Penerapan Rekursif dalam Penyelesaian Sudoku

Paulus Haiktwo Dimpan Siahaan /13517111
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*author@itb.ac.id*

***Abstract* - Makalah ini membahas mengenai penerapan prinsip rekursif untuk menyelesaikan permainan sudoku dalam bentuk program. Penerapan prinsip rekursif tersebut terdapat dalam algoritma *backtracking* yang akan digunakan dalam program untuk menyelesaikan permainan sudoku .**

Kata Kunci—Sudoku, Rekursif, Algoritma *Backtracking* .

I. PENDAHULUAN

Permainan sudoku adalah permainan yang meminta pemain untuk mengisi sebuah kotak N x N dengan beberapa peraturan pengisian. Untuk semua N x N, tidak boleh terdapat angka yang sama pada baris atau kolom yang sama. Untuk kotak 9 x 9, peraturan umumya adalah, pada bagian 3 x 3, tidak boleh terdapat angka yang sama.

Pengisian angka pada sudoku tergantung pada ukuran baris atau kolom. Jika terdapat sudoku 9 x 9, angka yang bisa digunakan hanya angka 1 sampai 9. Satu kotak hanya bisa berisi satu angka. Terdapat beberapa kotak yang sudah diisi sebagai petunjuk untuk pemain untuk mengisi kotak yang lain.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Contoh sudoku 9 x 9
Gambar 1.1
(Sumber : https://cdn-images-1.medium.com/max/800/1*V6o3RVkDbHbwhR3lH_Aq7A.png)

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Contoh solusi sudoku 9 x 9
Gambar 1.2
(Sumber : https://cdn-images-1.medium.com/max/800/1*uT1D1ZgbzNuJU_ Q_X 1Tl4A.png)

Sudoku sendiri dalam Bahasa jepang disebut dengan "*Suuji wa dokushin ni kagiru*" yang artinya "angka harus tunggal" yang disingkat dari awalan kata majemuknya yaitu su dan doku menjadi sudoku.

Terdapat berbagai variasi dalam permainan sudoku, diantaranya *Mini sudoku*, *Alphabetical sudoku*, *Hyper sudoku*, *Killer sudoku*, dan lainnya. Tetapi, pada makalah ini, yang dibahas hanyalah penerapan rekursif dalam permainan sudoku normal 9 x 9 seperti gambar 1.1.

II. DASAR TEORI

2.1. Rekursif

2.1.1. Fungsi Rekursif

Suatu fungsi dikatakan rekursif apabila dalam definisinya, terdapat dirinya sendiri juga. Proses pendefinisian tersebut disebut rekursi.

Fungsi rekursif terdiri dari dua bagian yaitu :

1. Basis

Bagian basis mengandung nilai fungsi yang terdefinisi secara eksplisit, sehingga bagian ini yang akan menghentikan rekursif dan memberikan nilai dari hasil rekursif yang bukan fungsi itu sendiri.

2. Rekurens

Bagian ini yang memanggil fungsi itu sendiri dalam pendefinisiannya. Berisi kaidah untuk menemukan nilai fungsi pada suatu input dari nilai – nilai lainnya pada input yang lebih kecil.

Contoh dalam fungsi :

Misalkan f didefinsikan secara rekusif sebagai berikut

$$f(n) = \begin{cases} 3, n = 0 \\ 2f(n-1) + 4, n > 0 \end{cases}$$

Tentukan nilai f(4)!

f(0) = 3

f(1) = 2f(0) + 4 = 2 · 3 + 4 = 10

f(2) = 2f(1) + 4 = 2 · 10 + 4 = 24

f(3) = 2f(2) + 4 = 2 · 24 + 4 = 52

f(4) = 2f(3) + 4 = 2 · 52 + 4 = 108

Jadi, f(4) = 108.

2.1.2. Relasi Rekurens

Suatu barisan dengan elemen $a_0, a_1, a_2, \ldots, a_n$ dilambangkan dengan $\{a_n\}$. Elemen baris ke-n, yaitu a_n, dapat ditentukan dari suatu persamaan. Bila persamaan tersebut dinyatakan secara rekursif dalam satu atau lebih *term* elemen sebelumnya, maka persamaan tersebut dinamakan relasi rekurens.

Contoh : $a_n = 2\ a_{n-1} + 1$.

Pada relasi rekurens, terdapat kondisi awal. Kondisi awal suatu barisan adalah satu atau lebih nilai yang diperlukan untuk memulai menghitung elemen – elemen selanjutnya.

Contoh : $a_n = 2a_{n-1} + 1$; $a_0 = 1$.

Karena relasi rekurens merupakan barisan dengan setiap elemen yang dinyatakan secara rekursif, maka kondisi awal merupakan basis pada definisi rekursif. Kondisi awal akan menentukan elemen – elemen barisan selanjutnya. Kondisi awal yang berbeda akan menghasilkan elemen – elemen barisan yang berbeda pula.

2.1.3. Solusi Relasi Rekurens

Solusi dari sebuah relasi rekurens adalah sebuah fungsi yang tidak melibatkan rekursif lagi dan memenuhi relasi rekurens yang dimaksud.

Relasi rekurens dapat diselesaikan secara iteratif atau dengan metode yang sistematis. Secara sistematis digunakan pada relasi rekurens yang berbentuk homogen lanjar. Relasi rekurens dikatakan homogen lanjar jika berbentuk : $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$. Dalam hal ini, $c_1, c_2, \ldots, c_k$ adalah bilangan riil dan bukan nol.

Solusi dari relasi rekurens yang berbentuk homogen lanjar adalah bentuk : $a_n = r^n$. Dalam persamaan ini, r merupakan konstanta. Sulihkan $a_n = r^n$ ke dalam relasi rekurens homogen lanjar a_n menjadi $r^n = c_1r^{n-1} + c_2r^{n-2} + \ldots + c_kr^{n-k}$. Kemudian, bagi kedua ruas dengan r^{n-k} menghasilkan : $r^k - c_1r^{k-1} - c_2r^{k-2} - \ldots - c_{k-1}r - c_k = 0$. Persamaan tersebut merupakan persamaan karakteristik dari relasi rekurens.

Solusi dari persamaan karakteristik disebut akar – akar karakteristik, dan merupakan komponen solusi relasi rekurens yang dicari.Untuk relasi rekurens homogen lanjar dengan derajat k = 2, persamaan karakteristiknya berbentuk : $r^2 - c_1r - c_2 = 0$ dan akar persamaan karakteristiknya adalah r_1 dan r_2.

Terdapat dua teorema untuk penyelesaiannya :

Teorema 1: Barisan $\{a_n\}$ adalah solusi relasi rekurens $a_n = c_1a_{n-1} + c2a_{n-2}$ jika dan hanya jika $an = \alpha_1 r^n{}_1 + \alpha_2 r^n{}_2$ untuk n = 0, 1, 2, … dengan α_1 danα_2 adalah konstan.

Teorema 2: Misalkan $r_2 - c_1 r - c_2 = 0$ mempunyai akar kembar r_0 . Barisan $\{a_n\}$ adalah solusi relasi rekurens $an = c_1a_{n-1} + c_2a_{n-2}$ jika dan hanya jika $a_n = \alpha_1 r^n{}_0 + \alpha_2 nr^n{}_0$ untuk n = 0, 1, 2, … dengan α_1 danα_2 adalah konstan.

Teorema 1 digunakan jika $r_1 \neq r_2$, sedangkan teorema 2 digunakan jika $r_1 = r_2$.

2.2. Algoritma *Bactracking*

Algoritma *backtracking* merupakan salah satu algoritma yang sering digunakan dalam pemrograman. Algoritma ini menggunakan prinsip rekursi sebagai dasarnya dan membangun solusi secara bertahap dengan cara mencoba berbagai solusi dan membuang solusi yang tidak sesuai dengan syarat yang dibuat. Algoritma ini dapat memberikan hasil dengan cepat dan biasanya digunakan untuk memecahkan beberapa puzzle yang syaratnya dan hasil akhirnya sudah jelas. Contohnya adalah penyelesaian labirin.

Pada labirin, algoritma ini akan mencoba setiap kemungkinan jalan. Jika tidak sampai ke titik akhir, algoritma ini akan mundur kembali (*backtracking*) ke titik dimana ia memilih jalan buntu tersebut. Setelah itu, algoritma ini akan mencari jalan lain yang mungkin mencapai titik akhir.

III. PEMBAHASAN

3.1. Penerapan Algoritma *Backtracking* pada Sudoku

Pada sudoku, langkah yang nantinya akan digunakan pada algoritma *backtracking* merupakan angka yang mau diisi pada kotak sudoku, yaitu angka 1 sampai 9. Program akan mencari kotak kosong, lalu akan mencoba angka 1 sampai 9 untuk mengisi kotak tersebut, lalu dilakukan pengecekan sesuai syarat pada sudoku 9 x 9. Jika memenuhi syarat, program akan mencari kotak kosong lainnya lalu mengulang langkah awal. Jika semua angka tersebut tidak sesuai dengan syarat, program akan kembali ke kotak kosong sebelum kotak tersebut dan mencoba mengisi dengan angka lain yang sesuai lalu melanjutkan ke kotak kosong tadi. Jika tidak ada lagi kotak yang kosong, program akan selesai. Jika semua angka telah dicoba dan tidak ada yang cocok, maka program akan memberikan pesan kegagalan.

Program dengan algoritma di atas akan memakan waktu lebih sedikit dibanding menggunakan tekhnik *brute force* yang akan mencoba semua angka di setiap kotak kosong. Algoritma ini akan berhenti mengecek angka jika angka tersebut sudah sesuai dengan syarat bermain pada sudoku 9 x 9 lalu melanjutkan ke kotak kosong yang lain. Jika dengan *brute force*, program akan mencoba semua kemungkinan kombinasi angka untuk mengisi semua kotak yang kosong sehingga akan memakan waktu yang lumayan lama.

3.2. Penerapan dalam Program

Program ini menggunakan bahasa C ++. Pada program ini didefenisikan UNASSIGNED = 0, N = 9.

```
bool FindUnassignedLocation(int grid[N][N], int &row, int &cc
{
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}
```

Gambar 3.2.1.

Pada fungsi di atas, fungsi tersebut akan mengambalikan nilai dengan tipe data boolean. Variabel row dan col pada variabel dalam fungsi digunakan untuk menyimpan nilai baris dan kolom pada matriks sudoku yang belum diisi. Jika terdapat kotak yang masih belum terisi, fungsi akan mengembalikan nilai true, jika tidak ada lagi yang kosong, fungsi akan mengembalikan nilai false.

```
bool UsedInCol(int grid[N][N], int col, int num)
{
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

Gambar 3.2.2.

Pada fungsi ini, fungsi ini akan mengecek kolom dari sebuah posisi yang merupakan variabel input pada fungsi tersebut. Jika pada posisi tersebut terisi angka yang sudah terdapat pada kolom dari posisi tersebut, fungsi ini akan mengirimkan kembalian true. Jika tidak terdapat angka yang sama pada kolom yang sama, fungsi ini akan mengembalikan nilai false.

```
bool UsedInRow(int grid[N][N], int row, int num)
{
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

Gambar 3.2.3.

Fungsi ini akan mengembalikan nilai true jika pada posisi yang diinput dari variabel memiliki angka yang sama dengan salah satu posisi pada baris yang sama dengan posisi input. Jika tidak terdapat angka yang sama pada posisi yang sebaris dengan posisi input, fungsi ini akan mengembalikan nilai false.

```
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num
{
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row+boxStartRow][col+boxStartCol] == num)
                return true;
    return false;
}
```

Gambar 3.2.4.

Fungsi ini akan mengembalikan nilai true jika pada posisi yang diinput dari variabel memiliki angka yang sama dengan kotak 3 x 3 yang mencakup posisi input. Jika tidak terdapat angka yang sama antara posisi input dengan posisi pada kotak 3 x 3 di sekitarnya, fungsi ini akan mengembalikan nilai false.

```
bool isSafe(int grid[N][N], int row, int col, int num)
{

    return !UsedInRow(grid, row, num) &&
           !UsedInCol(grid, col, num) &&
           !UsedInBox(grid, row - row%3 , col - col%3, num)&&
            grid[row][col]==UNASSIGNED;
}
```

Gambar 3.2.3.

Fungsi di atas akan menggabungkan semua fungsi pengecekan sebelumnya. Fungsi ini digunakan untuk pengecekan angka yang diisi pada kotak kosong. Jika angka tersebut tidak memenuhi persyaratan sudoku 9 x 9, maka fungsi ini akan mengembalikan nilai false. Jika memenuhi persyaratan, fungsi ini akan mengembalikan nilai true.

```
bool SolveSudoku(int grid[N][N])
{
    int row, col;

    if (!FindUnassignedLocation(grid, row, col))
       return true;

    for (int num = 1; num <= 9; num++)
    {
        if (isSafe(grid, row, col, num))
        {
            grid[row][col] = num;

            if (SolveSudoku(grid))
                return true;

            grid[row][col] = UNASSIGNED;
        }
    }
    return false;
}
```

Gambar 3.2.4.

Fungsi di atas merupakan fungsi penyelesaian sudoku dan merupakan penerapan dari algoritma *backtracking* itu sendiri. Fungsi di atas akan mengembalikan nilai true jika tidak terdapat lagi kotak kosong(sudoku telah diselesaikan). Jika masih terdapat kotak kosong, fungsi ini akan mencoba angka 1 sampai 9 untuk mengisi kotak kosong tersebut. Setelah diisi, fungsi ini akan memanggil fungsi ini sendiri lalu mengecek apakah semua kotak sudah terisi atau belum.

Pengecekan kotak kosong menggunakan fungsi FindUnassignedLocation, sedangkan untuk pengecekan angka setelah mengisi kotak kosong digunakan fungsi isSafe.

```
void printGrid(int grid[N][N])
{
    for (int row = 0; row < N; row++)
    {
        for (int col = 0; col < N; col++)
            printf("%2d", grid[row][col]);
        printf("\n");
    }
}
```

Gambar 3.2.5.

Fungsi ini digunakan untuk mencetak matriks sudoku ke layar.

```
int main()
{
    int grid[N][N] = {{3, 0, 6, 5, 0, 8, 4, 0, 0},
                      {5, 2, 0, 0, 0, 0, 0, 0, 0},
                      {0, 8, 7, 0, 0, 0, 0, 3, 1},
                      {0, 0, 3, 0, 1, 0, 0, 8, 0},
                      {9, 0, 0, 8, 6, 3, 0, 0, 5},
                      {0, 5, 0, 0, 9, 0, 6, 0, 0},
                      {1, 3, 0, 0, 0, 0, 2, 5, 0},
                      {0, 0, 0, 0, 0, 0, 0, 7, 4},
                      {0, 0, 5, 2, 0, 6, 3, 0, 0}};
    if (SolveSudoku(grid) == true)
          printGrid(grid);
    else
          printf("No solution exists");

    return 0;
}
```

Gambar 3.2.6.

Fungsi ini merupakan fungsi main atau fungsi utama dari program. Fungsi ini mengandung contoh masukan program, yaitu sudoku yang diubah ke dalam matriks. Program ini kemudian memanggil fungsi SolveGrid. Jika terdapat solusi untuk sudoku tersebut, program akan mencetak hasil akhir sudoku ke layar. Jika tidak terdapat solusi, program akan mencetak "*No solution exists*". Kemudian program akan berakhir.

Input dari program ini dapat diubah menjadi input dari masukan pengguna dan/ atau input dari file eksternal. Pengolahan sudoku tetap dalam bentuk matriks 9 x 9.

IV. KESIMPULAN

Permainan sudoku merupakan sebuah permainan puzzle yang menarik dan lumayan sulit dimainkan jika ukuran baris dan kolomnya bertambah besar. Permainan tersebut dapat diselesaikan menggunakan program komputer. Salah satu program tersebut menggunakan algoritma *backtracking* yang dasarnya menggunakan prinsip rekursif. Dengan program tersebut, permainan sudoku dalam ukuran berapapun dapat diselesaikan dengan cepat dengan syarat mencantumkan syarat bermain yang jelas pada program dan goal yang ingin dituju dalam program.

DAFTAR REFERENSI

[1] https://www.geeksforgeeks.org/sudoku-backtracking-7/
Diakses pada 07 Desember 2018.

[2] https://library.binus.ac.id/eColls/eThesisdoc/Bab2/2009-2-00177-IF%20Bab%202.pdf
Diakses pada 07 Desember 2018.

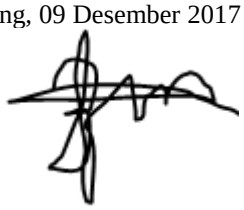
[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Makalah-Matdis-2015/Makalah-IF2120-2015-113.pdf
Diakses pada 08 Desember 2018

[4] Slide Presentasi Rekursif, Bahan kuliah Matematika Diskrit, Teknik Informatika ITB.

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, 09 Desember 2017



Paulus Haiktwo Dimpan Siahaan
13517111