PENGGUNAAN ALGORITMA *BACKTRACKING* DALAM MENCARI SOLUSI PERMAINAN "BRIDGE CROSSING"

Anthony Rahmat Sunaryo

Program Studi Teknik Informatika,
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: if16009@students.if.itb.ac.id

ABSTRAK

Makalah ini membahas tentang pencarian solusi optimum pada sebuah permainan logika bernama "Bridge Crossing". Algoritma-algoritma yang dapat digunakan dalam memecahkan masalah pada permainan ini antara lain algoritma *brute force, greedy,* dan *backtracking*. Namun algoritma yang akan dianalisis dalam makalah ini dibatasi hanya pada algoritma *bactracking* saja. Analisis yang akan dilakukan adalah bagaimana solusi demi solusi dibangkitkan dengan menggunakan algoritma runut-balik dan bagaimana sebuah solusi dapat ditemukan. Pembahasan dengan algoritma runut-balik dilakukan karena algoritma ini selalu dapat menemukan solusi dari sebuah permasalahan yang melibatkan logika dalam skala kecil maupun besar yang tidak lagi dapat dipikirkan langsung oleh otak manusia.

Permainan "Bridge Crossing" merepresentasikan sebuah kondisi dimana diperlukan sebuah pemikiran yang cepat untuk mengambil satu keputusan yang tepat dari berbagai peluang yang mungkin. Permainan ini hanya sebuah contoh kasus dalam kehidupan sehari-hari yang tidak luput dari masalah yang serupa dengan permainan ini. Maka sangatlah penting utnuk mengetahui pemikiran atau algoritma yang baik untuk menyelesaikannya.

Kata kunci: *Backtracking*, algoritma, solusi, Bridge Crossing.

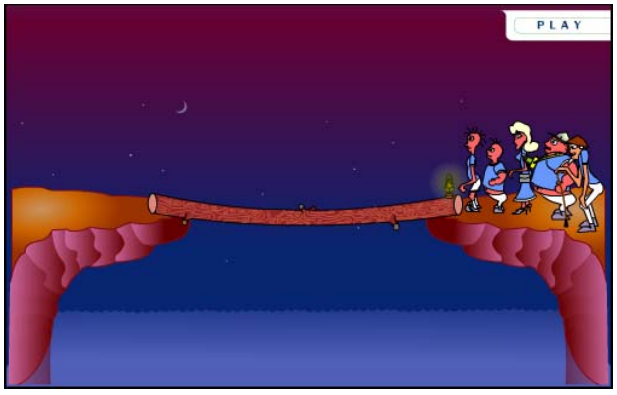
1. PERMAINAN "BRIDGE CROSSING"

Permainan "Bridge Crossing" adalah sebuah permainan yang mengandalkan logika untuk mencari solusi yang tepat. Berikut ini adalah peraturan pada permainan ini :

1. Terdapat *n* orang pada posisi start. Setiap *n* orang tersebut harus pindah dari posisi start ke posisi finish melalui sebuah jembatan penghubung.
2. Masing-masing dari *n* orang tersebut dapat menyebrangi jembatan dengan waktu penyeberangan *t*. Maksimal orang yang dapat menyebrangi jembatan adalah dua orang.
3. Lampu diperlukan untuk menyebrangi jembatan dan hanya terdapat satu buah lampu. Oleh karena itu, setiap pembawa lampu yang mencapai finish harus kembali ke posisi start jika masih ada orang yang berada di posisi start. Lampu hanya dapat dipegang oleh satu orang.
4. Terdapat batas waktu M untuk memindahkan semua orang dari titik start ke titik finish.

Dalam permainan "Bridge Crossing" ini, ditentukan spesifikasi sebagai berikut:

1. n = 5 orang = $\{n_1, n_2, n_3, n_4, n_5\}$.
2. t = $\{t_1, t_2, t_3, t_4, t_5\}$ = {1, 3, 6, 8, 12} menit
3. M = 30 menit
4. Himpunan solusi dicontohkan sebagai berikut ($\{n_1,n_2\},n_1,\{n_1,n_3\},n_1,\{n_1,n_4\},n_1,\{n_1,n_5\}$). Notasi {} memiliki arti dua orang menyeberang bersamaan, sedangkan yang tidak memakai notasi {} berarti orang yang kembali ke start.



Gambar 1. *Screenshot* permainan "Bridge Crossing"

2. ANALISIS PENGGUNAAN ALGORITMA *BACKTRACKING*

2.1 Algoritma *Backtracking*

Algoritma *Backtracking* sangat erat hubungannya dengan pembentukan pohon ruang status. Maka dari itu, pencarian solusi akan dilakukan dengan menelusuri simpul-simpul dalam pohon tersebut.

Algoritma *Backtracking* melibatkan algoritma *DFS* (*Depth-First Search*) dalam penelusuran simpul-simpul pada pohon sehingga penelusuran dilakukan sampai simpul terdalam terlebih dahulu. Secara umum, langkah-langkah dalam algoritma *Backtracking* adalah sebagai berikut :

1. Simpul yang sudah dilahirkan dinamai **simpul hidup,** sedangkan simpul yang sedang diperluas dinamai **simpul-E**. Pencarian solusi dilakukan dengan membentuk simpul akar sebagai **simpul-E** terlebih dahulu yang (simpul akar merupakan keadaan awal).
2. Perluas setiap simpul-E dengan menelusuri simpul-simpul anaknya sampai ditemukan simpul yang tidak mengarah ke solusi. Simpul tersebut dinamakan **simpul mati**. Sebuah fungsi yang digunakan untuk menandai sebuah simpul adalah simpul mati bernama **fungsi pembatas**. Simpul mati tidak akan diperluas lagi.
3. Jika menemukan simpul mati, proses perluasan dilakukan dengan membangkitkan simpul anak pada simpul bapak dari simpul mati yang ditemukan.
4. Pencarian selesai jika ditemukan sebuah solusi atau seluruh simpul sudah dibangkitkan.

```
procedure RunutBalikI(input n:integer)
{Mencari semua solusi persoalan dengan
algoritma backtracking-iteratif.
Input: n (panjang vektor solusi)
Output: solusi = (x[1], x[2],…, x[n])
}
Kamus
k : integer
Algoritma
k ← 1
while k > 0 do
   if (x[k] belum dicoba sedemikian
   sehingga x[k]←T(k)) and
   (B(x[1], x[2], ... ,x[k])= true)
   then
      if (x[1],x[2],...,x[k]) adalah
      lintasan dari akar ke daun then
       CetakSolusi(x)
      endif
   k←k+1
   else {x[1], x[2], …, x[k] tidak
   mengarah ke simpul solusi}
      k←k-1 {runut-balik ke anggota
      tupple sebelumnya}
   endif
endwhile { k = 0 }
```

2.2 Implementasian algoritma *Backtracking* pada permainan

Sebelum membuat pohon ruang status, perlu diperhatikan bahwa fungsi pembatas yang akan digunakan dalam algoritma *backtracking* harus dibuat terlebih dahulu. Dalam permainan "*Bridge Crossing*" disebutkan bahwa batas waktu M yang diberikan untuk menyeberangkan semua orang ke posisi finish adalah 30 menit. Oleh karena itu, dengan menganggap T sebagai waktu yang telah digunakan dan M sebagai batas waktu, maka berikut ini fungsi pembatasnya :

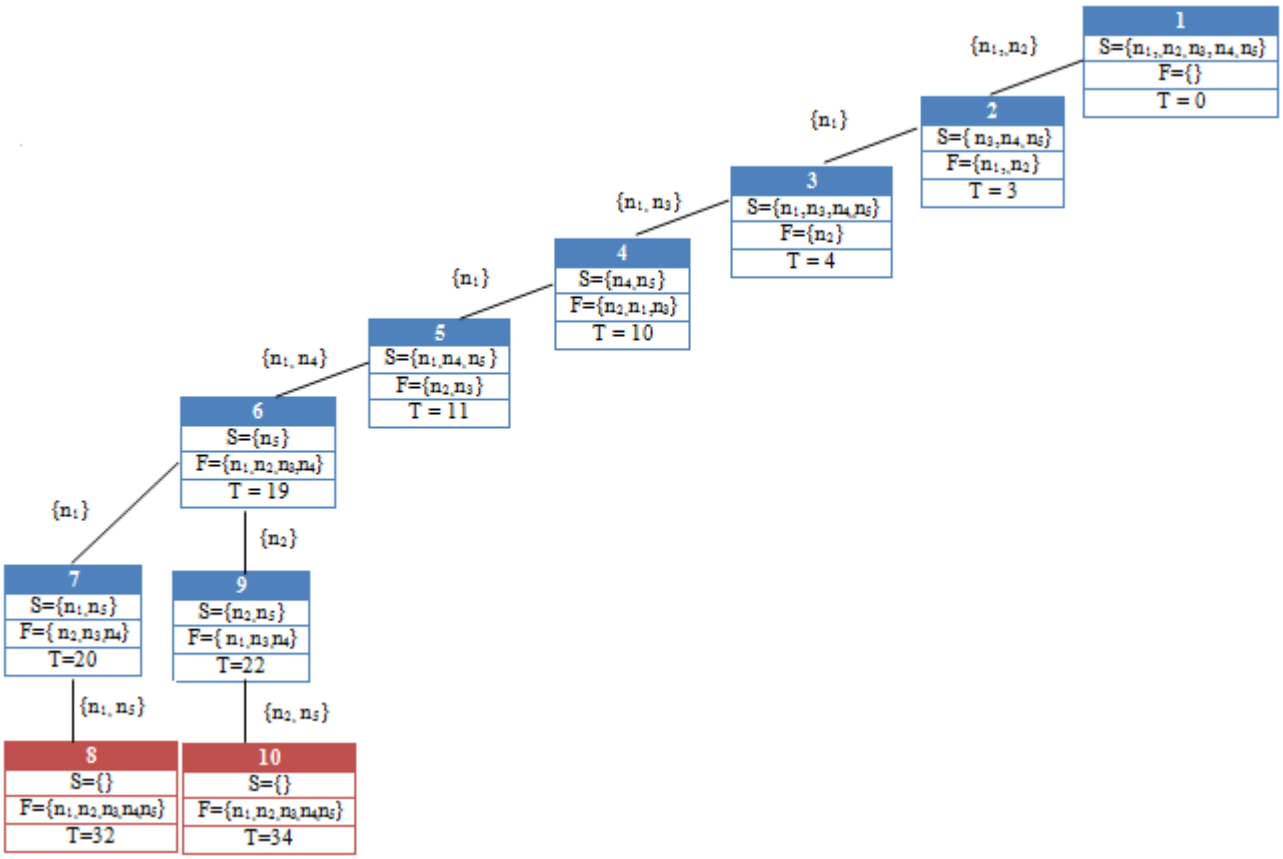
$$T \leq M$$

Setelah fungsi pembatas ditentukan, pohon ruang status dibuat. Namun, perlu diketahui bahwa cara penulisan simpul pada pohon ruang status yang akan dibuat adalah sebagai berikut :

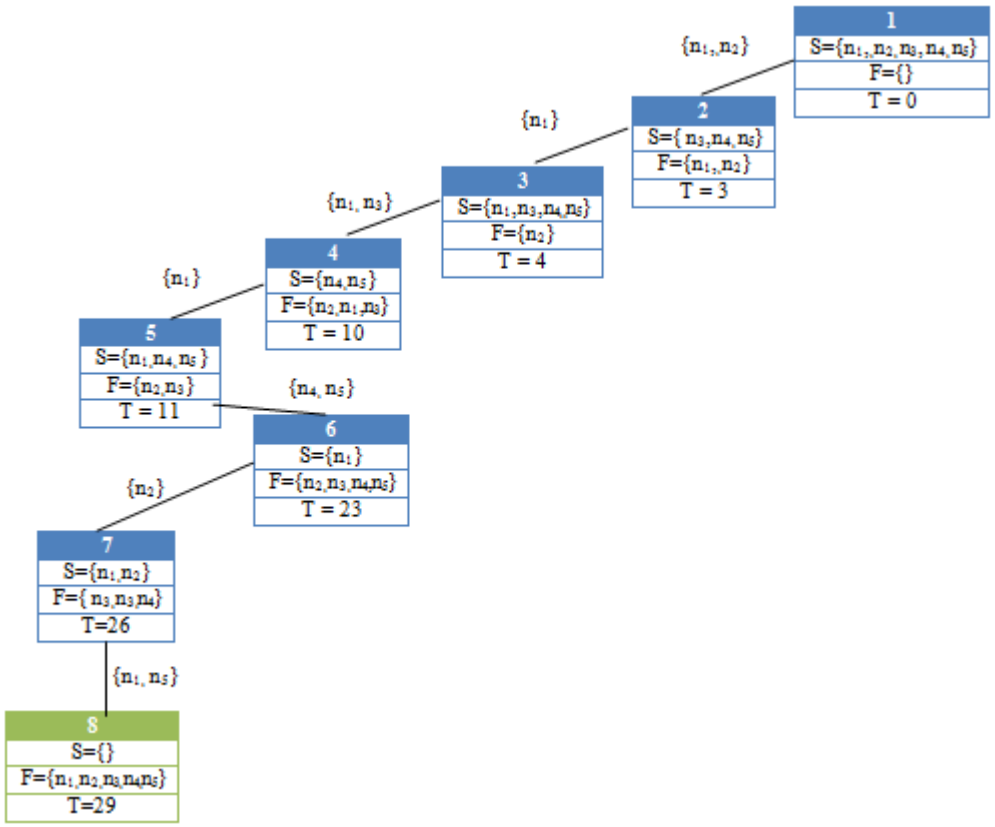
no. urut simpul
Himpunan orang di start
Himpunan orang di finish
Waktu yang telah digunakan

Gambar 2. Cara penulisan simpul

Simpul hidup ditandai dengan warna biru, simpul mati ditandai dengan warna merah, dan simpul solusi ditandai dengan warna hijau. Berikut ini adalah pohon ruang status yang merepresentasikan pencarian sebuah solusi dalam permainan "*Bridge Crossing*".



Gambar 3. Contoh penelusuran simpul yang berakhir pada ditemukannya solusi



Gambar 4. Sebagian pohon ruang status dalam pencarian solusi

Berikut ini adalah pengaplikasian algoritma dalam bentuk tabel :

#	Posisi start	Ke Finish	Ke Start	Posisi Finish	T
1	$\{n_1,n_2,n_3,n_4,n_5\}$	$\{\}$	$\{\}$	$\{\}$	0
2	$\{n_3,n_4,n_5\}$	$\{n_1,n_2\}$	$\{\}$	$\{n_1,n_2\}$	3
3	$\{n_1,n_3,n_4,n_5\}$	$\{\}$	$\{n_1\}$	$\{n_2\}$	4
4	$\{n_4, n_5\}$	$\{ n_1, n_3\}$	$\{\}$	$\{ n_1, n_2, n_3\}$	10
5	$\{n_1, n_4, n_5\}$	$\{\}$	$\{n_1\}$	$\{n_2, n_3\}$	11
6	$\{n_5\}$	$\{ n_1, n_4\}$	$\{\}$	$\{ n_1,n_2,n_3,n_4\}$	19
7	$\{n_1, n_5\}$	$\{\}$	$\{n_1\}$	$\{n_2,n_3,n_4\}$	20
8	$\{\}$	$\{n_1, n_5\}$	$\{\}$	$\{n_1,n_2,n_3,n_4,n_5\}$	32

#	Posisi start	Ke Finish	Ke Start	Posisi Finish	T
1	$\{n_1,n_2,n_3,n_4,n_5\}$	$\{\}$	$\{\}$	$\{\}$	0
2	$\{n_3,n_4,n_5\}$	$\{n_1,n_2\}$	$\{\}$	$\{n_1,n_2\}$	3
3	$\{n_1,n_3,n_4,n_5\}$	$\{\}$	$\{n_1\}$	$\{n_2\}$	4
4	$\{n_4, n_5\}$	$\{ n_1, n_3\}$	$\{\}$	$\{ n_1, n_2, n_3\}$	10
5	$\{n_1, n_4, n_5\}$	$\{\}$	$\{n_1\}$	$\{n_2, n_3\}$	11
6	$\{n_1\}$	$\{n_4, n_5\}$	$\{\}$	$\{n_2,n_3,n_4,n_5\}$	23
7	$\{n_1, n_2\}$	$\{\}$	$\{n_2\}$	$\{n_3,n_4,n_5\}$	26
8	$\{\}$	$\{n_1, n_2\}$	$\{\}$	$\{n_1,n_2,n_3,n_4,n_5\}$	29

Tabel 1. Contoh pencarian solusi dalam bentuk tabel

Dari gambar maupun tabeldi atas, dapat disimpulkan bahwa solusi $(\{n_1,n_2\},n_1,\{n_1,n_3\},n_1,\{n_1,n_4\},n_1,\{n_1,n_5\})$ tidak valid karena memakan waktu T lebih dari M, yaitu 30 menit. Solusi $(\{n_1,n_2\},n_1,\{n_1,n_3\},n_1,\{n_1,n_4\},n_2,\{n_2,n_5\})$ menunjukkan hal yang sama.

Jika proses pencarian solusi seperti di atas diteruskan, maka akan didapatkan satu buah contoh solusi yang valid, yaitu $(\{n_1,n_2\},n_1,\{n_1,n_3\},n_1,\{n_4,n_5\},n_2,\{n_1,n_2\})$ yang memakan waktu T hanya 29 menit.

3. KESIMPULAN

Algoritma runut-balik atau *backtracking* sangat efektif dan efisien untuk menyelesaikan permainan "*Bridge Crossing*" dan masalah-masalah lainnya yang serupa. Penyelesaian permainan ini sebenarnya masih dapat diselesaikan dengan kemampuan otak dan daya ingat manusia. Namun jika persoalan dikembangkan lebih jauh lagi, misalnya dengan jumlah orang yang bertambah, maka menyelesaikan persoalan dengan hanya mengandalkan kemampuan otak manusia akan sangat sulit. Untuk kasus ini, algoritma *Backtracking* adalah salah satu solusinya.

REFERENSI

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