Studi Algoritma Optimasi dalam Graf Berbobot

Vandy Putrandika – NIM : 13505001

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15001@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang Algoritma-algoritma untuk melakukan optimasi dalam graf berbobot dan pengaplikasiannya dalam dunia nyata. Ada dua jenis optimasi yang akan dibahas kemudian, yaitu mencari lintasan terpendek dari suatu simpul ke simpul lain dalam sebuah graf berbobot dan mencari *minimum spanning tree* yaitu lintasan pada graf berbobot dengan jumlah bobot yang paling kecil. Algoritma yang akan dipakai dalam mencari lintasan terpendek adalah *Algoritma Dijkstra* (sesuai nama penemunya, Edsger Wybe Dijkstra). Sedangkan yang akan dipakai dalam mencari *minimum spanning tree* adalah *Algoritma Prim* dan *Algoritma Kruskal*, yaitu penyempurnaan dari Algoritma pertama yang ditemukan oleh Otakar Boruvka pada tahun 1926.

Kedua jenis optimasi ini mempunyai terapan yang sangat luas dalam praktek, karena dapat menentukan jarak atau waktu minimum untuk melakukan sesuatu yang berhubungan dengan graf berbobot sehingga dapat menghemat banyak waktu dan biaya.

Aplikasi dari kedua jenis optimasi ini pun bermacam-macam, namun yang cukup popular adalah mencari jarak terpendek dari suatu kota ke kota lain atau mencari waktu tersingkat untuk melakukan berbagai pekerjaan.

Kata Kunci : *Algoritma Dijkstra, Algoritma Prim, Algoritma Kruskal, Minimum Spanning Tree, graf berbobot*

1. Pendahuluan

Saat kita ingin bepergian dengan kendaraan pribadi ke suatu tempat, pernahkah terpikir jalan mana yang akan dipilih agar kita bisa sampai dengan secepat mungkin? Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi.

Atau saat ingin membuat jalan kereta api, pemerintah tidak begitu saja menghubungkan semua buah kota dengan rel-rel kereta api. Hal ini disebabkan oleh mahalnya biaya pembangunan rel kereta api, sehingga pembangunan jalur tersebut tidak perlu menghubungkan langsung dua buah kota; tetapi cukup membangun jalur kereta seperti pohon merentang. Karena dalam sebuah graf mungkin saja terdapat lebih dari satu pohon merentang, harus dicari pohon merentang yang mempunyai jumlah jarak terpendek, dengan kata lain harus dicari pohon merentang minimum. Persoalan ini juga termasuk salah satu persoalan optimasi.

Terdapat banyak aplikasi yang berkaitan dengan graf. Di dalam aplikasi itu, graf digunakan sebagai alat untuk merepresentasikan atau memodelkan persoalan. Berdasarkan graf yang dibentuk, barulah persoalan tersebut diselesaikan. Salah satu jenis graf yang sering dipakai adalah graf berbobot karena graf berbobot adalah graf yang paling bisa merepresentasikan peta dan diagram waktu.

Untuk mencari lintasan terpendek dari suatu simpul ke simpul yang lain kita akan menggunakan *Algoritma Dijkstra*, sedangkan untu mencari *minimum spanning tree*, kita akan menggunakan *Algoritma Prim* dan *Algoritma Kruskal*.

2. Mencari Lintasan Terpendek

Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (*weighed graph*), yaitu graf yang setiap sisinya diberikan nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya.

Asumsi yang kita gunakan disini adalah bahwa semua bobot bernilai positif. Kata "terpendek" jangan selalu diartikan secara fisik sebagai panjang minimum, sebab kata "terpendek" berbeda-beda maknanya bergantung pada tipikal persoalan yang akan diselesaikan. Namun, secara umum "terpendek" berarti meminimasi bobot lintasan 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 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 disini adalah menentukan jalur komunikasi terpendek antara dua buah terminal komputer. Lintasan terpendek akan menghemat waktu pengiriman pesan dan biaya komunikasi.

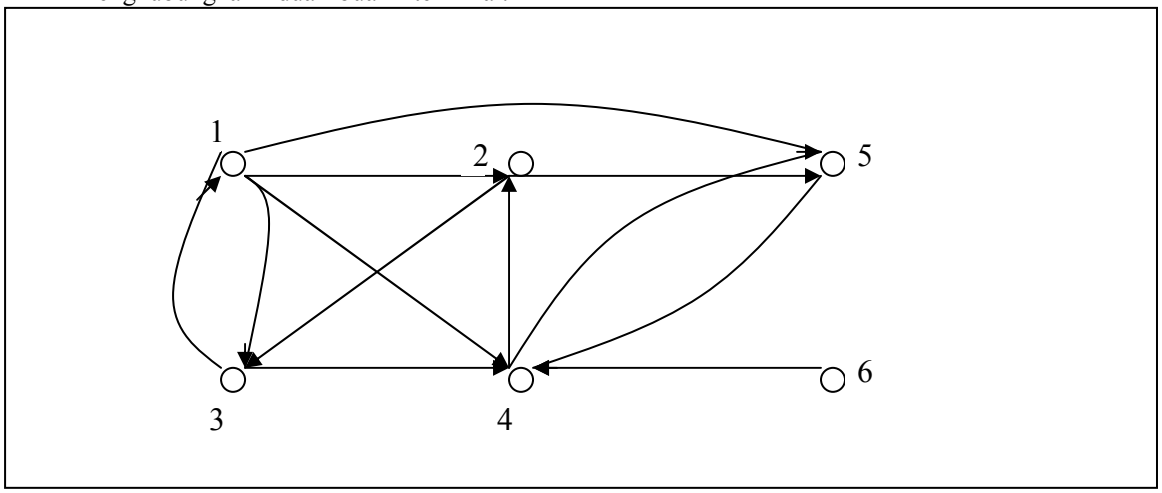
Ada beberapa macam persoalan lintasan terpendek, antara lain :

a. Lintasan terpendek antara dua buah simpul tertentu
b. Lintasan terpendek antara semua pasangan simpul
c. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain.
d. Lintasan terpendek antara dua buah simpul yang melalui yang melalui beberapa simpul tertentu.

Pada dasarnya, jenis persoalan a) mirip dengan jenis persoalan c), karena pencarian lintasan terpendek pada jenis persoalan c) dapat dihentikan bila simpul yang dikehendaki sudah ditemukan lintasan terpendeknya. Di dalam makalah ini kita memilih jenis persoalan c).

Deskripsi persoalan lintasan terpendek adalah sebagai berikut :

Diberikan graf berbobot G = (V, E) dan sebuah simpul awal a. Tentukan lintasan terpendek dari a ke setiap simpul lainnya di G.



Sebagai ilustrasi, tinjau graf berarah pada gambar di bawah ini. Lintasan terpendek dari simpul 1 ke setiap simpul lainnya diberikan pada tabel berikut..

Simpul Asal	Simpul tujuan	Lintasan terpendek	Jarak
1	3	1 -> 3	10
1	4	1 -> 3 -> 4	25
1	2	1 -> 3 -> 4 -> 2	45
1	5	1 -> 5	45
1	6	Tidak ada	-

Perhatikan dari tabel di atas bahwa lintasan terpendek dari 1 ke 2 berarti juga melalui lintasan terpendek dari 1 ke 3 dan dari 1 ke 4.

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). Aslinya algoritma Dijkstra diterapkan untuk mencari lintasan terpendek pada graf berarah. Namun algoritma ini juga benar untuk graf tak-berarah.

Algoritma Dijkstra mencari lintasan terpendek dalam sejumlah langkah. Algoritma ini menggunakan prinsip greedy. Prinsip greedy pada algoritma Dijkstra menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke himpunan solusi.

Ada beberapa versi algoritma Dijkstra yang ditulis pada berbagai pustaka. Algoritma yang dibahas di bawah ini adalah salah satu versinya.

Misalkan sebuah graf berbobot dengan n buah simpul dinyatakan dengan matriks ketetanggaan M = [m_{ij}], yang dalam hal ini,

m_{ij} = bobot sisi(i, j) (pada graf tak berarah mij = mji)

$m_{ii} = 0$

$m_{ij} = \infty$, jika tak ada simpul dari i ke j.

Selain matriks M, kita juga menggunakan larik S = [si] yang dalam hal ini,

s_i = 1, jika simpul i termasuk pada lintasan terpendek

s_i = 0 jika simpul i tidak termasuk ke dalam lintasan terpendek.

Dan larik/tabel D = [d_i] yang dalam hal ini,

d_i = panjang lintasan dari simpul awal a ke simpul i

Algoritma Dijkstra dinyatakan dalam *pseudo-code* sebagai berikut :

```
procedure Dijkstra (input m : matriks, a :simpul awal)
{ Mencari lintasan terpendek dari simpul awal a ke simpul lainnya
  Masukan : matriks ketetanggaan (m) dari graf berbobot G dan simpul awal a
  Keluaran : lintasan terpendek dari a ke semua simpul lainnya
}

Deklarasi
        s1, s2, ..., sn : integer   { larik integer }
        d1, d2, ...,dn : integer   { larik integer }
        i : integer
```

Algoritma

{ *Langkah 0 (inisialisasi) :* }
for i<- 0 to n do
    s_i <- 0
    d_i <- m_{ai}
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 <- 1 to n-1 do
    cari *j* sedemikian sehingga $s_j = 0$ dan $d_j = \min\{d_1, d_2, ..., d_n\}$
    s_j <- 1 { simpul *j* terpilih ke dalam lintasan terpendek }
    perbarui d_i, untuk *i* = 1, 2, 3, ..., n dengan : d_i (baru) = min {d_i (lama), $d_j + m_{ji}$}
endfor

Tinjau kembali graf berarah pada gambar di atas, dengan matriks ketetanggan M sebagai berikut :

i \ j	1	2	3	4	5	6
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 di tabulasikan sebagai berikut.

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	10	40	45	∞
										(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
1	1	1	1	0	0	0	0	0	∞	50	10	40	45	∞
										(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
2	3	1,3	1	0	1	0	0	0	∞	50	10	25	45	∞
										(1,2)	(1,3)	(1,3,4)	(1,5)	(1,6)
3	4	1,3,4	1	0	1	1	0	0	∞	45	10	25	45	∞
										(1,3,4,2)	(1,3)	(1,3,4)	(1,5)	(1,6)
4	2	1,3,4,2	1	1	1	1	0	0	∞	45	10	25	45	∞
										(1,3,4,2)	(1,3)	(1,3,4)	(1,5)	(1,6)
5	5	1,5	1	1	1	1	1	0	∞	45	10	25	45	∞
										(1,3,4,2)	(1,3)	(1,3,4)	(1,5)	(1,6)

(Keterangan : angka-angka di dalam tanda kurung menyatakan lintasan terpendek dari 1 ke simpul tersebut)

Jadi lintasan terpendek dari :

1 ke 3 adalah 1,3 dengan panjang = 10
1 ke 4 adalah 1,4 dengan panjang = 25
1 ke 2 adalah 1,3,4,2 dengan panjang = 45
1 ke 5 adalah 1,5 dengan panjang 45
1 ke 6 tak ada lintasan

3. Pohon Merentang Minimum

Jika G adalah graf berbobot, maka bobot pohon merentang T dari G didefinisikan sebagai jumlah bobot semua sisi di T. Pohon merentang yang berbeda memiliki bobot yang berbeda pula. Di antara semua pohon merentang di G, pohon merentang yang berbobot minimum –dinamakan pohon merentang minimum (minimum spanning tree)- merupakan pohon merentang yang paling penting. Pohon merentang minimum mempunyai terapan yang luas dalam praktek. Misalkan pemerintah akan membangun jalur kereta api yang menghubungkan sejumlah kota. Membangun jalur kereta api biayanya mahal, karena itu pembangunan jalur kereta api tidak perlu menghubungkan langsung dua buah kota; tetapi cukup membangun jalur kereta seperti pohin merentang. Karena di dalam sebuah graf mungkin saja terdapat lebih dari satu pohon merentang, harus dicari pohon merentang yang mempunyai jumlah jarak terpendek, dengan kata lain harus dicari pohon merentang minimum.

Terdapat dua buah algoritma membangun pohon merentang minimum. Yang pertama adalah algoritma Prim, dan yang kedua adalah algoritma Kruskal.

2.1. Algoritma Prim

Algoritma Prim adalah sebuah algoritma dalam graph theory untuk mencari pohon rentang minimum untuk sebuah graf terhubung berbobot, dengan kata lain sebuah himpunan bagian dari cabang-cabang yang membentuk suatu pohon yang terdiri dari semua node, di mana bobot keseluruhan semua cabang dalam pohon adalah paling kecil. Bila graf tersebut tidak terhubung, maka graf itu hanya memiliki satu pohon rentang minimum untuk satu dari komponen yang terhubung. Algoritma ini ditemukan pada 1930 oleh matematikawan Vojtech Jarník dan kemudian secara terpisah oleh computer scientist Robert C. Prim pada 1957 dan ditemukan kembali oleh Dijkstra pada 1959. Karena itu algoritma ini sering dinamai algoritma DJP atau algoritma Jarnik.

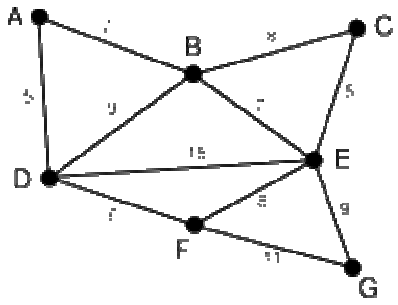
Algoritma Prim membentuk pohon merentang minimum langkah per langkah. Pada setiap langkah kita mengambil sisi dari graf G yang mempunyai bobot minimum namun terhubung dengan pohon merentang minimum T yang telah terbentuk.

Algoritma Prim
1. Ambil sisi dari graf G yang berbobot minimum, masukkan ke dalam T.
2. Pilih sisi (u,v) yang mempunyai bobot minimum dan bersisian dengan simpul di T, tetapi (u,v) tidak membentuk sirkuit di T. Tambahkan (u,v) ke dalam T
3. Ulangi langkah 2 sebanyak n-2 kali.

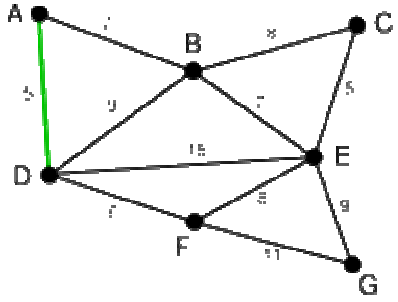
```
procedure Prim (input G : graf, output T : pohon)
{ Membentuk pohon merentang minimum T dari graf terhubung G
  Masukan : graf berbobot terhubung G = (V, E), yang mana |V| = n
  Keluaran : pohon rentang minimum T = (V, E')
}
Deklarasi
        i, p, q, u, v : integer

Algoritma
        Cari sisi (p, q) dari E berbobot terkecil
        T <- {(p, q)}
        for i <- 1 to n-2 do
                Pilih sisi (u, v) dari E yang bobotnya terkecil namun bersisian dengan simpul
di T
                T <- T U {(u,v)}
        endfor
```

Dengan struktur data binary heap sederhana, algoritma Prim dapat ditunjukkan berjalan dalam waktu *O*(Elog V), di mana E adalah jumlah cabang dan V adalah jumlah node. Dengan Fibonacci heap, hal ini dapat ditekan menjadi *O*(E + Vlog V), yang jauh lebih cepat bila grafnya cukup padat sehingga E adalah *O*(Vlog V).

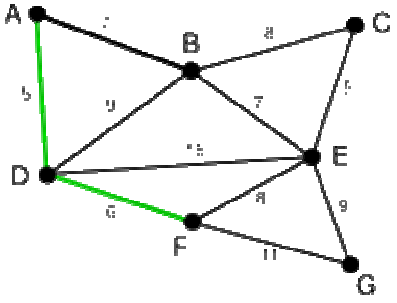
Ilustrasi Algoritma Prim :



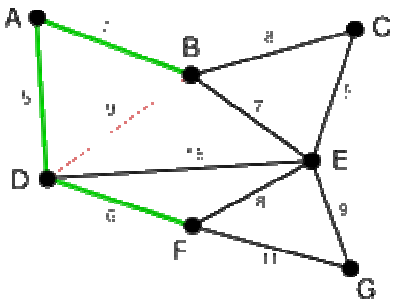
Ini adalah graf berbobot awal. Graf ini bukan pohon karena ada circuit. Nama yang lebih tepat untuk diagram ini adalah graf atau jaringan. Angka-angka dekat garis penghubung adalah bobotnya. Belum ada garis yang ditandai, dan node **D** dipilih secara sembarang sebagai titik awal.



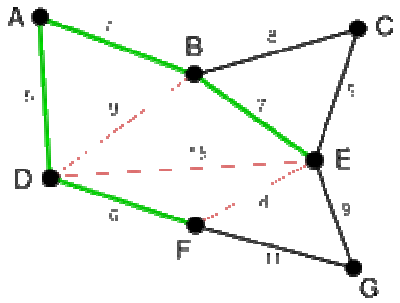
Node kedua yang dipilih adalah yang terdekat ke **D**: **A** jauhnya 5, **B** 9, **E** 15, dan **F** 6. Dari keempatnya, 5 adalah yang terkecil, jadi kita tandai node **A** dan cabang **DA**.



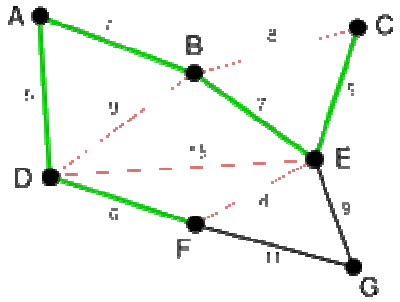
Node berikutnya yang dipilih adalah yang terdekat dari **D** *atau* **A**. **B** jauhnya 9 dari **D** dan 7 dari **A**, **E** jauhnya 15, dan **F** 6. 6 adalah yang terkecil, jadi kita tandai node **F** dan cabang **DF**.



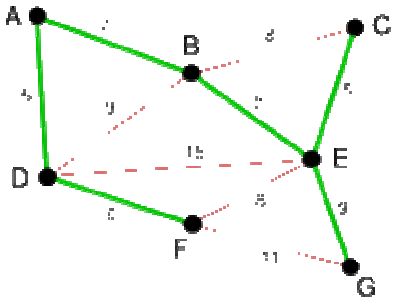
Algoritma ini berlanjut seperti di atas. Node **B**, yang jauhnya 7 dari **A**, ditandai. Di sini, cabang **DB** ditandai merah, karena baik node **B** dan node **D** telah ditandai hijau, sehingga DB tidak dapat digunakan.



Dalam hal ini, kita dapat memilih antara **C**, **E**, dan **G**. **C** jauhnya 8 dari **B**, **E** 7 dari **B**, dan **G** 11 dari **F**. **E** yang terdekat, jadi kita tandai node **E** dan cabang **EB**. Dua cabang lain ditandai merah, karena kedua node yang terhubung telah digunakan.



Di sini, node yang tersedia adalah **C** dan **G**. **C** jauhnya 5 dari **E**, dan **G** 9 dari **E**. **C** dipilih, jadi ditandai bersama dengan cabang **EC**. Cabang **BC** juga ditandai merah.



Node **G** adalah satu-satunya yang tersisa. Jauhnya 11 dari **F**, dan 9 dari **E**. **E** lebih dekat, jadi kita tandai cabang **EG**. Sekarang semua node telah terhubung, dan pohon rentang minimum ditunjukkan dengan warna hijau, bobotnya 39

Bukti Algoritma Prim

Misalkan *P* adalah sebuah graf terhubung berbobot. Pada setiap iterasi algoritma Prim, suatu cabang harus ditemukan yang menghubungkan sebuah node di graf bagian ke sebuah node di luar graf bagian. Karena *P* terhubung, maka selalu ada jalur ke setiap node. Keluaran *Y* dari algoritma Prim adalah sebuah pohon, karena semua cabang dan node yang ditambahkan pada *Y* terhubung.

Misalkan Y_1 adalah pohon rentang minimum dari P. Bila Y_1=*Y* maka *Y* adalah pohon rentang minimum. Kalau tidak, misalkan *e* cabang pertama yang ditambahkan dalam konstruksi *Y* yang tidak berada di Y_1, dan *V* himpunan semua node yang terhubung oleh cabang-cabang yang ditambahkan sebelum *e*. Maka salah satu ujung dari *e* ada di dalam *V* dan ujung yang lain tidak. Karena Y_1 adalah pohon rentang dari *P*, ada jalur dalam Y_1 yang menghubungkan kedua ujung itu.

Bila jalur ini ditelusuri, kita akan menemukan sebuah cabang *f* yang menghubungkan sebuah node di *V* ke satu node yang tidak di *V*. Pada iterasi ketika *e* ditambahkan ke *Y*, *f* dapat juga ditambahkan dan akan ditambahkan alih-alih *e* bila bobotnya lebih kecil daripada *e*. Karena *f* tidak ditambahkan, maka kesimpulannya

$$w(f) \geq w(e).$$

Misalkan Y_2 adalah graf yang diperoleh dengan menghapus *f* dan menambahkan *e* dari Y_1. Dapat ditunjukkan bahwa Y_2 terhubung, memiliki jumlah cabang yang sama dengan Y_1, dan bobotnya tidak lebih besar daripada Y_1, karena itu ia adalah pohon rentang minimum dari *P* dan ia mengandung

e dan semua cabang-cabang yang ditambahkan sebelumnya selama konstruksi *V*.

Ulangi langkah-langkah di atas dan kita akan mendapatkan sebuah pohon rentang minimum dari *P* yang identis dengan *Y*. Hal ini menunjukkan bahwa *Y* adalah pohon rentang minimum.

2.2. Algoritma Kruskal

Pada algoritma kruskal, sisi-sisi graf diurut terlebih dahulu berdasarkan bobotnya dari kecil ke besar. Sisi yang dimasukkan ke dalam himpunan T adalah sisi graf G sedemikian sehingga T adalah pohon. Pada keadaaan awal, sisi-sisi sudah diurut berdasarkan bobot membentuk hutan (forest), masing-masing pohon di hutan hanya berupa satu buah simpul. Hutan tersebut dinamakan hutan merentang (spanning forest). Sisi dari graf G ditambahkan ke T jika ia tidak membentuk siklus di T.

Perbedaan prinsip antara algoritma Prim dan Kruskal adalah: jika pada algoritma Prim sisi yang dimasukkan ke dalam T harus bersisian dengan sebuah simpul pada T, maka pada algoritma Kruskal sisi yang dipilih tidak perlu bersisian dengan sebuah simpul di T asalkan penambahan sisi tersebut tidak membentuk sirkuit (siklus)

Algoritma Kruskal

(asumsi : sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya - dari bobot kecil ke besar)

1. T masih kosong

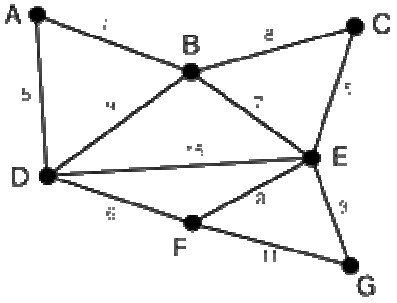
2. Pilih sisi (u,v) dengan bobot minimum yang tidak membentuk sirkuit di T. Tambahkan (u,v) ke dalam T.

3. Ulangi langkah 2 sebanyak n-1 kali.

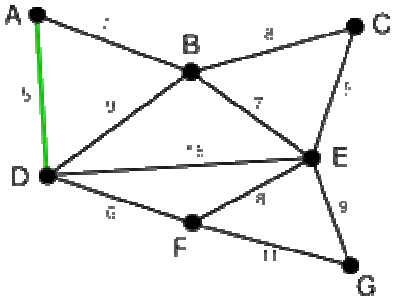
Dalam notasi pseudo-code, algoritma Kruskal dapat kita tuliskan sebagai berikut :

```
procedure Kruskal (input G : graf, output T : pohon)
{ Membentuk pohon merentang dari minimum T dari graf terhubung G.
  Masukan : graf berbobot terhubung G = (V, E), yang mana |V| = n
  Keluaran : pohon rentang minimum T (V, E')
}
Deklarasi
        i, p, q, u, v : integer

Algoritma
        ( Asumsi : sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya - dari bobot
kecil ke bobot besar)
        T <- {}
        while jumlah sisi T < n-1 do
                Pilih sisi (u, v) dari E yang bobotnya terkecil
                if (u,v) tidak membentuk siklus di T then
                        T <- T U {(u,v)}
                endif
        endwhile
```

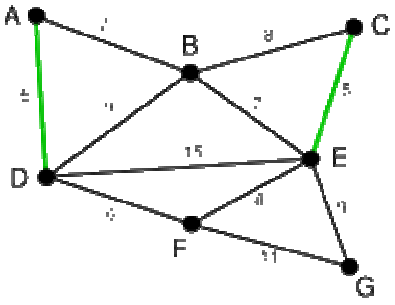
Ilustrasi Algoritma Kruskal :



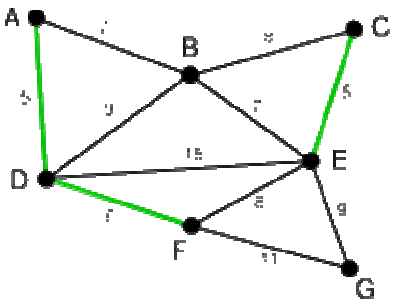
Ini adalah graf berbobot awal. Graf ini bukan pohon karena ada circuit. Nama yang lebih tepat untuk diagram ini adalah graf atau jaringan. Angka-angka dekat garis penghubung adalah bobotnya.



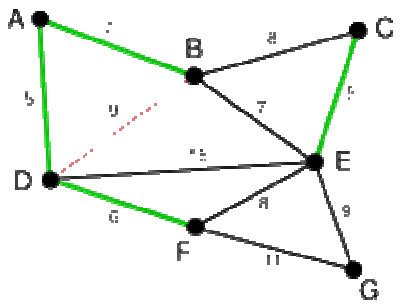
Dari semua bobot yang ada yang terpendek adalah 5. Yaitu AD dan CE. Oleh karena itu kita akan mewarnai AD dengan warna hijau sebagai tanda dimasukkan ke dalam hutan terentang.



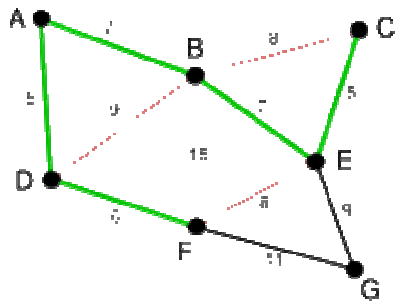
Setelah AD sisi yang bobotnya terkecil adalah CE. Oleh karena itu kita akan mewarnai CE dengan warna hijau



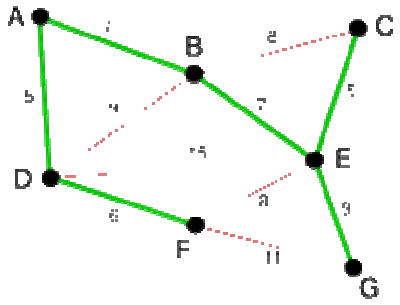
Lalu yang bobotnya terkecil setelah AD dan CE adalah DF yang berbobot 6. Oleh karena itu akan kita warnai hijau



Lalu yang bobotnya terkecil setelah AD, CE, dan DF adalah AB yang bernilai 7. Maka kita warnai hijau. Karena BD membentuk siklus bila ditambahkan, maka kita warnai merah, artinya tidak akan kita tambahkan.



Kemudian, yang bobotnya paling kecil lagi adalah BE yang bernilai 7. Maka kita warnai hijau. Karena BC dan EF akan membentuk siklus bila kita tambahkan ke pohon, maka akan kita warnai merah.



Terakhir, karena untuk menuju G bobot terendah adalah EG, maka kita warnai hijau sebagai tanda kita masukkan ke pohon terentang. Sekarang semua node terhubung dengan jumlah bobot 39.

Bukti Algoritma Kruskal

Biarkan P terhubung,jadilah suatu graf berbobot dan biarkan Y jadilah subgraph dari P yang diproduksi oleh algoritma itu. Y tidak bisa mempunyai suatu siklus, sejak sisi yang ditambahkan untuk siklus itu akan telah di dalam satu subtree dan bukan antara dua Tree yang berbeda. Y tidak bisa diputus, karena lebih dulu ditemui sisi itu menggagungkan dua komponen dari Y akan telah ditambahkan oleh algoritma itu. Seperti itu, Y adalah suatu Spanning Tree dari P.

Untuk menyederhanakan, berasumsi bahwa semua sisi mempunyai bobot yang berbeda.

Biarkan Y1 jadilah suatu Minimum Spanning Tree. Jika Y1=Y kemudian Y adalah suatu Minimum Spanning Tree. Cara lainnya, biarkan e jadilah tepi yang pertama yang dipertimbangkan oleh algoritma yang di dalam Y tetapi bukan di dalam Y1. Y1+E mempunyai suatu siklus, sebab kamu tidak bisa menambahkan suatu sisi equivelent dengan spanning tree dan masih mempunyai suatu Tree. Siklus ini berisi tepi lain f yang (mana) di langkah algoritma di mana e ditambahkan untuk Y, belum dipertimbangkan. Ini adalah sebab jika tidak e tidak akan menghubungkan Tree yang berbeda. hanyalah dua cabang dari Tree yang sma. Kemudian Y2=Y1+E-F adalah juga suatu spanning tree. Bobot total nya adalah kurang dari total berat/beban dari Y1.

Ini Adalah sebab algoritma mengunjungi e [sebelum/di depan] f. Kesimpulan ialah .. Y1 adalah tidak ada Minimum Spanning Tree, dan asumsi yang di sana ada suatu sisi di dalam Y, tetapi bukan di dalam Y1, adalah salah. Membuktikan ini yang Y=Y1, yaitu., Y adalah suatu Minimum Spanning Tree.

4. Kesimpulan

1. Untuk mencari langkah terpendek dari sebuah graf berbobot kita dapat menggunakan algoritma Dijkstra, yaitu kumpulan langkah-langkah yang ditemukan oleh Edsger Wybe Dijkstra.
2. Untuk mencari minimum spanning tree kita dapat menggunakan algoritma Prim dan Kruskal yang keduanya merupakan turunan dari algoritma yang diciptakan oleh Otakar Boruvka.
3. Algoritma Prim mencari spanning tree dengan cara melihat sisi terpendek dari graf yang bersisian dengan pohon terentang yang telah dibuat dan yang tidak membentuk siklus.
4. Sedangkan Algoritma Prim mencari minimum spanning tree dengan cara melihat sisi terpendek yang tidak membentuk siklus, walaupun tidak bersisian dengan pohon terentang yang telah dibuat.

Daftar Pustaka

1. *Wikipedia.org* http://id.wikipedia.org *Tanggal akses: 3 Januari 2007 pukul 13.20.*
2. Munir, Rinaldi. *Diktat Kuliah Matematika Diskrit Edisi Keempat.* Teknik Informatika ITB. 2004