Algoritma Prim dengan Algoritma Greedy dalam Pohon Merentang Minimum

Made Mahendra Adyatman – 13505015

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15015@students.if.itb.ac.id

Abstrak

Makalah ini dibuat untuk membahas algoritma Prim dengan strategi Greedy untuk membangun pohon yang merentang minimum. Pohon adalah Graf yang terhubung dan tidak berarah. 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.

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*).

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, pohon, pohon merentang minimum, Greedy, Prim

1. Pendahuluan

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 seharihari. 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.

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.

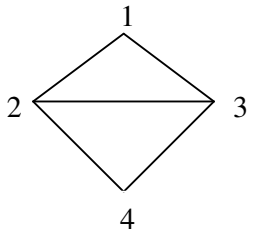
Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan yang "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan **optimum lokal** (*local optimum*) pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi **optimum global** (*global optimum*). Salah satu algoritma yang menggunakan strategi Greedy ini adalah algoriitma Prim yang akan kita bahas dalam makalah ini.

2. Graf

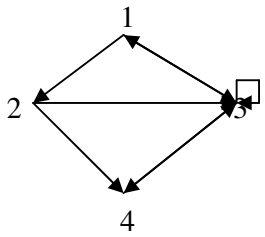
Graf G didefinisikan sebagai pasangan himpunan (V, E), yang dalam hal ini:
V = himpunan tidak-kosong dari simpul-simpul (vertices atau node) = { v1, v2, ... , vn}
dan
E = himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul = {*e*1, *e*2, ...,*en*}
atau dapat ditulis singkat notasi *G* = (*V*, *E*)

Jumlah simpul pada graf kita sebut sebagai kardinalitas graf, dan dinyatakan dengan *n* = *V* , dan jumlah sisikita nyatakan dengan *m* = *E* .

V tidak boleh kosong, sedangkan *E* boleh kosong. Jadi, sebuah graf dimungkinkan tidak mempunyai sisi satu buah pun, tetapi simpulnya harus ada, minimal satu. Graf yang
hanya mempunyai satu buah simpul tanpa sebuah sisi pun dinamakan **graf trivial**



G adalah graf dengan himpunan simpul V dan himpunan sisi E adalah:
V = { 1, 2, 3, 4 }
E = { (1, 2), (1, 3), (2, 3), (2, 4), (3, 4) }



G adalah graf dengan himpunan simpul *V* dan himpunan sisi *E* adalah:
V = { 1, 2, 3, 4 }
E = { (1, 2), (2, 3), (1, 3), (1, 3), (2, 4), (3, 4), (3, 4), (3, 3) } adalah himpunan ganda
= { *e*1, *e*2, *e*3, *e*4, *e*5, *e*6, *e*7, *e*8}
(1, 3) dinamakan **sisi-ganda** (*multiple edges* atau *paralel edges*) karena
kedua sisi ini menghubungi dua buah simpul yang sama, yaitu simpul 1 dan simpul 3.
(3, 3) dinamakan **gelang** atau **kalang** (*loop*) karena ia berawal dan berakhir pada simpul yang sama.

2.1. Jenis-Jenis Graf

Pengelompokan graf dapat dipandang berdasarkan ada tidaknya sisi ganda atau sisi kalang, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi.

1. Graf sederhana
   Graf yang tidak mengandung gelang maupun sisi-ganda dinamakan graf sederhana.
2. Graf tak sederhana
   Graf yang mengandung sisi ganda atau gelang dinamakan graf tak-sederhana (*unsimple graph*). Ada dua macam graf tak-sederhana, yaitu graf ganda (*multigraph*) dan graf semu (*pseudograph*). Graf ganda adalah graf yang mengandung sisi ganda. Graf semu adalah graf yang mengandung gelang.

3. Graf berhingga
   Graf berhingga adalah graf yang jumlah simpulnya, *n*, berhingga.
4. Graf tak berhingga
   Graf yang jumlah simpulnya, n, tidak berhingga banyaknya disebut graf tak-berhingga.
5. Graf berarah
   Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Kita lebih suka menyebutsisi berarah dengan sebutan busur (*arc*).
6. Graf tak berarah
   Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan.

2.2. Terminologi Dasar Graf

1. Bertetanggaan
   Dua buah simpul pada graf tak-berarah *G* dikatakan bertetangga bila keduanya terhubung
   langsung dengan sebuah sisi.
2. Bersisian
   Sisi *e* dikatakan bersisian dengan simpul *vj* dan simpul *vk* jika sisi e menghubungkan kedua simpul tersebut
3. Simpul Terpencil
   Simpul yang tidak mempunyai sisi yang bersisian dengannya. Atau,dapat juga dinyatakan bahwa simpul terpencil adalah simpul yang tidak satupun bertetangga dengan simpul-simpul lainnya.
4. Graf Kosong
   Graf yang himpunan sisinya merupakan himpunan kosong disebut sebagai **graf kosong** danditulis sebagi *Nn*, yang dalam hal ini *n a*dalah jumlah simpul.
5. Derajat
   Jumlah sisi yang bersisian dengan simpul tersebut.Notasi: *d*(*v*) menyatakan derajat simpul *v*.
   Lemma Jabat Tangan. Jumlah derajat semua simpul pada suatu graf adalah genap, yaitu dua kali jumlah sisi pada graf tersebut. Dengan kata lain, jika *G* = (*V*, *E*), maka ∑d = E x 2
   (catatan: ingatlah *E x* 2 selalu bernilai genap)
   Lemma ini dikenal dengan lemma jabat tangan (*handshaking lemma*). Hal ini disebabkan oleh setiap sisidihitung dua kali, yaitu pada ujung kiri sebagai bagian dari simpul kiri dan pada ujung kanan dihitungsebagai bagian dari simpul kanan. Layaknya orang berjabat tangan, maka jumlah tangan yang berjabatanadalah genap dan jumlah tangan yang berjabatan adalah dua kali jumlah jabatan tangan yang terjadi. Catatlah bahwa Lemma Jabat Tangan juga benar untuk graf berarah, yang dalam hal ini
   d(*v*) = *d*in(*v*) + *d*out(*v*).
6. Lintasan(Path)
   Lintasan yang panjangnya *n* dari simpul awal *v*0ke simpul tujuan *vn*di dalam graf *G* ialahbarisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk *v*0, *e*1, *v*1, *e*2, *v*2,... , *vn* –1, *en*, *vn* sedemikian sehingga *e*1= (*v*0, *v*1), *e*2= (*v*1, *v*2), ... , *en*= (*vn*-1, *vn*) adalah sisi-sisi dari graf *G*. Lintasan yang berawal dan berakhir pada simpul yang sama disebut **lintasan tertutup** (*closed path*),sedangkan lintasan yang tidak berawal dan berakhir pada simpul yang sama disebut **lintasan terbuka** (*open path*).
7. Sirkuit
   Lintasan yang berawal dan berakhir pada simpul yang sama disebut **sirkuit** atau **siklus**.
8. Terhubung
   Dua buah simpul *vi*dan simpul *vj*dikatakan **terhubung** jika terdapat lintasan dari *vi* ke *vj*. Jika dua buah simpul terhubung maka pasti suatu simpul dapat dicapai dari simpul lain.
9. Sub Graf
   Misalkan *G* = (*V*, *E*) adalah sebuah graf. *G*1= (*V*1, *E*1) adalah upagraf (*subgraph*) dari *G* jika *V*1 termasuk *V* dan *E*1termasuk *E*
10. Cut-Set
    Cut set dari graf terhubungG adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung , selalu menghasilkan dua buah komponen terhubung.
11. Graf Berbobot
    Adalah Graf yang setiap sisinya diberi nilai atau bobot, sehingga bias dihitung lintasannya

3. Pohon

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit.
Hutan (*forest*) adalah
- kumpulan pohon yang saling lepas, atau
- graf tidak terhubung yang tidak mengandung sirkuit. Setiap komponen di dalam graf terhubung tersebut adalah pohon.

Pohon mempunyai bilangan kromatis = 2

3.1. Sifat-sifat Pohon

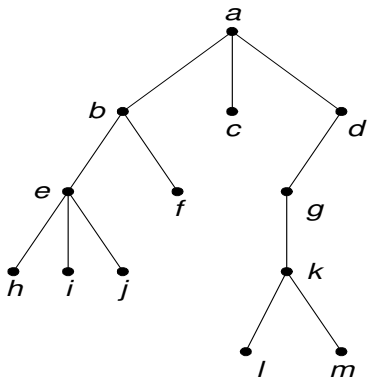
Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.

3.2. Pohon Berakar

Pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah dinamakan **pohon berakar** (*rooted tree*).

3.3. Terminologi Dasar Pohon



1. Anak (*child* atau *children*) dan Orangtua (*parent*)
   b, *c*, dan *d* adalah anak-anak simpul *a*,
   a adalah orangtua dari anak-anak itu
2. Lintasan (*path*)
   Lintasan dari *a* ke *j* adalah *a*, *b*, *e*, *j*.
   Panjang lintasan dari *a* ke *j* adalah 3.
3. Saudara kandung (*sibling*)
   f adalah saudara kandung *e*,
   tetapi, *g* bukan saudara kandung *e*, karena orangtua mereka berbeda.
4. Upapohon (*subtree*)
   Bagian dari pohon, anak pohon
5. Derajat (*degree*)
   Derajat sebuah simpul adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut.
   Derajat *a* adalah 3, derajat *b* adalah 2, Derajat *d* adalah satu dan derajat *c* adalah 0.
   Jadi, derajat yang dimaksudkan di sini adalah derajat-keluar.
   Derajat maksimum dari semua simpul merupakan derajat pohon itu sendiri. Pohon di atas berderajat 3
6. Daun (*leaf*)
   Simpul yang berderajat nol (atau tidak mempunyai anak) disebut **daun**. Simpul *h*, *i*, *j*, *f*, *c*, *l*, dan *m* adalah daun.
7. Simpul Dalam (*internal nodes*)
   Simpul yang mempunyai anak disebut **simpul dalam**. Simpul *b*, *d*, *e*, *g*, dan *k* adalah simpul dalam.
8. Tinggi (*height*) atau Kedalaman (*depth*)
   Aras maksimum dari suatu pohon disebut **tinggi** atau **kedalaman** pohon tersebut. Pohon di atas mempunyai tinggi 4.

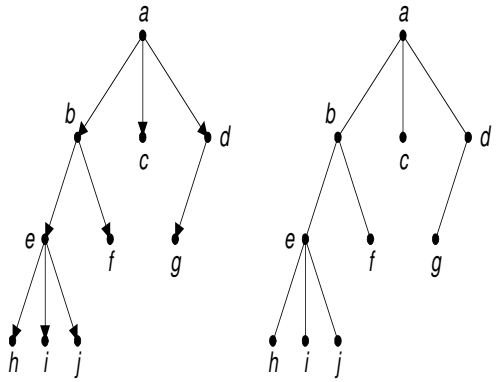
3.4. Pohon Terurut

Pohon berakar yang urutan anak-anaknya penting disebut **pohon terurut** (*ordered tree*).

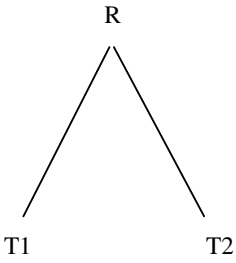
3.5. Pohon *m-ary*

Pohon berakar yang setiap simpul cabangnya mempunyai paling banyak *m* buah anak disebut **pohon *m-ary***.

Jika $m = 2$, pohonnnya disebut **pohon biner** (*binary tree.)*. Pohon *m-ary* dikatakan **teratur** atau **penuh** (*full*) jika setiap simpul cabangnya mempunyai tepat *m* anak.



3.56 Penulusuran Pohon Biner(n-ary, n=2)



Ada tiga macam, yaitu **preorder, inorder dan postorder :**

1. Preorder

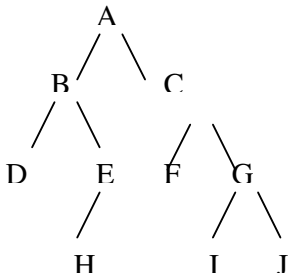
Kunjungi R, telusuri T1 secara preorder, telusuri T2 secara preorder

2. Inorder

Telusuri T1 secara inorder, kunjungi R, telusuri T2 secara inorder

3. Postorder

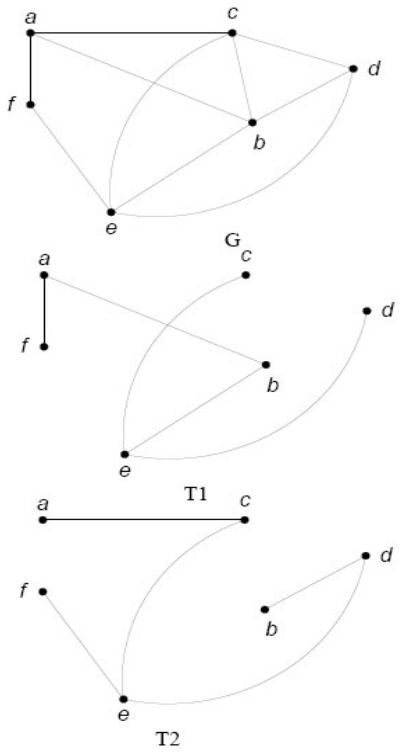
Telusuri T1 secara postorder, telusuri T2 secara postorder, lalu kunjungi R



preorder : A, B, D, E, H, C, F, G, I, J
inorder : D, B, H, E, A, F, C, I, G, J
postorder : D, H, E, B, F, I, J, G, C, A

4. Pohon Merentang

- Pohon merentang dari graf terhubung adalah upagraf merentang yang berupa pohon.
- Pohon merentang diperoleh dengan memutus sirkuit di dalam graf.
- Setiap graf terhubung mempunyai paling sedikit satu buah pohon merentang.
- Graf tak-terhubung dengan *k* komponen mempunyai *k* buah hutan merentang yang disebut hutan merentang (*spanning forest*).
- Contoh pembentukan pohon merentang

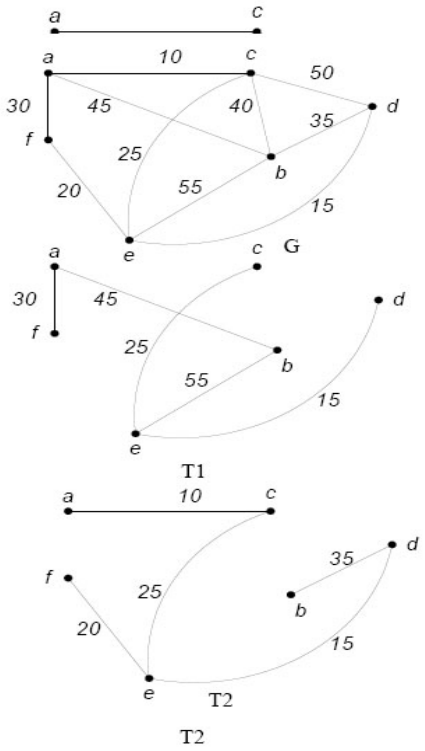


Keterangan :
T1 dan T2 merupakan pohon merentang dari graf G
Pohon merentang T1 dibentuk dengan cara menghapus sisi {(a,c),(b,c),(b,d),(c,d),(e,f)} dari graf G.
Pohon merentang T2 dibentuk dengan cara menghapus sisi {(a,f),(a,b),(b,c),(b,e),(c,d)} dari graf G.

4.1. Pohon Merentang Minimun

Jika G pada merupakan graf berbobot, maka bobot pohon merentang T1 atau T2 didefinisikan sebagai jumlah bobot semua sisi di T1atau T2. Diantara pohon merentang yang ada padaG, yang palin penting adalah pohon merentangdengan bobot minimum. Pohon merentang dengan bobot minimum ini disebut dengan pohon merentang minimum atau *Minimum Spanning Tree (MST).*Contoh aplikasi MST yang sering digunakan adalahpemodelan proyek pembangunan jalan rayamenggunakan graf. MST digunakan untuk memilihn jalur dengan bobot terkecil yang akan meminimalkan biaya pembangunan jalan.

Contoh graf dan pohon berbobot :



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.

4.2. Konsep

Algoritma prims 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:

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

5. Algoritma Greedy

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

1. 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.
2. Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan
3. 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
(inputC:himpunan_kandidat;
outputS: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 = {} }

5.1. 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?

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

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.

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

6.2. Analisa :

Langkah pertama pada Algoritma Prim adalah mencari sisi pada himpunan E yang menyatakan sisisisi 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.

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*.

Y_1 dijadikan spanning tree minimal untuk *P*. Jika *Y*= Y_1, kemudian tanda bukti tersebut lengkap. Jika bukan, terdapat suatu edge di *Y* yang tidak ada di Y_1. 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 Y_1 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 $w(f) \geq w(e)$.

Misalkan Y_2 menjadi tree yang diperoleh dengan pemindahan *f* dan menambahkan *e* dari Y_1. Hal ini menunjukkan bahwa Y_2 itu adalah tree yang lebih umum dengan *Y* dibanding dengan Y_1. Jika Y_2 sama dengan *Y*, QED. Jika bukan, kita dapat temukan tree, Y_3 dengan satu lagi edge lebih umum dengan *Y* dibanding Y_2 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.

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.

6.3. Kompleksitas

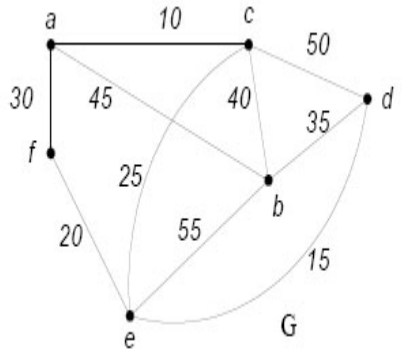
Dengan menggunakan suatu struktur data binary heap sederhana, algoritma prims dapat bekerja pada waktu $O((M+ n) \log n)$, dengan m adalah banyaknya edge, dan n adalah banyaknya vertek. Sedangkan dengan menggunaan suatu Fibonacci heap yang lebih canggih, algoritma ini dapat dikurangi kompleksitasnya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

Jumlah seluruh langkah pada algoritma Prim adalah
1 + (v-2) = v-1

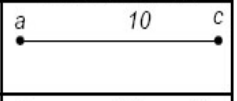
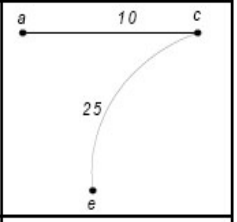
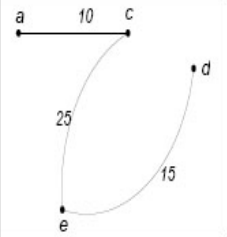
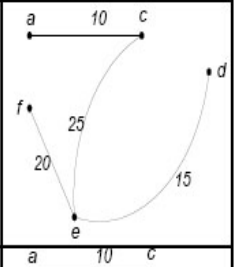
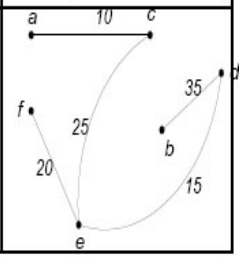
Keterangan :
v : vertex atau simpul dalam pohon merentang Algoritma Prim mempunyai kompleksitas waktu asimptotik $O(n^2)$. Artinya jika dimasukkan sebanyak n sisi akan memerlukaan waktu n^2. Kenapa n2? Karena setiap sisi akan mengecek semua sisi yang bertetangga dengannya. Pada langkah pertama, algoritma 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) x n = n^2– n, sehingga kompleksitas waktu asimptotik 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.

7. Studi Kasus

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.



Langkah-langkah pembentukan pohon merentang minimum dari gambar diatas.

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	

8. 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.

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.

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

Daftar Pustaka

1. Munir, Rinaldi. 2006. *Diktat Kuliah IF2153 Matematika Diskrit.* Bandung. Informatika
2. Munir, Rinaldi. 2004. *Handout Bahan Kuliah ke 1-4*. Bandung. ITB
3. Munir, Rinaldi. 2005. *Diktat Kuliah IF 2251 Strategi Algoritmik.* Bandung: Laboratorium Ilmu dan Rekayasa Komputasi Departemen Teknik Informatika ITB.
4. Liem, Inggriani. 2001. *Diktat Struktur Data IF222*. Bandung : Informatika
5. *http://kur2003.if.itb.ac.id/*
Diakses tanggal 1 Januari 2007 ~ 13.05
6. M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, 2002
7. http://www.wikipedia.org
Diakses tanggal 1 Januari 2007 ~ 11.00
8. http://cs.ui.id
Diakses tanggal 1 Januari 2007 ~ 12.02