BAB II

TINJAUAN PUSTAKA

2.1 DKI Jakarta

Menurut Perda No 1 Tahun 2009 tentang Rencana Pembangunan Jangka Menengah Daerah Tahun 2007-2012. Provinsi DKI Jakarta terbagi menjadi 5 wilayah Kota administrasi dan satu Kabupaten administratif, yakni: Kota administrasi Jakarta Pusat dengan luas 47,90 km2, Jakarta Utara dengan luas 142,20 km2, Jakarta Barat dengan luas 126,15 km2, Jakarta Selatan dengan luas 145,73 km2, dan Kota administrasi Jakarta Timur dengan luas 187,73 km2, serta Kabupaten Administratif Kepulauan Seribu dengan luas 11,81 km2. Di sebelah utara membentang pantai sepanjang 35 km, yang menjadi tempat bermuaranya 13 buah sungai dan 2 buah kanal. Di sebelah selatan dan timur berbatasan dengan Kota Depok, Kabupaten Bogor, Kota Bekasi dan Kabupaten Bekasi, sebelah barat dengan Kota Tangerang dan Kabupaten Tangerang, serta di sebelah utara dengan laut jawa.

Secara geologis, seluruh dataran terdiri dari endapan *pleistocene* yang terdapat pada ±50 m di bawah permukaan tanah. Bagian selatan terdiri atas lapisan *alluvial*, sedang dataran rendah pantai merentang ke bagian pedalaman sekitar 10 km. Di bawahnya terdapat lapisan endapan yang lebih tua yang tidak tampak pada permukaan tanah karena tertimbun seluruhnya oleh endapan *alluvium*. Di wilayah bagian utara baru terdapat pada kedalaman 10-25 m, makin ke selatan permukaan keras semakin dangkal 8-15 m. Pada bagian tertentu juga terdapat lapisan permukaan tanah yang keras dengan kedalaman 40m.

Keadaan Kota Jakarta umumnya beriklim panas dengan suhu udara maksimum berkisar 32,7°C - 34,°C pada siang hari, dan suhu udara minimum berkisar 23,8°C -25,4°C pada malam hari. Rata-rata curah hujan sepanjang tahun 237,96 mm, selama periode 2002-2006 curah hujan terendah sebesar 122,0 mm terjadi pada tahun 2002 dan tertinggi sebesar 267,4 mm terjadi pada tahun 2005, dengan tingkat kelembaban udara mencapai 73,0 - 78,0 persen dan kecepatan angin rata-rata mencapai 2,2 m/detik - 2,5 m/detik.

2.2 Angkutan Umum

Angkutan pada dasarnya adalah sarana untuk memindahkan orang dan atau barang dari satu tempat ke tempat lain. Tujuannya membantu orang atau kelompok orang menjangkau berbagai tempat yang dikehendaki atau mengirimkan barang dari tempat asalnya ke tempat tujuannya. Prosesnya dapat dilakukan dengan menggunakan sarana angkutan berupa kendaraan. Sementara Angkutan Umum Penumpang adalah angkutan penumpang yang menggunakan kendaraan umum yang dilakukan dengan sistem sewa atau bayar. Termasuk dalam pengertian angkutan umum penumpang adalah angkutan kota (bus, minibus, dsb), kereta api, angkutan air, dan angkutan udara. (Warpani, 1990) .

Angkutan Umum Penumpang bersifat massal sehingga biaya angkut dapat dibebankan kepada lebih banyak orang atau penumpang yang menyebabkan biaya per penumpang dapat ditekan serendah mungkin. Karena merupakan angkutan massal, perlu ada kesamaan diantara para penumpang, antara lain kesamaan asal dan tujuan. Kesamaan ini dicapai dengan cara pengumpulan di terminal dan atau tempat perhentian. Kesamaan tujuan tidak selalu berarti kesamaan maksud.

Angkutan umum massal atau masstransit memiliki trayek dan jadwal keberangkatan yang tetap. Pelayanan angkutan umum penumpang akan berjalan dengan baik apabila tercipta keseimbangan antara ketersediaan dan permintaan. Oleh karena itu, Pemerintah perlu turut campur tangan dalam hal ini. (Warpani, 1990). Jakarta memiliki beberapa angkutan umum antara lain kopaja, metro mini, mikrolet dan lain-lain.

2.3 Rute

Menurut kamus besar bahasa Indonesia rute adalah jarak atau arah yang harus diturut (ditempuh, dilalui). Setiap manusia selalu melakukan perjalanan dan berusaha mencari rute terbaik masing-masing yang akan meminimalkan biaya (jarak/harga) perjalanan dan berakir pada suatu pola rute yang stabil.

2.4 Codeigniter

Codeigniter merupakan sebuah *framework* PHP yang merupakan aplikasi *open source* dengan penerapan konsep *view, model* dan *control* (MVC) untuk membangun *website* yang dinamis. Tujuan penggunaan codeigniter dapat mempercepat, mempermudah dan memberikan standarisasi keamanan saat membangun aplikasi *web* dibandingkan membangun dari awal. PHP merupakan kepanjangan (PHP *Hypertext Preprocessor*) adalah bahasa pemprograman yang digunakan untuk membuat aplikasi berbasis web dikembangkan oleh Rasmus L dan sekarang PHP menjadi standarisasi pemprograman disetiap universitas yang memiliki jurusan IT. Codeigniter adalah framework PHP yang ditulis oleh rick

Ellis pendiri ellislab.com yang merupakan perusahaan pengembang aplikasi *open sourc*e codeigniter.

2.5 Graph

Graph merupakan salah satu dari beberapa struktur data yang paling sering diaplikasikan dalam pemprograman komputer. Secara konseptual, *graph* merupakan suatu struktur data yang agak berbeda dengan pohon (*tree*) dalam kenyataanya, pohon merupakan salah satu jenis graph. Didalam penerapan komputer graph sering sekali digunakan dalam berbagai cara yang relatif lebih bermanfaat dari pohon (Nugroho Adi, 2009). *Graph* secara matematika sudah ditemukan sejak beberapa ratus tahun yang lalu sebelum ditemukanya pohon (*tree*). Dimasa-masa sebelumnya para matematikawan bekeraja pada graph tanpa menggunakan sarana bebasis komputer (Gregory L. Hielman, 1996). Kecanggihan struktur *graph* memungkinkan ilmuwan komputer merepresentasikan berbagai persoalan sulit dalam ilmu komputer (Duane Bailey, 2005). Jika ingin menggunakan penyimpanan *data* yang bersifat *eksternal* (*external storage*), mungkin tidak terlalu membutuhkan *graph*. Namun untuk beberapa permasalahaan dimana memerlukan representasi *internal* (dalam memori komputer) untuk suatu struktur data, *graph* tidak bisa dihindari penggunaanya (Robert Lafore, 1998).

Secara matematis *graph* didefinisikan sebagai pasangan himpunan (V,E) ditulis dengan notasi

G = (V,E).....(Rumus 2.1)

yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul(*vertex* atau *node*) dan E adalah himpunan sisi (*edge*) yang menghubungkan sepasang simpul (Munir, 2005).

Menurut (Fauzi Imron, 2010) Simpul (*vertex*) pada *graph* dapat dinyatakan dengan huruf, bilangan atau gabungan keduanya. Sedangkan sisi-sisi yang menghubungkan simpul u dengan simpul v dinyatakan dengan pasangan (u,v) atau dintayatakan dengan lambang e1, e2, e3 dan seterusnya. Dengan kata lain, jika e adalah sisi yang menghubungkan simpul u dengan simpul v, maka e dapat ditulisakan sebagai e=(u,v).

2.5.1 Jenis Dan Istilah Umum *Graph*

Dalam terminologi graph dikenal berbagai istilah umum, di antaranya (Robert Lafore, 1998):

1. *Adjacency* (Keberdampingan)

   Dua simpul dinamakan berdampingan (*adjacency*) jika saling terhubung melalui satu lintasan. Contohnya lintasan A dan B, D dan S serta E dan E1 dikatakan berdampingan.

2. *Path* (Lintasan)

   Lintasan (path) adalah urutan dari lintasan. Contohnya lintasan A ke D. Yang dapat disebut sebagai path ABCD adalah path dari simpul A ke D.

3. *Connected* (Terhubung)

   Suatu graph dikatakan terhubung (connected) jika ada setidaknya satu lintasan anatara suatu simpul ke simpul-simpul lainnya. Contohnya gambar *graph* diatas adalah graph terhubung (c*onnected graph*). Dengan kata lain,

graph tidak terhubung (unconnected graph) adalah *graph* yang satu atau lebih simpulnya tidak terhubung ke simpul lain.

4. *Directed Graph* (Graph Berarah)

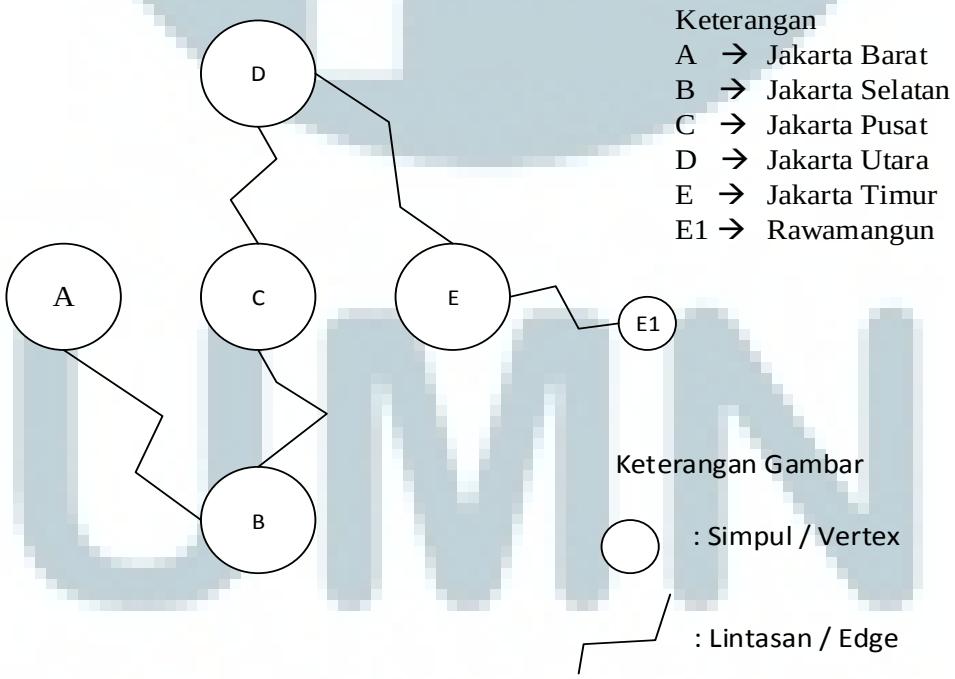
   Suatu *graph* dikatakan berarah jika lintasan memiliki arah tujuan ke simpul yang lain.

5. *Undirected Graph* (Graph Tidak Berarah)

   Suatu *graph* tidak berarah jika lintasan tidak memiliki arah panah yang menunjukan antar simpul.

6. *Weighted Graph* (Berbobot)

   Yaitu bilangan yang digunakan untuk mempresentasikan jarak fisik antara dua simpul atau waktu maupun biaya yang diperlukan untuk melintas dari suatu simpul kesimpul lainya.



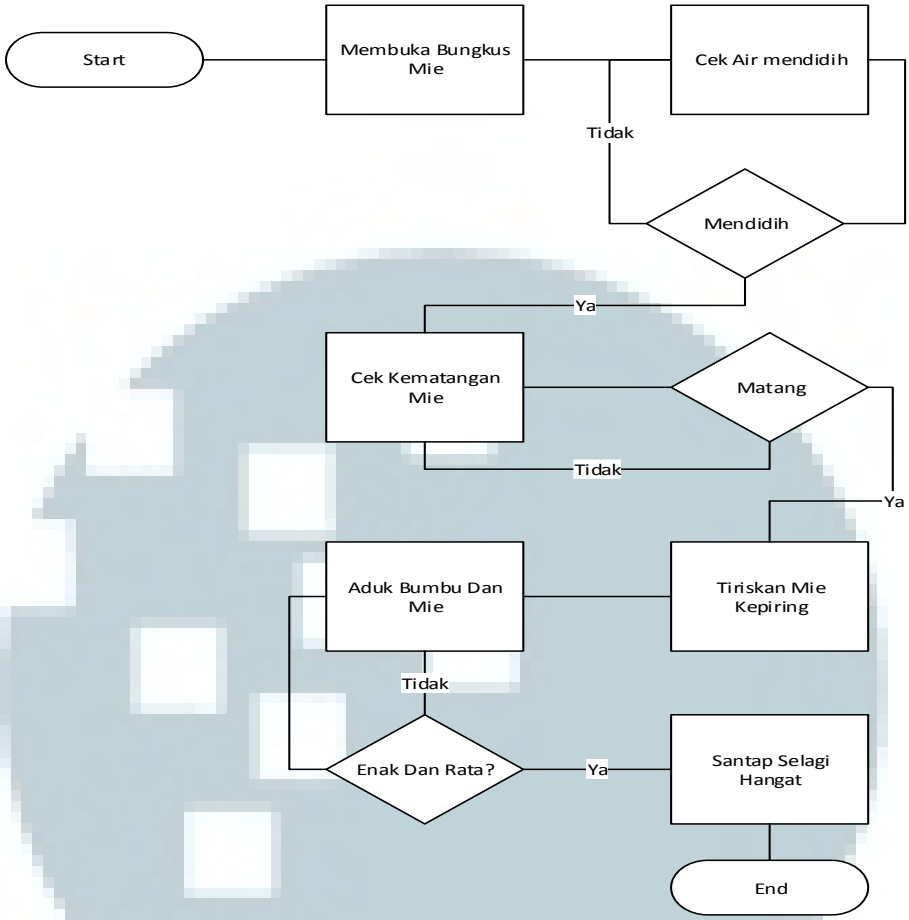
Gambar 2.1 Rute Pada Graph
Sumber : (*edit*) Nugroho Adi, 2008

2.6 Algoritma

Algoritma, ditinjau dari asal usul kata algoritma sendiri mempunyai sejarah yang aneh yaitu berawal dari kata *algorism* yang berarti proses menghitung dengan angka arab. Para ahli bahasa menemukan asal kata *algorism* dari penulis buku arab yang terkenal yaitu Abu Ja'far Muhammad ibnu Musa al-khuwarizmi.

Perubahan kata *algorism* menjadi *algorithm* dikarenakan seiring berjalannya waktu kata *algorism* sering dikelirukan dengan *arithmetic*, sehingga akhiran –sm berubah menjadi thm. Karena perhitungan dengan angka arab sudah menjadi hal yang biasa/lumrah, maka dengan berjalannya waktu kata algorithm dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna aslinya. Dalam bahasa indonesia kata *algorthm* diserap menjadi algoritma.

Deskripsi algoritma adalah langkah-langkah penyelesaian masalah yang tersusun secara logis atau urutan logis pengambilan keputusan untuk pemecahan suatu masalah. Algoritma ditulis dengan notasi khusus, notasi mudah dimengerti dan notasi dapat diterjemahkan menjadi sintaks suatu bahasa pemprograman (Zakaria dan Prijono, 2006). Dalam dunia komputer, algoritma sangan berperan penting dalam menunjang pembangunan suata *software*. Dalam kehidupan sehari hari, mungkin tanpa disadari bahwa algoritma telah masuk dalam kehidupan (mahluk sosial). Banyak orang yang salah mengartikan Algoritma dengan logaritma, logaritma merupakan operasi matematika yang merupakan kebalikan dari eksponen atau pemangkatan.



Gambar 2.2 Contoh sederhana Algoritma Membuat Mie Instan
Sumber : (*edit*) Imron Fauzi, 2011

Berikut merupakan gambar salah satu contoh nyata algoritma dalam kehidupan sehari-hari yang tanpa disadari.

Contoh diatas merupakan algoritma yang sering terjadi hampir setiap hari didunia nyata bagaimana langkah membuat mie instan. Jadi perlu diketahui algoritma diterapkan bukan hanya didalam dunia komputasi tapi dalam kehidupan sehari-hari juga sering terjadi. Contoh algoritma dalam dunia komputer biasanya diterapkan dalam pembuatan program seperti pada bahasa C, C++, C#, Java, Objective C, Pascal, dll kemudian algoritma tersebut disusun kedalam program dan terciptalah sebuah *software* ataupun aplikasi. Kesimpulan mengenai algoritma

merupakan salah satu cabang ilmu komputer yang membahas suatu langkah ataupun urutan untuk mencapai tujuan yang diinginkan.

2.6.1 Komponen Algoritma

Menurut (Nurhayati, 2010), Terdapat 5 buah komponen utama dalam sebuah algoritma antara lain *finiteness*, *definiteness*, *effectiveness*, *input* dan *output*, sehingga dalam membuat sebuah algoritma harus memiliki minimal 3 komponen yaitu:

1. *Input*

   Komponen ini terdiri dari variabel, jenis variabel, tipe variable, konstanta dan *parameter*

2. *Output*

   Komponen ini merupakan hasil dari perancangan algoritma dan program. Permasalahan yang diselesaikan dalam algoritma dan program harus tampil dalam komponen keluaran.

3. *Processing*

   Komponen ini merupakan bagian terpenting dalam merancang sebuah algoritma. Pada bagian ini memiliki logika permasalahan, logika algoritma, rumusan, metode yang diterapkan seperti penambahan, pengurangan dll.

2.6.2 Pseudocode

Pseudocode ataupun yang disebut kode semu merupakan deskripsi dari algoritma pemprograman computer yang menggunakan struktur sederhana dari

beberapa bahasa pemprograman tetapi bahasa tersebut bertujuan agar dapat dibaca oleh manusia. Biasanya kode semu ditulis dari variabel dan fungsi.

Berikut merupakan contoh kode semu/*pseudocode* sederhana algoritma dijkstra.

```
1  function Dijkstra(Graph, source):
2      for each vertex v in Graph:                                  // Initializations
3          dist[v]  := infinity ;                                    // Unknown distance function from
4                                                                    // source to v
5          previous[v]  := undefined ;                               // Previous node in optimal path
6      end for                                                       // from source
7
8      dist[source]  := 0 ;                                          // Distance from source to source
9      Q := the set of all nodes in Graph ;                          // All nodes in the graph are
10                                                                   // unoptimized - thus are in Q
11     while Q is not empty:                                         // The main loop
12         u := vertex in Q with smallest distance in dist[] ;       // Source node in first case
13         remove u from Q ;
14         if dist[u] = infinity:
15             break ;                                               // all remaining vertices are
16         end if                                                    // inaccessible from source
17
18         for each neighbor v of u:                                 // where v has not yet been
19                                                                   // removed from Q.
20             alt := dist[u] + dist_between(u, v) ;
21             if alt < dist[v]:                                     // Relax (u,v,a)
22                 dist[v]  := alt ;
23                 previous[v]  := u ;
24                 decrease-key v in Q;                              // Reorder v in the Queue
25             end if
26         end for
27     end while
28     return dist;
29 endfunction
```

Gambar 2.3 Pseudocode sederhana
Sumber : http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Tujuan utama dari dari penggunaan *pseudocode* adalah untuk mempermudah manusia dalam membaca dan memahami prinsip-prinsip dari algoritma yang telah di tulis terutama dari sisi kode dan sangat bermaanfaat sekali untuk *programmer*.

Pada *pseudocode*/kode semu sangatlah penting menjelaskan kondisi awal (*pre-condition*) dan kondisi akhir (*post condition*) serta *input* dan *output* dari algoritma.

Ini merupakan beberapa komponen yang harus dimiliki kode semu (*pseudocode*):

1. Kondisi awal dan kondisi akhir.
2. *Input* dan juga *output* algoritma.
3. Mendefinisikan semua variabel, fungsi ataupun kelas-kelas yang digunakan.
4. Komentar, agar mempermudah membaca pseudocode yang semakin rumit untuk dibaca.

2.7 Algoritma Greedy

Salah satu algoritma yang lebih sederhana adalah algoritma greedy, dimana algoritma ini pada dasarnya adalah algoritma yang selalu memilih solusi terbaik pada suatu saat tertentu tanpa memikirkan apakah pilihan dari solusi itu akan berakibat buruk terhadap pilihan lain di masa akan datang(Bruce Eckel dan Larry O'Brien, 2007).

Algoritma greedy secara umum mengindikasikan bahwa implementasi akan menggunakan sederetan pilihan yang terbaik yang akan memicu pilihan terbaik yang bersifat final. Jika memang demikian keadaanya, algoritma akan menghasilkan solusi yang *optimal* (bukan yang terbaik). Jika tidak, solusi *suboptimal* akan ditemukan. Meski demikan, untuk banyak permasalahaan, tidak selalu perlu menemukan solusi terbaik, sehingga dalam kasus seperti ini algoritma greedy bekerja dengan hasil yang cukup baik (Michael McMillan, 2006).

Gambar 2.4 Algoritma Sederhana Greedy
Sumber : (edit dari tulisan) Nugroho Adi, 2008

Contoh algortima greedy klasik :

Seorang kasir toko menentukan pecahan-pecahan uang yang akan dikembalikan kepada seorang pembeli.

Dari gambar diatas tidak akan mendapatkan seluruh kombinasi solusi yang mungkin. Inilah esensi metode algoritma greedy (Bruce Eckel dan Larry O'Brien, 2007).

2.8 Algoritma Shortest Path

Shortest path merupakan suatu *network* pengarahan perjalanan dimana seorang pengarah jalan ingin menentukan jalur terpendek antara dua kota berdasarkan jalur *alternatif* yang tersedia, dimana kota tujuan hanya satu. Masalah ini sendiri menggunakan representasi *graph* untuk memodelkan persoalan yang diwakili sehingga lebih memudahkan penyelesaiannya. Cara mengunjugi *vertex* pada *graph* dari *vertex* awal ke *vertex* akhir dengan bobot minimum, dimana dalam hal ini bobot yang digunakan adalah jarak dan kota-kota yang akan dikunjungi dan diasumsikan sebagi *graph* yang saling terhubung(*connected graph*) antara suatu kota ke kota lainnya. Jika setiap dua *vertex* yaitu V1 dan V2 dalam suatu *graph* terdapat sedikitnya sebuah *edge*. *Edge* pada *graph* berarah disebut arc. Untuk menyelesaikan *shortest path* banyak algoritma yang telah dikembangkan untuk menyelesaikan persoalan jalur terpendek diantaranya algoritma Dijkstra, algoritma Floyd-Warshall dan algoritma Bellman-Ford .

2.8.1 Dijkstra

Algoritma dijkstra merupakan algoritma yang paling sering digunakan dalam pencarian rute terpendek, sederhana penggunanya dengan menggunakan simpul-simpul sederhana pada jaringan jalan yang tidak rumit (chamero, 2006). Salah satu algoritma yang paling terkenal untuk menyelesaikan masalah *graph* berbobot yang dikenal dalam ilmu komputer adalah algoritma jalur terpendek yang ditemukan oleh Edsger Dijkstra pada tahun 1950 (Duane Bailey, 2005). Algoritma jalur tependek Dijkstra bekerja dengan menemukan jalur terpendek dari suatu simpul yang didefinisi dan melintasi seluruh simpul lain dalam *graph*. Algoritma Dijkstra melakukannya dengan menggunakan strategi greedy, yaitu mencari solusi optimum pada setiap langkah yang dilalui dengan tujuan untuk mendapatkan optimum dan menuju pada solusi terbaik sehingga kompleksitas waktu algoritma Dijkstra menjadi cukup besar yaitu

O (V*Log(v+e)) .....(Rumus 2.2)

dimana v dan e merupakan simpul pada sisi *graph* yang digunakan.

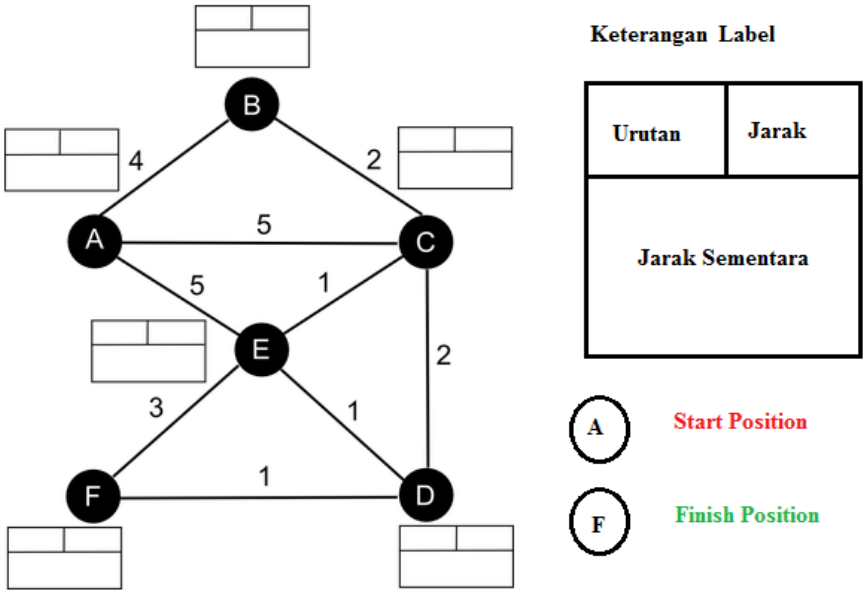
Permasalahan selanjutnya yang sering dijumpai pada suatu *graph* berbobot adalah bagaimana menemukan lintasan terpendek antara dua simpul (Bruce Eckel dan Larry O'Brien, 2007). Sehingga dalam dunia nyata algoritma Dijkstra sering dijumpai mulai dari perancangan papan sirkuit elektronika hingga penjadwalan proyek.

Algoritma Dijkstra merupakan solusi bagaimana cara menemukan lintasan terpendek antara dua simpul yang lebih kompleks dan lebih baik dibandingkan algoritma minimum spanning tree graph berbobot dalam menyelesaikan permasalahan (Nugroho Adi, 2008).

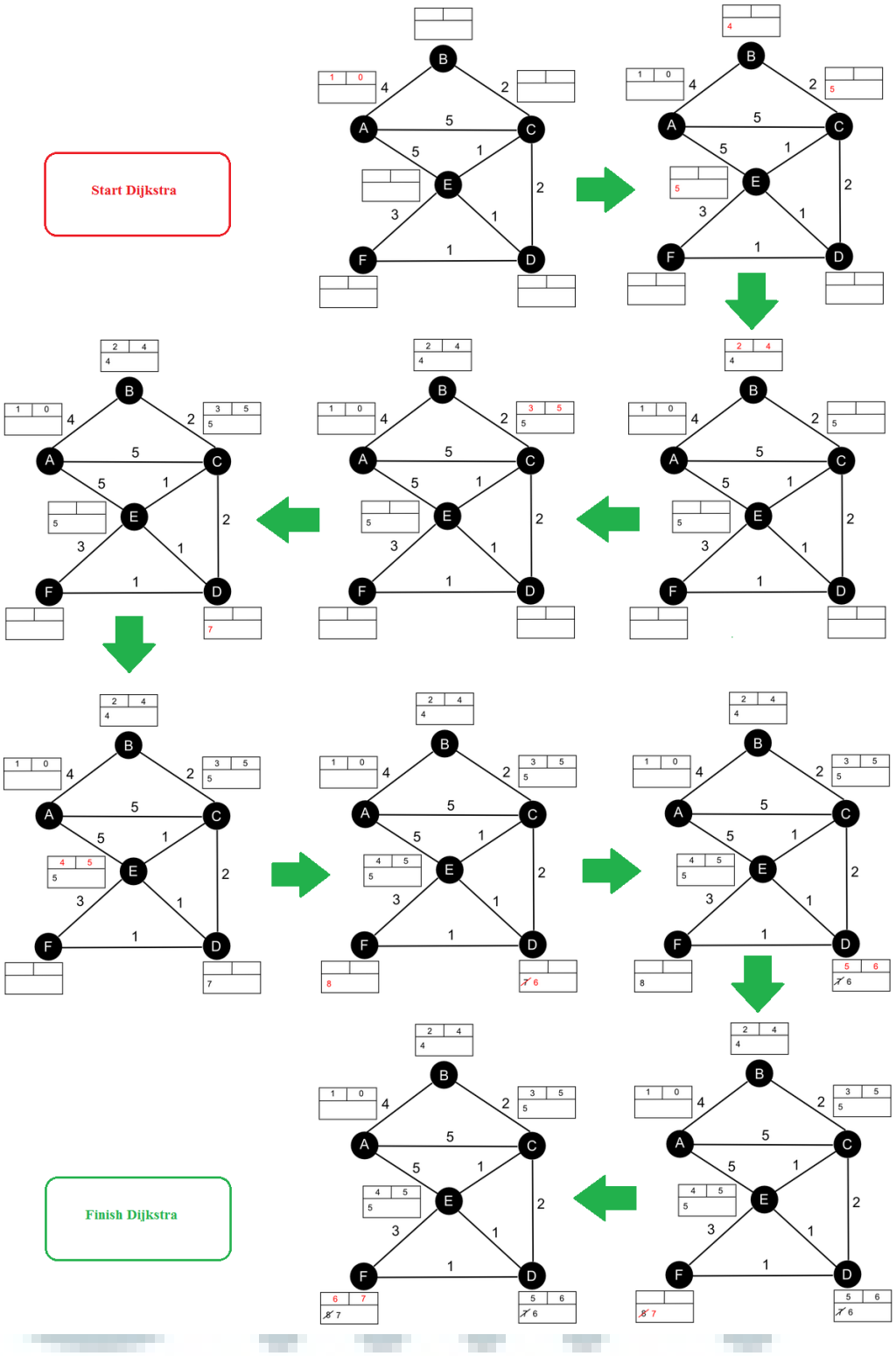
Menurut (Arsori Ahmad, 2013) untuk menerapkan algoritma Dijkstra dibutuhkan beberapa data yang harus disiapkan, yaitu:

1. Beberapa titik/simpul/daerah, titik/simpul/daerah yang bisa dijangkau secara langsung, dan juga jarak antara satu dengan yang lainnya.
2. Titik/simpul/daerah awal.
3. Titik/simpul/daerah tujuan.

Titik A adalah titik awal dan titik F adalah tujuan. Kemudian selanjutnya mencari rute manakah yang harus dilewati dan memiliki total jarak yang paling dekat. Untuk bisa mendapatkan rute tersebut, menambahkan label pada Gambar 2.5.



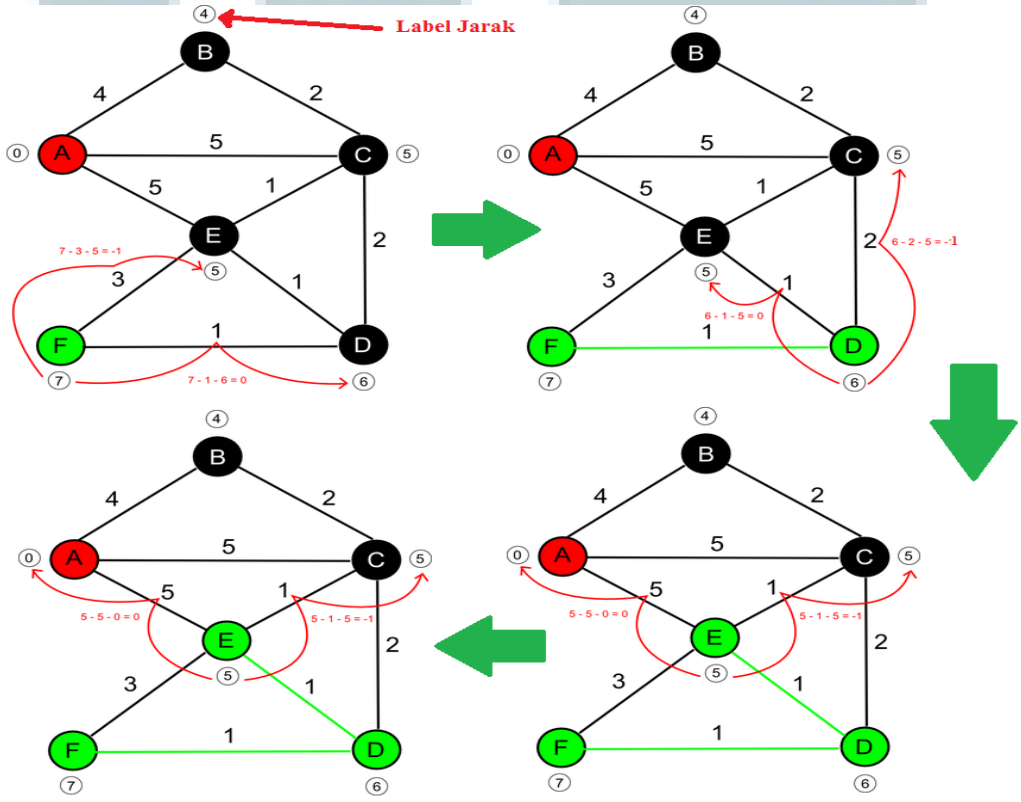
Gambar 2.5 Label Pada Dijkstra
Sumber : http://achmad-asrori.blogspot.com/2013/01/algoritma-dijkstra.html



Gambar 2.6 Penerapan Algoritma Dijkstra
Sumber : (edit) http://achmad-asrori.blogspot.com/2013/01/algoritma-dijkstra.html

Gambar 2.6 merupakan penerapan algoritma Dijkstra dalam pemberian label untuk mengetahui langkah-langkah selanjutnya.

Untuk mengetahui rute yang harus dilewati, Dijkstra akan menelusuri kembali dari tujuan hingga ke titik awal. Titik yang dapat dilalui F adalah E dan D maka untuk menentukan titik manakah yang seharusnya dilewati dengan cara mengurangkan label jarak titik F dengan jaraknya ke titik tujuan serta label jarak titik tersebut. Jika hasil < 0 maka titik tersebut tidak layak untuk dilewati, dan jika hasilnya >0 maka titik tersebut layak untuk dilewati, dan jika hasilnya lebih dari 0 serta lebih mendekati 0 maka titik tersebut yang harus dilewati seperti Gambar 2.7

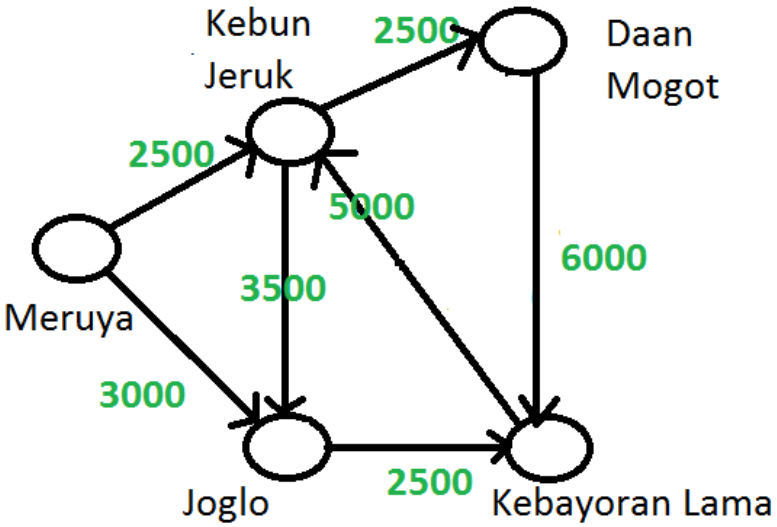


Gambar 2.7 Mengetahui Rute Di lewati Dijkstra
Sumber : http://achmad-asrori.blogspot.com/2013/01/algoritma-dijkstra.html

Dengan begitu hasil rekomendasi rute yang harus dilewati dan memili jarak terpendek dari titik A menuju F adalah A→E→D→F.

A. Penerapan Algoritma Dijkstra Dengan Harga Termurah

Dijkstra merupakan algoritma pencari jalur terpendek, penerapan dengan harga termurah hampir sama dengan penerapan jalur terpendek dimana nilai *edge* (berisi jarak) akan diganti dengan nilai edge (berisi harga) sehingga algoritma akan mencari nilai termurah dari rute tersebut. Berikut merupakan simulasi lintasan contoh penerapan algoritma dijkstra pada rute termurah, data pada harga yang tertera hanya berupa *dummy*.



Gambar 2.8 Penerapan rute termurah dijkstra pada lintasan

2.9 Penelitian Sebelumnya

Dasar dan acuan berupa teori-teori atau temuan-temuan melalui hasil berbagai penelitian sebelumnya merupakan hal yang sangat penting dijadikan sebagai data pendukung. Salah satu data pendukung yang menurut penulis perlu dijadikan bagian tersendiri adalah penelitian terdahulu yang relevan dengan permasalahan yang sedang dibahas dalam penelitian ini.

Penelitian sebelumnya mengenai algoritma dijkstra pada pencarian rute telah diteliti oleh Douglas Lawrie (MIT), Imron Fauzi dan Adi Nugroho pada buku terbitannya algoritma & struktur data dengan C#.

Penelitian menggunakan algoritma dijkstra sebagai metode pencarian rute terpendek/termurah dari suatu simpul yang dapat diterapakan pada rute angkutan umum DKI-Jakarta terutama Jakarta barat.