ISSN 2541-0350

Volume 1, No. 1

2013

JURNAL KOMPUTASI

Ilmu Komputer
Universitas Lampung
Bandar Lampung, April 2013

Hal. 1 - 90

VOL 1, NO 1 (2013)

TABLE OF CONTENTS

ARTICLES

ISSN: 2541-0350

EDITORIAL TEAM

EDITORS

SECTION EDITOR

ISSN: 2541-0350

JOURNAL CONTACT

PRINCIPAL CONTACT

Rizky Prabowo
Pengelola Jurnal
Jurusan Ilmu Komputer FMIPA Universitas Lampung

Jurusan Ilmu Komputer

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Lampung

Jl. Sumantri Brojonegoro No. 01 Bandar Lampung

35145

Phone: +6285840180508
Email: rizky.prabowo@fmipa.unila.ac.id

SUPPORT CONTACT

Muhammad iqbal
Phone: +6281284387257
Email: iqdwita@gmail.com

ISSN: 2541-0350

PEOPLE

REVIEWER

Admi Syarif, Unila, Indonesia

Yeni Herdiyeni, IPB, Indonesia

Rifki Sadikin, LIPI, Indonesia

Tristiyanto Tristiyanto, Unila, Indonesia

Esa Prakarsa, LIPI, Indonesia

Faisal Makhrus, UGM, Indonesia

Edi Wibowo, UGM

Sri Wahyuni, IPB, Indonesia

Suhendro Yusuf Irianto, Darmajaya, Indonesia

ISSN: 2541-0350

Implementasi Algoritma Backtrack untuk Pencarian Solusi Knight's Tour Problem pada Papan Catur *m x n*

[1]Wamiliana, [1]Dian Kurniasari, [2]Dolly Yudhistira

[1] Jurusan Matematika FMIPA Universitas Lampung
[2] Jurusan Ilmu Komputer FMIPA Universitas Lampung

Abstract

This research discusses about finding all solutions of The Knight's Tour Problem. The steps done by a knight on a chess board forms a path. If the track is able to pass through all points (boxes) and can return to the original point (box) is called as Hamilton Cycle, that produces Closed Knight's Tour; if the path can pass through all the points but can not return to the original point of the track is called as Open Knight's Tour. There are various ways to solve the Knight's Tour Problem, one of which is to use the Backtrack algorithm. In this paper we will discuss of finding all solutions of Knight's Tour Problem vary from 3x3 up to 8x8 chessboards.

Keywords: *knight's tour problem, backtrack algorithm, open and closed knight's tour*

1 Pendahuluan

Catur sudah lama sekali dikenal oleh masyarakat dunia dan dimainkan dari semua kalangan baik itu anak-anak, remaja, dewasa dan para manula. Permainan catur ini dimainkan oleh dua orang dalam suatu papan bujur sangkar yang terdiri dari delapan kolom dan delapan baris, yang berbentuk kotak dan pada umumnya berwarna hitam dan putih secara bergantian. Permainan catur terdiri dari satu raja, satu ratu, dua benteng, dua peluncur, dua kuda dan delapan bidak pada masing-masing pemainnya. Permainan catur ini banyak dimainkan karena selain menarik dan menantang tapi juga mampu mengasah strategi dan logika berpikir.

Berawal dari dasar permainan catur ini, banyak orang kemudian menciptakan permainan baru, salah satunya adalah *Knight's Tour*. *Knight's Tour* adalah permainan catur yang hanya menggunakan satu bidak kuda catur. Aturan dari *Knight's Tour* ini sederhana sekali, yaitu dapat melewati seluruh kotak papan catur tepat satu kali dengan langkah dari bidak kuda tersebut harus membentuk huruf "L". Langkah kuda tersebut dapat berupa melangkah dua kotak ke arah horizontal kemudian satu kotak ke arah vertikal atau melangkah satu kotak ke arah horizontal kemudian dua kotak ke arah vertikal dan sebaliknya sehingga terus membentuk huruf "L".

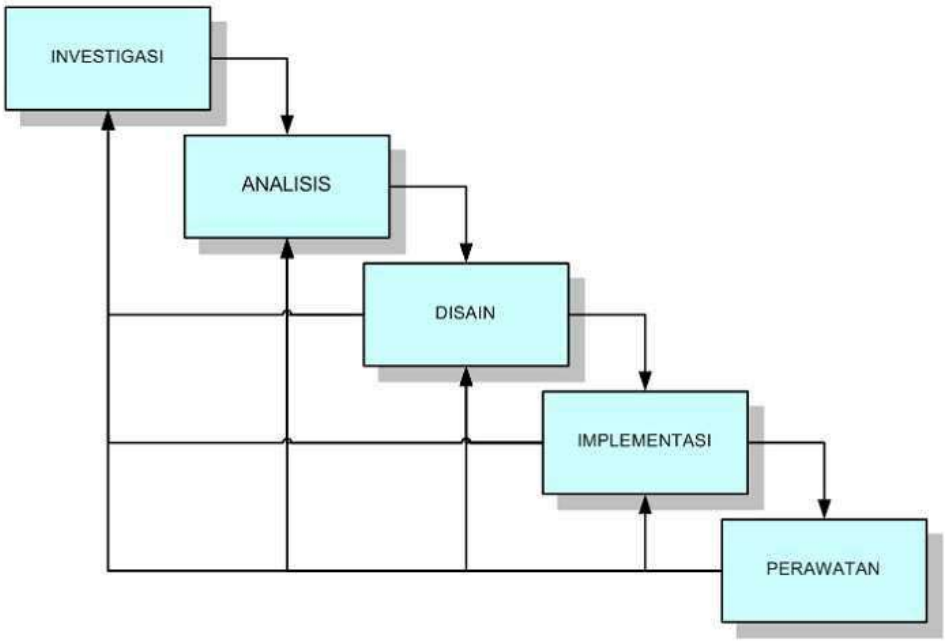
Bila dilihat secara matematis permainan *Knight's Tour* ini membentuk suatu graf, karena terdapat kotak-kotak sebagai *node* dan langkah kuda catur sebagai *edge*. Sehingga, bila *node* dan *edge* itu dihubungkan akan membentuk suatu lintasan. Menurut teori graf, siklus yang menggunakan semua titik dan kembali ke titik semula dikenal dengan siklus Hamilton (*Hamilton cycle*). Sedangkan jika semua titik dilewati tepat satu kali tetapi tidak kembali ke titik semula disebut jalur Hamilton (*Hamilton path*).

Banyak peneliti yang telah melakukan mengenai permasalahan *Knight's Tour* ini, baik peneliti dari dalam negeri maupun luar negeri. Berbagai macam metode dan algoritma telah digunakan untuk memecahkan permasalahan *Knight's Tour* ini, salah satunya adalah Sulistyono [1]. Dalam makalahnya tersebut dipaparkan bagaimana masalah *Knight's Tour* dapat diselesaikan dengan algoritma *Backtrack*.

Tujuan dari penelitian ini adalah menerapkan algoritma *Backtrack* untuk menemukan semua solusi yang mungkin dari *Knight's Tour Problem* pada papan catur *m x n*. Selain itu, pada penelitian ini akan membuat suatu program yang dapat untuk menemukan solusi *Knight's Tour Problem.*

2 Metode

Metode yang digunakan dalam pelaksanaan penelitian ini adalah *the waterfall model*. *The waterfall model* memiliki lima tahapan utama yaitu Investigasi, Analisis, Disain, Implementasi dan Perawatan. Metode ini disebut *waterfall* (air terjun) karena memang diagram tahapan prosesnya mirip dengan air terjun yang bertingkat [2]. Berikut gambaran tahapan metode waterfall seperti yang tampak pada Gambar 1.



Gambar 1. *The Waterfall Model*

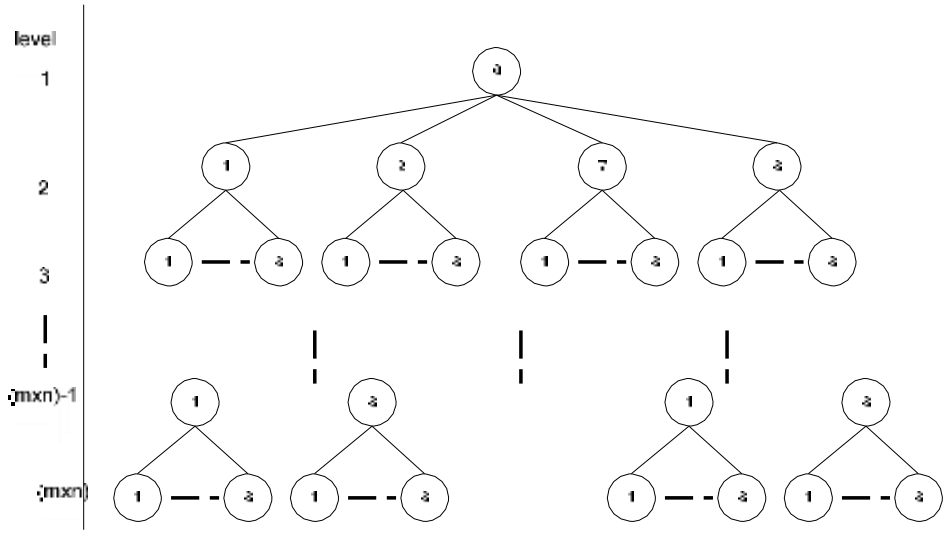
3 Pembahasan

3.1 Implementasi Lintasan Catur dalam Graf

Solusi untuk permasalahan *Knight's Tour* merupakan salah satu bentuk dari bentuk jalur Hamilton. Tiap langkah *knight* pada papan catur digambarkan sebagai suatu simpul yang membentuk suatu lintasan. Bila lintasan tersebut dapat melewati semua titik dan dapat kembali ke titik semula disebut siklus Hamilton (*Hamilton Cycle)* sehingga akan menghasilkan solusi *Closed Knight's Tour*. Sedangkan, jika lintasan tersebut dapat melewati semua titik tetapi tidak dapat kembali ke titik semula disebut lintasan Hamilton (*Hamilton Path*) maka akan menghasilkan solusi *Open Knight's*.

Suatu *knight* berjalan ke posisi selanjutnya ke arah A(x+2,y+1), (x-2,y+1), (x+2,y-1), (x-2,y-1) atau A(x+1,y+2), (x-1,y+2), (x+1,y-2), (x-1,y-2) yang akan membentuk lintasan berbentuk huruf L. Dengan demikian, *knight* memiliki maksimum 8 kemungkinan arah untuk tiap langkahnya.

Dengan menggambarkan titik awal *knight* sebagai simpul awal dari lintasan, dan langkah- langkah yang mungkin sebagai simpul selanjutnya, maka arah lintasan *knight* dapat digambarkan sebagai suatu pohon graf dengan jumlah simpul maksimum dari satu lintasan adalah *m x n*.



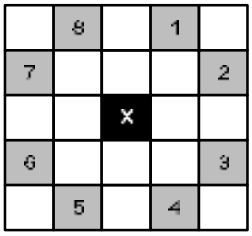
Gambar 2. Pohon graf arah langkah kuda pada permainan *Knight's Tour*

3.2 Implementasi Algoritma Backtrack

Suatu permainan *knight's tour* dapat diselesaikan dengan menggunakan algoritma *Backtrack*. Algoritma *Backtrack* sendiri merupakan suatu algoritma yang termasuk ke dalam algoritma *depth-first-search* yang merupakan salah satu bentuk dari metode pencarian *brute-force*.

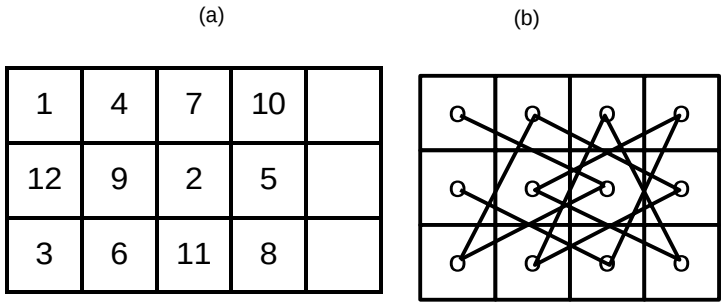
Dalam bentuk graf, langkah-langkah kuda pada papan catur digambarkan sebagai suatu *tree* dimana setiap simpul *tree* tersebut memiliki 8 buah *child*. Kedalaman dari *tree* tersebut ditentukan oleh banyaknya langkah terbanyak yang dapat dilalui oleh kuda (*m* x *n* langkah). Gambar 2 menunjukkan bentuk pohon graf langkah kuda pada permainan *knight-tour* dalam papan catur *m x n*.

Gambar 3 menunjukkan kedelapan arah langkah kuda. Kotak berwarna hitam dengan tanda 'X' merupakan titik awal kuda, dan tiap langkah arah langkah kuda di representasikan dengan nomor 1 – 8.



Gambar 3. Arah langkah kuda pada permainan catur

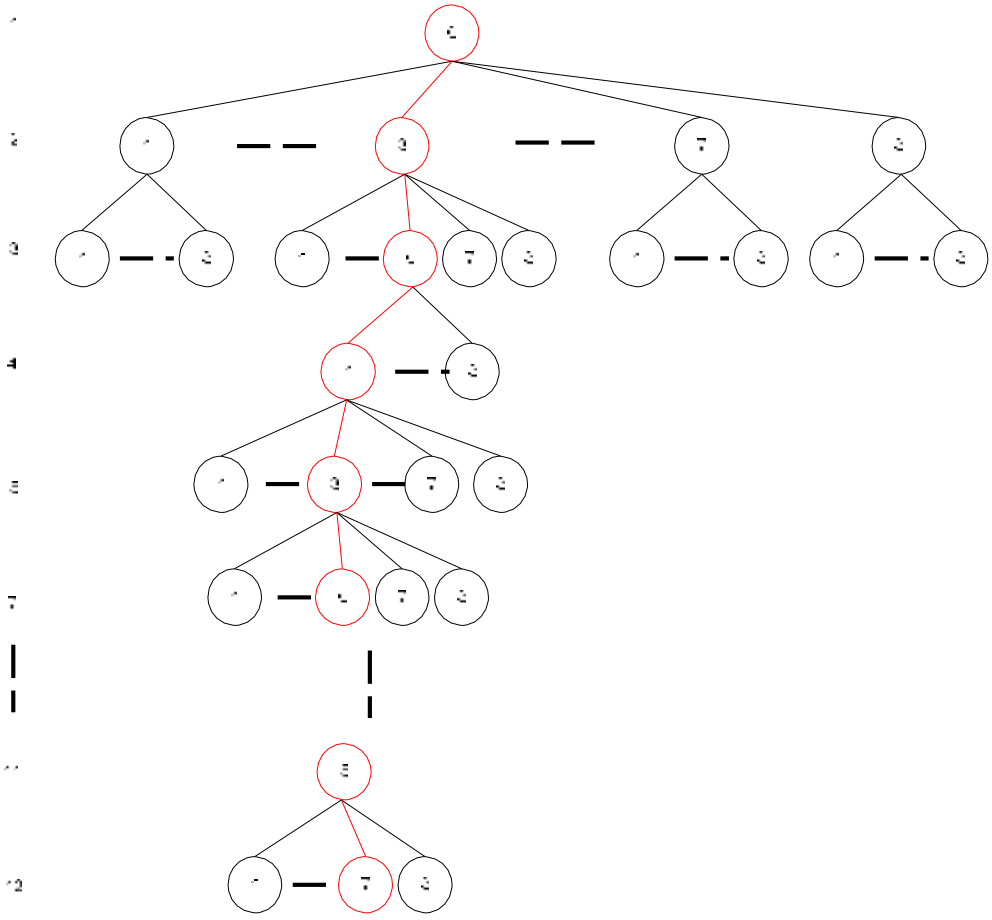
Pada Gambar 4 digambarkan suatu *open knight's tour*. Pada Gambar 4(a) angka-angka pada kotak merupakan urutan langkah yang ditempuh oleh kuda, dimana angka 1 menunjukkan langkah pertama *knight* dan angka 12 menunjukkan langkah ke-12 *knight*.



Gambar 4 (a) *Open Knight Tour* direpresentasikan dalam angka. (b) *Open Knight Tour* direpresentasikan dalam garis.

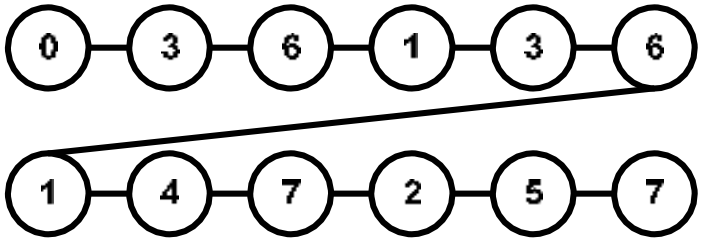
Apabila tiap langkah pada papan tersebut dibuat dalam bentuk grafik seperti pada Gambar 4 (b), berdasarkan arah langkah papan catur pada Gambar 3, maka pergerakan langkah kuda dari langkah pertama dapat di tuliskan dengan : 0-3-6-1-3-6-1-4-7-2-5-7

Apabila digambarkan pada *tree* maka akan tampak pada Gambar 5 :



Gambar 5. Solusi *Open Knight's Tour* pada *tree*

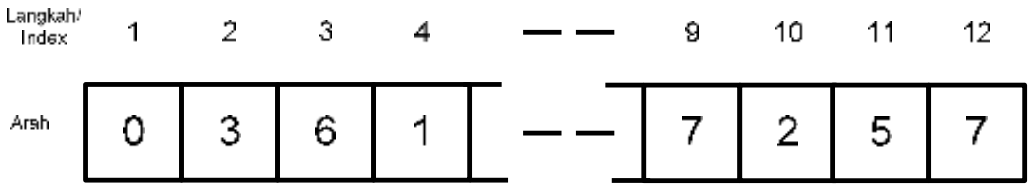
Berdasarkan graf pada Gambar 5, maka dapat diambil simpul-simpul yang merupakan simpul arah dari langkah kuda sebagai berikut:



Gambar 6. Simpul arah langkah kuda

Pada graf tersebut, *node* pertama merupakan langkah pertama dari *knight tour*. Langkah pertama tersebut diberi nilai 0 karena langkah pertama merupakan langkah yang telah di tentukan. Pada *node* kedua atau langkah kedua, *node* bernilai 3 yang berarti, berdasarkan papan catur pada Gambar 3, kuda tersebut bergerak dengan arah 2 langkah ke kanan dan 1 langkah ke bawah. Begitu seterusnya pergerakan catur hingga langkah atau *node* ke-12 yang bernilai 7.

Nilai-nilai *node* pada graf tersebut dapat direpresentasikan kedalam suatu larik *array* bertipe data *integer* dengan panjang *node* adalan *mxn* sebagai berikut:
Panjang *node*: *mxn* =4x3=12

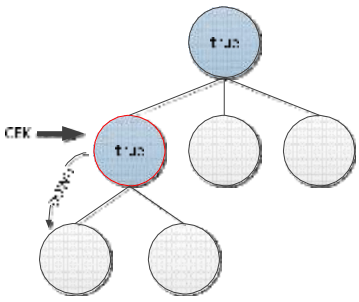


Gambar 7. Representasi simpul pada *array*

Pada Gambar 7, nilai *index* ke-*i* merepresentasikan arah langkah ke-*i* pada papan catur, *index* ke-(*i*-1) merepresentasikan *node parent* dan *index* ke(*i*+1) merepresentasikan nomor *child*.

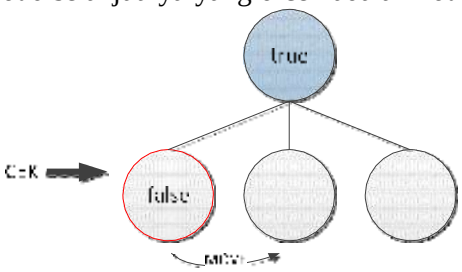
Proses pencarian solusi dengan *backtrack* diuraikan sebagai berikut :

1) Proses dimulai dengan menginisialisasi langkah pertama kuda, dimana langkah tersebut di tentukan sebagai *node* pertama pada *tree*. Langkah pertama di tentukan sebagai langkah *valid*, selanjutnya program akan mengecek *child* dari *node* pertama. *Child* dari *node* pertama berada pada *level* 2, yaitu berarti pengecekan naik 1 *level*, atau pada *tree* berarti *node* lebih dalam 1 tingkat. Karena pengecekan selanjutnya harus naik *level* (simpul turun), maka *flagDown* diinisiasikan dengan nilai *true*.
2) Proses selanjutnya dilakukan suatu rutin dimana rutin tersebut akan melakukan pengecekan untuk tiap-tiap *node* selanjutnya. Proses pengecekan tersebut diuraikan sebagai berikut:
   a) Program akan mengecek kondisi dari pengecekan sebelumnya yang direpresentasikan sebagai nilai *flag* (*flagDown, flagMove, flagUp*). Apabila pada iterasi sebelumnya *node* yang dicek adalah *valid* dan *node* tersebut bukan *node* terbawah (*level* tertinggi), maka iterasi berikutnya yaitu mengecek *child* dari *node* tersebut, yang berarti iterasi selanjutnya akan mengecek *node* yang lebih bawah atau *level* lebih tinggi, yang berarti *flagDown* akan diaktifkan. Pada pengecekan *flagDown*, *node child* yang di cek dimulai dari *child* sesuai urutan langkah. Maka, pada iterasi dengan *flagDown*, nilai *node* diinisialisasi menjadi 1 atau langkah pertama (pada *array* yang dimulai dari 0, langkah direpresentasikan dari 0-7, sehingga inisialisasi awal adalah 0).



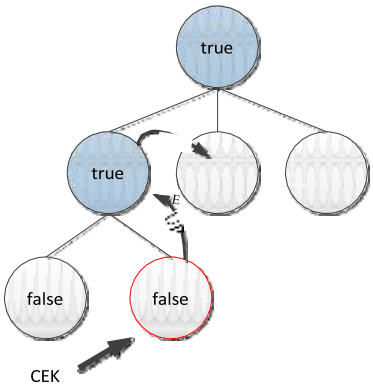
Gambar 8. *Flag Down*

b) Pada iterasi ke-*i*, apabila suatu *node* yang di cek *valid*, namun *node* tersebut adalah *node* terdalam atau *level* tertinggi, berarti kuda telah mencapai kondisi '*tour*', yaitu semua kotak telah dilewati. Pada kondisi ini tidak mungkin lagi untuk mengecek *child* dari *node* tersebut. Karena *node* tersebut tak memiliki *child* lagi, maka yang dilakukan pada iterasi selanjutnya yaitu mengecek *sibling* (*node* pada kedalaman yang sama atau *level* sama dan *parent* yang sama) dari *node* tersebut. Karena pengecekan selanjutnya dilakukan pada *node* dengan *level* yang sama, keadaan ini disebut pindah atau geser, sehingga diaktifkan *flagMove* pada kondisi ini. Kondisi geser juga dapat ditemui apabila *node* yang dicek adalah tidak *valid*, sehingga *node* selanjutnya yang dicek adalah *node sibling* dari *node* tersebut.



Gambar 9. *Flag Move*

c) Apabila semua *node child* dalam satu *parent* telah dicek nilai validitasnya, sedangkan tidak ada lagi simpul-simpul di bawahnya yang bernilai *valid*, maka pada kondisi tersebut dilakukan proses *backtracking*, yaitu rutin naik ke tingkat yang lebih tinggi (*level* turun, menuju *parent*). Iterasi selanjutnya dilakukan dengan mengecek *node sibling* dari *parent*. Karena *node* melakukan pergerakan ke *level* yang lebih rendah (simpul lebih tinggi), maka kondisi ini disebut dengan kondisi naik, sehingga *flagUp* diaktifkan pada kondisi ini.



Gambar 10. *Flag Up*

3) Rutin dilakukan hingga tidak ada lagi *node* yang belum dicek. Pada *tree*, kondisi tersebut terjadi ketika *node* berada pada level 2, simpul yang dicek adalah simpul ke-8 dan *flagMove* aktif. Karena *flagMove* aktif maka semua *node* yang lebih rendah pada simpul tersebut sudah dilakukan pengecekan dan tak ada lagi *node* yang bernilai *valid*. Pada kondisi tersebut, maka *flagUp* akan diaktifkan karena tidak ada lagi *node sibling* pada *level* 2 yang belum di cek. Fungsi *flagUp* akan menyebabkan *flagMove* pada *parent* aktif, karena *node* pada *level* 1 hanya memiliki 1 nilai atau tak mempunyai *sibling*, maka berarti tidak ada lagi *node* yang dapat dicek atau program berakhir.

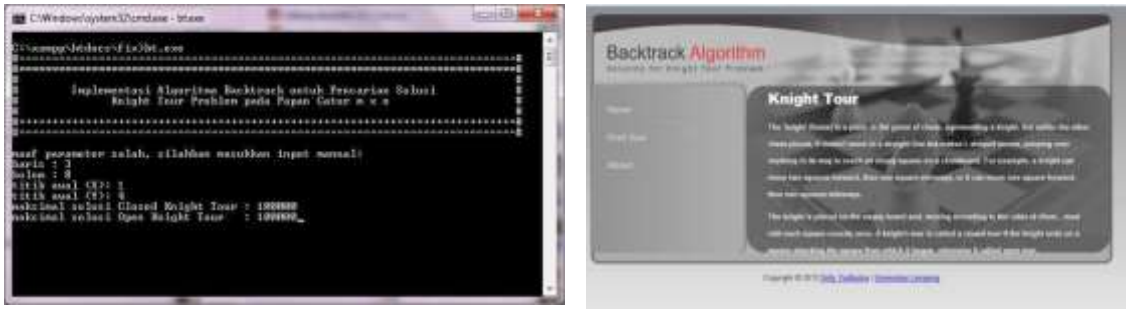
Berikut ini *pseudocode* dari uraian algoritma *backtrack* tersebut :

```
1. Inisialisasi
      maxNode  mxn
      level  0
      data[1]  0
      maxLangkah  8
2. While data[1]<maxLangkah do
3. If flagDown then
4.          flagDown=false;level++;
5.    else if flagMove then
6.          flagMove=false;data[level]++;
                  if data[level]>=maxLangkah then
7.                flagUp=true;goto end while;
8.          endif;
9.    else if flagUp then
10.         flagUp=false;level--;flagMove=true;goto end while;
11.         endif;
            valid=cek(data[level]);
12.         if valid then
13.               if level=maxNode-1 then tour
14.               else move
                  endif;
15.         else fMove=true
16.         endif
17.   end while
18.   end.
```

Proses *tree* pada program di atas yaitu berada pada baris ke-2 hingga baris ke-18 (terdapat pada rutin *while*). Dimana proses tersebut merupakan suatu rutin yang melakukan pengecekan pada tiap simpul dalam *tree*. Arah simpul berikutnya yang akan dicek (ke *child*, *sibling*, atau ke *parent*) diwakilkan dengan *flagDown*, *flagMove*, dan *flagUp*. Rutin tersebut terus menerus dilakukan hingga semua *node* telah selesai dicek (`while node[2]<=8`).

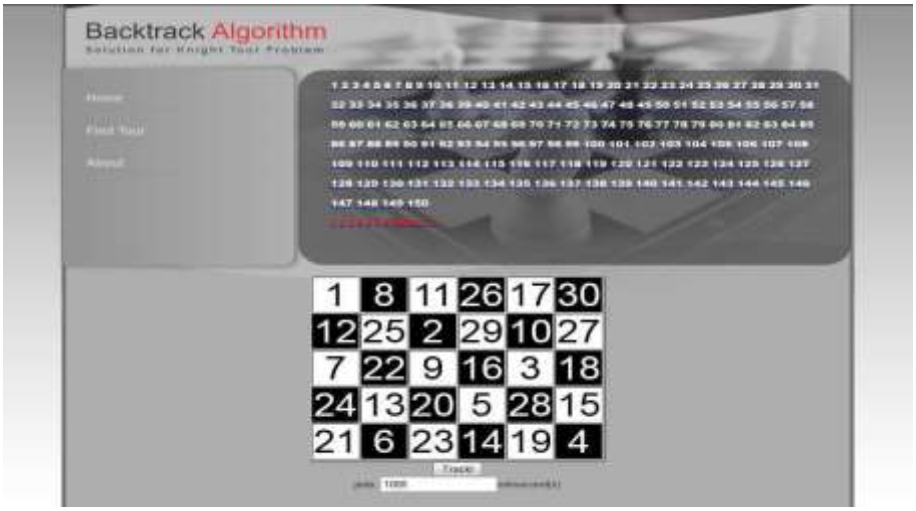
3.3 Implementasi Perangkat Lunak

Pada penelitian "Implementasi Algoritma *Backtrack* untuk Pencarian *Solusi Knight's Tour Problem* pada papan catur *m x n*" ini menggunakan dua program utama yaitu satu program digunakan untuk mencari solusi dari papan catur *m x n* dengan menggunakan bahasa pemrograman Pascal yang ditunjukkan pada Gambar 11(a). Sedangkan program lainnya merupakan antarmuka untuk melihat solusi yang telah dihasilkan dengan menggunakan bahasa pemrograman PHP yang keluarannya akan ditampilkan pada *browser* yang ditunjukkan pada gambar 11(b).



Gambar 11. a) Tampilan program Pascal b) Tampilan *Home* pada *Browser*

Sedangkan *output* atau keluaran dari program ini berupa urutan langkah kuda pada kotak hitam dan putih sesuai dengan ukuran papan catur dan titik awal kuda melangkah. Gambar 12 merupakan contoh *output* langkah kuda pada papan catur 5x6 pada titik (0,0).



Gambar 12. Tampilan *output* pada papan catur 5x6 di titik (0,0)

3.4 Pengujian

Pengujian ini dilakukan dengan mencoba mencari semua solusi yang ada pada papan catur 3x8 pada semua titik.

Tabel 1 Hasil pengujian pada papan catur 3 x 8

Titik Awal	Jumlah Solusi *Open KTP*	Jumlah Solusi *Closed KTP*	Waktu	
			(detik)	(mili detik)
0,0	82	0	0	468
0,1	20	0	0	264
0,2	4	0	0	188

0,3	24	0	0	202
0,4	24	0	0	264
0,5	4	0	0	186
0,6	20	0	0	248
0,7	82	0	0	452
1,0	68	0	0	421
1,1	48	0	0	358
1,2	0	0	0	171
1,3	20	0	0	218
1,4	20	0	0	250
1,5	0	0	0	171
1,6	48	0	0	358
1,7	68	0	0	422
2,0	82	0	0	451
2,1	20	0	0	249
2,2	4	0	0	171
2,3	24	0	0	249
2,4	24	0	0	248
2,5	4	0	0	203
2,6	20	0	0	264
2,7	82	0	0	468

Dari tabel di atas terlihat solusi yang dihasilkan pada papan catur 3 x 8 memiliki jumlah solusi yang beragam, hal ini dipengaruhi oleh titik awal dari kuda tersebut melangkah. Dari hasil di atas juga terlihat bahwa solusi yang dihasilkan berbentuk simetris pada baris yang sama, yaitu pada titik 0,0 memiliki jumlah solusi yang sama pada titik 0,7. Hal ini juga berlaku pada titik 1,0 dengan titik 1,7 dan pada titik 2,0 dengan titik 2,7. Pada papan 3x8 tidak ditemukan solusi *closed tour*, hal ini dikarenakan pada papan catur 3x8 memang tidak memiliki solusi *closed tour.*

3.5 Analisis Algoritma *Bactrack* pada *Knight's Tour Problem*

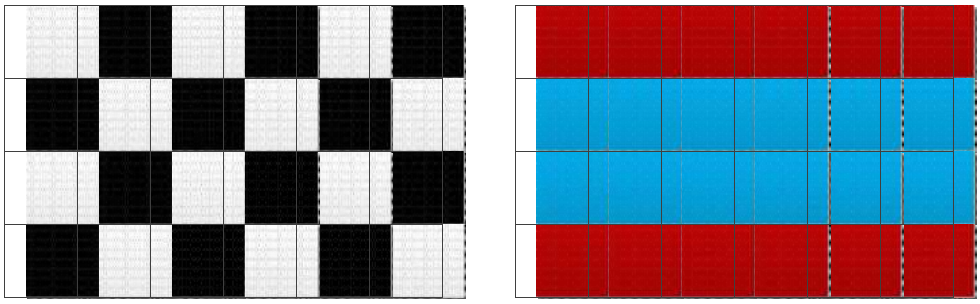
Dari hasil pengujian yang telah dilakukan memiliki hasil yang sesuai dengan Teorema Schwenk : " *An m x n chessboard with m ≤ n has a closed knight's tour unless one or more of these three conditions holds :*

(i) m and n are both odd;
(ii) m = 1, 2, or 4; or
(iii) m = 3 and n = 4, 6, or 8." (Schwenk, 1991)
[3].

Pembuktian (i), berdasarkan *knight tour* yang terjadi pada papan catur *m x n*. Dapat dilihat bahwa langkah yang sah selalu berawal dari kotak hitam ke kotak putih atau sebaliknya. Sedangkan pada *closed knight's tour* harus melewati jumlah yang sama baik itu kotak hitam atau kotak putih, jadi jumlah kotak hitam dan putih selalu membentuk suatu pasangan. Hal tersebut tidak akan terjadi jika jumlah *m* dan *n* ganjil, karena knight tidak akan mungkin kembali ke posisi awal. Contoh pembuktiannya pada papan catur 3x3, 3x5, 3x7, 5x3, 5x5, 5x7, 7x3, 7x5 dan 7x7 tidak ada solusi closed knight's tour.

Pembuktian (ii), untuk m = 1 atau 2 jelas dapat dilihat tidak mungkin ada *knight tour* yang terjadi karena papan tidak cukup lebar untuk memungkinkan terjadinya *open* atau *closed*

knight's tour. Jika m = 4, dapat dibuktikan bahwa tidak ada *closed knight's tour* yang terjadi dengan cara mewarnai papan catur 4x6 dengan warna merah pada baris 1 dan 4, sedangkan baris 2 dan 2 diberi warna biru seperti pada Gambar 8. Pada papan tersebut memiliki jumlah yang sama antara warna merah dan biru. Pada langkah *knight* yang berawal dari papan merah hanya bisa ke kotak biru. Berdasar fakta tersebut berarti suatu *closed knight's tour* dapat terjadi hanya jika urutan langkah *knight* bergantian antara merah dan biru. Sedangkan pada kotak 4x*n* tidak memungkinkan adanya langkah merah dan biru secara bergantian dari awal hingga akhir. Dari aturan langkah kuda yang berbentuk "L", diketahui bahwa langkah pertama kuda antara hitam dan putih, dan untuk menciptakan *closed knight's tour* harus ada kesesuaian antara kotak putih/hitam dan kotak biru/merah. Contoh pembuktiannya pada papan 4x3, 4x4, 4x5, 4x6, 4x7, dan 4x8 tidak ada solusi *closed knight's tour*.



Gambar 8. Papan 4x*n* tidak mungkin terjadi *closed knigh'st tour*

Untuk pembuktian (iii), telah dijelaskan secara terperinci oleh Schwenk pada tulisannya. Dengan menggunakan program yang berhasil dibuat menunjukkan hasil yang sama dengan teorema Schwenk yaitu pada papan catur 3x4, 3x6 dan 3x8 tidak ditemukan solusi *closed knight's tour.*

Dengan demikian algoritma *Backtrack* mampu menemukan semua solusi *Knight's Tour Problem* pada papan catur *m x n*. Solusi yang didapat mampu memberikan langkah yang tepat, baik itu untuk *open knight's tour* maupun *closed knight's tour*.

4 Kesimpulan

Berdasarkan hasil penelitian dan pengujian dari Implementasi Algoritma *Backtrack* pada Papan Catur *m x n*, maka dapat disimpulkan bahwa: algoritma *Backtrack* mampu menemukan semua solusi pada *Knight's Tour Problem* pada papan catur *m x n* dengan cepat dan tepat. Kelebihan algoritma *Backtrack* untuk menemukan semua solusi *Knight's Tour Problem* pada papan catur *m x n* yaitu mudah merunut balik suatu langkah, sehingga apabila menemui langkah terakhir maka dapat kembali pada posisi sebelumnya yang mempunyai langkah solusi sehingga dapat menemukan solusi lebih cepat.

5 Refference

[1] Sulistyono, Ari. *Penggunaan Teori Graf dan Algoritma Backtrack dalam Penyelesaian Masalah Knight's Tour*. Jurusan Ilmu Komputer, Universitas Pendidikan Indonesia. 2010http://cai.elearning.gunadarma.ac.id/webbasedmedia/download.php?file=penyelesaian-knight-tour.pdf diakses tanggal 1 Juni 2012, 21.15 WIB.

[2] Mulyanto, Aunur Rofiq. 2008. *Rekayasa Perangkat Lunak Jilid I untuk SMK*. Jakarta : Direktorak Sekolah Menengah Kejuruan

[3] Schwenk, Allen J. 1991. *"Which Rectangular Chessboards Have a Knight's Tour?"*. Mathematics Magazine, Vol. 64: 325-332 : Mathematical Association of America. http://www.jstor.org/stable/2690649 diakses 24 Juli 2012, 22.46 WIB.