Mid :- Assignment Report

CSE-0408 Summer 2021

Name:Rafi Ahmed ID:UG02-47-18-030
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
Email Address: rafiahmed362@gmail.com

***Abstract*—This paper introduced for Breadth-First Search(BFS) problem and 8 puzzle problem by heuristic function using C++ language.**
n

***Index Terms*—Languages: C++.**

I. Introduction

Heuristic Function

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order.

Breadth-First Search

Breadth-first searchstarts at a given vertex s, which is at level 0. In the first stage, we visit all the vertices that are at the distance of one edge away. When we visit there, we paint as ”visited,” the vertices adjacent to the start vertex s - these vertices are placed into level 1.

II. Literature Review

Heuristic Function

In 2012 S. et. al. [4] proposed new resolution for solving N-queens by using combination of DFS (Depth First Search) and BFS (Breadth First Search) techniques. The proposed algorithm act based on placing queens on chess board directly. The results report the performance and run time of this approach.

Breadth-First Search

The two variants of Best First Search are Greedy Best First Search and A* Best First Search. Greedy BFS: Algorithm selects the path which appears to be the best, it can be known as the combination of depth-first search and breadthfirst search. Greedy BFS makes use of Heuristic function

III. Proposed Methodology

Here i Discuss BFS Algorithm:

```
1. for each u in V s
2. do color[u] ← WHITE
3. d[u] ← infinity
4. [u] ← NIL
5. color[s] ← GRAY
6. d[s] ← 0
7. [s] ← NIL
8. Q ←
9. ENQUEUE(Q, s)
10 while Q is non-empty.
```

IV. RULES FOR SOLVING THE PUZZLE

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile, basically swapping the tile with the empty space. The empty space can only move in four directions viz.

1. Up
2.Down
3. Right or
4. Left

The empty space cannot move diagonally and can take only one step at a time (i.e. move the empty space one position at a time)

V. ALGORITHM FOR BFS

Step 1: Choose the starting node and insert it into queue

Step 2: Find the vertices that have direct edges with the vertex(node)

Step 3: Insert all the vertices found in step 3 into
queue Step 4: Remove the first vertex(node) in queue

Step 5: Continue this process until all the vertices are visited

VI. CONCLUSION AND FUTURE WORK

The BFS algorithm is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these

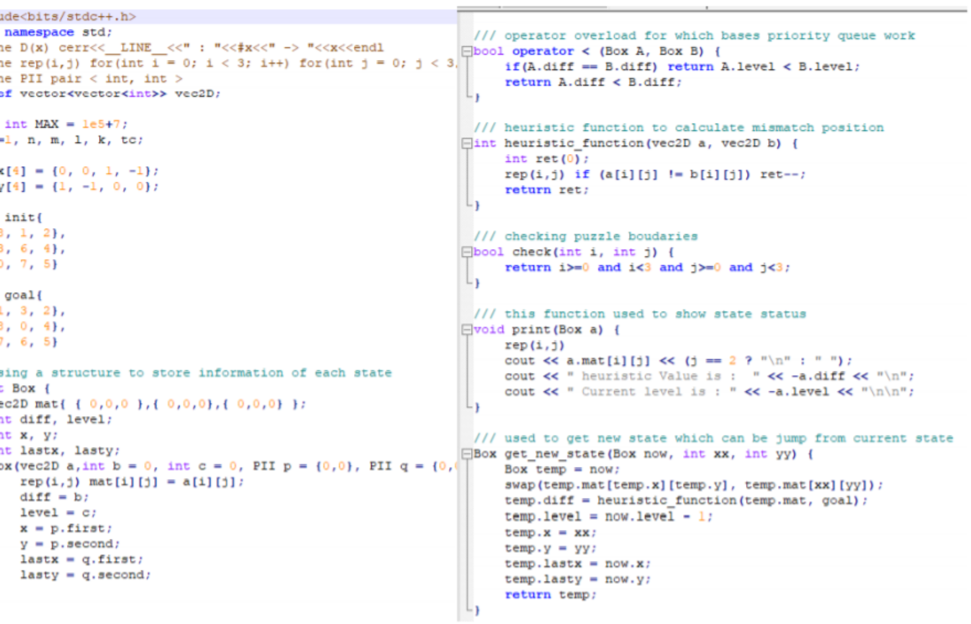
Acknowledgment

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

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.



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

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

x[4] = {0, 0, 1, -1};
y[4] = {1, -1, 0, 0};

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

 goal{
1, 3, 2},
3, 0, 4},
7, 6, 5}

sing a structure to store information of each state
t Box {
ec2D mat{ { 0,0,0 },{ 0,0,0},{ 0,0,0} };
nt diff, level;
nt x, y;
nt lastx, lasty;
ox(vec2D a,int b = 0, int c = 0, PII p = {0,0}, PII q = {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 queue work
bool operator < (Box A, Box B) {
    if(A.diff == B.diff) return A.level < B.level;
    return A.diff < B.diff;
}

/// heuristic function to calculate mismatch position
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 << "\n\n";
}

/// used to get new state which can be jump from current state
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;
}
```

Fig. 1. code.