ALGORITHMS AND PROBLEM SOLVING LAB
PROJECT
(15B17CI471)

PATH FINDER USING DIJKSTRA'S ALGORITHM

SUBMITTED BY:

9920103062 RHYTHM SHANDLYA
9920103071 ADITYA KURELE
9920103085 ABHINAV SINHA
9920103090 KARTIK VALECHA

SUBMITTED TO:

HIMANI BANSAL

Department of CSE/IT
Jaypee Institute of Information Technology University, Noida

MAY, 2022

Introduction:

Finding a path from a source to a destination while also avoiding obstacles and minimizing costs such as time, distance, risks, fuel, price, etc. is a common programming challenge for software engineers. One of the best solutions to tackle this problem is for us to implement Djikstra's algorithm. For this project, we will be building a visualization of the runtime of Dijkstra's algorithm on a user-generated maze. Dijkstra's algorithm is a pathfinding algorithm that finds the shortest path in a network of vertices and weighted edges between these vertices..

Motivation to choosing Dijkstra's algorithm:

Dijkstra's Algorithm lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favours lower cost paths. It is the algorithm of choice for finding the shortest path paths with multiple destinations. Traversal algorithms such as breadth-first-search and depth-first-search explore all possibilities : they iterate over all possible paths until they reach the target. However, it may not be necessary to examine all possible paths; Dijkstra's algorithm eliminates useless traversal using heuristics to find the optimal paths.

Methodology:

In this project, we will be building on a unweighted graph (a 2D grid) so it functions similarly to a breadth first search traversal, but the underlying JavaScript implementation of Dijkstra's is given so that a user can also be able expand further and incorporate edge weights into the project as per their liking. This project is a React.js visualizer based on functional components. Our project is mainly divided into three code files :- Node, Dijkstra and PathFinder.

Node

The Node component is equivalent to a single tile on your 2D grid. It can be either a wall or a valid space depending on whether or not it is clicked. While making the Node component, we need to consider certain conditions such as whether or not the node itself is a wall, and whether or not the user has clicked on a node to convert the node. Some additional things to consider would be the

node's location, and whether or not the node is your start/end point for the algorithm's execution. We will also implement some additional styling options depending on the what the node type actually is. If it's a wall, it should be a different tone, if it's the start point or finish, colour it differently. The purpose of the is to give the parent component a connection to the Node component with these event handler functions, so the Node component itself can update via a prop change from the parent component. In other words, the parent component defines some function(s) to change the overarching grid structure that Dijkstra's algorithm runs on depending on what is and isn't a wall, or where the start/end points are. However, these changes must be propagated when the user interacts with the individual Node components. So by passing down the functions to the child component (Node) you can induce changes onto the parent component from some event (click/drag) on the child

Dijkstra

This file will primarily perform Dijkstra's algorithm. It will return all nodes in the order in which they were visited. This will also make the nodes point back to their previous node, effectively allowing us to compute the shortest path by backtracking from the finish node.

Path Finder

This component will be our top-level component (i.e. the parent to the Node component). At the top where our import statements are, we have have imported our Node component along with some Dijkstra's functions. Here, we will define all the function as well as do the actual animation of Dijkstra's progression. When the progression is animated, we will have to highlight the nodes in the order in which they were visited with some amount of delay between each. Then, we will need to call the function to animate the shortest path after you visualize every visited node.

Contributions that the project will be able to make:

Many studies show that visualization is an efficient way of learning any concept faster than conventional methods. Pathfinding Visualization is allowing us to create a learning tool that helps in improving computer science education very much. The goal of this project is to visualize the runtime of which can be used to visualize and understand the concept behind Dijkstra's algorithm.

Moreover, the educational benefits of the project can also be increased by the addition and implementation of other shortest path algorithms.

Also, Dijkstra's Algorithm has several real-world use cases where visualizer models are needed, some of which are as follows:

- Digital Mapping Services in Google Maps.
- Social Networking Applications.
- Telephone Network.
- IP routing to find Open shortest Path First.
- Flighting Agenda.
- Designate file server.
- Robotic Path.
- Prediction model for spread of diseases.

Hardware and Software requirements:

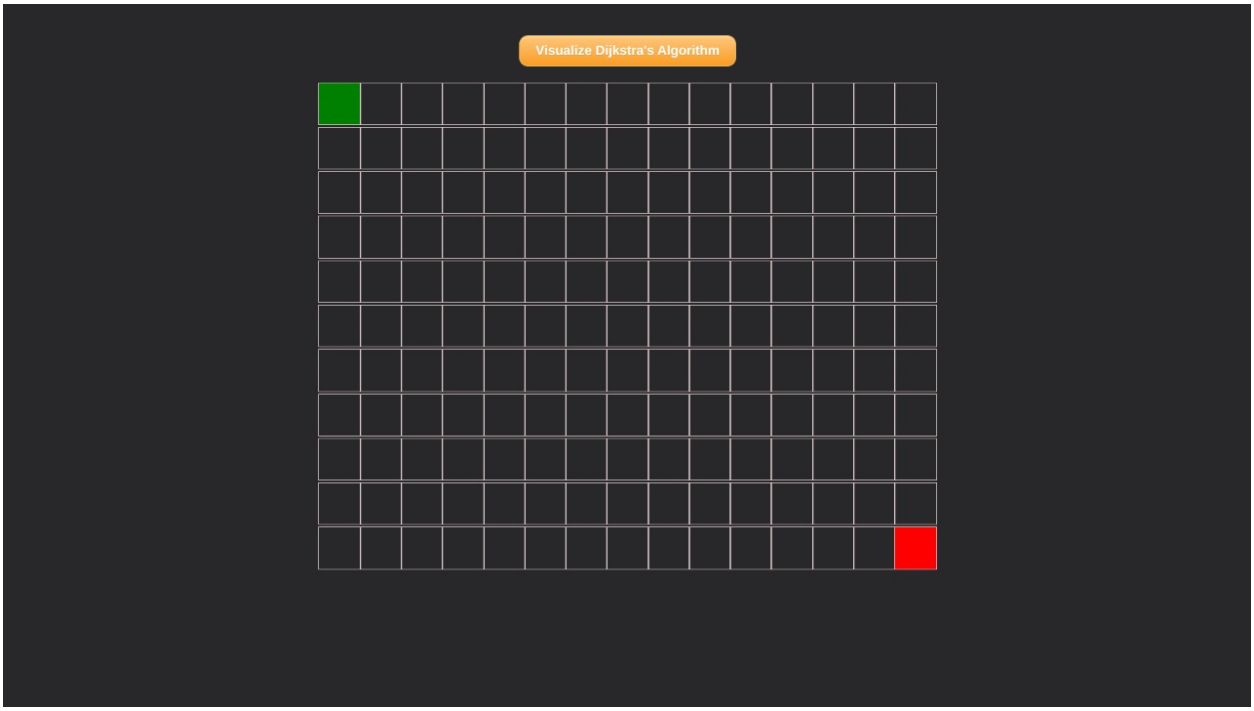
- Browser of your choice
- React.js
- NPM (Node Package Manager)
- PC

Summary:

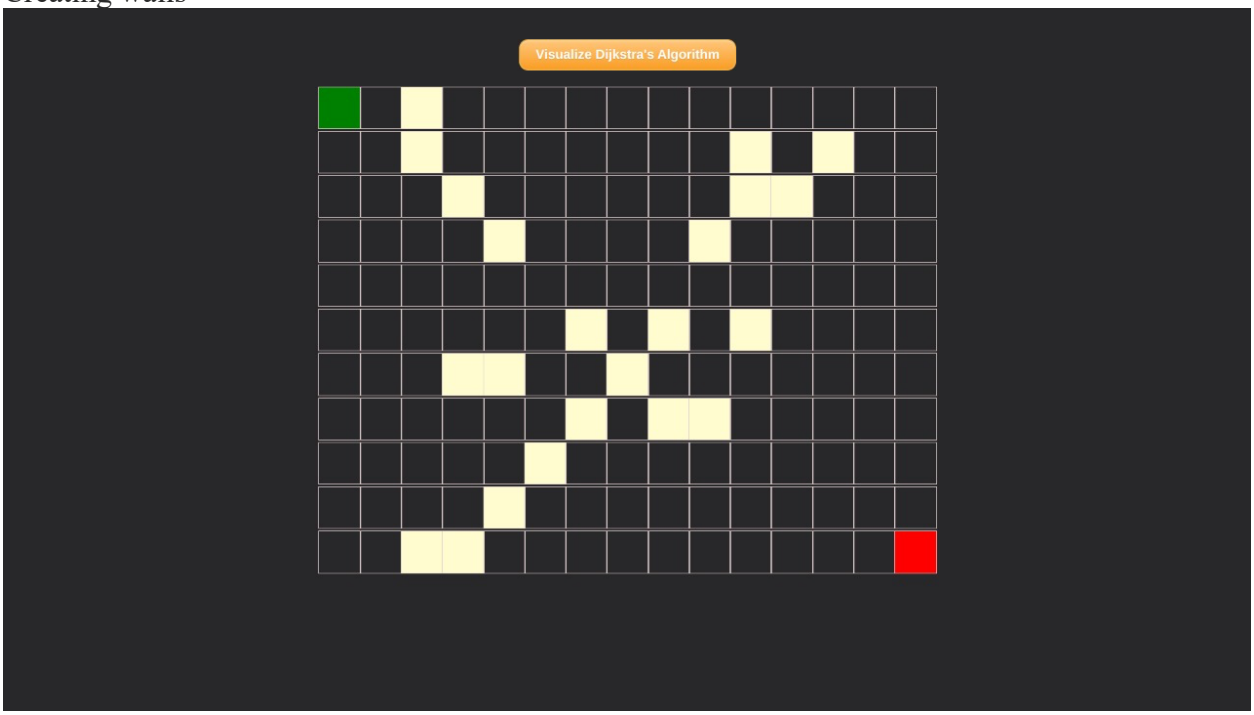
The project is called a **Path Finder Visualizer** using Dijkstra's algorithm, aptly because it does what it says, it finds a path from a source to a destination using Dijkstra's algorithm. It is based on greedy algorithm and visually shows the steps taken by the algorithm to reach its destination while also avoiding obstacles and minimizes time cost by assuming that a global optimum can be arrived at by selecting a local optimum.

SCREENSHOTS

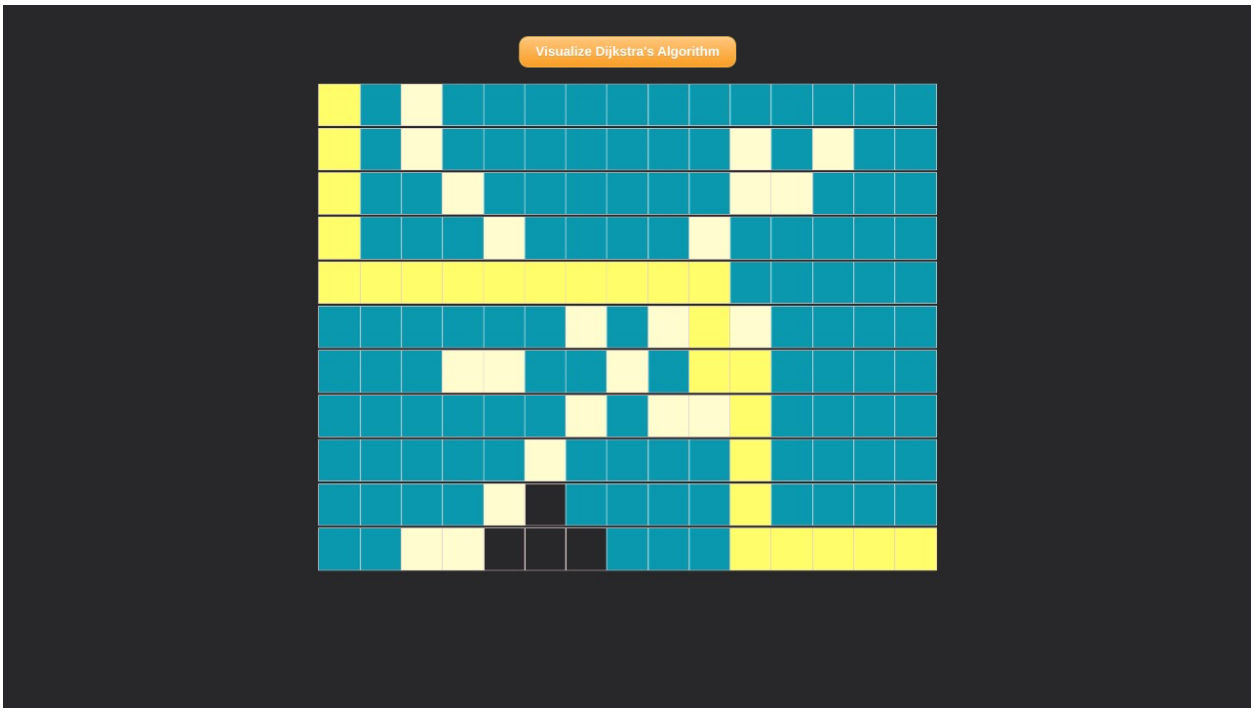
Starting screen



Creating walls



Path Finder Visualization



**********END OF REPORT**********