Membangun Pohon Merentang Minimum Dari Algoritma Prim dengan Strategi Greedy

Doni Arzinal[1]
Jursan Teknik Informatika, Institut Teknologi Bandung
Labtek V, Jl. Ganesha 10 Bandung

[1]if15109@students.if.itb.ac.id, NIM : 13505109

Abstrak

Graf merupakan salah satu metode untuk mencari solusi dari permasalahan diskrit yang ditemui dalam dunia nyata. Graf memiliki banyak konsep. Salah satu diantaranya adalah konsep pohon. Konsep pohon merupakan konsep yang paling penting dan populer karena konsep ini mampu mendukung penerapan graf dalam berbagai bidang ilmu. Aplikasi yang menggunakan konsep pohon diantaranya adalah pembangunan jalan dan rel kereta api, pembuatan jaringan komputer, pencarian jalur untuk penjual keliling, dll. Menghadirkan Graf dengan konsep pohon untuk memecahkan masalah yaitu dengan membangun graf menjadi pohon merentang minimum. Salah satu algoritma yang dipakai dalam membangun pohon merentang minimum adalah algoritma Prim yang merupakan algoritma dengan menggunakan strategi Greedy sehingga membentuk solusi pada tiap langkahnya. Algoritma Prim mengeksplorasi banyak pilihan pada tiap langkah dan akan selalu menghasilkan pohon merentang minimum.

Kata Kunci : graf, diskrit, pohon, pohon merentang minimum, Greedy, Prim

Abstract

Graph is the method of discrete problem solution searching that was in the real world. Graph has much of concepts. The ones is Tree concept. Tree concept is being the significant and the popular concept because support to applies graph for a lot of branch of science. The application that use tree concept such as road away built project and railway track built project, built a computer network, the salessman tavelling problem, etc. Presenting graph uses tree concept to solve the problem is build the graph to be a Minimum Spanning Tree (MST). One of the algorithm that used to built MST is Prim algorithm, which uses Greedy strategy so solution resulted in every steps. Prim algorithm explores much of selection in every steps and product at least one MST.

Key Words : graph, discrete, tree, minimum spanning tree, Greedy, Prim

1. Pendahuluan

Berbagai pertanyaan yang ditemui penulis ketika mendapatkan mata kuliah Matematika Diskrit (Matdis) terutama tentang graf, membuat penulis ingin lebih jauh mendalami permasalahan graf. Bila di Matdis penulis lebih ditekankan pada penerapan graf untuk memecahkan masalah, maka di sini penulis ingin membuka lebih jauh tentang graf dan konsep-konsepnya. Karena menurut penulis, graf merupakan ilmu yang sangat penting dan mampu menyelesaikan banyak permasalahan.

Pemilihan konsep pohon sebagai salah satu konsep terapan graf disebabkan karena konsep pohon merupakan konsep yang sangat dekat dengan kehidupan nyata. Secara tidak disadari, manusia banyak yang menggunakan pohon sebagai pemodelan berbagai hal dalam kehidupan sehari-hari. Peran konsep pohon sangat besar ketika diterapkan pada aplikasi besar yang menyangkut hidup orang banyak.

Konsep pohon yang sangat membantu manusia adalah pohon merentang minimum. Pembahasan tentang metode untuk mendapatkan pohon merentang minimum masih menjadi suatu hal yang asing dan kadang terabaikan begitu saja sehingga pemecahan masalah dengan pohon merentang minimum menjadi kurang optimal. Selain itu, ketika pemakaian algoritma untuk pohon merentang minimum kadang membingungkan ketika sedang diterapkan karena sering sekali bercampur dengan pemahaman metode pemecahan masalah jalur terpendek dengan algoritma exhaustive search.

2. Dasar Teori

Teori graf merupakan pokok bahasan yang sudah tua usianya namun memiliki banyak terapan sampai saat ini. 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. Contoh salah satu representasi visual dari graf adalah peta. Banyak hal yang dapat digali dari representasi tersebut, diantaranya 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, dan sebagainya. Selain peta, masih banyak hal lain dalam dunia nyata yang merupakan representasi visual dari graf.

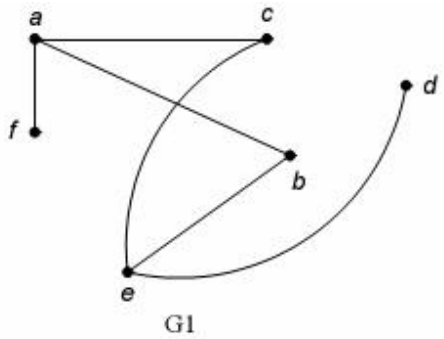
Sejumlah masalah yang berkaitan dengan graf yang ditemukan manusia dalam kehidupan nyata menimbulkan penemuan konsep-konsep pemecahan masalah graf. Diantara sekian banyak konsep graf yang ada, konsep pohon (tree) merupakan konsep yang paling populer dan penting. Karena konsep ini mampu mendukung pemecahan masalah dalam berbagai terapan graf. Konsep pohon digunakan sejak tahun 1857 oleh seorang matematikawan Inggris bernama Arthur Caylay untuk menghitung jumlah senyawa kimia.

Di dalam konsep pohon sendiri, terdapat banyak jenis pohon yang digunakan sebagai cara untuk mencari solusi dari masalah dalam dunia terapan graf. Dan yang akan dibahas disini adalah penggunaan pohon merentang atau lebih dikenal dengan Spanning Tree. Aplikasi Spanning Tree misalnya pada pemeliharaan jalan raya. Dengan Spanning Tree, pemerintah dapat mengalokasikan dana secara optimal dengan mencari jalur pemeliharaan dengan bobot biaya yang minimum. Spanning Tree juga memainkan peranan penting dalam jaringan komputer, travelling salesman problem, dan lain-lain. Jika Spanning Tree diterapkan pada persoalan yang mengandung unsur pencarian bobot yang minimum maka dinamakan Minimum Spanning Tree (MST).

2.1 Pohon (Tree)

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

Contoh pohon :



G1

Gambar 1. Gambar Pohon G1

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 Sifat-sifat Pohon

Sifat-sifat (properties) pohon adalah sebagai berikut :

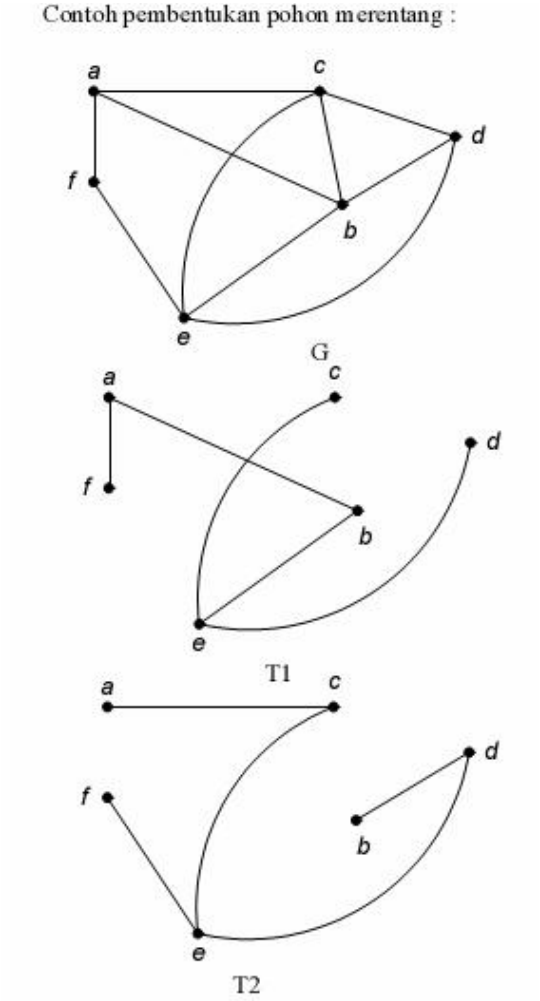
Misalkan G = (V, E) adalah graf tak-berarah sederhanan dan jumlah simpulnya n. maka, semua pernyataan dibawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki m = n 2 buah sisi.
4. G tidak mengandung sirkuit dan memiliki m = n-1 buah sisi.
5. G tidak mengandung sirkuit dan memilik m = n-1

2.3 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]

Contoh pembentukan pohon merentang :



Gambar 2. G adalah Graf. T1 dan T2 adalah Pohon

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.

Aplikasi pohon merentang misalnya pada pemiliharaan jalan raya. Misalkan graf G adalah peta jaringan jalan raya yang menghubungkan empat buah kota. Karena dana pemeliharaan yang terbatas, pemerintah daerah mempertimbangkan hanya memelihara jalan-jalan sesedikit mungkin sehingga keempat masih tetap terhubung satu sama lain. Masalah ini dapat dipecahkan dengan membuat upgraf yang mengandung jumlah sisi minimum dan mengandung semua simpul di dalam graf. Graf semacam itu haruslah pohon merentang.

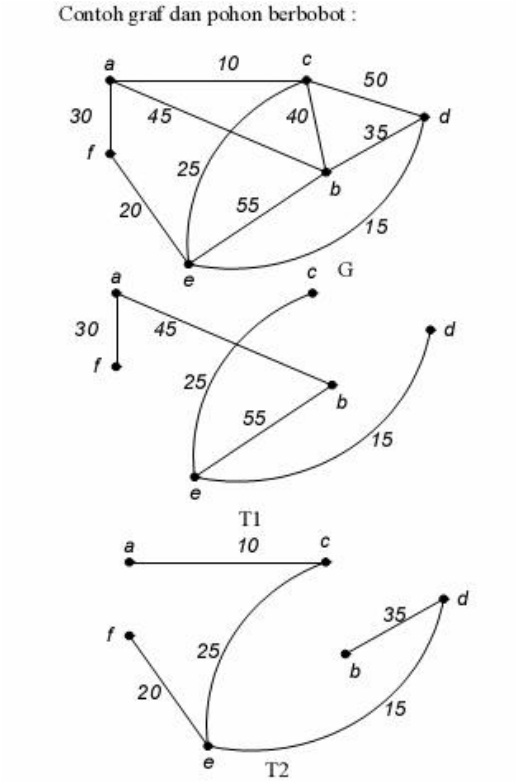
Pohon merentang juga memainkan peranan penting dalam jaringan komputer. Jaringan komputer dapat dimodelkan sebagai sebuah graf. Simpul pada graf dapat menyatakan suat terminal komputer (work station) atau suatu router (router adalah komputer yang difungsikan untuk meneruskan data dari suatu simpul komunikasi ke simpul komunikasi lain). Jika sebuah komputer mengirim pesan (atau data) ke komputer lain (melalui router), maka komputer tersebut mengirimkannya ke seluruh simpul-simpul di jaringan. Setiap pesan yang sampai ke suatu router antara akan diteruskan ke satu atau lebih router lainnya. Dengan cara seperti ini, maka pesan akan sampai ke komputer penerima. Pesan yang telah sampai ke suatu router diharapkan tidak pernah kembali diterima oleh router tersebut. Tetapi karena router-router pada umumnya membentuk sirkuit (atau cycle atau loop), maka menerima pesan yang sama lebih dari sekali pasti terjadi. Untuk mengatasi hal ini, maka algoritma jaringan membentuk pohon merentang di dalam graf sehingga antara sepasang simpul router hanya ada satu lintasan tunggal dan simpul-simpul router tidak pernah menerima pesan yang sama lebih dari sekali. Metode penyebaran pesan (routing) seperti ini dinamakan IP Multicasting.

Harus diingat bahwa pohon merentang didefenisikan hanya untuk graf terhubung, karena pohon selalu terhubung. Pada graf tak-terhubung dengan n buah simpul kita tidak dapat menemukan upagraf terhubung dengan n buah simpul. Tiap komponennya dari graf tak-terhubung mempunyai satu buah pohon merentang. Dengan demikian, graf tak-terhubung dengan k komponen mempunyai hutan merentang (spanning forest) yang terdiri dari k buah pohon merentang.

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

Keterangan :
Dari graf berbobot G, kita harus menentukan pohon merentang mana yang paling minimum. Apakah T1 atau T2? Hal tersebut yang akan dicari dengan membangun pohon merentang minimum.

2.5 Algoritma Greedy

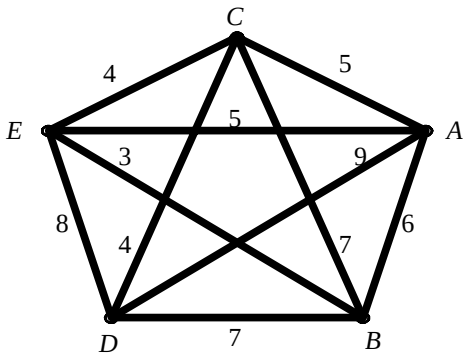
Algoritma Greedy merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Algoritma greedy membentuk solusi langkah per langkah yaitu :

- 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)

2.5.1 TSP (*Travelling Salesperson Problem*)

- Strategi *greedy* untuk memilih kota selanjutnya:

Pada setiap langkah, pilih kota yang belum pernah dikunjungi yang mempunyai jarak terdekat.



Contoh 6.
n = 5; perjalanan dimulai dari kota E.

Langkah 1: Dari E pergi ke B (Total jarak = 3)
Langkah 2: Dari B pergi ke A (Total jarak = 3 + 6 = 9)
Langkah 3: Dari A pergi ke C (Total jarak = 9 + 5 = 14)
Langkah 4: Dari C pergi ke D (Total jarak = 14 + 4 = 18)
Langkah 5: Dari D kembali lagi ke E (Total jarak = 18 + 8 = 26)

Perjalanan (sirkuit Hamilton) yang dihasilkan:

$$E \rightarrow B \rightarrow A \rightarrow C \rightarrow D \rightarrow E$$

Panjang = 3 + 6 + 5 + 4 + 8 = 26. (tidak optimal)

Solusi yang lebih baik:

$$E \rightarrow B \rightarrow D \rightarrow C \rightarrow A \rightarrow E$$

dengan panjang = 3 + 7 + 4 + 5 + 5 = 24. (optimal)

- **Kesimpulan**: Masalah TSP tidak dapat diselesaikan dengan algoritma *greedy*, karena solusi yang diahasilkan tidak dijamin merupakan solusi optimal. Namun jika solusi hampiran dianggap mencukupi, maka solusi dengan algoritma greedy dapat dijadikan sebagai titik awal untuk menemukan sirkuit Hamilton yang minimum.

2.5.2 Penjadwalan *Job* dengan Tenggat Waktu (*Job Schedulling with Deadlines*)

- **Persoalan:** Ada *n* buah *job* yang akan dikerjakan oleh sebuah mesin. Tiap *job* diproses oleh mesin selama satu satuan waktu dan tenggat waktu (*deadline*) – yaitu batas waktu *job* tersebut harus sudah selesai diproses– tiap *job i* adalah $d_i \geq 0$ (d_i dalam satuan waktu yang sama dengan waktu proses mesin). *Job i* akan memberikan keuntungan sebesar p_i jika dan hanya jika *job* tersebut diselesaikan tidak melebihi tenggat waktunya.

- Obyektif persoalan adalah memilih *job-job* yang akan dikerjakan oleh mesin sehingga keuntungan yang diperoleh dari pengerjaan itu maksimum.

- Secara matematis, fungsi obyektif persoalan ini dapat ditulis sebagai

$$\text{Maksimasi } F = \sum_{i \in J} p_i$$

- Solusi yang layak (*feasible*) ialah himpunan *J* yang berisi urutan *job* yang akan diproses oleh sebuah mesin sedemikian sehingga setiap *job* di dalam *J* selesai dikerjakan sebelum tenggatnya.

- Solusi optimum ialah solusi layak yang memaksimumkan *F*.

Contoh 7. Misalkan A berisi 4 buah job (n = 4). Tenggat setiap *job* dan keuntungan masing-masing:

$(p_1, p_2, p_3, p_4) = (50, 10, 15, 30)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Job	Tenggat	Harus selesai pada pukul
1	2 jam	8.00
2	1 jam	7.00
3	2 jam	8.00
4	1 jam	7.00

2.5.3 Pemecahan Masalah dengan Algoritma Exhaustive Search

- Persoalan penjadwalan job dengan tenggat waktu dapat dipandang sebagai mencari himpunan bagian (*subset*) *job* yang layak dan memberikan total keuntungan terbesar.
- Algoritma *exhaustive search* untuk persoalan penjadwalan job dengan tenggat waktu mempunyai kompleksitas $O(n \cdot 2^n)$.

2.5.4 Algoritma Greedy untuk Membentuk Kode Huffman

- Huffman menemukan algoritma *greedy* untuk membentuk kode prefiks yang optimal. Algoritma Huffman membangun pohon biner *T* (yang disebut pohon Huffman) yang berkoresponden dengan kode optimal tersebut dari bawah ke atas (*bottom-up*).

- Langkah-langkah pembentukan pohon Huffman adalah sebagai berikut:

1. Baca semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-*assign* dengan frekuensi kemunculan karakter tersebut.

2. Terapkan strategi *greedy* sebagai berikut: gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon penyusunnya.

3. Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman.

- Agar pemilihan dua pohon yang akan digabungkan berlangsung cepat, maka semua pohon yang ada selalu terurut menaik berdasarkan frekuensi.

2.5.5 Kenapa strategi pemilihan dua pohon dengan frekuensi terkecil greedy?

- Ongkos (*cost*) penggabungan dua buah pohon pada sebuah akar setara dengan jumlah frekuensi dua buah yang digabung. Oleh karena itu, total ongkos pembentukan pohon Huffman adalah jumlah ongkos seluruh penggabungan.

- Penggabungan dua buah pohon dilakukan pada setiap langkah, dan algoritma Huffman selalu memilih dua buah pohon yang mempunyai frekuensi terkecil untuk meminimumkan total ongkos. Inilah alasan mengapa strategi penggabungan dua buah pohon yang mempunyai frekuensi terkecil merupakan strategi *greedy*.

- Algoritma Huffman mempunyai kompleksitas $O(n \log n)$ untuk himpunan dengan n karakter.

2.6 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-dua sebanyak n-2 kali

3. Analisa

Analisa akan dilakukan pada algoritma Greedy dan algoritma Prim untuk memperlihatkan strategi greedy pada algoritma Prim. Serta akan diperlihatkan bagaimana algoritma Prim diterapkan pada suatu kasus graf.

3.1 Analisa Algoritma Greedy

```
Procedure greedy (input C: himpunan_kandidat;
output S : himpunan_solusi)
{ 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 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**.**

```
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.

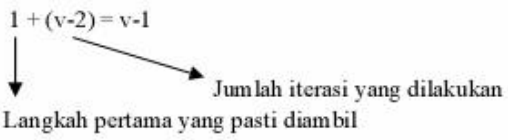
Apakah mungkin sisi yang bersisian membentuk sirkuit? Mungkin saja. Bagaimana mengetahui bahwa sisi tersebut tidak membentuk sirkuit? Menurut penulis, untuk mengatasi hal tersebut harus dilihat apakah titik ujung dari sisi tersebut sudah ada dalam T atau belum. Jika sudah ada maka tidak boleh memilih 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.

3.3 Kompleksitas Algoritma Prim

Jumlah seluruh langkah pada algoritma Prim adalah



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 2n ? 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

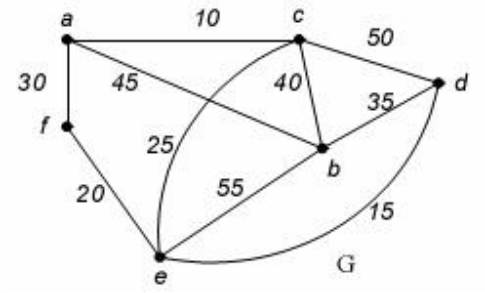
$(n-1) \times n = n^2 - n$,

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

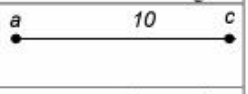
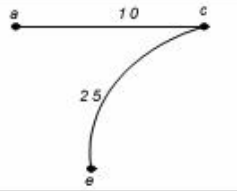
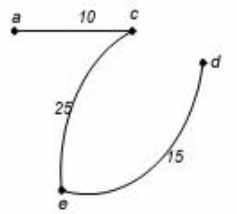
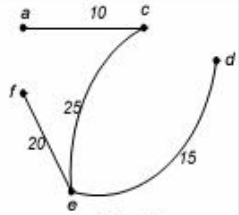
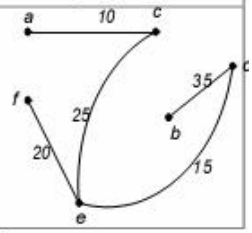
Simpul dan bobot pada graf direpresentasikan ke dalam matriks. Untuk mencari suatu sisi, maka algoritma Prim akan mencari kedua arah yaitu baris dan kolom graf G kemudian akan dilihat bobotnya.

3.4 Studi Kasus dengan Algoritma Prim

Studi kasus ini menggunakan gambar graf G pada gambar 3. Lalu kita akan lihat, apakah hasil dari algoritma Prim akan menghasilkan T1, T2 ataukah pohon T yang lain.



Tabel pembentukan pohon merentang minimum dari gambar 3

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	

Tabel 1

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 locl 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.

4.2 Saran Pengembangan

- Untuk mempercepat proses, susun sisi-sis 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