AI Assignment 3

-------------------------------

Name: Yash Oswal

Div: B Roll no: 38

SRN: 201901226

-------------------------------

**Question:** Implement the 8 puzzle using the Hill climbing algorithm.

**Code:**

#include <bits/stdc++.h>

using namespace std;

#define Size 3

struct Node{

Node\* parent;

int mat[Size][Size];

int x, y,cost,level;

};

int printMatrix(int mat[Size][Size]){

for (int i = 0; i < Size; i++){

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

printf("\t| %d ", mat[i][j]);

printf("\n");

}

return 0;

}

Node\* newNode(int mat[Size][Size], int x, int y, int newX,int newY, int level, Node\* parent){

Node\* node = new Node;

node->parent = parent;

memcpy(node->mat, mat, sizeof node->mat);

swap(node->mat[x][y], node->mat[newX][newY]);

node->cost = INT\_MAX;

node->level = level;

node->x = newX;node->y = newY;

return node;

}

int row[] = { 1, 0, -1, 0 };

int col[] = { 0, -1, 0, 1 };

int calculateCost(int initial\_State[Size][Size], int Goal\_State[Size][Size]){

int count = 0;

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

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

if (initial\_State[i][j] && initial\_State[i][j] != Goal\_State[i][j])

count++;

return count;

}

int isSafe(int x, int y){

return (x >= 0 && x < Size && y >= 0 && y < Size);

}

void printPath(Node\* root){

if (root == NULL)

return;

printPath(root->parent);

printMatrix(root->mat);

printf("\n");

}

struct comp{

bool operator()(const Node\* lhs, const Node\* rhs) const{

return (lhs->cost + lhs->level) > (rhs->cost + rhs->level);

}

};

void solve(int initial\_State[Size][Size], int x, int y,int Goal\_State[Size][Size]){

priority\_queue<Node\*, std::vector<Node\*>, comp> pq;

Node\* root = newNode(initial\_State, x, y, x, y, 0, NULL);

root->cost = calculateCost(initial\_State, Goal\_State);

pq.push(root);

while (!pq.empty()){

Node\* min = pq.top();

pq.pop();

if (min->cost == 0){

printPath(min);

return;

}

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

if (isSafe(min->x + row[i], min->y + col[i])){

Node\* child = newNode(min->mat, min->x,

min->y, min->x + row[i],

min->y + col[i],

min->level + 1, min);

child->cost = calculateCost(child->mat, Goal\_State);

pq.push(child);

}

}

}

}

int main(){

int x = 1, y = 2;

int initial\_State[Size][Size] ={

{1, 2, 3},

{8, 6, 4},

{7, 0, 5}

};

int Goal\_State[Size][Size] ={

{2, 8, 1},

{0, 4, 3},

{7, 6, 5}

};

solve(initial\_State, x, y, Goal\_State);

return 0;

}