Pemecahan Masalah Rubik Cube dengan Algoritma Breadth First Search

Jonathan Sudibya - 13512093
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512093@std.stei.itb.ac.id

***Abstract*—Banyak solusi algoritma penyelesaian kotak Rubik ditemukan oleh berbagai orang. Salah satu algoritma yang dapat digunakan untuk menyelesaikan permainan ini dalah Breadth First Search atau BFS. BFS biasa dipasangkan dengan Depth First Search, namun DFS ini tidak efektif untuk menyelsaikan permainan ini karena jika sebuah sisi terus diputar maka rubik akan kembali ke posisi semula dalam 4 kali putaran. Dengan BFS diharapkan permainan ini juga diselesaikan lebih cepat dibandingkan dengan menggunakan Algoritma Brute Force dimana semua jenis kemungkinan dicoba.**

***Index Terms*—Algorithm, BFS, Breadth First Search, Rubik**

I. PENDAHULUAN

Permainan Rubik cube diciptakan ada 1974 oleh Profesor Erno Rubik. Permainan ini dicipakan dengan 3 x 3 per sisinya dan miliki 6 sisi. Seiring dengan berjalannya waktu Rubik berkembang menjadi banyak sekali jenis varietas seperti rubik 2x2 , 4 x 4 , 5 x 5 dan 6 x 6. Tidak hanya variasi jumlah kotak pada rubik, bentuk juga dirubah menjadi segita, kotak yang tidak simetris dan lain sebagainya.

Gambar 1. Jenis-jenis Rubik saat ini.

Permainan ini mengasah otak pemain dengan menyelesaikan (mengembalikan) posisi balok rubik ke tempat semula, yaitu setiap sisi memiliki warna yang sama. Permainan ini sempat meredup selama beberapa tahun. Namun, sekitar tahun 2010 permainan ini kembali popular baik dikalangan muda maupun tua.

Dengan kembali populernya permainan rubik, berbagai kompetisi untuk menyelesaikan rubik dibuka. Kompetisi ini tidak hanya melihat kecepatan pemain untuk menyelesaikan sebuah rubik, melainkan banyak tipe permainan seperti blindfold (mata tertutup) dan menyelesaikan banyaknya rubik dalam kurun waktu tertentu.

Sebenarnya permainan rubik bisa diselesaikan dengan pengetahuan dan pengalaman (eureka), namun untuk sebuah komputer untuk mempelajari dan menyimpan seluruh kemungkinan yang ada, dirasa tidak memungkinkan mengingat banyaknya kemungkinan yang harus dicoba. Dengan demikian menggunakan Algoritma Breadth First Search (BFS) diharapkan permainan ini dapat diselesaikan tanpa harus mengetahui kumpulan ruang solusinya.

II. BREADTH FIRST SEARCH

Algoritma Breadth First Search ditemukan untuk menyelesaikan masalah graf, baik berarah maupun tidak berarah. BFS bersifat *uninformed / blind search*, sehingga ruang solusi belum sama sekali dibuat. BFS bisa juga disebut sebagai pencarian melebar karena pencarian BFS difokuskan kepada banyaknya kemungkinan gerak yang bisa dilakukan dari sebuah status / simpul graf.

Pada permasalahan ini BFS digunakan untuk mengetahui langkah apa yang diperlukan untuk menyelesaikan sebuah kotak rubik berukuran 3 x 3. Langkah – langkah ini memiliki beberapa simbol sebagai berikut :

1. U : bagian atas diputar ke kanan.
2. D : bagian bawah diputar ke kanan.
3. R : bagian kanan diputar ke lanan.
4. L : bagian kiri diputar ke kanan.
5. F : bagian depan diputar ke kanan. (searah jarum jam).
6. B : bagian belakang diputar ke kanan. (searah

jarum jam).

7. U’ : bagian atas diputar ke kiri.
8. D’ : bagian bawah diputar ke kiri.
9. R’ : bagian kanan diputar ke kiri.
10. L’ : bagian kiri diputar ke kiri.
11. F’ : bagian depan diputar ke kiri. (berlawanan arah jarum jam).
12. B’ : bagian belakang diputar ke kiri. (berlawanan arah jarum jam).

Perlu diingat pula bahwa kotak rubik tidak bisa diputar karena sisi dari rubik tersebut dapat berubah. Simpul dimulai dari posisi rubik yang diberikan diawal.

Setiap posisi Rubik akan ditampung di dalam sebuah queue dimana setiap queue menyatakan sebuah level atau jumlah langkah. Pencarian dihentikan ketika solusi yang pertamakali muncul ditemukan.

III. Analisis dan Hasil Data

A. Algoritma

Secara umum algoritma untuk menyelesaikan permainan Rubik ini adalah :

1. Kotak Rubik yang belum dipecahkan diputar sesuai dengan banyaknya kemungkinan pergerakan (12).
2. Posisi rubik yang pernah dikunjungi tidak dikunjungi lagi.
3. Setiap kemungkinan disimpan kedalam sebuah queue dengan setiap anak dari posisi diletakan dipaling belakang.
4. Langkah terus diulangi hingga solusi ditemukan.

B. Implementasi

```
public class rubick {

    private int [][] top;

    private int [][] bottom;

    private int [][] front;

    private int [][] back;

    private int [][] right;

    private int [][] left;


    public rubick(){

        top = new int[3][3];

        bottom = new int[3][3];

        front = new int[3][3];

        back = new int[3][3];

        right = new int[3][3];

        left = new int[3][3];

    }


    public void setDefault(){

        for (int i = 0;i < 6;i++){

            for(int j = 0; j < 3;j++){

                for(int k=0;k < 3;k++){

                    switch (i){

                        case 0 : top[j][k] = i;break;

                        case 1 : bottom[j][k] = i;break;

                        case 2 : front[j][k] = i;break;

                        case 3 : back[j][k] = i;break;

                        case 4 : right[j][k] = i;break;

                        case 5 : left[j][k] = i;break;

                    }

                }

            }

        }

    public void rotateU(int [][] side,boolean dir){

        int temp,temp2;

        temp = side[0][1];

        side[0][1] = side[0][0];

        temp2 = side[0][2];

        side[0][2] = temp;
```

```
    temp = side[1][2];
    side[1][2] = temp2;
    temp2 = side[2][2];
    side[2][2] = temp;
    temp = side[2][1];
    side[2][1] = temp2;
    temp2 = side[2][0];
    side[2][0] = temp;
    temp = side[1][0];
    side[1][0] = temp2;
    side[0][0] = temp;
  }
  public void rotateD(int [][] side,boolean dir){
    int temp,temp2;
    temp = side[0][1];
    side[0][1] = side[0][0];
    temp2 = side[0][2];
    side[0][2] = temp;
    temp = side[1][2];
    side[1][2] = temp2;
    temp2 = side[2][2];
    side[2][2] = temp;
    temp = side[2][1];
    side[2][1] = temp2;
    temp2 = side[2][0];
    side[2][0] = temp;
    temp = side[1][0];
    side[1][0] = temp2;
    side[0][0] = temp;
  }
  public void rotateR(int [][] side,boolean dir){
    int temp,temp2;
    temp = side[0][1];
    side[0][1] = side[0][0];
    temp2 = side[0][2];
    side[0][2] = temp;
    temp = side[1][2];
    side[1][2] = temp2;
    temp2 = side[2][2];
    side[2][2] = temp;
    temp = side[2][1];
    side[2][1] = temp2;
    temp2 = side[2][0];
    side[2][0] = temp;
    temp = side[1][0];
    side[1][0] = temp2;
    side[0][0] = temp;
  }
  public void rotateL(int [][] side,boolean dir){
    int temp,temp2;
    temp = side[0][1];
    side[0][1] = side[0][0];
    temp2 = side[0][2];
    side[0][2] = temp;
    temp = side[1][2];
    side[1][2] = temp2;
    temp2 = side[2][2];
    side[2][2] = temp;
    temp = side[2][1];
    side[2][1] = temp2;
```

```
    temp2 = side[2][0];
    side[2][0] = temp;
    temp = side[1][0];
    side[1][0] = temp2;
    side[0][0] = temp;
  }
  public void rotateF(int [][] side,boolean dir){
    int temp,temp2;
    temp = side[0][1];
    side[0][1] = side[0][0];
    temp2 = side[0][2];
    side[0][2] = temp;
    temp = side[1][2];
    side[1][2] = temp2;
    temp2 = side[2][2];
    side[2][2] = temp;
    temp = side[2][1];
    side[2][1] = temp2;
    temp2 = side[2][0];
    side[2][0] = temp;
    temp = side[1][0];
    side[1][0] = temp2;
    side[0][0] = temp;
  }
  public void rotateB(int [][] side,boolean dir){
    int temp,temp2;
    temp = side[0][1];
    side[0][1] = side[0][0];
    temp2 = side[0][2];
    side[0][2] = temp;
    temp = side[1][2];
    side[1][2] = temp2;
    temp2 = side[2][2];
    side[2][2] = temp;
    temp = side[2][1];
    side[2][1] = temp2;
    temp2 = side[2][0];
    side[2][0] = temp;
    temp = side[1][0];
    side[1][0] = temp2;
    side[0][0] = temp;
  }
}
```

Gambar 2 Unsolved Rubik Cube

C. Hasil dan Analisis

Hasil penulusan dengan BFS pada rubik yang diacak secara random berkisar, Antara 1-5 menit (tanpa animasi perputaran karena memakan banyak waktu untuk mencetak data.

Dengan hasil demikian dengan menggunakan alogirtma BFS dinilai cukup lambat untuk menyelesaikan permainan Rubik karena Permainan rubik dapat diselesaikan oleh manusia dibawah 1 menit.

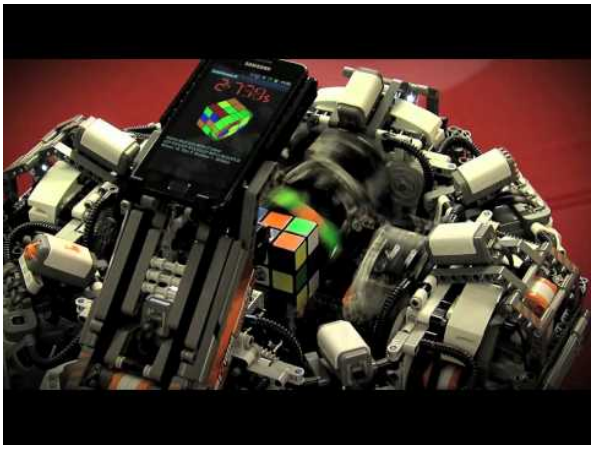
IV. KESIMPULAN

Kesimpulan kemampuan BFS untuk menyelesaikan masalah Rubik dinilai kurang efektif karena Rubik dapat diseesaikan lebih cepat dengan kemampuan manusia. Namun kemampuan ini tetap dinilai lebih cepat dibandingkan menggunakan Brute Force.

Dengan automasi penyelesaikan Rubik dengan menggunakan metoda ini jauh lebih cepat diterapkan untuk menyelesaikan rubik 2x2. Karena rubik 2x2 memiliki kumpulan posisi jauh lebih sedikit dibandingkan rubik 3 x 3. Berlaku juga untuk selanjutnya.

Dengan demikin semakin sedikit jumlah kotak yang harus diselesaikan maka semakin sendikit data yang harus diolah sehingga performa untuk menyelesaikan akan semakin bertambah seiring dengan mengicilnya data yang dikerjakan

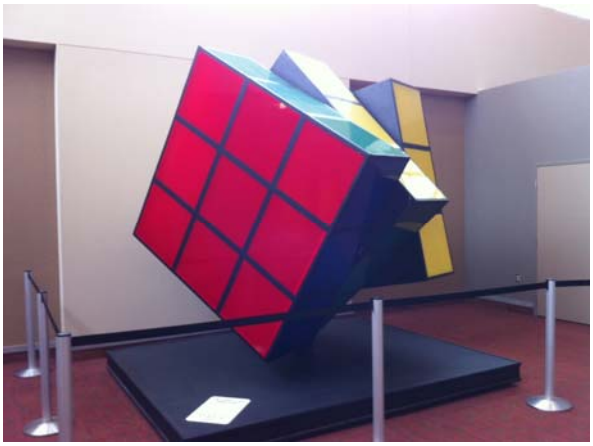
V. FUN FACT



Gambar 3 Robot Lego menyelesaikan Rubik

Gambar 4 Rubik terkecil

Gambar 5 Koleksi Rubik



Gambar 6 Rubik terbesar yang ada di pameran

REFERENCES

[1] Munir,Rinaldi, BFS and DFS.pptx Slide Kuliah 2014.
[2] http://gobigorgohomeblog.com/content/Rubiks-Cube2.jpg
[3] http://img4.tgdaily.com/sites/default/files/stock/450teaser/cube.jpg
[4] http://i.ytimg.com/vi/cHXTEBihbN8/0.jpg
[5] https://gigaom2.files.wordpress.com/2013/07/screen-shot-2013-07-26-at-3-30-42-pm.png
[6] http://media.techeblog.com/images/roulette-wheel-iq-cube.jpg
[7] http://images1.sw-cdn.net/model/picture/674x501_503955_166034_1338413389.jpg

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

Jonathan Sudibya (13512093)