8 Puzzle and Breadth-First search

CSE-0408 Summer 2021

Shubho Chandra Shil
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
Shubho.sarker12@gmail.com

***Abstract*—Main theme of your assignment or academic projects.**

n

***Index Terms*—8 Puzzle problem in Python .**

I. Introduction

The 8-puzzle problem is a puzzle invented and popularizedby Noyes Palmer Chapman in the 1870s. It is played on a3-by-3 grid with 8 square blocks labeled 1 through 8 and ablank square.

II. Literature Review

.

III. Rules of solving puzzle

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.

The empty space can only move in four directions (Movement of empty space)

Up Down Right or Left The empty space cannot move diagonally and can take only one step at a time.

Code

```
#include<bits/stdc++.h>
using namespace std;
#define D(x) cerr<<__LINE__<<" : "<<#x<<" -> "<<x<<endl
#define rep(i,j) for(int i = 0; i < 3; i++) for(int j=0;j<3;j++)
#define PII pair < int , int >
typedef vector<vector<int>> vec2D;

const int MAX = 1e5+7;
int t=1, n, m, l, k, tc;

int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0, 0};

vec2D init{
    {8, 1, 2},
    {3, 6, 4},
    {0, 7, 5}
};

vec2D goal{
    {1, 3, 2},
    {8, 0, 4},
    {7, 6, 5}
};
/// using a structure to store information of each
struct Box {
    vec2D mat{ { 0,0,0 },{ 0,0,0},{ 0,0,0} };
    int diff, level;
    int x, y;
    int lastx, lasty;
    Box(vec2D a,int b = 0, int c = 0, PII p = {0,0
        rep(i,j) mat[i][j] = a[i][j];
        diff = b;
        level = c;
        x = p.first;
        y = p.second;
        lastx = q.first;
        lasty = q.second;
    }
};

/// operator overload for which bases priority que
bool operator < (Box A, Box        {
    if(A.diff == B.diff) return A.level < B.level;
    return A.diff < B.diff;
}
int heuristic_function(vec2D a, vec2D b) {
    int ret(0);
    rep(i,j) if (a[i][j] != b[i][j]) ret--;
    return ret;
}

/// checking puzzle boudaries
bool check(int i, int j) {
    return i>=0 and i<3 and j>=0 and j<3;
}

/// this function used to show state status
void print(Box a) {
    rep(i,j)
```

```
    cout << a.mat[i][j] << (j == 2 ? "\n" : " ")
    cout << " heuristic Value is :
" << -a.diff << "\n";
    cout << " Current level is : " << -a.level <<
}

/// used to get new state which can be jump from
Box get_new_state(Box now, int xx, int yy) {
    Box temp = now;
    swap(temp.mat[temp.x][temp.y], temp.mat[xx][yy]);
    temp.diff = heuristic_function(temp.mat, goal);
    temp.level = now.level - 1;
    temp.x = xx;
    temp.y = yy;
    temp.lastx = now.x;
    temp.lasty = now.y;
    return temp;
}


/// this is modified version of dijkstra shortest path algorithms
/// basically work on those state first which heuristic value lesser
void dijkstra(int x, int y) {
    map < vec2D,  bool > mp;
    priority_queue < Box > PQ;
    int nD = heuristic_function(init, goal);
    Box src = {init, nD, 0, {x,y}, {-1,-1}};
    PQ.push(src);
    int state = 0;
    while(!PQ.empty()) {
        state++;
        Box now = PQ.top();
        PQ.pop();
        cout << "Step no : " << state-1 <<"\n";
        print(now);
        if(!now.diff) { /// if heuristic value is zero it means we are on goal
            puts("Goal state has been discovered");
            cout << "level : " << -now.level << "\n";
            cout << " Step no : " << state-1 <<"\n";
            break;
        }
        if(mp[now.mat]) continue;
        mp[now.mat] = true;
        for(int i = 0; i < 4; i++) {
            int xx = now.x + dx[i];
            int yy = now.y + dy[i];
            if(check(xx, yy)) {
                if(now.lastx == xx and now.lasty == yy) continue;
                Box temp = get_new_state(now, xx, yy);
                PQ.push(temp);
            }
        }
    }
}

signed main() {
```

```
puts("Current State:");
    rep(i,j) cout << init[i][j] << (j == 2 ? "\n"
    puts("");
    puts("Goal State:");
    rep(i,j) cout << goal[i][j] << (j == 2 ? "\n"
    puts("\n.....Search Started.....
    rep(i,j) if(-1 != init[i][j]) dijkstra(i,j); /// th
    return 0;
}
```


ACKNOWLEDGMENT

I would like to thank my honourable **Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

***Abstract*—Main theme of your assignment or academic projects.**

n

***Index Terms*—Breadth-first search in python.**

I. Introduction

Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

II. Literature Review

III. Proposed Methodology

Breadth-first search (BFS) is an important graph search algorithm that is used to solve many problems including finding the shortest path in a graph and solving puzzle games.

Code

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

#define MX 110

vector < int > graph[MX];
bool vis[MX];
int dist[MX];
int parent[MX];

void bfs(int source){
    queue < int > Q;
    // initialization
    vis[source] = 1;
    dist[source] = 0;
    Q.push(source);

    while(!Q.empty()){
        int node = Q.front();
        Q.pop();

        for (int i = 0; i < graph[node].size(); i++){
            int next = graph[node][i];
            if (vis[next] == 0){
                vis[next] = 1; // visit
                dist[next] = dist[node] + 1;
                Q.push(next); // push to queue

                // set parent
                parent[next] = node;
            }
        }
    }
}

/*
    input:
    7 9
    1 2
    1 3
    1 7
    2 3
    3 7
    2 4
    4 5
    3 6
    5 6
    1
*/

// path printing functions

// recursive function
void printPathRecursive(int source, int node){
    if (node == source){
        cout << node << " "; // print from source
        return;
    }
    printPathRecursive(source, parent[node]);
    cout << node << " ";
}

// iterative function
void printPathIterative(int source, int node){
    vector<int> path_vector;

    while(node != source){
        path_vector.push_back(node);
        node = parent[node];
    }
    path_vector.push_back(source); // inserting so

    for (int i = path_vector.size() - 1; i >= 0; i
}

// main()
int main(){

    int nodes, edges;
    cin >> nodes >> edges;

    for (int i = 1; i <= edges; i++){
        int u, v;
```

```
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    int source;
    cin >> source;

    bfs(source);

    cout << "From node " << source << endl;
    for (int i = 1; i <= nodes; i++){
        cout << "Distance of " << i << " is : " << dist[i] << endl;
    }
    cout << endl;

    // path printing example

    // recursive version
    for (int i = 1; i <= nodes; i++){
        cout << "Path from " << i << " to source: ";
        printPathRecursive(source, i);
        cout << endl;
    }

    cout << endl;

    // iterative version
    for (int i = 1; i <= nodes; i++){
        cout << "Path from " << i << " to source: ";
        printPathIterative(source, i);
        cout << endl;
    }

    return 0;
}
```