PATHFINDING VISUALIZER FOR SHORTEST PATH ALGORITHMS

Saurabh Gawade[*1], Prathmesh Gaikwad[*2], Sahil Kadam[*3], Farhan Patel[*4],

Prof. Akshata Laddha[*5]

[*1,2,3,4]Student, Department of Computer Engineering, Dilkap Research Institute of Engineering and Management Studies, City, Neral-Karjat, Maharashtra, India

[*5]Professor, Department of Computer Engineering, Dilkap Research Institute of Engineering and Management Studies, City, Neral-Karjat, Maharashtra, India

ABSTRACT

Algorithm visualization has been a popular topic in computer science education for some time, but it has not yet become a primary educational tool in colleges and universities. This paper identifies two key requirements for successful algorithm visualization: the widespread availability of the software used and the ability to visualize why an algorithm solves a problem rather than just what it is doing. One approach to achieving this is through the use of algorithm invariants, which are well-known in program verification theory and software validation. Many researchers believe that understanding invariants is essential to understanding algorithms. Due to recent advances in computing power, even complex visualizations involving 3D animation can be successfully implemented using interpreted graphic scripting languages like JavaScript, which are available to all web users without the need for installation. The use of images can convey useful information about algorithms and provide a third way to study them, in addition to theoretical and empirical analyses.

Keywords: Visualization, Animation, Algorithm Visualizer

I. INTRODUCTION

Algorithm visualization, also known as algorithm vitality, is a technique that uses dynamic displays to demonstrate the computation of a given algorithm. While initial attempts to visualize algorithms date back to the 1980s, the golden age of algorithm visualization was around 2000, when software tools like Java and its graphic libraries became available. Despite hopes that it would revolutionize the teaching of algorithms, algorithm visualization has not been widely adopted and is not widely used in computer science courses today. This may be due to the fact that algorithm visualization is typically used by running the algorithm slowly or in steps, which can make it difficult for learners to understand what is happening. While there have been attempts to automate the creation of animated algorithms and visualization tools to enable learners to create their own demonstrations, algorithm visualization has not gained widespread use in instruction.

II. LITERATURE REVIEW

The Literature Review essential for exploring findings that remain unclear or require improvement from previous investigations. Researchers often review various outcomes to enhance and provide new insights into existing research. The current literature offers a plethora of intriguing features that serve as the foundation for reflection and study.

An important aspect of comprehensive research is examining the structure of ideal connections between objects through graph networks. While the analysis of these structures may appear theoretical, they have practical uses in modeling smart connections in real-world systems. One of the most significant applications of this study lies in identifying short paths in numerous industries, including maps, robot navigation, texture mapping, typesetting in TeX, community business planning, optimal pipelining of VLSI chips, subroutines in improved algorithms, scheduling telemarketer drivers, routing telecommunications letters, piecewise direct places, network routing protocols (OSPF, BGP, RIP), exploiting arbitrage openings in currency trade, optimal commutation routing through business patterns.

DATA STRUCTURES

Graphs are typically presented using one of two common data structures: adjacency lists or adjacency matrices. Both structures are essentially arrays that are indexed by vertices. However, to enable indexing, each vertex must be assigned a unique integer identifier between 1 and v. In a sense, these integers themselves represent the vertices in the graph.

ADJACENCY LISTS

The most commonly used data structure for storing graphs is the adjacency list. An adjacency list is an array of lists, whereby each list contains the neighbors of a particular vertex, or the out-neighbors for a directed graph. When the graph is undirected, the edge is stored twice, once in the neighbor list of each vertex. Conversely, when the graph is directed, the edge is stored only in the neighbor list of the tail vertex. For either type of graph, the space requirements for the adjacency list is O(V + E).

There are many ways to represent these neighbor lists, but the standard way is using a singly-linked list. Using this data structure, we can list the out-neighbors of a vertex in O(1 + deg(v)) time by simply scanning its neighbor list. Furthermore, determining whether uv is an edge takes O(1 + deg(u)) time, by scanning the neighbor list of u. However, for undirected graphs, we can achieve faster results by scanning the neighbor lists of both u and v simultaneously. In this case, we stop either when the edge is found or when we reach the end of a list. This approach improves the time complexity to O(1 + min{deg(u), deg(v)}).

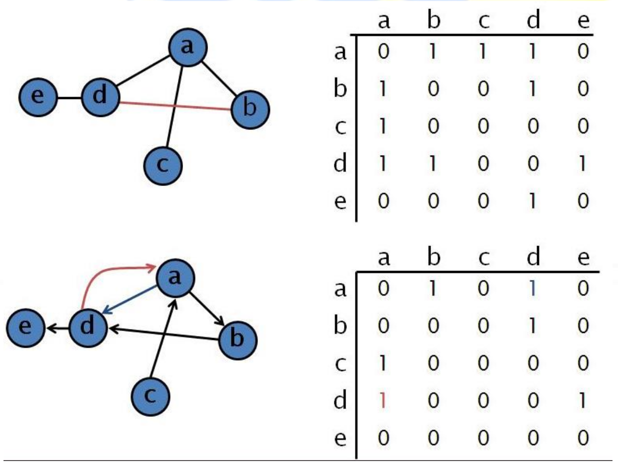
ADJACENCY MATRICES

Georges Brunelin introduced the adjacency matrix, a standard data structure for graphs that consists of a matrix of 0s and 1s. Each entry within the matrix indicates the presence of an edge in a graph G. The matrix is represented as a two-dimensional array, A[1 .. V, 1 .. V], and has a size of V ⇥ V. If the graph is undirected, then A[u, v] is set to 1 if and only if uv 2 E for all vertices u and v. Similarly, if the graph is directed, then A[u, v] is set to 1 if and only if uv 2 E.

The adjacency matrix is an essential tool in graph theory as it enables an efficient representation of graphs and their properties. Using the matrix, determining the degree of a vertex, finding connected components, and detecting cycles in a graph can be done easily. The adjacency matrix's versatility and simplicity make it a fundamental concept in graph theory that has a wide range of applications. Fields such as computer science, social network analysis, and transportation planning use the adjacency.

The adjacency matrix demonstrates symmetry in undirected graphs, where each vertex is connected to another. This means that A[u, v] is equivalent to A[v, u] for all vertices u and v. This symmetry is due to the fact that uv and vu represent the same edge, and the diagonal entries A[u,u] are all zeroes. Directed graphs, however, do not always exhibit this symmetry, as the directions of the edges matter. Additionally, the diagonal entries in directed graphs may or may not be zeroes, depending on whether or not there are self-loops present.

An adjacency matrix allows us to quickly determine if two vertices are connected by an edge, taking only O(1) time to look in the corresponding slot in the matrix. Additionally, we can list all the neighbors of a given vertex by scanning the row or column in O(V) time. However, this method requires scanning the entire row even if the vertex has few neighbors, resulting in suboptimal performance. Although adjacency matrices offer optimal performance in the worst case scenario, they require O(V^2) space regardless of graph density, making them only efficient for dense graphs.



	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0

	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	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 center, a pathfinding algorithm seeks to discover the shortest path among 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 ladened. Unweighted algorithms don't take turns or weight nodes into record, whereas weighted ones do. also, not all algorithms guarantee the shortest path.

MEET THE ALGORITHMS

Dijkstra's algorithm : is a widely-used algorithm in computer science for solving the shortest path problem, particularly in graph theory. it turned into advanced by dutch computer scientist edsger w. Dijkstra in 1956 and has since become known as the father of pathfinding algorithms. The algorithm works by starting at a specified node and examines its neighbors, selecting the node with the smallest distance and visiting its neighbors. This process is repeated until the final destination is reached, ensuring that the shortest path is found.

A* search algorithm : is a modification of Dijkstra's algorithm that incorporates heuristic functions to improve its efficiency. This algorithm is widely used in the field of artificial intelligence and is particularly effective in solving pathfinding problems in video games, robotics, and autonomous vehicles. The algorithm considers both the cost of the path taken and the estimated remaining cost to the destination, helping it make informed decisions about which paths to explore first.

Breath-first search (BFS) : is another algorithm often used in pathfinding, particularly when dealing with unweighted graphs. This algorithm works by exploring all possible paths from a starting node, one level at a time, until it reaches the destination. BFS is a remarkable set of rules because it guarantees the shortest path to the destination.

Depth-first search (DFS) : algorithm is a terrible set of rules for pathfinding because it does not guarantee the shortest route. this set of rules starts off evolved at the basis node and explores as some distance as possible earlier than backtracking. DFS can be useful in some situations, such as graph traversal, but it is not ideal for finding the shortest path.

ADDING WALLS

Click at the grid to feature a wall.

Walls are impenetrable, meaning that a path cannot cross through them.

Visualizing and greater: 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.

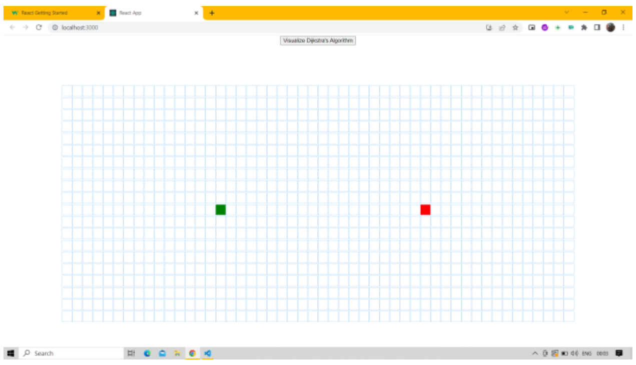
IV. OBJECTIVE

- ✓ It can be used as a E learning tool to understand Algorithms.
- ✓ ✓ It's used in chancing Shortest Path.
- ✓ It's used in the telephone network.
- ✓ ✓ It's used in IP routing to find Open shortest Path First.
- ✓ It's used in geographical Charts to find locales of Chart which refers to vertices of graph.
- ✓ We can make a GPS system which will guide you to the locales.
- ✓ search engine crawlers are used bfs to build index. starting from source page, it finds all hyperlinks in it to get new pages.
- ✓ In peer-to-peer community like bit-torrent, bfs is used to locate 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. RESULTS AND DISCUSSION

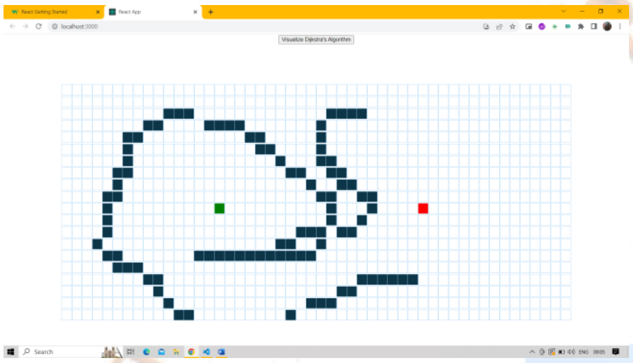
Now our visualizer is complete, so let’s use it.

First pick an algorithm : select an algorithm of your choise.

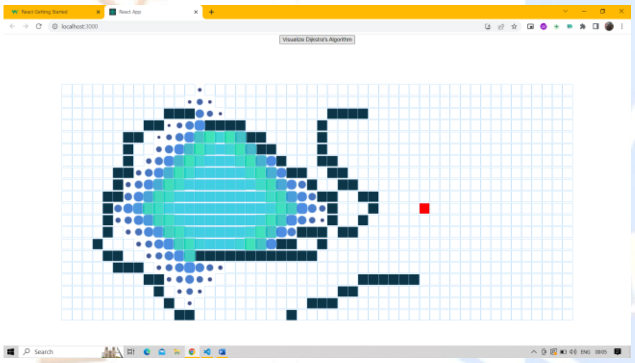


Now, add walls and weights : (these are optional)

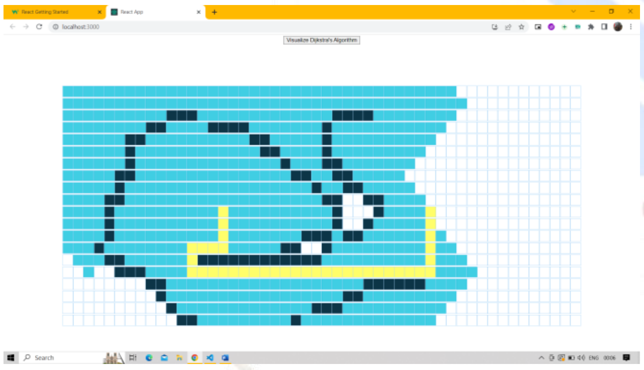
Now, add walls and weights : (these are optional)



Click on visualize button and it will start visualizing the algorithm.



After a couple seconds, it will find the shortest path and display the shortest path using the yellow line.



VI. FUTURE WORK

For time being the program contains only limited quantum of pathfinding algorithms we'd like to add a lot further algorithms and want to fantasize also in both 2- D and 3-D. We'd also like to add further tools for comparing these algorithms.

VII. CONCLUSION

The visualizer is capable of showcasing a variety of pathfinding algorithms such as Dijkstra's, A-star, Greedy Best-First Search, and Breadth-First-Search. The interface is simple, easy to use and provides users with a clear idea of how each algorithm works in finding the optimal path between a start and goal point within a maze. The project provides a useful tool to help people understand and learn about the core concepts of pathfinding and algorithms. The visualizer can aid educators in teaching the subject in an engaging way and can be used as a fun and interactive way to introduce the basics of algorithms to students. Overall, the pathfinding visualizer can be considered as a valuable tool not just for students, teachers or educational institutions but also for researchers who are looking to develop and test new pathfinding algorithms.

VIII. REFERENCES

1. Dijkstra, E. W. (1959). A Note on Two Problems in Connection with Graphs. Numeriche Mathematik, 1, 269-271.
2. Paramythis A., Loidl S., Mühlbacher J. R., & Sonntag M. (2005). A Framework for Uniformly Visualizing and Interacting with Algorithms. In Montgomerie, T.C., & Parker E-Learning, J.R. (Eds.), In Proc. IASTED Conf. on Education and Technology (ICET 2005), Calgary, Alberta, Canada, 2-6 July 2005, pp. 28- 33.
3. Fouh E., Akbar M. & Shaffer C. A. (2012). The Role of Visualization in Computer Science Education. Computers in the Schools, 29(1-2), 95-117.
4. 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.
5. L. Wenzheng, L. Junjun and Y. Shunli, "An Improved Dijkstra's Algorithm for Shortest Path Planning on 2D Grid Maps," IEEE, 2019.
6. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). Introduction to Algorithms, 2nd edition. The MIT Press, Cambridge, Massachusetts, and McGraw Hill, Boston.
7. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
8. https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd
9. https://en.wikipedia.org/wiki/Pathfinding
10. https://www.w3schools.com/js/
11. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
12. https://www.researchgate.net/publication/282488307_Pathfinding_Algorithm_Efficiency_Analysis_in_2D_Grid