KELOMPOK 14

PENYELESAIAN KASUS 8-PUZZLES

Dengan Metode A*

ANGGOTA

NADIAH NURI AISYAH

5025211210

ZAKIA KOLBI

5025211049

SEKAR AMBAR ARUM

5025211041

H1 : MISPLACED-TILES

H2 : MANHATTAN DISTANCE

8-PUZZLE METODE A*

8 PUZZLE ADALAH PERMAINAN YANG MEMILIKI 8 PERSEGI BERNOMOR 1-8 DALAM BINGKAI DENGAN TINGGI 3 PERSEGI DAN LEBAR 3 PERSEGI, MENYISAKAN SATU RUANG PERSEGI KOSONG. PERSEGI DI BARIS ATAU KOLOM YANG SAMA DENGAN POSISI TERBUKA DAPAT DIPINDAHKAN DENGAN MENGGESERNYA MASING-MASING SECARA HORIZONTAL ATAU VERTIKAL. TUJUAN DARI PUZZLE INI ADALAH UNTUK MENEMPATKAN PERSEGI DALAM URUTAN NUMERIK YANG RUNTUT.

8-PUZZLE

★★★★★

8 puzzle merupakan salah satu implementasi dari Artificial Intelegence. Dalam proses penyelesaiannya banyak terdapat algoritma-algoritma pencarian yang dapat diterapkan. Kali ini, kelompok kami telah menerapkan metode A*.

★★★★★

Metode A* merupakan metode pencarian rute dengan menggunakan teknik heuristik. Teknik heuristik digunakan untuk meningkatkan evisiensi waktu terhadap pencarian rute. Rute yang dicari hanyalah dua lokasi saja, yaitu lokasi awal dan lokasi akhir. Metode A* adalah algoritma komputer yang digunakan secara luas dalam mencari jalur (path finding) dan grafik traversal, proses plotting sebuah jalur melintang secara efisien antara titik-titik yang disebut node.

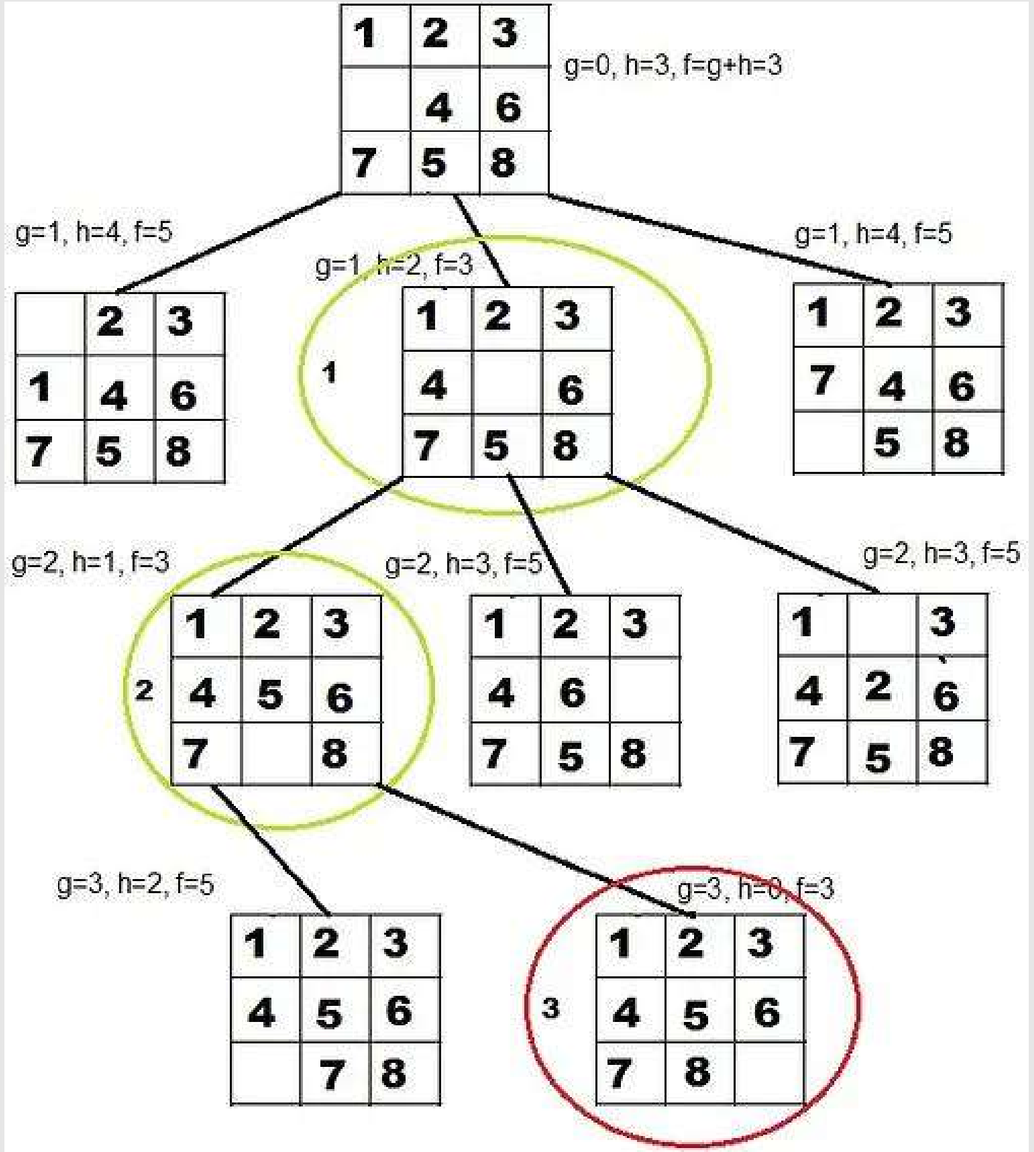
METODE A*

Notasi yang dipakai oleh algoritma A* adalah sebagai berikut:

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

dimana

- f(n) = Nilai Heuristic
- g(n) = Jumlah gerakan kotak putih
- h(n) = Jumlah kotak yang tidak bersesuaian



Kelebihan algoritma A* antara lain :

- Waktu pencarian algoritma A* dalam menemukan rute lebih cepat dibanding algoritma pencarian yang lain.
- Jumlah Loop A star lebih sedikit.
- Rute yang ditemukan dapat berbeda tapi mempunyai biaya yang sama.

(KODE) MISPLACED-TILES

- KE HALAMAN AGENDA -

```
    def f(self,start,goal):
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp

```

```
    def process(self):
        print("Masukkan Initial State \n")
        start = self.accept()
        print("\nMasukkan Goals State \n")
        goal = self.accept()

        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        self.open.append(start)
        print("\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
```

```
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            print("\nHeuristic Value(Misplaced) : ",self.h(cur.data,goal))
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]

            self.open.sort(key = lambda x:x.fval,reverse=False)


puz = Puzzle(3)
puz.process()
```

(OUTPUT) MISPLACED-TILES

```
Masukkan Initial State

1 2 3
_ 5 6
4 7 8

Masukkan Goals State

1 2 3
4 5 6
7 8 _
```

```
  |
  |
 \'/

1 2 3
_ 5 6
4 7 8

Heuristic Value(Misplaced) :  3

  |
  |
 \'/

1 2 3
4 5 6
_ 7 8

Heuristic Value(Misplaced) :  2
```

```
  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

Heuristic Value(Misplaced) :  1

  |
  |
 \'/

1 2 3
4 5 6
7 8 _

Heuristic Value(Misplaced) :  0
```

(KODE)
MANHATTAN DISTANCE

- KE HALAMAN AGENDA -

```cpp
#include <bits/stdc++.h>
using namespace std;
#define N 3

struct Node
{
    Node* parent;   //parent node

    int mat[N][N];
    int x, y;       //koordinat tile kosong
    int cost;       //jarak tile ke posisinya yg benar
    int level;      //jumlah move yang telah dilakukan
};

//fungsi untuk mencetak puzzle
int printMatrix(int mat[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}
```

```cpp
// Fungsi untuk mengalokasikan node baru
Node* newNode(int mat[N][N], int x, int y, int newX,
              int newY, int level, Node* parent)
{
    Node* node = new Node;

    // set pointer untuk path ke root
    node->parent = parent;

    // copy data dari parent node ke current node
    memcpy(node->mat, mat, sizeof node->mat);

    // menggeser tile sebayak 1
    swap(node->mat[x][y], node->mat[newX][newY]);

    // set jumlah dari tile yang tidak pada tempatnya
    node->cost = INT_MAX;

    // set jumlah move yang telah dilakukan
    node->level = level;

    // mengupdate koordinat xy yang baru
    node->x = newX;
    node->y = newY;

    return node;
}
```

```c
// bawah, kiri, atas, kanan
int row[] = { 1, 0, -1, 0 };
int col[] = { 0, -1, 0, 1 };

//fungsi untuk menghitung cost/jarak
int calculateCost(int initial[N][N], int final[N][N])
{
    int count = 0;
    for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        if (initial[i][j] && initial[i][j] != final[i][j])
        count++;
    return count;
}

// fungsi untuk mengecek apakah koordinat tile kosong safe untuk diproses
int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}
```

```c
// print path dari root node ke destination node
void printPath(Node* root)
{
    if (root == NULL)
        return;
    printPath(root->parent);
    printMatrix(root->mat);

    printf("\n");
}

// membandingkan objek untuk menyusun urutan
struct comp
{
    bool operator()(const Node* lhs, const Node* rhs) const
    {
        return (lhs->cost + lhs->level) > (rhs->cost + rhs->level);
    }
};
```

```cpp
//fungsi untuk menyelesaikan puzzle NxN
void solve(int initial[N][N], int x, int y,
           int final[N][N])
{
    // membuat priority queue untuk menyimpan live nodes dari
    // search tree;
    priority_queue<Node*, std::vector<Node*>, comp> pq;

    // membuat root node dan menghitung costnya
    Node* root = newNode(initial, x, y, x, y, 0, NULL);
    root->cost = calculateCost(initial, final);

    // menambahkan root ke daftar live nodes;
    pq.push(root);

    // mencari live node dengan cost paling sedikit,
    // menambahkan node childnya ke list live node dan
    // akhirnya dihapus dari list.
    while (!pq.empty())
    {
        // Mencari live node dengan estimasi cost paling sedikit
        Node* min = pq.top();

        // node yang ditemukan dihapus dari list live node
        pq.pop();

        // jika min merupakan jawabannya
        if (min->cost == 0)
        {
            // print path dari root ke destination;
            printPath(min);
            return;
        }

        // dilakukan untuk setiap child dari min
        // maksimal 4 child tiap node
        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y + col[i]))
            {
                // membuat child node dan menghitung
                // costnya
                Node* child = newNode(min->mat, min->x,
                              min->y, min->x + row[i],
                              min->y + col[i],
                              min->level + 1, min);
                child->cost = calculateCost(child->mat, final);

                // Masukkan child ke list live node
                pq.push(child);
            }
        }
    }
}
```

```
150  int main()
151  {
152
153      //initial state
154      int initial[N][N] =
155      {
156          {1, 2, 5},
157          {3, 4, 0},
158          {6, 7, 8}
159      };
160
161      //final state
162      int final[N][N] =
163      {
164          {0, 1, 2},
165          {3, 4, 5},
166          {6, 7, 8}
167      };
168
```

```
169      //koordinat tile kosong
170      int x = 1, y = 2;
171
172      solve(initial, x, y, final);
173
174      return 0;
175  }
```

(OUTPUT)
MANHATTAN DISTANCE

1	2	5
3	4	0
6	7	8

1	2	0
3	4	5
6	7	8

1	0	2
3	4	5
6	7	8

0	1	2
3	4	5
6	7	8


THANK YOU!!