komponen sebanyak *n* buah.

Pada sebuah matriks ketetanggaan X yang merepresentasikan sebuah graf, operasi penyatuan dilakukan dengan cara operasi penjumlahan biner terhadap elemen-elemen simpul ke-*i* dengan elemen-elemen simpul ke-*j* (ingat kembali operasi penjumlahan biner, yaitu 1 + 1 = 1 , 1 + 0 = 1, 0 + 1 = 1, dan 0 + 0 = 0), kemudian baris ke-*j* dengan kolom ke-*j* dihapus dari matriks. Untuk mengetahui simpul tetangga dari simpul ke-*i,* cukup dengan hanya mencari elemen yang bernilai 1 pada baris ke-*i.* Agar algoritma lebih mangkus, maka kita pilih simpul yang memiliki banyak tetangga maksimum (baris yang paling banyak memiliki nilai 1) terlebih dahulu.

Algoritma ini akan berjalan dengan baik untuk hampir semua jenis graf. Perhatikan bahwa bila ada suatu graf yang memiliki sisi ganda tetap akan dikenali sebagai sisi tunggal, karena bila sisi tersebut ganda ataupun tunggal tetap tidak akan mempengaruhi keterhubungan suatu graf.

Jumlah maksimum mekanisme penyatuan ini adalah sebanyak *n-1*, dengan *n* sebagai jumlah simpul pada graf. Dan karena setiap operasi penyatuan melakukan paling banyak *n* operasi penjumlahan biner, maka batas atas waktu eksekusi algoritma tersebut sebesar *n(n-1)*.

Berikut ini algoritma status keterhubungan yang dideskripsikan dengan diagram alir (*flow chart*)



Sebagai ilustrasi algoritma di atas, lihat mekanisme di bawah ini.

- Mis. dibaca sebuah graf G dengan simpul *n*=10 dengan nilai elemen-elemen yang sudah terdefinisi.
- Matriks X yang merepresentasikan G misalnya berbentuk seperti di bawah ini.

1 2 3 4 5 6 7 8 9 10

$$\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

- Output :
  Komponen : 1, Simpul :1,2, 3, 5, 6, 7, 8, 9, 10
  Komponen : 2, Simpul : 4

Bila kita hanya membutuhkan informasi apakah suatu graf G merupakan graf terhubung atau tak-terhubung, kita dapat membuat sebuah fungsi baru (prekondisi : algoritma di atas sudah dijalankan) dengan masukkan jumlah komponen dan keluaran nilai boolean 1 atau 0. Jika jumlah komponen adalah 1, maka fungsi menghasilkan 1 yang berarti graf G terhubung. Dan sebaliknya, jika jumlah komponen>1, maka fungsi menghasilkan 0 yang berarti graf G tidak terhubung.

Berikut ini implementasi algoritma keterhubungan yang sudah dijelaskan di atas dalam bahasa C. Program ini hanya terfokus dengan mekanisme utama dan tidak melakukan berbagai validasi terhadap input.

```
/* Nama File :  graph.c */
/* Author    : Muhammad Ghifary */
/* Deskripsi : Aplikasi Algoritma
Keterhubungan terhadap Graf */

/** HEADER **/
#include <stdio.h>
#include <stdlib.h>

/** MACRO SELECTOR **/
#define height(G) (G).height
#define width(G) (G).width
#define Mheight 20
#define Mwidth 20

/** DEFINISI GRAF **/
typedef struct{
    int info[Mheight-1][Mwidth-1]; /*
hanya berisi 1, 0, dan -99(jika kosong/
                                    t
idak ada vertex)*/
    int height;
    int width;
}Graph;


/* Definisi tidak ada vertex :
G.info[1][j]==-99 dan G.info[j][1]==-
99*/
/* Definisi graf kosong :
G.info[i][1]==-99 di mana i terdiri
dari 1,..,n */

/** KONSTRUKTOR **/
void createGraph(Graph *G,int width,
int height);
/* I.S. sembarang  */
/* F.S. Graf G terdefinisi dan kosong
*/

/** INPUT/OUTPUT  **/
void bacaGraph(Graph *G);
/* I.S. Graf G terdefinisi */
/* F.S. Graf G terisi dengan 1 atau 0*/
void tulisGraph(Graph G);
/* I.S. Graf G terdefinisi */
/* F.S. Elemen-elemen dari Graf G
tampil di layar monitor */
/*      Jika tidak ada vertex, maka
tidak ditampilkan  */


/** GETTER **/
int getInfo(Graph G, int row, int col);
/* Mengembalikan G.info[row][col]*/
int nbVertex(Graph G);
/* Mengembalikan jumlah vertex pada
graf G */


/** SETTER **/
int nbTetangga(Graph G, int i);
/* Mengirimkan banyaknya tetangga dari
vertex ke-i */
void setInfo(Graph *G, int row, int
col, int info);
/* I.S. Graf G terdefinisi */
/* F.S. G.info[row][col]=info */
void maxTetangga(Graph G,int
*vertex,int *nbMax);
/* I.S. Graf G terdefisini */
/* F.S. variabel vertex berisi indeks
verteks yang memiliki banyak */
/*      tetangga maksimum, nbMax =
banyaknya tetangga maksimum pada vertex
*/

/** OPERATOR GRAF **/
void delVertex(Graph *G,int vertex);
/* I.S. Graf G sudah terdefinisi */
/* F.S. nbComp terisi nilai yang
menyatakan jumlah komponen dari graf G
*/
/*      Output komponen beserta
vertexnya ke layar monitor */
/*      Graf G menjadi kosong */
```

```
/**  PROCEDURE  ALGORITMA  KETERHUBUNGAN
**/
void    connectedness(Graph    G,    int
*nbComp);
/* I.S. Graf G sudah terdefinisi */
```

```
/*F.S.  nbComp  terisi  nilai  yang
menyatakan jumlah komponen dari graf G
*/
/*            Output  komponen  beserta
vertexnya ke layar monitor */
/*      Graf G menjadi kosong */


int main(){
    /* Kamus */
    Graph G;
    int vertex,nbMax;
    int row,nbComp;
    /* Algoritma */
    bacaGraph(&G);
    tulisGraph(G);
    row = 1;
     printf("nbtetanga verteks ke-%d :
%d\n",row,nbTetangga(G,row));
    maxTetangga(G,&vertex,&nbMax);
      printf("Vertex yg punya tetangga
terbanyak  dengan  indeks  terkecil  :
%d\n",vertex);
         printf("dengan  jmlh  tetangga
sebanyak : %d\n",nbMax);
    printf("-- CONNECTEDNESS--\n");
    connectedness(G,&nbComp);



    getch();
    return 0;

}
void createGraph(Graph *G, int width,
int height){
/* I.S. sembarang  */
/* F.S. Graf G terdefinisi dan kosong
*/
     /* Kamus Lokal */
     int i;
     /* Algoritma */
     width(*G)=width;
     height(*G)=height;
     for(i=1;i<=height(*G);i++){
         setInfo(*G,i,1,-99);
     }
}
void bacaGraph(Graph *G){
/* I.S. Graf G terdefinisi */
/* F.S. Graf G terisi dengan 1 atau 0*/
     /* Kamus Lokal */
     int n;   /* jumlah simpul */
     int i,j; /* bil. pencacah */
     int info;
     /* Algoritma */
     do{
         printf("Masukkan jumlah simpul
n :  ");
        scanf("%d",&n);
     }while(n>Mheight);
     createGraph(G,n,n);
              printf("Isi   nilai-nilai
keterhubungan G\n");
     for (i=1;i<=n;i++){

         for(j=1;j<=n;j++){
            printf("G[%d][%d] = ",i,j);
            scanf("%d",&info);
            setInfo(G,i,j,info);
            printf("\n");
         }
         printf("\n");
     }
}
void tulisGraph(Graph G){
/* I.S. Graf G terdefinisi */
/*  F.S.  Elemen-elemen  dari  Graf  G
tampil di layar monitor */
/*        Jika tidak ada vertex, maka
tidak ditampilkan  */
    int i,j;

    for (i=1;i<=width(G);i++){
        if(getInfo(G,i,1)!=-99){
         printf("[%d]",i);
         for(j=1;j<=height(G);j++){
            if(getInfo(G,1,j)!=-99){
               printf("%d\t",getInfo(G,
i,j));
            }
         }

         printf("\n");
         }
     }
}
/** GETTER **/
int getInfo(Graph G, int row, int col){
/* Mengembalikan G.info[row][col]*/
    return (G).info[row][col];
}
int nbTetangga(Graph G, int i){
/* Mengirimkan banyaknya tetangga dari
vertex ke-i */
    /* Kamus Lokal */
    int j;
    int n;
    /* Algoritma */
    n=0;
    for(j=1;j<=width(G);j++){
        if(getInfo(G,1,j)!=-99){
            if(getInfo(G,i,j)==1){
                n++;
            }
        }
    }
    return n;
}
int nbVertex(Graph G){
/*  Mengembalikan  jumlah  vertex  pada
graf G */
    /* Kamus Lokal */
    int i,j,n;
    /* Algoritma */
    n=0;
    for(i=1;i<=height(G);i++){
          if(getInfo(G,i,1)!=-99){
            n++;
          }
    }
    return n;
}
/** SETTER **/
void setInfo(Graph *G, int row, int
col, int info){
/* I.S. Graf G terdefinisi */
/* F.S. G.info[row][col]=info */
   /* Kamus Lokal */
   /* Algoritma */
   (*G).info[row][col]=info;
}
```

```
void maxTetangga(Graph G,int *vertex,int *nbMax){
/* I.S. Graf G terdefisini */
/* F.S. variabel vertex berisi indeks verteks yang memiliki banyak */
/*          tetangga maksimum, nbMax = banyaknya tetangga maksimum pada vertex */
     int i,j;
    *nbMax = -99999;
    *vertex = 0;
     for(i=1;i<=height(G);i++){

        if(getInfo(G,i,1)!=-99){

             if(*nbMax<nbTetangga(G,i)){
                                  *nbMax = nbTetangga(G,i);
                *vertex = i;
              }
        }
     }
}

/* Operator-Operator Graph */

void delVertex(Graph *G, int vertex){
/* I.S. Graf G terdefinisi dan tidak kosong */
/* F.S. col dan row dengan indeks verteks menjadi kosong */
    setInfo(G,vertex,1,-99);
    setInfo(G,1,vertex,-99);

}

/* Procedure Algoritma Keterhubungan */
void connectedness(Graph G, int *nbComp){
/* I.S. Graf G sudah terdefinisi */
/* F.S. nbComp terisi nilai yang menyatakan jumlah komponen dari graf G */
/*            Output komponen beserta vertexnya ke layar monitor */
/*      Graf G menjadi kosong */

    /* Kamus Lokal */
    int i,j; /* bilangan pencacah */
    int vertex,nbMax;
    int komp=0;
    /* Algoritma */
    do{
        /* cari verteks yang mempunyai banyak tetangga maksimum */
       maxTetangga(G,&vertex,&nbMax);


       if(nbMax!=0){
           /* Lakukan penjumlahan biner antara verteks i dengan tetangga-tetanga
          verteks i */
          for(j=1;j<=width(G);j++){
            if(j!=vertex){
              if(getInfo(G,vertex,j)==1){

                  for(i=1;i<=height(G);i++){
                    if(getInfo(G,i,1)!=-99){
                          setInfo(&G,i,vertex,(getInfo(G,i,j)||getInfo(G,i,vertex)));
                    }
                  }
              }
            }
          }
          komp++;
          *nbComp=komp;
             printf("Komponen : %d  ,",(*nbComp));
          printf("Verteks : ");
            /* Hapus verteks i beserta tetangga2nya */
          for(i=1;i<=height(G);i++){
              if(getInfo(G,i,vertex)==1){
                 printf("%d  | ",i);
                 delVertex(&G,i);
              }
          }
          printf("\n");
       }
       else{/* nbMax=0 */
          komp++;
          *nbComp=komp;
             printf("Komponen : %d  ,",(*nbComp));
                  printf("Verteks : %d\n",vertex);
          /* Hapus verteks i */
          delVertex(&G,vertex);
       }
         if((*nbComp)==1){/*  jumlah komponen = 1*/
          printf("Graf G terhubung\n");
       }
       else{/* nbComp!=1 */
              printf("Graf G tidak terhubung\n");
       }
    }
    while(nbVertex(G)>0);
}
```

7. Kesimpulan

Kesimpulan yang dapat diambil dari studi tentang algoritma untuk memecahkan masalah keterhubungan(*connectednes*) ini adalah

1. Masalah keterhubungan(*connectedness*) merupakan masalah dalam graf yang paling mendasar. Untuk melakukan manipulasi yang lainnya pada sebuah graf, agar program berjalan lebih efisien, maka dapat digunakan algoritma status keterhubungan terlebih dahulu.
2. Algoritma status keterhubungan dengan menggunakan operasi penyatuan (*fusion*) lebih efisien dibandingkan algoritma lainnya, seperti algoritma dengan metode

membentuk matriks ke *block-diagonal* melalui permutasi baris dan kolom yang bersangkutan.

Daftar Pustaka

[1] Deo, N. (1987). Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall of Indra Private Limited.

[2] Munir, Rinaldi. (2006). Bahan Kuliah IF21523 Matematika Diskrit. Program Studi Informatika, Institut Teknologi Bandung.

[3] Liem, Inggriani (2001). Diktat Program Kecil Bahasa C. Program Studi Informatika, Institus Teknologi Bandung.

[4] Liem, Inggriani (2001). Diktat Struktur Data IF222 Kurikulum 1998. Program Studi Informatika, Institus Teknologi Bandung.