PERBANDINGAN KOMPLEKSITAS PENERAPAN ALGORITMA GREEDY UNTUK BEBERAPA MASALAH

Wiradeva Arif Kristawarman – NIM : 13505053

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15053@students.if.itb.ac.id

Abstrak

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*). Algoritma Greedy dapat menyelesaikan beberapa masalah dalam kehidupan nyata, yang akan kami bahas dalam makalah ini adalah :

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

Dari sini kemudian akan dibandingkan kompleksitas tiap algoritma yang digunakan untuk menyelesaikan ketiga problem tersebut.

Kata kunci: algoritma greedy, travelling salesperson problem, prim's, penjadwalan, minimum spanning tree, kompleksitas.

1. Pendahuluan

Persoalan optimasi (*optimization problems*) yaitu persoalan yang menuntut pencarian solusi optimum. Persoalan optimasi ada dua macam:
- Maksimasi (*maximization*)
- Minimasi (*minimization*)

Solusi optimum (terbaik) adalah solusi yang bernilai minimum atau maksimum dari sekumpulan alternatif solusi yang mungkin. Elemen persoalan optimasi:

- Kendala (*constraints*)
- Fungsi Objektif (atau fungsi optimasi)

Solusi yang mengatasi semua kendala disebut **solusi layak** (*feasible solution*). Solusi layak yang mengoptimumkan fungsi optimasi disebut **solusi optimum**. Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi 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*).

Algoritma *greedy* adalah algoritma yang memecahkan masalah langkah per langkah, pada setiap langkah :

- Mengambil pilihan yang terbaik yang dapat diperoleh saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now!*")
- Berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Pada setiap langkah diperoleh **optimum lokal**. Bila algoritma berakhir, kita berharap optimum lokal menjadi **optimum global**.

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, dan yang akan kita bahas dalam makalah kali ini adalah :

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

2. Pembahasan

2.1. Traveling Salesperson Problem

2.1.1. 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).

2.1.2. Model Penelitian

Algoritma *greedy* dapat digunakan untuk menangani problem TSP ini, dimana algoritma ini akan mencari sirkuit Hamilton minimum.

Berikut adalah algoritmanya yang diimplementasikan dalam bahasa C++*:*

```
void TSP ( )
{
 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] < min_wt)
        {
         x=w;
         min_wt=f_wt[w];
        }
        ptr2=ptr2->next;
       }
       del(x);
       status[x]=intree;
       top++;
      }
     }
    }
    for(x=2;x<=totNodes;x++)
     cout<<"("<<x<<","<<b[x]<<")\n";
}
```

Input :
Graf-berbobot terhubung *G = (V, E)*, dimana *V* =vertex dan *E* = edge.

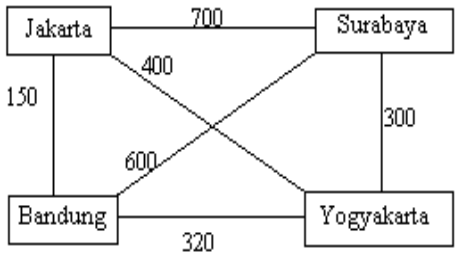
Output :
Sirkuit Hamilton minimum *T = (V, E')*.

2.1.3. Hasil Penelitian

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

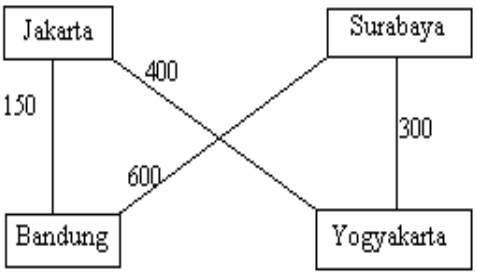
	Jakarta	Surabaya	Bandung	Yogyakarta
Jakarta	0	700	150	400
Surabaya	700	0	600	300
Bandung	150	600	0	320
Yogyakarta	400	300	320	0

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



Solving
- Langkah 1 : *Jakarta* menuju *Bandung* (Jarak = 150)
- Langkah 2 : *Surabaya* menuju *Yogyakarta* (Jarak= 150 + 300 = 450)
- Langkah 3 : *Jakarta* menuju *Yogyakarta* (Jarak =150 + 300 + 400 = 850)
- Langkah 4 : *Bandung* menuju *Surabaya* (Jarak = 150 + 300 + 400 + 600 = 1450)

Sehingga sirkuit Hamiltonian terpendek yang diperoleh adalah :



2.1.4. 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 yang membentuk sirkuit hamilton minimum.

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

2.2. Minimum Spanning Tree

2.2.1. Konsep

Algoritma greedy yang digunakan di sini adalah Algoritma Prim. Algoritma Prim adalah suatu algoritma di dalam teori graph yang menemukan suatu minimum spanning tree untuk suatu graph dengan bobot yang terhubung. Metode ini menemukan suatu subset dari edge yang membentuk suatu pohon yang melibatkan tiap-tiap vertex, di mana total bobot dari semua edge di dalam tree diperkecil. Jika graph tidak terhubung, maka akan hanya menemukan suatu minimum spanning tree untuk salah satu komponen yang terhubung.
Algoritma bekerja sebagai berikut:

- Menciptakan suatu pohon yang terdiri dari vertek tunggal, memilih arbitrarily dari graph
- Menciptakan semua edge di dalam graph
- Loop tiap-tiap edge yang menghubungkan dua vertek di dalam tree
- Memindahkan dari sekumpulan edge yang memiliki bobot minimum yang menghubungkan suatu vertek di dalam tree dengan suatu vertek bukan di dalam tree
- Menambahkan edge ke dalam tree

2.2.2. 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.

Dalam notasi *pseudo-code,* algoritma Prim kita tuliskan sebagai berikut :

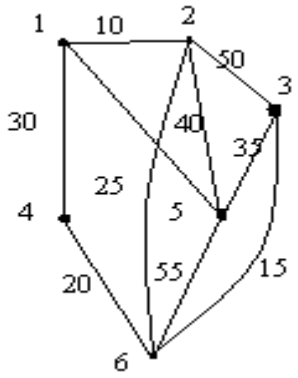
procedure Prim (input G : graf, output T : pohon)
{
1.Membentuk pohon merentang minimum T dari graf terhubung G.
2.Masukan: graf-berbobot terhubung G = (V, E),
yang mana .V .= n
3.Keluaran: pohon rentang minimum T = (V, E')
}

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

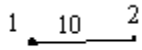
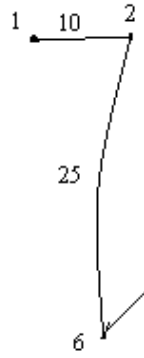
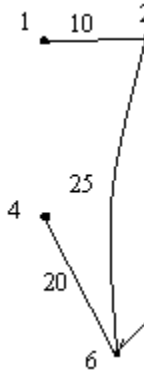
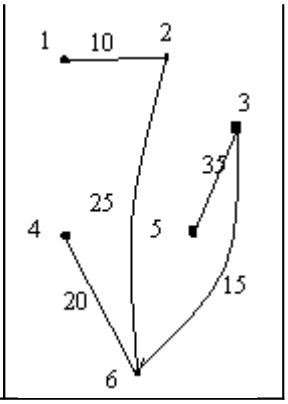
Misalkan model penelitiannya adalah graf sebagai berikut :

	1	2	3	4	5	6
1	0	10	∞	30	45	∞
2	10	0	50	∞	40	25
3	∞	50	0	∞	35	15
4	30	∞	∞	0	∞	20
5	45	40	35	∞	0	55
6	∞	25	15	20	55	0



2.2.3. Hasil Penelitian

Langkah-langkah pembentukan pohon merentang minimum dengan algoritma Prim :

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

2.2.4. Analisis

Misalkan *P* terhubung dalam graph berbobot, pada tiap iterasi algoritma prims, suatu edge harus ditemukan menghubungkan suatu edge di dalam subgraph kepada edge di luar subgraph itu. Saat *P* dihubungkan, akan selalu ada path ke setiap vertek. Keluaran *Y* dari algoritma Prims adalah suatu tree, karena edge dan vertek yang ditambahkan ke *Y* dihubungkan ke edge dan vertek lain pada *Y* dan saat tidak terdapat iterasi. Suatu sirkuit diciptakan saat setiap edge yang ditambahkan menghubungkan dua vertek yang tidak terhubung. Selain itu, *Y* meliputi semua vertek dari *P* sebab *Y* adalah suatu tree dengan n vertek, sama seperti *P*. Oleh karena itu, *Y* adalah suatu spannning tree untuk *P*.

Y1 dijadikan spanning tree minimal untuk *P*. Jika *Y*= *Y1*, kemudian tanda bukti tersebut lengkap. Jika bukan, terdapat suatu edge di *Y* yang tidak ada di *Y1*. Misalkan *e* menjadi edge pertama yang ditambahkan ketika *Y* dibangun. Dan misalkan *V* menjadi satuan vertek *Y- e*. Kemudian satu endpoint *e* ada di *Y* dan yang lain adalah bukan. Saat *Y1* adalah suatu spanning tree dari *P*, terdapat suatu path di *Y1* yang bergabung dengan kedua endpoint. Saat penelusuran sepanjang path, harus terdapat sesuatu yang menghadapi suatu edge *f* yang bergabung dengan suatu vertek di *V* dengan salah satu edge yang tidak berada di *V*. Saat di iterasi ketika *e* ditambahkan ke *Y*, *f* dapat juga

ditambahkan dan akan ditambahkan sebagai ganti *e* jika bobot nya kurang dari *e*. Saat f tidak ditambahkan, kita menyimpulkan bahwa $w(f) = w(e)$.

Misalkan *Y2* menjadi tree yang diperoleh dengan pemindahan *f* dan menambahkan *e* dari *Y1*. Hal ini menunjukkan bahwa *Y2* itu adalah tree yang lebih umum dengan *Y* dibanding dengan *Y1*. Jika *Y2* sama dengan *Y*, QED. Jika bukan, kita dapat temukan tree, *Y3* dengan satu lagi edge lebih umum dengan *Y* dibanding *Y2* dan sebagainya. Selanjutnya menghasilkan suatu tree yang lebih secara umum dengan Y dibanding dengan tree yang terdahulu. Jika terdapat sejumlah edge di *Y*, dengan jumlah sekuens terbatas, maka akan ada tree, *Yh*, yang sama dengan *Y*. Ini menunjukan *Y* adalah suatu minimal spanning tree.

Dengan menggunakan suatu struktur data binary heap sederhana, algoritma prims dapat bekerja pada waktu $O((M+ n) \log n)$, dengan m adalah banyaknya edge, dan n adalah banyaknya vertek. Sedangkan dengan menggunaan suatu Fibonacci heap yang lebih canggih, algoritma ini dapat dikurangi kompleksitasnya menjadi $O(M+ n \log n)$, yang lebih cepat saat graph cukup padat dimana *m* adalah Omega($n \log n$).

Kompleksitas :
Kompleksitas waktu asimptotik: $O(n2)$.

2.3. Minimasi Waktu dalam Sistem (scheduling)

2.3.1. Konsep

Pemilihan srategi *greedy* untuk penjadwalan pelanggan akan selalu menghasilkan solusi optimum. Keoptimuman ini dinyatakan :
Jika $t1 = t2 = \ldots = tn$ maka pengurutan $ij = j$, $1 = j = n$ meminimumkan

$$T = \sum_{k=1}^{n} \sum_{j=1}^{k} tij$$

untuk semua kemungkinan permutasi *ij*.

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.

2.3.2. 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 ti}
for i ←1 to n do
output(‘Pelanggan ‘, i, ‘ dilayani!’)
endfor

2.3.3. 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 ti. Kita ingin meminimumkan total waktu di dalam sistem.

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

(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 $t1 = 5$, $t2 = 10$, $t3 = 3$, maka enam urutan pelayanan yang mungkin adalah:

Urutan	*T*
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** . **(optimal)**
3, 2, 1:	3 + (3 + 10) + (3 + 10 + 5) = 34

2.3.4. 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 $O(n)$.

3. Kesimpulan dan Saran

Algoritma *Greedy* adalah suatu algoritma yang menyelesaikan masalah secara 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 (Prim'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 mencari sirkuit hamilton terpendek *(TSP)* dan yang terkecil adalah pada *scheduling.*

Daftar Pustaka

[1] Munir, Rinaldi. 2005. *Diktat Kuliah IF2153 Matematika Diskrit*. Bandung : Laboratorium Ilmu dan Rekayasa Komputasi Program Studi Informatika Sekolah Teknik Elektro dan Informatika ITB.

[2] http://kur2003.if.itb.ac.id/. Diakses tanggal 3 Januari 2007

[3] http://wikipedia.org. Diakses tanggal 3 Januari 2007

[4] M.Agrawal, N.Kayal, and N.Saxena. 2002. *PRIMES is in P*