PENGGUNAAN ALGORITMA BACKTRACKING PADA KNIGHT'S TOUR PROBLEM

Adiputra Sejati

13507105
Teknik Informatika Institut Teknologi Bandung
e-mail: if17105@students.if.itb.ac.id; adiputra.sejati@yahoo.com

ABSTRAK

Algoritma *backtracking* pada dasarnya adalah algoritma untuk mencari kemungkinan solusi yang berbasis DFS. *Knight's tour* adalah sebuah permainan catur yang menggunakan satu buah bidak kuda dimana bidak kuda tersebut dapat berjalan dengan polanya pada semua kotak papan catur tepat hanya sekali. Pada makalah ini akan dibahas bagaimana masalah *knight's tour* diselesaikan dengan menggunakan strategi algoritma *backtracking*.

Kata kunci: *Algoritma, backtracking algorithm, knight's tour,* strategi algoritma

1. PENDAHULUAN

Seiring dengan perkembangan globalisasi, manusia tidak lepas dari perkembangan teknologi yang terus berkembang. Untuk menghadapi perkembangan teknologi tersebut, manusia selalu menemukan masalah-masalah yang dapat menghambat perkembangan teknologi tersebut.

Untuk menghadapi sebuah masalah, seseorang membutuhkan strategi untuk menyelesaikan masalah tersebut. Strategi adalah langkah-langkah dari sebuah rencana yang sudah dipersiapkan untuk memecahkan masalah tertentu.

Beberapa strategi pencarian solusi yang telah dipelajari dalam perkuliahan strategi algoritma ini adalah :

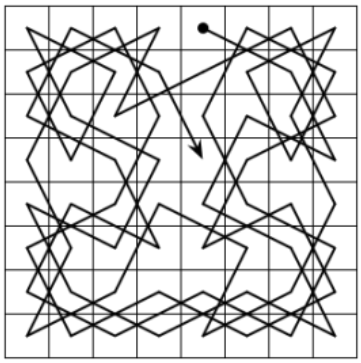
1. Algoritma *Brute Force*
2. Algoritma *Greedy*
3. Algoritma *Divide and Conguer*
4. *DFS* dan *BFS*
5. Algoritma Runut-balik (*backtracking*)
6. Algoritma *Branch and Bound*
7. Program Dinamis
8. Pencocokan String

Permasalahan yang ada dalam makalah ini adalah *knight's tour*. Bagaimana cara menyelesaikan permasalahan aplikasi teori graf pada permainan *knight's tour*. Sedangkan strategi yang akan digunakan penulis dalam memecahkan masalah ini adalah algoritma runut-balik (*backtracking algorithm).* Strategi algoritma runut-balik ini merupakan strategi algoritma pencarian yang berbasis pada ruang status (*state space base strategies*).

2. Knight's Tour

Knight's Tour adalah permainan catur yang hanya melibatkan sebuah bidak kuda dalam satu papan tersebut. Kuda tersebut akan ditempatkan pada sebuah kotak pada papan dan akan bergerak sesuai dengan aturan pola kuda dalam catur, dan kuda tersebut hanya bisa mengunjungi tiap kotak papan hanya sekali. *Knight's tour* dapat dibedakan menjadi dua, yaitu *closed tour* dan *open tour*. Yang dimaksud dengan *closed tour* adalah bidak kuda tersebut selesai melewati semua kotak dalam papan catur dan dapat kembali ke posisi awal bidak tersebut diletakkan. Sedangkan *open tour* adalah bidak kuda tersebut selesai melewati semua kotak di papan catur tersebut, tetapi tidak bisa kembali ke posisi awal bidak tersebut diletakkan.

Gambar1. Contoh pola *Closed Tour*



Gambar 2. Contoh pola *Open Tour*

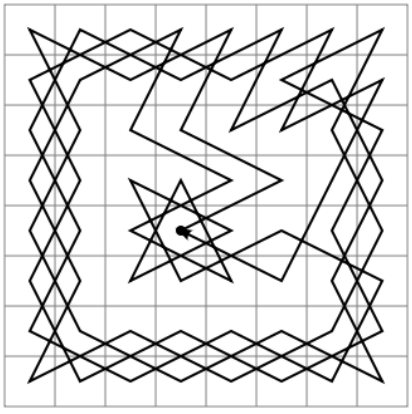
Pola kuda yang diperbolehkan adalah :

- 1 langkah ke atas dan 2 langkah ke kiri
- 1 langkah ke atas dan 2 langkah ke kanan
- 2 langkah ke atas dan 1 langkah ke kiri
- 2 langkah ke atas dan 1 langkah ke kanan
- 1 langkah ke bawah dan 2 langkah ke kiri
- 1 langkah ke bawah dan 2 langkah ke kanan
- 2 langkah ke bawah dan 1 langkah ke kiri
- 2 langkah ke bawah dan 1 langkah ke kanan

Permainan *Knight's Tour* ini telah dikenal sejak abad ke 9 setelah masehi. Pola dari *knight's tour* ini telah disajikan dalam bentuk model ayat dalam puisi sansekerta yang ditulis oleh Kashmir Rudrata pada abad ke 9.

Pola seorang ksatria tur pada papan setengah telah disajikan dalam bentuk ayat (sebagai kendala sastra) dalam puisi Sanskerta yang sangat bergaya *Kavyalankara* [3] yang ditulis oleh penyair abad ke-9 Kashmir Rudrata, yang membahas seni puisi, terutama dengan hubungannya dengan teater (Natyashastra).

Algoritma pertama untuk menyelesaikan permasalahan *Knight's Tour* adalah algoritma Warnsdorff, pada tahun 1823 oleh HC Warnsdorff.



Gambar 3. Contoh graf hasil pemecahan masalah dari *knight's tour*.

3. Algoitma Runut-Balik (*Backtracking Algoithm*)

Algoritma Runut-Balik adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Algoritma ini merupakan perbaikan dari algoritma *brute force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada.

Istilah runut balik pertama kali dikenalkan pada tahun 1950 oleh D.H. Lehmer. Kemudian R.J. Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma ini dan penerapannya terhadap berbagai persoalan. Pada saat ini algoritma runut-balik banyak digunakan untuk program games seperti *tic-tac-toe*, menemukan jalan keluar pada labirin, catur, dan lain-lain. Selain itu algoritma ini juga digunakan pada bidang kecerdasan buatan (*artificial intelegence*)

Backtracking dapat diterapkan hanya untuk masalah-masalah yang mempunyai konsep "calon parsial solusi" dan tes yang relatif cepat apakah mungkin bisa diselesaikan dengan solusi yang valid. *Backtracking* seringkali lebih cepat daripada *brute force* dari semua calon solusi yang banyak, hal ini dikarenakan algoritma ini dapat menghilangkan sejumlah besar kandidat dengan sebuah tes.

Backtracking tergantung pada pengguna diberikan "*black box procedure*" yang mendefinisikan masalah yang harus dipecahkan, dari beberapa calon solusi, dan bagaimana mereka diperluas menjadi kandidat solusi yang lengkap. Oleh karena itu yang *metaheuristic* daripada algoritma tertentu. Meskipun, tidak seperti banyak meta-heuristik lainnya, maka dijamin untuk menemukan semua solusi untuk masalah yang terbatas dalam jumlah yang dibatasi waktu.

3.1 Pseudocode Algoritma Runut-Balik

Dalam rangka untuk menerapkan runut-balik pada masalah khusus, pertama harus menyediakan data P untuk contoh khusus dari masalah yang harus dipecahkan, dan enam parameter prosedural, *root*, *reject*,*accept*, *first*, *next*, and *output*. Prosedur ini harus mengambil data contoh P sebagai parameter dan harus melakukan berikut ini:

1. *root*(*P*) : mengembalikan kandidat parsial pada akar dalam pohon pencarian
2. *reject*(*P*,*c*) : mengembalikan '*true'* jika kandidat partial c tidak dapat diselesaikan.
3. *accept*(*P*,*c*): mengembalikan '*true'* jika kandidat parsial c merupakan solusi dari data P, dan mengembalikan '*false'* jika sebaliknya.
4. *first*(*P*,*c*) : mengenerate elemen pertama dari kandidat c.
5. *next*(*P*,*s*) : megenerate elemen selanjutanya dari sebuah kandidat setelah elemen s.

6. *output*(*P*,*c*): menggunakan solusi c dari data P.

Algoritma backtracking mengurangi yang memanggil bt(root (P)), di mana bt adalah prosedur rekursif berikut:

```
Procedure bt(c)
   If reject(P,c) then return
   If accept(P,c) then output(P,c)
   s ← first(P,c)
   While s ≠ □ do
     bt(s)
     s ← next(P,s)
```

3.2 Properti Umum Metode Algoritma Runut-Balik

Untuk menerapkan metode runut-balik (*backtracking algorithm*), properti didefinisikan sebagai berikut:

1. Solusi Persoalan
   Solusi dinyatakan sebagai vektor dengan n-tuple:
   $X = (x_1, x_2,..., x_n)$, $x_i \in$ himpunan berhingga S_i.
   Mungkin saja $S_1 = S_2 = ... = S_n$.
   Contoh:
   $S_i = \{0, 1\}$,
   $x_i = 0$ atau 1
2. Fungsi Pembangkit nilai x_k
   Dinyatakan sebagai:
   T(*k*)
   T(*k*) membandingkan nilai untuk x_k , yang merupakan komponen vektor solusi.
3. Fungsi pembatas (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)
   Dinyatakan sebagai:
   B($x_1, x_2,..., x_k$)
   Fungsi pembatas menentukan apakah ($x_1, x_2,..., x_k$) mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak, maka ($x_1, x_2,..., x_k$) dibuang dan tidak lagi dipertimbangkan dalam pencarian solusi.
   Fungsi pembatas tidak selalu dinyatakan sebagai fungsi matematis. Ia dapat dinyatakan sebagai predikat yang bernilai *true* atau *false*, atau dalam bentuk lain yang ekivalen.

3.3 Pengorganisasian Solusi

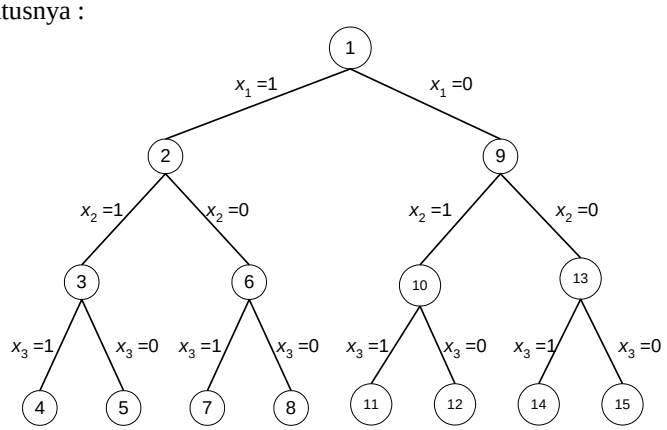
Semua kumungkinan solusi dari persoalan disebut ruang solusi. Jika $x_i \in S_i$, maka $S_1 \times S_2 \times \ldots \times S_n$ disebut ruang solusi. Dan jumlah anggota di ruang solusi tersbut adalah $|S_1| \cdot |S_2| \cdot \ldots \cdot |S_n|$.

Contohnya adalah persoalan *Knapsack* 0/1 untuk $n = 3$. Solusi persoalan dinyatakan sebagai vektor (x_1, x_2, x_3) dengan $x_i \in \{0,1\}$. Maka ruang solusinya adalah

$\{0,1\} \times \{0,1\} \times \{0,1\} = \{(0, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1)\}$.

Dan dengan dengan $n = 3$ terdapat $2^n = 2^3 = 8$ kemungkinan solusi, yaitu:
(0, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1), dan (1, 1, 1).

Penyelesaian secara *exhaustive search* adalah dengan menguji setiap kemungkinan solusi. Kemudian ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status persoalan, dan sisi/cabang dilabeli dengan nilai-nilai x_i. Lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian disebut sebagai pohon ruang status (*state space tree*). Maka pada persoalan *knapsack* diatas, pohon ruang statusnya :



Gambar 4. Ruang solusi untuk persoalan *Knapsack* 0/1 dengan n=3

3.4 Prinsip Pencarian Solusi dengan Metode Runut-Balik

Dari pohon ruang status yang telah dibangun maka kita dapat meninjau pencarian solusi secara dinamis. Langkah-langkah pencarian solusi adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan pencarian mendalam (DFS). Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*). Simpul hidup yang *sedang* diperluas dinamakan **simpul-E** (*Expand-node*).
2. 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** (*bounding function*). Simpul yang sudah mati tidak akan pernah diperluas lagi.

3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

4. Pengaplikasian Algoritma Runut-Balik Pada Knight's Tour

Pada persoalan Knight's Tour ini, algoritma runut-balik dapat kita pergunakan untuk menemukan solusi dengan tepat. Algoritma runut-balik akan membangun solusi parsial dari semua kemungkinan yang ada lalu akan mengembangkan solusi tersebut. Dalam persoalan knight's tour ini berarti algoritma ini akan membangun beberapa solusi dari step pertama bidak kuda yang diletakkan dan diperluas tiap-tiap solusi tersebut. Jika solusi yang diperluas tidak mencapai hasil maka algoritma ini akan melakukan backtracking dan mencari solusi dari solusi parsial lainnya.

Tahap-tahap yang akan dilakukan algoritma runut-balik pada persoalan ini adalah:

1. Dari kotak pertama tempat bidak kuda diletakkan, dibangun solusi-solusi parsial yang berisi langkah ke kotak berikutnya yang dapat memungkinkan dilalui bidak kuda tersebut.
2. Salah satu dari solusi tersebut dipilih dan kemudian dibangkitkan langkah berikutnya.
3. Bidak kuda jalan sesuai degan langkah yang dibangkitkan.
4. Tahap 1 dan tahap 2 dilakukan sampai semua solusi ditemukan(semua kotak terlewati) atau solusi tidak ditemukan (sampai langkah berikutnya tidak dapat diperluas lagi).
5. Apabila solusi tidak ditemukan, dilakukan runut balik ke langkah sebelumnya, kemudian dari kotak tersebut dilakukan tahap 1 dan tahap 2 lagi.
6. Pencarian solusi akan berhenti ketika solusi ditemukan atau tidak ditemukan satupun solusi.

Contoh algoritma ini pada papan 3x3 yang tidak akan menemukan solusi adalah sebagai berikut :

Anggap kotak yang belum pernah dilewati kuda direpresentasikan dengan angka 0, dan yang sudah pernah dilewati bidak kuda akan direpresentasikan dengan angka sesuai dengan urutan langkahnya.

Pada contoh ini bidak kuda awal akan diletakkan pada kolom 1 dan baris 1

Kuda mulai pada (1,1)

```
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
```

Di setiap kotak yang ditempati oleh bidak, kita menyimpan daftar kotak yang dapat dilewati untuk giliran berikutnya.

Pada kasus ini ada 2 kotak yang dapat dilewati oleh kuda, yaitu kotak (2,3) dan (3,2).

Pertama kita mencoba memilih kotak (2,3)

```
| 1 | 0 | 0 |
| 0 | 0 | 2 |
| 0 | 0 | 0 |
```

Pada papan sekarang terlihat ada 2 kotak yang mungkin untuk langkah selanjutnya, yaitu (1,1) dan (3,1). Karena (1,1) sudah pernah dilewati maka kotak yang diambil adalah (3,1).

```
| 1 | 0 | 0 |
| 0 | 0 | 2 |
| 3 | 0 | 0 |
```

Dari kotak ini terdapat 2 langkah yang dapat diambil, yaitu (2,3) dan (1,2). Karena (2,3) sudah perna dilewati maka bidak akan bergerak ke (1,2)

```
| 1 | 4 | 0 |
| 0 | 0 | 2 |
| 3 | 0 | 0 |
```

Langkah yang mungkin dari kotak ini: (3,1) dan (3,3). (3,1) sudah pernah dilewati. Ambil kotak (3,3).

```
| 1 | 4 | 0 |
| 0 | 0 | 2 |
| 3 | 0 | 5 |
```

Langkah yang mungkin dari kotak ini: (2,1) dan (1,2). (1,2) sudah pernah dilewati. Ambil (2,1).

```
| 1 | 4 | 0 |
| 6 | 0 | 2 |
| 3 | 0 | 5 |
```

Langkah yang mungkin dari kotak ini: (1,3) dan (3,3). (3,3) sudah pernah dilewati. Ambil (1,3).

```
| 1 | 4 | 7 |
| 6 | 0 | 2 |
| 3 | 0 | 5 |
```

Langkah yang mungkin dari kotak ini: (2,1) dan (3,2). (2,1) sudah pernah dilewati. Ambil (3,2).

```
| 1 | 4 | 7 |
| 6 | 0 | 2 |
| 3 | 8 | 5 |
```

Langkah yang mungkin dari kotak ini: (1,1) dan (1,3). (1,1) dan (1,3) sudah pernah dilewati. Maka tidak ada kotak yang memungkinkan untuk langkah selanjutnya. Maka runut-balik dilakukan ke tahap sebelumnya

```
| 1 | 4 | 7 |
| 6 | 0 | 2 |
| 3 | 0 | 5 |
```

Masih belum ada langkah yang mungkin dari tahap ke 7 selain kemungkinan yang pertama, maka runut-balik kembali dilakukan.

```
| 1 | 4 | 0 |
| 6 | 0 | 2 |
| 3 | 0 | 5 |
```

Masih belum ada langkah yang mungkin dari tahap ke 6 selain kemungkinan yang pertama, maka runut-balik kembali dilakukan.

```
| 1 | 4 | 0 |
| 0 | 0 | 2 |
| 3 | 0 | 5 |
```

Masih belum ada langkah yang mungkin dari tahap ke 5 selain kemungkinan yang pertama, maka runut-balik kembali dilakukan.

```
| 1 | 4 | 0 |
| 0 | 0 | 2 |
| 3 | 0 | 0 |
```

Begitu juga pada tahap ke 4, runut-balik kembali dilakukan.

```
| 1 | 0 | 0 |
| 0 | 0 | 2 |
| 3 | 0 | 0 |
```

Tahap ke 3 juga tidak memberikan langkah alternatif lainnya, runut-balik dilakukan.

```
| 1 | 0 | 0 |
| 0 | 0 | 2 |
| 0 | 0 | 0 |
```

Pada tahap ke 2 pun tidak ada langkah alternatif lainnya, runut-balik dilakukan kembali.

```
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
```

Kotak (3,2) adalah alternatif lainnya dari langkah pertama tadi, maka kita mencoba melangkah pada kotak (3.2)

```
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 2 | 0 |
```

Proses itu akan berjalan terus sampai solusi ditemukan. Ketika sebuah persegi dapat ditandai kita terus. Jika tidak dapat ditempatkan, runut-balik dilakukan. Dengan cara ini, semua kemungkinan gerakan yang mencoba. Jika pada suatu saat kita sudah ditandai semua 9 kotak, maka pencarian berhasil dilakukan. Jika kita kehabisan bergerak dari tahap 1, maka pencarian gagal dilakukan.

5. KESIMPULAN

Knight's Tour merupakan salah satu dari pengaplikasian teorigraf dari permainan papan catur. Ada beberapa algoritma untuk memecahkan masalah ini, yaitu *Neural network solutions* dan *warnsdorff's algorithm*. Algoritma runut-balik merupakan salah satu dari banyak solusi yang dapat menyelesaikan permasalahan *Knight's Tour*.

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Teknik Informatika STEI ITB, 2009.

[2] http://en.wikipedia.org/wiki/Knight's_tour
Tanggal akses: 26 Desember 2009 pukul 13:50

[3] http://www.experts-exchange.com/Programming/Algorithms/ Q_22635691.html
Tanggal akses: 26 Desemver 2009 pukul 14:00

[4] http://en.wikipedia.org/wiki/Backtracking
Tanggal akses: 27 Desember 2009 pukul 13:10

[5] http://cerebro.cs.xu.edu/csci220/01f/project1.html
Tanggal akses: 28 Desember 2009 pukul 11:40