IMPLEMENTASI GRAF DENGAN MENGGUNAKAN STRATEGI *GREEDY*

Arief Latu Suseno – NIM : 13505019

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15019@students.if.itb.ac.id

Abstrak

Graf merupakan salah satu metode untuk mencari solusi dari permasalah diskrit yang ditemui dalam dunia nyata. Graf memiliki banyak konsep, salah satu diantaranya yang paling penting dan sangat populer adalah konsep pohon karena konsep ini mampu mendukung penerapan graf dalam berbagai bidang ilmu dan dapat memecahkan masalah ataupun membuat sebuah keputusuan yaitu dengan cara membangun graf menjadi pohon merentang minimum (Minimum Spanning Tree / MST). Salah satu strategi yang dipakai untuk menghasilkan pohon merentang yang minimum adalah dengan menggunakan algoritma Greedy. Algoritma greedy merupakan metode yang paling populer untuk menemukan solusi optimum dalam persoalan optimasi (optimization problem) dengan membentuk solusi langkah per langkah (step by step). 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 "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal (local optimum) pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi optimum global (global optimum).

Kata kunci: Strategi Greedy, Travelling Sallesperson Problem, Minimum Spanning Tree, Minimasi Waktu dalam Sistem

1. Pendahuluan

Teori graf adalah ilmu yang berkembang sangat pesat, bahkan dalam perkembangannya dapat disejajarkan dengan ilmu Aljarbar yang lebih dahulu berkembang. Ilmu Aljabar (abstrak) yang merupakan bagian dari ilmu matematika, pada dasarnya berkembang pesat karena berhubungan dengan himpunan, operasi, dan sifat struktur-struktur di dalamnya.

Keunikan teori Graf adalah kesederhanaan pokok bahasan yang dipelarjarinya, karena dapat disajikan sebagai titik (verteks) dan garis (edge). Meskipun pokok bahasan dari topik-topik teori graf sangat sederhana, tapi isi di dalamnya belum tentu sesederhanan itu. Kerumitan demi kerumitan masalah-masalah selalu pasti ada dan bahkan sampai saat ini masih ada masalah yang belum terpecahkan.

Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Banyak persoalan pada dunia nyata yang sebenarnya merupakan representasi visual dari graf, misalnya peta. Hal yang dapat digali dari representasi tersebut adalah menentukan jalur terpendek dari satu tempat ke tempat lain, menggambarkan 2 kota yang bertetangga dengan warna yang berbeda pada peta, menentukan tata letak jalur transportasi, dsb. Selain peta, masih banyak hal lain dalam dunia nyata yang merupakan representasi visual dari graf.

Sejumlah masalah yang berkaitan dengan graf yang ditemukan manusia dalam kehidupan nyata menimbulkan penemuan konsep-konsep pemecahan masalah graf, yang dalam makalah ini menggunakan Strategi *Greedy*. Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan yang “tampaknya” memberikan perolehan terbaik, yaitu dengan membuat pilihan **optimum lokal** (*local*

optimum) pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi **optimum global** (*global optimum*).

Secara umum algoritma greedy disusun oleh elemen-elemen berikut :

- Himpunan kandidat.
  Berisi elemen-elemen pembentuk solusi.
- Himpunan solusi
  Berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
- Fungsi seleksi (*selection function*)
  Memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.

Fungsi kelayakan (*feasible*) Memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.

Algoritma Greedy dapat menyelesaikan beberapa masalah dalam kehidupan nyata, yang akan saya bahas dalam makalah ini adalah :

- TSP (Travelling Sallesperson Problem)
- Minimum Spanning Tree (prim's)
- Minimasi Waktu dalam Sistem (Penjadwalan)

2. Dasar Teori - Strategi *Greedy*

2.1 Definisi Strategi *Greedy*

Strategi *greedy* adalah strategi yang memecahkan masalah langkah demi langkah, pada setiap langkah, adapun urutan langkah strategi *Greedy* adalah :

1. mengambil pilihan yang terbaik yang dapat diperoleh saat itu,
2. berharap bahwa dengan memilih solusi optimum lokal, pada setiap langkah akan mencapai solusi optimum global. Strategi *greedy* mengasumsikan bahwa optimum lokal merupakan bagian dari optimum global.

2.2 Skema Umum Strategi *Greedy*

Persoalan optimasi dalam konteks strategi *greedy* disusun oleh elemen-elemen sebagai berikut:

1. Himpunan kandidat, *C*.
   Himpunan ini berisi elemen-elemen pembentuk solusi. Pada setiap langkah, satu buah kandidat diambil dari himpunannya.
2. Himpunan solusi, *S*.
   Merupakan himpunan dari kandidat-kandidat yang terpilih sebagai solusi persoalan. Himpunan solusi adalah himpunan bagian dari himpunan kandidat.
3. Fungsi seleksi – dinyatakan sebagai predikat SELEKSI –
   Merupakan fungsi yang pada setiap langkah memilih kandidat yang paling mungkin untuk mendapatkan solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.
4. Fungsi kelayakan (*feasible*) – dinyatakan dengan predikat LAYAK –
   Merupakan fungsi yang memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar aturan yang ada.
5. Fungsi obyektif, merupakan fungsi yang memaksimumkan atau meminimumkan nilai solusi.

Dalam strategi ini, kita berharap optimum global merupakan solusi optimum dari persoalan. Namun, adakalanya optimum global belum tentu merupakan solusi optimum (terbaik), tetapi dapat merupakan solusi sub-optimum atau pseudo-optimum. Hal ini dapat dijelaskan dari dua faktor berikut:

1. strategi *greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada.
2. pemilihan fungsi SELEKSI: fungsi SELEKSI biasanya didasarkan pada fungsi obyektif (fungsi SELEKSI bisa saja identik dengan fungsi obyektif). Jika fungsi SELEKSI tidak identik dengan fungsi obyektif, kita harus memilih fungsi yang tepat untuk menghasilkan nilai yang optimum.

Karena itu, pada sebagian masalah strategi *Greedy* tidak selalu berhasil memberikan solusi yang benar-benar optimum. Tetapi, strategi *greedy* pasti memberikan solusi yang mendekati (*approximation*) nilai optimum.

3. Pembahasan Penelitian

3.1. Traveling Salesperson Problem

3.1.1. Konsep

Penggambaran yang sangat sederhana dari istilah *Traveling Salesman Problem (TSP)* adalah seorang *salesman* keliling yang harus mengunjungi *n* kota dengan aturan sebagai berikut :

- Ia harus mengunjungi setiap kota hanya sebanyak satu kali.
- Ia harus meminimalisasi total jarak perjalanannya.
- Pada akhirnya ia harus kembali ke kota asalnya.

Dengan demikian, apa yang telah ia lakukan tersebut akan kita sebut sebagai sebuah *tour*. Guna memudahkan permasalahan, pemetaan *n* kota tersebut akan digambarkan dengan sebuah *graph*, dimana jumlah *vertice* dan *edge*-nya terbatas (sebuah *vertice* akan mewakili sebuah kota dan sebuah *edge* akan mewakili jarak antar dua kota yang dihubungkannya). Penanganan problem TSP ini ekuivalen dengan mencari sirkuit Hamiltonian terpendek.

Terdapat berbagai algoritma yang dapat diterapkan untuk menangani kasus TSP ini, mulai dari *exhaustive search* hingga *dynamic programming*. Akan tetapi saat ini yang akan digunakan adalah algoritma *Greedy*.

Strategi *greedy* yang digunakan untuk memilih kota berikutnya yang akan dikunjungi adalah sebagai berikut :
”Pada setiap langkah, akan dipilih kota yang belum pernah dikunjungi, dimana kota tersebut memiliki jarak terdekat dari kota sebelumnya”, berdasarkan aturan tersebut dapat dilihat bahwa *greedy* tidak mempertimbangkan nilai *heuristic* (dalam hal ini bisa berupa jarak langsung antara dua kota).

3.1.2. Model Penelitian

Salah satu algoritma *greedy* yang dapat digunakan untuk menangani problem TSP ini adalah ***Kruskal's Algorithm***, dimana algoritma ini akan mencari sirkuit Hamilton minimum.

Berikut adalah algoritma *Kruskal*

```
void kruskal ( )
{
 int i,x,b[MAX_NODE],top,w,v;
 int min_wt,y,f_wt[MAX_NODE],bentuk;
 node *ptr1;
 f_node *ptr2;
 f_list=NULL;
 for(i=1;i<=totNodes;i++)
  status[i]=unseen;
  x=1;
  status[x]=intree;
  top=0;
  bentuk=0;
  while( (top <= (totNodes-1)) && (!bentuk)){
     ptr1=adj[x];
     while(ptr1!=NULL)
     {
      y=ptr1 → vertex;
      w=ptr1 → weight;
      if((status[y] == fringe) && (w < f_wt[y])){

                b[y]=x;
                f_wt[y]=w;
                }
                else if(status[y]==unseen)
                {
                  status
                  [y]=fringe;
                  b[y]=x;
                  f_wt[y]=w;
                  Insert_Beg(y);
                }
                ptr1=ptr1 → next;
           }
           if
           (f_list==NULL)
           bentuk=1;
           else {
             x=f_list → vertex;
             min_wt=f_wt[x];
             ptr2=f_list->next;
             while(ptr2!=NULL)
             {
              w=ptr2 → vertex;
              if(f_wt[w] < minwt)
              x=w;
              min_wt=f_wt[w];
              ptr2=ptr2 → next;
              }
             del(x);
             status[x]=intree;
             top++;
           }
         }
```

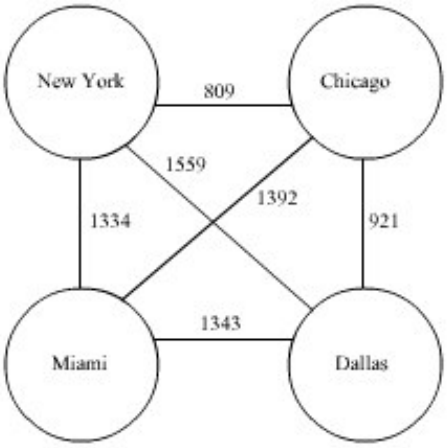
Input :
Graf-berbobot terhubung *G = (V, E)*, dimana *V* = vertex dan *E* = edge.
Output :
Pohon merentang minimum *T = (V, E')*.

3.1.3. Hasil Penelitian

Terdapat empat buah kota (*n* = 4) dengan jarak antar kota adalah sebagai berikut :

	New York	Miami	Dallas	Chicago
New York	-	1334	1559	809
Miami	1334	-	1343	1397
Dallas	1659	1343	-	921
Chicago	809	1397	921	-

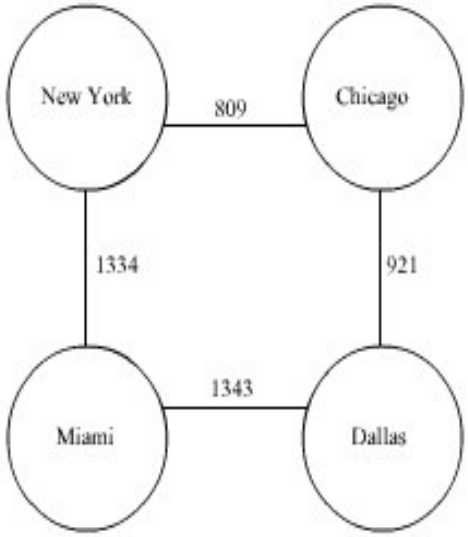
Berdasarkan data jarak di atas, maka graph yang dihasilkan adalah sebagai berikut :



Solving

- Langkah 1 : *New York* menuju *Chicago* (Jarak = 809)
- Langkah 2 : *Chicago* menuju *Dallas* (Jarak = 809 + 921 = 1730)
- Langkah 3 : *Dallas* menuju *Miami* (Jarak = 809 + 921 + 1343 = 3073.
- Langkah 4 : *Miami* menuju *New York* (Jarak = 809 + 921 + 1343 + 1334 = 4407)

Sehingga sirkuit Hamiltonian terpendek yang diperoleh adalah :



3.1.4. Analisis

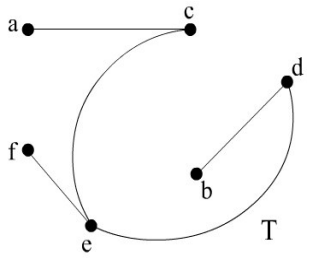
Algoritma ini digunakan untuk memilih node selanjutnya pada graf G yang akan dikunjungi, dimana pada setiap langkah akan dipilih node yang belum pernah dikunjungi dan mempunyai jarak terdekat. Pada setiap langkah tersebut, pilih sisi dari graf G yang mempunyai bobot minimum tetapi sisi tersebut tidak membentuk sirkuit di T.

Kompleksitas :
O(|E| log |E|), dimana |*E*| adalah jumlah sisi di dalam graf *G*.

3.2 Pohon (Tree)

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

Contoh pohon :



Gb.1 Gambar Pohon T

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

T = (V,E)

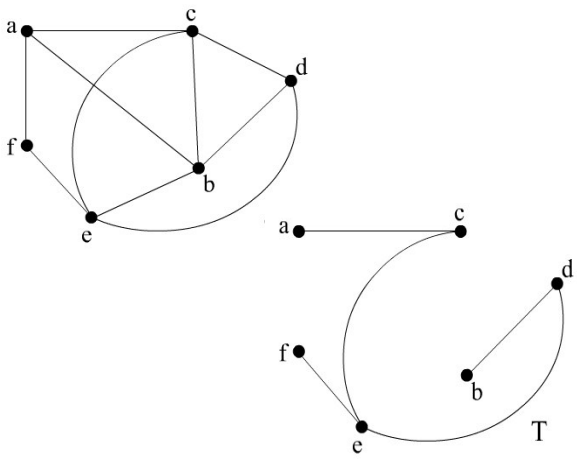
Keterangan :
T : Tree
V : Vertices atau node atau vertex atau simpul, V merupakan himpunan tidak kosong.

V = {v1,v2,…,vn}
E : Edges atau Arcs atau sisi yang menghubungkan simpul
E = {e1,e2,…,en}

3.2.1 Pohon Merentang (Spanning Tree)

Misalkan G = (V,E) adalah graf tak-berarah terhubung yang bukan pohon, yang berarti 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 dan sisi-sisi pada pohon T $\subseteq$ sisi-sisi pada graf G.

Contoh pembentukan pohon merentang :



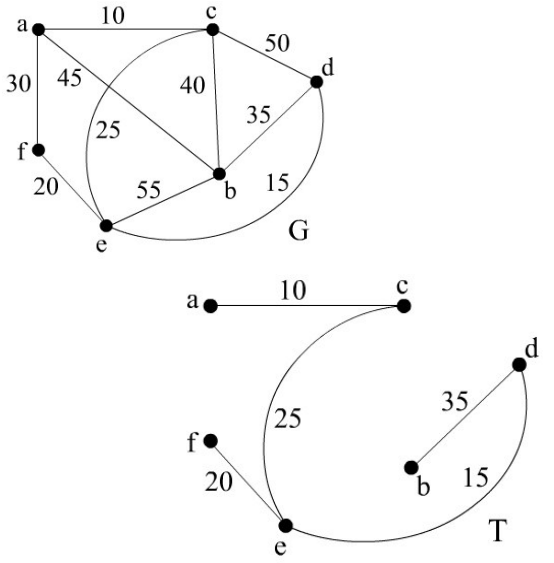
Gambar 2. G adalah Graf, T adalah Pohon Merentang dari Graf G

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

3.2.1.1 Pohon Merentang Minimum (Minimum Spanning Tree)

Jika G pada gambar 2 merupakan graf berbobot, maka bobot pohon merentang T didefinisikan sebagai jumlah bobot semua sisi di T. 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.

Contoh graf dan pohon berbobot :



Gambar 3. G adalah Graf Berbobot, T adalah Pohon Merentang Berbobot dari Graf G

Keterangan :
Dari graf berbobot G, kita harus bisa menentukan apakah T adalah pohon merentang paling minimum dari graf G

3.2.2 Algoritma Greedy

Algoritma *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)*

```
Procedure greedy
(input C: himpunankandidat;
output S : himpunansolusi)
{
  Menentukan solusi optimum dari persoalan
  optimasi dengan algoritma greedy
  Masukan: himpunan kandidat C
  Keluaran: himpunan solusi S
}

Deklarasi x
: kandidat;

Algoritma:
S← {}          { inisialisasi S dengan kosong }
while (belum SOLUSI(S)) and (C ^ { } ) do
   x ← SELEKSI(C); { pilih kandidat dari
C}
   C ← C - {x}
   if LAYAK(S ∪ {x}) then
     S ← S ∪ {x}
   endif
endwhile
{SOLUSI(S) sudah diperoleh or C = {} }
```

Analisa :
Ambil satu kandidat dari himpunan kandidat C lalu masukkan ke x dan kurangi C dengan kandidat tersebut. Kemudian cek apakah layak jika x digabungkan dengan himpunan solusi S? jika layak maka gabungkan x dengan solusi S dan lakukan perulangan hingga C kosong atau solusi S sudah ditemukan.

Layak atau tidaknya x digabung dengan S, melihat tujuan yang ingin dicapai pada kasus yang sedang dipecahkan tetapi tidak melihat apakah hasil tersebut merupakan hasil yang mampu mengoptimalkan tujuan. Yang terpenting ketika langkah tersebut diambil maka setidaknya hasil pada saat itu mendekati tujuan yang ingin dicapai. Misalkan pada kasus mencari jalur terpendek, maka saat menguji apakah x layak digabungkan menjadi solusi S, yang menjadi pertimbangan adalah apakah jika x digabungkan dengan S akan menghasilkan solusi S yang terpendek?

3.2.3 Algoritma Prim

Algoritma Prim merupakan salah satu algoritma yang bekerja secara greedy. Algoritma Prim membentuk MST langkah per langkah. Pada setiap langkah dipilih sisi graf G yang mempunyai bobot minimum dan terhubung dengan MST yang telah terbentuk. Ada tiga langkah yang dilakukan pada algoritma Prim, yaitu :

1. Pilih sisi graf G yang berbobot paling minimum dan masukan ke dalam T
2. Pilih sisi (u,v) yang mempunyai bobot minimum dan bersisian dengan simpul di T, tetapi tidak membentuk sirkuit di T, lalu tambahkan ke dalam T.
3. Ulangi langkah ke-2 sebanyak n-2 kali

3.2.4 Model Penelitian

Strategi greedy yang digunakan :

Pada setiap langkah, pilih sisi dari graf G yang mempunyai bobot minimum dengan syarat sisi tersebut tetap terhubung dengan pohon merentang minimum T yang telah terbentuk.

```
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,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 termasuk salah satu algoritma Greedy sedangkan pendekatan untuk mengecek sirkuit tersebut sama dengan kasus pemecahan masalah sirkuit Hamilton menggunakan algoritma Exhaustive Search yang notabene termasuk algoritma Brute Force, jadi termasuk ke yang mana algoritma Prim yang akan dipakai tersebut? Menurut penulis tetap masuk ke algoritma Greedy karena dalam pengecekan tersebut kita hanya menambahkan prasyarat agar terbentuk pohon merentang.

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.

3.2.5 Kompleksitas Algoritma Prim

Jumlah seluruh langkah pada algoritma Prim adalah

$1 + (v-2) = v-1$

Jumlah iterasi yang dilakukan

Langkah pertama yang pasti diambil

Keterangan :

v : vertex atau simpul dalam pohon merentang

Algoritma Prim mempunyai kompleksitas waktu asimptotik $O(n^2)$. Artinya jika dimasukkan sebanyak n sisi akan memerlukaan waktu n^2. Kenapa 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.

3.3. Minimasi Waktu dalam Sistem (scheduling)

3.3.1. Konsep

Pemilihan srategi *greedy* untuk penjadwalan pelanggan akan selalu menghasilkan solusi optimum.

Keoptimuman ini dinyatakan : Jika $t_1 \leq t_2 \leq \ldots \leq t_n$ maka pengurutan $i_j = j$, $1 \leq j \leq n$ meminimumkan

$T = \sum_{k=1}^{n} \sum_{j=1}^{k} t\, i_j$, untuk semua kemungkinan permutasi i_j.

Masalah penjadwalan pelanggan diatas dalam penyelesaiannya menggunakan Algoritma Greedy, karena mencari solusi yang paling optimum.

Algoritma Greedy membentuk solusi langkah per langkah. Pendekatan yang digunakan di dalam algoritma Greedy adalah membuat pilihan yang memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi optimum secara keseluruhan.

3.3.2. Model Penelitian

procedure PenjadwalanPelanggan(input n:integer)

{
1. *Mencetak informasi deretan pelanggan yang akan diproses oleh server tunggal*
2. *Masukan: n pelangan, setiap pelanggan dinomori 1, 2, …, n*
3. *Keluaran: urutan pelanggan yang dilayani*

}

Deklarasi
i : integer

Algoritma:
{pelanggan 1, 2, ..., n sudah diurut menaik berdasarkan t_i}
for i ← 1 to n do
write('Pelanggan ', i, ' dilayani!')
endfor

3.3.3. Hasil Penelitian

Misal sebuah server (dapat berupa processor, pompa, kasir di bank) mempunyai n pelanggan (customer, client) yang harus dilayani. Waktu pelayanan untuk setiap pelanggan sudah ditetapkan sebelumnya, yaitu pelanggan I membutuhkan waktu t_i. Kita ingin meminimumkan total waktu di dalam sistem.

$T = \sum_{i=1}^{n}$

(waktu dalam system untuk pelanggan i)

Karena jumlah pelanggan adalah tetap, meminimumkan waktu di dalam sistem ekivalen dengan meminimumkan waktu rata-rata.

Misalkan kita mempunyai tiga pelanggan dengan
$t_1 = 5$
$t_2 = 10$
$t_3 = 3$

maka enam urutan pelayanan yang mungkin adalah:

Urutan	*T*	Total
1, 2, 3	5 + (5 + 10) + (5 + 10 + 3)	38
1, 3, 2	5 + (5 + 3) + (5 + 3 + 10)	31
2, 1, 3	10 + (10 + 5) + (10 + 5 + 3)	43
2, 3, 1	10 + (10 + 3) + (10 + 3 + 5)	41
3, 1, 2	**3 + (3 + 5) + (3 + 5 + 10)**	**29**
3, 2, 1	3 + (3 + 10) + (3 + 10 + 5)	34

Bold → (optimal)

3.3.4. Analisis

Strategi *greedy* untuk memilih pelanggan berikutnya adalah:

- Pada setiap langkah, masukkan pelanggan yang membutuhkan waktu pelayanan terkecil di antara pelanggan lain yang belum dilayani.
- Agar proses pemilihan pelanggan berikutnya optimal, maka perlu mengurutkan waktu pelayanan seluruh pelanggan dalam urutan yang menaik.

Kompleksitas :

Jika waktu pengurutan tidak dihitung, maka kompleksitas algoritma *greedy* untuk masalah minimisasi waktu di dalam sistem adalah $O(n)$.

4. Kesimpulan dan Saran Pengembangan

4.1 Kesimpulan

Algoritma Prim merupakan salah satu algoritma yang digunakan untuk memecahkan permasalahan yang berhubungan dengan graf dengan membangun graf menjadi pohon merentang minimum. Algoritma Prim termasuk algoritma Greedy karena pada satu langkah algoritm Prim akan mencari hasil yang paling optimal sehingga dikatakan algoritma Prim selalu memenuhi local 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.

Sedangkan algoritma *Greedy* adalah suatu algoritma yang menyelesaikan masalah langkah demi langkah (step by step) sehingga ketika pada satu langkah telah diambil keputusan maka pada langkah selanjutnya keputusan itu tidak dapat diubah lagi. Jadi algoritma ini menggunakan pendekatan untuk mendapatkan solusi lokal yang optimum dengan harapan akan mengarah pada solusi global yang optimum. Dengan kata lain algoritma *greedy* tidak dapat menjamin solusi global yang optimum.

Penerapan algoritma *greedy* diantaranya dapat dilihat dalam kasus *Travelling Salesperson Problem (TSP), Minimum Spanning Tree (Pim's)* dan Minimasi Waktu dalam Sistem (*scheduling).* Pada ketiga contoh tersebut, kompleksitas tertinggi terdapat pada algoritma *prim's (Minimum Spanning Tree)* dan diikuti dengan algoritma *kruskal's (TSP)* dan yang terkecil adalah pada *scheduling.*

4.2 Saran Pengembangan

- Untuk mempercepat proses, susun sisi-sisi yang bersisian dengan urutan kecil ke besar
- Gunakan pengecekan sirkuit sebagai syarat untuk menjadikan kandidat sisi sebagai sisi pohon merentang

5. Daftar Pustaka

[1] Munir, Rinaldi. 2003. *Matematika Diskrit.* Bandung. Informatika

[2] Munir, Rinaldi. 2004. *Handout Bahan Kuliah ke 1-4.* Bandung. ITB

[3] Strategi *Greedy*. *http://kur2003.if.itb.ac.id/strategialgoritmik*. Diakses tanggal 25 Desember 2006.

[4] *http://www.cs.ui.ac.id/WebKuliah/MD1/* Diakses tanggal 25 Desember 2006.

[5] Foundation Of Computer Science *http://203.130.205.68/dosen/asih/foundationcomputer2/* Diakses tanggal 25 Desember 2006.

[6] Aplikasi Teorema Polya Pada Aplikasi Graf Sederhana *home.unpar.ac.id/~integral/Volume%208/Integral%208%20No.%201/* Diakses tanggal 26 Desember 2006.

[7] Graf
http://www.intanbk.intan.my/kmk/Alatan_KMK/
Diakses tanggal 26 Desember 2006.

[8] Algoritm Prim dengan Strategi Greedy Untuk Membangun Pohon Merentang Minimum
www.stttelkom.ac.id/staf/FAY/kuliah/DAA/20052/Tugas1/pdfs/
Diakses tanggal 26 Desember 2006.

[9] Model Graf Bagi Masalah Pohon Merentas MinimumMenggunakan Algoritma Prim
www.institut.fs.utm.my/~ss/ssu4904/
Diakses tanggal 26 Desember 2006.

[10] Strategi Greedy Pada Kasus Pencarian Lintasan Terpendek
www.stttelkom.ac.id/staf/FAY/kuliah/DAA/20052/Tugas1/pdfs/
Diakses tanggal 26 Desember 2006.

[11] Dasar Teori Graf
http://www.zaki-math.web.ugm.ac.id/matematika/graph_theory/
Diakses tanggal 26 Desember 2006.