Penerapan Algoritma Runut-Balik pada *Magic Square*

Joshua Bezaleel Abednego 13512013[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512013@std.stei.itb.ac.id

***Abstrak*—*Magic square* merupakan salah satu hal yang menarik untuk ditelaah dalam dunia ilmu matematika. Magic square terdiri dari n x n kotak yang diisi angka 1 sampai n2 dimana memiliki karakteristik unik, yaitu jumlah angka dalam satu baris, kolom dan diagonalnya sama. Dalam makalah ini, penulis mencoba untuk mengaplikasikan algoritma backtracking karena dapat mempersingkat waktu penelusuran dan menghasilkan program yang lebih mangkus.**

***Indeks*—Backtracking, strategi algoritma, *magic square*.**

I. PENDAHULUAN

Dunia matematika tidaklah hanya mengenai rumus sulit belaka. Dalam ilmu matematika, terdapat suatu istilah yaitu matematika rekreasional. Matematika rekreasional merupakan hal-hal yang menyangkut ilmu kematematikaan yang lebih condong untuk dipakai sebagai rekreasi dibandingkan sesuatu hal yang serius. Hal inilah yang sering tidak diperhatikan baik oleh para murid ataupun guru di dunia pendidikan Indonesia, bahwa banyak hal indah yang dapat dipelajari dari ilmu-ilmu alam di sekitar kita.

Salah satu contoh dari matematika rekreasional yang cukup akrab didengar oleh pelajar adalah *magic* square atau yang lebih dikenal dengan persegi ajaib. *Magic square* merupakan sebuah kotak table berukuran n x n yang berisi angka-angka yang biasanya merupakan bilangan bulat. Setiap dari angka-angka ini merupakan angka berbeda yakni dari 1 hingga n^2. Yang membuat *magic square* menarik adalah jumlah dari angka-angka pada tiap baris, kolom dan juga diagonalnya sama.

Penulis merasa bahwa algoritma *backtracking* yang telah penulis pelajari pada kuliah Strategi Algoritma IF2211 dapat diterapkan untuk mencari solusi dari *magic square* ini untuk menghasilkan proses yang lebih mangkus dibandingkan secara *trial and error*. Penulis berharap dengan adanya makalah ini dapat menarik ketertarikan lebih dari siswa-siswa di Indonesia berbagai kalangan terhadap matematika rekreasional dan juga algoritma, sekaligus membuktikan bahwa algoritma *backtracking* dapat diterapkan untuk mencari solusi dari permasalahan *magic square.*

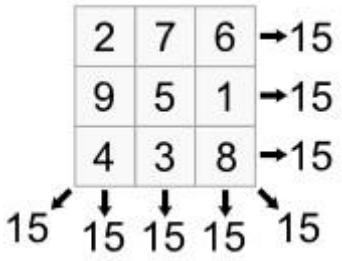
II. DASAR TEORI

2.1. *Magic square*

Dalam ilmu matematika, sebuah *magic square* atau yang lebih dikenal dalam bahasa Indonesia dengan peregi ajaib merupakan suatu susunan dari angka-angka berbeda (setiap angka hanya dipakai sekali), biasanya bilangan bulat, dalam sebuah kotak persegi, dimana setiap angka dalam tiap baris, dalam tiap kolom, dalam tiap diagonal utama, semuanya menghasilkan jumlah angka yang sama[2].

Setiap *magic square* mempunyai jumlah baris dan kolom yang sama. Dalam notasi matematika, *n* merupakan banyak baris(dan kolom) yang dipunyai, sehingga setiap *magic square* mempunyai n^2 angka dan ukurannya dalam ukuran n. Sebuah *magic square* yang mengandung bilangan bulat dari 1 hingga n^2 disebut *normal magic square*.

Contoh dari solusi *magic square* berukuran 3x3 adalah sebagai berikut :



Gambar 1 Contoh magic square berukuran 3x3

Setiap konstanta yang merupakan penjumlahan dari baris, kolom dna diagonal disebut *magic constant* atau disebut konstanta ajaib, M. Setiap *normal magic square* mempunyai sebuah konstanta unik yang ditentukan dari nilai *n*, yang merupakan ukuran dari *magic square* tersebut yang dapat diperoleh dari rumus berikut :

$$M = \frac{n(n^2+1)}{2}.$$

Gambar 2 Rumus mencari Magic Constant

Sebagai contoh, untuk n = 3, maka dengan rumus berikut diperoleh nilai M = $[3(3^2+1)]/2$ yang merupakan 15. Dapat dilihat pada gambar di atas bahwa nilai penjumlahannya merupakan 15.

2.1.1 Sejarah

Magic square pertama kali diketahui oleh matematikana Cina sekitar 650 SM, dan oleh matematikawan Arab sekitar awal abad ke 7. Literatur Cina sekitar 650 SM menceritakan sebuah cerita legenda tentang Lo Shu atau "gulungan sungai Lo". Menurut legenda, saat itu terjadilah banjir besar di Cina kuno. Selagi raja Yu mencoba untuk menyalurkan air banjir tersebut ke laut, seekor kura-kura keluar dari dalam air dengan pola aneh di punggungnya, sebuah table 3x3 dengan titik melingkar yang jika angkanya diurutkan, membentuk pola seperti *magic square*, yaitu jumlah baris, kolom dan diagonalnya sama, yaitu 15, yang juga merupakan banyak hari dalam 24 siklus dari tahun matahari Cina.

4	9	2
3	5	7
8	1	6

Gambar 3 Persegi Lo Shu

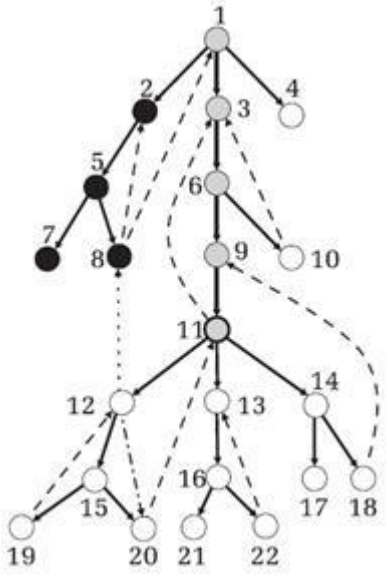
Persegi Lo Shu, seperti *magic square* pada punggung kura-kura tersebut, merupakan *normal magic square* yang unik berukuran 3x3 dimana angka 1 berada di paling bawah dan angka 2 berada di pojok kanan atas. Semua *normal magic square* yang berukuran 3x3 lainnya didapatkan dari persegi Lo Shu berdasarkan rotasi atau pencerminan.

2.2. Algoritma pencarian mendalam (DFS)

Algoritma pencarian mendalam atau yang lebih dikenal dengan Depth First Search adalah sebuah cara untuk mentraversal sebuah graf. Prinsip dari algoritma DFS adalah menelusuri secara mendalam selama penelusuran masih memungkinkan[3].

Awalnya semua simpul belum ada yang dikunjungi. DFS dimulai dari simpul akar dengan mengikuti langkah-langkah sebagai berikut :

1. Tandai simpul u
2. Untuk setiap sisi (u,v) dimana v belum dikunjungi jalankan DFS untuk u baik secara rekursif atau bisa juga iterative
3. Tandai simpul u sebagai simpul yang sudah diproses dan lakukan backtrack pada simpul parent.



Gambar 4 Ilustrasi algoritma DFS

2.3. Algoritma runut balik (*backtrack*)

Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus[1]. Runut-balik, yang merupakan perbaikan dari algoritma *brute-force.* Pada *brute-force*, semua kemungkinan solusi dieksplorasi satu per satu sedangkan pada backtracking hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan kembali. Istilah *backtracking*, pertama kali diperkenalkan oleh D.H. Lehner pada tahun 1950 dan terdapat juga tokoh-tokoh seperti R.J. Walker, Golomb, dan Baumert yang menyajikan uraian umum tentang *backtracking*.

Algoritma *backtracking* banyak diterapkan untuk berbagai program games seperti :

- Tic-tac-toe
- Maze (labirin)
- Catur
- Knight's tour
- Persoalan 8 ratu
- Persoalan kecerdasan buatan (*artificial intelligence*)

2.3.1 Properti Umum Algoritma *Backtracking*

Algoritma *backtracking* memiliki property umum yakni sebagai berikut :

1. **Solusi Persoalan**
   Solusi dinyatakan dalam bentuk veltor dengan *tuple* : $X = (x_1,x_2,x_3,\ldots,x_n)$, $x_i \in S_i$
   Mungkin terjadi $S_1 = S_2 = \ldots. = S_n$
   Contoh : $S_1 = \{0,1\}$, $x_i = 0$ atau 1

2. **Fungsi Pembangkit**
   Fungsi pembangkit nilai x_k.
   Dinyatakan dalam predikat T(k) dimana T(k) membangkitkan nilai untuk x_k, yang merupakan

komponen vektor solusi.

3. **Fungsi Pembatas**
   Dinyatakan dalam predikat : $B(x_1,x_2,\ldots,x_k)$
   Fungsi pembatas menentukan apakah $(x_1,x_2,\ldots,x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak, maka $(x_1,x_2,\ldots,x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

   Fungsi pembatas tidak selalu dinyatakan sebagai fungsi matematis. Ia dapat dinyatakan sebagai predikat yang bernilai *true* atau *false*, atau dalam bentuk lain yang ekivalen.

2.3.2 Prinsip Pencarian Solusi dengan Metode *Backtracking*

-Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan *depth-first order* (DFS).

-Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*)

-Simpul-simpul yang sedang diperluas dinamakan simpul-E (*expand node*)

-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.

-Simpul yang sudah mati tidak pernah diperluas lagi

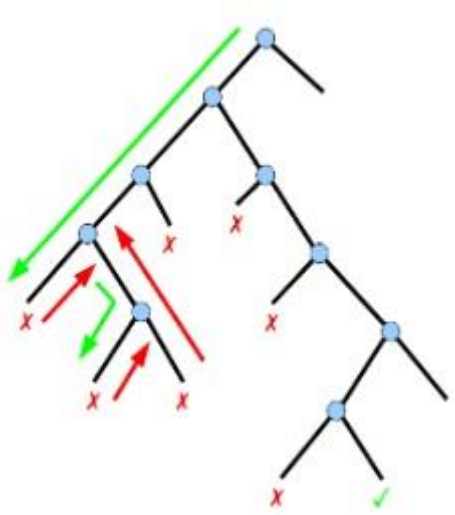
- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul aras diatasnya.

- Lalu, teruskan dengan membangkitkan simpul anak lainnya.

-Selanjutnya simpul ini menjadi simpul-E yang baru.

- Pencarian dihentikan bila kita telah sampai pada *goal node*.

Berikut ini adalah ilustrasi dari algoritma *backtrack* pada struktur data pohon :



Gambar 5 Ilustrasi algoritma backtracking

2.3.3. Kompleksitas Waktu Algoritma *Backtracking*

-Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.

- Jika jumlah simpul dalam pohon ruang status adalah 2^n atau n!, maka untuk kasus terburuk, algoritma backtrack membutuhkan waktu dalam :

- $O(p(n)2^n)$ atau
- $O(q(n)n!)$

- Dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi simpul.

2.3.4. Skema Umum Algoritma *Backtracking*

Di bawah ini disajikan skema umum algoritma runut-balik dalam dua versi rekursif dan versi iteratif. Skema dalam versi rekursif lebih tepat karena algoritma runut-balik lebih alami dinyatakan dalam bentuk rekursi. Algoritma di bawah ini akan menghasilkan semua solusi.

2.3.4.1 Skema Umum Algoritma *Backtracking* versi Rekursif

```
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 x[k+1]}
    endfor
```

Pemanggilan prosedur pertama kali : RunutBalikR(1)

2.3.4.2 Skema Umum Algoritma *Backtracking* versi Iteratif

```
procedure RunutBalikI(input n: integer)
 {Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif}
  Masukan : n, yaitu panjang vektor solusi
  Keluaran : solusi x = (x[1],x[2],…,x[n])
 }
Deklarasi :
  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 {indeks anggota tuple berikut}
          else {x[1],x[2],…,x[k] tidak mengarah ke solusi}
                 k ← k-1 {runut balik ke anggota tuple sebelumnya}
          endif
   endwhile
   {k = 0}
```

Pemanggilan prosedur pertama kali : RunutBalikI(n)

Prosedur CetakSolusi adalah sebagai berikut :

```
procedure CetakSolusi(input x : TabelInt)
{Mencetak solusi persoalan
 Masukan : x[1],x[2],…,x[n]
 Keluaran : nilai x[1],x[2],…,x[n] tercetak ke layar
}

Deklarasi :
   k : integer

Algoritma :
   for k ← 1 to n do
          write(x[k])
   endfor
```

III. IMPLEMENTASI BACKTRACKING PADA MAGIC SQUARE

Algoritma runut-balik dapat diimplementasikan pada kecerdasan buatan (artificial intelligence) untuk menghasilkan solusi dari *magic square*. Untuk kasus ini penulis akan mengambil contoh dari *normal magic square* yang berukuran 3x3.

Algoritma backtracking dipilih untuk menyelesaikan persoalan ini karena pemilihan dari algoritma ini dapat menghilangkan kemungkinan dari solusi yang sudah tidak mungkin, yang pada kasus ini jumlah dari barisnya lebih dari 15 atau angka tersebut sudah pernah muncul. Dalam permainan ini algoritma backtracking akan membangun beberapa solusi dari step pertama yang dipilih dan diperluas tiap solusi tersebut. Jika solusi yang diperluas tidak mencapai hasil maka algoritma ini akan melakukan backtracking dan mencari solusi dari solusi parsial lainnya.

Tahap algoritma backtracking dalam persoalan magic square adalah sebagai berikut :

1. Dari kotak pertama yang dipilih yakni kotak pertama paling pojok kiri atas, akan dicoba kemungkinan angka yang bisa dimasukkan di sana dari yang paling kecil, yakni dari angka 1.
2. Setelah solusi tersebut dipilih, akan dibangkitkan langkah berikutnya yakni di kotak kedua dan seterusnya.
3. Setiap ada baris atau kolom yang sudah terisi semua elemnnya, akan di cek apakah memenuhi fungsi pembatas atau tidak. Jika tidak, solusi tersebut akan backtrack ke solusi parsial sebelumnya dan begitu seterusnya sehingga solusi ditemukan.

Berikut merupakan mekanisme *backtracking* yang dipakai untuk menyelesaikan persoalan *magic square* 3x3 :

```
if (semua kotak telah terisi) then
        print solusi
else
```

a. Tempatkan angka yang belum dipilih pada kotak yang sedang ditunjuk. Untuk pencobaan pertama yaitu angka 1 di kotak pertama.
b. Periksa apakah penempatan tersebut diperbolehkan
c. Jika tidak ada pelanggaran, maju ke sel berikutnya.
d. Tempatkan angka dari yang paling kecil pada sel tersebut, dan periksa apakah ada pelanggaran.
e. Jika terdapat pelanggaran, angka tersebut ditambahkan dan coba ditempatkan lagi, serta dicek kembali.
f. Jika tidak ada angka yang memungkinkan, tinggalkan kotak tersebut dalam keadaan kosong dan kembali ke kotak sebelumnya. Nilai dari kotak tersebut dinaikkan

Pseudocode yang dibuat adalah sebagai berikut :

```
#include <stdio.h>
using namespace std;

bool IsValid(int a[10][10], int x, int y);
int MagicConstant(int size);

int main ()
{
        /*
        int size;
        cout << "Masukkan ukuran dari magic square: ";
cin >> size;
```

```
        */
        //Dalam kasus ini hanya akan dipakai normal magic square berukuran 3 x 3
        int size=3,i=1,j=1,counter;
        int MagicCons = MagicConstant(size);
        int MagicSquare[size+1][size+1]; // array dimulai dari 0
        //Inisialisasi matriks dengan 0
        for(i=1;i<=size;i++)
        {
                for(j=1;j<=size;j++)
                {
                        MagicSquare[i][j] = 0;
                }
        }
        //Selama sebelum dicek sampai kolom terakhir
        while(i<=size && j<=size)
        {
                if(MagicSquare[i][j]==0)
                {
                        counter = 1;
                }
                MagicSquare[i][j] = counter;
                //Pengecekan valid atau tidaknya penempatan jawaban
                while(!IsValid(MagicSquare,i,j,size) && counter<=9)
                {
                        counter++;
                        MagicSquare[i][j] = counter;
                }
                if(counter>9)
                {
                        MagicSquare[i][j] = 0;
                }
        }
        //Kalau udah dicounter sampe 5 tapi masih belom valid, maka backtrack
        if(MagicSquare[i][j]==0)
        {
                j = j-1;
                if(j<0)
                {
                        i = i-1;
                        j = size;
                }
                counter = MagicSquare[i][j] + 1;

        }
        else //Kalau valid, maju ke kolom berikutnya
        {
                j = j+1;
                if(j>size)
                {
                        i = i + 1;
                        j = 1;
                }
        }

}

bool IsValid(int a[size][size], int x, int y, int size)
{
        //Mengecek apakah angka pada suatu kotak valid atau tidak
        int i, j,sumbaris=0,sumkolom=0;
        bool baris=true, kolom=true, sumbar=true, sumkol=true;
        i = x;
        for(j=1;j<=size;j++)
        {
                if(a[i][j]==a[x][y]&&j!=y)
                {
                        baris = false;
                }
        }
        j = y;
        for(i=1;i<=size;i++)
        {
                if (a[i][j]==a[x][y]&&i!=x)
                {
                        kolom = false;
                }
        }
        //Kalau suatu angka ditempatin di ujung kolom, dicek apakah jumlah
        //baris tersebut sama dengan MagicConstant
        if(y==size&&x!=size)
        {
                for(j=1;j<=size;j++)
                {
                        sumbaris = sumbaris + a[x][j];
                }
        }
        if(sumbaris!=MagicConstant(size))
        {
                sumbar = false;
        }
        //Kalau suatu angka ditempatin di ujung baris, dicek apakah jumlah
        //kolom tersebut sama dengan MagicConstant
        if(x==size&&y!=size)
        {
                for(i=1;i<=size;i++)
                {
                        sumkolom = sumkolom + a[i][y];
                }
        }
        if(sumkolom!=MagicConstant(size))
        {
                sumkol = false;
        }
        return (baris&&kolom&&sumbar&&sumkol);
```

```
    }
    int MagicConstant(int size)
    {
        //Menghitung magic constant (hasil penjumlahan) dari magic square
        //dengan size tertentu
        return (size(size*size+1))/2;
}
```

Contoh pengaplikasian :

1	2	3

Pada tabel di atas dapat dilihat bahwa angka-angka pada baris 1 masih memenuhi validasi karena tidak ada angka yang bertabrakan. Tetapi ketika angka 3 dimasukkan yakni angka terakhir pada baris 1, di cek apakah jumlah angka-angka pada baris tersebut sama dengan MagicConstant yakni 15. Karena 1+2+3 tidak sama dengan 15, maka angka pada baris 1 kolom 3 yaitu 3 akan coba diperbesar menjadi 4.

Prinsip backtrackingnya adalah, kita memutuskan pencobaan penelusuran solusi dengan instansiasi bahwa baris 1 kolom 3 diisi dengan 3. Ketika baris 1 kolom 3 diisi dengan 3, masih banyak kemungkinan kotak yang belum diisi yaitu 6 kotak lagi yang berarti masih ada tersisa 6! = 720 percobaan lagi. Ini berarti kita menghemat pencobaan penelusuran 720 percobaan karena angka 3 sudah tidak cocok ditempatkan di baris 1 kolom 3. Maka dari itu, kita mencoba mengganti angka 3 tersebut dan hasilnya adalah sebagai berikut :

1	2	4

Angka 3 pada baris 1 kolom 3 diganti menjadi angka 4. Tetapi ketika nantinya kita mencoba angka pada baris 1 kolom 3 hingga 9, tetap tidak akan didapatkan jumlah 15 karena berarti baris 1 maksimal menghasilkan jumlah 1 + 2 + 9 = 13. Ketika sudah dicoba hingga 9 tetapi masih tidak valid, maka akan dilakukan *backtrack* yaitu baris 1 kolom 3 dikosongkan dan percobaan pengisian mundur ke baris 1 kolom 2 dan dicoba untuk dinaikkan 1 per satu menjadi 3 sehingga hasilnya adalah sebagai berikut :

1	3	

Ketika langkah-langkah tersebut dilakukan secara terus menerus hingga mencapai baris ke 3 dan kolom ke 3, maka matriks yang didapat seharusnya menjadi seperti ini

2	7	6
9	5	1
4	3	8

IV. KESIMPULAN

Algoritma backtracking dapat diimplementasikan untuk persoalan *magic square*. Algoritma ini dipilih karena dapat menghasilkan proses pengerjaan yang lebih mangkus disbanding secara *trial and error* ataupun *brute force* karena tidak perlu untuk menghasilkan semua kemungkinan solusi, solusi yang sudah tidak mungkin tidak akan dilanjutkan penelusurannya.

V. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada Tuhan dengan penyertaan-Nya selama proses pengerjaan makalah dan juga kepada Pak Rinaldi Munir dan Bu Masayu sebagai dosen dari mata kuliah IF2211 Strategi Algoritma atas bimbingan dan ilmunya serta kepada teman-teman Teknik Informatika 2012 dan juga keluarga.

REFERENSI

[1] Munir Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2009

[2] http://mathworld.wolfram.com/MagicSquare.html (diakses pada 19 Mei 2014)

[3] http://www.ics.uci.edu/~eppstein/161/960215.html (diakses pada 19 Mei 2014)

PERNYATAAN

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

Bandung, 19 Mei 2014

Joshua Bezaleel Abednego
13512013