PENERAPAN ALGORITMA RUNUT BALIK UNTUK PENYELESAIAN *TETRAVEX PUZZLE*

Ikhsan Fanani
13505123

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
e-mail: ikhsan_fanani@yahoo.com

ABSTRAK

Algoritma yang banyak diterapkan dalam penyelesaian permainan adalah algoritma *backtracking* (runut-balik). Dengan menggunakan algoritma ini, penyelesaian suatu permainan yang melibatkan banyak kemungkinan solusi dapat diselesaikan dengan cepat. Hal ini dimungkinkan karena algoritma *backtracking* adalah perbaikan dari algoritma *brute force*, dengan memangkas (*pruning*) simpul-simpul yang tidak mengarah ke solusi.

***Tetravex Puzzle* adalah salah satu permainan puzzle klasik yang dulu banyak dimainkan. Permainan ini cukup rumit untuk diselesaikan secara biasa karena membutuhkan ketelitian dan konsentrasi tinggi, terutama untuk puzzle dengan ukuran besar. Untuk itu, algoritma *backtracking* akan digunakan sebagai salah satu solusi penyelesaian *Tetravex Puzzle* ini.**

Kata kunci: *Tetravex Puzzle, Backtracking*.

1. PENDAHULUAN

Bermain *mini-game* dapat menjadi salah satu penghilang rasa jenuh ketika/setelah melakukan aktivitas yang memeras otak. Saat ini sudah tersedia ratusan, bahkan ribuan permainan *mini-game* gratis yang dapat kita peroleh dari dunia maya, salah satunya adalah *Tetravex Puzzle* ini.

Permainan *Tetravex Puzzle* memang sudah cukup tua. Namun bagi kalangan akademisi, terutama dalam bidang matematika dan *computer science*, permainan ini masih cukup menarik karena mampu melatih logika dan ketelitian mereka secara santai. Permainan-permainan lain yang banyak dimainkan adalah *Sudoku*, *Minesweeper*, dll.

Namun bagi sebagian besar orang, permainan seperti ini cukup rumit dan memusingkan, karena banyak berurusan dengan angka dan logika. Untuk itu, untuk menyelesaikan *puzzle* ini, dapat digunakan berbagai jenis algoritma, salah satunya adalah algoritma runut-balik (*backtracking)*. Algoritma ini dipilih karena cukup mangkus dalam menyelesaikan per-soalan-persoalan seperti ini. Puzzle ini dapat menghasil-kan pohon solusi yang cukup kompleks jika diselesaikan dengan cara biasa (*brute force*). Sedangkan dengan algoritma runut-balik, proses pencarian solusi dapat dipersingkat karena algoritma ini mematikan simpul-simpul yang tidak mengarah ke solusi.

2. TETRAVEX PUZZLE

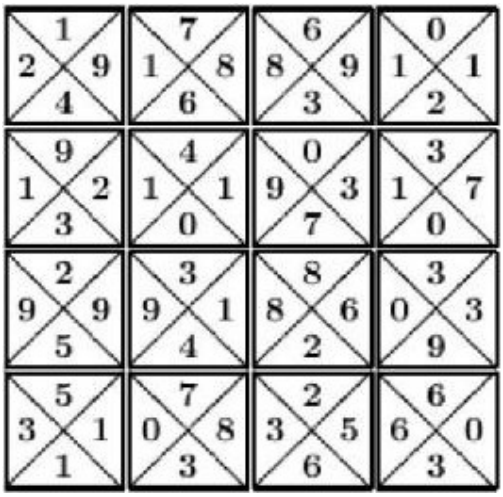
2.1 Gambaran Umum Tetravex Puzzle

Tetravex Puzzle adalah permainan menyusun kotak-kotak yang terdiri dari empat angka. Karena itulah permainan ini disebut Tetravex Puzzle (tetra = empat). Puzzle ini terdiri atas M x N buah kotak yang harus disusun sedemikian hingga angka-angka yang terdapat pada sisi kotak yang bersentuhan adalah angka yang sama, menjadi satu buah kotak besar [1]. Contoh pada Gambar 1 dan Gambar 2 adalah Tetravex Puzzle berukuran 4 x 4.

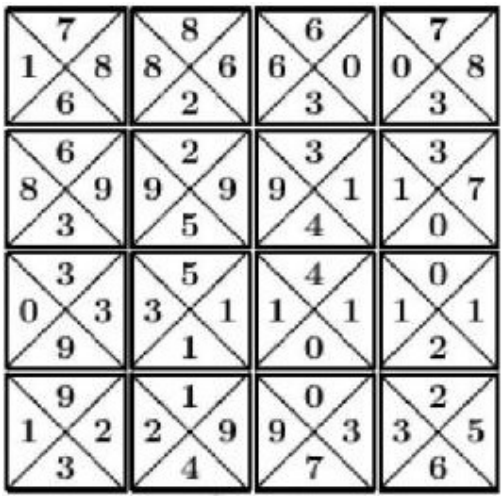
2.2 Strategi Umum Penyelesaian Tetravex

Umumnya permainan ini diselesaikan dengan teknik *brute force*, yaitu mencoba berbagai macam kemungkinan, ditambah dengan sedikit analisis. Untuk *puzzle* berukuran kecil, penggunaan algoritma *brute force* tidak begitu bermasalah. Namun untuk puzzle berukuran besar ($M >= 4$ dan $N >= 4$), permainan ini menjadi semakin rumit dan tidak begitu banyak orang yang berminat untuk menyelesaikannya.

Penambahan ukuran *puzzle* memberikan penambahan kemungkinan solusi yang sangat signifikan. Untuk *puzzle* berukuran $M \times N$, dengan algoritma *brute force*, terdapat $(M \times N)!$ kemungkinan penempatan kotak. Jumlah kemungkinan solusi akan meningkat secara faktorial, sehingga penggunaan algoritma umum ini tidak mangkus lagi.



Gambar 1. Keadaan awal tetravex puzzle



Gambar 2. Salah satu solusi tetravex puzzle

Untuk puzzle dengan ukuran M = 4 dan N = 4 seperti pada contoh diatas, maka jumlah kemungkinan solusi adalah :

$$(4 \times 4)! = 16! = \mathbf{20.922.789.888.000}$$

Jumlah tersebut bukanlah jumlah yang cukup kecil untuk diselesaikan oleh komputer. Sebagai perbaikan, dapat digunakan algoritma runut-balik yang mampu mengurangi kemungkinan solusi *puzzle*.

3. ALGORITMA RUNUT-BALIK

3.1 Gambaran Singkat

Istilah runut-balik pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Selanjutnya, R.J. Walker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik dan penerapannya pada berbagai persoalan. Saat ini algoritma runut-balik (*backtracking*) banyak diterapkan pada penyelesaian permainan seperti *sudoku*, *tic-tac-toe*, menemukan jalan keluar dari labirin, catur, serta masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

Algoritma runut-balik (*backtracking*) merupakan perbaikan dari algoritma *brute force*. Algoritma ini mencari solusi dari sekian banyak kemungkinan solusi yang ada. Semua solusi dibuat dibentuk dalam pohon solusi (pohon ini berbentuk abstrak) dan algoritma akan menelusuri secara DFS (*Depth First Search*). Algoritma ini memangkas simpul-simpul yang tidak mengarah ke solusi, sehingga waktu pencarian akan lebih singkat.

3.2 Properti Umum Metode

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan n-tuple :
   $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$

   Mungkin saja $S_1 = S_2 = \ldots = S_n$

   Contoh: $S_i = \{0, 1\}$, $x_i = 0$ 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
   Dinyatakan sebagai $B(x_1, x_2, \ldots, x_k)$

   B bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi

   Jika *true*, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, x_k)$ dibuang.

3.3 Prinsip Pencarian

Disini, kita hanya akan meninjau pencarian solusi pada pohon yang dibangun secara dinamis. Langkah-langkah pencarian solusi adalah sebagai berikut [2]:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian mendalam (*DFS*).

Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E (*expand node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.

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 orang tua). Selanjutnya, simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

4. PENERAPAN RUNUT-BALIK UNTUK PENYELESAIAN TETRAVEX PUZZLE

Implementasi algoritma runut-balik dilakukan dengan membuat pohon dari kotak-kotak *puzzle* yang disediakan. Fungsi pembatas mengecek apakah kotak yang disimpan memiliki angka-angka yang sama dengan angka dari kotak tetangga pada sisi yang bersentuhan.

Untuk implementasi yang lebih mudah, skema rekursif akan lebih baik dibandingkan skema iteratif. Berikut salah satu contoh implementasi :

```
Struktur Data dan Variabel

M, N  : integer {ukuran puzzle}
Kotak : array [1..4] of integer
        {1: kiri, 2:atas, 3:kanan,
         4:bawah}
Puzzle: array [1..M] of array [1..N] of
        Kotak
Solusi: array [1..M] of array [1..N] of
        Kotak
LKotak: array[1..M*N] of Kotak
```

Struktur data Puzzle yang digunakan adalah `Matriks of Kotak`, dengan Kotak bertipe `array [1..4] of integer`. Array pada Kotak merepresentasikan angka-angka yang terdapat pada sisi-sisi Kotak (1: bilangan pada sisi kiri, 2: sisi atas, 3: sisi kanan, 4: sisi bawah). LKotak adalah list dari kotak-kotak yang disediakan.

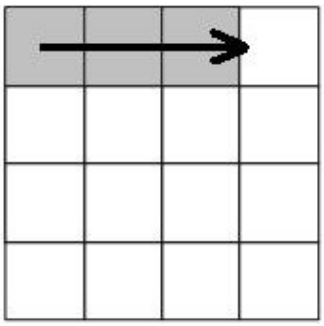
Implementasi dari algoritma penyelesaian (dengan skema rekursif) dan fungsi pembatas sebagai berikut:

```
procedure Solve (input baris: integer,
kolom: integer)
{ result true jika solusi ditemukan,
  dan mencetak ke matriks Solusi
  result false jika tidak ada solusi

  masukan: indeks baris dan kolom dari
  matriks puzzle
}

Deklarasi
  i          : integer
  result     : boolean

Algoritma
  result = false
  if baris = M + 1 then
  {solusi ditemukan}
    result = true
  else
    for tiap LKotak do
      if isCocok (Kotak) then
      {fungsi pembatas}
        Solusi[baris][kolom] = Kotak[i]
        Solve (nextBaris, nextKolom)
    if tidak ada kotak yg cocok then
      result = false
      Solusi[baris][kolom] = NULL
      {backtrack()}
```

```
function isCocok (input b: integer, k:
integer) → boolean
{ true jika semua sisi memiliki angka
  yang sama dengan angka pada sisi
  kotak yang bersinggungan
  false jika ada sisi yang tidak cocok

  masukan: b: indeks baris kotak
           k: indeks kolom kotak
}

Algoritma
  if bukan kotak paling kiri then
    return Kotak[1] = Solusi[b][k-1][3]

  if bukan kotak paling atas then
    return Kotak[2] = Solusi[b-1][k][4]

  return true
```

Pengecekan pada fungsi pembatas hanya dilakukan dengan membandingkan kotak dengan kotak pada sisi kiri dan atas, karena pengisian kotak dilakukan mengurut dari kiri ke kanan dan dari atas ke bawah. Sehingga ketika sebuah kotak diletakkan, maka pada sisi kanan dan bawahnya belum terdapat kotak lain.

Algoritma ini akan berhenti ketika ditemukan sebuah solusi, meskipun dimungkinkan adanya lebih dari satu buah solusi.



Gambar 3. Urutan pengisian kotak pada matriks solusi

5. KESIMPULAN

Salah satu (namun bukan satu-satunya) algoritma yang cukup mangkus dalam penyelesaian *Tetravex Puzzle* adalah algo-ritma runut-balik. Dibandingkan dengan algoritma *brute force*, algoritma ini lebih efisien karena memiliki fungsi pembatas yang membatasi arah pencarian solusi.

Secara umum, persoalan-persoalan yang memiliki banyak kemungkinan pengisian solusi seperti *Sudoku*, *Tetravex Puzzle*, *N-Queens*, dan *Maze* dapat diselesaikan dengan menggunakan algoritma runut-balik ini.

REFERENSI

[1] http://acm.uva.es/p/v9/954.html diakses pada Selasa, 22 Mei 2007.

[2] Rinaldi Munir, Algoritma Runut-balik (bagian 1).pdf.

[3] Rinaldi Munir, Algoritma Runut-balik (bagian 2).pdf