Penyelesaian Permainan "3 *missionaries* and 3 *cannibals*" Dengan Algoritma Runut-Balik

Hendro

Program Studi Teknik Informatika
Alamat : Jl. Ciumbeuluit Gg.Suhari No. 95/155A
E-mail: kyoshiro@students.itb.ac.id / if15103@students.if.itb.ac.id

ABSTRAK

Sekarang ini, sudah banyak *game* / permainan-permainan menarik yang beredar di masyarakat. Ada yang dalam bentuk *console* seperti PlayStation maupun permainan pada komputer. Permainan ini awalnya adalah suatu hiburan untuk menyegarkan pikiran. Namun banyak juga orang yang sampai kecanduan terhadap suatu permainan karena permainan itu sangat menarik dan terus berlanjut. Hal seperti ini kurang bagus. Permainan yang baik adalah permainan yang bukan hanya menghibur, tetapi juga mengasah otak ibarat sambil menyelam minum air. Namun biasanya permainan seperti itu kurang populer sehingga banyak yang tidak mengetahuinya.

Oleh karena itu, pada makalah ini, akan dibahas suatu permainan logika yaitu "3 *missionaries* and 3 *cannibals*" dimana tujuan akhir dari permainan ini adalah menyebrangkan tiga *missionaries* dan tiga *cannibals* ke sisi lainnya dengan perahu yang hanya mampu mengangkut dua orang dan dikendalikan oleh salah satu orang pada perahu tersebut. Pada permainan ini, ada banyak kemungkinan yang dapat dilakukan dan juga terdapat beberapa kemungkinan yang menuju ke penyelesaian. Salah satu algoritma yang cukup mangkus untuk menyelesaikan permasalahan ini adalah dengan menggunakan algoritma runut-balik (*backtracking*). Algoritma ini mampu memangkas kemungkinan-kemungkinan yang tidak menuju solusi sehingga waktu yang diperlukan untuk mencari penyelesaian dari permainan ini menjadi lebih singkat.

Kata kunci : *missionary, missionaries, cannibal, cannibals*, algoritma Runut-Balik, *Backtracking*.

1. PENDAHULUAN

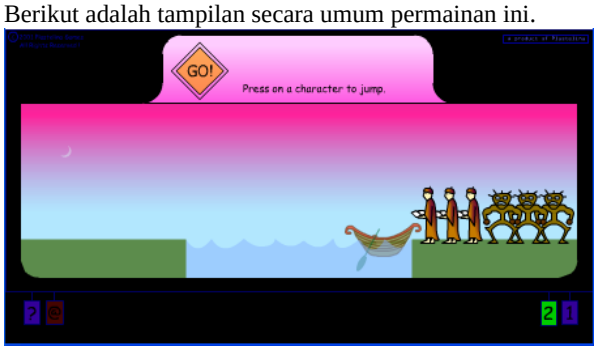
Permainan "3 missionaries and 3 cannibals" ini tidak begitu populer diantara permainan-permainan lainnya disebabkan karena permainannya yang sederhana dan antar muka yang kurang menarik sehingga tidak menarik orang untuk memainkannya. Namun, permainan ini sangatlah bagus untuk mengasah logika. Karena untuk menyelesaikan permainan ini, dibutuhkan sedikit logika untuk menyelesaikannya.

Adapun permainan ini memiliki aturan sebagai berikut. Pada saat awal permainan, terdapat dua buah daratan di sisi sebelah kanan dan kiri yang dipisahkan oleh sungai. Terdapat tiga missionaries dan tiga cannibals di sisi sebelah kanan dan sebuah perahu yang telah menepi di sisi kanan. Perahu tersebut hanya bisa mengangkut maksimal dua orang dan dikendalikan oleh salah orang dalam perahu. Perahu hanya dapat berjalan dari kiri ke kanan atau kanan ke kiri.

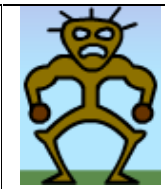
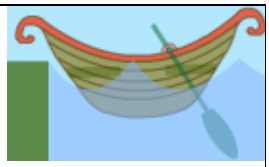
Selain itu, juga terdapat aturan bahwa jumlah missionary tidak boleh lebih sedikit dari jumlah cannibal pada kedua sisinya. Jika jumlah missionary lebih sedikit daripada jumlah cannibal, maka missionary akan dimakan oleh cannibal sehingga permainan pun akan berakhir dengan kekalahan.

Tujuan akhir dari permainan ini adalah memindahkan tiga missionaries dan tiga cannibals dengan selamat ke daratan di sisi kiri sesuai dengan aturan diatas. Jadi dibutuhkan pertimbangan pada setiap langkah agar jumlah missionary di kedua sisi selalu leih besar atau sama dengan jumlah cannibal.

Berikut adalah tampilan secara umum permainan ini.



Gambar 1. Tampilan awal permainan

		
Gambar 2. Tampilan missionary	**Gambar 3. Tampilan cannibal**	**Gambar 4. Tampilan perahu**

Berikut penjelasan singkat mengenai teknis permainan. Untuk menaikkan missionary atau cannibal, cukup mengkliknya maka missionary atau cannibal akan otomatis masuk ke perahu. Begitu pula untuk mengeluarkannya dari perahu tinggal mengklik missionary atau cannibal yang ada di dalam perahu. Untuk menjalankan perahu, cukup mengklik tombol "GO!" yang terletak kanan atas.

Dengan aturan-aturan di atas, permainan ini dapat diselesaikan dengan algoritma *Brute Force* yaitu dengan metode *exhaustive search. Exhaustive search* mencoba semua kemungkinan pilihan pada setiap langkah dalam permainan ini kemudian mencoba lagi setiap kemungkinan pada setiap langkah yang dipilih. Begitu seterusnya sampai semua kemungkinan selesai ditelusuri.

Metode ini tentu saja akan menemukan solusi karena setiap kemungkinan ditelusuri. Namun, karena setiap kemungkinan ditelusuri, maka dibutuhkan waktu yang cukup lama untuk memproses sampai menemukan solusinya. Sehingga metode ini tidak mangkus dan jarang dipakai. Metode ini sering kali dipakai jika tidak ada metode lain yang mampu menyelesaikan suatu permasalahan.

Oleh karena itu, untuk menyelesaikan permainan ini, digunakan algoritma runut-balik (*Backtracking*) yang jauh lebih mangkus dibandingkan *exhaustive search.*

2. ALGORITMA RUNUT-BALIK (BACKTRACKING)

2.1 Deskripsi Umum

Algoritma runut-balik pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Dalam perkembangannnya, beberapa ahli seperti RJ Walker, Golomb, dan Baumert menyajikan uraian umum mengenai runut-balik dalam menyelesaikan berbagai persoalan.

Dasar dari algoritma runut-balik adalah algoritma pencarian mendalam atau DFS (Depth First Search), yaitu metode pencarian yang melakukan percarian solusi secara vertikal. Selama pencarian dilakukan terbentuk pohon ruang status dimana setiap simpul yang ditelusuri akan ditelusuri lagi sampai mencapai daun atau simpul akhirnya. Namun akan dibutuhkan waktu yang cukup lama juga untuk mencari solusi karena setiap simpul akan ditelusuri.

Pada algoritma runut-balik, diberi perbaikan sehingga dapat mencari solusi persoalan 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 runut-balik, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat.

Saat ini algoritma runut-balik banyak diterapkan untuk program *games* (seperti permainan *tic-tac-toe*, menemukan jalan keluar dalam sebuah labirin, catur, dll) dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

2.2 Properti Umum

1. Solusi persoalan.
   - Solusi dinyatakan sebagai vektor dengan *n-tuple*: $X = (x1, x2, \ldots, xn),\ xi \in Si$.
   - Mungkin saja $S1 = S2 = \ldots = Sn$.
   - Contoh: $Si = \{0, 1\},\ xi = 0$ atau 1
2. Fungsi pembangkit nilai x_k
   Dinyatakan sebagai:
   $$T(k)$$
   $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
3. Fungsi pembatas (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)
   - Dinyatakan sebagai
     $$B(x1, x2, \ldots, xk)$$
   - B bernilai *true* jika $(x1, x2, \ldots, xk)$ mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk $xk+1$ dilanjutkan, tetapi jika *false*, maka $(x1, x2, \ldots, xk)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

2.3 Pengorganisasian Solusi

- Semua kemungkinan solusi dari persoalan disebut **ruang solusi** (*solution space*).
- Jika $xi \in Si$, maka $S1 \times S2 \times \ldots \times Sn$ disebut **ruang solusi**.
- Jumlah anggota di dalam ruang solusi adalah $|S1| \cdot |S2| \cdot \ldots \cdot |Sn|$.

- Ruang solusi diorganisasikan ke dalam struktur pohon.
- Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai *xi*.
- Lintasan dari akar ke daun menyatakan solusi yang mungkin.
- Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai **pohon ruang status** (*state space tree*).

2.4 Prinsip Pencarian Solusi

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.

Biasanya algoritma dari runut-balik bersifat rekursif sehingga setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.

Jika jumlah simpul dalam pohon ruang status adalah $2n$ atau $n!$, maka untuk kasus terburuk, algoritma runut-balik membutuhkan waktu dalam $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

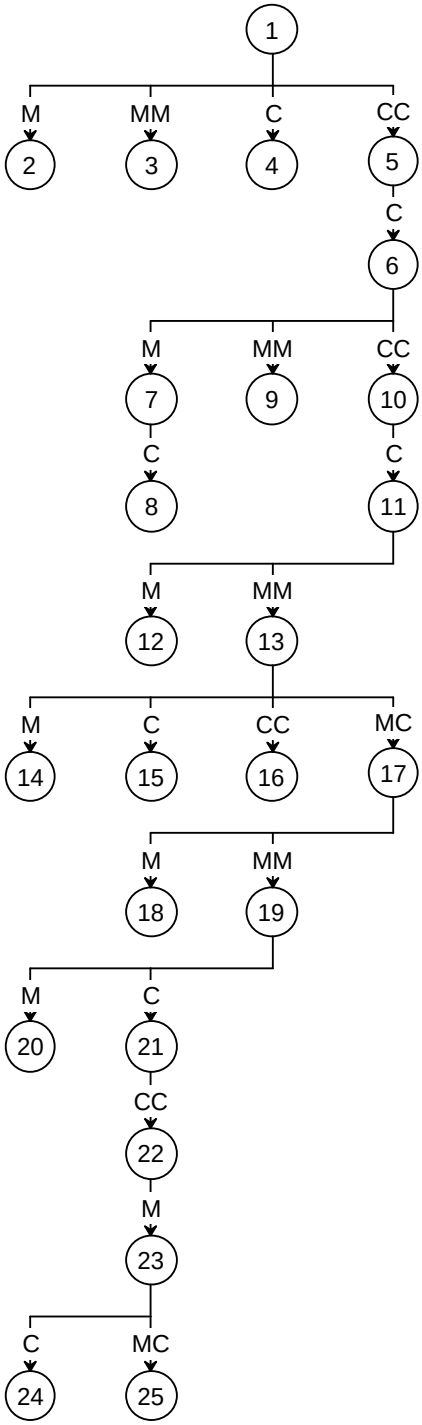
3. PENYELESAIAN DENGAN RUNUT-BALIK

Seperti yang telah disebutkan diatas, bahwa algoritma runut-balik banyak diterapkan dalam mencari penyelesaian dari suatu permainan, maka kali ini akan dibahas mengenai penyelesaiannya pada permainan "3 missionaries and 3 cannibals".

Properti :
1. Solusi permainan
   Permainan terselesaikan jika tiga missionaries dan tiga cannibals dapat sampai dengan selamat ke daratan di sisi kiri.
2. Fungsi pembangkit
   Membangkitkan segala kemungkinan langkah pada setiap langkah. Contohnya pada awal permainan, ada lima kemungkinan yang menaiki perahu.
3. Fungsi pembatas
   Fungsi untuk membatasi sejauh mana algoritma runut balik akan menelusuri setiap kemungkinan yang ada. Contohnya adanya batasan bahwa jumlah missionary tidak boleh lebih sedikit daripada jumlah cannibal dikedua sisinya. Tetapi jika jumlah missionary sama dengan nol, tidak apa-apa.

Tahap-tahap pencarian solusi :
1. Pada awal permainan, terdapat lima kemungkinan (selanjutnya akan disebut sebagai simpul) yang menaiki perahu, yaitu satu missionary, dua missionaries, satu cannibal, dua cannibals, atau satu missionary dan satu cannibal (selanjutnya akan disingkat menjadi M, MM, C, CC, MC).
2. Jika dicoba simpul pertama yaitu M, maka fungsi pembatas akan bernilai true karena jumlah missionary pada sisi kanan lebih sedikit daripada jumlah cannibal. Sehingga simpul pertama akan dipangkas dan akan dilakukan runut-balik dan mencoba simpul kedua.
3. Pada simpul kedua, MM juga sama dengan simpul pertama akan melanggar fungsi pembatas.
4. Di simpul ketiga, C tidak melanggar fungsi pembatas. Namun, tidak memiliki anak simpul lagi karena jika C kembali menaiki perahu, maka akan kembali ke kondisi awal permainan.
5. Pada simpul keempat, CC tidak melanggar fungsi pembatas. Kemudian anak simpulnya hanya satu yaitu C. Dan jika ditelusuri lagi, akan menghasilkan 5 anak simpul yang akan ditelusuri satu per satu lagi.
6. Begitu seterusnya, selama tidak melanggar fungsi pembatas, maka simpul tersebut akan ditelusuri terus sampai tidak ada anak simpulya lagi atau melanggar fungsi pembatas. Dan jika melanggar fungsi pembatas atau simpul tersebut tidak memiliki anak simpul lagi, maka akan dilakukan runut-balik sampai menemukan simpul yang belum ditelusuri.
7. Proses ini akan berhenti sampai menemukan solusi dari permainan atau semua simpul telah ditelusuri.

Dari tahap-tahap diatas, maka akan terbentuk suatu pohon status sebagai berikut.



Gambar 5. Pohon ruang status

Berikut alur singkat terbentuknya pohon ruang status diatas.

1. Simpul 1 (0:0<3:3 melambangkan perbandingan jumlah missionary dengan cannibal di sisi kiri (0:0) dan kanan (3:3) dan tanda < menandakan arah perahu akan berjalan berikutnya ke kiri atau kanan). Banyak anak simpul 5.
2. Simpul 2 (1:0>2:3). Melanggar fungsi pembatas. Runut-balik.
3. Simpul 3 (2:0>1:3). Melanggar fungsi pembatas. Runut-balik.
4. Simpul 4 (0:1>3:2). Tidak memiliki anak simpul lagi. Runut-balik.
5. Simpul 5 (0:2>3:1). Banyak anak simpul 1.
6. Simpul 6 (0:1<3:2). Banyak anak simpul 4.
7. Simpul 7 (1:1>2:2). Banyak anak simpul 1.
8. Simpul 8 (1:0<2:3). Melanggar fungsi pembatas. Runut-balik.
9. Simpul 9 (2:1>1:2). Melanggar fungsi pembatas. Runut-balik.
10. Simpul 10 (0:3>3:0). Banyak anak simpul 1.
11. Simpul 11 (0:2<3:1). Banyak anak simpul 4.
12. Simpul 12 (1:2>2:1). Melanggar fungsi pembatas. Runut-balik.
13. Simpul 13 (2:2>1:1). Banyak anak simpul 4.
14. Simpul 14 (1:2<2:1). Melanggar fungsi pembatas. Runut-balik.
15. Simpul 15 (2:1<1:2). Melanggar fungsi pembatas. Runut-balik.
16. Simpul 16 (2:0<1:3). Melanggar fungsi pembatas. Runut-balik.
17. Simpul 17 (1:1<2:2). Banyak anak simpul 4.
18. Simpul 18 (2:1>1:2). Melanggar fungsi pembatas. Runut-balik.
19. Simpul 19 (3:1>0:2). Banyak anak simpul 3.
20. Simpul 20 (2:1<1:2). Melanggar fungsi pembatas. Runut-balik.
21. Simpul 21 (3:0<0:3). Banyak anak simpul 1.
22. Simpul 22 (3:2>0:1). Banyak anak simpul 4.
23. Simpul 23 (2:2<1:1). Banyak anak simpul 2.
24. Simpul 24 (2:3>1:0). Melanggar fungsi pembatas. Runut-balik.
25. Simpul 25 (3:3<0:0). Permainan selesai

Secara algoritma, dapat ditulis menjadi :

```
// fungsi solusi
function IsSolved() : boolean
begin
  return (jumlah missionary di kiri == 3
  and jumlah cannibal di kiri == 3);
end;

// fungsi pembatas
function IsLose() : boolean
begin
  return (jumlah missionary di kiri <
  jumlah cannibal di kiri and jumlah
  missionary di kanan < jumlah cannibal di
  kanan);
end;

// fungsi pembangkit
```

```
procedure Solve(Simpul S)
begin
  if (IsSolved() then
    solved=true;
  else if (IsLose()) then
    runut-balik   ke   simpul   yang   belum
    dikunjungi;
  else
  begin
    i : integer;
    array A=anak-anak simpul S yang mungkin;
    for i=1 to A.length() do
      Solve(A[ i ]);
    endfor;
  endif;
end;
```

4. KESIMPULAN

Jika dilihat dari pohon ruang status di gambar 5, hanya dengan 25 tahap, sudah dapat menemukan solusi permainan ini. Jika dengan metode *exhaustive search* yang mencoba setiap kemungkinan dari setiap simpul dan akan membutuhkan waktu yang sangat lama untuk menelusuri setiap kemungkinannya. Jika dibandingkan, algoritma runut-balik jauh lebih efisien karena algoritma ini memangkas simpul-simpul yang tidak menuju solusi.

Selain itu, algoritma runut-balik ini juga jauh lebih cocok untuk menyelesaikan suatu permainan karena biasanya suatu permainan memiliki banyak sekali kemungkinan-kemungkinan yang bisa terjadi sehingga sangat tidak efisien ditelusuri semua kemungkinannya

REFERENSI

[1] Munir, Rinaldi. 2006. Strategi Algoritmik. Teknik Informatika ITB : Bandung

[2] http://www.informatika.org/~rinaldi

[3] Widya Wardani1, Dian Intania Savitri H2, Allentine Tanujaya3, “Analisis Penerapan Algoritma Backtracking Dalam Pencarian Solusi Game ‘Crossword Puzzle’”, IF2251-Strategi Algoritmik, 1, 2005, halaman 2s.