Assignment Report:02
8 puzzel problem And BFS Algorithm problem solving

Yeasmin Akter
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
yasminepsha@gmail.com

I. Assignment Name:

8 Puzzel Problem Solving

***Abstract*—A Heuristics is a Function that estimates how close a state to a goal state. The heuristics or heuristics function differs from problem to problem and usually need to be designed to fit a certain problem. Most of the artificial intelligence problem can be described as follows: given a huge amount of information, data, and some constraints, tell me how to reach our goal state. In this case, the heuristics function tells us how close we are to the goal state.**

***Index Terms*—Index Terms—About Heuristic Function is the 8-puzzle problem in C++**

II. Introduction

Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

III. II. LITERATURE REVIEW

Heuristics in human decision-making was developed in the 1970s and the 1980s.The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time. Heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve their efficiency.Results about hardness in theoretical computer science make heuristics the only viable option for a variety of complex optimization problems that need to be routinely solved in real-world applications.

IV. PROPOSED METHODOLOGY

Current State[[8,1,2],[3,6,4],[0,7,5]] Goal State [[1,2,3][8,0,4],[7,6,5]]

V. . RESULT AND SOLVING PROBLEM

In this paper, i applied pattern classification techniques to solve a fundamental open problem in computer science that relates heuristic function accuracy and solution optimality. More specifically,in this paper, we have discussed the efficiency of using heuristic functions for optimization problems and resolved an open problem. The problem involves how the accuracy of a heuristic function relatesto the quality of the corresponding solution obtained. The efficiency has been quantified by means of the probability of the heuristic function leading to the optimal solution. It is represented by h(n).The value of this function is always positive. The heuristics function is given as:h(n)!=h*(n)
1.up
2.Down.
3.Left.
4.Righ

VI. Main 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 ;
//vec2D init // 1, 2, 3, // 8, 6, 0, // 7, 5, 4 //;
//vec2D goal // 1, 2, 3, // 8, 0, 4, // 7, 6, 5 //;
//vec2D init // 1, 3, 2, // 4, 0, 7, // 6, 5, 8 //;
```

```
//vec2D goal // 0, 2, 4, // 1, 3, 8, // 6, 5, 7 //;
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, PII q = 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;
;
bool operator ¡ (Box A, Box B)
if(A.diff == B.diff) return A.level ¡ B.level;
return A.diff ¡ B.diff;
int isEqual(vec2D a, vec2D b)
int ret(0);
rep(i,j) if (a[i][j] != b[i][j]) ret–;
return ret;
bool check(int i, int j)
return i¿=0 and i¡3 and j¿=0 and j¡3;
void print(Box a)
rep(i,j)
cout ¡¡ a.mat[i][j] ¡¡ (j == 2 ? ”” : ” ”);
D(-a.diff);
D(-a.level);
cout ¡¡ ”(” ¡¡ a.x ¡¡ ”,” ¡¡ a.y ¡¡”)”;
void dijkstra(int x, int y)
map ¡ vec2D, bool ¿ mp;
priority$_{q}ueue < Box > PQ$;
int nD = isEqual(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();
print(now);
if(!now.diff)
puts(”Goal state has been discovered”);
cout ¡¡ ”level : ” ¡¡ -now.level ¡¡ ””; D(state);
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 = now;
swap(temp.mat[temp.x][temp.y], temp.mat[xx][yy]);
temp.diff = isEqual(temp.mat, goal);
temp.level = now.level - 1;
temp.x = xx;
temp.y = yy;
temp.lastx = now.x;
temp.lasty = now.y;
PQ.push(temp);
signed main()
puts(”Current State:”);
rep(i,j) cout ¡¡ init[i][j] ¡¡ (j == 2 ? ”” : ” ”);
puts(””);
puts(”Goal State:”);
rep(i,j) cout ¡¡ goal[i][j] ¡¡ (j == 2 ? ”” : ” ”);
puts(”............Search Started..............”);
rep(i,j) if(!init[i][j]) dijkstra(i,j); return 0;
```

VII. Assignment Name:

8 Puzzel Problem Solving

***Abstract*—BFS is one of the classical graph theory algorithms, typically expressed under the imperative style. However, it has applications in other domains such as artificial intelligence. All known algorithms of BFS use iteration.**

This article focuses on the information maintained by BFS during exploration of an arbitrary graph component. To better understand the structure of this information, BFS is re-defined both recursively and iteratively. Also provided in this article is an analysis and comparison of the various BFS algorithms presented.

***Index Terms*—BFS,Code Blocks,C++**

VIII. Introduction

Breadth-first search (BFS) is an algorithm that is used to graph data or searching tree or traversing structures. The full form of BFS is the Breadth-first search.

The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion. This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. Remember, BFS accesses these nodes one by one.

Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them. Once visited, all nodes are marked. These iterations continue until all the nodes of the graph have been successfully visited and marked.

IX. LITERATURE REVIEW

BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalk¨ul programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze, and later developed by C. Y. Lee into a wire routing algorithm (published 1961). In 2012 Farhad S. et. al. proposed new resolution for solving N-queens by using combination of DFS (Depth First Search) and BFS (Breadth First Search) techniques.

X. PROPOSED METHODOLOGY

```
1. for each u in V s
2. do color[u] $\leftarrow$ W HIT E
3.d[u] $\leftarrow$ inf inity
4.[u] $\leftarrow$ NIL
5.color[s] $\leftarrow$ GRAY
6.d[s] $\leftarrow$ 0 7.[s] $\leftarrow$ NIL
8.Q $\leftarrow$ 9.ENQUEUE(Q, s)
10.whileQisnon  empty
11.dou $\leftarrow$ DEQUEUE(Q)
12.foreachvadjacenttou
13.doif color[v] $\leftarrow$ W HIT E
14.thencolor[v] $\leftarrow$ GRAY
15.d[v] $\leftarrow$ d[u] + 1
16.[v] $\leftarrow$ u
17.ENQUEUE(Q, v)
18.DEQUEUE(Q)
19.color[u] $\leftarrow$ BLACK
```

XI. Rules of BFS Algorithm

A standard BFS implementation puts each vertex of the graph into one of two categories:

1.Visited
2.Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.The algorithm works as follows:

*Start by putting any one of the graph's vertices at the back of a queue.

*Take the front item of the queue and add it to the visited list.

*Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.

*Keep repeating steps 2 and 3 until the queue is empty.

*The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

XII. Advantage of BFS Algorithm

There are numerous reasons to utilize the BFS Algorithm to use as searching for your dataset. Some of the most vital aspects that make this algorithm your first choice are:

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

*BFS can traverse through a graph in the smallest number of iterations.

*The architecture of the BFS algorithm is simple and robust.

*The result of the BFS algorithm holds a high level of accuracy in comparison to other algorithms.

*BFS iterations are seamless, and there is no possibility of this algorithm getting caught up in an infinite loop problem.

2021-07-15.png

Fig. 1. Output1

XIII. Summary

A graph traversal is a unique process that requires the algorithm to visit, check, and/or update every single un-visited node in a tree-like structure. BFS algorithm works on a similar principle.The algorithm is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.The algorithm traverses the graph in the smallest number of iterations and the shortest possible time.BFS selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. BFS accesses these nodes one by one.The visited and marked data is placed in a queue by BFS. A queue works on a first in first out basis. Hence, the element placed in the graph first is deleted first and printed as a result.The BFS algorithm can never get caught in an infinite loop.Due to high precision and robust implementation, BFS is used in multiple real-life solutions like P2P networks, Web Crawlers, and Network Broadcasting.

XIV. Conclusion

Breadth Search Algorithm comes with some great advantages to recommend it. One of the many applications of the BFS algorithm is to calculate the shortest path. It is also used in networking to find neighbouring nodes and can be found in social networking sites, network broadcasting, and garbage collection. The users need to understand the requirement and the data pattern to use it for better performance.

XV. Main Code

```
include
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; // update
```

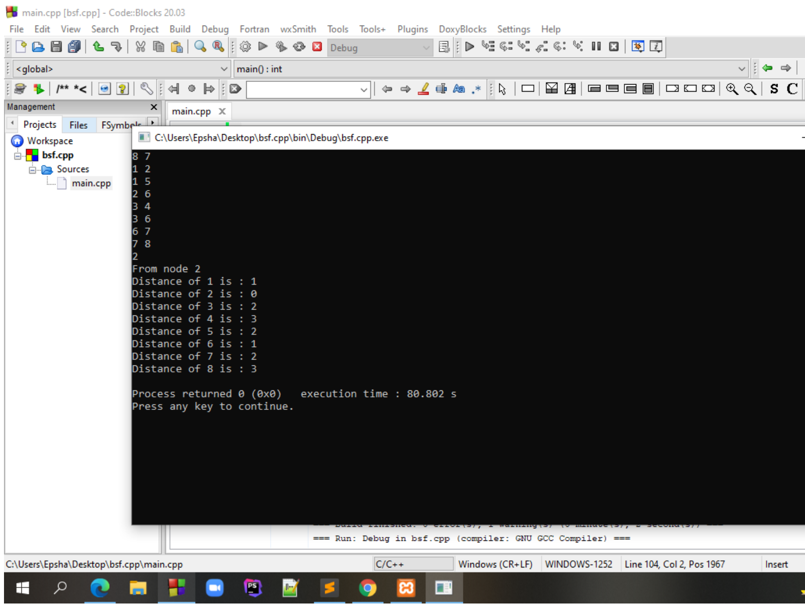
Q.push(next); // push to queue
// set parent parent[next] = node;
/*
input: 8 7
1 5
1 2
2 6
3 6
3 4
6 7
7 8
2
*/ // recursive function
void printPathRecursive(int source, int node)
if (node == source)
cout ¡¡ node ¡¡ ” ”; // print from source return;
printPathRecursive(source, parent[node]); cout ¡¡ node ¡¡ ” ”;
// iterative function
void printPathIterative(int source, int node)
vector path$_vector;$
while(node != source)
path$_vector.push_back(node);$
node = parent[node];
path$_vector.push_back(source); //insertingsource$
for (int i = path$_vector.size() - 1; i >= 0; i--)$
cout ¡¡ path$_vector[i] << ””;$
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;
return 0;

ACKNOWLEDGMENT

I would like to express my gratefulness and gratitude to all those people who gave me the inspiration to complete this proposal. A special thanks to my instructor **Khan Md. Hasib Sir**, Lecturer, department of Computer Science and Engineering (CSE), whose help, stimulating suggestions, corrections and encouragement gave me the opportunity to coordinate information in writing this research proposal.

REFERENCES

[1] thebibliography


main.cpp [bsf.cpp] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Management
Projects Files FSymbols
Workspace
bsf.cpp
Sources
main.cpp
C:\Users\Epsha\Desktop\bsf.cpp\main.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 104, Col 2, Pos 1967
Insert

C:\Users\Epsha\Desktop\bsf.cpp\bin\Debug\bsf.cpp.exe

```
8 7
1 2
1 5
2 6
3 4
3 6
6 7
7 8
2
From node 2
Distance of 1 is : 1
Distance of 2 is : 0
Distance of 3 is : 2
Distance of 4 is : 3
Distance of 5 is : 2
Distance of 6 is : 1
Distance of 7 is : 2
Distance of 8 is : 3

Process returned 0 (0x0)   execution time : 80.802 s
Press any key to continue.
```

```
=== Run: Debug in bsf.cpp (compiler: GNU GCC Compiler) ===
```

Fig. 2. Output