

Algorithms Visualizer for Finding Spanning Trees and Paths

*A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of*

Bachelor of Technology

by

K. Sai Deekshith
(111701025)



INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CERTIFICATE

*This is to certify that the work contained in the project entitled “**Algorithms Visualizer for Finding Spanning Trees and Paths**” is a bonafide work of **K. Sai Deekshith (Roll No. 111701025)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

Krithika Ramaswamy

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

I would like to express my gratitude to my project mentor Dr. Krithika Ramaswamy for the support and guidance provided to carry out the project. I want to thank my friends Vaibhav Jindal and Drisya P for there help.

Contents

List of Figures	iv
List of Tables	v
1 Introduction	1
1.1 Organization of The Report	2
2 Review of Prior Works	3
2.1 Conclusion	3
3 Eulerian Trail	5
3.1 Undirected Graph	5
3.2 Directed Graph	6
3.3 Algorithm	7
3.4 Proof of correctness	9
3.5 Time Complexity	9
3.6 Conclusion	9
4 Implementation	11
4.1 React Components	11
4.1.1 App.jsx	11
4.1.2 Home.jsx	11

4.1.3	MainPage.jsx	11
4.1.4	EulerianTrail.jsx	12
4.1.5	draw-scene.js	12
4.1.6	NewCircle.js	12
4.1.7	NewEdge.js	12
4.2	Illustration	12
5	Tools and Technologies	17
5.1	Introduction	17
5.2	ReactJS	17
5.3	KendoReact	18
5.4	Node Packages	18
5.5	VS code	18
6	Conclusion and Future Work	19
6.1	Conclusion	19
6.2	Future Work	19
	References	21

List of Figures

4.1	Initial Page	13
4.2	Added vertices	13
4.3	Added edge	14
4.4	removed edge	14
4.5	Error on more than one connected component	15
4.6	Visualization of the algorithm	15
4.7	Final result	16

List of Tables

Chapter 1

Introduction

The aim of the project is to visualize the implementation of the some of the important algorithms like *Eulerian Trail* , *Prim's Algorithm*, *Kruskal' Algorithm*, *Hamiltonian Path* in various graphs and also *Longest Path in directed acyclic graph* and data structures involved in the implementation of the algorithms. It is useful as a teaching aid in the classroom to make it easy for the students to understand. It is useful to explain the time complexity, space complexity and correctness of the algorithms. The main features of the Visualizer are:

- Detailed data structure updates during algorithm execution
- Step-by-step intermediate algorithmic stages leading to final output
- Visualisation of key correctness arguments wherever possible
- Visualization of time and space utilization
- Support for dynamic inputs

For this term Eulerian Trail for any given graph is implemented.

1.1 Organization of The Report

This report consists of 6 chapters. chapter 1 is the introduction about the project and the goal. Chapter 2 discusses with the prior work. Chapter 3 explains the algorithm and its correctness. Chapter 4 deals with the implementation of the algorithm and visualization of the algorithm. Chapter 5 discusses the tools and technologies used in the development of the project. Chapter 6 is the conclusion and future work.

Chapter 2

Review of Prior Works

There are algorithm visualizer available for free online. Data Structure Visualisations by David Galles, University of San Francisco, has implementations of sorting algorithms, dynamic algorithms, and graph algorithms. Graph online which provides with visualization of some of the graph algorithms. But the Graph online only visualizes the final result but not implementation. 'Visual Algo' is another visualizer for sorting algorithms, graph algorithms

2.1 Conclusion

Most of the visualizers online does not provide the visualization of implementation of algorithm. Even if they provide there are only system provided inputs rather than the user. They lack to provide correctness statements, pseudo code, etc.

Chapter 3

Eulerian Trail

Eulerian Trail or Eulerian Path is defined as a path in a graph such that it passes through all the edges of the graph exactly once. Any graph which has Eulerian trail is called traversable graph and also Semi-Eulerian. A graph is called Eulerian if it has an Eulerian Cycle. We can find whether a given graph has a Eulerian Trail or not in polynomial time.

3.1 Undirected Graph

Any undirected graph must satisfy the following theorems to have an eulerian path.

Theorem 1: *The graph must be connected.*

Proof : Consider there are two connected components in the graph. We cannot reach the edges of one connected component from another. Hence the graph should be connected.

Theorem 2: *All the vertices in the graph except zero or 2 must have even degree.*

Proof : Consider there are more than two vertices with odd degree.

1. Start at an odd degree vertex.

Let v be vertex with odd degree where the path begins. After leaving the vertex from one edge we are having even degree at vertex v . Half of the edges are used to return to the vertex and the other are used to leave. As we start by entering the vertex when

there are even number of edges finally we will leave the vertex and hence we cannot end the path at that vertex.

Now consider the vertices with odd degree other than v . When we enter a vertex with odd degree the degree of that becomes even. Now half of the edges are used to leave and the other half are used to enter. Hence the last edge of that vertex is used to enter. Now if we have more than one odd degree vertex (other than v) then when we enter an odd degree vertex with its last edge we cannot leave that vertex but there are edges of other odd degree vertices which are not traversed. Hence no eulerian path.

2.Start at an even degree vertex.

If we start at an even degree vertex once we enter an odd degree vertex with only one edge remaining we cannot leave that edge but there are edges at other odd degree vertices which are not traversed. Hence there is no eulerian trail. Hence to have an Eulerian Path there should be either zero or 2 vertices with odd degree and all other verices must have even degree.

3.2 Directed Graph

Any directed graph must satisfy the following theorems to have an eulerian path:

Theorem 3: *The graph must be connected.*

Proof : Consider there are two connected components in the graph. We cannot reach the edges of one connected component from another. Hence the graph should be connected.

Theorem 4: *All the vertices except 2 in the graph must have equal in-degree and out-degree. One of the remaining 2 must have in-degree one greater than out-degree (end vertex) and the other must have out-degree one greater than the in-degree (start vertex).*

Proof : If the in-degree and out-degree of a vertex is equal then the total degree of the vertex is even. If the in-degree and out-degree is not equal then the total degree is odd.

From the proof of theorem 2 we can state that if there are more than two vertex with odd degree then there is no eulerian path. Hence if there are more than two vertices with in-degree not equal to out-degree then there is no eulerian path.

3.3 Algorithm

Pseudo Code: Here the graph g is described as $g[\text{vertex}] = [\text{vertex}, \text{edgeID}]$;

```
function checkPathPresentDirected() {  
    oddVertex = 0;  
    start = 0, end;  
    for (i = 0; i < |V|; i += 1) {  
        if (outDeg[i] - inDeg[i] !== 0) {  
            if (outDeg[i] > inDeg[i]) {  
                start = i;  
            }  
            else {  
                end = i;  
            }  
            oddVertex += 1;  
        }  
    }  
    return {odd: oddVertex, start: start};  
}
```

```
function checkPathPresentUnDirected() {  
    oddVertex = 0;  
    start = 0;  
    for (i = 0; i < idCount; i += 1) {  
        if (deg[i] % 2 !== 0) {
```

```

        start = i;
        oddVertex += 1;
    }
}

return {odd: oddVertex, start: start};
}

```

```

function Trail(v){
    for i = 0 to outDegree[v] {
        edge = g[v][i];
        to = edge[0];
        id = edge[1];
        if (!edgeVisited[id]) {
            edgeVisited[id] = true;
            Trail(to);
        }
    }
    trail.push(v);
}

```

Here the vertices traverse order is stored in trail. The order of the traversal is in the reverse order. Hence the result is the reverse of the stored in the trail.

The starting vertex is the vertex with indegree greater than the outdegree by 1 for directed graphs and vertex with odd degree for undirected graph. First we need to check if there is path. For directed using `checkPathPresentDirected` we get the number of odd degree vertices and the start vertices. If the number of odd degree vertices is not 2 then there is no path. Else the start vertex is start.

For undirected using `checkPathPresentUnDirected` we get the number of odd degree

vertices and the start vertices. If the number of odd degree vertices is not 2 then there is no path. Else the start vertex is start.

3.4 Proof of correctness

In theorem 2 and theorem 4 we already proved that if the number of odd degree vertices is not 2 then there is no eulerian trail.

Now we started traversing the graph at vertex with odd degree. For directed graph the vertex with outdegree 1 greater than indegree. Since the outdegree is greater than indegree whenever we enter that vertex we will always leave. One other vertex has indegree 1 greater than the outdegree. Hence once we enter the vertex there will be a case where there is no out edge. Hence it is final vertex in the path. Since all other vertices have even degree when we enter a vertex there is always another edge to leave.

In the algorithm we are pushing the vertex into the trail when there is no edge out of the vertex. Hence the first vertex that is pushed into the trail is vertex with odd degree. As we are backtracking we will traverse in reverse order hence the vertex going to be pushed will have an edge to the previous edge. Since the graph has only one connected component all the edges can be visited. Hence proved.

3.5 Time Complexity

As the algorithm is a direct implementation of the DFS algorithm. The time complexity is $O(V+E)$.

3.6 Conclusion

In this chapter, we discussed Eulerian Trail and the algorithm to find the Eulerian Trail, its proof and the time complexity.

Chapter 4

Implementation

4.1 React Components

As we are using ReactJS there are many components which are being used to render the graph and visualise.

4.1.1 App.jsx

This is the very first component of the project which loads the home page and Main page.

4.1.2 Home.jsx

This page has brief description of the project.

4.1.3 MainPage.jsx

This page has the Short Description of what Eulerian Trail is. It also takes the user inputs and send it to the child component EulerianTrail.jsx . It also provides buttons to Start, Pause and Resume the visualization.

4.1.4 EulerianTrail.jsx

This page takes the input from the parent and sends the data to draw-scene.js which is used to draw the graph on the UI. When start button is clicked the edges and verices are passed to a function Eulerian Trail where the path is computed and the result is sent. Finally the graph is visualized.

4.1.5 draw-scene.js

This file consists of a function which takes edges and vertices of the graph and uses NewCircle and NewEdge functions to draw the graph.

4.1.6 NewCircle.js

This file consists of a function which is used to draw an new cirle with the given colour and id.

4.1.7 NewEdge.js

This file consists of a function which is used to draw edge between two vertices given the coordinates of the vertices and the colour of the edge.

4.2 Illustration

Initial page of the visualizer:

The visualizer also supports pausing and resuming of the visualization.

Eulerian Path

Eulerian Path is a path in graph that visits every edge exactly once. The conditions for the graph to contain an eulerian path are :

- A directed graph has an Eulerian trail if and only if at most one vertex has $(\text{out-degree}) - (\text{in-degree}) = 1$, at most one vertex has $(\text{in-degree}) - (\text{out-degree}) = 1$, every other vertex has equal in-degree and out-degree.
- all of vertices of graph with nonzero degree belong to a single connected component of the underlying undirected graph.

Eulerian Path

Number of Vertices:

Add Edge:

Remove Edge:

Fig. 4.1 Initial Page

Number of Vertices:

Add Edge:

Remove Edge:

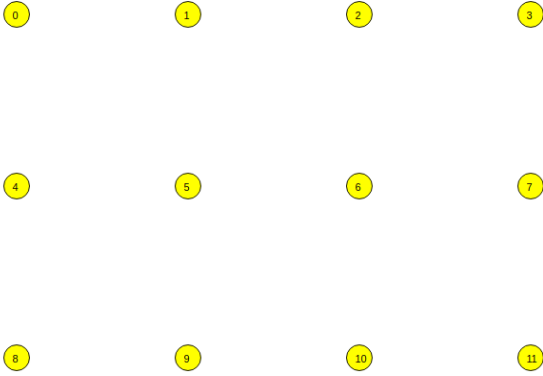


Fig. 4.2 Added vertices

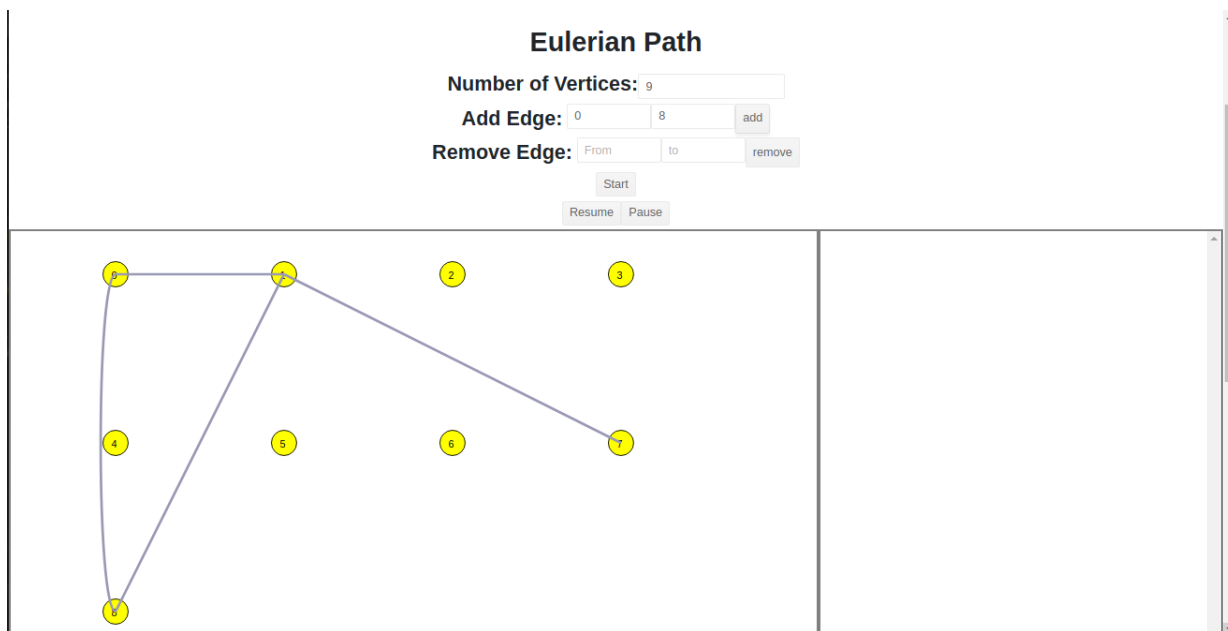


Fig. 4.3 Added edge

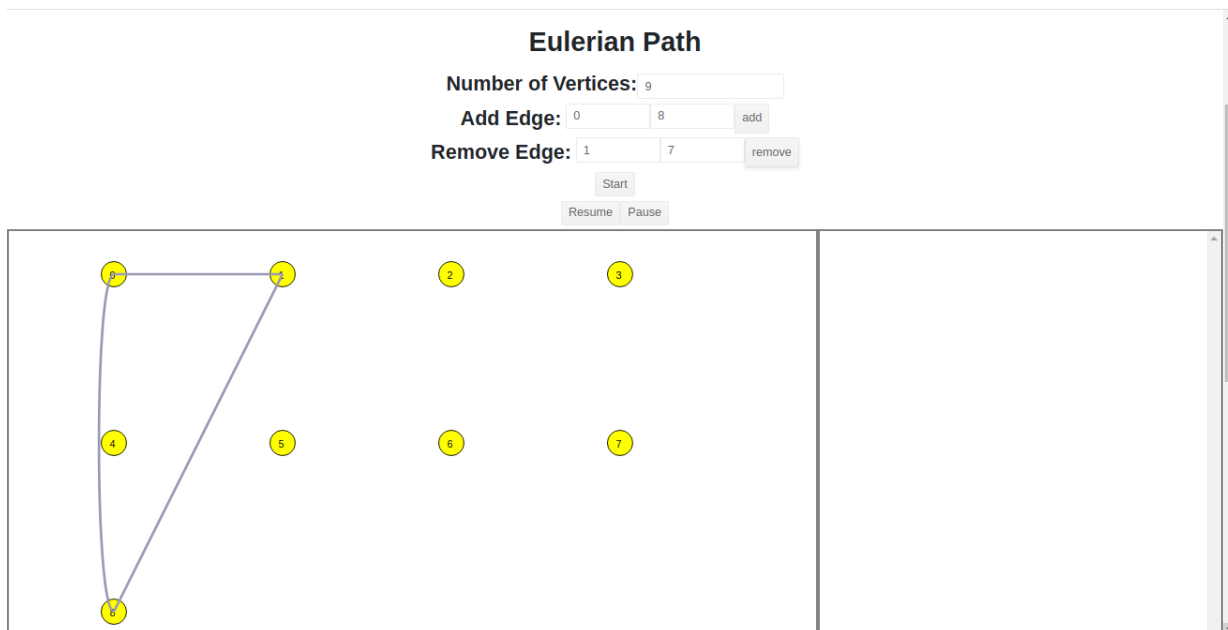


Fig. 4.4 removed edge

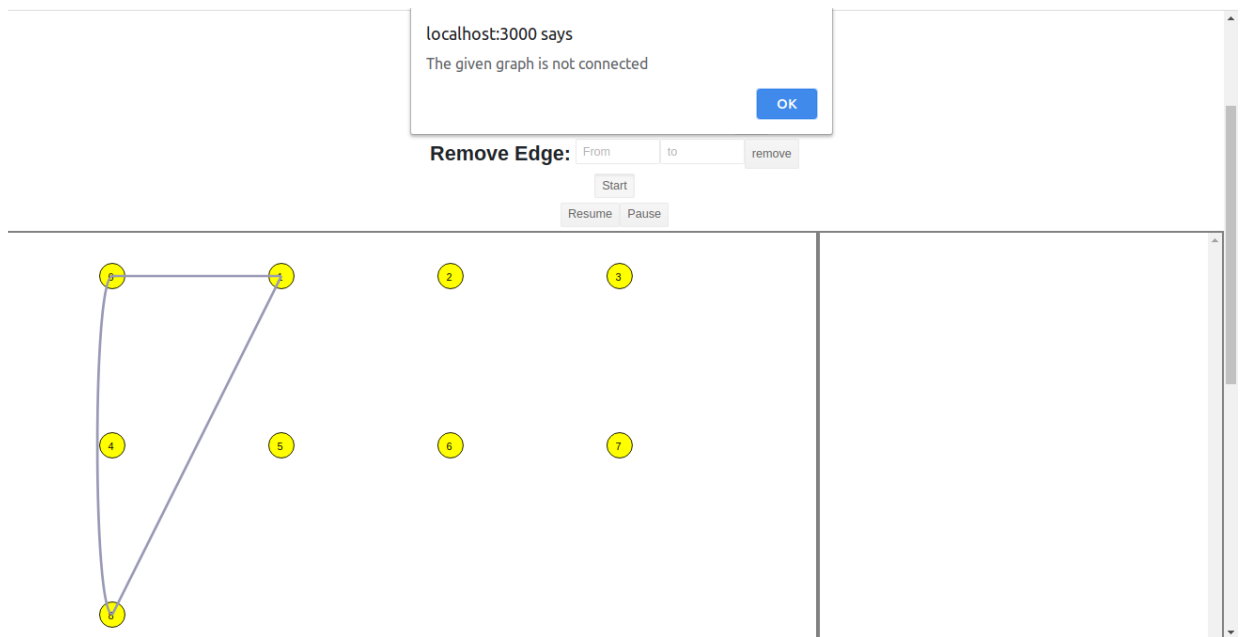


Fig. 4.5 Error on more than one connected component

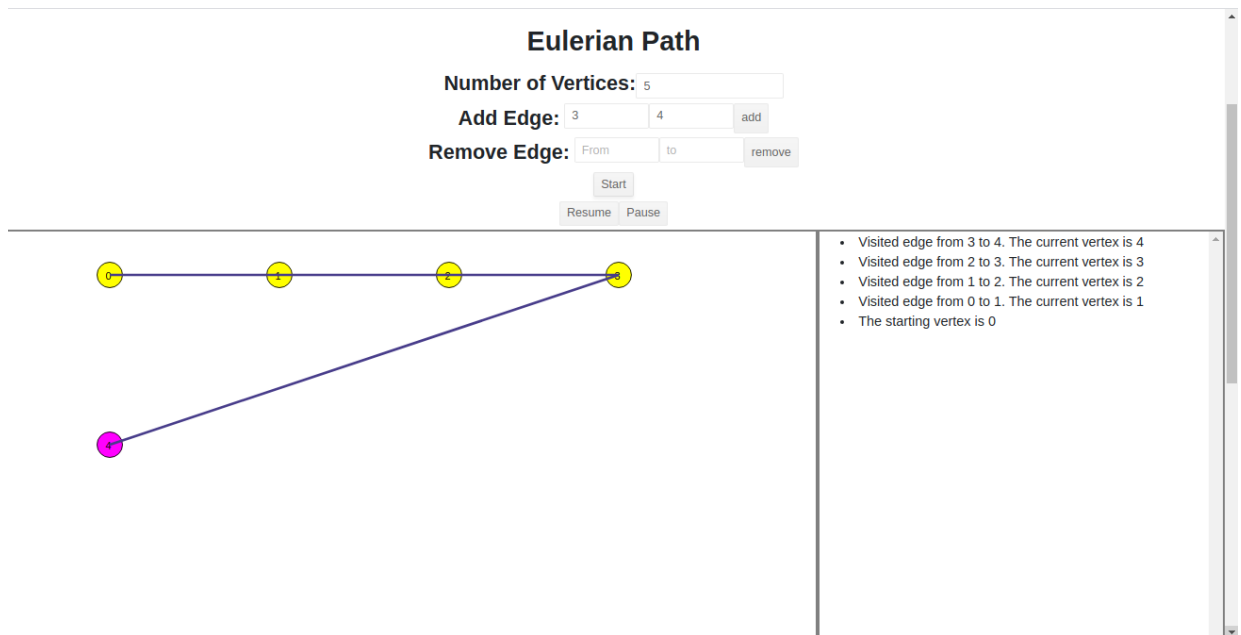


Fig. 4.6 Visualization of the algorithm

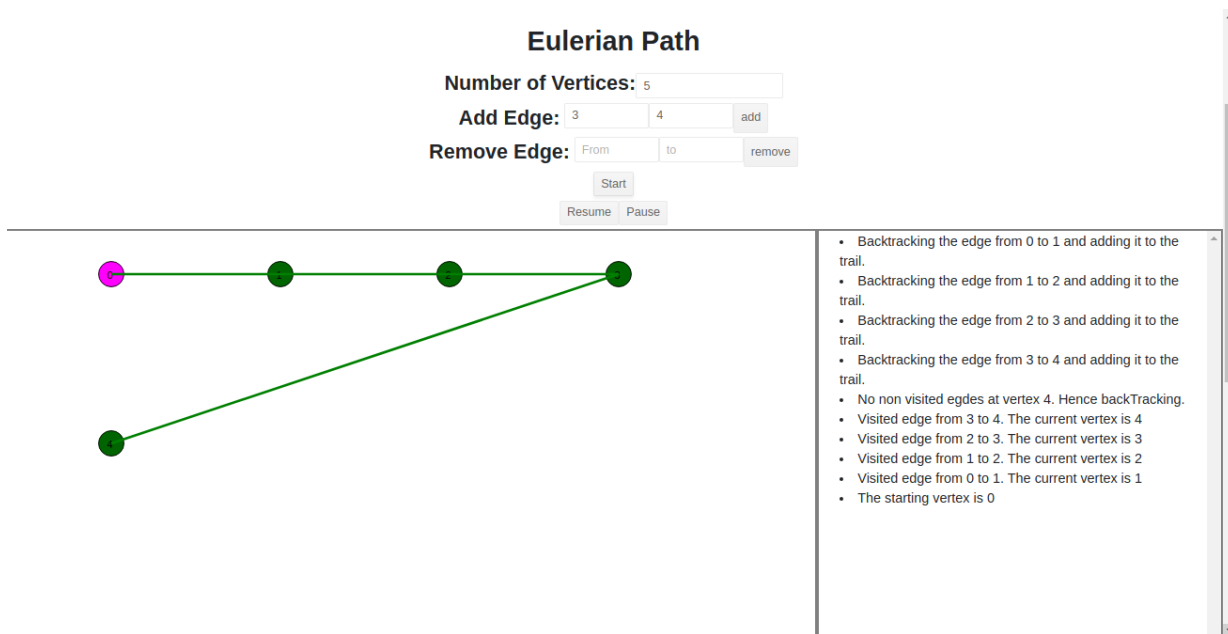


Fig. 4.7 Final result

Chapter 5

Tools and Technologies

5.1 Introduction

In this chapter we are going to discuss the technologies used in the development of the project.

5.2 ReactJS

React is a Javascript library developed by Facebook. It is very useful in development of single page based applications. In React a single UI page is divided into multiple components. Whenever there is an update the component related to that update will only be rendered. Hence it is more efficient by preventing loading of unnecessary components. Every component has its own state and when the state is updated the component is rendered.

React components return JSX (JavaScript XML) code which is used to render over UI. Any web page associated with HTML follows DOM (Document Object Model). It is a tree like structure where there are nodes and children to that nodes. React uses the concept of Virtual DOM where each node is a component. When there is change in one node it will be reflected in all of its children. Hence the node and the subtree at that node will get rerendered.

5.3 KendoReact

KendoReact is a professionally developed commercial library of UI components built specifically for React. It consists of different tools which helps in easy visualization of the vertices, edges and different shapes. It also consists of the style sheet which can be modified accordingly and incorporate in the UI.

5.4 Node Packages

Node packages or NPM packages were used for the responsiveness and styling of the UI.

- KendoReact Drawing package to draw vertices and edges.
- Bootstrap and CSS packages for styling and responsiveness of the UI.
- React related packages for better functioning.

5.5 VS code

In development of the project the editor used is Visual Studio Code. It provides with several useful extensions. Kendo UI Template Wizard is one of the extension which is used to build a basic project with required number of view pages along with our own choice of styling.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This report explains the visualizer for Eulerian Trail. It clearly states what Eulerian Trail is, algorithm to find Eulerian Trail in both directed and undirected graphs. It states step by step procedure on how to use the visualizer. It provides with statements related to what currently is being visualized. This provides more clarity to students if they do not understand the visualizer. It helps students to understand the time complexity and the intuition behind the proof of correctness of algorithm.

6.2 Future Work

- To provide the pseudo code and highlight the line currently executing.
- Validation of the input.
- To provide more options to users such as the speed adjustment, next step, prev step etc.
- Include more algorithms.

References

- [1] M. Poppe. Eulerain trail. [Online]. Available: <https://github.com/mauriciopoppe/eulerian-trail>
 - [2] Telerik. Kendo react. [Online]. Available: <https://www.telerik.com/kendo-react-ui/components/drawing/>
 - [3] Graph magic. [Online]. Available: <http://www.graph-magics.com/articles/euler.php>
 - [4] Reactjs. [Online]. Available: <https://www.w3schools.com/react/>
- [1] [2] [3] [4]