Perbandingan Algoritma Dijkstra (*Greedy*), Bellman-Ford (BFS-DFS), dan Floyd-Warshall (*Dynamic Programming*) dalam Pengaplikasian Lintasan Terpendek pada *Link-State Routing Protocol*

Michell Setyawati Handaka / 135 08 045[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*if18045@students.if.itb.ac.id*

Pada algoritma* Interior Gateway Routing *dengan menggunakan* link-state routing protocol *diperlukan sebuah algoritma pencarian jalur terpendek yang mana harus memenuhi penggunaan ruang memori yang kecil mengingat memori router terbatas dan mahal harganya dan kebutuhan waktu komputasi rendah karena* transfer time *yang menggunakan hasil komputasi memiliki nilai yang relatif kecil sehingga penambahan kompleksitas waktu sekecil apapun akan memberikan dampak yang signifikan. Akan dibahas tiga buah pendekatan algoritma pencarian jalur terpendek, yakni algoritma Dijkstra yang berbasiskan strategi* greedy*, algoritma Bellman-Ford yang merupakan penurunan dari strategi BFS – DFS, dan algoritma Floyd-Warhall yang merupakan salah satu aplikasi dari* dynamic programming*.

Bellman-Ford, Dijkstra, Floyd-Warshall,* link-state routing protocol*.

I. PENDAHULUAN

Dalam Model *ISO / OSI Layer* pada jaringan komputer, *routing protocol* merupakan jantung inti pembentuk layanan yang ditawarkan oleh layer ketiga (*network layer*) kepada layer yang berada tepat satu di atasnya (*transport layer*).

Routing Protocol adalah sebuah protokol yang menspesifikasikan keterhubungan antar router dalam sebuah jaringan –dan bagaimana mereka dapat saling berkomunikasi untuk menyebarkan informasi– sedemikian cara sehingga dimungkinkan dilakukannya pemilihan lintasan antara setiap pasangan dua nodal atau lebih dalam jaringan yang terhubung; adapun proses pemilihannya dilakukan oleh *routing protocol algorithm*. Protokol ini sangatlah diperlukan dalam *network layer* mengingat fungsi utama dari *network layer* adalah merutekan lintasan bagi paket-paket dari mesin asal ke mesin tujuan, dimana pada umunya paket-paket ini membutuhkan lebih dari satu *hop* dalam perjalanannya untuk dapat sampai ke tempatnya.

Setiap router pada mulanya hanya memiliki pengetahuan mengenai jaringan yang terhubung langsung dengannya, kemudian *routing protocol* membagikan informasi ini dimulai dari tetangga terdekat hingga seluruh router yang terdapat dalam jaringan sehingga pada akhirnya setiap router mengetahui topologi dari jaringan yang sesungguhnya.

Dewasa ini, terdapat banyak tipe dari routing protocol, beberapa di antaranya yang paling banyak digunakan secara luas pada jaringan IP dapat dikelompokan ke dalam tiga kelas besar sepeti pada berikut ini :

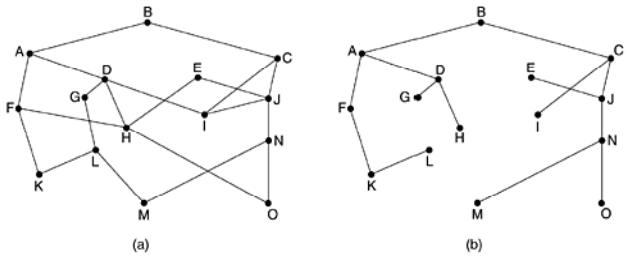
1. *Interior Gateway Routing* menggunakan *link-state routing protocol*
2. *Interior Gateway Routing* menggunakan *path vector* atau yang umum juga dikenal sebagai *distance vector protocols*
3. *Exterior Gateway Routing*

Link-state routing protocol yang akan dibahas lebih mendalam selanjutnya adalah sebuah algoritma perutean *adaptive* di dalam suatu *autonomous system* yang mana bertujuan untuk menciptakan kembali topologi yang benar pada suatu internetwork.

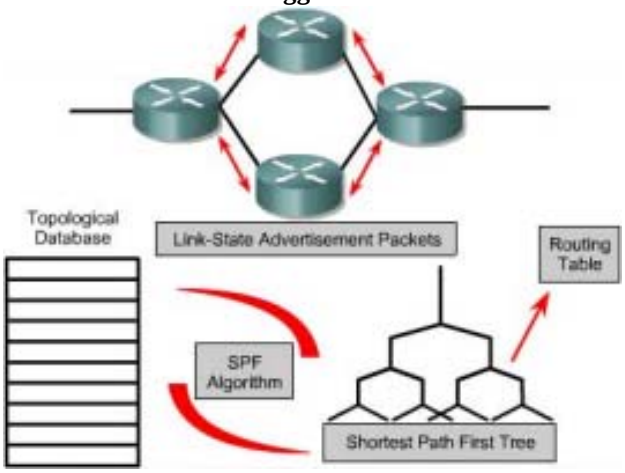
Seperti algoritma perutean lain pada umumnya, *link-state routing protocol* menggunakan prinsip optimalitas yang antara lain bunyinya adalah sebagai berikut ini, yakni

> **Prinsip Optimalitas**
> Jika router J berada dalam lintasan optimum antara router I dan router K, maka lintasan dari J ke K juga optimum dan melalui rute yang sama.

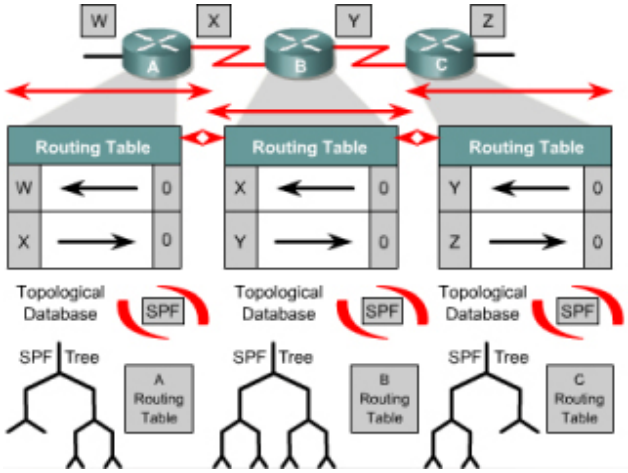
untuk membangun sebuah pohon tenggelam yang merepresentasikan lintasan optimal dari setiap sumber ke setiap tujuan. Pohon ini nantinya akan digunakan untuk proses pengambilan keputusan mengenai lintasan mana yang sebaiknya dipilih.



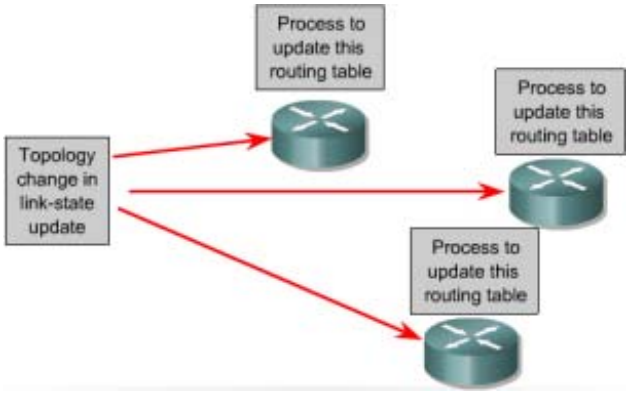
Gambar 1. Pohon Tenggelam dari sebuah Subnet



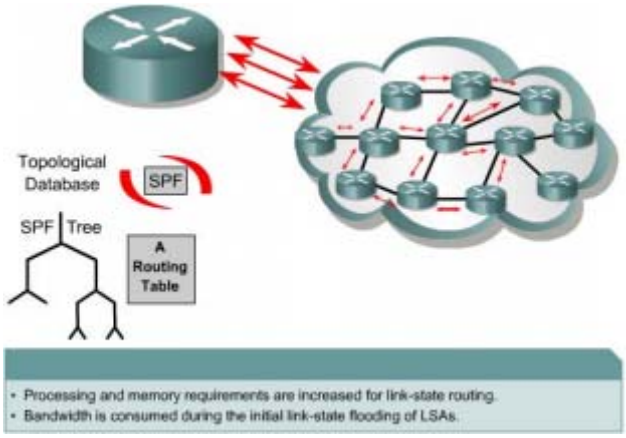
Gambar 2. Konsep *Link-State Routing*



Gambar 3. Discovery Jaringan *Link-State*



Gambar 4. Perubahan Topology *Link-State*



Gambar 5. *Link-State Concern*

Ide dasar dari *link-state routing protocol* sangatlah mudah dan dapat dinyatakan dalam lima bagian. Setiap router harus melakukan hal di bawah ini :

1. Mengetahui semua tetangga terdekatnya dan mempelajari alamat mereka.
2. Mengukur delay dan biaya ke masing-masing tetangganya.
3. Membuat sebuah paket yang memberitahu semua router lainnya mengenai informasi yang dimiliki.
4. Mengirimkan paket tersebut.
5. Menghitung lintasan terpendek ke semua router lainnya.

Terdapat banyak strategi yang dapat digunakan dalam pemilihan algoritma untuk pendekatan penyelesaian kebutuhan no. 5, antara lain yang paling populer adalah algoritma Dijkstra (*greedy*), algoritma Bellman-Ford (penurunan BFS - DFS), dan algoritma Floyd-Warshall (*dynamic programming*).

II. DIJKSTRA (*GREEDY*)

Algoritma Dijkstra mencari lintasan terpendek dalam sejumlah langkah dengan menggunakan strategi *greedy* sebagai berikut :

Strategi *greedy* pada Algoritma Dijkstra
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.

Misalkan sebuah graf berbobot dengan *n* buah simpul dinyatakan dengan matriks ketetanggan $M = [m_{ij}]$, yang dalam hal ini memenuhi :

$[m_{ij}]$ = bobot sisi (i, j)

$[m_{ii}] = 0$

$[m_{ij}] = \infty$, jika tidak ada sisi dari simpul *i* ke simpul *j*

Selain matriks *M*, juga akan digunakan tabel $S = [s_i]$, yang dalam hal ini berlaku :

$[s_i]$ = 1, jika dan hanya jika simpul *i* termasuk ke dalam lintasan terpendek

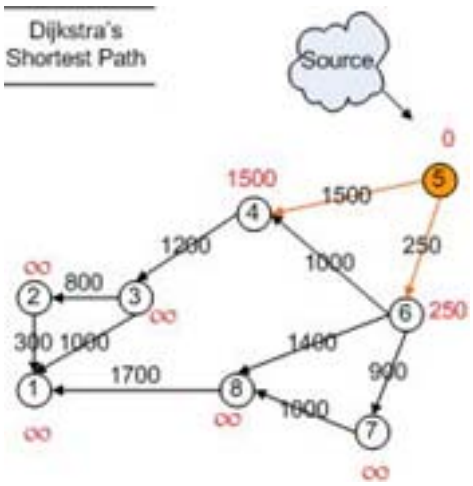
$[s_i]$ = 0, jika dan hanya jika simpul *i* tidak termasuk ke dalam lintasan terpendek

dan keluarannya adalah sebuah tabel $D = [d_i]$, yang dalam hal ini menyatakan :

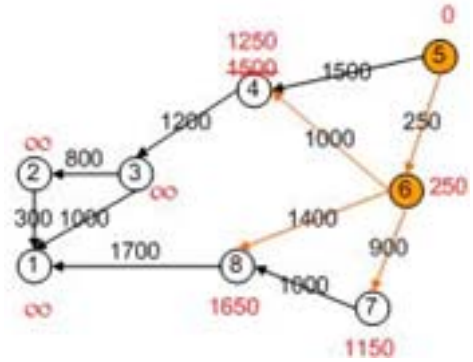
$[d_i]$ = panjang lintasan dari simpul awal sumber *a* ke simpul akhir tujuan *i*.

Dengan simpul awal adalah *a*, dan dengan jarak dari simpul *i* diartikan sebagai jarak antara simpul *a* dan *i*, maka algoritma Djikstra akan menginitialisasikan nilai jarak awal dan memperbaikinya tahap demi tahap.

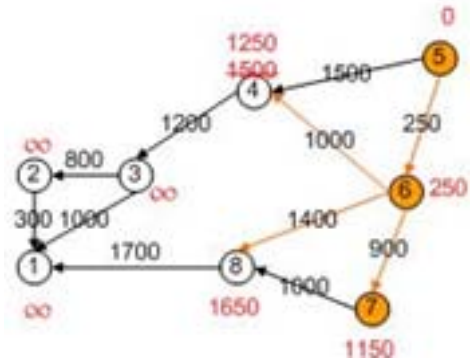
1. Inisiasikan suatu nilai jarak untuk setiap simpul dengan nilai nol untuk simpul awal dan nilai tak hingga untuk setiap simpul sisanya.
2. Tandailah seluruh simpul sebagai belum dikunjungi dan tentukan simpul *a* sebagai simpul saat ini.
3. Untuk simpul saat ini, perhitungkan seluruh tetangga langsungnya yang belum dikunjungi dan kalkulasikan jarak *tentative* dari simpul *a*. Jika jarak yang didapatkan lebih kecil daripada jarak yang sudah dicatat sebelumnya, maka jarak yang minimum akan disimpan.
4. Jika kita sudah selesai dengan pengecekan terhadap semua tetangga terdekat dari simpul saat ini, simpul ditandai sebagai sudah dikunjungi.
5. Sebuah simpul yang ditandai sebagai sudah dikunjungi tidak akan pernah diperiksa ulang dan jarak yang tercatat adalah akhir dan minimal.
6. Jika seluruh simpul sudah selesai dikunjungi, maka selesai; selain daripada itu pilih simpul dengan jarak minimum dari simpul *a* untuk ditetapkan sebagai simpul saat ini dan ulangi langkah 3.



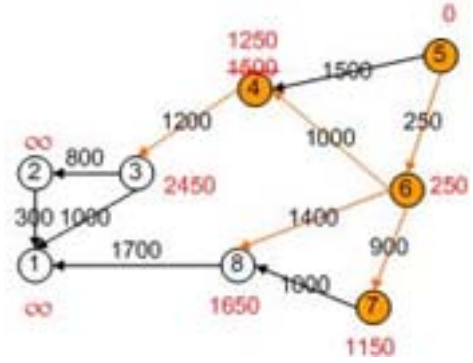
(a)



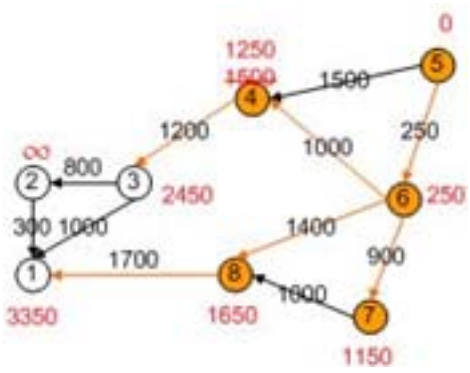
(b)



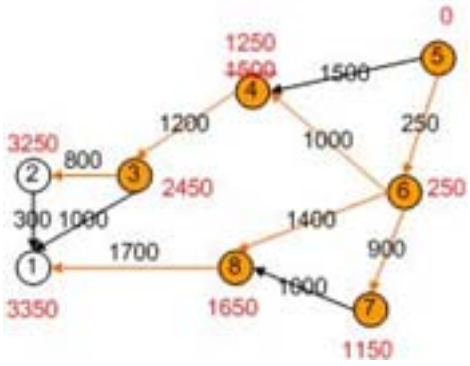
(c)



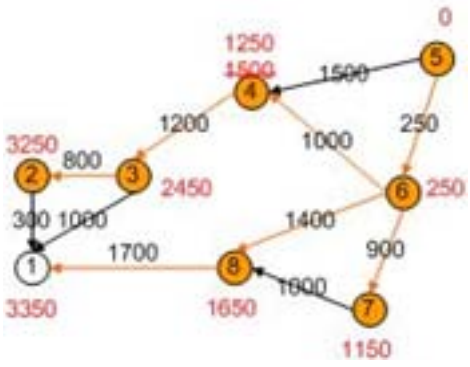
(d)



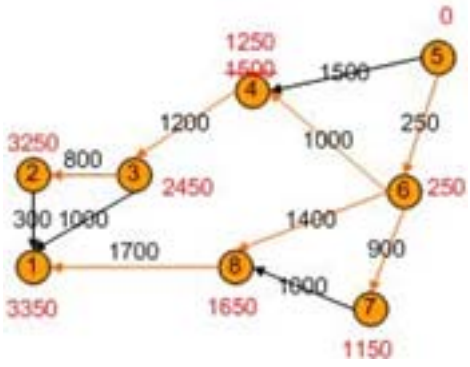
(e)



(f)



(g)



(h)

Gambar 6. Langkah – langkah Solusi pada Algoritma Dijkstra

```
Dijkstra Pseudo Algorithm

function Dijkstra(Graph, source):
      for each vertex v in Graph:
// Initializations
          dist[v] := infinity ;
// Unknown distance function from
source to v
          previous[v] := undefined ;
// Previous node in optimal path from
source
      end for ;
      dist[source] := 0 ;
// Distance from source to source
      Q := the set of all nodes in
Graph ;
      // All nodes in the graph are
unoptimized - thus are in Q
      while Q is not empty:
// The main loop
          u := vertex in Q with
smallest dist[] ;
          if dist[u] = infinity:
              break ;
// all remaining vertices are
inaccessible from source
          fi ;
          remove u from Q ;
          for each neighbor v of u:
// where v has not yet been removed
from Q.
              alt := dist[u] +
dist_between(u, v) ;
              if alt < dist[v]:
// Relax (u,v,a)
                  dist[v] := alt ;
                  previous[v] := u ;
              fi  ;
          end for ;
      end while ;
      return dist[] ;
  end Dijkstra.
```

Analisis

1. Algoritma *greedy* memandang bahwa untuk mencapai optimum global, dapat dilakukan dengan pendekatan yakni mengambil suatu nilai yang pada saat itu adalah merupakan nilai optimum lokal dengan harapan resultan akhirnya akan menghasilkan nilai yang optimum juga. Karena algoritma *greedy* hanya berbasiskan pada pengambilan keputusan setiap saat tanpa mempertimbangkan dampak ke depannya, algoritma *greedy* tidak selalu menjamin bahwa nilai yang dihasilkan adalah nilai optimum yang sesungguhnya, tetapi dapat dikatakan bahwa nilai yang dihasilkan merupakan suatu hampiran dari nilai optimum global.
2. Algoritma *greedy* hanya mempertimbangkan suatu solusi secara sesaat sehingga kebutuhan ruang dan waktunya sangat sedikit.

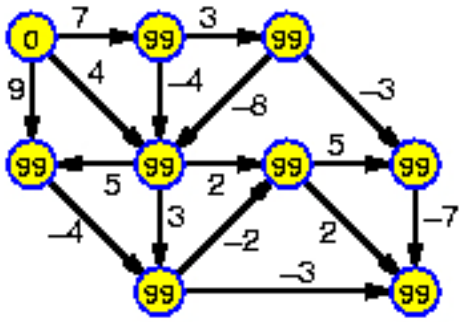
Untuk algoritma Dijkstra, kompleksitas waktu untuk suatu graf dengan sisi *E* dan simpul *V* dapat diekspresikan dalam bentuk suatu fungsi dari |*E*| dan |*V*| sebagai $O(|E|.dk_Q + |V|.em_Q)$ dimana dk_Q dan em_Q adalah

waktu yang dibutuhkan untuk melakukan pengurangan kunci dan mengekstraksi operasi minimum dari sebuah himpunan Q yang dapat disederhanakan sebagai $O(|V^2| + |E|) = O(|V^2|)$. Dengan sedikit perbaikan, algoritma ini dapat mencapai $O\big((|E| + |V|)\log|V|\big) = O(|E|\log|V|) \approx O(|E| + |V|\log|V|)$.

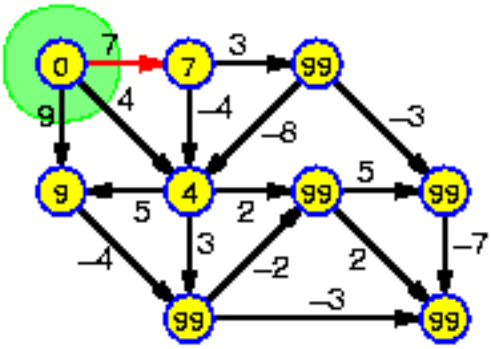
III. BELLMAN-FORD (VARIASI DARI BFS- DFS)

Struktur dasar dari Bellman-Ford sangatlah menyerupai yang ada pada algoritma Dijkstra, akan tetapi dibandingkan dengan memilih simpul dengan bobot minimum yang belum digunakan secara *greedy*, algoritma Bellman-Ford secara sederhana melakukan pengecekan terhadap semua sisi dan melakukannya sejumlah |*V*| - 1 kali, dimana |*V*| adalah banyaknya simpul yang terdapat di dalam graf. Pengulangan ini memungkinkan jarak terpendek untuk dihitung secara akurat bertahap di dalam graf dengan kemungkinan suatu simpul dikunjungi dalam lintasan tersebut adalah sebanyak-banyaknya satu kali saja. Adapun kompleksitas waktu untuk keberjalanan algoritma Bellman-Ford adalah $O(|V|.|E|)$, dimana |*V*| dan |*E*| adalah banyaknya simpul dan sisi berturutan.

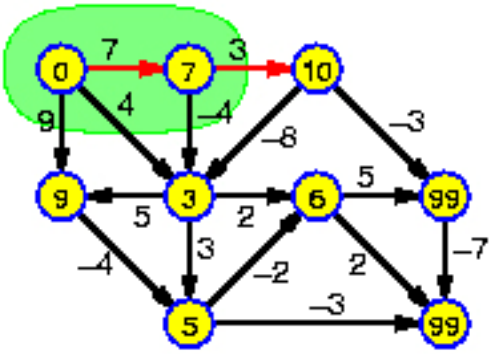
```
Bellman-Ford Pseudo Algorithm

procedure BellmanFord(list vertices,
list edges, vertex source)
   // This implementation takes in a
graph, represented as lists of
vertices
   // and edges, and modifies the
vertices so that their distance and
   // predecessor attributes store the
shortest paths.

   // Step 1: initialize graph
   for each vertex v in vertices:
       if v is source then v.distance
:= 0
       else v.distance := infinity
       v.predecessor := null

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge uv in edges: //
uv is the edge from u to v
           u := uv.source
           v := uv.destination
           if u.distance + uv.weight <
v.distance:
               v.distance :=
u.distance + uv.weight
               v.predecessor := u

   // Step 3: check for negative-
weight cycles
   for each edge uv in edges:
       u := uv.source
       v := uv.destination
       if u.distance + uv.weight <
v.distance:
           error "Graph contains a
negative-weight cycle"
```



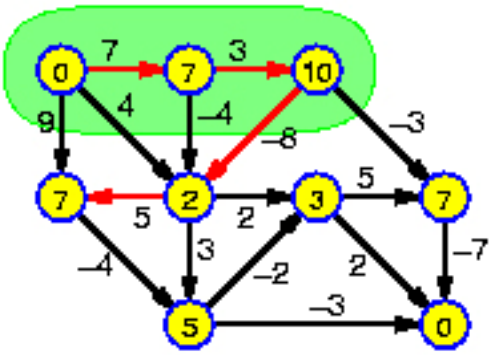
(a)



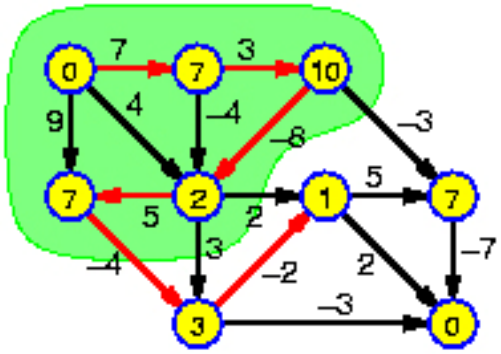
(b)



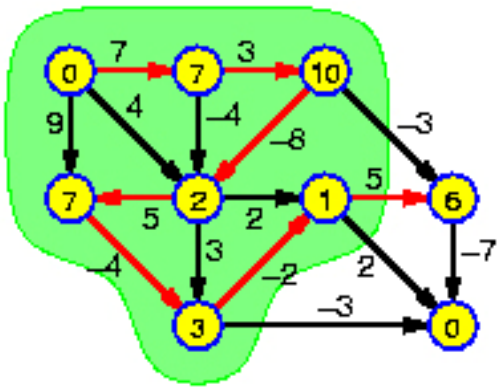
(c)



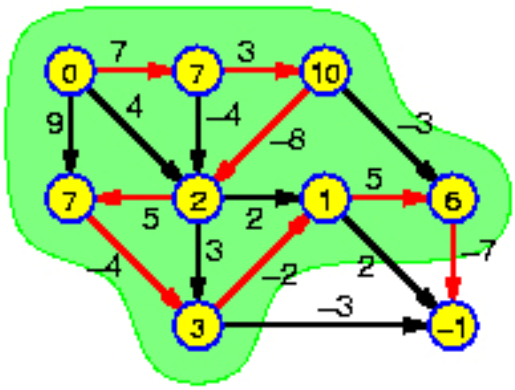
(d)



(e)



(f)



(g)

Gambar 7. Langkah – langkah Solusi pada Algoritma Bellman-Ford

Analisis

1. Algoritma BFS / DFS dapat menjamin ditemukannya solusi optimum pada pohon ruang status. (untuk algoritma Bellman-Ford pembuktian solusi selalu merupakan solusi optimum global dapat dilakukan dengan menggunkan induksi)
2. Algoritma BFS / DFS pada umumnya memerlukan waktu yang lebih lama daripada algoritma *greedy*.

Untuk algoritma Bellman-Ford, kompleksitas waktu untuk suatu graf dengan sisi *E* dan simpul *V* dapat diekspresikan dalam bentuk suatu fungsi dari |*E*| dan |*V*| sebagai $O(|V||E|)$ sementara kompleksitas ruangnya adalah $O(|V|)$. Sekalipun lebih cepat daripada algoritma lainnya yang setara, algoritma Bellman-Ford memiliki kelemahan *count-to-infinity* problem.

IV. Floyd-Warshal (*Dinamyc Programming*)

Algoritma Floyd-Warshall adalah sebuah algoritma analisis graf untuk mencari bobot minimum dari graf berarah. Dalam satu kali eksekusi algoritma, akan didapatkan jarak sebagai jumlah bobot dari lintasan terpendek antar setiap pasang simpul tanpa memperhitungkan informasi mengenai simpul-simpul yang dilaluinya. Algoritma yang juga dikenal dengan nama Roy-Floyd ini merupakan penerapan strategi *dynamic programming*.

Algoritma Floyd-Warshall membandingkan semua lintasan yang mungkin dalam graf untuk setiap pasang simpul dalam $\theta(|V^3|)$ jumlah perbandingan dengan fakta bahwa jumlah sisi yang mungkin ada dalam graf dapat mencapai $\Omega(|V^2|)$ dan setiap kombinasinya akan diuji. Cara ini dilakukan dengan secara bertahap melakukan penaikan satu terhadap peningkatan estimasi biaya antara dua buah simpul hingga estimasi mencapai nilai yang optimum.

Diberikan sebuah graf *G* dengan simpul *V*, yang dinomori mulai dari 1 hingga *N*, tinjau fungsi lintasanTerpendek(*i*, *j*, *k*) yang mengembalikan lintasan terpendek yang mungkin dilalui dari *i* ke *j* dengan hanya menggunakan simpul dari himpunan {1, 2, ..., k} sebagai simpul antara. Dengan bermodalkan fungsi ini, akan dicari lintasan terpendek untuk setiap pasang *i* dan *j* dengan menggunakan hanya simpul antara 1 hingga k + 1.

Terdapat dua kandidat untuk setiap lintasan yang ada : baik lintasan terpendek sesungguhnya hanya menggunakan simpul pada himpunan {1, 2, ..., *k*}; maupun adanya suatu lintasan dari *i* ke *k* + 1, lalu dari *k* + 1 menuju *j* yang kemudian memberikan hasil yang lebih baik. Dengan demikian, maka dapat didefinisikan :

$$lintasanTerpendek(i,j,k) =$$
$$\min\{lintasanTerpendek(i,j,(k-1)),$$
$$lintasanTerpendek(i,k,(k-1)) +$$
$$lintasanTerpendek(k,j,(k-1))\},$$
$$lintasanTerpendek(i,j,0) = biayaSisi(i,j).$$

Algoritma dimulai dengan cara pertama kali menghitung lintasanTerpendek(*i*, *j*, *k*) untuk seluruh pasangan (*i*, *j*) dengan *k* meningkat dimulai dari 1 hingga *n*.

```
Floyd-Warshall Pseudo Algorithm

/* Assume a function edgeCost(i,j)
which returns the cost of the edge from
i to j
    (infinity if there is none).
    Also assume that n is the number of
vertices and edgeCost(i,i) = 0
 */
```

```
int path[][];
/* A 2-dimensional matrix. At each
step in the algorithm, path[i][j] is
the shortest path
    from i to j using intermediate
vertices (1..k-1).  Each path[i][j] is
initialized to
    edgeCost(i,j) or infinity if there
is no edge between i and j.
 */

 procedure FloydWarshall ()
    for k := 1 to n
       for i := 1 to n
          for j := 1 to n
             path[i][j] = min (
path[i][j], path[i][k]+path[k][j] );
```

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

(a)

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

(b)

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

(c)

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

(d)

Gambar 8. Langkah – langkah Solusi pada Algoritma Floyd-Warshall

Analisis

1. Algoritma *dynamic programming* dapat menjamin ditemukannya solusi optimum pada pohon ruang status.
2. Algoritma *dynamic programming* pada umumnya memerlukan ruang yang lebih besar daripada algoritma BFS / DFS tetapi membutuhkan waktu komputasi rata-rata yang lebih cepat.

Untuk algoritma Floyd-Warshall, kompleksitas waktu untuk suatu graf dengan sisi E dan simpul V dapat diekspresikan dalam bentuk suatu fungsi dari $|E|$ dan $|V|$ sebagai $O(V^3)$ sementara kompleksitas ruangnya adalah $\theta(|V|^2)$.

V. Analisis dan Kesimpulan

Meskipun algoritma *greedy* pada umumnya tidak selalu memberikan hasil yang optimum, tetapi pada *routing protocol* yang mana hal ini tidak memiliki tingkat urgensi tinggi sehingga hampiran nilai optimum dapat dikatakan cukup, maka algoritma Dijkstra dapat menjadi pilihan favorit mengingat kebutuhan waktu dan ruangnya sangatlah kecil. Selain daripada itu, sekalipun didapatkan lintasan terpendek yang optimum, jika membutuhkan waktu komputasi lama, bukanlah tidak mungkin jika pada

akhirnya total waktu yang dibutuhkan untuk komputasi dan *transfer time* melalui lintasan tersebut menjadi lebih lama dibandingkan dengan algoritma lain yang membutuhkan waktu komputasi yang jauh lebih ringkas sekalipun waktu *transfer time* tidak optimum dan hanya merupakan hampiran.

REFERENCES

http://amicta.web.id/pic/floyd-warshall25.jpg (Rabu, 08-Desember-2010 19:13)

http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm (Rabu, 08-Desember-2010 19:13)

http://en.wikipedia.org/wiki/Dijkstra's_algorithm (Rabu, 08-Desember-2010 19:13)

http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm (Rabu, 08-Desember-2010 19:13)

http://en.wikipedia.org/wiki/Routing_protocol (Sabtu, 04-Desember-2010 17:31)

http://en.wikipedia.org/wiki/Routing_Information_Protocol (Sabtu, 04-Desember-2010 17:31)

http://student.eepis-its.edu/~izankboy/laporan/Jaringan/ccna2-6.pdf (Sabtu, 04-Desember-2010 17:31)

http://www8.cs.umu.se/~jopsi/dinf504/bellman_ford.gif (Rabu, 08-Desember-2010 19:13)

http://www.egr.unlv.edu/~jjtse/CS477/Dijkstra%20SP.jpg (Rabu, 08-Desember-2010 19:13)

Tanenbaum, Andrew S., "Computer Networks," 4th ed., Pearson Education, Inc. New Jersey: Prentice Hall PTR, 2003, pp. 259–291.

Munir, Rinaldi, "Strategi Algoritma," Diktat Kuliah IF3051, 2nd ed., Institut Teknologi Bandung. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika, 2009, pp. 58–63.

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, 09 Desember 2010

Michell Setyawati Handaka / 135 08 045