MEMBANDINGKAN KEMANGKUSAN ALGORITMA PRIM DAN ALGORITMA KRUSKAL DALAM PEMECAHAN MASALAH POHON MERENTANG MINIMUM

Pudy Prima (13508047)

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganeca 10 Bandung
E-mail: if18047@students.if.itb.ac.id

ABSTRAK

Algoritma Prim dan algoritma Kruskal merupakan algoritma yang paling umum digunakan dalam pemecahan masalah pohon merentang minimum. Secara umum, kedua algoritma ini akan memberikan keluaran pohon merentang minimum yang sama bentuknya. Namun, ternyata setiap algoritma tersebut memiliki kelebihan dan kekurangan masing-masing. Kelebihan dan kekurangan ini memungkinkan pengguna untuk memilih algoritma mana yang lebih mangkus untuk diterapkan pada graf jenis tertentu. Dalam makalah kali ini akan dibahas dan dibandingkan kemangkusan kedua algoritma tersebut, yaitu algoritma Prim dan algoritma Kruskal.

Kata kunci: *algoritma Prim, algoritma Kruskal, pohon merentang minimum, graf, mangkus*

1. PENDAHULUAN

Aplikasi graf dan pohon banyak diterapkan pada pemodelan masalah dalam kehidupan sehari-hari. Salah satu bahasan yang cukup penting dalam teori graf dan pohon adalah pohon merentang minimum (*minimum spanning tree* / MST). Pohon merentang minimum misalnya digunakan dalam menyelesaikan masalah penentuan panjang kabel optimum untuk merancang jaringan komputer di suatu area. Contoh lain misalnya untuk menentukan rute terpendek untuk menjelajahi kota sehingga sejumlah titik tertentu di kota tersebut tepat dilewati satu kali.

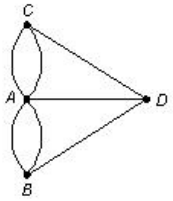
Ada beberapa cara yang lazim digunakan dalam memecahkan masalah pohon merentang minimum ini. Algoritma Prim dan algoritma Kruskal merupakan dua cara yang paling umum digunakan untuk membentuk pohon merentang minimum. Kedua algoritma ini terbukti mampu menghasilkan pohon merentang minimum. Namun dalam praktiknya, pengguna seringkali merasa sulit untuk memilih algoritma mana yang lebih baik dan lebih mangkus untuk diterapkan pada graf jenis tertentu. Oleh karena itu, pembahasan mengenai perbandingan kemangkusan antara algoritma Prim dan algoritma Kruskal dirasa penting untuk untuk menentukan algoritma mana yang lebih baik digunakan dalam pemecahan masalah pohon merentang minimum.

2. PEMBAHASAN

2.1 Teori Graf dan Pohon

2.1.1 Graf

Graf G didefinisikan sebagai pasangan himpunan (V,E), yang dalam hal ini, V merupakan himpunan tidak kosong dari simpul-simpul (*vertices* atau *node*) dan E adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul. Graf dapat ditulis dalam notasi G=(V,E). Sebuah graf dimungkinkan tidak memiliki sisi satu buah pun, namun harus memiliki minimal satu buah simpul.



Gambar 1. Graf G

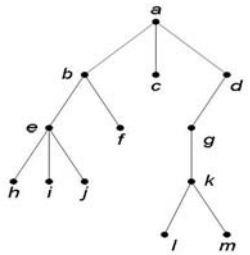
Gambar 1 merupakan contoh graf, yaitu graf G yang terdiri dari
V = {A,B,C,D} dan
E = {(C,A),(C,A),(A,B),(A,B),(C,D),(A,D),(B,D)}

Graf dapat dikelompokkan menjadi beberapa jenis bergantung pada sudut pandang pengelompokkannya, yaitu berdasarkan ada tidaknya sisi ganda atau sisi kalang, berdasarkan jumlah simpul, atau berdasarkan orientasi arah sisi.

1. Graf Sederhana (*simple graph*)
   Graf sederhana adalah graf yang tidak mengandung gelang maupun sisi ganda.
2. Graf Tak-Sederhana (*unsimple graph*)
   Graf tak-sederhana merupakan graf yang mengandung sisi ganda atau gelang. Graf yang mengandung sisi ganda disebut graf ganda (*multigraph*), sedangkan graf yang mengandung gelang disebut graf semu (*pseudograph*).
3. Graf Berhingga (*limited graph*)
   Graf berhingga adalah graf yang jumlah simpulnya berhingga.
4. Graf Tak-Berhingga (*unlimited graph*)
   Graf tak-berhingga adalah graf yang jumlah simpulnya tidak berhingga.
5. Graf Berarah (*directed graph*)
   Graf berarah adalah graf yang setiap sisinya diberi orientasi arah. Pada grah berarah, gelang diperbolehkan, tetapi sisi ganda tidak. Namun, ada pula yang disebut graf ganda berarah. Pada graf ganda berarah, gelang dan sisi ganda diperbolehkan ada.
6. Graf Tak-Berarah (*undirected graph*)
   Graf tak-berarah adalah graf yang sisinya tidak mempunyai orientasi arah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan.
7. Graf Terhubung (*connected graph*)
   Graf tak-berarah G desebut graf terhubung (*connected graph*) jika untuk setiap pasang simpul v_i dan v_j terdapat lintasan dari vi ke vj (yang juga berarti ada lintasan dari v_j ke v_i). Jika tidak, maka G disebut graf tak-terhubung (*disconnected graph*).
8. Graf berbobot (*weighted graph*)
   Graf berbobot merupakan graf yang setiap sisinya diberi bobot (harga).

2.1.2 Pohon

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Pohon dalam definisi ini sering juga disebut pohon bebas (*free tree*), untuk membedakannya dengan pohon berakar (*rooted tree*), yaitu pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah.



Gambar 2. Pohon

2.1.3 Pohon Merentang (*Spanning Tree*)

Misalkan G = (V,E) adalah graf tak-berarah terhubung yang bukan pohon, yang berarti di G terdapat sirkuit. G dapat diubah menjadi pohon T = (V1,E1) dengan cara memutuskan sirkuit-sirkuit yang ada. Setiap graf terhubung mempunyai paling sedikit satu buah pohon merentang.

2.1.4 Pohon Merentang Minimum

Jika G adalah graf berbobot, maka bobot pohon merentang dari G adalah jumlah bobot semua sisi pada pohon merentang tersebut. Pohon merentang yang mempunyai bobot minimum disebut pohon merentang minimum (*minimum spanning tree* / MST).

Terdapat beberapa cara pembentukan pohon merentang minimum. Cara pertama adalah dengan mendata semua pohon merentang dari suatu graf terhubung, kemudian dicari yang memiliki bobot terkecil. Cara ini merupakan cara yang paling mudah dimengerti, namun paling menguras waktu dan tenaga.

Secara umum, terdapat dua buah algoritma yang dapat digunakan untuk membangun sebuah pohon merentang minimum. Yang pertama adalah algoritma Prim, dan yang kedua adalah algoritma Kruskal.

2.2 Algoritma Prim

Konsep dasar yang digunakan dalam algoritma Prim adalah dalam setiap langkah, pilih sisi dari graf G yang berbobot minimum, yang terhubung dengan pohon merentang T yang telah terbentuk, dan tidak membentuk sirkuit.

Langkah-langkah algoritma Prim :

1. Ambil sisi dari graf G yang berbobot minimum, masukkan ke dalam T.
2. Pilih sisi (u,v) yang mempunyai bobot minimum dan bersisian dengan T, tetapi (u,v) tidak membentuk sirkuit di T. Tambahkan (u,v) ke dalam T.
3. Ulangi langkah 2 sampai pohon merentang minimum terbentuk, yaitu setelah mengalami pengulangan sebanyak n-2 kali (n adalah jumlah simpul graf G).

Penulisan algoritma Prim dalam bentuk notasi algoritmik (*pseudocode*) :

```
procedure Prim(input G : graf, output T : pohon)
{ Membentuk pohon merentang minimum T dari graf terhubung G.
  Masukan: graf-berbobot terhubung G = (V, E), yang mana |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
```

2.3 Algoritma Kruskal

Konsep dasar yang digunakan dalam algoritma Kruskal adalah pada setiap langkah, pilih sisi dari graf G yang berbobot minimum, tetapi sisi tersebut tidak membentuk sirkuit di T.

Langkah-langkah algoritma Kruskal:

1. Lakukan pengurutan terhadap setiap sisi di graf G mulai dari sisi dengan bobot terkecil.
2. Pilih sisi (u,v) yang mempunyai bobot minimum yang tidak membentuk sirkuit di T. Tambahkan (u,v) ke dalam T.
3. Ulangi langkah 2 sampai pohon merentang minimum terbentuk, yaitu ketika sisi di dalam pohon merentang T berjumlah n-1 (n adalah jumlah simpul graf G).

Penulisan algoritma Kruskal dalam bentuk notasi algoritmik (*pseudocode*) :

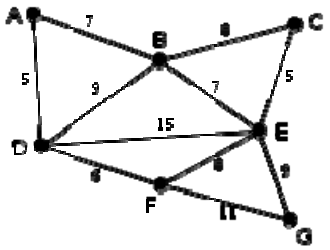
```
procedure Kruskal(input G : graf, output T : pohon)
{ Membentuk pohon merentang minimum T dari graf terhubung G.
  Masukan: graf-berbobot terhubung G = (V, E), yang mana |V| = n
  Keluaran: pohon rentang minimum T = (V, E') }
Deklarasi
  i, p, q, u, v : integer

Algoritma
  {Asumsi: sisi-sisi dari graf sudah diurut menaik berdasarkan
  bobotnya }
  T ← {}
  while jumlah sisi di dalam T < n-1 do
    Pilih sisi (u,v) dari E yang bobotnya terkecil
    if (u,v) tidak membentuk siklus di T then
       T ← T ∪ {(u,v)}
    endif
  endfor
```

3. UJI KASUS

3.1 Uji Kasus I

Diberikan sebuah graf terhubung G1 yang ditunjukkan pada gambar 3.



Gambar 3. Graf terhubung G1

3.1.1 Aplikasi Algoritma Prim

Langkah-langkah yang dilakukan untuk membuat pohon merentang minimum dari graf G1 menggunakan algoritma Prim adalah sebagai berikut :

1. Pohon merentang T masih kosong. Bandingkan sisi-sisi pada graf G1. Ambil sisi yang memiliki bobot minimum, yaitu sisi (A,D), masukkan ke dalam pohon merentang T.
2. Daftarkan semua sisi yang bersisian dengan sisi pada pohon merentang T, maka didapatkan sisi (A,B), (D,B), (D,E), dan (D,F). Cari sisi yang memiliki bobot minimum, yaitu sisi (D,F). Periksa apakah sisi (D,F) membentuk sirkuit di T. Ternyata tidak, sehingga (D,F) ditambahkan ke dalam T. T sekarang terdiri dari (A,D) dan (D,F) yang terhubung.
3. Ulangi langkah 2 untuk pohon T yang baru, didapatkan sisi yang bersisian dengan pohon T adalah (A,B), (D,B), (D,E), (F,E), dan (F,G). Sisi yang berbobot minimum dan tidak membentuk sirkuit di T adalah sisi (A,B), sehingga pohon T sekarang terdiri dari sisi (A,B), (A,D), dan (D,F) yang saling terhubung.
4. Ulangi langkah 2 sehingga pohon merentang T kini terdiri dari sisi (A,B), (A,D), (D,F), dan (B,E).
5. Ulangi langkah 2 sehingga pohon merentang T kini terdiri dari sisi (A,B), (A,D), (D,F), (B,E), dan (E,C).
6. Ulangi langkah 2 sehingga pohon merentang T kini terdiri dari sisi (A,B), (A,D), (D,F), (B,E), (E,C), dan (E,G).
7. Karena tidak didapatkan lagi sisi yang berbobot minimum namun tidak membentuk sirkuit di T, maka pohon merentang T yang terdiri dari sisi (A,B), (A,D), (D,F), (B,E), (E,C), dan (E,G) merupakan pohon merentang minimum dari graf G1.

3.1.2 Aplikasi Algoritma Kruskal

Langkah-langkah yang dilakukan untuk membuat pohon merentang minimum dari graf G1 menggunakan algoritma Kruskal adalah sebagai berikut :

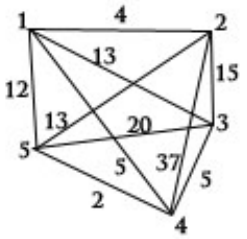
1. Urutkan sisi-sisi pada graf G1 mulai dari sisi yang berbobot paling kecil hingga sisi yang berbobot paling besar, sehingga didapatkan urutan sisi adalah (A,D), (C,E), (D,F), (A,B), (B,E), (B,C), (F,E), (D,B), (E,G), (F,G), dan (D,E).
2. Pohon merentang T masih kosong. Ambil sisi pada graf G1 yang memiliki bobot terkecil, yaitu (A,D).

Tambahkan ke dalam pohon merentang T, sehingga T kini terdiri dari sisi (A,D).

3. Ambil sisi berilutnya yang memiliki bobot terkecil, yaitu sisi (C,E). Periksa apakah (C,E) membentuk sirkuit di T. Ternyata tidak, maka tambahkan (C,E) ke dalam T, sehingga kini T terdiri dari sisi (A,D) dan (C,E).
4. Ulangi langkah 3 sehingga pohon merentang T kini terdiri dari sisi (A,D), (C,E), dan (D,F).
5. Ulangi langkah 3 sehingga pohon merentang T kini terdiri dari sisi (A,D), (C,E), (D,F), dan (A,B).
6. Ulangi langkah 3 sehingga pohon merentang T kini terdiri dari sisi (A,D), (C,E), (D,F), (A,B), dan (B,E).
7. Ulangi langkah 3 untuk sisi berikutnya, yaitu (B,C). Namun karena (B,C) membentuk sirkuit dengan (B,E) dan (C,E), maka pohon merentang T tetap terdiri dari sisi (A,D), (C,E), (D,F), (A,B), dan (B,E).
8. Ulangi langkah 3 untuk sisi berikutnya, yaitu (F,E). Namun karena (F,E) membentuk sirkuit dengan (F,D), (D,A), (A,B), dan (B,E), maka pohon merentang T tetap terdiri dari sisi (A,D), (C,E), (D,F), (A,B), dan (B,E).
9. Ulangi langkah 3 untuk sisi berikutnya, yaitu (D,B). Namun karena (D,B) membentuk sirkuit dengan (D,A) dan (A,B), maka pohon merentang T tetap terdiri dari sisi (A,D), (C,E), (D,F), (A,B), dan (B,E).
10. Ulangi langkah 3 sehingga pohon merentang T kini terdiri dari sisi (A,D), (C,E), (D,F), (A,B), (B,E), dan (E,G).
11. Karena jumlah sisi pada pohon merentang T telah mencapai 6 buah (jumlah simpul, n=7), maka pohon merentang T yang terdiri dari sisi (A,D), (C,E), (D,F), (A,B), (B,E), dan (E,G) merupakan pohon merentang minimum graf G1.

3.2 Uji Kasus II

Diberikan sebuah graf terhubung G2 yang ditunjukkan pada gambar 4.



Gambar 4. Graf terhubung G2

3.2.1 Aplikasi Algoritma Prim

Langkah-langkah yang dilakukan untuk membuat pohon merentang minimum dari graf G2 menggunakan algoritma Prim adalah sebagai berikut :

1. Pohon merentang T masih kosong. Bandingkan sisi-sisi pada graf G2. Ambil sisi yang memiliki bobot minimum, yaitu sisi (5,4), masukkan ke dalam pohon merentang T.
2. Daftarkan semua sisi yang bersisian dengan sisi pada pohon merentang T, maka didapatkan sisi (4,3), (4,2), (4,2), (5,3), (5,2) dan (5,1). Cari sisi yang memiliki bobot minimum, yaitu sisi (4,3). Periksa apakah sisi (4,3) membentuk sirkuit di T. Ternyata tidak, sehingga (4,3) ditambahkan ke dalam T. T sekarang terdiri dari (5,4) dan (4,3) yang terhubung.
3. Ulangi langkah 2 untuk pohon T yang baru, didapatkan sisi yang bersisian dengan pohon T adalah (3,2), (3,1), (4,2), (4,1), (5,2), dan (5,1). Sisi yang berbobot minimum dan tidak membentuk sirkuit di T adalah sisi (4,1), sehingga pohon T sekarang terdiri dari sisi (5,4), (4,3), dan (4,1) yang saling terhubung.
4. Ulangi langkah 2 sehingga pohon merentang T kini terdiri dari sisi (5,4), (4,3), (4,1), dan (1,2).
5. Karena tidak didapatkan lagi sisi yang berbobot minimum namun tidak membentuk sirkuit di T, maka pohon merentang T yang terdiri dari sisi (5,4), (4,3), (4,1), dan (1,2) merupakan pohon merentang minimum dari graf G2.

3.2.2 Aplikasi Algoritma Kruskal

Langkah-langkah yang dilakukan untuk membuat pohon merentang minimum dari graf G2 menggunakan algoritma Kruskal adalah sebagai berikut :

1. Urutkan sisi-sisi pada graf G2 mulai dari sisi yang berbobot paling kecil hingga sisi yang berbobot paling besar, sehingga didapatkan urutan sisi adalah (4,5), (1,2), (1,4), (3,4), (1,5), (1,3), (2,5), (2,3), (3,5), dan (2,4).
2. Pohon merentang T masih kosong. Ambil sisi pada graf G2 yang memiliki bobot terkecil, yaitu (4,5). Tambahkan ke dalam pohon merentang T, sehingga T kini terdiri dari sisi (4,5).
3. Ambil sisi berikutnya yang memiliki bobot terkecil, yaitu sisi (1,2). Periksa apakah (1,2) membentuk sirkuit di T. Ternyata tidak, maka tambahkan (1,2) ke dalam T, sehingga kini T terdiri dari sisi (4,5) dan (1,2).
4. Ulangi langkah 3 sehingga pohon merentang T kini terdiri dari sisi (4,5), (1,2), dan (1,4).
5. Ulangi langkah 3 sehingga pohon merentang T kini terdiri dari sisi (4,5), (1,2), (1,4), dan (3,4).
6. Karena jumlah sisi pada pohon merentang T telah mencapai 4 buah (jumlah simpul, n=5), maka pohon merentang T yang terdiri dari sisi (4,5), (1,2), (1,4), dan (3,4) merupakan pohon merentang minimum graf G2.

4. HASIL DAN ANALISIS

- Algoritma Prim dan algoritma Kruskal memberikan keluaran pohon merentang minimum yang sama.
- Algoritma Prim memiliki kondisi berhenti yang sama untuk setiap graf, yaitu setelah pembentukan sisi ke-(n-

1) pada pohon merentang T, atau setelah pengulangan ke-(n-1). Dapat kita lihat pada uji kasus I bahwa kondisi berhenti algoritma Prim terjadi setelah pembentukan sisi ke-6 pada pohon merentang T (artinya memang tidak ada lagi sisi yang bisa ditambahkan ke dalam pohon merentang T ini). Selain itu, pada uji kasus II juga terlihat bahwa kondisi berhenti algoritma Prim terjadi saat pembentukan sisi ke-4 pada pohon merentang T. Hal ini menyatakan bahwa algoritma Prim ditentukan oleh banyaknya simpul pada graf, bukan dipengaruhi oleh banyaknya sisi.

- Algoritma Kruskal memiliki kondisi berhenti yang belum tentu sama antara graf yang satu dengan graf yang lain. Dapat kita lihat pada uji kasus I, kondisi berhenti algoritma Kruskal terjadi setelah sisi pohon merentang T mencapai 6 buah, namun algoritma ini sendiri telah mengoperasikan 9 sisi dari 11 sisi yang terdapat pada graf G1. Sedangkan dalam uji kasus II, algoritma Kruskal berhenti setelah pembentukan sisi ke-4 pada pohon merentang T, namun algoritma ini sebenarnya telah mengoperasikan 4 sisi dari 10 sisi yang ada pada graf G2. Hal ini menyatakan bahwa algoritma Kruskal bekerja dengan dipengaruhi oleh jumlah sisi yang terbentuk pada pohon merentang T, dan tidak dipengaruhi oleh banyaknya simpul maupun sisi yang ada pada graf G.
- Dalam algoritma Prim, setiap langkah yang dilakukan selalu menghasilkan sisi bagi pohon merentang T. Hal ini terjadi karena keterhubungan setiap simpul selalu terjaga, sehingga pasti ada sisi dengan bobot minimum yang menghubungkan antar simpul, yang merupakan anggota dari pohon merentang minimum graf tersebut. Hal ini menandakan bahwa tidak ada langkah yang sia-sia (*useless*) dalam algoritma Prim.
- Dalam algoritma Kruskal, mungkin saja ada banyak langkah sia-sia yang dilakukan. Kita lihat dalam uji kasus I, ketika pohon merentang T yang terbentuk sudah memiliki sisi (A,D), (C,E), (D,F), (A,B), dan (B,E), sebenarnya hanya perlu menambahkan satu sisi lagi, yaitu sisi (E,G), namun sebelum sampai ke sisi (E,G), kita masih harus mengoperasikan sisi (B,C), (F,E), (D,B), yang ternyata justru membentuk sirkuit di T, sehingga langkah ini menjadi sia-sia. Kasus terburuk adalah bila ada graf dengan n buah simpul dengan cukup banyak sisi, dan sisi yang merupakan anggota pohon merentang minimumnya terdapat di awal dan di akhhir pengurutan, maka algoritma Kruskal akan tetap melakukan pengoperasian terhadap sisi-sisi yang berada di antara sisi awal dan sisi akhir, walaupun sebenarnya sisi-sisi tersebut bukan merupakan anggota pohon merentang minimum graf tersebut.
- Sekilas memang terlihat algoritma Prim melakukan langkah yang sia-sia dengan jumlah lebih sedikit daripada algoritma Kruskal. Namun demikian, perlu kita tinjau lebih lanjut bahwa untuk setiap langkah yang diambil, algoritma Prim juga melakukan beberapa langkah sia-sia. Misalnya jika dilihat pada uji kasus I, pada langkah 2, sisi yang bersisian dengan pohon merentang T ada 4 buah, yaitu (A,B), (D,B), (D,E), dan (D,F). Untuk menentukan sisi yang merupakan anggota pohon merentang T, kita perlu membandingkan bobotnya, sehingga didapatkan bobot minimum adalah sisi (D,F). Setelahnya, kita masih harus memeriksa apakah sisi (D,F) membentuk sirkuit di T. Jika membentuk sirkuit, maka kita perlu membandingkan kembali bobot sisi mana yang minimum, kemudian memeriksa adanya sirkuit di T, begitu seterusnya hingga didapatkan sisi yang sesuai. Proses perbandingan ini berlanjut terus pada setiap langkah. Pada langkah 3 uji kasus I, sisi yang didapat adalah (A,B), (D,B), (D,E), (F,E), dan (F,G). Beberapa sisi pada langkah 2 masih ditemukan di langkah 3 ini, sehingga ada proses perbandingan ulang yang terjadi. Bahwa ternyata sisi-sisi yang diperiksa berulang-ulang tersebut bukan anggota pohon merentang minimum, maka langkah perbandingan ini menjadi langkah sia-sia yang berulang-ulang. Secara teknis, ini bisa melahirkan langkah sia-sia (*useless*) yang lebih banyak daripada langkah sia-sia pada algoritma Kruskal.

5. KESIMPULAN

Graf merupakan suatu kumpulan simpul yang mungkin dihubungkan dengan sisi (antar satu simpul dengan simpul lain). Pohon adalah graf tak-berarah terhubung yang tidak memiliki sirkuit. Pohon merentang adalah upagraf (subgraf) yang tidak memiliki sirkuit. Pohon merentang minimum adalah pohon merentang yang memiliki bobot minimum dari suatu graf berbobot. Pembahasan mengenai pohon merentang minimum ini sangat penting, disebabkan banyaknya kegunaan yang dapat diperoleh dari pohon merentang minimum ini.

Ada dua cara yang umum digunakan dalam memecahkan masalah pohon merentang minimum ini. Cara pertama adalah dengan menggunakan pendekatan algoritma Prim, sedangkan cara kedua adalah melalui pendekatan algoritma Kruskal.

Konsep dasar yang dipakai dalam algoritma Prim adalah dalam dalam setiap langkah, pilih sisi dari graf G yang berbobot minimum, yang terhubung dengan pohon merentang T yang telah terbentuk, dan tidak membentuk sirkuit. Sedangkan konsep dasar yang digunakan dalam algoritma Kruskal adalah pada setiap langkah, pilih sisi dari graf G yang berbobot minimum, tetapi sisi tersebut tidak membentuk sirkuit di T.

Algoritma Prim menitikberatkan pada proses pencarian simpul. Artinya, algoritma ini tidak dipengaruhi oleh banyaknya sisi dalam graf, melainkan hanya dipengaruhi oleh banyaknya simpul. Kelebihan algoritma Prim adalah bahwa dalam setiap langkah yang diambil, algoritma Prim selalu menghasilkan sisi yang merupakan anggota pohon merentang minimum. Hal ini membuat setiap langkah menjadi efektif dan tidak ada langkah yang sia-sia. Namun

demikian, kelemahan algoritma ini terletak pada sublangkah yang dilakukannya, yaitu proses pencarian sisi berbobot minimum dan pemeriksaan keberadaan sirkuit terhadap pohon merentang yang telah terbentuk. Untuk graf dengan banyak cabang di tiap simpulnya, misalnya graf lengkap, sublangkah seperti ini justru menjadi penghambat dalam masalah kemangkusan. Algoritma ini dirasa cukup mangkus untuk diterapkan pada graf yang memiliki sedikit cabang pada tiap simpulnya serta memiliki sedikit simpul.

Di sisi lain, algoritma Kruskal bekerja dengan menitikberatkan pada proses pencarian sisi. Algoritma ini mengurutkan terlebih dahulu semua sisi pada graf, untuk kemudian mengoperasikannya satu per satu hingga tercapai sisi pohon merentang berjumlah n-1 buah (dengan n adalah jumlah simpul pada graf). Kelebihan agoritma ini adalah bahwa proses yang dilakukan merupakan proses random dalam pencarian sisi, artinya sisi yang diambil adalah sisi minimum, tidak peduli bahwa sisi tersebut belum terhubung dengan pohon merentang yang telah terbentuk, sejauh tidak adanya sirkuit yang terbentuk terhadap pohon merentang. Sayangnya, kelemahan algoritma ini dapat ditemukan pada pengoperasian terhadap sisi cabang-cabang pada tiap simpul. Seringkali ditemukan bahwa pengoperasian tersebut hanya tidak menghasilkan apa-apa, atau dengan kata lain menjadi sia-sia. Algoritma Kruskal ini masih lebih mangkus jika diterapkan pada graf dengan sedikit cabang pada tiap simpul serta memiliki agak banyak simpul dibandingkan dengan algoritma Prim.

Pada pengujian kali ini masih belum bisa terlihat mana algoritma yang lebih baik, karena jumlah sisi dan simpul pada graf yang diuji masih relatif sedikit. Kemangkusan algoritma akan lebih jelas terlihat untuk penanganan kasus dengan jumlah simpul dan sisi graf yang jauh lebih banyak.

REFERENSI

[1] Munir, Rinaldi. 2008. Diktat Kuliah IF2091 Struktur Diskrit. Program Studi Teknik Informatika, Sekolah Tinggi Teknik Elektro dan Informatika, Institut Teknologi Bandung.