Penerapan Algoritma *Backtracking* pada *Knight's Tour Problem*

Mahdan Ahmad Fauzi Al-Hasan - 13510104
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Mahdan.ahmad@s.itb.ac.id

***Abstract* — *Knight* adalah sebuah bidak pada permainan catur. Dengan menggunakan bisa dan papan catur tersebut, terciptalah *Knight's Tour*, suatu *puzzle* matematis yang meminta kita menggerakkan bidak tersebut, dengan gerakan membentuk huruf L, tepat 1 kali untuk setiap petak dalam papan permainan tersebut. *Puzzle* ini sudah menjadi permainan yang menarik perhatian khalayak ramai dari masa ke masa. Perkembangan permainan ini pun memaksa terciptanya berbagai macam bentuk papan, maupun solusi yang diberikan. Solusi dari permainan ini telah tersedia dalam berbagai macam algoritma, salah satu algoritma yang bisa digunakan untuk menyelesaikan permainan ini adalah algoritma *Backtracking*. Solusi dari permainan ini menggunakan algoritma tersebut lah yang akan dibahas pada makalah ini**

Index Terms* — *Knight's Tour, Backtracking, Puzzle, Chess.

I. PENDAHULUAN

A. *Knight's Tour*

Knight (♘ ♞, dalam bahasa Indonesia biasa disebut kuda) adalah sebuah bidak pada permainan catur. Dalam permainan catur sendiri, gerakan *knight* mempunyai gerakan yang paling unik. *Knight* bergerak dengan membentuk huruf L, dan tidak seperti bidak catur yang lain, *knight* bisa melompati bidak-bidak yang ada di jalur L nya.

Knight's tour adalah suatu *puzzle* yang telah menyita perhatian para pemain catur dari masa kemasa. Dengan menggunakan bidak *knight* dan papan catur, pemain diminta untuk menggerakkan knight dengan ketentuan knight tersebut harus bisa melewati setiap kotak sebanyak 1 kali saja, dengan gerakan *knight* membentuk huruf L tentunya.

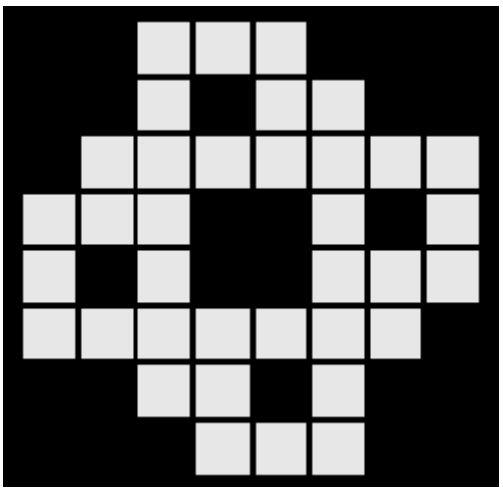
Solusi dari *knight's tour* yang pertama kali tercatat dilakukan pada papan catur berukuran 8 x 8. Terutulis dalam manuskrip arab berjudul *"Nuzhat al-albab al-'aqulfi'sh-shatranj al-manqul" (The delight of the intelligent, a description of chess)*, ditulis oleh al-Adli ar-Rumi, seorang pemain catur dari bagdad, sekitar tahun 840 M. Sedangkan untuk *formal study* dari permainan ini dimulai oleh Euler pada tahun 1759 menggunakan papan catur standar berukuran 8 x 8.

Berdasarkan hasil akhirnya, *knight's tour* dibagi menjadi 2 jenis:

1. *Open Knight's Tour*
   Sebuah *Knight's Tour* dikatakan *Open* atau terbuka bila pada langkah terakhirnya, bidak *knight* bisa kembali ke posisi awalnya.

2. *Close Knight's Tour*
   Sebaliknya, sebuah *Knight's Tour* dikatakan *Close* atau tertutup bila pada langkah terakhirnya, bidak *knight* tidak bisa kembali ke posisi awalnya.

Dalam perkembangannya *knight's tour* telah menjadi *puzzle* yang menarik bagi khalayak ramai, *puzzle* ini tidak hanya dicoba pada papan catur biasa (8x8), *knight's tour* sudah banyak dicoba di papan berbentuk persegi (n x n) maupun persegi panjang (m x n), bahkan dicoba pula di papan yang tidak beraturan, seperti pada gambar di bawah ini.



Gambar 1 – Papan Aneh Pada *Knight's Tour*

Secara matematis, *Knight's Tour Problem* ini bisa diubah ke dalam bentuk grafik seperti dalam Persoalan Hamilton. Petak-petak pada papan akan membentuk simpul untuk graf, dan sisi-sisinya akan menunjukkan gerakan-gerakan legal dari bidak k*night* itu sendiri. Sehingga bisa dikatakan *Knight's Tour Problem* adalah persoalan Hamilton dalam permainan catur.

B. Algoritma *Backtracking*

Algoritma Runut-balik (*backtracking*) adalah strategi pemecahan masalah yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus.

Backtracking merupakan perbaikan dari algoritma *Brute Force*, yang secara sisteatis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dengan metode *backtracking*, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya perlu mempertimbangkan proses pencarian yang sudah mengarah ke solusi saja. Oleh karena itu, waktu pencarian menjadi lebih cepat.

Algoritma *Backtracking* telah banyak diterapkan untuk berbagai macam hal seperti:

- *Games (tic-tac-toe, maze-based game)*
- Berbagai permasalahan pada catur (*eight queens puzzle, Knigth's Tour Problem*)
- Permasalahan dalam *parsing* seperti *knapsack problem*
- *Puzzle* seperti Sudoku, maupun *crossword puzzle*
- dll.

Langkah-langkah umum dalam pencarian solusi menggunakan algoritma *Backtraking* bisa ditulis seperti berikut:

a. Solusi dicari dengan membentuk lintasan dari akar ke daun. Pembentukan lintasan yang dipakai menggunakan metode *Depth-First-Search* atau DFS. Simpul-simpul yang sudah dibangkitkan disebut simpul hidup *(Live node)*, Sedangkan simpul hidup yang sedang diperluas disebut simpul-E (*Expand-node)*. Simpul diberi nomer sesuai urutan kelahirannya.

b. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut akan dibunuh sehingga menjadi simpul mati *(Dead node)*. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas *(Bounding Function)*. Simpul yang sudah mati tidak akan pernah diperluas lagi.

c. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak ayng dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik *(Backtrack)* ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.

d. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

Secara umum penerapan algortima ini bisa dilakukan dengan cara *rekursif* maupun *iteratif*

II. PENERAPAN *BACKTRACKING* PADA *KNIGHT'S TOUR PROBLEM*

Ide algoritma *backtracking* yang dipakai untuk memecahkan permasalahan dari *knight's tour problem* ini tidak jauh berbeda dengan algoritma lain untuk memecahkan permasalahan ini.

Algoritma untuk memecahkan *knight's tour problem* secara umum akan berbentuk seperti berikut:

```
while ada simpul yang belum dicoba
{
   NEXT_TOUR();
   PRINT()
   if semua petak terisi
   {
      Ada Solusi
      PRINT();
   } else
   {
      Tidak ada solusi
   {
}
```

Sedangkan bila dicoba dengan menggunakan algoritma *backtracking* algoritmanya akan berbentuk seperti berikut (*rekursif*):

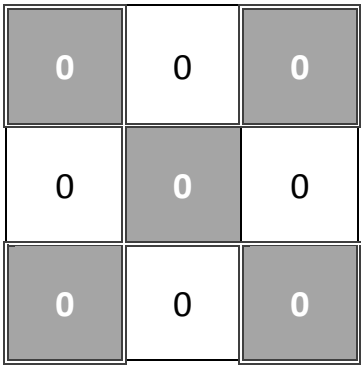
```
If semua petak terisi
  PRINT()
  return true
Else
{
  NEXT_TOUR()
  PRINT()
  if jalan memunculkan solusi
     Backtrak() //rekursif
     return true
  else
     remove jalan

  return false
}
```

Seperti itulah gambaran algoritma *backtracking* untuk memecahkan *knight's tour problem* ini. Prosedur tersebut akan mengeluarkan Boolean. Bila *true* berarti *knight's tour problem* tersebut mempunyai solusi, bila *false* berarti tidak. Ada atau tidaknya solusi dari *knight's tour problem* dilihat dari bentuk dan ukuran papan permainan itu sendiri.

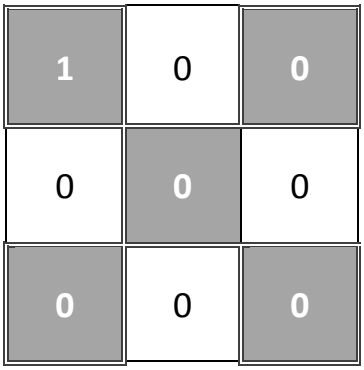
Dengan menggunakan algoritma diatas, maka bila kita melakukan percobaan pada papan permainan dengan ukuran 3 x 3, program akan berjalan sesuai ilustrasi berikut:

State awal, inisialisasi papan solusi:



Gambar 2. State 0

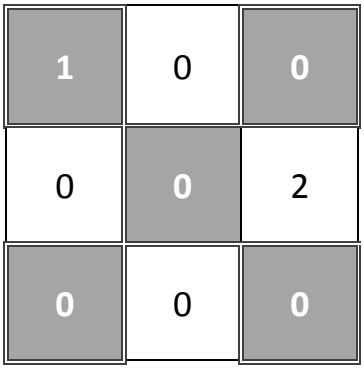
State selanjutnya kuda ditempatkan pada titik (1, 1)



Gambar 3. State 1

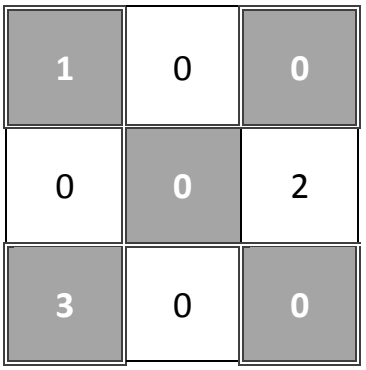
Pada proses NEXT_TOUR() program akan membuat daftar titik mana saja yang bisa dilaluli oleh *knight*, kemudian memilih salah satu.

Dalam kondisi ini, titik yang mungkin dilalui adalah titik (2, 3) dan titik (3, 2). Program akan memilih satu titik, dan titik yang terpilih adalah titik (2, 3). Jika langkah pada titik ini tidak menemukan solusi, maka program akan melakukan *backtrack* dan mencoba melewati titik (3, 2). Kemudian program akan menandai titik (1, 1) sudah terlewati.



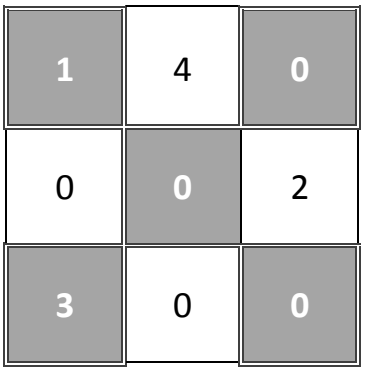
Gambar 4. State 2

Dalam state ini, program kembali melakukan pengecekan rute selanjutnya melalui proses NEXT_TOUR dan menemukan kondisi terdapat 2 kemungkinan gerakan: (1, 1) dan (3, 1). Namun, karena titik (1, 1) telah ditandai sudah dilewati, maka, program akan langsung memilih titik (3, 1) dan menandainya.



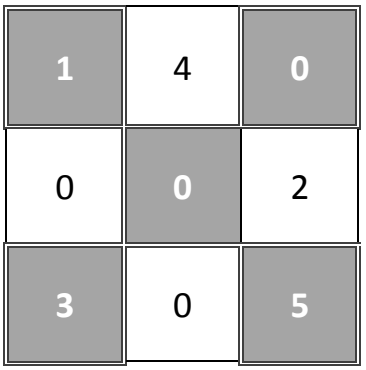
Gambar 5. State 3

Dengan cara yang sama, kita menemukan bahwa di state ini dapat ditemukan 2 jalan yang mungkin, (2, 3) dan (1, 2). Karena (2, 3) sudah terisi, maka proses akan berlanjut ke titik (1, 2) dan menandainya.



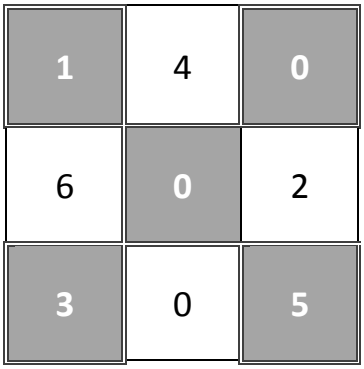
Gambar 6. State 4

Dengan cara yang sama, kita menemukan bahwa di state ini dapat ditemukan 2 jalan yang mungkin, (3, 1) dan (3, 3). Karena (3, 1) sudah terisi, maka proses akan berlanjut ke titik (3, 3) dan menandainya.



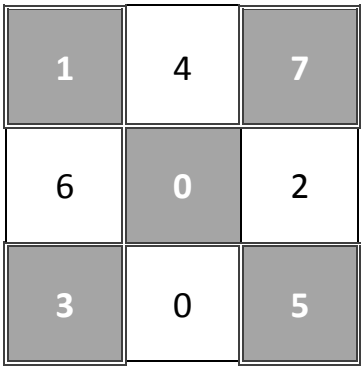
Gambar 7. State 5

Dengan cara yang sama, kita menemukan bahwa di state ini dapat ditemukan 2 jalan yang mungkin, (2, 1) dan (1, 2). Karena (1, 2) sudah terisi, maka proses akan berlanjut ke titik (2, 1) dan menandainya.



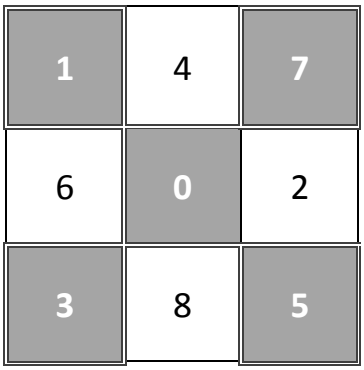
Gambar 8. State 6

Dengan cara yang sama, kita menemukan bahwa di state ini dapat ditemukan 2 jalan yang mungkin, (3, 3) dan (1, 3). Karena (3, 3) sudah terisi, maka proses akan berlanjut ke titik (1, 3) dan menandainya.



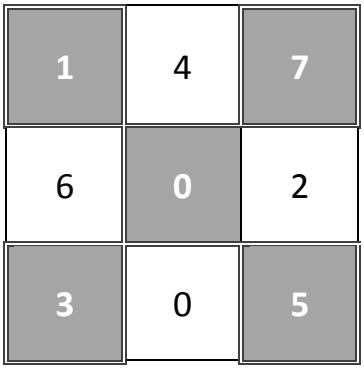
Gambar 9. State 7

Dengan cara yang sama, kita menemukan bahwa di state ini dapat ditemukan 2 jalan yang mungkin, (2, 1) dan (3, 2). Karena (2, 1) sudah terisi, maka proses akan berlanjut ke titik (3, 2) dan menandainya.



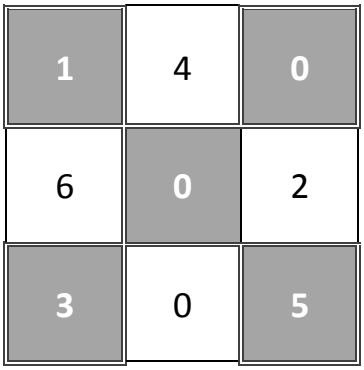
Gambar 10. State 8

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (3, 2).



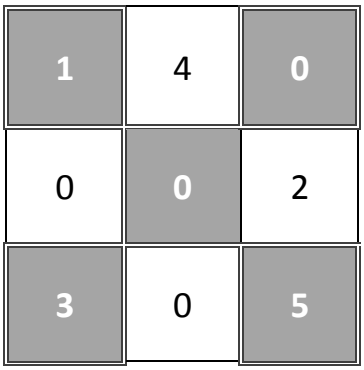
Gambar 11. State 9

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (1, 3).



Gambar 12. State 10

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (2, 1).



Gambar 13. State 11

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (1, 3).

1	4	0
0	0	2
3	0	0

Gambar 14. State 12

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (1, 2).

1	0	0
0	0	2
3	0	0

Gambar 15. State 13

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (3, 1).

1	0	0
0	0	2
0	0	0

Gambar 16. State 14

Pada proses NEXT_TOUR() tidak ditemukan adanya kemungkinan langkah, oleh karena itu, program akan melakukan *backtrack* dengan meremove langkah terakhir, yaitu titik (2, 3).

1	0	0
0	0	0
0	0	0

Gambar 17. State 15

Pada proses NEXT_TOUR() akhirnya ditemukan adanya kemungkinan langkah, yaitu pada titik (3, 2). Proses akan berlanjut pada titik tersebut. Kemudian, seperti biasa menandai titik tersebut.

1	0	0
0	0	0
0	2	0

Gambar 18. State 16

Sesuai yang telah ditulis di awal proses, bila program tidak menemukan solusi melewati titik (2, 3), program akan melakukan backtrack dan mencoba jalur lain, dalam hal ini titik (2, 1). Proses akan berlangsung kembali dengan langkah2 diatas, dan kemungkinan terburuknya bila tidak ditemukan solusi dari semua langkah, maka program akan mengeluarkan *output message* "tidak ada solusi".

III. KESIMPULAN

Algortitma *backtracking* adalah salah satu algoritma yang lebih baik daripada algoritma *brute force* walaupun pada kenyataannya, bila tidak ditemukan solusi, maka algoritma ini akan mencoba semua kemungkinan layaknya *brute force. Knight's tour problem* sendiri ada suatu persoalan yang unik dan mengasikkan untuk dicoba, terdapat berbagai macam cara penyelesaian dari problem ini, salah satunya adalah dengan menggunakan algoritma *backtracking.* Pada tulisan ini telah ditunjukkan pengaplikasian algoritma *backtracking* untuk memecahkan suatu permasalahan.

REFERENCES

[1] I. Parberry, "An Efficient Algorithm for the Knight's Tour Problem", Discrete Applied Mathematics, University of North Texas, Vol. 73, pp. 251-260, 1997.

[2] O. Kyek, I. Panberry, I Wegener. “Bounds on the number of knight’s tours”. Discrete Applied Mathematics, University of North Texas, Vol. 74, pp. 171-181, 1997.

[3] B. Hill, K Tostado. “Knight’s Tours”. 2004

[4] Munir, Rinaldi. “Diktat Kuliah IF2251 Strategi Algoritmik”, Program Studi Teknik Informatika STEI ITB, 2009.

[5] http://www.archimedes-lab.org/knight_tour.html
Tanggal akses: 20 Desember 2012 pukul 10.45

[6] http://www.geeksforgeeks.org/archives/12916
Tanggal akses: 20 Desember 2012 pukul 22.30

[7] http://cerebro.cs.xu.edu/csci220/01f/project1.html
Tanggal akses: 20 Desember 2012 pukul 22.30

[8] http://en.algoritmy.net/article/39907/Knights-tour
Tanggal akses: 21 Desember 2012 pukul 09.30

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, 21 Desember 2012

Mahdan Ahmad Fauzi Al-Hasan
13510104