**Name – Armaan Asif Shaikh**

**PRN- 22311412**

**Roll No - 382041**

**AI ASSIGNMENTS**

**Assignment-1**

#include <bits/stdc++.h>

using namespace std;

struct Node {

vector<vector<int>> state;

string path;

};

vector<vector<int>> goal = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 0}

};

vector<pair<int, int>> moves = {{-1,0}, {1,0}, {0,-1}, {0,1}};

vector<char> moveChar = {'U','D','L','R'};

string boardToString(vector<vector<int>> &board) {

string s="";

for (auto &row : board)

for (int x : row) s += to\_string(x);

return s;

}

pair<int,int> findBlank(vector<vector<int>> &state) {

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

if(state[i][j]==0) return {i,j};

return {-1,-1};

}

void BFS(vector<vector<int>> start) {

queue<Node> q;

unordered\_set<string> visited;

q.push({start, ""});

visited.insert(boardToString(start));

while(!q.empty()) {

Node cur = q.front(); q.pop();

if(cur.state == goal) {

cout << "BFS Found Solution!\nPath: " << cur.path << "\n";

cout << "Moves: " << cur.path.size() << "\n";

return;

}

auto [x,y] = findBlank(cur.state);

for(int k=0;k<4;k++) {

int nx=x+moves[k].first, ny=y+moves[k].second;

if(nx>=0 && ny>=0 && nx<3 && ny<3) {

auto newState = cur.state;

swap(newState[x][y], newState[nx][ny]);

string key = boardToString(newState);

if(!visited.count(key)) {

visited.insert(key);

q.push({newState, cur.path+moveChar[k]});

}

}

}

}

cout << "No solution found.\n";

}

int main() {

// Example start state

vector<vector<int>> start = {

{1, 2, 3},

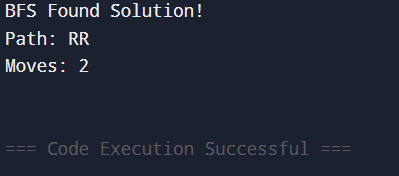
{4, 5, 6},

{0, 7, 8}

};

BFS(start);

}



**Approach – 2(DFS)**

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> goal = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 0}

};

vector<pair<int,int>> moves = {{-1,0},{1,0},{0,-1},{0,1}};

string boardToString(vector<vector<int>> &board) {

string s="";

for(auto &row : board)

for(int x : row) s += to\_string(x);

return s;

}

pair<int,int> findBlank(vector<vector<int>> &state) {

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

if(state[i][j]==0) return {i,j};

return {-1,-1};

}

bool DFSUtil(vector<vector<int>> state, unordered\_set<string> &visited, int depth, int limit) {

if(state == goal) {

cout << "Goal Found at depth " << depth << "!\n";

return true;

}

if(depth >= limit) return false;

auto [x,y] = findBlank(state);

for(auto &mv : moves) {

int nx=x+mv.first, ny=y+mv.second;

if(nx>=0 && ny>=0 && nx<3 && ny<3) {

auto newState = state;

swap(newState[x][y], newState[nx][ny]);

string key = boardToString(newState);

if(!visited.count(key)) {

visited.insert(key);

if(DFSUtil(newState, visited, depth+1, limit))

return true;

}

}

}

return false;

}

void DFS(vector<vector<int>> start, int depthLimit=20) {

unordered\_set<string> visited;

visited.insert(boardToString(start));

if(!DFSUtil(start, visited, 0, depthLimit)) {

cout << "No solution found within depth limit " << depthLimit << ".\n";

}

}

int main() {

vector<vector<int>> start = {

{1, 2, 3},

{4, 5, 6},

{0, 7, 8}

};

cout << "Running DFS...\n";

DFS(start, 15); // depth limit 15

}

