NAME: PARAM RASIKLAL KANADA

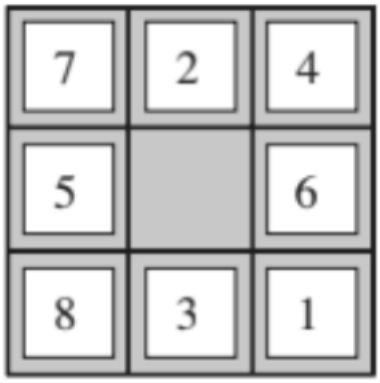
ADD. NO. :U24AI047

ARTIFICIAL INTELLIGENCE (AI202) LAB PRACTICALS

ASSIGNMENT 2

Q1 The following figure shows the start state and goal state for the 8-puzzle problem. Find the number of states explored before reaching the goal state using the BFS algorithm.

Q2 Find the number of states explored for the DFS algorithm. Give comparison of BFS and DFS in terms of cost if the path cost for the state is given as depth of the search tree at which the state is occurring.



Start State

Goal State

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <set>
#include <algorithm> // Needed for swap

using namespace std;

const vector<int> GOAL_STATE = {0,1,2,3,4,5,6,7,8};
```

```
int findZero(const vector<int>& state){
    for (int i=0;i<9;i++){
        if(state[i]==0) return i;
    }
    return -1;
}

bool tryMove(const vector<int>& current, vector<int>& next, int direction ){
    int zeroIndex = findZero(current);
    int row = zeroIndex / 3;
    int col = zeroIndex % 3;

    int newRow = row;
    int newCol = col;

    if (direction == 0) newRow--;       // Up
    else if (direction == 1) newRow++;  // Down
    else if (direction == 2) newCol--;  // Left
    else if (direction == 3) newCol++;  // Right

    if (newRow>=0 && newRow<3 && newCol>=0 && newCol<3){
        next = current;
        int newIdx = newRow * 3 + newCol;
        swap(next[zeroIndex], next[newIdx]); // Swap empty tile with target
        return true;
    }
    return false;
}

void solveBFS(const vector<int>& startState) {
    queue<vector<int>> q;
```

```
set<vector<int>> visited;

q.push(startState);
visited.insert(startState);

int statesExplored = 0;
bool found = false;

cout << "Starting BFS search..." << endl;

while (!q.empty()) {
    vector<int> current = q.front();
    q.pop();

    statesExplored++;

    if (current == GOAL_STATE) {
        found = true;
        break;
    }

    for (int i = 0; i < 4; i++) {
        vector<int> nextState;
        if (tryMove(current, nextState, i)) {
            if (visited.find(nextState) == visited.end()) {
                visited.insert(nextState);
                q.push(nextState);
            }
        }
    }
}
```

```
    if (found) {
        cout << "Goal Reached!" << endl;
        cout << "Total states explored: " << statesExplored << endl;
    } else {
        cout << "Goal not reachable." << endl;
    }
}

struct Node {
    vector<int> state;
    int depth;
};

void solveDFS(const vector<int>& startState) {
    const int MAX_DEPTH = 47;
    stack<Node> s;
    set<vector<int>> visited;

    s.push({startState, 0});
    visited.insert(startState);

    int statesExplored = 0;
    bool found = false;
    int finalDepth = 0;

    cout << "Starting DFS (Max Depth:" << MAX_DEPTH << ") search..." << endl;

    while (!s.empty()) {
        Node current = s.top();
        s.pop();

        statesExplored++;
```

```
    if (current.state == GOAL_STATE) {
        found = true;
        finalDepth = current.depth;
        break;
    }

    if(current.depth < MAX_DEPTH) {
        for (int i = 0; i < 4; i++) {
            vector<int> nextState;

            // ERROR FIX 1: Pass 'current.state' instead of 'current'
            if (tryMove(current.state, nextState, i)) {

                if (visited.find(nextState) == visited.end()) {
                    visited.insert(nextState);

                    // ERROR FIX 2: Push a Node struct {state, depth}, not just state
                    s.push({nextState, current.depth + 1});
                }
            }
        }
    }
}

if (found) {
    cout << "Goal Reached!" << endl;
    cout << "Solution Depth (Moves): " << finalDepth << endl;
    cout << "Total states explored: " << statesExplored << endl;
} else {
    cout << "Goal not found within depth limit of " << MAX_DEPTH << "." << endl;
    cout << "Total states explored: " << statesExplored << endl;
```

```
    }
}

int main(){
    vector<int> startState = {7, 2, 4, 5, 0, 6, 8, 3, 1};

    solveBFS(startState);
    cout << endl;
    solveDFS(startState);

    return 0;
}
```