EC504 Advanced Data Structures

Web-based Algorithm Visualizer

Niantong Dong	U55783192	niantong@scc1.bu.edu
Zixuan Shu	U48380935	zmshu@bu.edu
Chen-Wei Weng	U58151415	cwweng@bu.edu
Yiqin Zhang	U83608825	evezhang@bu.edu

Introduction

Learning algorithms is hard, especially when you read the textbook and easily get lost in those complicated explanations of how the algorithm works. In fact, the better communication way is using a graph to show you the idea about what the author wants to tell you about the algorithms. In this case, it is important to have a tool to show you how those algorithms work.For our project, we create a web-based algorithm visualizer to help us to learn the mechanism of algorithms using Nodejs. Since it is a one-month project and we are not quite familiar with web development, we try our best to make this web as interactive with users as possible. Our project is not perfect but still a helpful tool.

Abstract

In this project, we built sorting algorithms and path finding algorithms visualizer which enables users to watch and learn how algorithms work and the difference between algorithms. We constructed our work as a web-based application using Node.js, CSS, and HTML. For sorting algorithms, we implemented Bubble sort, Selection sort, Insertion sort, Quick sort, Heap sort, and Merge sort. For path finding algorithms, we have Dijkstra's algorithm, A* Search, Greedy Best-first Search, traditional Best-first Search, and Depth-first search.

Algorithms

Bubble sort: the simplest sorting algorithm that sorts by repeatedly swapping the adjacent elements.

Selection sort: the sorting algorithm that sort an array by repeatedly finding the minimum element.

Insertion sort: simple sorting algorithm that builds the result one item at a time.

Quicksort: sorting algorithm that works by selecting a pivot element from the array and partitioning the other elements into subarrays.

Heapsort: a comparison based sorting algorithm based on Binary Heap data structure.

Merge sort: another comparison based sorting algorithm, it divides the array into two halves, calls itself for the two halves, and then merges the two sorted arrays.

Dijkstra's Algorithms: a pathfinding algorithm uses a data structure (priority heap) for storing and querying partial solutions sorted by distance from the start.

A* Search: a pathfinding algorithm works by making a lowest-cost path tree from the start node to the target node.

Greedy Best-first Search: Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule

BFS: an algorithm for traversing or searching a tree or graph, starting at a node and exploring all of the neighbor nodes.

DFS: an algorithm for traversing or searching a tree or graph, starting at anode and exploring as far as possible along each branch before backtracking.

Instructions

Our source code is on github(https://github.com/NiantongDong/EC504-Final-Project). To run this application on your side.

Firstly, download the source code from the Github. Make sure you have the latest version of Nodejs and NPM. Follow the instruction on this link to install them https://phoenixnap.com/kb/install-node-js-npm-on-windows

To check your Nodejs and NPM version, run this Two commands.

```
node -v
npm -v
```

Then, we need to install relative dependencies for our project. Make sure you have the *package.json* file for this operation. To do so, run

```
npm install
```

When you do npm install, you may have some issue with the installation. Run this to fix them all

```
npm audit fix --force
```

After that, you can deploy this web on the localhost. To change the port, go to the server.js

```
app.listen(3000, () => { //Change 3000 to any port number you want

  console.log("The server is up and running!");
});
```

Then, run this command. Once the server is up and running, use browser to visit localhost:3000

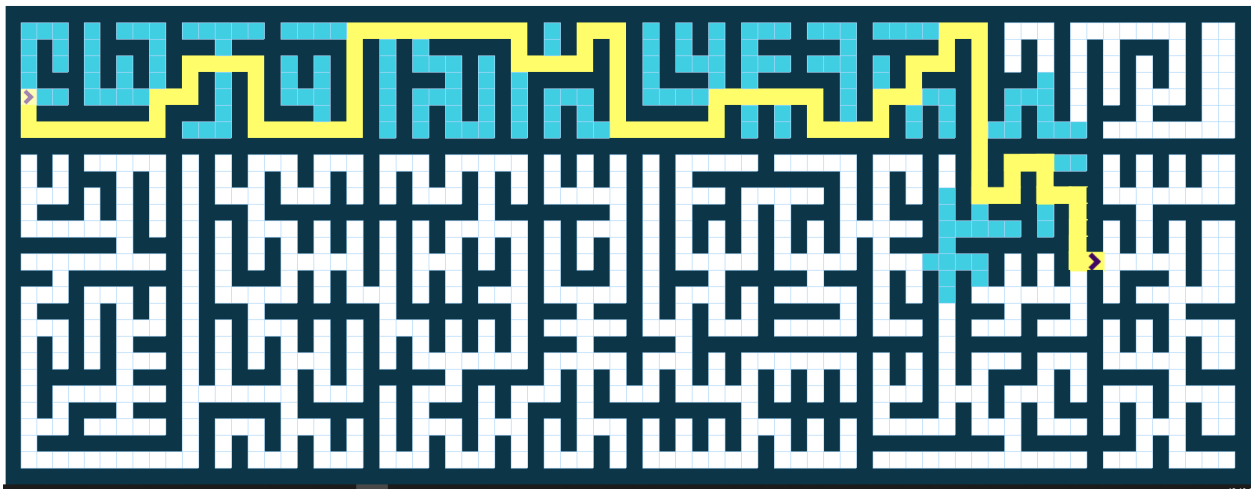
```
npm start
```

When the web launches on localhost, you will first enter a selection page to see which visualizer you want to try. We have the sorting algorithm visualizer and the pathfinding algorithm visualizer. Click the one you want and it will redirect you to the visualizer.

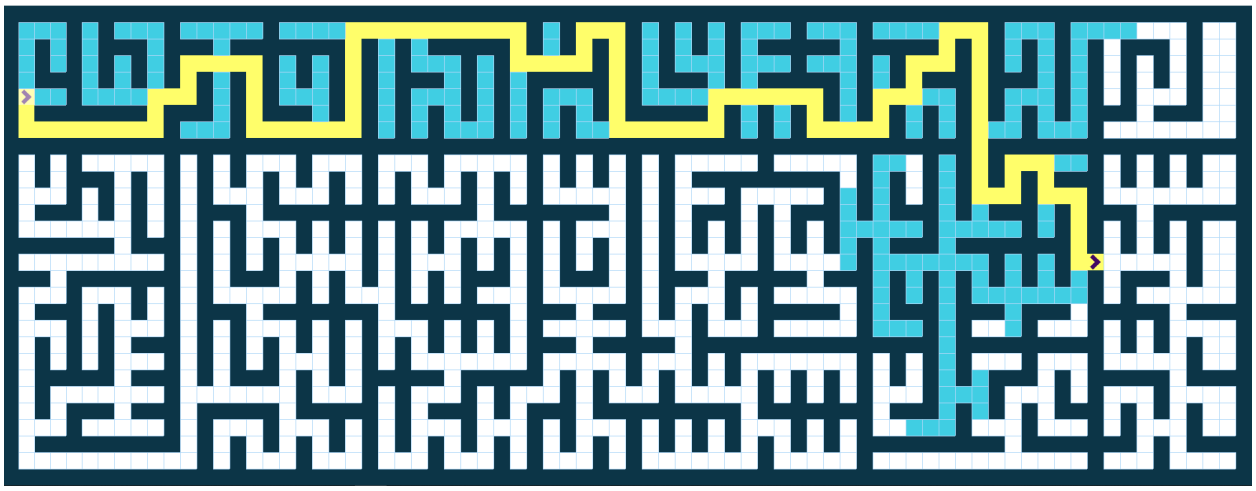
For pathfinding visualizer, we will provide the initial start node and destination node, you can draw it to anywhere you want. Then, on the navigation bar, choose the algorithm you want and click “Visualize”. The animation will show you how our program finds the shortest path. To add a wall node on the board, simply click the blank node and it will add a wall node. To add a weight node, press W and left click the node you want. The weight node will be considered as the last option for the path if there is no path shorter than the path with weight node.

For the sorting visualizer, You will have some selections for you. First, choose the array size as many as you want. Then select the sorting algorithm. After that, you can also select how you wanna create your initial array. Finally, choose the speed for each operation. Once you select everything, click “Start” to see how our program sorts the array.

Results



This sample result use A* algorithm to find the shortest path from start to destination



For the same maze, the result using Dijkstra's Algorithms will have a different way to find the path.

For the sorting visualizer, our implementation does not show a lot about how to sort the array. We have a simple demo video that will show you how to do so. You can find if on our Github repo named *SortingDemo*

Conclusion

This project has shown that we can demonstrate different algorithms in the way of animation by using modern web design technologies. Our application is capable of performing sorting algorithms and path finding algorithms on browsers. From this project, we have a chance to study these algorithms deeply. Second, we learn the experience of combining traditional algorithm knowledge and modern web design skills to create a marvelous application. Last but not least, we understand how animation can help people learn difficult ideas more efficiently and more fastly. There are several avenues for future work. For instance, in path finding algorithms, we can have users decide the weight of each cell or provide the distance between the cell to start point and end point. So the efficiency of users' learning can be even further improved.

Reference

https://github.com/clementmihailescu/Pathfinding-Visualizer-Tutorial

https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd

http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

https://www.geeksforgeeks.org/sorting-algorithms/