STUDI DAN IMPLEMENTASI PERSOALAN LINTASAN TERPENDEK SUATU GRAF DENGAN ALGORITMA DIJKSTRA DAN ALGORITMA BELLMAN-FORD

Bayu Aditya Pradhana – NIM : 13505124

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15124@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang studi dan implementasi persoalan lintasan terpendek suatu graf dengan algoritma *dijkstra* dan algoritma B*ellman-Ford*. Lintasan terpendek merupakan bagian dari teori graf. Jika diberikan sebuah graf berbobot, masalah jarak terpendek adalah bagaimana kita mencari sebuah jalur pada graf yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut. Persoalan ini adalah persoalan optimasi, dimana kita akan mencari beberapa alternatif solusi penyelesaian yang paling efektif dan mangkus dari masalah penentuan lintasan terpendek pada suatu graf.

Saat ini banyak sekali algortima-algoritma yang dapat digunakan untuk menyelesaikan persoalan penentuan lintasan terpendek (shortest path problem) dari suatu graf. Solusi yang didapat dari penelusuran algoritma tersebut dapat diberi nama Pathing Algorithm. Ada dua algortima yang cukup terkenal yang bisa digunakaan untuk menyelesaikan persoalan lintasan terpendek, yaitu *Algoritma Dijkstra* dan Algoritma *Bellman-Ford.*

Algoritma *Dijkstra* merupakan salah satu algoritma yang digunakan untuk memecahkan permasalahan lintasan terpendek yang terdapat pada suatu graf . Algoritma ini digunakan pada graf berbobot dengan syarat bobot dari masing-masing sisi haruslah bernilai positif (>=0). Algoritma B*ellman-Ford* juga merupakan salah satu algoritma yang digunakan untuk memecahkan permasalahan lintasan terpendek yang terdapat pada suatu graf. Algoritma ini digunakan pada graf berbobot dengan bobot yang dapat bernilai positif maupun bernilai negatif. Apabila kita hendak mencari lintasan terpendek dari suatu graf berbobot yang terdapat sisi yang berbobot negatif kita dapat menggunakan algoritma *Bellman-Ford*, sedangkan apabila kita menemui suatu graf berbobot yang setiap sisinya berbobot positif maka kita dapat menggunakan algoritma Bellman-Ford atau algoritma dijkstra, beberapa analisa pun menunjukkan beberapa keuntungan dan kelemahan dari kedua algoritma tersebut.

***Kata kunci**: graf, graf berbobot*, lintasan, *shortest pah problem*, *algoritma dijkstra*, *algoritma bellman-ford*, *pathing algorithm*.

1. Pendahuluan

Graf adalah himpunan simpul yang dihubungkan dengan busur-busur. Setiap busur diasosiasikan dengan tepat dua simpul. Dalam kehidupan sehari-hari banyak sekali persoalan yang diimplementasikan dengan graf. Bidang-bidang yang menggunakan penerapan graf antara lain Switching network, Coding theory, Electrical analysis, Operation research, Aljabar, Computer science, dan Kimia.

Graf merupakan model matematika yang sangat kompleks dan rumit, tapi bisa juga menjadi solusi yang sangat bagus terhadap beberapa kasus tertentu. Banyak sekali aplikasi yang menggunakan dengan graf sebagai alat untuk merepresentasikan atau memodelkan persoalan sehinggan persoalan itu dapat diselesaikan dengan baik. Aplikasi-aplikasi tersebut misalnya menentukan lintasan terpendek (the Shortest Path Problem), persoalan pedagang keliling (travelling salesperson problem), persoalan tukang pos Cina (chinese postman problem), pewarnaan graf (graph colouring), Pembuatan system jalan raya satu arah (Making a Road System One-way), menentukan peringkat peserta sebuah turnamen (Rangking the Participants in a

tournament), dan masih banyak lagi. Di dalam makalah ini penulis mencoba mengulas tentang salah satu aplikasi graph yaitu tentang persoalan menentukan lintasan terpendek.

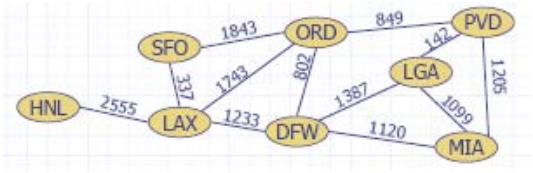
Menurut teori Graf, persoalan lintasan terpendek adalah merupakan suatu persoalan untuk mencari lintasan antara dua buah simpul pada graf berbobot yang memiliki gabungan nilai jumlah bobot pada sisi graf yang dilalui dengan jumlah yang paling minimum. Persoalan lintasan terpendek ini pun banyak sekali dijumpai di kehidupan sehari-hari. Aplikasi yang paling sering ditemui adalah pada bidang transportasi dan komunikasi, seperti pada pencarian rute terbaik untuk menempuh dua kota atau untuk mengetahui dan menelusuri proses pengiriman paket data komunikasi dalam suatu jaringan komunikasi agar dihasilkan suatu proses yang paling cepat.

Solusi untuk persoalan lintasan terpendek ini sering disebut juga pathing algorithm. Banyak sekali algortima yang dapat digunakan untuk menyelesaikan persoalan ini seperti algoritma djikstra (djikstra' algorithm) dan algoritma bellman-Ford (bellman-Ford algorithm). Dalam menyelesaikan persoalan lintasan terpendek kedua algoritma tersebut memiliki beberapa kelebihan dan kekurangan yang akan dibahas lebih lanjut di dalam makalah ini.

Makalah ini bertujuan untuk membandingkan keunggulan dan kekurangan algoritma dijkstra (dijkstra' algorithm) dan algoritma bellman-ford (bellman-ford algorithm) dalam menyelesaikan persoalan menentukan lintasan terpendek. Sehingga dengan makalah ini diharapkan kita bisa mengambil kesimpulan algoritma mana yang lebih efisien untuk digunakan, algoritma dijkstra atau algoritma bellman-ford.

2. Dasar Teori

Teori graf merupakan pokok bahasan yang sudah tua usianya namun memiliki banyak terapan dalam kehidupan sehari-hari sampai saat ini. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Banyak persoalan pada dunia nyata yang sebenarnya merupakan representasi visual dari graf. Contoh salah satu representasi visual dari graf adalah peta. Banyak hal yang dapat digali dari representasi tersebut, diantaranya adalah menentukan jalur terpendek dari satu tempat ke tempat lain, menggambarkan 2 kota yang bertetangga dengan warna yang berbeda pada peta, menentukan tata letak jalur transportasi, pengaturan jaringan komunikasi atau jaringan internet dan masih banyak lagi. Selain peta, masih banyak hal lain dalam dunia nyata yang merupakan representasi visual dari graf.



Gambar 1 Graf yang merupakan yang menggambarkan peta beberapa negara bagian di Amerika Serikat

Secara matematis, graf didefinisikan sebagai berikut [1] :

Graf *G* didefinisikan sebagai pasangan himpunan (V,E) yang dalam hal ini :

V = himpunan tidak kosong dari simpul - simpul (*vertices* atau *node*): $\{v_1,v_2,\ldots,v_n\}$

E = himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul: $\{e_1,e_2,\ldots,e_n\}$

atau dapat ditulis singkat notasi $G = (V,E)$

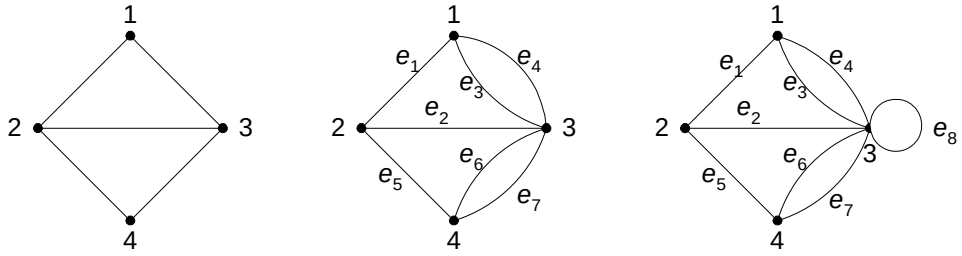
2.1 Jenis-Jenis Graf

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

1. Graf sederhana (*simple graph*).
   Graf yang tidak memiliki gelang maupun sisi-ganda.
2. Graf tak-sederhana (*unsimple-graph*).
   Graf yang memiliki gelang maupun sisi-ganda. Ada dua macam graf tak-sederhana, yaitu graf ganda (*multigraph*) dan graf semu (*pseudograph*). Graf ganda adalah graf yang memiliki sisi ganda, sedangkan graf semu adalah graf yang memiliki gelang.

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

1. Graf berhingga (*limited graph*).
2. Graf tak-berhingga (*unlimited graph*).

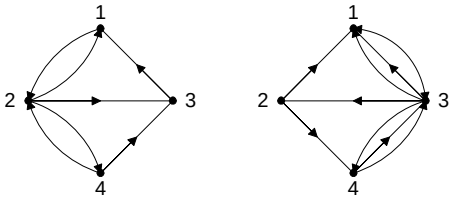


Gambar 2 Beberapa Graf a) graf sederhana b) graf ganda c) graf semu

Berdasarkan orientasi arah pada sisi, maka secara umum graf dapat digolongkan menjadi dua jenis:

1. Graf berarah (*directed graph*).
   Graf yang setiap sisinya diberikan orientasi arah. Pada graf berarah, (v_j, v_k) dan (v_k , v_j) menyatakan dua busur yang berbeda, dengan kata lain
   (v_j , v_k) ≠ (v_k , v_j)

   Untuk busur (v_j , v_k), simpul v_j disebut simpul asal (*initial vertex*) dan untuk simpul *vk* disebut simpul terminal (*terminal vertex*).
2. Graf tak-berarah (un*directed graph*).
   Graf yang sisinya tidak memiliki orientasi arah. Pada graf tek-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan, dengan kata lain:
   (v_j , v_k) = (v_k , v_j)



Gambar 3 Graf berarah (directed graph)

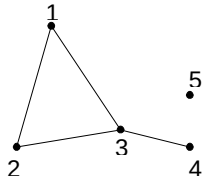
2.2 Terminologi Dasar

Terdapat beberapa istilah penting yang berkaitan dengan graf. Berikut ini didefinisikan beberapa terminologi yang sering digunakan:

1. Bertetangga (*Adjacent*)
   Dua bua simpul pada graf tak-berarah *G* dikatakan bertetangga bila keduanya terhubung langsung dengan sebuah sisi. Dengan kata lain, v_j bertetangga dengan v_k jika (v_j , v_k) adalah sebuah sisi pada graf *G*.

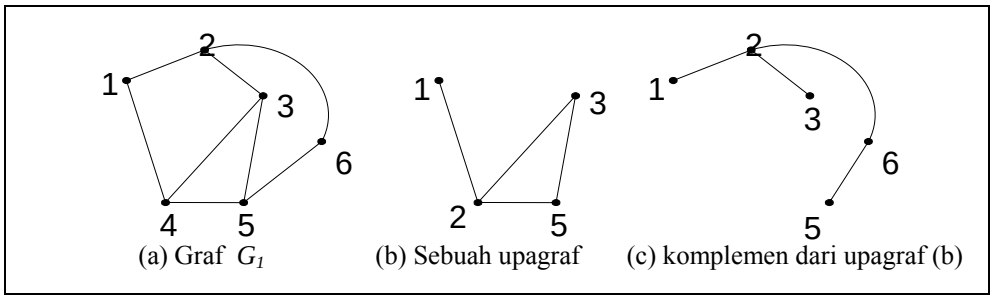
2. Bersisian (*Incident*)
   Untuk sembarang sisi e = (v_j , v_k), sisi e dikatakan bersisian dengan simpul v_j dan simpul v_k .

3. Simpul Terpencil (*Isolated Vertex*)
   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.



Gambar 4 Simpul 5 simpul terpencil

4. Graf Kosong (*Null* atau *Empty Graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong, ditulis sebagai N_n, yang dalam hal ini *n* adalah jumlah simpul.



(a) Graf G_1 (b) Sebuah upagraf (c) komplemen dari upagraf (b)

Gambar 5

5. Derajat (*Degree*)
   Derajat suatu simpul pada graf tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut.
   Pada graf berarah, derajat simpul v dinyatakan dengan $d_{in}(v)$ dan $d_{out}(v)$, yang dalam hal ini:
   $d_{in}(v)$ = derajat masuk (in-degree)
   = jumlah simpul yang masuk ke simpul v

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

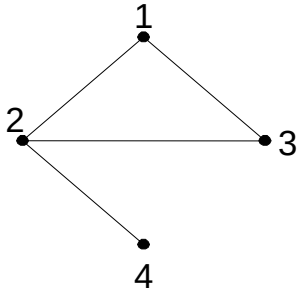
   dan

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

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

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

7. Siklus (*Cycle*) atau Sirkuit (*Circuit*)
   Lintasan yang berawal dan berakhir pada simpul yang sama



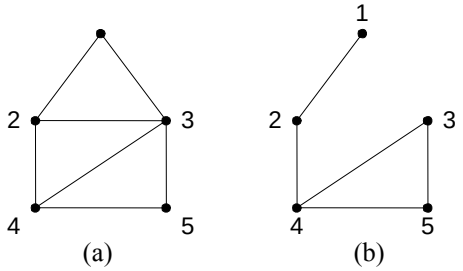
Gambar 6 Sirkuit 1-2-3-1

8. Terhubung (*Connected*)
   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_i ke v_j). Jika tidak, maka G disebut graf tak-terhubung.
   (disconnected graph).
   Graf berarah G dikatakan terhubung jika graf tak-berarahnya terhubung (graf tak-berarah dari G diperoleh dengan menghilangkan arahnya).

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

10. Komplemen Upagraf
    Komplemen dari upagaraf 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.

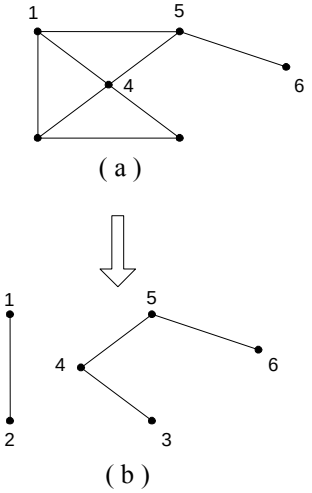
11. Upagraf Merentang
    (*Spanning Subgraph*)
    Upagraf $G_1 = (V_1, E_1)$ dari $G = (V, E)$ dikatakan upagraf merentang jika $V_1 = V$ (yaitu G_1 mengandung semua simpul dari G).



(a) (b)

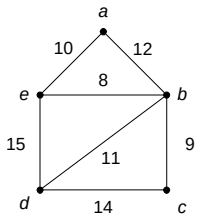
Gambar 7 a) Graf *G*
b) Upagraf merentang dari *G*

12. *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.



Gambar 8 {(1,2),(1,5),(3,5),(3,4)} adalah cut-set

13. Graf Berbobot (*Weighted Graph*)
    Graf yang setiap sisinya diberi harga (bobot).

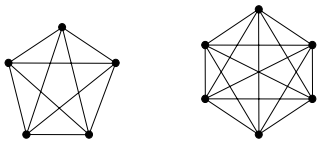


Gambar 9 contoh graf berbobot

3.2 Beberapa Graf Khusus

Terdapat beberapa jenis graf sederhana khusus. Berikut ini didefinisikan beberapa graf khusus yang sering ditemui:

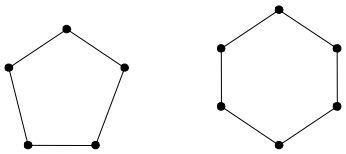
1. Graf Lengkap (*Complete Graph*)
   Graf sederhana yang setiap simpulnya mempunyai sisi ke semua simpul lainnya. Graf lengkap dengan n buah simpil dilambangkan dengan K_n. Setiap simpul pada K_n berderajat n-1. Jumlah sisi pada graf lengkap yang terdiri dari n buah simpul adalah $n(n - 1)/2$.



K_5 K_6

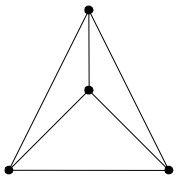
Gambar 10 contoh graf lengkap

2. Graf Lingkaran
   Graf sederhana yang setiap simpulnya berderajat dua. Graf lingkaran dengan n simpul dilambangkan dengan C_n.



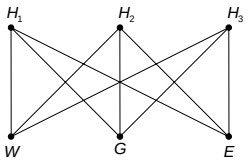
Gambar 11 contoh graf lingkaran

3. Graf Teratur (*Regular Graphs*)
   Graf yang setiap simpulnya mempunyai derajat yang sama. Apabila derajat setiap simpunya adalah r, maka graf tersebut disebut juga graf teratur derajat r.



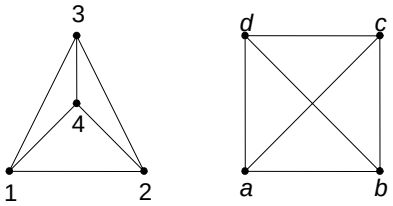
Gambar 12 contoh graf teratur

4. Graf Bipartit (*Bipartite Graph*)
   Graf *G* yang himpunan simpulnya dapat dipisah menjadi dua himpunan bagian V_1 dan V_2, sedemikian sehingga setiap sisi pada *G* menghubungkan sebuah simpul di V_1 ke sebuah simpul di V_2. Dapat juga dinyatakan sebagai *G(V1,V2)*.



Gambar 13 contoh graf bipartit

5. Graf Isomorfik (*Isomorphic Graph*)
   Dua bua graf, G_1 dan G_2 dikatakan isomorfik jika terdapat korespondensi satu-satu antara simpul-simpul keduanya dan antara sisi-sisi keduanya sedemikian sehingga jika sisi *e* bersisian dengan simpul *u* dan *v* di G_1 , maka sisi *e'* yang berkorespon di G_2 juga harus bersisian dengan simpul *u'* dan *v'* di G_2.

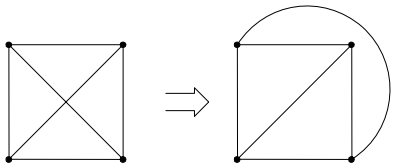


Gambar 14 dua buah graf yang saling isomorfik

Syarat-syarat dua buah graf dapat dikatakan graf isomorfik :

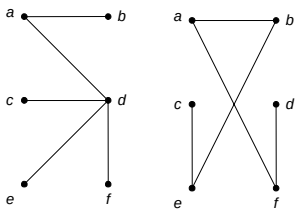
a. Mempunyai jumlah simpul yang sama.
b. Mempunyai jumlah sisi yang sama.
c. Mempunyai jumlah simpul yang sama berderajat tertentu.

6. Graf Planar (*Planar Graph*)
   Graf *G* yang dapat digambarkan pada bidang datar dengan sisi-sisi yang tidak saling memotong (bersilangan).
   Graf planar yang digambarkan dengan sisi-sisi yang tidak saling berpotongan dinamakan graf bidang (plane graph).



Gambar 15 dua graf yang saling planar

7. Pohon (*Tree*)
   Graf tak-berarah terhubung yang tidak mengandung sirkuit.



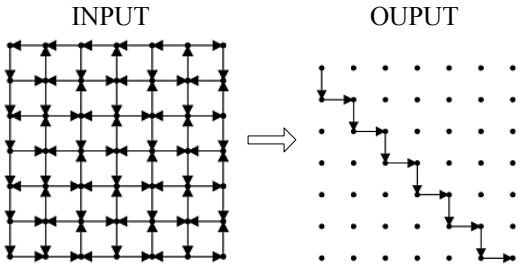
Gambar 16 dua buah pohon

3. The Shortest Path Problem

Menurut teori Graf, persoalan lintasan terpendek (The Shortest Path Problem) adalah merupakan suatu persoalan untuk mencari lintasan antara dua buah simpul pada graf berbobot yang memiliki gabungan nilai jumlah bobot pada sisi graf yang dilalui dengan jumlah yang paling minimum atau dapat dinyatakan juga sebagai berikut :

> diberikan sebuah graf berbobot (dengan himpunan simpul *V*, himpunan sisi *E*, dan fungsi bobot bernilai bilangan riil yang dapat ditulis dengan f : E → **R**), dan diberikan elamen *v* dari *V* , sehingga dapat dicari sebuah lintasan P dari *v* ke setiap *v*' dari *V*, sehingga
>
> $$\sum_{p \in P} f(p)$$
>
> adalah nilai minimal dari semua lintasan yang menghubungkan *v* ke *v*'.

Persoalan lintasan terpendek merupakan salah satu persoalan optimasi yang menggunakan graf berbobot, dimana bobot pada setiap sisi graf tersebut dapat kita gunakan untuk menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya.



Gambar 17 Shortest Path Problem

3.1 Aplikasi dari Shortest Path Problem

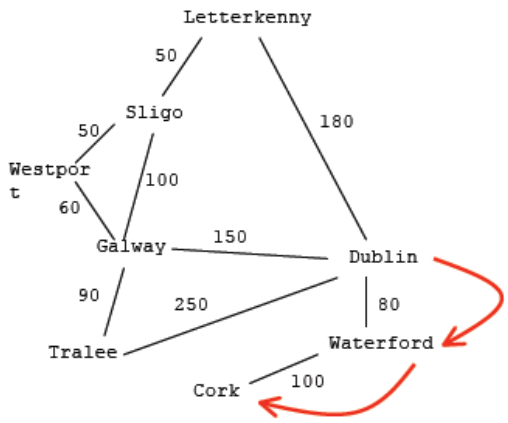
Ada beberapa macam persoalan lintasan terpendek, antara lain [1] :

a. Lintasan terpendek anatara dua buah simpul tertentu.
b. Lintasan terpendek antara semua pasangan simpul.
c. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain.

d. Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu.

Aplikasi persoalan penentuan lintasan terpendek ini banyak sekali kita jumpai dalam kehidupan sehari-hari:

a. Menentukan rute atau jalur terbaik yang harus ditempuh dari suatu kota untuk menuju ke kota lain.
b. Menentukan jalur komunikasi dua buah terminal komputer.
c. Menentukan jalur penerbangan dunia yang paling efektif untuk dilakukan.



Gambar 18 aplikasi rute beberapa kota

3.2 Pathing Algorhitm

Pathing Algorithm adalah sebuah algoritma yang digunakan untuk mencari suatu solusi dalam menentukan lintasan terpendek dari suatu graf. Saat ini terdapat banyak sekali algoritma yang digunakan untuk menyelesaikan persoalan lintasan terpendek diantaranya Algoritma *Djikstra* (djikstra algorithm) dan Algoritma *Bellman-Ford* (bellman-ford algorithm).

3.2.1 Algoritma *Dijkstra*

Edsger Wybe Dijksrta, menemukan suatu algoritma untuk mencari lintasan terpendek pada suatu graf. Algoritma Dijksstra pada awalnya diterapkan pada graf berarah, tetapi ternyata algoritma ini juga benar untuk algoritma graf tak-berarah. Algoritma ini menggunakan prinsip *greedy* yang digunakan untuk menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi. Akan tetapi bobot dari graf tersebut harus lah bernilai bilangan positif (bobot >=0).

Properti algoritma Dijkstra:

1. Matriks ketetanggaan M[m_{ij}]

   m_{ij} = bobot sisi (i, j)
   (pada graf tak-berarah $m_{ij} = m_{ji}$)
   $m_{ii} = 0$
   $m_{ij} = \infty$, jika tidak ada sisi dari simpul *i* ke simpul *j*

2. Larik $S = [s_i]$ yang dalam hal ini,

   s_i = 1, jika simpul i termasuk ke dalam lintasan terpendek
   s_i = 0, jika simpul i tidak termasuk ke dalam lintasan terpendek

3. Larik/tabel $D = [d_i]$ yang dalam hal ini,

   d_i = panjang lintasan dari simpul awal *s* ke simpul *i*

Algoritma Lintasan Terpendek Dijkstra :

(Mencari lintasan terpendek dari simpul a ke semua simpul lain}

Langkah 0 (inisialisasi):

- inisialisasi $s_i = 0$ dan $d_i = m_{ai}$ untuk i = 1, 2, ..., *n*

Langkah 1:

- isi s_a dengan 1 *(karena simpul a adalah simpul asal lintasan terpendek, jadi sudah pasti terpilih)*
- isi d_a dengan ∞ *(tidak ada lintasan terpendek dari simpul a ke a)*

Langkah 2, 3, ... , *n*-1:

- cari *j* sedemikian sehingga $s_j = 0$ dan $d_j = \min\{d_1, d_2, ..., d_n\}$
- isi s_j dengan 1
- perbarui d_i, untuk $i = 1, 2, 3, \ldots, n$ dengan:
  d_i (baru) = $\min\{d_i$ (lama), $d_j + m_{ji}\}$.

3.2.2 Algoritma *Bellman-Ford*

Algoritma Bellman-Ford menghitung jarak terpendek (dari satu sumber) pada sebuah digraf berbobot. Maksudnya dari satu sumber ialah

bahwa ia menghitung semua jarak terpendek yang berawal dari satu titik node. Algoritma Dijkstra dapat lebih cepat mencari hal yang sama dengan syarat tidak ada sisi (edge) yang berbobot negatif. Maka Algoritma Bellman-Ford hanya digunakan jika ada sisi berbobot negatif.

Algoritma Bellman-Ford menggunakan waktu sebesar O(V.E), di mana V dan E adalah menyatakan banyaknya sisi dan titik. Dalam konteks ini, bobot ekivalen dengan jarak dalam sebuah sisi.

4. Analisa

4.1 Analisa Algoritma *Dijkstra*

Input algoritma ini adalah sebuah weighted directed graph *G* dan sebuah source vertex *s* dalam *G*. *V* adalah himpunan semua vertices dalam graph *G*. Setiap edge dari graph ini adalah pasangan vertices (*u*,*v*) yang melambangkan hubungan dari vertex *u* ke vertex *v*. Himpunan semua edge disebut *E*. Weights dari edges dihitung dengan fungsi *w*: $E \rightarrow [0, \infty)$; jadi *w*(*u*,*v*) adalah jarak non-negatif dari vertex *u* ke vertex *v*. Cost dari sebuah edge dapat dianggap sebagai jarak antara dua vertex, yaitu jumlah jarak semua edge dalam path tersebut. Untuk sepasang vertex *s* dan *t* dalam *V*, algoritma ini menghitung jarak terpendek dari *s* ke *t*.

```
1 function Dijkstra(G, w, s)
2   for each vertex v in V[G]
3       d[v] := infinity
4       previous[v] := undefined
5   d[s] := 0
6   S := empty set
7   Q := V[G]
8   while Q is not an empty set
9     u := Extract_Min(Q)
10    S := S union {u}
11    for each edge(u,v)
                outgoing from u
12     if d[u] + w(u,v) < d[v]
13        d[v] := d[u] + w(u,v)
14        previous[v] := u
```

Gambar 19 Pseudo-code Algoritma Dijkstra

Algoritma ini bekerja dengan cara menyimpan setiap simpul v, dengan d[v] menyatakan lintasan terpendek yang telah ditemukan antara s dan v. Pada awalnya, nilai ini adalah 0 untuk sumber simpul s (d[s]=0), dan ketidakterbatasan untuk semua simpul lain, yang menyatakan fakta bahwa kita tidak mengetahui lintasan manapun yang mengacu ke arah simpul itu (d[v]=∞ untuk tiap-tiap v di dalam V, kecuali s). Ketika algoritma berhenti, d[v] akan menyatakan nilai lintasan yang paling pendek dari s sampai v atau bernilai takberhingga, jika lintasan tersebut tidak ada.

Operasi dasar dari Algoritma Dijkstra adalah relaksasi sisi: jika ada suatu sisi dari u ke v, kemudian diketahui lintasan yang paling pendek dari s ke u (d[u]) dapat diperluas menjadi lintasan dari s ke v dengan menambahkan sisi (u,v) pada bagian akhir. Lintasan ini akan mempunyai panjang d[u]+w(u,v). Jika nilai ini adalah kurang dari d[v] yang sekarang, kita dapat menggantikan nilai d[v] yang sekarang dengan nilai yang baru itu. Relaksasi sisi diterapkan sampai semua nilai-nilai d[v] menggembalikan nilai lintasan yang paling pendek dari s ke v. Algoritma ini diorganisir sedemikian sehingga masing-masing sisi (u,v) direlaksasi hanya sekali, ketikad[u] telah mencapai nilai akhir nya.

Ide relaksasi ini datang dari suatu analogi antara perkiraan lintasan yang paling pendek dan panjang suatu pegas seperti bentuk sekerup yang bukan dirancang untuk tekanan. Pada awalnya, nilai lintasan yang paling pendek adalah suatu optimasi, yang dianalogikan dengan pegas. Ketika lintasan lebih pendek ditemukan, nilai yang telah diperkirakan diturunkan, dan pegas diperlonggar. Ketika lintasan yang paling pendek ada dan telah ditemukan maka pegas diperlonggar ke nya panjang normalnya.

Algoritma memelihara dua himpunan simpul S dan Q. Himpunan S berisi semua simpul yang telah kita ketahui bahwa nilai d[v] adalah sudah merupakan nilai lintasan terpendek dan himpunan Q berisi semua simpul lainnya. Pada awalnya himpunan S kosong, dan pada setiap langkah satu simpul dipindahkan dari Q ke S. Simpul ini dipilih sebagai simpul dengan nilai d[u] paling rendah. Ketika suatu simpul u dipindahkan ke S, algoritma ini merelaksasi setiap sisi (u,v).

Kita dapat menyatakan lamanya waktu untuk menjalankan Algoritma Dijkstra's pada suatu graf dengan E (himpunan sisi) dan V (himpunan simpul) sebagai fungsi |E| dan |V| menggunakan notasi Big-O.

Implementasi yang paling sederhana dari Algoritma Dijkstra menyimpan simpul dari himpunan Q ke dalam suatu array atau list

berkait, dan operasi Extract-Min(Q) adalah suatu pencarian linier di seluruh simpul di Q. Dalam hal ini, waktu untuk menjalankan adalah O(V2).

Untuk sparse graph, yaitu graf dengan sisi lebih sedikit dibanding sisi V2, Algoritma Dijkstra dapat diterapkan secara lebih mangkus dengan mengubah graf itu dalam wujud senarai ketetanggaan dan penggunaan suatu tumpukan biner atau tumpukan Fibonacci sebagai antrian prioritas untuk menerapkan fungsi Extract-Min. Dengan suatu tumpukan biner, algoritma akan memerlukan O((E+V)Logv) Waktu, dan Fibonacci Tumpukan meningkatkannya menjadi O(E + Vlogv).

4.2 Analisa Algoritma *Bellman-Ford*

```
// Definisi tipe data graf
record titik {
    list sisi2
    real jarak
    titik sebelum
}
record sisi {
    titik dari
    titik ke
    real bobot
}

function BellmanFord
          (list semuatitik, list
          semuasisi, titik dari)

   // Persiapan
   for each titik v in semuatitik:
       if v is dari then v.jarak = 0
       else v.jarak := tak-hingga
       v.sebelum := null

   // Perulangan relaksasi sisi
   for i from 1 to size(semuatitik):
       for each sisi uv in
           semuasisi:
              u := uv.dari
              v := uv.ke
              if v.jarak > u.jarak +
                  uv.bobot
                v.jarak := u.jarak +
                           uv.bobot
                v.sebelum := u

   // Cari sirkuit berbobot negatif
   for each sisi uv in semuasisi:
       u := uv.dari
       v := uv.ke
       if v.jarak > u.jarak +
                    uv.bobot
          error "Graph mengandung
           siklus berbobot total
           negatif"
```

Gambar 20 Pseudo-code Algoritma Bellman-Ford

Untuk kasus yang induktif, kita pertama membuktikan bagian yang pertama. Pertimbangkan waktu ketika suatu jarak simpul diperbaharui oleh v.distance:= u.distance + uv.weight. Dengan asumsi induktif, u.distance adalah panjang lintasan dari sumber sampai u. Kemudian u.distance + uv.weight adalah panjang lintasan dari sumber sampai v yang mengikuti lintasan dari sumber hingga u dan kemudian dilanjutkan ke v.

Untuk bagian yang kedua , pertimbangkan lintasan yang paling pendek dari sumber samapi u dengan sisi i. Biarkan V menjadi simpul yang terJakhir sebelum u pada lintasan ini. Kemudian, bagian dari lintasan dari sumber sampai v adalah lintasan yang paling pendek dari sumber sampai v dengan sisi i-1. Dengan asumsi induktif, v.distance setelah i-1 siklus adalah pada panjang terjauh lintasan ini. Oleh karena itu, uv.weight+ v.distance adalah panjang lintasan paling jauh dari s sampai u. Di dalam ith siklus, u.distance akan dibandingkan dengan uv.weight+ v.distance, dan akan dibuat sama jika uv.weight+ v.distance bernilai lebih kecil. Oleh karena itu, setelah i berputar, u.distance adalah lintasan yang paling pendek dari sumber sampai u yang menggunakan sisi i.

Ketika i setara dengan banyaknya simpul di dalam graf, masing-masing lintasan akan menjadi lintasan yang paling pendek secara keseluruhan, kecuali jika ada siklus berbobot negatif. Jika suatu siklus berbobot negatif ada dan dapat diakses dari sumber, kemudian tersedia jalan manapun yang lebih pendek, maka tidak ada jalan yang paling pendek. Cara lainnya adalah rute terpendek tidak akan meliputi siklus manapun (sebab mengelilingi suatu siklus akan membuat perjalanan lebih pendek), maka masing-masing lintasan paling pendek mengunjungi simpul masing-masing paling banyak sekali, dan jumlah dari sisi nya kurang dari banyaknya simpul di dalam grafik itu.

5. Kesimpulan dan Saran Pengembangan

5.1 Kesimpulan

Persoalan menentukan lintasan terpendek adalah salah satu bentuk aplikasi dari penggunaan teori graf. Persoalan ini adalah merupakan suatu persoalan optimasi, sehingga semakin mangkus algoritma penyelesaiannya maka akan semakin baik.

Algoritma *Dijkstra* merupakan salah satu algoritma yang digunakan untuk memecahkan permasalahan lintasan terpendek yang terdapat pada suatu graf . Algoritma ini digunakan pada graf berbobot dengan syarat bobot dari masing-masing sisi haruslah bernilai positif (>=0).

Algoritma B*ellman-Ford* merupakan salah satu algoritma yang digunakan untuk memecahkan permasalahan lintasan terpendek yang terdapat pada suatu graf. Algoritma ini digunakan pada graf berbobot dengan bobot yang dapat bernilai positif maupun bernilai negatif.

Meskipun pada algoritma *Bellman-Ford* dapat mengangani bobot bernilai negatif namun algoritma D*ijkstra* lebih sering digunakan karena membutuhkan waktu yang lebih sedikit daripada algoritma *Bellman-Ford*, jadi persoalan dapat diselesaikan dengan lebih mangkus.

5.2 Saran Pengembangan

Apabila kita hendak mencari lintasan terpendek dari suatu graf berbobot yang terdapat sisi yang berbobot negatif kita dapat menggunakan algoritma *Bellman-Ford*, sedangkan apabila kita menemui suatu graf berbobot yang setiap sisinya berbobot positif maka kita dapat menggunakan algoritma Bellman-Ford atau algoritma dijkstra, namun disarankan untuk menggunakan algoritma dijksta karena algoritma tersebut bisa dikatakan lebih mangkus.

DAFTAR PUSTAKA

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

[2] Murty, U.S.R. & J.A. Bondy. (1982) . Graph theory with applications. North-Holland. New York

[3] Dobson, Simon. (2006). Weighted graphs and shortest paths. UCD School of Computer Science and Informatics.Dublin

[4] Wikipedia. (2007), Shortest Path Problem, *http://en.wikipedia.org/wiki/Shortest_path_problem.htm,* Tanggal Akses: 26 Desember 2006 pukul 10.32

[5] NUS.(2006), *http://www.comp.nus.edu.sg/%7Estevenha/programming/prog_graph2.html,* Tanggal Akses: 26 Desember 2006 10:43.

[6] Shortest Path.(1997), *http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK4/NODE161.HTM,* Tanggal Akses: 26 Desember 2006 10:36