Name: Nitin Choudhary
Roll no: 50
Class: TY-IT A B3

8-Puzzle using AStar :

```
#include <bits/stdc++.h>
using namespace std;

static const auto init = []
{
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    return false;
}();

class Node
{
public:
    Node *parent;
    int cost;
    int hval;
    vector<vector<int>> board;
    vector<Node> Children;
    void Hfun(Node &Goal)
    {
        int count = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (board[i][j] != Goal.board[i][j])
                {
                    count++;
                }
            }
        }
        hval = count;
    }
    void PrintBoard()
    {
        for (int i = 0; i < 3; i++)
        {
```

```
            for (int j = 0; j < 3; j++)
            {
                cout << board[i][j] << " ";
            }
            cout << "\n";
        }
        cout << endl;
    }
    void GenerateChildren(int x, int y, Node &Goal)
    {
        Node child;
        child.parent = this;
        child.board = board;
        child.cost = cost + 1;
        if (x + 1 < 3)
        {
            swap(child.board[x + 1][y], child.board[x][y]);
            child.Hfun(Goal);
            Children.push_back(child);
        }
        if (y + 1 < 3)
        {
            child.board = board;
            swap(child.board[x][y + 1], child.board[x][y]);
            child.Hfun(Goal);
            Children.push_back(child);
        }
        if (x - 1 >= 0)
        {
            child.board = board;
            swap(child.board[x - 1][y], child.board[x][y]);
            child.Hfun(Goal);
            Children.push_back(child);
        }
        if (y - 1 >= 0)
        {
            child.board = board;
            swap(child.board[x][y - 1], child.board[x][y]);
            child.Hfun(Goal);
            Children.push_back(child);
        }
    }
};

class Position
```

```
{
public:
    int x;
    int y;
    void zeroposition(vector<vector<int>> &temp)
    {
        int i, j;
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                if (temp[i][j] == 0)
                {
                    x = i;
                    y = j;
                    break;
                }
            }
        }
    }
};
void PrintList(list<Node> List, string Type)
{
    cout << Type << " List\n";
    for (int i = 0; i < 3; i++)
    {
        for (auto j : List)
        {
            cout << j.board[i][0] << " " << j.board[i][1] << " " <<
j.board[i][2] << "   ";
        }
        cout << "\n";
    }
    cout << "**************************************\n";
}
int main()
{
    // Goal State
    Node Goal;
    Goal.board = {{1, 2, 3}, {4, 8, 5}, {0, 7, 6}};
    Goal.hval = 0;

    // Intial State
    Node Intial;
    Intial.board = {{1,2,3}, {4,5,6}, {7, 8, 0}};
```

```
    Intial.Hfun(Goal);
    Intial.parent = NULL;
    Intial.cost = 0;

    int minf = INT_MAX;
    Position curr; // empty tile position
    Node Parent, Child;

    list<Node> Open;
    list<Node> Closed;

    // Push Intial Node in Open List
    Open.push_front(Intial);
    PrintList(Open, "OPEN");
    PrintList(Closed, "CLOSED");

    while (!Open.empty())
    {
        // Pick Node with Least f value
        minf = INT_MAX;
        std::list<Node>::iterator minIt = Open.begin();
        for (auto i = Open.begin(); i != Open.end(); ++i)
        {
            if (i->cost + i->hval < minf)
            {
                minf = i->cost + i->hval;
                minIt = i;
            }
        }
        // Pop Node with least f-value
        Parent = *minIt;
        Open.erase(minIt);

        // Generate Child Nodes of Parent Node
        curr.zeroposition(Parent.board);
        Parent.GenerateChildren(curr.x, curr.y, Goal);
        bool goalflag = false;
        Closed.push_back(Parent);
        // Evaluate each sucessor of parent
        for (auto i : Parent.Children)
        {
            i.PrintBoard();
            if (i.board == Goal.board)
            {
                cout << "\nGoal Reached\n";
```

```
                cout << "Cost  = " << i.cost << "\n";
                Closed.push_back(i);
                goalflag = true;
                break;
            }
            minf = i.hval + i.cost;
            bool flag = true;
            // check for child node in Open List with same position with
lower cost
            for (auto j : Open)
            {
                if (i.board == j.board && i.cost < j.cost)
                {
                    j = i;
                    break;
                }
            }

            // check for child node in Closed List with same position with
lower cost
            for (auto j : Closed)
            {
                if (i.board == j.board && i.cost > j.cost)
                {
                    flag = false;
                    break;
                }
            }
            if (flag)
            {
                Open.push_back(i);
            }
        }
        cout << "****************************" << endl;
        PrintList(Open, "OPEN");
        PrintList(Closed, "CLOSED");
        if (goalflag)
        {
            break;
        }
    }

    return 0;
}
```

Output:

OPEN List
1 2 3
4 5 6
7 8 0
****
CLOSED List

****
1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 0 8

****
OPEN List
1 2 3 1 2 3
4 5 0 4 5 6
7 8 6 7 0 8
****
CLOSED List
1 2 3
4 5 6
7 8 0
****
1 2 3
4 5 6
7 8 0

1 2 0
4 5 3
7 8 6

1 2 3
4 0 5
7 8 6

****
OPEN List
1 2 3 1 2 0 1 2 3
4 5 6 4 5 3 4 0 5
7 0 8 7 8 6 7 8 6
****
CLOSED List
1 2 3 1 2 3
4 5 6 4 5 0
7 8 0 7 8 6
****
1 2 3
4 8 5
7 0 6

1 2 3
4 5 0
7 8 6

1 0 3
4 2 5
7 8 6

1 2 3
0 4 5
7 8 6

****
OPEN List
1 2 3 1 2 0 1 2 3 1 0 3 1 2 3
4 5 6 4 5 3 4 8 5 4 2 5 0 4 5
7 0 8 7 8 6 7 0 6 7 8 6 7 8 6
****
CLOSED List
1 2 3 1 2 3 1 2 3
4 5 6 4 5 0 4 0 5

7 8 0 7 8 6 7 8 6
****
1 2 3
4 8 5
7 6 0

1 2 3
4 0 5
7 8 6

1 2 3
4 8 5
0 7 6

Goal Reached
Cost = 4
****
OPEN List
1 2 3 1 2 0 1 0 3 1 2 3 1 2 3
4 5 6 4 5 3 4 2 5 0 4 5 4 8 5
7 0 8 7 8 6 7 8 6 7 8 6 7 6 0
****
CLOSED List
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
4 5 6 4 5 0 4 0 5 4 8 5 4 8 5
7 8 0 7 8 6 7 8 6 7 0 6 0 7 6
****