Pencarian Jalur Terpendek dalam GPS dengan Menggunakan Teori graf
Using Graph Theory for Finding Shortest Path in GPS

Mohamad Ray Rizaldy
13505073

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha no.10, Bandung

e-mail : if15073@students.informatika.org

ABSTRAK

Teknologi transportasi dari zaman ke zaman terus menunjukan kemajuan yang pesat. Salah satu hasil dari pesatnya kemajuan teknologi transportasi ini adalah terciptanya *Global Positioning System* atau yang dikenal sebagai GPS. Fungsi utama dari GPS adalah untuk mendapatkan informasi mengenai posisi sebuah objek yang berada di permukaan bumi. Pada awalnya GPS hanya diciptakan untuk kepentingan militer, tetapi masyarakat punya kebutuhan akan teknologi dan transportasi yang harus dipenuhi. Saat ini, aplikasi GPS dengan dibantu teknologi Automotive Navigation System dapat digunakan untuk membantu manusia dalam menentukan jalur tempuh yang terpendek. Dengan mengetahui informasi jalur tempuh ini, manusia tentunya dapat mengefisiensikan waktu mereka karena bisa lebih cepat sampai di tujuan. Terdapat beberapa algoritma yang dapat digunakan dalam penentuan jalur terpendek sebagian besar dari algoritma ini menggunakan teori graf untuk menyelesaikannya.

Kata Kunci: *GPS, Automotive Navigation System, Algoritma Shortest Path, TeoriGraf*

1. Pendahuluan

Selama bertahun-tahun, para navigator dan penjelajah bertanya-tanya apakah ada cara yang dapat membuat mereka bisa mengetahui posisi mereka di muka bumi secara akurat. Hal ini sangat penting, karena dengan mengetahuinya, mereka bisa menghindari bencana juga bisa mempercepat waktu tempuh ke daerah yang dituju.

Pada 26 Juni 1993, pertanyaan tersebut pun terjawab dengan diorbitkannya satelit ke-24 oleh United States Department of Defense (DoD). Satelit itu bekerja bersama 23 satelit sejenis yang lebih dahulu mengorbit. Selanjutnya ke 24 satelit tersebut dinamakan NAVSTAR GPS yang merupakan kepanjangan dari Navigational Satellite Timing and Ranging Global Positioning System.

Hingga tahun 1995, teknologi tersebut hanya bisa digunakan oleh pihak militer. Tetapi seiring berjalannya waktu, pada tanggal 17 Juli 1995 teknologi yang akhirnya dikenal dengan sebutan GPS ini diaktifkan untuk kepentingan umum. GPS kemudian dapat digunakan dalam hal transportasi, SIG (Sistem Informasi Geografis), pelacakan kendaraan,dan pemantau gempa.

Dalam makalah ini yang akan dibahas adalah tentang transportasi. GPS dalam bidang

ini berguna untuk menggantikan fungsi routing jalan pada penggunaan peta biasa. GPS akan memudahkan penggunanya dalam menentukan jalur terbaik ke sebuah tujuan, karena pengguna tidak lagi perlu repot-repot harus melakkuan routing manual pada peta yang membosankan.

2. Teori Graf

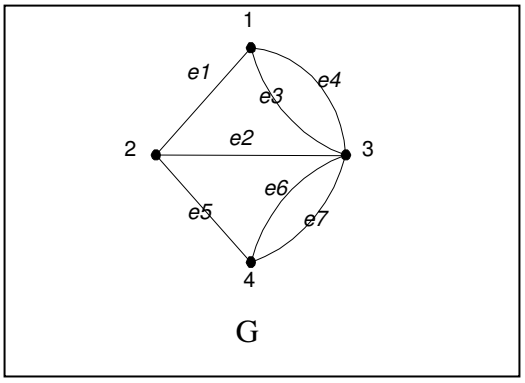
2.1 Definisi Graf

Graph merupakan struktur data yang paling umum. Jika struktur linear memungkinkan pendefinisian keterhubungan sikuensial antara entitas data, struktur data tree memungkinkan pendefinisian keterhubungan hirarkis, maka struktur graph memungkinkan pendefinisian keterhubungan tak terbatas antara entitas data.

Definisi Graf adalah G = (V,E) yang dalam hal ini :

V = himpunan tak kosong dari simpul-simpul
= {v_1, v_2, v_3, ...}

E = himpunan sisi yang menghubungkan sepasang simpul
= {e_1, e_2, e_3, ...}



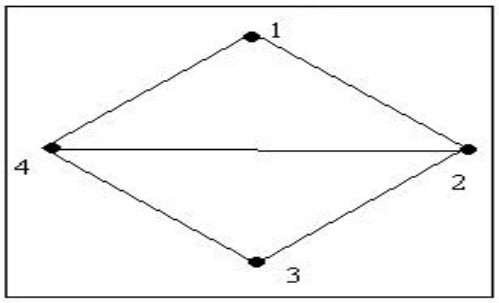
Gbr. 1

G adalah graf dengan
V = { 1, 2, 3, 4 }
E = { (1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4) }
= { e_1, e_2, e_3, e_4, e_5, e_6, e_7}

Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis:

1. Graf tak-berarah (*undirected graph*)

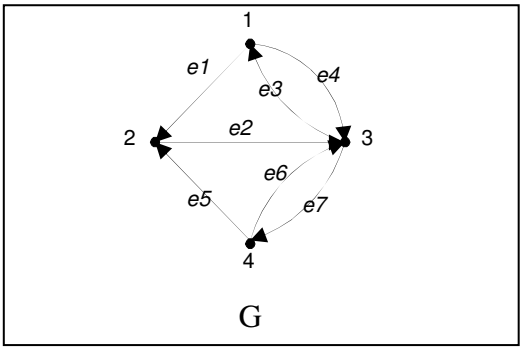
Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah



Gbr. 2

2. Graf berarah (*directed graph* atau *digraph*)

Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah.

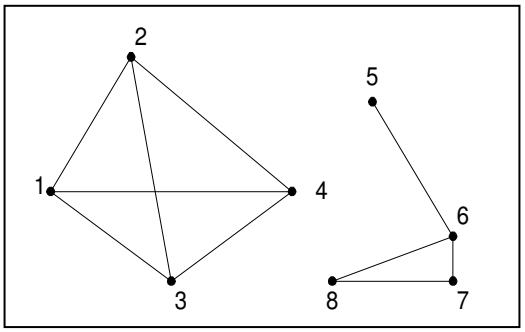


Gbr. 3

Dua buah simpul v_1 dan simpul v_2 disebut **terhubung** jika terdapat lintasan dari v_1 ke v_2.

G disebut **graf terhubung** (*connected graph*) jika untuk setiap pasang simpul v_i dan v_j dalam himpunan *V* terdapat lintasan dari v_i ke v_j.

Jika tidak, maka *G* disebut **graf tak-terhubung** (*disconnected graph*).



Gbr. 4

Graf berarah G dikatakan terhubung jika graf tidak berarahnya terhubung (graf tidak berarah dari G diperoleh dengan menghilangkan arahnya).

2.2 Terminologi Graf

1. Ketetanggaan (adjacent)
Dua buah simpul dikatakan bertetangga bila keduanya terhubung langsung.

2. Bersisian (incidency)
Untuk sembarang sisi $e = (v_i, v_j)$ dikatakan bahwa e bersisian dengan simpul (vertex) v_i dan v_j.

3. Derajat (degree)
Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Notasinya adalah d(v)

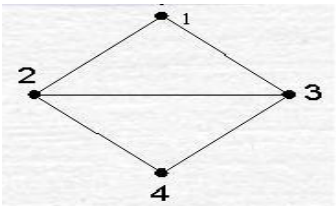
4. Lintasan (path)
Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam sebuah graf adalah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1, e_2, v_2, ...dst.$

5. Sirkuit (Circuit)
Lintasan yang berawal dan berakhir di tempat (simpul) yang sama disebut sirkuit

6. Graf berbobot (weighted graph)
Apabila sisi-sisi pada graph disertai juga dengan suatu (atau beberapa) harga yang menyatakan secara unik kondisi keterhubungan tersebut maka graph tersebut disebut graph berbobot.

2.3 Representasi Graf

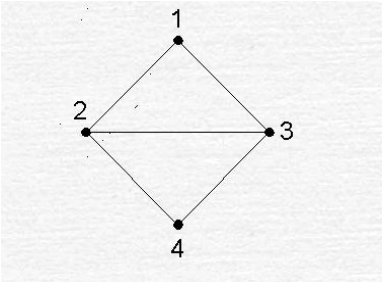
1. Adjacency Matrix
Suatu matriks digunakan untuk menyatakan adjacency set setiap verteks dalam baris-barisnya. Nomor baris menyatakan nomor verteks adjacency berasal dan nomor kolom menunjukkan nomor verteks kemana arah adjacency. Elemen matriks [x, y] berharga 1 bila terdapat sisi dari x ke y dan berharga 0 bila tidak ada.



	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

Gbr. 5

2. Adjacency List
Mengingat bahwa rasio degree (atau outdegree pada digraph) rata-rata verteks terhadap jumlah verteks dalam sejumlah masalah adalah bisa sangat kecil maka representasi matriks tersebut akan berupa matriks sparse yaitu sebagian besarnya berisikan bilangan nol (atau bilangan ¥ pada weighted graph). Untuk kepentingan efisiensi ruang maka tiap baris matriks tersebut digantikan list yang hanya berisikan verteks-verteks dalam adjacency set Vx dari setiap verteks x.



Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

Gbr. 6

2.4 Lintasan dan Sirkuit Euler

Lintasan Euler adalah lintasan yang melalui masing-masing sisi di dalam graf secara tepat satu kali.

Sirkuit Euler ialah sirkuit yang melewati masing –masing sisi tepat satu kali.

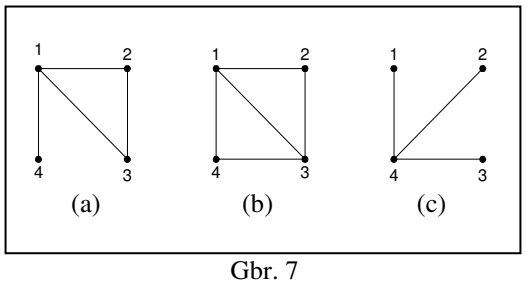
Graf yang memiliki sirkuit Euler disebut dengan graf Euler. Graf yang memiliki lintasan euler dinamakan juga graf semi euler

2.5 Graf Hamilton

Lintasan Hamilton ialah lintasan yang melalui tiap simpul di dalam graf tepat satu kali.

Sirkuit Hamilton ialah sirkuit yang melalui tiap simpul di dalam graf tepat satu kali, kecuali simpul asal (sekaligus simpul akhir) yang dilalui dua kali.

Graf yang memiliki sirkuit Hamilton dinamakan graf Hamilton, sedangkan graf yang hanya memiliki lintasan Hamilton disebut graf semi-Hamilton.



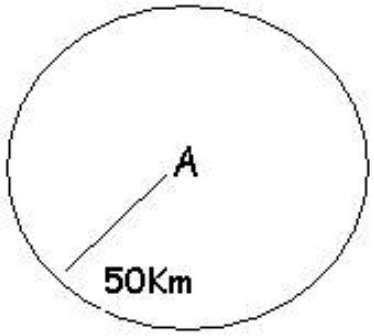
Gbr. 7

(a) graf yang memiliki lintasan Hamilton (misal: 3, 2, 1, 4)
(b) graf yang memiliki lintasan Hamilton (1, 2, 3, 4, 1)
(c) graf yang tidak memiliki lintasan maupun sirkuit Hamilton

3. Cara Kerja GPS

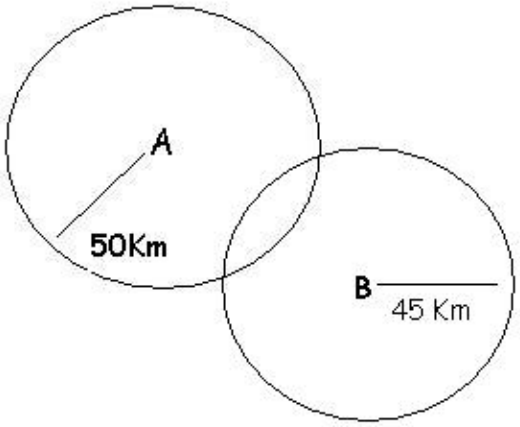
3.1 Trilaterasi 2-Dimensi

Bayangkan suatu saat anda sedang dalam keadaan tersesat dan sama sekali tidak mengetahui dimana anda berada sekarang. Kemudian anda menemukan sebuah patok (pembatas wilayah) bertuliskan: “jarak ke kota A adalah 50 Km”. Mungkin menurut anda, ini akan sangat berguna. Tapi tidak, pendapat anda tidak sepenuhnya benar. Karena pada kenyataannya ada berjuta titik yang memiliki jarak 50 Km dari kota A.



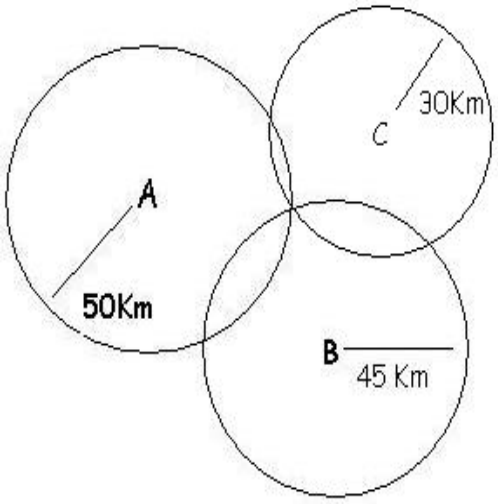
Gbr. 8

Anda kemudian menemukan sebuah patok lagi. Yang kali ini bertuliskan “Jarak anda ke kota B adalah 45 Km”. Dengan kedua informasi ini anda bisa menyimpulkan anda kemungkinan berada di salah satu titik yang perpotongan lingkar radius berjarak 50 Km dari A dan 45 Km dari B.



Gbr. 9

Seandainya anda kemudian menemukan sebuah patok lagi bertuliskan "Jarak ke kota C adalah 30 Km". Maka anda akan dapat mengeliminasikan satu titik dan anda bisa mendapatkan posisi anda berkat 3 informasi tersebut.



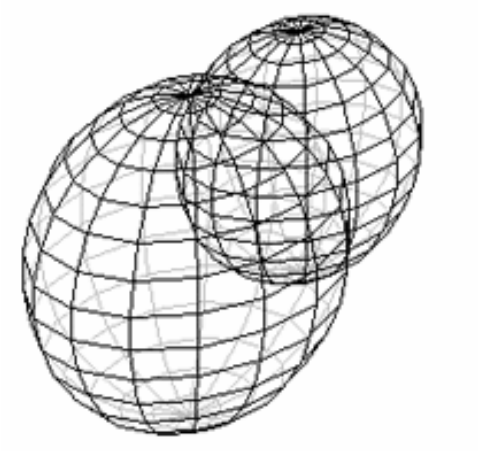
Gbr. 10

3.2 Triliterasi 3-Dimensi

Pada dasarnya triliterasi pada dimensi 3 tidak terlalu berbeda dengan triliterasi dimensi 2. Masalahnya adalah pada penggambaran objeknya.

Misalnya anda mengetahui bahwa anda sekarang berada 10 mil dari satelit A di luar angkasa. Tentunya anda tahu bahwa ada jutaan titik yang membentuk bola berjari jari 10 mil yang berpusat di A.

Jika anda juga mengetahui anda berada pada jarak 15 mil dari satelit B. Bola dengan pusat di A tadi bisa anda gabungkan dengan bola lain berjari-jari 15 mil yang berpusat di B. Hasil gabungan kedua bola tersebut akan membentuk sebuah lingkaran perpotongan bola-bola itu.



Gbr. 11

Seandainya anda juga mengetahui jarak anda ke satelit C, anda akan bisa membayangkan satu bola lagi tergabung dalam gabungan tadi. Anda akan menemukan perpotongan ketiga bola tersebut berada pada 2 titik.

Anda akan memerlukan sebuah bola lagi untuk mengeliminasikan kedua titik yang ditemukan. Dalam GPS yang berfungsi sebagai bola ke-4 adalah bumi. Mengapa? Karena kita (pengguna yang sedang menentukan posisi) tentunya akan berada di permukaan bumi. Dengan menjadikan bumi sebagai bola ke-4 anda, anda akan menemukan posisi pasti anda di permukaan bumi secara pasti.

3.3 Pencarian Posisi pada GPS

Sebuah koordinat didapatkan berdasarkan sistem *World Geodetic System WGS84 coordinates*. Untuk mencari posisinya, sebuah receiver membutuhkan waktu yang presisi atau tepat. Oleh karena itu tiap satelit dilengkapi oleh sebuah jam atom yang super akurat, sedangkan receivernya menggunakan jam *crystal oscillator*

internal yang secara kontinu selalu terbaharui oleh sinyal yang datang dari satelit.

Receiver mengidentifikasi setiap sinyal satelit dengan mengenali *C/A (Course Acqusition) code pattern*, dan menghitung waktu tunda (*delay time*) setiap satelit.

Data tentang posisi orbital digunakan untuk mengkalkulasikan posisi presisi dari satelit. Informasi tentang posisi dan jarak tiap satelit mengindikasikan bahwa receiver terletak pada sebuah permukaan dari bola khayalan yang berpusat pada sebuah setelit.

Ketika 4 satelit diukur secara bersamaan, perpotongan 4 bola khayal yang terbentuk akan menjadi sebuah titik yang mewakilkan posisi dari receiver.

4. Automotive Navigation System

Untuk keperluan navigasi, sebuah *automobile* dapat di pasang pada receiver GPS. Unit ini akan menampilkan peta bergerak dan informasi tentang lokasi, kecepatan dan arahnya.

Oleh karena itu, untuk penggunaan automobile ini kemudian diperkenalkanlah sebuah sistem yang bernama *Automotive Navigation System*. Dalam sistem ini, GPS digunakan pada pengiriman posisi berdasarkan pantauan satelit untuk diintegrasikan dengan database jalan yang dimiliki oleh receiver. Hasil penyatuan ini kemudian ditampilkan dalam layar.

Sayangnya terkadang sistem navigasi ini terhalang oleh cuaca dan keadaan geografis, sehingga hasil yang diperoleh jika pengguna berada di sebuah terowongan dan jika pengguna berada di sebuah padang rumput akan berbeda.

Database jalan raya yang digunakan harus dibuat secara detil persis dengan keadaan sebenarnya di lapangan. Database ini juga harus dilengkapi dengan keteranga-keterangan secara geografis, mulai dari garis lintang, garis bujur hingga altitud atau ketinggian tiap-tiap titik jalan.

Dalam database dapat juga dimasukkan informasi-informasi tambahan yang nantinya akan bermanfaat bagi pengguna kendaraan, misalnya SPBU (pom bensin) atau tempat ibadah, dan mungkin juga hotel.

5. Pencarian Shortest Path

Pencarian shortest path (lintasan terpendek) dalam adalah termasuk masalah yang paling umum dalam suatu weighted, connected graph. Lebih lanjut lagi dalam kelas masalah ini terdapat subkelas-subkelas masalah yang lebih spesifik. Misalnya pada jaringan jalan raya yang menghubungkan kota-kota disuatu wilayah, hendak dicari :

- lintasan terpendek yag menghubungkan antara dua kota berlainan tertentu (Single-source Single-destination Shortest Path Problems)
- semua lintasan terpendek masing-masing dari suatu kota ke setiap kota lainnya (Single-source Shortest Path problems)
- semua lintasan terpendek masing-masing antara tiap kemungkinan pasang kota yang berbeda (All-pairs Shortest Path Problems)

Untuk memecahkan masing-masing dari masalah-masalah tersebut terdapat sejumlah solusi yaitu:

- Algoritma Dijkstra untuk Single-source Shortest Path
- Algoritma Floyd-Warshall untuk masalah All-pairs Shortest Path
- Algoritma Johnson untuk masalah All-pairs Shortest Path pada sparse graph

Dalam makalah ini hanya akan dibahas mengenai algoritma Djikstra dan algoritma Floyd-Warshall.

5.1 Algoritma Djikstra

Algoritma Dijkstra, dinamai menurut penemunya, Edsger Dijkstra, adalah greedy algorithm yang memecahkan shortest path problem untuk sebuah directed graph dengan edge weights yang tidak negatif.

Misalnya, bila vertices dari sebuah graph melambangkan kota-kota dan edge weights melambangkan jarak antara kota-kota tersebut, algoritma Dijkstra dapat digunakan untuk menemukan jarak terpendek antara dua kota.

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 → [0, ∞); 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.

Algoritma ini mirip dengan algoritma Prim untuk mencari MST, yaitu pada tiap iterasi memeriksa sisi-sisi yang menghubungkan subset verteks W dan subset verteks (V-W) dan memindahkan verteks w dari (V-W) ke W yang memenuhi kriteria tertentu. Perbedaannya terletak pada kriteria itu sendiri.

- Jika yang dicari algoritma Kruskal adalah sisi dengan bobot terkecil dari sisi-sisi di atas dalam setiap iterasinya,
- dalam algoritma Dijkstra, yang dicari adalah sisi yang menghubungkan ke suatu verteks di (V-W) sehingga jarak dari verteks asal Vs ke verteks tersebut adalah minimal.

Dalam implementasinya penghitungan jarak dari verteks asal vs disederhanakan dengan menambahkan field minpath pada setiap verteks.

Field-field minpath ini diinisialisasi sesuai dengan adjacencynya dengan vs, kemudian dalam setiap iterasi di-update bersamaan masuknya w dalam W. Field minpath ini menunjukkan jarak dari vs ke verteks yang bersangkutan terpendek yang diketahui hingga saat itu.

Jadi pada verteks dalam W, minpath sudah menunjukkan jarak terpendek dari Vs untuk mencapai verteks yang bersangkutan, sementara pada (V-W) masih perlu diupdate pada setiap iterasi dalam mendapatkan verteks w.

Setiap mendapatkan w maka update minpath setiap adjacent verteks x dari w di (V-W) sisa dengan minimum (minpath dari x , total minpath w + panjang sisi yang bersangkutan), agar dapat berlaku umum maka di awal algoritma seluruh minpath diinisialisasi dengan +tak hingga.

Selengkapnya algoritma Dijkstra adalah sebagai berikut:

- Inisialisasi
  - W berisi mula-mula hanya vs
  - field minpath tiap verteks v dengan
  - Weight[vs, v], jika ada sisi tersebut, atau,
  - +tak hingga , jika tidak ada.
- Lalu, dalam iterasi lakukan hingga (V-W) tak tersisa (atau dalam versi lain: jika ve ditemukan):
  - dari field minpath tiap verteks cari verteks w dalam (V-W) yang memiliki minpath terkecil yang bukan tak hingga.
  - jika ada w maka
    - masukkan w dalam W
    - update minpath pada tiap verteks t adjacent dari w dan berada dalam (V-W) dengan: minimum (minpath[t] , minpath[w] + Weight[w, t])

Verteks-verteks dalam W dapat dibedakan dari verteks dalam (V-W) dengan suatu field yang berfungsi sebagai flag atau dengan struktur liked-list. Informasi lintasan dapat diketahui dengan pencatatan predesesor dari setiap verteks yang dilakukan pada saat update harga minpath tersebut (fungsi minimum): jika minpath[t] diupdate dengan (minpath[w]+Weight[w, t]) maka predesesor dari t adalah w. pada tahap inisialisasi predesesor setiap verteks diisi oleh vs.

5.2 Algoritma Floyd – Marshall

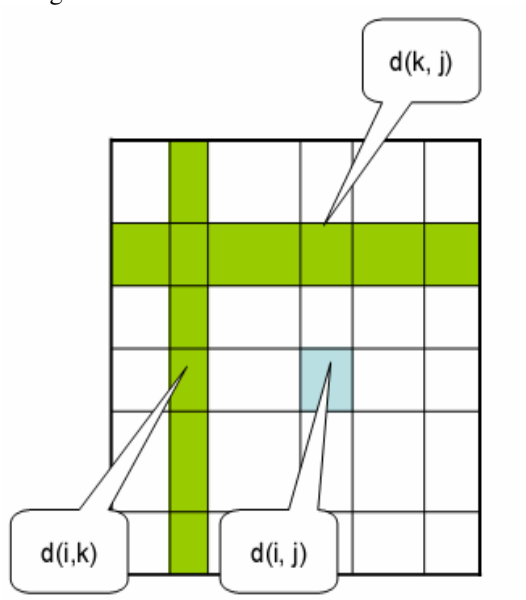
Algoritma Floyd-Warshall memiliki input graf berarah dan berbobot (V,E), yang berupa daftar titik (node/vertex V) dan daftar sisi (edge E). Jumlah bobot sisi-sisi pada sebuah jalur adalah bobot jalur tersebut. Sisi pada E diperbolehkan memiliki bobot negatif, akan tetapi tidak diperbolehkan bagi graf ini untuk memiliki siklus dengan bobot negatif. Algoritma ini menghitung bobot terkecil dari semua jalur yang menghubungkan sebuah pasangan titik, dan melakukannya sekaligus untuk semua pasangan titik. Algoritma ini berjalan dengan waktu $\Theta(|V|3)$.

Algoritma dilakukan berdasarkan formulasi *dyanmic programming*. Setiap langkahnya akan memeriksa path antara vi dan vj apakah bisa lebih pendek jika melalui vi-vk dan vk-vj.

Ide Formulasi Rekrusif (DP) adalah sebagai berikut :

- vertex-vertex antara dalam short path
- jika V={1,2,3,...,N},untuk k=0,...,n maka $d_{ij}^{(k)}$ =
  - w_{ij} jika k=0
  - min ($d_{ij}^{(k-1)}$, $d_{ik}^{(k-1)} + d_{jk}^{(k-1)}$) untuk k>0
- solusi dari $d_{ij}^{(n)}$ merupakan matriks shortest pathdari vertex i ke vertex j.

Formulasi tersebut dapat divisualisasikan sebagai berikut.



Gbr. 12

pada iterasi ke k, elemen d(i,j) dibandingkan dengan d(i,k)+d(k,j)

5.3 Pseudocode Algoritma

jika algoritma Djikstra dan Floyd-Marshall dituliskan dalam bentuk pseudocode, kita akan melihat hasil sebagai besrikut:

```
//Algoritma DIJKSTRA
function Dijkstra(G, t, s)
  for each vertex v in V[G]                          // Initializations
    d[v] := infinity
    previous[v] := undefined
  d[s] := 0                                          // Distance from s to s
  S := empty set
  Q := V[G]                                          // Set of all vertices
  while Q is not an empty set                        // The algorithm itself
    u := Extract_Min(Q)
    S := S union {u}
    for each edge (u,v) outgoing from u
      if d[u] + w(u,v) < d[v]                  // Relax (u,v)
        d[v] := d[u] + w(u,v)
        previous[v] := u
```

```
//Algoritma Floyd Warshall
function fw(int[1..n,1..n] graph) {
    // Initialization
    var int[1..n,1..n] dist := graph
    var int[1..n,1..n] pred
    for i from 1 to n
        for j from 1 to n
            pred[i,j] := nil
            if (dist[i,j] > 0) and (dist[i,j] < Infinity)
                pred[i,j] := i
    // Main loop of the algorithm
    for k from 1 to n
        for i from 1 to n
            for j from 1 to n
                if dist[i,j] > dist[i,k] + dist[k,j]
                    dist[i,j] = dist[i,k] + dist[k,j]
                    pred[i,j] = pred[k,j]
    return ( dist, pred ) // Tuple of the distance and predecessor
matrices
}
```

6. Daftar Pustaka

1. Bovy, P. and E. Stern.1990. "*Route Choice: Wayfinding in Transportation Networks"*. Kluwer Academic Publishers.
2. Munir, Rinaldi. 2005. "*Diktat Kuliah IF2153 Matematika Diskrit*". Bandung : Sekolah Teknik Elektro dan Informatika ITB. ITB
3. http://electronics.howstuffworks.com/gps.htm/printable
   diakses 3 Januari 2007 pukul 13:13
4. http://www.kowoma.de/en/gps/history.htm
   diakses 3 Januari 2007 pukul 14:14
5. http://id.wikipedia.org/wiki/GPS
   diakses 3 Januari 2007 pukul 14:14
6. http://en.wikipedia.org/wiki/Automotive_navigation_system
   diakses 3 Januari 2007 pukul 19:39
7. http://id.wikipedia.org/wiki/Algoritma_Dijkstra
   diakses 3 Januari 2007 pukul 19:41
8. http://id.wikipedia.org/wiki/Algoritma_Floyd-Warshall
   diakses 3 Januari 2007 pukul 19:42
9. http://ranau.cs.ui.ac.id/sda/archive/1998/handout/handout19.html
   diakses 3 Januari 2007 pukul 19:43
10. http://ranau.cs.ui.ac.id/sda/archive/1998/handout/handout20.html
    diakses 3 Januari 2007 pukul 19:43
11. http://ranau.cs.ui.ac.id/sda/archive/2006/Shortest%20Path.pdf
    diakses 3 Januari 2007 pukul 21:38