Penerapan Algoritma Backtracking untuk penyelesaian Knight's Tour problem

Andika Pratama (13507005)

Program Studi Teknik Informatika ITB
Jl. Kidang Pananjung 2, Bandung
e-mail: andikapratama@gmail.com

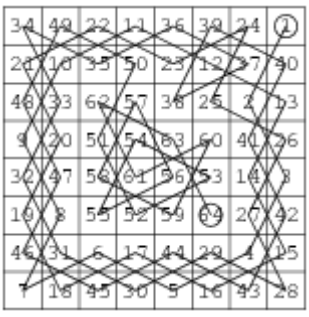
ABSTRAK

Knight's Tour adalah masalah matematis catur klasik, dan telah terdapat beberapa cara untuk menyelesaikan masalah Knight's tour, mulai dari brute-force sederhana, Algoritma Warnsdorff, dan lain lain. Disini saya akan mencoba menjelaskan bagaimana Algoritma Backtracking dapat diimplementasikan untuk memecahkan permasalahan ini, dengan dioptimasikan untuk mengurangi jumlah langkah yang diperlukan.

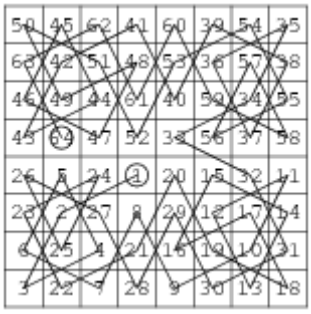
Kata kunci: Backtracking, Knight's tour, Optimasi,

1. PENDAHULUAN

Knight's Tour adalah masalah matematis mengenai perjalan bidak catur knight(kuda) dalam catur. Knight ditempatkan pada papan kosong, dan sesuai perturan gerak catur, Knight harus mengunjungi setiap kotak tepat sekali. Maka, rute Knight's Tour adalah rute hamilton pada graf dalam papan catur, dengan petak-petak catur sebagai simpul dan langkah knight sebagai sisi-nya.
Jika rute knight dapat diakhiri pada petak awalnya, maka itu disebut sebagai rute tertutup, dan menjadikannya sebagai sirkuit Hamilton. Jika tidak maka dikatakan sebagai rute terbuka.

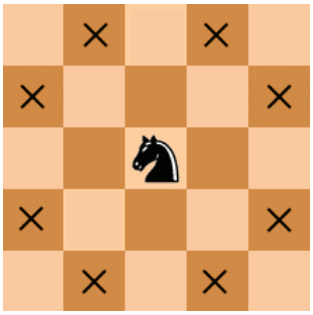


Gambar 1:Contoh Knight's Tour, Rute Terbuka



Gambar 2:Contoh Knight's Tour, rute tertutup

Knight's tour adalah puzzle yang cukup menarik bagi kebanyakan orang, yaitu karena sifat gerak Knight yang unik, yaitu hanya boleh bergerak berbentuk huruf 'L', yaitu bergerak 3x horizontal ber urutan dan 1x vertikal, atau kebalikannya.



Gambar 3: Gerakan yang valid untuk Knight

2. TINJAUAN PUSTAKA

2.1 Algoritma Backtracking

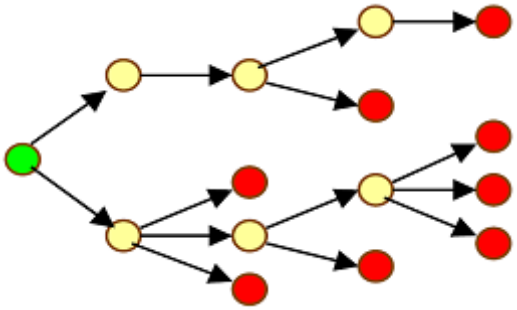
Runut-balik (backtracking) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Runut-balik, yang merupakan perbaikan dari algoritma brute-force, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dengan metode ini kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang

mengarah ke solusi saja yang akan dipertimbangkan. Akibatnya waktu pencarian dapat dihemat. Runut- balik lebih alami dinyatakan dengan algoritma rekursif. Kadang – kadang disebutkan pula bahwa runut-balik merupakan bentuk tipikal dari algoritma rekursif.

Istilah Runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. R.J alker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik dan penerappannya pada berbagai persoalan.saat ini, algoritma runut-balik banyak diterapkan untuk program games (permainan tic-tac-toe, menemukan jalan keluar dalam sebuah labirin, catur, dan masalah-masalah pada bidang kecerdasan buatan (artificial intelligence)).

2.1 Pencarian Solusi Algoritma Backtracking

Seperti yang telah dijelaskan diatas bahwa pencarian solusi dengan menggunakan algoritma backtracking ini berbasis pada DFS, maka kita menggunakan pohon ruang status.



Langkah-langkah pencarian solusi adalah sebagai berikut :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul yang sudah dilahirkan dinamakan simpul hidup dan simpul hidup yang diperluas dinamakan simpul-E (Expand- node).
2. Jika lintasan yang diperoleh dari perluasan simpul-E tidak mengarak ke solusi, maka simpul itu akan menjadi simpul mati dimana simpul itu tidak akan diperluas lagi.
3. Jika posisi terakhir ada di simpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lainnya dan jika tidak ada simpul anak maka dilakukan backtracking ke simpul orang tuanya.
4. Pencarian dihentikan jika kita telah menemukan solusi atau tidak ada simpul hidup yang dapat di diperluas.

2.2 Property Umum Algoritma Backtracking

Untuk menerapkan metode runut-balik, properti berikut didefinisikan:
1. Solusi persoalan
Solusi dinyatakan sebagai vektor dengan n-tuple: X = (x1, x2, …, xn), xi ∈ Si . Mungkin saja S1 = S2 = … = Sn.
Contoh: Si = {0, 1},
xi = 0 atau 1
2. Fungsi pembangkit nilai xk
Dinyatakan sebagai: T(k)
T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.
3. Fungsi pembatas (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)
Dinyatakan sebagai : B(x1, x2, …, xk)
B bernilai true jika (x1, x2, …, xk) mengarah ke solusi. Jika true, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika false, maka (x1, x2, …, xk) dibuang.

Semua kemungkinan solusi dari persoalan disebut ruang solusi (solution space). Secara formal dapat dinyatakan, bahwa xi ∈ Si, maka S1 x S2 x … x Sn
Disebut ruang solusi. Jumlah anggota di dalam ruang solusi adalah | S1|·| S2| ·… · | Sn |.

2.3 Skema Umum Algoritma Backtracking

Dalam pseudocode:

```
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 kedaun
        then
        CetakSolusi(x)
    endif
    RunutBalikR(k+1)
Endfor
```

3. METODE

3.1 Algoritma

Untuk memudahkan, posisi knight dalam catur akan direperesentasikan secara cartesian.

Penyelesaikan dimulai dari penempatan knight ke salah satu petak catur. Petak ini dicatat telah dilalui dan menjadi awal dari tour. Dengan algoritma backtrack, jika ditulis dalam pseudocode secara garis besar adalah seperti berikut:

```
function KnightTour (input: P:PapanCatur,Langkah:List) ->
```

```
boolean

Deklarasi
arah : integer
{
Posisi x + 3 , y + 1 -> 1
Posisi x + 3 , y - 1 -> 2
Posisi x - 3 , y + 1 -> 3
Posisi x - 3 , y - 1 -> 4
Posisi x + 1 , y + 3 -> 5
Posisi x - 1 , y + 3 -> 6
Posisi x + 1 , y - 3 -> 7
Posisi x - 1 , y - 3 -> 8
}

Algoritma
If SolusiTercapai(P) then return true
else
for tiap arah gerakan arah  do
        if GerakanValid (P,arah)
        then
        Langkah.Add(arah)
        Pindah (P,arah)
        if SolusiTakBisaDicapai(P)
        then
        Backtrack(P,arah,Langkah)
         If KnightTour(P,Langkah) then
        return true
        else Backtrack(P,arah,Langkah)
endwhile
CetakList(Langkah)
```

Fungsi SolusiTercapai(P) adalah mengembalikan true apabila knight telah mengunjungi semua petak dalam catur tepat sekali, yang berarti, rute telah lengkap. Jika belum tercapai solusi, dikembalikan false.

Fungsi GerakanValid(P,arah) akan memeriksa apakah terdapat arah petak yang belum pernah dikunjungi sebelumnya dan langkah kesana adalah gerakan yang valid. Jika yaa, akan dikembalikan true, jika tidak, false.

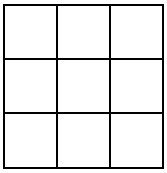
Fungsi SolusiTakBisaDicapai (P) akan memeriksa apakan sebuah solusi masih bisa dicapai dari kondisi sekarang. Contoh kondisi yang membuat solusi tak bisa dicapai misalnya danya petak yang tak bisa lagi dijangkau, atau knight tidak bisa berjalan lagi. Dengan kata lain, fungsi ini berfungsi sebagai fungsi pembatas. Jika solusi sudah tak bisa dicapai, dikembalikan true, jika tidak, false.

Fungsi Backtrack(P,Langkah) adalah mengembalikan ke kondisi sebelumnya, sebelum langkah terakhir.

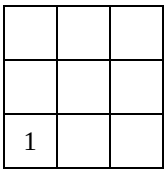
Setiap kondisi dalam algoritma backtracking yang digunakan dapat direpresentasikan sebagai pohon ruang status. Ruang status ini bebasis DFS, yaitu pencarian dilakukan secara mendalam, dari simpul awal ke simpul akhir, dan hanya akan melakukan runut balik jika simpul yang terakhir bukanlah solusi.

3. 2 Simulasi

Berikut contoh simulasi untuk board 3x3:

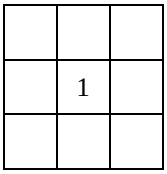


Dalam state awal, knight belum disimpan. Disini knight dapat disimpan dimana saja. Untuk pertama, kita masukkan ke (1,1)



Terdapat 2 langkah valid untuk knight, yaitu ke (2,3) atau ke (3,2). Namun disini fungsi SolusiTakBisaDicapai(P) akan mencegah simpul dikembangkan, karena petak (2,2) tidak akan pernah bisa dikunjungi, tidak ada langkah yang bisa mencapainya.

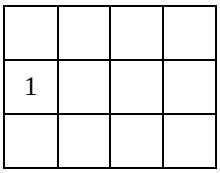
Kasus yang sama akan terjadi pada posisi lain, dan bila knight mulai pada posisi (2,2)



Maka simpul juga akan mati, karena knight tidak bisa berjalan kemanapun.

Karena tidak ditemukan rute, maka diketahui bahwa untuk papan 3x3 tidak ada Knight's Tour.

Berikut contoh simulasi untuk board 4x3:
Untuk posisi awal, kita coba knight pada posisi (1,2)



Dari sini, terdapat dua piliha jalan, yaitu ke (3,3) dan ke (3,1). Kita coba ke (3,3).

		2	
1			

Selanjutnya, pilihannya adalah (2,1) dan (4,1). Kita coba (2,1) terlebih dahulu.

		2	
1			
	3		

Dari sini, kemungkinan jalan yang valid adalah (1,3) atau (4,2). Dipilih (1,3). Langkahnya menjadi:

4		2	
1			
	3		

4		2	
1		5	
	3		

4		2	
1		5	
6	3		

4	7	2	
1		5	
6	3		

Pilihannya antara (3,1) atau (4,2). Kita ambil (3,1) dulu

4	7	2	
1		5	
6	3	8	

4	7	2	9
1		5	
6	3	8	

Buntu, back track ke state percabangan sebelumnya (saya persingkat untuk menghemat kertas)

4	7	2	
1		5	
6	3		

Kali ini kita pergi ke pilihan yang tersisa, (4,2).

4	7	2	
1		5	8
6	3		

Buntu lagi. Tak ada jalan yang valid. Backtrack kembali ke percabangan terakhir (saya persingkat untuk menghemat kertas)

		2	
1			
	3		

Sekarang pilihan yang belum dicoba adalah (4,2).

		2	
1			4
	3		

	5	2	
1			4
	3		

	5	2	
1			4
	3	6	

	5	2	7
1			4
	3	6	

	5	2	7
1	8		4
	3	6	

	5	2	7
1	8		4
	3	6	9

Buntu, tidak ada lagi langkah yang valid. Dilakukan backtrack ke percabangan terakhir (saya persingkat untuk menghemat kertas)

		2	
1			

Kini dipilih yang kemungkinan yang tersisam yaitu langkah ke (4,1)

		2	
1			
			3

		2	
1	4		
			3

		2	5
1	4		
			3

		2	5
1	4		
		6	3

	7	2	5
1	4		
		6	3

Terdapat 2 kemungkinan, yaitu (1,1) atau (4,2). Dipilih (1,1).

	7	2	5
1	4		
8		6	3

	7	2	5
1	4	9	
8		6	3

10	7	2	5
1	4	9	
8		6	3

10	7	2	5
1	4	9	
8	11	6	3

10	7	2	5
1	4	9	12
8	11	6	3

Semua petak telah dikunjungi. Algoritma selesai.
Maka telah ditemukan Kinght's Tour untuk papan 3x4.

Berdasarkan eksperimen, papan berukuran 3x4 adalah papan ukuran terkecil yang memiliki knight's tour.

Sedangkan untuk papan catur standar yang berukuran 8x8, menurut studi literatur memiliki 33,439,123,484,294 kemungkinan rute knight's tour.

4. KESIMPULAN

Untuk kebanyakan masalah yang banyak memiliki kemungkinan solusi, algoritma backtracking dapat menyelesaikannya dengan cukup baik dan lebih efisien daripada bruteforce atau exhaustive search. Hal utama yang membuat backtrack lebih baik adalah adanya fungsi pembatasnya yang mengurangi jumlah langkah yang diperlukan.

Algoritma backtracking terutama berguna karena dapat diterapkan pada banyak masalah, mudah dimengerti, dan lebih mangkus daripada metode bruteforce sederhana.

REFERENSI

[1] Munir, Rizaldi. 2005, strategi Algoritmik, Bandung.

[2] http://en.wikipedia.org/wiki/Knight's_tour diakses 2 Januari jam 20:00 WIB

[3] http://mathworld.wolfram.com/KnightsTour.html diakses 2 Januari jam 20:20 WIB

[4] http://www.markkeen.com/knight/index.html diakses 3 Januari jam 10:00 WIB