ANTIMAGIC PUZZLE

Alwi Afiansyah Ramdan – 135 08 099

Program Studi Teknik Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: alfiansyah.ramdan@gmail.com

ABSTRAK

Makalah ini membahas tentang Algoritma *Brute Force*, mengenai definisi dan karakteristiknya; Algoritma *Backtracking*, mengenai langkah-langkah, properti umum, dan skema umumnya; *Puzzle Antimagic*, mengenai pengertian, karakteristik dan kaidah-kaidahnya; serta penggunaan Algoritma *Brute Force* dan Algoritma *Backtracking* dalam menemukan solusi untuk *Antimagic Puzzle* dan membandingkan keduanya. *Antimagic Puzzle* adalah sebuah kotak nxn yang berisi angka 1..n^2 dimana setiap kolom, baris dan diagonal utamanya menghasilkan angka yang berbeda dan mengurut.

Kata kunci: *antimajic square, majic constant, backtracking, brute force*.

1. PENDAHULUAN

Permainan (*game*) adalah hal yang menarik untuk banyak orang di dunia. Banyak jenis permainan yang sudah dimainkan di dunia, mulai dari yang sederhana sampai yang rumit, mulai dari yang mengasah otak, sampai yang mengasah sistem motorik manusia.

Dewasa ini, mulai banyak permainan yang mengasah otak yang digemari oleh masyarakat dunia. Salah satunya adalah permainan *Sudoku*, yaitu permainan dengan tujuan akhir mengisi kotak berukuran 9x9 dengan angka dari 1 sampai angka 9 dimana di tiap kolom atau baris serta di tiap area tidak ada angka yang sama. Area ini adalah 9 area berukuran 3x3 yang berada di kiri atas, tengah atas, kanan atas, kiri tengah, tengah tengah, kanan tengah, kiri bawah, tengah bawah, kanan bawah.

Contoh lainnya adalah permainan *Minesweeper* yang biasanya ada pada *Operating System Microsoft Windows*, dimana pemain diminta mencari kumpulan bom yang disembungikan di balik *grid-grid* dengan memberinya angka-angka yang mengindikasikan jumlah bom yang ada di sekelilingnya dan menjinakkan bom tersebut aga tidak meledak. Dengan banyaknya permainan asah otak yang digemari oleh masyarakat dunia, maka mulai bermunculan permainan permainan baru yang tujuannya untuk mengasah otak dan mengetes kemampuan berfikir seorang pemain.

Banyak cara yang digunakan oleh orang-orang untuk menyelesaikan permainan-permainan ini. Misalnya secara *brute force* dimana pemain mencoba segala kemungkinan yang ada untuk mencapai solusi. Ada pula yang mencari solusi dengan cara lebih cerdas, misalnya menggunakan Algoritma *Greedy* dalam pencarian solusinya.

Salah satu permainan berupa *puzzle* yang ditujukan untuk mengasah otak dan mengetes/menguji kemampuan berpikir seseorang adalah *Antimagic Puzzle*.

Di sini, penulis mencoba mencari suatu pencarian solusi untuk permainan *Antimagic Puzzle* ini dengan menggunakan Algoritma *Backtracking*/Algoritma Runut-balik sebagai sarana untuk mencari solusi.

2. Algoritma *Brute Force*

Dalam pencarian suatu solusi, banyak strategi algoritmik yang dapat digunakan. Beberapa teknik dan strategi algoritmik tersebut adalah Algoritma *Brute Force* yang paling banyak digunakan oleh orang-orang di dunia. Metode ini adalah metode pemecahan masalah yang aling sederhana dan paling mudah digunakan.

2.1 Definisi Algoritma *Brute Force*

Brute force adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma *Brute Force* memecahkan masalah sengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*).

2.2 Karakteristik Algoritma *Brute Force*

Algoritma *Brute Force* umumnya tidak "cerdas", dan tidak mangkus, karena ia membtuhkan jumlah langkah yang besar dalam penyelesaiannya, terutama jika masalah yang dipecahkan berukuran besar (dalam hal ini ukuran masukannya). Terkadang Algoritma *Brute Force* disebut juga dengan algoritma naif.

Algoritma *Brute Force* seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan atau trik-trik khusus, biasanya akan membantu kita dalam manamukan algoritma yang lebih cerdas dan lebih mangkus.

Selain itu, algoritma ini sering kali lebih mudah diimplementasikan daripada algoritma yang lebih canggih, dan karena kesederhanaannya kadang-kadang Algoritma *Brute Force* dapat lebih mangkus (ditinjau dari segi implementasi).

3. Algoritma *Backtracking*

Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari 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. Denan metode ini klita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya waktu pencarian dapat dihemat. Runut- balik lebih alami dinyatakan dengan algoritma rekursif. Kadang – kadang disebutkan pula bahwa runut-balik merupakan bentuk *typical* dari algoritma rekursif.

Istilah Runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. R.J Walker, 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)).

Langkah-langkah dalam pencarian solusi menggunakan metode runut-balik:

a. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah metode pencarian mendalam atau DFS. Simpul-simpul yang sudah dibangkitkan disebut simpul hidup. Simpul hidup yang sedang diperluas disebut simpul-E (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai urutan kelahirannya.
b. 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. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas. Simpul yang sudah mati idak akan pernah diperluas lagi.
c. 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. Lintasan baru dibangun kenmbali sampai lintasan tersebut membentuk solusi.
d. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

3.1 Properti Umum Metode Runut-Balik

Untuk menerapkan metode runut-balik, properti berikut didefinisikan:

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 (pada beberapa persoalan fungsi ini dinamakan sebgai fungsi kriteria).
   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.

Semua kemungkinan solusi dari persoalan disebut ruang solusi (solution space). Secara formal dapat dinyatakan, bahwa xi ∈ Si, maka

$S_1 \times S_2 \times \ldots \times S_n$

disebut ruang solusi. Jumlah anggota di dalam ruang solusi adalah $| S_1 | \cdot | S_2 | \cdot \ldots \cdot | S_n |$.

3.2 Skema Umum Algoritma *Backtracking*

Di bawah ini menunjukkan skema umum algoritma runut balik:

```
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 ke daun then
         CetakSolusi(x)
       endif
      RunutBalikR(k+1){ tentukan nilai untuk
        [k+1]}
   endfor
```

T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi. B(x1, x2, …, xk) adalah fungsi pembatas menentukan apakah (x1, x2, …, xk) mengarah ke solusi.

4. *Antimagic Puzzle*

Antimagic Puzzle adalah sebuah *puzzle* yang terdiri dari *Antimagic Square* yang masih kosong dan harus diisi oleh pemain. *Antimagic Square* adalah sebuah *array of integer* dengan ukuran n x n dari 1 sampai n^2 sedemikian sehingga setiap baris, kolom dan digonal utamanya tidak menghasilkan angka jumlah yang sama dan jumlah tersebut merupakan sebuah angka terurut naik dengan selisih satu nilai.

Antimagic Square adalah kasus khusus untuk *Heterosquare*. *Heterosquare* adalah sebuah *square* yang terdiri dari n x n integer dari 1 sampai dengan n^2 sedemikian sehingga setiap baris, kolom dan diagonal utamanya memiliki angka jumlah yang berbeda. Contoh *antimagic square* dapat dilihat pada gambar di bawah ini.

15	2	12	4
1	14	10	5
8	9	3	16
11	13	6	7

21	18	6	17	4
7	3	13	16	24
5	20	23	11	1
15	8	19	2	25
14	12	9	22	10

10	25	32	13	16	9	
22	7	3	24	21	30	
20	27	18	26	11	6	
1	31	23	33	17	8	
19	5	36	12	15	29	
34	14	2	4	35	28	

14	3	34	21	47	29	22
43	16	13	25	6	26	44
30	48	24	8	12	9	45
10	5	11	38	49	46	19
4	41	37	36	33	27	1
39	17	40	20	7	35	23
31	42	18	32	28	2	15

49	16	50	10	19	28	24	56
42	43	11	15	44	38	55	5
25	21	48	46	9	37	6	63
29	47	8	40	51	30	52	1
45	22	54	23	20	34	2	62
14	59	18	33	41	26	61	13
36	12	58	32	27	64	3	35
17	39	7	57	53	4	60	31

52	19	81	22	29	15	42	31	76
61	10	67	23	54	79	25	33	16
57	9	71	24	38	1	51	47	75
26	78	7	69	66	77	13	27	12
39	21	74	20	37	17	49	55	64
8	65	4	62	50	34	73	41	40
56	68	2	63	14	72	35	44	6
53	30	60	32	36	3	46	43	58
11	70	5	59	48	80	28	45	18

Gambar 1. Enam contoh *Antimagic Square*

Misalkan didefinisikan sebuah bilangan yang disebut dengan *magic constant*:

$$M(n) \equiv \frac{1}{2} n\left(n^2 + 1\right) \quad (1)$$

Maka jika ada sebuah *antimagic square* dengan order n, maka rentang angka jumlah positifnya adalah

$$[M(n) - n, M(n) + n + 1] \quad (2)$$

Antimagic square dengan orde 1, 2 dan 3 tidak eksis. Sehingga *Antimagic Puzzle* hanya bisa dibuat mulai dari orde 4.

5. Hasil dan Analisis

Akan dibahas dua macam metode dalam pencarian solusi untuk permasalah *antimagic puzzle* ini, yaitu dengan Algoritma *Brute Force* dan dengan Algoritma *Backtracking*.

Permainan ini dapat diselesaikan oleh pemain dengan menggunakan Algoritma *Brute Force*, mengingat tidak adanya batasan waktu (pemain bisa bermain sampai kapanpun ia mau). Dengan menggunakan metode tersebut, seorang pemain sudah pasti akan mendapatkan solusi *puzzle* yang sedang ditanganinya.

Misal kita memiliki variabel t yang bertipe integer, dan $t = n^2$ Dengan menggunakan algoritma ini, seorang pemain akan bisa menyelesaikan *puzzle* dalam waktu minimal P(t,t) = (t)!. Misal nilai n adalah 5 dan waktu pengisian *square* adalah 1 detik tanpa adanya waktu untuk

berpikir, maka paling tidak pemain baru bisa menyelesaikan permainan dalam waktu 1.5512E25 detik.

Secara garis besar, algoritma yang dipakai adalah seperti di bawah ini:

```
While solusi belum ditemukan do
      Kosongkan square
      for setiap kotak di square P do
           integer a = angka terkecil
             yang belum ada di square P
           isi square P dengan angka a
           lanjut ke kotak berikutnya
      endfor
endwhile
```

Selain dengan menggunakan Algoritma *Brute Force*, permainan pun bisa diselesaikan dengan menggunakan Algoritma *Backtracking*.

Pada permainan ini, penggunaan algoritma runut-balik sebagai pencarian solusi dapat dipandang sebagai pembentukan pohon ruang status. Akar pohon ruang status merupakan kondisi dimana *square* masih kosong, dan daun-daunnya merupakan kondisi dimana *square* telah terisi penuh dan setiap kotak dalam *suqare* memenuhi kaidah *antimagic square*.

Permasalahan dalam permainan ini adalah banyaknya kemungkinan yang dapat terjadi dengan persebaran angka-angka yang berbeda satu sama lain. Pemain harus memilih suatu angka untuk ditempatkan di suatu tempat tertentu dalam kotak yang ada di dalam *square*. Setalah itu, pemain harus memilih angka lainnya untuk ditempatkan di kotak berikutnya dalam *square* sedemikian sehingga jumlah angka-angka tersebut berada pada rentang yang semestinya ketika telah memasukkan n buah angka yang sama dalam sebaris, sekolom atau satu diagonal.

Secara garis besar, algoritma yang digunakan untuk memecahkan permasalahan ini adalah sebagai berikut:

```
if bukan solusi
then
       masuk ke kotak pertama
       while belum mencapai solusi do
              for semua a = 1 sampai dengan a =
                  n^2 do
                    if a layak masuk ke dalam
                       square
                    then
                          isi square dengan a
                          pindah ke kotak
                             berikutnya
                    endif
                    else if a tidak layak
                       masuk ke dalam square
                    then
                                backtrack
                    endif
              endfor
       endwhile
endif
else
       solusi ditemukan {semua kotak dalam
              square terisi dan memenuhi kaidah
              antimagic square}
```

Untuk mempermudah, kita gunakan fungsi-fungsi didefinisikan, diantaranya adalah:

```
procedure buatSquare(input: int n;
output: int[n][n] P)
{menghasilkan square P kosong}

function rentangJumlah (int n) →
int[2]
{mengembalikan nilai minimum dan nilai
maksimum untuk rentang angka jumlah
dalam square ukuran n x n, dimana
rentangnya adalah dari 1/2n(n^2+1)-n
sampai dengan 1/2n(n^2+1)+n+1}

function apakahSolusi(int[n][n] P) →
boolean
{menghasilkan true jika square P sudah
mencapai solusi, false jika belum}

function layakMasuk(int[n][n] P, int a)
→ boolean
{menghasilkan true jika a layak untuk
dimasukkan ke dalam square atau false
jika tidak, layak apabila a belum ada
di dalam square, angka jumlah baris,
kolom dan diagonalnya tidak melebihi
batas maksimum rentang jika a
dimasukkan}

procedure isiSquare(input/output:
int[n][n] P;input: int a)
{memasukkan nilai a ke kotak di square
P dan bergerak ke kotak selanjutnya
yang belum terisi}

procedure backtrackSquare(input/output:
int[n][n] P)
{mengembalikan square P ke keadaan
state sebelumnya}
```

Permasalahannya sekarang adalah kotak mana di dalam *square* yang sedang berusaha untuk diisi. Solusinya adalah dengan membuat suatu variabel global yang manandakan kotak mana yang sedang aktif. Kotak tersebut akan dipindahkan ketika prosedur isiSquare

dijalankan. Dan dibuat prosedur tambahan yang mentransformasikan dari kotak ke posisi di *square*.

```
procedure squareToKotak(input:
int[n][n] P; output: kotak K)
{mencari posisi di square yang belum
terisi dan menyimpannya di dalam kotak
yang merupakan penyimpan posisi}
```

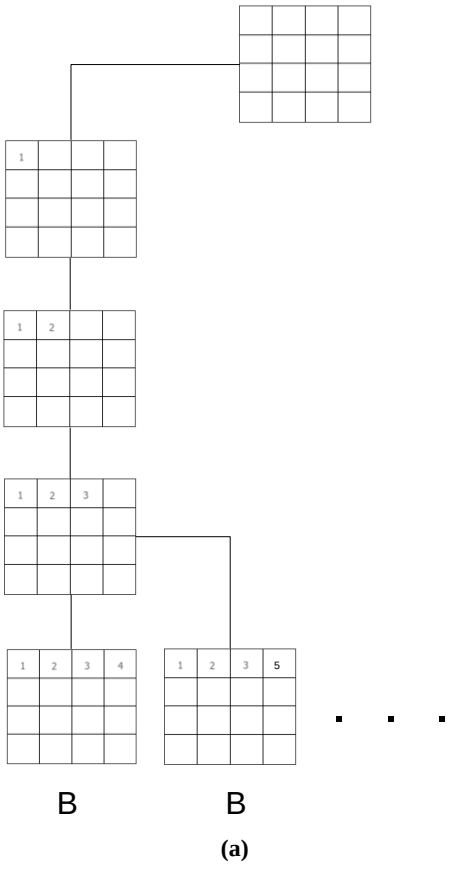
Dengan menggunakan fungsi-fungsi tersebut di atas, dapat dibuat Algoritma *Backtracking* untuk permainan ini, yaitu sebagai berikut:

```
function antimagicSquare (int n) →
boolean
{mengeluarkan true jika solusi
ditemukan dan false jika tidak}

Deklarasi
typedef kotak{
      x :   int
      y :   int
}
kotak K {sebagai variabel yang
     menyimpan posisi dalam square}
int[n][n] P {inisiasi square}

Algoritma
buatSquare(n, P) {membuat square baru}
squareToKotak(P,K) {mengambil posisi
    kotak, pertama kali digenerate
    secara acak}
if apakahSolusi(P) then
      return true
else
      while not apakahSolusi(P) do
            for int a = 1 to n^2 do
                if layakMasuk(P,a) then
                     isiSquare(P,a){meng
                        isi kotak dan
                        memindahkan
                        posisi ke kotak
                        yang baru}
                else
                     backtrackSquare(P)
                endif
            endfor
      endwhile
      return false {semua kemungkinan
          sudah dicoba tapi tatap tidak
          menemukan solusi}
endif
```

Pohon ruang status yang terbentuk berdasarkan pada DFS (*Depth First Search*), yaitu pencarian secara mendalam dimulai dari akar sampai daun. Jika simpul tidak mendekati ruang solusi, maka akan dilakukan runut-balik (*backtracking*) ke simpul tetangga yang belum ditelusuri.

Misal $t = n^2$. Di setiap simpul, akan ada kemungkinan muncul t simpul anak berikutnya, dan seterusnya sampai semua kotak dalam square penuh. Hal ini akan memberikan jumlah maksimum ruang status di dalam pohon ruang status menjadi t^t. Namun adanya fungsi pembatas yang membatasi penambahan angka dengan jumlah tertentu untuk setiap baris, kolom, dan diagonal utamanya mengurangi jumlah ruang status menjadi 2^t atau t!. Karena di setiap simpul ada pemanggilan proses rekursif, maka kasus terburuk yang dapat terjadi membutuhkan waktu sebesar $O(p(t)t!)$ atau $O(q(t)2^t)$ dimana p(t) dan q(t) adalah polinom berderajat t yang menyatakan waktu eksekusi setiap simpul.

Di bawah ini adalah contoh pohon ruang status yang terbentuk untuk permasalahan *Antimagic Puzzle*. (Gambar 2.a)



(a)

1	13	3	12
15	9	4	10
7	2	16	8
14	6	11	5

(b)

Gambar 2. Ilustrasi pohon ruang status yang terbentuk. (a) Percobaan pengisian *square*. (b) *Antimagic Square* yang telah komplit.

6. KESIMPULAN

Penggunaan Algoritma *Brute Force* dan Algoritma *Backtracking* dapat diimplementasikan di dalam berbagai permainan, salah satunya adalah permainan *Antimagic Puzzle*. Penggunaan Algoritma *Brute Force* dalam *puzzle* akan memakan waktu yang relatif lebih lama daripada penggunaan Algoritma *Backtracking* yang bekerja dengan lebih cerdas.

REFERENSI

[1] Antimagic Square, Wikipedia – Free Encyclopedia, http://en.wikipedia.org/wiki/Antimagic_square, 2010. Tanggal akses: 5 Desember 2010, pukul 10.00 WIB.

[2] Antimagic Square – Wolfram Mathworld, http://mathworld.wolfram.com/AntimagicSquare.html, 2010. Tanggal akses: 5 Desember 2010, pukul 10.00 WIB.

[3] Heterosquare – Wolfram Mathworld, http://mathworld.wolfram.com/Heterosquare.html, 2010. Tanggal akses: 5 Desember 2010, pukul 10.00 WIB.

[4] Rinaldi Munir, "Diktat Kuliah IF3051, Strategi Algoritma", Program Studi Teknik Informatika, STEI ITB, 2008.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalahorang lain, dan bukan plagiasi.

Bandung, 5 Desember 2010

Alwi Alfiansyah Ramdan
135 08 099