Shortest Path

Simulator

June 2021

# Project Analysis

## Description

Schools and colleges around the world must deal with a common problem, managing student travel times in between classes.

Some schools deal with this problem by choosing an arbitrary number of minutes in between lessons, 3-5 minutes. However, in most cases these times do not reflect real travel times between classrooms, especially in large campuses.

Other schools choose not to handle this issue at all, leaving the responsibility of students getting to lessons on time in the hands of individual teachers.

This program allows users to create graph diagrams and find the shortest path between any two given nodes. The nodes represent student destinations, such as classrooms or public facilities, and the weight of the edges represent the average travel time between those nodes.

Using this program, new and existing schools and colleges can create correct maps showing the preferred routes students are expected to take while traveling in between lessons. This can help in creating reasonable student timetables and decreases cases where sincere students are blamed for arriving late to classes.

## Research

There are two main paths I could take this project, and that is building a desktop application or a web application.

To build a desktop application, some options for a programming language I could use include C#, Python and Java. As for a web application, my options include PHP, ASP.NET, Python and JavaScript.

PHP, Python and ASP.NET are great options for creating server-side applications, whereas JavaScript is preferred for creating client-side applications, since this is the scripting language web browsers use, and JavaScript allows for direct manipulation of the DOM (the structure of a web page). Additionally, JavaScript can be used to serve web pages by using NodeJS, a JavaScript runtime (NodeJS, 2009).

Desktop applications run offline and are generally more secure than web applications. However, web applications are more accessible, since clients don’t need to install a program onto their computer, and they can run on any machine with a browser and an internet connection. This also makes web applications cost-efficient, since they only need to be written once to run on Windows, Mac, Linux, Android and iOS.

React JS is a JavaScript library for building user interfaces (React, 2021). React is great for handling complex state and creating reusable components. Additionally, TypeScript can be used in React to add type checking, which would make the code more reliable and robust. TypeScript is an open-source language which builds on JavaScript (Microsoft, 2012).

Graph Online is an existing web application that allows users to build graph networks and run algorithms, such as Dijkstra's shortest path algorithm on them. The functionality provided by Graph Online is similar to what I am trying to achieve in this project, and so this makes it a good frame of reference.

## Model

Here is an example of a weighted, undirected graph. The edges between the nodes have a weight, representing the average travel time between the nodes it connects.

|  |
| --- |
| Example of a Graph |
| Figure 1. Example of a graph |

## Objectives

1. Create a traversable graph object
   1. Use object orientated programming to create an object representing a graph and its nodes and edges.
   2. Create new nodes and edges objects.
   3. Edit existing nodes and edges.
   4. Delete nodes and edges.
   5. Graph should be traversable; its nodes should point to all its adjacent nodes, so that you could create a path of adjacent nodes.
2. Find the shortest path between any two given nodes
   1. Select the start and end nodes.
   2. Use Dijkstra's shortest path algorithm to find the shortest path between the start and end nodes.
   3. Return the set of nodes that make up the shortest path in order.
3. User Interface
   1. User should be able to create and manipulate graphs
   2. Graph should be always represented in a diagram
   3. User can run algorithm and see results on the graph
4. Create a graph creation tool
   1. User interface to create nodes and edges.
   2. Set names of nodes.
   3. Set weights of edges.
5. Represent graph in a diagram
   1. Navigable SVG diagram container
   2. Diagram can be panned, zoomed
   3. Diagram contains grid
   4. Diagram elements can snap to grid
   5. Use SVG components to represent the graph.
   6. Allow user to manipulate the layout of the graph by dragging and moving.
   7. Show the shortest path running the algorithm

## Critical path

1. Graph node objects
2. Graph edge objects
3. Create a traversable graph object
4. Apply Dijkstra’s algorithm to graph’s first and last nodes
5. Find shortest path between any given start and end node
6. Design user interface layout
7. Design graph creation menus
8. Design algorithm menu
9. Design diagram options menu
10. Create a navigable diagram
11. Create reusable diagram element components
12. Show graph on diagram
13. Show graph’s shortest path on diagram

Project Design

**High-level Overview**

On start, the user will be presented with an interface where they can create a new graph diagram, made up of nodes and edges, from scratch. The graph can be manipulated by moving the nodes around to create a layout.

|  |
| --- |
|  |
| Figure 2. High level overview of application |

The user has access to an additional menu where they can select a start and end node to run the shortest path algorithm on.

Dijkstra’s algorithm is used to find the shortest path between these nodes. The result of this algorithm, distance, and direction, will be presented to the user in a menu. Additionally, the shortest path will be highlighted on the graph for the user to see.

*Fig. 2 (left)*, shows a simple high-level flowchart overviewing the basic functionality of the application.

|  |
| --- |
|  |
| Figure 3. Use case diagram |

## Use-Case Diagram

*Fig. 3* *(right)* is a UML use-case diagram of the application. This shows the potential use-case scenarios a user can experience while using this application.

For the purposes of this shortest path simulator, the use-case is quite straightforward. A user can create a graph, manipulate it and see the shortest path between two nodes.

## User Interface

The user interface for this application will have a standard 3 column layout. There will be 2 sidebars and 1 main viewport. *Fig. 4 (below)* is a mockup of how I want the UI to look. I created this using Figma, a designing tool.

|  |
| --- |
|  |
| Figure 4. User Interface Design |

## Pseudocode

### Node Distances Object

Here is the pseudocode for a class that will handle mapping distances to individual graph nodes.

01: # Object that will store distance values for array of graph nodes

02: class NodeDistances

03: public store

04:

05: # Map each node to a default distance of Infinity

06: public procedure new(nodes)

07: store = array[nodes.length]

08: for i=0 to nodes.length

09: node = nodes[i]

10: store[i] = { node: node, distance: Infinity }

11: next i

12:

13: public function get(node)

14: distance = null

15: for i=0 to store.length

16: if node == store[i].node

17: distance = store[i].distance

18: next i

19: return distance

20:

21: public procedure set(node, newDistance)

22: for i=0 to store.length

23: if node == store[i].node

24: store[i].distance = newDistance

25: next i

26:

27: # Get node with shortest distance to start excluding any visited nodes

28: public function getShortestDistanceNode(visited)

29: shortest = null

30: for i=0 to store.length

31: if store[i] in visited

32: endfor

33: if shortest == null

34: shortest = store[i]

35: if store[i].distance < shortest.distance

36: shortest = store[i]

37: next i

38: return shortest

### Find Shortest Path Function

This is the main algorithm function that will use Dijkstra's algorithm to find the shortest path between two nodes.

01: function findShortestPath(graph, startNode, endNode)

02: # Set up algorithm

03: distances = new NodeDistances(graph.nodes)

04: array visited[]

05: array parents[]

06: distances.set(startNode, 0)

07: node = distances.getShortestDistanceNode(visited)

08:

09: # Loop through unvisited nodes

10: while node != null

11: distance = distances.get(node)

12: edges = graph.getNodeEdges(node)

13: for i=0 to edges.length

14: child = edges[i].node

15: newDistance = distance + edges[i].weight

16: currentDistance = distances.get(child)

17: if newDistance < currentDistance

18: distances.set(child, newDistance)

19: parents.add(node)

20: visited.add(node)

21: node = distances.getShortestDistanceNode(visited)

22: endwhile

23:

24: # Gather results

25: distance = distances.get(endNode)

26: array path[]

27: parent = parents[endNode]

28: while parent != null

29: node = parents[parent]

30: path.add(node)

31: parent = node

32: endwhile

33: return { distance: distance, path: path }

# Implementation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reference | Page | File | Line | Notes |
| Objective 1 : Create a traversable Graph | 15 | Graph/Graph.ts | 4 | Define a Graph object that represent a graph of nodes and edges. Has methods to manipulate the graph’s structure. |
| Objective 2 : Find the shortest path between two nodes | 13 | Graph/algorithm.ts | 5 | Use Djikstra’s algorithm to iterate through nodes by their distances from the identified start node. Returns an object with an array of GraphEdge objects which make up the shortest path, and the total distance from start to end as a number. |
| Objective 2b : Store node distances | 12 | Graph/algorithm.ts | 24 | Create a class that inherits the InMemoryKeyValueStore class defined in storage.ts. A NodeDistances object maps each node in a graph to its distance from start. Additionally, there is a method that will return the unvisited node with the shortest distance given the current algorithm state. |
| Objective 3b : Graph should be represented on Diagram | 44 | User Interface/algorithm.ts | 8 | This is a functional React component which also contains two more functions. One maps the nodes in a graph to SVG DiagramNode components. The other maps the edges in the graph to SVG DiagramEdge components. The parent Diagram component represents the current state of the graph as an SVG contained in an HTML Div element. |
| InMemoryKeyValueStore | 85 | Utility/storage.ts | 1 | Polymorphism is used to create a generic object which can store key value pairs in memory. Used in algorithm.ts |

## Graph

### File: algorithm.ts

001: import { Graph } from "./Graph";

002: import { GraphNode } from "./GraphNode";

003: import { GraphEdge } from "./GraphEdge";

004: import { InMemoryKeyValueStore } from "../util/storage";

005:

006: export interface ShortestPathResult {

007: distance: number;

008: path: GraphEdge[];

009: }

010:

011: class NodeParents extends InMemoryKeyValueStore<

012: GraphNode,

013: { node: GraphNode; edge: GraphEdge } | null

014: > {

015: /\*\*

016: \* Init a new NodeParents KeyValue store.

017: \* Map out existing nodes with default values of null.

018: \*/

019: constructor(nodes: GraphNode[]) {

020: super({ keys: nodes, defaultValue: null });

021: }

022: }

023:

024: class NodeDistances extends InMemoryKeyValueStore<GraphNode, number> {

025: /\*\*

026: \* Init a new NodeDistances KeyValue store.

027: \* Map out existing nodes with default values of Infinity.

028: \*/

029: constructor(nodes: GraphNode[]) {

030: super({ keys: nodes, defaultValue: Infinity });

031: }

032:

033: /\*\*

034: \* Return node with shortest distance, excluding visited nodes

035: \*/

036: getShortestDistanceNode(visited: GraphNode[]) {

037: const unvisited = this.store.filter(

038: (obj) => !visited.includes(obj.key)

039: );

040:

041: if (!unvisited.length) return null;

042:

043: const shortest = unvisited.reduce((prev, current) =>

044: current.value < prev.value ? current : prev

045: );

046:

047: return shortest?.key;

048: }

049: }

050:

051: // class NodeParents extends InMemoryKeyValueStore<GraphNode, {node: GraphNode, edge: GraphEdge}> {

052:

053: // }

054:

055: export function findShortestPath(

056: graph: Graph,

057: startNode: GraphNode,

058: endNode: GraphNode

059: ) {

060: // Setup algorithm

061: const distances = new NodeDistances(graph.nodes);

062: const visited: GraphNode[] = [];

063: const parents = new NodeParents(graph.nodes);

064: // const parents = new InMemoryKeyValueStore<GraphNode, GraphNode | undefined>(

065: // {

066: // keys: graph.nodes,

067: // defaultValue: undefined,

068: // }

069: // );

070:

071: distances.set(startNode, 0);

072: let current = distances.getShortestDistanceNode(visited);

073:

074: // Iterate through nodes in order of shortest distance from start

075: while (current !== null) {

076: const node = current;

077: const distance = distances.get(node)!;

078: const edges = graph.getNodeEdges(node);

079:

080: edges.forEach((edge) => {

081: // Get other node from edge

082: const child = edge.nodes.find((n) => n !== node)!;

083:

084: // Prevent infinite loop if child is start node

085: if (child === startNode) return;

086:

087: // Check if new distance is shorter than current

088: const newDistance = distance + edge.weight;

089: const currentDistance = distances.get(child)!;

090: if (newDistance < currentDistance) {

091: // Update distance & parent

092: distances.set(child, newDistance);

093: parents.set(child, { node, edge } ?? null);

094: }

095: });

096:

097: visited.push(node);

098:

099: current = distances.getShortestDistanceNode(visited);

100: }

101:

102: // Obtain result

103: const distance = distances.get(endNode)!;

104: const path = [];

105:

106: // Find find path by using parents

107: let parent = parents.get(endNode);

108: while (parent) {

109: path.push(parent.edge);

110: parent = parents.get(parent.node);

111: }

112: path.reverse();

113:

114: const result: ShortestPathResult = {

115: distance,

116: path,

117: };

118:

119: return result;

120: }

### File: Graph.ts

01: import { GraphEdge } from "./GraphEdge";

02: import { GraphNode } from "./GraphNode";

03:

04: export class Graph {

05: private \_nodes: GraphNode[] = [];

06: public get nodes() {

07: return this.\_nodes;

08: }

09:

10: private \_edges: GraphEdge[] = [];

11: public get edges() {

12: return this.\_edges;

13: }

14:

15: addNode(node: GraphNode) {

16: this.\_nodes.push(node);

17: return this;

18: }

19:

20: addNodes(nodes: GraphNode[]) {

21: nodes.forEach((node) => {

22: this.addNode(node);

23: });

24: return this;

25: }

26:

27: addEdge(edge: GraphEdge) {

28: // Validate edge nodes in graph

29: let errorNodes: GraphNode[] = [];

30: edge.nodes.forEach((node) => {

31: if (!this.nodes.includes(node)) {

32: errorNodes.push(node);

33: }

34: });

35:

36: // Return error if nodes not in graph

37: if (errorNodes.length > 0) {

38: const message = `Graph does not contain node${

39: errorNodes.length > 1 ? "s" : ""

40: }${errorNodes.map((node) => {

41: return ` "${node.name}"`;

42: })}.`;

43:

44: const error = {

45: nodes: errorNodes,

46: message,

47: };

48:

49: throw error;

50: }

51:

52: this.\_edges.push(edge);

53: return this;

54: }

55:

56: addEdges(edges: GraphEdge[]) {

57: edges.forEach((edge) => this.addEdge(edge));

58: return this;

59: }

60:

61: removeNode(node: GraphNode) {

62: this.\_nodes = this.\_nodes.filter((\_node) => \_node !== node);

63: this.\_edges.filter((\_edge) => !\_edge.nodes.includes(node));

64: return this;

65: }

66:

67: removeEdge(edge: GraphEdge) {

68: this.\_edges = this.\_edges.filter((\_edge) => \_edge !== edge);

69: return this;

70: }

71:

72: getNodeEdges(node: GraphNode) {

73: return this.edges.filter((edge) => edge.nodes.includes(node));

74: }

75: }

### File: GraphNode.ts

01: interface Position {

02: x: number;

03: y: number;

04: }

05:

06: /\*\*

07: \* Represents a node in a graph.

08: \*/

09: export class GraphNode {

10: position: Position;

11: name: string;

12: id: string;

13:

14: constructor(name: string, id: string, initialPosition?: Position) {

15: this.name = name;

16: this.id = id;

17: this.position = initialPosition ?? { x: 0, y: 0 };

18: }

19:

20: edit({ name, id }: { name?: string; id?: string }) {

21: if (name) this.name = name;

22: if (id) this.id = id;

23: }

24:

25: move(newPosition: Position) {

26: this.position = newPosition;

27: }

28: }

### File: GraphEdge.ts

01: import { GraphNode } from "./GraphNode";

02:

03: /\*\*

04: \* Represents an edge between 2 nodes

05: \*/

06: export class GraphEdge {

07: private \_nodes: [GraphNode, GraphNode];

08: public get nodes() {

09: return this.\_nodes;

10: }

11:

12: private \_weight: number;

13: public get weight() {

14: return this.\_weight;

15: }

16:

17: constructor(node1: GraphNode, node2: GraphNode, weight: number) {

18: if (node1 === node2) throw new Error("Cannot have edge with one node");

19: if (weight < 0) throw new Error("Cannot have negative value weight");

20: this.\_nodes = [node1, node2];

21: this.\_weight = weight;

22: }

23:

24: edit({ weight }: { weight?: number }) {

25: if (weight) this.\_weight = weight;

26: }

27: }

## Context

### File: graphActions.ts

01: import React from "react";

02: import { GraphNode } from "../GraphNode";

03: import ActionController from "./ActionController";

04: import { GraphEdgeActionMap } from "./graphEdgeActions";

05: import { GraphNodeActionMap } from "./graphNodeActions";

06:

07: export interface GraphActionMap extends GraphNodeActionMap, GraphEdgeActionMap {

08: changeSnap: { type: "changeSnap"; snap: number };

09: findShortestPath: {

10: type: "findShortestPath";

11: start: GraphNode;

12: end: GraphNode;

13: };

14: deselect: { type: "deselect" };

15: }

16:

17: export type GraphAction = GraphActionMap[keyof GraphActionMap];

18:

19: /\*\*

20: \* Handle generic actions

21: \* @param dispatch

22: \* @param action execute action

23: \* @param undoAction revert action

24: \* @returns

25: \*/

26: export function dispatchAction(

27: dispatch: React.Dispatch<GraphAction>,

28: action: GraphAction,

29: undoAction?: GraphAction

30: ) {

31: if (undoAction) {

32: const execute = () => dispatch(action);

33:

34: const undo = () => dispatch(undoAction);

35:

36: ActionController.addAction(execute, undo);

37:

38: return dispatch(action);

39: }

40:

41: dispatch(action);

42: }

43:

44: /\*\*

45: \* Deselect current

46: \*/

47: export const deselect = (dispatch: React.Dispatch<GraphAction>) =>

48: dispatchAction(dispatch, {

49: type: "deselect",

50: });

51:

52: /\*\*

53: \* Change snap value

54: \*/

55: export const changeSnap = (

56: dispatch: React.Dispatch<GraphAction>,

57: snap: number

58: ) =>

59: dispatchAction(dispatch, {

60: type: "changeSnap",

61: snap,

62: });

63:

64: /\*\*

65: \* Run Dijkstra's algorithm on graph

66: \*/

67: export const findShortestPath = (

68: dispatch: React.Dispatch<GraphAction>,

69: start: GraphNode,

70: end: GraphNode

71: ) =>

72: dispatchAction(dispatch, {

73: type: "findShortestPath",

74: start,

75: end,

76: });

### File: graphEdgeActions.ts

01: import { GraphEdge } from "../index";

02: import { GraphAction, dispatchAction } from "./graphActions";

03:

04: export interface GraphEdgeActionMap {

05: selectEdge: { type: "selectEdge"; edge: GraphEdge };

06: addEdge: { type: "addEdge"; edge: GraphEdge };

07: removeEdge: { type: "removeEdge"; edge: GraphEdge };

08: editEdgeWeight: {

09: type: "editEdgeWeight";

10: edge: GraphEdge;

11: weight: number;

12: };

13: }

14:

15: /\*\*

16: \* Select an edge

17: \*/

18: export const selectEdge = (

19: dispatch: React.Dispatch<GraphAction>,

20: edge: GraphEdge

21: ) =>

22: dispatchAction(dispatch, {

23: type: "selectEdge",

24: edge,

25: });

26:

27: /\*\*

28: \* Add a new edge object to graph

29: \*/

30: export const addEdge = (

31: dispatch: React.Dispatch<GraphAction>,

32: edge: GraphEdge

33: ) =>

34: dispatchAction(

35: dispatch,

36: {

37: type: "addEdge",

38: edge,

39: },

40: {

41: type: "removeEdge",

42: edge,

43: }

44: );

45:

46: /\*\*

47: \* Remove an edge from graph

48: \*/

49: export const removeEdge = (

50: dispatch: React.Dispatch<GraphAction>,

51: edge: GraphEdge

52: ) =>

53: dispatchAction(

54: dispatch,

55: {

56: type: "removeEdge",

57: edge,

58: },

59: {

60: type: "addEdge",

61: edge,

62: }

63: );

64:

65: /\*\*

66: \* Edit edge weight

67: \*/

68: export const editEdgeWeight = (

69: dispatch: React.Dispatch<GraphAction>,

70: edge: GraphEdge,

71: weight: number

72: ) =>

73: dispatchAction(dispatch, {

74: type: "editEdgeWeight",

75: edge,

76: weight,

77: });

### File: graphNodeActions.ts

001: import { GraphNode } from "../index";

002: import { GraphAction, dispatchAction } from "./graphActions";

003:

004: export interface GraphNodeActionMap {

005: selectNode: { type: "selectNode"; node: GraphNode };

006: addNode: { type: "addNode"; node: GraphNode };

007: removeNode: { type: "removeNode"; node: GraphNode };

008: moveNode: {

009: type: "moveNode";

010: node: GraphNode;

011: position: { x: number; y: number };

012: };

013: editNodeName: {

014: type: "editNodeName";

015: node: GraphNode;

016: name: string;

017: };

018: editNodeID: {

019: type: "editNodeID";

020: node: GraphNode;

021: id: string;

022: };

023: }

024:

025: /\*\*

026: \* Select a node

027: \*/

028: export const selectNode = (

029: dispatch: React.Dispatch<GraphAction>,

030: node: GraphNode

031: ) =>

032: dispatchAction(dispatch, {

033: type: "selectNode",

034: node,

035: });

036:

037: /\*\*

038: \* Add a new node object to graph

039: \*/

040: export const addNode = (

041: dispatch: React.Dispatch<GraphAction>,

042: node: GraphNode

043: ) =>

044: dispatchAction(

045: dispatch,

046: {

047: type: "addNode",

048: node,

049: },

050: {

051: type: "removeNode",

052: node,

053: }

054: );

055:

056: /\*\*

057: \* Remove node from graph

058: \*/

059: export const removeNode = (

060: dispatch: React.Dispatch<GraphAction>,

061: node: GraphNode

062: ) =>

063: dispatchAction(

064: dispatch,

065: {

066: type: "removeNode",

067: node,

068: },

069: {

070: type: "addNode",

071: node,

072: }

073: );

074:

075: /\*\*

076: \* Move node to new position

077: \*/

078: export const moveNode = (

079: dispatch: React.Dispatch<GraphAction>,

080: node: GraphNode,

081: position: { x: number; y: number }

082: ) =>

083: dispatchAction(dispatch, {

084: type: "moveNode",

085: node,

086: position,

087: });

088:

089: /\*\*

090: \* Edit node name

091: \*/

092: export const editNodeName = (

093: dispatch: React.Dispatch<GraphAction>,

094: node: GraphNode,

095: name: string

096: ) =>

097: dispatchAction(dispatch, {

098: type: "editNodeName",

099: node,

100: name,

101: });

102:

103: /\*\*

104: \* Edit node name

105: \*/

106: export const editNodeID = (

107: dispatch: React.Dispatch<GraphAction>,

108: node: GraphNode,

109: id: string

110: ) =>

111: dispatchAction(dispatch, {

112: type: "editNodeID",

113: node,

114: id,

115: });

### File: graphContext.tsx

01: import React, { createContext, useReducer } from "react";

02: import { ShortestPathResult } from "../algorithm";

03: import { Graph, GraphNode, GraphEdge } from "../index";

04: import { GraphAction } from "./graphActions";

05: import { graphReducer } from "./graphReducer";

06:

07: /\*\*

08: \* Store graph state

09: \*/

10: export type GraphState = {

11: graph: Graph;

12: current?: GraphNode | GraphEdge;

13: snap: number;

14: shortestPath?: ShortestPathResult;

15: };

16:

17: interface IGraphContext {

18: state: GraphState;

19: dispatch: React.Dispatch<GraphAction>;

20: }

21:

22: export const GraphContext = createContext<IGraphContext>(null!);

23:

24: /\*\*

25: \* Provider for context API

26: \*/

27: type GraphContextProviderProps = {

28: children: React.ReactNode;

29: };

30:

31: export function GraphContextProvider({ children }: GraphContextProviderProps) {

32: const graph = new Graph();

33:

34: const initialState = {

35: graph,

36: snap: 50,

37: };

38:

39: const [state, dispatch] = useReducer(graphReducer, initialState);

40: const value = { state, dispatch };

41:

42: return (

43: <GraphContext.Provider value={value}>{children}</GraphContext.Provider>

44: );

45: }

### File: graphReducer.ts

01: import { findShortestPath } from "../algorithm";

02: import { GraphAction } from "./graphActions";

03: import { GraphState } from "./graphContext";

04:

05: export function graphReducer(state: GraphState, action: GraphAction) {

06: switch (action.type) {

07: case "deselect":

08: return { ...state, current: undefined };

09:

10: case "changeSnap":

11: return { ...state, snap: action.snap };

12:

13: case "findShortestPath":

14: const shortestPath = findShortestPath(

15: state.graph,

16: action.start,

17: action.end

18: );

19: return { ...state, shortestPath };

20:

21: case "selectNode":

22: return { ...state, current: action.node };

23:

24: case "selectEdge":

25: return { ...state, current: action.edge };

26:

27: case "addNode":

28: state.graph.addNode(action.node);

29: return { ...state };

30:

31: case "removeNode":

32: state.graph.removeNode(action.node);

33: return { ...state };

34:

35: case "addEdge":

36: state.graph.addEdge(action.edge);

37: return { ...state };

38:

39: case "removeEdge":

40: state.graph.removeEdge(action.edge);

41: return { ...state };

42:

43: case "moveNode":

44: action.node.move(action.position);

45: return { ...state };

46:

47: case "editNodeName":

48: action.node.edit({ name: action.name });

49: return { ...state };

50:

51: case "editNodeID":

52: action.node.edit({ id: action.id });

53: return { ...state };

54:

55: case "editEdgeWeight":

56: action.edge.edit({ weight: action.weight });

57: return { ...state };

58:

59: default:

60: return { ...state };

61: }

62: }

### File: ActionController.ts

01: export class Action {

02: execute: Function;

03: undo: Function;

04:

05: constructor(execute: Function, undo: Function) {

06: this.execute = execute;

07: this.undo = undo;

08: }

09: }

10:

11: class ActionController {

12: actionStack: Action[] = [];

13: undoStack: Action[] = [];

14:

15: /\*\*

16: \* Add an action to the top of the actionStack

17: \* @param execute - Function to apply action

18: \* @param undo - Function to revert action

19: \*/

20: addAction(execute: Function, undo: Function) {

21: const action = new Action(execute, undo);

22: this.actionStack.push(action);

23: }

24:

25: /\*\*

26: \* Revert latest change made, and add change to undoStack

27: \*/

28: undo() {

29: // Get latest action

30: const latestAction = this.actionStack[this.actionStack.length - 1];

31:

32: if (!latestAction) return;

33:

34: // Undo action

35: latestAction.undo();

36: this.actionStack.pop();

37: this.undoStack.push(latestAction);

38: }

39:

40: redo() {

41: // Get latest undone action

42: const latestUndoneAction = this.undoStack[this.undoStack.length - 1];

43:

44: if (!latestUndoneAction) return;

45:

46: // Execute action

47: latestUndoneAction.execute();

48: this.undoStack.pop();

49: this.actionStack.push(latestUndoneAction);

50: }

51: }

52:

53: export default new ActionController();

## User Interface

### File: App.tsx

01: import SidebarContainer from "./components/Sidebar/SidebarContainer/SidebarContainer";

02: import MenubarContainer from "./components/Menu/MenubarContainer/MenubarContainer";

03: import Diagram from "./components/Diagram/Diagram/Diagram";

04: import styles from "./App.module.scss";

05: import SidebarPrimary from "./components/Sidebar/SidebarPrimary/SidebarPrimary";

06: import { GraphContextProvider } from "./graph/context/graphContext";

07:

08: function App() {

09: return (

10: <div className={styles.appLayout}>

11: <MenubarContainer></MenubarContainer>

12: <GraphContextProvider>

13: <SidebarPrimary />

14: <Diagram />

15: <SidebarContainer></SidebarContainer>

16: </GraphContextProvider>

17: </div>

18: );

19: }

20:

21: export default App;

### File: Diagram.tsx

01: import { View } from "@saiefelgebali/react-diagrams";

02: import { useContext } from "react";

03: import { GraphContext } from "../../../graph/context/graphContext";

04: import DiagramEdge from "../DiagramEdge/DiagramEdge";

05: import DiagramNode from "../DiagramNode/DiagramNode";

06: import styles from "./Diagram.module.scss";

07:

08: function Diagram() {

09: const { state } = useContext(GraphContext);

10:

11: const Nodes = () =>

12: state.graph.nodes.map((node, index) => (

13: <DiagramNode node={node} key={index} />

14: ));

15:

16: const Edges = () => {

17: return state.graph.edges.map((edge, index) => (

18: <DiagramEdge

19: edge={edge}

20: path={state.shortestPath?.path.includes(edge)}

21: key={index}

22: />

23: ));

24: };

25:

26: return (

27: <View className={styles.view}>

28: {Edges()}

29: {Nodes()}

30: </View>

31: );

32: }

33:

34: export default Diagram;

### File: DiagramNode.tsx

01: import React, { useContext } from "react";

02: import { GraphNode } from "../../../graph/GraphNode";

03: import { ElementContainer } from "@saiefelgebali/react-diagrams";

04: import { GraphContext } from "../../../graph/context/graphContext";

05: import { moveNode, selectNode } from "../../../graph/context/graphNodeActions";

06: import styles from "./DiagramNode.module.scss";

07:

08: type DiagramNodeProps = {

09: node: GraphNode;

10: };

11:

12: /\*\*

13: \* Represents a graph node on the diagram

14: \* @prop {GraphNode} node

15: \*/

16: function DiagramNode({ node }: DiagramNodeProps) {

17: // The radius of a node's circle

18: const r = 50;

19:

20: // Handle app state

21: const { state, dispatch } = useContext(GraphContext);

22:

23: // Fired while dragging

24: function onDragMove(event: MouseEvent) {

25: // Snap to grid

26: moveNode(dispatch, node, {

27: x: Math.round(event.offsetX / state.snap) \* state.snap,

28: y: Math.round(event.offsetY / state.snap) \* state.snap,

29: });

30: }

31:

32: function onClick(event: React.MouseEvent) {

33: selectNode(dispatch, node);

34: }

35:

36: return (

37: <ElementContainer

38: className={styles.node}

39: classNameDragging={styles.isDragging}

40: onDrag={onDragMove}

41: onClick={onClick}>

42: <circle cx={node.position.x} cy={node.position.y} r={r} />

43: <text

44: x={node.position.x}

45: y={node.position.y}

46: textAnchor='middle'

47: fill='white'

48: fontSize='24px'

49: fontFamily='Arial'

50: dy='8px'>

51: {node.id}

52: </text>

53: </ElementContainer>

54: );

55: }

56:

57: export default DiagramNode;

### File: DiagramEdge.tsx

01: import React from "react";

02: import { useContext } from "react";

03: import { ElementContainer } from "@saiefelgebali/react-diagrams";

04: import { selectEdge } from "../../../graph/context/graphEdgeActions";

05: import { GraphContext } from "../../../graph/context/graphContext";

06: import { GraphEdge } from "../../../graph/GraphEdge";

07: import styles from "./DiagramEdge.module.scss";

08:

09: type DiagramEdgeProps = {

10: edge: GraphEdge;

11: path?: boolean;

12: };

13:

14: function DiagramEdge({ edge, path = false }: DiagramEdgeProps) {

15: const { dispatch } = useContext(GraphContext);

16:

17: function onClick(event: React.MouseEvent) {

18: selectEdge(dispatch, edge);

19: }

20:

21: /\*\*

22: \* Overlay line to increase clickable area

23: \*/

24: return (

25: <ElementContainer

26: className={`${styles.edge} ${path ? styles.path : ""}`}

27: onClick={onClick}>

28: <line

29: className={styles.overlay}

30: x1={edge.nodes[0].position.x}

31: y1={edge.nodes[0].position.y}

32: x2={edge.nodes[1].position.x}

33: y2={edge.nodes[1].position.y}

34: />

35: <line

36: className={styles.main}

37: x1={edge.nodes[0].position.x}

38: y1={edge.nodes[0].position.y}

39: x2={edge.nodes[1].position.x}

40: y2={edge.nodes[1].position.y}

41: />

42: </ElementContainer>

43: );

44: }

45:

46: export default DiagramEdge;

### File: SidebarPrimary.tsx

01: import React, { useState } from "react";

02: import {

03: faArrowsAlt,

04: faMapMarked,

05: faPlus,

06: } from "@fortawesome/free-solid-svg-icons";

07: import SidebarContainer from "../SidebarContainer/SidebarContainer";

08: import SidebarMenuSwitcher from "../SidebarMenu/SidebarMenuSwitcher";

09: import SidebarMenuSwitcherOption from "../SidebarMenu/SidebarMenuSwitcherOption";

10: import MenuCreateEdge from "./menus/MenuCreateEdge";

11: import MenuCreateNode from "./menus/MenuCreateNode";

12: import styles from "./SidebarPrimary.module.scss";

13: import MenuMoveNode from "./menus/MenuMoveNode";

14: import MenuAlgorithm from "./menus/MenuAlgorithm";

15:

16: enum Menu {

17: create = "CREATE",

18: move = "MOVE",

19: algorithm = "ALGORITHM",

20: }

21:

22: /\*\*

23: \* Give user options to add items to the graph

24: \*/

25: function SidebarPrimary() {

26: // Current menu state

27: const [menu, setMenu] = useState<Menu>(Menu.create);

28:

29: const CreateMenu = () => (

30: <div className={styles.menuList}>

31: <MenuCreateNode />

32: <MenuCreateEdge />

33: </div>

34: );

35:

36: /\*\*

37: \* Show currently selected menu

38: \*/

39: const CurrentMenu = () => {

40: switch (menu) {

41: case Menu.create:

42: return <CreateMenu />;

43: case Menu.move:

44: return <MenuMoveNode />;

45: case Menu.algorithm:

46: return <MenuAlgorithm />;

47: }

48: };

49:

50: return (

51: <SidebarContainer>

52: <SidebarMenuSwitcher>

53: <SidebarMenuSwitcherOption

54: menu={Menu.create}

55: currentMenu={menu}

56: icon={faPlus}

57: setMenu={setMenu}

58: />

59: <SidebarMenuSwitcherOption

60: menu={Menu.move}

61: currentMenu={menu}

62: icon={faArrowsAlt}

63: setMenu={setMenu}

64: />

65: <SidebarMenuSwitcherOption

66: menu={Menu.algorithm}

67: currentMenu={menu}

68: icon={faMapMarked}

69: setMenu={setMenu}

70: />

71: </SidebarMenuSwitcher>

72: <CurrentMenu />

73: </SidebarContainer>

74: );

75: }

76:

77: export default SidebarPrimary;

### File: MenuAlgorithm.tsx

01: import React from "react";

02: import SidebarMenu from "../../SidebarMenu/SidebarMenu";

03: import styles from "../SidebarPrimary.module.scss";

04: import sidebarStyles from "../../Sidebar.module.scss";

05: import { useContext } from "react";

06: import { GraphContext } from "../../../../graph/context/graphContext";

07: import { findShortestPath } from "../../../../graph/context/graphActions";

08: import { SelectNode } from "./SelectNode";

09: import { useRef } from "react";

10: import { GraphEdge, GraphNode } from "../../../../graph";

11:

12: const Edge = ({ edge }: { edge: GraphEdge }) => {

13: return (

14: <div className={styles.edge}>

15: <div>{edge.nodes[0].id}</div>

16: <div>{edge.nodes[1].id}</div>

17: </div>

18: );

19: };

20:

21: const Result = () => {

22: const { state } = useContext(GraphContext);

23:

24: if (!state.shortestPath) return null;

25:

26: return (

27: <div>

28: <div className={styles.path}>

29: <label className={sidebarStyles.formLabel}>Result</label>

30: <div className={styles.header}>

31: <div>From</div>

32: <div>To</div>

33: </div>

34: {state.shortestPath.path.map((edge, index) => (

35: <Edge edge={edge} key={index} />

36: ))}

37: </div>

38: <div>Distance: {state.shortestPath.distance}</div>

39: </div>

40: );

41: };

42:

43: function MenuAlgorithm() {

44: const { state, dispatch } = useContext(GraphContext);

45:

46: // Reference selected nodes

47: const startNodeRef = useRef<GraphNode>();

48: const endNodeRef = useRef<GraphNode>();

49:

50: const onStartAlgorithm = () => {

51: if (startNodeRef.current && endNodeRef.current)

52: findShortestPath(

53: dispatch,

54: startNodeRef.current,

55: endNodeRef.current

56: );

57: };

58:

59: return (

60: <SidebarMenu title='Shortest Path'>

61: <div className={sidebarStyles.form}>

62: <label className={sidebarStyles.formLabel}>Start Node</label>

63: <SelectNode

64: nodeRef={startNodeRef}

65: current={state.current as GraphNode}

66: />

67: <label className={sidebarStyles.formLabel}>End Node</label>

68: <SelectNode

69: nodeRef={endNodeRef}

70: current={state.current as GraphNode}

71: />

72: <label className={sidebarStyles.formLabel}>

73: Find Shortest Path

74: </label>

75: <button

76: onClick={onStartAlgorithm}

77: className={sidebarStyles.formSubmit}>

78: Start

79: </button>

80: <Result />

81: </div>

82: </SidebarMenu>

83: );

84: }

85:

86: export default MenuAlgorithm;

### File: MenuCreateEdge.tsx

01: import React from "react";

02: import { useRef } from "react";

03: import { useContext } from "react";

04: import { GraphEdge, GraphNode } from "../../../../graph";

05: import { addEdge } from "../../../../graph/context/graphEdgeActions";

06: import { GraphContext } from "../../../../graph/context/graphContext";

07: import SidebarMenu from "../../SidebarMenu/SidebarMenu";

08: import sidebarStyles from "../../Sidebar.module.scss";

09: import { SelectNode } from "./SelectNode";

10:

11: /\*\*

12: \* Create new Edge and add it to graph

13: \*/

14: function MenuCreateEdge() {

15: const { state, dispatch } = useContext(GraphContext);

16:

17: // Reference selected nodes

18: const fromNodeRef = useRef<GraphNode>();

19: const toNodeRef = useRef<GraphNode>();

20:

21: function handleCreateEdge(event: React.FormEvent<HTMLFormElement>) {

22: event.preventDefault();

23: const formData = new FormData(event.currentTarget);

24:

25: const fromNode = fromNodeRef.current;

26: const toNode = toNodeRef.current;

27: const weight = parseInt(formData.get("weight") as string) || 0;

28:

29: if (!fromNode || !toNode) return;

30:

31: const edge = new GraphEdge(fromNode, toNode, weight);

32:

33: addEdge(dispatch, edge);

34: }

35:

36: return (

37: <SidebarMenu title='Create Edge'>

38: <form onSubmit={handleCreateEdge} className={sidebarStyles.form}>

39: <label className={sidebarStyles.formLabel}>From</label>

40: <SelectNode

41: nodeRef={fromNodeRef}

42: current={state.current as GraphNode}

43: />

44: <label className={sidebarStyles.formLabel}>To</label>

45: <SelectNode

46: nodeRef={toNodeRef}

47: current={state.current as GraphNode}

48: />

49: <label className={sidebarStyles.formLabel}>Weight</label>

50: <input

51: className={sidebarStyles.formInput}

52: type='number'

53: name='weight'

54: placeholder='0'

55: min={0}

56: maxLength={24}

57: />

58: <button className={sidebarStyles.formSubmit}>Add Edge</button>

59: </form>

60: </SidebarMenu>

61: );

62: }

63:

64: export default MenuCreateEdge;

### File: MenuCreateNode.tsx

01: import React from "react";

02: import { useContext } from "react";

03: import { GraphNode } from "../../../../graph";

04: import { addNode } from "../../../../graph/context/graphNodeActions";

05: import { GraphContext } from "../../../../graph/context/graphContext";

06: import SidebarMenu from "../../SidebarMenu/SidebarMenu";

07: import styles from "../SidebarPrimary.module.scss";

08: import sidebarStyles from "../../Sidebar.module.scss";

09:

10: function MenuCreateNode() {

11: const { dispatch } = useContext(GraphContext);

12:

13: function handleSubmitCreateNode(event: React.FormEvent<HTMLFormElement>) {

14: event.preventDefault();

15:

16: // Get data

17: const formData = new FormData(event.currentTarget);

18: const name = formData.get("name") as string;

19: const id = formData.get("id") as string;

20: if (!name || !id) return;

21:

22: // Create node

23: const node = new GraphNode(name, id, { x: 200, y: 200 });

24:

25: // Update graph

26: addNode(dispatch, node);

27: event.currentTarget.reset();

28:

29: // Return focus to name entry

30: const nameInput = event.currentTarget.querySelector(

31: "input[name='name']"

32: ) as HTMLInputElement;

33: nameInput.focus();

34: }

35:

36: return (

37: <SidebarMenu title='Create Node'>

38: <form

39: onSubmit={handleSubmitCreateNode}

40: className={sidebarStyles.form}>

41: <label className={sidebarStyles.formLabel}>Name</label>

42: <input

43: type='text'

44: name='name'

45: autoComplete='off'

46: maxLength={24}

47: placeholder='Room 1'

48: className={sidebarStyles.formInput}

49: />

50: <label className={sidebarStyles.formLabel}>ID</label>

51: <input

52: type='text'

53: name='id'

54: autoComplete='off'

55: placeholder='RM1'

56: maxLength={3}

57: className={`${sidebarStyles.formInput} ${styles.idInput}`}

58: />

59: <button className={sidebarStyles.formSubmit}>Add Node</button>

60: </form>

61: </SidebarMenu>

62: );

63: }

64:

65: export default MenuCreateNode;

### File: MenuMoveNode.tsx

01: import React, { FormEvent } from "react";

02: import SidebarMenu from "../../SidebarMenu/SidebarMenu";

03: import sidebarStyles from "../../Sidebar.module.scss";

04: import { useContext } from "react";

05: import { GraphContext } from "../../../../graph/context/graphContext";

06: import { changeSnap } from "../../../../graph/context/graphActions";

07:

08: function MenuMoveNode() {

09: const { state, dispatch } = useContext(GraphContext);

10:

11: /\*\*

12: \* Change context snap value based on input.

13: \* Min value always at 1.

14: \*/

15: function onChangeSnap(event: FormEvent<HTMLInputElement>) {

16: let newSnap = event.currentTarget.valueAsNumber;

17: if (newSnap <= 1 || !newSnap) {

18: newSnap = 1;

19: changeSnap(dispatch, newSnap);

20: } else {

21: changeSnap(dispatch, newSnap);

22: }

23: event.currentTarget.valueAsNumber = newSnap;

24: }

25:

26: return (

27: <SidebarMenu title='Move Node'>

28: <div className={sidebarStyles.form}>

29: <label className={sidebarStyles.formLabel}>Snap to grid</label>

30: <input

31: className={sidebarStyles.formInput}

32: type='number'

33: name='snap'

34: defaultValue={state.snap}

35: min={0}

36: step={10}

37: onBlur={onChangeSnap}

38: />

39: <div>Click and drag nodes to rearrange them!</div>

40: </div>

41: </SidebarMenu>

42: );

43: }

44:

45: export default MenuMoveNode;

### File: SelectNode.tsx

01: import React from "react";

02: import { useRef } from "react";

03: import { useEffect } from "react";

04: import { useState } from "react";

05: import { GraphNode } from "../../../../graph";

06: import styles from "../SidebarPrimary.module.scss";

07: import sidebarStyles from "../../Sidebar.module.scss";

08:

09: type SelectNodeProps = {

10: nodeRef: React.MutableRefObject<GraphNode | undefined>;

11: current: GraphNode | undefined;

12: };

13:

14: /\*\*

15: \* Handle selecting nodes from graph

16: \*/

17: export const SelectNode = React.memo(

18: ({ nodeRef, current }: SelectNodeProps) => {

19: const selectNodeDivRef = useRef<HTMLDivElement>(null!);

20:

21: const [active, setActive] = useState(false);

22:

23: const [node, setNode] = useState<GraphNode>();

24:

25: // Select current node

26: useEffect(() => {

27: if (!active) return;

28:

29: if (current instanceof GraphNode) {

30: setNode(current);

31: }

32: }, [current, active]); // I would prefer to not have "active" as a dependency here

33:

34: // Update node ref

35: useEffect(() => {

36: nodeRef.current = node;

37: }, [node, nodeRef]);

38:

39: // Toggle active states on focus and blur

40: function handleOnFocus(event: React.FocusEvent) {

41: setActive(true);

42: }

43: function handleOnBlur(event: React.FocusEvent) {

44: setActive(false);

45: }

46:

47: const DisplayName = () => {

48: if (!node) return null;

49: return (

50: <span>

51: {node.id} - {node.name}

52: </span>

53: );

54: };

55:

56: return (

57: <div

58: ref={selectNodeDivRef}

59: onFocus={handleOnFocus}

60: onBlur={handleOnBlur}

61: className={`${sidebarStyles.formInput} ${styles.selectNode}`}

62: tabIndex={0}>

63: <DisplayName />

64: </div>

65: );

66: }

67: );

### File: SidebarSecondary.tsx

01: import React from "react";

02: import { useContext } from "react";

03: import { GraphEdge, GraphNode } from "../../../graph";

04: import { GraphContext } from "../../../graph/context/graphContext";

05: import SidebarContainer from "../SidebarContainer/SidebarContainer";

06: import GraphEdgeInfo from "./GraphEdgeInfo";

07: import GraphNodeInfo from "./GraphNodeInfo";

08:

09: const Current = () => {

10: const { state } = useContext(GraphContext);

11: const current = state.current;

12:

13: // Node info

14: if (current instanceof GraphNode) {

15: return <GraphNodeInfo node={current} />;

16: }

17:

18: // Edge info

19: else if (current instanceof GraphEdge) {

20: return <GraphEdgeInfo edge={current} />;

21: }

22:

23: return null;

24: };

25:

26: function SidebarSecondary() {

27: console.log("rerender");

28:

29: return (

30: <SidebarContainer>

31: <Current />

32: </SidebarContainer>

33: );

34: }

35:

36: export default SidebarSecondary;

### File: GraphNodeInfo.tsx

001: import React, { FormEvent } from "react";

002: import { GraphNode } from "../../../graph";

003: import SidebarMenu from "../SidebarMenu/SidebarMenu";

004: import sidebarStyles from "../Sidebar.module.scss";

005: import styles from "./SidebarSecondary.module.scss";

006: import {

007: editNodeID,

008: editNodeName,

009: moveNode,

010: removeNode,

011: } from "../../../graph/context/graphNodeActions";

012: import { useContext } from "react";

013: import { GraphContext } from "../../../graph/context/graphContext";

014: import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

015: import { faTrash } from "@fortawesome/free-solid-svg-icons";

016: import { deselect } from "../../../graph/context/graphActions";

017:

018: const GraphNodeInfo = React.memo(({ node }: { node: GraphNode }) => {

019: const EditName = () => {

020: const { dispatch } = useContext(GraphContext);

021:

022: function handleChange(event: FormEvent<HTMLInputElement>) {

023: editNodeName(dispatch, node, event.currentTarget.value);

024: }

025:

026: return (

027: <>

028: <label className={sidebarStyles.formLabel}>Name</label>

029: <input

030: className={sidebarStyles.formInput}

031: type='text'

032: name='name'

033: defaultValue={node.name}

034: onChange={handleChange}

035: />

036: </>

037: );

038: };

039:

040: const EditID = () => {

041: const { dispatch } = useContext(GraphContext);

042:

043: function handleChange(event: FormEvent<HTMLInputElement>) {

044: editNodeID(dispatch, node, event.currentTarget.value);

045: }

046:

047: return (

048: <>

049: <label className={sidebarStyles.formLabel}>ID</label>

050: <input

051: className={sidebarStyles.formInput}

052: type='text'

053: name='ID'

054: defaultValue={node.id}

055: maxLength={3}

056: onChange={handleChange}

057: />

058: </>

059: );

060: };

061:

062: const EditPosition = () => {

063: const { dispatch } = useContext(GraphContext);

064:

065: // Move Node to new position on change input

066: function handlePositionInput(event: React.FormEvent<HTMLInputElement>) {

067: const position = node.position;

068: const axis = event.currentTarget.name as "x" | "y";

069: position[axis] = parseInt(event.currentTarget.value);

070: moveNode(dispatch, node, position);

071: }

072:

073: return (

074: <>

075: <label className={sidebarStyles.formLabel}>Position</label>

076: <div className={styles.editPosition}>

077: <label>X:</label>

078: <input

079: className={sidebarStyles.formInput}

080: type='number'

081: step={10}

082: name='x'

083: onChange={handlePositionInput}

084: value={node.position.x}

085: />

086: <label>Y:</label>

087: <input

088: className={sidebarStyles.formInput}

089: type='number'

090: step={10}

091: name='y'

092: onChange={handlePositionInput}

093: value={node.position.y}

094: />

095: </div>

096: </>

097: );

098: };

099:

100: const DeleteNode = () => {

101: const { dispatch } = useContext(GraphContext);

102:

103: function handleDeleteNode() {

104: removeNode(dispatch, node);

105: deselect(dispatch);

106: }

107:

108: return (

109: <button className={styles.deleteButton} onClick={handleDeleteNode}>

110: <FontAwesomeIcon icon={faTrash} />

111: <span>Delete Node</span>

112: </button>

113: );

114: };

115:

116: return (

117: <SidebarMenu title='Edit Node'>

118: <div className={sidebarStyles.form}>

119: <EditName />

120: <EditID />

121: <EditPosition />

122: <DeleteNode />

123: </div>

124: </SidebarMenu>

125: );

126: });

127:

128: export default GraphNodeInfo;

### File: GraphEdgeInfo.tsx

01: import React, { FormEvent } from "react";

02: import { GraphEdge } from "../../../graph";

03: import SidebarMenu from "../SidebarMenu/SidebarMenu";

04: import styles from "./SidebarSecondary.module.scss";

05: import sidebarStyles from "../Sidebar.module.scss";

06: import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

07: import { faTrash } from "@fortawesome/free-solid-svg-icons";

08: import {

09: editEdgeWeight,

10: removeEdge,

11: } from "../../../graph/context/graphEdgeActions";

12: import { deselect } from "../../../graph/context/graphActions";

13: import { useContext } from "react";

14: import { GraphContext } from "../../../graph/context/graphContext";

15:

16: const GraphEdgeInfo = React.memo(({ edge }: { edge: GraphEdge }) => {

17: const EdgeNodes = () => {

18: return (

19: <>

20: <label className={sidebarStyles.formLabel}>From</label>

21: <div>

22: {edge.nodes[0].id} - {edge.nodes[1].name}

23: </div>

24: <label className={sidebarStyles.formLabel}>To</label>

25: <div>

26: {edge.nodes[0].id} - {edge.nodes[1].name}

27: </div>

28: </>

29: );

30: };

31:

32: const EdgeWeight = () => {

33: const { dispatch } = useContext(GraphContext);

34:

35: function handleChange(event: FormEvent<HTMLInputElement>) {

36: const newWeight = parseInt(event.currentTarget.value);

37: if (!isNaN(newWeight)) {

38: editEdgeWeight(dispatch, edge, newWeight);

39: }

40: }

41:

42: return (

43: <>

44: <label className={sidebarStyles.formLabel}>Weight</label>

45: <input

46: className={sidebarStyles.formInput}

47: type='number'

48: name='weight'

49: value={edge.weight}

50: min={0}

51: onChange={handleChange}

52: />

53: </>

54: );

55: };

56:

57: const DeleteEdge = () => {

58: const { dispatch } = useContext(GraphContext);

59:

60: function handleDeleteEdge() {

61: removeEdge(dispatch, edge);

62: deselect(dispatch);

63: }

64:

65: return (

66: <button className={styles.deleteButton} onClick={handleDeleteEdge}>

67: <FontAwesomeIcon icon={faTrash} />

68: <span>Delete Edge</span>

69: </button>

70: );

71: };

72:

73: return (

74: <SidebarMenu title='Edit Edge'>

75: <div className={sidebarStyles.form}>

76: <EdgeNodes />

77: <EdgeWeight />

78: <DeleteEdge />

79: </div>

80: </SidebarMenu>

81: );

82: });

83:

84: export default GraphEdgeInfo;

### File: SidebarMenu.tsx

01: import styles from "./SidebarMenu.module.scss";

02:

03: type SidebarMenuProps = {

04: children?: JSX.Element | JSX.Element[];

05: title: string;

06: };

07:

08: function SidebarMenu({ children, title }: SidebarMenuProps) {

09: /\*\*

10: \* Return title element if title prop is passed

11: \*/

12: const Title = () => {

13: if (!title) return null;

14: return <div className={styles.menuTitle}>{title}</div>;

15: };

16:

17: return (

18: <div className={styles.sidebarMenu}>

19: <Title />

20: {children}

21: </div>

22: );

23: }

24:

25: export default SidebarMenu;

### File: SidebarMenuSwitcher.tsx

01: import React from "react";

02: import styles from "./SidebarMenuSwitcher.module.scss";

03:

04: type SidebarMenuSwitcherProps = {

05: children?: JSX.Element[];

06: };

07:

08: function SidebarMenuSwitcher({ children }: SidebarMenuSwitcherProps) {

09: return <div className={styles.sidebarMenuSwitcher}>{children}</div>;

10: }

11:

12: export default SidebarMenuSwitcher;

### File: SidebarMenuSwitcherOption.tsx

01: import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

02: import React from "react";

03: import styles from "./SidebarMenuSwitcher.module.scss";

04:

05: type SidebarMenuSwitcherOptionProps = {

06: menu: string;

07: currentMenu: string;

08: icon: any;

09: setMenu: React.Dispatch<React.SetStateAction<any>>;

10: };

11:

12: function SidebarMenuSwitcherOption({

13: menu,

14: currentMenu,

15: icon,

16: setMenu,

17: }: SidebarMenuSwitcherOptionProps) {

18: /\*\*

19: \* Fires on click option

20: \*/

21: function onClick() {

22: setMenu(menu);

23: }

24:

25: /\*\*

26: \* Determine whether this option is currently selected

27: \*/

28: const isActive = () => (menu === currentMenu ? styles.isActive : null);

29:

30: const className = `${styles.option} ${isActive()}`;

31:

32: return (

33: <div className={className} onClick={onClick}>

34: <FontAwesomeIcon icon={icon} />

35: </div>

36: );

37: }

38:

39: export default SidebarMenuSwitcherOption;

### File: SidebarContainer.tsx

01: import styles from "./SidebarContainer.module.scss";

02:

03: type SidebarContainerProps = {

04: children?: JSX.Element | JSX.Element[];

05: };

06:

07: function SidebarContainer({ children }: SidebarContainerProps) {

08: return <div className={styles.container}>{children}</div>;

09: }

10:

11: export default SidebarContainer;

## Utility

### File: storage.ts

01: export class InMemoryKeyValueStore<K, V> {

02: protected store: { key: K; value: V }[] = [];

03:

04: /\*\*

05: \* If keys and defaultValue params passed,

06: \* map each key to default value in store.

07: \*/

08: constructor(initialStore?: { keys: K[]; defaultValue: V }) {

09: if (!initialStore) return;

10: const { keys, defaultValue } = initialStore;

11: this.store = keys.map((key) => ({ key, value: defaultValue }));

12: }

13:

14: private getStoreObject(key: K) {

15: const object = this.store.find((obj) => obj.key === key);

16: if (!object) console.error("Invalid Key:", key);

17: return object;

18: }

19:

20: public get(key: K) {

21: const object = this.getStoreObject(key);

22: return object?.value;

23: }

24:

25: public set(key: K, value: V) {

26: const object = this.getStoreObject(key);

27: if (!object) return;

28: object.value = value;

29: }

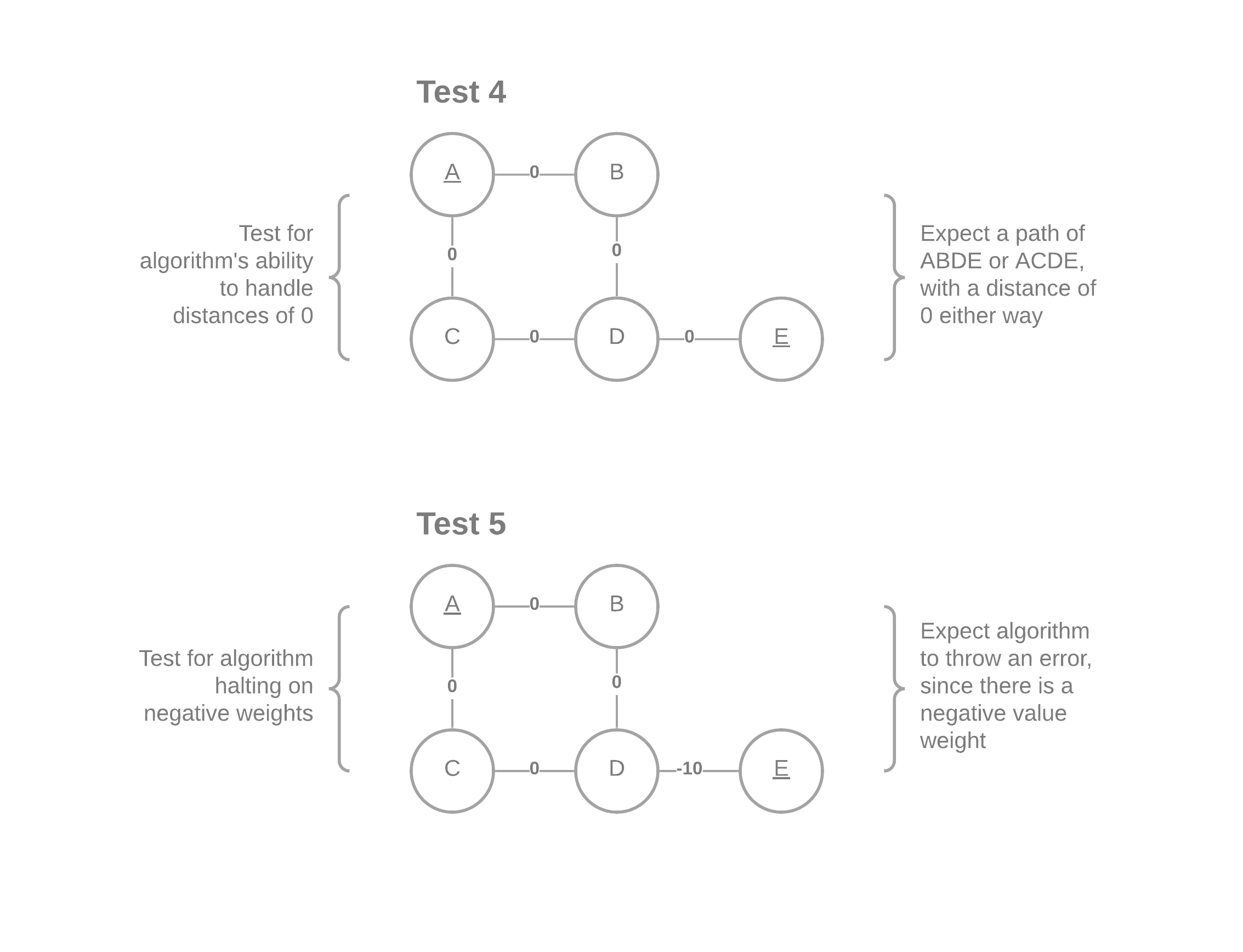
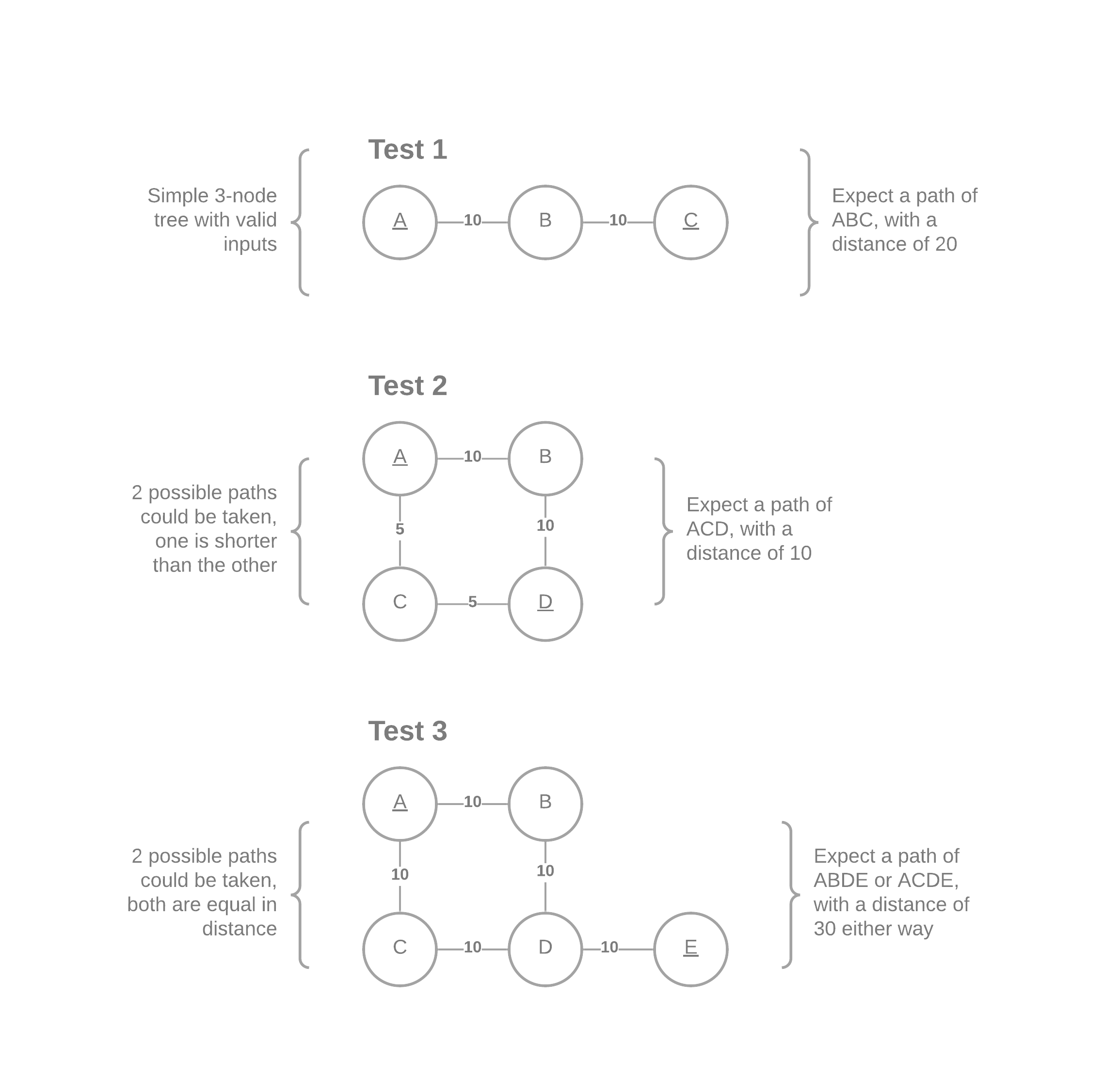
30: }

# Testing

## Algorithm Tests

To test the implementation of the algorithm, I will run a set of predefined tests on the function of the algorithm.

Here is a series of diagrams representing the tests I am running on a graph. The nodes with underlined labels represent start/end node.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Description | Test Data | Expected Result | Result |
| 1.1 | Simple 3-node tree with valid inputs | nodes:  [A, B, C]  edges:  [AB: 10, BC: 10] | distance: 20  path: [A, B, C] | FAIL  distance: 20  path: [null, A] |
| 1.2 | - | - | - | FAIL  distance: 20  path: [A, B] |
| 1.3 | - | - | - | PASS  distance: 20  path: [A, B, C] |
| 2 | 2 possible paths could be taken, one is shorter than the other | nodes:  [A, B, C, D]  edges:  [AB: 10, BD: 10, AC: 5, CD: 5] | distance: 10  path: [A, C, D] | PASS  distance: 10  path: [A, C, D] |
| 3 | 2 possible paths could be taken, both are equal in distance | nodes:  [A, B, C, D, E]  edges:  [AB: 10, BD: 10, AC: 10, CD: 10, DE: 10] | distance: 30  path: [A, B, D, E] or [A, C, D, E] | PASS  distance: 30  path: [A, B, D, E] |
| 4 | Test for algorithm's ability to handle distances of 0 | nodes:  [A, B, C, D]  edges:  [AB: 0, BD: 0, AC: 0, CD: 0] | distance: 0  path: [A, B, D, E] or [A, C, D, E] | PASS  distance: 0  path: [A, B, D, E] |
| 5.1 | Test for algorithm halting on negative weights | nodes:  [A, B, C, D]  edges:  [AB: 0, BD: 0, AC: 0, CD: -10] | expect error: negative value weight | FAIL  error: invalid array length Allocation failed – JavaScript heap out of memory |
| 5.2 | - | - | - | PASS  error: Cannot handle negative value weight. |

## Test Result Screenshots

### Test 1.1

|  |
| --- |
|  |
| Test 1.1 Result |

This test failed because the path returned was incorrect. The distance returned 20, which is true, however the path was [null, A] when expecting a path of [A, B, C]. I think the reason this happened was invalid logic when compiling the path.

### Test 1.2

|  |
| --- |
|  |
| Test 1.2 Result |

After making changes to the way the algorithm compiles the path *(See Change 1 – Test 1.1 in Pseudocode Changes section below),* I ran Test 1 again and I received this as a result. The distance was correct again, and the path has improved, however it was still missing one node. I believe the reason for node C not being included in the path was because I only start adding nodes to the path starting from the parent of C, not C itself.

### Test 1.3

|  |
| --- |
|  |
| Test 1.3 Result |

This test was sucessful in finding the shortest path in a simple 3-node tree, where there is only one possible path. This shows that the algorithm can at least function and return a valid result.

### Test 2

|  |
| --- |
|  |
| Test 2 Result |

### Test 3

|  |
| --- |
|  |
| Test 3 Result |

### Test 4

|  |
| --- |
|  |
| Test 4 Result |

### Test 5.1

|  |
| --- |
|  |
| Test 5.1 Result |

The error states that the JavaScript heap is out of memory, which suggests that this test made the algorithm run in an infinite loop. To fix this, the algorithm must be able to handle negative distances.

### Test 5.2

|  |
| --- |
|  |
| Test 5.2 Result |

Here the program successfully throws an error showing that negative value weights are not accepted.

## Pseudocode changes

I made some changes to the way I am handling the parent nodes, in order to fix the invalid path it returned in Test 1.1. In this new method, I made it so that the parent is added to the path before changing to the new node.

### Change 1 – Test 1.1

1: # Gather results

2: distance = distances.get(endNode)

3: array path[]

4: parent = parents[endNode]

5: while parent != null

6: path.add(parent)

7: parent = parents[parent]

8: endwhile

File: changes

14: # Gather results

15: distance = distances.get(endNode)

16: array path[]

17: parent = endNode

18: while parent != null

19: path.add(parent)

20: parent = parents[parent]

21: endwhile

### Change 2 – Test 1.2

Previously, the endNode was not included in the final path, so I changed the way the path is compiled by starting from the endNode itself, rather than its parent.

1: # Gather results

2: distance = distances.get(endNode)

3: array path[]

4: parent = endNode

5: while parent != null

6: path.add(parent)

7: parent = parents[parent]

8: endwhile

# Evaluation

The project came out successful in that I have fulfilled all the objectives that I had set.

I found implementing Djikstra’s algorithm to be the most challenging part of this project. At one point I was stuck trying to find out how to make the edges of the graph undirected. After protoyping different ways of handling graph edges, I ended up using a tuple to store the two nodes an edge connects.

|  |
| --- |
|  |
| Sreenshot of the app finding the shortest path in a graph |

# Bibliography

Graph Online 2015, viewed 8 July 2021, <<https://graphonline.ru/en/>/>

Microsoft, 2012, viewed 8 July 2021, <<https://www.typescriptlang.org/>/>

Mahipal Nehra 2021, *Top 10 Programming Languages for Desktop Apps in 2021*, Decipherzone Softwares,viewed 8 July 2021, <<https://www.decipherzone.com/blog-detail/top-programming-languages-for-desktop-apps-in-2021>/>