PRIM vs KRUSKAL
PERBANDINGAN ALGORITMA PENCARIAN POHON MERENTANG MINIMUM

Hanson Prihantoro Putro – NIM : 13505045

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if115045@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang studi dan perbandingan Algoritma Prim dan Kruskal untuk menyelesaikan problem pencarian pohon merentang minimum. Perbandingan yang akan diulas disini meliputi alur algoritma, bagaimana mereka melakukan tahap per tahap instruksi untuk mendapatkan pohon merentang itu serta perbandingan dalam penggunaannya dalam beberapa studi kasus yang berbeda, terkait dengan effisiensi algoritma ini dalam kalkukasi setiap masalah yang berbeda. Efisiensi inilah yang akan lebih dititikberatkan dalam pembahasan ini. Dengan perbedaan ini, setiap algoritma memiliki kelebihan dan kekurangan dalam menyelesaikan suatu masalah. Ada kasus yang dengan mudah diselesaikan oleh algoritma Kruskal misalnya, namun untuk kasus lain, algoritma Prim jauh memberikan keuntungan dibanding Prim. Walaupun masih dalam satu permasalahan, yaitu pencarian pohon merentang minimum, data-data yang diberikan sering kali mengakibatkan algoritma yang satu berjalan lebih cepat dibanding algotima yang lain, atau kadang sebaliknya.
Dalam pembahasan kali ini, perbandingan dilakukan dengan memodelkan permasalahan ini ke dalam pemrograman komputer. Bahasa pemrograman yang digunakan adalah bahasa Pascal. Diharapkan pemahaman mengenai bahasa Pascal sudah sedikit dimengerti. Untuk teori-teori graf dan beberapa pendukungnya, akan dijelaskan di makalah ini.

***Kata kunci**:* efisiensi, algoritma, instruksi, bahasa Pascal, komplemen, matriks, representasi, indeks, algoritma *greedy*, kode, realisasi, analisa algoritma, *quicksort*, eksekusi, kompleksitas algoritma.

1. Pendahuluan

Zaman kini sudah berkembang begitu pesatnya. Semakin berkembang suatu zaman, semakin banyak permasalahan yang dihadapi. Permasalahan timbul karena orang menginginkan kenyamanan dan keuntungan yang lebih. Orang kemudian berlomba-lomba untuk bisa menyelesaikan persoalan ini. Banyak penelitian dan pengembangan terus dilakukan. Untuk sebuah pemasalahan saja, tidak hanya satu solusi saja yang tersedia. Namun diantara banyak solusi yang tersedia itu, tidak semuanya memuaskan. Kadang kala untuk kasus yang lain, solusi yang lain malah bisa menyelesaikan sesuai keinginan kita. Oleh karena itu, diperlukan pemahaman yang baik dalam memilih suatu solusi yang akan digunakan untuk menyelesaikan permasalah yang kita hadapi.

Beralih ke perkembangan zaman yang terjadi, informasi memegang peranan penting dalam perkembangan ini. Pengiriman data dan informasi baik itu yang terlihat secara fisik atau tidak harus bisa berjalan dengan lancar. Komponen dari arus informasi ini adalah pengirim, penerima, jalur pengiriman serta data informasi itu sendiri. Dengan banyaknya hubungan antara pengirim dan penerima, jalur-jalur ini kemudian berkembang membentuk sebuah jaringan.

Jaringan itu tidak hanya menghubungkan satu daerah ke daerah satu yang lain, namun sudah meliputi semua wilayah di dunia ini, dari belahan bumi manapun. Jaringan yang ada saat ini harus bisa memastikan data bisa diterima oleh yang meminta dengan tepat dan cepat. Untuk membentuk jaringan besar yang baik, bagian jaringan-jaringan yang kecil ini harus baik dulu.Karena keterbatasan sumber daya yang ada, kita harus bisa menghubungkan semua wilayah ini dengan seefektif dan seefisien mungkin.

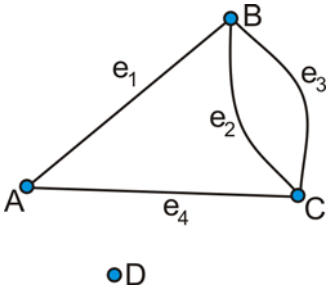
Permasalahan inilah yang kemudian kita coba untuk bahas. Penyelasaian dari permasalahan ini bisa dilakukan dengan melakukan beberapa kalkulasi dan perhitungan. Dalam ilmu hitung, lebih tepatnya Matematika Diskrit, proses menghubungkan dan dihubungkan ada dalam teori yang kita sebut Teori Graf. Graf ini sendiri memiliki beberapa teori yang digunakan untuk menyelesaiakn beberapa permasalahan yang ada. Salah satunya adalah permasalahan mengubungkan semua wilayah yang ada namun dengan menggunakan sumber daya sesedikitnya.

2. Teori Graf

Graf (*Graph*) didefiniskan sebagai:

$G = \{V, E\}$

Dalam hal ini, V merupakan himpunan tidak kosong dari simpul-simpul (*vertices*/*node*) di gambarkan dalam titik-titik, dan E adalah himpunan sisi-sisi (*edges*/*arcs*) digambarkan dalam garis-garis yang menghubungkan sepasang simpul. Dapat dikatakan graf adalah kumpulan dari simpul-simpul yang dihubungkan oleh sisi-sisi.



Gambar 1: Graf G_1

Pada G_1 diatas, graf terdiri dari himpunan V dan E yaitu:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4\}$; bisa kita tulis

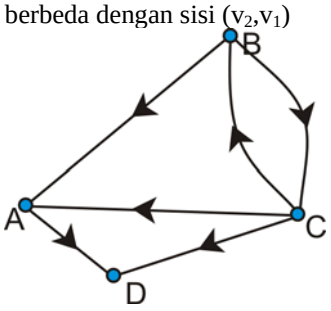
$= \{(A,B),(B,C),(B,C),(A,C)\}$

Aplikasi graf sangat luas. Graf dipakai dalam berbagai disiplin ilmu maupun dalam kehidupan sehari-hari. Penggunaan graf di berbagai bidang tersebut adalah untuk memodelkan persoalan.

Beberapa terminologi dasar yang harus diketahui:

a. Graf Berarah (*Directed Graph*/*Digraph*)

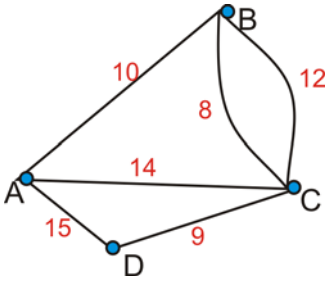
Graf berarah adalah graf yang setiap sisinya diberi orientasi arah. Dalam hal ini sisi yang ditulis (v_1,v_2) berbeda dengan sisi (v_2,v_1)



Gambar 2: Graf Berarah

b. Graf Berbobot (*Weight Graf*)

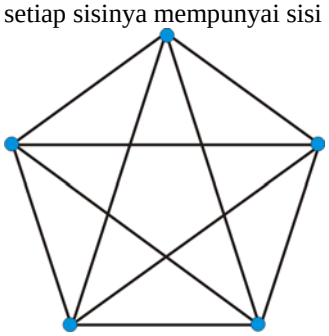
Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga.



Gambar 3: Graf Berbobot

c. Graf Lengkap (*Complete Graph*)

Graf lengkap adalah graf sederhana, tidak mengandung gelang (sisi yang kedua simpulnya sama) maupun sisi ganda (dua sisi yang memiliki simpul asal dan simpul tujuan yang sama), serta setiap sisinya mempunyai sisi ke simpul lain.



Gambar 4: Graf Lengkap K_5

d. Bertetangga (*Adjacent*)

Dua buah simpul pada graf tak berarah dikatakan bertentangga bila keduanya terhubung dengan sebuah sisi. Dapat dikatakan, jika ada v_1 dan v_2 yang bertetangga, maka harus ada sisi (v_1,v_2)

e. Bersisian (*Incident*)

Untuk sembarang sisi $e = (v_1,v_2)$, sisi e dikatakan bersisian dengan simpul v_1 dan simpul v_2.

f. Simpul Terpencil (*Isolated Vertex*)

Simpul terpencil adalah simpul yang tidak mempunyai sisi yang bersisian dengannya. Dengan kata lain, simpul ini ialah simpul yang tidak satupun bertetangga dengan simpul-simpul lain.

g. Graf Kosong (*Empty Graf*)

Graf kosong yaitu graf yang himpunan sisinya merupakan himpunan kosong.

h. Lintasan (*Path*)

Lintasan yang panjangnya n dari simpul awal v_0 ke simpul akhir v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v_0, e_1, v_1, e_2, …, v_{n-1}, e_n, v_n sedemikian sehingga $e_1=(v_0,v_1)$, $e_2=(v_1,v_2)$, …, $e_n=(v_{n-1},v_n)$ adalah sisi-sisi dari graf G.

i. Siklus (*Cycle*) atau Sirkuit (*Circuit*)

Lintasan yang berawal dan berakhir pada simpul yang sama disebut siklus atau sirkuit.

j. Terhubung (*Connected*)
Graf disebut graf terhubung jika untuk setiap pasang simpul v_1 dan v_2 di dalam himpunan V terdapat lintasan dari v_1 ke v_2, yang juga berarrti ada lintasan dari v_2 ke v_1 (untuk graf berarah).

k. Upagraf (*Subgraf*) dan Komplemen Upagraf
Misalkan G = {V,E} sebuah graf, G_1 = {V_1,E_1} dikatakan upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.
Komplemen dari upagraf G_1 terhadap G adalah graf G_2 = {V_2,E_2} sedemikian sehingga $E_2 = E - E_1$ dan V_2 adalah himpunan simpul yang anggota-anggota E_2 bersisian dengannya.

l. Upagraf Merentang (*Spanning Subgraf*)
Upagraf G_1 = {V_1,E_1} dari G = {V,E} dikatakan upagraf merentang jika V_1=V, G_1 mengandung semua simpul G.

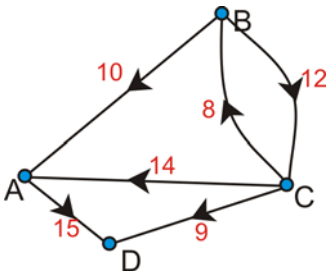
m. Pohon (*Tree*)
Pohon adalah graf tak berarah terhubung yang tidak mempunyai sirkuit.

3. Representasi Graf

Untuk maksud pemodelan graf, pemrosesan graf dalam program komputer graf harus direpresentasikan di dalam memori. Ada beberapa cara dalam merepresentasikan sebuah graf:

a. Himpunan Simpul dan Sisi (*Set*)
Representasi ini adalah permodelan yang paling mendekati dengan definisi graf itu sendiri.

Secara umum graf dengan representasi himpunan dapat digambarkan sebagai berikut:
- Graf memiliki dua komponen, dalam hal ini tipe dari graf adalah tipe bentukan (**`record`**) dengan himpunan simpul dan sisi sebagai komponennya.
- Himpunan simpul dapat direpresentasikan dalam himpunan yang beranggotakan huruf-huruf yang melambangkan setiap simpul (**`set of char`**)
- Himpunan sisi beranggotakan sisi-sisi yang juga merupakan tipe bentukan dari 2 simpul. Jika graf merupakan graf berbobot, ditambahkan 1 komponen lagi pada sisi, yaitu bobot dari sisi itu.



Gambar 5: Graf G_4

Graf G_4 dapat direpresentasikan sebagai berikut:
G = (V,E)
V = {'A','B','C','D'}
E = {('A','D',15), ('B','A',10), ('B','C',12), ('C','A',14), ('C','B',8),('C','D',9)}

Karena keterbatasan penggunaan himpunan dalam bahasa Pascal, kita dapat menggunakan larik (**`array`**) sebagai pengganti himpunan. Sehingga dapat dituliskan:
E_1 = ('A','D',15)
E_2 = ('B','A',10)
E_3 = ('B','C',12)
E_4 = ('C','A',14)
E_5 = ('C','B',8)
E_6 = ('C','D',9)

b. Matriks Ketetanggaan (*Matrix*)
Dalam merepresentasikan graf dengan matriks kita dapat menggunakan larik 2 dimensi. Dimana setiap indeks yang digunakan melambangkan simpul-simpul yang ada dan isi dari larik itu adalah kondisi keterhubungan antara simpul atau bobot dari sisi yang menghubungkan 2 simpul tersebut.

Untuk graf G_4 diatas, matriks ketetanggaan yang bisa dituliskan:

idx:	A	B	C	D
A	∞	∞	∞	15
B	10	∞	12	∞
C	8	14	∞	9
D	∞	∞	∞	∞

Lambang tak hingga (∞) berarti tidak ada sisi yang menghubungkan kedua simpul tersebut. Dengan representasi matriks ini, hubungan antar simpul lebih terlihat. Graf yang direpresentasikan pun lebih tergambar dibenak kita. Oleh karena itu, matriks ketetanggaan lebih sering digunakan.

c. Senarai Ketetanggan (*List*)
Hampir sama dengan matriks ketetanggaan, namun kali ini kita menggunakan kombinasi antara larik dengan senarai berkait (**`linked list`**).

Gambarannya dalah sebagai berikut:
- Kita menggunakan simpul sebagai indeks larik.
- Larik itu merupakan list yang menunjuk ke list lain yang berisi info berupa simpul yang bertetanggan dengan indeks. List ini tidak menunjuk kemanapun (**`NULL`**) jika indeks tidak memiliki tetangga dengan simpul yang lain
- Kembali, list yang ditunjuk tadi menunjuk list lain yang bertetanggan dengan indeks.
- Hal ini terus diulang sebanyak simpul yang bertetangga dengan indeks tersebut.

Untuk graf G_4 representasinya adalah sebagai berikut:

A : D
B : A, C
C : A, B, D
D : -

Karena tidak bisa menjelaskan bobot dari sisi yang menghubungkan simpul-simpul graf, representasi ini tidak digunakan dalam pembahasan kali ini.

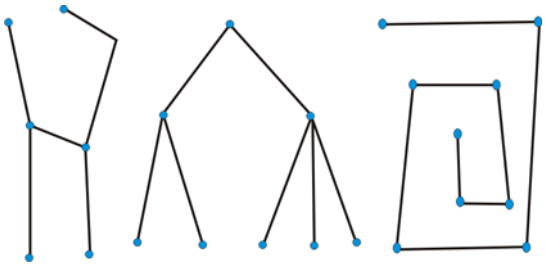
4. Pohon

Sesuai permasalahan yang akan diketengahkan, kita akan membahas graf yang tidak berarah serta tidak mempunyai sirkuit. Sebelum beranjak ke permasalahan utama, terlebih dulu kita sajikan beberapa sifat-sifat pohon:

Misalkan G = (V,E) adalah graf tak berarah sederhana dengan jumlah simpulnya n, maka semua pernyataan dibawah ini adalah ekuivalen:

a. G adalah pohon.
b. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
c. G terhubung dan memiliki m = n–1 buah sisi.
d. G tidak mengandung sirkuit dan memiliki m=n–1 buah sisi.
e. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuatnya memiliki satu sirkuit.
f. G terhubung dan semua sisinya adalah jembatan, yaitu sisi yang bila dihapus menyebabkan graf terpecah menjadi dua komponen.

Semua butir diatas juga dapat diangap sebagai definisi dari pohon.



Gambar 6: Tiga Pohon yang Membentuk Hutan

Kita dapat mengatakan bahwa beberapa pohon dapat membentuk hutan. Hutan adalah kumpulan pohon yang saling lepas.

4.1 Pohon Merentang (Spanning Tree)

Misalkan G = (V,E) adalah graf tak berarah terhubung yang bukan pohon, berarti G memiliki beberapa sirkuit. G dapat diubah menjadi pohon T = (V_1,E_1) dengan memutuskan sirkuit-sirkuit yang ada. Caranya, mula-mula dipilih sebuah sirkuit, lalu hapus satu buah sisi dari sirkuit ini. G akan tetap terhubung dan jumlah sirkuitnya berkurang satu. Bila proses ini dilakukan berulang-ulang sampai semua sirkuit di G hilang, maka G menjadi sebuah pohon T, yang dinamakan pohon merentang.

Walaupun semua pohon bisa dikatakan graf merentang, pohon merentang merupakan graf dengan semua simpul pada simpul T sama dengan semua simpul pada G dan himpunana sisi-sisi pada pohon T merupakan himpunan bagian dari himpunana sisi-sisi pada graf G. Dengan kata lain, jika upagraf dari graf terhubung berbentuk pohon, maka upagraf rentang tersebut dinamakan pohon merentang.

4.2 Pohon Merentang Minimum (*Minimum Spanning Tree / MST*)

Jika G adalah graf berbobot maka bobot pohon merentang T dari G didefinisikan sebagai jumlah bobot semua sisi di T. Di antara semua pohon merentang di G, pohon merentang yang berbobot minimum dinamakan pohon merentang minimum. Untuk graf yang memiliki lebih dari satu sisi yang memiliki bobot yang sama, ada kemungkinan, tidak hanya ada satu pohon merentang minimum yang dapat dibuat. Dalam perhitungan, hal ini tidak terlalu dipermasalahkan.

Pohon merentang minimum mempunyai terapan yang luas dalam praktek. Hal ini yang akan dibahas dalam makalah ini. Ada banyak contoh kasus yang membutuhkan permodelan pohon merentang minimum sebagai solusinya. Kasus ini biasanya berhubungan dengan masalah jaringan (*network*), bagaimana kita harus membangun suatu jaringan dengan harga pemakaian sumber daya sekecil mungkin. Dengan permodelan ini dan teknik pemecahan masalah yang sudah dimiliki, permasalahan mengenai pembangunan jaringan dapat terselesaikan.

4.3 Algoritma Pohon Merentang Minimum

Sebenarnya ada banyak teknik pemecahan masalah pembuatan pohon merentang minimum ini. Namun yang akan dibahas kali ini hanya 2 teknik atau algoritma yang sudah umum digunakan saja.

4.3.a Algoritma Prim (Prim's Algorithm)

Algoritma Prim adalah sebuah algoritma dalam teori graf yang bisa mendapatkan pohon merentang minimum dari sebuah graf yang diberikan. Algoritma ini ditemukan pada tahun 1930 oleh seorang matematikawan Voljtêch Jarnik, dan lalu secara terpisah oleh ahli komputer Robert C. Prim di tahun 1957, kemudian dikembangkan lagi oleh Dijkstra di tahun 1959. Algoritma ini tergolong

dalam algoritma g*reedy*, dimana kita mencari nilai minimum semua dengan mencari nilai minimum perbagian, berharap itu juga merupakan nilai minimum total.

Algoritmanya:
1. Buat pohon T dengan satu simpul, ambil secara acak dari graf G.
2. Ulang (banyak simpul G = banyak simpul T)
   a. Cari e = (x,y) yang memiliki bobot minimum dengan x atau y ada di T namun tidak keduanya.
   b. Simpul yang tidak ada di T tersebut, masukkan ke dalam T.
   c. Masukkan e ke dalam T.

4.3.b Algoritma Kruskal (Kruskal's Algorithm)

Algoritma Kruskal adalah juga tergolong algoritma *greedy* dalam teori graf yang digunakan untuk mencari pohon merentang minimum. Algoritma ini pertama kali muncul pada tahun 1956 dalam sebuah tulisan yang ditulis oleh Joseph Kruskal.

Algoritmanya:
1. Himpunan sisi dari G diurutkan membesar sesuai bobot sisi tersebut.
2. Buat T dengan memasukkan 1 sisi terpendek dari G tersebut.
3. Ulang (banyak sisi T = (banyak simpul G) -1)
   a. Ambil sisi selanjutnya dari G.
   b. Jika sisi itu tidak membuat sirkuit di T
      - Masukkan sisi itu ke T.
      - Masukkan simpul-simpul sisi itu ke T.

5. Perbandingan 2 Algoritma

5.1 Studi Kasus

Saat teknologi komunikasi mulai berkembang, pemerintah ingin mengganti sistem jaringan kabel telepon yang sudah ada di satu daerah dengan kabel yang baru. Namun, karena ini baru merupakan awal, pemerintah tidak mau ambil resiko mengganti semua kabel yang menghubungkan wilayah satu dengan yang lain. Ia hanya akan memasang jaringan di wilayah itu jika wilayah itu memang belum tersentuh jaringan yang baru.

Bantulah pemerintah untuk membangun jaringan yang menghubungkan semua wilayah dari jaringan lama yang diberikan. Dengan asumsi, semakin panjang kabel yang dipasang, semakin mahal biaya yang harus dikeluarkan, buatlah jaringan yang dibangun ini memakan biaya sesedikit mungkin.

Masukkan:
Masukkan terdiri dari beberapa baris bilangan. Baris pertama terdiri dari 2 angka V dan E, masing-masing menyatakan banyaknya wilayah dan banyaknya jalur yang lama. E baris selanjutnya menggambarkan bagaimana wilayah-wilayah yang ada dihubungkan oleh jalur-jalur yang lama. Wilayah diberi nomor mulai dari 1 hingga V. Diberikan pula W pada tiap jalur yang menyatakan panjang jalur tersebut. Batasannya, 1<V<=100 ; 1<=E<=1000 ; 1<=W<=30000.

Contoh 1:
```
4 5
1 2 10
1 3 14
1 4 15
2 3 12
3 4 9
```

Contoh 2:
```
7 11
1 4 5
3 5 5
4 6 6
2 4 9
1 2 7
2 5 7
2 3 8
4 5 15
5 6 8
5 7 9
6 7 11
```

Keluaran:
Hasil keluaran hampir sama dengan masukkan. Cukup tampilkan saja panjang kabel minimum yang harus dibuat, kemudian, tampilkan, wilayah, wilayah mana saja yang harus dihubungkan, beserta dengan panjangnya. Hasil maupun urutan dari jaringan yang terbentuk boleh berbeda, namun panjang minimum yang didapat harus sesuai.

Contoh 1:
```
31
1 2 10
2 3 12
3 4 9
```

Contoh 2:
```
39
1 4 5
3 5 5
4 6 6
1 2 7
2 5 7
5 7 9
```

5.2 Solusi

Dengan algoritma yang sudah diberikan, kita bisa menyelesaikan permasalahan ini, salah satunya dengan menggunakan bahasa pemrograman Pascal. Dua contoh kode di bawah ini merupakan contoh solusi dari permasalahan pemasangan kabel telepon diatas. Dalam kode yang dibuat dibawah ini, graf dan pohon direpresentasikan dalam representasi himpunan. Hal ini dilakukan untuk mempermudah pengkodean.

```
program prim;
{menyelesaikan permasalahan pohon
merentang minimum dengan algoritma prim}

type
  sEdge = record
    iv,tv,we : integer;
  end;
  {iv : simpul asal
   tv : simpul terminal
   we : bobot sisi}

  list = array[0..100] of integer;
  tEdge = array[0..1000] of sEdge;

  tGraph = record
    nVer,nEd : integer;
    ve : list;
    ed : tEdge;
  end;
  {nVer : banyak simpul
   nEd  : banyak sisi
   ve   : himpunan simpul
   ed   : himpunan sisi
  }

var
  graph : tGraph;   {graf awal}
  mst   : tGraph;   {pohon merentang
minimum = hasil}
  total : integer;  {bobot pohon
merentang minimum}

function member(x,n : integer; L:list) :
boolean;
{mengecek keberadaan elemen x dalam L}
var i : integer;
begin
  i := 1;
  while (x<>L[i]) and (i<n) do
    inc(i);

  member := (x = L[i]);
end;


procedure baca(var G : tGraph);
{membaca masukkan graf}
var
  i,x,y,w : integer;
  fi : text;

begin
  readln(G.nVer,G.nEd);
  G.ed[0].we := 32760;

  for i:=1 to G.nEd do
  begin
     readln(x,y,w);

     G.ed[i].iv := x;
     G.ed[i].tv := y;
     G.ed[i].we := w;
     G.ve[i] := i;
  end;
end;
```

```
procedure span(G : tGraph; var T :
tGraph);
{mendapatkan pohon merentang minimum T
dari graf G}
var
  i, mEd : integer;

begin
  {initial}
  total := 0;
  T.ve[1] := G.ve[1];
  T.nVer := 1;

  while (T.nVer<G.nVer) do
  begin
    mEd := 0;

    for i:=1 to G.nEd do
      if (member(G.ed[i].iv,T.nVer,T.ve)
xor member(G.ed[i].tv,T.nVer,T.ve)) then
        if (G.ed[i].we<G.ed[mEd].we) then
          mEd := i;

    T.ed[T.nVer] := G.ed[mEd];
    total := total + G.ed[mEd].we;

    inc(T.nVer);
    if member(G.ed[mEd].iv,T.nVer,T.ve)
then
      T.ve[T.nVer] := G.ed[mEd].tv
    else
      T.ve[T.nVer] := G.ed[mEd].iv;
  end;

  T.nEd := T.nVer -1;
end;

procedure tulis(G : tGraph);
{menuliskan hasil pohon dan bobot yang
diperoleh}
var i,j : integer;

begin
  writeln(total);
  for i:=1 to G.nEd do
    writeln(G.ed[i].iv,' ',G.ed[i].tv,'
',G.ed[i].we);
  writeln;
end;

begin
  baca(graph);
  span(graph,mst);
  tulis(mst);
end.
```

```
program kruskal;
{menyelesaikan permasalahan pohon
merentang minimum dengan algoritma
kruskal}

type
  sEdge = record
    iv,tv,we : integer;
  end;
  {iv : simpul asal
   tv : simpul terminal
   we : bobot sisi}

  list = array[0..100] of integer;
  tEdge = array[0..1000] of sEdge;

  tGraph = record
    nVer,nEd : integer;
    ve : list;
    ed : tEdge;
  end;
  {nVer : banyak simpul
   nEd  : banyak sisi
   ve   : himpunan simpul
   ed   : himpunan sisi
  }

var
  graph : tGraph;   {graf awal}
  mst   : tGraph;   {pohon merentang
minimum = hasil}
  total : integer;  {bobot pohon
merentang minimum}


function member(x,n : integer; L:list) :
boolean;
{mengecek keberadaan elemen x dalam L}
var i : integer;
begin
  i := 1;
  while (x<>L[i]) and (i<n) do
    inc(i);

  member := (x = L[i]) and (n<>0);
end;

procedure swap(var a,b: sEdge);
{menukar dua sisi}
var t : sEdge;

begin
  t := a;
  a := b;
  b := t;
end;
```

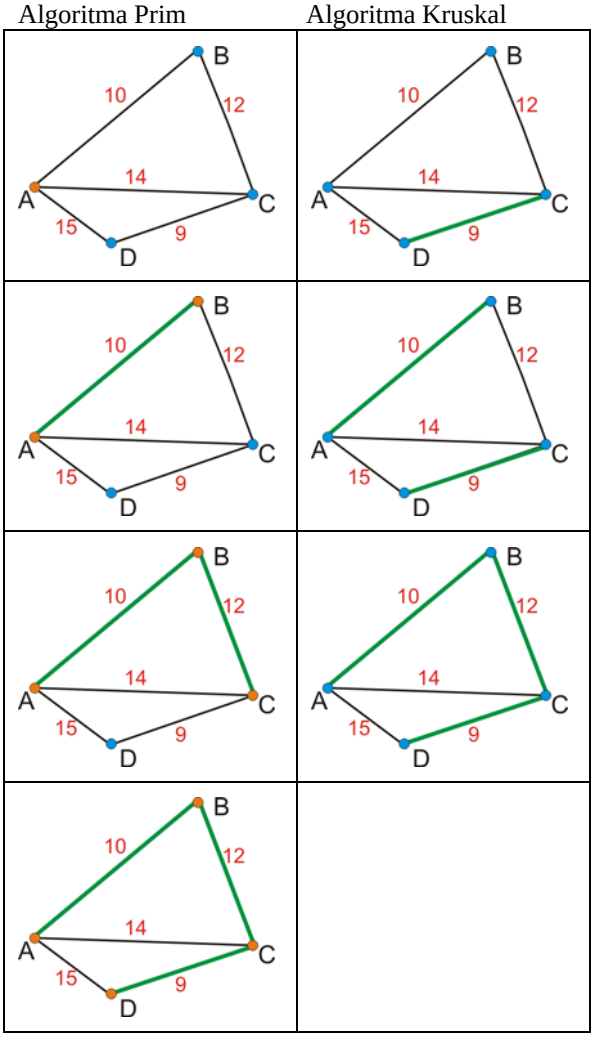
```
procedure sort(var data: tEdge; i, j :
integer);
{sorting sisi secara menaik dengan
quicksort}
var k,l,p : integer;

begin
  p := data[i].we;
  k := i+1;
  l := j;

  while ((data[k].we<=p) and (k<j)) do
    inc(k);
  while ((data[l].we>p) and (l>i)) do
    dec(l);

  while (k<l) do begin
    swap(data[k],data[l]);

    while ((data[k].we<=p) and (k<j)) do
      inc(k);
    while ((data[l].we>p) and (l>i)) do
      dec(l);
  end;
  swap(data[i],data[l]);

  if (i<l) then sort(data,i,l-1);
  if (l<j) then sort(data,l+1,j);
end;


procedure baca(var G : tGraph);
{membaca masukkan graf}
var
  i,x,y,w : integer;
  fi : text;

begin
  readln(G.nVer,G.nEd);
  G.ed[0].we := 32760;

  for i:=1 to G.nEd do
  begin
     readln(x,y,w);

     G.ed[i].iv := x;
     G.ed[i].tv := y;
     G.ed[i].we := w;
     G.ve[i] := i;
  end;
end;

procedure tulis(G : tGraph);
{menuliskan hasil pohon dan bobot yang
diperoleh}
var i,j : integer;

begin
  writeln(total);
  for i:=1 to G.nEd do
    writeln(G.ed[i].iv,' ',G.ed[i].tv,'
',G.ed[i].we);
  writeln;
end;
```

```
procedure span(G : tGraph; var T :
tGraph);
{mendapatkan pohon merentang minimum T
dari graf G}
var
  i,j : integer;
  path: list;
  nol : integer;
{tree disini tidak terhubung}

  function cycle : boolean;
  begin
    cycle:= (member(G.ed[i].iv,
T.nVer,T.ve) and member(G.ed[i].tv,
T.nVer,T.ve));
    {perlu dibuat pengecekan yang labih
baik}
  end;

begin
  sort(G.ed,1,G.nEd);

  {inisialisasi}
  i := 1;
  T.nEd := 1;
  T.ed[1] := G.ed[1];

  T.nVer := 2;
  T.ve[1] := G.ed[1].iv;
  T.ve[2] := G.ed[1].tv;

  repeat
    inc(i);

    nol:= 0;

    if not(cycle) then
    begin
      {menggabungkan pohon}
      if not(member(G.ed[i].iv,
T.nVer,T.ve)) then begin
        inc(T.nVer);
        T.ve[T.nVer] := G.ed[i].iv;
      end;

      if not(member(G.ed[i].tv,
T.nVer,T.ve)) then begin
        inc(T.nVer);
        T.ve[T.nVer] := G.ed[i].tv;
      end;

      inc(T.nEd);
      T.ed[T.nEd] := G.ed[i];
      total := total + G.ed[i].we;
    end;
  until (T.nEd = G.nVer-1);
end;

begin
  baca(graph);
  span(graph,mst);
  tulis(mst);
end.
```

5.3 Realisasi

Dengan algoritma yang diberikan tersebut serta dengan contoh masukkan yang sudah diberikan, kita bisa menggambarkan bagaimana proses pembentukkan pohon merentang minimum dari dua algoritma yang berbeda tersebut.

Algoritma Prim Algoritma Kruskal



Gambar 7: Menggambarkan bagaimana kedua algoritma ini berjalan untuk mendapatkan pohon merentang minimum untuk Contoh 1.

Untuk Contoh 2, bisa kita gambarkan dengan teknik yang berbeda.

Langkah	Algoritma Prim		Algoritma Kruskal	
	Terpilih (simpul)	Sisi terbentuk	Teripilih (sisi)	Simpul masuk
0	1	-	(1,4)	{1,4}
1	4	{(1,4)}	(3,5)	{1,3,4,5}
2	6	{(1,4), (4,6)}	(4,6)	{1,3,4,5, 6}
3	2	{(1,4), (4,6), (1,2)}	(1,2)	{1,2,3,4, 5,6}
4	5	{(1,4), (4,6), (1,2), (2,5)}	(2,5)	(1,2,3,4, 5,6)
5	3	{(1,4), (4,6), (1,2), (2,5), (5,3)}	(2,3)	ditolak

6	7	{(1,4), (4,6), (1,2), (2,5), (5,3), (5,7)}	(5,7)	{1,2,3,4, 5,6,7}

Tabel 1: Perbandingan Eksekusi untuk Contoh 2

5.4 Analisa Algoritma

Jika kita melihat algoritma Prim dan Kruskal yang telah disebutkan di atas serta dengan melihat potongan kode yang telah dibuat, kita bisa memeperkirakan waktu yang dibutuhkan untuk menjalankan program tersebut.

Untuk algoritma Prim:

Lkh	T(E,V)	Keterangan
1	1	inisialisasi
2	V	Perulangan(i) sebanyak simpul
a	i	Masuk perulangan
b	1	Masuk dalam perulangan
c	1	Masu dalam perulangan

Tabel 2: Analisa Algoritma Prim

Jadi perkiraan total waktu yang dibutuhkan :

$T_P(V,E) = 1 + V(i+1+1)$

$$= 1 + \frac{V(V+1)}{2} + 2V$$

$= ½V^2 + 2½V + 1$

Sehingga kompleksitas algoritmanya: $O(V^2)$, dengan V menyatakan banyaknya simpul.

Untuk algoritma Kruskal:

Lkh	T(E,V)	Keterangan
1	$E\ ^2\log E$	Dengan *Quicksort*
2	1	Inisialisasi
3	E	Perulangan(i), maksimum sebanyak sisisnya
A	1	Masuk dalam perulangan
B	1	Pengecekan Sirkuit
-	1	Walaupun dlm perulangan E, ini hanya terjadi maksimum V kali, terkait dg pengecekan sirkuit.
-	1	

Tabel 3: Analisa Algoritma Kruskal

Jadi perkiraan total waktu yang dibutuhkan :

$T_K(V,E) = E\ ^2\log E + 1 + E(1+1) + V(1+1)$

$= E\ ^2\log E + 2V + 2E + 1$

Sehingga kompleksitas algoritmanya:
O(E log E + V), dengan E menyatakan banyaknya sisi dan V menyatakan banyaknya simpul.

Sebenarnya, akan dilakukan percobaan dengan menghitung waktu eksekusi dari program yang dijalankan dengan memasukkan beberapa contoh masukkan yang berbeda-beda.

Namun karena keterbatasan program, program tidak bisa mengelola masukkan yang besar. Padahal, dengan masukkan yang sesuai untuk program tersebut, waktu eksekusi masih tidak bisa terlihat, kurang dari seper seratus detik.Oleh karena itu, untuk melihat kecepatan dari program ini, cukup melihat waktu perkiraan T(V,E) saja.

No	V	E	T_{Prim}	$T_{Kruskal}$	Keterangan
1	50	100	1.376	1.067	Umum
2	50	1225	1.376	17.367	Graf Lengkap
3	1226	1225	754.604	17.367	Garis

Tabel 4: Perbandingan Lama Eksekusi Program

Dari tabel terlihat bahwa secara umum, algoritma Kruskal bisa berjalan lebih cepat dibanding algoritma Prim. Namun untuk graf lengkap atau yang mendekati lengkap, dimana setiap simpul terhubungkan dengan semua simpul yang lain Prim menunjukkan perbedaan yang cukup besar. Walaupun kemudian, saat graf sepi akan sisi namun dengan sangat banyak simpul, Kruskal kembali berjaya dalam lomba kecepatan program.

Sebenarnya hal ini sudah terlihat dari tahap-tahap algoritma itu sendiri. Prim lebih berorintasi kepada pencarian simpul sedangkan Kruskal pada pencariam sisi, dimana sisi-sisi tersebut harus diurutkan dan ini memakan waktu yang lumayan lama, apalagi kalau pengurutan menggunakan algoritma standar dengan $O(n^2)$. Sehingga saat pengujian kasus untuk masukkan yang berbeda, algoritma Prim unggul saat graf memiliki banyak sisi karena banyaknya sisi hampir tidak diperhitungkan di sini. Namun saat simpul yang diberikan banyak tetapi sisi yang menghubungkan simpul-simpul itu hanya sedikit, algoritma Kruskal bisa lebih dihandalkan, apalagi dengan kecepatan logaritmik dari proses pengurutannya.

5. Kesimpulan

Kesimpulan yang dapat dimbil dari studi dan perbandingan dua algoritma pencarian pohon merentang minimum adalah:

1. Algoritma Prim dan Algoritma Kruskal dapat menyelesaikan permasalahan pencarian pohon merentang minimum dengan tepat.
2. Algoritma Prim lebih efisien dibanding algoritma Kruskal saat graf yang diberikan memiliki banyak sisi dengan simpul yang sedikit (graf lengkap).
3. Algoritma Kruskal lebih efisien dibanding algoritma Prim saat graf yang diberikan memiliki banyak simpul dengan sisi yang sedikit.

DAFTAR PUSTAKA

[1] Algorithms Design Manual. (1997), *http://www2.toki.or.id/book/AlgDesignManual/* Tanggal akses: 2 Januari 2006 pukul 10:30.

[2] Brassard, Gilles. (1995). Fundamental of Algorithms. Prentice Hall, New Jersey.

[3] English Wikipedia, Ensiklopedia Bebas. (1997), *http://en.wikipedia.org/wiki/* Tanggal akses: 2 Januari 2006 pukul 11:00.

[4] Munir, Rinaldi. (2005). Bahan Kuliah IF2151 Matematika Diskrit. Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[5] Stony Brook Algorithm. (1997), *http://www.cs.sunusb.edu/* Tanggal akses: 2 Januari 2006 pukul 11:30.