TEORI GRAPH DAN IMPLEMENTASINYA DALAM ILMU KOMPUTER

Dian Wirdasari

Program Studi Ilmu Komputer, Universitas Sumatera Utara

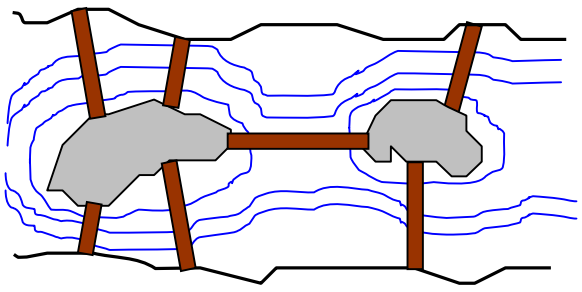
dianws@gmail.com

ABSTRAK: Makalah ini membahas tentang pokok bahasan dalam matematika diskrit yaitu teori graph dan implementasinya dalam ilmu komputer. Teori graph merupakan konsep yang sudah cukup lama dipakai dan diterapkan pada banyak bidang. Makalah ini menyajikan bagaimana tataran konseptual graph, yaitu tentang gambaran umum, definisi graph, hingga sampai pada tataran implementasi, yaitu bagaimana konsep tersebut diterapkan dalam bidang ilmu komputer khususnya dalam Struktur Data dan menentukan minimum spanning tree (MST) yang banyak diaplikasikan dalam masalah TSP (Traveling Salesman Problem).

Kata Kunci: *graph, struktur data, TSP, traveling salesman problem*

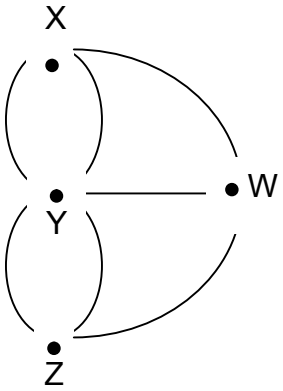
A. PENDAHULUAN

Teori graph diperkenalkan pada abad ke 18 oleh seorang matematikawan bernama Leonhard Euler. Euler mencoba memecahkan teka-teki yang dikenal dengan nama **Masalah Jembatan Konigsberg**. Terdapat tujuh buah jembatan yang menghubungkan dua pulau dan sebuah sungai, seperti yang ditunjukkan pada Gambar 1. Akan dicari sebuah lintasan yang melewati setiap jembatan tepat satu kali.



Gambar 1. Jembatan Konigsberg

Sebuah metode untuk mencari solusi dari masalah ini adalah dengan membentuk model dari jembatan Konigsberg yang dikenal sebagai **multigraph**, diperlihatkan pada Gambar 2. Sebuah multigraph memiliki dua elemen yaitu himpunan verteks (titik/node) dan himpunan edge (garis) yang menghubungkan antar verteks.



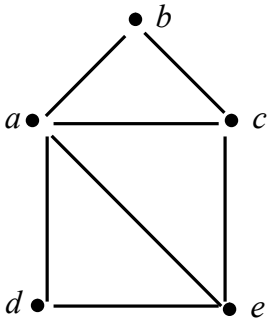
Gambar 2. Representasi Multigraph Jembatan Konigsberg

Titik-titik yang diberi label X, Y, Z, dan W pada Gambar 2 itulah yang disebut **verteks**, dan garis yang menghubungkan antar titik itulah

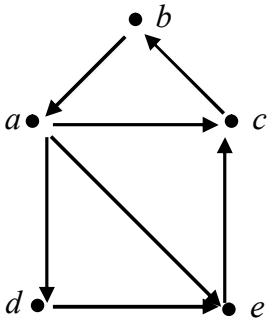
yang disebut dengan **edge**. Euler menetapkan sebuah aturan yang bisa dipakai disemua multigraph, untuk mencari solusi dari masalah pada jembatan Konigsberg, aturan ini disebut dengan **Eulerian path**, yang berbunyi:

> *"Andaikan kita mempunyai sebuah multigraph sehingga untuk beberapa pasang verteks terdapat sebuah path (lintasan) diantara verteks-verteks tersebut. Multigraph tersebut memiliki Eulerian path jika dan hanya jika terdapat 0 atau 2 verteks yang mana banyak edge yang meninggalkan verteks tersebut berjumlah ganjil."*

Multigraph pada jembatan Konigsberg memiliki empat verteks, yang mana keempat verteks tersebut memiliki edge yang meninggalkan verteks tersebut berjumlah ganjil. Maka multigraph jembatan Konigsberg tidak memiliki Eulerian path. Multigraph yang ditunjukkan pada Gambar 3a tidak memiliki panah, sehingga disebut dengan **undirected graph (graph tak berarah)**. Sebaliknya, multigraph yang memiliki panah disebut dengan **directed graph (graph berarah)** (Gambar 3b).



Gambar 3a. Graph Tak Berarah



Gambar 3b. Graph Berarah

Definisi 1. Sebuah simple graph (undirected graph) adalah pasangan dari $G = (V, E)$ dimana:

1. V adalah himpunan berhingga dari elemen yang disebut verteks
2. E adalah sebuah relasi yang **irrefleksif** dan **simetri** pada V.

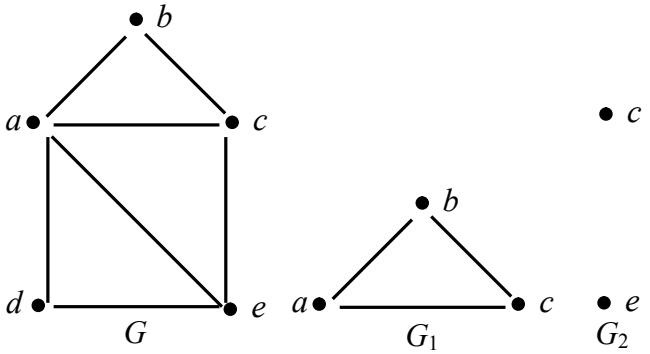
Pasangan berurutan pada E disebut edge dari graph. Lebih spesifik, jika $e = (u, v) \in E$, dikatakan bahwa edge e adalah antara u dan v (dan juga antara v dan u), dan dikatakan bahwa u **adjacent ke** v. Lebih jauh, dapat dikatakan bahwa e **incident ke** u (dan juga v). Karena E simetri, maka kita dapat menotasikan e sebagai pasangan tak berurut $\{u, v\}$.

Andaikan $G = (V, E)$ sebuah graph. Dengan u, v verteks. **Degree** dari v, dinotasikan dengan $d(v)$, adalah jumlah edge yang incident ke v. Karena sebuah edge harus incident ke dua verteks, maka muncullah Teorema 1.

Teorema 1. Andaikan $G = (V, E)$

$$\sum_{v \in V} d(v) = 2|E|$$

Sebuah subgraph dari graph $G = (V, E)$ adalah sebuah graph $G' = (V', E')$ sehingga $V' \subseteq V$ dan $E' \subseteq E$. Subgraph $G' = (V', E')$ disebut sebagai **spanning** subgraph jika $V = V'$. Gambar 4 menunjukkan dua subgraph, G_1 dan G_2 dari graph G.



Gambar 4. Graph G dan dua subgraphnya G_1 dan G_2

Sebuah graph $G = (V, E)$ dikatakan komplit jika untuk setiap verteksnya $u, v \hat{I} V \{u, v\} \hat{I} E$. Sebuah komplit graph dengan n verteks dinotasikan sebagai K_n. Setiap graph adalah spanning subgraph dari sebuah komplit graph.

B. PATH, CYCLE, DAN GRAPH TERHUBUNG

Sebuah path (lintasan) yang panjangnya k pada suatu graph adalah barisan dari verteks-verteks v_o, v_1, K, v_k sehingga untuk $i = 1, 2, \text{K}, k, \{v_{i-1}, v_i\} \hat{I} E$. Barisan c, a, d, e adalah sebuah path dengan panjang (length) 3 pada graph G pada Gambar 4.

Sebuah path pada sebuah graph tak berarah (undirected graph) disebut sebagai **cycle** (sirkuit) jika verteks awal dan verteks akhirnya sama, dan tidak ada edge yang berulang pada path tersebut. Jadi, sebuah cycle harus mempunyai sekurang-kurangnya tiga edge.

Graph yang tidak memiliki cycle disebut dengan graph **acyclic** (asiklik). Path a, b, c, e, a adalah sebuah cycle pada graph G Gambar 4. Sebuah cycle tidak boleh memiliki edge yang berulang, tetapi boleh memiliki verteks yang berulang. Sebagai contoh, path a, b, c, a, e, d, a adalah cycle pada graph Gambar 4. Subgraph G_2 pada Gambar 4 adalah acyclic. Sebuah graph dikatakan **connected** (terhubung) jika terdapat sebuah path antara setiap pasangan verteks-verteksnya. Graph G dan G_1 pada Gambar 4 adalah connected, tetapi G_2 tidak.

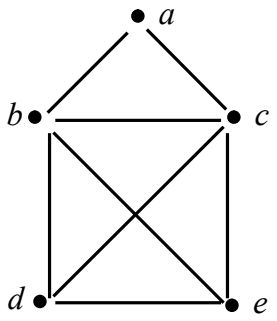
C. EULERIAN PATH

Teorema 2. Sebuah graph terhubung (connected) dengan sekurang-kurangnya dua verteks memiliki Eulerian path jika dan hanya jika terdapat 0 atau 2 verteks yang berdegree ganjil. Sebuah path adalah cycle jika dan hanya jika setiap verteksnya mempunyai degree genap.

Sebuah Eulerian path, yang pathnya adalah cycle sehingga verteks awal dan akhirnya adalah sama disebut sebagai Eulerian cycle (Eulerian Sirkuit).

Contoh 1

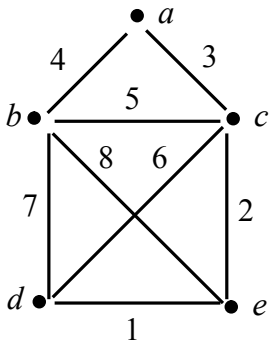
Tunjukkanlah graph pada Gambar 5 memiliki Eulerian path tetapi tidak memiliki Eulerian sirkuit. Tentukanlah Eulerian pathnya.



Gambar 5. Graph untuk contoh 1

Penyelesaian:

Karena terdapat dua verteks (d dan e) yang berdegree ganjil, sesuai dengan teorema, maka terdapat sebuah Eulerian path, tetapi bukan Eulerian cycle. Untuk menentukan Eulerian pathnya, kita harus memulai dari verteks d atau e. Jika dimulai dari d, diperoleh path $d, e, c, a, b, c, d, b, e$. Gambar 6 menunjukkan Eulerian path, dengan setiap edgenya diberi label sesuai dengan jalur pathnya.



Gambar 6. Graph dengan Eulerian path

D. HAMILTONIAN CYCLE

Masalah yang sama untuk mencari Eulerian sirkuit adalah masalah untuk mencari Hamiltonian sirkuit (Hamiltonian cycle).

Sebuah **Hamiltonian sirkuit** adalah sebuah cycle pada sebuah graph yang mana setiap verteksnya dilalui tepat satu kali. Sebuah graph dengan Hamiltonian sirkuit maka graph tersebut akan connected (terhubung), tetapi tidak semua graph yang terhubung memiliki Hamiltonian sirkuit.

Masalah menjadi lebih sulit jika edge pada graph diberi tanda dengan bobot (weight) yang menunjukkan jarak atau biaya perjalanan, maka kemudian mencari Hamiltonian sirkuitnya dan menghitung total biaya minimumnya. Masalah ini disebut dengan **traveling salesman problem**. Salah satu algoritma yang digunakan untuk mencari solusi dari traveling salesman problem adalah *nearest neighbor method* (metode tetangga terdekat), dengan asumsi bahwa graph nya terhubung (connected).

Nearest Neighbor Method untuk Traveling Salesman Problem

begin

Pilih sebarang $v1 \hat{I} V$.

$v' \neg v1$

$w \neg 0$

Tambahkan v' ke daftar verteks dalam path.

while verteks yang tidak bertanda bersisa **do**

begin

Tandai v'.

Pilih sebarang verteks yang tidak bertanda, u, yang terdekat dengan v'

Tambahkan u ke daftar verteks dalam path

$w \neg w$ + bobot dari edge $\{v', u\}$

$v' \neg u$

end

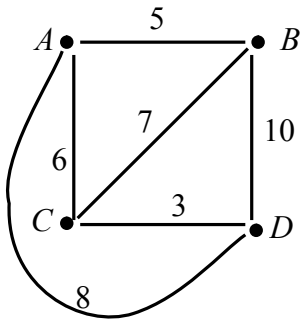
Tambahkan $v1$ ke daftar verteks dalam path.

$w \neg w$ + bobot dari edge $\{v', v1\}$

end.

Contoh 2

Gunakan algoritma *nearest neighbor* pada graph berbobot berikut.

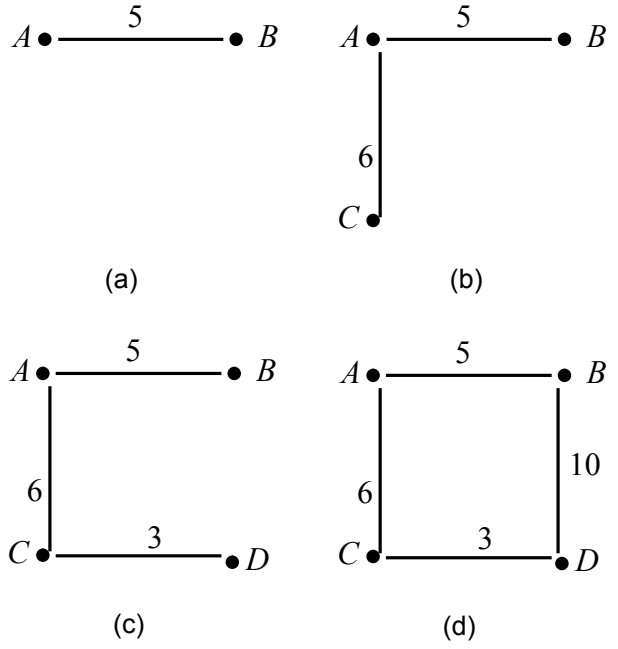


Gambar 7. Graph untuk Contoh 2

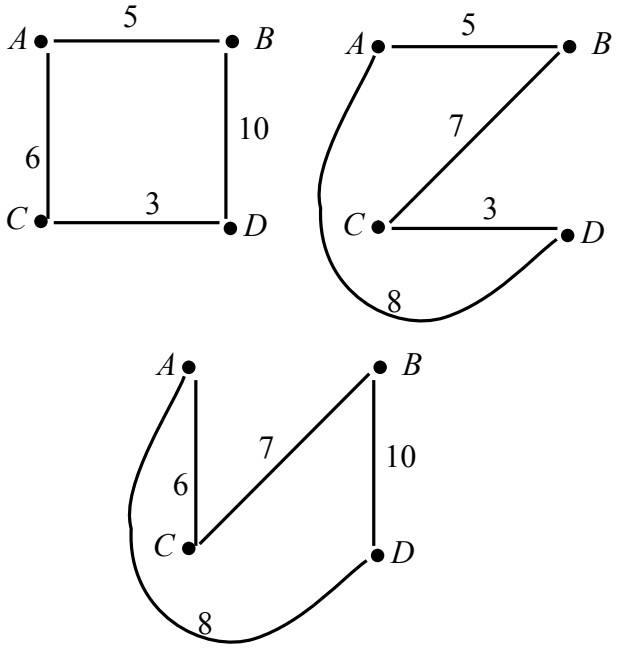
Penyelesaian:

Dimulai dari B. Beri nilai awal $w = 0$ dan daftar verteks dalam path adalah B. Verteks yang tidak bertanda yang terdekat dengan B adalah A, jadi tambahkan A ke dalam path, seperti diperlihatkan pada Gambar 8(a), dan tandai B. Kemudian tambahkan 5 ke w. Verteks yang tidak bertanda yang terdekat dengan A adalah C, jadi tambahkan C ke dalam path, seperti diperlihatkan pada Gambar 8(b), dan tandai A. Tambahkan w dengan 6. Verteks yang tidak bertanda yang terdekat dengan C adalah D, jadi tambahkan D ke dalam path, seperti diperlihatkan pada Gambar 8(c), dan tandai C. Kemudian tambahkan 3 ke w.

Karena tidak ada sisa verteks yang tidak bertanda, maka cycle akan terpenuhi dengan menambahkan verteks B ke dalam path. Kemudian tambahkan w dengan 10. Hasil sirkuitnya diperlihatkan pada Gambar 8(d). Bobot dari sirkuitnya adalah 24. Gambar 9 memperlihatkan semua ketiga buah Hamiltonian sirkuit yang dapat dibentuk dari graph tersebut, dan yang paling baik adalah yang memiliki total bobot 23.



Gambar 8. Penerapan algoritma nearest neighbor



Gambar 9. Hamiltonian sirkuit dari graph Contoh 2

E. STRUKTUR DATA GRAPH

Dalam bidang ilmu komputer, sebuah graph dapat dinyatakan sebagai sebuah struktur data, atau secara spesifik dinamakan sebagai ADT (abstract data type) yang terdiri dari kumpulan simpul dan sisi yang membangun hubungan antar simpul. Konsep ADT graph ini merupakan turunan konsep graph dari bidang kajian matematika.

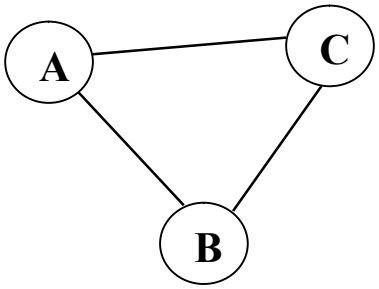
Pokok bahasan sebelumnya menjelaskan bahwa graph menampilkan visualisasi data dan hubungannya. Sedangkan jika berbicara masalah implementasi struktur data graph itu sendiri, isu utama yang dihadapi adalah bagaimana informasi itu disimpan dan dapat diakses dengan baik, ini yang dapat disebut dengan representasi internal.

Secara umum terdapat dua macam representasi dari struktur data graph yang dapat diimplementasi. Pertama, disebut *adjacency list*, dan diimplementasi dengan menampilkan masing-masing simpul sebagai sebuah struktur data yang mengandung senarai dari semua simpul yang saling berhubungan.

Yang kedua adalah representasi berupa *adjacency matrix* dimana baris dan kolom dari matriks (jika dalam konteks implementasi berupa senarai dua dimensi) tersebut merepresentasikan simpul awal dan simpul tujuan dan sebuah entri di dalam senarai yang menyatakan apakah terdapat sisi di antara kedua simpul tersebut.

1. Adjacency List

Dalam teori graph, *adjacency list* merupakan bentuk representasi dari seluruh sisi atau busur dalam suatu graph sebagai suatu senarai. Simpul-simpul yang dihubungkan sisi atau busur tersebut dinyatakan sebagai simpul yang saling terkait. Dalam implementasinya, hash table digunakan untuk menghubungkan sebuah simpul dengan senarai berisi simpul-simpul yang saling terkait tersebut.



Gambar 10. Undirected Cyclic Graph

Graph pada gambar 10 dapat dideskripsikan sebagai senarai {a,b},{a,c},{b,c}. Dan representasi adjacency list dapat digambarkan melalui tabel di bawah ini.

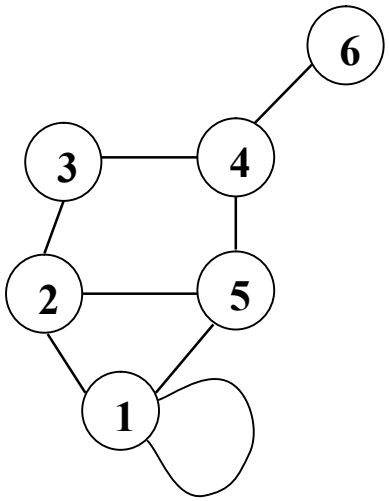
Tabel 1. Representasi Adjacency List

Vertex	Adjacency	Array of Adjacent Vertices
a	adjacent to	b,c
b	adjacent to	a,c
c	adjacent to	a,b

Salah satu kekurangan dari teknik representasi ini adalah tidak adanya tempat untuk menyimpan nila yang melekat pada sisi. Contoh nilai ini antara lain berupa jarak simpul, atau beban simpul.

2. Adjacency Matrix

Adjacency Matrix merupakan representasi matriks nxn yang menyatakan hubungan antar simpul dalam suatu graph. Kolom dan baris dari matriks ini merepresentasikan simpul-simpul, dan nilai entri dalam matriks ini menyatakan hubungan antar simpul, apakah terdapat sisi yang menghubungkan kedua simpul tersebut. Pada sebuah matriks nxn, entri non-diagonal a_{ij} merepresentasikan sisi dari simpul i dan simpul j. Sedangkan entri diagonal a_{ii} menyatakan sisi kalang (loop) pada simpul i.



Gambar 11. Graph tak berarah berlabel

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

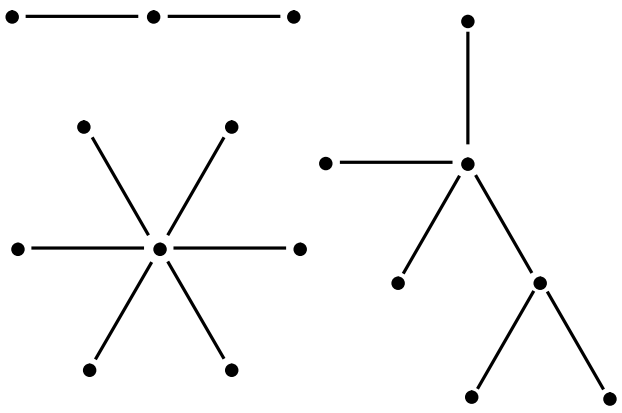
Gambar 12. Adjacency matrix

Gambar 12 merupakan adjacency matrix yang berkorelasi dengan graph tak berarah pada gambar 11. Kolom dan baris pada matriks merupakan simpul-simpul berlabel 1-6.

Kelebihan dari *adjacency matrix* ini adalah elemen matriksnya dapat diakses langsung melalui indeks, dengan begitu hubungan ketetanggan antara kedua simpul dapat ditentukan dengan langsung. Sedangkan kekurangan pada representasi ini adalah bila graph memiliki jumlah sisi atau busur yang relatif sedikit, karena matriksnya bersifat jarang yaitu hanya mengandung elemen bukan nol yang sedikit. Kasus seperti ini merugikan, karena kebutuhan ruang memori untuk matriks menjadi boros dan tidak efisien karena komputer menyimpan elemen 0 yang tidak perlu.

F. TREE

Sebuah graph $G = (V, E)$ disebut **tree** (pohon) jika G terhubung (connected) dan acyclic (tidak memiliki cycle). Gambar 13 memperlihatkan contoh tiga buah graph yang ketiga-tiganya adalah tree.



Gambar 13. Contoh tree

Teorema 3. Kalimat berikut ini adalah ekivalen untuk graph $G = (V, E)$ dengan n verteks dan m edge.

(1) G adalah tree.
(2) Terdapat tepat sebuah path antara semua verteks pada G.
(3) G terhubung (connected) dan $m = n - 1$.
(4) G terhubung (connected) dan menghapus salah satu edge menyebabkan G tidak terhubung (disconnected).
(5) G acyclic (tidak memiliki cycle) dan $m = n - 1$.
(6) G acyclic (tidak memiliki cycle) dan penambahan sebuah edge menyebabkan terbentuknya cycle.

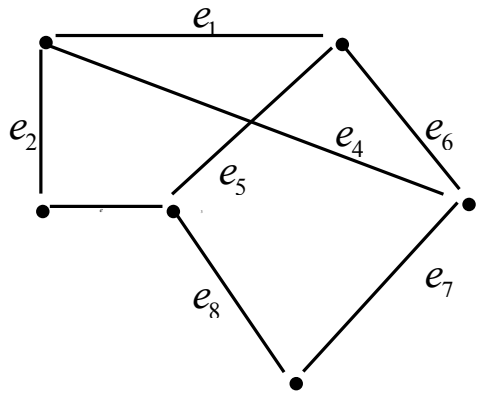
Teorema di atas dapat digunakan untuk memperlihatkan bahwa setiap graph yang connected (terhubung) berisi sebuah *spanning tree* (tree yang merentang), yang mana sebuah spanning tree dari sebuah graph adalah sebuah tree.

Terdapat dua cara untuk membentuk sebuah spanning tree dari sebuah graph. Pertama mulai dengan menghilangkan beberapa edge sehingga pada akhirnya akan menghasilkan sebuah subgraph yang tetap terhubung (connected), sehingga jika dihapus salah satu edge akan menyebabkan subgraph menjadi tidak terhubung. Sesuai dengan bagian (4) dari Teorema 3, subgraph akhir ini adalah sebuah spanning tree. Perhatikan bahwa, spanning tree akan memiliki jumlah edge sebanyak $n - 1$ edge, dengan n adalah jumlah verteks yang dimiliki oleh graph semula.

Cara yang kedua, mulai menggambarkan graph dengan tanpa edge (hanya verteks-verteksnya saja), dan tambahkan edge satu persatu sehingga setiap verteksnya terhubung, dan tidak terdapat cycle, sehingga jika menambahkan salah satu edge akan menyebabkan terbentuknya cycle. Graph yang dibentuk disebut dengan tree.

Contoh 3

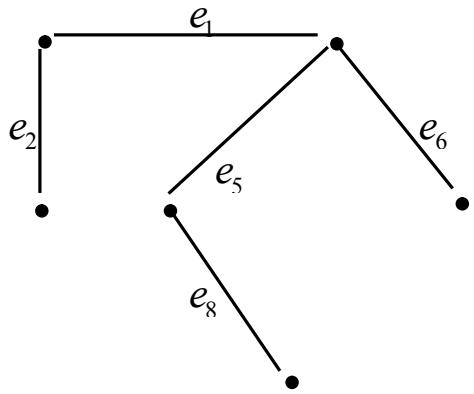
Gunakan cara pertama di atas untuk mencari sebuah spanning tree dari graph pada Gambar 14 berikut.



Gambar 14. Graph untuk Contoh 3

Penyelesaian:

Karena memiliki 8 buah edge dan enam verteks, maka kita harus menghilangkan tiga edge. Selanjutnya, sebarang edge dapat dihapus. Pertama, hapuslah e_3. Kemudian sebarang edge yang lain, tetapi bukan e_2, hapuslah e_4. Selanjutnya sebarang edge boleh dihapus tetapi bukan e_1 dan e_2, hapuslah e_7. Kita sampai di keadaan dimana jika menghapus salah satu edge akan menyebabkan graph menjadi tidak terhubung (disconnect), subgraph hasil inilah yang disebut spanning tree, ditunjukkan pada Gambar 15. Subgraph tersebut mempunyai lima buah edge, sesuai dengan pernyataan di bagian (3) dari Teorema 3.



Gambar 15. Spanning tree untuk graph Contoh 3

1. Minimal Spanning Tree

Minimal spanning tree (disebut juga dengan **MST**) adalah mencari sebuah spanning tree dengan jumlah bobot (weight) minimal dari sebuah graph yang terhubung (connected).

Masalah ini sama seperti pada traveling salesman problem. Algoritma untuk mencari minimal spanning tree adalah sebagai berikut.

Algoritma untuk Mencari Minimal Spanning Tree

begin

e ¬ sebuah edge di E dengan bobot paling kecil.

T ¬ {e}

E' ¬ E - {e}

while E' ¹ Æ

begin

e' ¬ sebuah edge Î E' dengan bobot paling kecil.

T ¬ T È {e'}

E' ¬ himpunan edge di E' - T yang penambahannya ke dalam T tidak menyebabkan terbentuknya cycle

end

end.

Sebuah rooted tree adalah sebuah tree dengan himpunan verteks V sehingga sebuah verteks v Î V ditandai sebagai akar (root) dari tree tersebut. Untuk sebarang verteks u pada tree, terdapat path dari v ke u. Panjang path ini disebut tingkatan (**level**) verteks u. Verteks pada level 1, menunjukkan verteks tersebut adjacent ke root, disebut sebagai **descendant** (keturunan/anak) dari root. Selanjutnya, descendant dari root dapat dianggap sebagai root bagi subtree yang diperoleh dengan menghapus root dari tree.

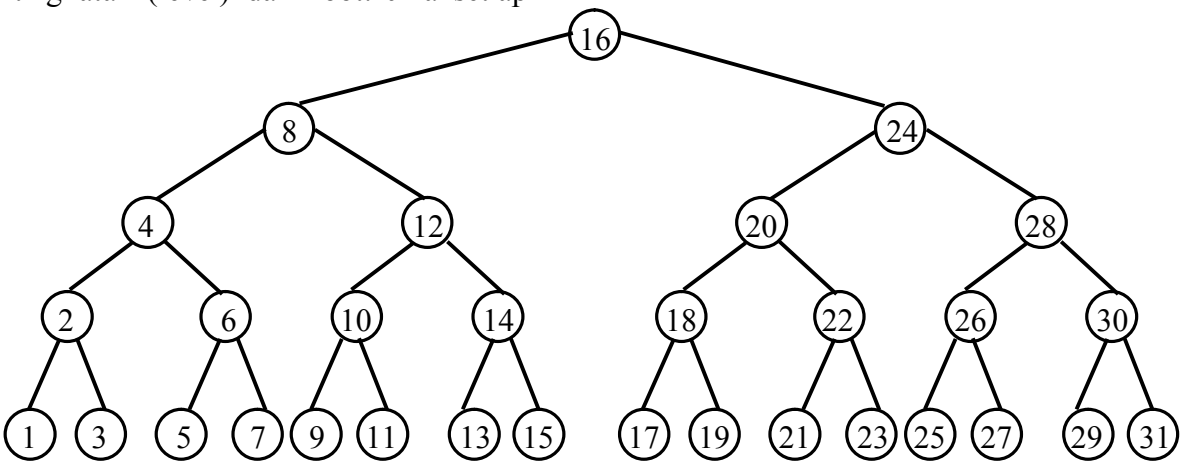
Panjang dari path yang terpanjang dari root disebut tingkatan (level) dari root. Jika setiap verteks pada rooted tree mempunyai dua subtree, maka tree yang seperti ini disebut dengan **binary tree**. Rooted tree biasanya digambarkan dengan root di atasnya dan subtree terbuka ke bawah ke arah daun (**leave**), yang merupakan verteks terjauh dari root. Leave harus berdegree 1.

Contoh 4

Terdapat sebuah permainan sederhana sebagai berikut: Seseorang memikirkan sebuah angka antara 1 sampai 31. Anda harus menebak angka dengan benar. Anda bertanya, ”Apakah angkanya x?” kemudian orang tersebut menjawab dengan ”Ya”,”Lebih kecil dari x”, atau ”Lebih besar dari x”. Tunjukkan bahwa Anda mampu menebak angka tersebut tidak lebih dari 5 kali tebakan.

Penyelesaian:

Petunjuknya adalah dengan selalu menebak angka yang menjadi titik tengah dari jangkauan angka yang tersisa. Kemudian, jika tebakan salah akan mengurangi separuh angka, hingga akhirnya akan tersisa satu angka. Gambar 16 memperlihatkan bagaimana proses tebakan berlangsung, mulai dari 16. setiap verteks adalah titik yang memutuskan nilai benar atau salah, jika salah maka nilai tersebut berada di salah satu subtree dari dua subtree. Subtree pada sisi kiri berisi nilai yang lebih kecil, dan subtree pada sisi kanan berisi nilai yang lebih besar. Tree yang terbentuk hanya empat level, maka diperlukan tidak lebih dari 5 kali tebakan.



Gambar 16. Tree pada Contoh 4

2. Rooted Tree dalam Pembentukan Folder

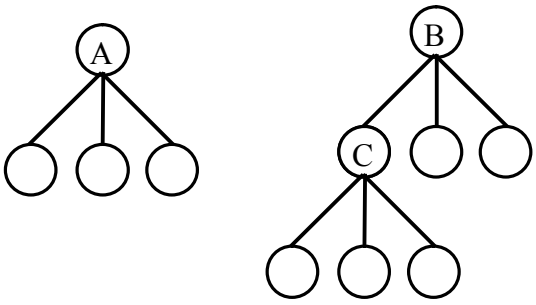
Suatu pohon dikatakan N-ary jika setiap simpul cabangnya mempunyai paling banyak n-buah anak. Karena itu pohon dengan hanya berupa sebuah simpul dapat kita sebut juga sebagai pohon N-ary namun tanpa anak. Kita dapat mengimplementasikan pohon dengan notasi himpunan sebagai berikut:

Ta = {A, Ø , Ø , Ø }

Tb = {B, {C, Ø, Ø, Ø}, Ø, Ø}

Ø = null

Ta menyatakan pohon dengan satu buah akar (A) dan simpul dengan 3 anak null. Tb menyatakan pohon dengan dua buah simpul (B dan C). Nilai null menyatakan bahwa pohon tersebut kosong atau tidak mempunyai anak. Null juga dapat berarti daun. Representasi pohon dari notasi himpunan tersebut adalah pada Gambar 17 berikut.



Gambar 17. Representasi Pohon Biner

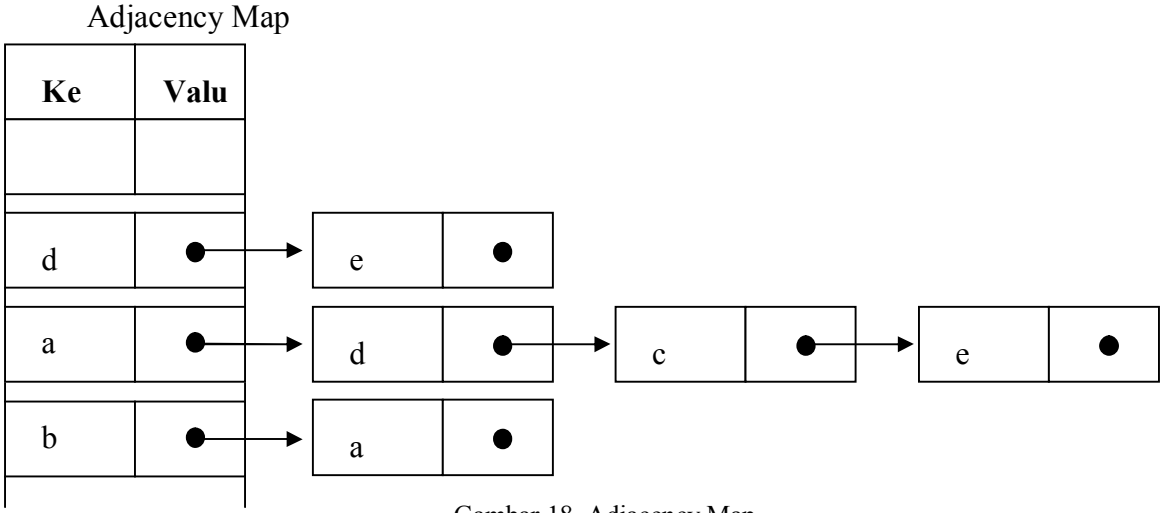
Pohon Biner (Binary Tree) adalah pohon n-ary dengan n = 2. Setiap simpul di dalam pohon biner mempunyai paling banyak 2 buah anak. Dibedakan antara anak kiri (left child) dan anak kanan (right child). Karena ada perbedaan urutan anak, maka pohon biner adalah pohon terurut.

G. PEMBAHASAN

1. Implementasi Pada Struktur Data

Representasi internal yang dipakai dalam struktur data graph ini merupakan implementasi dari representasi senarai ketetanggaan dengan menggunakan sebuah *hashmap*. Simpul disimpan sebagai kunci dalam sebuah *Map structure* (struktur pemetaan) dengan tujuan agar mempermudah pencarian sebuah simpul. Struktur pemetaan ini selanjutnya disebut sebagai *adjacency map.* Sisi yang berawal dari setiap simpul disimpan sebagai senarai simpul yang berhubungan. Senarai ini disimpan sebagai nilai yang memiliki kaitan dengan kunci yang sesuai dalam *adjacency map.* Sebagai contoh digunakan graph pada gambar 3b, dengan adjacency map pada Gambar 18. Representasi dari graph di atas akan terdiri dari sebuah pemetaan dengan masukan sebagai berikut:

Key : "a"
Value : ["c", "d", "e"]



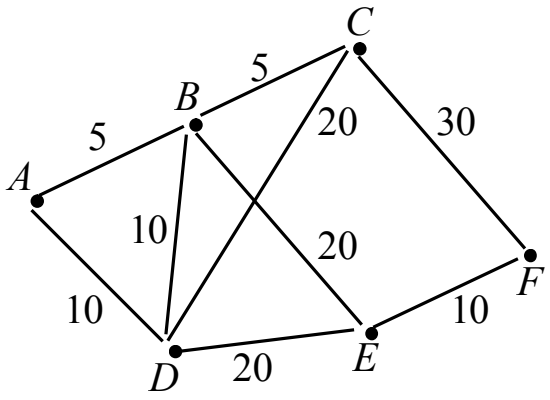
Gambar 18. Adjacency Map

2. Implementasi Pada MST

Implementasi graph dalam menentukan minimum spanning tree (MST) dipaparkan pada contoh 5 berikut.

Contoh 5

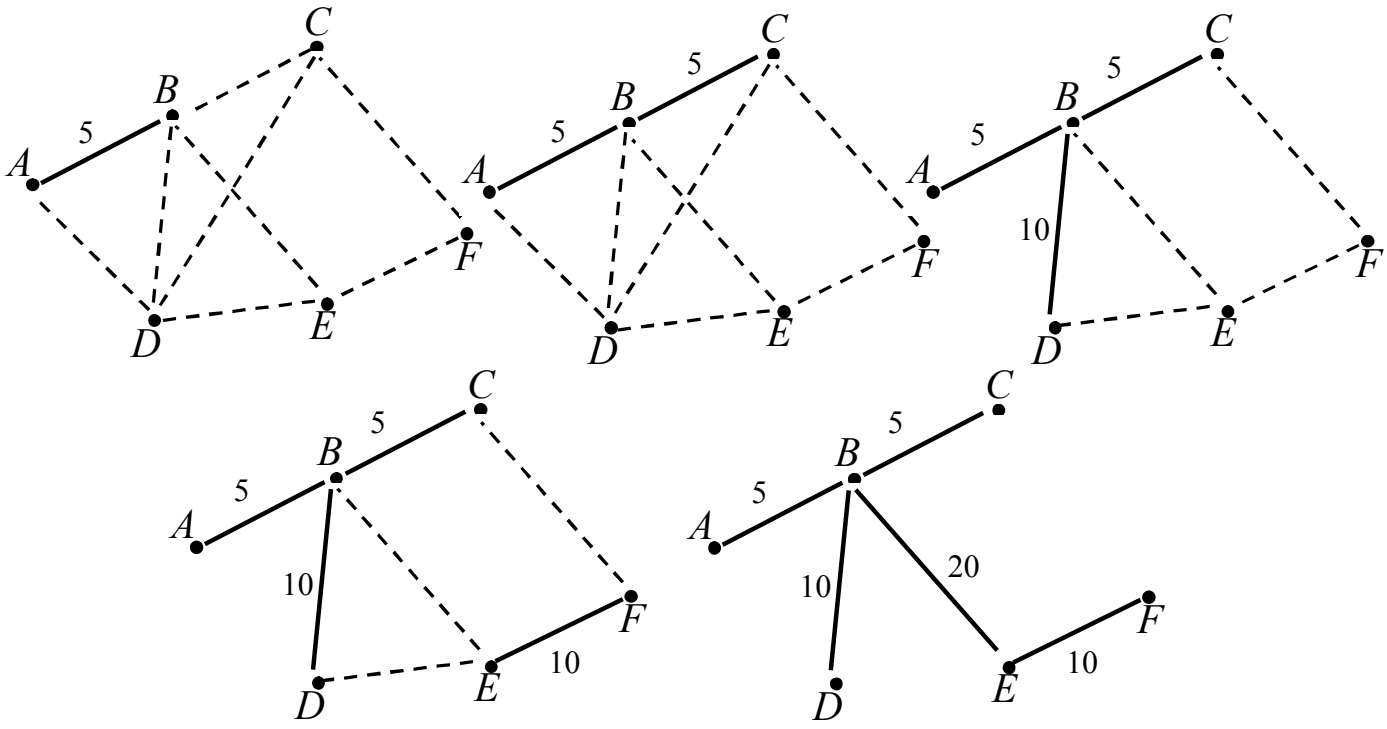
Gunakan algoritma di atas untuk mencari MST pada graph berikut.



Gambar 19. Graph untuk Contoh 5

Penyelesaian:

Langkah pertama pada algoritma adalah memilih sebuah edge yang memiliki robot paling kecil. Pada kasus ini, terdapat dua edge yang memiliki bobot = 5, yaitu $\{A,B\}$ dan $\{B,C\}$. Kita bebas memilih satu diantaranya. Proses membentuk tree diperlihatkan pada Gambar 20. pada setiap gambar, edge yang masuk ke dalam T digambarkan sebagai garis tebal, dan edge yang berada di dalam E' digambar sebagai garis putus-putus. Setelah menambahkan edge kelima, $E' = Æ$, yang menyebabkan berhentinya perulangan.

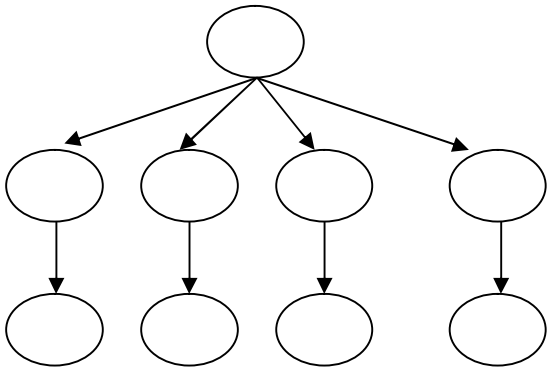


Gambar 20. MST dari graph Contoh 5

3. Implementasi Dalam Pembentukan Folder

Proses pembuatan *folde*r pada aplikasi desktop dari suatu sistem operasi sangatlah mudah. Misalnya pada sistem operasi Windows, Pengguna tinggal mengklik kanan kemudian pilih new folder. Setelah memilih new folder maka akan terbentuk file baru dengan nama default 'New Folder'. Folder tersebut dapat kita beri nama sesuka hati. Kita juga dapat membuat folder baru di dalam folder yang telah kita buat.

Seandainya nama folder yang sudah dinamakan tadi kita anggap sebagai Akar sekaligus Orangtua dari suatu pohon. Maka folder-folder lain yang ada di dalam akar kita anggap sebagai Sub-Pohon dari akar tersebut. Cara kerja dari struktur pohon ini adalah Setiap pembuatan folder baru maka pada satu Akar (folder1) yang sama, akan membangkitkan atau membuat subpohon berikutnya (folder2, folder3, folder4,…,folder-n). Selanjutnya apabila subpohon tersebut bukan daun atau tidak memiliki anak lainya maka folder tadi tidak akan membangkitkan kembali anak-anaknya.



Gambar 21. Pohon pembentuk *folder*

Pada Gambar 21 terlihat jelas bahwa setiap folder yang dibuat oleh folder sebelumnya akan membentuk Subpohon. Secara rekursif Subpohon akan terus terbentuk sebanyak folder yang akan dibuat. Kemampuan ini akan membuat pembuatan folder atau subpohon akan menjadi tidak terbatas. Pengguna dapat membuat folder berapa pun banyaknya dan dimana pun tempatnya.

Selain dengan cara di atas, beberapa sistem operasi lain ada yang menerapkan command-line dalam membuat folder-nya (misalnya minix). Secara konsep, aturan dan struktur yang digunakan sama dengan sebelumnya.

H. SIMPULAN

Makalah ini telah menunjukan bagaimana konsep graph diimplementasi dalam ilmu komputer. Setelah melalui pembahasan, dapat diambil kesimpulan sebagai berikut:

1. Teori Graph merupakan salah satu cabang dari bidang Matematika Diskrit yang mempunyai banyak terapan di berbagai bidang.
2. Struktur data graph dan Minimum Spanning Tree merupakan bentuk implementasi dari teori graph yang mencakup definisi, dan hukum-hukum yang menyertainya.
3. Struktur data graph menggunakan representasi internal senarai ketetanggaan dengan alasan efisiensi penggunaan untuk komputasi, karena penggunaan matriks ketetanggan kurang efisisen dan cenderung boros untuk kasus jumlah sisi sedikit sedangkan matriks ketetanggaan yang dibentuk berupa matriks jarang (*sparse*)
4. Struktur pohon sangat sesuai untuk pembuatan folder pada aplikasi desktop pada suatu sistem operasi karena strukturnya yang rekursif.

I. DAFTAR PUSTAKA

Baker, Roger. 2001. *Linear Algebra*. USA: Rinton Press.

Bogart, Kenneth P., dan Stein, Cliff. 2002. *Discrete Math in Computer Science*. Dept. Of Computer Mathematics and Dept. Of Computer Science. Dartmouth College.

Diestel, Reinhard. 2000. *Graph Theory*. New York: Springer-Verlag.

Horn, R., dan Johnson, C. 1985. *Matrix Analysis*. Cambridge University Press.

Kaw, Autar K. 2002. *Introduction to Matrix Algebra*. University of South Florida. http://www.eng.usf.edu/~kaw

Munir, Rinaldi. 2003. *Diktat Kuliah IF2153. Matematika Diskrit Edisi Keempat.* Bandung: Penerbit ITB. http://www.informatika.org/~rinaldi.

Nobel, B., dan Daniel, J. 1977. *Applied Linear Algebra*. USA: Prentice Hall.

Skvarcius, Romualdas., dan Robinson, William B. 1986. *Discrete Mathematics with Computer Science Applications*. California: The Benjamin/Cummings Publishing Company, Inc.