Penerapan Algoritma Dijkstra dalam Pemilihan Trayek Bus Transjakarta

Muhammad Yafi 13512014
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512014@std.stei.itb.ac.id

Abstrak – Makalah ini membahas tentang penggunaan salah satu materi Strategi Algoritma yaitu Algoritma Dijkstra, yang menggunakan prinsip greedy, untuk merancang lintasan yang dipilih oleh pengguna bus Transjakarta untuk pergi dari suatu tempat asal ke tempat tujuan. Lintasan tersebut dipilih sedemikian hingga waktu yang ditempuh oleh pengguna seminimal mungkin.

Index Terms – graf, Dijkstra, lintasan, simpul, sisi, waktu, SSSP.

I. PENDAHULUAN

Transjakarta merupakan sarana transportasi umum berbentuk bus yang ada di Jakarta sebagai upaya pemerintah dalam mengatasi kemacetan. Berbeda dengan bus kota biasa, Transjakarta memiliki jalan khusus yang terpisah dengan jalan umum sehingga tidak terbawa arus macet. Karena jalur khusus tersebut, maka mode transportasi umum ini sering juga disebut dengan busway. Keunggulan lain dari bus-bus Transjakarta adalah halte yang terletak di tempat-tempat strategis dan sering dikunjungi oleh warga Jakarta.

Untuk membeli tiket, pengguna tinggal datang ke halte untuk membelinya. Kelebihan dari tiket Transjakarta adalah harganya yang tetap walaupun jaraknya jauh, dan tidak ada biaya tambahan jika harus pindah trayek bus. Trayek tersebut sering juga disebut oleh pengguna Transjakarta dengan koridor, jalur, atau lintasan. Hal ini dikarenakan biaya dihitung saat pengguna masuk ke "area tunggu". Masuk dari luar halte ke area tunggu dihitung satu tiket. Namun, jika pengguna keluar dari bus dan menunggu bus lain untuk ganti trayek, hal tersebut tidak hitung tiket baru. Hal ini dikarenakan pengguna masih berada dalam "area tunggu" dan tidak berasal dari luar halte. Akibatnya, biaya yang dikeluarkan oleh pengguna lebih murah jika dibandingkan dengan naik angkutan umum biasa.

Permasalahan yang sering terjadi adalah jika pengguna ingin menentukan lintasan yang digunakan untuk berpindah dari suatu tempat ke tempat lain, bisa jadi tidak ada bus langsung yang tersedia. Pengguna harus ganti trayek bus selama dua kali atau lebih atau dikenal dengan istilah transit. Karena biaya dari Transjakarta dihitung bukan dari seberapa banyak trayek yang diambil, maka pengguna bisa bebas memilih bus-bus mana saja yang akan dia ambil dalam perjalanan tersebut. Sehingga masalah yang ada hanya berapakah waktu yang dibutuhkan untuk menempuh lintasan tersebut. Penyambungan lintasan ini dapat menghabiskan waktu yang sangat lama jika pengguna tidak tahu trayek-trayek mana saja yang harus diambil. Selain itu, ada juga kemungkinan bahwa walaupun dengan satu buah bus pengguna dapat sampai ke tempat tujuan, namun jika pengguna berganti bus ternyata juga bisa sampai ke tempat tujuan dengan lebih cepat.

Gambar 1- Bus Transjakarta sedang berhenti di halte[7]

Algoritma Dijkstra dapat digunakan untuk menentukan lintasan manakah yang diambil oleh penumpang sehingga lintasan tersebut memiliki waktu tempuh yang sesingkat mungkin. Dengan mengetahui lintasan yang diambil tersebut maka pengguna dapat menentukan bus-bus mana yang akan diambil.

II. DASAR TEORI

2.1 Definisi Graf

Graf G didefinisikan sebagai pasangan himpunan (V,E) dengan

V = himpunan tak-kosong dari simpul
$= \{ v_1 , v_2 , v_3, \ldots , v_n \}$

E = himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul
$= \{ e_1 , e_2 , e_3, \ldots , e_n \}$

Atau dapat ditulis singkat notasi G = (V,E).

Jika sebuah *edge e* menghubungkan simpul v_i dan v_j maka *e* dapat ditulis sebagai $e = (v_i , v_j)$.

2.2 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

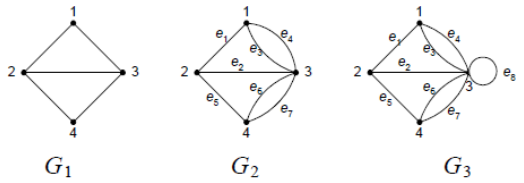
Graf yang tidak mengandung gelang maupun sisi-ganda dinamakan graf sederhana.

2. Graf tak-sederhana

Graf yang mengandung sisi ganda atau gelang dinamakan graf tak-sederhana. Ada dua macam graf tak-sederhana, yaitu graf ganda dan graf semu.

Graf ganda adalah graf yang mengandung sisi ganda. Sebuah graf memiliki sisi ganda jika ada 2 buah simpul yang dihubungkan lebih dari satu sisi.

Graf semu adalah graf yang memiliki sisi gelang (*loop).* Sisi gelang adalah sisi yang menghubungkan sebuah simpul dengan simpul itu sendir. Untuk lebih jelasnya, perhatikan Gambar 2.

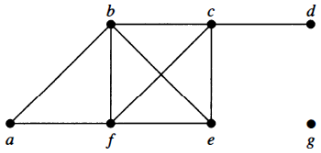


Gambar 2-(a) graf sederhana, (b) graf ganda, dan (c) graf semu.[2]

Sisi graf dapat memiliki orientasi arah. Berdasarkan arah dari sisi, graf dibedakan menjadi 2 jenis :

1. Graf tak-berarah

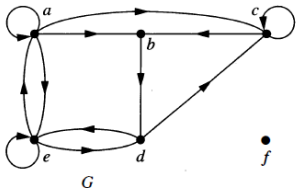
Graf yang sisinya tidak memiliki orientasi arah disebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul pada sisi tidak diperhatikan. Sebuah sisi e = (u, v) sama dengan e = (v, u)



Gambar 3-Graf tak-berarah[3]

2. Graf berarah

Graf yang setiap sisinya memiliki orientasi arah disebuh graf berarah. Pada graf berarah, sebuah sisi dikenal juga sebagai busur (*arc*). Pada graf berarah, (u, v) dan (v, u) menyatakan dua buah sisi yang berbeda. Pada sebuah sisi (u, v), simpul u menyatakan simpul asal (*initial vertex)* dan simpul v menyatakan simpul terminal (*terminal vertex).*



Gambar 4 - Graf berarah[3]

Sisi pada graf dapat memiliki bobot atau tidak. Berdasarkan bobot pada sisinya, graf dapat digolongkan menjadi dua :

1. Graf berbobot (*weighted graph)*

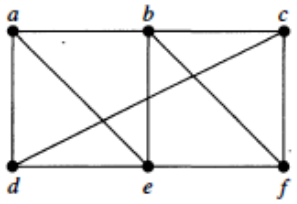
Graf berbobot adalah graf yang setiap sisinya memiliki bobot. Bobot pada sisi graf dapat merepresentasikan kapasitas, biaya, atau keuntungan.

2. Graf tak-berbobot (*unweighted graph)*

Graf tak-berbobot adalah graf yang setiap sisinya tidak memiliki bobot.

2.3 Lintasan dan Sirkuit

Sebuah lintasan (*path)* didefinisikan sebagai urutan dari n buah sisi $e_1, \ldots, e_n$ sedemikian hingga e_1 merupakan (v_0, v_1), e_2 merupakan (v_1, v_2), …, e_n merupakan (v_{n-1}, v_n). Lintasan juga dapat didefinisikan sebagai urutan simpul yang dikunjungi $v_1, v_2, v_3, v_4, \ldots, v_n$, Sebuah lintasan disebut sirkuit (*cycle/circuit)* jika $v_0 = v_n$ atau dengan kata lain simpul asal dan simpul tujuan sama. Sebagai contoh, perhatikan gambar di bawah ini. Rute *a, b, c, f, e, d* merupakan lintasan dari *a* ke *d*. Rute *a, b, f, c, d, a* merupakan sirkut karena memiliki simpul asal dan tujuan yang sama, yaitu *a*.



Gambar 5-Graf *cyclic*[3]

Sebuah graf yang tidak memiliki lintasan siklik disebut graf asiklik (*acyclic graph).* Graf pada Gambar 5 merupakan graf siklik karena memiliki lintasan siklik.

2.4 Algoritma Greedy

Algoritma greedy adalah algoritma yang digunakan untuk menentukan hasil dari persoalan optimasi. Ada 2 macam persoalan optimasi yaitu maksimum dan minimum. Pada setiap langkah, pilihan yang menghasilkan nilai optimal paling besar (local optimal) diambil sebagai bagian dari optimal total (global optimal) dengan harapan bahwa total dari local optimal tersebut merupakan hasil/tidak jauh dari hasil optimal global sebenarnya.

Dua syarat algoritma greedy adalah :

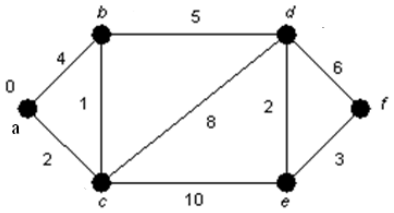
- *Greedy Choice Property*: dari local optimal kita bisa mencapai global optimal tanpa harus mengganti/mempertimbangkan keputusan yang sudah diambil.
- *Optimal Substructure Property*: solusi optimal dari permasalahan dapat ditentukan dari subsolusi permasalahan tersebut.

Elemen-elemen algoritma greedy:

1. Himpunan kandidat, C.
2. Himpunan solusi, S
3. Fungsi seleksi (selection function)
4. Fungsi kelayakan (feasible)
5. Fungsi obyektif

2.5 Algoritma Lintasan Terpendek/Shortest Path

Algoritma lintasan terpendek merupakan algoritma yang digunakan untuk menentukan lintasan yang diambil untuk pergi dari suatu titik ke titik lain. Diberikan graf berbobot $G = (V, E)$. Tentukan lintasan terpendek dari sebuah simpul asal *a* ke setiap simpul lainnya di *G*.



Gambar 6 - Graf Permasalahan Lintasan Terpendek[2]

Ada 4 macam bentuk algoritma shortest path:

a) Lintasan terpendek antara dua buah simpul tertentu (*a pair shortest path*).
b) Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
c) Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).
d) Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

Asumsi yang kita buat adalah bahwa semua sisi berbobot positif.

2.6 Algoritma Dijkstra

Algoritma Dijkstra menggunakan prinsip greedy. Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih. Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek diantara semua lintasannya ke simpul-simpul yang belum terpilih.

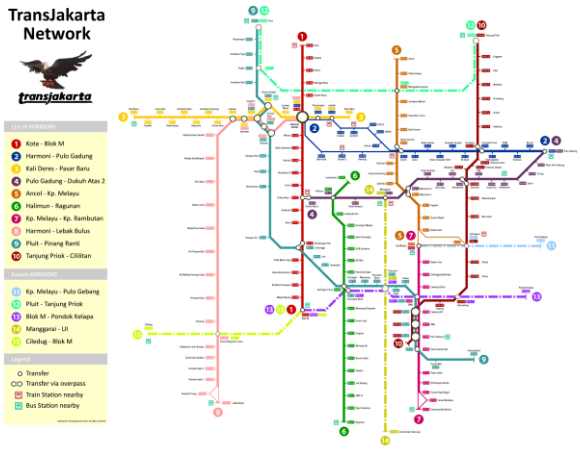
Pseudocode dari algoritma Dijkstra sebagai berikut :

```
procedure Dijkstra (input G: weighted_graph,
input a: intial_vertex)
Deklarasi:
     S : himpunan  simpul solusi
     L : array[1..n] of real  { L(z) berisi
panjang lintasan terpendek dari a ke z}
Algoritma
     for i←1 to n
          L(vi) ← ∞
     end for
     L(a) ← 0; S ← { }
     while z ∉ S do
          u ← simpul yang bukan di dalam S dan
memiliki L(u) minimum
          S ← S ∪ {u}
          for semua simpul v yang tidak terdapat
di dalam S
                  if L(u) + G(u,v) < L(v) then
L(v) ← L(u) + G(u,v)
        end for
    end while
```

III. ANALISIS PERMASALAHAN

3.1. Deskripsi Permasalahan

Seorang pengguna ingin menggunakan bus Transjakarta untuk pergi ke suatu tempat. Karena Transjakarta berhenti hanya di halte-halte yang sudah ditentukan, permasalahan tersebut akan diubah. Seorang pengguna ingin berpindah dari suatu halte menuju halte lain. Perhatikan bahwa penulis tidak menghitung berapa jarak yang ditempuh oleh pengguna untuk menuju halte awal (misal dengan berjalan kaki, naik taksi, dan sebagainya) dan berapa jarak tambahan yang ditempuh untuk menuju ke tempat tujuan dari halte akhir (misal harus berjalan kaki terlebih dahulu).



Gambar 7 - Seluruh trayek Transjakarta[5]

Dari gambar di atas, perhatikan bahwa untuk pergi dari suatu tempat ke tempat lain, mungkin ada lebih dari satu kemungkinan cara. Setiap cara memiliki urutan halte yang dikunjungi dan trayek-trayek bus mana yang harus diambil. Mungkin saja dengan mengambil lebih dari satu trayek, akan lebih cepat sampai dibandingkan hanya dengan mengikuti satu trayek bus saja. Akan tetapi, bisa saja mengambil trayek lebih banyak akan menambah waktu tempuh yang dibutuhkan.

3.2. Bentuk Umum Permasalahan

Sebelum melakukan analisis, penulis menentukan bentuk umum dari permasalahan tersebut:

- Permasalahan tersebut dapat dikategorikan sebagai single source shortest path (SSSP). Hal ini karena kita akan menentukan waktu tempuh sesedikit mungkin dari simpul asal ke simpul-simpul lain.
- Halte merupakan simpul-simpul dari sebuah graf.
- Jalan menyatakan sisi-sisi dari graf.
- Ongkos dari suatu sisi merupakan waktu yang dibutuhkan untuk berangka dari suatu simpul ke simpul lain. Ongkos ini dijamin selalu positif.
- Simpul awal merupakan halte awal pengguna berada.
- Simpul akhir merupakan halte akhir pengguna berada.

3.3. Penerapan Algoritma

Algoritma pencarian trayek ini terdiri dari 3 tahap besar:

a. Mencari jarak terpendek dari simpul asal ke seluruh simpul lain (termasuk simpul tujuan) yang dilakukan dengan menerapkan Dijkstra.
b. Menentukan lintasan yang akan dilalui dari simpul asal ke simpul tujuan.
c. Menentukan trayek-trayek yang akan diambil untuk digunakan oleh pengguna.

Algoritma yang akan dipakai adalah sebagai berikut:

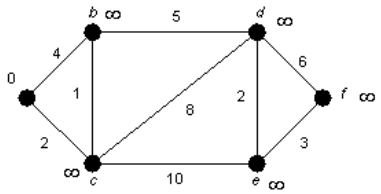
a. Pertama-tama, tentukan simpul awal dari graf.
b. Kemudian, setiap simpul menyimpan *best,* yaitu jarak minimum yang dibutuhkan untuk berangkat dari simpul awal ke simpul tersebut. Pada awalnya, nilai dari *best* adalah tak hingga (nilai yang sangat besar). Selain menyimpan jarak dari simpul awal, setiap simpul juga menyimpan simpul *previous,* yaitu simpul sebelumnya yang diambil oleh lintasan dari simpul awal sebelum pergi ke simpul tersebut. Kegunaaan dari simpul *previous* ini adalah untuk mengkonstruksi lintasan dari simpul awal ke simpul tujuan.
c. Ambil simpul yang belum dikunjungi (pada awal algoritma, semua simpul belum dikunjungi) dengan bobot terkecil. Pada awal algoritma, simpul tersebut adalah simpul asal. Misalkan simpul tersebut adalah *u* dan nilai jarak simpul tersebut ke simpul asal adalah *best[u]*. Untuk setiap *v* sehingga ada jalan dari *u* ke *v,* ganti nilai dari *best[v]* jika lebih dari biaya yang dibutuhkan untuk berangkat dari simpul awal ke simpul u *best[u]* ditambah dengan biaya dari simpul u ke simpul v. Atau *best[v] > best[u] + cost[u][v]*. Untuk setiap *v* yang diupdate, ganti nilai *previous*[*v*] menjadi u.
d. Tandai simpul *u* sebagai simpul yang sudah dikunjungi. Artinya, kita tidak perlu mempertimbangkan nilai dari *best[u]* lagi. Di sinilah prinsip greedy dari Dijkstra.
e. Ulangi langkah c dan d, sampai semua simpul sudah dikunjungi.
f. Cari simpul tujuan, misalkan simpul *f.* Ambil nilai dari *previous[f]*. Artinya adalah untuk mencapai simpul awal menuju simpul *f,* maka sebelumnya kita harus melewati *previous[f]*. Kemudian, kita harus mencari nilai *previous* dari *previous f* tersebut (rekursif). Ulangi langkah ini sehingga didapat rangkaian simpul yang berawal dari simpul *f* dan berakhir di simpul awal.
g. Misalkan kita dapat lintasan terpendek dari poin f di atas adalah R = { v_1 , v_2 , v_3, ... , v_n } dengan v_1 adalah simpul asal dan v_n adalah simpul tujuan. Maka kita tentukan S = { v_1 , v_2 , v_3, ... , v_k } sedemikian hingga *k* adalah maksimum, $k \leq n$ dan ada trayek yang melewati seluruh simpul anggota himpunan S. Artinya adalah kita mencari sebuah sub-lintasan yang merupakan bagian dari lintasan sebenarnya sedemikian hingga ada sebuah bus yang melewati sub-lintasan tersebut. Perhatikan bahwa jika R = S maka kita hanya perlu sebuah bus untuk berangkat dari simpul awal ke simpul tujuan. Namun jika S merupakan himpunan bagian dari R maka artinya kita harus ganti trayek bus. v_1 menandakan simpul awal dan v_k menandakan simpul di mana kita harus transit untuk ganti bus.
h. Ulangi langkah (g), namun kali ini dengan R diganti dengan R – S + {v_k} Artinya adalah kita mencari bus lagi yang melewati R – S dan berangkat dari v_k

3.4. Contoh Instantiasi Permasalahan

Untuk lebih mudah, penulis akan memberikan satu contoh permasalahan.

Misalkan terdapat 6 buah halte (ditandai dengan huruf a, b, c, d, e, f). Garis pada graf menandakan hubungan antar halte. Simpul tujuan adalah simpul f, dan simpulasal adalah simpul a. Asumsikan bahwa untuk setiap pasang simpul yang bertetangga, ada bus yang melalui dua simpul tersebut.

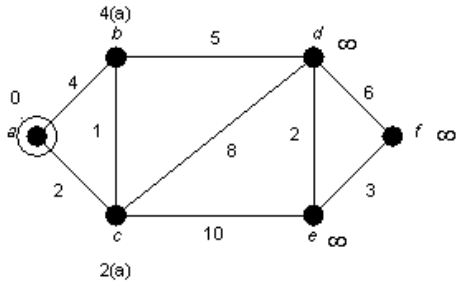
Pada awalnya, setiap simpul diberi nilai *best* = tak terhingga kecuali simpul a.



Gambar 8 - Graf awal[6]

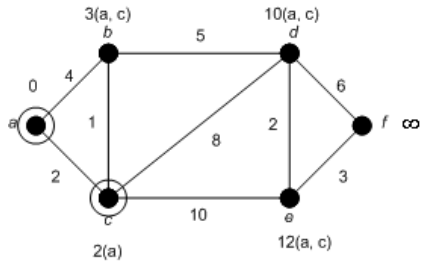
Simpul yang belum terkunjungi dengan bobot minimal adalah a. Maka, perbarui nilai *best* dari seluruh simpul yang bertetangga dengan a yaitu b dan c. Kemudian tandai simpul a sudah dikunjungi (artinya simpul a tidak perlu diupdate lagi dan tidak dicek lagi nilai *best* nya)

Simpul	*best[v] awal*	*best[u] + cost[u][v]*	*best[v]*	previous
b	∞	0+4=4	4	a
c	∞	0+2=2	2	a



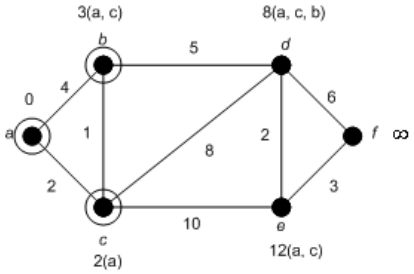
Simpul yang belum terkunjungi dan memiliki nilai minimum adalah *c*. Update tetangga *c* yaitu (b,d,e). Perhatikan a tidak diupdate karena sudah ditandai. Tandai c sudah dikunjungi

Simpul	*best[v] awal*	*best[u] + cost[u][v]*	*best[v]*	previous
b	4	2+1=3	3	c
d	∞	2+8=10	10	c
e	∞	2+10=12	12	c



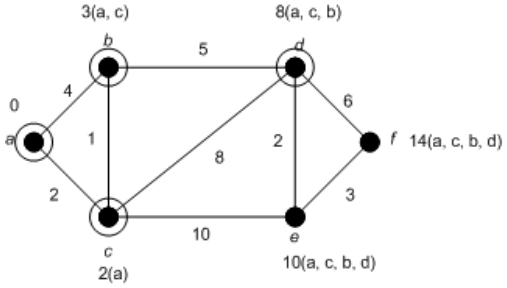
Simpul yang belum terkunjungi dan memiliki nilai minimum adalah *b*. Update tetangga *b* yaitu (d). Tandai *b* sudah dikunjungi.

Simpul	*best[v] awal*	*best[u] + cost[u][v]*	*best[v]*	previous
d	10	3+5=8	8	b



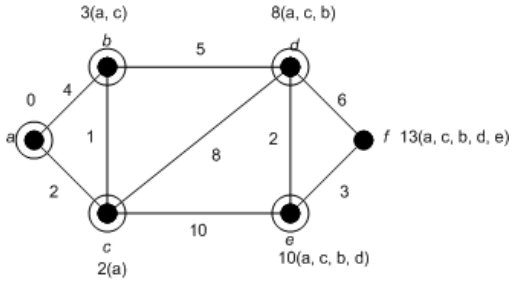
Simpul selanjutnya adalah *d* dengan tentangganya e dan f.

Simpul	*best[v] awal*	*best[u] + cost[u][v]*	*best[v]*	previous
e	12	8+2 = 10	10	d
f	∞	8+6 = 14	14	d



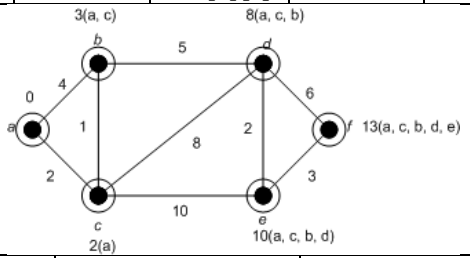
Simpul yang dipilih selanjutnya adalah e

Simpul	*best[v] awal*	*best[u] + cost[u][v]*	*best[v]*	previous
f	14	10+3	13	d



Simpul f tidak ada tetangga yang belum dikunjungi.

Simpul	*best[v] awal*	*best[u] + cost[u][v]*	*best[v]*	previous



Simpul	*best[u]*	previous
a	0	-
b	3	c
c	2	a
d	8	b
e	10	d
f	13	e

Sekarang kita akan menentukan lintasan yang dibutuhkan untuk menempuh a ke f. Perhatikan bahwa walaupun pada gambar sudah ada urutannya namun pada algoritma sebenarnya lintasan tersebut disimpan implisit.

awal	previous	hasil
{f}	previous[f] = e	{e,f}
{e,f}	previous[e] = d	{d,e,f}
{d,e,f}	previous[d] = b	{b,d,e,f}
{b,d,e,f}	previous[b] = c	{c,b,d,e,f}
{c,b,d,e,f}	previous[c] = a	{a,c,b,d,e,f}

Perhatikan bahwa untuk mencapai f dengan ongkos minimal maka lintasan yang harus dilalui adalah a – c – b – d – e – f .

Untuk menentukan trayek-trayek mana yang perlu diambil, mula-mula kita tentukan daftar trayek yang melewati a,b,c,d,e, atau f. Misalkan daftar trayeknya adalah sebagai berikut :

Trayek	Halte yang dilalui
1	a – e – f
2	a – c – b - f – d – e
3	a – c – b – d
4	c – b – d – f

Algoritma yang dilakukan adalah sebagai berikut :
Iterasi pertama : {a,c,b,d,e,f}

Lintasan	Bus yang lewat lintasan tersebut
a	{1,2,3}
a – c	{2,3}
a – c – b	{2,3}
a – c – b – d	{3}
a – c – b – d - e	{}

Lintasan terjauh adalah dengan menempuh a – c – b – d dengan bus 3. Artinya adalah pengguna harus transit pada halte d.
Iterasi kedua : {a,c,b,d,e,f} – {a,b,c,b,d} + {d} = {d,e,f}

Lintasan	Bus yang lewat lintasan tersebut
d	{2,3,4}
d – e	{4}
d – e – f	{}

Lintasan terjauh adalah dengan menempuh d – e dengan bus 4. Pengguna akan transi di halte e
Iterasi ketiga : {d,e,f} – {d,e} + {e} = {e,f}

Lintasan	Bus yang lewat lintasan tersebut
e	{1,2}
e – f	{1}

Ambil bus 1.

Berdasarkan hasil di atas, maka kita dapat rangkum alur perjalanan pengguna sebagai berikut:
Naik bus 3 dari halte a. Turun di halte d.
Naik bus 4 dari halte d. Turun di halte e.
Naik bus 1 dari halte e. Turun di halte f.

3.5. Kompleksitas Algoritma

Berdasarkan algoritma yang penulis bahas, maka ada 2 sub-algoritma utama yang dijalankan. Pertama adalah algoritma untuk mencari lintasan terpendek (yaitu Dijkstra) dan algoritma untuk mencari trayek bus yang melewati lintasan terpendek dan melewati simpul asal dan simpul tujuan.

Algoritma Dijkstra memiliki kompleksitas waktu $O((V+E) \log (V+E))$[4] atau termasuk dalam $O(n \log n)$.

Algoritma mencari trayek memiliki kompleksitas waktu sebagai berikut :

Misalkan untuk mencapai simpul tujuan akan diambil lintasan $\{ v_1 , v_2 , v_3 , \dots , v_n \}$ dengan v_1 adalah simpul asal dan v_n adalah simpul tujuan.

Pada iterasi pertama akan mencari simpul terjauh v_{k_1} ($k_i \in \{1, 2, 3, \dots ,n\}$) dan ada trayek yang melewati v_1, , … v_{k_1}. Iterasi tersebut memiliki kompleksitas $T(n) = ck_1$. Dengan c adalah faktor konstanta yaitu banyak trayek yang ada.

Iterasi selanjutnya akan mencari simpul terjauh v_{k_2} dan ada trayek yang melewati v_{k_1} , … v_{k_2}. Iterasi tersebut memiliki kompleksitas $T(n) = c(k_2 – k_1)$.

Iterasi selanjutnya akan mencari simpul terjauh v_{k_3} dan ada trayek yang melewati v_{k_2} , … v_{k_3}. Iterasi tersebut memiliki kompleksitas $T(n) = c(k_3 – k_2)$.

Iterasi tersebut akan diulang hingga terdapat simpul $v_{k_{m-1}}$ dan ada trayek yang melewati $v_{k_{m-1}}$, … v_{k_m} dan v_{k_m} adalah simpul akhir ($k_m = n$)

Total kompleksitas yang terjadi adalah

$T(n) = c(k_m – k_{m-1}) + c(k_m – k_{m-1}) + \dots + c(k_3 – k_2) + c(k_2 – k_1)$

$T(n) = c(k_m – k_1)$

Karena kita tahu bahwa $k_m = n$ dan $k_1 = 1$ maka

$T(n) = c(n-1) = O(n)$

Total dari kompleksitas algoritma ini adalah

Kompleksitas algoritma Dijkstra + Kompleksitas algoritma mencari trayek

$= O(n \log n) + O(n)$

$= O(n \log n)$

Dengan demikian kecepatan dari algoritma ini sama dengan kecepatan algoritma Dijkstra biasa. Karena $O(n \log n)$ dapat diselesaikan dalam waktu sekitar 1 detik untuk $n = 100000$, maka algoritma tersebut cukup mangkus untuk diterapkan pada jalur Transjakarta (yang total halte-nya hanya sekitar 100-an)

3.6. Penerapan dalam Kehidupan Nyata

Pihak pengelola bus Transjakarta dapat membuat aplikasi (misalnya berbasis *mobile phone)* pemilihan lintasan untuk penggunanya. Pengguna dapat memasukkan tempat asal dan tempat tujuan. Aplikasi akan memberikan halte asal dan halte tujuan yang paling dekat dengan pengguna. Selain itu, pengguna juga akan

diberikan petunjuk trayek bus mana saja yang harus diambil sedemikian hingga total waktu yang ditempuh adalah seminimal mungkin. Dengan demikian, kenyamanan dan aspek fungsionalitas dari bus dapat ditingkatkan.

Penerapan algoritma ini dalam kehidupan nyata bisa jadi menemui suatu kasus yang tidak optimal. Perhatikan bahwa ongkos antar dua simpul adalah waktu tempuh. Pada kenyataannya, waktu tempuh antar dua simpul bisa bervariasi. Pada jam padat mungkin waktu tempuh yang dibutuhkan bisa lebih tinggi. Pada pagi hari atau sore mungkin waktu tempuh bisa lebih rendah. Solusi ini adalah dengan mengubah variabel cost[u][v] menjadi cost[u][v][t], yaitu waktu yang dibutuhkan untuk bergerak dari *u* ke *v* pada waktu ke-*t*

Selain itu, ada kemungkinan bahwa waktu tunggu transit untuk berpindah bus memiliki waktu yang lama. Pada algoritma yang penulis buat, waktu transit dianggap tidak signifikan terhadap waktu tempuh perjalanan sehingga diberi nilai nol. Solusi untuk permasalahan ini adalah dengan memberikan nilai best[v] bukan sebagai minimum dari {best[v],best[u]+cost[u][v]}, namun minimum dari {best[v],best[u]+cost[u][v]+waiting[u]}. Artinya, ditentukan rata-rata waktu yang dibutuhkan untuk menunggu bus transit pada simpul ke-u.

IV. KESIMPULAN

Algoritma Dijkstra dapat digunakan untuk menentukan waktu tempuh terpendek yang dibutuhkan untuk mencapai suatu halte Transjakarta dari halte lain. Selain itu, dengan informasi tersebut maka pengguna Transjakarta juga dapat menentukan trayek-trayek bus mana saja yang perlu diambil sedemikian hingga lintasan tersebut dapat ditempuh.

REFERENSI

[1] Liem, Inggriani, *Diktat Struktur Data*. Bandung: Program Studi Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2008.

[2] Munir, Rinaldi. 2009. *Matematika Diskrit*. Bandung : Informatika.

[3] Rosen, Kenneth H., *Discrete Mathematics and Its Applications, 4th*, McGraw-Hill International 1999.

[4] Halim, Steven. *Competitive Programming 3.* National University of Singapore, 2012.

[5] Indahnesia, http://www.indahnesia.com. Diakses 15 Mei 2014 pukul 11.00.

[6] Munir, Rinaldi. 2009. *Diktat Kuliah IF 2211 Strategi Algoritma.* Bandung : Informatika.

[7] Website Transjakarta, http://transjakarta.co.id. Diakses 15 Mei 2014 pukul 12.00.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 15 Mei 2014

Muhammad Yafi
13512014