# ANALISIS PENERAPAN ALGORITMA *GREEDY* (DILENGKAPI DENGAN *BACKTRACKING)* DALAM PENCARIAN SOLUSI PERMAINAN "CROSS THE BRIDGE"

**Arief Latu Suseno**

Program Studi Teknik Informatika,
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
arif_latu@yahoo.com

## ABSTRAK

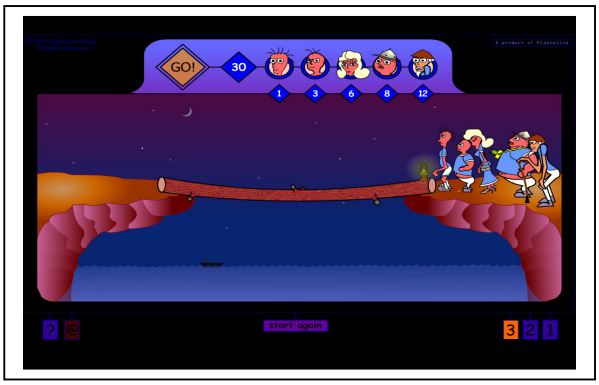
**Program *game*(permainan) komputer adalah salah satu implementasi dari bidang informatika dan teknologi. Perkembangan *game* pada masa kini sudah sangat pesat, terbukti dengan banyaknya *game* yang telah beredar di masyarakat. Dengan jenis yang bermacam-macam dan tampilan yang menarik, *game* komputer termasuk *software* yang diminati oleh berbagai kalangan. *Game* komputer juga dapat menjadi salah satu sarana *refreshing* yang menyenangkan terutama bagi orang yang telah terbiasa menggunakan komputer. Pada makalah ini penulis mencoba membahas sebuah *game* logika, yaitu *game "Cross the Bridge"* dan menganalisanya dengan menerapkan algoritma *greedy* (dilengkapi dengan algoritma *backtracking)* sebagai salah satu cara penyelesaian persoalannya. Penulis berasumsi sendiri bahwa nama dari game ini adalah "*Cross the Bridge*" karena penulis tidak mengetahui nama pasti dari game ini. Penggunaan 2 algoritma dimaksudkan agar solusi yang dicapai benar-benar sesuai harapan, tepat sasaran, dan juga efisien.**

**Kata kunci:** Backtracking, Cross the Bridge, Greedy

## 1. PENDAHULUAN

*Game* "*Cross the Bridge*" merupakan *game* sederhana berbasiskan logika yang bertujuan menyeberangkan sebuah keluarga dari suatu tempat (gambar 1) yang terdiri dari 5 orang ke sebuah tempat (gambar 2) melewati suatu jembatan dalam waktu yang telah ditentukan. Jika waktu yang dihabiskan sudah melewati batas dalam proses penentuan siapa yang harus menyeberang terlebih dahulu, maka akan dilakukan *backtrack* sampai waktu yang dihabiskan tidak melewati batas yang ditentukan.

Untuk mempermudah pembahasan, penulis menggunakan isitilah **ArAwal** sebagai tempat awal sebelum penyeberangan, yaitu daerah sebelah kanan di gambar 1. Sedangkan untuk daerah tujuan, yaitu daerah sebelah kiri di gambar 1, penulis menggunakan istilah **ArAkhir**.




**Gambar 1. Awal Permainan**


**Gambar 2. Akhir Permainan**

## 2. PEMBAHASAN

### 2.1 Persoalan

Game "Cross the Bridge" menceritakan sebuah keluarga yang terdiri dari **5 orang** akan menyeberangi sebuah jembatan, dengan masing-masing orang membutuhkan waktu **1 detik**, **3 detik**, **6 detik**, **8 detik**, dan **12 detik** untuk menyeberang. Jembatan tersebut hanya bisa dilalui oleh **maksimal 2 orang**, dan harus menggunakan lampu yang akan dibawa oleh 1 dari 2 orang yang menyeberang (siapa yang membawa tidak dipermasalhkan). Namun lampu yang tersedia hanya 1 buah, itupun hanya bertahan selama **30 detik** (ini adalah waktu maksimal). Jadi setiap dua orang yang menyeberang, **1 orang harus kembali** untuk membawa lampu. Tugas dari pemain sekarang adalah memilih orang-orang yang harus menyeberang terlebih dahulu agar mereka semua bisa sampai ke seberang sebelum lampu yang mereka bawa mati, atau dengan kata lain dalam waktu **kurang dari atau sama dengan 30 detik.**

### 2.1 Model Penelitian

Pada makalah ini, penulis menggunakan algoritma *greedy* yang dilengkapi dengan *backtrack*. Algoritma *backtrack* (runut balik) yang merupakan perbaikan dari algoritma *brute-force* merupakan salah satu metode pemecahan masalah yang termasuk dalam strategi yang berbasis pencarian pada ruang status. Algoritma *backtrack* bekerja secara rekursif atau iteratif dan melakukan pencarian solusi persoalan secara sistematis pada semua kemungkinan solusi yang ada. Oleh karena algoritma ini berbasis pada algoritma *Depth-First Search* (DFS), maka pencarian solusi dilakukan dengan menelusuri suatu struktur berbentuk pohon berakar secara *preorder*. Proses ini dicirikan dengan ekspansi simpul terdalam lebih dahulu sampai tidak ditemukan lagi suksesor dari suatu simpul.

Sedangkan prinsip dari algoritma greedy adalah:

1. mengambil pilihan yang terbaik (dalam makalah ini adalah selisih waktu yang terkecil) yang dapat diperoleh saat itu,
2. berharap bahwa dengan memilih solusi optimum lokal, pada setiap langkah akan mencapai solusi optimum global. Algoritma *greedy* mengasumsikan bahwa optimum lokal merupakan bagian dari optimum global.

Dalam strategi ini, kita berharap optimum global merupakan solusi optimum dari persoalan. Namun, adakalanya optimum global belum tentu merupakan solusi optimum (terbaik), tetapi dapat merupakan solusi sub-optimum atau pseudo-optimum. Hal ini dapat dijelaskan karena strategi *greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada.. Karena strategi *Greedy* tidak selalu berhasil memberikan solusi yang benar-benar optimum, penulis memadukan *greedy* dan *backtrack* agar proses bisa beroperasi secara menyeluruh tetapi tetap efisien sehingga hasil yang dicapai merupakan solusi yang benar- benar diharapkan.

Algoritma greedy mempunyai elemen-elemen:

1. Himpunan kandidat, *ArAwal*.
2. Himpunan solusi, *ArAkhir*
3. Fungsi seleksi (*selection function*)
4. Fungsi kelayakan (*feasible*)
5. Fungsi obyektif

Pada masalah pencarian waktu minimum:

- *Himpunan kandidat*: himpunan waktu yang merepresentasikan nilai 1, 3, 6, 8, 12, paling sedikit mengandung satu waktu.
- *Himpunan solusi*: total nilai waktu yang dipilih kurang dari atau sama dengan jumlah waktu maksimum yang ditetapkan.
- *Fungsi seleksi*: pilihlah dua waktu yang mempunyai selisih terkecil dari himpunan kandidat yang tersisa, yaitu dengan menggunakan fungsi **GreedyBySelisihTerkecil**
- *Fungsi layak*: memeriksa apakah nilai total dari himpunan waktu yang dipilih tidak melebihi jumlah waktu yang ditetapkan.
- *Fungsi obyektif*: jumlah waktu yang digunakan minimum.

### 2.3 Hasil Penelitian

*Game* ini dimulai dengan menentukan 2 orang pertama yang harus menyeberangi jembatan. Disinilah algoritma *greedy* berperan, yaitu memilih dua orang yang memiliki selisih waktu terkecil dengan harapan waktu yang dihabiskan menjadi lebih efisien. Jika di tahap selanjutnya ternyata tidak ditemukan solusi, maka proses akan mundur hingga tahap ini lalu menentukan 2 orang yang lain untuk menyeberangi jembatan.

*Greedy* juga digunakan untuk memilih satu orang di ArAkhir untuk menyeberang kembali ke ArAwal. *Greedy* yang digunakan adalah **GreedyByWaktuTercepat** yang penulis implementasikan menjadi fungsi **min**.

Algoritma *backtrack* digunakan ketika proses sampai pada solusi yang tidak memungkinkan, yaitu ketika sisa waktu yang ada lebih kecil dari atau sama dengan waktu terlama yang ada di dalam ArAwal.

**maxwaktu-waktu <= max(ArAwal)**

Ketika proses menemukan kondisi seperti yang disebutkan di atas, maka proses akan memulai *backtracking* (mematikan simpul-simpul yang tidak mengarah ke solusi) sampai pada pemilihan GreedyBySelisihTerkecil berikutnya. Begitu seterusnya sampai proses menemukan solusi. Ada dua pendekatan

*backtracking* untuk masalah ini, yaitu secara iteratif dan rekursif. Dalam hal ini, penulis menggunakan metode iteratif.

Berikut adalah algoritma yang dibuat penulis untuk menyelesaikan *game "Cross the Bridge"*:

```
function solution(input ArAwal:
himpunan_kandidat)→ himpunan_kandidat
{ Mengembalikan solusi dari persoalan
optimasi dengan algoritma greedy
Masukan: himpunan kandidat ArAwal
Keluaran: himpunan solusi yang bertipe
himpunan_kandidat
}
Deklarasi
x : kandidat { dua elemen }
y : kandidat { satu elemen }
ArTemp  : himpunan_kandidat
ArAkhir : himpunan_kandidat
Maxwaktu : jumlah semua elemen ArAwal

Algoritma:
ArAkhir → {} { inisialisasi ArAkhir dengan
kosong }
while (not solusi) do
    ArTemp → ArAwal
    tempwaktu  → waktu
    while (maxwaktu - waktu) > max(ArAwal) do
        x → ArAwal.GreedyBySelisihTerkecil[i]
        { pilih 2 buah kandidat dari ArAwal}
        waktu → waktu + max{x}
        ArAwal → ArAwal - {x}
        {elemen himpunan kandidat berkurang 2}
        ArAkhir → ArAkhir ∪ {x}
        if (nbElmt{ArAkhir}  ≠ nbElmt{ArTemp})
then
           y → min{ArAkhir}
           waktu → waktu + max{y}
           ArAkhir → ArAkhir - {y}
           ArAwal → ArAwal ∪ {y}
        else
           solusi = true
        endif
    endwhile
    { solusi belum ditemukan }
    i++
    {untuk memilih Greedy selisih berikutnya}
    ArAwal → ArTemp
    waktu  → tempwaktu
endwhile
{solusi = true, yaitu ArAwal = {} }

If solusi then
    return ArAkhir
else
    write('tidak ada solusi')
endif
```

Ada beberapa fungsi primitif yang penulis pergunakan namun tidak didefinisikan, dengan asumsi fungsi-fungsi tersebut sudah sangat sering dijelaskan dan dipakai di kalangan pemrogram. Fungsi-fungsi tersebut adalah:

1. nbElmt(Input: Array)
   Mengembalikan banyaknya elemen dalam array
2. GreedyBySelisihTerkecil
   Mengembalikan dua elemen dari sebuah array yang mempunyai selisih waktu paling sedikit.
3. max(Input: Array)
   Mengembalikan nilai waktu paling maksimum dari suatu array. Fungsi max adalah hasil implementasi dari **GreedyByWaktuTerlama.**
4. min(Input: Array)
   Mengembalikan nilai waktu paling minimum dari suatu array. Fungsi min adalah hasil implementasi dari **GreedyByWaktuTercepat.**

Penjelasan Algoritma:

1. Kita tentukan dua orang yang memiliki selisih waktu terrkecil dengan menggunakan fungsi **GreedyBySelisihTerkecil**.
2. Dua orang yang terpilih kita pindahkan ke ArAkhir.
3. Kita tentukan 1 orang untuk kembali ke ArAwal dengan menggunakan fungsi **min**
4. Satu orang tersebut kita pindahkan ke ArAwal
5. Ulangi langkah 1 sampai menemukan solusi

Untuk lebih jelasnya, penulis mencontohkannya dalam tabel 1.

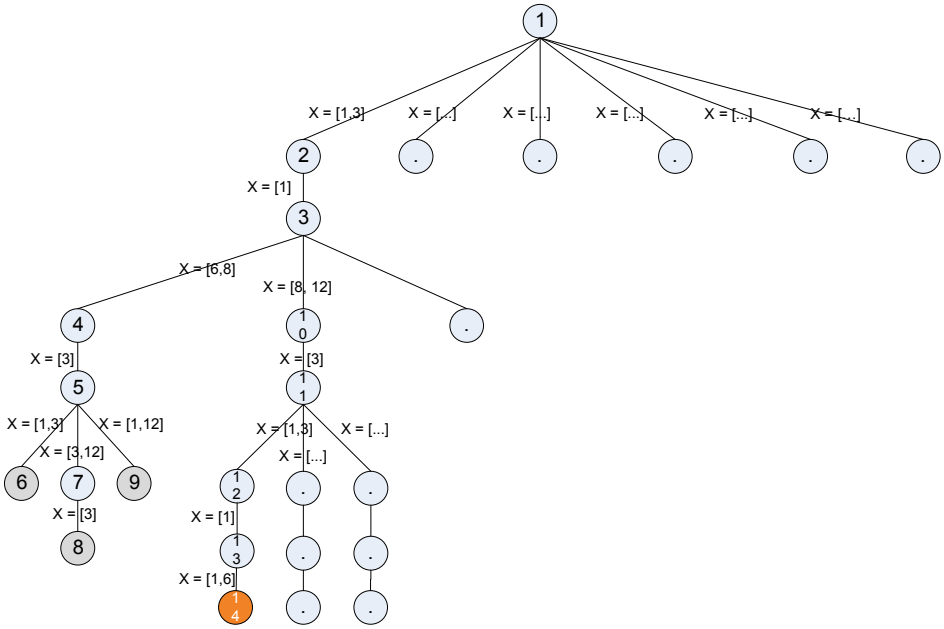
**Tabel 1 Contoh Implementasi Algoritma**

| ArAwal | Pindah | ArAkhir | Waku |
|---|---|---|---|
| 1, 3, 6, 8, 12 | | | 0 |
| 6, 8, 12 | 1, 3 | 1, 3 | 3 |
| 1, 6, 8, 12 | 1 | 3 | 4 |
| 1, 12 | 6, 8 | 3, 6, 8 | 12 |
| 1, 3, 12 | 3 | 6, 8 | 15 |
| 12 | 1, 3 | 1, 3, 6, 8 | 18 |
| 1, 12 | 1 | 3, 6, 8 | 19 |
| | 1, 12 | 1, 3, 6, 8, 12 | 31 |
| 1, 3, 12 | 3 | 6, 8 | 15 |
| 1 | 3, 12 | 3, 6, 8, 12 | 27 |
| 1,3 | 3 | 6, 8, 12 | 30 |
| | 1, 3 | 1, 3, 6, 8, 12 | 33 |
| 1, 3, 12 | 3 | 6, 8 | 15 |
| 3 | 1, 12 | 1, 6, 8, 12 | 27 |
| 1,3 | 1 | 6, 8, 12 | 28 |
| | 1, 3 | 1, 3, 6, 8, 12 | 31 |
| 1, 6, 8, 12 | 1 | 3 | 4 |
| 1, 6 | 8, 12 | 3, 8, 12 | 16 |
| 1, 3, 6 | 3 | 8, 12 | 19 |
| 6 | 1, 3 | 1, 3, 8, 12 | 22 |
| 1, 6 | 1 | 3, 8, 12 | 23 |
| | 1, 6 | 1, 3, 6, 8, 12 | 29 |

Keterangan:

| | |
|---|---|
| | Tidak akan diekseskusi<br>→ solusi tidak memungkinkan |
| | Backtracking |
| | Fungsi **GreedyBySelisihTerkecil** |
| | Fungsi **min** |
| | Himpunan Solusi |

Ilustrasi dengan diagram pohon sampai menemukan solusi:




## 3. KESIMPULAN

Penggunaan algoritma Greedy dalam penyelesaian game ini tidak selalu berhasil memberikan solusi yang benar-benar optimum, namun pasti memberikan solusi yang mendekati (*approximation*) nilai optimum. Namun karena *game* ini membutuhkan solusi yang benar-benar optimum, maka dibutuhkan penambahan algoritma *backtrack* (runut balik).

Algoritma *greedy* bekerja dengan membentuk solusi langkah per langkah (*step by step*) dan pada setiap langkahnya, terdapat banyak pilihan yang perlu dieksplorasi. Di sinilah *greedy* mungkin membuat keputusan yang keliru dalam menentukan pilihan. Hal ini terjadi karena pengambilan keputusan pada setiap langkah greedy tidak pernah mempertimbangkan lebih jauh apakah pilihan tersebut pada langkah-langkah selanjutnya merupakan pilihan yang tepat.

Penggunaan *backtrack* dilakukan jika solusi yang dihasilkan tidak benar-benar optimum karena pilihan *greedy* yang keliru tersebut.

Dengan kombinasi 2 algoritma ini, penulis berkeyakinan bahwa solusi yang dihasilkan pasti solusi yang benar-benar optimum karena permasalahan pemilihan yang mungkin keliru yang dilakukan *greedy* sudah di atasi dengan adanya *backtrack*.

**Tambahan:**

Game ini bisa juga diselesaikan dengan program dinamis.

## 4. REFERENSI


[1] Munir, Rinaldi, "Strategi Algoritmik", Departemen Teknik Informatika ITB, Bandung, 2006.