Technical Documentation

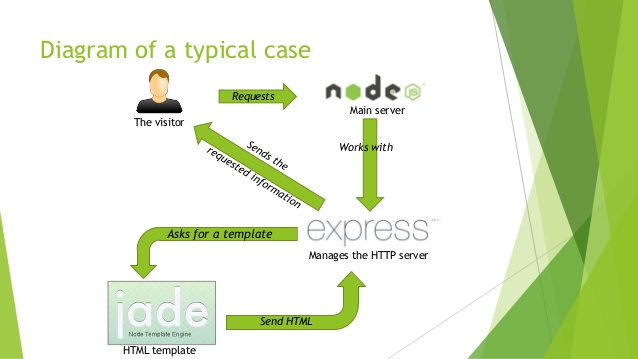
Current documentation that is subject to change as the application develops.

**JavaScript Technologies**

I will be using various JavaScript technologies to enable me to create the best application possible.

Node JS

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side. Normally, JavaScript is used as a client side language with scripts being processed in browser. Node Js changed this so that JavaScript can be also used a back-end language. I using this to my advantage which means my application will be a ‘one programming language’ application. Meaning, I can use the same language at both the back-end and the front-end increasing productivity and code readability. Node JS for my application will specifically be used for handling client/server requests. This will be done using a popular framework called express which is the standard web application for node js. A small diagram illustrating this.



Additionally, with Node JS and express support, I have the ability to connect to a MongoDB. I feel this could be useful in my application. An idea for a database is if a user has a user profile that can log in, an idea can be that it’s possible for the user to save their graphs and animations if they wish. This is an idea for the future.

External Libraries

There are also important external libraries that are critical for the application to function. My main one being the actual graph library which is responsible for displaying nodes and edges on the screen. This library is called sigma Js. Sigma is a JavaScript library dedicated to graph drawing. It makes easy to publish networks on Web pages, and allows developers to integrate network exploration in rich Web applications. Sigma provides a lot of built-in features, such as Canvas and WebGL renderers or mouse and touch support, to make networks manipulation on Web pages smooth and fast for the user. It has been designed as an engine that you can customize and use to develop highly interactive Web applications that show graph visualizations. There is excellent documentation for this library with an active helpful community who willing to come to developers aid if there is a problem. Another advantage of this that it support mobile touch support, which is a key point for my application to have. I have other helpful libraries, as mentioned and gone into more detailed below there are small libraries such as Tree.js. This will be help with my implementation of algorithms that are critical for my application to run. Particles is also a nice front end functionality I have which will give a good aesthetic display at the intro section of the website. I have been in contact with the community in Sigma JS when I have required some expert knowledge. Combining this with the vast documentation, this has been a very pleasant experience to implement.

**Bootstrap & CSS 3**

To make the true aesthetics of this application shine the technologies of this application will be accompanied with bootstrap the most popular front-end library in the world. There will be a selection of hundreds of templates that are available to use for my liking. The beauty of bootstrap also is that the developer has the ability to modify the template to their free will. Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only. The plan when I start developing the UI is selecting an aesthetic UI which will be edited to fit the theme of my application.

**Version Control**

This project will be interwoven with full Git support. Git is a distributed version control system. Git is similar to other version control systems such as subversion or CVS, but it's distributed. What this means is that if you clone a git project, you have the entire project history. You can commit, branch and tag all on your local machine without interacting with a server at all. If you were working with subversion or another centralized VCS all of your interactions occur with the server. GitHub is useful for many reasons, the main reasons being that there are such as I can save all my work at any time. I can also access my project from any computer if it has Git. It is also interwoven with GitHub which is a great website for storing and displaying projects. This has vast application benefits.

**Main web application architecture**

Web Storm

Web Storm is the main IDE that I will be using for the development of my project. It is a cross-platform IDE primarily for web, JavaScript and Typescript development. Many of Jet Brain’s other IDEs include the feature set of Web Storm via plugins. The main reason I use this is that it is from the same development company as Intelli J which means I am used to the shortcuts from the program. It also has many useful plugins such as auto run tests and instant load up of browser to launch the application. Npm tools are also implemented in project templates so it’s easy to start up a test project.

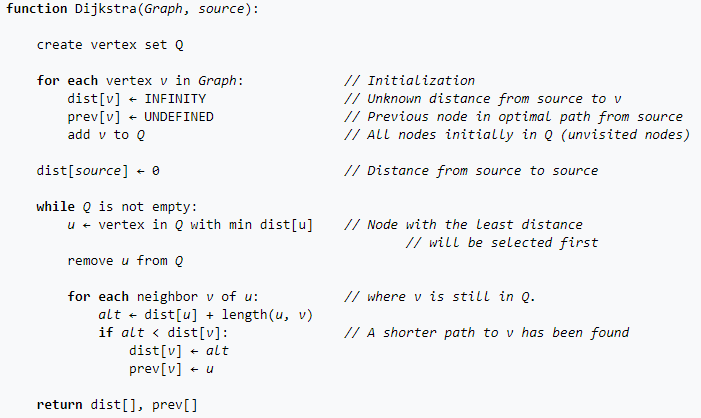
**Dijkstra’s Algorithm**

Dijkstra’s algorithm was the first and main algorithm that I developed for this application. I had a set of goals I wanted to achieve for this and it seems more than likely that they will get accomplished. The goals that will be displayed are the user will have text fields to put in the source and destination node ids in which after they can execute the algorithm. The path that the algorithm calculates from the source to destination will be highlighted in a specific colour which can be modified by the user. The path not taken will be black by default. So, the user can clearly see the path Dijkstra’s algorithm takes. This will happen in step by step iterations (at a speed determined by the user) until the end of the path. There will also be text information of step-by-step instructions on what the algorithm is doing. With my current implementation is quiet minute, this is a positive thing as I wanted this implementation to be as concise and simple as possible which will allow for other developers to see and develop. It obviously has to also work, so I made sure I did detailed manual testing. I even sat down with Mike Sanderson to go through this algorithm to ensure that the appropriate behaviour is displayed. I can confidently confirm that this algorithm does what it says on the tin to do. This was coded in with JavaScript obviously with priority for Node JS. Dijkstra's algorithm computes length of the shortest path from the source to each of the remaining vertices in the graph. [4] It is one of the most popular Greedy algorithm type in existence. This differs greatly from Prim’s and Kruskal’s because it does not create any minimum spanning trees My mind set for creating this to split this in two equal good functions all doing its specialised task.

Steps on how to execute Dijkstra’s algorithm

1. Create a graph by setting up nodes and edges. (Left click and connection of edges together).
2. Choose the starting node that you want to the algorithm to execute from. Every node has a label which is associated with its ID. Put the ID into the input field ‘source node’.
3. Press the ‘Run Dijkstra’s’ to get an initial start of the algorithm till it ends.
4. Optionally start/pause if necessary.

I then researched and planned a pseudo code implementation for Dijkstra’s before I started writing any code. A simple implementation of Dijkstra’s is displayed here below. Taken screenshot to save space.



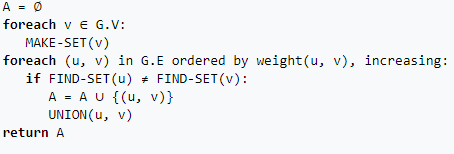
**Kruskal’s Algorithm**

Kruskal’s algorithm was the second and main algorithm that I developed for this application. A set of goals of course were established for this too. The goals that I aim to have in this algorithm implementation are. The algorithm will simply get executed by a button click. This will show the paths from lowest edge to the highest until it explores the whole graph. The path that the algorithm calculates from the source to destination will be highlighted in a specific colour which can be modified by the user. The path not taken will be black by default. There will also be text information of step-by-step instructions on what the algorithm is doing. Highlight of what the Minimum Spanning Tree is will be highlighted too. Currently, this is the algorithm with the longest amount of coding lines however this is expected as the functionality is more complex than the other two algorithms, them being Dijkstra’s and Prim’s algorithms. For my specific implementation to work with what I want to do I have to use an external library called Tree.js. Similar to Dijkstra’s, it is a greedy algorithm. The difference being, it finds a Minimum Spanning Tree for a connected weighted graph adding increasing cost arcs at each step. [10]. Usages will be Network Design in fields such as telephone, electrical, computer and roads. My goal will be to implement a simple design of Kruskal’s algorithm. The time complexity for Kruskal’s Algorithm is O (E log V) where E is the number of edges and V is the number of vertices. This is a library responsible for representing a Tree data structure from arrays in JavaScript. With this approach, it is possible to finalise Kruskal’s algorithm with fundamental languages and execute the program. I have decent time complexity for this algorithm running at O log (n) however there could be improvements to be done that can cause my algorithm to run faster than specialised before. However, this won’t be the ultimate priority as with this current implementation most modern computers and phones can execute it.

Steps on how to execute Kruskal’s algorithm

1. Create a graph by setting up nodes and edges. (Left click and connection of edges together).
2. Simply click ‘Run Kruskal’s’ algorithm to automatically run the algorithm.
3. Optionally pause if necessary.
4. Click ‘Run Kruskal’s if user wants to run again’.

The equivalent of its pseudocode is displayed below too for Kruskal’s algorithm. This is the starting point of the way on how to implement this.



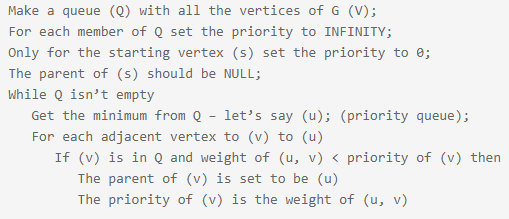
**Prim’s Algorithm**

Lastly, Prim’s algorithm was the last and main algorithm that I developed for this application. Another set of course were laid out for it to functional as well as the others. These goals are the algorithm will have a text field for the user to put in a designated node id for the algorithm to start executing from. After that, the algorithm will show the path from the start to the finish. The path that the algorithm calculates from the source to destination will be highlighted in a specific colour which can be modified by the user. The path not taken will be black by default. There will also be text information of step-by-step instructions on what the algorithm is doing. Highlight of what the Minimum Spanning Tree is will be highlighted too. I am definitely happy with the way Prim’s algorithm is implemented. With just 70 lines my prims.js does exactly what it has to do at efficient time complexities. Another aspect I am happy about is that I use the implementations of functional languages such as Scala. This is because the latest JavaScript actually contains functional methods such as ‘map’ and ‘for each’. For writers, a random sentence can help them get their creative juices flowing. Since the topic of the sentence is completely unknown, it forces the writer to be creative when the sentence appears. This algorithm is very similar to Kruskal’s algorithm as they both result in a minimum spanning tree. There also have the same usages as Kruskal’s. Everything Kruskal’s can do, Prim’s can also do apart from very edge case conditions. However, there are a few differences between Kruskal’s and Prim’s algorithm. Firstly, Kruskal’s algorithm starts with an edge but Prim’s algorithm starts with a vertex. Secondly, Prim’s algorithm selects the edge to nodes that are connected to each other whereas Kruskal’s selects edges in a haphazard way that does not always have to be connected. There is also a difference in time complexity. This is used everywhere in this file to ensure non mutability and good programming styles. Did I realise that the champion of functional programming was in JavaScript? I just did with this. With the basis of this are just basic loops with if conditions. It took some time to think about the design that will be necessary but I stayed vigilant and conquered it in the end.

Steps on how to execute Prim’s algorithms

1. Create a graph by setting up nodes and edges. (Left click and connection of edges together).
2. Choose the starting node from where the algorithm will start parsing the edges.
3. Simply click ‘Run Prim’s algorithm to automatically run the algorithm.
4. Optionally pause if necessary.

I will display the pseudocode counterpart for Prim’s algorithm below. Again, this was my starting point of highlighting the logic needed for my program.



**Graph Tool features**

Click to add

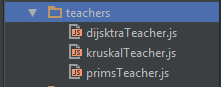
Significant time has been spent to try and make this application as user initiative and friendly as possibly. So to meet this criteria I implemented a good feature were if a user wants to add a node, they can easily just click where and it will get added. This saves the user from trying to find out how sort this out and they can quickly get into the application and start exploring.

Random graph on click

If a user does not want to actually create a specific at all, I also implemented the feature of generating a random graph for the user if they wish.

Teachers

I have a module of teachers which is responsible for displaying the informative text to the user about what is happening when the algorithms are running.



I currently have a teacher for every algorithm that I have implemented. They will also vary by functionality because reasons. For example Prim’s and Kruskal’s will want to know if there is a minimum spanning tree created were as Dijkstra’s is not interested in that information. So they are all implemented in slightly different ways. Does what it says on the tin to do. This was coded in with JavaScript obviously with priority for Node JS. Dijkstra's algorithm computes length of the shortest path from the source to each of the remaining vertices in the graph. It is one of the most popular Greedy algorithm type in existence. A desirable nice feature that I have for this teachers is that as it prints out information for the user to follow as the algorithm runs. This gives a user another visual aid of what is going on.