PENYELESAIAN MASALAH *MISSIONARIES* DAN *CANNIBAL* MENGGUNAKAN ALGORITMA *DFS* DENGAN VARIASI PENGHINDARAN *REPEATED STATE*

Gia Pusfita (13505082)

Program Studi Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha No.10 Bandung
e-mail: if15082@students.if.itb.ac.id

ABSTRAK

Banyak persoalan yang dapat diselesaikan dengan metode search. Salah satunya adalah persoalan missionaries dan cannibal yang sering dikaitkan dengan topik intelijensia buatan. Banyak pendekatan yang dapat dilakukan untuk persoalan tersebut, namun permasalahannya adalah tidak setiap pendekatan akan memberikan solusi terbaik yang memiliki cost, waktu dan ruang yang efisien. Makalah ini akan mengeksplor kasus missionaries dan cannibal tersebut dengan menggunakan algoritma *Depth First Search (DFS)* yang akan melakukan penelusuran terhadap pohon ruang status secara mendalam dengan variasi penghindaran looping dengan tujuan untuk mengurangi cost, waktu komputasi dan ruang pencarian solusi.

Kata kunci: *Depth First Search(DFS)*, *cost*

1. PENDAHULUAN

Tiga orang missionaries dan tiga kanibal terjebak bersama pada salah satu sisi sungai hendak menyebrang ke sisi sungai lainnya. Pada waktu tersebut hanya terdapat sebuah perahu yang mampu mengangkut maksimum dua orang untuk menyeberang ke sisi lain sungai. Jika suatu saat jumlah kanibal ditemukan lebih banyak dari jumlah missionaries maka kanibal tersebut akan memakan missionaries. Permasalahan dalam kasus ini adalah bagaimana menemukan serangkaian cara sehingga kanibal dan missionaries dapat menyeberang ke sisi sungai lainnya dengan aman tanpa ada satu missionariespun yang berhasil dimakan oleh kanibal dengan kata lain missionaries dan kanibal masing-masing tetap berjumlah tiga orang.

Gambar 1. Kondisi awal missionaries dan kanibal berada pada sisi yang sama

2. Depth First Search

2.1 Konsep Depth First Search

Depth First Search selalu mengembangkan salah satu node pada level terdalam dari pohon. Hanya ketika mencapai titik mati (non-goal node yang tidak mungkin diekspansi) akan dilakukan backtrack ke node induk dan mengembangkan node pada level berikutnya.

Depth First Search memiliki *requirement* memory terkecil, karena hanya perlu untuk menyimpan path tunggal dari akar hinga ke daun. Untuk ruang status dengan faktor b dan kedalaman maksimum m, depth first search memerlukan tempat penyimpanan sebanyak bm, hal ini sangat kontras jauh berbeda dengan metode pencarian *Breadth First Search* yang memakan memory sebanyak b^d dengan lebar d. Kompleksitas waktu untuk DFS sendiri adalah $O(b^m)$.

Untuk permasalahan yang memiliki solusi banyak, DFS akan lebih cepat dari DFS karena DFS dapat menemukan solusi hanya dengan melakukan eksplorasi terhadap bagian kecil dari seluruh ruang. Kelemahan dari DFS adalah saat DFS mencapai node yang salah yang telah dikembangkan sebelumnya. Pencarian akan terus dilakukan tanpa pernah berbalik.

Salah satu pendekatan untuk menghindari hal tersebut adalah dengan melakukan penghindaran untuk mengembangkan state yang telah dikembangkan sebelumnya sehingga tidak terjadi *looping*, salah satu caranya adalah dengan menggunakan *closed list.* List

tersebut akan menyimpan seluruh node yang pernah dikembangkan sebelumnya, sehingga ketika sebuah node akan dikembangkan, terlebih dahulu akan dilakukan pengecekan apakah node tersebut tersebut pernah dikembangkan sebelumnya dengan memanfaatkan *closed list* yang ada.

2.2 Algoritma Depth First Search

Algoritma *Depth First Search* jika dituliskan secara prosedural adalah sebagai berikut:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul tujuan maka solusi telah ditemukan.Stop
2. Jika Q kosong, tidak ada solusi. Stop
3. Ambil simpul v dari kepala antrian
   Jika kedalaman simpul v sama dengan batas kedalaman maksimum, kembali ke langkah 2
4. Bangkitkan semua anak dari simpul v, jika tidak memiliki anak lagi, kembali ke langkah 2. tempatkan semua anak dari v di awal antrian Q
5. Jika anak dari simpul v adalah simput tujuan, solusi telah ditemukan, kalau tidak kembali ke langkah 2

[Munir, Rinaldi, "Diktat Kuliah IF2251 Strategi Algoritmik", 2007.]

3 PENYELESAIAN PERSOALAN

3.1 Algoritma

Pada kasus berikut algoritma DFS diatas mengalami beberapa penambahan karena simpul akan dibunuh tidak hanya karena Q kosong tapi juga karena jumlah kanibal lebih banyak dari jumlah missionaries pada salah satu atau kedua sisi sungai atau karena simpul tersebut pernah dibangkitkan sebelumnya.
Jika ditulis secara prosedural maka akan tampak sebagai berikut :

1. Masukkan simpul akar ke dalam antrian Q. Masukkan kedalam closed list Jika simpul akar adalah simpul tujuan maka solusi telah ditemukan.Stop
2. Jika Q kosong, tidak ada solusi. Stop
3. Ambil simpul v dari kepala antrian
   Jika jumlah kanibal lebih banyak dari jumlah missionaris pada salah satu atau kedua sisi atau simpul pernah dibangkitkan sebelumnya, kembali ke langkah 2
4. Bangkitkan semua anak dari simpul v, masukkan v kedalam closed list. jika tidak memiliki anak lagi, kembali ke langkah 2. tempatkan semua anak dari v di awal antrian Q
5. Jika anak dari simpul v adalah simput tujuan, solusi telah ditemukan, kalau tidak kembali ke langkah 2

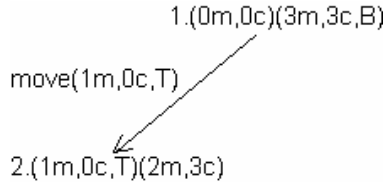
Pada permasalahan ini state dinotasikan dengan sebuah tuple <Xm,Xc, status posisi perahu>
Xm = Jumlah missionaries tersisa
Xc = Jumlah kanibal tersisa
Posisi perahu dapat berupa huruf B yang berarti perahu berada di sebelah barat dan T yang berarti perahu berada di sebelah timur. Jika digambarkan dengan pohon maka kondisi awal adalah sebagai berikut :

<0m,0c><3m,3c,B>

Kondisi tersebut berarti bahwa perahu ada disebelah Barat dengan jumlah missionaries dan kanibal pada sisi tersebut masing-masing tiga, sedangkan pada sisi Timur sungai masih kosong.
Untuk mempermudah dan menjaga konsistensi dalam kasus ini ditentukan urutan perpindahan missionaries dan kanibal adalah sebagai berikut :

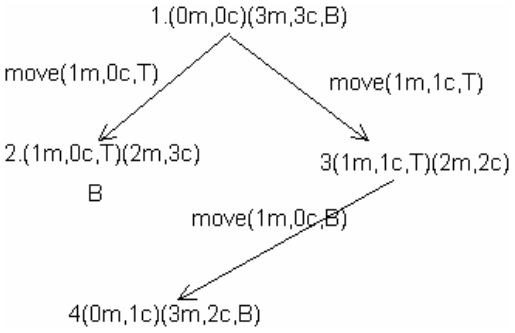
<1m,0c><1m,1c><2m,0c><1c,0m><2c>

pengembangan node selalu dimulai dari sisi kiri.Dari kondisi awal tersebut maka node yang akan dikembangkan adalah <1m,0c> . jika digambarkan adalah sebagai berikut :



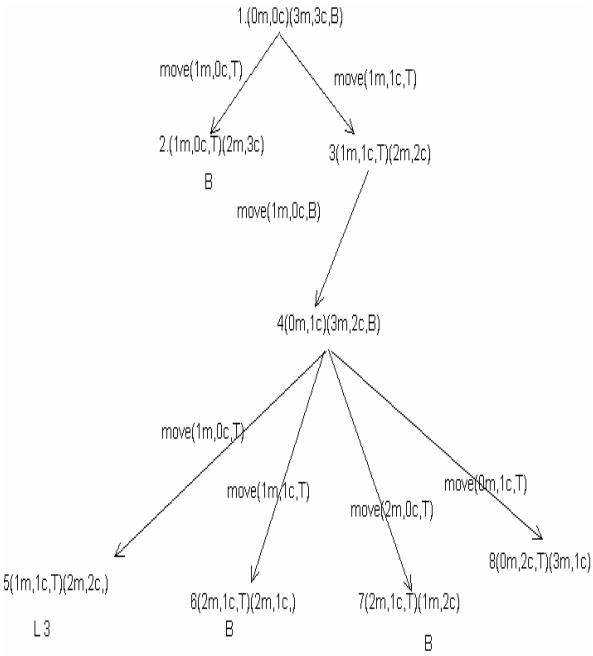
Gambar 2. Kondisi awal missionaries dan kanibal berada pada sisi yang sama

karena pada simpul dua di sebelah barat jumlah kanibal lebih banyak dari jumlah missionaries maka simpul tersebut mati. Pencarian kemudian berbalik ke simpul satu dan membangkitkan simpul anak lain yang masih mungkin untuk dibangkitkan.



Gambar 3. Pembangkitan anak pada simpul 3

Pencarian terus dilakukan seperti pada gambar diatas, ketika simpul nomor lima dibangkitkan diketahui bahwa simpul tersebut sama dengan simpul yang pernah dibangkitkan sebelumnya yaitu simpul 3, simpul lima kemudian dibunuh ditandai dengan L3 yang berarti looping terhadap simpul 3, maka simpul tersebut dibunuh lalu backtrack ke pohon induk kemudian mengembangkan simpul-simpul anak lainnnya

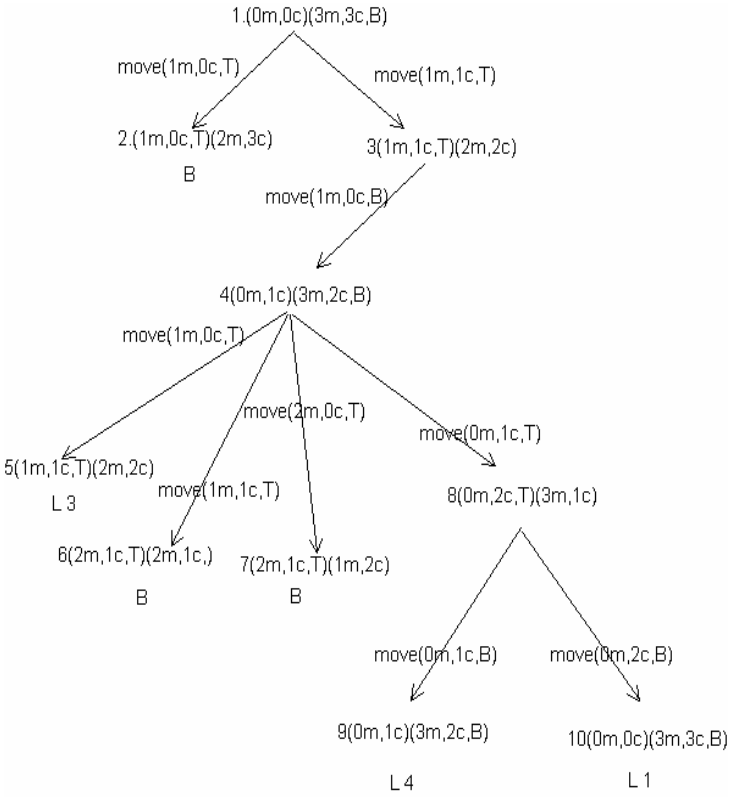


Gambar 4. Pembangkitan anak pada simpul 4

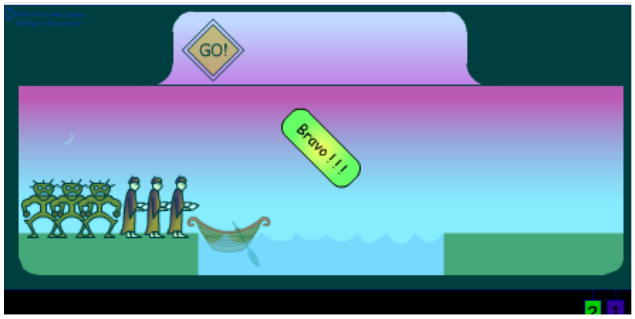
Seperti yang dapat dilihat dari gambar 4 . simpul 6 dan 7 ikut dibunuh karena pada salah satu sisi, jumlah kanibal lebih banyak dari missionaries dan akhirnya dikembangkan simpul 8.
karena simpul 9 dan 10 terjadi looping dan simpul 8 tidak mungkin membangkitkan simpul anak lagi maka, simpul 4 lah yang akan membangkitkan anak, begitu seterusnya hingga tercapai solusi atau tidak terdapat solusi sama sekali (pohon ruang status akhir ruang status keseluruhan disertakan dalam lampiran).
Solusi yang diperoleh adalah :

Sisi Timur	Perpindahan	Sisi Barat
	1m,1c	3m,3c *
1m,1c*		2m,2c
	1m,0c	
0m,1c		3m,2c*
	0m,2c	
0m,3c*		3m,0c
	0m,1c	
0m,2c		3m,1c*
	2m,0c	
2m,2c*		1m,1c

Sisi Timur	Perpindahan	Sisi Barat
	1m,1c	
1m,1c		2m,2c*
	2m,0c	
3m,1c*		0m,2c
	0m,1c	
3m,0c		0m,3c*
	0m,2c	
3m,2c*		0m,1c
	1m,0c	
2m,2c		1m,1c*
	1m,0c	
3m,2c*		0m,1c
	0m,1c	
3m,1c		0m,2c*
3m,3c*	0m,2c	



Gambar 5. Pembangkitan anak pada simpul 8



Gambar 6. Kondisi akhir missionaries dan kanibal berada pada sisi yang sama di seberang sungai

3.2 Pseudo Code

```
Skema rekursif
def dfs(v):
    mark v as visited
    preorder-process(v)
    for all vertices i adjacent to v such
that i not visited and (canibal >
missionaries both river side)
        dfs(i)
     postorder-process(v)
```

```
Skema non-rekursif :
dfs(graph G)
{
  list Closed = empty
  list L = empty
  tree T = empty
  choose a starting vertex x
  search(x)
  while(L is not empty)
  {
    remove edge (v, w) from beginning of L
    if w is not in Closed
    {
      add (v, w) to T
      add (v,w) to Closed
      search(w)
    }
  }
}

search(vertex v)
{
  visit v
  for each edge (v, w)
    add edge (v, w) to the beginning of L
}
```

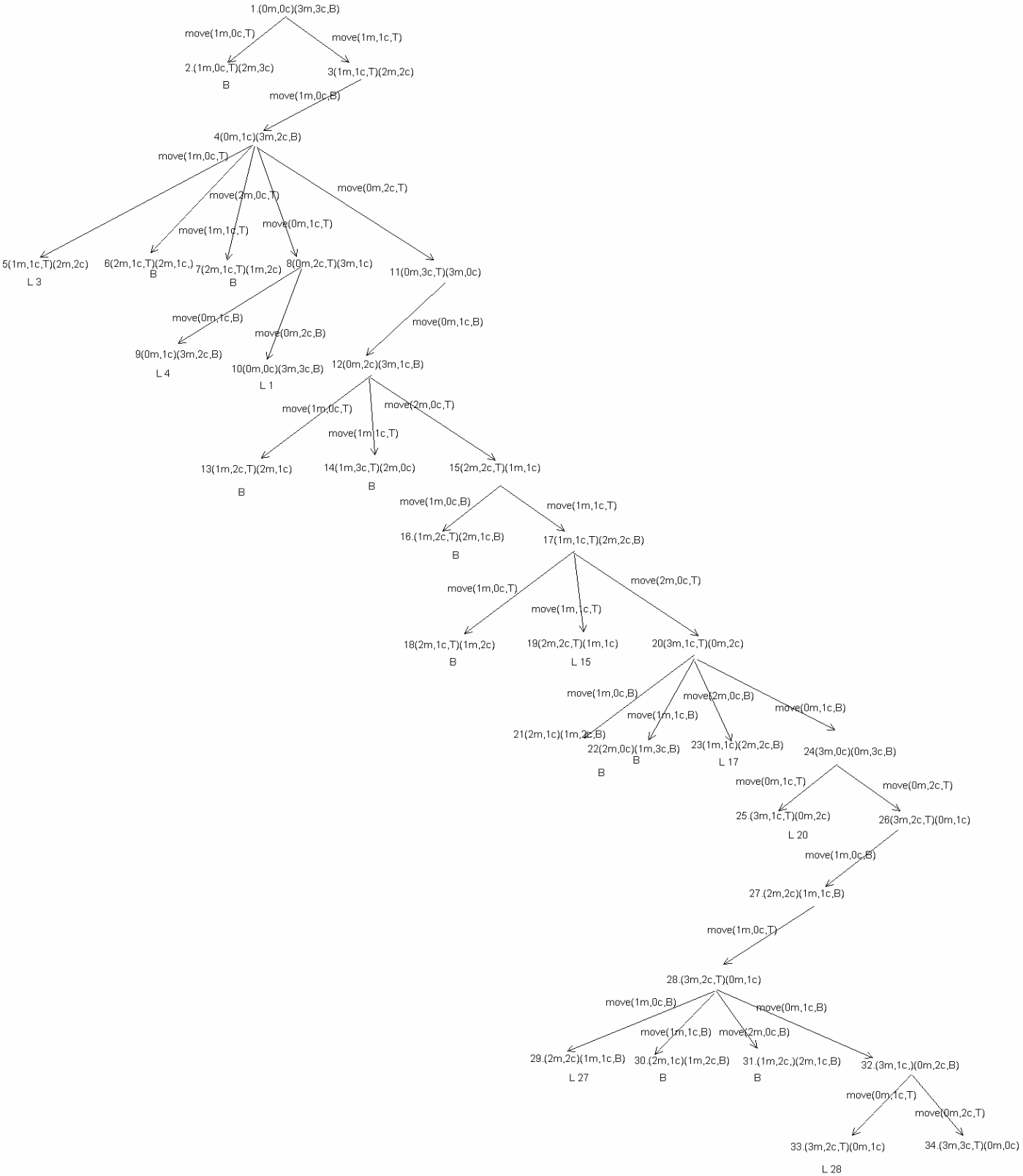
IV. KESIMPULAN

Untuk penyelesaian persoalan yang memiliki banyak solusi, DFS adalah salah satu algoritma yang sering digunakan karena kecepatannya, namun untuk kasus dengan kedalaman maksimum tidak diketahui seperti kasus Cannibal dan missionaries DFS dapat memberikan hasil tak hingga, namun hal tersebut dapat dipangkas dengan menghindari adanya pembangkitan simpul yang telah dikunjungi sebelumnya. Dengan memanfaatkan closed list yang berfungsi sebagai tempat penyimpanan simpul-simpul yang telah dikunjungi waktu komputasi menjadi lebih cepat dan, DFS tidak akan mengalami pencarian tak berhingga.

REFERENSI

[1] Munir, Rinaldi, "Diktat Kuliah IF2251 Strategi Algoritmik", 2007

[2] Russel Stuart, "Artificial Intelligence A modern Aproach", Prentice Hall, 1995.


1.(0m,0c)(3m,3c,B)
move(1m,0c,T)
move(1m,1c,T)
2.(1m,0c,T)(2m,3c)
B
3(1m,1c,T)(2m,2c)
move(1m,0c,B)
4(0m,1c)(3m,2c,B)
move(1m,0c,T)
move(2m,0c,T)
move(0m,2c,T)
move(1m,1c,T)
move(0m,1c,T)
5(1m,1c,T)(2m,2c)
L 3
6(2m,1c,T)(2m,1c,)
B
7(2m,1c,T)(1m,2c)
B
8(0m,2c,T)(3m,1c)
11(0m,3c,T)(3m,0c)
move(0m,1c,B)
move(0m,2c,B)
move(0m,1c,B)
9(0m,1c)(3m,2c,B)
L 4
10(0m,0c)(3m,3c,B)
L 1
12(0m,2c)(3m,1c,B)
move(1m,0c,T)
move(2m,0c,T)
move(1m,1c,T)
13(1m,2c,T)(2m,1c)
B
14(1m,3c,T)(2m,0c)
B
15(2m,2c,T)(1m,1c)
move(1m,0c,B)
move(1m,1c,T)
16.(1m,2c,T)(2m,1c,B)
B
17(1m,1c,T)(2m,2c,B)
move(1m,0c,T)
move(2m,0c,T)
move(1m,1c,T)
18(2m,1c,T)(1m,2c)
B
19(2m,2c,T)(1m,1c)
L 15
20(3m,1c,T)(0m,2c)
move(1m,0c,B)
move(2m,0c,B)
move(0m,1c,B)
move(1m,1c,B)
21(2m,1c)(1m,2c,B)
B
22(2m,0c)(1m,3c,B)
B
23(1m,1c)(2m,2c,B)
L 17
24(3m,0c)(0m,3c,B)
move(0m,1c,T)
move(0m,2c,T)
25.(3m,1c,T)(0m,2c)
L 20
26(3m,2c,T)(0m,1c)
move(1m,0c,B)
27.(2m,2c)(1m,1c,B)
move(1m,0c,T)
28.(3m,2c,T)(0m,1c)
move(1m,0c,B)
move(0m,1c,B)
move(1m,1c,B)
move(2m,0c,B)
29.(2m,2c)(1m,1c,B)
L 27
30.(2m,1c)(1m,2c,B)
B
31.(1m,2c,)(2m,1c,B)
B
32.(3m,1c,)(0m,2c,B)
move(0m,1c,T)
move(0m,2c,T)
33.(3m,2c,T)(0m,1c)
L 28
34.(3m,3c,T)(0m,0c)