International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/IJSRCSEIT

Development of Tool for Visualizing Pathfinding Algorithms

Mohak Mohan[1], Mehak Pargal[1], Dr. Simmi Dutta[2]
[1]Department of Computer Science & Engineering, GCET, Chak Bhalwal, Jammu, J&K, India
[2]HOD, Department of Computer Science & Engineering, GCET, Chak Bhalwal, Jammu, J&K, India

ABSTRACT

Article Info
Volume 7, Issue 4
Page Number : 439-454

Publication Issue :
July-August-2021

Article History
Accepted : 20 July 2021
Published : 30 July 2021

Dijkstra algorithm, A Search, Greedy Best-first Search, Swarm Search, Breadth-first and Depth-first are some of the popular algorithms today. As a beginner's step to algorithms and their implementation, this paper demonstrates how the pathfinding algorithm works.
Moreover, the user can get a better perception of how different algorithms function and how programming works, in general. By understanding these algorithms, they will also get a basic idea of how to implement various navigation tools. The visualizer comprises a grid page that contains a 'start node' and an 'end node'. The viewer can add various features such as a maze, walls and weights to improve the general outlook and understand how these pathfinding algorithms tackle our day-to-day problems. To build a visualizer, a programmer needs a fair knowledge of front-end programming languages and a good understanding of pathfinding Algorithms.
Keywords : Greedy Best-first Search, Swarm Search, Breadth-first, Depth-first

I. INTRODUCTION

"An algorithm must be seen to be believed." are the famous words said by Donald Knuth, a computer scientist. Just like that with the help of this tool we are doing the same.

This tool can help beginners and even experienced programmer to understand algorithms a cut above. It is an web app consisting of a (19x55) grid containing a start node and an end node which a user can place anywhere on this grid. A user can further add walls and weights to imitate a real world situation. When the grid is set choose an algorithm and the with the help of this till you'll be able to visualize how an algorithm actually works.A user can fuher can the speed and see the time taken by the algorithm to complete the task.

II. LITERATURE REVIEW

Literature Review is required to take the matter into considerations that can't be cleared in the past researches. Many researchers try to interpret various kind of conclusions and to improve those past results literature review is needed. The present literature serves many varied interesting features, which forms the vital background for the study and conducted a consideration.

An important field of mathematical theory is the mathematical study of the structure of abstract relationships between objects by means of graphs (networks). Although investigating of these constructions can be purely theoretical, they can be used to model pair wise relationships in many real world systems. One of most widely using applications is determination of shortest paths in many practical applications as: maps; robot navigation; texture mapping; typesetting in TeX; urban traffic planning; optimal pipelining of VLSI chips; subroutines in advanced algorithms; telemarketer operator scheduling; routing of telecommunications messages; approximating piecewise linear functions; network routing protocols (OSPF, BGP, RIP); exploiting arbitrage opportunities in currency exchange; optimal truck routing through given traffic congestion pattern.

DATA STRUCTURES

In practice, graphs are usually represented by one of two standard data structures: adjacency lists and adjacency matrices. At a high level, both data structures are arrays indexed by vertices; this requires that each vertex has a unique integer identifier between 1 and V. In a formal sense, these integers are the vertices.

ADJACENCY LISTS

By far the most common data structure for storing graphs is the adjacency list. An adjacency list is an array of lists, each containing the neighbors of one of the vertices (or the out-neighbors if the graph is directed). For undirected graphs, each edge uv is stored twice, once in u's neighbor list and once in v's neighbor list; for directed graphs, each edge uv is stored only once, in the neighbor list of the tail u. For both types of graphs, the overall space required for an adjacency list is O(V + E).

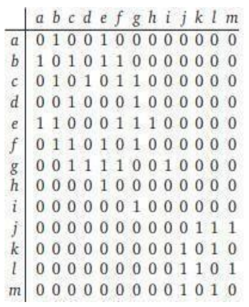
There are several dierent ways to represent these neighbor lists, but the standard implementation uses a simple singly-linked list. The resulting data structure allows us to list the (out-)neighbors of a node v in O(1 + deg(v)) time; just scan v's neighbor list. Similarly, we can determine whether uv is an edge in O(1 + deg(u)) time scanning the neighbor list of u. For undirected graphs, we can improve the time to O(1 + min{deg(u), deg(v)}) by simultaneously scanning the neighbor lists of both u and v, stopping either when we locate the edge or when we fall of the end of a list.

ADJACENCY MATRICES

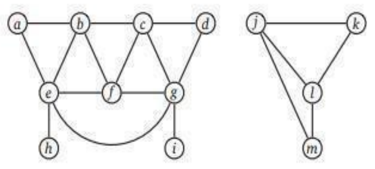
The other standard data structure for graphs is the adjacency matrix, first proposed by Georges Brunelin. The adjacency matrix of a graph G is a V ⇥ V matrix of 0s and 1s, normally represented by a two-dimensional array A[1 .. V, 1 .. V], where each entry indicates whether a particular edge is present in G. Specifically, for all vertices u and v:
if the graph is undirected, then A[u, v] := 1 if and only if uv 2 E, and if the graph is directed, then A[u, v] := 1 if and only if uv 2 E.
For undirected graphs, the adjacency matrix is always symmetric, meaning A[u, v] = A[v, u] for all vertices u and v, because uv and vu are just dierent names for the same edge, and the diagonal entries A[u, u] are all zeros. For directed graphs, the adjacency matrix may or may not be symmetric, and the diagonal entries may or may not be zero.

Given an adjacency matrix, we can decide in
(1) time whether two vertices are connected by an edge just by looking in the appropriate slot in the matrix. We can also list all the neighbors of a vertex in ⇥ (V) time by scanning the corresponding row (or column). This running time is optimal in the worst case, but even if a vertex has few neighbors, we still have to scan the entire row to find them all.

Similarly, adjacency matrices require ⇥ (V2) space, regardless of how many edges the graph actually has, so they are only space-evident for very dense graphs.



	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0	1	0	0	1	0	0	0	0	0	0	0	0
b	1	0	1	0	1	1	0	0	0	0	0	0	0
c	0	1	0	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	1	0	0	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	0	0	0
f	0	1	1	0	1	0	1	0	0	0	0	0	0
g	0	0	1	1	1	1	0	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	1	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	1	1	1
k	0	0	0	0	0	0	0	0	0	1	0	1	0
l	0	0	0	0	0	0	0	0	0	1	1	0	1
m	0	0	0	0	0	0	0	0	0	1	0	1	0



III. METHODOLOGY

In this section, the overall working of the project has been described. How the project started and how the project works and how the various phases of project were carried out and the challenges faced at each level.

What does the project do?
At its core, a pathfinding algorithm seeks to find the shortest path between two points.
This project visualizes various pathfinding algorithms in action, and more!
All of the algorithms on this project are adapted for a 2D grid, where 90 degree turns have a "cost" of 1 and movements from a node to another have a "cost" of 1.

PICKING AN ALGORITHM
Choose an algorithm from the "Algorithms" drop-down menu. Note that some algorithms are unweighted, while others are weighted. Unweighted algorithms do not take turns or weight nodes into account, whereas weighted ones do.
Additionally, not all algorithms guarantee the shortest path.

MEET THE ALGORITHMS
Dijkstra's Algorithm:The father of pathfinding algorithms; guarantees the shortest path.

A*: A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. It is a very smart algorithm and is very efficient.

Greedy Best-first Search (weighted): A faster, more heuristic-heavy version of A*; does not guarantee the shortest path.

Breath-first Search (unweighted): A great algorithm; guarantees the shortest path.

Depth-first Search (unweighted): A very bad algorithm for pathfinding; does not guarantee the shortest path.

ADDING WALLS
Click on the grid to add a wall.
Walls are impenetrable, meaning that a path cannot cross through them.
Visualizing and more: Use the navbar buttons to visualize algorithms and to do other stuff!
You can clear the current path, clear walls and weights, clear the entire board, and adjust the visualization speed, all from the navbar. If
You want to access this tutorial again, click on "Pathfinding Visualizer" in the top left corner of your screen.

IV. OBJECTIVE

- ✓ It can be used as a E learning tool to understand Algorithms.
- ✓ It is used in finding Shortest Path.
- ✓ It is used in the telephone network.
- ✓ It is used in IP routing to find Open shortest Path First.
- ✓ It is used in geographical Maps to find locations of Map which refers to vertices of graph.
- ✓ We can make a GPS system which will guide you to the locations.

- ✓ Search engine crawlers are used BFS to build index. Starting from source page, it finds all links in it to get new pages.
- ✓ In peer-to-peer network like bit-torrent, BFS is used to find all neighbor nodes.
- ✓ As users of wireless technology, people demand high data rates beyond GigaBytes per second for Voice, Video and other applications. There are many standards to achieve data rates beyond GB/s. One of the standards is MIMO(Multi input Multi output).MIMO employs K-best Algorithm(which isa Breadth-First Search algorithm) to find the shortest partial euclidian distances.

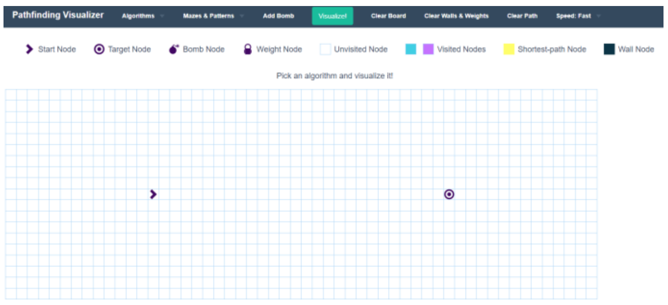
V. OVERVIEW

The development of this project has been carved out in 6 phases. These phases include all the steps of the project, beginning from data collection and processing to output for the user. The 6 phases are:

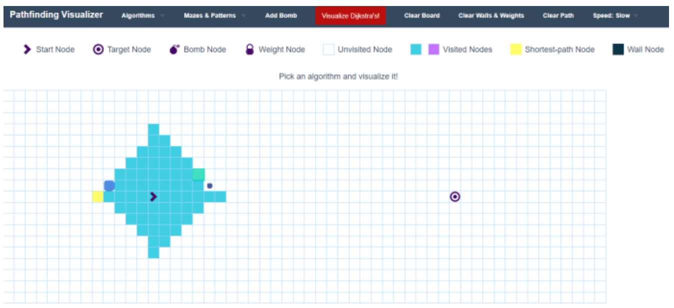
1. Building of the graph matrix.
2. Adding walls and event listeners.
3. Embed the graph algorithms.
4. Integrate the pathfinding functionality.
5. Improve the design and UI.
6. Added the timer functionality. After all these phases the project is completely ready for the user to use.

VI. PATHFINDING VISUALIZER

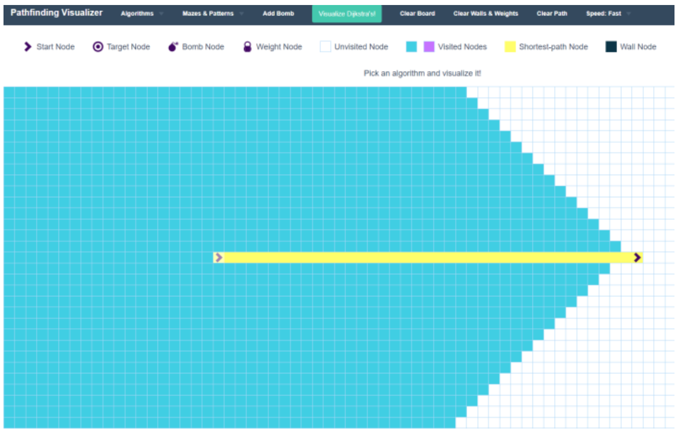
Now our visualizer is complete, so let's use it.
Firstly pick an algorithm of your choice.



Now, add walls and weights. (optional)



Click on visualize and it will start visualizing the algorithm you chose.



After a couple seconds, will will find the shortest path and will trace that path using the yellow line.

VII. FUTURE WORK

For time being the program contains only limited amount of pathfinding algorithms we would like to add a lot more algorithms and want to visualize then in both 2-D and 3-D. We would also like to add more tools for comparing these algorithms.

VIII. CONCLUSION

In this paper we have described a pathfinding visualizer that was implemented and created using entirely open source off the shelf software. By making this particular project we made it easy to understand and learn about the algorithms.

IX. REFERENCES

[1]. https://en.wikipedia.org/wiki/Algorithm#:~:text= Algorithms%20are%20always%20 unambiguous%20and,automated%20reasoning%2C%20and%20other%20tasks.&text=As%20an%20effective%20method%2C%20an,language%20for%20calculating%20a%20function.
[2]. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
[3]. https://www.geeksforgeeks.org/a-search-algorithm/
[4]. https://www.geeksforgeeks.org/best-first- search-informed-search/
[5]. https://www.geeksforgeeks.org/breadth-fi rst-search-or-bfs-for-a-graph/
[6]. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
[7]. https://www.w3schools.com/js/default.asp
[8]. https://www.meta-chart.com/histogram
[9]. Roles J.A. & ElAarag H. (2013). A Smoothest Path algorithm and its visualization tool. Southeastcon, In Proc. of IEEE, DOI: 10.1109/SECON.2013.6567453

Cite this article as :

Mohak Mohan, Mehak Pargal, Dr. Simmi Dutta, "Development of Tool for Visualizing Pathfinding Algorithms", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7, Issue 4, pp.439-454, July-August-2021.