Perbandingan BFS, DFS dan Greedy pada Permainan Logika Crossing Bridge

Susanti Gojali and 13512057[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*susantigojali@itb.ac.id*

***Abstrak*—Crossing Bridge adalah salah satu dari banyak permainan logika. Permainan Crossing Bridge ini bertujuan untuk memindahkan 4 orang dari satu sisi jembatan menuju sisi lain dengan menggunakan sebuah lampu. Permainan ini dapat diselesaikan dengan banyak alternatif solusi.**

Pada permainan ini dapat diterapkan beberapa strategi algoritma dalam pencarian solusi. Makalah ini akan membahas pencarian solusi Crossing Bridge dengan menggunakan algorirma Breadth First Search(BFS), Depth First Search(DFS), dan greedy. Selain itu, makalah ini akan membandingkan ketiga strategi pencarian dalam menemukan solusi.

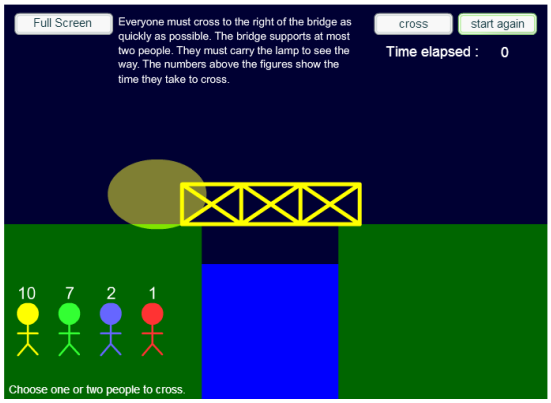
***Kata Kunci*—BFS, DFS, Greedy, Perbandingan, Permainan Logika.**

I. PENDAHULUAN

Saat ini, semua anak pasti pernah memainkan permainan. Permainan terdiri dari banyak genre. Ada yang bergenre masak, musik, olahraga, balapan dan tembakan. Dari banyak jenis permainan tersebut, ada permainan yang membutuhkan logika untuk menyelesaikannya. Permainan ini tidak bisa diselesaikan hanya dengan sembarangan/asal tebak. Permainan logika membutuhkan pemikiran lebih untuk mendapatkan solusi dari permainan tersebut. Salah satu dari permainan logika adalah Crossing Bridge.

Crossing Bridge adalah salah satu permainan logika yang cukup popular dan sudah ada cukup lama. Permainan ini bertujuan untuk memindahkan orang dari satu sisi jembatan ke sisi jembatan lainnya. Pada awalnya, terdapat empat orang di satu sisi jembatan. Setiap orang memiliki kecepatan jalan yang berbeda-beda yaitu masing-masing 1,2,7 dan 10 detik.

Pada permainan ini terdapat satu lampu untuk menyebrang jembatan. Setiap kali melakukan penyebrangan diperlukan lampu dan hanya maksimal dua orang yang bisa menyebrang jembatan. Jika dua orang menyebrang secara bersamaan, maka waktu yang diperlukan untuk menyebrang mengikuti waktu terlama dari orang yang menyebrang. Permainan ini mencari waktu minimum yang dibutuhkan untuk menyebrangkan empat orang tersebut ke sisi jembatan lainnya.



Gambar 1. Permainan Crossing Bridge

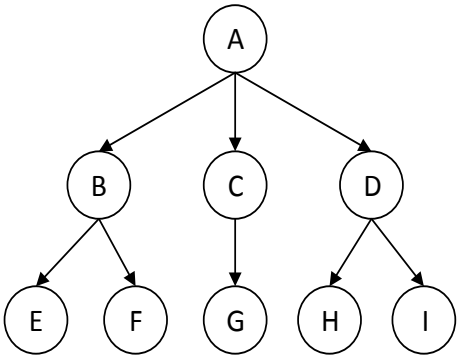
Pada umumnya, permainan logika memiliki banyak alternatif solusi. Begitu pun pada permainan Crossing Bridge. Pencarian solusi untuk menyebrangkan orang ke sisi jembatan lainnya dapat menggunakan beberapa strategi algoritma yaitu Breadth First Search(BFS), Depth First Search(DFS), dan greedy.

II. DASAR TEORI

A. Breadth First Search (BFS)

Breadth First Search adalah metode pencarian secara melebar. BFS mengunjungi semua simpul yang belum dikunjungi dari akar. Algoritma BFS menggunakan sebuah queue Q untuk menyimpan antrian. Algoritma BFS ditunjukkan sebagai berikut.

1. Memasukkan simpul akar misal simpul A ke Q.
2. Mengambil simpul dari antrian pertama pada Q.
3. Jika simpul tersebut merupakan solusi, maka pencarian dihentikan dan solusi telah ditemukan.
4. Jika simpul tersebut bukan solusi, maka memasukkan simpul yang bertetangga dengan simpul tersebut ke antrian.
5. Jika antrian kosong, maka solusi tidak ditemukan.
6. Jika antrian tidak kosong, maka ulangi pencarian dari langkah 2.



Gambar 2. Pencarian secara BFS

Jika melakukan pencarian secara BFS pada graf di gambar 2, maka penelusurannya dimulai dari simpul A. Setelah itu, akan dibangkitkan semua simpul anak dari A yaitu B, C dan D. Lalu dimulai lagi dengan membangkitkan anak simpul B yaitu E dan F. Setelah itu, G dibangkitkan karena anak dari simpul C serta H dan I dibangkitkan oleh D. E dan F tidak memiliki anak lagi sehingga tidak ada simpul yang bisa dibangitkan. Begitu pula G, H dan I. Secara keseluruhan, penelusurannya akan menjadi A-B-C-D-E-F-G-H-I.

Pseudocode BFS ditunjukkan sebagai berikut. Misal terdapat graf G, simpul akar v dan tabel Boolean yang bernama dikunjungi berukuran sebesar jumlah simpul.

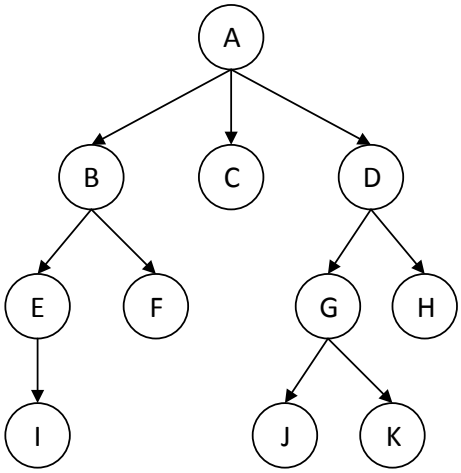
```
Prosedur BFS(G,v)
{Traversal Graf G secara BFS
Masukan: G adlah graf, v adalah simpul awal
Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
W : integer
Q : antrian

Prosedur BuatAntrian(input/output q:antrian)
{Membuat antrian kosong}

Prosedur MasukAntrian(input/output q:antrian, input v:integer)
{Memasukkan v ke belakang antrian q}

Prosedur HapusAntrian(input/output q:antrian, output v:integer)
{Menghapus v kepala antrian }

Function AntrianKosong()→boolean
{mengembalikan true jikan antrian kosong, false jika antrian tidak kosong}

Algoritma
    BuatAntrian(Q)
    Write(v)
    MasukAntrian(Q,v)
    Dikunjungi[v]← true
    While not AntrianKosong(Q) do
        HapusAntrian(Q,v)
        for semua simpul w yang bertetangga dengan simpul v
            if not dikunjungi[w] then
                Write(w)
                MasukAntrian(Q, w)
                Dikunjungi[w]← true
            end if
        end for
    end while
```

Kelebihan metode BFS adalah tidak menemui jalan buntu, menjamin menemukan solusi jika memang solusinya ada dan solusi yang ditemukan adalah solusi yang optimal. Kelemahan metode ini adalah membutuhkan memori yang cukup banyak karena harus menyimpan semua simpul yang pernah dibangkitkan dan membutuhkan waktu yang lama.

B. DFS

Depth First Search adalah metode pencarian secara mendalam. Algoritma DFS ditunjukkan sebagai berikut. Misal simpul akar adalah simpul v.

1. Mengunjungi simpul v
2. Mengunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi langkah 1 dengan simpul w.
4. Ketika telah mencapai suatu simpul u dan tidak ada lagi simpul yang bertetangga dengannya, maka pencarian dirunut balik ke simpul terakhir yang dikunjungi sebelumnya.
5. Pencarian berhenti ketika tidak ada simpul lagi yang belum dikunjungi.



Gambar 3. Pencarian secara DFS

Penelusuran secara DFS pada graf di gambar 3 dimuali dair simpuul A. Anak pertama dari simpul A yaitu B ditelusuri. Dari B, E ditelusuri dan dari E , I ditelusuri. Simpul I tidak memiliki anak simpul lagi, maka dilakukan *backtracking* ke simpul E. E juga tidak memiliki simpul anak, maka balik lagi ke B. B

masih memiliki simpul F maka F ditelusuri. Simpul F adalah daun maka kembali ke B dan dengan alasan yang sama kembali ke A. Selanjutnya C ditelusuri dan D ditelusuri. Dari D maka akan ke G lalu J dan K. Setelah itu kembali lagi ke G dan menelusuri H. Secara keseluruhan maka penelusuran secara DFS akan menjadi A-B-E-I-F-C-D-G-J-K-H.

DFS dapat diimplementasikan dengan dua cara yaitu secara rekursif dan secara iteratif. Misal terdapat graf G, simpul akar v dan tabel Boolean yang bernama dikunjungi berukuran sebesar jumlah simpul. Pseudocode DFS secara rekursif ditunjukkan sebagai berikut.

```
Prosedur DFS (G,v)
 Dikunjungi[v]←true
 For setiap simpul w yang bertetanga dengan simpul v
     If not Dikunjungi[w]
           DFS(w)
     end if
 end for
```

Implementasi DFS secara iterasi memanfaatkan struktur data stack. Pseudocode DFS secara iterasi ditunjukkan sebagai berikut.

```
Prosedur DFS-iteratif(G,v)
   Membuat Stack S
   S.Push(v)
   While S is not empty
       V← S.Pop()
       If  not Dikunjungi [v]
            Dikunnjungi[v]←true
            For semua simpul w yang bertetangga
            dengan v
                 S.Push(w)
            end for
       end if
   end while
```

Kelebihan DFS adalah memakai memori yang cukup kecil karena hanya simpul yang aktif saja yang disimpan dan jika solusi berada pada level yang dalam dan paling kiri maka DFS akan menemukannya dengan cepat. Kelemahan DFS adalah jika pohon yang dibangkitkan memiliki level yang dalam (tak tetrhingga) dimungkinkan tidak ditemukan solusi. Selain itu, jika terdapat lebih dari satu solusi pada level yang berbeda, DFS tidak menjamin menemukan solusi yang optimal.

C. Greedy

Algoritma greedy merupakan salah satu algoritma yang cukup popular. Algoritma ini digunakan untuk memecahkan persoalan optimasi. Persoalan optimasi tidak hanya mencari solusi saja, namun juga mencari solusi terbaik. Dalam mencari solusi, greedy memakai dua buah macam persoalan optimasi, yaitu maksimasi dan minimasi.

Secara harafiah, greedy berarti rakus, tamak atau loba. Sesuai artinya, prinsip greedy adalah *take what you can get now*. Dalam kehidupan sehari-hari, algoritma greedy dapat digunakan dalam masalah seperti memilih jalur terpendek dari satu kota ke kota lain atau memilih beberapa jenis inventasi.

Algoritma greedy membentuk solusi langkah per langkah. Pada setiap langkah terdapat banyak pilihan yang perlu dieksplorasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan terbaik dalam menentukan pilihan tanpa memperhatikan kosekuensi ke depan. Langkah yang telah diambil tidak bisa diubah lagi pada langkah selanjutnya. Pada setiap langkah, kita memperoleh optimum lokal. Dengan memilih optimum lokal pada setiap langkah, kita berharap optimum lokal akan menjadi optimum global.

Namun terkadang, solusi optimum global yang didapat bukan merupakan solusi terbaik, tetapi sub-optimum atau pseudo-optimum. Ini disebabkan algoritma greedy tidak mencari semua alternatif solusi yang ada. Namun, algoritma ini tetap menjadi pilihan utama untuk permasalahan sederhana karena metode ini paling cepat dibandingkan dengan metode lainnya dan dapat memberikan solusi hampiran yang masih merupakan solusi yang layak.

Algoritma greedy disusun oleh elemen-elemen berikut.

1. Himpunan kandidat: berisi elemen-elemen pembentuk solusi
2. Himpunan solusi: berisi kandidat-kandidat yang terpilih menjadi solusi.
3. Fungsi seleksi (*selection function*): fungsi yang memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang telah dipilih pada suatu langkah tidak dipertimbangkan pada langkah berikutnya.
4. Fungsi kelayakan (*feasible*): fungsi yang memeriksa apakah kandidat yang telah dipilih memberikan solusi yang layak, yaitu kandidat tersebut dengan himpunan solusi yang telah ada tidak melanggar kendala (*constraints*) yang ada. Kandidat yang layak dimasukkan kedalam solusi sedangkan kandidat yang tidak layak dibuang.
5. Fungsi objektif: fungsi yang memaksimumkan atau meminimumkan nilai solusi.

Skema umum algoritma greedy ditunjukkan seperti di bawah ini.

```
Function Greedy (input C : himpunan kandidat) ⟶ himpunan_kandidat
{ Mengembalikan solusi dari persoalan optimasi dengan
```

```
greedy
  Masukan : himpunan kandidat C
  Keluaran : himpunan solusi yang bertipe
himpunan_kandidat }
Deklarasi
    x : kandidat
    S : himpunan_kandidat
Algoritma
    S ⟵ {} {inisialisasi S dengan kosong}
    while (not SOLUSI(S)  and  (C ≠ {}) do
        x ⟵ SELEKSI(C) {pilih sebuah kandidat dari C}
        C ⟵ C – {x} {elemen himpunan kandidat
                        berkurang 1}
        if LAYAK(S ∪ {x}) then
            S ⟵ S ∪ {x}
        endif
    endwhile
   if SOLUSI(S) then
          return S
  else
          write('tidak ada solusi')
  endif
```

III. PENERAPAN BFS, DFS DAN GREEDY PADA CROSSING BRIDGE

A. Penentuan State dan Pergerakan

Pada permainan crossing bridge terdapat beberapa aturan yang berlaku:

1. Empat orang tersebut harus berpindah dari satu sisi jembatan ke sisi lainnya untuk menyelesaikan permainan.
2. Dalam setiap kali penyebrangan, dibutuhkan satu lampu untuk penerangan.
3. Jumlah orang yang diperbolehkan dalam setiap kali melakukan penyebrangan adalah minimal 1 orang dan maksimal 2 orang.
4. Ketika dua orang menyebrang bersamaan, maka waktu yang diperlukan untuk meyebrang adalah waktu penyebrangan terlama dari antara kedua orang tersebut.

Untuk memudahkan, orang yang menyebrang dalam waktu 1 detik akan disebut sebagai A, orang yang menyebrang dalam waktu 2 detik sebagai B, orang yang menyebrang dalam waktu 7 detik sebagai C dan orang yang menyebrang dalam waktu 10 detik sebagai D.

Dari aturan yang telah disebutkan, maka akan dibentuk pohon dinamis dengan ketentuan state seperti di bawah ini:

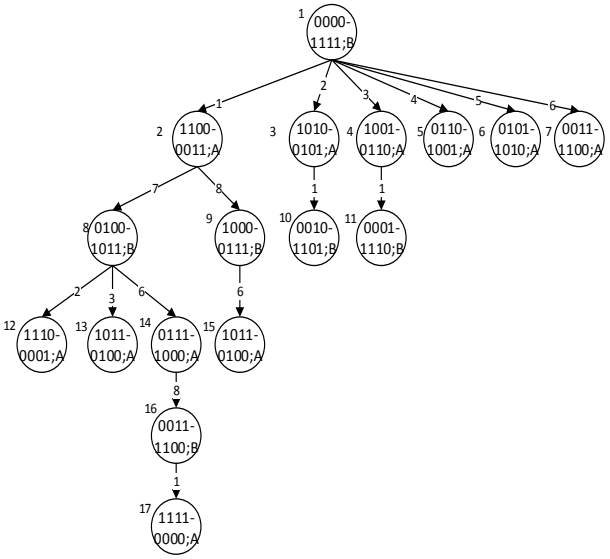
1. Ada tidaknya A di sebelah kiri jembatan
2. Ada tidaknya B di sebelah kiri jembatan
3. Ada tidaknya C di sebelah kiri jembatan
4. Ada tidaknya D di sebelah kiri jembatan
5. Ada tidaknya A di sebelah kanan jembatan
6. Ada tidaknya B di sebelah kanan jembatan
7. Ada tidaknya C di sebelah kanan jembatan
8. Ada tidaknya D di sebelah kanan jembatan
9. Posisi Lampu(A=kiri, B=kanan)

Dari aturan state tersebut maka state awal permainan akan menjadi 0000-1111;B yaitu tidak ada seorang pun di sebelah kiri jembatan dan keempat orang tersebut berada di sebelah kanan jembatan serta posisi lampu berada di sebelah kanan. Pada setiap state terdapat pergerakan/move yang berlaku yaitu:

1 = A&B menyebrang
2 = A&C menyebrang
3 = A&D menyebrang
4 = B&C menyebrang
5 = B&D menyebrang
6 = C&D menyebrang
7 = A menyebrang
8 = B menyebrang
9 = C menyebrang
10 = D menyebrang

B. Penerapan BFS

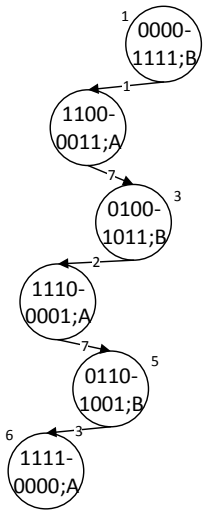
Pembentukan pohon dinamis BFS mengikuti algoritma BFS yang telah disebutkan pada bab IIA. Dalam setiap state dalam pohon ini adalah unik. Jika suatu state pernah dibangkitkan sebelumnya maka state tersebut tidak akan dibangkitkan kembali. Arah pergerakan yag dilakukan mengikuti prioritas yang telah didefinisikan di bab IIIA. Hasil pembangkitan state secara BFS ditunjukkan seperti di bawah ini.



Gambar 4. BFS pada Crossing Bridge

C. Penerapan DFS

Pembangkitan pohon secara DFS mengikuti algoritma pada bab IIC. Sama seperti BFS, state di dalam pohon dinamis ini unik. Penelusuran secara DFS terus menggali kedalam sampai ditemukan solusi. Penelusuran solusi secara DFS diperlihatkan seperti di bawah ini.



Gambar 5. DFS pada Crossing Bridge

D. Penerapan Greedy

Pencarian solusi dengan algoritma greedy, maka elemen-elemennya adalah sebagai berikut.

1. Himpunan kandidat: Himpunan orang-orang yang akan menyebrang
2. Himpunan solusi: pergerakan orang yang dipilih menghasilkan waktu penyebrangan yang minimum
3. Fungsi seleksi (*selection function*): jika berada di sebelah kanan, memilih 2 orang dengan waktu menyebrang tersingkat sedangkan jika berada di sebelah kiri jembatan memilih 1 orang dengan waktu penyebrangan terpendek.
4. Fungsi kelayakan (*feasible*): memeriksa apakah kondisi pernah dicapai oleh state sebelumnya
5. Fungsi objektif: waktu yang dibutuhkan untuk menyebrang minimum.

Hasil algoritma greedy tersebut menghasilkan pergerakan:

1. 0000-1111B
2. 1100-0011A
3. 0100-1011B
4. 1110-0001A
5. 0110-1001B
6. 1111-0000A

IV. Analisa Perbandingan BFS, DFS dan Greedy

Dari hasil yang didapat, penelusuran solusi secara BFS menghasilkan total 17 simpul. Simpul 5, 6 dan 7 tidak diekspans lagi dikarenakan state yang akan terbentuk telah didefinisikan sebelumnya. Begitu pula pada state 10, 11, 12, 13 dan 15. Pencarian secara BFS baru menemukan solusi pada tingkat 5 yaitu pada simpul 17. Runtutan alur hingga menuju solusi adalah sebagai berikut.

1. Menyebrangkan A dan B ke kiri
2. Menyebrangkan A ke kanan kembali
3. Menyebrangkan C dan D ke kiri
4. Menyebrangkan B ke kiri
5. Menyebrangkan A dan B ke kiri.

Penelusuran secara BFS memakan waktu sebesar 17 detik untuk menyebrangkan keempat orang tersebut ke sisi jembatan lainnya.

Penelusuran dengan DFS menghasilkan 6 simpul. Sama seperti BFS, DFS juga mencapai tingkat 5. Dalam membentuk solusi secara DFS, penelusuran terus masuk kedalam tanpa melakukan *backtracking*. Alur menuju solusi yang dihasilkan oleh DFS adalah

1. Menyebrangkan A dan B ke kiri
2. Menyebrangkan A ke kanan kembali
3. Menyebrangkan A dan C ke kiri
4. Menyebrangkan A ke kiri
5. Menyebrangkan A dan D ke kiri.

Total waktu yang dihasilkan oleh solusi dengan pencarian secara DFS adalah 21 detik.

Pencarian alur solusi dengan greedy juga menghasilkan 5 tahapan. Penelusuran dengan greedy tidak mengiijinkan *backtracking* sama sekali. Tahapan yang dihasilkan adalah sebagai berikut.

1. Menyebrangkan A dan B ke kiri
2. Menyebrangkan A ke kanan kembali
3. Menyebrangkan A dan C ke kiri
4. Menyebrangkan A ke kiri
5. Menyebrangkan A dan D ke kiri.

Waktu yang dihasilkan oleh algoritma greedy adalah 21 detik.

Dari ketiga algoritma tersebut, dapat dilihat penelusuran dengan BFS melahirkan lebih banyak simpul dibanding dengan DFS. Namun tingkat kedalaman solusi yang dihasilkan oleh kedua algoritma tersebut sama yaitu tingkat 5. Penelusuran dengan algoritma greedy juga memerlukan 5 tahap sama seperti dengan DFS. Dilihat dari alur yang dihasilkan, algoritma greedy dan DFS menghasilkan alur solusi yang sama sedangkan pada BFS alur solusi terlihat berbeda.

Jika dibandingkan dengan waktu yang dihasilkan, pencarian solusi dengan BFS menghasilkan waktu yang paling singkat yaitu 17 detik sedangkan untuk DFS dan greedy memberikan total waktu 21 detik. Ini menunjukkan pencarian solusi untuk *crossing bridge* dengan BFS lebih baik dalam mencari waktu total yang dihasilkan walaupun simpul yang dihasilkan jauh lebih banyak.

Algoritma greedy seharusnya memberikan solusi yang cukup optimum. Namun, dalam kasus ini algoritma greedy lebih kurang optimum dibanding dengan BFS. Ini disebabkan oleh pemilihan fungsi seleksi yang kurang cukup baik untuk kasus *crossing bridge* sehingga solusi yang dihasilkan juga kurang baik. Selain itu, algoritma BFS menghasilkan alur solusi yang cukup baik dikarenakan prioritas pergerakan yang dipilih sehingga total *move*/pergerakan menjadi sedikit.

Dari ketiga algoritma yang diterapkan untuk mencari solusi *crossing bridge*, algoritma BFS memberikan hasil yang paling optimum dengan waktu 17 detik.

V. KESIMPULAN

Strategi algoritma BFS, DFS dan greedy memiliki banyak penerapan, salah satunya pada permainan logika yaitu *crossing bridge*. Ketiga algoritma itu digunakan untuk mencari alur solusi yang paling optimum. Dalam kasus ini, algoritma BFS yang memberikan solusi yang paling optimum yaitu 17 detik. Hal ini dikarenakan penentuan prioritas pergerakan yang cukup baik.

V. UCAPAN TERIMA KASIH

Pada kesempatan ini, saya ingin mengucapkan terima kasih kepada Tuhan yang Maha Esa karena atas rahmatNyalah makalah ini dapat selesai. Saya juga mengucapkan terima kasih kepada Ibu Masayu Leylia Khodra dan Bapak Rinaldi Munir selaku dosen strategi algoritma. Terima kasih saya ucapkan kepada semua pihak yang terlibat dalam pengerjaan makalah ini.

REFERENCES

[1] Munir, Rinaldi. *Diktat Kuliah IF 2211 Strategi Algoritma*. 2009. Bandung:Institut Teknologi Bandung

[2] http://www.metode-algoritma.com/2013/02/contoh-program-algoritma-greedy.html diakses 14 May 2014

[3] http://www.ics.uci.edu/~eppstein/161/960215.html diakses 14 May 2014

[4] Introduction to Algorithm(Cormen, Leiserson, and Rivest) 1990, Chapter 17 “Greedy Algorithm” p. 329.

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, 16 Mei 2014

Susanti Gojali-13512057