Aplikasi Graf pada Persoalan Lintasan Terpendek dengan Algoritma Dijkstra

Adriansyah Ekaputra – 13503021

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

Abstraksi

Makalah ini membahas tentang aplikasi graf pada persoalan shortest path atau lintasan terpendek. Persoalan lintasan terpendek yaitu menemukan lintasan terpendek antara dua atau beberapa simpul lebih yang berhubungan. Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Persoalan ini biasanya direpresentasikan dalam bentuk graf. Graf yang digunakan dalam pencarian lintasan terpendek atau shortest path adalah graf berbobot (weighted graph), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya.

Algoritma yang dipakai pada persoalan lintasan terpendek ada beberapa yaitu Algoritma Dijkstra, algoritma A*, algoritma Floyd-Warshal, dan beberapa algortima lainnya. Akan tetapi, algoritma yang paling populer digunakan yaitu algoritma Dijkstra.

1. Pendahuluan

Persoalan lintasan terpendek yaitu menemukan lintasan terpendek antara dua atau beberapa simpul lebih yang berhubungan. Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Persoalan ini biasanya direpresentasikan dalam bentuk graf. Graf yang digunakan dalam pencarian lintasan terpendek atau shortest path adalah graf berbobot (weighted graph), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya.

Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (*weighted graph*), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya. Asumsi yang kita gunakan di sini adalah bahwa semua bobot bernilai positif. Kata "terpendek" jangan selalu diartikan secara fisik sebagai panjang minimum, sebab kata "terpendek" berbeda-beda makanya bergantung pada tipikal perasoalan yang akan diselesaikan. Namun, secara umum "terpendek" berarti meminimalisasi bobot pada suatu lintasan di dalam graf.

Contoh-contoh terapan pencarian lintasan terpendek misalnya:

1. Misalkan simpul pada graf dapat merupakan kota, sedangkan sisi menyatakan jalan yang menghubungkan dua buah kota. Bobot sisi graf dapat menyatakan jarak antara dua buah kota atau rata-rata waktu tempuh antara dua buah kota. Apabila terdapat lebih dari satu lintasan dari kota A ke kota B, maka persoalan lintasan terpendek di sini adalah menentukan jarak terpendek atau waktu tersingkat dari kota A ke kota B.

2. Misalkan simpul pada graf dapat merupakan terminal komputer atau simpul komunikasi dalam suatu jaringan, sedangkan sisi menyatakan saluran komunikasi yang menghubungkan dua buah terminal. Bobot pada graf dapat menyatakan biaya pemakaian saluran komunikasi antara dua buah terminal, jarak antara dua buah terminal, atau waktu pengiriman pesan (*message*) antara dua buah terminal. Persoalan lintasan terpendek di sini adalah menentukan jalur komunikasi terpenek antara dua buah terminal komputer. Lintasan

terpendek akan menghemat waktu pengiriman pesan dan biaya komunikasi.

Ada beberapa macam persoalan lintasan terpendek, antara lain:

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 terpendekan antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

2. Contoh-contoh Algoritma Persoalan Lintasan Terpendek

Algoritma yang dipakai pada persoalan lintasan terpendek ada beberapa yaitu Algoritma Dijkstra, algoritma A*, algoritma Floyd-Warshal, dan beberapa algortima lainnya. Akan tetapi, algoritma yang paling populer digunakan yaitu algoritma Dijkstra.

2.1 Persoalan Lintasan Terpendek dengan Algoritma *Greedy*

Persoalan yang diambil di sini yaitu single-source shortest path atau lintasan terpendek dari simpul tertentu ke semua simpul yang lain

Persoalan: diberikan graf berbobot G(*V, E*). Tentukan lintasan terpendek dari sebuah simpul asal, *a*, ke setiap simpul lainnya di *G.* Asumsi yang kita buat adalah bahwa semua sisi berbobot positif.

Strategi *greedy* untuk mencari lintasan terpendek adalah sebagai berikut: Karena kita harus meminimumkan panjang lintasan, maka sebagai ukuran optimasi dapat digunakan total jarak pada lintasan yang baru dibentuk. Dalam hal ini, lintasan dibentuk satu per satu. Lintasan berikutnya yang dibentuk ialah lintasan yang meminimumkan jumlah jaraknya.

Algoritma *greedy* untuk mencari lintasan terpendek dirumuskan sebagai berikut:

1. Periksa semua sisi yang langsung bersisian (*incident*) dengan simpul *a*. Pilih sisi yang bobotnya terkecil. Sisi ini menjadi lintasan terpendek pertama. Sebut lintasan itu L_1.

2. Tentukan lintasan terpendek kedua dengan cara berikut:

   (i) hitung: d_1 = Panjang (L_1) + bobot sisi dari simpul akhir L_1 ke simpul *i* yang lain (simpul *i* yang lain adalah simpul *i* yang belum termasuk di dalam L_1)

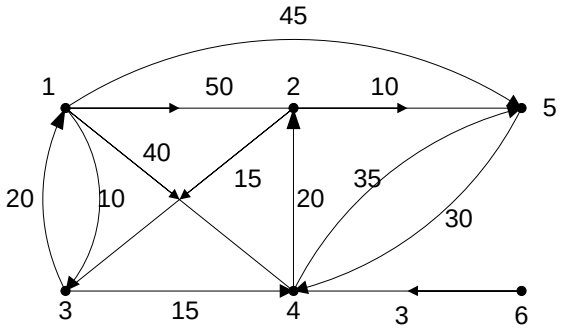
   (ii) pilih d_1 yang terkecil

   bandingkan d_1 dengan bobot sisi (*a*, *i*). Jika bobot (*a*, *i*) lebih kecil daripada d_1, maka lintasan terpendek kedua adalah L_2 = (*a*, *i*), jika tidak, maka

   $L_2 = L_1 \cup$ (sisi dari simpul akhir L_1 ke simpul *i*)

3. Dengan cara yang sama, ulangi langkah 2 untuk menentukan lintasan terpendek berikutnya.

Contoh: Tinjau sebuah graf berarah di bawah ini. Bobot pada setiap sisi dapat menyatakan jarak, ongkos, waktu, dan sebagainya.



Lintasan terpendek dari simpul 1 ke semua simpul lain yang diberikan pada tabel di bawah ini dihasilkan dengan mengunakan strategi *greedy* di atas:

Simpul asal	Simpul tujuanTujuan	Lintasan terpendek	Jarak
1	3	$1 \rightarrow 3$	10
1	4	$1 \rightarrow 3 \rightarrow 4$	25
1	2	$1 \rightarrow 3 \rightarrow 4 \rightarrow 2$	45
1	5	$1 \rightarrow 5$	45
1	6	tidak ada	-

2.1.1 Algoritma Dijkstra

Sampai saat ini, sudah banyak algoritma mencari lintasan terpendek yang pernah ditulis orang. Algoritma lintasan terpendek yang paling terkenal adalah **algoritma Dijkstra** (sesuai dengan nama penemunya, Edsger Wybe Dijkstra). Dari naskah aslitnya, algoritma Dijkstra diterapkan untuk mencari lintasan terpednek pada graf berarah. Namun, algoritma ini tetap benar untuk graf tak-berarah.

Edsger Wybe Dijkstra adalah seorang ilmuwan komputer dan ahli matematika Jerman yang telah memberikan banyak kontribusi pada ilmu komputer. Di antara banyak algoritma yang telah ia temukan, algoritma lintasan terpendek yang dijelaskan di sini merupakan yang terbaik yang pernah diketahui.

Analisis dasar di balik algoritma tersebut adalah sebagai berikut:

- Seperti BFS, algoritma ini secara bertahap membentuk sebuah pohon dari simpul-simpul dan beberapa sisi dalam graf. Akar dari pohon ini adalah source dari pencarian, simpul *s*, seperti yang telah ditentukan pada awal algoritma.

Maka untuk setiap simpul *u*, dipertahankan sebuah *parent pointer* p[*u*] dan sebuah *distance field* d[*u*] (jaraknya diukur dari source)

- Seperti BFS, algoritma ini mempertahankan sebuah queue di mana simpul yang belum diperiksa disimpan. Akan tetapi, ini adalah queue yang diprioritaskan, bukan queue first-in-first-out yang sederhana. Queue yang diprioritaskan ini diurutkan menggunakan field d[] dari simpul-simpulnya (yang akan berubah selama masa eksekusi).
- Tidak seperti BFS, algoritma Dijkstra mempertahankan sebuah set solusi S yang memuat semua simpul yang telah diperiksa, yang jarak minimum terhadap sourcenya sudah ditentukan. Set ini berkembang bersamaan dengan algoritma yang terus bekerja dan memeriksa lebih banyak simpul.
- Di awalnya, semua *parent pointer* diatur menjadi NIL dan semua *distance fields* menjadi tak hingga, kecuali yang dimiliki source, yang diset menjadi nol. Set solusi pun diatur menjadi kosong.

Ini berarti kita tidak mengakumulasi informasi tentang simpul-simpul tersebut. Di sisi lain, queue yang diprioritaskan diisi oleh setiap simpul dalam graf sejak tidak satu pun telah diperiksa. Akan tetapi, sejak semua field d[] adalah tak hingga, mereka tidak akan tersusun dengan urutan tertentu, kecuali sourcenya, yang akan memiliki urutan pertama untuk diekstrak.

- Setiap simpul dalam queue yang diprioritaskan diekstrak dalam sebuah urutan (berdasarkan field d[] dari simpul-simpul).
- Simpul yang telah diekstrak, simpul *u*, ditambahkan ke set solusi *as is*.
- Untuk setiap simpul tetangga *v* dari simpul baru ini, field d[] dicoba diketatkan, untuk membuat perkiraan lebih baik dari yang telah ada. Ini dilakukan dengan cara mengambil perkiraan yang telah ada, dan membandingkannya dengan jumlah dari jarak *u* dari akar (d[*u*] dan beban dari sisi antara *u* dan *v* (w[*u, v*])). Apabila jumlahnya kurang dari perkiraan yang telah ada, maka perkiraan yang telah ada diganti oleh jumlah tersebut. Proses ini disebut *relaxing* sisi yang menghubungkan dua simpul.

Hasilnya adalah pada waktu apapun saat eksekusi, field d[] dari simpul-simpul dalam set solusi terkunci secara permanen dan mewakilkan panjang lintasan terpendek dari simpul tersebut pada source. Unutk simpul yang berada di luar set solusi (kecuali yang diset menjadi tak hingga, berada di tempat paling bawah dari queue yang diprioritaskan), field d[] adalah *worst-case guess*, sebuah panjang lintasan yang telah diketahui dapat dicapai, dengan mengikuti sisi-sisi yang mengarah pada salah satu dari tetangganya dalam set solusi. Mungkin ada lintasan lebih pendek yang tidak mengarah secara langsung pada set solusi, yang dalam hal ini dapat ditemukan pada beberapa saat lain saat semua node yang lain yang berada pada lintasan yang lebih baik tersebut telah dikerjakan.

Apabila jarak *v* benar berubah, maka diperlukan pengaturan posisinya dalam queue yang diprioritaskan dengan *bubbling up*, pada dasarnya mekanisme yang sama yang digunakan dalam memasukkan *item* pada queue yang diprioritaskan. Teknik dalam pemasangan balls & strings mungkin digunakan untuk mengatur ulang tumpukan setelah setiap event.

Set solusi dapat digambarkan sebagai sebuah kepulauan yang terus berkembang, secara bertahap menyerap lebih banyak simpul yang mengambang di sekitarnya. Pada setiap iterasi dari loopnya, harus dibuat sebuah "jembatan" dari "pulau-pulau" ke salah satu simpul yang ada. Aturan yang digunakan dalam memilih simpul yang dihubungkan pada jembatan ini berdasarkan sebuah sistem dalam algoritma ini: pada saat sebuah simpul dalam queue yang diprioritaskan mencapai posisi teratas, jembatan terbaiknya adalah yang menuju simpul yang berada di atas pulau dengan lintasan terpendek ke source.

Apabila kita tidak ingin mengeksplorasi seluruh grafnya, tetapi ingin mengukur jarak ke tujuan tertentu, yang harus dilakukan hanyalah berhenti saat simpul tujuan memasuki *S* dan mengikuti *parent pointer* kembali ke source. Bila kita tidak ingin mengeksplorasi seluruh grafnya, kita akan mendapatkan lintasan terpendek ke setiap titik dari satu source.

Secara garis besar, *pseudo-code* algoritma Dijkstra adalah sebagai berikut:

```
Function Dijkstra (input M : matriks, a : simpul awal) → tabel
( Mencari lintasan terpendek dari simpul awal a ke semua simpul lainnya
  Masukan: matriks ketetanggaan (M) dari graf berbobot G dan simpul
           awal a
  Keluaran: tabel D yang berisi panjang lintasan terpendek dari a ke
            semua simpul lainnya

}

Deklarasi
   D, S : tabel
   I : integer
```

```
Algoritma:
      { Langkah 0 (inisialisasi: )
       For i ← 1 to n do
          S[i] ← 0
          D[i] ← m[a,i]
       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] = Minimum{D[1],
           D[2], ..., D[n]}
           S[j] ← 1 { simpul j sudah terpilih ke dalam lintasan
           Terpendek }
           Hitung D[i] yang baru dari a ke simpul i ∈ S dengan cara
            sebagai berikut:
              D[i] ← Minimum{D[i], (D[j] + M[j,i])}
       endfor
       return D
```

Kode algoritma Dijkstra selengkapnya adalah sebagai berikut :

(a) Deklarasi struktur data

```
const n = ...;    { jumlah simpul di dalam graf }

type matriks = array[1..n, 1..n] of integer

type tabel = array [1..n] of integer
```

(b) Algoritma Dijkstra

```
function Dijkstra (input M : matriks, a : integer) → tabel

{  Mencari lintasan terpendek dari simpul awal a ke semua smpul lainnya

   Masukan : matriks ketetanggaan (M) dari graf berbobot G dan simpul
   awal a

   Keluaran : tabel D yang berisi panjang lintasan terpendek dari a ke
   semua simpul lainnya

}



Deklarasi

   D, S : array[1..n] of integer

   I, j, k min : integer



Algoritma

   { Langkah 0 (inisialisais: )

   for i ← 1 to n do

       S[i] ← 0
```

```
    D[i] ← M[a, i]
Endfor


{ Langkah 1: }
S[a] ← 1        { masukkan simpul awal ke dalam S }
D[a] ← ∞        { jarak dari simpul awal ke simpul awal diberi nilai
                  tak hingga }
{ Langkah 2, 3, ..., n-1 : }
for k ← 2 to n − 1 do
    { cari simpul j sedemikian sehingga S[j] = 0
      Dan D[j] = Minimum{D[1], D[2], ..., D[n]}
    min ← D[1]
    j ← 1
    for i ← 2 to n do
      if (S[i] = 0) and (D[i] < min) then
          min ← D[i]
          j ← i
      endif
    endfor
    S[j] ← 1 { simpul j sudah terpilih ke dalam lintasan terpendek }
    {hitung D[i] yang baru dari a ke simpul i ∉ S }
    for i ← 1 to n do
       if S[i] = 0 then
         if D[i] > (D[j] + jarak[j,]) then
            D[i] ← D[j] + jarak[j,1]
         endif
       endif
```

```
    endfor
  endfor
  return D
```

Kompleksitas waktu algoritma Dijkstra adalah $O(n^2)$. Hal ini dijelaskan sebagai berikut :

(a) Instruksi di dalam kalang `for` pertama :

```
For i←1 to n do
    ...
Endfor
```

dikerjakan sebanyak *n* kali, sehingga waktu kompleksitasnya adalah *O(n)*.

(b) kalang bersarang

(i) kalang `for` terluar :

```
For k ← 2 to n − 1 do
...
 Endfor
```

Dilakukan sebanyak *n -2* kali

(ii) kalang `for` terdalam untuk menentukan *D[i]* minimum :

```
For i ← 2 to n do
   ....
Endfor
```

Membutuhkan waktu dalam *O(n)*.

(iii) kalang for berikutnya untuk memperbarui nilai D[1..n]

```
For i ← 1 to n do
    .....
Endfor
```

Membutuhkan waktu dalam *O(n)*.

Dengan demikian, total waktu yang dibutuhkan adalah :

$$T(n) = O(n) + (n-2)[O(n)+O(n)]$$

$$= O(n) + O((n-2)(n))$$

$$= O(n) + O(n^2)$$

$$= O(n^2)$$

Sebagai contoh ilustrasi, tinjau kembali graf berarah pada contoh dengan matriks ketetanggaan *M* sebagai berikut:

	j=1	2	3	4	5	6
i = 1	0	50	10	40	45	∞
2	∞	0	15	∞	10	∞
3	20	∞	0	15	∞	∞
4	∞	20	∞	0	35	∞
5	∞	∞	∞	30	0	∞
6	∞	∞	∞	3	∞	0

Perhitungan lintasan terpendek dari simpul awal *a* = 1 ke semua simpul lainnya ditabulasikan sebagai berikut (termasuk nilai-nilai tabel *S* dan *D*):

Lelaran	Simpul yang dipilih	Lintasan	*S*						*D*					
			1	2	3	4	5	6	1	2	3	4	5	6
Inisial	-	-	0	0	0	0	0	0	0	50 *(1,2)*	10 *(1,3)*	40 *(1,4)*	45 *(1,5)*	∞ *(1,6)*
1	1	1	1	0	0	0	0	0	∞	50 *(1,2)*	10 *(1,3)*	40 *(1,4)*	45 *(1,5)*	∞ *(1,6)*
2	3	1, 3	1	0	1	0	0	0	∞	50 (1,2)	10 (1,3)	25 (1,3,4)	45 (1,5)	∞ (1,6)
3	4	1, 3, 4	1	0	1	1	0	0	∞	45 *(1,3,4,2)*	10 *(1,3)*	25 *(1,3,4)*	45 *(1,4)*	∞ *(1,6)*
4	2	1, 3, 4, 2	1	1	1	1	0	0	∞	45 *(1,3,4,2)*	10 *(1,3)*	25 *(1,3,4)*	45 *(1,4)*	∞ *(1,6)*
5	5	1, 5	1	1	1	1	1	0	∞	45	10	25	45	∞

Jadi, lintasan terpendek dari:
1 ke 3 adalah 1, 3 dengan panjang = 10
1 ke 4 adalah 1, 3, 4 dengan jarak = 25
1 ke 2 adalah 1, 3, 4, 2 dengan jarak = 45
1 ke 5 adalah 1, 5 dengan jarak = 45
1 ke 6 tidak ada

3. Kesimpulan

Kesimpulan yang dapat diambil dari pembahasan tentang Aplikasi Graf pada Persoalan Lintasan Terpendek dengan Algoritma Dijkstra adalah :

1. Persoalan Lintasan Terpendek dapat direpresentasikan dalam bentuk graf berbobot (weighted graph).

2. Ada beberapa macam persoalan lintasan terpendek, antara lain:

    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 terpendekan antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

3. Algoritma lintasan terpendek yang paling terkenal adalah **algoritma Dijkstra** untuk persoalan lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).

4. Algoritma Dijkstra mempertahankan sebuah set solusi S yang memuat semua simpul yang telah diperiksa, yang jarak minimum terhadap sourcenya sudah ditentukan. Set ini berkembang bersamaan dengan algoritma yang terus bekerja dan memeriksa lebih banyak simpul.

5. Kompleksitas waktu algoritma Dijkstra adalah $O(n^2)$.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. (2003). *Matematika Diskrit*. Bandung : Penerbit Informatika

[2] _____. (2005). *Diktat Kuliah IF2251 Strategi Algoritmik*. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] Shortest Path Problem – Wikipedia, the free encyclopedia. (2006). url : *http://en.wikipedia.org/wiki/Shortest_path_problem*. Tanggal akses: 31 Desember 2006 pukul 16.30.

[4] Dijkstra's Algorithm – Wikipedia, the free encyclopedia. (2006). url : *http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm*. Tanggal akses: 31 Desember 2006 pukul 16.30.

[5] CS251B -- Topic #29: Shortest path algorithms. (2006). url : *http://www.cs.mcgill.ca/~cs251/OldCourses/1997/topic29/* . Tanggal akses: 31 Desember 2006 pukul 16.30