CmpE480 Project1 Report

Barın Özmen - 2012400045

November 4, 2018

1 Introduction

This program is written in python2 language and you can simply run it from terminal with giving two parameters; first one for python file and second one for input file. Example:

> *python search.py data/test/labyrinth/labyrinth_test_1.txt*

You should see an output like this on the terminal:

> *Output files are created under search.py path.*

After that output, files are created on that specific folder path which has the same path as search.py

2 Algorithm

The code consists of 5 methods which 3 different algorithms to solve the labyrinth, 1 for reading input and 1 for main. readInput() method takes parameter which is going to be input txt file and parse it into coordinates and map2D array to keep track of the map. Also this method creates discoveredMatrix array which has the same size as map2D array and all of its elements are 0 due to not starting to discover yet.

ImplementDFS() and ImplementBFS() methods are working in almost in same ways except, BFS has list as queue and DFS has list as stack. The main part of these algorithm works as first checking the boundaries, if currentX or currentY on the map can go further. After that, if structure created as ascending and descending order as in the assignment pdf. First look if we can travel around the map checking if there is a wall(in labyrinth cases) and if height is bigger than 1(in mountain cases). If not, algorithm adds these coordinates into stack or queue according to DFS or BFS. While doing it, it also add these nodes to dictComeFrom which helps the algorithm to find its way back starting point when it reaches to the end. When additions are completed, in both DFS and BFS, it is time to pick a new currentX and currentY to be able to travel in the map. Take the first element from the stack or queue and pop this element and make it new currentX and currentY and add it to discoveredMatrix as well. When it reaches the end points, the algorithm exit on while loop and from the backtraceStack which has the same name as in BFS(variable name is not important at that point because it does the same thing as DFS; i copied and paste

and actually must be named as backtraceQueue), it keeps the shortest path for the map travel. Than it writes what it recorded to the file named as inputfile + "_bfs_out.txt" or "_dfs_out.txt".

implementAStar() method works little bit different than those two. In this algorithm, instead of using stack or queue, i used 3 dictionaries for keeping g(n), h(n) and f(n) values(f(n) = g(n) + h(n)) for traveling map. On deciding which is going to be new currentX and new currentY, f(n) values are compared. Also there is a difference on exploring locations which is the algorithm checks g(n) values of the nodes and if it already explored compare old g(n) value(s) with the new g(n) value and if the new value is smaller than old one(s), algorithm updates the g(n), h(n), f(n) and dictComeFrom. After exploring part, algorithm decides on new currentX and new currentY according to f(n) values. It takes the lowest f(n) value as new nodes. Rest is the same as implementDFS() and implementBFS() part while writing on a file.

3 Results

In this part, output files explained in detail for the test examples in which algorithm finds optimal paths or which algorithm finds a path with discovering less nodes.

3.1 labyrinth_test_1.txt

DFS algorithm: Finds exit with skipping alternative roads with 26 steps for shortest path.

BFS algorithm: Finds a path with length 18, yet it discovered almost all the map to find it.

A* algorihm: Again it finds exit with path length 18 and it discovered few nodes to get the finish.

3.2 labyrinth_test_2.txt

DFS algorithm: Finds exit with skipping alternative roads with 36 path long

BFS algorithm: Finds better path than DFS with 16 path long. Yet it discovered almost all the nodes in the map

A* algorihm: Finds it way out in 16 steps again with less discovered node than BFS

3.3 labyrinth_test_3.txt

DFS algorithm: Finds exit with skipping alternative roads with 40 path long

BFS algorithm: Finds better path than DFS with 20 path long and more discovered node

A* algorihm: Finds it way out in 20 steps again with little less discovered node than BFS

3.4 labyrinth_test_4.txt

DFS algorithm: It founds it way out with 45 path length.

BFS algorithm: Finds best path as length 19 yet with too much discovered node.

A* algorihm: Same as BFS length for best path with fewer discovered node.

3.5 labyrinth_test_5.txt

DFS algorithm: Shortest path with 38 path long.

BFS algorithm: Shortest path with 26 path long with more discovered node.

A* algorihm: Same shortest path long with BFS with less discovered node

3.6 mountain_test_1.txt

DFS algorithm: shortest path length 30 with Euclidean distance of 34.54

BFS algorithm: shortest path length 24 with Euclidean distance of 26.82

A* algorihm: shortest path length 24 with Euclidean distance of 26.31 and less discovered node than BFS algorithm

3.7 mountain_test_2.txt

DFS algorithm: shortest path length 24 with Euclidean distance of 26.82

BFS algorithm: shortest path length 24 with Euclidean distance of 26.82

A* algorihm: shortest path length 24 with Euclidean distance of 26.31 and less discovered node than BFS algorithm

3.8 mountain_test_3.txt

DFS algorithm: shortest path length 20 with Euclidean distance of 23.42

BFS algorithm: shortest path length 14 with Euclidean distance of 15.88

A* algorihm: shortest path length 14 with Euclidean distance of 15.88 and less discovered node than BFS algorithm

3.9 mountain_test_4.txt

DFS algorithm: shortest path length 23 with Euclidean distance of 24.76

BFS algorithm: shortest path length 9 with Euclidean distance of 9.02

A* algorihm: shortest path length 9 with Euclidean distance of 9.02 and lot smaller discovered node than BFS algorithm

3.10 mountain_test_5.txt

DFS algorithm: shortest path length 19 with Euclidean distance of 21.92

BFS algorithm: shortest path length 19 with Euclidean distance of 21.92

A* algorihm: shortest path length 19 with Euclidean distance of 20.47 and lot smaller discovered node than BFS algorithm

4 Conclusion and Discussion

All three algorithm differs in each other while finding the shortest path. DFS algorithm usually finds the exit with the smallest number of operation among 3 of them. Yet, it misses other paths so that it cannot find the shortest path in all these examples compared with BFS and A*. Smallest operation number, smallest discovered map, largest shortest path length compared these 3 algorithms.

BFS algorithm finds shortest path same as A* in labyrinth examples yet with too much discovered nodes. This algorithm discover almost every part of the map. In mountain examples, even if BFS algorithm discover too much area, it couldn't find the shortest path compared with A* with small difference. Largest operation number, largest discovered map, almost same as A* in finding shortest path with little difference. BFS is better than DFS in finding shortest path, yet A* is the number one in finding shortest path compared to these 2 algorithms.

A* algorithm finds the best shortest path among those 3 algorithms. Yet, when we look from the aspect of discovered nodes, this algorithm is better than BFS and worse than DFS algorithm.

To sum up, A* algorithm seems much more better than BFS and DFS algorithms in case of finding the shortest path with fewer discovered map.