Pengelompokan Organisme Dengan Menggunakan Algoritma Kruskal

Alif Raditya Rochman - 13511013
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511013@std.stei.itb.ac.id

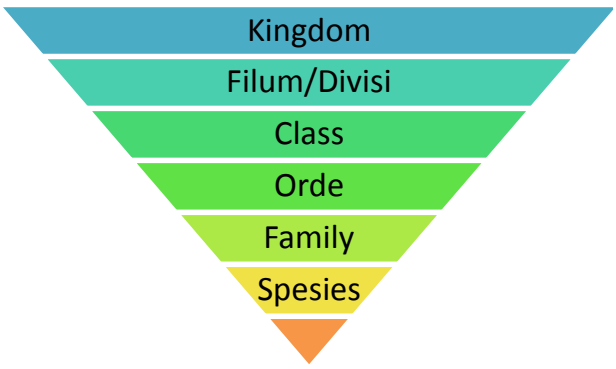
***Abstract*—Dengan banyaknya jumlah organisme yang mencapai puluhan juta, kita memerlukan suatu cara yang mangkus untuk mengelompokkan keseluruhan organisme. Untuk menyeselesaikan masalah ini, kita akan mencoba memodelkan permasalahan ini sebagai permasalahan graf. Permasalahan berubah menjadi modifikasi dari permasalahan *minimum spanning tree*. Dengan begitu kita dapat menyelesaikan permasalahan dengan algoritma yang serupa dengan *minimum spanning tree* dan kompleksitas waktu yang relatif sama.**

***Index Terms*—Pengelompokan Organisme, Graf, *Minimmum Spanning Tree*, Kruskal**

I. Pendahuluan

Jumlah organisme yang ada di bumi diperkirakan mencapai puluhan juta [1]. Untuk memudahkan manusia khususnya peneliti untuk mengenali dan mempelajarinya, diperlukan suatu pengelompokan makhluk hidup. Pengelompokan ini dapat didasarkan pada perilaku, habitat, atau ciri-cirinya.

Saat ini kita telah mengenal sebuah sistem taksonomi yang pertama-kali dikenalkan oleh Carolus Linnaeus[2]. Sistem taksonomi yang dia kenalkan terdiri dari 6 tingkat yaitu *kingdom*, *phylum* atau *division*, *class*, *orde*, *family*, dan *spesies*. Apabila dua organisme dimasukkan ke dalam kelompok taksonomi yang tingkatnya rendah, berarti persamaan antara dua organisme tersebut lebih banyak. Apabila dua organisme dimasukkan ke dalam kelompok taksonomi yang tinkatnya tinggi, berarti persamaan antara dua organisme tersebut lebih sedikit.



Gambar 1.1 Taksonomi Organisme

Terkadang kita tertarik untuk mengelompokkan organisme dengan sistem pengelompokkan yang berbeda dari sistem taksonomi yang ada. Misalkan kita ingin mengelompokkan *n* organisme ke dalam *k* kelompok. Kita ingin perbedaan antar kelompok adalah maksimal. Kita membutuhkan suatu struktur data dan algoritma yang mangkus, karena nilai *n* dapat mencapai puluhan juta (jumlah organisme dalam muka bumi mencapai puluhan juta).

Untuk menyelesaikan permasalahan ini, kita perlu melakukan sebuah pemodelan sehingga dapat lebih mudah diselesaikan. Kita akan mencoba melakukan pemodelan permasalahan ini sebagai suatu permasalahan graf.

II. Teori Dasar Terkait

A. Graf

Graf adalah sebuah kumpulan simpul yang dihubungkan dengan sejumlah sisi. Sebuah sisi menghubungkan tepat dua buah simpul

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka graf dapat digolongkan menjadi dua jenis. Graf sederhana adalah graf yang tidak mengandung gelang maupun sisi ganda, sedangkan Graf tak-sederhaa adalah graf yang mengandung gelang atau sisi ganda. Gelang adalah simpul yang menghubungkan dua simpul yang sama. Sedangkan sisi ganda adalah bilamana dua buah sisi menghubungkan dua buah simpul yang sama secara langsung.

Sisi pada graf dapat memiliki orientasi arah. Sisi dikatakan memiliki orientasi arah apabila suatu sisi *e* menghubungkan simpul dari *u* ke *v* tetapi tidak menghubungkan *v* ke *u*. Graf yang seluruh sisinya berarah disebut graf berarah, sedangkan graf yang seluruh sisinya tidak berarah disebut graf tak-berarah.

Sisi pada graf juga dapat memiliki bobot. Graf yang seluruh sisinya memiliki bobot disebut graf berbobot, sedangkan graf yang seluruh sisinya tidak memiliki bobot disebut graf tak-berbobot.

Graf yang setiap pasang simpulnya di hubungkan oleh sisi maka disebut graf lengkap. Graf lengkap akan selalu memiliki $n*(n-1)/2$ simpul dimana *n* adalah jumlah simpul.

Sebuah graf T dikatakan subgraf dari G apabila setiap simpul pada graf T terdapat juga di graf G dan setiap sisi pada graf T terdapat juga di graf G. Dengan kata lain apa bila V_G himpunan simpul pada G, V_T himpunan simpul pada T, E_G himpunan sisi pada G, dan E_T himpunan sisi pada T maka berlaku $V_T \subseteq V_G$ dan $E_T \subseteq E_G$ [3].

B. Minimmum Spanning Tree

Apabila kita menghapus sejumlah sisi pada G sehingga seluruh simpul pada G tetap terhubung dan total bobot pada graf G adalah minimum, kita akan mendapatkan sebuah graf baru T yang merupakan subgraf dari G. Dengan kata lain, misalkan dua buah simpul v_i dan v_j dimana $1 \leq i , j \leq n$ dan $i \neq j$ di hubungkan oleh sisi e_k dengan bobot w_k, kita ingin menghapus sejumlah sisi pada G sehingga $\sum_{i=1}^{k} w_k$ minimum dan v_i dan v_j tetap terhubung.

Perhatikan bahwa T merupakan pohon karena untuk setiap pasang simpul v_i dan v_j dimana $1 \leq i , j \leq n$ dan $i \neq j$ hanya akan terdapat satu buah jalur. Pohon T yang kita dapat itulah yang disebut *minimum spanning tree*. Pada *minimum spanning tree*, seluruh simpulnya terhubung. Selain itu tidak akan ditemukan siklus di dalamnya [4].

Untuk mendapatkan *minimum spanning tree* dari sebuah graf berbobot, terdapat dua algoritma yang terkenal yaitu algoritma kruskal dan algoritma prim. Kali ini kita akan membatasi bahasan kita ke algoritma kruskal saja.

C. Algoritma Kruskal

Algoritma kruskal diperkenalkan oleh Joseph Kruskal untuk mencari *minimum spanning tree* dari graf berbobot. Algoritma ini termasuk ke dalam algoritma rakus. [5]

Cara kerja algoritma ini adalah sebagai berikut :

a. Inisialisasi graf T sebagai graf kosong
b. Buat sejumlah himpunan dimana setiap simpul pada graf G dibuat sebagai himpunan yang berbeda
c. Buat list E yang berisikan seluruh sisi pada graf G
d. Selama list E belum kosong, lakukan langkah berikut :
    - Pilih sisi dengan bobot paling kecil dari list E dan hapus dari list E.
    - Bila sisi tersebut menghubungkan dua buah himpunan yang berbeda, maka gabungkan dua buah himpunan tersebut menjadi satu hipunan
    - Tambahkan sisi tersebut kedalam T
e. T adalah *minimum spanning tree* dari G.

D. Disjoint Set

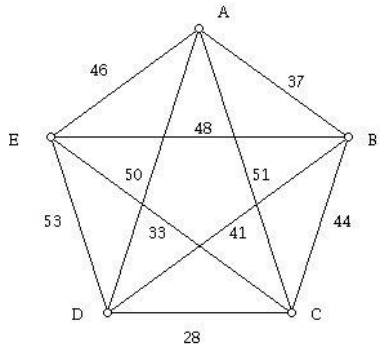
Perhatikan bahwa pada algoritma kruskal terdapat operasi penggabungan himpunan dan pencarian himpunan. Sekarang kita akan lihat bagaimana struktur data yang tepat agar operasi tersebut tetap mangkus.

Disjoint set adalah sebuah struktur data yang menangani himpunan elemen yang elemen-elemennya tidak *overlapping*. Struktur data ini memiliki tiga fungsi dasar yaitu *MakeSet* yaitu membuat himpunan baru, *FindSet* yaitu mencari tahu himpunan yang mencangkup elemen tertentu, dan *Union* yaitu menggabungkan dua himpunan menjadi satu himpunan.

Untuk mendapatkan performa terbaik dari struktur data ini, akan sangat baik untuk menggunakan disjoint set yang menggunakan *union by rank* dan *path compression*[6].

III. PEMODELAN PERMASALAHAN

Mari kita lihat lagi permasalahannya. Kita ingin mengelompokkan *n* organisme ke dalam *k* kelompok. Kita dapat memperlakukan organisme sebagai simpul dari suatu graf. Kita juga dapat mendefinisikan sisi antar simpul sebagai hubungan kesamaan antara dua sisi. Bobot sisi kita definisikan sebagai nilai perbedaan antara dua organisme. Semakin banyak perbedaan antara dua organisme maka bobotnya pun akan semakin besar.



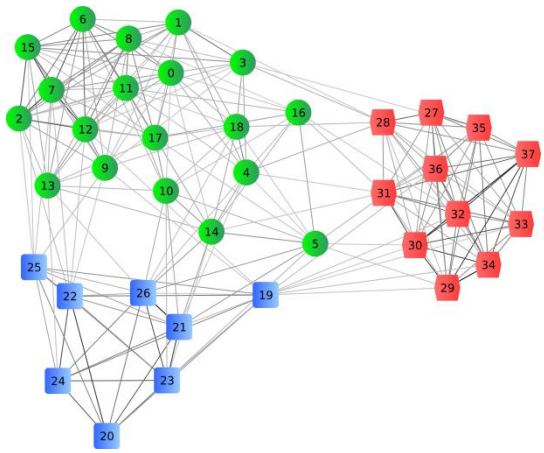
Gambar 3.1 Graf sederhana lengkap berbobot tak-berarah

Bila kita memodelkan permasalahannya sebagai graf, dapat dilihat bahwa graf yang terbentuk adalah graf sederhana lengkap berbobot tak-berarah seperti pada gambar 3.1. Dikatakan graf sederhana karena kita tidak ingin untuk membuat nilai kesamaan antara sebuah simpul yang sama dan tidak mungkin dua buah simpul memiliki lebih dari satu nilai kesamaan sehingga tidak mungkin terbentuk sisi gelang maupun sisi ganda. Dikatakan graf lengkap karena untuk setiap $1 \leq i , j \leq n$ dan $i \neq j$ akan selalu ada sisi yag menghubungkan V_i dengan V_j. Dikatakan graf berbobot karena setiap sisi yang ada memiliki bobot. Dikatakan graf tak-berarah dikarenakan untuk setiap $1 \leq i , j \leq n$ dan $i \neq j$ sisi yang menghubungkan V_i ke V_j juga menghubungkan V_j ke V_i dan bobot V_i ke V_j pasti sama dengan V_j ke V_i.

Sekarang kita telah berhasil membuat graf dari permasalahan yang ada. Tetapi kita sama sekali belum menyelesaikan permasalahannya. Kita ingin membuat k kelompok sehingga bobot antar kelompok adalah maksimal.

Kita telah mengetahui definisi bobot antar simpul, tetapi kita belum mengetahui definisi bobot antar kelompok. Perhatikan bahwa kelompok A dengan jumlah

kelompok S_A dan kelompok B dengan jumlah kelompok S_A akan ada sejumlah pasangan V_{Ai} dan V_{Bj} dimana $1 \le i \le S_A$ dan $1 \le j \le S_B$ yang memiliki bobot W_{ij}. Kita dapat mendefinisikan bobot antar kelompok *WC* sebagai bobot minimal W_{ij} untuk setiap $1 \le i \le S_A$ dan $1 \le j \le S_B$.



Gambar 3.2 Graf yang dikelompokkan menjadi tiga

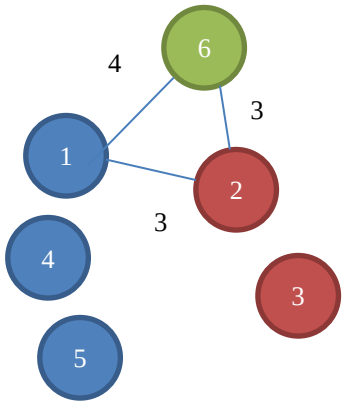
Permasalahannya sekarang menjadi seperti ini, diberikan sebuah graf sederhana lengkap berbobot tak-berarah dengan *n* simpul, bagi menjadi *k* kelompok sedemekian sehingga bobot minimal antara dua simpul pada kelompok berbeda itu maksimal. Perhatian bahwa jumlah anggota tiap kelompok tidak harus sama. Terdapat kemungkinan lebih dari satu jawaban, tetapi tetap menghasilkan jarak antar kelompok yang sama.

Sebagai contoh, diberikan tabel *W* pada tabel 3.1. Terdapat 6 organisme yang akan dibagi menjadi 3 kelompok. Tentukan masing-masing anggota tiap sedemikian sehingga bobot antar kelompoknya adalah maksimal.

W	1	2	3	4	5	6
1	-	3	3	1	1	4
2	3	-	2	3	4	3
3	3	2	-	5	6	
4	1	3	5	-	5	7
5	1	4	6	5	-	7
6	4	3	4	7	7	-

Tabel 3.1 Tabel nilai perbedaan W

Kita dapat membagi keenam organisme seperti pada gambar 3.3.



Gambar 3.3 Solusi dari persoaalan contoh

Pada gambar 3.3, garis yang digambarkan adalah bobot antar kelompok. Perhatikan bahwa antara kelompok biru dan kelompok hijau tidak ada lagi bobot antar simpul yang berbeda kelompok yang lebih kecil daripada 4. Sedangkan antara kelompok merah dan hijau, tidak ada lagi bobot antar simpul yang berebda kelompok yang lebih kecil daripada 3. Begitu juga antara kelompok merah dan kelompok biru, tidak ada lagi bobot antar simpul yang berbeda kelompok yang lebih kecil daripada 3. Sehingga gambar 3.3 adalah solusi dari contoh persoalan yang diajukan.

IV. Algoritma Penyelesaian

Ide penyelesaiannya adalah pertama-tama kita anggap terdapat *n* buah kelompok yang tiap kelompok beranggotakan satu individu. Secara terus-menerus, cari dua simpul yang berbeda kelompok yang dihubungkan dengan sisi dengan bobot terkecil dan gabungkan kelompok tersebut sampai terbentuk k kelompok.

Ide penyelesaian ini adalah *greedy*. Kita berpikir *greedy* karena kita menganggap dengan selalu mengambil bobot terkecil adalah pilihan yang optimal tanpa mempertimpangkan bobot yang lebih besar dahulu.

Perhatikan bahwa ide penyelesaian diatas serupa dengan algoritma kruskal. Perbedaannya hanya pada kondisi berhenti. Pada algoritma kruskal, kondisi berhentinya yaitu ketika seluruh simpul telah terhubung menjadi satu kelompok atau seluruh sisi telah selesai dianalisis. Sedangkan pada ide penyelesaian masalah ini kondisi berhentinya yaitu ketika terbentuk k kelompok.

Kita dapat memandang penyelesaian ini dari arah yang berbeda. Kita bisa saja mengatakan bahwa untuk membuat *k* kelompok dari *n* buah simpul pada graf berbobot kita hanya perlu mencari *minimum spanning tree* dari algoritma tersebut dan hapus k-1 sisi dengan bobot paling besar. Dengan cara itu akan terbentuk k graf atau bisa kita sebut k kelompok.

Sekarang kita akan mencoba merumuskan pseudocodenya. Diberikan graf G, buatlah graf T dimana T adalah *minimum spanning tree* dari G. Pertama-tama kita inisialisasikan T sebagai graf kosong. Kemudian untuk setiap simpul *v* dari G kita buat himpunan sendiri

menggunakan struktur data *disjoint set*. Lalu urutkan sisi *e* dengan bobot terurut menaik. Untuk setiap sisi *e* kita cek apakah sisi tersebut menghubungkan dua simpul yang berbeda himpunan. Bila berbeda himpunan, maka kita akan gabungkan himpunan dimana elemen u dan v berada kemudian kita masukkan sisi *e* = (*u*,*v*) ke dalam graf T. Realisasi pseudocodenya dapat dilihat dibawah.

```
MST_Kruskal (G)
    T = Nil
    for setiap simpul v dari G
        MakeSet(v)
    Urutkan sisi e = (u,v) dengan
bobot terurut menaik
    for setiap sisi e = (u,v) dari G
        if (FindSet(u) != FindSet(v))
            T = T U { (u,v) }
            Union(u,v)
    return T
```

Setelah mendapatkan *spanning tree* T, kita akan menghapus *k* sisi dari T sehingga membentuk hutan F yang terdiri dari *k* pohon. Setiap pohon akan melambangkan satu kelompok tersendiri.

```
DeleteKEdge (T,k)
    F = T
    Urutkan sisi e (u,v) dengan bobot
terurut menaik
    for k sisi tertinggi
        F = F -  { (u,v) }
    return F
```

VI. Pembuktian Kebenaran

Sampai sekarang, kita telah merumuskan algoritma untuk menyelesaikan permasalahan awal. Tetapi kita belum membuktikan bahwa algoritma yang kita buat adalah benar. Sekarang kita akan mencoba memberikan pembuktian terhadap algoritma yang telah dirumuskan sebelumnya.

Pertama-tama, kita perlu membuktikan bahwa dengan algoritma kruskal akan terbentuk *spanning tree*. Misalkan G adalah graf berbobot tak-berarah dan T adalah subgraf yang dihasilkan dari algoritma kruskal. Dari desain, kita akan selalu menambahkan sisi ke T yang menggabungkan dua buah himpunan. T tidak mungkin terdapat siklus, karena untuk membentuk siklus, perlu ditambahkan satu sisi yang menghubungkan dua simpul dari himpunan yang sama. Sehingga dapat disimpulkan T adalah *spanning tree* dari G.

Kita akan membuktikan bahwa dengan menghapus k-1 sisi dengan bobot terbesar dari *minimum spanning tree* akan menghasilkan kelompok yang optimal. Misalkan C adalah pengelompokan hasil algoritma dan C' adalah pengelompokan yang berbeda. Akan kita tunjukkan bahwa C' pasti memiliki bobot antar kelompok yang sama atau lebih kecil dari C. Karena C ≠ C', maka pasti ada pasangan Vi dan Vj yang berada di kelompok yang sama pada C tetapi berbeda di C'. Karena berada di kelompok yang sama pada C, maka bobot antara Vi dan Vj <= d dimana d adalah jarak maksimal dalam kelompok yang lebih kecil daripada jarak minimal antar kelompok. Karena sisi tersebut dikelompokkan pada kelompok yang berbeda di C', maka C' <= d yang menyebabkan C' tidak optimal.

VII. Kompleksitas Algoritma

Sekarang kita akan menghitung kompleksitas waktu algoritma yang kita buat. Perhatikan bahwa untuk mendapatkan k kelompok dari n buah item, kita akan menjalankan fungsi `MST_Kruskal (G)` dan fungsi `DeleteKEdge (T,k)` masing-masin sebanyak satu kali. Sehingga untuk mendapatkan komplekitas waktu agoritma yang telah kita buat kita perlu memperhatikan kompleksitas waktu masing-masing.

Perhatikan fungsi pertama, pada baris ketiga dan keempat kita akan membuat himpunan sejumlah *V*. Kompleksitas waktu dari proses tersebut adalah O(*V*) dengan catatan bahwa untuk membuat himpunan hanya memerlukan waktu O(1).

Pada baris berikutnya kita akan mengurutkan sisi *e* (u,v) sejumlah *E*. Kompleksitas waktu pengurutan beragam tergantung algoritma yang digunakan. Bila kita menggunakan algoritma seperti Insertion Sort atau Bubble Sort, maka kompleksitasnya menjadi O(E^2). Sedangkan bila kita menggunakan algoritma yang lebih mangkus seperti quick sort atau merge sort, maka kompleksitasnya menjadi O(*E*log*E)*. Karena lebih mangkus, kita akan menggunakan algoritma quick sort untuk melakukan pengurutan ini.

Kemudian pada baris berikutnya kita akan melakukan iterasi untuk mengecek apakah dua buah simpul terdapat di hipunan yang sama atau tidak. Pada saat ini peran penggunan Disjoint Set akan sangat terasa. Kompleksitas waktu *FindSet* dan *Union* adalah sama yaitu O(α(*V*)) dimana α adalah *inverse* dari fungsi Ackerman sehingga fungsi ini adalah fungsi yang sangat lambat naik. Karena nilai fungsi α hanya bernilai 5 untuk semua input yang wajar di kehidupan, maka kompleksitas waktunya untuk iterasi ini adalah O(*E*).

Kompleksitas waktu dari fungsi pertama bila dikalkulasikan menjadi O(*E* log *E*) = O(*E* log *V*). Karena *V* = *n* dan *E* = *n**(*n*-1)/2 maka kompleksitasnya menjadi O(n^2).

Sekarang perhatikan fungsi kedua, pada baris kedua terdapat operasi pengurutan sisi (u,v) sejumlah E. Sehingga kompleksitas waktunya bila menggunakan algoritma quick sort menjadi O(E log E).

Pada baris berikutnya terdapat iterasi untuk operasi penghapusan sisi sebanyak k. Kompleksitas waktunya menjadi O(k) dengan catatan operasi penghapusan sisi memakan waktu O(1).

Kompleksitas waktu dari fungsi kedua bila dikalkulasian menjadi O(E log E + k). Karena E log E + E ≥ E log E + k, kompleksitas waktunya dapat kita tulis O(E log E). Karena *V* = *n* dan *E* = *n**(*n*-1)/2 maka kompleksitasnya menjadi O(n^2 log *n*).

Kompleksitas keseluruhan penyelesaiannya menjadi $O(n^2 \log n)$ dimana n adalah jumlah organisme yang ingin kita kelompokkan.

Kompleksitas waktu yang dihasilkan cukup besar karena graf yang terbentuk adalah graf sempurna, sehingga jumlah sisinya sangat banyak. Sebenarnya kita dapat melakukan kompleksitas waktu yang lebih cepat bila kita menggunakan counting sort dibandingkan dengan quick sort. Hal ini dikarenakan counting sort hanya memiliki kompleksitas waktu $O(n)$. Akan tetapi counting sort memakan memori yang banyak, sehingga sulit untuk digunakan dalam permasalahan ini. Salah satu trik lain yaitu dengan mengunakan *heuristic*, tetapi tidak akan dibahas dalam makalah ini.

VIII. KESIMPULAN

Kesimpulan dari tulisan ini yaitu bahwa kita dapat menyelesaikan permasalahan pengelompokan spesies dengan menggunakan Kruskal. Dengan pemanfaatan algoritma Kruskal dan struktur data Disjoint Set, permasalahan pengelompokan organisme diselesaikan dengan kompleksitas waktu $O(n^2 \log n))$ yang tidak terlalu mangkus, akan tetapi sudah cukup memuaskan untuk n yang tidak terlalu besar.

IX. UCAPAN

Rasa syukur dan terima kasih saya panjatkan kepada tuhan yang maha esa yang tetap memberikan saya kesempatan untuk dapat menyelesaikan makalah ini. Rasa terima kasih juga saya sampaikan kepada Dr. Ir. Rinaldi Munir dan Dra. Harlili S., M.Sc sebagai dosen Struktur Diskrit yang telah membimbing saya dalam menyelesaikan makalah ini. Selain itu saya juga berterima kasih kepada teman-teman saya yang selalu memberikan dukungan selama pengerjaan makalah ini.

REFERENCES

[1] K.V. Krishnamurthy, Textbook of biodiversity, New Hampshire:Science Publisher, 2001, pp. 23 - 40

[2] P. Mikulecky, AP Biology For Dummies, Indianapolis:Wiley,2008,pp. 185 - 200

[3] K.H. Rosen, Discrete Mathematics and Its Application, 6th ed., New York: McGraw-Hill, 2007, pp. 589–630.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithm, 3rd ed., Massachusetts:MIT Press, 2009, pp. 624-627

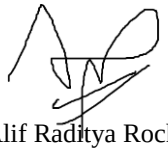
[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithm, 3rd ed., Massachusetts:MIT Press, 2009, pp. 631-633

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithm, 3rd ed., Massachusetts:MIT Press, 2009, pp. 568-570

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Desember 2012



ttd

Alif Raditya Rochman
13511013