Name: Satyam Jaiswal **UID:** 2021600028 **Batch**: B **Date:** 05/08/2023

EXP 2: Informed Search Strategy

Problem: 15 Puzzle problem. Solve it using the A* algorithm.

Show the status of OPEN and Close at every intermediate stage.

Show the solution path along with depth.

show no count of open nodes and closed nodes.

Using heuristic: Misplaced Tiles

Program:

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

int misplaced(vector<vector<int>>& puzzle){
   int val = 1;
   int count = 0;
   for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
         if(val==16) val = -1;
         if(val!=puzzle[i][j]) count++;
         val++;
      }
   }
   return count;
}

vector<int> emptyLoc(vector<vector<int>>& puzzle){
   for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
         if(puzzle[i][j]==-1) return {i,j};
      }
```

```
    }
    return {3,3};
}

vector<vector<int>> moveUp(vector<vector<int>>& puzzle,int i,int j){
    vector<vector<int>> curr = puzzle;
    int temp = curr[i][j];
    curr[i][j] = curr[i-1][j];
    curr[i-1][j] = temp;
    return curr;
}

vector<vector<int>> moveDown(vector<vector<int>>& puzzle,int i,int j){
    vector<vector<int>> curr = puzzle;
    int temp = curr[i][j];
    curr[i][j] = curr[i+1][j];
    curr[i+1][j] = temp;
    return curr;
}

vector<vector<int>> moveRight(vector<vector<int>>& puzzle,int i,int j){
    vector<vector<int>> curr = puzzle;
    int temp = curr[i][j];
    curr[i][j] = curr[i][j+1];
    curr[i][j+1] = temp;
    return curr;
}

vector<vector<int>> moveLeft(vector<vector<int>>& puzzle,int i,int j){
    vector<vector<int>> curr = puzzle;
```

```
    int temp = curr[i][j];
    curr[i][j] = curr[i][j-1];
    curr[i][j-1] = temp;
    return curr;
}

void display(vector<vector<int>>& puzzle){
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            cout<<puzzle[i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<"Misplaced Tiles: "<<misplaced(puzzle)<<endl;
}

vector<pair<int, vector<vector<int>>>> generateNextStates(int cost,vector<vector<int>>&
curr){
    vector<pair<int, vector<vector<int>>>> nextStates;
    vector<int> empty = emptyLoc(curr);
    if(empty[1]!=0){
        vector<vector<int>> leftMoved = moveLeft(curr,empty[0],empty[1]);
        cout<<"Left Move: f-cost = "<<cost+misplaced(leftMoved)<<endl;
        display(leftMoved);
        nextStates.push_back(make_pair(cost+misplaced(leftMoved),leftMoved));
    }
    if(empty[0]!=0){
        vector<vector<int>> upMoved = moveUp(curr,empty[0],empty[1]);
        cout<<"Up Move: f-cost = "<<cost+misplaced(upMoved)<<endl;
        display(upMoved);
```

```
        nextStates.push_back(make_pair(cost+misplaced(upMoved),upMoved));
    }
    if(empty[1]!=3){
        vector<vector<int>> rightMoved = moveRight(curr,empty[0],empty[1]);
        cout<<"Right Move: f - cost = "<<cost+misplaced(rightMoved)<<endl;
        display(rightMoved);
        nextStates.push_back(make_pair(cost+misplaced(rightMoved),rightMoved));
    }
    if(empty[0]!=3){
        vector<vector<int>> downMoved = moveDown(curr,empty[0],empty[1]);
        cout<<"Down Move: f-cost = "<<cost+misplaced(downMoved)<<endl;
        display(downMoved);
        nextStates.push_back(make_pair(cost+misplaced(downMoved),downMoved));
    }
    return nextStates;
}

bool isGoal(vector<vector<int>> curr){
    if(misplaced(curr)==0) return true;
    return false;
}

void astar(vector<vector<int>>& puzzle, priority_queue<pair<int, vector<vector<int>>>,
vector<pair<int, vector<vector<int>>>>, greater<pair<int, vector<vector<int>>>>>& open,
set<pair<int, vector<vector<int>>>>& closed) {
    open.push(make_pair(misplaced(puzzle), puzzle));
    while (!open.empty()) {
        cout<<"Open: "<<open.size()<<" & Closed: "<<closed.size()<<endl;
        pair<int, vector<vector<int>>> Node = open.top();
        open.pop();
        int cost = Node.first;
```

```
        vector<vector<int>> curr_state = Node.second;
        int depth = cost - misplaced(curr_state);
        if (isGoal(curr_state)) {
            cout << endl << "Goal: " << endl;
            display(curr_state);
            return;
        }
        if (closed.find(make_pair(cost, curr_state)) != closed.end()) {
            continue;
        }
        vector<pair<int, vector<vector<int>>>> nextStates = generateNextStates(depth + 1,
curr_state);
        for (auto it : nextStates) {
            open.push(it);
        }
        closed.insert(make_pair(cost, curr_state));
    }
}



int main(){
    #ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    #endif
    vector<vector<int>> puzzle = {{1, 2, 3, 4},
                                  {5, 6, 7, -1},
                                  {9, 10, 11, 8},
                                  {13, 14, 15, 12}};
    priority_queue<pair<int, vector<vector<int>>>, vector<pair<int, vector<vector<int>>>>,
greater<pair<int, vector<vector<int>>>>> open;
```

```
    set<pair<int, vector<vector<int>>>> closed;
    vector<vector<int>> curr = moveUp(puzzle,3,3);
    display(puzzle);
    cout<<endl<<"Applying a-star:"<<endl;
    astar(puzzle,open,closed);
}
```

Output:

```
1 2 3 4
5 6 7 -1
9 10 11 8
13 14 15 12
Misplaced Tiles: 3

Applying a-star:
Open: 1 & Closed: 0
Left Move: f-cost = 5
1 2 3 4
5 6 -1 7
9 10 11 8
13 14 15 12
Misplaced Tiles: 4
Up Move: f-cost = 5
1 2 3 -1
5 6 7 4
9 10 11 8
13 14 15 12
Misplaced Tiles: 4
Down Move: f-cost = 3
1 2 3 4
5 6 7 8
```

9 10 11 -1

13 14 15 12

Misplaced Tiles: 2

Open: 3 & Closed: 1

Left Move: f-cost = 5

1 2 3 4

5 6 7 8

9 10 -1 11

13 14 15 12

Misplaced Tiles: 3

Up Move: f-cost = 5

1 2 3 4

5 6 7 -1

9 10 11 8

13 14 15 12

Misplaced Tiles: 3

Down Move: f-cost = 2

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 -1

Misplaced Tiles: 0

Open: 5 & Closed: 2

Goal:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 -1

Misplaced Tiles: 0

Time and Space Complexity:

Space Complexity:

Open Set Space Complexity: O(|V|) or O(|E|)

Closed Set Space Complexity: O(|V|) or O(|E|)

Time Complexity:

Time Complexity: O(b^d) (worst-case), where b is the branching factor and d is the depth of the solution.

Conclusion: 1) Learnt about the informed search strategy such as a *.

2) Also learnt how heuristic is calculated for the 15-puzzle problem.

3) Also learnt how to solve the 15-puzzle problem using the a* algorithm.