Penerapan Strategy Greedy
Untuk Membangun Pohon Merentang Minimum

Bayu Aditya Pradhana

Program Studi Teknik Informatika Institut Teknologi Bandung
Kampus ITB Jl.Ganesha No.10 Bandung
e-mail: deryck_18@yahoo.com

ABSTRAK

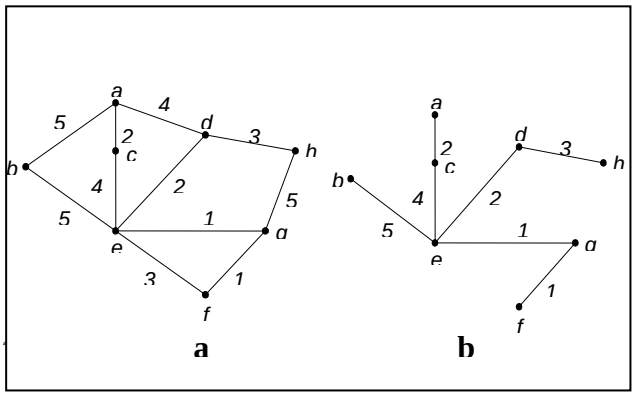
Teori graf berkembang dan banyak di aplikasikan pada kehidupan sehari-hari manusia hingga saat ini. Salah satu cabang dari teori graf yang banyak dikembangkan saat ini adalah penggunaan teori pohon. Konsep pohon dianggap sebagai salah satu konsep yang paling populer saat ini dan banyak digunakan untuk menyelesaikan beberapa masalah yang ditemui manusia dalam kehidupan sehari-harinya. Konsep pohon dapat digunakan ketika hendak direntangkan jaringan kabel listrik yang menghubungkan sejumlah lokasi dengan panjang kabel yang digunakan sependek-pendeknya mungkin, melihat pengelompokan data yang tersebar pada suatu ruang, ataupun pada perencanaan jaringan transportasi/distribusi barang. Ketiga aplikasi yang disebutkan tadi merupakan contoh aplikasi konsep pohon yaitu dengan menentukan suatu pohon merentang minimum. Saat ini terdapat beberapa algoritma untuk menentukan suatu pohon merentang minimum, namun pada makalah ini hanya akan membahas dua algoritma yang cukup terkenal untuk menyelesaikan permasalahan ini yaitu algoritma prim dan algoritma kruskal. Kedua algoritma ini sama-sama menerapkan strategi *greedy* dalam pencarian solusi optimumnya, dalam hal ini pohon akan dibangun secara bertahap, sisi demi sisi.

Kata kunci: Kata kunci 1, Kata kunci 2, ..., Kata kunci 5.

1. PENDAHULUAN

Dasar algoritma *greedy* adalah membangu suatu himpunan solusi besar diatas solusi kecil. Algoritma *greedy* hanya mengambil solusi yang terbaik yang mereka temukan pada saat itu tanpa memperhatikan konsekuensi lainnya. Algoritma greedy sangat terkenal untuk menyelesaikan beberapa masalah optimasi meskipun ternyata solusi yang ditemukan merupakan sebuah solusi optimal, tetapi secara garis besar algoritma greedy dapat menyelesaikan persoalan optimasi dengan baik.

Salah satu contoh masalah optimasi terdapat pada persoalan pohon merentang minimum. Pohon rentang minimum adalah pohon rentang dari graf sedemikian sehingga semua pohon rentang lain memiliki bobot lebih besar atau sama dengan pohon rentang minimum (*wikipedia.org*). Persoalan pohon rentang minimum telah banyak dijadikan dasar untuk menyelesaikan permasalahan pada kehidupan sehari-hari manusia, seperti ketika hendak direntangkan jaringan kabel listrik yang menghubungkan sejumlah lokasi dengan panjang kabel yang digunakan sependek-pendeknya mungkin, melihat pengelompokan data yang tersebar pada suatu ruang, ataupun pada perencanaan jaringan transportasi/distribusi barang seperti membangun jalur kereta api yang menghubungkan sejumlah kota seperti yang digambarkan oleh graf pada Gambar 1.



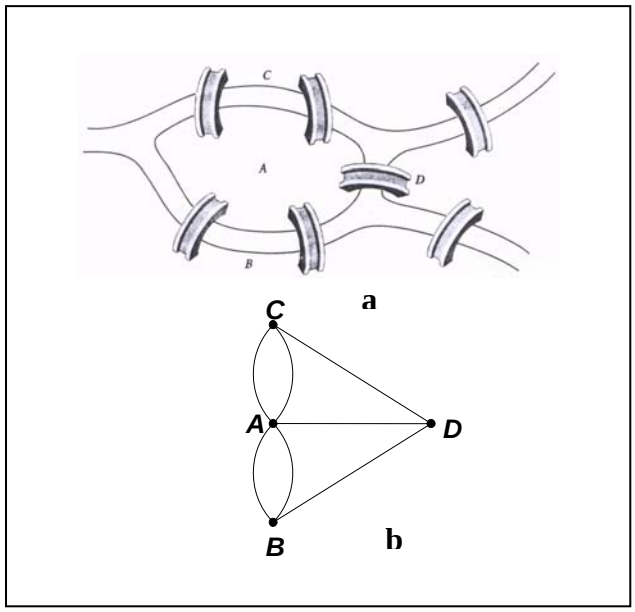
Gambar 1. a) Graf yang menyatakan jalur kereta api. Bobot pada sisi menyatakan panjang rel (per 10000 Km); b) Pohon Merentang minimum yang mempunyai jarak minimum

Apabila kita ingin menggabungkan beberapa kota dengan jalur kereta api maka di perlukan biaya yang cukup besar, oleh karena itu dalam menentukan jalur kereta api, perlu ditentukan jalur yang paling efisien dan ekonomis dengan membentuk suatu himpunan solusi yang paling optimal dan pada kasus ini kita dapat menggunakan metode pohon merentang minimum. Dalam membangun

pohon merentang minimum tersebut terdapat dua algoritma yang cukup terkenal yaitu algoritma prim dan algoritma kruskal. Pada makalah ini penulis mencoba menggambarkan perbandingan kedua algoritma yang sama-sama menerapkan strategi *Greedy.*

2. DASAR TEORI

Menurut catatan sejarah, pada kira-kira tahun 1736 teori graf pertama kali digunakan untuk menangani masalah jembatan *Konigsberg* (dapat dilihat pada gambar 2.1), jembatan yang menggubungkan pulau *Kneiphof* yang dikelilingi oleh sungai *Pregal* dengan daratan lainnya.



Gambar 2.1. a) Jembatan Konigsberg; dan b) graf yang merepresentasikan jembatan Konigsberg

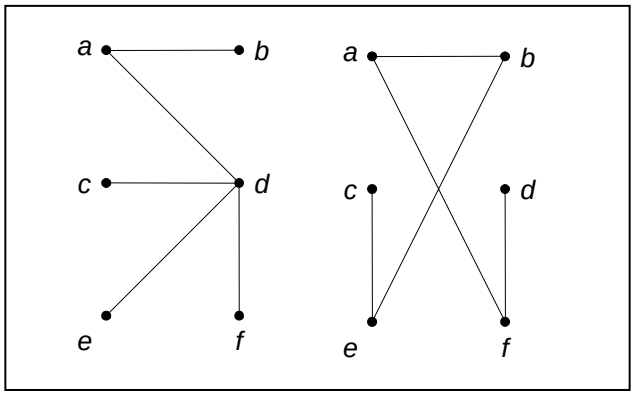
Graf sendiri didefinisikan sebagai pasangan himpunan tidak kosong dari simpul-simpul (*vertices* atau *node*) dan himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul [Munir,2005,halVIII-2]. Teori graf yang bila dilihat dari usianya dapat dikatakan teori tua ternyata memiliki aplikasi yang sangat sering kita temui dalam kehidupan manusia sehari-hari. Salah satu yang paling sering kita temui adalah peta. Peta yang merupakan representasi visual dari teori graf dapat dikembangkan menjadi beberapa aplikasi lainnya seperti aplikasi jalur kereta api yang telah disebutkan sebelumnya, selain itu dapat juga dikembangkan menjadi peta jaringan komunikasi LAN, dan masih banyak lagi aplikasi lain yang dapat dikembangkan dari representasi visual graf.

Seiring dengan banyaknya aplikasi yang dapat dikembangkan dari teori graf maka semakin banyak juga teori yang muncul yang juga berkaitan dengan teori graf. Salah satunya adalah konsep pohon. Konsep pohon sendiri sudah lama digunakan yaitu pada sekitar tahun 1857 ketika matematikawan asal Inggris, Arthur Cayley, mencoba menghitung jumlah senyawa kimia. Konsep ini menjadi sangat populer dan penting saat ini karena mampu menangani banyak aplikasi permasalahan yang berhubungan dengan Teori Graf.

Pada konsep pohon, terdapat beberapa jenis model representasi pohon yang dapat digunakan untuk pencarian suatu solusi dari sebuah permasalahan terkait dengan teori graf. Salah satu bentuknya adalah pohon merentang atau yang lebih dikenal dengan *Spanning Tree*. Jika Spanning Tree diterapkan pada persoalan yang mengandung unsur pencarian bobot yang minimum maka dinamakan *Minimum Spanning Tree (MST).* Banyak sekali berbagai masalah yang dapat diselesaikan dengan memodelkan masalah tersebut ke dalam Graf kemudian diselesaikan dengan menentukan pohon merentang minimumnya, seperti pada contoh kasus jalur kereta api. Ada beberapa algoritma yang digunakan untuk pembangunan MST. Pada makalah ini hanya akan digunakan algoritma Prim dan Kruskal yang merupakan wakil algoritma yang menerapkan strategi *greedy.*

2.1 Pohon (Tree)

Pohon adalah graf tak-berarah terhubung dan tidak mengandung sirkuit.



Gambar 2.2. Contoh Pohon

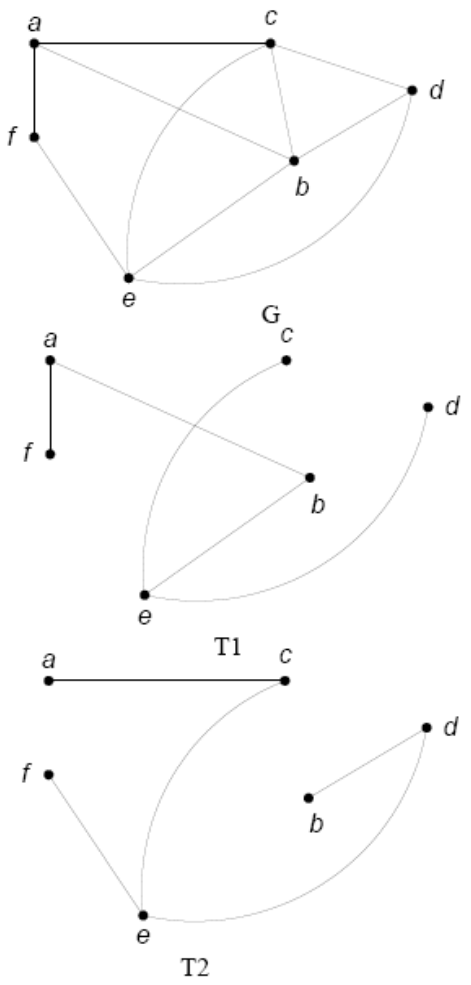
Sifat yang terpenting pada pohon adalah terhubung dan tidak mengandung sirkuit. Pohon dinotasikan sama dengan :

T = (V,E)

Keterangan :
T : Tree
V : Vertices atau node atau vertex atau simpul, V merupakan himpunan tidak kosong.
V = {v1,v2,…,vn}
E : Edges atau Arcs atau sisi yang menghubungkan simpul
E = {e1,e2,…,en}

2.2 Pohon Merentang (Spanning Tree)

Misalkan G = (V,E) adalah graf tak-berarah terhubung yang bukan pohon, artinya di G terdapat sirkuit. G dapat diubah menjadi pohon T = (V,E) dengan cara memutuskan sirkuit-sirkuit yang ada. Caranya yaitu dengan memutuskan salah satu sisi pada sirkuit hingga tidak ada sirkuit pada G. Jika di G tidak lagi ada sirkuit maka pohon T ini disebut dengan pohon merentang. Disebut merentang karena semua simpul pada pohon T sama dengan simpul pada graf G.[1]

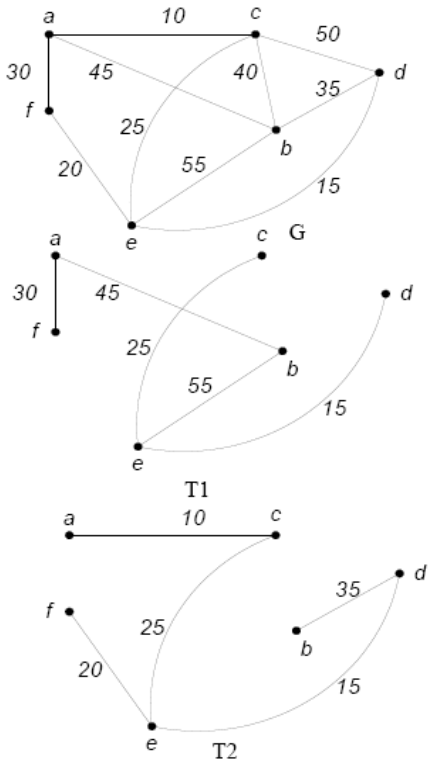


Gambar 2.3. G adalah Graf, T1 dan T2 adalah Pohon Merentang dari Graf G

Keterangan :
- T1 dan T2 merupakan pohon merentang dari graf G
- Pohon merentang T1 dibentuk dengan cara menghapus sisi {(a,c),(b,c),(b,d),(c,d),(e,f)} dari graf G.
- Pohon merentang T1 dibentuk dengan cara menghapus sisi {(a,f),(a,b),(b,c),(b,e),(c,d)} dari graf G.

2.3 Pohon Merentang Minimum (Minimum Spanning Tree)

Jika G pada gambar 2 merupakan graf berbobot, maka bobot pohon merentang T1 atau T2 didefinisikan sebagai jumlah bobot semua sisi di T1 atau T2. Diantara pohon merentang yang ada pada G, yang palin penting adalah pohon merentang dengan bobot minimum. Pohon merentang dengan bobot minimum ini disebut dengan pohon merentang minimum atau *Minimum Spanning Tree (MST).* Contoh aplikasi MST yang sering digunakan adalah pemodelan proyek pembangunan jalan raya menggunakan graf. MST digunakan untuk memilih jalur dengan bobot terkecil yang akan meminimalkan biaya pembangunan jalan.



Gambar 2.4. G adalah Graf Berbobot, T1 dan T2 adalah Pohon Merentang Berbobot dari Graf G

3. PEMBAHASAN

3.1 Strategi Greedy

Strategi *Greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Algoritma *greedy* membentuk solusi langkah per langkah yaitu :

- Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.
- Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan
- Yang terlihat memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal (*local optimum*) pada setiap langkah dan diharapkan akan mendapatkan solusi optimum global (*global optimum*)

3.2 Algoritma Prim

Algoritma dimulai dari suatu verteks awal tertentu: bisa ditentukan oleh pemanggil atau dipilih sebarang oleh algoritma. Misalnya verteks awal tersebut adalah v.

Pada setiap iterasi terdapat kondisi dimana himpunan verteks V terbagi dua dalam:

- W yaitu himpunan verteks yang sudah dievaluasi sebagai node di dalam pohon, serta
- (V-W) yaitu himpunan verteks yang belum dievaluasi.
- Di awal algoritma W diinisialisasi berisi verteks awal v.
- Selanjutnya, di dalam iterasinya:
    - Pada setiap adjacency dari tiap verteks dalam W dengan verteks dalam (V-W) dicari sisi dengan panjang minimal.
    - setelah diperoleh, sisi tersebut ditandai sebagai sisi yang membentuk tree dan verteks adjacent sisi tersebut dalam (V-W) dipindahkan ke W (menjadi anggota W).
    - Jika sisi tersebut tidak ada maka proses selesai.

Strategi *greedy* yang digunakan adalah:

> Pada setiap langkah, pilih sisi *e* dan graf *G(V,E)* yang mempunyai bobot terkecil dan bersisian dengan simpul-simpul di *T* tetapi *e* tidak membentuk sirkuit di *T*.

3.3 Algoritma Kruskal

Pada algoritma Kruskal, sisi-sisi graf diurut terlebih dahulu berdasarkan bobotnya yang menaik (dari kecil ke besar). Pada keadaan awal, sisi-sisi yang sudah diurut berdasarkan bobot tersebut membentuk suatu hutan (forest), masing-masing pohon di dalam hutan tersebut hanya terdiri dari satu simpul saja. Hutan tersebut dinamakan hutan merentang. Setiap pohon di dalam hutan disebut juga komponen dari poho merentang.

Mulai dari sisi dengan bobot terkecil hingga terbesar lakukan dalam iterasi:

- jika sisi tsb. menghubungkan dua verteks dalam satu subset (berarti membentuk siklik) maka skip sisi tersebut dan periksa sisi berikutnya
- jika tidak (berarti membentuk siklik) maka kedua subset dari verteks-verteks yang bersangkutan digabungkan menjadi satu subset yang lebih besar.
- Iterasi akan berlangsung hingga semua sisi terproses.

Strategi *greedy* yang digunakan adalah:

> Pada setiap langkah, pilih sisi *e* dari graf *G* yang mempunyai bobot minimum tetapi *e* tidak membentuk sirkuit di *T*.

4. ANALISA

Analisa akan dilakukan pada algoritma Prim dan algoritma Kruskal yang memperlihatkan strategi greedy untuk diterapkan pada suatu kasus graf.

4.1 Analisa Algoritma Prim

Algoritma Prim menggunakan strategi *Greedy* yaitu pada satu langkah dia hanya akan memilih satu yang terbaik dan tidak mungkin mengulang langkah yang sebelumnya hingga akhir dari algoritma. Strategi Greedy yang digunakan adalah pada setiap langkah, pilih sisi *e* dari graf *G(V,E)* yang mempunyai bobot terkecil dan bersisian dengan simpul-simpul di *T* tetapi *e* tidak membentuk sirkuit di *T*.

```
Procedure Prim(input G : graf, output
T : pohon)
{ Membentuk pohon merentang minimum T
dari graf terhubung G.
Masukan : graf-berbobot terhubung
          G = (V,E), dimana |V| = n
Keluaran : pohon rentang minimum T =
          (V,E')
}

Deklarasi
i, p, q, u, v : integer

Algoritma

Cari sisi (p,q) dari E yang berbobot
terkecil
T <- {(p,q)}

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

Analisa :

Langkah pertama pada Algoritma Prim adalah mencari sisi pada himpunan E yang menyatakan sisi-sisi pada graf G dengan bobot terkecil kemudian dimasukan pada himpuan T. Setelah itu akan dilakukan perulangan/iterasi sebanyak n-2 untuk mencari sisi dengan bobot terkecil pada himpunan E yang bersisian dengan simpul yang telah dimasukan pada T. Hasil pencarian tersebut kemudian digabungkan atau ditambahkan pada himpunan T.

Pada algoritma Prim diatas tidak ada pengecekan secara eksplisit apakah sisi yang dipilih akan membentuk sirkuit atau tidak. Karena pada algoritma Prim sisi yang dimasukkan ke dalam T harus bersisian dengan sebuah simpul di T. Algoritma Prim juga tidak mampu menentukan sisi mana yang akan dipilih jika mempunyai bobot yang sama maka sisi yang dimasukkan harus terurut dari kecil ke besar.

Apakah mungkin sisi yang bersisian membentuk sirkuit? Mungkin saja. Bagaimana mengetahui bahwa sisi tersebut tidak membentuk sirkuit? Menurut penulis, untuk mengatasi hal tersebut harus dilihat apakah titik ujung dari sisi tersebut sudah ada dalam T atau belum. Jika sudah ada maka tidak boleh memilih sisi tersebut karena pasti akan membentuk sirkuit. Apakah hal itu akan membuat algoritma Prim hampir sama dengan algoritma Kruskal? Tentu saja berbeda. Perbedaannya dengan algoritma Kruskal adalah sisi yang dimasukkan pada algoritma Prim harus bersisian sehingga akan meminimalkan waktu sedangkan pada algoritma Kruskal semua sisi boleh dimasukkan asal tidak membentuk sirkuit.

Algoritma Prim akan selalu berhasil menemukan pohon merentang minimum tetapi pohon merentang yang dihasilkan tidak selalu unik, maksudnya mungkin akan lebih dari 1 pohon yang dihasilkan dengan bobot yang sama hanya bentuknya saja yang berbeda.

4.2 Komplektivitas Algoritma Prim

Algoritma Prim mempunyai kompleksitas waktu asimptotik $O(n^2)$. Artinya jika dimasukkan sebanyak n sisi akan memerlukaan waktu n^2, karena setiap sisi akan mengecek semua sisi yang bertetangga dengannya.

Pada langkah pertama, algoritma akan mencari sebanyak n buah sisi. Pada langkah ke-2, algoritma akan mengecek n buah sisi untuk diambil sisi yang bersisian dengan bobot terkecil. Demikian pula untuk langkah ke-3 dan seterusnya. Maka jika ada n buah sisi yang dicari, algoritma akan memerlukan waktu sebanyak :

$(n-1) \times n = n^2 - n$,

sehingga kompleksitas dari algoritma adalah $O(n^2)$.

Simpul dan bobot pada graf direpresentasikan ke dalam matriks. Untuk mencari suatu sisi, maka algoritma Prim akan mencari kedua arah yaitu baris dan kolom graf G kemudian akan dilihat bobotnya.

4.3 Analisa Algoritma Kruskal

```
function Kruskal (input E : himpunan_sisi)
      ->himpunan_sisi
{menghasilkan pohon merentang minimum.
Asumsi : sisi-sisi di dalam graf sudah
terurut menaik berdasarkan bobotnya, dari
bobot kecil ke bobot besar}

Deklarasi
      T : himpunan_sisi
      e : sisi

Algoritma
      S <- {}
      while (jumlah sisi di dalam S<n - 1)
          and (E!={}) do
            e <- sisi yang mempunyai bobot
                 terkecil di dalam E
            E <- E -{e}
            if T U e bukan sirkuit then
                  S <- S U {e}
            endif
      endwhile
      {T(V,S) pohon merentang} or (E = {})

      return S
```

Analisa:

Algoritma ini lebih sederhana jika dilihat dari konsepnya namun lebih sulit dalam implementasinya. Idenya adalah mendapatkan satu demi satu sisi mulai dari yang berbobot terkecil untuk membentuk tree; suatu sisi walaupun berbobot kecil tidak akan diambil jika membentuk siklik dengan sisi yang sudah termasuk dalam tree. Yang menjadi masalah dalam implementasinya adalah keperluan adanya pemeriksaan kondisi siklik tersebut.

Salah satu pemecahaannya adalah dengan subsetting yaitu pembentukan subset-subset yang disjoint dan secara bertahap dilakukan penggabungan atas tiap dua subset yang berhubungan dengan suatu sisi dengan bobot terpendek.

Untuk memeriksa apakah sisi membentuk sirkuit pada pohon yang telah terbentuk, kita dapat melekukan cara berikut ini:

1. Simpul-simpul pada pohon yang sama didalam hutan rentang dikelompokkan ke dalam sebuah himpunan. Tiap simpul didalam himpunan yang sama membentuk pohon (yang merupakan komponen pohon merentang).
2. Bila sisi akan dimasukkan ke dalam pohon, periksa apakah simpul terletak didalam himpunan yang sama. Jika ya, maka sisi tersebut di tolak, tetapi jika tidak, maka sisi diterima, lalu dibentuklah himpunan yang baru.

4.4 Komplektivitas Algoritma Kruskal

Anggaplah E adalah jumlah sisi pada graf G dan V adalah jumlah simpul. Algoritma kruskal memiliki kompleksitas asimptotik $O(E \log E)$, atau ekuivalen dengan $O(E \log V)$. Keduanya dinyatakan equivalen karena:

- E paling banyak adalah V^2 dan $\log V^2 = 2 \times \log V$ is O(log V).
- Apabila kita mengabaikan simpul terpencil, $V \leq 2E$, jadi log V dapat disubstitusi dengan $O(\log E)$

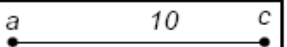
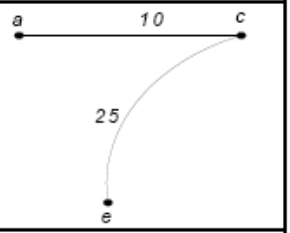
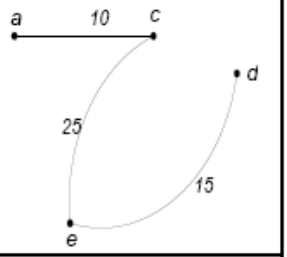
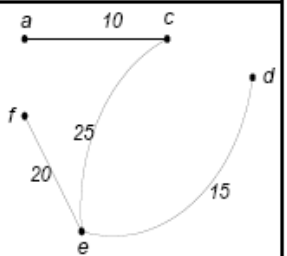
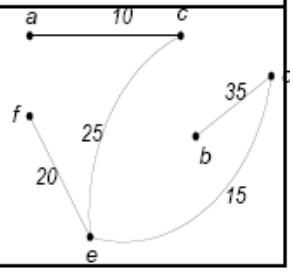
Kita dapat menerima batasan karena : Pertama urutkan sisi berdasarkan bobot menggunakan pengurutan perbandingan (*comparison sort*) dengan waktu $O(E \log E)$; hal ini menyebabkan langkah "menghilangkan sisi dengan bobot minimum dari S" untuk dioperasikan pada waktu konstan. Kemudian kita gunakan sebuah struktur data himpunan disjoint untuk menjaga simpul mana yang terdapat pada suatu komponen. Kita membutuhkan operasi sebanyak $O(E)$, dua buah operasi pencarian, dan mungkin sebuah penggabungan untuk setiap sisi. Sebuah struktur data himpunan disjoint sederhana seperti himpunan hutan disjoint dengan penggabungan berdasarkan rangking dapat menjalankan operasi $O(E)$ pada waktu $O(E \log V)$. Dengan demikian total waktu asimptotiknya menjadi $O(E \log E) = O(E \log V)$.

5. STUDI KASUS

5.1 Studi Kasus Dengan Algoritma Prim

Studi kasus ini menggunakan gambar graf G pada gambar 2.4. Lalu kita akan lihat, apa hasil dari algoritma Prim.

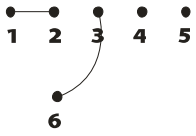
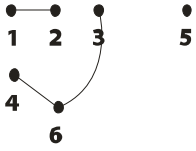
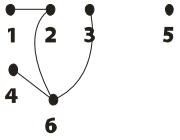
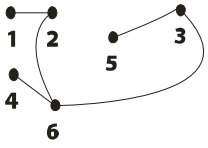
Tabel 1 Tabel Pembentukan Pohon Merentang Minimum dengan Algoritma Prim

Langkah	Sisi	Bobot	Pohon Merentang
1	(a,c)	10	
2	(c,e)	25	
3	(d,e)	15	
4	(e,f)	20	
5	(b,d)	35	

5.2 Studi Kasus Dengan Algoritma Kruskal

Untuk membandingkan hasil pembentukan pohon merentang minimum yang dibentuk dengan algoritma kruskal dan algoritma prim maka pada studi kasus ini digunakan graf yang sama dengan studi kasus sebelumnya yaitu dengan menggunakan graf pada gambar 2.4. Kita akan lihat, apa hasil dari algoritma Kruskal.

Tabel 2 Tabel Pembentukan Pohon Merentang Minimum dengan Algoritma Kruskal

Langkah	Sisi	Bobot	Pohon Merentang
0			
1	(1,2)	10	
2	(3,6)	15	
3	(4,6)	20	
4	(2,6)	25	
5	(1,4)	30	ditolak
6	(3,5)	35	

6. KESIMPULAN

Algoritma Prim dan algoritma Kruskal merupakan contoh algoritma yang digunakan untuk memecahkan permasalahan yang berhubungan dengan graf dengan membangun graf menjadi pohon merentang minimum. Kedua algoritma tersebut termasuk algoritma yang menerapkan strategi Greedy karena pada setiap langkah algoritmanya akan mencari solusi yang paling optimal sehingga dapat dikatakan selalu memenuhi lokal optimal tetapi belum tentu menghasilkan global optimal.

Algoritma Prim pasti menghasilkan pohon merentang minimum meskipun tidak selalu unik. Sisi graf yang dimasukkan untuk menjadi kandidat sisi pohon merentang minimum adalah sisi yang bersisian dengan simpul sebelumnya. Sementara itu algoritma Kruskal membangun pohon merentang minimum dengan mengurutkan terlebih dahulu sisi dengan bobot minimum hingga bobot maksimum kemudian dilakukan penyaringan terhadap sisi yang membentuk sirkuit.

7. REFERENSI

[1] Munir, Rinaldi. 2006. *"Matematika Diskrit"*. Bandung. Penerbit ITB

[2] Munir, Rinaldi. 2007. *"Strategi Algoritmik"*. Bandung. Penerbit ITB

[3] http://en.wikipedia.org/wiki/Kruskal%27s_algorithm.htm waktu akses : 19 Mei 2007 13.00

[4] http://en.wikipedia.org/wiki/Prim%27s_algorithm.htm waktu akses : 19 Mei 2007 13.00