Astar(Cpp)

```
#include <bits/stdc++.h>
using namespace std;
#define N 3
struct Node
{
Node *p;
int mat[N][N];
int x, y;
int h;
int g;
int f;
};


int pr(int mat[N][N])
{
for (int i = 0; i < N; i++)
{
for (int j = 0; j < N; j++)
{
cout << mat[i][j] << " ";
}
cout << endl;
}


}


Node *newNode(int mat[N][N], int x, int y, int nx, int ny, int g, Node *p)
{
Node *node = new Node;
node->p = p;
for (int i = 0; i < N; i++)
{
for (int j = 0; j < N; j++)
{
node->mat[i][j] = mat[i][j];

}
}
```

```
swap(node->mat[x][y], node->mat[nx][ny]);
node->h = INT_MAX;
node->g = g;
node->x = nx;
node->y = ny;
return node;
}
int diff(int imat[N][N], int res[N][N])
{
int count = 0;
for (int i = 0; i < N; i++)
{
for (int j = 0; j < N; j++)
{
if (imat[i][j]!= res[i][j])
{
count++;
}
}
}
return count;
}
int r[] = {1, 0, -1, 0};
int c[] = {0, -1, 0, 1};
int move(int x, int y)
{
return (x >= 0 && x < N && y >= 0 && y < N);
}
void trv(Node *root)
{
if (root == NULL)

{
return;
}
trv(root->p);
pr(root->mat);
cout << endl;
}
struct comp
{
bool operator()(Node *a, Node *b)
{
```

```
return (a->f) > (b->f);
}
};
void solve(int imat[N][N], int x, int y, int res[N][N])
{
priority_queue<Node *, vector<Node *>, comp> pq;
Node *root = newNode(imat, x, y, x, y, 0, NULL);
root->h = diff(imat, res);
root->f = root->g + root->h;
pq.push(root);
while (!pq.empty())
{
Node *temp = pq.top();
pq.pop();
if (temp->h == 0)
{
trv(temp);
cout<<endl;
return;
}

for (int i = 0; i < 4; i++)
{
if (move(temp->x + r[i], temp->y + c[i]))
{
Node *child = newNode(temp->mat, temp->x, temp->y,( temp->x + r[i]), (temp->y +
c[i]), (temp->g + 1), temp);
child->h = diff(child->mat, res);
child->f = child->g + child->h;
pq.push(child);
}
}
}
}
int main()
{
int imat[N][N] =
{
{2, 8, 3},
{1, 6, 4},
{7, 0, 5}};
int res[N][N] =
{
{1, 2, 3},
```

```
{8, 0, 4},
{7, 6, 5}};
int x = 2, y = 1;
solve(imat, x, y, res);
return 0;
}
```

Astar(Python)

```
class Node:
        def __init__(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval

        def generate_child(self):
        x,y = self.find(self.data,'_')
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
        child = self.shuffle(self.data,x,y,i[0],i[1])
        if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children

        def shuffle(self,puz,x1,y1,x2,y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
        temp_puz = []
        temp_puz = self.copy(puz)
        temp = temp_puz[x2][y2]
        temp_puz[x2][y2] = temp_puz[x1][y1]
        temp_puz[x1][y1] = temp
        return temp_puz
        else:
        return None


        def copy(self,root):
        temp = []
        for i in root:
        t = []
```

```
        for j in i:
                t.append(j)
        temp.append(t)
        return temp

        def find(self,puz,x):
        for i in range(0,len(self.data)):
        for j in range(0,len(self.data)):
                if puz[i][j] == x:
                return i,j

class Puzzle:
        def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []
        def accept(self):
        puz = []
        for i in range(0,self.n):
        temp = input().split(" ")
        puz.append(temp)
        return puz
        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("Entr the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        self.open.append(start)
        print("\n\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("")
        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()
```