# Penerapan Algoritma Runut-Balik dan Graf dalam Pemecahan Knight's Tour

**Krisnaldi Eka Pramudita NIM-13508014**

Prodi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Bandung 40135,
Email : if18014@students.if.itb.ac.id

## ABSTRAK

**Chess atau akrab dikenal catur adalah satu dari permainan board game yang sudah sangat terkenal, bahkan permainan ini sudah ada dari ratusan tahun lalu. Permainan ini menuntut kita untuk berpikir sedemikian rupa sehingga kita dapat mengalahkan pertahanan lawan. Tentunya dibutuhkan suatu kepintaran lebih untuk melakukan hal ini, Salah satu pengembangan dari board game catur adalah permainan Knight's Tour Knight's tour merupakan salah satu jenis permainan yang menggunakan dasar permainan catur, tapi dalam knight's tour hanya menggunakan satu buah bidak kuda.**

**Kata kunci:** Backtracking, Algoritma, Graf, Knight's Tour

## 1. PENDAHULUAN

Manusia adalah mahkluk yang punya akal budi sehingga manusia dapat berpikir dan menciptakan sesuatu. Salah satu sifat manusia adalah rasa ingin tahu yang sangat tinggi sehingga bisa mempelajari seuatu hal dan membuat/menemukan sesuatu dari apa yang mereka pelajari. Dalam menyelesaikan masalah, manusia umumnya manusia mempunyai langkah-langkah agar permasalahan itu dapat terselesaikan. Ada banyak sekali cara, manusia itu sendiri yang memilih efektivitasnya. Salah satu contohnya adalah ilmu matematika, khususnya yang berhubungan dengan matematika/struktur diskrit. Salah satu bahasan yang menarik dari struktur diskrit adalah penggunaan graf. Khususnya dalam makalah ini akan dijelaskan penggunaan graf atau algoritma backtracking yang berdasarkan pada teori pohon pada sebuah pemecahan knight's tour.

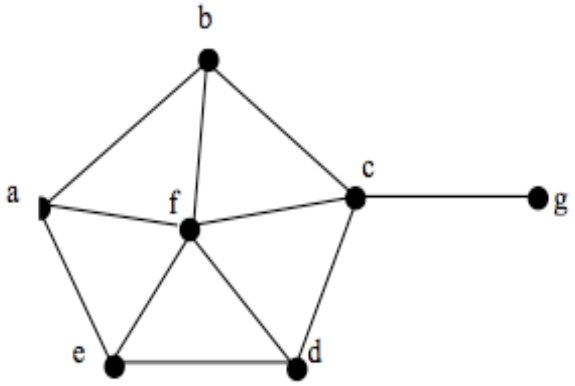
## 1.1 Graf

Secara informal, graf adalah himpunan benda-benda yang disebut verteks (node) yang terhubung oleh sisi (edge). Graf umumnya digambarkan sebagai kumpulan node yang dihubungkan oleh garis-garis.

Secara teoritis, graf G = (V,E) adalah suatu sistem yang terdiri dari himpunan V dan E dari pasangan tak terurut (vi,vj). Himpunan V merupakan himpunan titik dari Graf G dan anggotanya adalah node yang terhubung oleh sebuah sisi edge. Himpunan E adalah himpunan sisi dari graf G dan anggotanya disebut sisi. Notasi |V(G)| menyatakan banyaknya titik dalam graf G.

Jalan dibedakan menjadi 2 yaitu jalan terbuka dan tertutup. Jika pada sebuah jalan (dari suatu node ke node yang lain) kembali ke node semula maka disebut jalan tertutup. Jika saat selesai jalan tidak kembali ke posisi semula disebut jalan tertutup.

Saat sebuah jalan semua titiknya berbeda maka ia disebut jalur. Jika seriap sisi dari jalan tersbut berbeda maka dinamakan jejak. Jejak tertutup disebut sirkuit. Sebuah sirkuit yang titiknya semua berlainan disebut siklus.

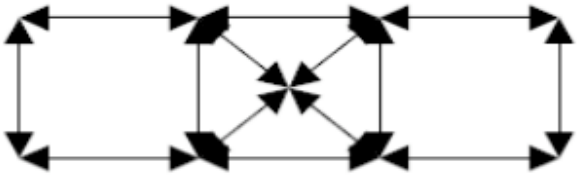



**Gambar 1.1 Sebuah Graf**

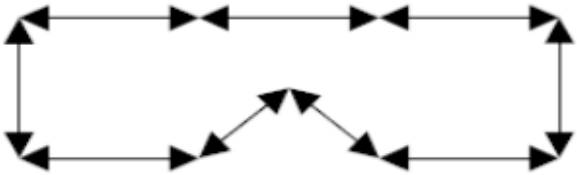
Pada gambar di atas. Salah satu jalan di atas adalah a-f-b-f-c-d-c. Sebuah jalan tertutup contohnya a-e-d-f-c-g-c-b-a (karena kembali pada node a). Contoh sebuah jejak a-f-b-c-d-e (setiap sisinya berbeda). Contoh jalur adalah a-e-f-c-d. Contoh sirkuit a-f-b-c-d-f-e-a. Dan contoh sebuah sirkuit adalah a-b-c-f-d-e-a.

## 1.2 Siklus Hamilton

Pada teori graf, siklus yang menggunakan semua node dan kembali ke node semula sering disebut dengan siklus Hamilton. Sedangkan jika semua node telah dilewati satu kali dan tidak kembali ke node semulia disebut jalur Hamilton. Kasus graf ini ditemukan oleh Sir Graf Hamilton pada tahun 1856.



Gambar 1.2 Graf G = (9,14)



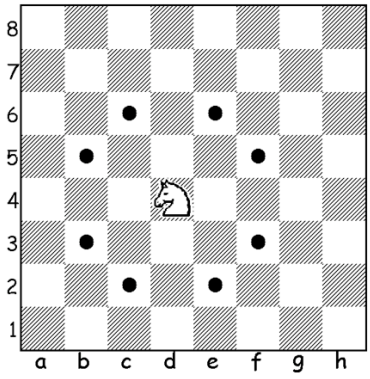
Gambar 1.3 Salah Satu Siklus Hamilton pada Graf G = (9,14)

Pada gambar 1.2 diatas ada graf G dan mempunya 14 sisi. Kita dapat membuat sebuah siklis Hamilton dari graf tersebut.

## 2. Knight's Tour

Knight's tour adalah suatu penerapa konsep graf pada chess. Permainan Knight's Tour adalah rangkaian perjalanan yang ditempuh oleh sebuah kuda catur hingga melewati semua kotak yang pada sebuah kotak catur tepat satu kali.

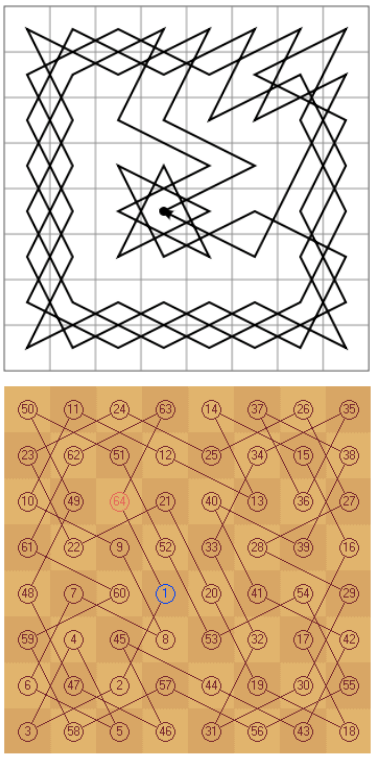
Pada gambar beriku adalah sebuah aturan langkah pada sebuah kuda pada permainan catur.




**gambar 1.4 Langkah kuda Catur**

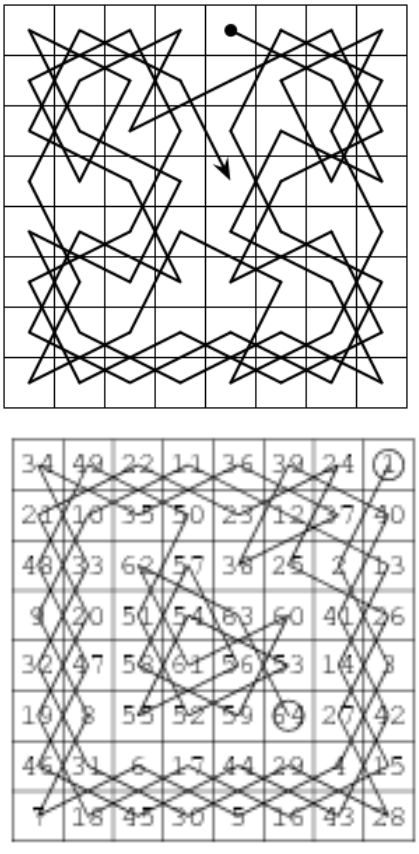
Closed Knight's Tour atau langkah kuda tertutup didapatkan ketika setiap kotak pada papan catur (8a – 1h) semua terlewati oleh kuda dan kuda kembali pada kotak semula. Sedangkan jika tidak kembali ke posisi/kotak semual disebut Open Knight's Tour atau langkah kuda tertutup.

Contoh closed knight's tour.



**Gambar 2.1 Closed knight's Tour**

Contoh open knight's tour



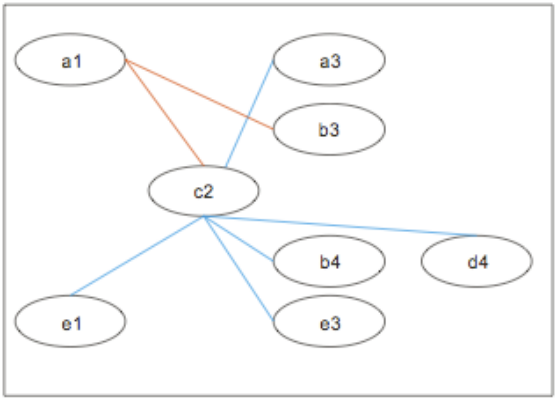
**Gambar 2.2 Open Knight's Tour**

Terlihat perbedaannya bahwa closed knight's tour dapat memulai kudanya dari kotak mana saja dan pasti kuda akan kembali ke posisi semual. Sedangkan pada open knight's tour kuda tidak kembali pada posisi semula tetapi kuda tetap melewati semua kotak pada papan catur 8x8.

### 2.1 Knight's Tour menggunakan Graf

Misal kita mempunyai papan catur 8x8 dengan kolom a-h dan baris 1-8. Andaikan tiap kotak mewakilkan satu state. Misalkan state a1 akan berhubungan dengan state b3 dan c2. State b2 akan berhubungan dengan a4,c4,d3, dan d1. State d4 akan berhubungan langsung (seperti pada gambar 1.4) dengan c6,e6,b5,f5,b3,f3,c2,dan e2. Begitu juga state-state lainnya.

Sebagai contoh kita mengambil kotak a1 sebagai state awal, maka kita akan punya 2 next state yaitu b3 dan c2. Contohnya pada gambar 2.2



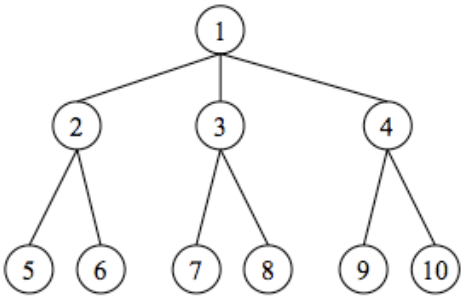
Gambar 2.2 Skema Graf

Pada gambar ini state a1 terhubung pada 2 state c2 dan b3. Jik kita memilih b3 maka kita akan terhubung ke state c1,c5,d4,dan d2. Jika kita memilij c2 maka kita akan punya 5 pilihan next state yaitu e3,e1,d4,a3,dan b3. Lalu bagaimana cara kita menemukan solusi dari knight's tour ini? Jika kita memilih sebuah path, berarti kita memutuskan hubungan dari state awal ke semua next state yang kita tidak pilih, ini juga berarti kita telah menyederhanakan skema graf kita. Contoh misalnya dari state a1 kita memilih jalan b3, maka path dari a1 ke c2 akan terputus. Karena hal ini maka kita akan punya ikatan dari state awal sampai akhir. Kalau ternyata Knight's Tour tidak terpenuhi maka pasti ada kesalahan dalam memilih next state nya.

Algorima backtracking ternyata menjadi salah satu penyelesaian dari kasus memilih next state yang efisien ini.

## 3. Algoritma Backtracking / Runut-Balik

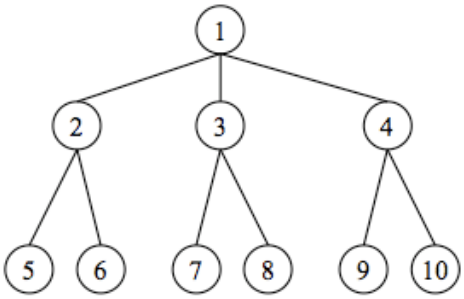
Algoritma *backtracking* mempunyai prinsip dasar yang sama seperti *brute-force* yaitu mencoba segala kemungkinan solusi. Perbedaan utamanya adalah pada ide dasarnya, semua solusi dibuat dalam bentuk pohon solusi (pohon ini tentunya berbentuk abstrak) dan algoritma akan menelusuri pohon tersebut secara *DFS* (*depth field search)* sampai ditemukan solusi yang layak. Nama *backtrack* didapatkan dari sifat algoritma ini yang memanfaat karakteristik himpunan solusinya yang sudah disusun menjadi suatu pohon solusi. Agar lebih jelas bisa dilihat pada pohon solusi berikut:




Misalkan pohon diatas menggambarkan solusi dari suatu permasalahan. Untuk mecapai solusi (5), maka jalan yang ditempuh adalah (1,2,5), demikian juga dengan solusi-solusi yang lain. Algoritma *backtrack* akan memeriksa mulai dari solusi yang pertama yaitu solusi (5). Jika ternyata solusi (5) bukan solusi yang layak maka algoritma akan melanjutkan ke solusi 6. Jalan yang ditempuh ke solusi 5 adalah 1,2,5 dan jalan ke solusi 6 adalah 1,2,6. Kedua solusi ini punya awal yng sama yaitu 1,2. Jadi kita menyimpan hasil 1,2 dan langsung memeriksa solusi 6. Pada pohon yang lebih rumit maka cara ini akan jauh lebih efisien dari brute force. Pada beberapa kasus, hasil sebelumnya harus disimpan, sedangkan ada juga beberapa kasus yang tidak perlu.

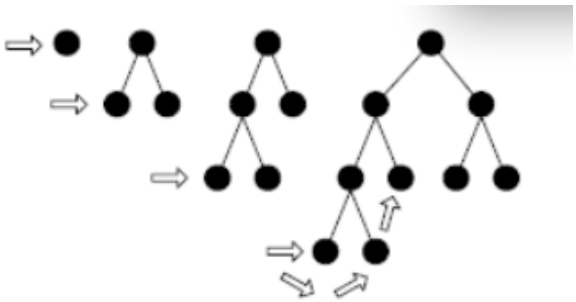
### 3.1 Implementasi dan Cara Kerja Algoritma Runut-Balik

Karena algoritma *backtrack* akan mencoba menelusuri semua kemungkinan solusi yang mungkin, maka hal pertama yang harus dilakukan adalah membuat algoritma dasar yang akan menjelajahi semua kemungkinan solusi. Kemudian algoritma ini diperbaiki sehingga cara pencarian solusinya dibuat lebih sistematis. Lebih tepatnya jika algoritma itu dibuat sehingga akan menelusuri kemungkinan solusi pada suatu pohon solusi abstrak. Algoritma ini memperbaiki teknik *brute-force* dengan cara menghentikan penelusuran cabang jika pada suatu saat sudah dipastikan tidak akan mengarah ke solusi. Dengan demikian jalan-jalan yang harus ditempuh yajg ada dibawah *node* pada pohon tersebut tidak akan ditelusuri lagi sehingga kompleksitas program akan berkurang. Kembali pada pohon solusi yang sebelumnya:




Jika pada pengecekan status di *node* (2) sudah dapat dipastikan bahwa jalan ini tidak akan menghasilkan solusi yang layak maka penelusuran jalan langsung dibatalkan dan dalam kasus ini langsung menelusuri *node* (3). Pembatalan penelusuran pada *node* (2) secara otomatis akan menghilangkan pengecekan jalan (1,2,5) dan (1,2,6) yang merupakan faktor untuk mengurangi kompleksitas waktu yang diperlukan. Semakin cepat terdeteksi bahwa jalan yang ditempuh tidak akan mengarah ke suatu solusi
yang layak maka program akan bekerja dengan lebih efisien.

Untuk kembali pada *state* sebelumnya (dalam kondisi *backtrack*) maka agar hasil perhitungan sampai dengan *state* tersebut harus disimpan dalam memori. Penyimpanan ini paling mudah dilakukan pada bahasa pemograman yang telah bisa menangani fungsi-fungsi atau prosedur-prosedur secara rekursif, sehingga manajemen memori akan dilakukan sepenuhnya oleh *compiler*. Pada bahasa pemograman yang lain, algoritma *backtrack* masih dapat diimplementasikan meskipun manajemen memori harus dilakukan oleh *programmer*. Cara menajemen memori yang baik adalah menggunakan *pointer* atau *dynamic array* karena kedalaman pohon solusi yang harus ditelusuri biasanya bervariasi dan tidak dapat ditentukan.



Algoritma *backtrack* dapat diimplementasikan dengan mudah pada bahasa-bahasa pemograman yang telah men-*support* pemograman rekursif. Bahasa pemograman yang nyaman digunakan adalah Pascal atau Java. Bahasa Pascal dipilih karena bisa diprogram secara rekursif dan mendukung penggunaan *pointer*. Sedangkan bahasa Java meskipun lebih rumit tetapi dapat bekerja secara rekursif dan sangat mudah dalam membuat *dynamic array*.

Skema yang umum digunakan pada pemograman dengan fungsi rekursif adalah telusuri solusi yang ada kemudian cek *state* program apakah sedang menuju ke suatu solusi. Jika ya maka panggil kembali fungsi itu secara rekursif. Kemudian cek apakah solusi sudah ditemukan (jika hanya perlu mencari sebuah solusi) atau semua kemungkinan solusi sudah diperiksa (jika ingin mengecek semua kemungkinan solusi). Jika ya maka langsung keluar dari prosedur atau fungsi tersebut. Kemudian pada akhir fungsi kembalikan semua perubahan yang dilakukan pada awal fungsi. Pada bahasa pemograman yang telah mendukung pemanggilan secara rekursif semua *state* setiap fungsi akan diatur oleh *compiler*. Dengan skema ini maka jika program tidak memiliki solusi maka *state* akhir program akan sama dengan *state* awal program.

### 3.2 Kegunaan Backtrack

Penggunaan terbesar *backtrack* adalah untuk membuat AI pada *board games*. Dengan algoritma ini program dapat menghasilkan pohon sampai dengan kedalaman tertentu dari *current status* dan memilih solusi yang akan membuat langkah-langkah yang dapat dilakukan oleh *user* akan menghasilkan pohon solusi baru dengan jumlah pilihan langkah terbanyak. Cara ini dipakai sebagai AI yang digunakan untuk *dynamic problem solving*. Beberapa kegunaan yang cukup terkenal dari algoritma *backtrack* dari suatu masalah "statik" adalah untuk memecahkan masalah *N-Queen problem* dan *maze solver*. *N-Queen problem* adalah bagaimana cara meletakkan bidak *Queen* catur sebanyak N buah pada papan catur atau pada papan ukuran NxN sedemikian rupa sehingga tidak ada satu bidakpun yang dapat memangsa bidak lainnya dengan 1 gerakan[1]. Meskipun mungkin terdapat lebih dari satu solusi untuk masalah ini, tetapi pencarian semua solusi biasanya tidak terlalu diperlukan, tetapi untuk beberapa kasus tertentu diperlukan pencarian semua solusi sehingga didapatkan solusi yang optimal.

*Maze solver* adalah bagaimana cara mencari jalan keluar dari suatu *maze* (labirin) yang diberikan. Pada *maze* yang sederhana dimana *field* yang dibentuk dapat direpresentasikan dalam bentuk biner dan pada setiap petak maksimal terdapat 4 kemungkinan: atas, kanan, bawah, dan kiri. Untuk masalah ini biasanya solusi pertama yang ditemukan bukanlah solusi yang paling optimal sehingga untuk mendapatkan hasil yang optimal dibutuhkan pencarian terhadap seluruh kemungkinan solusi. Hal ini disebabkan oleh urutan pencarian yang telah ditetapkan dalam program (apakah menyelidiki kemungkinan ke arah atas dahulu atau ke arah lainnya dahulu).

### 3.3 Pengaplikasian Algoritma Runut Balik pada Knight's Tour

Algoritma Backtracking bisa kita gunakan untuk mencari solusi dari kasus permainan. Kita akan membangun solusi parsial dari sebuah masalah lalu akan mencoba mengembangkan solusi tersebut. Jika solusi yang telah diperluas gagal, maka ia akan balik(dengan runut balik) dan akan mencoba solusi parsial lainnya, Knight'sTour yaitu dengan cara sebagai berikut :

1. Dari kotak awal kuda ditempatkan dan membangkitkan langkah-langkah yang mungkin dilalui oleh kuda.
2. Memilih salah satu langkah (kotak) yang kemudian diperluas langkah tersebut.
3. Menempatkan kuda pada kotak yang telah dipilih.
4. Mengulangi langkah satu untuk kotak yang sedang ditempati.
5. Kembali ke langkah sebelumnya jika belum ditemukan solusi (backtracking).
6. Pencarian berhenti jika telah ditemukan solusi atau tidak ada lagi langkah yang memungkinkan.

Berikut adalah contoh algoritma backtracking untuk kasus Knight's Tour

- boardnya adalah n x n (ukuran dari papan)
- (x,y) adalah koordinat letak kotak
- move adalah nomor kotak yang telah dilewati
- ok adalah boolean apakah sukses atau gagal

```
type chess_board is array
(1..n,1..n) of integer;
procedure knight (board : in out
chess_board;

x,y,move : in out integer;
                          ok : in
out boolean) is
w, z : integer;
begin
   if move = n^2+1 then
     ok := ( (x,y) = (1,1) );
   elsif board(x,y) /= 0 then
     ok := false;
   else
     board(x,y) := move;
     loop
       (w,z) := Next position
from (x,y);
       knight(board, w, z,
move+1, ok );
       exit when (ok or No moves
remain);
     end loop;
     if not ok then
       board ( x,y ) :=0; --
Backtracking
     end if;
    end if;
end knight;
```

## 4. Kesimpulan

Sampai sekarang masih banyak pakar yang meneliti cara-cara untuk menyelesaikan Knight's tour ini. Banyak juga penemuan-penemuan cara efisien untuk menemukan penyelesaiannya. Teori graf termasuk salah satu solusi yang kredibel untuk kasus ini. Teorema backtrack yang berdarkan pada teori pohon juga berguna karena akan menoptimalkan penghitungan kemungkinan yang ada. Algoritma *backtrack* sangat berguna untuk mencari solusi-solusi jumlah kombinasi jalan yang diperlukan untuk mencari solusi tersebut selalu berubah terhadap waktu ataupun respon *user* (dinamik) Jadi teori graf dan teori pohon dalam algoritma backtracking sangat mendukung untuk menyelesaikan masalah ini.

## Referensi


[1] Mumtaz, Fahmi. "ALGORITMA RUNUT-BALIK (BACKTRACKING ALGORITHM) PADA MASALAH KNIGHT'S TOUR". 2007.

[2] Munir, Rinaldi. (2009). Diktat Kuliah IF 2153 Struktur Diskrit. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[3] http://www.csc.liv.ac.uk/~ped/teachadmin/algor/search.html
Tanggal akses 15 Desember 2009 20.53

[4] http://www.usna.edu/Users/math/wdj/knight_tour.htm
Tanggal Akses 16 Desember 2009 18.20

[5] http://mathworld.wolfram.com/KnightsTour.html
Tanggal Akses 16 Desember 2009 18.45

[6] http://en.wikipedia.org/wiki/Knight%27s_tour
Tanggal Akses 16 Desember 2009 19.00