Studi Minimum Spanning Tree dengan Algoritma Prim dan Kruskal

Hadyan Ghaziani Fadli – NIM : 13505005

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15005@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang pengaplikasian Algoritma Prim dan Algoritma Kruskal dalam dunia nyata. Dalam hal ini saya menampilkannya dalam pembuatan *Minimum Spanning Tree.* Algoritma pertama yang dipakai untuk menemukan *Minimum Spanning Tree* ditemukan oleh Otakar Boruvka tahun 1926, namun kini disempurnakan oleh 2 algoritma lainnya yang dikenal dengan Algoritma Prim dan Algoritma Kruskal. Algoritma Prim dan Algoritma Kruskal dalam dalam Matematika Diskrit adalah suatu algoritma yang berada didalam teori grafik untuk mencari sebuah *Minimum Spaning Tree* untuk menghubungkan graf berbobot.

Minimum Spanning Tree ini sangat berguna di dunia, karena dengan mengaplikasikannya kita dapat menentukan jarak terpendek suatu titik dengan titik lainnya dengan melewati titik- titik yang ada diantaranya.

Realisasi dari algoritma pembentuk *Minimum Spanning Tree* ini bisa bermacam-macam. Yang paling populer adalah pencarian jarak terpendek suatu kota dengan kota lainnya dengan melewati kota kota lain yang terbentang di antara 2 kota yang kita cari jarak minimumnya tersebut.

***Kata kunci**: Minimum Spanning Tree, Algoritma Prim, Algoritma Kruskal*

1. Pendahuluan

Ketika kita sedang berjalan dengan jalan darat, dengan kereta api atau dengan kendaraan bermotor dari suatu kota ke kota lainnya, pernahkah terpikir bagaimana arah jalan yang kita lewati?

Untuk kendaraan motor seperti mobil, menghubungkan beberapa kota besar mungkin akan dihubungkan secara langsung dengan jalan tol,namun pada umumnya, seperti pada jalan kereta api, untuk penghematan pembuatan jalan, rel kereta api dibuat tidak langsung dibuat seperti jalan tol yang menghubungkan 2 kota yang cukup jauh secara langsung, namun melalui kota kota yang berada di antaranya, jadi jalanan ini sekaligus menghubungkan kota kota yang ada diantara kota asal dan kota tujuan kita.

Untuk menentukan jalur mana yang akan dipakai agar efisien maka diperlukan cara untuk mendapatkan jalur terpendek. Karenanya pencarian sebuah jalan dengan jarak minimal sangatlah dibutuhkan khususnya dalam dunia transportasi di dunia.

Namun bagaimanakah cara atau metode yang digunakan untuk mendapatkan jawaban atas kasus diatas?

Pada Tahun 1926, Seorang Czech scientist, Otakar Boruvka, menemukan sebuah algoritma yang dikenal dengan Boruvka's Algorithm.
Seiring berjalannya waktu, ada 2 algoritma lain yang lebih umum untuk dipakai yaitu Algoritma Prim dan Algoritma Kruskal.

Walau ada lebih dari 1 algoritma yang berbeda, namun jalan yang didapat akan sama panjangnya. Metode yang dipakai oleh ketiga Algoritma ini adalah metode Minimum Spanning Tree.

2. Algoritma Prim

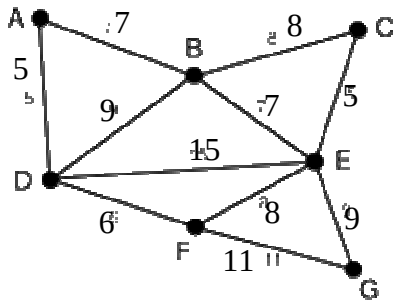
Algoritma Prim adalah suatu algoritma di dalam teori graf yang bertujuan menemukan *Minimum Spanning Tree* untuk menghubungkan graf berbobot.
Ini berarti algoritma ini menemukan subset dari sebuah tepi yang membentuk sebuah *Tree* yang meliputi setiap titik nya. Dimana total beban dari setiap tepi di *Tree* diminimalkan.
Jika Graf tidak terhubung, maka ini hanya akan menemukan sebuah *Minimum Spanning Tree* menjadi satu jalur untuk komponen yang terhubung.
Algoritma ini ditemukan pada tahun 1930 oleh seorang ahli matematika Vojtech Jarnik, dan kemudian dipublikasikan oleh seorang *computer scientist* Robert C. Prim pada tahun 1957 dan ditemukan kembali oleh Dijkstra pada tahun 1959. Oleh karena itu terkadang Algoritma ini juga disebut DJP algorithm atau algoritma Jarnik.

2.1 Cara Kerja Algoritma Prim

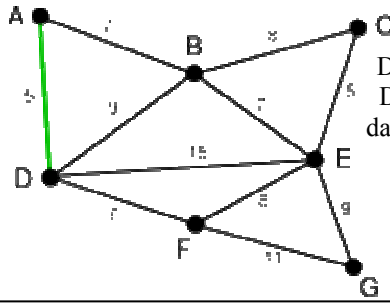
- Buat sebuah *Tree* yang mengandung vertex tunggal, pilih yang berbobot minimum
- Buat sebuah set (yang belum tercakup) yang mengandung semua vertices yang lain di dalam graf
- Buat sebuah set (fringe vertices) yang di inisialiasai kosong
- Loop(jumlah vertices - 1) :
    - Pindahkan tiap vertives yang belum tercakup dan secara langsung terhubung kepada *node* terakhir, tambahkan ke fringe set
    - Untuk tiap titik di set sisi, tentukan, jika sebuah sisi menghubungkan vertices dengan node terakhir yang ditambahkan,jika iya, maka jika sisi tersebut memiliki bobot lebih kecil dari sisi sebelumnya yang menghubungkan vertex ke Tree yang telah terbuat, masukkan sisi baru ini melalui node terakhir yang ditambahkan sebagai rute terbaik di Tree yang telah terbuat.
    - Pilih sisi dengan bobot minimum yang menghubungkan vertex dalam fringe set ke vertex pada Tree yang sudah terbuat
    - Tambahkan sisi tersebut ke Tre dan pindahkan fringe vertex dalam fringe set ke sebuah vertex dalam Tree yang sudah terbuat
    - Update node terakhir yang ditambahkan untuk menjadi fringe vertex yang baru ditambahkan

Hanya | V | -1, dimana | V | adalah jumlah dari vertices dalam graf, pengulangan diperlukan. Sebuah Tree menghubungkan | V | vertices hanya membutuhkan | V |-1 sisi dan tiap pengulangan dari algoritma yang dideskripsikan diatas tertarik dalam tepat 1 sisi.
Implementasi yang sippel menggunakan representasi *adjacency matrix graph* dan mencari sebuah barisan bobot untuk mencari sisi dengan bobot minimum.

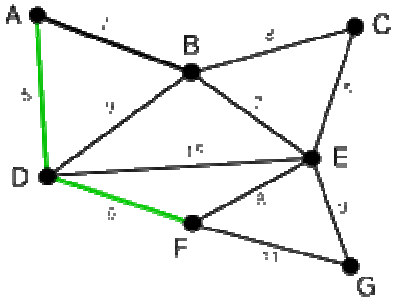
2.1 Ilustrasi Cara Kerja Algoritma Prim



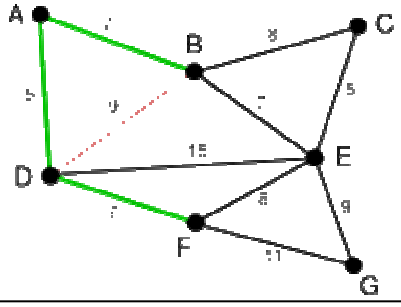
Ini adalah sebuah graf berbobot yang asli, ini bukan sebuah *Tree* karena definisi sebuah *Tree* tidak memiliki sirkuit, dan diagram ini memiliki sirkuit. Nama yang lebih benar untuk diagram ini adalah sebuah graf atau network. Nomer didekat busur adalah bobotnya. Tidak satupun dari titik sudut yang diberi highlight dan vertex D dipilih sebagai starting point.



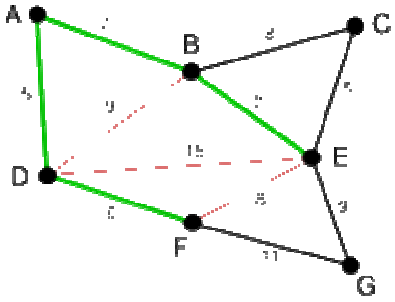
Kedua, pilih vertex terdekat dengan D : A memiliki jarak 5 dengan D, B memiliki jarak 9, E memiliki jarak 15, dan F memiliki jarak 6. Dari semuanya 5 adalah jarak terpendek, jadi kita highlight vertex A dan busur DA.



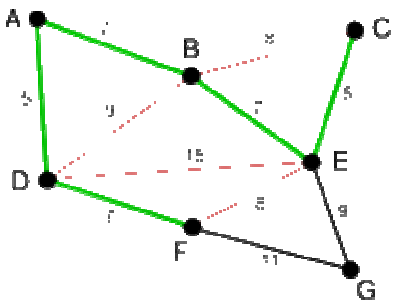
Vertex berikutnya adalah vertex yang terdekat ke D atau A. B memiliki jarak 9 dari D dan 7 dari A, E memiliki jarak 15 dan F memiliki jarak 6. Jadi F yang memiliki jarak terpendek. Sehingga kita highlight vertex F dan sisi DF.



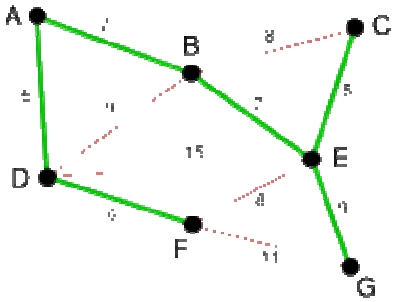
Algoritma yang disebutkan diatas, vertex B, yang memiliki jarak 7 dari A, di highlight. Disini , sisi DB di highlight warna merah, karena kedua vertex yaitu B dan D telah di hilghlight, jadi ini tidak bisa digunakan



Dalam kasus ini, kita dapat memilih antara C, E dan G. C memiliki jarak 8 dai B, E memiliki jarak v dari B dan G memiliki jarak 11 dari F. E yang terdekat.jadi kita highlight E dan sisi EB. Dia sisi lainnya telah di highlight merah, artinya kedua vertices gabungan itu telah dipakai



Disin, vertices yang masih tersedia adalah C dan G. C memiliki jarak5 dari E dan G memiliki jarak 9 dari E. C yang dipilih. Jadi kita highlight sisi EC, dan BC juga kita highlight merah



Vertex G adalah vertex terakhir. Ini berjarak 11 dari F, dan 9 dari E. E yang terdekat , jadi kita highlight sisi EG. Sekarang semua vertices telah di highlight, sebuah Minimum Spanning Tree di tamppilkan dengan warna hijau, dalam kasus ini bobotnya adalah 39.

Pseudocode

```
 Minimum-Spanning-Tree-by-Prim(G, weight-function, source vertex)
 1  for each vertex u in graph G
 2    set key of u to ∞
 3    set parent of u to nil
 4  set key of source vertex to zero
 5  enqueue to minimum-heap Q all vertices in graph G.
 6  while Q is not empty
 7    extract vertex u from Q  // u is the vertex with the lowest key
that is in Q
 8    for each adjacent vertex v of u do
 9      if (v is in Q) and (weight-function(u, v) < key of v) then
10        set u to be parent of v  // in minimum-spanning-tree
11        update in Q the key of v to equal weight-function(u, v)
```

Pseudocode ini intinya sama dengan algoritma diatas. Kecuali penentuan pilihan pertama didalam loop dan untuk melakuka ini. Tukar pilihan lainnya dan update yang ada didalam loop. Kunci dari sebuah vertex adalah bobot minimum yang menghubungkan sisi vertex ini ke Tree yang sudah dibangun. Jadi ini akan mengeluarkan sebuah Tree per komponen yang dihubungkan.

Line 1 -3 mendeskripsikan penempatan awal dai semua vertices yang belum tercakup.

Line 4 memeastikan bahwa source ini, sejak ini akan memiliki kunci dari 0, selama setiap yang

lainnya memiliki kunci infinity, adalah vertex pertama yang ditambahkan kedalam *Tree*

Line 5 menaruh semua kunci vertex dalam *min-heap* untuk mempersilahkan untuk pemindahan cepat dari kunci minimum.

Line 7 mengeluarkan vertex dengan bobot minimum yang menghubungkannya dengan *Tree* yang sudah terbuat.

Line 8 – 11 pindahkan semua item yang berada di set yang tak terlihat dan terhubung dengan Node terkhir yang ditambahkan ke Fringe set dan update bobot dari fringe vertices yang memiliki

kunci dengan nilai yang aktual.
Tes kedua di Line 9 memastikan rute terbaik yang digunakan dalam Tree.

Line 10 menyajikan informasi sisi yang pernah terhubung kepada fringe vertex yang telah terpilih ke Tree yang telah terbuat.

2.1 Bukti kebenaran Algoritma Prim

Biarkan P terhubung sebuah graf berbobot. Dalam setiap iterasidai Algoritma Prim , sebuah sisi harus ditemukan menghubungkan sebuah vertex dalam sebuah subgraf dengan sebuah vertex diluar subgraf. Sejak P terhubung, ini akan selalu menjadi jalan ke setiap vertex. Y, keluaran dari sebuah Algoritma Prim adalah sebuah *Tree* .Karena sisi dan vertex ditambahkan ke Y yang terhubung. Jadikan Y_1 sebagai Minimum Spanning Tree dai P. Jika $Y_1 = Y$, kemudian Y adalah sebuah Minimum Spanning Tree. Dilain pihak, biarkan e menjadi sisi petama yang ditambahkanselama pembangunan dai Y yang bukan di Y_1 , dan V menjadi set dati vertices yang dihubungkan oleh sisi yang ditambahkan sebelum e . Kemudian sebuah endpoint dari e adalah didalam V dan lainnya tidak.
Sejak Y_1 adalah Spanning Treedari P , disana ada sebuah jalan dalam Y_1 yang bergabung dalam 2 endpoint.
Selama sesuatu berjalan di dalam jalan tersebut, ia harus mempertemukan sebuah sisi f dan menggabungkannya dengan sebuah vertex didalam V ke sesuatu yang bukan V. Dalam iterasi ketika e ditambahkan ke Y , f bisa juga telah ditambahkan dan ini akan ditambahkan sebagai ganti dari e jika bobotnya kurang dari e. Maka dapat disimpulkan

$$W(f) >= w(e)$$

3. Algoritma Kruskal

Algoritma Kruskal pertama kali dipopulaerkan oleh Joseph Kruskal pada tahun 1956. Algoritma Kruskal adalah sebuah algoritma dalam teori graf yang mencari sebuah Minimum Sanning Tree untuk sebuah graf berbobot yang terhubung. Ini berarti mencari subset dari sisi yang membentuk sebuah Tree yang menampung setiap vertex, dimana total bobot dari semua sisidalam Tree adalah minimum. Jika graf tidak terhubung, kemudian ini mencari sebuah Minimum Spanning Forest (sebuah Minimum Spanning Tree untuk tiap komponen yang terhubung). Algoritma Kruskal adalah suatu contoh dari Algoritma Greedy.

3.1 Cara Kerja Algoritma Kruskal

- Buat sebuah *Forest F (*set dari Tree*)*, yang tiap vertex dalam graf adalah Tree pemisah
- Buat sebuah set S yang mengandung semua sisi didalam graf
- While S tidak kosong :
    - Buang sebuah sisi dengan bobot minimum dari S
    - Jika sebuah sisi menghubungkan dua pohon yang berbeda, kemudian tambahkan ini kedalam Forest, kombinasikan 2 Tree kedalam 1 Tree.
    - Dilain pihak, buang sisi tersebut

Dalam terminasi dari algoritma ini, Forest hanya memiliki satu komponen dan membentuk sebuah Minimum Spanning Tree dari graf

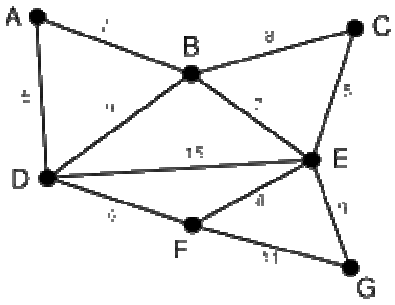
3.1 Performa Algoritma Kruskal

Di mana E adalah banyaknya tepi di sisi dan V adalah banyaknya vertices, algoritma Kruskal dapat ditunjukkan untuk O(E log E) waktu, atau dengan setara, O(E log V) waktu, adalah semua dengan struktur data yang sederhana.waktu yang dipakaiadaah equivalen sebab :

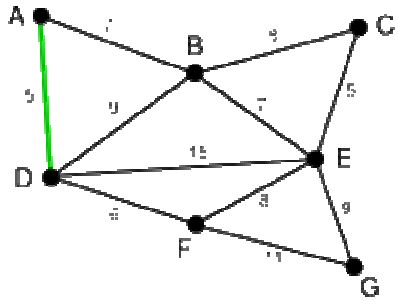
- E adalah maksimal V^2 dan log $V^2 = 2 \times \log V$ adalah $O(\log V)$.
- Jika kita mengabaikan vertices yang terisolasi, yang masing masing akan menjadi komponen mereka sendiri dari Minimum Spanning Tree, bagaimanapun, V = 2E, maka log V adalah O(Log E).

Kita dapat mencapai ikatan ini sebagai berikut:pertama sort sisi dengan beban dengan menggunakan suatu comparison sort dalam O(E log E) waktu. ini mengijinkan langkah tersebut " pindahkan suatu tepi dengan bobot yang minimum dari S" untuk beroperasi dalam waktu yang konstan. Berikutnya, kita menggunakan suatu disjoint-set struktur data untuk menjaga track dimamna vertices ada didalam komponen tersebut.
Kita harus melakukan operasi O(E) , dua ditemukan operasi dan mungkin satu gabungan sambung untuk masing-masing sisi. Bahkan suatu simple disjoint-set dari struktur data yang sederhana seperti disjoint-set Forest dengan gabungan oleh ranking dapat dilakukan operasi O(E) di dalam O(E log V) waktu. Dengan begitu total waktu adalah O(E log E)= O(E log V).
Dengan ketentuan bahwa sisi adalah apapun telah mensortir atau dapat disortir di dalam waktu yang linier (sebagai contoh dengan counting sort), algoritma dapat menggunakan disjoint-set struktur data yang lebih canggih untuk menjalankan O(E a(V)) waktu, di mana suatu adalah sangat invers yang tumbuh lambat dari itu fungsi Ackermann yang bernilai tunggal.

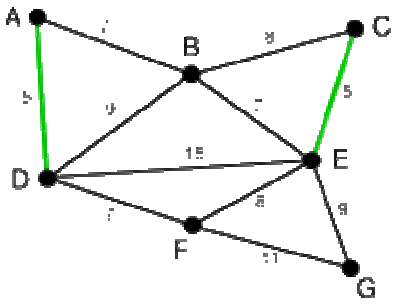
3.1 Ilustrasi Cara Kerja Algoritma Kruskal



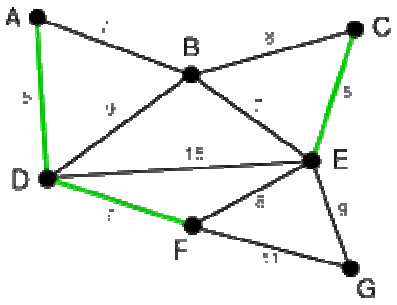
Ini adalah graf asli. Angka-Angka dekat sisi menandai bobot mereka. Tidak satupun dari sisi di Highlight



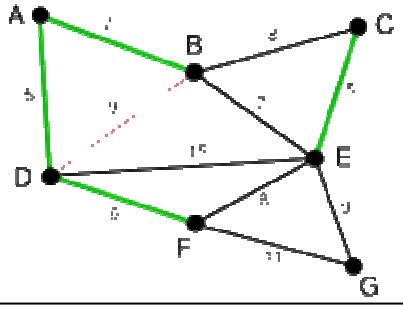
AD dan CE adalah sisi yang paling pendek, dengan panjangnya 5, dan AD ditetapkan sebagai awal, jadi kita beri highlight



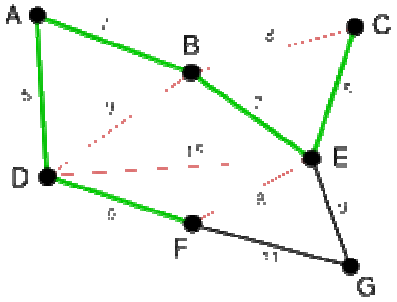
CE sekarang adalah sisi terpendek yang tidak membentuk loop, dengan panjang 5, maka kita highlight.



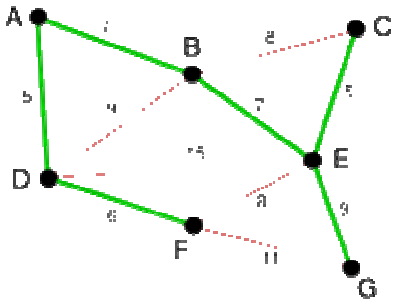
Sisi selanjutnya , DF dengan panjang 6, di highlight menggunakan metode yang sama



Sisi terpendek berikutnya adalah AB dan BE, keduanya memiliki panjang 7, AB dipilih, dai di highlight, Sisi BD di Highlight merah, karena ini akan membentuk loop ABD



Proses berlanjut dengan meng- highlight ke sisi terpendek berikutnya, BE dengan panjang 7. akan lebih banyak sisi yang di highlight dengan merah dalam stage ini : BC, DE dan FE



Terakhir, proses berakhir dengan sisi EG denganpanjang 9, dan Minimum Spanning Tree telah ditemukan

Pseudocode

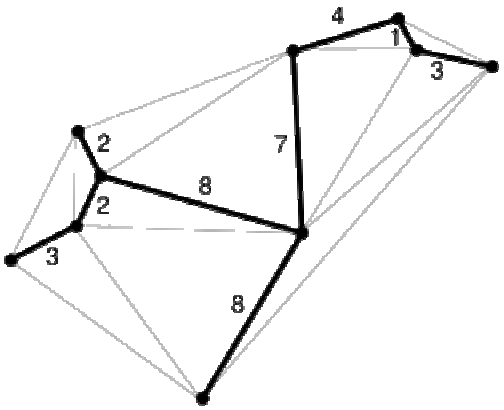
```
 1  function Kruskal(G)
 2    for each vertex v in G do
 3      Define an elementary cluster C(v) ← {v}.
 4    Initialize a priority queue Q to contain all edges in G, using
the weights as keys.
 5    Define a tree T ← Ø                                  //T will
ultimately contain the edges of the MST
 6    while T has fewer than n-1 edges do
 7      (u,v) ← Q.removeMin()
 8      Let C(v) be the cluster containing v, and let C(u) be the
cluster containing u.
 9      if C(v) ≠ C(u) then
10        Add edge (v,u) to T.
11        Merge C(v) and C(u) into one cluster, that is, union C(v) and
C(u).
12    return tree T
```

3.1 Bukti kebenaran Algoritma Kruskal

Biarkan P terhubung,jadilah suatu graf berbobot dan biarkan Y jadilah subgraph dari P yang diproduksi oleh algoritma itu. Y tidak bisa mempunyai suatu siklus, sejak sisi yang ditambahkan untuk siklus itu akan telah di dalam satu subtree dan bukan antara dua Tree yang berbeda. Y tidak bisa diputus, karena lebih dulu ditemui sisi itu menggagungkan dua komponen dari Y akan telah ditambahkan oleh algoritma itu. Seperti itu, Y adalah suatu Spanning Tree dari P.

Untuk menyederhanakan, berasumsi bahwa semua sisi mempunyai bobot yang berbeda. Biarkan Y1 jadilah suatu Minimum Spanning Tree. Jika Y1=Y kemudian Y adalah suatu Minimum Spanning Tree. Cara lainnya, biarkan e jadilah tepi yang pertama yang dipertimbangkan oleh algoritma yang di dalam Y tetapi bukan di dalam Y1. Y1+E mempunyai suatu siklus, sebab kamu tidak bisa menambahkan suatu sisi equivelent dengan spanning tree dan masih mempunyai suatu Tree. Siklus ini berisi tepi lain f yang (mana) di langkah algoritma di mana e ditambahkan untuk Y, belum dipertimbangkan. Ini adalah sebab jika tidak e tidak akan menghubungkan Tree yang berbeda. hanyalah dua cabang dari Tree yang sma. Kemudian Y2=Y1+E-F adalah juga suatu spanning tree. Bobot total nya adalah kurang dari total berat/beban dari Y1. Ini Adalah sebab algoritma mengunjungi e [sebelum/di depan] f. Kesimpulan ialah .. Y1 adalah tidak ada Minimum Spanning Tree, dan asumsi yang di sana ada suatu sisi di dalam Y, tetapi bukan di dalam Y1, adalah salah. Membuktikan ini yang Y=Y1, yaitu., Y adalah suatu Minimum Spanning Tree.

4. Minimum Spanning Tree



hubungkanlah, graf yang tidak berarah, sebuah Spanning Tree dari graf tersebut adalah subgraf dimana sebuah Tree dan sisinya terhubung bersama-sama. graf tunggal dapat mempunyai banyak Spanning Tree yang berbeda. Kita dapat juga memberikan suatu bobot kepada masing-masing sisi, yang mana adalah berupa nomor yang mewakili bobot tersebut. dan menggunakan ini untuk menentukan suatu bobot equivalent dengan Spanning Tree dengan komputasi penjumlahan dari bobottersebut dari sisi di Spanning Tree. Suatu Minimum Spanning Tree atau bobot Minimum Spanning Tree kemudian adalah suatu Spanning Tree dengan bobot kurang dari atau sepadan otdengan bobt dari tiap Spanning Tree lain. Lebih umum lagi, manapun graf yang tidak berarah mempunyai suatu Minimum Spanning Forest.

Satu contoh, suatu perusahaan cable TV yang akan memasang sambungan baru ke tetangga baru. Jika itu dibatasi untuk mengubur kabel hanya sepanjang alur,tertentu, kemudian akan ada suatu graf yang mewakili poin-poin yang dihubungkan oleh alur itu. Sebagian dari alur itu boleh jadi lebih mahal, sebab mereka lebih panjang, atau perlu kabel untuk dikuburkan lebih dalam, alur ini akan diwakili oleh membingkai dengan memberi bobot yang lebih besar. Suatu Sanning Tree untuk graf itu akan menjadi suatu subset alur itu semua yang tidak punya siklus tetapi masih menghubungkan tiap-tiap rumah. Disana boleh jadi beberapa Spanning Tree mungkin. Suatu Minimum Spanning Tree akan satu pilihan dengan total biaya yang paling rendah.

4. 1 Algoritma Minimum Spanning Tree

Algoritma yang pertama untuk menemukan suatu Minimum Spanning Tree telah dikembangkan oleh seorang ilmuwan Cekoslovakia yang bernama Otakar Boruvka di tahun 1926 (lihat algoritma Boruvka). Tujuan nya adalah suatu pemenuhan elektrik efisien tentang Bohemia. Sekarang Ada dua algoritma biasanya digunakan, Algoritma aprim dan algoritma Kruskal. Ketiganya adalah dikenal dengan algoritma Greedy.

algoritma Minimum Spanning Tree yang tercepat sampai saat ini telah dikembangkan oleh Bernard Chazelle,yang didasarkan pada algoritma Boruvka. waktu untuk menjalankan adalah O(E a(e,v)), di mana e adalah banyaknya tepi, v mengacu pada banyaknya vertice dan a adalah invers fungsional klasik dari fungsi Ackermann. Fungsi ini tumbuh sangat pelan-pelan, sedemikian sehingga semua practical purposes disadari sebagai konstanta yang tidak lebih besar dari 4, dengan begitu algoritma Chazelle mengambil sangat dekat dengan O(e) waktu.

Apa yang merupakan algoritma mungkin paling cepat untuk masalah ini? Itu adalah salah satu dari pertanyaan terbuka yang paling tua di dalam Computer Science. terlihat dengan jelas suatu ikatan linear yang lebih rendah, sejak kita harus sedikitnya menguji semua bobot itu. Jika bobot sisinya adalah bilangan integer dengan suatu yang dibatasi panjangnya bit, kemudian algoritma yang deterministic dikenal dengan waktu untuk menjalankan yang linier, O(e). untuk bobot biasa, algoritma acak diketahui dalam waktu harapan linear.

Apakah di sana ada suatu algoritma yang deterministic dengan waktu menjalankan yang linier untuk bobot umum masih suatu pertanyaan terbuka. Bagaimanapun, Seth Pettie dan Vijaya Ramachandran sudah menemukan suatu deterministic algoritma Minimum Spanning Tree, kompleksitas komputasional yang tak diketahui.

baru-baru ini, riset telah terpusat untuk memecahkan masalah Minimum Spanning Tree dengan metode highly parelellized. Sebagai contoh, paper pragmatis 2003 " fast Shared-Memory Algorithm untuk Menghitung Yang Minimum Spanning Forest dari sparse graphs" dengan David A. Bader dan Guojing Cong mempertunjukkan suatu algoritma yang dapat menghitung MSTS 5 kali lebih cepat pada 8 processor dibanding suatu sequensial algorithm yang di optimalkan. Secara khas, algoritma yang paralel didasarkan pada algoritma Boruvka—Prim Dan terutama algoritma Kruskal tidak mengelupas juga kepada pengolah tambahan.

5. Kesimpulan

- Algoritma Prim dan Algoritma Kruskal adalah 2 algoritma yang banyak dipakai sekarang untuk menyelesaikan berbagai macam kasus pencarian Minimum Spanning Tree, walaupun jalannya berbeda , 2 algoritma ini berasal dari 1 induk yang sama yaitu algoritma Boruvka yang dirilis tahun 1926.

- Pada zaman sekarang yang cenderung dipakai adalah 2 algoritma diatas yaitu algoritma prim dan algoritma kruskal karena memiliki runtime yang paling kecil diantara algoritma lainnya untuk kasus minimum spanning tree.

- 3 algoritma yaitu Prim, Kruskal dan Boruvka biasa disebut algoritma breedy

- Algoritma Prim dilakukan dengan cara mencari sisi dengan bobot minimum sebagai langkah awal, kemudian dilanjutkan dengan mencari sisi tetangganya yang bernilai munimum namun tidak membuat loop. Langkah kedua dilakukan terus hingga semua vertice telah terhubung sehingga ditemukan Minimum Spanning Tree

- Algoritma Kruskal dilakukan dengan cara yang sama seperti algoritma prim sebagai langkah awal, yaitu mencari sisi dengan bobot minimum, langkah berikutnya adalah mencari sisi lain yang memiliki bobot minimum, tidak perlu bersebelahan, dan harus tidak membuat loop, langkah ini dilakukan iterasi hingga seluruh vertice telah terhubung dan membentuk Minimum Spanning Tree.

DAFTAR PUSTAKA

[1] Wikipedia.org *http://en.wikipedia.org* Tanggal akses: 24 Desember 2006 pukul 10.00.

[2] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] R. C. Prim: *Shortest connection networks and some generalisations*. In: *Bell System Technical Journal*, 36 (1957), pp. 1389–1401

[4] D. Cherition and R. E. Tarjan: *Finding minimum spanning trees*. In: *SIAM Journal of Computing*, 5 (Dec. 1976), pp. 724–741

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 23.2: The algorithms of Kruskal and Prim, pp.567–574.

[6] J. B. Kruskal: *On the shortest spanning subtree and the traveling salesman problem*. In: Proceedings of the American Mathematical Society. *7 (1956), pp. 48–50*

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 23.2: The algorithms of Kruskal and Prim, pp.567–574.

[8] Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*, Fourth Edition. John Wiley & Sons, Inc., 2006. ISBN 0-471-73884-0. Section 13.7.1: Kruskal's Algorithm, pp.632.