Depth-First VS Breadth-First Search in labyrinths

Sárközi Viktor
Computer Science Bsc
Gyöngyöshalász, Hungary
vikusz00275@gmail.com

Oravecz Zsolt
Computer Science Bsc
Salgótarján, Hungary
zsolti1012@gmail.com

I. ABSTRACT

This document compare to Deep-first and Breadth-first searching algorithms in labyrinths. The program language that we are choose to implements and testing is c#. The two algorithms are structurally very similar, but in the comparison we took into account the running time of the algorithm as well as which search returns the shortest (optimal) path after solving the maze. In most cases, the most significant differences are presented in the comparison, usually supported by pictures.

II. KEYWORDS

- BFS = (Breadth-First-Search)
- DFS = (Depth-First-Search)
- FIFO = (First in, first out)(Stack structure)
- LIFO = (Last in, first out) (Queue structure)

III. INTRODUCTION

This documentation primarily represents the differences and similarities between two search algorithms as evidenced by the tests performed. To solve the problem, similar comparisons have already been made almost certainly in, for example, a tree data structure, but in this comparison, testing focuses on a maze, as any differences in the maze can be excellently presented. The documentation includes links between the sections for better transparency. We hope this documentation will help people who's interested in this topic.

IV. METHOD

As the introduction read, we limited the problem to a maze. First, to illustrate the problem, we used a maze drawing program that can even draw the maze that we created. Within this program, we implemented the two different algorithms in a separate class and their interaction provides the right environment for the tests. However, the test cases should, of course, be different in order to illustrate a better result.

These cases are:

- Testing in larger mazes (50x50).
- Use a randomly generated maze.
- Will recognize the shortest path?

A. Other facilities

It is also worth analyzing the disadvantages of the different options for more effective implementation.

1) Testing in larger mazes (50x50).

   In the case of larger labyrinths, the number of paths to the destination can increase significantly, and at the same time the running time can even double (this is no problem for the test). (If the program does not fit on the screen, it will still run, but we may not see the target on the screen, which in this case would hinder the output of the research.

2) Use a randomly generated maze. [2]

   In the case of a randomly generated maze, the number of possible paths clearly increases, as in the case of larger labyrinths. It is worth providing our program for this case as well, which can be an important milestone in the comparison if we also take into account the time of recognizing the unsolvable problem in the comparison.

3) Will recognize the shortest path?

   We need to implement the recognition of the shortest path in all respects as it is very important for the test results that which algorithm recognizes and how fast. The most spectacular implementation of this is if the algorithm draws the shortest path with a different color after processing the maze.

B. Show results

After these theoretical suggestions, we obviously also need to talk about the method of performing the tests. For the sake of better transparency, the results shown by our testing environment are summarized in tables.

Finally, before looking at the results obtained, it is worth mentioning the operation of the two algorithms in general.

C. Working of the BFS and DFS

- DFS (Depth-First Search)

  It is essentially an intuitive algorithm that randomly selects a route after it starts at an intersection. If the path leads to a dead end, the algorithm returns to the intersection and choose another path. It follows that the algorithm explores the entire maze. The advantage of this algorithm is that it always finds its way. However, it also has the disadvantage that it does not necessarily find the optimal path, which is a significant problem in terms of time. More information about the DFS available in : [2]

- BFS (Breadth-First Search)

  An latitude search is an algorithm used to traverse or search a tree or graph data structure. It is used at the root of the tree (or at any node in a graph, sometimes called a ”search key”) and explores all adjacent nodes at the current depth before moving on to the next nodes.You can found more information about the BFS in: [1]

V. Results

Our first more important result, which is essential for the comparison of algorithms, is the way of realizing the maze itself, which we have already outlined in the previous sections, we now represent the result with images.

A. Make own labyrinth

The character ”X” indicates the wall, while the space bar indicates the free path. Indicated by the ”S” character the starting point and the character ”C” the goal you want to reach. The road network is stored in a two-dimensional array.

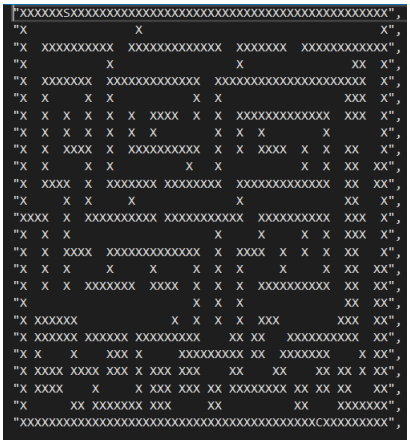

Fig. 1: Own labyrinth

In the case of latitude traversal, we pass our array to a cycle we define, with which the wall marking is given the number 0 and the possible path the number 1.

B. Random maze

Considering the problems encountered earlier in the generated maze, we get a generated maze that meets our expectations to perform our test. Where the size of the maze can be changed manually.

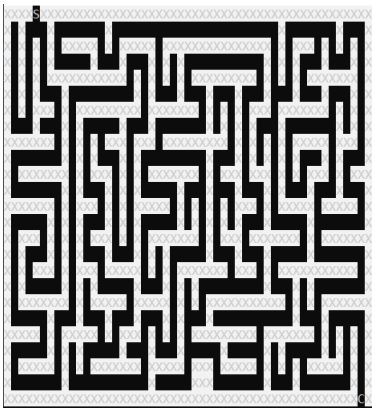

Fig. 2: Random generated labyrinth

Now our maze is given, we only need to implement the BFS(Breath-First Search) and DFS(Depth-First Search) algorithms for the maze. The images show that the paths traversed by the algorithm are illustrated with different colors. Obviously, the green color indicates the way we can get out of the maze, by the way, the other color indicates the tried and tested routes. [5]

C. Make statistic

Compiling the statistics is an important part of the income statement, as it provides information about possible differences between the two algorithms using different test cases at the same time. The implementation of the statistics itself can be done in any programming language, but in this case we kept the C# programming language, because the other implementations were written in this language as well. The following image shows the more important lines of code for check runtime.

```
1  using System.Diagnostics;
2          .....
3      Stopwatch sw = new Stopwatch();
4                  sw.Start();
5                  BFS.solve(maze2, start.Y, start.X,  destination
                         .Y,  destination .X);
6                  sw.Stop();
7                  BFS.print(maze2);
8                    .......
9                  Console.WriteLine(sw.Elapsed);
10                 ....
```

This snippet only helps us to display the runtime of a particular method. The values obtained are shown in the results section.

D. Implements the algorithms: [3]

- BFS

  Some explanations to understand the picture:

  - In red we can see the paths traversed by the algorithm (It can be seen that it only maps the maze until it finds the target.)
  - The green color illustrates the path to the target. (Not the optimal one).
  - The blue color illustrates the shortest route to the desired point, which is also the optimal route.

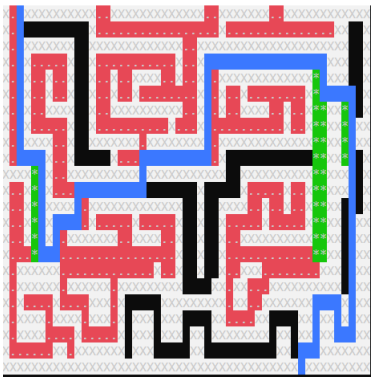

Fig. 3: Solve with BFS

Implementing a BFS algorithm in a maze promises to be a bit more complicated than the graphical implementation known to many, because during the implementation the objects in the maze had to be included in the algorithm. Fortunately, we managed to overcome the obstacles and here is a detail of the working implementation:

```
1  public static int BFS(int[,] Maze, Point src,Point dest)
2        {.....
3   while (q.Count != 0)
4      {
5      curr = q.Peek();
6      Point pt = curr.pt;
7    if (pt.x == dest.x && pt.y == dest.y)
8          return curr.dist;
9          q.Dequeue();
10              for (int i = 0; i < 4; i++) {
11              int row = pt.x + rowNum[i];
12              int col = pt.y + colNum[i];
13               ....}
14              queueNode Adjcell = new queueNode(new
                     Point(row, col), curr.dist + 1,curr);
15                   distance[row, col] = curr.dist+1 ;
16                  q.Enqueue(Adjcell);
17                   ....
18                       }
```

- DFS

Some explanations to understand the picture:

- In red we can see the paths traversed by the algorithm (It can be seen that it only maps the maze until it finds the target.)
- The green color illustrates the path to the target. (Not the optimal one).
- This algorithm cannot find the optimal route.

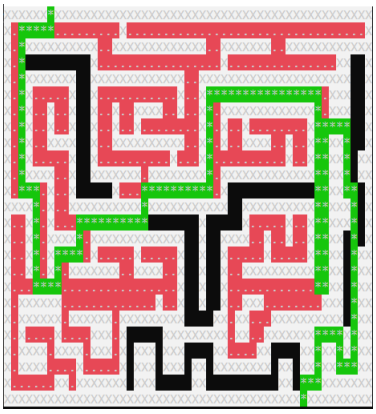

Fig. 4: Solve with DFS

The same problems were encountered in the implementation of this algorithm as in the case of BFS, but the significant difference can also be seen in the code of the algorithm itself. This is because, as described earlier, the two algorithms are said to be oriented differently within the maze. This difference is illustrated in the code snippet below.

```
1  static public bool DFS(string[,] Maze,int x,int y,
       int destx,int desty)
2      {
3          Maze[y, x] = Questionable;
4          bool found = false;
5          if (x == destx && y == desty)
6              found = true;
7     if (!found && x < SIZEX - 1 && Maze[y, x +
          1] == Route)
8              {     ....    }
9              if (!found && x > 0 && Maze[y, x -
                   1] == Route)
10             {
11    if (DFS(Maze, x - 1, y, destx, desty))
12     found = true;
13             }
14              {.....}
15    Maze[y, x] = found ? Exit : Deadend;
16             return found; }
```

E. The summary

Finally, knowing the facts, we can turn to discuss the differences and similarities between the two algorithms, which we can sort out a bit. [5]

1) Differences:

- One important difference between the two algorithms is finding the optimal route. The Depth-First-Search algorithm, does not guarantee the optimal path. Breath-First-Search algorithm always find the optimal way. [4]
- The BFS algorithm uses a queue data structure, while the DFS algorithm uses a stack data structure.
- BFS is implemented using FIFO list,but DFS is implemented using LIFO list.
- There is no need of backtracking in BFS, although need of backtracking in DFS.
- In terms of memory usage, BFS requires more memory to execute than the other algorithm. [6]

2) Similarities:

- Both algorithms only explore the maze until they find a route to the destination. If you find one, it will no longer map the rest of the course.
- Both have the similar efficiency (linear time).

TABLE I: The compare results in general [5]

	Solving in C#		
	Solving steps	***Optional way***	***Run time***
Depth-first search	depends on size	not guaranteed	less than BFS
Breath-first search	depends on size	guaranteed	more than DFS

The test cases listed in the tables can also be found in image format in previous chapters. (Own labyrinth, Random labyrinth)

The compare results in different size of the labyrinth

TABLE II: Breadth-First-Search

	Solving with Breadth-First-Search		
	Solving steps	***Optional way***	***Run time***
50x50 labyrinth	156	guaranteed	0:00:00.0016294
Own labyrinth [1]	109	guaranteed	0:00:00.0013517
Random labyrinth [2]	103	guaranteed	0:00:00.0012555

Our randomly generated maze in which we performed the test is a maze of 15 x 35 characters, the results of which are shown in the corresponding row of the table.

In the case of randomly generated mazes there was also a case when the two algorithms solved the given path in the same number of steps, the identity of the number of steps can be explained by the fact that the DFS algorithm does not guarantee the optimal path, but it may find the optimal path. is the same as BFS which always finds the optimal path.

TABLE III: Depth-First-Search

	Solving with Depth-First-Search		
	Solving steps	***Optional way***	***Run time***
50x50 labyrinth	194	not guaranteed	0:00:00.0015694
Own labyrinth [1]	129	not guaranteed	0:00:00.009157
Random labyrinth [2]	121	not guaranteed	0:00:00.0008352

F. Results in Random labyrinth

The measurements were performed in several random mazes, but in these cases we no longer averaged the time but the data measured in five randomly generated mazes, and we expanded both algorithms with an extra counter. For BFS, the number of recursive calls is summed, while for DFS, the number of operations is summed. Their average is shown in the table below. For more authentic results, random maze example can be found in appendix.

TABLE IV: Compare five random labyrinth

	Random labyrinth solving with BFS		
	Solving steps	***Recursive call***	***Queue operation***
maze1 [5]	103	not use	181
maze2	95	not use	219
maze3	79	not use	221
maze4	99	not use	149
maze5	87	not use	166
Average	**92,6**	**-**	**187,2**

It can be seen that for 5 random mazes, BFS is able to solve a 15x35 orbit in an average of 92 steps using about 187 rows of operations per maze.

TABLE V: Compare five random labyrinth

	Random labyrinth solving with DFS		
	Solving steps	***Recursive call***	***Queue operation***
maze1 [5]	121	233	not use
maze2	137	177	not use
maze3	149	215	not use
maze4	117	129	not use
maze5	87	228	not use
Average	**122,2**	**196,4**	**-**

Seeing these results, we can state that the solutions of the DFS algorithm are a bit more expensive for random mazes.

VI. Discussion

Reading the research results, it can be clearly stated that the research was successful as solutions to the problems discussed in the previous chapters were also found. Also, this comparison can help many to choose between the two algorithms, as we have sufficiently demonstrated which one is worth choosing to solve a labyrinthine problem. Although in the case of a graph problem it is no longer certain that the information we have provided is completely correct, we have made a comparison limited to the labyrinth.

REFERENCES

[1] BFS: Breadth-First-Search:
https://en.wikipedia.org/wiki/Breadth-firstsearch

[2] Keshav Sharma, Chirag Munshi A Comprehensive and Comparative Study Of Maze-Solving Techniques by Implementing Graph Theory. Ver. IV (Jan – Feb. 2015)

[3] Implement the labyrinth:
https://infoc.eet.bme.hu/labirintus/?fbclid=
IwAR32H9vM7rp5UBwJal5jkIqiLOxDmYgDFDuNj3PqJIQ2203L9PGCBGC-EyQ

[4] Shortest-path:
https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/fbclid=
IwAR2XW1g3YZ8Z1aYspv5XsKY7YPbbP2kXVHG9x0H3oYudypp7uZGDS8oxH-Y

[5] Compare in graph
https://www.guru99.com/difference-between-bfs-and-dfs.html?
fbclid=IwAR1LzO84YR8bSp8Qe-puh8RXFB4BIc_t8Hs6FiTjdjcar_
s--XJiJXnCuXE#5

[6] Columbia University, Similarities between bfs and dfs both procedures.

VII. APPENDIX

Random maze used for data in the results table:

- As can be seen in a maze executed simultaneously by both algorithms this explains the data shown in the image.

Fig. 5: maze1