# Penerapan Algoritma Backtrack pada Knight's Tour

Adhika Aryantio 13511061
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*adhika.aryantio@students.itb.ac.id*

***Backtracking* adalah cara yang metodologis mencoba beberapa sekuens keputusan, sampai Anda menemkan sekuens yang "bekerja". Penerapan *backtracking* dalam kehidupan sehari-hari sangat banyak, namun saya dalam kesempatan ini akan mengambil topik yang unik yaitu penerapan algoritma *backtrack* pada penyelesaian *Knight's Tour*. *Knight's Tour* adalah jalan *knight* pada papan catur yang dimulai dari titik tertentu, lalu menginjak semua petak yang ada tepat 1 kali dan kembali ke posisi awal. Dalam paper ini bisa dilihat perbedaan dari algoritma *brute force* dan algoritma *bactrack* dalam menyelesaikan masalah tersebut.**

***Keyword* : *Knight's tour, backtrack, brute force***

## I. PENDAHULUAN

Banyak algoritma yang terdapat di seluruh belahan dunia, terdapat algoritma yang masing-masing dari algoritma tersebut memiliki fungsi dan kegunaan untuk memecahkan solusi terterntu. Diantaranya yaitu algoritma *Brute Force*, algoritma *greedy*, algoritma *dfs* dan *bfs*, algoritma *backtracking*, algoritma *B&B*, algoritma *string matching*, dan masih banyak lagi algoritma-algoritma lainnya yang terdapat di dunia ini.

Pada dokumen ini, algoritma yang akan lebih lanjut dibahas adalah algoritma runut-balik atau disebut dengan algoritma *backtracking*. Salah satu masalah menarik yang dapat dilakukan dengan algoritma *backtracking* ini adalah langkah knight yang mana *knight* tersebut dapat melewati seluruh kotak papan catur tepat satu kali.
Persoalan yang akan dibahas yaitu penyelesaian *Knight's Tour* dengan menggunakan Algoritma *backtracking*.

### 1.1. Algoritma *Backtracking*

Istilah backtracking pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950 dan terdapat juga tokoh-tokoh seperti R.J.Walker, Golomb, Baumert yang menyajikan uraian umum tentang *backtracking*.

Algoritma *backtracking* adalah algoritma pencarian solusi yang berbasis pada *DFS*. Algoritma *backtracking* banyak diterapkan untuk program games :

1. Permainan *tic-tac-toe*
2. Menemukan jalan keluar dari sebuah *maze*
3. Catur termasuk didalamnya permainan dari *knight's tour* ini.

Algoritma *backtracking* merupakan perbaikan dari algoritma *brute-force*. Pada *brute-force* semua kemungkinan solusi dieksplorasi satu per satu sedangkan pada backtracking hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan kembali.

#### *1.1.1.* Properti Umum Algortima *Backtracking*

1. **Solusi Persoalan**
   Solusi dinyatakan dalam bentuk vektor dengan *tuple* : $X = (x_1,x_2,x_3,...,x_n)$, $x_i \in S_i$
   Mungkin terjadi $S_1 = S_2 = ... = S_n$
   Contoh : $S_i = \{0,1\}$, $x_i$=0 atau 1
2. **Fungsi Pembangkit**
   Fungsi pembangkit nilai $x_k$
   Dinyatakan dalam predikat *T(k)* dimana *T(k)* membangkitkan nilai untuk $x_k$, yang merupakan komponen vektor solusi.
3. **Fungsi Pembatas**
   Dinyatakan dalam predikat : $B$ $(x_1,x_2,..,x_k)$.
   $B$ bernilai *true* jika $(x_1,x_2,..,x_k)$ mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk $x_k$+1 dilanjutkan, tetapi jika *false*, maka $(x_1,x_2,..,x_k)$ dibuang.

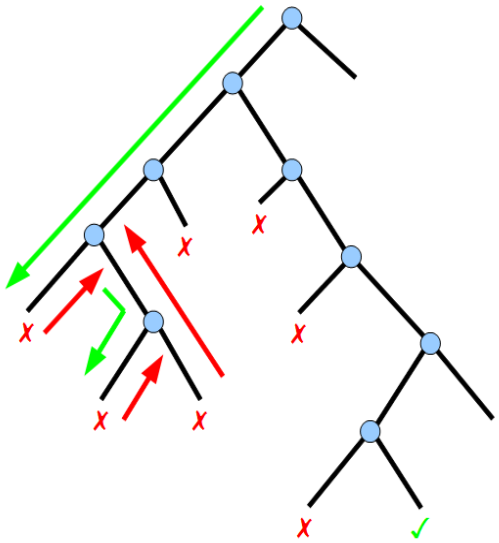
#### 1.1.2. Prinsip Pencarian Solusi dengan Metode *Backtracking*

- Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan *depht-first order* (DFS).
- Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*)
- Simpul-simpul yang sedang diperluas dinamakan **simpul-E** (*Expand node*)
- Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
- Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi **simpul mati** (*dead node*)
- Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas**.
- Simpul yang sudah mati tidak pernah diperluas

lagi.

- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul aras diatasnya.
- Lalu, teruskan dengan membangkitkan simpul anak lainnya.
- Selanjutnya simpul ini menjadi simpul-E yang baru.
- Pencarian dihentikan bila kita telah sampai pada *goal node*.

Berikut merupakan gambar ilustrasi algoritma *backtrack* :



**Gambar 1.1** Ilustrasi Algoritma *Backtrack*

### 1.1.3. Kompleksitas Waktu Algoritma *Backtracking*

- Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.
- Jika jumlah simpul dalam pohon ruang status adalah $2^n$ atau n!, maka untuk kasus terburuk, algoritma *backtrack* membutuhkan waktu dalam :
  1. $O(p(n)2^n)$ atau
  2. O(q(n)n!)
- Dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi simpul.

### 1.1.4. Skema Umum Algoritma Runut Balik menggunakan rekursif

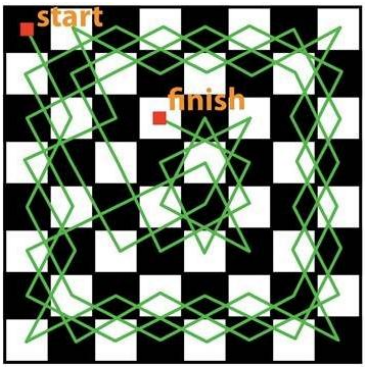
```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema rekursif
 Masukan: k, yaitu indeks komponen vektor solusi, x[k]
 Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Algoritma:
   for tiap x[k] yang belum dicoba sedemikian sehingga
         ( x[k]←T(k))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

## II.*KNIGHT'S TOUR*

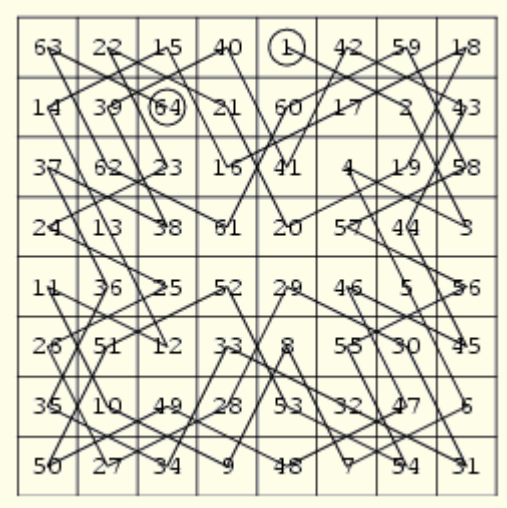
*Knight's Tour* adalah merupakan sebuah siklus hamilton, langkah-langkah *knight* pada papan catur sehingga seluruh kotak terlewati tepat satu kali. Aturan langkah *knight* pada permainan catur adalah sebagai berikut :

- Melangkah dua persegi ke arah horizontal kemudian satu persegi ke arah vertikal atau,
- Melangkah dua ersegi ke arah vertikal kemudian satu persegi ke arah horizotal, atau
- Melangkah satu persegi ke arah horizontal kemudian dua persegi ke arah vertikal
- Melangkah satu persegi ke arah vertikal kemudian dua persegi ke arah horizontal.

Pada penerapan *Knight's Tour* pada sirkuit Hamilton, terdapat 2 jalan dari *knight's tour* ini yaitu *closed tour* dan *open tour*.
Berikut gambar *closed* dan *open tournya* :




**Gambar 2.1** *Open Knight's Tour*

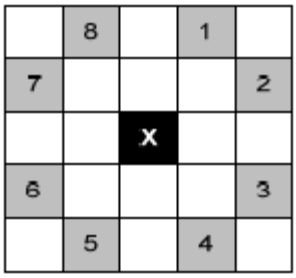



**Gambar 2.2** *Closed Knight's Tour*

## III. PENERAPAN ALGORTIMA *BACKTRACK* PADA *KNIGHT'S TOUR*

Cara knight jalan dalam permainan catur adalah dengan membentuk huruf L pada papan catur, langkah-langkah yang mungkin dilakukan oleh knight dalam melangkah pada papan catur adalah sebagai berikut :

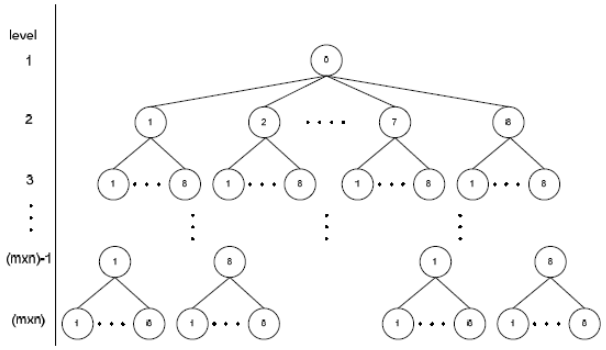
1. A(x+2,y+1),
2. A(x-2,y+1),
3. A(x+2,y-1),
4. A(x-2,y-1),
5. A(x+1,y+2),
6. A(x-1,y+2),
7. A(x+1,y-2),
8. A (x-1,y-2).




**Gambar 3.1** Langkah *Knight*

Sehingga dari data yang didapat dari data diatas bahwa kemungkinan sebuah knight melangkah untuk sekali jalannya adalah 8 langkah. Oleh karen itu terbentuklah pohon merentang yang memiliki 8 buah kemungkinan.

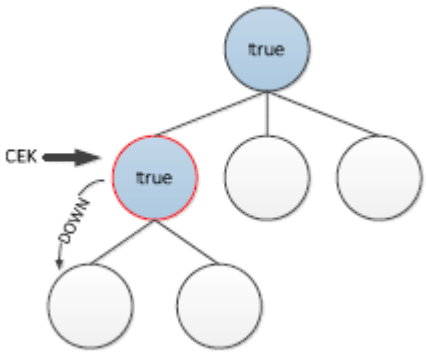
Pohon merentang memiliki level dari 1 sampai level (m x n) dimana m merupakan kolom dari papan catur dan n merupakan baris dari papan catur sehingga maksimum yang ditempuh adalah jumlah seluruh kotak terlah dipijak oleh *knight*.




**Gambar 3.1** Pohon Merentang *knight's tour*

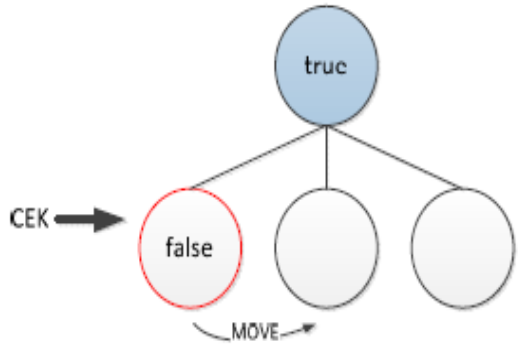
### 3.1. Proses Pencarian Solusi dengan menggunakan *backtrack*

1. Proses dimulai dengan menginisiasi langkah pertama dari sebuah knight dimana langkah tersebut dijadikan sebagai *node* pertama atau akar dalam sebuah *tree*, lalu tandai *node* pertama sebagai node yang valid.
2. Lalu lakukan pengecekan pada anak pertama dari *node* yang pertama kali anda buat, lalu telusuri anak dari *node* pertama tersebut, karena sekarang sudah pada level 2 maka tandai dengan memberikan tanda *levelDown*.
3. Berikutnya lakukan *rekursif* pada anak-anak dari *node* yang utama perlakukan seperti *node* pertama untuk setiap *parentnya*
4. Apabila iterasi yang dicek pada *node-node* yang dipilih adalah *valid* dan *node* yang valid tersebut bukan *node* yang paling bawah ( *goal node* ), maka selalu beri tanda *levelDown* karena pengecekan harus *rekursif* hingga sampai pada simpul *goal*



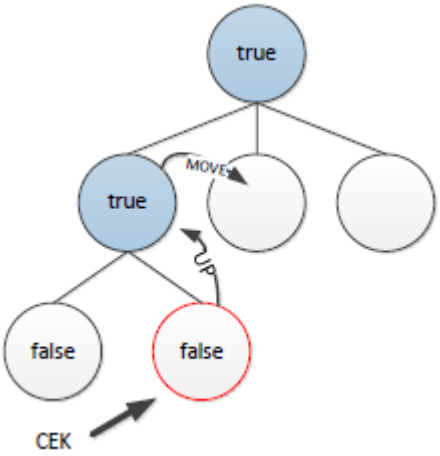

**Gambar 3.3** *levelDown*

5. Bila *node* yang anak yang dicek merupakan *node* yang tidak *valid*, atau *node* sudah mencapai titik terdalam tetapi tidak menemukan *goal node*. Maka lakukan perintah *move* ke *sibling* dari anak yang merupakan *node* yang tidak *valid* tersebut.




**Gambar 3.4** *moveSibling*

6. Apabila semua node telah dicek validasinya dalam satu parent, sedangkan tidak ada lagi simpul bawahnya yang bernilai *valid*, maka pada kondisi tersebut dilakukan proses *backtracking*, yaitu rekursif naik ke level yang paling rendah. Iterasi selanjutnya dilakukan dengan mengecek *node sibling* dari *parent*. Pergerakan ini diberi sebutan *levelUp*




**Gambar 3.5** *levelUP*

7. *Rekursif* dilakukan hingga tidak ada node lain lagi yang dicek, Setelah tidak ada lagi yang dicek, maka sampailah ke *goal node*.

Dengan mengikuti langkah-langkah yang dilkukan di atas, maka permasalahan *knight's tour* yang ada dapat diselesaikan dengan waktu yang dibutuhkan lebih baik daripada menggunakan algoritma *brute force*. Pengecekan langkah dengan melakukan pendekatan pohon merentang ini memudahkan dalam implementasi program dalam penerapan algoritma *Backtrack* ini.

Berikut merupakan mekanisme *backtrack* pada *knight's tour* pada umumnya :

```
If all squares are visited
    print the solution
Else
   a) Add one of the next moves to solution vector and recursively
   check if this move leads to a solution. (A Knight can make maximum
   eight moves. We choose one of the 8 moves in this step).
   b) If the move chosen in the above step doesn't lead to a solution
   then remove this move from the solution vector and try other
   alternative moves.
   c) If none of the alternatives work then return false (Returning false
   will remove the previously added item in recursion and if false is
   returned by the initial call of recursion then "no solution exists" )
```

*Pseudo Code* yang dibuat :

```
Inisialisasi
    maxnode -> mxn
    level -> 0
    data[1] -> 0
    maxLangkah -> 8

While data[1] < maxLangkah do
    if levelDown then
        levelDown =false; level += 1
    else if moveSiblings then
        moveSiblings = false; data[level]++;
            if data[level] >= maxLangkah then
            levelUp = true; goto end while;
        endif;
    else if moveUp then
        levelUp = false ; level--; moveSiblings=true; goto end while;
    endif;

    valid = cek(data[level]);
    if valid then
        if level = maxNode-1 then tour
        else move
        endif;
    else levelDown = true
    endif
end while
end.
```

Tidak semua *knight's tour* dengan ukuran papan catur yang bervariasi bisa diselaikan dengan semua papan catur terinjak 1 kali oleh knight yang ada. Mari kita lihat pembuktiannya pada Implementasinya pada program yang dijalankan.

## IV. IMPLEMENTASI PADA PROGRAM

Implementasi pada papan 8x8:
*Open knight's tour* :
Dengan *random* posisi awal dari *knight*.

*Output* :

```
33 50  5 62 35 52  7 54
 4 59 34 51  6 55 26 37
49 32 61 58 63 36 53  8
60  3 48 41 56 27 38 25
31 46 57 64 39 42  9 18
 2 13 40 47 28 19 24 21
45 30 15 12 43 22 17 10
14  1 44 29 16 11 20 23
```

**Gambar 4.1** *Output* 1

*Output* dengan awal dari posisi *knight* yang berbeda

```
61 20  3 22 47 56  5 52
 2 23 60 55  4 53 48 57
19 62 21 46 59 64 51  6
24  1 36 63 54 43 58 49
35 18 45 42 31 50  7 40
28 25 32 37 44 41 10 13
17 34 27 30 15 12 39  8
26 29 16 33 38  9 14 11
```

**Gambar 4.2** *Output* 2

Penerapan pada *knight's tour* pada papan catur 8x8 selalu menghasilkan solusi walaupun posisi *knight* ditaruh di posisis manapun.

Bagaimana dengan waktunya ?

```
Waktu yang dibutuhkan : 12839 ms
```

Waktu yang dibutuhkan untuk menyelesaikan persoalan *knight's tour* dengan menggunakan algoritma *backtrack* adalah berkisar 10000 ms hingga 13000 ms.

Berikutnya percobaan dengan menggunakan papan catur yang lebih luas lagi.
Diambil 1 *output* , dari papan catur berukuran 10x10 untuk melihat perbedaan waktu yang dibutuhkan untuk menyelesaikannya.

```
24 65 62 55 26 95 72 53 28  3
63 56 25 94 71 54 27  2 37 52
66 23 64 61 96 73 98 51  4 29
57 60 93 70 99 50  1 38 75 36
22 67 58 89 78 97 74 43 30  5
59 92 69 100 49 88 39 76 35 42
68 21 86 79 90 77 44 41  6 31
85 18 91 48 87 40 81 34  9 12
20 47 16 83 80 45 14 11 32  7
17 84 19 46 15 82 33  8 13 10

Waktu yang dibutuhkan : 16852 ms
```

**Gambar 4.3** *Output* 10x10

Bagaimana waktu bila diberikan matriks yang banyak

Contoh 50x50 disini :

**Gambar 4.4** output 50x50

Ternyata sangat banyak sekali, tetapi waktu yang dibutuhkan adalah :

```
Waktu yang dibutuhkan : 100698 ms
```

Hanya 100 *second*., ini membuktikan bahwa algotirma backtrack ini cukup mangkus.

## V. KESIMPULAN

Banyak algoritma untuk menyelesaikan permasalahan *knight's tour*, salah satunya yaitu menggunakan algoritma *backtracking*, dimana dalam penyelesaiannya, langkah knight pada papan catur yaitu berbentuk *letter* L memiliki kemungkinan 8 langkah dalam setiap kali jalannya. Dengan merepresentasikan 8 langkah tersebut dengan menggunakan pohon merentang, pencarian solusi dengan metodologi *backtracking* menjadi memungkinkan, pencarian solusi ini memang bukan merupakan algoritma yang paling optimum tetapi dengan menggunakan algoritma ini waktu penyelesaian menjadi lebih baik debandingkan dengan menggunakan algoritma *brute-force*. Persoalan yang terlihat menarik ini akan menjadi sangat menarik bila diselesaikan dengan algoritma backtracking.

## VII. ACKNOWLEDGMENT

Ucapan terima kasih saya ucapkan kepada Pak Rinaldi Munir dan Ibu Masayu atas segala bimbingannya dalam slide presentasi dan diktat untuk mengerjakan makalah ini. Terima kasih turut diberikan kepada teman-teman yang telah membantu memberikan ide, dan ikut turut dalam memberikan referensi-referensi yang saya dapat sehingga saya dapat menyelesaikan makalah ini.

## REFERENCES

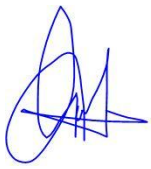
[1] Munir, Rinaldi , Diktat Kuliah Strategi Algoritma IF-2011, Program Studi Teknik Informatika, STEI, ITB
[2] Slide Presentasi, Bactracking atau Algoritma Runut Balik Tahun Ajaran 2013/2014
[3] http://rosettacode.org/wiki/Knight's_tour.
Tanggal Akses : 19 Desember 2013
[4] http://tgsourcecode.blogspot.com/2012/09/a-knights-tour.html.
Tanggal Akses : 19 Desember 2013
[5] http://www.cs.xu.edu/csci220/01f/project1.html
Tanggal Akses : 19 Desember 2013
[6] http://www.geeksforgeeks.org/backtracking-set-1-the-knights-tour-problem/
Tanggal Akses : 19 Desember 2013

## 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, 20 Desember 2013

ttd



Adhika Aryantio
13511061