Perbandingan Algoritma - algoritma Pencarian Minimum Pohon Merentang dari Suatu Graf

Rosalina Paramita N. – NIM : 13505125

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15125@students.if.itb.ac.id

Abstrak

Makalah ini berisi pembahasan mengenai algoritma - algoritma yang digunakan untuk mencari minimum pohon merentang dari suatu graf. Ada 3 algoritma umum untuk mencari minimum suatu pohon merentang, yaitu algoritma Borůvka, yang memiliki kompleksitas waktu O(E logV), algoritma prim, dengan kompleksitas waktu O(E + V logV), algoritma kruskal, dengan kompleksitas waktu O(E logV). Namun, ketiga algoritma tersebut termasuk algoritma greedy. Sekarang ini, sudah ada banyak algoritma untuk mencari minimum suatu pohon merentang, antara lain : algoritma yang dikembangkan oleh Bernard Chazell yang merupakan pengembangan dari algoritma Borůvka, yang memiliki kompleksitas waktu yang O(e α(e,*v*)).

Pada umumnya, algoritma - algoritma tersebut merupakan hasil pengembangan dari 3 algoritma dasar, yaitu Boruvka, Prim, dan Kruskal, atau kombinasi diantara ketiga algoritma tersebut. Selain itu, ada juga algoritma yang memparalelkan algoritma - algoritma tersebut dengan procosessor, sehingga memiliki waktu eksekusi yang cukup cepat, seperti yang dibuat oleh David A. Bader dan Guojing Cong dalam *paper* "Fast Shared - Memory Algorithms for Computing the Minimum Spanning Forest of Sparse Graphs". Mereka mendemonstrasikan algoritma yang dapat menghitum pohon merentang minimum (minimum spanning tree) atau MST, 5 kali lebih cepat dengan 8 prosesor dibandingkan sebuah algoritma optimasi sekuensial.

Algoritma - algoritma spesialisasi yang lain didesain untuk menghitung pohon merentang minimum dari sebuah graf yang sangat besar sehingga sebagian besar darinya disimpan dalam *disk* hampir setiap waktu. Algoritma ini dikenal dengan algoritma *external storage* (penyimpanan eksternal), contohnya "Engineering an External Memory Minimum Spanning Tree Algorithm " oleh Roman Dementiev, yang mampu menghitung MST dengan ukuran yang sangat besar, namun memiliki waktu eksekusi yang 2 sampai 5 kali lebih lambat daripada algoritma tradisional.

Pada makalah ini, saya hanya akan membahas 3 algoritma dasar, algoritma Bernard Chazell, dan algoritma waktu linear. Kemudian mencoba untuk membandingkan kelima algoritma tersebut dari sisi waktu eksekusi atau kompleksitas algoritma.

***Kata Kunci** : minimum spanning tree (MST), minimum spanning forrest, kompleksitas waktu*

1. Pendahuluan

Pohon merentang adalah subgraf dari suatu graf tidak berarah, yang tidak memiliki sirkuit dan menghubungkan semua simpul yang ada dalam graf tersebut. Dengan kata lain, pohon merentang adalah upagraf dari suatu graf yang berbentuk pohon. Jika G adalah graf berbobot, maka pohon merentang T dengan bobot minimum dinamakan pohon merentang minimum.

Mencari minimum suatu pohon merentang merupakan salah satu masalah yang sudah cukup dikenal dalam pokok bahasan graf dan mempunyai terapan yang luas dalam praktek, seperti komunikasi tanpa kabel, jaringan terdistribusi, masalah - masalah terbaru dalam bidang biologi dan pengobatan seperti pendeteksian kanker, dll.

Algoritma pertama untuk mencari pohon merentang minimum dari suatu graf ditemukan oleh Otakar Borůvka pada tahun 1926. Algoritma tersebut digunakan sebagai metode untuk membangun jaringan listrik yang efisien

di kota Bohemia. Kemudian muncul 2 algoritma lain, yaitu algoritma prim dan algoritma kruskal, yang kemudian lebih sering digunakan. Ketiga algoritma ini termasuk algoritma greedy yang bekerja dalam waktu polinomial.

Karena kompleksitas waktu dari algoritma-algorima tsb cukup besar, maka orang-orang berusaha untuk menemukan cara agar dapat mencari pohon merentang minimum dengan kompleksitas waktu lebih kecil / cepat.

Salah satu algoritma yang mempunyai kompleksitas waktu lebih kecil dikembangkan oleh Bernard Chazell. Algoritma ini merupakan hasil pengembangan dari algoritma Borůvka.

Algoritma ini memiliki kompleksitas waktu O(e α(e,*v*)), dimana e adalah jumlah sisi, v jumlah simpul dan α adalah fungsi invers dari fungsi Ackermann. Fungsi α tumbuh dengan sangat lambat, jadi untuk semua kondisi nilainya bisa dihampirkan dengan konstan yang lebih kecil dari 4; sehingga algoritma ini mempunyai kompleksitas waktu yang mendekati nilai O(e).

Apakah solusi tercepat untuk menyelesaikan masalah pohon merentang minimum? Merupakan suatu pertanyaan lama yang ada dalam bidang ilmu komputer.
Namun, pertanyaan tersebut belum bisa dijawab. Karena para peneliti, ilmuwan, atau para ahli di bidang ini masih mencari solusi tersebut.

2. Algoritma Borůvka

Algoritma ini dimulai dengan memeriksa setiap simpul dan menambahkan sisi dengan bobot terkecil pada pohon merentang, tanpa memperhatikan pada sisi yang telah ditambahkan, dan melanjutkan menggabungkan sisi tsb sampai terbentuk suatu pohon merentang.

Algoritma :

- Mulai dengan sebuah graph yang berbobot, dan sebuah himpunan kosong dari sisi T
- Sementara simpul graph G yang terhubung oleh T disjoin :
  - Mulai dengan himpunan kosong sisi E
  - Untuk setiap komponen :
    - Mulai dengan himpunan sisi kosong S
    - Untuk setiap simpul di G
      - ♣ Tambahkan sisi dengan bobot terkecil pada suatu simpul ke simpul lain dengan elemen terpisah pada S
    - Tambahkan sisi dengan bobot terkecil di S ke E
  - Tambahkan hasil dari E ke T
- T adalah pohon merentang minimum dari graph G

Kompleksitas waktu algoritma ini adalah O(E logV), dimana E menyatakan jumlah sisi dan V jumlah simpul dari graph G.

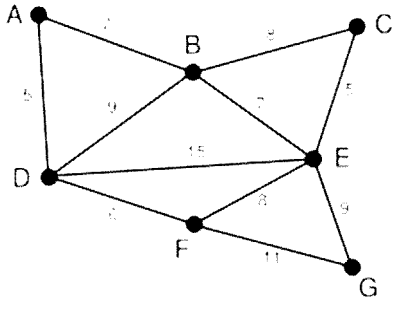
3. Algoritma Prim

Algoritma Prim membentuk pohon merentang minimum langkah per langkah. Pada setiap langkah diambil sisi dari graf G yang mempunyai bobot minimum namun terhubung dengan pohon merentang minimum T yang telah terbentuk.

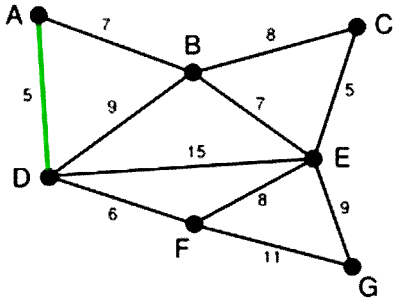
Algoritma :

- Ambil sisi dari graf G yang berbobot minimum, masukkan ke dalam T
- Pilih sisi (*u, v*) yang mempunyai bobot minimum dan bersisian dengan simpul di T, tetapi (*u,v*) tidak membentuk sirkuit di T. Tambahkan (*u, v*) ke dalam T.
- Ulangi sebanyak *n* - 2 kali

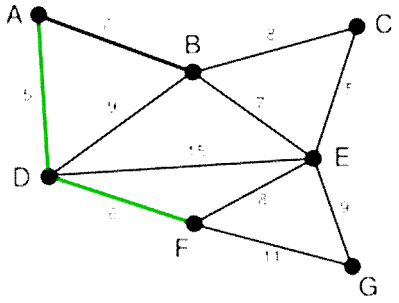
Contoh :
Berikut adalah graf berbobot, bobot sebesar 90, yang akan dicari pohon merentang minimum nya, pilih sembarang simpul, pada contoh ini, simpul D.



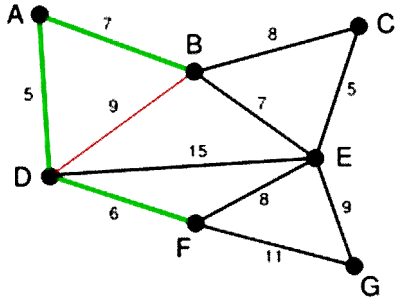
Simpul kedua yang dipilih adalah simpul yang terdekat ke D, yaitu simpul A.



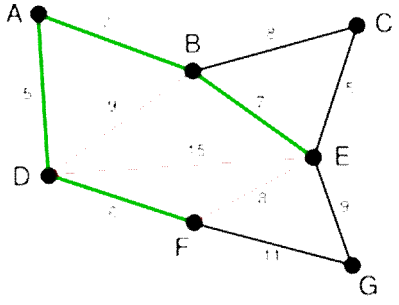
Simpul berikutnya adalah simpul yang terdekat ke simpul D atau A, yaitu simpul F, yang berbobot 6 dari D.



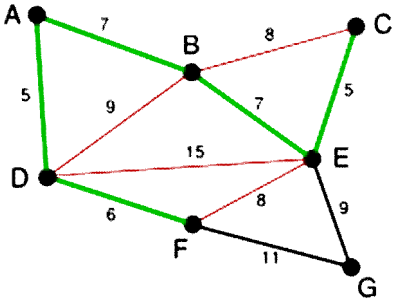
Seperti langkah sebelumnya, pilih simpul B yang memiliki bobot 7 dari A



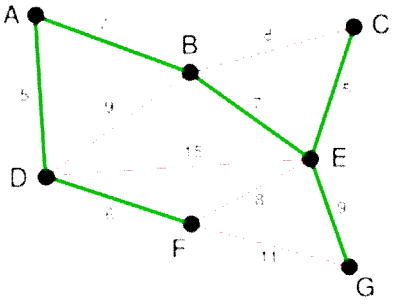
Selanjutnya, pilih simpul E yang terdekat dengan simpul B.



Selanjutnya, pilih simpul C yang terdekat dengan simpul E.



Karena simpul G adalah simpul terakhir yang tersisa, dan memiliki jarak/bobot terdekat dengan E, maka EG dipilih



Karena semua simpul telah dipilih, maka pohon merentang minimum ditunjukkan dengan warna hijau. Dengan bobot sebesar 39.

Sebuah implementasi sederhana menggunakan graf representasi matrix *adjacency* (berdekatan) dan mencari tabel bobot untuk menemukan sisi dengan bobot minimum untuk ditambahkan memerlukan kompleksitas waktu O(V^2).
Namun, jika menggunakan sebuah stuktur data sederhana *binary heap* dan sebuah represenatsi *adjacency list*, maka kompleksitas waktu O(E log*V)*.
Dan jika menggunakan sebuah *Fibonacci heap* yang rumit, maka kompleksitas waktu O(E + Vlog V).
Dimana E adalah jumlah sisi dan V jumlah simpul.

4. Algoritma Kruskal

Pada algoritma kruskal, sisi-sisi graf diurut terlebih dahulu berdasarkan bobotnya dari kecil ke besar. Sisi yang dimasukkan ke dalam himpunan T adalah sisi graf G sedemikian sehingga T adalah pohon. Pada keadaan awal, sisi-sisi sudah diurut berdasarkan bobot membentuk hutan, masing-masing pohon di hutan hanya berupa satu buah simpul, hutan

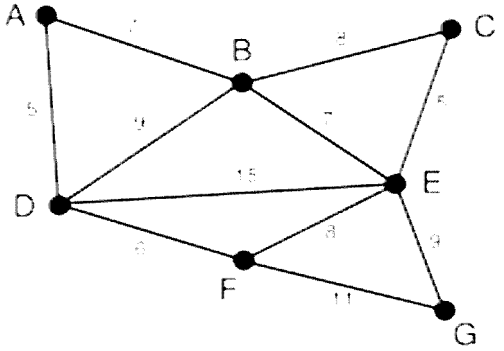
tersebu dinamakan hutan merentang (*spanning forest*). Sisi dari graf G ditambahkan ke T jika ia tidak membentuk siklus di T.

Algoritma :
(Asumsi: sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya- dari kecil ke besar)

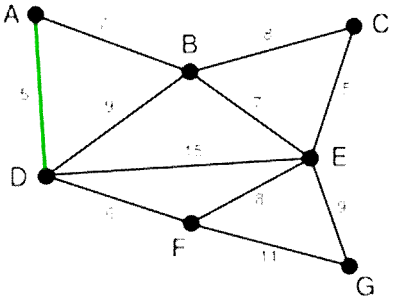
- T masih kosong
- Pilih sisi (*u,v*) dengan bobot minimum yang tidak membentuk sirkuit di T. Tambahkan (*u,v*) ke dalam T.
- Ulangi sebanyak n -1 kali

Kompleksitas waktu untuk algoritma ini adalah O(*E logE*) yang ekuivalen dengan O(*E logV*), dimana E menyatakan jumlah sisi dan *V* jumlah simpul dari graph G.

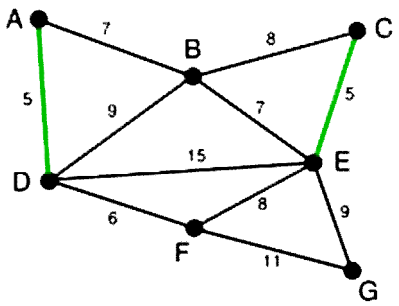
Contoh :
Berikut adalah graf berbobot yang akan dicari pohon merentang minimum nya,dengan bobot sebesar 90.



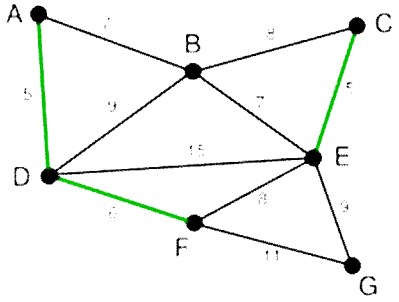
AD dan CE adalah sisi dengan bobot terkecil, pilih sembarang antara AD dan CE, pilih AD.



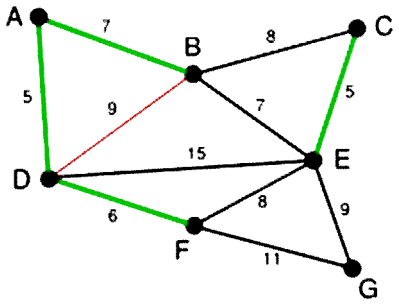
Karena CE sisi terkecil berikutnya, pilih CE



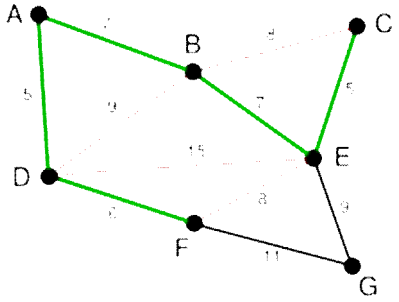
Selanjutnya, pilih sisi DF dengan bobot 6



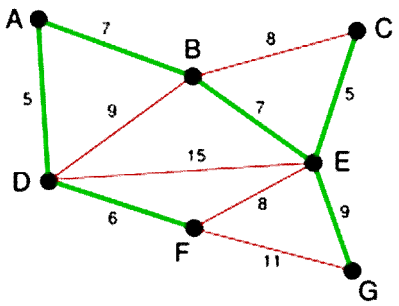
AB dan BE adalah sisi dengan bobot terkecil yang belum dipilih, pilih sembarang antara AB dan BE, pilih AB.



Karena BE sisi terkecil berikutnya, pilih BE



Simpul G adalah simpul terakhir yang belum dipilih, oleh karena itu pilih sisi yang terhubung dengan simpul G, yaitu GE dan GF, karena GE memiliki bobot lebih kecil, pilih GE.



Karena semua simpul telah dipilih, maka pohon merentang minimum ditunjukkan dengan warna hijau. Dengan bobot sebesar 39.

Dengan menggunakan struktur data sederhana kompleksitas waktu algoritma ini adalah O(E log E) yang ekuivalen dengan O (E log V). Keduanya ekuivalen karena :

- ♣ E bernilai maksimal V^2 dan log V^2 = 2 x log$V$ = O(log V)
- ♣ Jika kita tidak menghiraukan simpul-simpul yang terisolasi, V <= 2E, jadi log V adalah O(log E)

Hasil tsb diperoleh dengan melalui beberapa tahap, pertama, urutkan sisi-sisi dengan menggunakan *comparison sort* dengan waktu O(E log E). Kemudian, gunakan himpunan struktur data yang *disjoint* untuk menjaga tempat simpul yang mana ada di komponen mana.
Kemudian, tunjukkan O(E) operasi, untuk menemukan operasi - operasi dan kemungkinan satu *union* untuk setiap sisi.
Sehingga, waktu total adalah $O(E \log E) = O(E \log V)$.

Misalkan sisi-sisi yang ada telah diurutkan atau bisa diurut dalam waktu yang linear (dengan *counting sort* atau *radix sort*), algoritma ini dapat menggunakan himpunan struktur data yang *disjoint* dengan lebih rumit, sehingga memiliki kompleksitas waktu $O(E\ \alpha(V))$, dimana α adalah inverse dari fungsi Ackermann yang bernilai *single*, dan tumbuh dengan sangat lambat.

5. Algoritma Bernard Chazell

Algoritma yang dikembangkan oleh Bernard Chazell ini merupakan pengembangan dari algoritma Borůvka dan menggunakan fungsi Ackermann.

5.1 Fungsi Ackermann

Fungsi ini didefinisikan secara rekursif untuk bilangan integer tidak negatif m dan n sebagai berikut :

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Fungsi Ackermann dapat diekspresikan juga secara tidak rekursif menggunakan notasi *Conway chained arrow* :

$A(m, n) = (2 \rightarrow (n+3) \rightarrow (m - 2)) - 3$ untuk $m > 2$

Maka,

$2 \rightarrow n \rightarrow m = A(m+2,n-3) + 3$ untuk $n>2$

(n=1 and n=2 akan berkorespondensi dengan $A(m,-2) = -1$ and $A(m,-1) = 1$, yang secara logika dapat dijumlahkan).

Atau *hyper* operator :

$$A(m, n) = \text{hyper}(2, m, n + 3) - 3.$$

Untuk nilai m yang kecil seperti 1, 2, atau 3, fungsi Ackermann tumbuh relatif lambat sesuai n .
Untuk $m \geq 4$, fungsi ini tumbuh lebih cepat, bahkan $A(4,2)$ bernilai 2×10^{19728}

5.2 Algoritma

Untuk mencari pohon merentang minimum dari graf G dengan n simpul dan m sisi, gunakan fungsi msf (G, t) dengan parameter t,

$$t = \min\{i > 0 \mid n \leq S(i,d)^3\},$$

dimana $d = c[(m/n)^3]$

masukan fungsi msf ini adalah sebuah integer t ≥ 1 dan sebuah graf berbobot, dan mengembalikan nilai hutan merentang minimum (MSF).

msf(G, t) :

[1] jika t = 1 or n = O(1), menghasilkan MSF (G) dengan komputasi langsung
[2] lakukan c fase Boru°vka yg sekuensial
[3] bangun T dan bentuk graf B dari sisi yang tidak sesuai
[4] $F \leftarrow U_{z \in T}$ msf ($C_z \setminus B$, t - 1)
[5] kembalikan hasil msf($F \cup B$, t) ∪ {sisi-sisi hasil dari langkah [2]}

Langkah [1,2] : *fase Borůvka*

Untuk t = 1 diselesaikan dalam waktu $O(n^2)$ dengan melakukan fase Borůvka sampai graf G terikat pada satu simpul. Jika kita berhati - hati menghapus sisi-sisi yang ganda dan menjaga agar graf bebas dari sisi yang ganda, maka kita dapat menghitung semua fase total yaitu $O(n^2 + (n/2)^2 + \ldots..) = O(n^2)$ waktu.

Tujuan dari langkah [2] adalah mereduksikan jumlah simpul. Fase Borůvka mengubah G menjadi sebuah graf G_0 dengan $n_0 <= n/2^c$ simpul dan $m_0 <= m$ sisi. Transformasi ini membutuhkan O(n + m) waktu.

Langkah [3] *: membangun hirarki* T
Dengan $t > 1$ yang spesifik, begitu juga 'target' nilai n_z untuk setiap C_z, $n_z = S(t -1, S(t, d_{zk} - 1))^3$, dimana d_{zk} adalah tinggi dari $z \in T$.

Nyatakan $z = z_1,\ldots,z_k$ sebagai notasi lintasan yang aktif. Algoritma terkadang membuang/ menolak sisi-sisi dari G_0. setiap graf Cz termasuk semua sisi dari G_0 yang telah ditolak, ditahan karena graf G_0 sedang aktif.

Langkah [4] : *rekursif pada subgraf* T
Setelah membangun pohon T , pertimbangkan setiap anak z: pastikan C_z tidak mengandung satupun sisi yang ditolak. Misalkan $C_z \setminus B$ adalah subgraf dari C_z yang dipertahankan dengan menghapus semua sisi yang ditolak. Aplikasikan algoritma secara rekursif pada $C_z \setminus B$ setelah mengembalikan semua bobot tiap sisi ke nilai awalnya, dan melakukan *decrement* parameter t dengan satu.

Efek yang diharapkan adalah untuk memodifikasi ukuran target untuk pohon baru T yang akan dibangun. Keluaran F adalah sebuah himpunan dari simpul-simpul yang bergabung di C_z

Korespondensi antara simpul - simpul dari C_z dan anak dari z harus jelas. Pertimbangkan fusi pertama ke dalam simpul a dari C_z. Lebih penting, simpul a berkorespondensi dengan setiap adanak v dari z. Yang terjadi setelah fusi selama a dipertimbangkan dalam langkah [4] adalah tidak ada.

Kembali ke sebuah fusi khusus dari b ke a, dari semua sisi yang sedang berhubungan / bergabung dengan a dan terikat dengan b ditetapkan dalam F sebagai *original min-link* (a,b) jika sesuai. *Note :* menetapkannya dalam F hanya satu simpul dari bentuk (a,b) untuk setiap fusi adalah sama dengan menyelesaikan masalah hutan merentang minimum, relatif ke bobot - bobot awal, untuk kelomppok yang sesuai (*non -bad*), sisi - sisi ganda yang tidak ditolak bergabung dengan a dan b. Alasannya, jika *min-link* sesuai, maka bobotnya adalah bobot yang asli dan merupakan minimum dari kelompok / grup: secara konsekuen, sisi tersebut adalah pohon merentang minimum dari grup.

Melakukan fusi mungkin akan menyebabkan hal ini terlihat tidak jelas, tapi dari sudut pandang yang kompleks, semakin banyak jumlah mereka akan semakin bagus.

Langkah [5] : *rekursi akhir*
Pada langkah [3], semua sisi yang ditolak telah dibentuk selama proses konstruksi T dan membentuk graf B. Lalu tambahkan sisi - sisi dari F untuk menggabunkan subgraf F dan B dari G_0. Lalu aplikasikan cara pengurutan yang sama, hasil dari $msf(F \cup B, t)$ adalah sebuah himpunan sisi dengan titik akhir di G, bukan di G0. menambahkan sisi - sisi yang terikat pada langkah [2] menghasilkan pohon merentang minimum G.

5.3 Kompleksitas waktu
Dengan menggunakan induksi, telah dibuktikan bahwa msf (G,t) memerlukan waktu maksimal $bt(m + d^3(n - 1))$, dimana b adalah suatu konstanta yang bernilai cukup besar dan d adalah suatu integer yang juga bernilai cukup besar sehingga $n \leq S(t,d)^3$. Kasus - basis, t = 1, adalah mudah.

Untuk $n \leq S(1,d)^3 = 8d^3$ dan komputasinya memerlukan waktu $O(n^2) = O(d^3n) \leq b(m + d^3(n - 1))$. Jadi, dapat diasumsikan bahwa t > 1, dan karena langkah [1], maka n bernilai cukup besar.

Jika $n = S(t,d)^3$, maka d adalah tinggi dari pohon T .

Jika $n < S(t,d)^3$, maka sudah jelas kalau konstruksi akan di-terminasi sebelum akar dari T mencapai derajat target nya.
Fase Borůvka pada langkah [2] mengubah G menjadi sebuah graf G_0 dengan $n_0 \leq n/2^c$ simpul dan $m_0 \leq m$ sisi. Transformasi ini membutuhkan O(n + m) waktu. Kompleksitas membangun pohon T dalam langkah [3] didominasi dengan operasi *heap.*

Sehingga kompleksitas waktu untuk menyisipkan adalah $O(m_0)$, dan untuk menghapus dan menyatukan adalah $O(m_0)$.
Setiap operasi *heap* membutuhkan waktu konstan, secara konservatif, menghitung waktu t membutuhkan $O(m_0 + d^2n_0)$ ditambah waktu

untuk *bookkeeping* dari mengakses *heap* yang benar pada saat yang tepat.

Untuk setiap *adjacent* v ke rantai yang sedang aktif, pertahankan sisi-sisi luar yang berhubungan ke v dalam sebuah list berkait yang telah diurut.

Hal ini memungkinkan kita untuk menemukan, dalam waktu O(1), anak *z* dimana C_z adalah sisi luar yang berkaitan.

Kombinasi dari *height lists* dan struktur luar membuat kita dapat menjawab dalam waktu konstan : diberikan sebuah sisi luar (u,v) berkaitan ke Cz, apakah z' setelah z, dimana v *adjacent* ke Cz ?. Sebuah teknik minor : karena beberapa sisi di *height lists* mungkin ada di Cz, mungkin kita harus maju lagi dalam list untuk menemukan z' .

Hal ini berguna untuk menemukan lokasi *heap* - *heap* H(i,j) dimana merupakan tempat untuk meyisipkan sisi.

Dalam semua kasus, z_j tetap dan tidak boleh diganti dan kita harus menyisipkan sisi - sisi kedalam H(i,j) yang tepat. Untuk menghindari *traversing* ke sisi - sisi lain yang berkaitan dengan Cz, yang semuanya datang dari H(*, *), kelompokkan ulang mereka ke dalam sebuah sublist dalam *height list.*

Pertanyaan diatas tidak akan ditanya dengan menjadikan (u,v) salah satu dari sublist yang tidak perlu di *transverse* , dan pertanyaan ini selalu dapat dijawab dengan waktu konstan.

Sebuah ekstensi secara khusus menyebabkan sisi - sisi baru ditambahkan ke dalam *height list* dari simpul - simpul yang berkaitan dengan sisi ekstensi. Tidak ada struktur luar yang perlu diperbaharui, kecuali yang berkorespondensi dengan yang C_{zk} yang baru dibentuk.

Hal ini membutuhkan waktu yang proporsional untuk pertimbangan jumlah sisi - sisi baru.
Selama sebuah retraksi atau fusi, 2 atau lebih Cz hancur secara bersamaan. *Height lists* menjadi mudah untuk dipertahankan.

Dalam kasuk struktur luar, pembaharuan bobot per Cz adalah O(d) untuk konfigurasi ulang ditambah 1/d per simpul untuk pembaharuan akar dari pohon yang pendek (jika ada).

Total bobot dari bookkeeping seluruhnya adalah $O(m_0 + d^2n_0)$. Dalam penjumlahan, dengan memilih b yang cukup besar,

waktu untuk langkah [2,3] < $b/2\ (n + m + d^2n_0)$

Untuk langkah [4]
Pertimbangkan sebuah *internal node* z dari *T.*

Jika dz = 1, maka z dikatakan penuh.

Jika nilai n_z sama dengan $S(t,1)^3 = 8$.

Jika dz > 1, *node* z penuh, jika dan hanya jika nz = S() dan anaknya juga penuh. Untuk z yang tidak penuh, konstruksi dari Cz pasti telah diterminasi secara prematur.

Hal ini menunjukkan $Nz \geq (nz - 1)S(t, dz - 1)^3$, untuk semua dz > 1. Dengan konstruksi, jumlah simpul di Cz \ B maksimal $S(t - 1, S(t,dz - 1))^3$, sehingga kita bisa mencari waktu untuk $msf(C_z \setminus B, t - 1)$ dengan

$$b(t - 1)(m_z + S(t, d_z - 1)^3(n_z - 1)) \leq b(t - 1)(m_z + N_z),$$

dimana m_z adalah jumlah sisi di Cz \ B.
Pada langkah [4] kita mempunyai $\sum mz \leq m0 - |B|$ dan $\sum Nz \leq dn_0$ sehingga

waktu untuk langkah [4] < $b(t - 1)(m0 - |B| + dn_0)$

Akhirnya, langkah [5] rekursif terhadap $F \cup B$. jumlah simpulnya adalah $n0 < n \leq S(t,d)^3$ dan *F* tidak mempunyai siklus, sehingga dengan induksi

waktu untuk langkah [5] < $bt(n_0 - 1+ |B| + d^3(n_0 - 1))$.

Dengan menambahkan semua besar waktu setiap langkah, maka waktu eksekusi maksimal adalah

$$btm_0 + b(m/2 - m_0 + |B|) + 2btd^3n_0 + bn/2$$

Kemudian, dengan menggunakan fakta bahwa n0 <= n/2c, kita akan menemukan bahwa kompleksitas waktu dari msf(G,t) adalah $bt(m + d^3(n - 1))$, yang dapat dibuktikan kebenarannya dengan induksi.

Ketika menggunakan msf untuk menghitung pohon merentang minimum dari sebuah graf terhubung dengan n simpul dan m sisi, maka d pasti akan memenuhi persamaan $d^3n = O(m)$, dan seperti yang akan dibuktikan di bawah, $t = O(\alpha(m,n))$. Hal ini menyebabkan pohon

merentang minimum dari graf G mempunyai kompleksitas waktu O(m α (m,n)).

Lemma jika d = c[(m/n)1/3] dan t = min{i > 0 | n <= S(i,d)^3}, maka t = O((m,n)).

Bukti :
Fungsi Ackermann A(i , j) didefinisikan untuk i , j >= 0 :

$$\begin{cases} A(0 , j) = 2j, \text{ untuk } j >= 0; \\ A(i , 0) = 0 \text{ dan } A(i , 1) = 2, \text{ untuk } i>=1; \\ A(i , j) = A(i - 1, A(i, j - 1)), \text{ untuk } i>=1, \\ j \geq 2, \text{ dan } n, m > 0 \end{cases}$$

$$\alpha(m, n) = \min\{i \geq 1: A(i, 4[m / n]) > \log n\}.$$

untuk $i \geq 1$ dan $j \geq 4$,

$$A(3i, j) = A(3i - 1, A(3i, j - 1)) > 2^{A(3i, j - 1)} = 2^{A(3i - 1, A(3i, j - 2))}$$

Menggunakan A yg monoton, $A(3i, j - 2) \geq j$, diperoleh

$$A(3i, j) > 2^{A(i, j)}$$

Dengan menggunakan induksi, dapat ditunjukkan, untuk $u \geq 2$, $v \geq 3$, $A(u, v) \geq 2^{v+1}$, sehingga
$A(3i, j) = A(3i - 1, A(3i, j -1)) \geq A(3i - 1, 2^j) \geq A(i, 2^j)$.
secara trivial, $A(u - 1, v) \leq S(u, v)$, untuk $u, v \geq 1$, sehingga
$S(9\alpha(m, n) + 1, d) \geq A(9\alpha(m, n), d)$

Kemudian, dengan (5, 6) dan $d \geq 4$,

$$\begin{aligned} S(9\alpha(m, n) + 1, d) &> 2A^{(3\alpha(m, n), d)} \\ &\geq 2^{(\alpha(m, n), 2d)} \\ &\geq 2^{A(\alpha(m, n), 4[m/n])} \\ &> n , \end{aligned}$$

Maka, untuk t akan memenuhi persamaan

$$\mathbf{n \leq S(t, d)^3} \text{ dan } \mathbf{t \leq 9\alpha(m, n) + 1}$$

6. Algoritma waktu linier

Ide dari algoritma waktu linier ini algoritma Boruvka, dimana pada algoritma tersebut hanya memilih sisi - sisi untuk dimasukkan ke dalam pohon merentang minimum, namun tidak mengatakan tentang sisi - sisi yang tidak boleh berada dalam pohon merentang minimum tersebut.

Definisi:

Untuk setiap hutan F yang merupakan subgraf dari G dan setiap simpul *u,v* pada graf G, kami definisikan wf(u,v) sebagai berat maksimum dari setiap sisi dalam path yg unik diantara u dan v dalam F.
Jika tidak ada lintasan seperti itu, maka wf(u,v) = ∼
Untuk setiap sisi e = {u,v} kami sebut e F-berat jika we > wf(u,v). kalau tidak, e disebut F-ringan.

Algoritma :

1. terapkan 2 fase boruvka dan simpan sebagai sisi dari E1. buat paragraf baru G1
2. bentuh graf H dengan menyertakan setiap sisi dari G1 secara bebas dengan probabilitas ½. Temukan msf dari h secara rekursif dan simpan sebagai F
   identifikasi sisi-sisi f-berat dari G1 dan hapus untuk membentuk graf G2
3. temukan msf dari G2 simpan sebagai F'. msf dari G adalah F U E1.

Menentukan kompleksitas algoritma :
Menggunakan teknik "komposisi anak kiri".
Pohon dipartisi menjadi subtree-subtree dimana setiap subtree mempunyai akar. Setiap subtree mempunyai anak yang dapat dicapai melalui bagian kiri pohon.
Jika ada k sisi di bagian akan dari subtree - subtree tersebut, berapa banyak sisi dari keseluruhan subtree?
Jika ada k sisi di graf orang tua, G1, kita dapat mencari k/2 sisi di subgraf H hasil dari langkah 2. karena sisi yang ada lebih sedikit daripada yang ada di graf G1, pada simpul dengan level d dari akar, kita dapat menghitung jumlah sisi maksimal, $k/2^d$.

Lemma : untuk sebuah subgraf H dari G, F adalah hutan merentang minimum dari H. Jumlah sisi F - ringan di G paling banyak n/p

Bukti :
Kita proses sisi - sisi tersebut dalam urutan menaik, seperti Algoritma Kruskal. Kemudian, bentuk subgraf H dan msf F dari H pada waktu yang bersamaan. Untuk setiap sisi, tentukan apakah dia hutan-berat atau hutan-ringan.
Karena sisi yang lebih berat akan diproses belakangan, maka sisi F-berat akan tetap menjadi F-berat dan sisi F-ringan akan tetap menjadi F-ringan. Misalkan, ada sisi maksimum n di F, dan sisi F-ringan ada di bagian atas dari F, jadi kita hanya akan mendapatkan sisi - sisi F-ringan setelah melihat n *heads.*

Dengan menggunakan lemma ini, kita berharap agar ada 2v/4 sisi dalam sebuah subproblem yang mempunyai v simpul sebelum langkah 1.
Karena sebuah subproblem yang benar ada di level d, maka ada $n/4^d$ simpul, sehingga jumlah simpul di subproblem yang benar adalah

$$\sum_{d=1}^{\sim} 2^{d-1}(2n)/4^d = n/2$$

kombinasi ini dengan ekspektasi 2k sisi di seluruh anak kiri dalam subproblem yang benar dan m sisi pada bagian akar, akan diperoleh kompleksitas waktu O(m + n).
Dan hal ini juga membuktikan bahwa ekspektasi waktu eksekusi untuk algoritma ini adalah O(m).

7. Kesimpulan

- Algoritma Borůvka memiliki kompleksitas waktu O(E logV)
- Algoritma kruskal memiliki kompleksitas waktu O(E logE), dan nilai ini ekuivalen dengan O(E logV)
- Algoritma prim memiliki kompleksitas waktu $O(V^2)$, namun bila diterapkan metode yang lebih rumit kompleksitas algoritma ini adalah O(E + V logV)
- Algoritma Bernard Chazell memiliki kompleksitas waktu O(e α(e,*v*)) dimana fungsi α tumbuh dengan sangat lambat, jadi untuk semua kondisi nilainya bisa dihampirkan dengan konstan yang lebih kecil dari 4; sehingga algoritma ini mempunyai kompleksitas waktu yang mendekati nilai O(e)
- Algoritma waktu linier memiliki kompleksitas waktu O(m)
- Algoritma Borůvka memiliki banyak cara untuk dikembangkan menjadi suatu algoritma baru
- Waktu eksekusi setiap algoritma dapat diperkecil dengan menggunakan metode yang lebih rumit untuk pengujian algoritma tersebut.
- Algoritma Bernard Chazell memiliki kompleksitas waktu tercepat dibandingkan algoritma lainnya
- Algoritma Prim akan memiliki waktu eksekusi lebih cepat bila digunakan pada *dense graf*
- Algoritma Kruskal akan memiliki waktu eksekusi lebih cepat bila digunakan pada sparse *graf*
- Algoritma Prim mempunyai kemiripan dengan algoritma Dijkstra

DAFTAR PUSTAKA

[1] Morris, John. (1998). Data Structures and Algorithm .*http://www.cs.auckland.ac.nz.* Tanggal akses: 28 Desember 2006 pukul 13.32

[2] Munir, Rinaldi. (2006). Bahan Kuliah IF2153 Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] Kruskal's algorithm - Wikipedia, the free encyclopedia *http://en.wikipedia.org/kruskal's_algorithm.htm* . Tanggal akses: *28 Desember 2006 pukul 13.32*.

[4] Boruvka's algorithm - Wikipedia, the free encyclopedia *http://en.wikipedia.org/boruvka's_algorithm.htm* . Tanggal akses: *28 Desember 2006 pukul 13.32*.

[5] Prim's algorithm - Wikipedia, the free encyclopedia *http://en.wikipedia.org/prim's_algorithm.htm* . Tanggal akses: *28 Desember 2006 pukul 13.32*.

[6] Ackermann's function - Wikipedia, the free encyclopedia *http://en.Wikipedia.org/ Ackermann_function.htm* Tanggal akses: *2 Januari 2007 pukul 10.30*

[7] Chazelle, Bernard (1999). A Minimum Spanning Tree Algorithm with inverse Ackermann Type Complexity *http://www.cs.princeton.edu* . Tanggal akses: *28 Desember 2006 pukul 13.32*.

[8] Bader, David. A (1999). Fast Shared-Memory Algorithms for Computing Minimum Spanning Forest of Sparse Graphs

http://www.ece.unm.edu. Tanggal akses: *28 Desember 2006 pukul 13.32.*

[9] Skiena, Steven (1997). Minimum Spanning Tree *h*ttp://www2.toki.or.id/book/AlgDesign Manual/BOOK/BOOK4/NODE161.HTM Tanggal akses: 4 Januari 2007 pukul 13.32

[10] L, Allison (1999). Undirected Graph and Minimum Spanning Tree *h*ttp://www2.toki.or.id/book/AlgDesign Manual/BOOK/BOOK4/NODE161.HTM Tanggal akses: 4 Januari 2007 pukul 13.32