ALGORITMA DJIKSTRA, BELLMAN-FORD, DAN FLOYD-WARSHALL UNTUK MENCARI RUTE TERPENDEK DARI SUATU GRAF

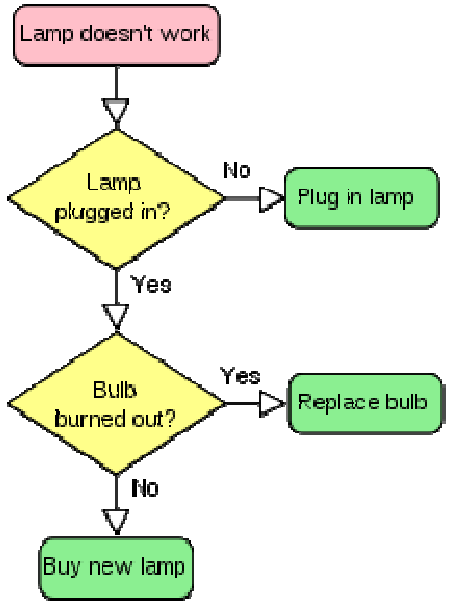
Dibi Khairurrazi Budiarsyah - 13509013
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509013@stei.itb.ac.id

Abstrak--Makalah ini terdiri dari 3 bagian, pada bagian pertama yakni pendahuluan terdapat definisi dari algoritma serta penemu-penemu algoritma. Algoritma adalah sekumpulan perintah yang digunakan untuk menyelesaikan suatu masalah. Terdapat 3 algoritma yang dibahas pada bagian kedua makalah ini yakni Algoritma Dijkstra, Algoritma Bellman-Ford, dan Algoritma Floyd-Warshall. Algoritma-algoritma ini bertujuan untuk menemukan rute terpendek dari suatu graf. Pada bagian ini, terdapat metode dan pseudocode dari masing-masing algoritma. Bagian terakhir adalah kesimpulan dan referensi. Kesimpulannya adalah algoritma dijkstra merupakan algoritma yang paling cepat dalam menentukan rute terpendek, namun tidak dapat menangani sisi berbobot negatif. Algoritma Bellman-Ford dan Floyd-Warshall dapat melakukannya, namun waktu yang dibutuhkan kedua algoritma ini lebih lama daripada algoritma dijkstra. Ketiga algoritma ini tidak bisa menangani graf dengan siklus berbobot negatif. Algoritma ini cukup luas pemanfaatannya dalam kehidipan sehari-hari, salah satunya adalah menentukan rute tercepat dari suatu kota ke kota yang lainnya, dll.

Kata kunci: Algoritma, Bellman-Ford, Dijkstra, Floyd-Warshall.

1. PENDAHULUAN

Algoritma merupakan kumpulan perintah-perintah yang digunakan untuk menyelesaikan suatu masalah. Kata algoritma sendiri berasal pelatinan nama seorang ahli matematika asal Uzbekistan yang bernama Al -Khawarizmi. Pada awalnya kata algorisma merujuk pada aturan-aturan aritmetis untuk menyelesaikan masalah dengan menggunakan bilangan numerik dari Arab. Namun pada abad ke-18, kata Algoritma mencakup semua prosedur atau urutan langkah yang jelas dan dibutuhkan guna menyelesaikan suatu masalah.



Gambar 1. Ilustrasi dari sebuah algoritma

Algoritma (kumpulan perintah) ini dapat diterjemahkan secara bertahap dimulai dari awal hingga akhir perintah. Masalah yang ada bisa apa saja dan menyangkut apa saja, asalkan ada kriteria kondisi awal yang harus dipenuhi sebelum algoritma itu dijalankan. Algoritma dapat berakhir apabila semua kondisi awal memenuhi kriteria, berbeda dengan heuristik yang hanya dapat digunakan sebagai pedoman saja. Algoritma terkadang memiliki iterasi (pengulangan) atau memerlukan keputusan seperti boolean atau perbandingan sampai pekerjaan nya selesai dilaksanakan.

Algoritma memiliki kompleksitas yang berbeda-beda, tergantung dari banyaknya komputasi yang harus dilakukan untuk menyelesaikan suatu masalah yang ada. Dengan kata lain, algoritma dengan jumlah komputasi yang relatif sedikit untuk menyelesaikan masalah dapat dikatakan memiliki kompleksitas yang rendah, dan algoritma yang mempunyai jumlah komputasi yang relatif banyak memiliki kompleksitas yang tinggi. Biasanya algoritma yang kompleksitasnya rendah lebih disenangi karena lebih mudah

dimengerti, lebih singkat dan cepat, dan tidak memberatkan sistem.

Secara umum terdapat tiga jenis algoritma yakni *Divide and Conquer*, *Dynamic Programming*, dan Metode Serakah. *Divide and Conquer* adalah suatu paradigma untuk membagi suatu permasalahan yang besar menjadi beberapa permasalahan kecil, pembagian terus dilakukan hingga ditemukan cara menyelesaikan suatu persamaan kecil. *Dynamic Programming* mirip dengan *Divide and Conquer*, suatu masalah dibagi menjadi beberapa sub-struktur yang optimal untuk dipecahkan, perbedaanya adalah karakter permasalahan yang dihadapi. Metode serakah ini mirip dengan jenis yang sebelumnya, namun jawaban dari suatu sub-masalah tidak perlu diketahui pada tiap tahap dan menggunakan pilihan "serakah" tentang apa yang paling baik pada saat itu.

Terdapat beberapa algoritma untuk menemukan rute terpendek dalam suatu graf, antara lain algoritma Dijkstra, algoritma Bellman-Ford, dan algoritma Floyd-Warshall.

Algoritma Dijkstra ditemukan oleh orang yang bernama Edsger Wybe Dijkstra. Ia merupakan seorang ahli komputer asal belanda yang lahir di Rotterdam pada tanggal 11 Mei 1930 dan meninggal dunia pada tanggal 6 Agustus 2002.

Gambar 2. Edsger Wybe Dijkstra

Algoritma Bellman-Ford ditemukan oleh Richard E. Bellman, seorang matematikawan yang lahir di New York pada tahun 1920, meninggal pada tanggal 19 maret 1984 dan Ford.

Gambar 3. Robert W. Floyd

Algoritma Floyd-Warshall ditemukan oleh Stephen Warshall dan Robert W. Floyd. Stephen Warshall lahir di New York pada tahun 1935 dan meninggal pada tanggal 11 desember 2006. Robert W. Floyd, lahir di New York pada tanggal 8 juni 1936 dan meninggal pada tangggal 25 September 2001.

Gambar 4. Stephen Warshall

Selain algoritma-algoritma tersebut, masih ada algoritma lain yang bertujuan sama, contohnya algoritma A* yang mirip dengan algoritma dijkstra.

2. METODE

Untuk mencari rute terpendek, dapat menggunakan beberapa metode, antara lain dengan menggunakan algoritma Dijkstra, algoritma Bellman-ford, dan algoritma Warshall. Pada masing-masing sub-bab akan dijelaskan metode dan pseudocode dari masing-masing algoritma.

2.1 Algoritma Dijkstra

Algoritma Dijkstra adalah suatu algoritma rakus dimana algoritma ini digunakan untuk mencari rute permasalahan terpendek antara simpul sumber dan simpul tujuan untuk sebuah graf berarah berdasarkan bobot pada sisi yang bernilai tidak negatif.

Algoritma Dijkstra bekerja dengan cara mengunjungi simpul-simpul yang ada, dimulai dari simpul sumber. Kemudian algoritma ini memilih simpul-simpul yang lokasi nya terdekat dan dilakukan secara berulang lalu kemudian menghitung total bobot semua sisi yang dilewati untuk mencapai simpul tujuan.

Pada algoritma dijkstra total biaya untuk mencapai suatu simpul adalah seperti ini

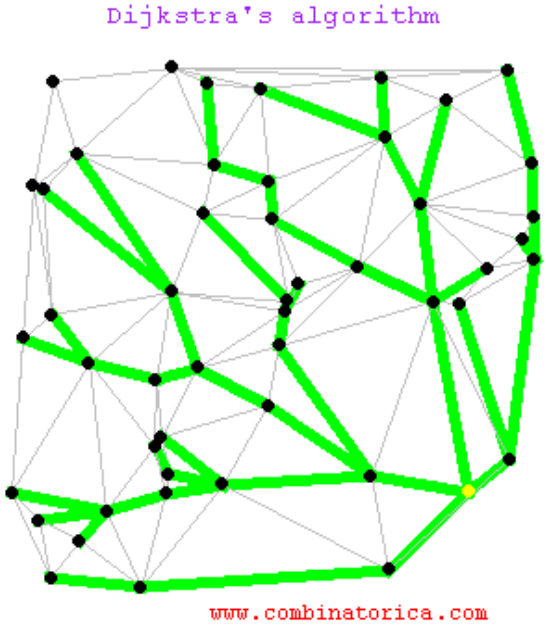
$$f(n_i) = g(n) + c(n,n_i) \qquad (1)$$

Algoritma Dijkstra dijamin dapat menemukan rute terpendek asalkan tidak terdapat bobot negatif pada setiap sisi dalam graph [10].

Input untuk algoritma ini adalah sebuah graph yang berarah dan berbobot *G*. Dan sebuah sumber *vertex s* dalam *G*. Dan *V* merupakan himpunan semua *vertice* dalam grapf *G*. Setiap sisi dari graph ini adalah pasangan *vertice* (*u*,*v*) yang melambangkan hubungan *vertex u* dengan *vertex v* dan himpunan semua tepi *E*.

$$w : E \rightarrow [0, \infty) \qquad (2)$$

jadi *w*(*u*, *v*) merupakan jarak non-negatif dari *vertex u* ke *vertex v*. Biaya sebuah sisi dapat dianggap sebagai jarak antara dua buah vertex dan merupakan jumplah jarak tiap sisi dalam jalur tersebut.



Gambar 5. Ilustrasi Algoritma Dijkstra [12].

Contoh, algoritma ini digunakan untuk menghitung jarak dari *s* ke *t* dalam *V*.

```
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
```

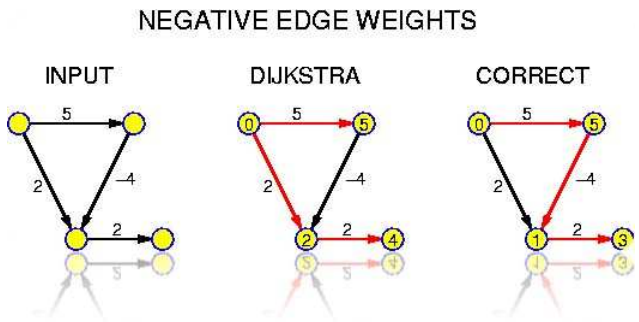
Pada baris kedua merupakan proses inisialisasi, pada baris ke lima merupakan jarak dari s ke s. Baris ke tujuh merupakan kumpulan dari semua vertice, dan baris ke delapan dan seterusnya merupakan algoritmanya itu sendiri.

Terdapat Pseudocode lain tentang Algoritma Djikstra.

```
1       DIJKSTRA(G, s, w)
2         for each vertex u in V
3           d[u] := infinity
4           p[u] := u
5           color[u] := WHITE
6         end for
7         color[s] := GRAY
8         d[s] := 0
9         INSERT(Q, s)
10        while (Q != Ø)
11          u := EXTRACT-MIN(Q)
12          S := S U { u }
13          for each vertex v in Adj[u]
14            if (w(u,v) + d[u] < d[v])
15              d[v] := w(u,v) + d[u]
16              p[v] := u
17            if (color[v] = WHITE)
18                  color[v] := GRAY
19              INSERT(Q, v)
20            else if (color[v] = GRAY)
21              DECREASE-KEY(Q, v)
22            else
23                  ...
24          end for
25          color[u] := BLACK
26        end while
27      return (d, p)
```

Pada baris ke-9 dilakukan pencarian *vertex s*, lalu pada baris ke 11, dilakukan pengecekan terhadap *vertex u*. Kemudian pada baris 13, dilakukan pengecekan pinggiran/ujung *(u,v)*. Lalu pada baris ke-19 dilakukan

pencarian *vertex v*. Dan pada baris ke-24, menyelesaikan *vertex u*.



Gambar 6. Ilustrasi kelemahan algoritma dijkstra.

Waktu yang dibutuhkan oleh algoritma dijkstra adalah $O(|V|^2)$. Waktu ini dapat dikurangi menjadi O(|E|log|V|) apabila *heap* digunakan untuk menjaga {v in $V\backslash S_i$: L(v) < infinity} [11].

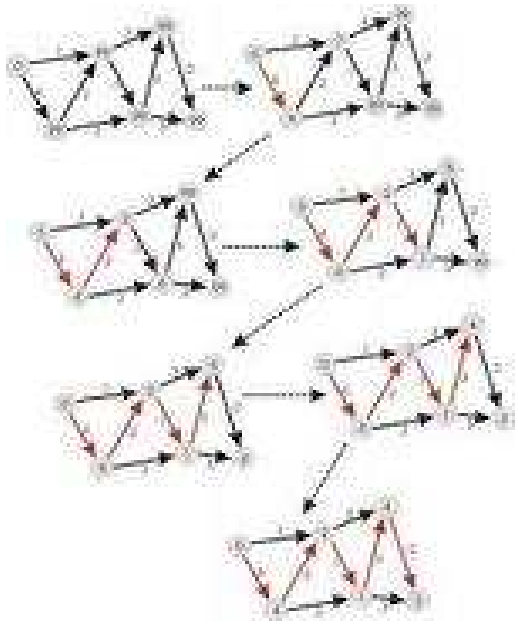
2.2. Algoritma Bellman-Ford

Algoritma Bellman-Ford adalah algoritma yang digunakan untuk menghitung jarak terpendek dari suatu graf berbobot. Algoritma dijkstra memang lebih cepat dalam melakukan hal yang sama, namun algoritma dijkstra hanya berlaku apabila tidak ada sisi yang berbobot negatif, sedangkan algoritma Bellman-Ford tetap berlaku.

Pseudocode untuk algoritma ini adalah sebagai berikut. Bobot ekivalen dengan jarak pada sebuah sisi.

```
// definisi tipe data dalam 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)
// argumennya ialah graf, dengan bentuk daftar titik dan
//sisi. Algoritma ini mengubah titik-titik dalam semuatitik
//sehingga atribut jarak dan sebelumnya menyimpan jarak
//terpendek.

// persiapan
for each titik v in semuatitik :
      if v is dari then v.jarak = 0
      else v.jarak = takhingga
      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        // uv adalah sisi dari u ke v
            if v.jarak > u.jarak + uv.bobot
                  v.jarak := u.jarak + uv.bobot
                  v.sebelum := u

// cari sirkuit berbobot(jarak) negatif
for each sisi uv in semuasisi :
      u := uv.dari
      v := uv.ke
      if v.jarak > u.jarak +uv.bobot
            Error “graf mengandung siklus berbobot total
            negatif"
```

Algoritma Bellman-Ford menggunakan waktu sebesar O(V.E), V adalah banyaknya sisi dan E adalah banyaknya titik.



Gambar 7. Ilustrasi Algoritma Bellman-Ford.

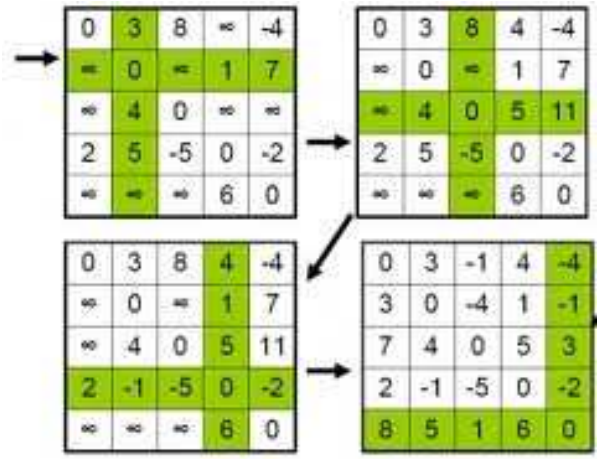
Algoritma bellman memiliki suatu kelemahan pada graf yang memiliki negative cycle, memang graf yang memilki negative cycle tidak dapat dihitung shortest pathnya[13].

2.3. Algoritma Floyd Warshall

Algoritma Floyd-Warshall memiliki input graf yang berupa titik (V) dan sisi (E). Sisi boleh berbobot negatif, namun tidak diperbolehkan memiliki siklus negatif. Algoritma ini menghitung bobot terkecil dari semua jalur dan melakukan sekaligus untuk semua titik. Algoritma ini memiliki waktu $O(|V|^3)$.

Berikut ini adalah salah satu pseudocode dari algoritma Floyd-Warshall.

```
Function fw(int [1 .. n, 1 .. n] graph) {
    // inisialisasi
    Var int [1 .. n, 1 .. n] jarak := graph
    Var int [1 .. n, 1 .. n] sebelum
    For i from 1 to n
        For j from 1 to n
            If jarak[i,j] < takhingga
                Sebelum[i,j] := i
    // perulangan utama
    For k from 1 to n
        For i from 1 to n
            For j from 1 to n
                If jarak[i,j] > jarak [i,k] + jarak[k,j]
                    Jarak[i,j]=jarak[i,k] + jarak[k,j]
                    Sebelum[i,j] = sebelum[k,j]
```



Gambar 8. Ilustrasi Algoritma Floyd-Warshall.

IV. KESIMPULAN

Terdapat banyak algoritma di dunia. Untuk memecahkan masalah rute terpendek, dapat menggunakan Algoritma Dijkstra, Algoritma Bellman-Ford, atau Algoritma Floyd-Warshall. Algoritma Dijkstra dapat digunakan untuk mencari rute terpendek dari suatu grafik berbobot tanpa nilai negatif, waktu yang dibutuhkan untuk operasi ini adalah $O(|V|^2)$ dimana V adalah jumlah node. Algoritma Bellman-Ford fungsinya sama dengan Algoritma Dijkstra, hanya saja prosesnya memakan waktu yang lebih lama yakni O(V.E) dengan E adalah jumlah sisi/edge. Meskipun memakan waktu lebih lama, Algoritma Bellman-Ford dapat menangani sisi berbobot negatif. Algoritma Floyd-Warshall juga dapat menangani kasus serupa, waktu yang dibutuhkan adalah $O(|V|^3)$. Untuk sisi tanpa bobot negatif, algoritma dijkstra lebih cepat dalam menentukan rute terpendek, namun apabila ada sisi berbobot negatif, algoritma Bellman lebih disukai. Dari beberapa algoritma ini terdapat kesamaan, yakni semuanya tidak dapat menentukan rute terpendek dari graf bersiklus negatif. Aplikasi dari algoritma-algoritma ini adalah dalam menentukan rute terpendek yang dapat diambil, misalnya saja rute perjalanan dari suatu kota ke kota yang lain.

.

REFERENSI

[1] http://id.wikipedia.org/wiki/Algoritma. (14/12/2010 23:55).
[2] http://id.wikipedia.org/wiki/Algoritma_Dijkstra. (14/12/2010 23:58)
[3] http://id.wikipedia.org/wiki/Edsger_Dijkstra (14/12/2010 00:00)
[4] http://id.wikipedia.org/wiki/Algoritma_Bellman-Ford (15/12/2010 00:01)
[5] http://id.wikipedia.org/wiki/Algoritma_Floyd-Warshall (15/12/2010 00:03)
[6] http://en.wikipedia.org/wiki/Richard_Bellman (15/12/2010 00:18)
[7] http://en.wikipedia.org/wiki/Stephen_Warshall (15/12/2010 00:25)
[8] http://en.wikipedia.org/wiki/Robert_Floyd (15/12/2010 00:35)
[9] http://www.boost.org/doc/libs/1_43_0/libs/graph/doc/dijkstra_shortest_paths.html (15/12/2010 00:40)
[10] http://www.ittelkom.ac.id/library/index.php?view=article&catid=20%3Ainformatika&id=161%3Aalgoritma-dijkstra&option=com_content&Itemid=15 (15/12/2010 00:45)
[11] http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml (15/12/2010 00:52)
[12] http://www.cs.sunysb.edu/~skiena/combinatorica/animations/dijkstra.html (15/12/2010 00:53)
[13] http://yohandamandala.blogspot.com/2009/11/algortima-shortest-path-bellman-ford.html (15/12/2010 01:30)

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 Desember 2010

Dibi Khairurrazi Budiarsyah
13509013