**CS-304-AI -LAB(LAB TASK-3)**

**ROLL-423135**

**NAME-HARSHITH SUDA**

**CODE-1**

#include<bits/stdc++.h>

using namespace std;

using std::chrono::high\_resolution\_clock;

using std::chrono::duration\_cast;

using std::chrono::duration;

using std::chrono::milliseconds;

struct Node{

    Node\* parent;

    string s;

    int x,y;

    int depth;

    int heuristic;

};

int computeMismatchings(string s){

    string goal="012345678";

    int mismatchings=0;

    for(int i=0;i<s.size();i++){

        if(goal[i]!=s[i]){

            mismatchings++;

        }

    }

    return mismatchings;

}

void printVectorOfVector(vector<vector<int>>&v){

    for(int i=0;i<v.size();i++){

        for(int j=0;j<v[0].size();j++){

*// cout<<"value of i:"<<i<<"value of j:"<<j<<" ";*

            cout<<v[i][j]<<" ";

        }

        cout<<endl;

    }

    return;

}

pair<vector<vector<int>>,pair<int,int>> randomPuzzleGenerator(){

    srand(time(NULL));

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

    for(int i=0;i<100;i++){

        int m=rand()%9;

        int n=rand()%9;

        swap(arr[m],arr[n]);

    }

    vector<int>m1(arr.begin(),arr.begin()+3);

    vector<int>m2(arr.begin()+3,arr.begin()+6);

    vector<int>m3(arr.begin()+6,arr.begin()+9);

    vector<vector<int>>ans={m1,m2,m3};

    int z=0;

    for(int i=0;i<8;i++){

        if(arr[i]==0){

            z=i;

*// cout<<"value of z:"<<z<<endl;*

            break;

        }

    }

*// cout<<"value of z/3:"<<z/3<<endl;*

*// cout<<"value of z%3:"<<z%3<<endl;*

    return {ans,{z/3,z%3}};

}

int manhattenDistance(string s){

    int cost=0;

    unordered\_map<int,pair<int,int>>m{{0,{0,0}},{1,{0,1}},{2,{0,2}},{3,{1,0}},{4,{1,1}},{5,{1,2}},{6,{2,0}},{7,{2,1}},{8,{2,2}}};

    for(int i=0;i<9;i++){

        int digit=s[i]-'0';

        cost+=abs(digit/3-m[i].first)+abs(digit%3-m[i].second);

    }

*// cout<<"cost"<<cost<<endl;*

    return cost;

}

Node\* InitNode(vector<vector<int>>&state,int x,int y){

    Node\* node=new Node();

    string s;

    for(int i=0;i<3;i++){

        for(int j=0;j<3;j++){

            s+=char(state[i][j]+'0');

        }

    }

    node->s=s;

    node->x=x;

    node->y=y;

    node->parent=NULL;

    node->depth=0;

    node->heuristic=manhattenDistance(s);

    cout<<"INITIALIZED"<<node->heuristic<<endl;

    return node;

}

struct comp{

    bool operator()(const Node\* a,const Node\* b)const{

        return (a->depth+a->heuristic)>(b->depth+b->heuristic);

    }

};

bool isValid(int x,int y){

    return (x>=0 && y>=0 && x<3 && y<3);

}

void printStringToPuzzle(string s){

    cout<<s[0]<<" "<<s[1]<<" "<<s[2]<<endl;

    cout<<s[3]<<" "<<s[4]<<" "<<s[5]<<endl;

    cout<<s[6]<<" "<<s[7]<<" "<<s[8]<<endl;

    return;

}

string convertPuzzleToString(vector<vector<int>>&puzzle){

    string s;

    for(int i=0;i<3;i++){

        for(int j=0;j<3;j++){

            s+=char(puzzle[i][j]+'0');

        }

    }

    return s;

}

bool isSolvable(vector<vector<int>>&puzzle){

    vector<int>v;

    for(int i=0;i<puzzle.size();i++){

        for(int j=0;j<puzzle[0].size();j++){

            v.push\_back(puzzle[i][j]);

        }

    }

    int invcnt=0;

    for(int i=0;i<8;i++){

        for(int j=i+1;j<9;j++){

            if(v[i] && v[j] && v[i]>v[j]){

                invcnt++;

            }

        }

    }

    return (invcnt%2)==0;

}

void backtrackSolution(Node\* node){

    vector<string>v;

    while(node){

        v.push\_back(node->s);

        node=node->parent;

    }

    reverse(v.begin(),v.end());

    for(int i=0;i<v.size();i++){

        printStringToPuzzle(v[i]);

        if(i!=v.size()-1){

            cout<<"Next->"<<endl;

        }

    }

    return;

}

void AStarAlgorithm(vector<vector<int>>&state,int x,int y,string goal){

    Node\* mainNode=InitNode(state,x,y);

    unordered\_set<string>explored;

    priority\_queue<Node\*,vector<Node\*>,comp>q;

    explored.insert(mainNode->s);

    q.push(mainNode);

    vector<int>X={-1,1,0,0};

    vector<int>Y={0,0,1,-1};

    while(!q.empty()){

*// cout<<"here"<<endl;*

        Node\* curr=q.top();

        q.pop();

        if(goal==curr->s){

            backtrackSolution(curr);

*// cout<<"here"<<endl;*

            return;

        }

        for(int i=0;i<4;i++){

            if(isValid(curr->x+X[i],curr->y+Y[i])){

                string temp=curr->s;

                swap(temp[3\*(curr->x)+curr->y],temp[3\*(curr->x+X[i])+curr->y+Y[i]]);

                if(explored.find(temp)==explored.end()){

                    Node\* nodeTemp=new Node();

                    nodeTemp->parent=curr;

                    nodeTemp->s=temp;

*// printStringToPuzzle(temp);*

*// cout<<"mistmacthings"<<computeMismatchings(temp)<<endl;*

                    nodeTemp->x=curr->x+X[i];nodeTemp->y=curr->y+Y[i];

                    nodeTemp->depth=curr->depth+1;

                    nodeTemp->heuristic=manhattenDistance(temp);

                    q.push(nodeTemp);

                }

            }

        }

        explored.insert(curr->s);

    }

    return;

}

int main(){

    ios\_base::sync\_with\_stdio(false);

    cin.tie(0);

    string goal="012345678";

*// pair<vector<vector<int>>,pair<int,int>>puzzle=randomPuzzleGenerator();*

*// if(isSolvable(puzzle.first)){*

*//     AStarAlgorithm(puzzle.first,puzzle.second.first,puzzle.second.second,goal);*

*// }*

*// else{*

*//     printVectorOfVector(puzzle.first);*

*//     cout<<"IMPOSSIBLE TO SOLVE";*

*// }*

*// //IMPOSSIBLE CASE*

*// vector<vector<int>>p={{0, 7, 6},{1, 8, 3},{2, 5, 4}};*

*// if(isSolvable(p)){*

*//     BFS8Puzzle(p,0,0,goal);*

*// }*

*// else{*

*//     printVectorOfVector(p);*

*//     cout<<"IMPOSSIBLE TO SOLVE";*

*// }*

*// SOLVABLE CASE*

    auto t1=high\_resolution\_clock::now();

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

    if(isSolvable(p)){

        AStarAlgorithm(p,2,2,goal);

    }

    else{

        printVectorOfVector(p);

        cout<<"IMPOSSIBLE TO SOLVE";

    }

    auto t2=high\_resolution\_clock::now();

    duration<double,std::milli>mil\_double=t2-t1;

    double timeTaken=mil\_double.count();

    cout<<"Time Taken for A\* : "<<timeTaken<<" milliseconds"<<endl;

    return 0;

}

**CODE-2**

#include<bits/stdc++.h>

using namespace std;

using std::chrono::high\_resolution\_clock;

using std::chrono::duration\_cast;

using std::chrono::duration;

using std::chrono::milliseconds;

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

unordered\_map<int,pair<int,int>>m{{0,{0,0}},{1,{0,1}},{2,{0,2}},{3,{1,0}},{4,{1,1}},{5,{1,2}},{6,{2,0}},{7,{2,1}},{8,{2,2}}};

struct Node{

    Node\* parent;

    int x,y;

    vector<vector<int>>state;

    int cost;

    int g;

};

int manhattenDistance(vector<vector<int>>&state){

    int cost=0;

    for(int i=0;i<3;i++){

        for(int j=0;j<3;j++){

            if(state[i][j] == 0) continue;

            cost+=abs(i-m[state[i][j]].first)+abs(j-m[state[i][j]].second);

        }

    }

    return cost;

}

Node \*createRoot(vector<vector<int>>state,int x,int y){

    Node\* node=new Node();

    node->parent=NULL;

    node->x=x;node->y=y;

    node->state=state;

    node->cost=manhattenDistance(state);

    node->g=0;

    return node;

}

void printVectorOfVector(vector<vector<int>>&v){

    for(int i=0;i<v.size();i++){

        for(int j=0;j<v[0].size();j++){

*// cout<<"value of i:"<<i<<"value of j:"<<j<<" ";*

            cout<<v[i][j]<<" ";

        }

        cout<<endl;

    }

    return;

}

bool isSolvable(vector<vector<int>>&puzzle){

    vector<int>v;

    for(int i=0;i<puzzle.size();i++){

        for(int j=0;j<puzzle[0].size();j++){

            v.push\_back(puzzle[i][j]);

        }

    }

    int invcnt=0;

    for(int i=0;i<8;i++){

        for(int j=i+1;j<9;j++){

            if(v[i] && v[j] && v[i]>v[j]){

                invcnt++;

            }

        }

    }

    return (invcnt%2)==0;

}

Node \*createNode(Node\* parent,int x,int y){

    Node\* node=new Node();

    vector<vector<int>>temp=parent->state;

    node->x=x;node->y=y;

    swap(temp[x][y],temp[parent->x][parent->y]);

    node->state=temp;

    node->parent=parent;

    node->g=parent->g+1;

    node->cost=manhattenDistance(temp)+node->g;

    return node;

}

bool valid(int x,int y,int n){

    if(x>=0 && y>=0 && x<n && y<n){

        return true;

    }

    return false;

}

bool sameState(vector<vector<int>>&a,vector<vector<int>>&b){

    for(int i=0;i<3;i++){

        for(int j=0;j<3;j++){

            if(a[i][j]!=b[i][j]){

                return false;

            }

        }

    }

    return true;

}

void backtrackSolution(Node\* node){

    vector<vector<vector<int>>>v;

    while(node){

        v.push\_back(node->state);

        node=node->parent;

    }

    reverse(v.begin(),v.end());

    for(int i=0;i<v.size();i++){

        printVectorOfVector(v[i]);

        if(i!=v.size()-1){

            cout<<"Next->"<<endl;

        }

    }

    return;

}

vector<Node\*> neighbours(Node\* node){

    vector<int>X={1,-1,0,0};

    vector<int>Y={0,0,1,-1};

    vector<Node\*>res;

    for(int i=0;i<4;i++){

        int newX=node->x+X[i];

        int newY=node->y+Y[i];

        if(valid(newX,newY,3)){

            Node\* temp=createNode(node,newX,newY);

            if(node->parent && sameState(temp->state,node->parent->state)){

                delete temp; continue;

            }

            res.push\_back(temp);

        }

    }

    return res;

}

struct comp{

    bool operator()(const Node\* a,const Node\* b)const{

        return a->cost>b->cost;

    }

};

pair<Node\*,int> RBFSPuzzle(Node\* node,int flimit){

    if(goal==node->state){

        backtrackSolution(node);

        cout<<"Found solution"<<endl;

        return {node,flimit};

    }

    priority\_queue<Node\*,vector<Node\*>,comp>successors;

    for(auto &x : neighbours(node)){

        x->cost=max(x->cost,node->cost);

        successors.push(x);

    }

    if(successors.empty()){

        return {NULL,INT\_MAX};

    }

    while(true){

        if(successors.empty()){

            return {NULL,INT\_MAX};

        }

        Node\* best=successors.top();

        int bestf=best->cost;

        successors.pop();

*// cout<<"current state"<<endl;*

*// printVectorOfVector(best->state);*

        if(bestf>flimit){

            return {NULL,bestf};

        }

        int alternativeCost = successors.empty() ? INT\_MAX : successors.top()->cost;

        auto result=RBFSPuzzle(best,alternativeCost);

        if(result.first!=NULL){

            return result;

        }

        best->cost=result.second;

        successors.push(best);

    }

    return {NULL,flimit};

}

int main(){

    ios\_base::sync\_with\_stdio(false);

    cin.tie(0);

    auto t1=high\_resolution\_clock::now();

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

    if(isSolvable(p)){

        Node\* node=createRoot(p,2,2);

        RBFSPuzzle(node,INT\_MAX);

    }

    else{

        printVectorOfVector(p);

        cout<<"IMPOSSIBLE TO SOLVE";

    }

    auto t2=high\_resolution\_clock::now();

    duration<double,std::milli>mil\_double=t2-t1;

    double timeTaken=mil\_double.count();

    cout<<"Time Taken for RBFS\* : "<<timeTaken<<" milliseconds"<<endl;

    return 0;

}