Penggunaan Algoritma Greedy dalam Membangun Pohon Merentang Minimum

Gerard Edwin Theodorus - 13507079
Jurusan Teknik Informatika ITB, Bandung, email: if17079@students.if.itb.ac.id

Abstract *– Makalah ini membahas tentng penggunaan algoritma greedy dalam membangun Pohon Merentang Minimum (Minimum Spanning Tree/MST). Implementasi algoritma greedy dalam membangun MST ini menghasilkan macam-macam algoritma. Beberapa diantaranya adalah algoritma Prim, algoritma Kruskal, dan algoritma Reverse-Delete (Hapus-Mundur).*

Prinsip utama dari algoritma greedy adalah mencari nilai optimum lokal di setiap daerah. Dengan begitu, diharapkan akan diperoleh nilai optimum global, dalam hal ini akan diperoleh pohon merentang yang minimum.

Ketiga algoritma untuk mencari Pohon Merentang Minimum didasarkan pada algoritma greedy ini, hanya penerapannya saja yang berbeda.

Kata Kunci*: algoritma greedy, Pohon Merentang Minimum, Prim, Kruskal, Reverse-Delete*

1. PENDAHULUAN

Graf merupakan salah satu materi yang penting dalam Struktur Diskrit karena memiliki banyak aplikasi dalam kehidupan sehari-hari. Salah satu turunan dari graf adalah pohon. Pohon merupakan sebuah graf tak-berarah terhubung yang tidak mengandung sirkuit. [1]

Dalam makalah ini yang akan dibahas adalah salah satu jenis pohon yaitu pohon merentang. Pohon merentang minimum sendiri adalah salah satu jenis dari pohon merentang yang dibentuk dari graf berbobot.

Graf dan pohon memiliki banyak aplikasi dalam kehidupan sehari-hari. Aplikasi graf misalnya : untuk merepresentasikan ikatan kimia senyawa karbon, memodelkan rangakaian listrik. Aplikasi pohon misalnya : memodelkan sistem pengarsipan komputer, pencarian (dengan binary search tree), pohon keputusan, dan sebagainya.

Aplikasi pohon merentang minimum contohnya dalam permasalahan penentuan jalur (jalur kereta api, perkabelan, atau jaringan (*network*). Biasanya dari satu tempat ke tempat lain hanya diperlukan satu jalur, dan jalur tersebut diharapkan memiliki biaya yang minimum untuk pemasangannya. Dalam makalah ini akan dibahas algoritma-algoritma yang dapat dipakai untuk mencari pohon merentang minimum tersebut, juga pembuktian atau hal-hal yang mendasari algoritma tersebut serta contoh penggunaan masing-masing algoritma tersebut.

2. DASAR TEORI

Sebelum membahas lebih lanjut mengenai algoritma greedy dan penggunaannya dalam mencari pohon merentang minimum akan dijelaskan terlebih dahulu beberapa teori dasar mengenai graf dan pohon yang terkait dengan permasalahan tersebut agar isi makalah ini dapat lebih dipahami.

2.1. Graf [1]

2.1.1 Definisi Graf

Sebuah graf G didefinisikan sebagai pasangan himpunan (V,E), yang dalam hal ini V adalah himpunan tidak-kosong dari simpul (*vertice*) dan E adalah himpunan dari sisi (*edge*) yang menghubungkan sepasang simpul. Dari definisi ini terlihat bahwa graf memiliki minimal satu simpul dan sebuah graf dapat tidak memiliki sisi (hanya memiliki simpul).

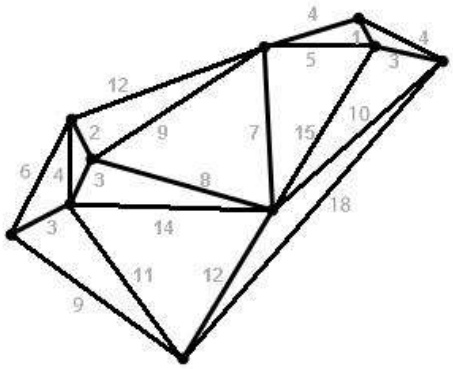
2.1.2 Jenis Graf

Graf dapat dibagi menjadi beberapa jenis berdasarkan beberapa sifat tertentu.
Berdasarkan sisinya :
Graf sederhana – tidak memiliki sisi ganda maupun gelang.
Graf tak-sederhana – mengandung sisi ganda atau gelang (*loop*).
Berdasarkan jumlah simpul :
Graf berhingga – jumlah simpulnya berhingga
Graf tak-berhingga – jumlah simpulnya tak-berhingga
Berdasarkan orientasi arah sisi :
Graf tak-berarah – sisi graf tidak memiliki orientasi arah $\{(v_j,v_k) = (v_k,v_j)$ merupakan sisi yang sama$\}$
Graf berarah – sisi graf memiliki orientasi arah dari simpul asal ke simpul terminal, sehingga $(v_j,v_k) \neq (v_k,v_j)$

2.1.3 Terminologi Dasar

Beberapa terminologi dasar mengenai graf (tak-berarah) yang terkait yaitu :
Bertetangga – v_j bertetangga dengan v_k jika ada sisi (v_j,v_k) pada graf G.
Bersisisan – (v_j,v_k) dikatakan bersisian dengan simpul v_j dan v_k
Derajat – banyaknya sisi yang bersisian dengan suatu simpul.
Lintasan – barisan berselang-seling simpul-simpul dan sisi-sisi dari simpul awal v_o ke simpul tujuan v_n dalam

sebuah graf G.
Siklus/Sirkuit – Lintasan yang berawal dan berakhir di simpul yang sama.
Terhubung – Graf G dikatakan terhubung jika terdapat lintasan untuk setiap v_i dan v_j dalam V.
Upagraf/Subgraf – $G_1 = (V_1,E_1)$ adalah upagraf dari G = (V,E) jika $V1 \subseteq V$ dan $E_1 \subseteq E$.
Upagraf Merentang – Upagraf yang mengandung semua simpul dari graf G ($V_1 = V$).
Cut-set – Cut-set dari graf terhubung G adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung.
Graf berbobot – Graf yang sisi-sisinya memiliki harga / bobot. Graf berbobot bisa berarah atau tak-berarah.



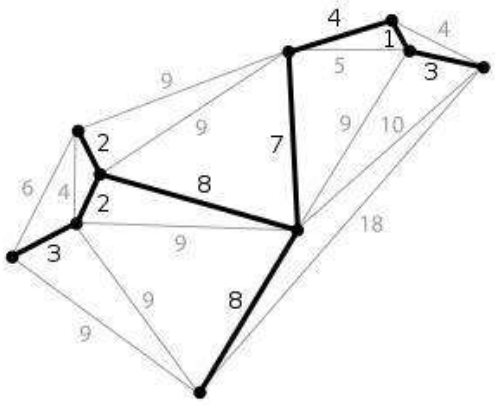
Gambar 2.1 : Contoh graf berbobot terhubung

2.2 Pohon [1]
2.2.1 Definisi Pohon
Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Sehingga setiap pasang simpul pada pohon P terhubung dengan lintasan tunggal.

2.2.2 Pohon Merentang
Pohon merentang $T = (V_1,E_1)$ adalah semua upagraf merentang dari sebuah graf terhubung G = (V,E) yang setiap sisinya terhubung dan tidak mengandung sirkuit. Jadi $V_1 = V$ dan $E_1 \subseteq E$.
Sebuah pohon merentang T dapat dibuat dari sebuah graf terhubung G dengan cara menghapus sebuah sisi dari setiap sirkuit yang terdapat dalam graf sampai tidak terdapat sirkuit lagi di dalam graf atau dengan kata lain menjadi pohon merentang.

2.2.3 Pohon Merentang Minimum
Pohon merentang yang dibentuk dari sebuah graf G yang berbobot dikatakan memiliki bobot sebesar jumlah dari bobot setiap sisinya. Pohon Merentang Minimum atau *Minimum Spanning Tree (MST)* adalah pohon merentang yang memiliki bobot minimum.
Seperti telah disebutkan sebelumnya pohon merentang ini memiliki banyak aplikasi dalam praktek, umumnya dalam pembangunan jalur, misalnya dalam pembangunan jalur kereta api.

Graf berbobot seperti telah dijelaskan bisa memiliki arah maupun tidak, tetapi sesuai definisi pohon, pohon merentang minimum hanya bisa dibentuk dari graf tak-berarah. Graf berbobot berarah umumnya digunakan dalam persoalan mencari lintasan terpendek (*shortest path*). Pencarian lintasan terpendek ini menggunakan algoritma Djikstra yang juga didasarkan pada algoritma greedy, tetapi algoritma ini tidak akan dibahas dalam makalah ini.



Gambar 2.2 : Contoh pohon merentang minimum (diperoleh dari Gambar 2.1)

3. PEMBAHASAN

3.1 Algoritma Greedy [3]
Algoritma greedy adalah setiap algoritma yang mencari nilai optimum untuk setiap tahap, sampai tahap terakhir. Dengan mencari nilai optimum lokal, diharapkan akan diperoleh nilai optimum global. Misalnya, dalam permasalahan TSP (Travelling Salesman Problem), berdasarkan algoritma greedy, yang dilakukan adalah : ”Pada setiap kota (simpul) cari kota terdekat yang belum dikunjungi (bobot sisi minimum).”
Jadi, bisa dikatakan bahwa prinsip utama algoritma greedy adalah memilih pilihan yang terbaik pada tahap tersebut.

Ada 5 elemen utama yang menyusun algoritma greedy, yaitu :
1. Himpunan kandidat (*candidate set*), yang akan digunakan untuk mencari solusi
2. Fungsi seleksi (*selection function*), yang akan memilih/menyeleksi kandidat yang dianggap sebagai nilai optimum lokal.
3. Fungsi kelayakan (*feasibility function*), yang akan menguji atau menentukan apakah kandidat yang dipilih termasuk dalam himpunan solusi.
4. Fungsi objektif (*objective function*), yang memberikan/memasukkan nilai ke dalam solusi atau solusi sementara (*partial solution*).
5. Fungsi solusi (*solution function*), yang menandakan bahwa himpunan solusi (*solution set*) sudah diperoleh.

Algoritma greedy memiliki beberapa kelemahan. Yang pertama yaitu bahwa tidak semua permasalahan dapat diselesaikan dengan algoritma greedy, karena tidak adanya *backtracking* dalam algoritmanya. Algoritma greedy memilih solusi yang terbaik secar

iteratif dan tidak pernah mengkaji ulang pilihan yang telah dipilih. Kelemahan yang kedua adalah ada permasalahan yang memiliki substruktur non-optimal (*non-optimal substructure*). Sebuah masalah dikatakan memiliki substruktur optimal jika solusi optimal untuk masalah tersebut juga mengandung solusi optimal untuk sub-masalah tersebut. Maka, masalah yang memiliki substruktur non-optimal belum tentu mengandung solusi optimal pada sub-masalahnya dalam solusi yang optimal. Artinya, algoritma greedy tidak berlaku pada masalah-masalah seperti ini, karena hasil yang diperoleh ini belum tentu merupakan solusi yang optimal.

Kedua hal inilah yang menyebabkan algoritma greedy gagal dalam mencari solusi optimum global, karena tidak melihat secara mendalam/detail setiap data yang ada. Namun, untuk masalah-masalah tertentu, algoritma greedy dapat dibuktikan akan menghasilkan solusi global yang optimum. Dalam permasalahan-permasalahan seperti ini, penggunaan algoritma greedy lebih mangkus dibandingkan dengan algoritma lain, misalnya dengan metode *exhaustive* (secara mendalam).

Beberapa contoh permasalahan yang dapat diselesaikan dengan algoritma greedy antara lain : Permasalahan pohon merentang minimum (MST), pencarian lintasan terpendek, dan pencarian pohon Huffman yang optimum.

3.2 Algoritma Greedy untuk membangun Pohon Merentang Minimum (*Minimum Spaning Tree*) [2]

Ada beberapa algoritma greedy yang dapat digunakan untuk menghasilkan pohon merentang minimum, tiga algoritma di antaranya, yang menghasilkan pohon merentang minimum, yaitu :

- Algoritma pertama memulai dengan himpunan sisi dari pohon yang kosong, kemudian menambahkan satu per satu sisi dari himpunan sisi graf E secara menaik (dari sisi yang berbobot paling kecil). Sisi e ditambahkan ke selama tidak terbentuk sirkuit dengan sisi-sisi yang telah ditambahkan sebelumnya. Jika terbentuk sirkuit, maka sisi yang baru ditambahkan tersebut dihapus dan proses dilanjutkan ke sisi berbobot terkecil berikutnya sampai semua sisi telah diproses. Pendekatan semacam ini disebut **Algoritma Kruskal**.

- Algoritma greedy lainnya yang termasuk sederhana adalah algoritma yang mirip dengan Algoritma Djikstra dalam pencarian rute terpendek, tetapi lebih mudah. Algoritma ini memulai dengan salah satu simpul s dan mencoba secara "greedy" untuk menghasilkan pohon dari v ke luar. Pada setiap tahap ditambahkan satu simpul ke himpunan simpul S (yang sekarang berisi s) yang terhubung ke simpul s dengan bobot terkecil dan memasukkan sisi tersebut ke dalam pohon merentang yang dibuat. Algoritma ini disebut **Algoritma Prim**.

- Algoritma yang ketiga adalah algoritma greedy yang seperti proses kebalikan dari algoritma Kruskal. Secara spesifik, algoritma in dimulai dari graf G = (V,E) yang lengkap kemudian menghapus satu per satu sisi dalam E secara menurun (mulai dari yang terbesar). Sisi e dihapus selama graf tidak menjadi terputus/tidak terhubung. Jika graf menjadi terputus, sisi e tidak dihapus, dan proses dilanjutkan ke sisi berbobot terbesar berikutnya hingga semua sisi selesai diproses. Algoritma ini tidak dinamai dengan nama tertentu, tetapi umumnya disebut sebagai **Algoritma Reverse-Delete**.

3.2.1 Algoritma Prim

Langkah-langkah untuk membentuk Pohon Merentang Minimum berdasarkan 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.

Jumlah langkah seluruhnya dalam algoritma Prim berdasarkan langkah-langkah di atas adalah (n-2) + 1 = n – 1 langkah.

Jika dituliskan dalam notasi *Pseudo-code*, algoritma Prim dapat dituliskan sebagai berikut :

```
procedure Prim(input G : graf,
output T : pohon)
{ Membentuk pohon merentang
minimum T dari graf terhubung-
berbobot G. }

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 simpul di T
    T ← T ∪ {(u,v)}
 endfor
```

Algoritma 3.1 *Pseudocode* untuk algoritma Prim

Kompleksitas waktu untuk algoritma Prim ini adalah $O(v^2)$ jika diimplementasikan dengan menggunakan array/tabel dan O(e log v) dengan menggunakan *binary heap*.

3.2.2 Algoritma Kruskal

Langkah-langkah pembentukan MST berdasarkan algoritma Kruskal yaitu :

1. Mengurutkan sisi-sisi dari graf secara menaik berdasarkan bobotnya.
2. T masih kosong.
3. Pilih sisi (u,v) dengan bobot minimum yang tidak membentuk sirkuit di T. Tambahkan (u,v) ke dalam T.
4. Ulangi langkah 3 sebanyak n-1 kali.

Notasi *pseudo-code* untuk Algoritma Kruskal dapat dituliskan sebagai berikut :

```
procedure Kruskal(input G :
graf, output T : pohon)
{ Membentuk pohon merentang
minimum T dari graf terhubung }
Deklarasi
  i, p, q, u, v : integer

Algoritma
  Mengurutkan sisi-sisi graf
   secara menaik berdasarkan
   bobotnya

   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,v)}

    endif

  endwhile
```

Algoritma 3.2 *Pseudocode* untuk algoritma Kruskal

Untuk algoritma Kruskal kompleksitas waktunya adalah O(e log v).

3.2.3 Algoritma Reverse-Delete

Algoritma ini bisa dikatakan kebalikan dari Algoritma Kruskal. Beberapa sumber mengatakan bahwa algoritma ini adalah versi asli dari algoritma Kruskal. Langkah-langkah yang dilakukan dalam algoritma ini hampir sama dengan algoritma Kruskal. Langkah-langkah untuk Algoritma Reverse-Delete ini, yaitu :

1. Mengurutkan sisi-sisi graf secara menurun berdasarkan bobotnya (dari bobot terbesar ke bobot terkecil).
2. Graf G disalin ke dalam T.
3. Pilih sisi (u,v) dengan bobot maksimum yang tidak menyebabkan T menjadi tidak terhubung. Hapus (u,v) dari T.
4. Ulangi langkah 3 sebanyak n-1 kali.

Algoritma ini jika dituliskan dalam *pseudo-code*, kurang lebih seperti berikut :

```
procedure Reverse-Delete(input G
: graf, output T : pohon)
{ Membentuk pohon merentang
minimum T dari graf terhubung }
Deklarasi
  i, p, q, u, v : integer

Algoritma
Mengurutkan sisi-sisi graf
secara menurun berdasarkan
bobotnya

T ← {E,V}

while jumlah sisi T > n-1 do

    Pilih sisi (u,v) dari E
     yang bobotnya terbesar

    if (u,v) tidak memutus T
       then

       Hapus (u,v) dari T

    endif

endwhile
```

Algoritma 3.3 *Pseudocode* untuk algoritma Reverse-Delete

Algoritma Reverse-Delete ini juga dapat dibuktikan memiliki kompleksitas waktu O(e log v).

3.3 Pembuktian Kebenaran Algoritma Greedy dalam Membangun Pohon Merentang Minimum

Ada 2 sifat yang menjadi dasar/membuktikan kebenaran dari algoritma Greedy. Untuk menyederhanakan, diasumsikan bahwa setiap sisi memiliki bobot (c_e) yang unik. Kedua sifat tersebut yaitu :

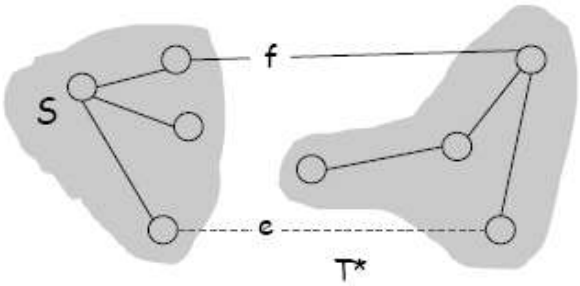
- **Cut property (Sifat jembatan)**
  Misalkan S adalah himpunan bagian dari V, dan e adalah sisi dengan bobot minimum yang salah satu ujungnya berada di S (dapat dilihat dari Gambar 3.1 bahwa e merupakan salah satu cut-set). Maka e termasuk dalam MST T*.

  Pembuktian :
  - Asumsikan e tidak termasuk dalam MST T*.
  - Jika e ditambahkan ke dalam T* akan terbentuk sirkuit, katakanlah C.
  - e termasuk dalam sirkuit C dan cut-set D yang terhubung ke S.
  - Asumsikan ada sisi lain (misalkan f) yang juga termasuk dalam sirkuit C dan cut-set D.
  - Jika f diputus, akan terbentuk pohon merentang T' yang mengandung e.
  - Karena $c_e < c_f$ maka bobot T' < T*
  - Maka T* bukan MST dan asumsi pertama salah.
  - Jadi, MST pasti mengandung e (sisi berbobot minimum).

- **Cycle property (Sifat siklus)**
  Misalkan C adalah sebuah sirkuit dalam graf G dan f adalah sisi berbobot maksimum dalam sirkuit C. Maka MST T* tidak mengandung sisi f.

  Pembuktian :
  - Asumsikan f termasuk dalam MST T*.
  - Jika f dihapus dari T*, T* akan terputus dengan salah satu bagiannya adalah S.
  - f termasuk dalam sirkuit C dan cut-set D yang terhubung ke S.
  - Asumsikan ada sisi lain (misalkan e) yang juga termasuk dalam sirkuit C dan cut-set D.
  - Jika e ditambahkan, akan terbentuk pohon merentang lain, misalkan T' yang mengandung e.
  - Karena $c_e < c_f$ maka bobot T' < T*
  - Maka T* bukan MST dan asumsi pertama salah.

Jadi, MST pasti tidak mengandung f (sisi berbobot maksimum).



Gambar 3.1 Graf untuk pembuktian kebenaran algoritma Greedy

Dengan 2 sifat di atas yang sudah terbukti benar, maka algoritma greedy terbukti benar karena dalam algoritma greedy yang dicari adalah sisi yang minimum, dan menurut cut property sisi tersebut termasuk dalam MST.

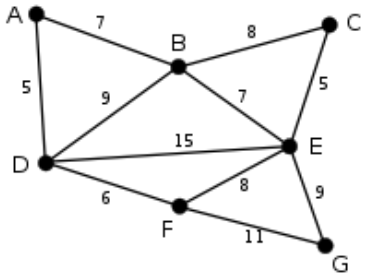
Dengan demikian terbukti pula kebenaran Algoritma Prim dan Algoritma Kruskal, karena keduanya memasukkan semua sisi berbobot minimum dari graf G. Berdasarkan *cut property* sisi berbobot minimum pasti termasuk dalam MST. Maka kedua algoritma tersebut pasti akan menghasilkan MST.

Algoritma Reverse-Delete juga terbukti kebenarannya berdasarkan *cycle property*. Berdasarkan sifat tersebut sisi berbobot maksimum tidak pernah termasuk dalam MST. Karena algoritma Reverse-Delete menghapus semua sisi berbobot maksimum dari graf, maka algoritma ini juga menghasilkan sebuah MST.

3.4 Contoh Penggunaan Algoritma Greedy

Contoh penggunaan algoritma greedy akan ditunjukkan dengan proses pembentukan pohon merentang minimum dari sebuah graf G menggunakan ketiga algoritma yang ada.

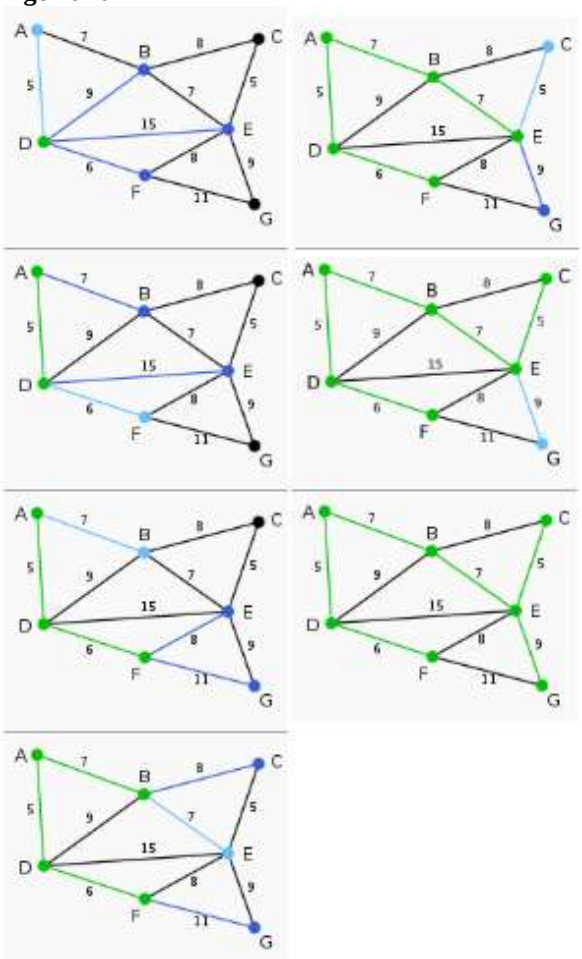
Misalkan diberikan sebuah graf G (V,E) seperti gambar di bawah ini, tentukanlah pohon merentang minimum T dari graf G tersebut!



Gambar 3.2 Graf G untuk contoh pembentukan pohon merentang minimum

Langkah-langkah pembentukan pohon merentang minimum :

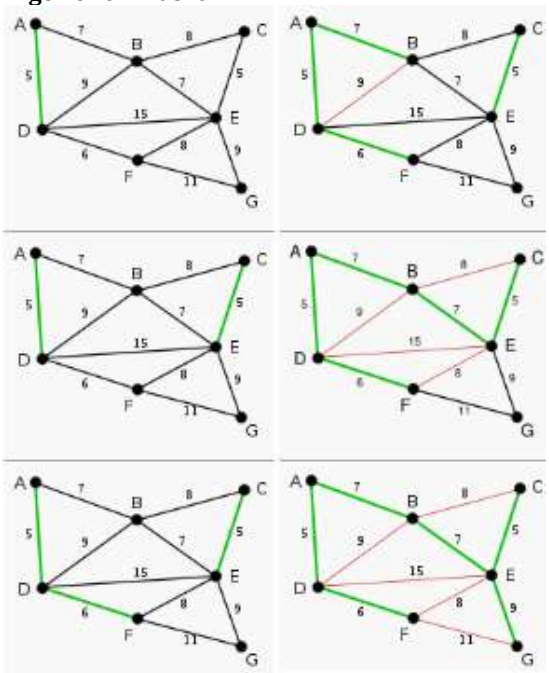
Algoritma Prim



Gambar 3.3 Tahap-tahap pembentukan pohon merentang minimum dengan algoritma Prim

Keterangan :
Kiri = Proses 1-4
Kanan = Proses 5-7
= simpul dan sisi yang dimasukkan dalam T.
= sisi berbobot minimum yang akan dimasukkan ke dalam T.
= sisi-sisi yang bersisian dengan simpul di T.

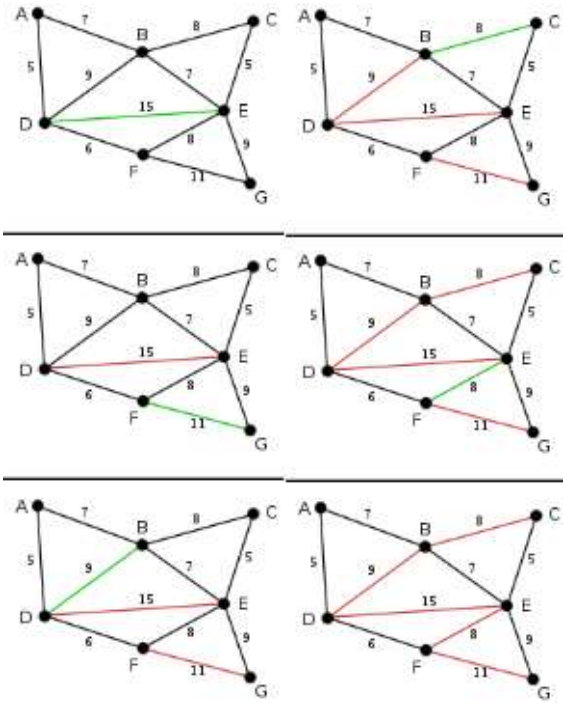
Algoritma Kruskal



Gambar 3.4 Tahap-tahap pembentukan pohon merentang minimum dengan algoritma Kruskal

Keterangan :
Kiri = Proses 1-3
Kanan = Proses 4-6
= sisi berbobot minimum yang dimasukkan dalam T.
= sisi berbobot minimum yang tidak dimasukkan ke dalam T karena membentuk siklus.

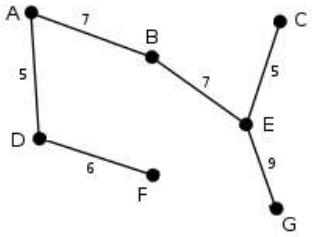
Algoritma Reverse-Delete



Gambar 3.5 Tahap-tahap pembentukan pohon merentang minimum dengan algoritma Reverse-Delete

Keterangan :
Kiri = Proses 1-3
Kanan = Proses 4-6
= sisi berbobot maksimum yang dihapus dari T.
= sisi berbobot maksimum yang telah dihapus dari T.

Dapat dilihat bahwa ketiga algoritma menghasilkan pohon merentang minimum (MST) yang sama. Artinya, solusi MST untuk contoh graf pada Gambar 3.2 adalah solusi yang unik, yaitu pohon merentang minimum seperti di bawah ini.



Gambar 3.6 Pohon merentang minimum untuk graf pada Gambar 3.2

4. KESIMPULAN

Kesimpulan yang dapat dapat diperoleh dari pembahasan dalam makalah ini, adalah :

1. Algoritma greedy dapat digunakan untuk memecahkan masalah-masalah tertentu secara mangkus, misalnya masalah *Minimum Spanning Tree* (MST).
2. Beberapa algoritma greedy untuk membangun MST, yaitu Algoritma Prim, Algoritma Kruskal, dan Algoritma Reverse-Delete. Kompleksitas waktu untuk ketiganya adalah O(e log v).

DAFTAR REFERENSI

[1] Rinaldi Munir, *Struktur Diskrit*, Informatika, 2008, Bandung, hal. VIII-1–VIII-18, IX-1–IX-10.

[2] Jon Kleinberg, Eva Tardos, "The Minimum Spanning Tree Problem", *Algorithm Design*, Pearson-Addison Wesley, 2005, hal. 142-151.

[3] *Greedy algorithm*, 2009. http://en.wikipedia.org/wiki/Greedy_algorithm Tanggal akses : 1 Januari 2009 18:00

[4] *Reverse-delete algorithm*, 2008. http://en.wikipedia.org/wiki/Reverse-delete_algorithm Tanggal akses : 1 Januari 2009 18:00

[5] *Kruskal's algorithm*, 2008. http://en.wikipedia.org/wiki/Kruskal%27s_algorithm Tanggal akses : 1 Januari 2009 18:00

[6] *Prim's algorithm*, 2008. http://en.wikipedia.org/wiki/Prim%27s_algorithm Tanggal akses : 1 Januari 2009 18:00