PERBANDINGAN ALGORITMA GREEDY DAN BRUTE FORCE DALAM SIMULASI PENCARIAN KOIN

Indra Mukmin
13506082

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB
Jalan Ganeca no.10
Email : if16082@students.if.itb.ac.id

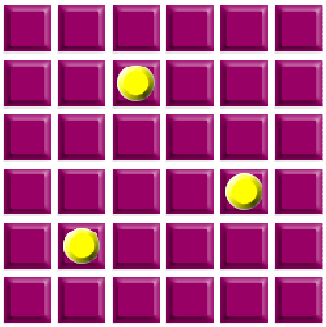
ABSTRAK

Makalah ini membahas tentang perbandingan antara algoritma *greedy* dan *brute force* yang digunakan sebagai solusi masalah dalam simulasi pencarian koin. Simulasi pencarian koin ini direpresentasikan dengan sebuah papan berukuran n x n dengan koin tersebar secara acak di tiap kotak pada papan tersebut. Terdapat sebuah robot yang akan bertugas untuk mengambil koin tersebut satu persatu. Robot hanya bisa bergerak ke kanan dan ke bawah dengan terminal berada pada kotak paling kanan bawah. Algoritma *greedy* yang digunakan adalah *greedy by range* yaitu robot akan melakukan penyisiran area dengan titik awal berada pada kotak robot. Penyisiran dilakukan ke kanan dan ke bawah. Dapat dipastikan bahwa dengan adanya penyisiran, tidak ada koin yang tidak terambil. Berbeda dengan *brute force*, pencarian koin dilakukan dengan menelusuri setiap kotak, demikian seterusnya sehingga setiap kotak berhasil disinggahi. Dengan *brute force*, dapat dipastikan bahwa juga tidak mungkin ada koin yang tidak terambil. Perbedaan keduanya adalah pada jumlah robot yang digunakan untuk mengambil seluruh koin yang tersebar tersebut.

Kata kunci: *greedy*, *greedy by range*, koin, *brute force*

1. PENDAHULUAN

Simulasi permainan pencari koin merupakan sebuah permainan yang dapat direpresentasikan dengan sebuah papan dengan jumlah kotak n x n dan di atas papan tersebut tersebar koin – koin. Terdapat sebuah robot yang bertugas untuk mengambil koin tersebut satu persatu dengan kendala, robot hanya dapat berjalan ke kanan dan atau ke bawah (robot tidak dapat mundur). Robot akan mulai berjalan dari kotak paling kiri atas menuju kotak paling kanan bawah. Selama perjalanan, robot akan berusaha mengambil koin sebanyak – banyaknya yang dapat diperoleh. Ketika mencapai kotak paling kanan bawah, robot akan kembali lagi ke kotak paling kiri atas untuk mengambil koin – koin yang masih tersisa. Begitu seterusnya hingga koin – koin yang tersebar di papan berhasil diambil semuanya.



Gambar 1. Simulasi Pencarian Koin

Inti dari simulasi permainan ini adalah penggunaan jumlah robot seminimal mungkin, yang digunakan untuk mengambil koin tersebut. Oleh karena itulah, tujuan utama dari penulisan makalah ini, selain untuk memenuhi tugas penulisan makalah mata kuliah IF2251 Strategi Algoritmik, juga untuk menganalisis perbandingan antara penggunaan algoritma *brute force* dan *greedy* dalam memecahkan masalah ini.

2. METODE

Metode yang digunakan dalam pembahasan makalah ini adalah deskriptif-argumentatif. Solusi yang digunakan untuk menyelesaikan permasalahan pencarian koin tersebut dapat bermacam – macam, dalam hal ini yaitu algoritma yang digunakan. Seperti yang telah dikemukan di atas, penulis menggunakan algoritma *brute force* dan *greedy* sebagai pilihan solusi.

2.1 Algoritma Brute Force

Algoritma *brute force* adalah algoritma yang memecahkan masalah dengan sangat sederhana, langsung, dan dengan cara yang jelas (*obvious way)* [1]. Penyelesaian permasalahan simulasi pencari jalan dengan menggunakan algoritma *brute force* (sesuai metode yang penulis buat) yaitu dengan mengunjungi setiap kotak pada papan, baik secara perbaris maupun perkolom. Algoritma *brute force* adalah algoritma yang lempang. Pengecekan akan dilakukan mulai dari baris pertama atau kolom pertama. Pengecekan akan berhenti dilakukan ketika jumlah koin yang tersisa pada papan sama dengan nol. Kemungkinan jumlah robot yang digunakan dapat bervariasi, mulai dari satu hingga delapan, dengan asumsi bahwa simulasi ini akan menampilkan minimal satu koin pada papan.

2.1.1 Penyelesaian dengan Algoritma Brute Force

Secara garis besar, *pseudo code* yang digunakan adalah sebagai berikut:

```
Procedure coin_BruteForce (input jumlah_koin :
integer, output jumlah_robot : integer)

Deklarasi
baris, kolom, temp, i : integer

Algoritma:
baris ← 1
jumlah_robot ← 0
while(jumlah_koin <> 0) do
   if(baris <> 1) then
     pindahkan robot ke kotak[baris,1]
   endif
   while(not paling kanan bawah)do
      kolom ← 0
      for i ← kolom+1 to n do
         gerakkan robot ke kotak[baris,kolom]
         if(ada koin) then
            jumlah_koin ← jumlah_koin - 1
         endif
      endfor
      if(kolom = n) then
         for(temp ← baris+1) to n do
            gerakkan robot ke kotak[temp,kolom]
            if(ada koin) then
               jumlah_koin ← jumlah_koin - 1
            endif
         endfor
      endif
   endwhile
   baris ← baris+1
   jumlah_robot ← jumlah_robot + 1
endwhile
```

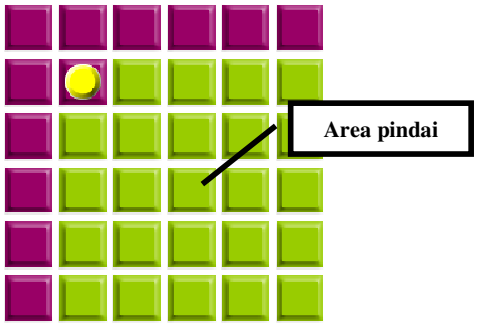
Pada algoritma *brute force* di atas, robot akan berjalan mulai dari baris pertama. Selanjutnya, robot akan berpindah dari kolom satu hingga ke-n. di tiap kotak, robot akan melakukan pengecekan apakah ada koin atau tidak. Apabila ada, koin diambil dan jumlah koin berkurang satu. Apabila robot telah mencapai kolom ke-n, robot akan berpindah ke satu baris lebih bawah. Dilakukan pengecekan apakah koin ada atau tidak. Bila terdapat koin, koin diambil dan jumlah koin berkurang satu. Begitu seterusnya hingga robot berada kotak paling kanan bawah. Selanjutnya, robot akan kembali ke atas ke kotak paling kiri atas. Pertama, robot akan turun satu baris dari baris pengecekan sebelumnya (bila sebelumnya robot berada di baris ke-4, maka sekarang robot melakukan pengecekan mulai dari baris ke-5). Apabila jumlah koin sama dengan nol, simulasi dihentikan.

Jumlah robot yang digunakan pada simulasi ini bervariasi, bergantung pada kondisi tertentu. Jumlah koin sama dengan satu, maka jumlah robot bervariasi dari 1 sampai 8. Hal ini dikarenakan, robot melakukan pencarian mulai dari baris pertama, sehingga untuk kasus satu koin, posisi koin terletak pada baris ke berapa amatlah menentukan. Apabila koin terdapat pada baris pertama, maka jumlah robot yang diperlukan hanya satu. Akan tetapi, apabila koin berada pada baris ke delapan, maka robot ke-8 lah yang nantinya akan mengambil koin tersebut karena robot akan melakukan pengecekan pada 7 baris sebelumnya terlebih dahulu. Apabila program mementingkan koin habis, algoritma ini dapat digunakan karena dapat dipastikan bahwa semua koin pasti terambil. Namun, apabila solusi yang diminta mengharapkan jumlah robot seminimal mungkin, algoritma ini bahkan tidak mangkus untuk kasus hanya terdapat satu koin dan koin tersebut tidak terletak pada baris pertama. Untuk menjawab permasalahan ini, diperlukan suatu algoritma yang lebih mangkus. Salah satu contohnya adalah adalah algoritma *greedy*.

2.2 Algoritma Gredy

Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan yang "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal (local optimum) pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi optimum global (global optimum)[1].

Algoritma *greedy* yang digunakan sebagai solusi pada permasalahan ini algoritma *greedy by range* yaitu program akan melakukan penyisiran area (*scanning area*) ke kanan dan ke bawah dengan posisi robot sebagai titik awal. Selanjutnya, robot akan menuju koin terdekat yang indeks posisinya diperoleh setelah penyisiran. Selanjutnya program kembali akan melakukan penyisiran area dengan posisi robot terakhir sebagai titik awal.



Gambar 2. Area pemindain koin

Dengan adanya pemindaian ini, dapat dipastikan bahwa apabila terdapat satu koin pada papan, maka robot yang diperlukan juga hanya satu buah saja. Maksimal jumlah robot yang diperlukan adalah delapan, bergantung pada keadaannya apakah karena jumlah koin pada papan adalah n x n, atau karena posisi tertentu koin (hal ini dikarenakan kendala gerak robot).

2.2.1 Penyelesaian dengan Algoritma Greedy

Secara umum, *pseudo code* yang digunakan adalah sebagai berikut:

```
procedure greedy_by_range (input C : set_koin,
jumlah_koin : integer, ouput S : set _koin)
{program menerima masukan berupa C himpunan koin dan
jumlah_koin yang nilainya dibangkitkan secara acak}

Deklarasi
gc : robot
a, b, i, j, x, y, s: integer;
jarak, jarakMin: real;

Algoritma
S ← {}
while jumlah_koin <> 0 do {fungsi kelayakan}
 x := 0;        {inisialisasi absis Robot}
 y := 0;        {inisialisasi ordinat Robot}
 a := 0;
 b := 0;
 while (x <> n) and (y <> n) do
  {selama posisi Robot belum mencapai sudut kanan bawah
papan, Robot akan terus mencari koin dengan jarak
terdekat dari posisinya saat ini / fungsi kelayakan}
   jarakMin := 0;
   for i := (x + 1) to n do
  {pengulangan indeks i dengan rentang antara absis
posisi robot saat ini hingga indeks terbesar papan,
yaitu n}
    for j := (y + 1) to n do
  {pengulangan indeks j dengan rentang antara ordinat
posisi robot saat ini hingga indeks terbesar papan,
yaitu n}
     if (ada koin dalam range kanan dan bawah)
then
   {jika koin ditemukan di suatu lokasi, jarak akan
dihitung}
     if jarakMin = 0 then
     {kasus untuk pencarian jarak minimum pertama kali}
       jarakMin := exp((i - x)*ln(2)) + exp((j -
y)*ln(2));
       a := i; {penyimpanan indeks lokasi koin untuk
kemudian digunakan oleh prosedur menjalankan robot
menuju tempat koin}
       b := j; {penyimpanan indeks lokasi koin untuk
kemudian digunakan oleh prosedur menjalankan robot
menuju tempat koin}
     else
       jarak := exp((i - x)*ln(2)) + exp((j -
y)*ln(2));
       if (jarak < jarakMin) then {jarak yang
dihitung untuk kesekian kalinya, dibandingkan dengan
nilai yang tersimpan di variabel jarakMin}
          jarakMin := jarak;
        a := i; {penyimpanan indeks lokasi koin untuk
kemudian digunakan oleh prosedur menjalankan robot
menuju tempat koin}
        b := j; {penyimpanan indeks lokasi koin untuk
kemudian digunakan oleh prosedur menjalankan robot
menuju tempat koin}
      endif
     endif
    endif
   endfor
  endfor
 endwhile
if jarakMin = 0 then {jika setelah proses pencarian,
tidak ditemukan adanya koin dalam cakupan maka proses
pengulangan dihentikan dan robot di-"pulang"-kan}
  break; {penghentian pengulangan}
 else
  (proses pengaktifan Robot menuju koin terdekat)
  C ← C - (gc)
  x := a; {absis robot setelah perpindahan}
  y := b; {ordinat robot setelah perpindahan}
  JumlahKoin := JumlahKoin - 1; {mengurangi jumlah
koin, karena telah terambil}
endif;
S ← S U (gc) {gc = robot/ robot pengambil koin }
endwhile;
```

berdasarkan algoritma *greedy* di atas, komponen – komponen algoritmanya yaitu:

Himpunan kandidat, C

Himpunan kandidat dari program di atas adalah himpunan koin yang berjumlah n x n buah, terletak di masing – masing kotak pada papan n x n tersebut. Yang akan ditampilkan adalah koin sebanyak hasil random nilai n x n.

Himpunan solusi, S

Himpunan solusi yang terbentuk merupakan himpunan robot yang digunakan untuk pengambilan seluruh koin yang ada di papan. Himpunan solusi memiliki minimal satu anggota karena akan fungsi acak koin akan memunculkan minimal satu koin di papan.

Fungsi seleksi

Fungsi seleksi yang digunakan adalah fungsi penghitungan jarak_minimal koin. Fungsi ini akan memilih koin dengan jarak terdekat dan menyimpan

indeks dari koin tersebut yang nantinya akan dijadikan sebagai acuan untuk robot.

Fungsi kelayakan

Fungsi ini, pada algoritma di atas dapat disamakan dengan while …do, akan mengecek apakah robot telah berada pada kotak paling bawah kanan (dituliskan sebagai : while (x <> n) and (y <> n) do …) atau belum, bila ya, maka robot dikembalikan pada kotak paling kiri atas. Jika belum, maka robot akan terus berjalan hingga mencapai kotak paling kanan bawah.

Selain itu, fungsi kelayakan juga muncul pada pengecekan jumlah koin (dituliskan sebagai : while jumlah_koin <> 0 do …). Selama koin belum habis, robot akan terus melakukan pencarian koin. Apabila koin telah habis, barulah proses pencarian selesai.

Fungsi obyektif

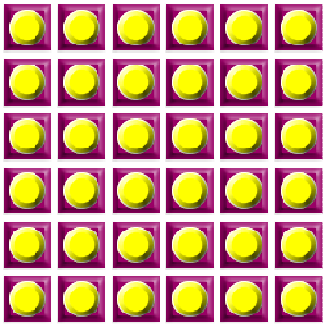
Fungsi ini berfungsi untuk menentukan koin mana yang akan dipilih. Dalam simulasi ini (berdasarkan metode penulis), apabila robot berada pada suatu kotak, dan tepat di sebelah bawah dan kanan robot terdapat koin, Robot akan memilih apakah akan bergerak ke bawah atau ke kanan. Fungsi obyektif ini akan memilih hanya salahsatu koin karena kedua koin memiliki kesempatan yang sama untuk dipilih karena memiliki jarak yang sama terhadap robot. Namun demikian, simulasi ini akan memilih kotak ke kanan sebagai prioritas.

2.3 Sekilas Perbandingan Beberapa Kasus Antara Algoritma Greedy dan Brute Force

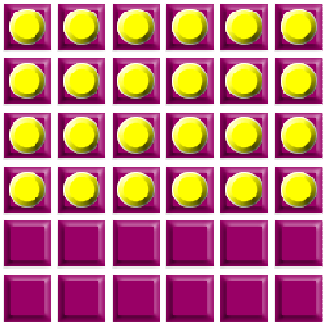
Berdasarkan pembahasan di atas, algoritma greedy jelas lebih mangkus untuk memecahkan permasalahan ini. Tinjau untuk kasus hanya terdapat satu koin saja pada papan, algoritma greedy hanya memerlukan satu robot saja untuk mengambil koin tersebut. Lain halnya pada algoritma *brute force* yang penulis kemukakan, akan terdapat 8 kemungkinan jumlah robot, bergantung pada posisi baris dimana koin itu berada. Apabila koin berada pada baris ke-8, maka jumlah robot yang diperlukan sama dengan 8 karena pengecekan tetap dilakukan mulai dari baris pertama.

Bertolak dari paragraf di atas, algoritma ini juga dapat memberikan hasil yang sama untuk beberapa kasus. Contoh :

1. Terdapat n x n koin pada papan. Jumlah robot yang diperlukan oleh kedua algoritma sama – sama berjumlah n robot.
2. Terdapat n baris yang berisi penuh koin. Jumlah koin sama dengan n x baris. Barisan ini harus dimulai dari satu dan berurutan. Tidak boleh ada selang baris. Maka jumlah robot robot yang diperlukan sama dengan banyaknya baris.

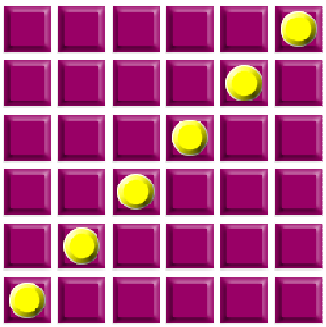


Gambar 3. Kasus koin penuh (kasus 1)



Gambar 4. Kasus baris penuh (kasus 2)

3. Hanya terdapat koin pada diagonal papan dengan koin berada pada kotak paling kiri bawah menuju paling kanan atas. Diagonal papan terisi semua. Maka jumlah robot yang diperlukan sama dengan n. Serta beberapa kasus lainnya.



Gambar 5. Kasus koin diagonal penuh (kasus 3)

Namun demikian, secara keseluruhan. Penggunaan algoritma greedy tetap memberikan hasil yang lebih mangkus dibanding penggunaan algoritma *brute force* sebagai solusi permasalahan.

3. KESIMPULAN

Berdasarkan hasil uraian di atas, dapat kita simpulkan bahwa masalah simulasi pencarian jalan dapat diselesaikan baik dengan algoritma *Brute Force* maupun algoritma greedy. Yang membedakan keduanya dalam pengimplementasian adalah kemangkusan hasil yang diperoleh.

Algoritma *brute force* dapat memastikan bahwa semua koin dapat terambil. Namun demikian, algoritma ini tidak menjamin bahwa pengambilan koin dapat diselesaikan dengan cepat. Apabila hanya terdapat satu koin yang terdapat pada baris paling bawah (namun koin tidak berada pada kotak yang posisinya paling kanan bawah), maka koin baru terambil setelah robot menyisir n – 1 baris sebelumnya terlebih dahulu. Oleh karena itu, dari segi efisiensi waktu, algoritma ini menjadi sangat tidak mangkus untuk kebanyakan kasus posisi koin.

Berbeda dengan algoritma *brute force*, algoritma greedy sebagai salah satu pilihan solusi membuat program jauh lebih efisien. Apabila terdapat satu koin, dimanapun koin itu terletak, hanya satu buah robot yang diperlukan. Selain itu, penggunaan greedy by range di atas juga dapat memastikan bahwa semua koin pasti terambil.

Selain greedy by range, terdapat alternatif greedy versi lainnya yang dapat digunakan antara lain: greedy by column, greedy by row, dan lain sebagainya. Namun demikian, penulis hanya membahas mengenai greedy by range saja.

Untuk beberapa kasus, kedua algoritma ini juga dapat menghasilkan jumlah robot yang sama, bergantung pada kondisi tertentu antara lain : letak koin, jumlah koin n x n , deretan koin hanya pada diagonal papan, dan lain – lain.

REFERENSI

[1] Munir, Rinaldi.”IF 2251 STRATEGI ALGORITMIK Diktat Kuliah Strategi Algoritmik”, Departemen Teknik Informatika, 2006.