Penerapan Algoritma A* dalam Permainan Rubik's *Cube*

Nigel Sahl - 13521043
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521043@std.stei.itb.ac.id

***Abstrak*—Salah satu penemuan paling berpengaruh di dunia yaitu *rubik's cube* adalah sebuah permainan teka-teki mekanis yang menjadi pusat perhatian pada awal ditemukannya. Dalam makalah ini akan dibahas mengenai penerapan dari algoritma *informed search* yaitu A* dalam menyelesaikan permaianan kubus rubik ini. Pada makalah ini akan difokuskan untuk jenis kubus rubik dengan ukuran nxn yang setiap permukaan kubus memiliki warna yang berbeda.**

Kata Kunci—rubik's cube, A*, algoritma, permainan

I. PENDAHULUAN

Rubik's Cube, salah satu dari 100 penemuan paling berpengaruh selama abad ke-20. Selain itu, permaianan ini secara luas dianggap sebagai mainan terlaris di dunia dan memenangkan penghargaan khusus *Game of the Year* di Jerman serta memenangkan penghargaan seupa untuk mainan terbaik di Inggris, Prancis, dan AS. *Rubik's Cube*, sebuah teka-teki mekanik yang sangat populer dan telah menarik perhatian di seluruh dunia karena karakteristiknya yang unik. Sebagai mainan latihan otak klasik yang terkenal di kalangan masyarakat umum, *Rubik's Cube* telah digunakan untuk penelitian ilmiah dan pengembangan teknologi oleh banyak sarjana.

Gambar 1.1 Sebuah kubus rubik berukuran 3x3

Sumber: A rubik cube on a white background photo – Free Rubik cube Image on Unsplash

Rubik's Cube, atau dapat dikatakan sebagai kubus rubik, teka-teki mekanik 3D yang ditemukan pada tahun 1974 oleh pemahat dan profesor arsitektur dari Hungaria yaitu Ernő Rubik, terdiri dari kubus dengan permukaan terbagi menjadi beberapa bagian yang dapat diputar yang pada awalnya disebut *Magic Cube*. Tujuan dari permainan ini adalah untuk mengembalikan setiap sisi kubus sehingga memiliki satu warna yang sama.

Kubus Rubik ini sangat populer sejak ditemukan oleh Ernő Rubik. Awalnya, Rubik menciptakan kubus ini sebagai alat yang membantu mahasiswa memahami konsep tiga dimensi. Namun, setelah beberapa kali memutar-mutarnya, Rubik menyadari kompleksitas dan tantangan yang terkandung dalam permainan ini.

Pada beberapa waktu setelah penemuannya, kubus ini dianalisis karakteristik khusus dan ditinjau penelitian tentang *Rubik's Cube* di berbagai disiplin ilmu di berbagai negara, termasuk penelitian tentang metafora ilmiah *Rubik's Cube*, algoritma reduksi, aplikasi khas, dan masalah mekanisme.

Rubik's Cube mencapai puncak popularitasnya pada tahun 1980-an. Meskipun sudah lama, namun kubus rubik ini masih dikenal dan digunakan secara luas. Kubus ini tidak hanya menarik para penggemarnya saja tapi juga para peneliti yang melakukan penelitian tentang algoritma pengurangan *Rubik's Cube* tetapi juga menarik perhatian para ilmuwan dan pekerja teknis dari berbagai lapisan masyarakat karena desain dan idenya yang canggih.

Struktur dari kubus rubik memiliki beberapa kegunaan seperti rotasi, permutasi dan kombinasi, serta siklus dan simetri. Kubus ini juga diperlakukan sebagai model fisik atau alat untuk mempelajari masalah ilmiah tertentu. Secara keseluruhan, prinsip-prinsip kubus ini terkandung dalam berbagai sistem ilmiah yang melibatkan permutasi dan kombinasi, simetri, dan siklus. Di sisi lain, para sarjana mulai mengeksplorasi prinsip-prinsip dari gerakan inti dari struktur kubus rubik.

Terdapat berbagai macam cara dalam menyelesaikan permainan teka-teki ini. Pada makalah ini, akan dibahas sebuah algoritma untuk menyelesaikan permainan ini yaitu algoritma A*. Terdapat berbagai jenis rubik di dunia ini. Pada pembahasan ini akan diambil rubik yang berbentuk kubus dengan enam sisi permukaan yang berbeda-beda warnanya. Penyelesaian dari permainan ini adalah membuat konfigurasi dari warna-warna pada rubik dalam kondisi semula yaitu setiap permukaan hanya memiliki satu buah warna.

II. LANDASAN TEORI

Pencarian rute adalah proses menentukan langkah-langkah atau urutan yang diperlukan untuk mencapai tujuan tertentu. Dalam pencarian rute, terdapat dua pendekatan utama yang digunakan: pencarian terinformasi (*informed search*) dan pencarian tanpa informasi (*uninformed search*).

Pencarian *informed search* melibatkan penggunaan informasi tambahan yang tersedia tentang masalah atau ruang pencarian untuk mengarahkan proses pencarian. Dalam pencarian terinformasi, algoritma terinformasi menggunakan nilai heuristik atau fungsi penilaian yang mengestimasikan biaya atau jarak tersisa untuk mencapai tujuan. Heuristik ini memberikan petunjuk tentang arah yang mungkin lebih baik dan efisien dalam mencapai solusi.

Contoh algoritma pencarian *informed search* termasuk algoritma A* (A-star) dan algoritma Greedy Best-First Search. Algoritma A* menggunakan fungsi penilaian (f(n)) yang menggabungkan biaya langkah yang sudah diambil (g(n)) dan perkiraan biaya yang tersisa (h(n)) untuk mencapai tujuan. Fungsi penilaian ini membantu algoritma memilih langkah-langkah yang paling menjanjikan berdasarkan kombinasi biaya langkah dan perkiraan biaya tersisa.

Pencarian *uninformed search*, dikenal sebagai pencarian buta (*blind search*), tidak menggunakan informasi tambahan tentang masalah atau ruang pencarian. Algoritma ini hanya bergantung pada struktur ruang pencarian yang tersedia untuk menjelajahi semua kemungkinan langkah-langkah yang sistematis.

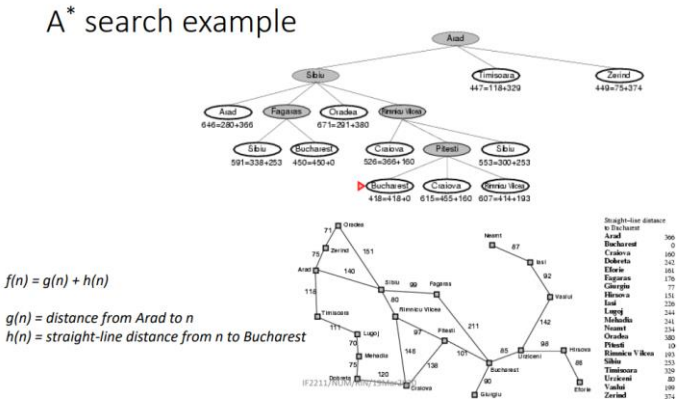
Contoh algoritma pencarian tanpa informasi termasuk algoritma Breadth-First Search (BFS), Depth-First Search (DFS), dan Uniform-Cost Search (UCS). BFS menjelajahi semua simpul pada kedalaman yang sama sebelum beralih ke kedalaman berikutnya. DFS menjelajahi simpul sejauh mungkin pada satu jalur sebelum kembali dan mencoba jalur lain. UCS, juga dikenal sebagai Dijkstra's Algorithm, mempertimbangkan biaya langkah saat menjelajahi ruang pencarian dan memilih langkah dengan biaya terendah.

Algortima pencarian A* adalah algoritma pencarian terbaik pertama yang terinformasi dan secara efisien menentukan jalur biaya terendah antara dua *node* mana pun dalam grafik berbobot terarah dengan bobot sisi non-negatif. Algoritma ini merupakan varian dari algoritma Dijkstra. Sedikit perbedaan muncul dari fakta bahwa fungsi evaluasi digunakan untuk menentukan *node* mana yang akan dijelajahi selanjutnya.

Terdapat sebuah fungsi dalam algoritma ini yang digunakan untuk menandakan setiap *path* yang dikunjungi. Fungsi evaluasi, f(x), untuk algoritma pencarian A* adalah sebagai berikut:

$$f(x) = g(x) + h(x)$$

Dalam fungsi tersebut g(x) merepresentasikan biaya untuk sampai ke *node* x dan h(x) merepresentasikan estimasi biaya untuk sampai ke *node* tujuan dari *node* x. Algoritma ini dapat dikatakan sebagai penggabungan algoritma UCS dan Greedy Best First Search. UCS hanya mengevaluasi nilai f(n) nya dengan g(n) sedangkan Greedy Best First Search mengevaluasi nilainya dengan h(n).



Gambar 2.1 Contoh penerapan algoritma A* pada pencarian rute terpendek
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

Algoritma A* memiliki pendekatan heuristik untuk mengisi nilai h(n) yaitu nilai heuristik, nilai ini menentukan harga taksiran dari *node* n untuk mencapai tujuan. Nilai ini membuat sebuah *node* menjanjikan untuk dikunjungi atau tidak, kesulitan dalam menyelesaikan sub persoalan, kualitas dari solusi yang direpresentasikan oleh *node*, dan jumlah informasi yang didapat. Nilai ini bergantung padda n, *goal*, pencarian sejauh ini, dan domain. Agar algoritma menghasilkan hasil yang benar, fungsi evaluasi harus dapat diterima, artinya tidak pernah melebih-lebihkan biaya untuk mencapai *node* tujuan.

Dalam ilmu komputer, khususnya dalam algoritma yang berkaitan dengan pencarian jalan, fungsi heuristik dikatakan dapat diterima (***admissible***) jika tidak pernah melebih-lebihkan biaya untuk mencapai tujuan, yaitu biaya yang diperkirakan untuk mencapai tujuan tidak lebih tinggi dari biaya serendah mungkin dari saat ini. titik di jalan. Dalam konteks ini, h(n) tidak pernah lebih besar dari h*(n) dalam konteks ini. Ini ada hubungannya dengan gagasan Heuristik yang andal. Semua Heuristik yang reliabel dapat digunakan, namun tidak semua Heuristik yang reliabel dapat digunakan.

Hal ini terkait dengan konsep heuristik yang konsisten. Sementara semua heuristik yang konsisten dapat diterima, tidak semua heuristik yang dapat diterima konsisten.

Algoritma A* diimplementasikan dengan cara yang mirip dengan algoritma Dijkstra. Diberikan grafik berbobot dengan bobot tepi non-negatif, untuk menemukan jalur dengan biaya terendah dari simpul awal S ke simpul tujuan G, dua list digunakan:

1. List terbuka, diimplementasikan sebagai antrean prioritas, yang menyimpan *node* berikutnya untuk dijelajahi. Karena ini adalah antrean prioritas, kandidat *node* yang paling menjanjikan (yang memiliki nilai terendah dari fungsi evaluasi) selalu berada di atas. Awalnya, satu-satunya simpul dalam daftar ini adalah simpul awal S.
2. List tertutup yang menyimpan *node* yang telah dievaluasi. Ketika sebuah *node* berada dalam daftar tertutup, itu berarti jalur dengan biaya terendah ke *node* tersebut telah ditemukan.

Untuk menemukan jalur biaya terendah, pohon pencarian dibangun dengan cara berikut:

1. Inisialisasi pohon dengan simpul akar sebagai simpul awal S.
2. Hapus simpul teratas dari list terbuka untuk eksplorasi.
3. Tambahkan *node* saat ini ke list tertutup.
4. Tambahkan semua simpul yang memiliki sisi masuk dari simpul saat ini sebagai simpul anak di pohon.
5. Perbarui biaya terendah untuk menjangkau *node* anak.
6. Hitung fungsi evaluasi untuk setiap simpul anak dan tambahkan ke daftar terbuka.

III. Pembahasan dan Implementasi

Penyelesaian permainan kubus rubik dengan A* dapat diabstraksi permasalahannya menjadi beberapa tahap. Tahap pertama yaitu abstraksi menjadi permasalahan yang dapat direpresentasi dalam kode program. Pada pembahasan ini penyelesaian permaianan dilakukan dengan pendekatan pemrograman berorientasi objek di java dengan 2 kelas yaitu kelas Rubik sebagai representasi dari permainan kubus rubik dengan 6 warna yang berbeda dan kelas Algo sebagai kelas untuk penerapan algoritma A*. Tahap kedua, yaitu pencarian fungsi f(n) pada algoritma A*. Dan tahap terakhir, implemmentasi algoritma pada kelas Algo.

A. *Kelas Rubik*

Kelas ini merepresentasikan permainan rubik menjadi 6 buah matriks dengan tipe string. Keenam matrix itu sebaga representasi 6 buah permukaan pada rubik. Isi string dari matriks tersebut berupa inisial warna dan kode angka misalnya warna kuning dapat ditulis “y01” yaitu blok warna kuning pertama (biasnya di ujung kiri atas tergantung perspektif pemain).

```
public class Rubik {
    // rubik's cube
    // matrix layer of rubik's cube
    private List<List<String>> up;
    private List<List<String>> down;
    private List<List<String>> left;
    private List<List<String>> right;
    private List<List<String>> front;
    private List<List<String>> back;
```

Gambar 3.1 Beberapa atribut dalam kelas Rubik

Sumber: dokumen penulis

Terdapat beberapa method penting dalam kelas ini yaitu konstruktor yang menginisiasi semua atribut dalam kelas ini yaitu 6 matriks untuk *layer* utama, 6 matrix untuk *temporary layer*, 6 matriks untuk *goal layer*, size dari rubik, dan array string method yang merepresentasi gerakan pemain terhadap rubik. Method berikutnya adalah getter dan setter kelas ini. Method berikutnya terdapat visualize rubik untuk 3 layer dari rubik.

```
PS C:\Users\Nerbi\Documents\A Project\Tugas\makalahStima\Rubiks-Cube-
      up
   y1  y2  y3
   y4  y5  y6
   y7  y8  y9
    left          front         right         back
   g1  g2  g3    o1  o2  o3    b1  b2  b3    r1  r2  r3
   g4  g5  g6    o4  o5  o6    b4  b5  b6    r4  r5  r6
   g7  g8  g9    o7  o8  o9    b7  b8  b9    r7  r8  r9
      down
   w1  w2  w3
   w4  w5  w6
   w7  w8  w9
```

Gambar 3.2 Contoh visualisasi rubik 3x3

Sumber: dokumen penulis

Terdapat method untuk changeColor untuk menampilkan tulisan berwarna di layar, updateTemp untuk menyamakan temp dengan *main layer*, shuffle dengan parameter *integer* untuk mengacak susunan rubik secara random sebanyak parameter masukannya, move sebagai method dengan parameter *integer* sebagai masukan untuk menggerakan rubik sesuai dengan notasi rubik. Dan method convertMove.

```
1054    public void move(int n){
1055        // enums the function of move 0..12
1056        switch(n){
1057            case 0:
1058                this.R();
1059                break;
1060            case 1:
1061                this.Rp();
1062                break;
1063            case 2:
1064                this.U();
1065                break;
1066            case 3:
1067                this.Up();
1068                break;
1069            case 4:
1070                this.L();
1071                break;
```

Gambar 3.3 Method move pada kelas Rubik

Sumber: dokumen penulis

Terdapat 24 method untuk menggerakan kubus rubik ini. Method dasr dari pergerakan rubik adalah R, R', U, U', L, L', D, D', F, F', B, B'. Karena nama metode dalam java tidak bisa menggunakan akses maka diganti menjadi p contohnya jika R' menjadi Rp. Method lainnya adalah method yang sama dengan yang di atas namun dengan parameter *integer* r yaitu menggerakan rubik pada sisi ke-r dari sisi yang dipilih menuju pusat sehingga totalnya ada 24. Penjelasan gerakannya adalah sebagai berikut:

1. R: menggerakan sisi bagian kanan (*right*) rubik searah jarum jam
2. U: menggerakan sisi bagian atas (*up*) rubik searah jarum jam
3. L: menggerakan sisi bagian kiri (*left*) rubik searah jarum jam
4. D: menggerakan sisi bagian bawah (*down*) rubik searah jarum jam
5. F: menggerakan sisi bagian depan (*front*) rubik searah jarum jam
6. B: menggerakan sisi bagian belakang (*back*) rubik searah jarum jam

Untuk metode dengan tambahan petik atau p maka sama seperti di atas namun berlawanan arah jarum jam.

```
public void R(){
    // rotate the right layer clockwise and all right part front, back, up, down is changed
    // right side of front -> right up side, right up side -> right back side,
    // right back side -> right down side, right down side -> right front side
    int n = this.size;
    this.updateTemp();
    // rotate right layer clockwise
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this.right.get(i).set(j, this.tempRight.get(n - 1 - j).get(i));
        }
    }
    for(int i = 0; i < n; i++){
        this.front.get(i).set(n - 1, this.tempDown.get(i).get(n - 1));
        this.up.get(i).set(n - 1, this.tempFront.get(i).get(n - 1));
        this.back.get(i).set(n - 1, this.tempUp.get(n- 1 - i).get(n - 1));
        this.down.get(i).set(n - 1, this.tempBack.get(n - 1 - i).get(n - 1));
    }

}
```

Gambar 3.4 Salah satu method pada kelas Rubik (method R yang mengerakan sisi kanan rubik searah jarum jam)

Sumber: dokumen penulis

B. Kelas Algo

Kelas ini berisi implementasi dari algoritma A* untuk menyelesaikan permainan rubik. Terdapat beberapa atribut dalam kelas ini yaitu atribut bertipe kelas Rubik, *Priority Queue* untuk *tuple* list *integer* sebagai representasi path gerakan yang diambil dan *integer* cost nilai f(n), terdapat juga atribut list string hasil.

```
1   import java.util.*;
2
3   public class Algo {
4       // rubik
5       private Rubik rubik;
6       // prio queue for A* algorithm :
7       // example : [ [ [1,2,3], 1 ], [ [1,2,3], 2 ] ]
8       private PriorityQueue<Tuple<List<Integer>, Integer>>
        prioQueue;
9       // list hasil in array of String
10      private List<String> hasil;
11  >   public Algo(int s){···
22  >   public Rubik getRubik(){···
25  >   public Integer getHn(){···
58  >   public static Integer getHn(Rubik rbk){···
97  >   public void printPrioQueue(){···
104 >   public void solve(){···
182
        Run | Debug
183 >   public static void main(String[] args){···
```

Gambar 3.5 Atribut dan mthod kelas Algo

Sumber: dokumen penulis

Priority Queue dari kelas ini digunakan untuk tempat penyimpanan path yang akan dikunjungi, *Queue* ini memiliki prioritas untuk pengisiannya yaitu terurut menaik sehingga antrian dikeluarkan elemennya dari elemen dengan nilai terkecil.

Fungsi f(n) yang digunakan dalam algoritma A* ini menggunakan fungsi g(n) ssebagai jumlah gerakan pemain dan fungsi h(n) atau heiristik menggunakan banyak kesalahan.

Untuk menentukan banyak kesalahan dari rubik, hanya perlu mengecek bagian-bagian dari rubik. terdapat beberapa istilah bagian dalam rubik yaitu:

1. *Edge*: blok yang memiliki 2 buah sisi yang berbeda warna
2. *Corner*: blok yang memiliki 3 buah sisi yang berbeda warna
3. *Center*: blok yang hanya memiliki 1 warna pada satu sisi yang terletak di tengah permukaan rubik jika rubik berukuran ganjil.

Dengan adanya hal tersebut, banyak kesalahan yang dihitung dapat diperkecil menjadi pengecekan *edge* dan *corner* yang tidak sama. Contohnya untuk rubik 3x3, terdapat 6 *center*, 8 *corner*, dan 12 *edge*.

Gambar 3.6 Fungsi untuk menghitung nilai heuristik h(n)

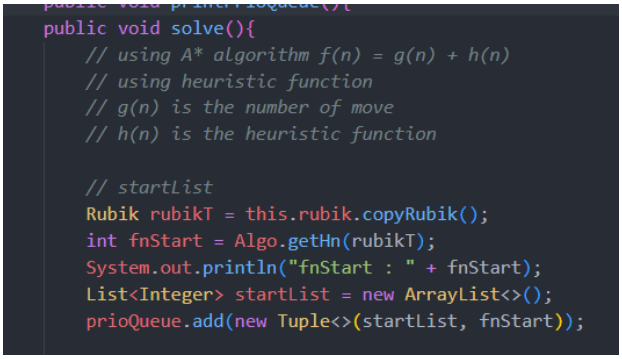
Sumber: dokumen penulis

Pada gambar 3.6 dapat ditunjukkan perhitungan banyak kesalahan posisi hanya dihitung bagian semua bagian depan dan belakang serta bagian antara baris 1 sampai size rubik kurang satu untuk sisi atas dan bawah. Pengecekan semua bagian sisi depan dan belakang sudah akan mengecek Dengan ini, semua pengecekan bagian rubik akan dilakukan.

Algoritma A* diimplementasikan pada fungsi solve pada kelas Algo. Pada fungsi ini, ambil f(n) dari *node* root yaitu kondisi awal setiap matriks permukaan pada rubik. Karena root belum jalan sama sekali sehingga g(n) bernilai nol. h(n) pada fungsi ini diperoleh oleh fungsi getHn yang telah dijelaskan di atas. Setelah itu, inisiasi list path awal yang berisi kosong. Selanjutnya, isi prioQueue dengan tuple baru yang berisi list kosong dan nilai f(n) yang sudah didapat.

$$f(n) = g(n) + h(n)$$

$$f(root) = h(root)$$



Gambar 3.7 Inisiasi awal algoritma untuk method solve

Sumber: dokumen penulis

Langkah selanjutnya adalah sebagai berikut:

1. Selama elemen pertama dalam queue bukan *goal* yang dituju matriks *goal* setiap layer permukaan maka iterasi langkah-langkah dibawah. Pengecekan *goal* adalah dengan mengintip elemen pertama dalam antrian dan melihat elemen tuple kedua jika dikurangi size dari list path nya apakah nol atau bukan. Jika nol maka elemen tersebut adalah *goal* dan sebaliknya.
2. Inisiasi tuple dengan megeluarkan elemen pertama dari antrian, inisiasi list move untuk tuple elemen pertama dan *integer* fn untuk elemen tuple kedua.
3. Inisiasi rubikTemp dengan Salinan dari rubik awal.
4. Iterasi semua list pathnya dan terapkan metode move.
5. Kurangi fn dengan hn dari rubik awal
6. Iterasi i dari 0 sampai 12 untuk Langkah 7 sampai 10 (iterasi ini bertujuan untuk mengunjungi semua kemungkinan Langkah yang dapat dilakukan pemain).
7. Terapkan move dengan parameter i untuk rubikTemp. Hitung Hn baru setelah move.
8. Tambahkan i ke dalam list move.
9. Tambahkan fn dengan 1 dan hn yang baru. Masukan tuple baru dengan list move baru dan fn ke dalam antrian.
10. Kembalikan gerakan rubik dan fn.
11. Jika elemen pertama sudah *goal*, terapkan semua list gerakan ke rubik.

Gambar 3.8 Implementasi algoritma A*

Sumber: dokumen penulis

```java
// moveList
for (int i = 0; i < 12; i++){
    rubikTemp.move(i);
    Integer hnNew = Algo.getHn(rubikTemp);
    // moveList
    List<Integer> moveListNew = new ArrayList<>
    (moveList);
    moveListNew.add(i);
    // fn = gn + hn
    fn = fn + 1 + hnNew;
    // make the Tuple of move
    Tuple<List<Integer>, Integer> tupleNew = new
    Tuple<>(moveListNew, fn);
    // add to the queue
    prioQueue.add(tupleNew);
    // unmove the move
    // if even then move(i+1) else move(i-1)
    if (i % 2 == 0){
        rubikTemp.move(i+1);
    }
    else{
        rubikTemp.move(i-1);
    }
    fn -= 1;
    fn -= hnNew;
}
```

Gambar 3.9 Isi kode dari iterasi dalam gambar 3.7

Sumber: dokumen penulis

IV. PENGUJIAN DAN EVALUASI

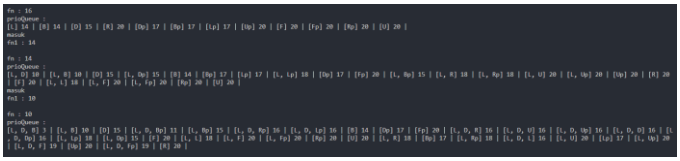
Pengujian dilakukan dengan menguji beberapa kali *shuffle* sederhana. Pengujian dengan apa saja gerakan-gerakan yang diterapkan dan apakah hasil list gerakan yang di berikan benar dalam mencapai simpul *goal.*

A. Pengujian

1. Pengujian pertama

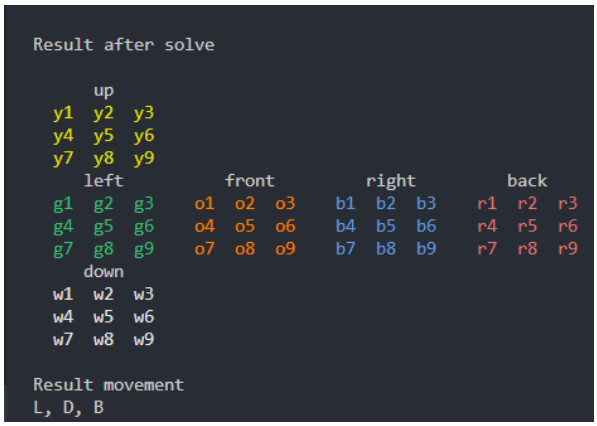
Gambar 4.1 Kondisi rubik untuk pengujian dengan 3 kali gerakan pengujian pertama

Sumber: dokumen penulis



Gambar 4.2 Gambar kondisi priority queue untuk pengujian dengan 3 kali gerakan pengujian pertama

Sumber: dokumen penulis

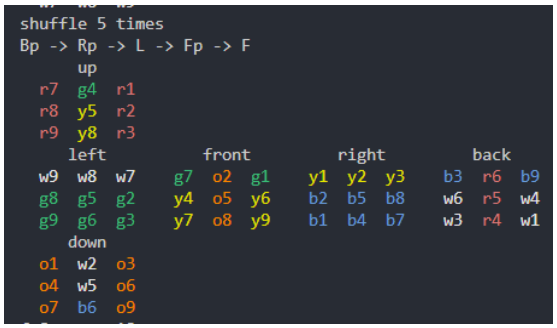


Gambar 4.3 Hasil kondisi rubik setelah pengujian pertama

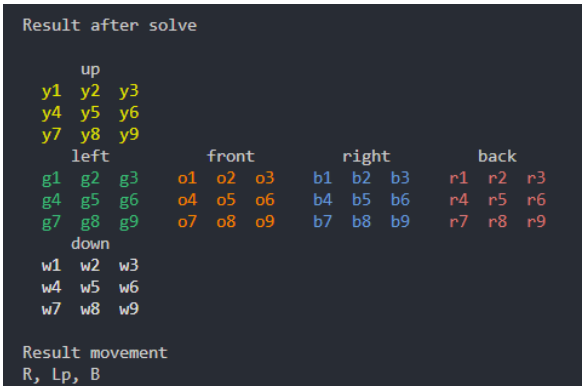
Sumber: dokumen penulis

Dari gambar 4.2 dapat diketahui bahwa hasil dari pengujian untuk 3 kali *shuffle* dengan kondisi pengocokan awal B’ -> D’ -> L’ menunjukkan hasil yang tepat. Solusi yang diberikan adalah L -> D -> B, solusi ini benar karena jika kita traversal secara harfiah dari pengocokan awal dengan membalik berlawanan arah jarum jam menjadi searah akan mendapatkan *node goal*.

2. Pengujia kedua



Gambar 4.4 Kondisi rubik untuk pengujian dengan 5 kali gerakan pengujian kedua

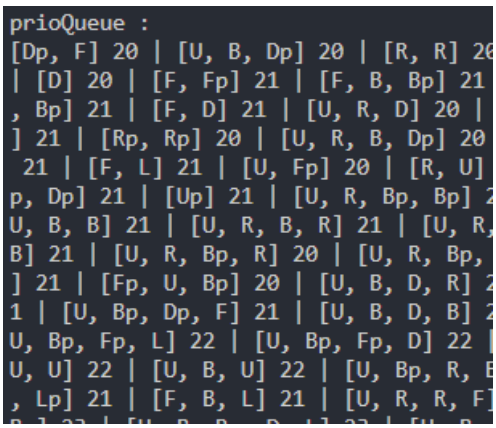


Gambar 4.5 Hasil kondisi rubik setelah pengujian kedua

Pada hasil di atas, dapat dilihat juga bahwa algoritma bekerja dengan cepat dan tepat untuk menghasilkan list solusi.

3. Pengujian ketiga

Gambar 4.6 Kondisi rubik untuk pengujian dengan 30 kali gerakan pengujian ketiga



Gambar 4.7 Kondisi setelah pengujian ketiga dengan 30 kali gerakan

Kondisi di atas adalah kondisi antrian untuk 30 kali *shuffle*, dapat dilihat bahwa antrian tidak terkendali dan membutuhkan waktu penyelesaian yang sangat lama (pengujian tidak menunggu hingga selesai dan langsung di-*terminate*).

B. Evaluasi

Terdapat evaluasi untuk pengujian pertama dan kedua yaitu kondisi rubik masih cukup sederhana sehingga algoritma dapat menyelesaikannya ddengan cepat dan tepat. Untuk pengujian dengan 30 kali *shuffle*, algoritma tidak dapat menyelesaikannya dengan cepat. Pada kasus ini, permasalahannya terletak pada fungsi heuristik yang digunakan yang masih belum admissible.

Fungsi h(n) pada algoritma ini masih dapat lebih besar dari nilai sebenarnya. Contohnya saja jika hanya menggerakan satu sisi rubik, akan memiliki nilai hn 8 yang melebihi nilai sebenernya yaitu 1.

V. KESIMPULAN DAN SARAN

Rubik's cube merupakan salah satu penemuan paling berpengaruh di dunia. Kubus ini merupakan teka-teki mekanik yang digunakan untuk penelitian ilmiah dan pengembangan teknologi oleh banyak sarjana. Terdapat banyak cara untuk menyelesaikan permaianan ini. Pada makalah ini telah dibuat sebuah algoritma penyelesaian kubus ini dengan menggunakan algoritma A*. Algoritma ini memiliki fungsi g(n) sebagai banyak gerakan yang sudah dijalankan dan h(n) sebagai banyaknya ketidakcocokan posisi dari bagian rubik yaitu *edge* dan *corner*-nya.

Pemecahan permainan *rubik's cube* dapat dilakukan dengan cukup baik menggunakan algoritma A*. Pembaharuan fungsi heuristik sangat diperlukan untuk tercapainya algoritma yang efektif dan efisien serta memberikan nilai taksiran yang tidak melebihi nilai asli.

VI. UCAPAN TERIMA KASIH

Puji dan syukur saya panjatkan kepada Tuhan Yang Maha Esa karena atas berkat rahmat dan karunia-Nya sehingga saya dapat menyelesaikan penulisan makalah ini dan implementasi kode program untuk penyelesaian makalah ini. Ucapan terima kasih saya sampaikan kepada keluarga saya yang telah memberikan dukungan dan do'a kepada saya dalam menjalani pendidikan saya di ITB. Saya juga ucapkan terima kasih kepada dosen pengampu kelas 1 mata kuliah Strategi Algoritma tahun 2023 yaitu Bapak Rinaldi Munir atas pengajarannya selama satu semester ini sehingga saya dapat menyelesaikan pembuatan makalah ini dengan baik.

LINK REPOSITORI

https://github.com/NerbFox/Rubiks-Cube-Algorithm

REFERENSI

[1] Andy. "Admissibility of A* Algorithm". https://www.geeksforgeeks.org/a-is-admissible/. Diakses pada 22 Mei 2023.

[2] Munir, Rinaldi. 2021. "Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search Bahan Kuliah IF2211 Strategi Algoritma". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf. Diakses pada 22 Mei 2023.

[3] Munir, Rinaldi. 2021. "Penentuan Rute (Route/Path Planning) Bagian 2: Algoritma A* Bahan Kuliah IF2211 Strategi Algoritma". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf. Diakses pada 22 Mei 2023.

[4] Russell, S.J. & Peter Norvig. (2002). "Artificial Intelligence: A Modern Approach". Prentice Hall.

[5] Touma, Hisham. “A* Search” https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search. Diakses pada 22 Mei 2023.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[6] Zeng, D.X., Li, M., Wang, J.J., Hou, Y.L., Lu, W.J., & Zhen Huang. “Overview of Rubik’s *Cube* and Reflections on Its Application in Mechanism”. Chin. J. Mech. Eng. **31**, 77 (2018). https://doi.org/10.1186/s10033-018-0269-7. Overview of Rubik’s *Cube* and Reflections on Its Application in Mechanism | Chinese Journal of Mechanical Engineering | Full Text (springeropen.com). Diakses pada 22 Mei 2023.

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, 22 Mei 2023

Nigel Sahl dan 13521043