Algoritma Backtracking Pada Permainan Peg Solitaire

Gilbran Imami, 13509072
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509072@std.stei.itb.ac.id

***Abstrak*—*Peg Solitaire* merupakan salah satu *board game* populer di dunia yang dapat dimainkan oleh satu orang pemain. *Peg Solitaire* dimainkan dengan menggunakan beberapa set kelereng dan papan yang berlubang. Objektif dari permainan ini adalah memindahkan *peg* dengan melangkahi suatu *peg* di depannya. *Peg* dapat dipindahkan ke lubang yang kosong, dan *peg* yang dilangkahi dapat dihilangkan. Tujuan akhir permainan ini adalah menghabiskan seluruh *peg* yang ada dengan melangkahi *peg-peg* tersebut hingga tersisa satu *peg*. Meskipun terdengar sederhana, namun untuk dapat mencapai kondisi seperti itu sangat sulit untuk dicapai. Dibutuhkan pemikiran yang matang dalam setiap langkah yang diambil. Karena jika salah, maka di akhir akan ada lebih dari satu *peg* yang tersisa. Dalam makalah ini, penulis mencoba untuk mencari solusi untuk menyelesaikan permainan ini dengan menerapkan algoritma *backtracking*. Algoritma *backtracking* adalah salah satu algoritma yang digunakan untuk mencari solusi dari permasalahan komputasional. Ada banyak cara yang dapat digunakan untuk mencari solusi, dan algoritma *backtracking* bukanlah solusi terbaik setiap saat. Namun, algoritma ini dapat digunakan untuk mencapai kondisi akhir yang ideal dari permainan *peg solitaire*.**

***Kata Kunci*—Algoritma *backtracking*, *peg solitaire*, *peg*, *hash-table*.**

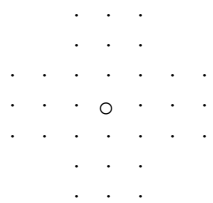
I. PENDAHULUAN

Peg Solitaire merupakan permainan yang cukup populer sejak dulu. Menurut beberapa orang, permainan ini pertama kali diciptakan oleh seorang aristokrat Perancis pada abad ke 17. Namun, hal ini masih menjadi sebuah perdebatan di tahun 1800-an. Salah satu orang yang mencari pembuktian ini bernama John Beasley. John Beasley mengemukakan bahwa permainan ini bukanlah berasal dari Perancis, melainkan salah satu warisan turun-temurun dari suku asli Amerika. Walaupun pernyataannya ini juga belum memiliki bukti yang cukup kuat.

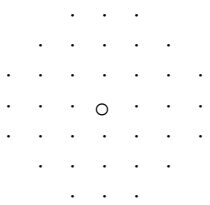
Permainan ini terkenal di beberapa negara dan memiliki nama yang berbeda-beda pula tergantung tempat permainan ini dimainkan. Misalnya di Inggris, permainan ini bernama "*Solitaire*". Sedangkan permainan kartu yang selama ini kita kenal dengan nama "*Solitaire"* disebut sebagai "*Patience"* di Inggris. Di India, permainan ini dikenal dengan nama "*Brainvita"*.

Dalam permainan ini kelereng, atau disebut sebagai *peg*, diisi secara penuh pada papan permainan, kecuali pada lubang bagian tengah.

Ada dua jenis papan yang lazim digunakan pada permainan ini, jenis papan Inggris dan jenis papan yang digunakan negara Eropa lain seperti gambar berikut:

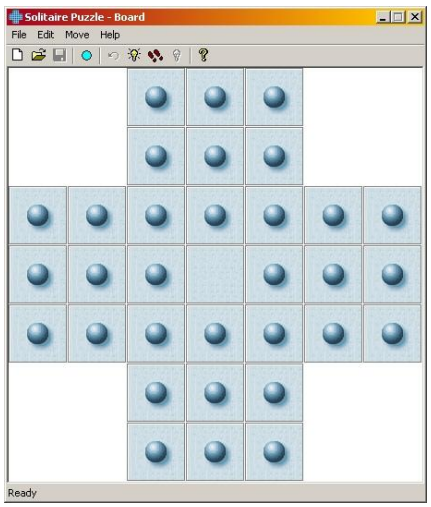


Gambar 1. Jenis papan *peg solitaire* Inggris.



Gambar 2. Jenis papan *peg solitaire* Eropa.

Setelah zaman menjadi terkomputerisasi, maka permainan yang memiliki popularitas cukup tinggi ini turut merambah dunia *digital*. Hingga sekarang permainan versi komputer dari *peg solitaire* ini sangat banyak jumlahnya, baik yang sangat sederhana maupun dengan tampilan dan fitur-fitur yang sangat menarik.



Gambar 3. Permainan "*Solitaire Puzzle*", sebuah permainan *peg solitaire* sederhana.

Untuk menyelesaikan permainan ini, ada banyak cara dan algoritma yang dapat digunakan. Dan disini akan diterangkan bagaimana menyelesaikan permainan ini dengan algoritma *backtracking*.

II. Dasar Teori

Bagian ini berisi penjelasan mengenai aturan permainan, dan beberapa teori-teori mendasar yang berhubungan dengan pencarian solusi permainan *peg solitaire*.

A. *Permainan*

Langkah yang valid yaitu melangkahi satu *peg* secara ortogonal dengan *peg* yang bersebelahan ke lubang sejauh dua posisi dan kemudian menghilangkan *peg* yang dilangkahi.

Tanda * mengindikasikan suatu *peg* di dalam lubang, ***** mengindikasikan *peg* yang akan dipindahkan, dan ○ mengindikasikan lubang yang kosong. Tanda ¤ adalah lubang asal *peg* yang baru saja dipindahkan, ***** adalah posisi final dari *peg* tersebut, dan **o** adalah lubang dari *peg* yang dilangkahi dan dihilangkan.

Maka, langkah yang diperbolehkan antara lain:

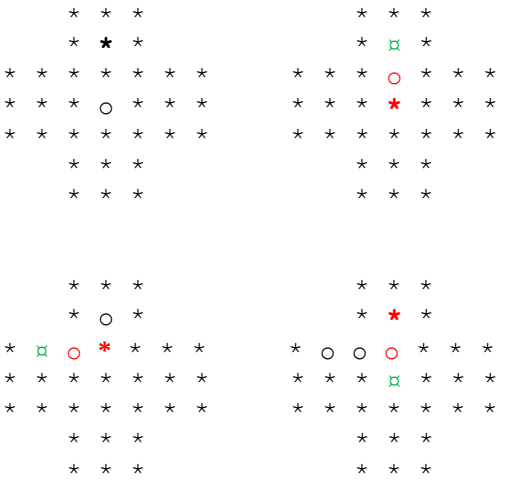
***** * ○ → ¤ **o** ***** Melangkah ke kanan

○ * ***** → ***** **o** ¤ Melangkah ke kiri

***** ¤
* → **o** Melangkah ke bawah
○ *****

○ *****
* → **o** Melangkah ke atas
***** ¤

Dalam papan Inggris, tiga langkah awal kemungkinan:



Kesalahan sangat mungkin dilakukan dan menyebabkan adanya dua atau tiga *peg* yang letaknya saling berjauhan. Hal ini menyebabkan banyak orang tidak bisa menyelesaikan permainan ini.

Ada banyak solusi untuk permasalahan yang sering ditemui, dan satu cara yang bisa dilakukan yaitu memberi huruf pada tiap lubang yang ada:

Inggris

```
    a b c
    d e f
g h i j k l m
n o p x P O N
M L K J I H G
    F E D
    C B A
```

Eropa

```
    a b c
  y d e f z
g h i j k l m
n o p x P O N
M L K J I H G
  Z F E D Y
    C B A
```

Notasi gambar cermin seperti ini digunakan karena pada papan Eropa, satu set alternatif dari permainan ini adalah dimulai dengan lubang di beberapa posisi dan diakhiri dengan satu *peg* di posisi cerminnya. Dalam papan Inggris set alternatif yang serupa dimulai dengan lubang dan diakhiri dengan *peg* di posisi yang sama.

Tidak ada solusi untuk papan Eropa yang memiliki lubang di tengah, jika hanya pergerakan ortogonal yang diizinkan. Hal ini sangat mudah kita lihat sebagai berikut,

dengan argumen dari seseorang bernama Hans Zantema. Bagi posisi papan ke dalam posisi A, B, dan C seperti berikut:

```
      A B C
    A B C A B
  A B C A B C A
  B C A B C A B
  C A B C A B C
    B C A B C
      A B C
```

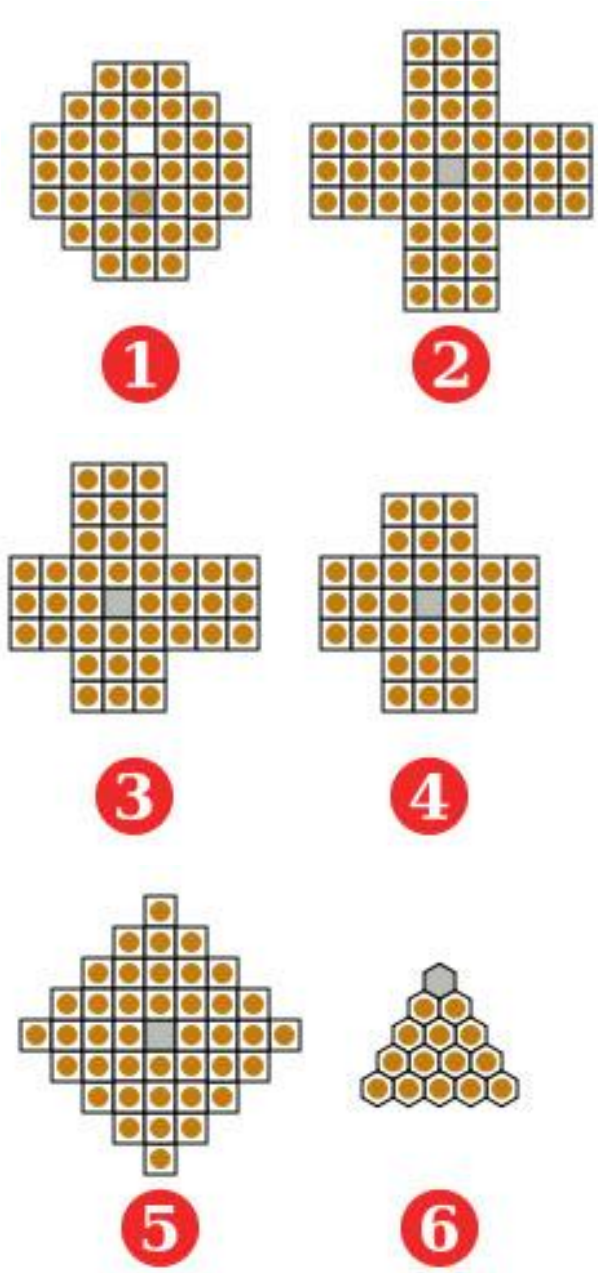
Pada awalnya dengan hanya posisi tengah papan permainan yang kosong, jumlah posisi yang dimiliki oleh A adalah 12, posisi B 12, juga posisi C 12. Setelah setiap perpindahan, jumlah posisi A naik atau turun sejumlah satu, dan hal yang sama juga berlaku pada B dan C. Sehingga, setelah perpindahan sebanyak n kali dengan n adalah bilangan genap, ketiga angka (untuk A, B, dan C) akan menjadi genap pula, dan setelah perpindahan sebanyak m kali dengan m adalah bilangan ganjil, ketiga angka (untuk A, B, dan C) akan menjadi ganjil pula. Sehinggan posisi akhir dengan hanya 1 *peg* yang tersisa tidak mungkin tercapai, karena salah satu dari ketiga angka untuk A, B, dan C ini adalah 1 (ganjil), sedangkan yang dua lainnya adalah 0 (genap).

Namun, terdapat cara lain di mana sebuah lubang yang ada dapat disederhanakan menjadi hanya satu *peg* yang tersisa. Taktik yang digunakan adalah dengan membagi papan permainan menjadi paket-paket terdiri dari tiga *peg*, dan menghilangkan ketiga *peg* tersebut dengan menggunakan sebuah *peg* ekstra, sebagai katalis, yang digunakan untuk lompat ke luar dan kemudian lompat ke dalam lagi. Contohnya dapat dilihat di gambar berikut, di mana * adalah katalisnya:

```
* * o      ¤ o *      o * *      * o ¤
  *    ->    *    ->    o    ->    o
  *          *          ¤          o
```

Teknik ini dapat digunakan dengan baris yang terdiri dari 3 *peg*, blok yang terdiri dari 2*3 *peg*, dan 6 *peg* berbentuk L dengan dasar yang panjangnya 3 *peg* dan tinggi yang panjangnya 4 *peg*.

Selain kedua jenis papan permainan yang populer tersebut, ada juga jenis dan ukuran papan yang lainnya, termasuk juga papan permainan berbentuk segitiga, dengan *peg* yang digunakan boleh melompat ke tiga arah.



Gambar 4. Macam-macam papan *peg solitaire* lain

1) Tipe Perancis, 37 lubang, digunakan pada abad 17
2) J.C. Wiegleb (Jerman), 45 lubang, 1779
3) 3-3-2-2 Asimetris oleh George Bell, pada abad 20
4) Tipe Inggris (standar), 33 lubang
5) Wajik, 41 lubang
6) Segitiga, 15 lubang

B. *Algoritma Backtracking*

Terdapat beberapa algoritma yang digunakan secara global untuk menemukan seluruh atau sebagian permasalahan komputasi, dua di antaranya adalah *Brute Force* dan *Backtracking*. Kedua algoritma ini sama-sama mencoba untuk mencari solusi dari permasalahan dengan membuat kandidat-kandidat solusi secara gradual.

Bagaimana pun, *backtracking* lebih efisien

dibandingkan dengan algortima *brute force*. Ini dikarenakan, pada algoritma *brute force*, sebuah program akan membuat setiap pemecahan atau solusi yang mungkin. Untuk setiap solusi, program akan kemudian mengecek apakah solusi tersebut telah memenuhi spesifikasi atau syarat solusi final yang sebenarnya. Sementara itu, algoritma *backtracking* akan berhenti memproses kandidat solusi jika telah terbukti tidak mengarah pada solusi final yang sebenarnya. Misalnya, sebuah program diminta untuk menyusun sebuah kata yang terdiri dari 5 huruf, dengan syarat terdiri dari maksimal tiga huruf konsonan. Saat program menemukan huruf konsonan ke empat dalam prosesnya mencari solusi yang sesuai, maka kandidat tersebut dan yang selanjutnya yang mengikuti akan dibuang, karena sudah tidak mungkin lagi untuk memenuhi syarat solusi yang diminta.

Namun, algoritma *backtracking* hanya dapat diaplikasikan pada beberapa permasalahan tertentu. Algoritma ini hanya dapat digunakan pada solusi atau pemecahan permasalahan yang dapat dicari secara sistematis dan bertahap. Algoritma *backtracking* hanya dapat diterapkan pada permasalahan yang memiliki konsep kandidat solusi parsial, dan dapat dicek secara cepat apakah kandidat tersebut mengarah pada solusi yang valid.

Algoritma *backtracking* didasarkan pada DFS (*Depth-First Search*). Mekanisme *backtracking* menggunakan prinsip rekursif. Untuk menemukan solusi dari keseluruhan permasalahan, diperlukan solusi dari sub permasalahannya. Setelah itu, solusi ini akan digunakan untuk memecahkan subpermasalahan lainnya secara rekursif. Saat kandidat yang saat ini sedang diperiksa gagal, atau jika seluruh solusi yang mungkin harus ditemukan, maka program akan melakukan backtrack ke node sebelumnya dan mengecek solusi berikutnya yang mungkin. Proses *backtracking* akan berhenti saat sudah tidak ada lagi solusi yang mungkin. Salah satu karakteristik yang paling signifikan dari algoritma *backtracking* adalah fungsi "pemotongan". Misalnya tahapan dari pencarian solusi tersebut direpresentasikan dalama bentuk tree, pemotongan akan dilakukan untuk node-node yang tampaknya tidak akan membentuk solusi yang valid. Saat sebuah node telah dipotong, anak node tersebut secara otomatis akan dibuang dari proses, karena memotong sebuah node sama artinya dengan membuang seluruh path yang mengikutinya.

Algoritma *backtracking* digunakan secara luas dalam pembuatan game PC seperti tic-tac-toe, labirin, dan catur. Algoritma ini biasanya digunakan untuk membangun *Artificial Intelligence* (AI) untuk game. Di samping itu, algoritma ini juga merupakan cara paling efisien untuk parsing dan permasalahan konbinatorial lainnya.

III. PENGAPLIKASIAN ALGORITMA *BACKTRACKING*

Algoritma *backtracking* atau runut balik dapat diaplikasikan untuk mencari solusi dari permainan *peg solitaire*. Contoh algoritma yang penulis buat untuk menyelesaikan permainan *peg solitaire* adalah sebagai berikut:

```
boolean move (board x, moveSequence mseq) {
if (solved(x)) return true;
curMoves = currentMoves(x); // himpunan
semua pergerakan
for (every m in curMoves) {
  y = makeMove(x, m);
  if (move(y, mseq)) {
       mseq = push(m, mseq);
       return true;
  }
 }
return false;
}
```

Algoritma *backtracking* seperti yang ditulis diatas akan menyelesaikan permainan dengan jumlah *peg* dengan jumlah sekitar 20. Tetapi untuk papan dengan junla *peg* yang lebih banyak, algoritma *backtracking* akan memakan waktu beberapa menit hingga beberapa jam, bahkan bisa berhari-hari. Tetapi dengan menambahkan Hash Table dapat meningkatkan performansi dari algoritma secara signifikan.

Ide utamanya adalah sebagai berikut. Misalkan X adalah posisi awal. Dan kemudian banyak urutan pergerakan yang akan membawa X ke posisi Y. Algoritma *backtracking* akan mengikuti jalur yang sesuai dengan salah satu urutan hingga sampai ke posisi Y. Pada tahap ini, rekursif akan terjadi dengan Y sebagai input. Misalkan Y tidak menunjukkan solusi yang diinginkan, maka pemanggilan ini akan mengeluarkan false dan pencarian akan dilanjutkan. Pada suatu saat, algoritma *backtracking* akan mengikuti urutan pergerakan yang lain dan akan menuju posisi Y. Pada tahap ini, pemanggilan rekursif akan terjadi lagi dengan Y dan dapat terjadi berulang kali. Pengulangan dapat dihindari dengan memperhatikan semua papan dimana pemanggilan rekursif telah gagal sejauh ini dalam sebuah struktur data. Operasi ini akan lebih efisien dengan didukung oleh struktur data dengan metode *search and insert*. Dengan begitu, *hash table* digunakan untuk menyimpan papan-papannya.

Berikut ini adalah versi modifikasi dari algoritma *backtracking* dengan menggunakan *hash table*.

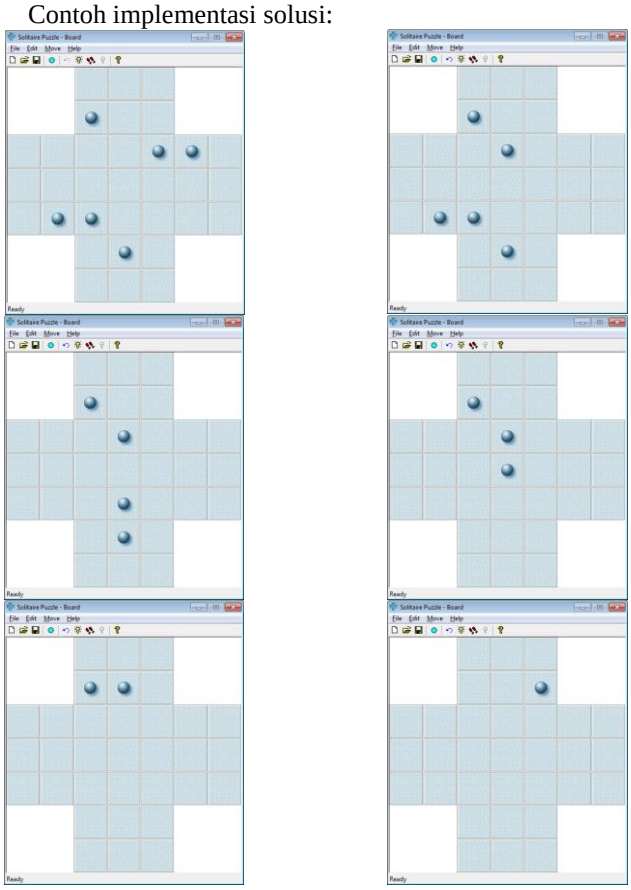
```
boolean move (board x, moveList mseq,
Hashtable H) {
if (solved(x)) return;
curMoves = currentMoves(x); //semua
pergerakan yang mungkin di x
for (every m in curMoves) {
  y = makeMove(x, m); // y adalah hasil
pembuatan m di x
  if (y is not in H) {
       if move(y, mseq, H) {
       mseq = push(mseq, y);
       return true;
  } // end inner if
  else
       insert y into H;
```

```
  } // end outer if
} // end for
return false;
} //end move
```

Dapat dilihat bahwa perubahan yang dibuat yaitu:

1. Sebelum pemanggilan rekursif, dilakukan pencarian untuk papan Y di hash-table untuk menghindari pemanggilan berulang.
2. Saat pemanggilan mengembalikan false, disisipkan papan ke hash-table.

Contoh implementasi solusi:



Gambar 5. Implementasi algoritma *backtracking*

IV. KESIMPULAN

Algoritma *backtracking* biasa digunakan untuk menyelesaikan masalah komputasional. Penggunaan algoritma ini cukup efektif saat diaplikasikan ke permasalahan yang sesuai. Dengan beberapa perbaikan, seperti penambahan *hash-table*, algoritma ini dapat menyelesaikan permainan *peg solitaire* dengan cukup baik. Tetapi, solusi menggunakan algoritma *backtracking* mungkin bukan solusi yang terbaik untuk permainan ini.

REFERENSI

[1] Munir, Rinaldi, "Strategi Algoritmik", Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2007, 125-149

[2] http://en.wikipedia.org/wiki/*Peg_solitaire*
Tanggal Akses: 6 Desember 2010, 20:00 WIB

[3] http://en.wikipedia.org/wiki/*Backtracking*
Tanggal Akses: 8 Desember 2010, 22:00 WIB

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, 8 Desember 2011

Gilbran Imami, 13509072