PERBANDINGAN APLIKASI ALGORITMA *BRUTE-FORCE* DAN KOMBINASI ALGORITMA *BREADTH FIRST SEARCH* DAN *GREEDY* DALAM PENCARIAN SOLUSI PERMAINAN "TREASURE HUNT"

Adi Purwanto Sujarwadi (13506010)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
e-mail: if16010@students.if.itb.ac.id

ABSTRAK

Permainan merupakan hal yang digemari oleh banyak orang untuk mengisi waktu luang. Seiring berkembangnya zaman, maka jenis permainan semakin beragam, mulai dari yang sederhana, hingga yang rumit.

Salah satu jenis permainan ialah permainan "Treasure Hunt". Permainan ini mengharuskan pemain menebak lokasi harta karun yang tersembunyi dengan jumlah tebakan sesedikit mungkin.

Untuk meyelesaikan suatu persoalan, dapat digunakan berbagai macam pendekatan. Tujuan utama pendekatan-pendekatan ini ialah untuk mendapatkan solusi yang paling mangkus untuk sebuah persoalan.

Dalam makalah ini, persoalan permainan "Treasure Hunt" dicoba diselesaikan menggunakan dua buah pendekatan algoritma yang telah dipelajari pada mata kuliah IF2251-Strategi Algoritmik, yaitu algoritma *Brute-Force* dan kombinasi algoritma *Breadth First Search* (BFS) dan algoritma *Greedy*

Tujuan dari penggunaan dua algoritma ini ialah untuk menemukan algoritma mana yang lebih mangkus untuk persoalan permainan "Treasure Hunt"

Kata kunci: Treasure Hunt, *Breadth First Search*, BFS, Greedy, *Brute-Force*, pencarian solusi

1. PENDAHULUAN

Semenjak zaman dahulu kala, salah satu bentuk kegiatan yang dilakukan manusia untuk mengisi waktu senggangnya ialah dengan memainkan permainan. Seiring dengan berjalannya waktu, jenis permainan terus bertambah, baik berupa permainan sederhana, hingga permainan digital yang dapat dimainkan pada peralatan elektronik, khususnya komputer.

Salah satu jenis permainan sederhana yang hingga saat ini masih digemari sebagian kalangan ialah permainan "*Treasure Search*". Permainan ini merupakan permainan yang sederhana dan sangat mudah dimainkan, sehingga menjadi pilihan banyak orang saat mengisi waktu senggangnya. Tujuan dari permainan ini ialah menemukan harta yang disembunyikan dalam langkah yang sesedikit mungkin.

Solusi dapat ditemukan dengan berbagai cara, namun dalam makalah ini, akan digunakan algoritma *Breadth First Search (BFS)* yang dikombinasikan dengan algoritma *Greedy* untuk menemukan solusi, dan sebagai pembanding akan digunakan algoritma *Brute-Force*.

2. PERMAINAN *TREASURE SEARCH*

Permainan *Treasure Search* merupakan permainan yang terdiri atas sebuah papan dengan ukuran bebas dan harta karun yang disembunyikan dalam salah satu kotak tersebut.

Pemain dapat menentukan sendiri titk awal tebakan, dan permainan ini akan memberitahukan jarak dari titik tebakan pemain menuju ke harta yang disembunyikan, namun permainan ini tidak akan memberitahu pemain, ke arah mana mereka harus menebak selanjutnya.

Permainan akan berakhir saat pemain telah menemukan harta yang disembunyikan. Tantangan utama dalam permainan ini ialah menemukan harta tersebut dalam jumlah tebakan sesedikit mungkin.

3. ALGORITMA

Dalam makalah ini, ada dua algoritma yang akan dicoba diterapkan, yaitu algoritma *Brute-Force*, dan kombinasi dari algoritma *Breadth First Search (BFS),* dan algoritma *Greedy*.

3.1 Algoritma *Brute-Force*

Algoritma *Brute-Force* merupakan algoritma yang lempang (*straightforward)* dalam menyelesaikan suatu masalah. Algoritma ini didasarkan pada pernyataan masalah dan definisi konsep yang dilibatkan. Algoritma *brute-force* menyelesaikan masalah dengan cara yang sederhana dan jelas.

Algoritma *brute-force* pada dasarnya tidak mangkus untuk menyelesaikan suatu permasalahan. Namun, apabila suatu permasalahan benar-benar mempunyai solusi, dapat dipastikan algoritma *brute-force* akan dapat menemukan solusi tersebut, meskipun dengan waktu yang relatif lebih lama dibandingkan dengan menggunakan algoritma lain yang lebih modern dan mangkus.

Untuk masalah yang sederhana, algoritma *brute-force* seringkali lebih disukai, karena biaya untuk mengoptimalkan masalah dianggap tidak sebanding dengan hasil optimalisasi masalah dengan algoritma lain.

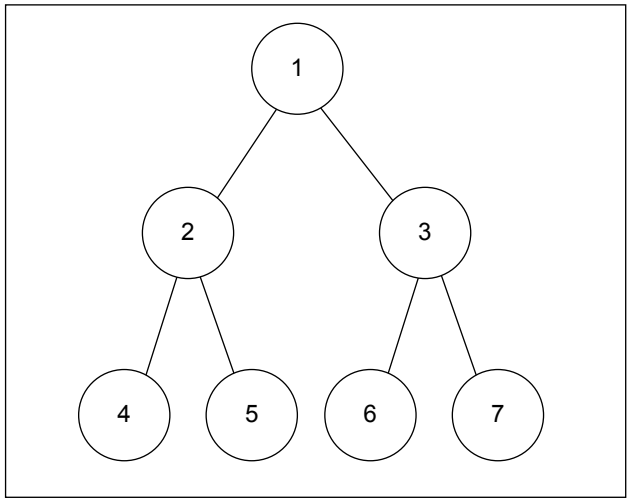
Algoritma *brute-force* juga seringkali digunakan sebagai basis untuk mencari algoritma lain yang lebih mangkus. Algoritma *brute-force* juga seringkali digunakan sebagai pembanding algoritma lain yang lebih mangkus, karena algoritma ini merupakan algoritma yang paling mendasar dalam menyelesaikan suatu persoalan.

Keunggulan utama dari algoritma *brute-force* ialah algoritma ini sangat mudah diimplementasikan dibandingkan algoritma lain yang lebih canggih. Oleh karena itu, terkadang algoritma *brute-force* dapat menjadi algoritma yang mangkus, apabila ditinjau dari segi kesederhanaan implementasinya.

3.2 Algoritma *Breadth First Search* (BFS)

Algoritma BFS (*Breadth First Search)* atau disebut juga Algoritma Pencarian Melebar adalah algoritma yang melakukan pencarian solusi dengan memaparkan terlebih dahulu seluruh pilihan solusi yang ada pada tiap tahap pencarian.

Algoritma BFS ini dapat digambarkan dengan menggunakan sebuah struktur data Pohon. Dengan simpul yang dapat dimisalkan sebagai Orang Tua dan simpul yang lainnya bertindak sebagai anak. Algoritma BFS mencari solusi dengan cara menghidupkan semua anak terlebih dahulu pada tiap tahapan, baru kemudian menghidupkan anak dari simpul sebelumnya yang sekarang akan bertindak sebagai Orang Tua.



Gambar 1. Pembangunan pohon solusi dengan algoritma *Breadth First Search*

§§

Gambar 1 adalah gambar pencarian solusi dengan AlgoritmaBFS, nomor pada tiap simpul menunjukkan urutan pembangkitan simpul tersebut

3.3 Algoritma *Greedy*

Algoritma *Greedy* merupakan algoritma yang membentuk solusi dengan cara langkah per langkah (*step by step*). Pada setiap langkahnya algoritma ini akan mengambil pilihan yang terbaik yang bisa didapatkan saat itu tanpa memikirkan konsekuensi ke depan (prinsip "*take what you can get now!"*. Dengan cara seperti ini, algoritma ini berharap bahwa dengan memilih optimum local pada setiap langkah maka akan berakhir dengan optimum global, meskipun tidak selamanya seperti itu.

Oleh karena algoritma *Greedy* ini selalu berusaha mengambil langkah optimum lokal, maka algoritma ini biasanya digunakan untuk menyelesaikan persoalan optimasi.

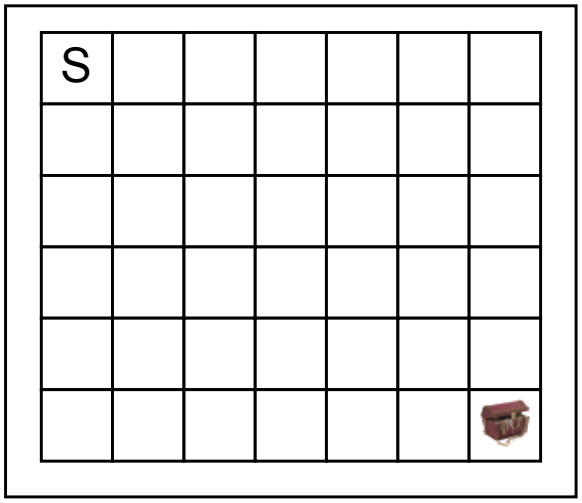
Persoalan optimasi dalam konteks algoritma *Greedy* disusun oleh elemen-elemen sebagai berikut :

a. Himpunan kandidat (C)
   Himpunan ini berisi elemen pembentuk solusi. Pada setiap langkah, satu buah kandidat akan diambil dari himpunannya.
b. Himpunan solusi (S)
   Berisi kandidat terpilih sebagai solusi persoalan, dengan kata lain himpunan solusi adalah bagian dari himpunan kandidat.
c. Fungsi seleksi
   Fungsi ini dinyatakan dengan predikat seleksi, yaitu fungsi yang pada setiap langkah memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu

langkah, tidak pernah dipertimbangkan lagi pada langkah selanjutnya.

d. Fungsi kelayakan
   Fungsi ini dinyatakan dengan predikat layak. Fungsi ini bertujuan memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constratints*) yang ada.
e. Fungsi obyektif
   Fungsi ini bertujuan untuk memaksimumpkan atau meminimumkan nilai solusi.

4. PERMASALAHAN



Gambar 2 : Ilustrasi persoalan

Misalkan kasus pencarian harta karun akan dilakukan pada papan berukuran 6x6, dengan posisi awal berada di kotak [1,1], persoalan ini digambarkan pada Gambar 2, di atas.

Berbeda dengan pencarian jalan terpendek, pada persoalan ini pemain tidak mengetahui letak garis akhir yang mana merupakan letak harta karun yang disembunyikan. Pemain hanya akan mengetahui jarak menuju kotak harta tersebut, tanpa mengetahui arah mana yang harus ditempuh.

5. PEMECAHAN MASALAH

Dalam mencari solusi untuk permainan ini, akan digunakan dua buah metode berbeda. Metode pertama dengan menggunakan algoritma *Brute-Force*, metode kedua ialah dengan menggunakan algoritma BFS yang dipadukan dengan algoritma *Greedy*.

5.1 Algoritma *Brute-Force*

Permainan *Treasure Hunt* ini merupakan permainan dengan tepat satu solusi, sehingga dapat dipastikan algoritma *Brute-Force* dapat digunakan untuk menemukan solusi persoalan.

Berikut akan diberikan *pseudo-code* penerapan algoritma *Brute-Force* untuk menyelesaikan persoalan tersebut.

```
Deklarasi
   JmlKolom, JmlBaris  : integer
   Kolom, Baris        : integer
   Papan               : struct
                           PosX  : integer
                           PosY  : integer
                           IsTreasure : boolean
                         Endstruct
   LapanganPermainan   : array[1..n][1..n] of
                         Papan
   Ditemukan           : boolean
   barisAwal,KolomAwal : integer

Algoritma
   Ditemukan <- False
{inisiasi ditemukan dengan salah}
   Kolom <- KolomAwal
{inisiasi dengan kolom awal}
   Baris <- BarisAwal
{inisiasi dengan baris awal}

   while (not Ditemukan) and
((Kolom<JmlKolom) and (Baris<JmlBaris)) do

   if (Papan[Kolom][Baris].IsTreasure = true)
   then
        Ditemukan <- True
   else
        Kolom<-Kolom+1
        Baris<-Baris+1
   endif
   endwhile

   if (not Ditemukan) then
{bila seluruh kotak dikanan dan bawah kotak
awal telah diperiksa namun belum ditemukan
solusi, periksa kotak lainnya}
   Kolom <- 1
   Baris <- 1
   while (not Ditemukan) and
((Kolom<KolomAwal) and (BarisAwal)) do
   if (Papan[Kolom][Baris].IsTreasure = true)
   then
        Ditemukan <- True
   else
        Kolom<-Kolom+1
        Baris<-Baris+1
   endif
   endwhile
endif
```

Pada potongan algoritma dalam *pseudo-code* di atas, terdapat tipe bentukan yaitu papan. Papan memiliki anggota PosX yang merupakan posisi baris lokasi papan tersebut, PosY yang merupakan posisi kolom lokasi papan

tersebut, dan IsTreasure yang akan bernilai *true* jika papan tersebut mengandung harta karun.

Permainan ini sendiri berjalan pada sebuah *matriks of papan* yang diberi nama Lapangan permainan.

Adapun langkah penyelesaian dengan algoritma *Brute-Force* dengan menggunakan solusi dalam *pseudo-code* di atas dapat dijelasakan sebagai berikut :

a. Periksa seluruh kotak yang memiliki indeks lebih besar dari titik awal.
b. Jika harta karun ditemukan, hentikan pencarian
c. Jika harta karun belum ditemukan, ulangi pencarian hingga solusi ditemukan (ulangi dari langkah a)
d. Jika harta karun belum ditemukan, namun pencarian telah mencapai papan dengan indeks maksimum, ulangi pencarian dari papan dengan indeks minimum.

Kondisi terbaik algoritma ini ialah saat harta karun tepat berada di kotak sebelah kotak awal, pencarian akan ditemukan dalam 1 langkah, sedangkan pada persoalan yang diberikan ini, algoritma ini akan memeriksa seluruh kotak, karena solusi berada di kotak paling akhir. Solusi akan ditemukan dalam 6x6 = 36 Langkah.

5.2 Algoritma BFS dan *Greedy*

Algoritma BFS dan *Greedy* merupakan algoritma yang telah diketahui lebih mangkus daripada *Brute-Force*. Untuk membuktikan hal tersebut, akan digunakan kombinasi dari kedua algoritma tersebut dalam *pseudo-code* berikut ini :

```
function FindMinimum (input C : array of papan)
-> papan
{Mengembalikan solusi paling minimum dari
kemungkinan solusi yang diberikan
 Masukan       : array of papan
 Keluaran      : papan {papan dengan nilai
jarak paling minimum dari masukan}
}

function PeriksaTetangga (input C: papan) ->
array of papan
{Memeriksa seluruh papan yang bertetanggaan
dengan papan utama
 Masukan       : papan
 Keluaran      : array of papan
(array dengan isi seluruh tetangga terdekat
papan yang diperiksa}
}

Deklarasi
   JmlKolom, JmlBaris  : integer
   Kolom, Baris        : integer
   Papan               : struct
                         PosX  : integer
                         PosY  : integer
                         IsTreasure : Boolean
                         Cost  : Integer
                         Endstruct
   LapanganPermainan   : array[1..n][1..n] of
                         Papan
   Ditemukan           : boolean
   barisAwal,KolomAwal : integer
   Parent              : Papan
   Anak                : Array of Papan
```

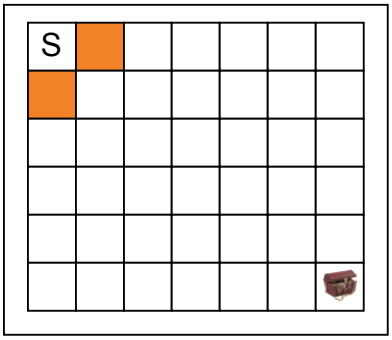
```
Algoritma

Parent <-LapanganPermainan[barisAwal][KolomAwal]
{inisiasi parent dengan kotak awal}
   while (not Ditemukan) do
     Anak <- PeriksaTetangga(Parent)
     Parent<-FindMinimum(Anak)
     if (Parent.IsTreasure = true) then
         Ditemukan <-true
     endif
   endwhile
```

Pada potongan algoritma dalam *pseudo-code* di atas, terdapat tipe bentukan yaitu papan. Papan memiliki anggota PosX yang merupakan posisi baris lokasi papan tersebut, PosY yang merupakan posisi kolom lokasi papan tersebut, dan IsTreasure yang akan bernilai *true* jika papan tersebut mengandung harta karun.

Permainan ini sendiri berjalan pada sebuah *matriks of papan* yang diberi nama Lapangan permainan.

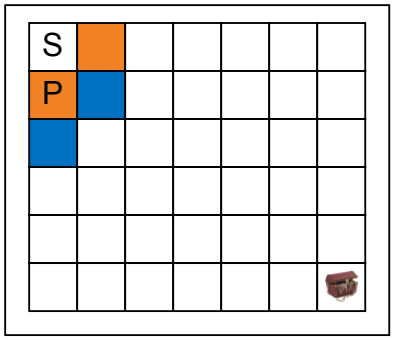
Adapun penjelasan *pseudo-code* di atas adalah sebagai berikut :

a. Kotak awal akan dijadikan simpul orang tua
b. Seluruh kotak yang bertetangga dengan kotak orang tua akan diperiksa jarak menuju letak harta karun. Di sinilah algoritma *Greedy* berperan, memilih simpul selanjutnya yang akan dibangkitkan dengan cara memilih simpul dengan jarak minimum. Jika ada dua atau lebih simpul dengan jarak yang sama, ambil secara sembarang.



Gambar 3: Ilustrasi pemecahan masalah (1)

Gambar di atas menunjukkan cara kerja tahapan (b), dengan simpul S sebagai orang tua dan simpul anak berupa kotak dengan warna jingga.



Gambar 4: Ilustrasi pemecahan masalah (2)

Gambar di atas menunjukkan cara kerja tahapan (b) selanjutnya, dimana kotak dengan nilai minimum akan dijadikan orang tua (ditandai dengan huruf P), selanjutnya yang menjadi simpul anak ialah kotak dengan warna biru. Catatan : sebenarnya kedua kotak memiliki jarak yang sama, dalam hal ini, diambil secara acak kotak P yang lokasinya dibawah kotak awal, pemilihan ini dilakukan secara acak, tidak ada aturan tertentu.

c. Apabila harta karun telah ditemukan, maka hentikan pencarian
d. Apabila belum ditemukan, maka simpul anak dengan jarak minimum akan dijadikan simpul orang tua, kemudian pencarian akan diulangi dari tahap (b)

Kondisi terbaik algoritma ini, sama dengan algoritma *Brute-Force*, yaitu saat harta karun tepat berada di kotak sebelah kotak awal, pencarian akan ditemukan dalam 1 langkah, sedangkan pada persoalan yang diberikan ini, algoritma ini akan lebih mangkus daripada algoritma *Brute-Force*, dan solusi akan ditemukan dalam 12 langkah saja(pembuktian diserahkan kepada pembaca).

6. KESIMPULAN

Dalam menyelesaikan persoalan permainan *Treasure Hunt*, kedua algoritma dapat menemukan solusi permainan dengan baik. Jadi, pada dasarnya apabila suatu persoalan memiliki solusi, kedua algoritma dapat digunakan untuk menyelesaikan persoalan tersebut.

Jika ditinjau dari segi kemangkusan, algoritma *Greedy* yang digabungkan dengan BFS memiliki tingkat kemangkusan yang jauh lebih baik daripada *Brute-Force* untuk menyelesaikan permasalahan ini.

Akan tetapi, jika kita tinjau dari segi kemudahan implementasi, algoritma *Brute-Force* dapat lebih mudah diterjemahkan ke bahasa pemrograman, karena algoritma ini bersifat sederhana, hanya memeriksa seluruh kemungkinan yang ada. Sedangkan jika kita mencoba mengimplementasikan algoritma BFS yang digabung dengan *Greedy*, akan sedikit menemui kesulitan pada tahapan pembuatan fungsi.

Kesimpulan akhir yang dapat diambil adalah, untuk persoalan *Treasure Hunt* dengan area permainan yang kecil, algoritma *Brute-Force* lebih layak diimplementasikan, karena meskipun algoritma BFS dan *Greedy* berjalan jauh lebih cepat, namun perbedaan kecepatan ini tidak akan terlalu terasa, bahkan mungkin akan dapat lebih lambat, karena ada kesulitan pada saat implementasi. Sedangkan jika area permainan sangat besar, waktu yang dikorbankan untuk implementasi algoritma BFS dan *Greedy* akan terasa tidak berarti dibandingkan tingkat kecepatan yang didapatkan.

REFERENSI

[1] Munir, Rinaldi. Strategi Algoritmik. Institut Teknologi Bandung : Bandung 2007.
[2] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm. Tanggal Akses : 16 Mei 2008
[3] http://www.cs.man.ac.uk/~graham/cs2022/greedy/. Tanggal Akses : 17 Mei 2008