Aplikasi Algoritma Dijkstra dalam Pencarian Lintasan Terpendek Graf

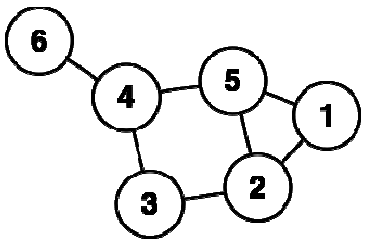
Nur Fajriah Rachmah - 13506091
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha nomor 10,
email : if16091@students.if.itb.ac.id

Abstrak *– Pencarian lintasan terpendek dalam graf berarti meminimalisasi bobot suatu lintasan dalam graf. Banyak algoritma untuk mencari lintasan terpendek, namun yang paling banyak diterapkan ialah algoritma Dijkstra. Algoritma ini menggunakan prinsip greedy yang menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi.*

Kata Kunci : *algoritma dijkstra, graf, greedy, shortest path.*

1. PENDAHULUAN

Graf G didefinisikan sebagai pasangan himpunan (*V,E*) yang dalam hal ini *V* adalah himpunan tak kosong dari simpul-simpul (*vertices* atau *node*), dan *E* adalah himpunan sisi-sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul, atau dapat ditulis singkat dengan notasi *G* = (*V,E*).



Gambar 1 : Graf dengan 6 verteks dan 7 *edge*

Sebuah struktur graf dapat dikembangkan dengan memberi bobot pada setiap *edge*. Graf berbobot dapat digunakan untuk melambangkan banyak konsep berbeda. Jika suatu graf melambangkan jaringan jalan, maka bobotnya dapat berarti panjang jalan maupun batas kecepatan pada batas tertentu. Graf berbobot inilah yang digunakan untuk mencari lintasan terpendek.

Pencarian lintasan terpendek dalam graf berarti meminimalisasi bobot suatu lintasan dalam graf. Aplikasinya banyak ditemukan dalam kehidupan sehari-hari seperti untuk mencari lintasan terpendek antara dua kota dan menentukan jalur komunikasi terpendek antara dua buah terminal komputer.

Ada beberapa macam persoalan lintasan terpendek, antara lain : lintasan terpendek antara dua buah simpul, lintasan terpendek antara semua pasangan simpul, lintasan terpendek dari simpul tertentu ke semua simpul lain, dan lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu.

Terdapat banyak cara untuk menentukan lintasan terpendek suatu graf. Hingga saat ini sudah banyak algoritma pencarian lintasan terpendek yang ditulis orang. Diantaranya algoritma Floyd-Warshall, algoritma Johnson, algoritma Dijkstra, dan algoritma Bellman-Ford. Namun yang paling banyak digunakan ialah algoritma Dijkstra.

Algoritma Dijkstra awalnya diterapkan untuk mencari lintasan terpendek pada graf berarah, namun algoritma ini juga benar untuk digunakan pada graf tak berarah.

2. JENIS DAN SIFAT GRAF

Teori graf memiliki banyak terapa hingga saat ini. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual graf adalah dengan menyatakan objek sebagai noktah, bulatan, atau titik. Sedangkan hubungan antara objek dinyatakan dengan garis.

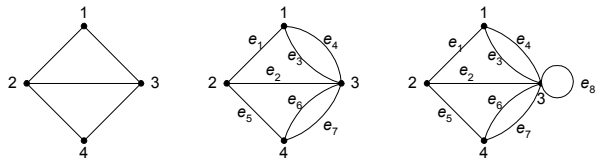
2.1 Jenis Graf

Pengelompokkan graf dapat didasarkan pada ada tidaknya sisi ganda atau sisi kalang, pada jumlah simpul, atau berdasarkan orientasi arah pada sisi.

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, secara umum graf dapat digolongkan menjadi dua jenis :

1. Graf sederhana
   Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana. Jaringan komputer merupakan contoh aplikasi graf sederhana.

2. Graf tak sederhana
   Dibagi menjadi 2, yaitu graf ganda dan graf semu. Graf ganda ialah graf yang mengandung sisi ganda. Graf semu ialah graf yang mengandung gelang.



(a) (b) (c)

Gambar 2 : Graf berdasarkan ada tidaknya sisi gelang atau sisi ganda, (a). Graf sederhana, (b). Graf ganda, (c). Graf semu

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

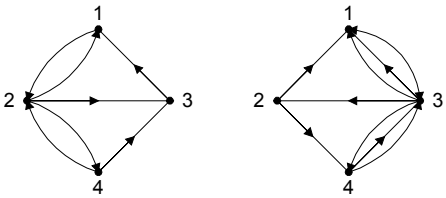
1. Graf berhingga
   Ialah graf yang jumlah simpulnya berhingga.

2. Graf tak berhingga
   Ialah graf yang jumlah simpulnya tak berhingga.

Berdasarkan orientasi arah pada sisi, secara umum graf dibedakan atas dua jenis :

1. Graf berarah
   Graf yang setiap sisinya diberikan orientasi arah disebut graf berarah.

2. Graf tak berarah
   Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak berarah. Graf pada Gambar 2 ialah graf tak berarah.



(a) (b)

Gambar 3 : (a). Graf berarah, (b). Graf berarah ganda

2.2 Terminologi Dasar

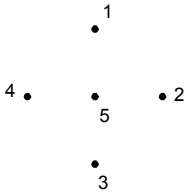
Banyak terminologi yang akan sering digunakan dalam pembahasan graf, diantaranya :

1. Bertetangga *(Adjacent)*
   Dua buah simpul pada graf tak-berarah *G* dikatakan bertetangga jika keduanya terhubung langsung dengan sebuah sisi.

2. Bersisian *(Incident)*
   Untuk sembarang sisi $e = (v_j, v_k)$, sisi e dikatakan bersisian dengan simpul v_j dan v_k.

3. Simpul Terpencil *(Isolated Vertex)*
   Simpul yang tidak mempunyai sisi yang bersisian dengannya disebut simpul terpencil.

4. Graf Kosong *(Null Graph* atau *Empty Graph)*
   Graf yang himpunan sisinya merupakan himpunan kosong disebut graf kosong.



Gambar 4 : Graf kosong

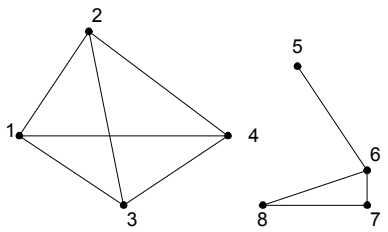
5. Derajat *(Degree)*
   Derajat suatu simpul pada graf tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut. Pada graf berarah, derajat suatu simpul ialah jumlah busur yang masuk ke simpul ditambah dengan jumlah busur yang keluar dari simpul.

6. Lintasan *(Path)*
   Lintasan yang panjangnya *n* dari simpul awal v_o ke simpul tujuan v_n di dalam graf *G* ialah 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 $e_1 = (v_0,v_1), e_2 = (v_1,v_2), \ldots, e_n = (v_{n-1},v_n)$ adalah sisi-sisi dari graf *G*.

   Panjang lintasan adalah jumlah sisi dalam lintasan tersebut.

7. Siklus *(Cycle)* atau Sirkuit *(Circuit)*
   Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus. Panjang sirkuit adalah jumlah sisi di dalam sirkuit tersebut.

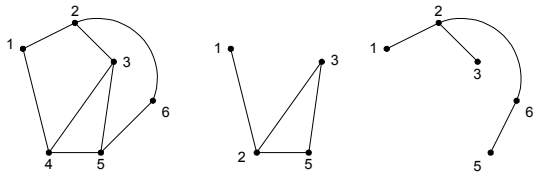
8. Terhubung *(Connected)*
   Graf tak berarah G disebut graf terhubung jika untuk setiap pasang simpul v_i dan v_j dalam himpunan *V* terdapat lintasan dari v_i ke v_j. Jika tidak, maka graf *G* tak terhubung.



Gambar 5 : (a). Graf terhuubung, (b). Graf tak terhubung

9. Upagraf *(Subgraph)* dan Komplemen Upagraf
Misalkan $G = (V,E)$ adalah sebuah graf. $G_1 = (V_1, E_1)$ adalah upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.

Komplemen dari upagraf G_1 terhadap graf G adalah graf $G_2 = (V_2, E_2)$ sedemikian sehingga $E_2 = E - E_1$ dan V_2 adalah himpunan simpul yang anggota-anggota E_2 bersisian dengannya.



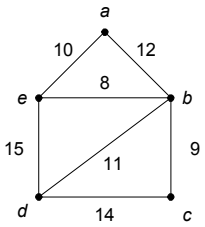
(a) (b) (c)

Gambar 6 : (a). Graf G_1, (b). Upagraf dari G_1, (c). Komplemen dari upagraf yang bersesuaian.

10. Upagraf Merentang *(Spanning Subgraph)*
Upagraf $G_1 = (V_1, E_1)$ dari $G = (V,E)$ dikatakan upagraf merentang jika $V_1 = V$ (yaitu G_1 mengandung semua simpul dari G)

11. *Cut-Set*
Cut-set dari graf terhubung G adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung. Jadi, *cut-set* selalu menghasilkan dua buah komponen terhubung. Nama lain untuk *cut-set* ialah *bridge* (jembatan).

12. Graf Berbobot *(Weighted Graph)*
Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga (bobot). Graf inilah yang digunakan untuk mencari lintasan terpendek.



Gambar 7 : Graf berbobot

3. LINTASAN TERPENDEK

Persoalan mencari lintasan terpendek (*shortest path problem*) dalam graf merupakan persoalan optimasi. Pencarian lintasan terpendek termasuk masalah yang paling umum dalam suatu *weighted-connected graph*. Dalam masalah ini terdapat subkelas-subkelas masalah yang lebih spesifik. Misalnya pada jaringan jalan raya yang menghubungkan kota-kota disuatu wilayah, hendak dicari lintasan terpendek yag menghubungkan antara dua kota berlainan tertentu (*Single-source shortest path problems*), semua lintasan terpendek masing-masing dari suatu kota ke setiap kota lainnya *(Single-source shortest path problems)*, semua lintasan terpendek masing-masing antara tiap kemungkinan pasang kota yang berbeda *(all-pairs shortest path problems)*.

Untuk memecahkan masing-masing dari masalah-masalah tersebut terdapat sejumlah solusi. Yaitu algoritma Dijkstra untuk *single-source shortest path*, algoritma Floyd-Warshall untuk masalah *all-pairs shortest path*, dan algoritma Johnson untuk masalah *all-pairs shortest path* pada *sparse graph*.

Dalam beberapa masalah graf lain, suatu graf dapat memiliki bobot negatif dan kasus ini dipecahkan oleh algoritma Bellman-Ford.

Yang akan dibahas di sini adalah algoritma Dijkstra yaitu mencari lintasan terpendek dari suatu verteks asal tertentu ke setiap verteks lainnya (algoritma ini juga berfungsi sangat optimal pada masalah *single-destination*).

3.1 Algoritma Dijkstra

Algoritma ini diberi nama sesuai nama penemunya, Edsger Wybe Dijkstra. Algoritma Dijkstra mencari lintasan terpendek dalam sejumlah langkah. Algoritma ini menggunakan prinsip greedy yang menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi.

Input algoritma ini adalah sebuah graf berarah yang berbobot (*weighted directed graph*) G dan sebuah sumber *vertex s* dalam G dan V adalah himpunan semua vertices dalam graph G.

Algoritma Dijkstra dimulai dari sebuah simpul asal dan dalam setiap iterasinya menambahkan sebuah verteks lain ke lintasan terpendek pohon merentang. Verteks ini merupakan titik terdekat ke akar namun masih di luar bagian pohon.

Algoritma Dijkstra dalam *pseudo-code* :

```
procedure Dijkstra (input m: matriks,
                    a:simpul awal)
{Mencari lintasan terpendek dari simpul
awal a ke semua simpul lainnya
Masukan: matriks ketetanggaan (m) dari
graf berbobot G dan simpul awal a
Keluaran: lintasan terpendek dari a ke
semua simpul lainnya}

Deklarasi
     s1, s2, ... , sn : integer {larik
integer}
     d1, d2, ... , dn : integer {larik
integer}
     i : integer
```

```
Algoritma
    {Langkah 0 (inisialisasi): }
    for i ← 1 to n do
        s_i ← 0
        d_i ← m_ai
    endfor

    {Langkah 1: }
    s_a ← 1     {karena simpul a adalah
                simpul asal lintasan
                terpendek, jadi simpul a
                sudah pasti terpilih dalam
                lintasan terpendek}

    d_a ← ∞     {tidak ada lintasan
                terpendek dari simpul a ke
                a}

    {Langkah 2,3,..., n-1 :}
    for i ← 2 to n-1 do
        cari j sedemikian sehingga s_j = 0
        dan d_j = min {d_1,d_2,...,d_n}
        s_j ← 1 {simpul j sudah terpilih ke
        dalam lintasan terpendek}
        perbarui d_i, untuk i = 1,2,3,...,n
        dengan: d_i (baru) = min {d_i (lama),
        d_j + m_ji}
    endfor
```

3.2 Analisa Algoritma Dijkstra

Algoritma ini mirip dengan algoritma Prim untuk mencari MST, yaitu pada tiap iterasi memeriksa sisi-sisi yang menghubungkan subset verteks W dan subset verteks (V-W) dan memindahkan verteks w dari (V-W) ke W yang memenuhi kriteria tertentu. Perbedaannya terletak pada kriteria itu sendiri.

Dalam algoritma Dijkstra, yang dicari adalah sisi yang menghubungkan ke suatu verteks di (V-W) sehingga jarak dari verteks asal Vs ke verteks tersebut adalah minimal.

Dalam implementasinya penghitungan jarak dari verteks asal vs disederhanakan dengan menambahkan *field minpath* pada setiap verteks. *Field-field minpath* ini diinisialisasi sesuai dengan *adjacency*-nya dengan vs, kemudian dalam setiap iterasi di-*update* bersamaan masuknya w dalam W. *Field minpath* ini menunjukkan jarak dari vs ke verteks yang bersangkutan terpendek yang diketahui hingga saat itu. Jadi pada verteks dalam W, *minpath* sudah menunjukkan jarak terpendek dari Vs untuk mencapai verteks yang bersangkutan, sementara pada (V-W) masih perlu di-*update* pada setiap iterasi dalam mendapatkan verteks w seperti diterangkan di atas. Yaitu, setiap mendapatkan w maka *update minpath* setiap *adjacent* verteks x dari w di (V-W) sisa dengan: minimum (*minpath* dari x , total *minpath* w + panjang sisi yang bersangkutan). Agar dapat berlaku umum maka di awal algoritma seluruh *minpath* diinisialisasi dengan $+\infty$.

Demikian halnya pencarian w itu sendiri disederhanakan menjadi pencarian node di (V-W) dengan *minpath* terkecil.

Selengkapnya algoritma Dijkstra adalah sebagai berikut :

1. Inisialisasi
   W berisi mula-mula hanya vs, *field minpath* tiap verteks v dengan Weight[vs, v], jika ada sisi tersebut, atau, $+\infty$ jika tidak ada.

2. Lalu, dalam iterasi lakukan hingga (V-W) tak tersisa (atau dalam versi lain: jika ve ditemukan) dari *field minpath* tiap verteks cari verteks w dalam (V-W) yang memiliki *minpath* terkecil yang bukan tak hingga. Jika ada w maka masukkan w dalam W. *Update minpath* pada tiap verteks t *adjacent* dari w dan berada dalam (V-W) dengan: minimum (*minpath*[t] , *minpath*[w] + *weight* [w, t])

Verteks-verteks dalam W dapat dibedakan dari verteks dalam (V-W) dengan suatu field yang berfungsi sebagai *flag* atau dengan struktur *liked-list*. Informasi lintasan dapat diketahui dengan pencatatan predesesor dari setiap verteks yang dilakukan pada saat *update* harga *minpath* tersebut (fungsi minimum). Jika *minpath*[t] di-*update* dengan (*minpath*[w] + *Weight* [w, t]) maka predesesor dari t adalah w. Pada tahap inisialisasi predesesor setiap verteks diisi oleh vs.

Algoritma Dijkstra membuat label yang menunjukkan simpul-simpul. Label-label ini melambangkan jarak dari simpul asal ke suatu simpul lain. Dalam graf, terdapat dua macam label, sementara dan permanen. Label sementara diberikan untuk simpul-simpul yang belum dicapai. Nilai yang diberikan untuk label sementara ini dapat beragam.

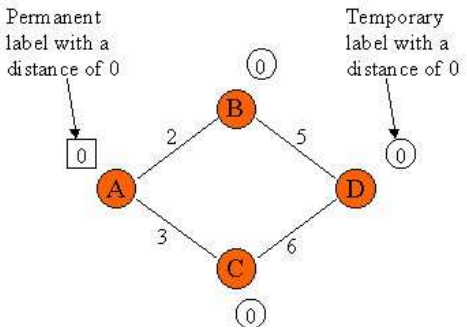
Label permanen diberikan untuk simpul-simpul yang sudah dicapai dan jarak ke simpul asal diketahui. Nilai yang diberikan untuk label ini ialah jarak dari simpul ke simpul asal.

Suatu simpul pasti memiliki label permanen atau label sementara. Tetapi tidak keduanya.

(a) (b)

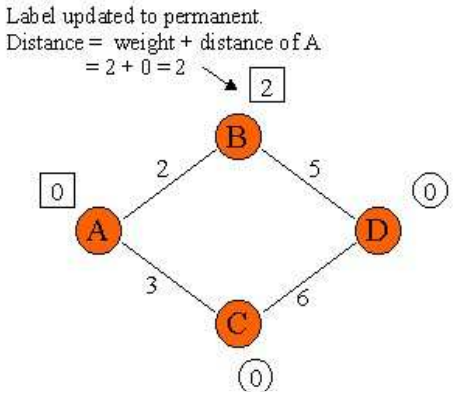
Gambar 8 : (a). Simpul A berlabel sementara dengan jarak 0, (b). Simpul B berlabel permanen dengan jarak 5

Algoritma dimulai dengan menginisialisasi simpul manapun di dalam graf (misalkan simpul A) dengan label permanen bernilai 0 dan simpul-simpul sisanya dengan label sementara bernilai 0.



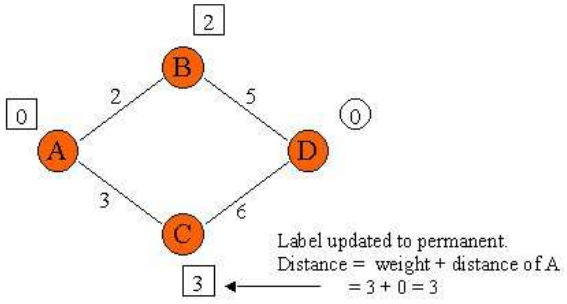
Gambar 9 : Inisialisasi awal

Algoritma ini kemudian memilih nilai sisi terendah yang menghubungkan simpul dengan label permanen (dalam hal ini simpul A) ke sebuah simpul lain yang berlabel sementara (misalkan simpul B). Kemudian label simpul B di-*update* dari label sementara menjadi label permanen. Nilai simpul B merupakan penjumlahan nilai sisi dan nilai simpul A.



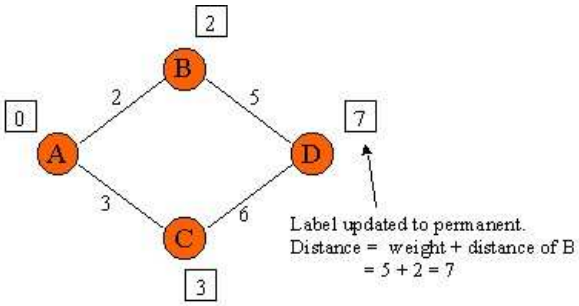
Gambar 10 : Nilai simpul B menjadi permanen

Langkah selanjutnya ialah menemukan nilai sisi terendah dari simpul dengan label sementara, baik simpul A maupun simpul B (misalkan simpul C). Ubah label simpul C menjadi permanen, dan ukur jarak ke simpul A.



Gambar 11 : Nilai simpul C berubah

Proses ini berulang hingga semua label simpul menjadi permanen.



Gambar 12 : Semua nilai simpul menjadi permanen

Lamanya waktu untuk menjalankan Algoritma Dijkstra's pada suatu graf dengan E (himpunan sisi) dan V (himpunan simpul) dapat dinyatakan sebagai fungsi E dan V menggunakan notasi Big-O. Waktu yang dibutuhkan algoritma Dijkstra untuk bekerja ialah sebesar O(V*log V + E).

Implementasi paling sederhana dari algoritma Dijkstra ialah penyimpanan simpul dari suatu himpunan ke dalam suatu array atau list berkait.

4. KESIMPULAN

Persoalan mencari lintasan terpendek merupakan masalah optimasi. Pencarian lintasan terpendek menggunakan graf berbobot.

Dalam mencari lintasan terpendek, algoritma yang paling banyak digunakan orang ialah algoritma Dijkstra, karena paling kerjanya paling efisien (mangkus), tidak membutuhkan waktu yang banyak.

Salah satu implementasi dari algoritma Dijkstra ialah penyimpanan simpul dari suatu himpunan ke dalam suatu array atau list berkait.

Waktu yang dibutuhkan algoritma Dijkstra untuk bekerja ialah O(V*log V + E).

DAFTAR REFERENSI

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF2153 Matematika Diskrit Edisi Keempat. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Wikipedia. (2007). http://id.wikipedia.org/wiki/Algoritma_Dijkstra Tanggal Akses: 31 Desember 2007 pukul 09.26

[3] CS usask. (1999). http://www.cs.usask.ca/resources/tutorials/csconcepts/1999_8/tutorial/advanced/dijkstra/dijkstra.html Tanggal Akses: 2 Januari 2008 pukul 20.33

[4] Ranau CS UI (1998) http://ranau.cs.ui.ac.id/sda/archive/1998/handout/

handout20.htm
Tanggal Akses: 1 Januari 2008 pukul 06.33
[5] tudelft
http://www.nas.its.tudelft.nl/people/Piet/CUPboo
kChapters/PACUP_sp_partial.pdf
Tanggal Akses: 1 Januari 2008 pukul 06.37
[6] Wikipedia. (2007).
http://id.wikipedia.org/wiki/Teori_graf
Tanggal Akses: 1 Januari 2008 pukul 07.00