SOLUSI ALGORITMA *BACKTRACKING* DALAM PERMAINAN "KSATRIA MENYEBRANG KASTIL"

Yosef Sukianto – Nim 13506035

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung, Jalan Ganesha 10, Bandung
Email : if16035@students.if.itb.ac.id

ABSTRAK

***Game* komputer merupakan salah satu aplikasi *software* yang saat ini banyak dikembangkan. Dengan jenis yang bermacam-macam dan tampilan yang menarik, *game* komputer termasuk *software* yang diminati oleh berbagai kalangan, mulai dari anak kecil sampai pada orang dewasa. Selain karena tampilan dan aplikasinya yang relatif menarik, *game* komputer juga disinyalir dapat menjadi salah satu sarana *refreshing* yang cukup menyenangkan terutama bagi orang yang telah terbiasa menggunakan komputer. dalam makalah ini penulis akan membahas mengenai game "*Ksatria Menyebrang Kastil*".**

Tentu saja untuk mencari solusi dari permainan "*Ksatria Menyebrang Kastil*", selain menggunakan algoritma *backtracking* kita juga dapat menggunakan beberapa metode antara lain : *greedy, DFS* (*Depth First Search*), *BFS* (*Breadth First Search*), dan lain-lain, namun pada makalah kali ini penulis mencoba membahas *game "Ksatria Menyebrang Kastil"* dan menganalisa dengan menggunakan algoritma *backtracking* sebagai salah satu cara penyelesaian persoalan, dan membandingkannya dengan algoritma *brute force*.

Kata kunci: Algoritma *backtracking,* Algoritma *Greedy, game*.

1. PENDAHULUAN

Permainan (*game*) merupakan hal yang menarik bagi sebagian besar masyarakat dunia, mulai dari *game* yang sederhana sampai yang berteknologi tinggi. Salah satu bentuk *game* yang dewasa ini sudah tidak asing adalah *game* yang dimainkan di komputer. Makalah ini dibuat untuk memenuhi tugas mata kuliah Strategi Algoritmik. Selain itu juga bertujuan untuk menambah pengetahuan penulis dalam menganalisa penerapan algoritma untuk menyelesaikan persoalan. Dalam hal ini persoalan yang penulis angkat adalah persoalan dalam penyelesaian *game "Ksatria Menyebrang Kastil". Game "Ksatria Menyebrang Kastil"* pada dasarnya merupakan suatu aplikasi *game.* Pada game ini seorang pemain harus membantu seorang kesatria untuk menyeberang dari istana yang satu ke istana yang lain.

Antar kedua istana tersebut terdapat sebuah persegi yang memiliki 4 kolom jadi terdapat 12 persegi, yang harus diinjak oleh sang kesatria dengan menyentuh tiap persegi sekali dan sang kesatria harus berjalan seperti pada catur (berbentuk huruf "L").

Berbagai data dalam makalah ini penulis peroleh dari berbagai sumber yang berkaitan. Selain itu penulis juga mendapatkan referensi dari berbagai buku dan situs yang berkaitan dengan penerapan algoritma *brute force dan backtracking*.

maka istilah tersebut perlu ditulis dengan cetak miring (*italic*).

Judul dari tiap bab ditulis dengan tipe huruf Times New Roman dengan ukuran 12pt, cetak tebal, huruf besar, serta penomoran dengan huruf (1,2,3,...).

2. TINJAUAN PUSTAKA

2.1 Algoritma *Backtracking*

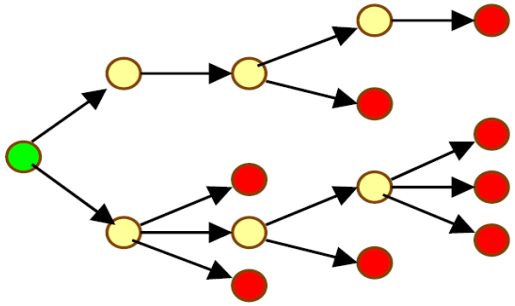
Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Runut-balik, yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Denan metode ini klita tidak perlu memeriksa semua kemungkinan solusi yang ad. Hana pencarian uang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya waktu pencarian dapat

dihemat. Runut- balk lebih alami dinyatakan dengan algoritma rekursif. Kadang – kadang disebutkan pula bahwa runut-balik merupakan bentuk typical dari algoritma rekursif.

Istilah Runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik dan penerappannya pada berbagai persoalan.saat ini, algoritma runut-balik banyak diterapkan untuk program *games* (permainan *tic-tac-toe*, menemukan jalan keluar dalam sebuah labirin, catur, dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*)).

2.1 Pencarian Solusi Algoritma *Backtracking*

Seperti yang telah dijelaskan diatas bahwa pencarian solusi dengan menggunakan algoritma backtracking ini berbasis pada DFS, maka kita menggunakan pohon ruang status.



Gambar 1 Pohon Ruang Status

Langkah-langkah pencarian solusi [1] adalah sebagai berikut :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul yang sudah dilahirkan dinamakan simpul hidup dan simpul hidup yang diperluas dinamakan simpul-E (Expand- node).
2. Jika lintasan yang diperoleh dari perluasan simpul-E tidak mengarak ke solusi, maka simpul itu akan menjadi simpul mati dimana simpul itu tidak akan diperluas lagi.
3. Jika posisi terakhir ada di simpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lainnya dan jika tidak ada simpul anak maka dilakukan backtracking ke simpul orang tua.
4. Pencarian dihentikan jika kita telah menemukan solusi atau tidak ada simpul hidup yang dapat di diperluas.

2.2 Property Umum Algoritma *Backtracking*

untuk menerapkan metode runut-balik, properti berikut didefinisikan:

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan *n-tuple*:
   $X = (x_1, x_2, \ldots, xn)$, $xi \in Si$. Mungkin saja $S_1 = S_2 = \ldots = Sn$.
   Contoh: $Si = \{0, 1\}$,
   $x_i = 0$ atau 1
2. Fungsi pembangkit nilai *xk*
   Dinyatakan sebagai:

   $T(k)$

   $T(k)$ membangkitkan nilai untuk *xk*, yang merupakan komponen vektor solusi.
3. Fungsi pembatas (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)
   Dinyatakan sebagai

   $B(x_1, x_2, \ldots, x_k)$

   B bernilai *true* jika $(x1, x2, \ldots, xk)$ mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk x_k+1 dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, xk)$ dibuang.

Semua kemungkinan solusi dari persoalan disebut ruang solusi (*solution space*). Secara formal dapat dinyatakan, bahwa $xi \in Si$, maka

$$S_1 \times S_2 \times \ldots \times S_n$$

Disebut ruang solusi. Jumlah anggota di dalam ruang solusi adalah $|S_1| \cdot |S_2| \cdot \ldots \cdot |S_n|$.

2.3 Skema Umum Algoritma *Backtracking*

Di bawah ini disajikan skema runut balik

```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan
metode runut-balik; skema rekursif
Masukan: k,yaitu indeks komponen vektor
   solusi, x[k]
Keluaran: solusi x = (x[1], x[2], …, x[n])}
  Algoritma:
  for tiap x[k] yang belum dicoba
      sedemikian sehingga ( x[k]□T(k)) and
      B(x[1], x[2], ... ,x[k])= true do
        if (x[1], x[2], ... ,x[k])
        lintasan dari akar kedaun then
              CetakSolusi(x)
        endif
      RunutBalikR(k+1)
  Endfor
```

T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi. B(x1, x2, …, xk) adalah fungsi pembatas menentukan apakah (x1, x2, …, xk) mengarah ke solusi.

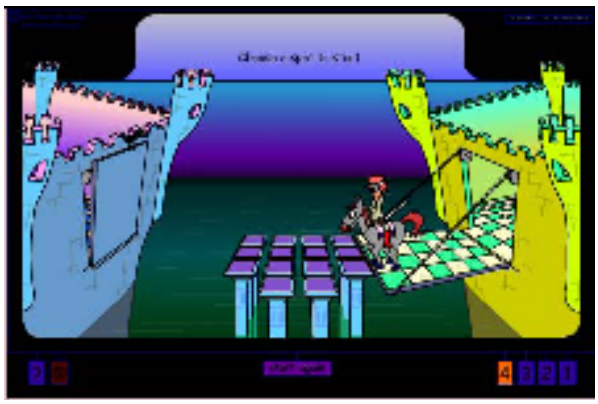
3. HASIL DAN ANALISIS

3.1 Analisis dengan *Brute force*

kebanyakan pemain biasanya memilih untuk melakukan pilihan secara *brute force,* yang artinya memilih acak atau terserah dari si pemain. Namun jarang atau sedikit sekali pilihan tersebut menuju pada solusi, karena keterbatasan waktu yang dimiliki oleh si pemain.

Namun algoritma *brute force* menjamin si pemain pasti mendapatkan sebuah solusi jika memiliki waktu yang cukup dengan mencoba semua kemungkinan yang ada.

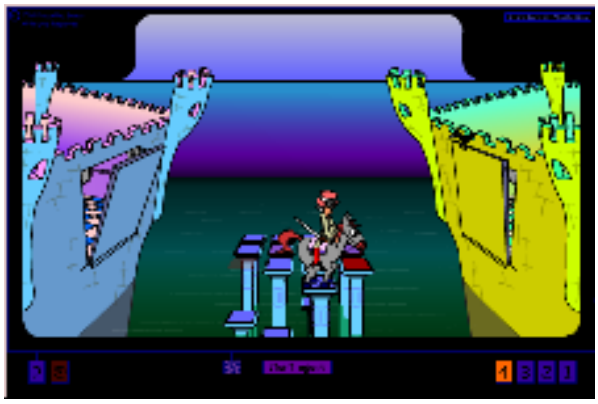
Apabila permainan ini diselesaikan secara *brute force* maka membutuhkan waktu minimal P (14,14)=8,7178291 x 10_{10} detik, jika kita kalikan dengan 1 detik.Berikut adalah cuplikan gambar dari permainan tersebut.



Gambar 2 Permainan "Ksatria Menyebrang Kastil"

3.2 Analisis dengan *Backtracking*

Permasalahan yang ada dalam permainan ini adalah banyaknya pilihan yang tersedia. Sang kesatria harus memilih kotak pertama dari 14 kotak yang tersedia, kemudian ia harus berjalan seperti dalam catur dalam bentuk "L". Setiap kotak yang telah ia injak tidak dapat dipilih lagi, setelah semua kotak terpilih ia harus kembali ke kotak awal, baru setelah itu ia dapat menyeberang ke istana yang berwarna biru. Jika ia tidak dapat kembali ke kotak awal maka ia tidak dapat menyeberang.



Gambar 3 Permainan Sedang Berjalan

Kotak yang berwarna merah adalah kotak awal yang ia pilih, sedangkan kotak lain yang telah ia injak akan menurun. Kotak yang dapat ia pilih selanjutnya (sesuai syarat harus berbentuk "L") diberi warna merah transparan sehingga memudahkan pemain untuk memilh kotak selanjutnya.

Adapun algoritmanya secara garis besar untuk memecahkan permasalahan ini adalah seperti yang ada dibawah ini.

```
If bukan solusi then
  while belum sampai pada tujuan do
     if terdapat arah yang benar sedemikian
        sehingga belum pernah berpindah ke
        kotak pada arah tersebut
     then
       pindah satu langkah ke kotak pada
       arah tersebut
     if terdapat arah yang benar namun
        kotak tersebut   pernah diinjak
     then
        backtrack
     endif
     endif
  endwhile
else
   solusi {semua kotak telah diinjak dan
   posisi kesatria   kembali ke kotak awal}
```

Permasalahannya sekarang adalah bagaimana kita mengetahui kotak mana yang belum diinjak dan arah mana yang harus dituju. Solusinya adalah untuk semua kotak diberi kondisi *true*, yang artinya kotak tersebut belum diinjak, dan sebaliknya *false* berarti sudah pernah diinjak. Untuk masalah arah "L", diasumsikan program sudah diatur untuk membuat arah semua kotak yang merupakan arah "L" untuk kotak yang sedang diinjak oleh

ksatria. Berikut adalah algoritma *backtracking* untuk permainan ini.

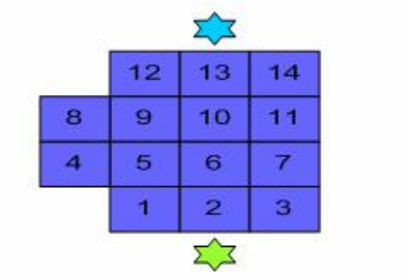
```
Function Solusi(input P: permainan)→boolean
{true jika solusi ditemukan, false jika
tidak}

Deklarasi
    pindah : boolean = true {true ≠ injak,
             false = injak}
    arah   : integer {atas = 1, bawah = 2,
             kanan = 3, kiri = 4}
Algoritma
if solusi sudah ditemukan then
  return true
else
  for(tiap arah gerakan and pindah= true)do
    gerak(P,arah){pindah kekotak yang
                  ditunjuk}
    if Solusi (P) then
       return true
    else
       tdkGerak (P,arah) {backtrack}
    endif
  endfor
  return false {semua arah sudah
   dicobatetapi tetap buntu, jadi solusinya
   tidak ada}
endif
```

Algoritma *backtracking* pada permainan ini dapat dipandang sebagai pembentukan pohon ruang status. Akar dari pohon adalah kondisi dimana kesatria berada pada kotak awal dan daunnya (anak- anaknya) adalah kotak selanjutnya yang dipilih dan mengikuti arah "L". Pohon ruang status ini berbasis pada *DFS* (*Depth First Search*), yaitu pencarian secara mendalam, dimulai dari simpul awal (akar) sampai ke simpul akhir, dan jika simpul akhir bukan solusi maka dilakukan runut-balik ke simpul tetangga yang belum ditelusuri.

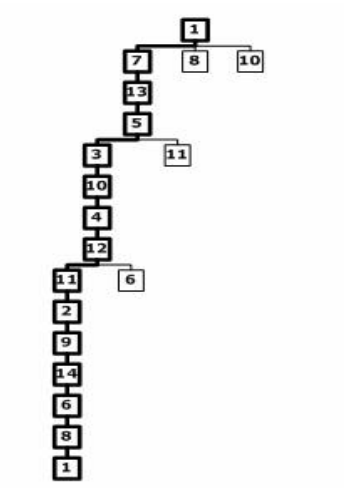
Setiap simpul dalam pohon ruang status yang berjumlah 2^n atau n!, berasosiasi dengan pemanggilan rekursif, maka kasus terburuk algoritma ini membutuhkan waktu sebesar $O(p(n)2^n)$ atau $O(p(n)n!)$, dimana p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul [1].

Di bawah ini adalah gambar kotak dari permainan ini sehingga dapat mempermudah kita dalam membuat pohon ruang statusnya



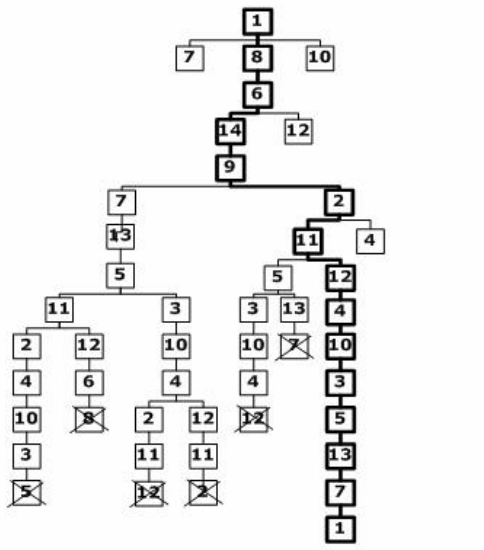
Gambar 4 Kotak-Kotak Pilihan

Setelah kita memberi angka pada kotak-kotak tersebut, maka kita akan membuat pohon ruang statusnya dengan menggunakan angka-angka tersebut. Bintang yang berwarna hijau mewakili istana yang berwarna hijau dan merupakan kondisi awal dimana kesatria berada dan bintang yang berwarna biru mewakili istana yang berwarna biru yang merupakan kondisi akhir dimana kesatria seharusnya berada (solusinya).



Gambar 5 Pilihan Kotak Awalnya 1-7

Jika kotak pertama yang dipilih adalah angka 1, maka kesatria ada 3 pilihan (angka 7, 8 atau 10), jika ia memilih angka 7, maka jalur lintasan untuk mendapatkan solusi adalah **1-7-13-5-3-10-4-12-11-2-9-14-6-8-1**



Gambar 7 Pilihan Kotak Awalnya 1-8

Jila ia memilih angka 8, maka lintasan solusinya adalah **1-8-6-14-9-2-11-12-4-10-3-5-12-7- 1**. Kita lihat bahwa ada kotak yang disilang, itu merupakan simpul mati karena tidak menuju pada solusi. Setelah menemukan simpul mati, kemudian dilakukan *backtracing* ke simpul terdekatnya, sampai kita menemukan solusi atau tidak ada lagi simpul hidup yang dapat ditelusuri.

4. KESIMPULAN

Untuk berbagai macam persoalan yang memiliki banyak kemungkinan langkah untuk menemukan solusi, runut-balik dapat menyelesaikan dengan cukup baik. Hal yang paling penting dalam pengoptimalan algoritma runut-balik dalam pencarian solusi adalah penentuan fungsi pembangkit dan fungsi pembatas. Kedua fungsi inilah yang akan menentukan aturan langkah mana yang harus diambil dalam pencarian solusi.

Melalui analisis yang didapat dari penggunaan algoritma runut-balik ini, masih menunjukkan kerumitan dalam penyelesaian masalah. Terutama untuk masalah dengan jumlah solusi yang cukup banyak. Namun untuk saat ini algoritma runut-balik merupakan algoritma yang bisa menyelesaikan banyak persoalan, karena itulah sering dipakai dalam pembuatan games.

Dengan pohon ruang status, kita berhenti saat kita menemukan solusi dari semua kemungkinan solusi yang ada. Sehingga metode ini jauh lebih efektif dibandingkan dengan *brute force* yang mencari semua kemungkinan solusi.

Agar penggunaan backtracking lebih mangkus maka sebaiknya ditambahkan fungsi heuristic pada algoritma backtracking.

REFERENSI

[1] Munir, Rinaldi. 2005. Strategi Algoritmik.Bandung. Institut Teknologi Bandung.

[2] Preiss, Bruno. 1997. Abstract Backtracking Solvers. http://www.brpreiss.com/books.

[3] http://delphiforfun.org/Programs/castle_escape.htm