Centre Number	Candidate Name	Candidate Number
29065	Xiangyu Zhao	6960

A Classroom Teaching Tool for Graph Theory

Table of Contents

Appendicies References 5
Analysis 6
Introduction 6
Target users 6
Some current existing teaching tools: advantages and drawbacks 7
Main functionalities of the project 16
Structure of the project 17
Requirements of the project 18
Record of feedbacks from target users 31
Documented Design 34
Back-end design 34
GUI design 35
`Vertex` control 42
Pseudo-code 42
`DijkstraVertexLabel` control 46
Pseudo-code 46
`AccountMenu` control 49
Pseudo-code 49
`TaskSettingControls` class 52
Pseudo-code 52
`DoTaskControls` class 54
Pseudo-code 54
`VertexTagControls` class 56
Pseudo-code 56
`Graph` class 57
Pseudo-code 57
`AdjacencyMatrix` class 65
Pseudo-code 65
`AdjacencyList` class 67
Pseudo-code 67
`MinimumSpanningTreeExample` class 69
Pseudo-code 69
`ShortestPathExample` class 71
Pseudo-code 71
Prim's minimum spanning tree algorithm 73
Algorithmic pseudo-code 73

Kruskal's minimum spanning tree algorithm 74
Algorithmic pseudo-code[5] 74
Dijkstra's shortest path algorithm 75
Algorithmic pseudo-code[5] 75
MD5 hashing algorithm 77
Pseudo-code solutions to requirements 79
SQL pseudo-commands list 112
Implementation 119
Source code for the project 119
Completeness of solution 120
Example technical skills 129
Coding styles 137
Testing 142
Testing plan 142
Testing data 143
Module 0 – Log in: 143
Module 1 – Sign up 147
Module 2 – Primary Menu 155
Module 2.1 – Teaching Section Menu: Select Topics 155
Module 2.1.*.1 – Topic Overview 155
Module 2.1.*.2 – Step-by-Step Demonstrations 156
Module 2.2 – Task Setting Window (Teacher accounts only) 157
Module 2.2.1 – Edit Adjacency Matrix 165
Module 2.2.2 – Edit Adjacency List 165
Module 2.2.3 – Sketch Board 166
Module 2.3 – Question Bank Section: List of Questions 170
Module 2.3.1 – Add Questions 170
Module 2.3.2 – Edit Questions 170
Module 2.3.3 – Delete Questions 170
Module 2.3.4 – Do Questions 171
Module 2.3.4.1 – Mark Questions 171
Module 3 – User Accounts 174
Module 3.1 – Account Setting 174
Module 3.2 – Quit 174
Evaluation 175
Feedbacks from users 175
Mr John Cowley 175
Mr Peter Hayes 176
Mr Thomas Hurst 183
Possible extensions 186

Client meeting log 188

References 195

747.45

Appendicies References

Appendix 1 – GUI implementation

Appendix 2 – GraphTeachingTool Source Code

Appendix 3 – Testing Video

Appendix 4 – Original feedback emails from users

Appendix 5 – Transcripts for Testing Video

Analysis

Introduction

Graph Theory is an important part in A-Level Mathematics and Computer Science subjects. A good majority of Sixth Form students who are studying Mathematics, Further Mathematics or Computer Science need a more hands-on approach to this very abstract subject which lies at the heart of Computer Science, and a detailed and easy-understandable teaching tool is essential to their understandings about Graph Theory. Unfortunately, however, such a teaching tool is quite rare to find in the majority of academic supportive websites. This project aims at creating a teaching tool for Graph Theory, particularly in the minimum spanning tree problem and the shortest path problem, based on Decision Mathematics from various specifications.

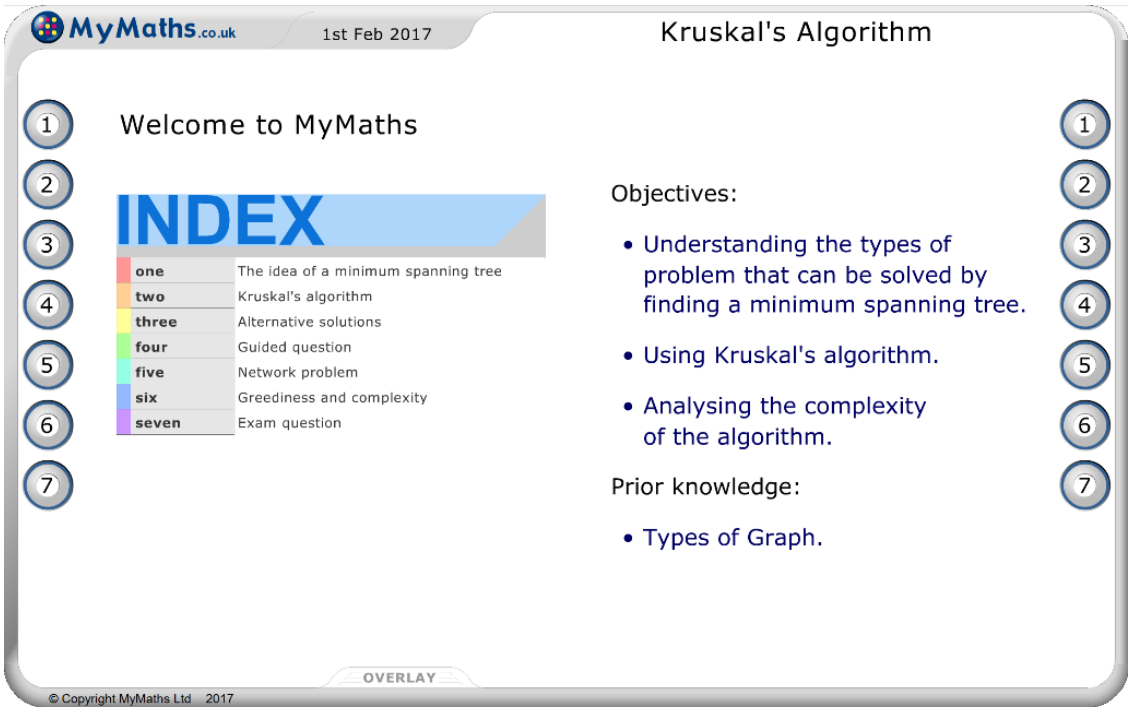
Target users

- **Teachers:** All the A-Level Mathematics/Computer Science teachers who are teaching Graph Theory to their students.
- **Students:** All the A-Level Mathematics/Computer Science students who are struggling with Graph Theory and are seeking a teaching tool for self-learning.

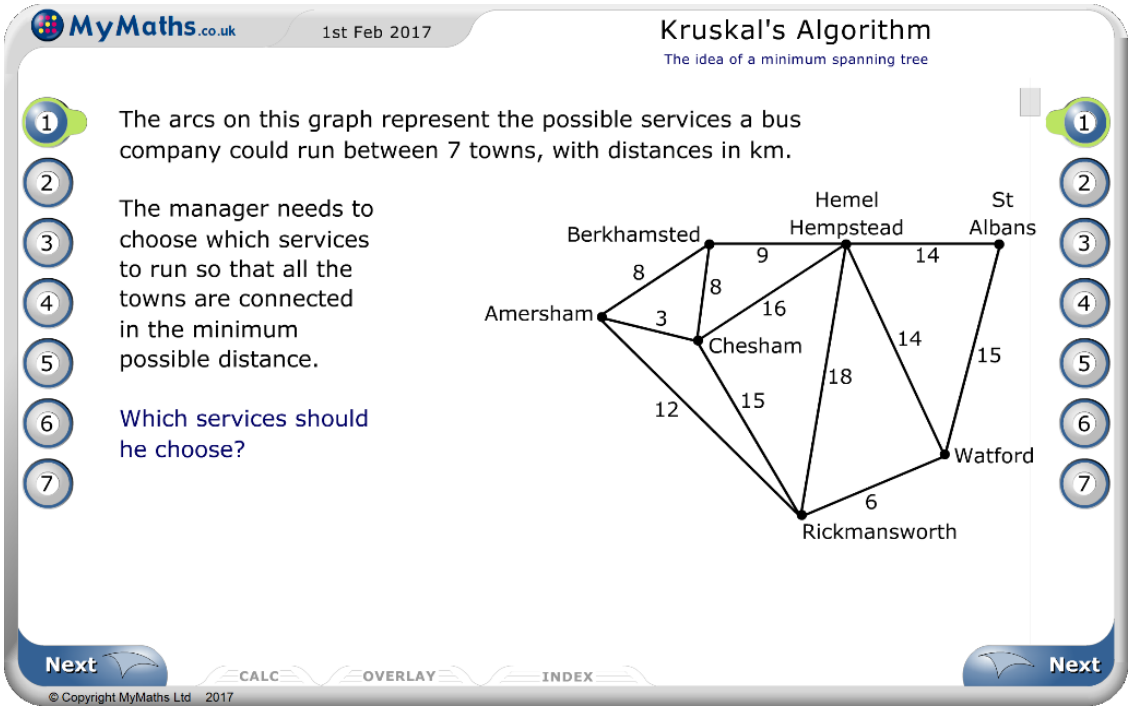
Some current existing teaching tools: advantages and drawbacks

Consider, for example, the determination of the minimum spanning tree for a given graph. This entails choosing a graph with a pre-determined number of nodes and edges which will then be used in order to find the minimum spanning tree. The algorithms used for this determination are the well-known Kruskal's and Prim's Algorithms.

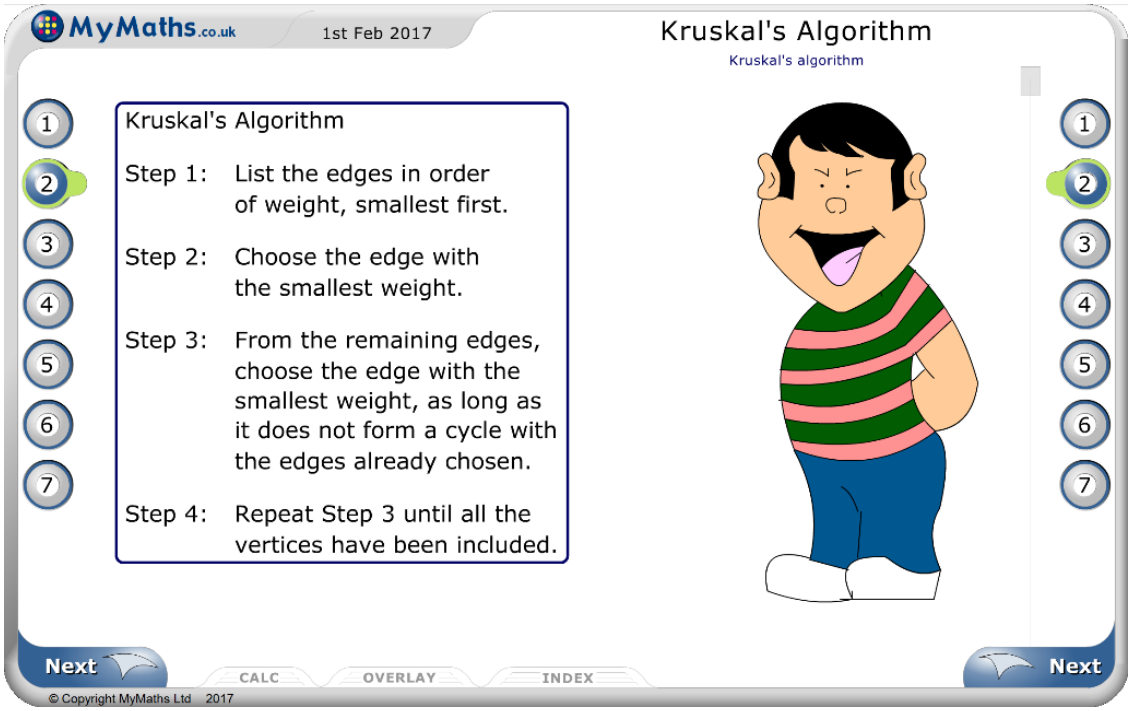
Here is a sequence of screenshots showing how this problem is solved via Kruskal's Algorithm by www.mymaths.co.uk[1], a popular supportive website for UK Mathematics students:



Menu of the teaching tool



Problem description



Algorithm description

MyMaths.co.uk 1st Feb 2017

Kruskal's Algorithm

Kruskal's algorithm

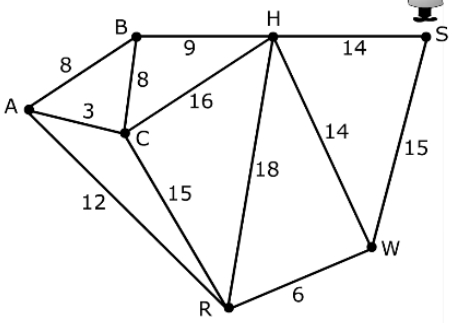
Notes

Use Kruskal's Algorithm to find a minimum spanning tree for this graph.

Step 1: List the edges in order of weight, smallest first.

It's easy to go wrong if you work straight from the graph so it makes sense to write the edges out in order.

Edge	Weight	Edge	Weight
AC	3	HW	14
RW	6	HS	14
AB	8	CR	15
BC	8	SW	15
BH	9	CH	16
AR	12	HR	18



Next CALC OVERLAY INDEX Next

© Copyright MyMaths Ltd 2017

Algorithm run-through (1)

MyMaths.co.uk 2nd Feb 2017

Kruskal's Algorithm

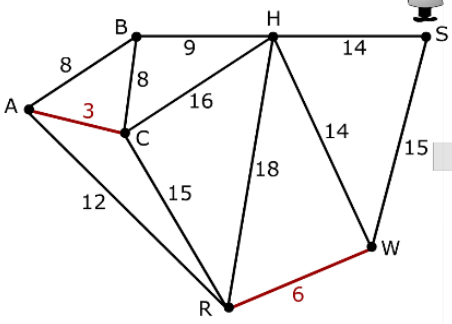
Kruskal's algorithm

Notes

Use Kruskal's Algorithm to find a minimum spanning tree for this graph.

Edge	Weight	Edge	Weight
AC	3 ✓	HW	14
RW	6 ✓	HS	14
AB	8	CR	15
BC	8	SW	15
BH	9	CH	16
AR	12	HR	18

Step 3: From the remaining edges, choose the edge with the smallest weight, as long as it does not form a cycle with the edges already chosen.

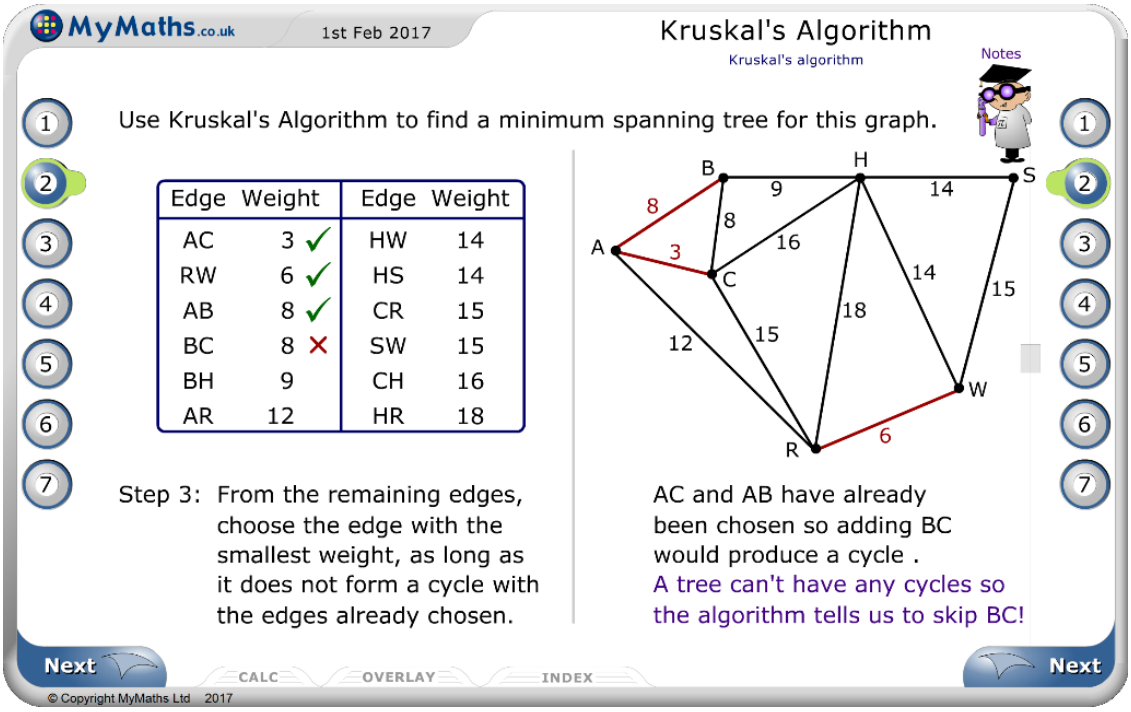


Choose RW as this is the next shortest edge.
It doesn't matter that the edges we've chosen aren't connected at this point.

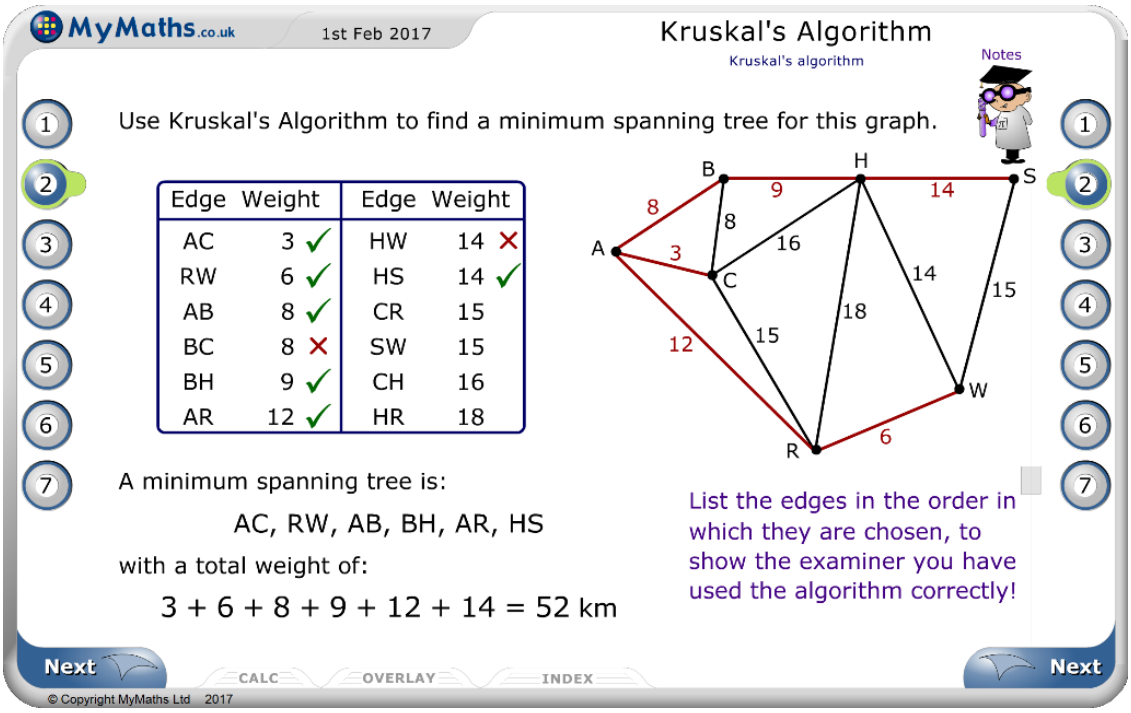
Next CALC OVERLAY INDEX Next

© Copyright MyMaths Ltd 2017

Algorithm run-through (2)



Algorithm run-though (3)



Algorithm run-through (4)

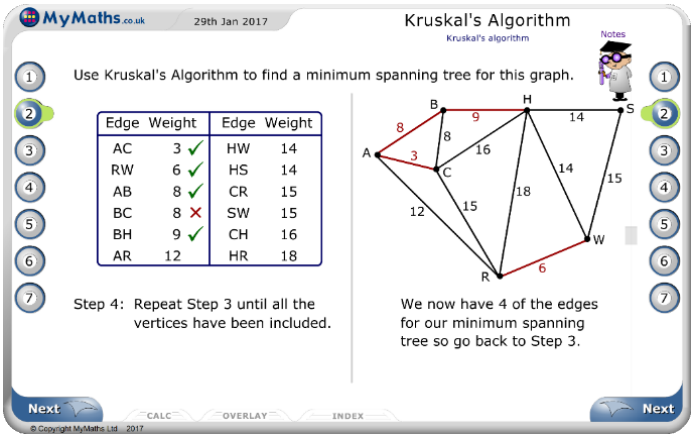
This teaching tool has the following advantages:

1. It has step-by-step illustrations about the algorithm;
2. It has a sample graph showing what the algorithm has done on that step by step, which helps students improve their understanding;
3. It provides objectives and a list of prior knowledge for the users;
4. It uses simple and detailed word to clearly explain the procedure of the algorithm;
5. It is divided into multiple sections, which enables users to select the certain knowledge they want to teach/learn, and skip the unnecessary points for them for their convenience;
6. It manages to emphasise important points by pausing the illustration, implementing animations, and highlighting the text of the important points.

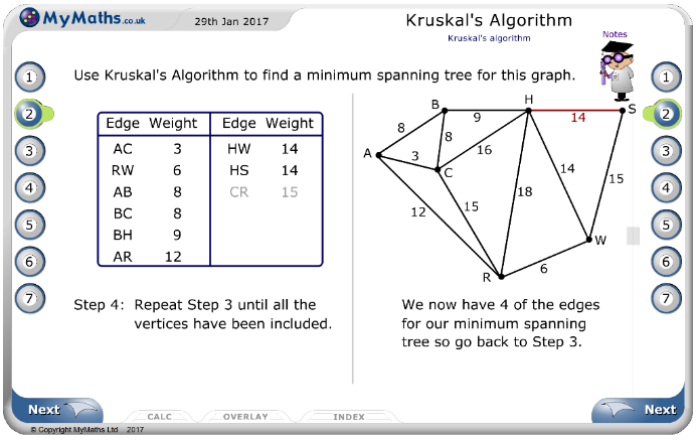
However, this teaching tool also has the following drawbacks:

(In my opinion, points 1 and 2 are considered major drawbacks, and points 3 and 4 are minor drawbacks)

1. **<u>There is only one sample graph which is pre-designed by the teaching tool, and the users cannot edit a graph themselves to reinforce their understandings, i.e., the graph is pre-determined in its pattern and number of vertices and edges;</u>**
2. Bugs occurs when moving the scrollbar – here is one example: if the scrollbar is moved when the previous animation of the table is being played, the animation stops playing, which also disables future display and makes the status of the graph incorrect, as shown in the following screen shots;

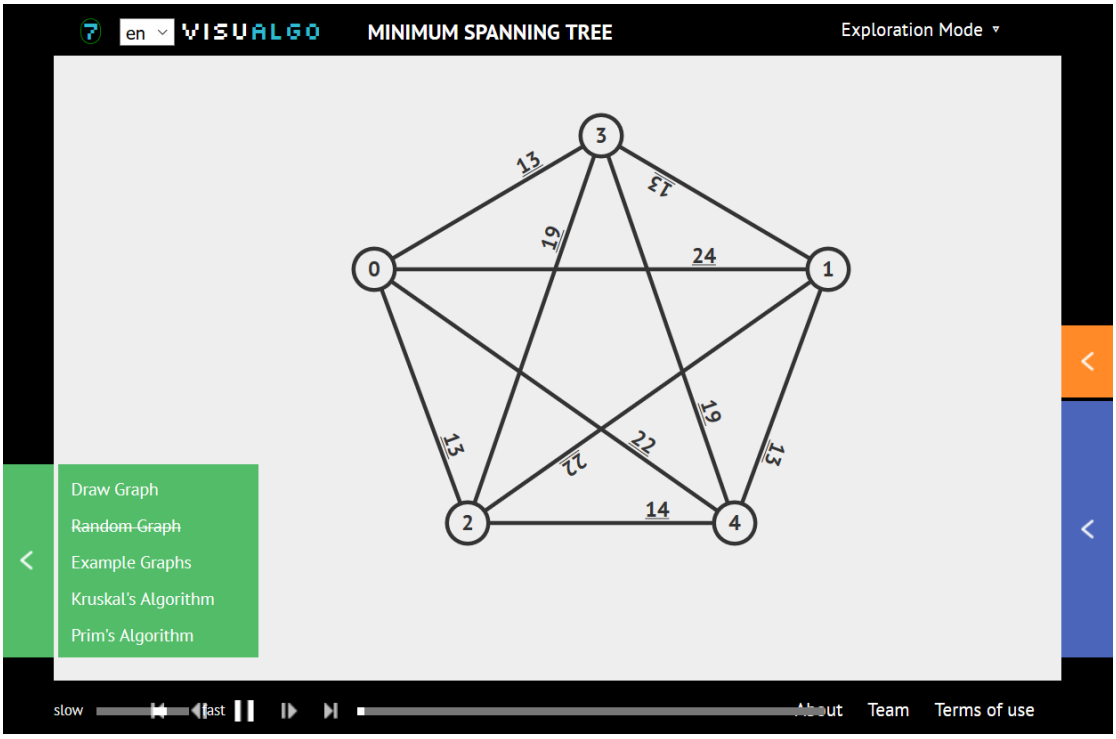


(This is what the page should be as expected)

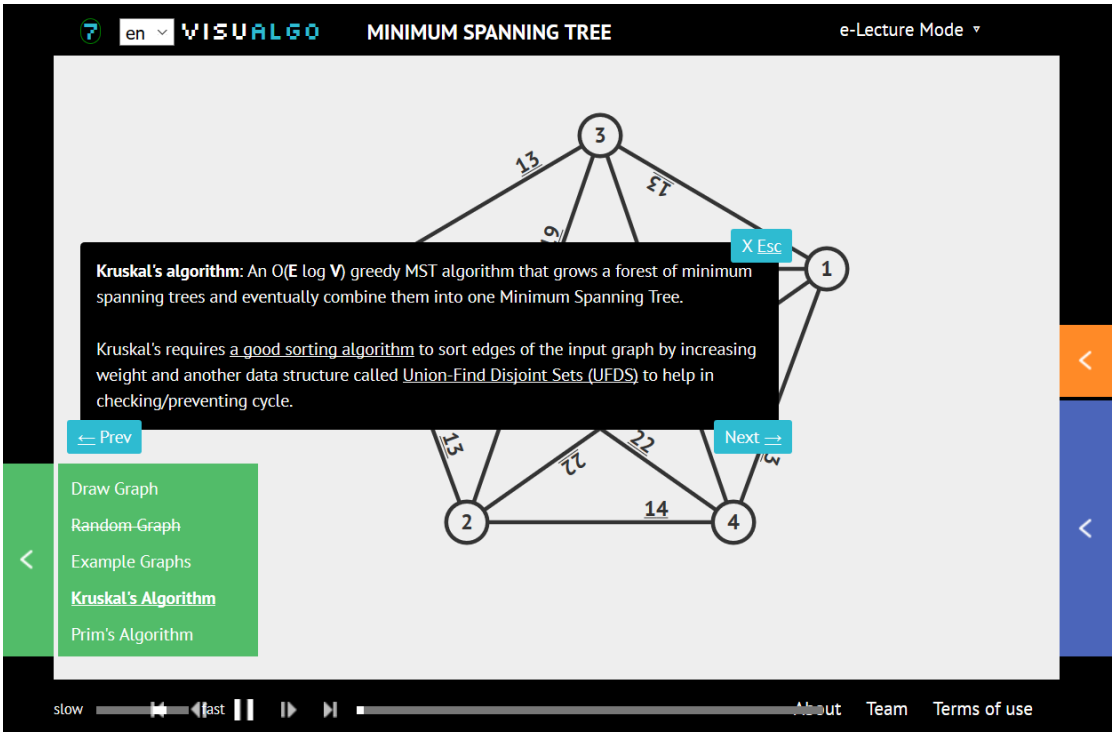


(This is what the page looks like when the above-mentioned bug occurs)

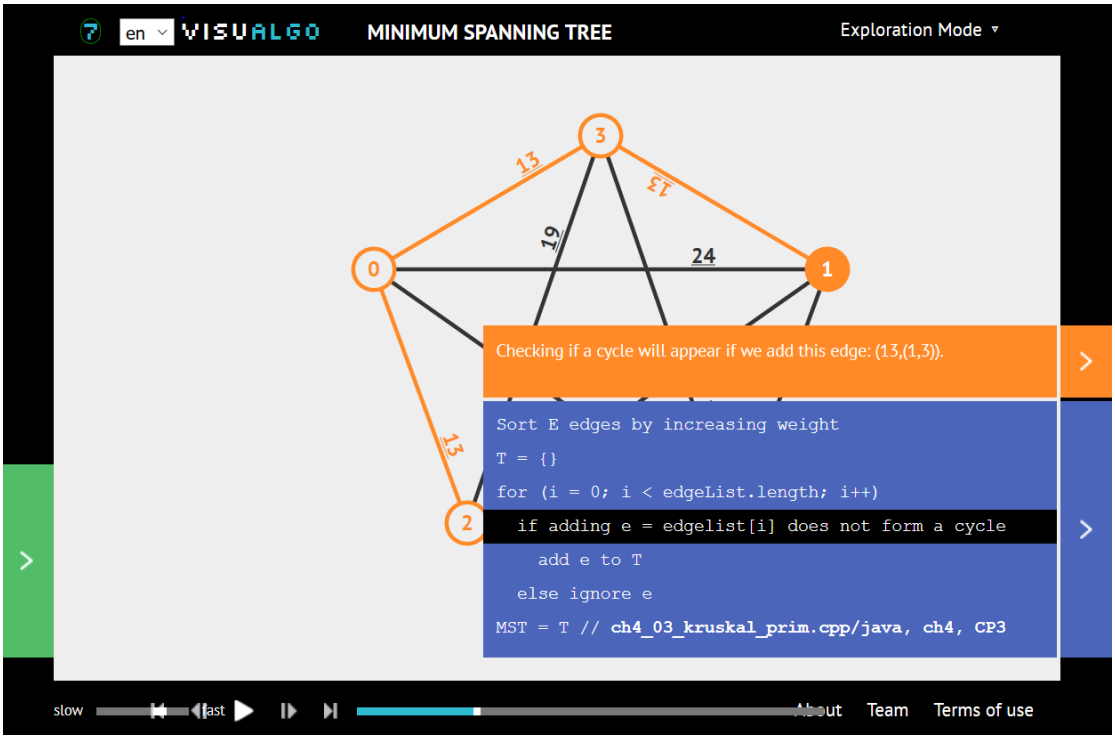
Furthermore, here is the screenshots of another online teaching tool, VisuAlgo[2], on analysing this algorithm:



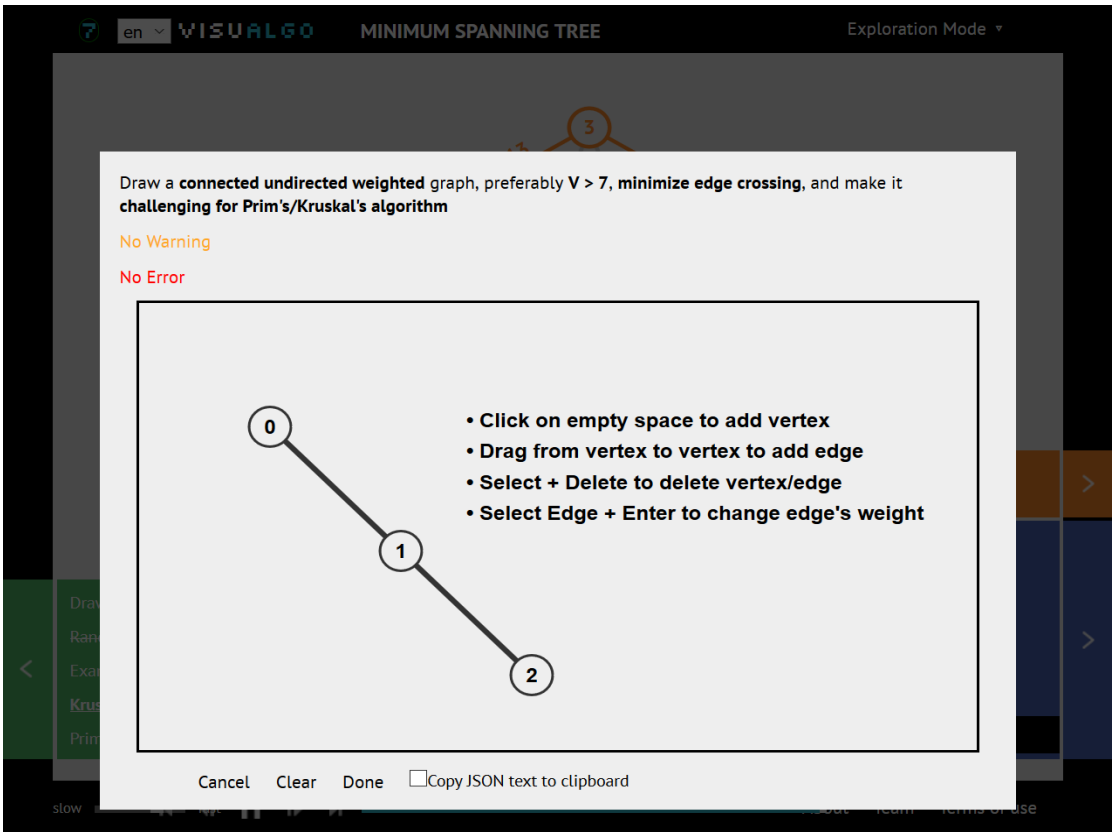
Sample graph



Algorithm description



Algorithm run-through



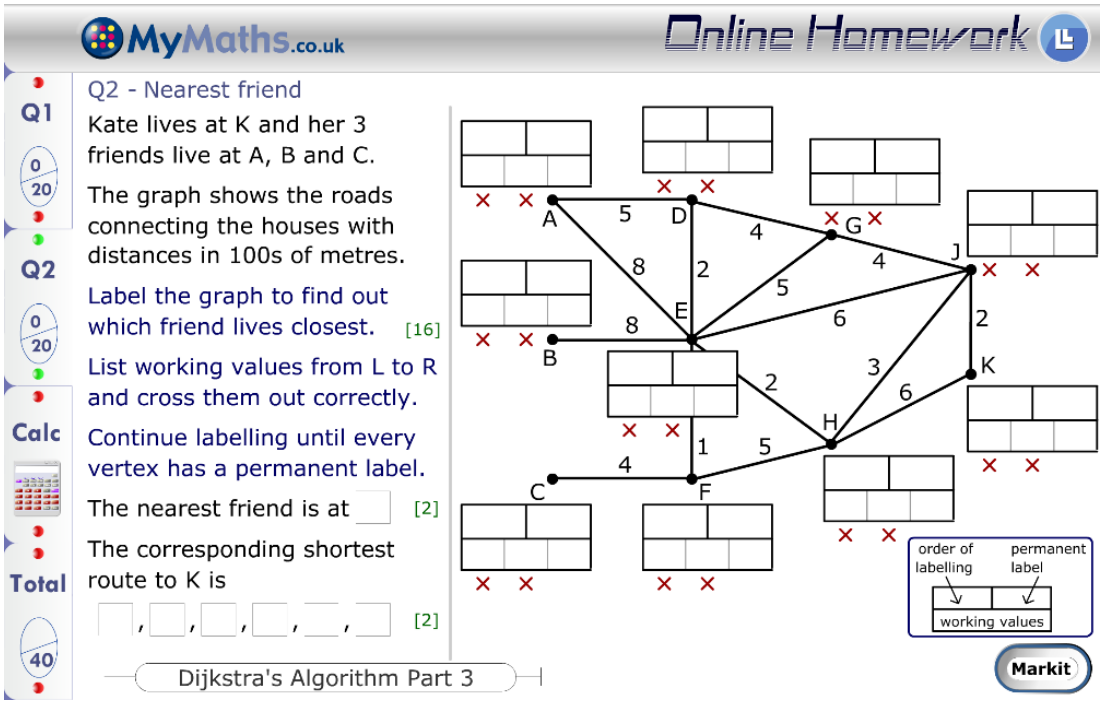
"Draw Graph" section

This teaching tool has the following advantages:

1. It has step-by-step clear visualisation and analysis about the algorithm;
2. It provides pseudo-code run-throughs that helps the understanding of the algorithm;
3. It supports step backwards and the change of animation for the users' convenience;
4. It provides training section, although not aiming at A-Level students, to help users to improve and solidify their understanding of the algorithm;
5. **It has a "Draw Graph" section, which enables the users to create and edit a graph freely.** This is a very good effort, although the operation on the drawing section is quite complicated.

However, this teaching tool also has a drawback: it is not designed for A-Level students, and **it does not provide a practice section for students to do the algorithms by hand, nor any supplementary question to be used as exercise.**

Consider again, the determination of the shortest path from a given vertex of a graph to another. There are many algorithms that can successfully solve this problem, a popular one of which is the Dijkstra's Algorithm, which is also taught in A-Level Decision Mathematics. Here is the homework section of Dijkstra's Shortest Path Algorithm in www.mymaths.co.uk:



Unfortunately, this section of the teaching tool has the following drawbacks:

1. **<u>Users cannot design graphs and questions on their own;</u>**
2. The contents are very cluttered, making it hard for students to clearly understand the problem;
3. The graph is poorly drawn, with the box of vertex E overlapping with the edges EF and EH, and is overall confusing for students to obtain information and completing the boxes;

My project will refer to this as well as other similar teaching tools, and make improvements based on the advantages and drawbacks of those teaching tools.

Main functionalities of the project

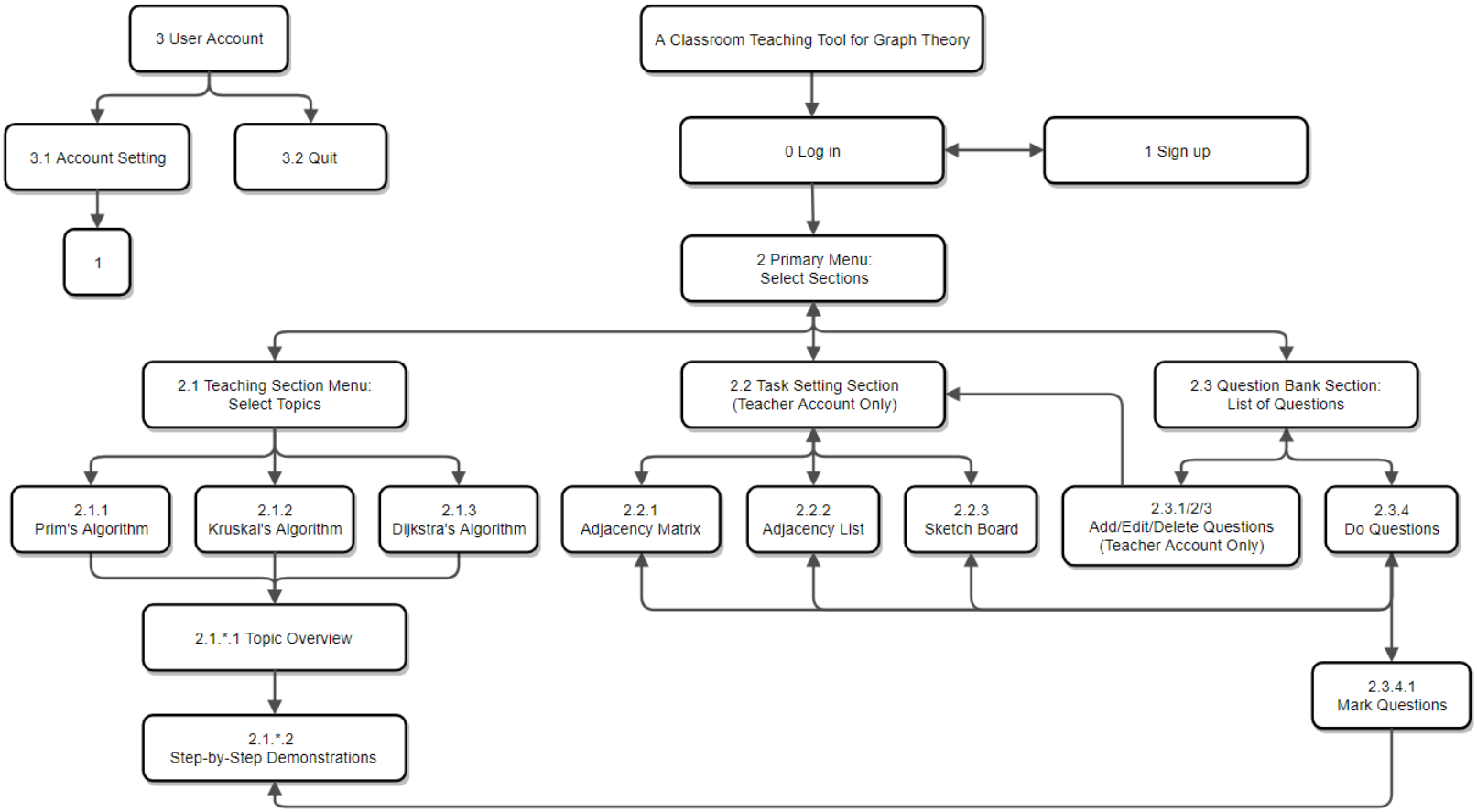
This project will be an offline Windows application with the following functionalities:

1. A thorough and clear explanation for a certain algorithm with step-by-step illustration on sample graphs:
    - Kruskal's Minimum Spanning Tree Algorithm
    - Prim's Minimum Spanning Tree Algorithm
    - Dijkstra's Shortest Path Algorithm
2. A task setting section for teachers to design tasks on their own and set prep to students, with the answers automatically computed by the system. This increases the flexibility and variety of the resources of the questions of each topic.
3. Users are allowed to build a graph on their own in the following three forms:
    - Adjacency Matrix
    - Adjacency List
    - Manually drawing the graph on the Sketch Board

    The user-created graphs will be sent to other sections of the project to work on with, including the step-by-step demonstrations in the Teaching Section, the Task Setting Section, and the Question Bank Section. Turning a real-world problem into a mathematical problem involves abstraction. This is one of the main areas which teachers find difficult to teach and students find difficult to grasp abstractly. This functionality will help students get practice with abstraction by allowing them to create the underlying graph for a given problem.
4. A Question Bank Section for students to practice on past papers and the teachers' self-designed questions, with automatic marking functionality and can help students review the incorrect answers.

Structure of the project

The structure of this project is shown in the following hierarchical diagram:



NB The user account system is a stand-alone module placed at the menu bar of the teaching tool.

Requirements of the project

The requirements of this project are listed in the following tables:

Table 1 – Log in, Sign up and Primary Menu:

Module	Inputs	Processing	Outputs
0 Log in	- 2 textboxes for users to input their usernames and passwords; - A "Log in" button; - A linked table named "New user – sign up" for a new user to sign in for an account. - Hidden labels alongside the textboxes for displaying error messages or alerts (visible only when needed)	**Log in operation:** 1. If there is no input for the username/password, then reject the log in request, with returning the error message "Please enter your username/password!" 2. Query account with the input username in the database; 3. If there is no account in the database that matches the input username, then reject the log in request, with returning the error message "The username you entered does not exist!" 4. Hash the input password, then check the hashed value with the hashed correct password value stored in the account information in the database; 5. If the hashed values do not match (i.e., the input password does not match the stored password), then reject the log in request, with returning the error message "The password you entered is incorrect!" 6. If the input password matched the stored password, then approve the log in request, and show the username on the menu bar: 1) If the account is a teacher account, the system should provide it the functionalities with access authorities for a teacher account; 2) If the account is a student account, the system should provide it the functionalities with access authorities for a student account. *(Details to be followed in the Table 6 – User Account)*	

Module	Inputs	Processing	Outputs
0 Log in *(cont.)*		**Sign up operation:** If the linked label "New user – sign up" is clicked, proceed to the Sign up window *(Reference: Module 1 – Sign up)*	
1 Sign up	If the linked label "New user – sign up" in "Module 0 – Log in" is clicked, proceed to this window. This window includes: - 2 radio buttons for a new user to choose an account type: - Teacher Account - Student Account - Textboxes for a new user to enter their personal information: - Username - Password & Repeat Password - Forename - Surname - Date of Birth - Email - School - A "Sign up" button - Hidden labels that shows error or alert message in need	**Validation: Rejection:** 1. Input username has already been used by other account; 2. Input username is invalid; 3. Input password is invalid; 4. Input password and the input repeat password do not equal; 5. Input for any compulsory information is empty; 6. Input for any personal information is invalid. Rejection: - Output the error message; - Stay in the Sign up window. **Validation: Approval:** No input data is rejected **Accept the sign up request:** - Save the approved new account in the database; - Go to the Log in window. *(Reference: Module 0 – Log in)*	

Module	Inputs	Processing	Outputs
2 Primary Menu	- 3 buttons, each represents a part of the main section: - Teaching Section - Task Setting Section - Homework Section	**Go to the selected part of the main section**	The selected part of the main section

Table 2 – Teaching Section:

Module	Inputs	Processing	Outputs
2.1 Teaching Section Menu: Select Topics	Buttons for different topics: - Module 2.1.1: Primm's Minimum Spanning Tree Algorithm - Module 2.1.2: Kruskal's Minimum Spanning Tree Algorithm - Module 2.1.3: Dijkstra's Shortest Path Algorithm	**Go to the topic overview window for the selected topic**	The topic overview window for the selected topic
2.1.*.1 Topic Overview	**Show the objectives/prerequisites for learning the selected algorithm;** Buttons of the example graphs to carry out the step-by-step demonstrations: - For Prim's algorithm, both graphical and tabular version are supported; - For Dijkstra's algorithm, both directed and undirected graphs are supported.	**Go to the step-by-step demonstration window for the example graph:** If an example graph is selected, go to the step-by-step demonstration window, to perform step-by-step demonstrations of the previously selected algorithm on the selected example graph. *(Reference: Module 2.1.*.2 – Step-by-Step Demonstrations)*	

Module	Inputs	Processing	Outputs
2.1.*.2 Step-by-Step Demonstrations	*(Details vary in different algorithms)* **Provide a full algorithm description, separated by steps;** **Show the selected example graph on the window;**	*(Details vary in different algorithms)* **Step forward:** On clicking the step forward button: - Go to the next step; - Highlight the current step; - Show corresponding text explanations of the step; - Visually show the changes on graph due to the current step of the algorithm; - The teaching section will not proceed unless the step forward/backward button is clicked. **Step backward:** On clicking the step backward button: go to the previous step with exactly the same previous state. **Illustrations on graph:** For each step, visually show the changes on graph due to the current step of the algorithm. This includes highlighting the vertices, edges or weight labels, updating values, and showing the periodical results up to the current step. **User operations on graph:** The diagram should be editable if the algorithm enables the user to choose a random node/edge, such as: choosing a starting/finishing vertex, or choosing from multiple vertices/edges that are equally optimal.	

Module	Inputs	Processing	Outputs
2.1.*.2 Step-by-Step Demonstrations *(cont.)*		**Finishing-up:** When the demonstration has finished (i.e. the algorithm has been proceeded to the final step): - Show the final results; - Show all the necessary information; - Enable the users to go back to a certain step, or start over again. **Step-by-step demonstrations on a user-chosen graph instead of the default example graphs:** All the above-described processing will be unchanged, with only the default text explanations, diagram and dry-run table replaced by those in the form for the user-chosen graph.	

Table 3 – Task Setting Section (Teacher accounts only):

Module	Inputs	Processing & Outputs
2.2 Task Setting Window	- A textbox for users to enter a general description for the question; - Buttons for users to create a graph, in either of the following forms: - Adjacency matrix - Adjacency list - Sketch Board - An "Add Task" button and a "Delete Task" button for users to flexibly add or delete subtasks; - Drop-down menus for users to choose a task from: - Find the minimum spanning tree(s) for the designed graph using Prim's/Kruskal's Algorithm - Find the shortest path from one node to another using Dijkstra's Algorithm - Draw a graph from its adjacency list/matrix, or vice versa - Textboxes for users to set the corresponding starting node and the destination node for the task, if needed; - A "Save" button.	**Create a graph via adjacency matrix:** Proceed to the Edit Adjacency Matrix window. *(References: Module 2.2.1 – Edit Adjacency Matrix)* **Create a graph via adjacency list:** Proceed to the Edit Adjacency List window. *(References: Module 2.2.2 – Edit Adjacency List)* **Create a graph via the Sketch Board:** Proceed to the Sketch Board window. *(References: Module 2.2.3 – Sketch Board)* **Add, edit, and delete a subtask:** Users can flexibly add, edit or delete a subtask. **Validation:** Check if the all the entries of the question is valid. This includes: - Check if the inputs are invalid - Check if the question asks for finding a Minimum Spanning Tree for a directed graph - Check if the saved graph is invalid - Check if the vertices in the subtasks do not exist in the saved graph

Module	Inputs	Processing & Outputs
2.2 Task Setting Window *(cont.)*		**Save the question:** On clicking the "Save" button: (all entries have been validated) 1. Store the question in the database; 2. Store the corresponding graph in the database; 3. Solve the user-set subtasks and store the answer in the database; 4. Go back to the Primary Menu *(Reference: Module 2 – Primary Menu: Select Sections)*
2.2.1 Edit Adjacency Matrix	- A 26×26 table for users to enter the entries of the adjacency matrix; (26 is the maximum number of vertices allowed to be created by the system) - A "Save" button.	**Validation:** Check if the all the entries of the adjacency matrix is valid. This includes: - Check if any of the entries is not a non-negative real number. **Save the graph:** On clicking the "Save" button: (all entries have been validated) - Store the adjacency matrix as an object in the code. - Proceed to the section that had called it. - If it is the Task Setting Window *(Reference: Module 2.2 – Task Setting Window)* that had called it, the adjacency matrix will be stored in the database when the whole question is saved.

Module	Inputs	Processing & Outputs
2.2.2 Edit Adjacency List	- A list of 26 vertices for users to enter the adjacent edges of each vertices in the adjacency list; (26 is the maximum number of vertices allowed to be created by the system) - A "Save" button.	**Validation:** Check if the all the entries of the adjacency list is valid. This includes: - Check if a weight is entered without a preceding vertex name; - Check if an entered weight is not a non-negative number. **Save the graph:** On clicking the "Save" button: (all entries have been validated) - Store the adjacency list as an object in the code. - Proceed to the section that had called it. - If it is the Task Setting Window *(Reference: Module 2.2 – Task Setting Window)* that had called it, the adjacency list will be stored in the database when the whole question is saved.
2.2.3 Sketch Board	- A menu consisting of: - A "Vertex" button - An "Edge" button - A "Tag" button - A plain board for users to design graphs	**Create a vertex:** - On selecting the "Vertex" button, when users single click on the sketch board, a new vertex with default name is created on the location of single click; **Edit a vertex/edge:** - Users can change a vertex's position by dragging that vertex;

Module	Inputs	Processing & Outputs
2.2.3 Sketch Board *(cont.)*		- On selecting the "Tag" button, when users double click on a vertex, a separate temporary window is made visible for the user to change the name and the adjacent edges of the clicked vertex; **Create an edge:** - On selecting the "Edge" button, when users single click on the Sketch Board at a position and drag to another position, a new edge with default name and weight is created between the vertex on the point of mouse down to the vertex on the point of mouse up; - If there is no vertex on either position, then create the vertex; - On clicking and hovering on the "Edge" button, a "Directed Edge" and a "Undirected Edge" button is made visible for users to set the property of the edges; **Save the graph:** - Store the graph as an object in the code. - Proceed to the section that had called it. - If it is the Task Setting Window *(Reference: Module 2.2 – Task Setting Window)* that had called it, the graph will be stored in the database when the whole question is saved.

Module	Inputs	Processing & Outputs
2.2.3 Sketch Board *(cont.)*		***NB*** - Only simple graphs are allowed in the Sketch Board module; - Only edges with positive weights are allowed in the Sketch Board module.

Table 4 – Question Bank Section:

Module	Inputs	Processing & Outputs
2.3 Question Bank Section: List of Questions	**Query the questions from the database:** - A list of all the questions stored in the database **Filter/sort the questions:** - Users should be able to filter the questions, or sort the questions in ascending/descending order, with respect to the following Properties: - Question name - Date modified - Related topic - Problem difficulty	**Add questions: (teacher accounts only)** On clicking the “Add Question” button: Open a new Task Setting Window. *(Reference: Module 2.2 – Task Setting Window)* **Edit questions: (teacher accounts only)** On selecting a question and clicking the "Edit Question" button: Proceed to the Task Setting Window, with the current content of the question loaded in place. *(Reference: Module 2.2 – Task Setting Window)* **Delete questions: (teacher accounts only)** On selecting a question and clicking the "Delete Question" button: Delete the question from the database. **Do questions:** On selecting a question and clicking the "Do Question" button: Proceed to the Do Question window. *(Reference: Module 2.3.4 – Do Questions)*

Module	Inputs	Processing & Outputs
2.3 Question Bank Section: List of Questions *(cont.)*	Buttons for operations on questions: - Add Question - Edit Question - Delete Question - Do Question	
2.3.1 Add Questions (Teacher Accounts Only)	- Open a new Task Setting Window. *(Reference: Module 2.2 – Task Setting Window)* - Once the new question is saved: 1. Store the new question and graph in the database; 2. Solve the new task and store the answer in the database; 3. Go back to the List of Questions window *(Reference: Module 2.3 – List of Questions)*	
2.3.2 Edit Questions (Teacher Accounts Only)	- Proceed to the Task Setting Window, with the current content of the question loaded in place. *(Reference: Module 2.2 – Task Setting Window)* - Once the edited question is saved: 1. Store the new question and graph in the database (overwrite the previous one); 2. Solve the new task and store the answer in the database (overwrite the previous one); 3. Go back to the List of Questions window *(Reference: Module 2.3 – List of Questions)*	
2.3.3 Delete Questions (Teacher Accounts Only)	- Delete the selected question; - Refresh the list of questions. *(Reference: Module 2.3 – List of Questions)*	

Module	Inputs	Processing & Outputs
2.3.4 Do Questions	**Query the content of the question:** - Show the content of the selected question on labels. This includes: - Question name - Problem description - Subtasks - Show the graph of the selected question in the form of adjacency matrix/list, if any, on a table - Show the graph of the selected question, if any, on a picture box Textboxes for users to input their answers when needed A "Mark it" button	**Mark the question:** - When the “Mark it” button is clicked, mark the users-entered answers based on the mark scheme stored in the database (partial marks are allowed) *(Reference: Module 2.3.4.1 – Mark Questions)* - The answers that the users enter can always be changed and re-marked on users’ demands.
2.3.4.1 Mark Questions	For each subtask, provide: - Labels showing the marks awarded for the questions - A “Show Answer” button - A "Step-by-Step Explanation" button	**Show answer:** On clicking the “Show Answer” button, show the answer of the subtask. **Step-by-Step Explanation for a subtask:** On clicking the "Step-by-Step Explanation" button: Proceed to the Step-by-Step Demonstration Module, and apply the needed algorithm on the graph in the question. *(Reference: Module 2.1.*.2 – Step-by-Step Demonstrations)*

Table 5 – User Accounts:

Module	Inputs	Processing & Outputs
Types of Accounts: Access Authorities	A teacher account can set questions via the Task Setting Section, while a student account cannot. A teacher account can add, edit, or delete questions via the Question Bank Section, while a student account cannot. All the other modules can be accessed by both types of accounts.	
3.1 Account Setting	**Query the account information:** Proceed to the Sign up window, with the current account information loaded in place. *(Refrence: Module 1 – Sign up)*	- Users can change all the account information (including the username), provided that they are valid; - Users must retype the password, whether or not they wish to change it. - The validation rule remains the same as the Sign up module; *(Reference: Module 1 – Sign up)* **Update account information:** - Once the updated account information is accepted, update the account information in the database, and proceed to the window where the Account Setting request is called.
3.2 Quit	**Quit the system.**	

Record of feedbacks from target users

In order to be more suitable and satisfactory to the users' requirements, as well as to learn from professional ideas, interviews with several teachers in my school has been conducted.

Below is the list of the interviewee teachers that have participated in the interviews:

- Mr John Cowley (JHC): Head of Mathematics Department of Ellesmere College;
- Mr Peter Hayes (PJH): Teacher of Mathematics of Ellesmere College, in charge of teaching Decision 1 for A-Level further mathematics students;
- Dr Sarah Shakibi (HSS): Head of Computer Science Department and Teacher of Mathematics of Ellesmere College.

The interviewed questions and responses are as follows:

1. **How do you teach graph algorithms, such as minimum spanning trees and Dijkstra's shortest paths, to your students?**
   JHC: Firstly, I would put various of PowerPoints and notes on the board to explain the algorithms. Then, I would show some examples, usually in the textbooks, and apply the algorithm on those examples to the students, and I would let the students to do other examples themselves.
   PJH: I start with the algorithms, discuss the objectives using real-life examples, such as satnav for the shortest path algorithm, and "explain" the algorithm by just following the instructions. I use some good videos/PowerPoints to show the algorithms step by step.
   HSS: I would either have to use the textbook or rely on the things in wiki, which is not very accurate, not very interesting, and not very interactive. There are resources around the world, but there is not a single program that covers all the knowledge.
2. **How do you find those algorithms explained in the Decision mathematics textbooks / Computer Science textbooks?**
   JHC: The current textbook is not the best but it is dedicated to our course. To be honest, I do not think there are enough examples on the book, but there are probably a bigger range of examples on the internet. In the textbook, there were not enough subsidiary questions related to the particular algorithms as the exam does. Besides, what is lacking in the textbook is the understanding outside the algorithm: the textbooks only explain the algorithm itself and does not consider different situation and change in the diagram or algorithm steps.
   PJH: The minimum spanning tree algorithm and the Dijkstra's algorithm are well explained by the textbook. In the heart of the Decision mathematics, those three are fairly straight forward and the applications of the algorithms are clearer. However, the travelling salesman problem is not very well explained.
   HSS: The knowledge in the textbooks is quite dry, and there are not many real-life examples in it. In fact, we are trying to leave the textbook as much as we can. The textbooks are good resources, but they should not be the only resources in the nowadays lessons.
3. **Do you think your students generally response or understand well on those topics?**
   JHC: Some do and some do not.
   PJH: I think students often respond well in those three easy ones (i.e., Kruskal's and Prim's minimum spanning tree algorithms, and Dijkstra's shortest path algorithm), but they do not do really well in the

hard ones (for example, the Hamiltonian cycle one). The biggest problem when I am teaching Decision mathematics is the language for international students. In the previous years, students sometimes choose the wrong algorithms for a question. Now that there is an answer book in the exam, where boxes and matrices are drawn, it is easy for a student to choose the right algorithm.

HSS: For further mathematics students, Decision 1 and Decision 2 are not a problem. However, compared with Mechanics and Statistics, Decision mathematics are very abstract.

4. **Have you ever used computer teaching tools to help you illustrating how the algorithms work? What are they?**

   JHC: No.

   PJH: I use TI-Nspire to teach the Simplex algorithm, but not many on the others.

   HSS: No. There is only a website that compares different algorithms, and to my knowledge, I do not find any of such kind of teaching tools, and even if there are, they will be very expensive and not affordable.

5. **Do you think computer teaching tools will be helpful to students' understanding?**

   JHC: I would think it would, because I think it would give practical examples to the algorithms rather than just some graphs in the exercise.

   PJH & HSS: Definitely yes.

6. **Do you think if there are drawbacks in the current teaching tools you use? What are they? How do you think those teaching tools should be improved?**

   JHC: PowerPoints can be very slow, particularly when they animate the display (I have never written PowerPoints myself, I just use the PowerPoints that I purchase), and the fancy displays are sometimes distractive and too long. I also give every pupil a photocopy of my note, and I put my note on the board simultaneously, so that the students can take notes while I am teaching. In fact, I use this in most of my mathematics teaching.

   PJH: PowerPoints and videos sometimes go a little bit too slowly, and it is impossible to deviate, for example, in Prim's minimum spanning tree algorithm, where the algorithm says select any node, the PowerPoints can only start at a pre-determined node, and when students randomly pick up which node to start, the node may not be prepared in the PowerPoint. I think it can be improved by using a branched PowerPoint instead of a linear one. The advantage of a PowerPoint is the students can take them after class and use them to review their coursework themselves. In fact, the more advanced the mathematics, the fewer resources there are. In lower school, there are a lot of online resources, but in A-Levels, the resources are not very much, because the number of people who can make resources are fewer, and the number of people who need the resources are fewer as well.

   HSS: We need tools for students to start working from scratch, and current teaching tools do not work very smoothly.

7. **Do you think a computer teaching tool where users can design graphs themselves would improve their understanding towards the graph algorithms?**

   JHC: I have never seen such a teaching tool like that, but I would definitely say yes. Hands-on experience is always a good thing in teaching.

   PJH: Probably. You need to be careful about students designing graphs themselves, for sometimes they make wrong connections. You might want to have a set of graphs for students to select from, instead of letting them to design graphs themselves.

   HSS: Very much yes.

8. **If I am going to offer you a computer teaching tool for graph theories and algorithms, what do you want it to have to make it best to help you teaching?**

   JHC: I would like a clear statement of the algorithm, a series of (for example, five or six) examples stating from very simple to more complex ones, and extra supplementary questions. Anything visual is a good thing, so a visual diagram will be very helpful. I would like to see an animated diagram to illustrate each step of an algorithm, so that the students can see what is happening on the algorithm.

   PJH: It should be able to demonstrate the algorithms, set questions (the disadvantage of the textbook is there is not enough questions), and track student's understanding. Ideally, it should enable the users to skip steps in case the whole process goes too slow.

   HSS: These are my general requirements for it:

   1) It should have a friendly user interface with login menu for teachers and students (both can use same entry menu);
   2) It should have a teaching module for students focussing on the basics of graph theory – students should be able to see examples of basic graphs (of all types) and be shown how to construct these from the adjacency list or matrix;
   3) It should have an exercise area where students can then practice building graphs *themselves* using the adjacency list or matrix for a given graph generated by the system;
   4) It should have a second teaching module focussing on the basics of three optimisation algorithms: Prim's, Kruskal's, Dijkstra's;
   5) Students should be able to see clearly laid out demonstrations of each algorithm on a not very complex graph;
   6) It should have an exercise area where students can then practice solving problems for a given optimisation algorithm;

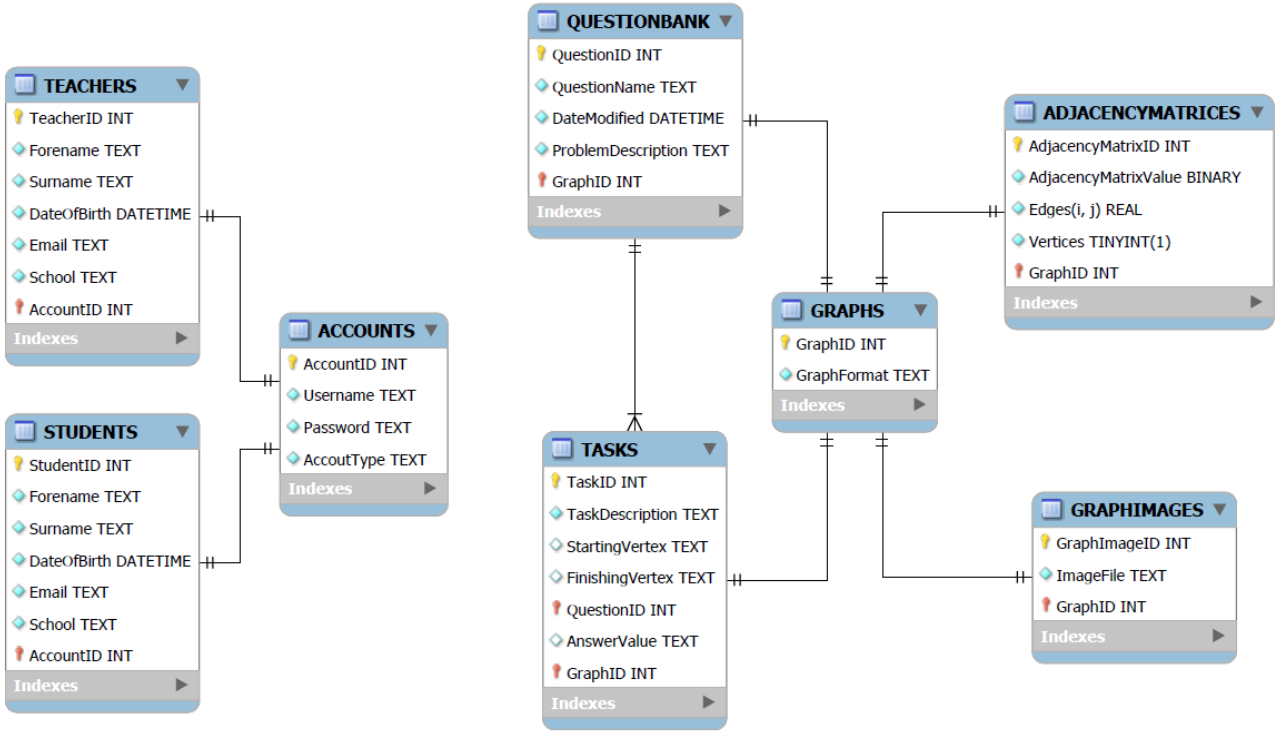
Based on those responses by the teachers, several pivotal conclusions with regards to the requirements of the computer teaching tool can be drawn:

1. The computer teaching tool should provide a substantial number of examples and exercises for its users.
2. The computer teaching tool should be able to use an animated, visual diagram to illustrate the process of the algorithms to help the students' understanding.
3. The computer teaching tool should be as brief as possible, and eliminate unnecessary and distractive animations, provided that a clear, essential statements of the algorithms are given. The computer teaching tool should also enable its users to skip steps for their convenience.

Documented Design

Back-end design

This project will use a relational database whose schema is shown by the following diagram (produced by MySQL Workbench[3])

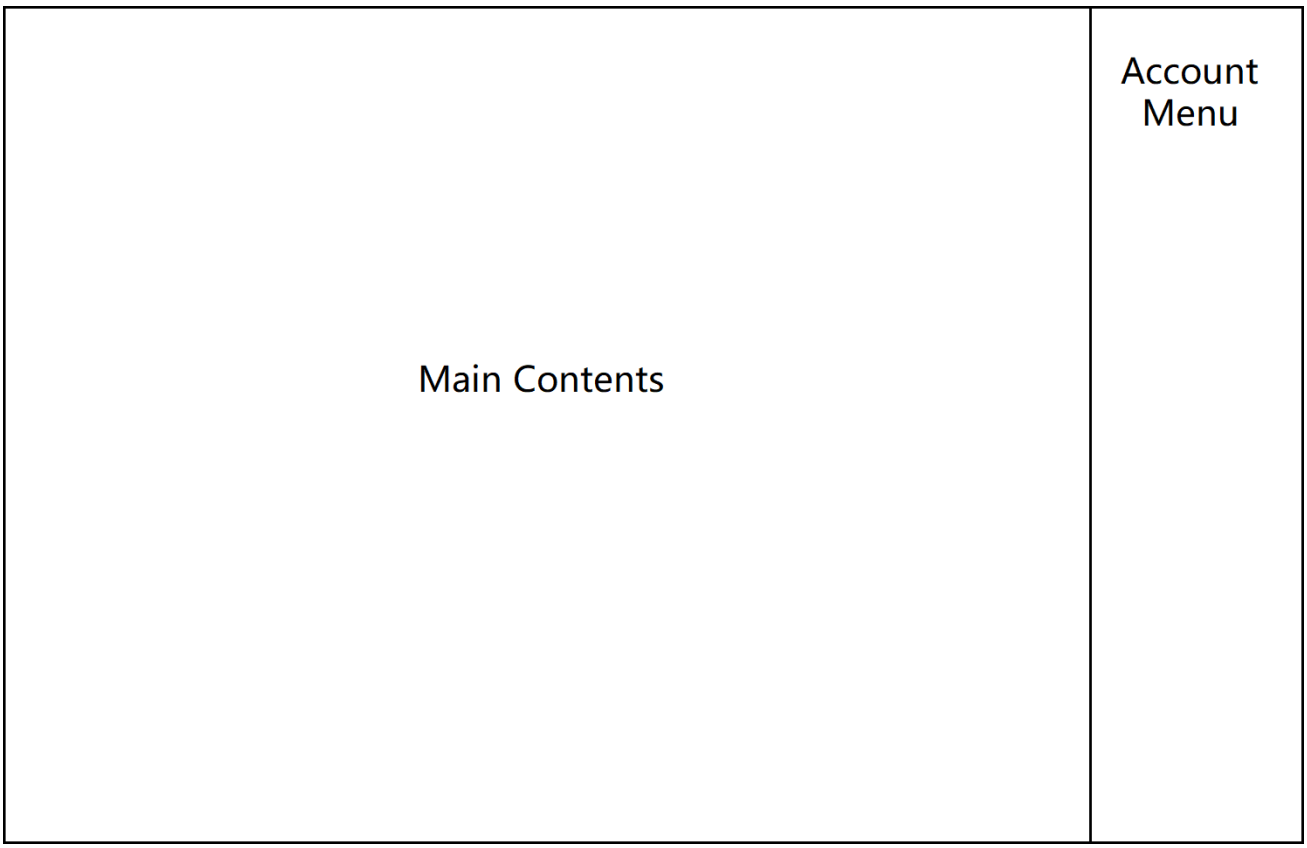


NB

- represents a not null attribute;
- represents a nullable attribute;
- represents a primary key attribute;
- represents a foreign key attribute;
- ADJACENCYMATRICES.Edges(i, j) represents 26×26=676 attributes in the actual design of the table ADJACENCYMATRICES, recording the weights between each two vertices (and equals 0 if there is no edge between two vertices);
- ADJACENCYMATRICS.Vertices represents 26 Boolean attributes in the actual design of the table ADJACENCYMATRICES, recording if each vertex is used in the graph.

GUI design

This is the general structure of the GUI design of each page of this project:



The structures of GUI designs for different pages varies, for example:

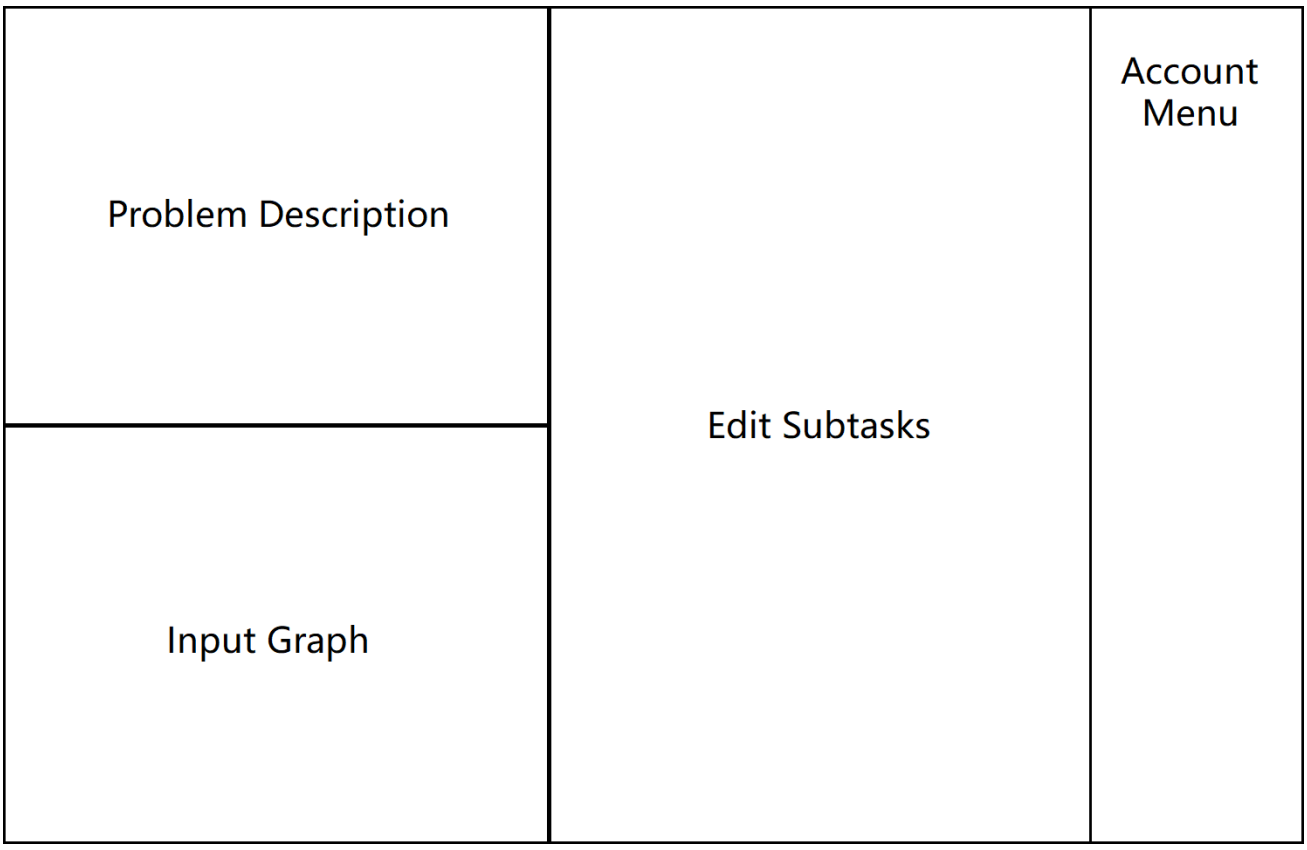
Algorithm Topic Overview:

Algorithm Objectives and Prerequisites	Example buttons	Account Menu

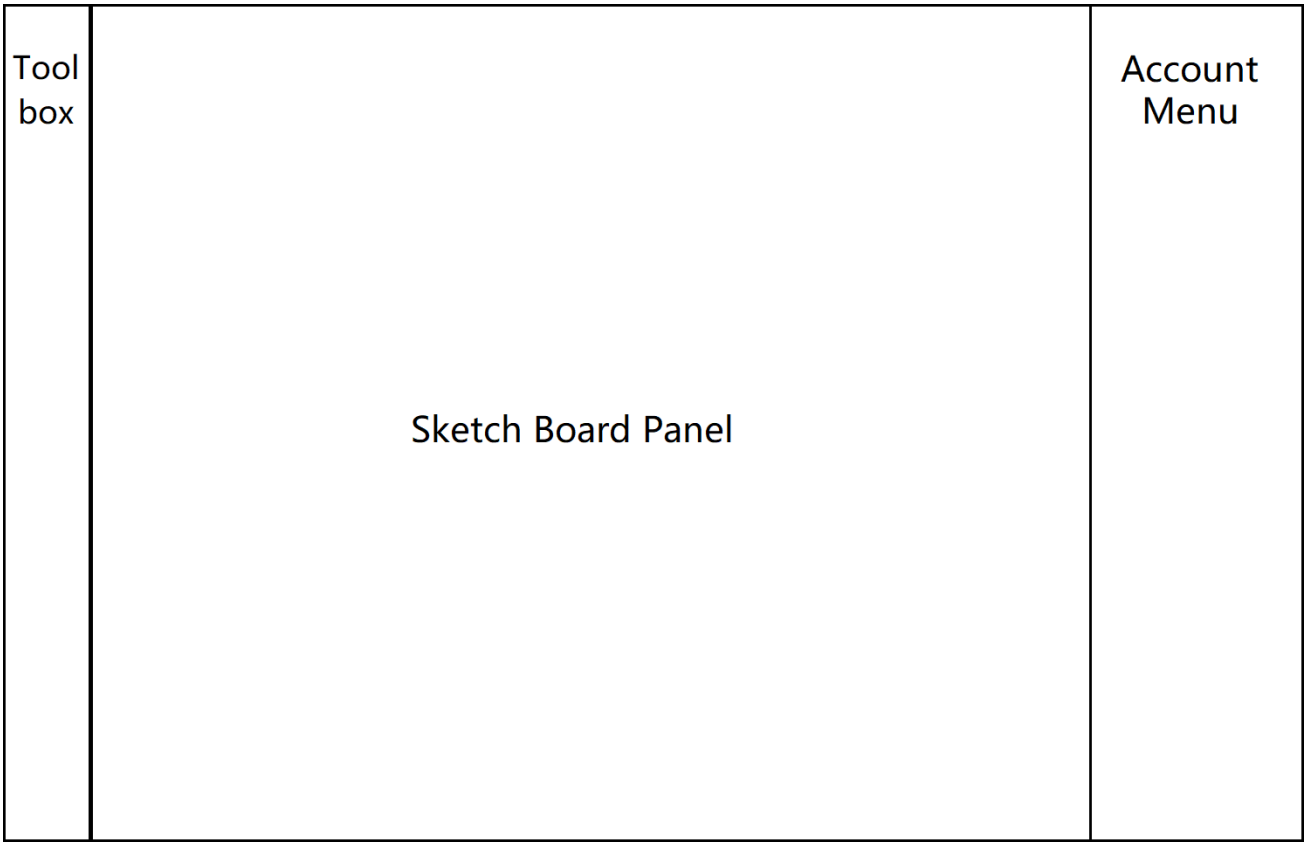
Algorithm Step-by-Step Demonstration:

Algorithm Procedures	Example Graph	Account Menu

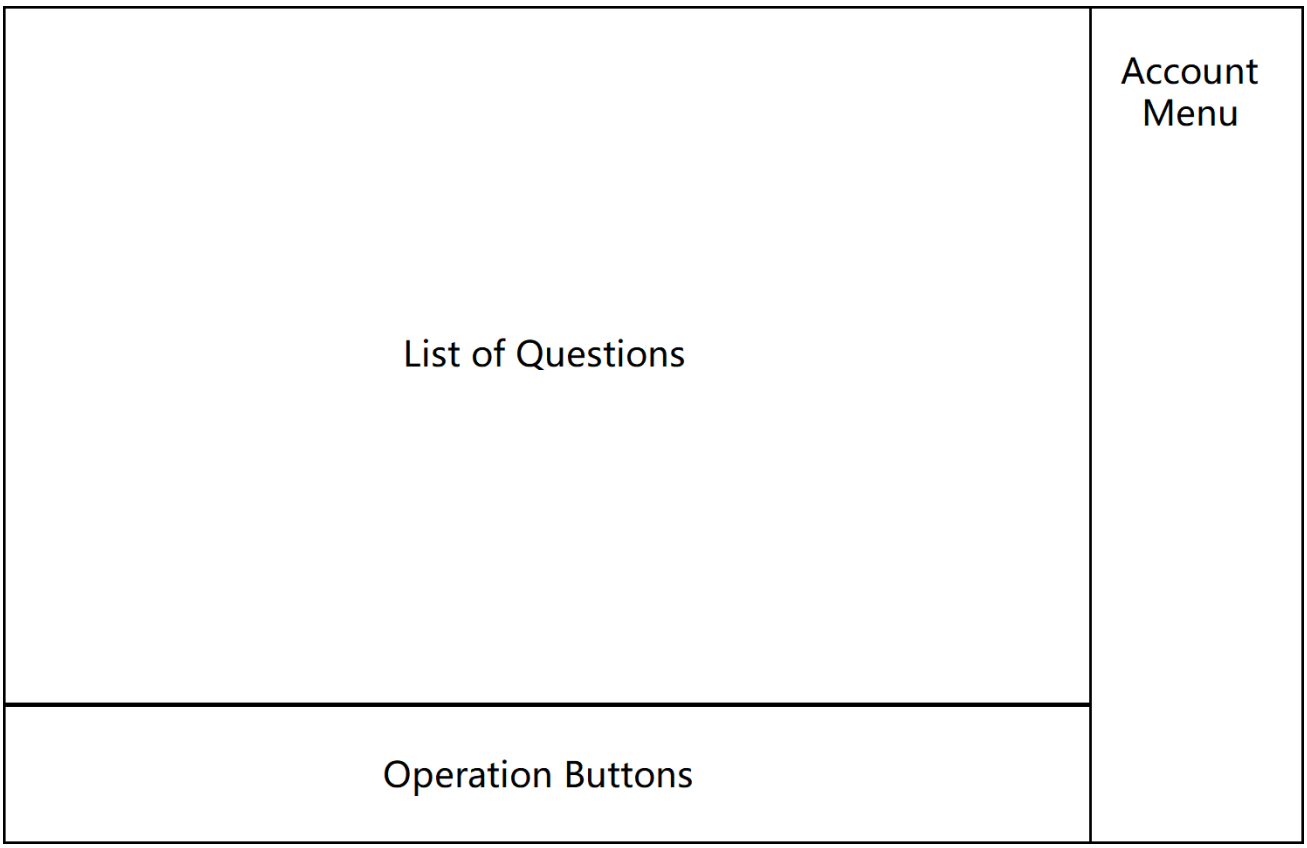
Task Setting Page:



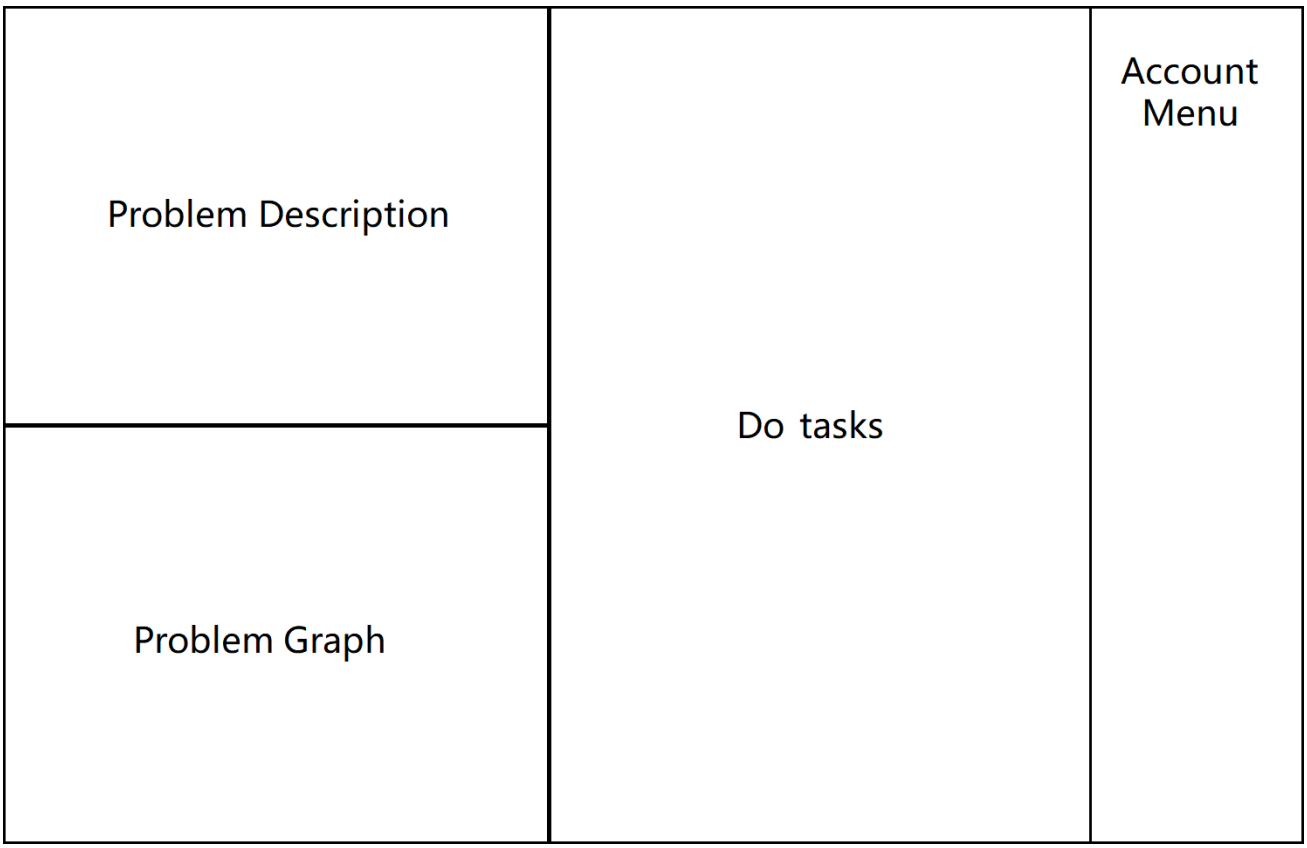
Sketch Board:



Question Bank:



Do Question Page:



The exact GUI implementation of the system is shown in **Appendix 1 - GUI implementation.pdf**.
It reflects the majority of the modules.

`Vertex` control

This user control contains:

- A round, transparent region with a solid black border
- A label `labelName` in the middle of the region showing the name of the vertex

Sample design:

Universally supported operations: (There are other operations that work on certain modules only)

- Drag to change the location of the vertex
- Press BACKSPACE or DELETE to delete this vertex, along with all the edges connected to it

Universal events: (There are other events that work on certain modules only)

- Paint
- MouseDown
- MouseMove
- MouseUp
- KeyPress

Pseudo-code

```
PUBLIC CLASS Vertex INHERITS UserControl
    DEFINE PRIVATE STRUCT AdjacentEdge
        vertex: Vertex
        weight: REAL
    END STRUCT
    DEFINE PRIVATE adjacentEdges: LIST<AdjacentEdge>
    DEFINE PRIVATE selected: BOOLEAN
    DEFINE PRIVATE clicked: BOOLEAN
    DEFINE PRIVATE selectable: BOOLEAN
    DEFINE PRIVATE draggable: BOOLEAN

    PUBLIC CONSTRUCTOR Vertex(STRING name, INTEGER x, INTEGER y)
        THIS.Name ← "vertex" + name
        THIS.Location ← NEW POINT(x - THIS.Width / 2, y - THIS.Height / 2)
        THIS.labelName.Text ← name
    END CONSTRUCTOR

    PUBLIC CONSTRUCTOR Vertex(STRING name, POINT centre)
        THIS.Name ← "vertex" + name
        THIS.Location ← NEW POINT(centre.X - THIS.Width / 2,
                                  centre.Y - THIS.Height / 2)
        THIS.labelName.Text ← name
    END CONSTRUCTOR

    PUBLIC FUNCTION POINT GetCentreLocation()
        DEFINE x, y: INTEGER
        x ← THIS.Location.X + THIS.Width / 2
        y ← THIS.Location.Y + THIS.Height / 2
        RETURN NEW POINT(x, y)
    END FUNCTION
```

```
PUBLIC FUNCTION CHAR GetName()
    RETURN THIS.Name.Trim("vertex")
END FUNCTION

PUBLIC FUNCTION BOOLEAN IsSelected()
    RETURN THIS.selected
END FUNCTION

PUBLIC FUNCTION BOOLEAN IsClicked()
    RETURN THIS.clicked
END FUNCTION

PUBLIC FUNCTION BOOLEAN IsSelectable()
    RETURN THIS.selectable
END FUNCTION

PUBLIC FUNCTION BOOLEAN IsDraggable()
    RETURN THIS.draggable
END FUNCTION

PUBLIC FUNCTION REAL GetDistance(Vertex v)
    DEFINE centre1, centre2 : POINT
    centre1 ← v.GetCentreLocation()
    centre2 ← THIS.GetCentreLocation()
    RETURN √(centre1.X - centre2.X)^2 + (centre1.Y - centre2.Y)^2
END FUNCTION

PUBLIC FUNCTION BOOLEAN ContainsEdge(Vertex v)
    FOREACH AdjacentEdge edge IN THIS.adjacentEdges
        IF edge.vertex = v
            RETURN TRUE
        END IF
    END FOR
    RETURN FALSE
END FUNCTION

PUBLIC FUNCTION REAL GetEdge(Vertex v)
    FOREACH AdjacentEdge edge IN THIS.adjacentEdges
        IF edge.vertex = v
            RETURN edge.weight
        END IF
    END FOR
    RETURN 0
END FUNCTION

PUBLIC FUNCTION LIST<Vertex> GetEdges()
    DEFINE output: LIST<Vertex>
    FOREACH AdjacentEdge edge IN THIS.adjacentEdges
        output.Add(edge.vertex)
    END FOR
    RETURN output
END FUNCTION

PUBLIC FUNCTION INTEGER GetNumberIndex()
    RETURN THIS.GetName() - 'A'
END FUNCION
```

```
PUBLIC FUNCTION VOID SetName(CHAR name)
    THIS.Name ← "vertex" + name
    THIS.labelName.Text ← name
END FUNCTION

PUBLIC FUNCTION VOID SetSelected(BOOLEAN status)
    THIS.selected ← status
END FUNCTION

PUBLIC FUNCTION VOID SetClicked(BOOLEAN status)
    THIS.clicked ← status
END FUNCTION

PUBLIC FUNCTION VOID SetSelectable(BOOLEAN status)
    THIS.selectable ← status
END FUNCTION

PUBLIC FUNCTION VOID SetDraggable(BOOLEAN status)
    THIS.draggable ← status
END FUNCTION

PUBLIC FUNCTION VOID SetEdge(Vertex v, REAL weight)
    DEFINE tempEdge, edgeToRemove: AdjacentEdge
    tempEdge.vertex ← v
    tempEdge.weight ← weight
    FOREACH AdjacentEdge edge IN THIS.adjacentEdges
        IF edge.vertex = v
            edgeToRemove ← edge
        END IF
    END FOR
    THIS.adjacentEdges.Remove(edgeToRemove)
    THIS.adjacentEdges.Add(tempEdge)
END FUNCTION

PUBLIC FUNCTION BOOLEAN SetEdge(Vertex v)
    SetEdge(v, 1)
END FUNCTION

PUBLIC FUNCTION VOID RemoveEdge(Vertex v)
    DEFINE edgeToRemove: AdjacentEdge
    FOREACH AdjacentEdge edge IN adjacentEdges
        IF edge.vertex = v
            edgeToRemove ← edge
        END IF
    END FOR
    adjacentEdges.Remove(edgeToRemove)
END FUNCTION

EVENT Vertex_Paint
    IF THIS.IsSelected() = TRUE
        <Highlight the border>
    ELSE
        <Do not highlight the border>
    END IF
END EVENT
```

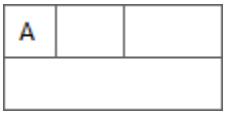
```
    EVENT Vertex_MouseDown
        IF THIS.IsSelectable() = TRUE
            THIS.SetSelected(NOT THIS.IsSelected())
        END IF
        THIS.Refresh()
        THIS.SetClicked(TRUE)
    END EVENT

    EVENT Vertex_MouseMove
        IF THIS.IsClicked() = TRUE AND THIS.IsDraggable() = TRUE
            THIS.SetSelected(TRUE AND THIS.IsSelectable())
            DEFINE x, y: INTEGER
            x ← THIS.Location.X + MOUSE_CLICK_POSITION.X - THIS.Width / 2
            y ← THIS.Location.Y + MOUSE_CLICK_POSITION.Y - THIS.Height / 2
            THIS.Location ← NEW POINT(x, y)
        END IF
    END EVENT

    EVENT Vertex_MouseUp
        THIS.SetClicked(FALSE)
    END EVENT

    EVENT Vertex_KeyPress
        IF KEY_VALUE = DELETE OR KEY_VALUE = BACKSPACE
            THIS.Dispose()
        END IF
    END EVENT
END CLASS
```

`DijkstraVertexLabel` control

This user control contains:

- 4 boxes, each containing one of the 4 key values:
  - Vertex name
  - Order of labelling
  - Permanent label
  - Working values (temporary label)
- A label `labelVertexName` in the top-left box showing the name of the vertex
- A textbox `textBoxOrder` in the top-middle box showing the order of labelling
- A textbox `textBoxFinalLabel` in the top-right box showing the permanent label
- A textbox `textboxWorkingValues` in the bottom box showing the working values

Sample design:



Supported operations:

- Read and write the value of the order of labelling
- Read and write the value of the permanent label
- Read and write the value of the temporary label
- Highlight and un-highlight itself on demand in Step-by-Step Demonstration mode

Events:

No event needed.

Pseudo-code

```
PUBLIC CLASS DijkstraVertexLabel INHERITS UserControl
    PUBLIC CONSTRUCTOR DijkstraVertexLabel(CHAR vertexName, POINT location)
        THIS.labelVertexName.Text ← vertexName
        THIS.Location ← location
    END CONSTRUCTOR

    PUBLIC FUNCTION CHAR GetVertexName()
        RETURN THIS.labelVertexName.Text
    END FUNCTION

    PUBLIC FUNCTION INTEGER GetNumberIndex()
        RETURN THIS.GetVertexName() - 'A'
    END FUNCTION

    PUBLIC FUNCTION INTEGER GetLabellingOrder()
        IF THIS.textBoxOrder.Text = NULL
            RETURN -1
        ELSE
            RETURN THIS.textBoxOrder.Text
        END IF
    END FUNCTION
```

```
PUBLIC FUNCTION REAL GetFinalLabel()
    IF THIS.textBoxFinalLabel.Text = NULL
        RETURN -1
    ELSE
        RETURN THIS.textBoxFinalLabel.Text
    END IF
END FUNCTION

PUBLIC FUNCTION STRING GetWorkingValues()
    RETURN THIS.textBoxWorkingValues.Text
END FUNCTION

PUBLIC FUNCTION POINT GetCentreLocation()
    DEFINE x, y: INTEGER
    x ← THIS.Location.X + THIS.Width / 2
    y ← THIS.Location.Y + THIS.Height / 2
    RETURN NEW POINT(x, y)
END FUNCTION

PUBLIC FUNCTION REAL GetDistance(DijkstraVertexLabel v)
    DEFINE centre1, centre2 : POINT
    centre1 ← v.GetCentreLocation()
    centre2 ← THIS.GetCentreLocation()
    RETURN √(centre1.X - centre2.X)^2 + (centre1.Y - centre2.Y)^2
END FUNCTION

PUBLIC FUNCTION VOID SetReadOnly(BOOLEAN status)
    textBoxFinalLabel.ReadOnly ← status
    textBoxWorkingValues.ReadOnly ← status
    textBoxOrder.ReadOnly ← status
END FUNCTION

PUBLIC FUNCTION VOID SetVertexName(CHAR vertexName)
    THIS.labelVertexName.Text ← vertexName
END FUNCTION

PUBLIC FUNCTION VOID SetLabellingOrder(INTEGER order)
    THIS.textBoxOrder.Text ← order
END FUNCTION

PUBLIC FUNCTION VOID SetFinalLabel(REAL distance)
    THIS.textBoxFinalLabel.Text ← distance
END FUNCTION

PUBLIC FUNCTION VOID SetWorkingValues(STRING workingValues)
    THIS.textBoxWorkingValues.Text ← workingValues
END FUNCTION

PUBLIC FUNCTION VOID UpdateWorkingValues(REAL workingValue)
    THIS.textBoxWorkingValues.Text += workingValue + " "
END FUNCTION

PUBLIC FUNCTION VOID Finalise(REAL distance, INT order)
    SetLabellingOrder(order)
    SetFinalLabel(distance)
END FUNCTION
```

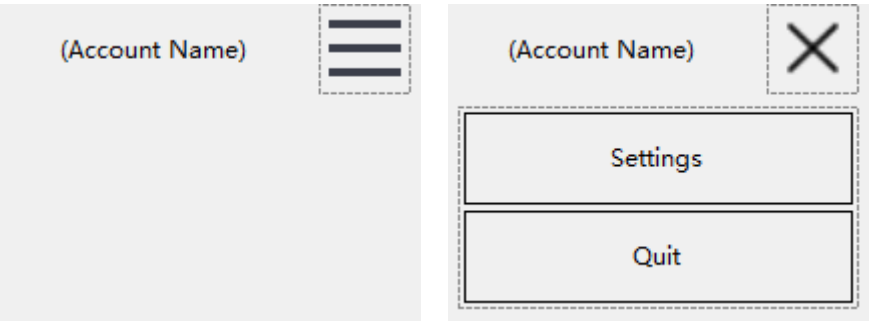
```
    PUBLIC FUNCTION VOID FocusOn()
        <Highlight THIS.labelVertexName>
        <Highlight THIS.textBoxOrder>
        <Highlight THIS.textBoxFinalOrder>
        <Highlight THIS.textBoxWorkingValues>
    END FUNCTION

    PUBLIC FUNCTION VOID FocusOff()
        <Do not highlight THIS.labelVertexName>
        <Do not highlight THIS.textBoxOrder>
        <Do not highlight THIS.textBoxFinalOrder>
        <Do not highlight THIS.textBoxWorkingValues>
    END FUNCTION
END CLASS
```

`AccountMenu` control

This user control contains:

- A label `labelAccountName` showing the name of the user account
- A picturebox `pictureBoxAccountOptions` that maintains whether `panelAccountOptions` should be shown or hidden
- A panel `panelAccountOptions` (hidden by default) showing the two account options of the user account when a user clicks `pictureBoxAccountOptions`:
    - A button `buttonAccountSettings` inside `panelAccountOptions` that opens the Sign up window for the Account Setting functionality
      (Refrence: Module 1 – Sign up, Module 3.1 – Account Setting)
    - A button `buttonQuit` inside `panelAccountOptions` that quits the system when clicked
      (Refrence: Module 3.2 – Quit)

Sample design:



Supported operations:

- Click `pictureBoxAccountOptions` to show or hide `panelAccountOptions`
- Click `buttonAccountSettings` to change the account information
- Click `buttonQuit` to quit the system

Events:

- `pictureBoxAccountOptions`: Click
- `buttonAccountSettings`: Click
- `buttonQuit`: Click
- External events called by this user control:
    - `WindowSignUp.buttonSignUp`: Click
    - `WindowSignUp`: Closed

Pseudo-code

```
PUBLIC CLASS AccountMenu INHERITS UserControl

    DEFINE PUBLIC accountID: INTEGER
    DEFINE PUBLIC username: STRING
    DEFINE PUBLIC accountType: STRING
    DEFINE PRIVATE sql: SQL_COMMAND
    DEFINE PRIVATE reader: SQL_DATA_READER
    DEFINE PRIVATE windowSignUp: WindowSignUp
```

```
PUBLIC CONSTRUCTOR Vertex(INTEGER accountID, STRING username,
                          STRING accountName, STRING accountType)
    THIS.accountID ← accountID
    THIS.userName ← username
    THIS.labelAccountName.Text ← accountName
    THIS.accountType ← accountType
END CONSTRUCTOR

EVENT PictureBoxAccountOptions_Click
    THIS.panelAccountOptions.Visible ← NOT THIS.panelAccountOptions.Visible
    IF THIS.panelAccountOptions.Visible = TRUE
        THIS.pictureBoxAccountOptions.Image ←
    ELSE
        THIS.pictureBoxAccountOptions.Image ←
    END IF
END EVENT

EVENT ButtonAccountSettings_Click
    THIS.ParentWindow.Hide()
    windowSignUp ← NEW WindowSignUp()
    windowSignUp.Closed.Add(NEW EVENT(WindowSignUp_WindowClosed))
    windowSignUp.buttonSignUp.Click.Add
        (NEW EVENT(WindowSignUp_ButtonSignUp_Click))
    sql ← <SQL 3.1_1 - Query current account information>
    reader ← DATABASE.ExecuteCommand(sql)
    reader.ReadNext()
    windowSignUp.textBoxUserName.Text ← reader["Username"]
    IF accountType = "TEACHER"
        windowSignUp.radioButtonTeacher.Checked ← TRUE
        windowSignUp.radioButtonStudent.Checked ← FALSE
    ELSE // IF accountType = "STUDENT"
        windowSignUp.radioButtonTeacher.Checked ← FALSE
        windowSignUp.radioButtonStudent.Checked ← TRUE
    END IF
    windowSignUp.radioButtonTeacher.Enabled ← FALSE
    windowSignUp.radioButtonStudent.Enabled ← FALSE
    windowSignUp.textBoxForename.Text ← reader["Forename"]
    windowSignUp.textBoxSurname.Text ← reader["Surname"]
    windowSignUp.textBoxDateOfBirth.Text ← reader["DateOfBirth"]
    windowSignUp.textBoxEmail.Text ← reader["Email"]
    windowSignUp.textBoxSchool.Text ← reader["School"]
END EVENT

EVENT WindowSignUp_ButtonSignUp_Click
    IF WindowSignUp.ValidateSignUp(username) = TRUE
        sql ← <SQL 3.1_2 - Update account credentials>
        DATABASE.ExecuteCommand(sql)
        sql ← <SQL 3.1_3 - Update personal information>
        DATABASE.ExecuteCommand(sql)
        WindowSignUp.Close()
    END IF
END EVENT
```

```
    EVENT WindowSignUp_Closed
        THIS.ParentWindow.Show()
    END EVENT

    EVENT ButtonQuit_Click
        <Exit the application>
    END EVENT
END CLASS
```

`TaskSettingControls` class

This class is a collection of user controls, to be used in doing a subtask in the Task Setting Section. (*Reference: Module 2.2 – Task Setting Window*) It contains:

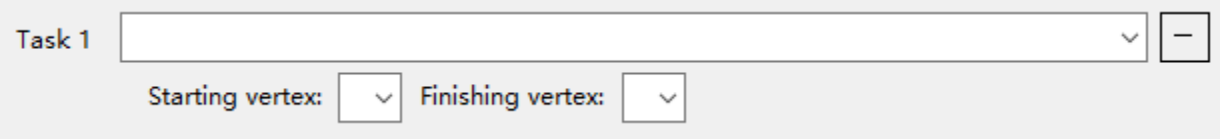
User controls:

- A label `labelTaskIndex` showing the task number;
- A combobox `comboBoxTask` that enables users to select an appropriate task
- A button `buttonRemoveTask` for users to remove the task;
- A label `labelStartingVertex` indicating the users to enter a starting vertex, if required by the content of the task;
- A combobox `comboBoxStartingVertex` that enables users to select a starting vertex
- A label `labelFinishingVertex` indicating the users to enter a finishing vertex, if required by the content of the task;
- A combobox `comboBoxFinishingVertex` that enables users to select a finishing vertex

Variables:

A constant dictionary<string, string> `tasks` that sets a reference between the topic of the task and its content.

Sample design:



Events:

- `comboBoxTask` : TextChanged: this will show/hide the starting/finishing vertex based on the selected task by the users;
- `buttonRemoveTask` : Click: this will remove the task on this unit, as well as disposing the entire user control.

Pseudo-code

```
PUBLIC CLASS TaskSettingControls
    DEFINE PUBLIC labelTaskIndex: LABEL
    DEFINE PUBLIC comboBoxTask: COMBOBOX
    DEFINE PUBLIC buttonRemoveTask: BUTTON
    DEFINE PUBLIC labelStartingVertex: LABEL
    DEFINE PUBLIC comboBoxStartingVertex: COMBOBOX
    DEFINE PUBLIC labelFinishingVertex: LABEL
    DEFINE PUBLIC comboBoxFinishingVertex: COMBOBOX
    DEFINE PUBLIC CONSTANT tasks: DICTIONARY<STRING, STRING>
    tasks ← { ("Prim", "Find the Minimum Spanning Tree using Prim's algorithm"),
              ("Kruskal", "Find the Minimum Spanning Tree using Kruskal's algorithm"),
              ("Dijkstra", "Find the Shortest Path using Dijkstra's algorithm"),
              ("Graph", "Draw the graph corresponding to the adjacency list/matrix"),
              ("Matrix", "Write the graph in adjacency matrix"),
              ("List", "Write the graph in adjacency list")}
```

```
    PUBLIC CONSTRUCTOR TaskSettingControls(INTEGER index)
        THIS.labelTaskIndex.Text ← "Task " + index
    END CONSTRUCTOR

    PUBLIC FUNCTION VOID EnableStartingVertex()
        THIS.labelStartingVertex.Enabled ← TRUE
        THIS.labelStartingVertex.Visible ← TRUE
        THIS.comboBoxStartingVertex.Enable ← TRUE
        THIS.comboBoxStartingVertex.Visible ← TRUE
    END FUNCTION

    PUBLIC FUNCTION VOID DisableStartingVertex()
        THIS.labelStartingVertex.Enabled ← FALSE
        THIS.labelStartingVertex.Visible ← FALSE
        THIS.comboBoxStartingVertex.Enable ← FALSE
        THIS.comboBoxStartingVertex.Visible ← FALSE
    END FUNCTION

    PUBLIC FUNCTION VOID EnableFinishingVertex()
        THIS.labelFinishingVertex.Enabled ← TRUE
        THIS.labelFinishingVertex.Visible ← TRUE
        THIS.comboBoxFinishingVertex.Enable ← TRUE
        THIS.comboBoxFinishingVertex.Visible ← TRUE
    END FUNCTION

    PUBLIC FUNCTION VOID DisableFinishingVertex()
        THIS.labelFinishingVertex.Enabled ← FALSE
        THIS.labelFinishingVertex.Visible ← FALSE
        THIS.comboBoxFinishingVertex.Enable ← FALSE
        THIS.comboBoxFinishingVertex.Visible ← FALSE
    END FUNCTION

    EVENT ComboBoxTask_TextChanged
        DEFINE newTaskText: STRING
        newTaskText ← comboBoxTask.Text
        IF newTaskText = tasks["Prim"]
            EnableStartingVertex()
            DisableFinishingVertex()
        ELSE IF newTaskText = tasks["Dijkstra"]
            EnableStartingVertex()
            EnableFinishingVertex()
        ELSE
            DisableStartingVertex()
            DisableFinishigVertex()
        END IF
    END EVENT

    EVENT buttonRemoveTask_Click
        THIS.Dispose()
    END EVENT
END CLASS
```

`DoTaskControls` class

This class is a collection of user controls, to be used in editing a subtask in the Do Task Section. (*Reference: Module 2.3.4 – Do Task*) It contains:

User controls:

- A label `labelTaskIndex` showing the task number;
- A label `labelTask` showing the task content
- A textbox `textBoxInputAnswer` for users to enter the answer;
- A button `buttonInputGraph` for users to open a graph editing window and design a graph, if required by the content of the task;
- A label `labelCorrectWrong` indicating if the user entered answer is correct or wrong, once the task is marked;
- A label `labelAnswer` (hidden by default) showing the answer of the task;
- A button `buttonShowAnswer` for users to show the answer;
- A button `buttonExplain` for users to open step-by-step demonstration on the task.

Variables:

- A string `answerValue` keeping the numeric answer value;
- An adjacency matrix `answerMatrix` showing graphical answer;
- An adjacency matrix `inputMatrix` showing the user-designed graph.

Sample design:

Please refer to Module 2.3.4 – Do Questions in Appendix 1 – GUI implementation.

Events:

Relevant events will be constructed in the Task Setting Section. (*Reference: Module 2.3.4 – Do Task*)

Pseudo-code

```
PUBLIC CLASS DoTaskControls

    DEFINE PUBLIC labelTaskIndex: LABEL
    DEFINE PUBLIC labelTask: LABEL
    DEFINE PUBLIC textBoxInputAnswer: TEXTBOX
    DEFINE PUBLIC buttonInputGraph: BUTTON
    DEFINE PUBLIC labelCorrectWrong: LABEL
    DEFINE PUBLIC labelAnswer: LABEL
    DEFINE PUBLIC buttonShowAnswer: BUTTON
    DEFINE PUBLIC buttonExplain: BUTTON
    DEFINE PRIVATE answerValue: STRING
    DEFINE PRIVATE answerMatrix: AdjacencyMatrix
    DEFINE PRIVATE inputMatrix: AdjacencyMatrix

    PUBLIC CONSTRUCTOR DoTaskControls(INTEGER index)
        THIS.labelTaskIndex.Text ← "Task " + index
    END CONSTRUCTOR

    PUBLIC FUNCTION STRING GetAnswerValue()
        RETURN THIS.answerValue
    END FUNCTION
```

```
    PUBLIC FUNCTION AdjacencyMatrix GetAnswerMatrix()
        RETURN THIS.answerMatrix
    END FUNCTION

    PUBLIC FUNCTION AdjacencyMatrix GetInputMatrix()
        RETURN THIS.inputMatrix
    END FUNCTION

    PUBLIC FUNCTION VOID SetAnswerValue(STRING newAnswerValue)
        THIS.answerValue ← newAnswerValue
    END FUNCTION

    PUBLIC FUNCTION VOID SetAnswerMatrix(AdjacencyMatrix newAnswerMatrix)
        THIS.answerMatrix ← newAnswerMatrix
    END FUNCTION

    PUBLIC FUNCTION VOID SetInputMatrix(AdjacencyMatrix newInputMatrix)
        THIS.inputMatirx ← newInputMatrix
    END FUNCTION
END CLASS
```

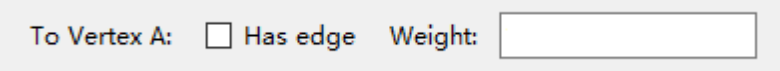
`VertexTagControls` class

This class is a collection of user controls, to be used in editing the properties of an adjacent edge of a vertex in the Sketch Board. (*Reference: Module 2.2.3 – Sketch Board*) It contains:

User controls:

- A label `labelFinishingVertex` showing the destination vertex of the edge;
- A checkbox `checkBoxContainsEdge` for users to set if there is an edge between the current vertex and the destination vertex;
- A label `labelWeight` indicating users to enter the weight, if the edge exists;
- A textbox `textBoxWeight` for users to enter the weight.

Sample design:



Events:

`checkBoxContainsEdge` : CheckChanged

Pseudo-code

```
PUBLIC CLASS VertexTagControls
    DEFINE PUBLIC labelFinishingVertex: LABEL
    DEFINE PUBLIC checkBoxContainsEdge: CHECKBOX
    DEFINE PUBLIC labelWeight: LABEL
    DEFINE PUBLIC textBoxWeight: TEXTBOX

    PUBLIC CONSTRUCTOR VertexTagControls(INTEGER vertex)
        THIS.labelFinishingVertex.Text ← "To Vertex " + (vertex + 'A')
    END CONSTRUCTOR

    EVENT CheckBoxContainsEdge_CheckChanged
        IF checkBoxContainsEdge.Checked = TRUE
            THIS.labelWeight.Enabled ← TRUE
            THIS.textBoxWeight.Enabled ← TRUE
        ELSE
            THIS.labelWeight.Enabled ← FALSE
            THIS.textBoxWeight.Enabled ← FALSE
            THIS.textBoxWeight.Text ← ""
        END IF
    END EVENT
END CLASS
```

`Graph` class

This abstract class represents a graph in general. It contains:

Variables:

- Integer constant `SIZE` representing the maximum limit number of vertices. In this system, the maximum limit is 26;
- Boolean array `vertexExisting` indicating whether each of the 26 vertices is contained in the graph.

Graph operation functions:

- Get the name of the graph, either in string format, or its numeric index;
- Get the number of vertices that are contained in the graph;
- Get/set if a vertex is contained in the graph;
- Get/set the weight of an edge between two vertices;
- Check if there is an edge between two vertices;
- Remove an edge between two vertices;
- Remove a vertex along with all of its adjacent edges from the graph;
- Clear the graph;
- Check if the graph is undirected.

Graph algorithms:

- Prim's algorithm, returning either the total weight of the Minimum Spanning Tree of the graph (the graph must be undirected), or the whole Minimum Spanning Tree;
- Kruskal's algorithm, returning either the total weight of the Minimum Spanning Tree of the graph (the graph must be undirected), or the whole Minimum Spanning Tree;
- Dijkstra's algorithm, returning either the shortest distance between two vertices of the graph, or the whole shortest path between the two vertices;
- All the relevant functions or algorithms that contribute to the above three algorithms, such as Quicksort and Union-Find data structure.

Pseudo-code

```
PUBLIC ABSTRACT CLASS Graph
    DEFINE PRIVATE CONSTANT SIZE ← 26
    DEFINE PRIVATE vertexExisting: BOOLEAN[SIZE]

    PUBLIC CONSTRUCTOR Graph()
        FOREACH BOOLEAN status IN vertexExisting
            status ← FALSE
        END FOR
    END CONSTRUCTOR

    PUBLIC FUNCTION INTEGER GetSize()
        RETURN THIS.SIZE
    END FUNCTION

    PUBLIC FUNCTION BOOLEAN IsVertexExisting(INTEGER vertex)
        RETURN THIS.vertexExisting[vertex]
    END FUNCTION
```

```
PUBLIC FUNCTION STRING GetVertexName(INTEGER vertexIndex)
   RETURN "vertex" + (vertexIndex + 'A')
END FUNCTION

PUBLIC FUNCTION INTGER GetVertexIndex(STRING vertexName)
   RETURN vertexName.Trim("vertex") - 'A'
END FUNCTION

PUBLIC FUNCTION INTEGER Count()
   DEFINE count: INTEGER
   count ← 0
   FOREACH BOOLEAN status IN THIS.vertexExisting
      IF status = TRUE
         count ← count + 1
      END IF
   END FOR
   RETURN count
END FUNCTION

PUBLIC ABSTRACT FUNCTION REAL GetEdge(INTEGER vStart, INTEGER vFinish)

PUBLIC FUNCTION BOOLEAN ContainsEdge(INTEGER vStart, INTEGER vFinish)
   RETURN GetEdge(vStart, vFinish) ≠ 0
END FUNCTION

PUBLIC ABSTRACT FUNCTION VOID SetEdge(INTEGER vStart, INTEGER vFinish,
                                      REAL weight, BOOLEAN isDirected)

PUBLIC FUNCTION VOID SetEdge(INTEGER vStart, INTEGER vFinish, BOOLEAN isDirected)
   SetEdge(vStart, vFinish, 1, isDirected)
END FUNCTION

PUBLIC FUNCTION VOID SetDirectedEdge(INTEGER vStart, INTEGER vFinish, REAL weight)
   SetEdge(vStart, vFinish, weight, TRUE)
END FUNCTION

PUBLIC FUNCTION VOID SetDirectedEdge(INTEGER vStart, INTEGER vFinish)
   SetEdge(vStart, vFinish, 1, TRUE)
END FUNCTION

PUBLIC FUNCTION VOID SetUndirectedEdge(INTEGER vStart, INTEGER vFinish, REAL weight)
   SetEdge(vStart, vFinish, weight, FALSE)
END FUNCTION

PUBLIC FUNCTION VOID SetUndirectedEdge(INTEGER vStart, INTEGER vFinish)
   SetEdge(vStart, vFinish, 1, FALSE)
END FUNCTION

PUBLIC ABSTRACT FUNCTION VOID RemoveEdge(INTEGER vStart, INTEGER vFinish,
                                         BOOLEAN isDirected)

PUBLIC FUNCTION VOID RemoveDirectedEdge(INTEGER vStart, INTEGER vFinish)
   RemoveEdge(vStart, vFinish, TRUE)
END FUNCION
```

```
PUBLIC FUNCTION VOID RemoveUndirectedEdge(INTEGER vStart, INTEGER vFinish)
    RemoveEdge(vStart, vFinish, FALSE)
END FUNCION

PUBLIC FUNCTION VOID SetVertexExistance(INTEGER vertex, BOOLEAN status)
    THIS.vertexExisting[vertex] ← status
END FUNCTION

PUBLIC FUNCTION VOID EnableVertex(INTEGER vertex)
    SetVertexExistance(vertex, TRUE)
END FUNCTION

PUBLIC FUNCTION VOID DisableVertex(INTEGER vertex)
    SetVertexExistance(vertex, FALSE)
END FUNCTION

PUBLIC FUNCTION VOID RemoveVertex(INTEGER vertex)
    FOR INTEGER v ← 0 TO GetSize() - 1 DO
        RemoveUndirectedEdge(v, vertex)
    END FOR
    DisableVertex(vertex)
END FUNCTION

PUBLIC FUNCTION VOID Clear()
    FOR INTEGER v ← 0 TO GetSize() - 1 DO
        RemoveVertex(v)
    END FOR
END FUNCTION

PUBLIC FUNCTION BOOLEAN CheckUndirectedGraph()
    FOR INTEGER v1 ← 0 TO GetSize() - 2 DO
        FOR INTEGER v2 ← v1 + 1 TO GetSize() - 1 DO
            IF GetEdge(v1, v2) ≠ GetEdge(v2, v1)
                RETURN FALSE
            END IF
        END FOR
    END FOR
    RETURN TRUE
END FUNCTION

PUBLIC FUNCTION REAL Prim(INTEGER vStart)
    IF CheckUndirectedGraph = FALSE
        <Output error message>
    ELSE
        DEFINE visitedVertices, remainingVertices: LIST<INTEGER>
        DEFINE weightMST: DOUBLE
        weightMST ← 0
        FOR INTEGER i ← 0 TO GetSize() - 1 DO
            IF IsVertexExisting(i) = TRUE
                remainingVertices.Add(i)
            END IF
        END FOR
        WHILE remainingVertices.Count ≠ 0 DO
            DEFINE min: REAL
            DEFINE newVertex: INTEGER
            min ← +∞
```

```
            FOREACH INTEGER i IN visitedVertices
                FOREACH INTEGER j IN remainingVertices
                    IF ContainsEdge(i, j) = TRUE AND GetEdge(i, j) < min
                        min ← GetEdge(i, j)
                        newVertex ← j
                    END IF
                END FOR
            END FOR
            visitedVertices.Add(newVertex)
            remainingVertices.Remove(newVertex)
            weightMST += min
        END WHILE
        RETURN weightMST
    END IF
END FUNCTION

PUBLIC FUNCTION Graph Prim_GetTree(INTEGER vStart)
    IF CheckUndirectedGraph = FALSE
        <Output error message>
    ELSE
        DEFINE visitedVertices, remainingVertices: LIST<INTEGER>
        DEFINE outputMST: Graph
        FOR INTEGER i ← 0 TO GetSize() - 1 DO
            IF IsVertexExisting(i) = TRUE
                remainingVertices.Add(i)
            END IF
        END FOR
        WHILE remainingVertices.Count ≠ 0 DO
            DEFINE min: REAL
            DEFINE newVStart, newVFinish: INTEGER
            min ← +∞
            FOREACH INTEGER i IN visitedVertices
                FOREACH INTEGER j IN remainingVertices
                    IF ContainsEdge(i, j) = TRUE AND GetEdge(i, j) < min
                        min ← GetEdge(i, j)
                        newVStart ← i
                        newVFinish ← j
                    END IF
                END FOR
            END FOR
            visitedVertices.Add(newVertex)
            remainingVertices.Remove(newVertex)
            outputMST.SetUndirectedEdge(newVStart, newVFinish, min)
        END WHILE
        RETURN outputMST
    END IF
END FUNCTION

DEFINE PRIVATE STRUCT Edge
    vStart, vFinish: INTEGER
    weight: REAL
END STRUCT

DEFINE PRIVATE edges: LIST<Edge>
```

```
PRIVATE FUNCTION VOID InitialiseEdges()
    FOR INTEGER i ← 0 TO GetSize() - 2 DO
        FOR INTEGER j ← i + 1 TO GetSize() - 1 DO
            IF ContainsEdge(i, j) = TRUE
                edges.Add(new Edge(vStart ← i, vFinish ← j, weight ← GetEdge(i, j)))
            END IF
        END FOR
    END FOR
END FUNCTION

DEFINE PRIVATE STRUCT UnionFind
    vertex, leader, prev, head, tail, count: INTEGER
END STRUCT
DEFINE PRIVATE unionFindVertices: LIST<UnionFind>

PRIVATE FUNCTION VOID InitialiseUnionFind()
    FOR INTEGER v ← 0 TO GetSize() - 1 DO
        IF IsVertexExisting(v) = TRUE
            unionFindVertices.Add(new UnionFind(vertex ← v, leader ← v,
                                                prev ← -1, head ← v,
                                                tail ← v, count ← 1)
        END IF
    END FOR
END FUNCTION

PRIVATE FUNCTION UnionFind Find(INTEGER vertex)
    FOREACH UnionFind v IN unionFindVertices
        IF v.vertex = vertex
            RETURN v
        END IF
    END FOR
    RETURN <Not found>
END FUNCTION

PRIVATE FUNCTION VOID Update(INTEGER setX, INTEGER setY)
    DEFINE index: INTEGER
    index ← unionFindVertices[setX].tail
    LOOP
        unionFindVertices[index].leader ← setY
        index ← unionFindVertices[index].prev
    UNTIL index = -1
    unionFindVertices[unionFindVertices[setY].head].Prev ←
                                                unionFindVertices[setX].tail
    unionFindVertices[setY].head ← unionFindVertices[setX].head
    unionFindVertices[setY].count ← unionFindVertices[setY].count
                                    + unionFindVertices[setX].count
END FUNCTION

PRIVATE FUNCTION VOID Union(INTEGER setX, INTEGER setY)
    IF unionFindVertices[setX].count < unionFindVertices[setY].count
        Update(setX, setY)
    ELSE
        Update(setY, setX)
    END IF
END FUNCTION
```

```
PUBLIC FUNCTION REAL Kruskal()
    IF CheckUndirectedGraph() = FALSE
        <Output error message>
    ELSE
        DEFINE weightMST: DOUBLE
        DEFINE count: INT
        weightMST ← 0
        count ← 0
        InitialiseEdges()
        InitialiseUnionFind()
        <Sort edges in ascending order>
        WHILE count < Count() - 1 DO
            IF Find(edges[0].vStart).leader ≠ Find(edges[0].vFinish).leader
                count ← count + 1
                weightMST ← weightMST + edges[0].weight
                Union(Find(edges[0].vStart).leader, Find(edges[0].vFinish).leader)
            END IF
            edges.Remove(edges[0])
        END WHILE
        RETURN weightMST
    END IF
END FUNCTION

PUBLIC FUNCTION Graph Kruskal_GetTree()
    IF CheckUndirectedGraph() = FALSE
        <Output error message>
    ELSE
        DEFINE outputMST: Graph
        DEFINE count: INT
    InitialiseEdges()
        InitialiseUnionFind()
        <Sort edges in ascending order>
        WHILE count < Count() - 1 DO
            IF Find(edges[0].vStart).leader ≠ Find(edges[0].vFinish).leader
                count ← count + 1
                outputMST.SetUndirectedEdge(edges[0].vStart, edges[0].vFinish)
                Union(Find(edges[0].vStart).leader, Find(edges[0].vFinish).leader)
            END IF
            edges.Remove(edges[0])
        END WHILE
        RETURN outputMST
    END IF
END FUNCTION

DEFINE PRIVATE STRUCT DijkstraVertex
    distance: REAL
    prev: INTEGER
END STRUCT
DEFINE PRIVATE dijkstraMap: DijkstraVertex[SIZE]

PRIVATE FUNCTION VOID InitialiseSingleSource(INTEGER vStart)
    FOREACH DijkstraVertex v IN dijkstraMap
        v.distance ← +∞, v.prev ← -1
    END FOR
    dijkstraMap[vStart].distance ← 0
END FUNCTION
```

```
PRIVATE FUNCTION VOID RelaxEdge(INTEGER vStart, INTEGER vFinish)
    IF ContainsEdge(vStart, vFinish)
        AND dijkstraMap[vFinish].distance > dijkstraMap[vStart].distance
                                            + GetEdge(vStart, vFinish)
        dijkstraMap[vFinish].distance ← dijkstraMap[vStart].distance
                                        + GetEdge(vStart, vFinish)
        dijkstraMap[vFinish].prev ← vStart
    END IF
END FUNCTION

PUBLIC FUNCTION REAL Dijkstra(INTEGER vStart, INTEGER vFinish)
    InitialiseSingleSource(vStart)
    DEFINE permanentVertices, temporaryVertices: LIST<INTEGER>
    FOR INTEGER i ← 0 TO GetSize() - 1 DO
        temporaryVertices.Add(i)
    END FOR
    WHILE temporaryVertices.Count ≠ 0 DO
        DEFINE minTemporaryVertex: INTEGER
        DEFINE min: REAL
        min ← +∞
        FOREACH INTEGER i IN temporaryVertices
            IF dijkstraMap[i].distance < min
                minTemporaryVertex ← i
                min ← dijkstraMap[i].distance
            END IF
        END FOR
        temporaryVertices.Remove(minTemporaryVertex)
        permanentVertices.Add(minTemporaryVertex)
        FOR INTEGER i ← 0 TO GetSize() - 1 DO
            RelaxEdge(minTemporaryVertex, i)
        END FOR
    END WHILE
    RETURN dijkstraMap[vFinish].distance
END FUNCTION

PUBLIC FUNCTION LIST<INTEGER> Dijkstra_GetShortestPath(INTEGER vStart,
                                                       INTEGER vFinish)
    InitialiseSingleSource(vStart)
    DEFINE permanentVertices: LIST<INTEGER>
    DEFINE temporaryVertices: LIST<INTEGER>
    DEFINE shortestPath: LIST<INTEGER>
    FOR INTEGER i ← 0 TO GetSize() - 1 DO
        temporaryVertices.Add(i)
    END FOR
    WHILE temporaryVertices.Count ≠ 0 DO
        DEFINE minTemporaryVertex: INTEGER
        DEFINE min: REAL
        min ← +∞
        FOREACH INTEGER i IN temporaryVertices
            IF dijkstraMap[i].distance < min
                minTemporaryVertex ← i
                min ← dijkstraMap[i].distance
            END IF
        END FOR
        temporaryVertices.Remove(minTemporaryVertex)
        permanentVertices.Add(minTemporaryVertex)
```

```
            FOR INTEGER i ← 0 TO GetSize() - 1 DO
                RelaxEdge(minTemporaryVertex, i)
            END FOR
        END WHILE
        IF dijkstraMap[vFinish].distance ← +∞
            RETURN <No path between vStart and vFinish>
        ELSE
            DEFINE i: INTEGER
            i ← vFinish
            shortestPath.Add(i)
            WHILE dijkstraMap[i].prev ≠ -1 DO
                shortestPath.Add(dijkstraMap[i].prev)
                i ← dijkstraMap[i].prev
            END WHILE
            shortestPath.ReverseOrder()
            RETURN shortestPath
        END IF
    END FUNCTION
END CLASS
```

`AdjacencyMatrix` class

This class, inheriting `Graph` class, represents an adjacency matrix. It contains:

Variables:

- Real 2-dimensional array `map` representing the adjacency matrix.

Overriden graph operation functions:

- `GetEdge(INTEGER vStart, INTEGER vFinish)`
- `SetEdge(INTEGER vStart, INTEGER vFinish, REAL weight, BOOLEAN isDirected)`
- `RemoveEdge(INTEGER vStart, INTEGER vFinish, BOOLEAN isDirected)`

Self-implemented graph operation functions:

- `CompareTo(AdjacencyMatrix matrix)` : compares with an adjacency matrix, and returns a list of integer containing the vertices that are different. (Returns NULL if there is no difference.)

Pseudo-code

```
PUBLIC CLASS AdjacencyMatrix INHERITS Graph
    DEFINE PRIVATE map: REAL[GetSize(), GetSize()]

    PUBLIC CONSTRUCTOR AdjacencyMatrix() INHERITS BASE()
        FOREACH REAL element IN map
            status ← 0
        END FOR
    END CONSTRUCTOR

    PUBLIC OVERRIDE FUNCTION REAL GetEdge(INTEGER vStart, INTEGER vFinish)
        RETURN THIS.map[vStart, vFinish]
    END FUNCTION

    PUBLIC FUNCTION LIST<INTEGER> CompareTo(AdjacencyMatrix matrix)
        DEFINE differentVertices: LIST<INTEGER>
        IF matrix = NULL
            <Add all existing vertices in THIS object to differentVertices>
            RETURN differentVertices
        ELSE
            FOR INTEGER row ← 0 TO GetSize() - 1 DO
                FOR INTEGER col ← 0 TO GetSize() - 1 DO
                    IF THIS.GetEdge(row, col) ≠ matrix.GetEdge(row, col)
                        differentVertices.Add(row)
                    END IF
                END FOR
            END FOR
            RETURN differentVertices
        END IF
    END FUNCTION

    PUBLIC OVERRIDE FUNCTION VOID SetEdge(INTEGER vStart, INTEGER vFinish,
                                          REAL weight, BOOLEAN isDirected)
        IF weight ≠ 0
            THIS.EnableVertex(vStart)
            THIS.EnableVertex(vFinish)
            THIS.map[vStart, vFinish] ← weight
```

```
            IF isDirected = FALSE
                This.map[vFinish, vStart] ← weight
            END IF
        END IF
    END FUNCTION

    PUBLIC OVERRIDE FUNCTION VOID RemoveEdge(INTEGER vStart, INTEGER vFinish,
                                             BOOLEAN isDirected)
        THIS.map[vStart, vFinish] ← 0
        IF isDirected = FALSE
            THIS.map[vFinish, vStart] ← 0
        END IF
    END FUNCTION
END CLASS
```

`AdjacencyList` class

This class, inheriting `Graph` class, represents an adjacency list. It contains:

Variables:

- Struct `AdjacentEdge` representing a directed edge. It contains the following properties:
  - Integer `vertex` representing the destination vertex;
  - Real `weight` representing the weight of the directed edge.
- Array of list of `AdjacentEdge`, named `list`, representing the adjacency list.

Overriden graph operation functions:

- `GetEdge(INTEGER vStart, INTEGER vFinish)`
- `SetEdge(INTEGER vStart, INTEGER vFinish, REAL weight, BOOLEAN isDirected)`
- `RemoveEdge(INTEGER vStart, INTEGER vFinish, BOOLEAN isDirected)`

Self-implemented graph operation functions:

- `CompareTo(AdjacencyList list)` : compares with an adjacency list, and returns a list of integer containing the vertices that are different. (Returns NULL if there is no difference.)

Pseudo-code

```
PUBLIC CLASS AdjacencyMatrix INHERITS Graph
    DEFINE PRIVATE STRUCT AdjacentEdge
        vertex: INTEGER
        weight: REAL
    END STRUCT
    DEFINE PRIVATE list: LIST<AdjacentEdge>[GetSize()]

    PUBLIC CONSTRUCTOR AdjacencyMatrix() INHERITS BASE() // No other operation needed

    PUBLIC OVERRIDE FUNCTION REAL GetEdge(INTEGER vStart, INTEGER vFinish)
        FOREACH AdjacentEdge edge IN list[vStart]
            IF edge.vertex = vFinish
                RETURN edge.weight
            END IF
        END FOR
    END FUNCTION

    PUBLIC FUNCTION LIST<INTEGER> CompareTo(AdjacencyList list)
        DEFINE differentVertices: LIST<INTEGER>
        IF list = NULL
            <Add all existing vertices in THIS object to differentVertices>
            RETURN differentVertices
        ELSE
            FOR INTEGER v ← 0 TO GetSize() - 1 DO
                FOREACH AdjacentEdge edge IN THIS.list[v]
                    IF NOT list.list[v].Contains(edge)
                        differentVertices.Add(v)
                    END IF
                END FOR
            END FOR
        RETURN differentVertices
    END FUNCTION
```

```
PUBLIC OVERRIDE FUNCTION VOID SetEdge(INTEGER vStart, INTEGER vFinish,
                                      REAL weight, BOOLEAN isDirected)
    IF weight ≠ 0
        DEFINE edgeToRemove: AdjacentEdge
        FOREACH AdjacentEdge edge IN THIS.list[vStart]
            IF edge.vertex = vFinish
                edgeToRemove ← edge
            END IF
        END FOR
        THIS.list[vStart].Remove(edgeToRemove)
        THIS.list[vStart].Add(NEW AdjacentEdge(vertex ← vFinish, weight ← weight)
        IF isDirected = FALSE
            FOREACH AdjacentEdge edge IN THIS.list[vFinish]
                IF edge.vertex = vStart
                    edgeToRemove ← edge
                END IF
            END FOR
            THIS.list[vFinish].Remove(edgeToRemove)
            THIS.list[vFinish].Add(NEW AdjacentEdge(vertex ← vStart,
                                                    weight ← weight)
        END IF
    END IF
END FUNCTION

PUBLIC OVERRIDE FUNCTION VOID RemoveEdge(INTEGER vStart, INTEGER vFinish,
                                         BOOLEAN isDirected)
    DEFINE edgeToRemove: AdjacentEdge
    FOREACH AdjacentEdge edge IN THIS.list[vStart]
        IF edge.vertex = vFinish
            edgeToRemove ← edge
        END IF
    END FOR
    THIS.list[vStart].Remove(edgeToRemove)
    IF isDirected = FALSE
        FOREACH AdjacentEdge edge IN THIS.list[vFinish]
            IF edge.vertex = vStart
                edgeToRemove ← edge
            END IF
        END FOR
        THIS.list[vFinish].Remove(edgeToRemove)
    END IF
END FUNCTION
```

`MinimumSpanningTreeExample` class

This class represents an example graph to be used in the Step-by-Step Demonstration Section. (*Reference: Module 2.1.1/2.2 Step-by-Step Demonstrations*) It contains:

Variables:

- A List of `Vertex` control, named `vertices`;
- An adjacency matrix `mapMatrix` representing the example graph;
- A panel `panel` where the example graph is shown and all the visual operations takes place;
- A 26 × 26 array of labels `labelWeights` in order to show the weights of the edge on the panel.

Functions:

- Create a vertex with specific name and its location on the panel;
- Create an undirected edge between two specific vertices;
- Place the correct weight labels into the correct positions;
- Highlight/unhighlight the edges and the labels when needed.

Pseudo-code

```
PUBLIC CLASS MinimumSpanningTreeExample

    DEFINE PUBLIC vertices: LIST<Vertex>
    DEFINE PUBLIC mapMatrix: AdjacencyMatrix
    DEFINE PUBLIC panel: PANEL
    DEFINE PUBLIC labelWeights: LABEL[26, 26]

    PUBLIC CONSTRUCTOR MinimumSpanningTreeExample(PANEL panel)
        THIS.panel ← panel
        FOREACH LABEL label IN labelWeights
            label.Enabled ← FALSE
            label.Visible ← FALSE
        END FOR
    END CONSTRUCTOR

    PUBLIC FUNCTION VOID CreateVertex(STRING name, INTEGER x, INTEGER y)
        DEFINE v: Vertex
        v ← new Vertex(name, NEW POINT(x, y))
        v.SetSelectable(FALSE)
        v.SetDraggable(FALSE)
        THIS.vertices.Add(v)
        THIS.panel.Add(v)
        THIS.mapMatrix.EnableVertex(v.GetNumberIndex())
    END FUNCTION

    PUBLIC FUNCTION VOID CreateEdge(Vertex v1, Vertex v2, REAL weight)
        v1.SetEdge(v2, weight)
        v2.SetEdge(v1, weight)
        THIS.mapMatrix.SetUndirectedEdge(v1.GetNumberIndex(),
                                         v2.GetNumberIndex(), weight)
        THIS.panel.<Draw edge between v1 and v2>
    END FUNCTION
```

```
    PUBLIC FUNCTION VOID CreateEdge(INTEGER v1, INTEGER v2, REAL weight)
        FOR INTEGER i ← 0 TO THIS.vertices.Count - 1 DO
            FOR INTEGER j ← 0 TO THIS.vertices.Count - 1 DO
                IF vertices[i].GetNumberIndex() = v1
                    AND vertices[j].GetNumberIndex() = v2
                    vertives[i].SetEdge(vertices[j], weight)
                    vertices[j].SetEdge(vertices[i], weight)
                END IF
            END FOR
        END FOR
        THIS.mapMatrix.SetUndirectedEdge(v1, v2, weight)
        THIS.panel.<Draw edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION VOID DrawLabelWeights()
        FOREACH Vertex v1 IN vertices
            FOREACH Vertex v2 IN v1.GetEdges
                DEFINE vStart, vFinish: INTEGER
                vStart ← v1.GetNumberIndex()
                vFinish ← v2.GetNumberIndex()
                IF vFinish > vStart
                    labelWeights[vStart, vFinish].Enabled ← TRUE
                    labelWeights[vStart, vFinish].Visible ← TRUE
                    DEFINE midpoint: POINT
                    midpoint ← NEW POINT(
                        (v1.GetCentreLocation().X + v2.GetCentreLocation().X) / 2,
                        (v1.GetCentreLocation().Y + v2.GetCentreLocation().Y) / 2);
                    labelWeights[vStart, vFinish].Location ← midpoint
                END IF
            END FOR
        END FOR
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOn(Vertex v1, Vertex v2)
        <Highlight labelWeights[v1.GetNumberIndex(), v2.GetNumberIndex()]>
        <Highlight edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOn(INTEGER v1, INTEGER v2)
        <Highlight labelWeights[v1, v2]>
        <Highlight edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOff(Vertex v1, Vertex v2)
        <Do not highlight labelWeights[v1.GetNumberIndex(),v2.GetNumberIndex()]>
        <Do not highlight edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOff(INTEGER v1, INTEGER v2)
        <Do not highlight labelWeights[v1, v2]>
        <Do not highlight edge between v1 and v2>
    END FUNCTION
END CLASS
```

`ShortestPathExample` class

This class represents an example graph to be used in the Step-by-Step Demonstration Section. (*Reference: Module 2.1.3.2 Step-by-Step Demonstrations*) It contains:

Variables:

- A List of `DijkstraVertexLabel` control, named `vertices`;
- An adjacency matrix `mapMatrix` representing the example graph;
- A panel `panel` where the example graph is shown and all the visual operations takes place;
- A 26 × 26 array of labels `labelWeights` in order to show the weights of the edge on the panel.

Functions:

- Create a vertex with specific name and its location on the panel;
- Create an edge between two specific vertices;
- Place the correct weight labels into the correct positions;
- Highlight/unhighlight the edges and the labels when needed.

Pseudo-code

```
PUBLIC CLASS ShortestPathExample
    DEFINE PUBLIC vertices: LIST<DijkstraVertexLabel>
    DEFINE PUBLIC mapMatrix: AdjacencyMatrix
    DEFINE PUBLIC panel: PANEL
    DEFINE PUBLIC labelWeights: LABEL[26, 26]

    PUBLIC CONSTRUCTOR ShortestPathExample(PANEL panel)
        THIS.panel ← panel
        FOREACH LABEL label IN labelWeights
            label.Enabled ← FALSE
            label.Visible ← FALSE
        END FOR
    END CONSTRUCTOR

    PUBLIC FUNCTION VOID CreateVertex(CHAR name, INTEGER x, INTEGER y)
        DEFINE v: DijkstraVertexLabel
        v ← new DijkstraVertexLabel(name, NEW POINT(x, y))
        v.SetReadOnly(TRUE)
        THIS.vertices.Add(v)
        THIS.panel.Add(v)
        THIS.mapMatrix.EnableVertex(v.GetNumberIndex())
    END FUNCTION

    PUBLIC FUNCTION VOID CreateUndirectedEdge(INTEGER v1, INTEGER v2, REAL weight)
        THIS.mapMatrix.SetUndirectedEdge(v1, v2, weight)
        THIS.panel.<Draw edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION VOID CreateDirectedEdge(INTEGER v1, INTEGER v2, REAL weight)
        THIS.mapMatrix.SetDirectedEdge(v1, v2, weight)
        THIS.panel.<Draw edge between v1 and v2>
    END FUNCTION
```

```
    PUBLIC FUNCTION VOID DrawLabelWeights()
        FOREACH DijkstraVertexLabel v1 IN vertices
            FOREACH DijkstraVertexLabel v2 IN v1.GetEdges
                DEFINE vStart, vFinish: INTEGER
                vStart ← v1.GetNumberIndex()
                vFinish ← v2.GetNumberIndex()
                labelWeights[vStart, vFinish].Enabled ← TRUE
                labelWeights[vStart, vFinish].Visible ← TRUE
                DEFINE point: POINT
                point ← <appropriate location for the label>
                labelWeights[vStart, vFinish].Location ← point
            END FOR
        END FOR
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOn(Vertex v1, Vertex v2)
        <Highlight labelWeights[v1.GetNumberIndex(), v2.GetNumberIndex()]>
        <Highlight edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOn(INTEGER v1, INTEGER v2)
        <Highlight labelWeights[v1, v2]>
        <Highlight edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOff(Vertex v1, Vertex v2)
        <Do not highlight labelWeights[v1.GetNumberIndex(),v2.GetNumberIndex()]>
        <Do not highlight edge between v1 and v2>
    END FUNCTION

    PUBLIC FUNCTION EdgeFocusOff(INTEGER v1, INTEGER v2)
        <Do not highlight labelWeights[v1, v2]>
        <Do not highlight edge between v1 and v2>
    END FUNCTION
END CLASS
```

Prim's minimum spanning tree algorithm

Prim's algorithm is used to find the Minimum Spanning Tree for a weighted undirected graph.

Algorithm description:[4]

START with an arbitrary vertex of G;

STEP 1: Add an edge of minimum weight joining a vertex already included to a vertex not already included;

STEP 2: If a spanning tree is obtained STOP; otherwise return to STEP 1;

Algorithmic pseudo-code[5]

For a graph $G \leftarrow (V, E)$ and the root vertex r, during execution of the algorithm, maintain a min-priority queue Q of all vertices that are not in the Minimum Spanning Tree. Fore each vertex v there are two attributes: the attribute $v.key$ is the minimum weight of any edge connecting v to a vertex in the tree ($v.key \leftarrow \infty$ if there is no such edge); the attribute $v.parent$ names the parent of v in the tree. The min-priority queue Q is based on the key attribute. The Minimum Spanning Tree MST for G is thus computed by the following processing:

```
PRIM(G, r)
    FOREACH vertex u ∈ G.V
        u.key ← ∞
        u.parent ← NULL
    r.key ← 0
    Q ← G.V
    MST ← ∅
    WHILE Q ≠ ∅
        u ← EXTRACT-MIN(Q)
        MST ← MST ∪ {(u, u.parent)}
        FOREACH vertex v ∈ G.Adj[u]
            IF v ∈ Q AND weight(u, v) < v.key
                v.parent ← u
                v.key ← weight(u, v)
    OUTPUT MST
```

Programming implementation:

Please refer to the `Graph` class.

Time complexity:

The WHILE $Q \neq \emptyset$ loop needs to be computed $O(V)$ times;

This project implements Prim's algorithm using linear searching of weights on an adjacency matrix or an adjacency list, so EXTRACT-MIN(Q) takes $O(V)$ running time.

Therefore it has an $O(V^2)$ time complexity.

Kruskal's minimum spanning tree algorithm

Kruskal's algorithm is used to find the Minimum Spanning Tree for a weighted undirected graph.

Algorithm description:[4]

START with all the vertices of G, but no edges; list the edges in increasing order of weight.

STEP 1 Add an edge of G of minimum weight in such a way that no cycles are created.

STEP 2 If a spanning tree is obtained STOP; otherwise return to STEP 1.

Algorithmic pseudo-code[5]

For a graph $G = (V, E)$, Kruskal's algorithm uses a Union-Find data structure to maintain several Union-Find sets of elements. Each set contains the vertices in one tree of the current forest. The operation FIND-SET(u) returns a representative element from the set that contains u. Thus, we can determine whether two vertices u and v belong to the same tree by testing whether FIND-SET(u) equals FIND-SET(v). To combine trees, Kruskal's algorithm calls the UNION procedure.

```
KRUSKAL(G)
    MST ← ∅
    FOREACH vertex v ∈ G.V
        MAKE-SET(v)
    sort the edges of G.E into nondecreasing order by weight w
    FOREACH edge (u, v) ∈ G.E, taken in nondecreasing order by weight
        IF FIND-SET(u) ≠ FIND-SET(v)
            MST ← MST ∪ {(u, v)}
            UNION(u, v)
    OUTPUT MST
```

Programming implementation:

Please refer to the `Graph` class.

Time complexity:

The time taken to sort the edges is $O(E \log E)$. Since $E \leq V^2$, we have $O(E \log E) = O(E \log V^2) = O(2E \log V) = O(E \log V)$;

This project implements Kruskal's algorithm using the union-by-rank and path-compression heuristics for the implementation of the Union-Find data structure, therefore the FOREACH loop performs $O(E)$ FIND-SET and UNION operations in the Union-Find forest.

Overall it has an $O(E \log V)$ time complexity.

Dijkstra's shortest path algorithm

Dijkstra's algorithm is used to find the shortest path between to vertices in a graph without negative-weight edge cycles.

Algorithm description:[4]

START with a graph G. At each vertex draw a box, the lower area for temporary labels, the upper left hand area for the order of becoming permanent and the upper right hand area for the permanent label.

STEP 1 Make the given start vertex permanent by giving it permanent label 0 and order label 1.

STEP 2 For each vertex that is not permanent and is connected by an arc to the vertex that has just been made permanent (with permanent label = P), add the arc weight to P. If this is smaller than the best temporary label at the vertex, write this value as the new best temporary label.

STEP 3 Choose the vertex that is not yet permanent which has the smallest best temporary label. If there is more than one such vertex, choose any one of them. Make this vertex permanent and assign it the next order label.

STEP 4 If every vertex is now permanent, or if the target vertex is permanent, use ‘trace back’ to find the routes or route, then STOP; otherwise return to STEP 2

Algorithmic pseudo-code[5]

For each vertex $v \in G.V$, we maintain an attribute v.distance, which is an upper bound on the weight of a shortest path from sourse vertex s to v, and an attribute $v.prev$, which is the predecessor of v that is either another vertex or NULL. We initialise $v.distance$ by the following procedure:

```
INITIALISE-SINGLE-SOURCE(G, s)
    FOREACH vertex v ∈ G.V
        v.distance ← ∞
        v.prev ← NULL
    s.distance ← 0
```

The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to (u, v) found so far by going through u and, if so, updating $v.distance$ and $v.prev$. The following procedure performs a relaxation step on edge (u, v)

```
RELAX(u, v)
    IF v.distance > u.distance + weight(u, v)
        v.distance ← u.distance + weight(u, v)
        v.prev ← u
```

Dijkstra’s algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum $distance$ attribute, adds u to S, and relaxes all edges leaving u. In the following implementation, we use a min-priority queue Q of vertices, keyed by their $distance$ attribute:

```
DIJKSTRA(G, s)
    INITIALISE-SINGLE-SOURCE(G, s)
    S ← ∅
    Q ← G.V
```

WHILE $Q \neq \emptyset$
    $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    FOREACH vertex $v \in G.Adj[u]$
        RELAX(u, v)

Programming implementation:

Please refer to the `Graph` class.

Time complexity:

The processing of INITIALISE-SINGLE-SOURCE(G, s) takes $O(V)$ running time;

The WHILE loop iterates exactly $|V|$ times, with the following processing:

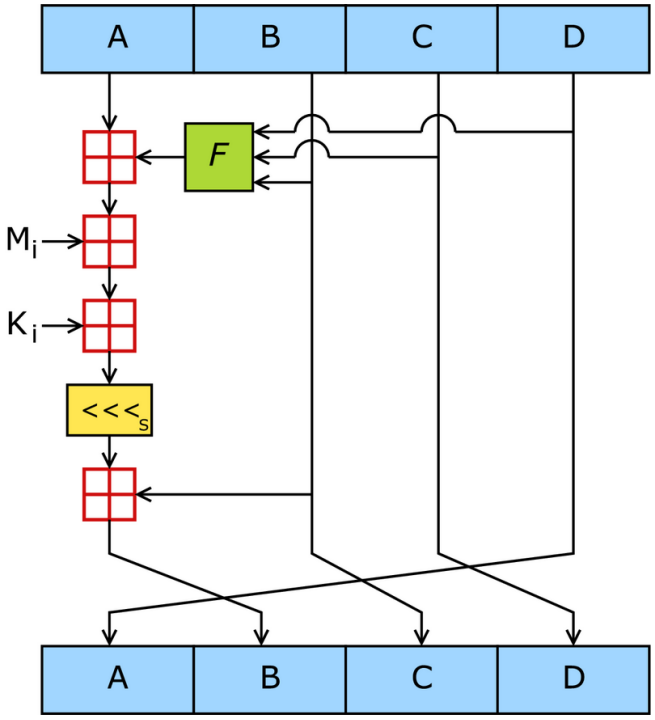
- This project implements Dijkstra's algorithm using linear searching of weights on an adjacency matrix or an adjacency list, so EXTRACT-MIN(Q) takes $O(V)$ running time.
- The processing of RELAX(u, v) takes $O(1)$ running time;

Overall it has an $O(V^2)$ time complexity.

MD5 hashing algorithm

This project use salted MD5 hashing algorithm to store the passwords in the database. The details of this algorithm are discussed as follows: [6][7]

1. Dividing message into blocks:
   MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.
2. The buffer:
   MD5 uses a buffer that is made up of four words that are each 32 bits long. These words are called A, B, C and D. They are initialized as:
   Word A: `01 23 45 67`
   Word B: `89 AB CD EF`
   Word C: `FE DC BA 98`
   Word D: `76 54 32 10`
3. The table:
   MD5 also uses a table K that has 64 elements. Element number i is indicated as K_i. The table is computed beforehand to speed up the computations. The elements are computed using the mathematical sine function:
   $K_i = |\sin(i+1)| \times 2^{32}$
4. Four auxiliary functions:
   In addition, MD5 uses four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word. They apply the logical operators AND, OR, NOT and XOR to the input bits.
   $F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
   $G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
   $H(X,Y,Z) = X \oplus Y \oplus Z$
   $I(X,Y,Z) = Y \oplus (X \vee \neg Z)$
   $\oplus$, $\wedge$, $\vee$, $\neg$ denote the XOR, AND, OR and NOT operations respectively.
5. The contents of the four buffers (A, B, C and D) are now mixed with the words of the input, using the four auxiliary functions (F, G, H and I). The main algorithm then uses each 512-bit message block in turn to modify the state. There are four rounds, each involves 16 basic operations. One operation is illustrated in the figure on the next page:[8]



- *F* is one of the four auxiliary functions; (a different function is used in each round)
- M_i denotes a 32-bit block of the message input;
- K_i denotes a 32-bit constant in the table *K*;
- $<<<_s$ denotes a left-bit rotation by *s* places; (*s* varies for each operation)
- ⊞ denotes addition modulo 2^{32}.

6. MD5 with salt

   This project adds salt to MD5 hashing algorithm in order to defend dictionary attacks.

   The salt is a constant string set in the code.

   The salted MD5 is shown as: `Encrypt(text) = MD5(MD5(text) + salt)`

Pseudo-code solutions to requirements

The pseudo-code solutions to the requirements of this project are shown in the following tables:

Table 1 – Log in, Sign up and Primary Menu:

Module	Inputs	Processing (Pseudo-code)	Outputs

0
Log in
	- 2 textboxes for users to input their usernames and passwords:
	`textBoxUsername`
	`textBoxPassword`
	- A "Log in" button:
	`buttonLogin`
	- A linked table named "New user – sign up" for a new user to sign in for an account:
	`linkLabelSignup`
	- Hidden labels alongside the textboxes for displaying error messages or alerts (visible only when needed)
	`labelErrorMessage`
		Log in operation:
		`EVENT ButtonLogin_Click`
		`  DEFINE sql, sql2: SQL_COMMAND`
		`  DEFINE reader, reader2: SQL_DATA_READER`
		`  IF textBoxUsername.Text = ""`
		`  OR textBoxPassword.Text = ""`
		`  labelErrorMessage.Text ← "Please enter your username/password!"`
		`  ELSE`
		`  DEFINE usernameText, passwordText: STRING`
		`  usernameText ← textBoxUsername.Text`
		`  passwordText ← MD5(textboxPassword.Text)`
		`  sql ← <SQL 0_1 – Query account credential>`
		`  reader ← DATABASE.ExecuteCommand(sql)`
		`  IF reader.ReadNext() ≠ NULL`
		`  sql2 ← <SQL 0_2 – Query account info>`
		`  reader2 ← DATABASE.ExecuteCommand(sql2)`
		`  DEFINE windowPrimaryMenu ← NEW WindowPrimaryMenu(accountID, username, accountName, accountType)`
		`  GOTO windowPrimaryMenu`
		`  ELSE`
		`  labelErrorMessage.Text ← "Invalid login"`
		`  END IF`
		`  END IF`
		`END EVENT`
			Log in operation:
			If accepted, proceed to the Primary Menu window:
			`windowPrimaryMenu`

Module	Inputs	Processing (Pseudo-code)	Outputs

0
Log in
(cont.)
		Sign up operation:
		`EVENT LinkLabelSignUp_Click`
		`  DEFINE windowSignUp ← NEW WindowSignUp()`
		`  GOTO windowSignUp`
		`END EVENT`
			Sign up operation:
			Proceed to the Sign up window:
			`windowSignUp`

1
Sign up
	The Sign up window contains:
	- 2 radio buttons for a new user to choose an account type:
	  - `radioButtonTeacher`
	  - `radioButtonStudent`
	- Textboxes for a new user to enter their personal information:
	  - `textBoxUsername`
	  - `textBoxPassword`
	  - `textBoxRepeatPassword`
	  - `textBoxForename`
	  - `textBoxSurname`
	  - `textBoxDateOfBirth`
	  - `monthCalendar`
	  - `textBoxEmail`
	  - `textBoxSchool`
	- A "Sign up" button:
	  `buttonSignUp`
	Hidden labels that shows error or alert message in need
		Validation:
		`FUNCTION BOOLEAN ValidateSignUp(STRING oldName)`
		`  DEFINE sql: SQL_COMMAND`
		`  DEFINE reader: SQL_DATA_READER`
		`  DEFINE validation ← TRUE: BOOLEAN`
		`  IF NOT 6 ≤ textBoxUsername.Text.Length ≤ 20`
		`  labelErrorMessage.Text="Invalid username!"`
		`  validation ← FALSE`
		`  END IF`
		`  sql ← <SQL 1_1 – Check repetitive username>`
		`  reader ← DATABASE.ExececuteCommand(sql)`
		`  IF reader.ReadNext ≠ NULL`
		`  labelErrorMessage.Text ← "This username has already been taken!"`
		`  validation ← FALSE`
		`  END IF`
		`  IF textBoxPassword.Text≠textBoxRepeatPassword.Text`
		`  labelErrorMessage.Text ← "Repeat password does not match the password!"`
		`  validation ← FALSE`
		`  END IF`
		`  IF <textBoxCompulsoryField>.Text = ""`
		`  labelErrorMessage.Text ← "Empty field!"`
		`  validation ← FALSE`
		`  END IF`
		`  RETURN validation`
		`END FUNCTION`
			Validation:
			`<textBoxCompulsoryField>` includes:
			- `textBoxUsername`
			- `textBoxPassword`
			- `textBoxRepeatPassword`
			- `textBoxForename`
			- `textBoxSurname`
			- `textBoxSchool`

Module	Inputs	Processing (Pseudo-code)	Outputs

1
Sign up
(cont.)
		Accept the sign up request:
		`EVENT WindowSignUp_buttonSignUp_Click`
		`  DEFINE sql: SQL_COMMAND`
		`  DEFINE reader: SQL_DATA_READER`
		`  IF windowSignUp.ValidateSignUp("")`
		`  DEFINE accountID: INTEGER`
		`  DEFINE accountType: STRING`
		`  IF radioButtonTeacher.Checked = TRUE`
		`  accountType ← "TEACHER"`
		`  ELSE accountType ← "STUDENT"`
		`  END IF`
		`  sql ← <SQL 1_2 - Insert new account credential>`
		`  DATABASE.ExececuteCommand(sql)`
		`  sql ← <SQL 1_3 - Query account ID>`
		`  reader ← DATABASE.ExececuteCommand(sql)`
		`  reader.ReadNext()`
		`  accountID ← reader["AccountID"]`
		`  sql ← <SQL 1_4 - Insert new account information>`
		`  DATABASE.ExececyteCommand(sql)`
		`  GOTO windowLogin`
		`  END IF`
		`END EVENT`
			Accept the sign up request:
			Go to the Log in window:
			`windowLogin`

2
Primary
Menu
	3 buttons, each represents a part of the main section:
	- `buttonTeaching`
	- `buttonTaskSetting`
	- `buttonQuestionBank`
		Go to the selected part of the main section:
		`EVENT buttonTeaching_Click`
		`  GOTO windowSelectTopics`
		`END EVENT`
		`EVENT buttonTaskSetting_Click`
		`  GOTO windowTaskSetting`
		`END EVENT`
		`EVENT buttonQuestionBank_Click`
		`  GOTO windowQuestionBank`
		`END EVENT`
			The selected part of the main section:
			- `windowTeaching`
			- `windowTaskSetting`
			- `windowQuestionBank`

Table 2 – Teaching Section:

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1 Teaching Section Menu: Select Topics	Buttons for different topics: - buttonPrim - buttonKruskal - buttonDijkstra	**Go to the topic overview window for the selected topic:** EVENT button<Algo>_Click GOTO windowTopicOverview(<Algo>) END EVENT ***NB*** <Algo> represents: - Prim - Kruskal - Dijkstra	The topic overview window for the selected topic: windowTopicOverview - Parameter: Prim - Parameter: Kruskal - Parameter: Dijkstra
2.1.*.1 Topic Overview	Label for showing the objectivs/ prerequisites for learning the selected algorithm: labelTopicOverview Buttons of the example graphs to carry out the step-by-step demonstrations: buttonExample	**Show the objectives/prerequisites for learning the selected algorithm:** PUBLIC CONSTRUCTOR windowTopicOverview(STRING algo) DEFINE objectives: DICTIONARY<STRING, LIST<STRING>> DEFINE prerequisites: LIST<STRING> *<Set values to objectives[algo, objectives]>* *<Set values to prerequisites>* THIS.labelTopicOverview.Text ← "Objectives: " + objectives[algo] + "Prerequisites: " + prerequisites END CONSTRUCTOR **Go to the step-by-step demonstration window for the example graph:** EVENT buttonExample_Click GOTO window<Algo> END EVENT	the step-by-step demonstration windows: - windowPrimOnGraph - windowPrimOnMatrix - windowKruskal - windowDijkstra

<table>
<tr><th>Module</th><th>Inputs</th><th>Processing (Pseudo-code)</th><th>Outputs</th></tr>
<tr><td>2.1.*.2
Step-by-Step
Demonstrations</td><td>- An array of labels for the steps of the selected algorithm:
  - labelStep
- Variables representing the example graph:
  - exampleGraph
  - vertices
  - mapMatrix
- A panel for the step-by-step demonstration on graph:
  - panelGraph</td><td>Provide a full algorithm description, separated by steps;
This has been predetermined by the design of the window.
Please refer to the GUI design.

Show the selected example graph on the window;
PUBLIC CONSTRUCTOR window<Algo>(INTEGER example)
  exampleGraph=NEW <Example>[example](panelGraph)
  vertices ← exampleGraph.vertices
  mapMatrix ← exampleGraph.mapMatrix
  <Paint exampleGraph on panelGraph>
END CONSTRUCTOR</td><td>NB
<Algo> represents:
  - PrimOnGraph
  - PrimOnMatrix
  - Kruskal
  - Dijkstra
<Example> represents:
  - MinimumSpanningTree Example
  - ShortestPathExample</td></tr>
<tr><td>2.1.1.2(1)
Step-by-Step
Demonstrations
(Prim on graph)</td><td>- A label for relevant explanation:
  - labelInformation
- A label for final answer:
  - labelTotalWeight
- An integer recording the current step that the step-by-step demonstration is at:
  - currentStep
- A real value recording the weight of the minimum spanning tree
  - weightMST
- Two lists of integer:
  - visitedVertices
  - remainingVertices</td><td colspan="2">Step forward & Illustrations on graph & User options on graph & Finishing up:
EVENT ButtonNext_Click
  <Highlight labelStep[currentStep]>
  <Do not highlight other labels in labelStep>
  IF currentStep = 1
  FOR INTEGER i ← 0 TO mapMatrix.GetSize() - 1 DO
  IF mapMatrix.IsVertexExisting(i)
  remainingVertices.Add(i)
  END IF
  END FOR
  labelInformation.Text ← "Choose a vertex:"
  <Wait for a vertex to be clicked>
  visitedVertices.Add(<clicked vertex>)
  remainingVertices.Remove(<clicked vertex>)
  <Highlight clicked vertex>
  <Do not highlight other vertices>
  currentStep ← 2</td></tr>
</table>

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1.1.2(1) Step-by-Step Demonstrations (Prim on graph) *(cont.)*		ELSE IF currentStep = 2 DEFINE candidateEdges: LIST<INTEGER> FOREACH INTEGER i IN visitedVertices FOREACH INTEGER j IN remainingVertices <*Find edges with least weight*> <*Add them to candidateEdges*> END FOR END FOR IF candidateEdges.Count ≥ 2 <*Highliht candidate edges*> labelInformation.Text="Choose an edge:" <*Wait for an edge to be clicked*> weightMST ← weightMST + <*weight of clicked edge*> labelTotalWeight.Text += "+" + <*weight of clicked edge*> examplgeGraph.EdgeFocusOn(<*clicked edge*>) visitedVertices.Add(<*starting vertex*>) remainingVertices.Remove(<*finishing vertex*>) ELSE weightMST ← weightMST + <*minimum weight*> labelTotalWeight.Text += "+" + <*minimum weight*> examplgeGraph.EdgeFocusOn(<*edge of minimum weight*>) visitedVertices.Add(<*starting vertex*>) remainingVertices.Remove(<*finishing vertex*>) END IF ELSE IF currentStep = 3 IF remainingVertices.Count > 0 currentStep ← 2 ELSE labelTotalWeight.Text ← labelTotalWeight.Text + "=" + weightMST buttonNext.Enabled ← FALSE END IF END IF END EVENT	

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1.1.2(2) Step-by-Step Demonstrations (Prim on matrix)	- A label for relevant explanation: - labelInformation - A label for final answer: - labelTotalWeight - A table for the graph matrix: - tableGraph - An integer recording the current step that the step-by-step demonstration is at: - currentStep - A real value recording the weight of the minimum spanning tree - weightMST - Two lists of integer: - visitedVertices - remainingVertices - An struct consisting of the starting vertex, finishing vertex, and the weight of a new edge: - newEdge	**Step forward & Illustrations on graph & User options on graph & Finishing up:** EVENT ButtonNext_Click <*Highlight labelStep[currentStep]*> <*Do not highlight other labels in labelStep*> IF currentStep = 1 FOR INTEGER i ← 0 TO mapMatrix.GetSize() - 1 DO IF mapMatrix.IsVertexExisting(i) remainingVertices.Add(i) END IF END FOR labelInformation.Text ← "Choose a vertex:" <*Wait for a vertex to be clicked*> visitedVertices.Add(<*clicked vertex*>) remainingVertices.Remove(<*clicked vertex*>) <*Cross through the entries of the row of the clicked vertex*> currentStep ← 2 ELSE IF currentStep = 2 DEFINE candidateEdges: LIST<INTEGER> FOREACH INTEGER i IN visitedVertices FOREACH INTEGER j IN remainingVertices <*Find entries with least weight*> <*Add them to candidateEdges*> END FOR END FOR IF candidateEdges.Count ≥ 2 <*Highliht candidate entries*> labelInformation.Text="Choose an entry:" <*Wait for an entry to be clicked*> newEdge.vStart ← tableGraph.<*clicked column*>.HeaderText - 'A' newEdge.vFinish ← tableGraph.<*clicked row*>.HeaderText - 'A' newEdge.weight ← <*weight of the entry*> <*Stop highlighting candidate entries*>	

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1.1.2(2) Step-by-Step Demonstrations (Prim on matrix) *(cont.)*		ELSE IF candidateEdges.Count = 1 newEdge.vStart ← <*minimum entry*>.ColumnIndex.HeaderText - 'A' newEdge.vFinish ← <*minimum entry*>.RowIndex.HeaderText - 'A' newEdge.weight ← <*minimum entry*> END IF currentStep ← 3 ELSE IF currentStep = 3 IF candidateEdges.Count = 0 labelTotalWeight.Text ← labelTotalWeight.Text + "=" + weightMST buttonNext.Enabled ← FALSE ELSE currentStep ← 4 END IF ELSE IF currentStep = 4 weightMST ← weightMST + newEdge.weight labelTotalWeight.Text += "+" + newEdge.weight examplgeGraph.EdgeFocusOn(newEdge.vStart, newEdge.vFinish) visitedVertices.Add(newEdge.vFinish) remainingVertices.Remove(newEdge.vFinish) DEFINE v1, v2: CHAR v1 ← newEdge.vStart + 'A' v2 ← newEdge.vFinish + 'A' FOR INTEGER col ← 0 TO tableGraph.ColumnCount - 1 DO FOR INTEGER row ← 0 TO tableGraph.RowCount - 1 DO IF tableGraph.Columns[col].HeaderText = v1 AND tableGraph.Rows[row].HeaderText = v2 <*Circle tableGraph[col, row]*> <*Mark tableGraph.Columns[col].Header*> <*Cross through the entries of tableGraph.Rows[row]*> END IF END FOR END FOR currentStep ← 5	

Module	Inputs	Processing (Pseudo-code)	Outputs

2.1.1.2(2) Step-by-Step Demonstrations (Prim on matrix) *(cont.)*		`    ELSE IF currentStep = 5`
		`        currentStep ← 2`
		`    END IF`
		`END EVENT`	

2.1.2.2 Step-by-Step Demonstrations (Kruskal)	- A label for relevant explanation:
	  - `labelInformation`
	- A label for final answer:
	  - `labelTotalWeight`
	- A list of labels of the names of the edges:
	  - `labelEdges`
	- A list of labels of the weights of the edges:
	  - `labelWeights`
	- A list of labels that marks if each edge has been used:
	  - `labelEdgeUsed`
	- A struct `Edge` consisting of the starting vertex, finishing vertex, and the weight of an edge;
	- A list of struct `Edge` containing all the edges in the graph:
	  - `edgeList`
		Step forward & Illustrations on graph & User options on graph & Finishing up:
		`EVENT ButtonNext_Click`
		`    <Highlight labelStep[currentStep]>`
		`    <Do not highlight other labels in labelStep>`
		`    IF currentStep = 1`
		`        edgeCount ← 0`
		`        FOR INEGER v1 ← 0 TO mapMatrix.GetSize() - 1 DO`
		`            FOR INTEGER v2 ← v1 + 1 TO mapMatrix.GetSize() DO`
		`                IF mapMatrix.ContainsEdge(v1, v2) = TRUE`
		`                    edgeCount ← edgeCount + 1`
		`                    edgeList.Add(new Edge(vStart ← v1, vFinish ← v2,`
		`                                          weight ← mapMatrix.GetEdge(v1, v2))`
		`                END IF`
		`            END FOR`
		`        END FOR`
		`        <Sort edgeList in non-decreasing order of weight>`
		`        <Show edge names and weights on labelEdges and labelWeights>`
		`        currentEdgeIndex ← 0`
		`        currentWeight ← edgeList[0].weight`
		`        currentStep ← 2`
		`    ELSE IF currentStep = 2`
		`        IF <Cycle check with edgeList[currentEdgeIndex]> = TRUE`
		`            labelEdgeUsed[currentEdgeIndex].Text ← "×"`
		`            LOOP currentEdgeIndex+=1 UNTIL labelEdgeUsed[currentEdgeIndex].Text=""`
		`            currentWeight ← edgeList[currentEdgeIndex].weight`
		`            currentStep ← 2`	

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1.2.2 Step-by-Step Demonstrations (Kruskal) *(cont.)*	- An integer recording the current step that the demonstration is at: - currentStep - An integer recording the weight of the current edge: - currentWeight - An integer recording the index of the current edge: - currentEdgeIndex - An integer recording the number of edges in the minimum spanning tree: - treeEdgeCount - A real value recording the weight of the minimum spanning tree - weightMST	ELSE DEFINE i ← currentEdgeIndex + 1: INTEGER DEFINE tempEdgeList: LIST<Edge> tempEdgeList.Add(edgeList[currentEdgeIndex]) WHILE edgeList[i].weight = currentWeight AND labelEdgeUsed[i].Text = "" DO IF <*Cycle check with edgeList[currentEdgeIndex]*> = FALSE <*Highlight edgeList[i]*> tempEdgeList.Add(edgeList[i]) ELSE labelEdgeUsed[currentEdgeIndex].Text ← "×" END IF i ← i + 1 END WHILE IF tempEdgeList.Count >= 2 labelInformation.Text ← "Choose an edge:" <*Wait for an edge to be clicked*> treeEdgeCount ← treeEdgeCount + 1 weightMST ← weightMST + <*clicked edge*>.weight examplgeGraph.EdgeFocusOn(<*clicked edge*>) labelEdgeUsed[<*index of clicked edge*>].Text ← "√" DEFINE i ← 0: INTEGER LOOP i ← i + 1 UNTIL labelEdgeUsed[i].Text = "" currentEdgeIndex ← i currentWeight ← edgeList[currentEdgeIndex].weight currentStep ← 3 ELSE treeEdgeCount ← treeEdgeCount + 1 weightMST ← weightMST + edgeList[currentEdgeIndex].weight examplgeGraph.EdgeFocusOn(edgeList[currentEdgeIndex]) labelEdgeUsed[currentEdgeIndex].Text ← "√"	

<table>
<tr><th>Module</th><th>Inputs</th><th>Processing (Pseudo-code)</th><th>Outputs</th></tr>
<tr><td>2.1.2.2
Step-by-Step Demonstrations (Kruskal)
(cont.)</td><td></td><td colspan="2">

```
                DEFINE i ← 0: INTEGER
                LOOP
                    i ← i + 1
                UNTIL labelEdgeUsed[i].Text = ""
                currentEdgeIndex ← i
                currentWeight ← edgeList[currentEdgeIndex].weight
                currentStep ← 3
            END IF
        END IF
    ELSE IF currentStep = 3
        IF treeEdgeCount < mapMatrix.Count() - 1
            currentStep ← 2
        ELSE
            FOR INT i ← 0 TO edgeList.Count DO
                IF labelEdgeUsed[i].Text = "√"
                    labelTotalWeight.Text += "+" + labelWeights[i].Text
                END IF
            END FOR
            labelTotalWeight.Text += "=" + weightMST
            buttonNext.Enabled ← FALSE
        END IF
    END IF
END EVENT
```

</td></tr>
<tr><td>2.1.3.2
Step-by-Step Demonstrations (Dijkstra)</td><td>- A label for relevant explanation:
  - <code>labelInformation</code>
- A label for final answer:
  - <code>labelFinalResult</code>
- An integer recording the current step that the demonstration is at:
  - <code>currentStep</code></td><td colspan="2">Step forward & Illustrations on graph & User options on graph & Finishing up:

```
EVENT ButtonNext_Click
    <Highlight labelStep[currentStep]>
    <Do not highlight other labels in labelStep>
    IF currentStep = 1
        labelInformation.Text ← "Choose a vertex:"
        <Wait for a vertex to be clicked>
        DEFINE vStart ← <clicked vertex>.GetNumberIndex(): INTEGER
        InitialiseSingleSource(vStart)
```

</td></tr>
</table>

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1.3.2 Step-by-Step Demonstrations (Dijkstra) *(cont.)*	- 3 lists of integer: - permanentVertices - temporaryVertices - candidateVertices	permanentVertices.Add(vStart) FOR INTEGER v ← 0 TO mapMatrix.GetSize() − 1 DO IF mapMatrix.IsVertexExisting(v) AND v ≠ vStart temporaryVertices.Add(v) END IF END FOR <*clicked vertex*>.Finalise(0, 1) currentStep ← 2 ELSE IF currentStep = 2 DEFINE currentVertex=permanentVertex[permanentVertex.Count−1]:INTEGER FOREACH INTEGER vertex IN temporaryVertices RelaxEdge(currentVertex, vertex) END FOR currentStep ← 3 ELSE IF currentStep = 3 FOREACH INTEGER i IN temporaryVertices <*Find vertices with least distance, and add them to candidateVertices*> END FOR IF candidateVertices.Count ≥ 2 <*Highlight all DijkstraVertexLabel in candidateVertices*> labelInformation.Text ← "Choose a vertex:" <*Wait for a vertex to be clicked*> DEFINE newVertex ← <*clicked vertex*>: DijkstraVertexLabel permanentVertices.Add(newVertex.GetNumberIndex()) temporaryVertices.Remove(newVertex.GetNumberIndex()) newVertex.Finalise(newVertex.distance, permanentVertices.Count) ELSE DEFINE newVertex ← candidateVertices[0]: DijkstraVertexLabel permanentVertices.Add(newVertex.GetNumberIndex()) temporaryVertices.Remove(newVertex.GetNumberIndex()) newVertex.Finalise(newVertex.distance, permanentVertices.Count) END IF currentStep ← 4	

Module	Inputs	Processing (Pseudo-code)	Outputs
2.1.3.2 Step-by-Step Demonstrations (Dijkstra) *(cont.)*		ELSE IF currentStep = 4 IF temporaryVertices.Count > 0 currentStep ← 2 ELSE labelInformation.Text ← "Choose a vertex:" <*Wait for a vertex to be clicked*> DEFINE vFinish ← <*clicked vertex*>: DijkstraVertexLabel labelFinalResult.Text ← "Shortest Route: " + vFinish.distance <*Use trace back to find the shortest path*> <*Show the path on labelFinalResult*> <*Highlight the path*> buttonNext.Enabled ← FALSE END IF END IF END EVENT	

Table 3 – Task Setting Section (Teacher accounts only):

Module	Inputs	Processing & Outputs (Pseudo-code)

2.2
Task Setting Window
	- A textbox for users to enter the name of the question:
	  - `textBoxQuestionName`
	- A textbox for users to enter a general description for the question;
	  - `textBoxProblemDescription`
	- Buttons for users to create a graph, in either of the following forms:
	  - `buttonMatrix`
	  - `buttonList`
	  - `buttonSketchBoard`
	- An "Add Task" button:
	  - `buttonAddTask`
	- A "Save" button.
	  - `buttonSave`
		Create a graph via adjacency matrix:
		`EVENT ButtonMatrix_Click`
		`  GOTO windowEditAdjacencyMatrix`
		`END EVENT`
		
		Create a graph via adjacency list:
		`EVENT ButtonList_Click`
		`  GOTO windowEditAdjacencyList`
		`END EVENT`
		
		Create a graph via the Sketch Board:
		`EVENT ButtonSketchBoard_Click`
		`  GOTO windowSketchBoard`
		`END EVENT`
		
		Validation:
		`DEFINE FUNCTION BOOLEAN ValidateTasks(AdjacencyMatrix mapMatrix)`
		`  DEFINE validation ← TRUE: BOOLEAN`
		`  IF <No graph is entered>`
		`  OUTPUT ERROR "Please enter a graph!"`
		`  validation ← FALSE`
		`  END IF`
		`  IF <All subtasks are empty>`
		`  OUTPUT ERROR "Please enter at least a task!"`
		`  validation ← FALSE`
		`  END IF`
		`  FOREACH <subtask for this question>`
		`  IF <task is Minimum Spanning Tree> AND <graph is directed>`
		`  OUTPUT ERROR "Cannot find minimum spanning tree for a directed graph!"`
		`  validation ← FALSE`

Module	Inputs	Processing & Outputs (Pseudo-code)
2.2 Task Setting Window *(cont.)*		ELSE IF <*Repeated task*> OUTPUT ERROR "Repeated task!" validation ← FALSE ELSE IF <*empty entry in starting vertex*> IF <*task is Prim*> OR <*task is Dijkstra*> OUTPUT ERROR "Empty starting vertex!" validation ← FALSE END IF ELSE IF <*empty entry in finishing vertex*> AND <*task is Dijkstra*> OUTPUT ERROR "Empty finishing vertex!" validation ← FALSE END IF END FOR RETURN validation END FUNCTION **Save the question:** Store the question in the database: DEFINE FUNCTION VOID SaveQuestionToDatabase() DEFINE sql: SQL_COMMAND DEFINE reader: SQL_DATA_READER IF <*new question*> sql ← <*SQL 2.2_1 - Insert new question*> ELSE sql ← <*SQL 2.2_2 - Update question information*> END IF DATABASE.ExecuteCommand(sql) IF <*new question*> sql ← <*SQL 2.2_3 - Retrieve new QuestionID*> reader ← DATABASE.ExecuteCommand(sql) questionID ← reader["QuestionID"] END IF END FUNCTION

Module	Inputs	Processing & Outputs (Pseudo-code)

2.2 Task Setting Window *(cont.)*		Store the corresponding graph in the database:
		`DEFINE FUNCTION VOID SaveGraphToDatabase(AdjacencyMatrix mapMatrix)`
		`  DEFINE sql: SQL_COMMAND`
		`  DEFINE reader: SQL_DATA_READER`
		`  IF <new question>`
		`  sql ← <SQL 2.2_4 - Insert new graph>`
		`  ELSE`
		`  sql ← <SQL 2.2_5 - Update graph values>`
		`  END IF`
		`  DATABASE.ExecuteCommand(sql)`
		`  IF <new question>`
		`  sql ← <SQL 2.2_6 - Retrieve new GraphID>`
		`  reader ← DATABASE.ExecuteCommand(sql)`
		`  graphID ← reader["GraphID"]`
		`  END IF`
		`  IF <new question>`
		`  sql ← <SQL 2.2_7 - Insert new adjacency matrix>`
		`  ELSE`
		`  sql ← <SQL 2.2_8 - Update adjacency matrix values>`
		`  END IF`
		`  DATABASE.ExecuteCommand(sql)`
		`  IF graphFormat = "SketchBoard"`
		`  DEFINE imageFileName: STRING`
		`  imageFileName ← textBoxQuestionName.TEXT + ".png"`
		`  FILE.SAVE(imageFileName)`
		`  IF NOT <new question>`
		`  sql ← <SQL 2.2_9 - Retrieve previous image>`
		`  reader ← DATABASE.ExecuteCommand(sql)`
		`  FILE.Delete(reader["ImageFileName"])`
		`  sql ← <SQL 2.2_10 - Delete previous image>`
		`  DATABASE.ExecuteCommand(sql)`
		`  END IF`

Module	Inputs	Processing & Outputs (Pseudo-code)
2.2 Task Setting Window *(cont.)*		sql ← *<SQL 2.2_11 – Insert new graph image>* DATABASE.ExecuteCommand(sql) END IF END FUNCTION Solve the user-set subtasks and store the answer in the database: DEFINE FUNCTION VOID SaveTasksToDatabase (AdjacencyMatrix mapMatrix) DEFINE sql: SQL_COMMAND DEFINE reader: SQL_DATA_READER IF ValidateTasks(mapMatrix) IF NOT *<new task>* sql ← *<SQL 2.2_12 – Delete previous tasks>* DATABASE.ExecuteCommand(sql) END IF FOREACH task IN taskControls IF task.GetCurrentTaskText() = *<Dijkstra>* sql ← *<SQL 2.2_13 – Insert Dijkstra task>* ELSE IF task.GetCurrentTaskText() = *<Prim>* sql ← *<SQL 2.2_14 – Insert Prim task>* ELSE IF task.GetCurrentTaskText() = <Kruskal> sql ← *<SQL 2.2_15 – Insert Kruskal task>* ELSE sql ← *<SQL 2.2_16 – Insert graph task>* END IF DATABASE.ExecuteCommand(sql) END FOR END IF END FUNCTION

Module	Inputs	Processing & Outputs (Pseudo-code)

2.2 Task Setting Window *(cont.)*		On clicking the "Save" button:
		`EVENT ButtonSave_Click`
		`  MESSAGEBOX.Show("Save?")`
		`  IF Result = YES`
		`  IF ValidateTasks(taskMatrix)`
		`  SaveQuestionToDatabase()`
		`  SaveGraphToDatabase(taskMatrix)`
		`  SaveTasksToDatabase(taskMatrix)`
		`  GOTO windowPrimaryMenu`
		`  END IF`
		`  END IF`
		`END EVENT`

2.2.1 Edit Adjacency Matrix	- A 26×26 table for users to enter the entries of the adjacency matrix: `tableAdjacencyMatrix`
	- A "Save" button: `buttonSubmit`
		Validation & Save the graph:
		`EVENT ButtonSubmit_Click`
		`  DEFINE flag ← TRUE: BOOLEAN`
		`  DEFINE mapMatrix ← AdjacencyMatrix`
		`  DEFINE mapList ← AdjacencyList`
		`  FOR INTEGER col ← 0 TO 25 DO`
		`  FOR INTEGER row ← 0 TO 25 DO`
		`  DEFINE colName, rowName: STRING`
		`  colName ← col + 'A'`
		`  rowName ← row + 'A'`
		`  IF <Invalid input at tableAdjacencyMatrix[col, row]>`
		`  OUTPUT ERROR "Invalid input at row: " + rowName`
		`  + ", column: "+ colName + "!"`
		`  flag ← FALSE`
		`  ELSE IF tableAdjacencyMatrix[col, row].Value < 0`
		`  OUTPUT ERROR "Negative weight at row: " + rowName`
		`  + ", column: "+ colName + "!"`
		`  flag ← FALSE`

Module	Inputs	Processing & Outputs (Pseudo-code)

2.2.1 Edit Adjacency Matrix *(cont.)*		`ELSE`
		`  mapMatrix.SetDirectedEdge(col,row,tableAdjacencyMatrix[col,row].Value)`
		`  mapList.SetDirectedEdge(col,row,tableAdjacencyMatrix[col,row].Value)`
		`END IF`
		`END FOR`
		`END FOR`
		`END EVENT`

2.2.2 Edit Adjacency List	- A list of 26 vertices for users to enter the adjacent edges of each vertices in the adjacency list: `tableAdjacencyList`
	- A "Save" button: `buttonSubmit`
		Validation & Save the graph:
		`EVENT ButtonSubmit_Click`
		`  DEFINE flag ← TRUE: BOOLEAN`
		`  DEFINE mapMatrix ← AdjacencyMatrix`
		`  DEFINE mapList ← AdjacencyList`
		`  FOR INTEGER vertex ← 0 TO 25 DO`
		`  DEFINE adjacentEgdes, newValue, state, vertexName: STRING`
		`  DEFINE finishingVertex: CHAR`
		`  DEFINE weight: REAL`
		`  adjacentEdges ← tableAdjacencyList[1, vertex].Value`
		`  state ← "vertex"`
		`  WHILE adjacentEdges.ReadNextValue() ≠ NULL`
		`  vertexName ← vertex + 'A'`
		`  IF <any invalid input>`
		`  OUTPUT ERROR "Invalid input at vertex " + vertexName + "!"`
		`  flag ← FALSE`
		`  END IF`
		`  IF state = "vertex"`
		`  finishingVertex ← newValue`
		`  IF finishingVertex = vertex + 'A'`
		`  OUTPUT ERROR "Self loop at vertex " + vertexName + "!"`
		`  flag ← FALSE`
		`  state ← "weight"`
		`  ELSE state ← "weight"`
		`  END IF`

Module	Inputs	Processing & Outputs (Pseudo-code)
2.2.2 Edit Adjacency List *(cont.)*		``` ELSE IF state = "weight" IF newValue.Length = 1 AND 'A' ≤ newValue.ToChar() ≤ 'Z' IF finishingVertex = vertex + 'A' OUTPUT ERROR "Self loop at vertex " + vertexName + "!" flag ← FALSE ELSE mapMatrix.SetDirectedEdge(vertex, finishingVertex, 1) mapList.SetDirectedEdge(vertex, finishingVertex, 1) finishingVertex ← newValue.ToChar() END IF state ← "weight" ELSE weight ← newValue.ToReal() IF weight > 0 mapMatrix.SetDirectedEdge(vertex,finishingVertex,weight) mapList.SetDirectedEdge(vertex,finishingVertex,weight) ELSE IF weight < 0 DEFINE vStart, vFinish, edge: STRING vStart ← vertex + 'A' vFinish ← finishingVertex edge ← vStart + vFinish OUTPUT ERROR "Negative weight at edge " + egde + "!" flag ← FALSE END IF END IF END WHILE IF finishingVertex != NULL mapMatrix.SetDirectedEdge(vertex, finishingVertex, 1) mapList.SetDirectedEdge(vertex, finishingVertex, 1) END IF END FOR END EVENT ```

<table>
<tr><th>Module</th><th>Inputs</th><th>Processing & Outputs (Pseudo-code)</th></tr>
<tr><td>2.2.3
Sketch Board</td><td>- A menu consisting of:
- A "Vertex" button
buttonVertex
- An "Edge" button
buttonEdge
- A "Tag" button
buttonTag
- A plain board for users to design graphs: panelSketchBoard</td><td>Create a vertex:
EVENT PanelSketchBoard_MouseDown
IF selectedTool = "buttonVertex"
IF <vertex count> < 26
CreateVertex(<new Vertex Name>, <Mouse click position>)
<Update vertex counter>
ELSE OUTPUT MESSAGE "Maximum number of vertices has been reached!"
END IF
END IF
END EVENT
Edit a vertex/edge:
For vertex: Please refer to Vertex_MouseMove event in Vertex class.
For edge:
EVENT PanelSketchBoard_MouseMove
IF vStart ≠ NULL AND selectedTool = "buttonEdge"
DrawEdge(vStart.GetCentreLocation(), <Mouse click position>)
END IF
END EVENT
For tag:
EVENT Vertex_MouseDoubleClick
IF selectedTool = "buttonTag"
<Open new windowVertexTag>
FOR INTEGER finishingVertex ← 0 TO mapMatrix.GetSize() DO
IF mapMatrix.IsVertexExisting(finishingVertex)
AND finishingVertex ≠ startingVertex
windowVertexTag.AddVertexControl(finishingVertex)
IF mapMatrix.ContainsEdge(startingVertex, finishingVertex)
<Show edge weight on VertexTagControl>
END IF
END IF
END FOR
END IF
END EVENT</td></tr>
</table>

Module	Inputs	Processing & Outputs (Pseudo-code)
2.2.3 Sketch Board *(cont.)*		EVENT WindowVertexTag_buttonSave_Click_ValidateVertexName DEFINE inputVertexNameString: STRING inputVertexNameString ← windowVertexTag.textBoxVertexName.Text IF inputVertexNameString.Length > 1 OUTPUT ERROR "Invalid name!" ELSE IF NOT 'A' ≤ inputVertexNameString.ToChar() ≤ 'Z' OUTPUT ERROR "Invalid name!" ELSE IF vertexNameUsed[inputVertexNameString.ToChar() - 'A'] = TRUE OUTPUT ERROR "This vertex name has already been taken!" ELSE OUTPUT submitSuccessful ← TRUE END IF END EVENT EVENT WindowVertexTag_buttonSave_Click_UpdateVertex *<Change the name of the vertex>* FOREACH VertexTagControl edgeControl IN windowVertexTag.edgeControls DO DEFINE finishingVertex: INTEGER finishingVertex ← edgeControl.GetLabelFinishingVertex().Name - 'A' IF edgeControl.GetTextBoxWeight().Enabled = TRUE DEFINE weight: REAL weight ← edgeControl.GetTextBoxWeight().Text.ToReal() vTagChanged.SetEdge(finishingVertex, weight) mapMatrix.SetDirectedEdge(vTagChanged, finishingVertex, weight) mapList.SetDirectedEdge(vTagChanged, finishingVertex, weight) ELSE vTagChanged.RemoveEdge(finishingVertex) mapMatrix.RemoveEdge(vTagChanged, finishingVertex) mapList.RemoveEdge(vTagChanged, finishingVertex) END IF END FOR END EVENT

Module	Inputs	Processing & Outputs (Pseudo-code)

2.2.3 Sketch Board *(cont.)*		**Create an edge:**
		Draw edge:
		`EVENT PanelSketchBoard_MouseDown`
		`  IF selectedTool = "buttonEdge"`
		`  IF vStart = NULL`
		`  IF <vertex count> < 26`
		`  CreateVertex(<new Vertex Name>, <Mouse click position>)`
		`  <Update vertex counter>`
		`  <new vertex>.SetSelected(TRUE)`
		`  <new vertex>.SetDraggable(FALSE)`
		`  vStart ← <new vertex>`
		`  ELSE OUTPUT MESSAGE "Maximum number of vertices has been reached!"`
		`  END IF`
		`  ELSE`
		`  IF <vertex count> < 26`
		`  CreateVertex(<new Vertex Name>, <Mouse click position>)`
		`  <Update vertex counter>`
		`  <new vertex>.SetSelected(TRUE)`
		`  <new vertex>.SetDraggable(FALSE)`
		`  vFinish ← <new vertex>`
		`  IF isDirected = TRUE`
		`  vStart.SetEdge(vFinish)`
		`  mapMatrix.SetDirectedEdge(vStart, vFinish)`
		`  mapList.SetDirectedEdge(vStart, vFinish)`
		`  ELSE`
		`  vStart.SetEdge(vFinish)`
		`  vFinish.SetEdge(vStart)`
		`  mapMatrix.SetUndirectedEdge(vStart, vFinish)`
		`  mapList.SetUndirectedEdge(vStart, vFinish)`
		`  END IF`
		`  vStart.SetSelected(FALSE)`
		`  vFinish.SetSelected(FALSE)`
		`  vStart ← NULL`
		`  vFinish ← NULL`

Module	Inputs	Processing & Outputs (Pseudo-code)
2.2.3 Sketch Board *(cont.)*		``` ELSE OUTPUT MESSAGE "Maximum number of vertices has been reached!" vStart.SetSelected(FALSE) vStart ← NULL vFinish ← NULL END IF END IF END IF END EVENT EVENT Vertex_MouseDown_DrawEdge IF selectedTool = "buttonEdge" vertex.SetDraggable(FALSE) IF vStart = NULL vStart ← vertex ELSE vFinish ← vertex IF isDirected = TRUE vStart.SetEdge(vFinish) mapMatrix.SetDirectedEdge(vStart, vFinish) mapList.SetDirectedEdge(vStart, vFinish) ELSE vStart.SetEdge(vFinish) vFinish.SetEdge(vStart) mapMatrix.SetUndirectedEdge(vStart, vFinish) mapList.SetUndirectedEdge(vStart, vFinish) END IF vStart.SetSelected(FALSE) vFinish.SetSelected(FALSE) vStart ← NULL vFinish ← NULL END IF END IF END EVENT ```

Module	Inputs	Processing & Outputs (Pseudo-code)

2.2.3
Sketch Board
(cont.)
		Select directed/undirected edge:
		`EVENT ButtonEdge_MouseDown`
		`  timerShowEdgeProperties.Start()`
		`END EVENT`
		
		`EVENT ButtonEdge_MouseUp`
		`  selectedTool ← "buttonEdge"`
		`  timerShowEdgeProperties.Stop()`
		`END EVENT`
		
		`EVENT TimerShowEdgeProperties_Tick`
		`  buttonDirected.Visible ← TRUE`
		`  buttonUndirected.Visible ← TRUE`
		`END EVENT`
		
		`EVENT ButtonDirected_Click`
		`  isDirected ← TRUE`
		`  buttonDirected.Visible ← FALSE`
		`  buttonUndirected.Visible ← FALSE`
		`END EVENT`
		
		`EVENT ButtonUndirected_Click`
		`  isDirected ← FALSE`
		`  buttonDirected.Visible ← FALSE`
		`  buttonUndirected.Visible ← FALSE`
		`END EVENT`
		
		Save the graph:
		`EVENT ButtonEdge_MouseDown`
		`  THIS.Close()`
		`END EVENT`
		Other modules can call the GetMatrix() and GetList() functions to get the graph.

Table 4 – Question Bank Section:

Module	Inputs	Processing & Outputs (Pseudo-code)

2.3 Question Bank Section: List of Questions	A list of all the questions stored in the database:
	`tableQuestions`
	
	Buttons for operations on questions:
	- `buttonAddQuestion`
	- `buttonEditQuestion`
	- `buttonDeleteQuestion`
	- `buttonDoQuestion`
		Query the questions from the database:
		`FUNCTION VOID ResetTableQuestions()`
		`  tableQuestions.Clear()`
		`  DEFINE sql: SQL_COMMAND`
		`  DEFINE reader: SQL_DATA_READER`
		`  sql ← <SQL 2.3_1 – Query all the questions from the Question Bank>`
		`  reader ← DATABASE.ExecuteCommand(sql)`
		`  DEFINE count ← 0: INTEGER`
		`  WHILE reader.ReadNext()`
		`  tableQuestions.AddRow()`
		`  tableQuestions["QuestionID", count].Value ← reader["QuestionID"]`
		`  tableQuestions["DateModified", count].Value ← reader["DateModified"]`
		`  tableQuestions["QuestionName", count].Value ← reader["QuestionName"]`
		`  count ← count + 1`
		`  END WHILE`
		`END FUNCTION`
		Add questions: (teacher accounts only)
		`EVENT ButtonAddQuestion_Click`
		`  GOTO windowTaskSetting`
		`END EVENT`
		Edit questions: (teacher accounts only)
		Please refer to Module 2.3.2 – Edit Questions.
		Delete questions: (teacher accounts only)
		Please refer to Module 2.3.3 – Delete Questions.
		Do questions:
		Please refer to Module 2.3.4 – Do Questions.

2.3.1 Add Questions	Save the new question: Please refer to Module 2.2 – Task Setting Window.	

Module	Inputs	Processing & Outputs (Pseudo-code)
2.3.2 Edit Questions	Load the content of the question:	

```
FUNCTION VOID EditQuestion(STRING questionID)
    DEFINE sql: SQL_COMMAND
    DEFINE reader: SQL_DATA_READER
    DEFINE graphID: INTEGER
    DEFINE graphFormat: STRING
    OPEN NEW windowTaskSetting
    sql ← <SQL 2.3.2_1 - Query general information and the graph of the question>
    reader ← DATABASE.ExecuteCommand(sql)
    windowTaskSetting.textBoxQuestionName.Text ← reader["QuestionName"]
    windowTaskSetting.textBoxProblemDescription.Text ← reader["ProblemDescription"]
    graphID ← reader["GraphID"]
    sql ← <SQL 2.3.2_2 - Query the correct representation for the graph>
    reader ← DATABASE.ExecuteCommand(sql)
    graphFormat ← reader["GraphFormat"]
    sql ← <SQL 2.3.2_3 - Query the graph in the form of an adjacency matrix>
    reader ← DATABASE.ExecuteCommand(sql)
    FOR INTEGER v1 ← 0 TO windowTaskSetting.taskMatrix.GetSize() - 1 DO
        FOR INTEGER v2 ← 0 TO windowTaskSetting.taskMatrix.GetSize() - 1 DO
            DEFINE fieldName ← "Edge" + GetVertexName(v1) + GetVertexName(v2): STRING
            windowTaskSetting.taskMatrix.SetDirectedEdge(v1, v2, reader[fieldName])
        END FOR
    END FOR
    IF graphFormat = "AdjacencyMatrix"
        <Show adjacency matrix on windowTaskSetting>
    ELSE IF graphFormat = "AdjacencyList"
        <Show adjacency list on windowTaskSetting>
    ELSE IF graphFormat = "SketchBoard"
        sql ← <SQL 2.3.2_4 - Query the graph image>
        reader ← DATABASE.ExecuteCommand(sql)
        <Show graph image ← reader["ImageFileName"] on windowTaskSetting>
    END IF
```

Module	Inputs	Processing & Outputs (Pseudo-code)

```
    sql ← <SQL 2.3.2_5 - Query the subtasks>
    reader ← DATABASE.ExecuteCommand(sql)
    DEFINE count ← 0: INTEGER
    WHILE reader.ReadNext()
        IF count ≠ 0
            windowTaskSetting.AddTaskControls()
        END IF
        windowTaskSetting.taskControls[count].ComboBoxTask.Text ← reader["TaskDescription"]
        windowTaskSetting.taskControls[count].ComboBoxStartingVertex.Text ← reader["StartingVertex"]
        windowTaskSetting.taskControls[count].ComboBoxFinishingVertex.Text ← reader["FinishingVertex"]
        count ← count + 1
    END WHILE
END FUNCTION
```

2.3.2 Edit Questions *(cont.)*

```
EVENT ButtonEditQuestion_Click
    DEFINE questionID: STRING
    FOR INTEGER row ← 0 TO tableQuestions.RowCount DO
        IF tableQuestions["QuestionID", row].Selected = TRUE
            questionID ← tableQuestions["QuestionID", row].Value
        END IF
    END FOR
    IF question ≠ NULL
        EditQuestion(questionID)
    END IF
END EVENT
```

Save the edited question:

Please refer to Module 2.2 – Task Setting Window.

Module	Inputs	Processing & Outputs (Pseudo-code)

2.3.3 Delete Questions

Delete the selected question & Refresh the list of questions:

```
EVENT ButtonDeleteQuestion_Click
    DEFINE questionID, graphID, graphFormat, imageToDelete: STRING
    DEFINE sql ← SQL_COMMAND
    DEFINE reader ← SQL_DATA_READER
    FOR INTEGER row ← 0 TO tableQuestions.RowCount DO
        IF tableQuestions["QuestionID", row].Selected = TRUE
            questionID ← tableQuestions["QuestionID", row].Value
        END IF
    END FOR
    IF question ≠ NULL
        MESSAGEBOX.Show("Delete?")
        IF Result = YES
            sql ← <SQL 2.3.3_1 - Query graph ID for deletion>
            reader ← DATABASE.ExecuteCommand(sql)
            graphID ← reader["GraphID"]
            sql ← <SQL 2.3.3_2 - Query the format of the graph to decide which deletion process to be followed>
            reader ← DATABASE.ExecuteCommand(sql)
            graphFormat ← reader["GraphFormat"]
            IF graphFormat = "SketchBoard")
                sql ← <SQL 2.3.3_3 - Query graph image for deletion>
                reader ← DATABASE.ExecuteCommand(sql)
                imageToDelete ← reader["ImageFileName"]
                FILE.Delete(imageToDelete)
                sql ← <SQL 2.3.3_4 - Delete the record in GRAPHIMAGES table>
                DATABASE.ExecuteCommand(sql)
            END IF
            sql ← <SQL 2.3.3_5 - Delete question information, subtasks and the graph>
            DATABASE.ExecuteCommand(sql)
            ResetTableQuestions()
        END IF
    END IF
END EVENT
```

Module	Inputs	Processing & Outputs (Pseudo-code)
2.3.4 Do Questions	User controls to show the content of the selected question on labels. This includes: - labelQuestionName - labelProblemdescription - taskControls Table to show the graph of the selected question in the form of adjacency matrix/list, if any: - tableGraph Picture box to show the graph of the selected question, if any: - pictureBoxGraph A "Mark it" button: - buttonSubmit	**Query the content of the question:** FUNCTION VOID DoQuestion(STRING questionID) DEFINE sql: SQL_COMMAND DEFINE reader: SQL_DATA_READER DEFINE graphID: INTEGER DEFINE graphFormat: STRING OPEN NEW windowDoQuestion sql←*<SQL 2.3.4_1-Query general information and the graph of the question>* reader ← DATABASE.ExecuteCommand(sql) windowDoQuestion.labelQuestionName.Text ← reader["QuestionName"] windowDoQuestion.labelProblemDescription.Text←reader["ProblemDescription"] graphID ← reader["GraphID"] sql ← *<SQL 2.3.4_2 - Query the correct representation for the graph>* reader ← DATABASE.ExecuteCommand(sql) graphFormat ← reader["GraphFormat"] sql ← *<SQL 2.3.4_3 - Query the graph in the form of an adjacency matrix>* reader ← DATABASE.ExecuteCommand(sql) FOR INTEGER v1 ← 0 TO windowDoQuestion.taskMatrix.GetSize() - 1 DO FOR INTEGER v2 ← 0 TO windowDoQuestion.taskMatrix.GetSize() - 1 DO DEFINE fieldName: STRING fieldName ← "Edge" + GetVertexName(v1) + GetVertexName(v2) windowDoQuestion.taskMatrix.SetDirectedEdge(v1,v2,reader[fieldName]) END FOR END FOR IF graphFormat = "AdjacencyMatrix" *<Show adjacency matrix on windowDoQuestion.tableGraph>* ELSE IF graphFormat = "AdjacencyList" *<Show adjacency list on windowDoQuestion.tableGraph>* ELSE IF graphFormat = "SketchBoard" sql ← *<SQL 2.3.4_4 - Query the graph image>* *<Show graph image on windowDoQuestion.pictureBoxGraph>* END IF

Module	Inputs	Processing & Outputs (Pseudo-code)
2.3.4 Do Questions *(cont.)*		sql ← *<SQL 2.3.4_5 – Query the subtasks and their answers>* reader ← DATABASE.ExecuteCommand(sql) DEFINE count ← 0: INTEGER WHILE reader.ReadNext() DEFINE taskDescription ← reader["TaskDescription"]: STRING windowDoQuestion.taskControls[count].labelTask.Text←taskDescription IF taskDescription = *<Prim>* windowDoQuestion.taskControls[count].labelTask.Text += ", starting from " + reader["StartingVertex"] ELSE IF taskDescription = *<Dijkstra>* windowDoQuestion.taskControls[count].labelTask.Text +="from"+reader["StartingVertex"]+"to"+reader["FinishingVertex"] END IF IF taskDescription = *<write adjacency matrix>* OR *<write adjacency list>* OR *<draw graph>* *<Prepare button to proceed to the graph editing windows>* windowDoQuestion.taskControls[count].SetAnswerMatrix(taskMatrix) ELSE *<Prepare textbox for users to enter the answer>* windowDoQuestion.taskControls[count].SetAnswerValue(reader["Answer"]) END IF count ← count + 1 END WHILE END FUNCTION EVENT ButtonDoQuestion_Click DEFINE questionID: STRING FOR INTEGER row ← 0 TO tableQuestions.RowCount DO IF tableQuestions["QuestionID", row].Selected = TRUE questionID ← tableQuestions["QuestionID", row].Value END IF END FOR IF question ≠ NULL DoQuestion(questionID) END IF END EVENT

Module	Inputs	Processing & Outputs (Pseudo-code)

2.3.4
Do Questions
(cont.)
		Mark the question:
		`EVENT ButtonSubmit_Click`
		`  DEFINE score ← 0: INTEGER`
		`  FOREACH DoTaskControl task IN THIS.taskControls DO`
		`  IF <answer is numeric>`
		`  IF task.textBoxInputAnswer.Text = task.GetAnswerValue()`
		`  score ← score + 1`
		`  task.labelCorrectWrong.Text ← "√"`
		`  ELSE`
		`  task.labelCorrectWrong.Text ← "×"`
		`  END IF`
		`  ELSE IF <answer is graphic>`
		`  IF task.GetAnswerMatrix().CompareTo(task.GetInputMatrix()) = NULL`
		`  score ← score + 1`
		`  task.labelCorrectWrong.Text ← "√"`
		`  ELSE`
		`  task.labelCorrectWrong.Text ← "×"`
		`  END IF`
		`  END IF`
		`  task.labelCorrectWrong.Visible ← TRUE`
		`  task.buttonShowAnswer.Enabled ← TRUE`
		`  END FOR`
		`  labelScore.Text ← "Your score: " + score.ToString()`
		`  labelScore.Visible ← TRUE`
		`END EVENT`

2.3.4.1
Mark
Questions
	For each subtask, provide:
	- Labels showing the marks awarded for the questions `labelAnswer`
	- A “Show Answer” button `buttonShowAnswer`
		Show answer:
		`EVENT ButtonShowAnswer_Click`
		`  IF taskControl.textBoxInputAnswer.Visible = TRUE`
		`  IF taskControl.buttonShowAnswer.Text = "Show Answer"`
		`  taskControl.labelAnswer.Visible ← TRUE`
		`  taskControl.buttonShowAnswer.Text ← "Hide Answer"`
		`  ELSE`
		`  taskControl.labelAbswer.Visible ← FALSE`
		`  taskControl.buttonShowAnswer.Text ← "Show Answer"`
		`  END IF`

Module	Inputs	Processing & Outputs (Pseudo-code)
2.3.4.1 Mark Questions *(cont.)*		``` ELSE IF taskControl.labelTask.Text = <write adjacency matrix> OPEN NEW windowEditAdjacencyMatrix <Show taskControl.GetAnswerMatrix() as an adjacency matrix on windowEditAdjacencyMatrix> windowEditAdjacencyMatrix.tableAdjacencyMatrix.ReadOnly ← TRUE ELSE IF taskControl.labelTask.Text = <write adjacency list> OPEN NEW windowEditAdjacencyList <Show taskControl.GetAnswerMatrix() as an adjacency list on windowEditAdjacencyList> windowEditAdjacencyList.tableAdjacencyList.ReadOnly ← TRUE ELSE IF taskControl.labelTask.Text = <draw graph> <Show answer graph image> END IF END IF END EVENT ```

Table 5 – User Accounts:

Module	Inputs	Processing (Pseudo-code)	Outputs
3.1 Account Setting	The Sign up window: `windowSignUp`	**Query the account information:** Please refer to the `AccountMenu` control. **Update account information:** Please refer to the `AccountMenu` control.	N/A
3.2 Quit	N/A	**Quit the system:** Please refer to the `AccountMenu` control.	N/A

SQL pseudo-commands list

The following SQL pseudo-commands will be used in the above mentioned places:

SQL Reference No.	Module Implemented	Pseudo-Command	Comments

0_1	Log in	`SELECT AccountType FROM ACCOUNTS`
		`WHERE Username = '<usernameText>'`
		`AND Password = '<passwordText>';`
			- Query account credential

0_2	Log in	`SELECT <accountType>S.AccountID,`
		`  <accountType>S.Forename,`
		`  <accountType>S.Surname,`
		`FROM <accountType>S, ACCOUNTS`
		`WHERE ACCOUNTS.Username = '<usernameText>'`
		`AND ACCOUNTS.AccountID = <accountType>S.AccountID;`
			- Query account information
			- `<accountType>` represents the value of `reader["AccountType"]`

1_1	Sign up	`SELECT * FROM ACCOUNTS`
		`WHERE Username = '<textBoxUsername.Text>'`
		`AND Username != '<oldName>'`
			- Check repetitive username

1_2	Sign up	`INSERT INTO ACCOUNTS (Username, Password, AccountType)`
		`VALUES ('<textBoxUsername.Text>',`
		`  '<MD5(<textBoxPassword.Text>)>',`
		`  '<accountType>');`
			- Insert new account credential
			- `<MD5()>` represents: `<MD5Hashing.Encrypt()>`

1_3	Sign up	`SELECT AccountID FROM ACCOUNTS` `WHERE Username = '<textBoxUsername.Text>';`	- Query account ID

1_4	Sign up	`INSERT INTO <accountType>S (Forename, Surename,`
		`  DateOfBirth, Email, School, AccountID)`
		`VALUES ('<textBoxForename.Text>',`
		`  '<textBoxSurname.Text>',`
		`  '<textBoxDateOfBirth.Text>',`
		`  '<textBoxEmail.Text>',`
		`  '<textBoxSchool.Text>'`
		`  <accountID>);`
			- Insert new account information

SQL Reference No.	Module Implemented	Pseudo-Command	Comments
2.2_1	Task Setting Window	`INSERT INTO QUESTIONBANK (QuestionName, DateModified, ProblemDescription, GraphID)` `VALUES ('<textBoxQuestionName.Text>', datetime(), '<textBoxProblemDescription.Text>', <graphID>, <questionID>);`	- Insert new question - `<graphID>` represents the value of `FormTaskSetting.graphID` - `<questionID>` represents the value of `FormTaskSetting.questionID`

2.2_2	Task Setting Window	`UPDATE QUESTIONBANK`
		`SET QuestionName = '<textBoxQuestionName.Text>',`
		`DateModified = datetime(),`
		`ProblemDescription =`
		`<textBoxProblemDescription.Text>`
		`WHERE QuestionID = <questionID>;`
			- Update question information
			- `<questionID>` represents the value of `FormTaskSetting.questionID`

2.2_3	Task Setting Window	`SELECT QuestionID FROM QUESTIONBANK` `WHERE rowid = last_insert_rowid();`	- Retrieve new QuestionID from the Question Bank
2.2_4	Task Setting Window	`INSERT INTO GRAPHS (GraphFormat)` `VALUES <GraphFormName>;`	- Insert new graph

2.2_5	Task Setting Window	`UPDATE GRAPHS`
		`SET GraphFormat = <GraphFormName>`
		`WHERE GraphID = <graphID>;`
			- Update graph values
			- `<graphID>` represents the value of `FormTaskSetting.graphID`

2.2_6	Task Setting Window	`SELECT GraphID FROM GRAPHS` `WHERE rowid = last_insert_rowid();`	- Retrieve new GraphID from the GRAPHS table
2.2_7	Task Setting Window	`INSERT INTO ADJACENCYMATRICES (<All required fields>)` `VALUES (<Values of the fields>);`	- Insert new adjacency matrix - Please refer to the Back-end Design of this document for the required fields of the table ADJACENCYMATRICES.

SQL Reference No.	Module Implemented	Pseudo-Command	Comments

2.2_8	Task Setting Window	`UPDATE ADJACENCYMATRICES`
		`SET <fields> = <values>`
		`WHERE GraphID = <graphID>;`
			- Update adjacency matrix values
			- Please refer to the Back-end Design of this document for the required fields of the table ADJACENCYMATRICES.
			- `<graphID>` represents the value of `FormTaskSetting.graphID`

2.2_9	Task Setting Window	`SELECT ImageFileName FROM GRAPHIMAGES` `WHERE GraphID = <graphID>;`	- Retrieve previous image
2.2_10	Task Setting Window	`DELETE FROM GRAPHIMAGES` `WHERE GraphID = <graphID>;`	- Delete previous image
2.2_11	Task Setting Window	`INSERT INTO IMAGES (ImageFileName, GraphID)` `VALUES (<imageFileName>, <graphID>);`	- Insert new image - `<imageFileName>` represents the value of `FormTaskSetting.imageFileName`
2.2_12	Task Setting Window	`DELETE FROM TASKS` `WHERE QuestionID = <questionID>;`	- Delete previous tasks - `<questionID>` represents the value of `FormTaskSetting.questionID`

2.2_13	Task Setting Window	`INSERT INTO TASKS (TaskDescription, StartingVertex,`
		`FinishingVertex, QuestionID, AnswerValue)`
		`VALUES ('<task.GetCurrentTaskText()>',`
		`'<task.GetComboBoxStartingVertex()>',`
		`'<task.GetComboBoxFinishigVertex()>',`
		`<quesionID>,`
		`<mapMatrix.Dijkstra(<StartingVertex>,`
		`<FinishingVertex>));`
			- Insert task of Dijkstra's algorithm
			- `<StartingVertex>` represents the value of `task.GetComboBoxStartingVertex()`
			- `<FinishingVertex>` represents the value of `task.GetComboBoxFinishingVertex()`

SQL Reference No.	Module Implemented	Pseudo-Command	Comments

2.2_14	Task Setting Window	`INSERT INTO TASKS (TaskDescription, StartingVertex, QuestionID, AnswerValue)`
		`VALUES ('<task.GetCurrentTaskText()>',`
		`'<task.GetComboBoxStartingVertex()>',`
		`<quesionID>,`
		`<mapMatrix.Prim(<StartingVertex>));`
			- Insert task of Prim's algorithm

2.2_15	Task Setting Window	`INSERT INTO TASKS(TaskDescription,QuestionID,AnswerValue)`
		`VALUES ('<task.GetCurrentTaskText()>',`
		`<quesionID>,`
		`<mapMatrix.Kruskal());`
			- Insert task of Kruskal's algorithm

2.2_16	Task Setting Window	`INSERT INTO TASKS(TaskDescription,QuestionID,GraphID)`
		`VALUES ('<task.GetCurrentTaskText()>',`
		`<quesionID>, <graphID>);`
			- Insert task of graph representations

2.3_1	List of Questions	`SELECT QuestionID, DateModified, QuestionName` `FROM QUESTIONBANK;`	- Query all the questions from the Question Bank

2.3.2_1	Edit Questions	`SELECT QuestionName, ProblemDescription, GraphID`
		`FROM QUESTIONBANK`
		`WHERE QuestionID = <questionID>;`
			- Query general information and the graph of the question
			- `<questionID>` represents the value of `FormQuestionBank.questionID`

2.3.2_2	Edit Questions	`SELECT GraphFormat FROM GRAPHS` `WHERE graphID = <graphID>;`	- Query the correct representation for the graph - `<graphID>` represents the value of `FormQuestionBank.graphID`
2.3.2_3	Edit Questions	`SELECT * FROM ADJACENCYMATRICES` `WHERE GraphID = <graphID>;`	- Query the graph in the form of an adjacency matrix
2.3.2_4	Edit Questions	`SELECT ImageFileName FROM GRAPHIMAGES` `WHERE GraphID = <graphID>;`	- Query the graph image

SQL Reference No.	Module Implemented	Pseudo-Command	Comments
2.3.2_5	Edit Questions	`SELECT TaskDescription, StartingVertex, FinishingVertex FROM TASKS WHERE QuestionID = <QuestionID>;`	- Query the subtasks
2.3.3_1	Delete Questions	`SELECT GraphID FROM QUESTIONBANK WHERE QuestionID = <QuestionID>;`	- Query graph ID for deletion - `<questionID>` represents the value of `FormQuestionBank.questionID`
2.3.3_2	Delete Questions	`SELECT GraphFormat FROM GRAPHS WHERE GraphID = <graphID>;`	- Query the format of the graph to decide which deletion process to be followed - `<graphID>` represents the value of `FormQuestionBank.graphID`
2.3.3_3	Delete Questions	`SELECT ImageFileName FROM GRAPHIMAGES WHERE GraphID = <graphID>;`	- Query the graph image for deletion
2.3.3_4	Delete Questions	`DELETE FROM GRAPHIMAGES WHERE GraphID = <graphID>;`	- Delete the record in GRAPHIMAGES table

2.3.3_5	Delete Questions	`DELETE FROM QUESTIONBANK WHERE QuestionID = <questionID>;`
		`DELETE FROM TASKS WHERE QuestionID = <questionID>;`
		`DELETE FROM GRAPHS WHERE GraphID = <graphID>;`
		`DELETE FROM ADJACENCYMATRICES WHERE GraphID = <graphID>;`
			- Delete question information, subtasks and the graph

2.3.4_1	Do Questions	`SELECT QuestionName, ProblemDescription, GraphID FROM QUESTIONBANK WHERE QuestionID = <questionID>;`	- Query information and the graph of the question - `<questionID>` represents the value of `FormQuestionBank.questionID`

SQL Reference No.	Module Implemented	Pseudo-Command	Comments
2.3.4_2	Do Questions	`SELECT GraphFormat FROM GRAPHS` `WHERE graphID = <graphID>;`	- Query the correct representation for the graph - `<graphID>` represents the value of `FormQuestionBank.graphID`
2.3.4_3	Do Questions	`SELECT * FROM ADJACENCYMATRICES` `WHERE GraphID = <graphID>;`	- Query the graph in the form of an adjacency matrix
2.3.4_4	Do Questions	`SELECT ImageFileName FROM GRAPHIMAGES` `WHERE GraphID = <graphID>;`	- Query the graph image

2.3.4_5	Do Questions	`SELECT TaskDescription, StartingVertex,`
		`FinishingVertex, AnswerValue`
		`FROM TASKS`
		`WHERE QuestionID = <QuestionID>;`
			- Query the subtasks and their answers

3.1_1	Account Settings	`SELECT ACCOUNTS.Username,`
		`<accountType>S.Forename,`
		`<accountType>S.Surname,`
		`<accountType>S.DateOfBirth,`
		`<accountType>S.Email,`
		`<accountType>S.School`
		`FROM ACCOUNTS, <accountType>S`
		`WHERE ACCOUNTS.AccountID = <accountID>`
		`AND <accountType>S.AccountID = <accountID>;`
			- Query current account information
			- `<accountType>` represents the value of `AccountMenu.accountType`

3.1_2	Account Settings	`UPDATE ACCOUNTS`
		`SET Username = '<textBoxUsername.Text>',`
		`Password = '<MD5(<textboxPassword.Text>)>',`
		`WHERE AccountID = <accountID>;`
			- Update account credentials
			- `<MD5()>` represents: `<MD5Hashing.Encrypt()>`

SQL Reference No.	Module Implemented	Pseudo-Command	Comments

3.1_3	Account Settings	`UPDATE <accountType>S`
		`SET Forename = '<textBoxForename.Text>',`
		`Surname = '<textBoxSurname.Text>',`
		`DateOfBirth = '<textBoxDateOfBirth.Text>',`
		`Email = '<textBoxEmail.Text>',`
		`School = '<textBoxSchool.Text>'`
		`WHERE AccountID = <accountID.Text>;`
			- Update personal information
			- `<accountType>` represents the value of `AccountMenu.accountType`

Implementation

Source code for the project

Please refer to **Appendix 2 - GraphTeachingTool Source Code.pdf** for the source code of this project.

Representative samples of techniques used in Group A (as indicated in the Example technical skills table) are also annotated in the code.

Completeness of solution

Module No.	Requirement	Met	Partially met	Not met	Reference to code
0	Log in operation	✓			FormLogin.cs: ButtonLogin_Click()
	Sign up operation	✓			FormLogin.cs: LinkLabelSignup_LinkClicked()
1	Validation: Rejection	✓			FormSignup.cs: - ValidateEmail() - ValidateSignUp() - MonthCalendar_DateSelected()
	Validation: Approval	✓			
	Accept the sign up request	✓			FormLogin.cs: - FormSignUp_buttonSignUp_Click() - OtherForms_FormClosed()
2	Go to the selected part of the main section	✓			FormPrimaryMenu.cs
2.1	Go to the topic overview window for the selected topic	✓			FormSelectTopics.cs
2.1.1.1	Show the objectives/prerequisites for learning Prim's Algorithm	✓			FormTopicOverview.cs: FormTopicOverview()
	Go to the step-by-step demonstration window for the example graph	✓			FormTopicOverview.cs: ButtonExample_Click()
2.1.2.1	Show the objectives/prerequisites for learning Kruskal's Algorithm	✓			FormTopicOverview.cs: FormTopicOverview()
	Go to the step-by-step demonstration window for the example graph	✓			FormTopicOverview.cs: ButtonExample_Click()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.1.3.1	Show the objectives/prerequisites for learning Dijkstra's Algorithm	✓			FormTopicOverview.cs: FormTopicOverview()
	Go to the step-by-step demonstration window for the example graph	✓			FormTopicOverview.cs: ButtonExample_Click()
2.1.1.2	Provide a full algorithm description for Prim's Algorithm, separated by steps	✓			FormPrimOnGraph.cs: Implemented by GUI design FormPrimOnMatrix.cs: Implemented by GUI design
	Show the selected example graph on the window	✓			FormPrimOnGraph.cs: - FormPrimOnGraph() - PanelGraph_Paint() FormPrimOnMatrix.cs: - FormPrimOnMatrix() - PanelGraph_Paint()
	Step forward	✓			FormPrimOnGraph.cs: ButtonNext_Click() FormPrimOnMatrix.cs: ButtonNext_Click()
	Step backward			✓	
	Illustrations on graph	✓			FormPrimOnGraph.cs: - PanelGraph_Paint() - EdgeFocusOn() - EdgeFocusOff() FormPrimOnMatrix.cs: - PanelGraph_Paint() - EdgeFocusOn() - EdgeFocusOff()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.1.1.2	User operations on graph	✓			FormPrimOnGraph.cs: - Vertex_MouseDown_ChooseStartingVertex() - LabelWeights_Click() FormPrimOnMatrix.cs: - DataGridViewGraph_CellContentClick()
	Finishing-up	✓			FormPrimOnGraph.cs: ButtonNext_Click() FormPrimOnMatrix.cs: ButtonNext_Click()
	Step-by-step demonstrations on a user-chosen graph instead of the default example graphs			✓	
2.1.2.2	Provide a full algorithm description for Kruskal's Algorithm, separated by steps	✓			FormKruskal.cs: Implemented by GUI design
	Show the selected example graph on the window	✓			FormKruskal.cs: - FormKruskal() - PanelGraph_Paint()
	Step forward	✓			FormKruskal.cs: ButtonNext_Click()
	Step backward			✓	
	Illustrations on graph	✓			FormKruskal.cs: - PanelGraph_Paint() - EdgeFocusOn() - EdgeFocusOff()
	User operations on graph	✓			FormKruskal.cs: LabelEdge_Click()
	Finishing-up	✓			FormKruskal.cs: ButtonNext_Click()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.1.2.2	Step-by-step demonstrations on a user-chosen graph instead of the default example graphs			✓	
2.1.3.2	Provide a full algorithm description for Dijkstra's Algorithm, separated by steps	✓			FormDijkstra.cs: Implemented by GUI design
	Show the selected example graph on the window	✓			FormDijkstra.cs: - FormDijkstra() - PanelGraph_Paint()
	Step forward	✓			FormDijkstra.cs: ButtonNext_Click()
	Step backward			✓	
	Illustrations on graph	✓			FormDijkstra.cs: - PanelGraph_Paint() - UndirectedEdgeFocusOn() - DirectedEdgeFocusOn() - UndirectedEdgeFocusOff() - DirectedEdgeFocusOff()
	User operations on graph	✓			FormDijkstra.cs: LabelVertexName_Click()
	Finishing-up	✓			FormDijkstra.cs: - ButtonNext_Click() - LabelVertexName_Click()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.1.3.2	Step-by-step demonstrations on a user-chosen graph instead of the default example graphs			✓	
2.2	Create a graph via adjacency matrix	✓			FormTaskSetting.cs: ButtonMatrix_Click()
	Create a graph via adjacency list	✓			FormTaskSetting.cs: ButtonList_Click()
	Create a graph via the Sketch Board	✓			FormTaskSetting.cs: ButtonSketchBoard_Click()
	Add, edit, and delete a subtask:	✓			FormTaskSetting.cs: - AddTaskControls() - ButtonAddTask_Click() - ButtonRemoveTask_Click() TaskSettingControls.cs: - ComboBoxTask_TextChanged() - ComboBoxVertex_TextChanged()
	Validation	✓			FormTaskSetting.cs: ValidateTasks()
	Save the question	✓			FormTaskSetting.cs: - SaveGraphToDatabase() - SaveQuestionToDatabase() - SaveTasksToDatabase() - FormTaskSetting_FormClosed() - ButtonSave_Click()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.2.1	Validation	✓			FormEditAdjacencyMatrix.cs: ButtonSubmit_Click()
	Save the graph	✓			FormEditAdjacencyMatrix.cs: ButtonSubmit_Click() FormTaskSetting.cs: - GraphEditingForms_ButtonSubmit_Click() - GraphEditingForms_FormClosed() - SaveGraph()
2.2.2	Validation	✓			FormEditAdjacencyList.cs: ButtonSubmit_Click()
	Save the graph	✓			FormEditAdjacencyList.cs: ButtonSubmit_Click() FormTaskSetting.cs: - GraphEditingForms_ButtonSubmit_Click() - GraphEditingForms_FormClosed() - SaveGraph()
2.2.3	Create a vertex	✓			FormSketchBoard.cs: - SetCurrentTool() - CreateVertex() - UpdateVertexNameCounter() - PanelSketchBoard_MouseDown() - ButtonVertex_MouseUp()
	Edit a vertex/edge	✓			FormSketchBoard.cs: - ResetBoard() - DisselectOthers() - DisselectAllVertices() - UpdateVertexNameCounter() - Vertex_MouseDown_TagState()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.2.3	Edit a vertex/edge *(cont.)*	✓			- Vertex_MouseDown_Disselect() - Vertex_MouseDown_DrawEdge() - Vertex_MouseDoubleClick() - Vertex_KeyPress_Delete() - PanelSketchBoard_MouseMove() - ButtonTag_MouseUp() - FormVertexTag_Load() - FormVertexTag_buttonSave_Click_ValidateVertexName() - FormVertexTag_buttonSave_Click_UpdateVertex() - FormVertexTag_buttonCancel_Click() - ButtonClearPanel_Click() FormVertexTag.cs: - TextBox_TextChanged() - ButtonSave_Click_ValidateWeights() VertexTagControls.cs: - CheckBoxContainsEdge_CheckChanged()
	Create an edge	✓			FormSketchBoard.cs: - SetCurrentTool() - ResetBoard() - PanelSketchBoard_MouseDown() - PanelSketchBoard_MouseMove() - ButtonEdge_MouseUp() - ButtonEdge_MouseDown() - TimerShowEdgeProperties_Tick()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.2.3	Create an edge *(cont.)*	✓			- ButtonDirected_Click() - ButtonUndirected_Click()
	Save the graph	✓			FormSketchBoard.cs: ButtonSubmit_Click() FormTaskSetting.cs: - GraphEditingForms_ButtonSubmit_Click() - GraphEditingForms_FormClosed() - SaveGraph()
2.3	Query the questions form the database	✓			FormQuestionBank.cs: - FormQuestionBank() - ResetDataGridViewQuestions()
	Filter/sort the questions		✓		FormQuestionBank.cs: Implemented by GUI design
2.3.1	Add questions	✓			FormQuestionBank.cs: - DataGridViewQuestions_CellDoubleClick() - ButtonAddQuestion_Click()
2.3.2	Edit questions	✓			FormQuestionBank.cs: - EditQuestion() - DataGridViewQuestions_CellDoubleClick() - ButtonEditQuestion_Click()
2.3.3	Delete questions	✓			FormQuestionBank.cs: ButtonDeleteQuestion_Click()
2.3.4	Do questions	✓			FormQuesitonBank.cs: ButtonDoQuestion_Click() FormDoQuestion.cs: - ButtonInputGraph_Click() - GraphEditingForms_buttonSubmit_Click() - GraphEditingForms_FormClosed()

Module No.	Requirement	Met	Partially met	Not met	Reference to code
2.3.4	Query the content of the information	✓			FormQuestionBank.cs: DoQuestion()
	Mark the question	✓			FormDoQuestion.cs: ButtonSubmit_Click()
2.3.4.1	Show answer	✓			FormDoQuestion.cs: ButtonShowAnswer_Click()
	Step-by-step explanation for a subtask			✓	
3.1	Query the account information	✓			AccountMenu.cs: ButtonAccountSettings_Click()
	Update account information	✓			AccountMenu.cs: - FormSignUp_buttonSignUp_Click() - FormSignUp_FormClosed()
3.2	Quit the system	✓			AccountMenu.cs: ButtonQuit_Click()

Example technical skills

NB As most of the techniques are implemented in a lot of places in the code, only representative samples will be shown here.

Group	Model/Algorithms	Representative reference to code
A	Complex data model in database	Program.cs: Relational database for the accounts and the question bank is defined there
	Cross-table parameterised SQL	Cross-table parameterised SQL used in: FormLogin.cs: - ButtonLogin_Click() FormTaskSetting.cs: - SaveGraphToDatabase() - SaveQuestionToDatabase() - SaveTasksToDatabase() FormQuestionBank.cs: - EditQuestion() - DoQuestion() - ButtonDeleteQuestion_Click()
	Aggregate SQL functions	FormTaskSetting.cs: - last_insert_rowid() used in SaveGraphToDatabase() - datetime() used in SaveQuestionToDatabase()
	User/CASE-generated DDL script	Program.cs: Main()

Group	Model/Algorithms	Representative reference to code
A	Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard	Graphs are constantly used thoughout the entire project: It is designed in: - Graph.cs as the general representation of a graph - AdjacencyMatrix.cs as the adjacency matrix representation of a graph - AdjacencyList.cs as the adjacency list representation of a graph And is also implemented in a wide range of relative areas in the project (please refer to the `AdjacencyMatrix` and `AdjacencyList` objects.) Trees are used for the minimum spanning tree algorithms: Graph.cs: - Prim_GetTree_Matrix() - Prim_GetTree_List() - Kruskal_GetTree_Matrix() - Kruskal_GetTree_List() Lists are constantly used throughout the entire project, for supporting the implementation of the graph structures and algorithms, and the dynamic generation of objects in the front end system, such as user controls. Representative samples: AdjacencyList.cs: - `List<AdjacentEdge> list` Graph.cs: - Prim(): `List<int> visitedVertices, remainingVertices` - Dijkstra(): `List<int> permanetVertices, tempraryVertices` - Dijkstra_GetShortestPath(): `List<int> shortestPath` - `List<UnionFind> unionFindVertices`

Group	Model/Algorithms	Representative reference to code
A	Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard *(cont.)*	FormSketchBoard.cs: - `List<Vertex> vertices` ShortestPathExample.cs: - `List<DijkstraVertexLabel> vertices` FormTaskSetting.cs: - `List<TaskSettingControls> taskControls` FormVertexTag.cs: - `List<VertexTagControls> edgeControls` FormDoQuestion.cs: - `List<DoTaskControls> taskControls`
	Graph/Tree Traversal	Graph traversed in Prim's, Kruskal's and Dijkstra's algorithms: Graph.cs: - Prim() - Prim_GetTree_Matrix() - Prim_GetTree_List() - Kruskal() - Kruskal_GetTree_Matrix() - Kruskal_GetTree_List() - Dijkstra() - Dijkstra_GetShortestPath() FormPrimOnGraph.cs FormPrimOnMatrix.cs FormKruskal.cs FormDijkstra.cs

Group	Model/Algorithms	Representative reference to code
A	List operations	Lists are constantly used and operated throughout the entire project, for supporting the implementation of the graph structures and algorithms, and the dynamic generation of objects in the front end system, such as user controls. (Please refer to the lists part in the “Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard” row in this table for representative samples.)
	Linked list maintenance	Linked list maintenance is used to maintain the Union-Find structure properties. Union-Find structure defined in: UnionFind.cs Linked list maintenance implemented in: Graph.cs: - InitialiseUnionFind() - Update() - Union() FormKruskal.cs: - InitialiseUnionFind() - Update() - Union()
	Hashing	MD5 Hashing algorithm is used in hashing user passwords. MD5 algorithm designed in: MD5Hashing.cs It is implemented in: FormLogin.cs: - ButtonLogin_Click() - FormSignUp_buttonSignUp_Click()

Group	Model/Algorithms	Representative reference to code
A	Files organised for direct access	Program.cs: Database.sqlite organised in Main() FormTaskSetting.cs: Graph image PNG files organised in SaveGraph()
	Recursive algorithms	Graph.cs and FormKruskal.cs: QuickSort()
	Complex user-defined algorithms (eg opetimisation, minimisation, scheduling, pattern matching) or equivalent difficulty	Prim's and Kruskal's minimum spanning tree algorithm, as well as Dijkstra's shortest path algorithms are implemented in the project. Refer to Graph.cs: - Prim() - Prim_GetTree_Matrix() - Prim_GetTree_List() - Kruskal() - Kruskal_GetTree_Matrix() - Kruskal_GetTree_List() - Dijkstra() - Dijkstra_GetShortestPath()
	Mergesort or similarly efficient sort	Graph.cs and FormKruskal.cs: QuickSort()
	Complex user-defined use of object-orientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces	OOP models are widely implemented throughout the entire project. All the object are in classes. Inheritance: The `AdjacencyMatrix` class and the `AdjacencyList` class inherits the `Graph` class.

Group	Model/Algorithms	Representative reference to code
A	Complex user-defined use of object-orientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces *(cont.)*	Polymorphism: Graph.cs: - SetEdge() - SetDirectedEdge() - SetUndirectedEdge() Vertex.cs: - SetEdge() FormKruskal.cs, FormPrimOnGraph.cs, FormPrimOnMatrix.cs: - EdgeFocusOn() - EdgeFocusOff() MinimumSpanningTreeExample.cs: - CreateEdge() - LabelFocusOn() - LabelFocusOff() Interfaces: Interfaces used in Graph.cs: - `IGraphAlgorithms` (Reference: IGraphAlgorithms.cs) - `IGraphOperations` (Reference: IGraphOperations.cs)
	Dynamic generation of objects based on complex user-defined use of OOP model	Dynamic generation of all the forms; (Please refer to all the .cs files with names beginning with “Form”) Dynamic generation of all the vertices: - Vertex.cs - DijkstraVertexLabel.cs

Group	Model/Algorithms	Representative reference to code
A	Dynamic generation of objects based on complex user-defined use of OOP model *(cont.)*	Dynamic generation of all the graph data structures: - AdjacencyMatrix.cs - AdjacencyList.cs Dynamic generation of the example graphs: - MinimumSpanningTreeExample.cs - MinimumSpanningTreeExample1.cs - MinimumSpanningTreeExample2.cs - ShortestPathExample.cs - ShortestPathExample1.cs - ShortestPathExample2.cs Dynamic generation of the combinations of user controls: - TaskSettingControls.cs - DoTaskControls.cs - VertexTagControls.cs Dynamic generation of other data structures or models: - UnionFind.cs - MD5Hashing.cs
B	Single table or non-parameterised SQL	Single-table parameterised SQL used in: FormLogin.cs: - ButtonLogin_Click() - FormSignUp_buttonSignUp_Click()

Group	Model/Algorithms	Representative reference to code
B	Single table or non-parameterised SQL *(cont.)*	FormSignUp.cs: - ValidateSignUp() FormTaskSetting.cs: - SaveGraphToDatabase() - SaveQuestionToDatabase() - SaveTasksToDatabase() FormQuestionBank.cs: - ResetDataGridViewQuestions() - EditQuestion() - DoQuestion() - ButtonDeleteQuestion_Click()
	Multi-dimensional arrays	Multi-dimentional arrays are widely used in the project, such as: AdjacencyMatrix.cs: - `double[,] map` MinimumSpanningTreeExample.cs and ShortestPathExample.cs: - `Label[,] labelWeights`
	Dictionaries	FormTopicOverview.cs: - `Dictionary<string, string> algorithmNames` - `Dictionary<string, List<string>> objectives` FormTaskSetting.cs: - `Dictionary<string, string> tasks`
	Simple user defined algorithms (eg a range of mathematical/statistical calculations)	FormSketchBoard.cs: - ResetBoard() FormDijkstra.cs: - PanelGraph_Paint()

Group	Model/Algorithms	Representative reference to code
C	Single-dimentional arrays	Single-dimentional arrays are widely used in the project, such as: AdjacencyList.cs: - `List<AdjacentEdge>[] list` FormSketchBoard.cs: - `bool[] vertexNameUsed`
	Appropriate choice of simple data types	Appropriate choice of int, double, Boolean, char, string data types thoughout the entire project.
	Simple mathematical calculations (eg average)	FormKruskal.cs, FormPrimOnGraph.cs, FormPrimOnMatrix.cs: - PanelGraph_Paint()

Coding styles

NB As most of the characteristics are demonstrated throughout the entire code, only representative samples will be shown here.

Style	Characteristic	Representative reference to code
Excellent	Modules (subroutines) with appropriate interfaces	Encapsulation used in the majority of classes, including but not limited to: - Class.cs - AdjacencyMatrix.cs - AdjacencyList.cs - UnionFind.cs - TaskSettingControls.cs - DoTaskControls.cs - VertexTagControls.cs

Style	Characteristic	Representative reference to code
Excellent	Loosely coupled modules (subroutines) – module code interacts with other parts of the program through its interface only	Module code only calls the encapsulated functions of other classes when interacting with those classes.
	Cohesive modules (subroutines) – module code does just one thing	In FormSketchBoard.cs, One event is divided into multiple functions, each doing only one task: 1.1 Vertex_MouseDown_TagState() 1.2 Vertex_MouseDown_Disselect() 1.3 Vertex_MouseDown_DrawEdge() 2.1 FormVertexTag_buttonSave_Click_ValidateVertexName() 2.2 FormVertexTag_buttonSave_Click_UpdateVertex()
	Modules (collections of subroutines) – subroutines with common purpose grouped	Functions with common purpose grouped within each class by `#region` blocks, for example: FormTaskSetting.cs: - #region Variables - #region Constructor - #region Operation Functions - #region Events for graph editing forms - #region Events for editing tasks - #region Events for editing tasks from Question Bank - #region Events for submission Functions within the same class are grouped in a single .cs file.

Style	Characteristic	Representative reference to code
Excellent	Defensive programming	Users are not able to enter invalid inputs, for example: - In FormLogIn.cs: MaxLength for textBoxUsername and textBoxPassword is 20, so users cannot enter anything longer than 20 characters; - In FormTaskSetting.cs: User can only select one of the subtasks from the drop-down menus for the subtasks, and can only select one of the vertices form the drop-down menus for the vertices; - In FormVertexTag.cs: MaxLength for textBoxVertexName is 1, so users can only enter 1 character for the name of a vertex. Validation functions used in: - FormLogin.cs: ButtonLogin_Click() - FormSignUp.cs: ValidateEmail(), ValidateSignUp() - FormSketchBoard.cs: FormVertexTag_buttonSave_Click_ValidateVertexName() - FormTaskSetting.cs: ValidateTasks() - FormVertexTag.cs: ButtonSave_Click_ValidateWeights()
	Good exception handling	`try-catch` clause used in: FormEditAdjacencyList.cs: ButtonSubmit_Click() FormEditAdjacencyList.cs: ButtonSubmit_Click() FormQuestionBank.cs: - EditQuestion() - DoQuestion() FormSignUp.cs: ValidateEmail() FormVertexTag.cs: ButtonSave_Click_ValidateWeights()

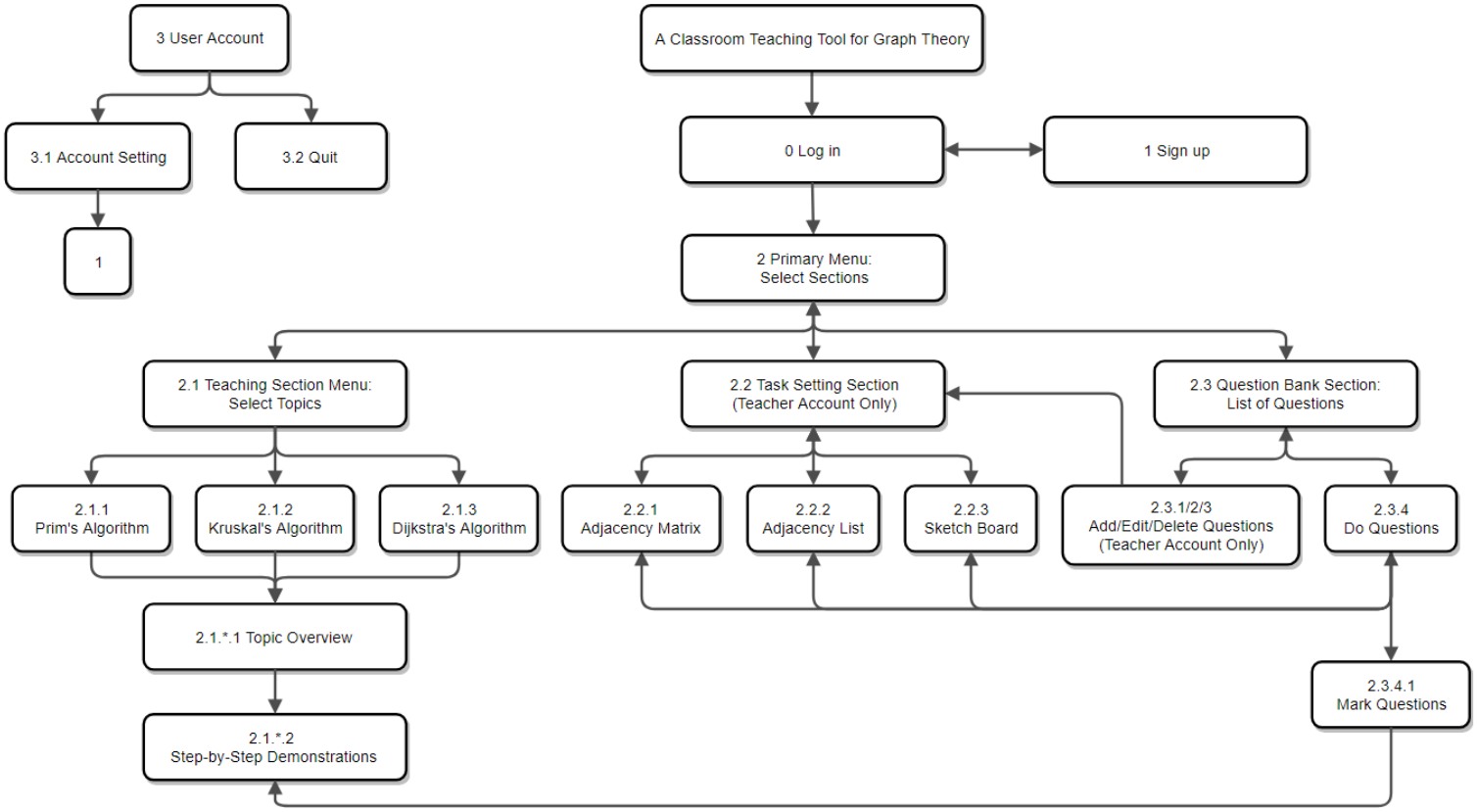
Style	Characteristic	Representative reference to code
Good	Well-designed user interface	Users only need to interact with the system through front-end, which have been achieved by GUI design.
	Modularisation of code	The entire code is divided into modules.
	Good use of local variables	Every function uses local variables.
	Minimal use of global variables	No global variable is used thourghout the entire system. Only necessary global variables are used within each class, for example: In FormQuestionBank.cs: - `string sql` - `FormTaskSetting formTaskSetting` - `FormDoQuestion formDoQuestion` `using (…) {…}` is used to reduce the number of global variables (object defined in (…) will be disposed immediately after the execution of {…}), including but not limited to: - All the implementation of SQL: SQLiteConnection, SQLiteCommand, and SQLiteDataReader; - Graphics - StreamReader - Bitmap.
	Managed casting of types	Almost all casting of types are done explicitly using the `Convert` class in C#. Please refer to all the appearances of `Convert` in the code. Almost all casting types to string operations are done using the `ToString()` function in C#. Please refer to all the appearances of `ToString()` in the code.

Style	Characteristic	Representative reference to code
Good	Use of constants	Constant maximum limit for the number of vertices in Graph.cs: - `const int SIZE = 26` Readonly dictionaries and lists in FormTopicOverview.cs (values of the dictionaries and lists are defined in the constructor function FormTopicOverview()): - algorithmNames - objectives_Prim - objectives_Kruskal - objectives_Dijkstra - objectives - prerequisites
	Appropriate indentation	Appropriate indentation is done throughout the code.
	Consistent style throughout	Coding style is consistent throughout the code.
Basic	Meaningful identifier names	Meaningful identifier names are used throughout the code, for example: - User controls: textBoxUsername, textBoxPassword in FormLogin.cs, indicating the text boxes for users to enter the username and the password; - Variables: graphEditingFormName in FormTaskSetting.cs, representing the name of the graph editing form that has been called; - Functions: Meaningful "Get" and "Set" functions used in Graph.cs, such as: SetEdge() SetDirectedEdge() SetUndirectedEdge()
	Annotation used effectively where required	Annotation used in almost every global variable, important functions, and used within a function to explain what those variables represent, what those functions do, and what effect a block of code has.

Testing

Testing plan

The system is to be tested module by module. Please refer to the hierarchical diagram in the **Requirements – Structure of the project** part of this document again:



Testing data

The following testing data are used to test the robustness of this project. Please also refer to the video about the results of testing.

NB The types of the testing data are shown in the following formats:

- Normal data: Green
- Extreme data (Correct): Light orange
- Extreme data (Erroneous): Orange
- Erroneous data: Red

Module 0 – Log in:

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Log in operation	Attempt to log in with an empty username and an empty password	Username: [Empty string] Password: [Empty string]	Error message: “Please enter your username and password!”	Passed
Log in operation	Attempt to log in with an empty username and either a valid or invalid password	Username: [Empty string] Password: ValidPassword	Error message: “Please enter your username!”	Passed
		Username: [Empty string] Password: short	Error message: “Please enter your username!”	Passed
		Username: [Empty string] Password: R34LlyLongP4ssWOrdButTooLongToBeTypedIn	User can only enter the first 20 characters of this password (R34LlyLong P4ssWOrdBu); Error message: “Please enter your username!”	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Log in operation	Attempt to log in with an empty password and either a valid or invalid username	Username: Victor.Zhao Password: [Empty string]	Error message: “Please enter your password!”	Passed
		Username: short Password: [Empty string]	Error message: “Please enter your password!”	Passed
		Username: ThisIsAReallyLongUsernameThatUsersCannotEvenTypeItIn Password: [Empty string]	User can only type in the first 20 characters of that username (ThisIsAReallyLongUse); Error message: “Please enter your password!”	Passed
Log in operation	Attempt to log in with either an invalid user-name or an invalid password or both	Username: short Password: ValidPassword	Error message: "Invalid username/password!"	Passed
		Username: ThisIsAnotherReallyLongUsernameThatExceedsTheCharacterLimit Password: ValidPassword	User can only enter the first 20 characters of this username (ThisIsAnotherReallyL); Error message: "Invalid username/password!"	Passed
		Username: Victor.Zhao Password: short	Error message: "Invalid username/password!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
		Username: Victor.Zhao Password: An0TH3rP4sSw0rDThatIsTooLongSoItCannotbeUsed	User can only enter the first 20 characters of this password (An0TH3rP4s Sw0rDThatI); Error message: "Invalid username/password!"	Passed
		Username: short Password: short	Error message: "Invalid username/password!"	Passed
		Username: ThisIsAReallyLongUsernameJustToShowHowTheValidationWorks Password: ThisIsAReallyLongPasswordWithTheSamePurpose	User can only enter the first 20 characters of this username (ThisIsAReall yLongUse), and the first 20 characters of this password (ThisIsAReally LongPas) Error message: "Invalid username/password!"	Passed
Log in operation	Attempt to log in with a username that is not in the database	Username: UsernameNotInDB Password: ValidPassword	Error message: "Invalid username/password!"	Passed
Log in operation	Attempt to log in with an incorrect password	Username: Victor.Zhao Password: IncorrectPassword	Error message: "Invalid username/password!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Log in operation	Attempt to log in with a username that is in the database, but in wrong cases, and an incorrect password	Username: VICTOR.ZHAO Password: IncorrectPassword	Error message: "Invalid username/password!"	Passed
Log in operation	Attempt to log in with a username that is in the database, but in wrong cases, and a correct password	Username: VICTOR.ZHAO Password: V3RyStr0n9P@$$W()rD	Error message: "Invalid username/password!"	Passed
Log in operation	Attempt to log in via SQL injection	Username: ' OR TRUE; -- Password: ValidPassword	Error message: "Invalid username/password!"	Passed
		Username: Victor.Zhao Password: ' OR TRUE; --	Error message: "Invalid username/password!"	Passed
		Username: ' OR TRUE; -- Password: ' OR TRUE; --	Error message: "Invalid username/password!"	Passed
Log in operation	Correct log in	Username: Sarah.Shakibi Password: Sarah.Shakibi	Proceed to the Primary Menu window (teacher account)	Passed
		Username: Victor.Zhao Password: V3RyStr0n9P@$$W()rD	Proceed to the Primary Menu window (student account)	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
		Username: UsernameLength=Limit Password: PasswordLength=Limit	Proceed to the Primary Menu window (student account)	Passed
		Username: short0 Password: Password	Proceed to the Primary Menu window (student account)	Passed
		Username: ' OR 0=0; -- Password: CorrectPassword	Proceed to the Primary Menu window (student account)	Passed
Sign up operation	Button testing	Click "Sign up" link label	Proceed to the Sign up window	Passed

Module 1 – Sign up

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Attempt to sign up with at least one required field empty	Username: [Empty string] Password: [Empty string] Repeat password: [Empty string] Forename: [Empty string] Surname: [Empty string] Date of birth: [Empty string] Email: [Empty string] School: [Empty string]	Error message: “Please choose a username!” “Please choose a password!” “Please enter your forename!” “Please enter your surname!” “Please enter your school!”	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Attempt to sign up with at least one required field empty	Username: NewUser Password: [Empty string] Repeat password: [Empty string] Forename: [Empty string] Surname: [Empty string] Date of birth: [Empty string] Email: [Empty string] School: [Empty string]	Error message: “Please choose a password!” “Please enter your forename!” “Please enter your surname!” “Please enter your school!”	Passed
		Username: NewUser Password: NewPassword Repeat password: [Empty string] Forename: [Empty string] Surname: [Empty string] Date of birth: [Empty string] Email: [Empty string] School: [Empty string]	Error message: “The repeated password does not match the password!” “Please enter your forename!” “Please enter your surname!” “Please enter your school!”	Passed
		Username: NewUser Password: NewPassword Repeat password: NewPassword Forename: [Empty string] Surname: [Empty string] Date of birth: [Empty string] Email: [Empty string] School: [Empty string]	Error message: “Please enter your forename!” “Please enter your surname!” “Please enter your school!”	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Attempt to sign up with at least one required field empty	Username: NewUser Password: NewPassword Repeat password: NewPassword Forename: NewForename Surname: [Empty string] Date of birth: [Empty string] Email: [Empty string] School: [Empty string]	Error message: “Please enter your surname!” “Please enter your school!”	Passed
		Username: NewUser Password: NewPassword Repeat password: NewPassword Forename: NewForename Surname: NewSurname Date of birth: [Empty string] Email: [Empty string] School: [Empty string]	Error message: “Please enter your school!”	Passed
Validation	Attempt to sign up with a username that already exists in the database	Username: Victor.Zhao Password: NewPassword Repeat password: NewPassword	Error message: “This username has already been taken!”	Passed
Validation	Attempt to sign up with either an invalid user-name or an invalid pass-word or both	Username: short Password: ValidPassword Repeat password: ValidPassword	Error message: “Invalid username!”	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Attempt to sign up with either an invalid username or an invalid password or both	Username: AReallyLongUsernameThatExceedsTheCharacterLimit Password: ValidPassword Repeat password: ValidPassword	User can only enter the first 20 characters of this username (AReallyLongUsernameT)	Passed
		Username: NewUser Password: short Repeat password: short	Error message: “Invalid password!”	Passed
		Username: NewUser Password: AVeryLongPasswordDefinitelyLongerThan20Characters Repeat password: AVeryLongPasswordDefinitelyLongerThan20Characters	User can only enter the first 20 characters of this password (AVeryLongPasswordDef)	Passed
		Username: AReallyLongUsernameJustToShowThatItCannotBeTypedIn Password: AReallyLongPasswordAlsoToShowThatItCannotBeTypedIn Repeat password: AReallyLongPasswordAlsoToShowThatItCannotBeTypedIn	User can only enter the first 20 characters of this username (AReallyLongUsernameJ), and the first 20 characters of this password (AReallyLongPasswordA)	Passed
Validation	Attempt to sign up with repeat password not matching the password	Username: NewUser Password: NewPassword Repeat password: DifferentPassword	Error message: “The repeated password does not match the password!”	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Attempt to sign up with repeat password not matching the password	Username: NewUser Password: NewPassword Repeat password: newpassword	Error message: “The repeated password does not match the password!”	Passed
Validation	Attempt to sign up with date of birth not in the past	On the month calendar provided, choose date of birth: 19 January 2038	This date does not show on the "Date of birth" textbox	Passed
		On the month calendar provided, choose date of birth: [Present day]	This date does not show on the "Date of birth" textbox	Passed
Validation	Attempt to sign up with invalid email address	Email: This is definitely not an email address	Error message: “Invalid email address!”	Passed
Validation	Attempt to sign up via SQL injection	Username: ' OR TRUE; -- Password: ' OR TRUE; -- Repeat password: ' OR TRUE; -- Forename: ' OR TRUE; -- Surname: ' OR TRUE; -- Date of birth: ' OR TRUE; -- Email: ' OR TRUE; -- School: ' OR TRUE; --	The "Date of birth" textbox is not editable; Error message: “Invalid email address!”	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Attempt to sign up via SQL injection	Username: ' OR TRUE; -- Password: ' OR TRUE; -- Repeat password: ' OR TRUE; -- Forename: ' OR TRUE; -- Surname: ' OR TRUE; -- Date of birth: [Empty string] Email: [Empty string] School: ' OR TRUE; --	No error message shown Proceed to the Log in window	Passed
Validation	Correct sign up	Username: NewTeacherAccount Password = Repeat password: NewTeacher Account Type: Teacher Forename: New Surname: Teacher Date of birth: 7/25/1972 Email: NewTeacher@example.com School: Example School	No error message shown Proceed to the Log in window	Passed
		Username: NewStudentAccount Password = Repeat password: NewStudent Account Type: Student Forename: New Surname: Student Date of birth: 1/1/2000 Email: NewStudent@example.com School: Example School	No error message shown Proceed to the Log in window	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Correct sign up	Username: Victor.ZHAO Password = Repeat password: AnotherVictorZhao:) Account Type: Student Forename: Another Victor Zhao Surname: with different cases Date of birth: 1/24/1999 Email: victor.zhao@ellesmere.com School: Ellesmere College	No error message shown Proceed to the Log in window	Passed
		Username: short1 Password = Repeat password: Password Account Type: Student Forename: Shortest username Surname: and shortest password Date of birth: 1/1/2000 Email: short1@example.com School: Example School	No error message shown Proceed to the Log in window	Passed
		Username: LongestValidUsername Password = Repeat password: LongestValidPassword Account Type: Student Forename: Longest username Surname: and longest password Date of birth: 1/1/2000 Email: long@example.com School: Example School	No error message shown Proceed to the Log in window	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	*(cont.)* Correct sign up	Username: TestAccount Password = Repeat password: TestAccount Account Type: Teacher Forename: Test for Surname: optional fields Date of birth: [Empty string] Email: [Empty string] School: Example School	No error message shown Proceed to the Log in window	Passed
Accept the sign up request	Database and button testing	With a correct sign up, click “Sign up” button	The account credential of the new account is saved in the ACCOUNT table; The personal information of the new account is saved in the TEACHER/STUDENT table, based on the account type; Proceed to the Log in window	Passed

Module 2 – Primary Menu

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Go to the selected part of the main section	Button testing	Click “Teaching Section” button	Proceed to the Teaching Section window	Passed
		Click “Set Tasks” button (on a teacher account)	Proceed to the Task Setting window	Passed
		Click “Question Bank” button	Proceed to the List of Questions window	Passed

Module 2.1 – Teaching Section Menu: Select Topics

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Go to the topic overview window for the selected topic	Button testing	Click “Prim’s Algorithm” button	Proceed to the Prim’s Minimum Spanning Tree Algorithm Topic Overview window	Passed
		Click “Kruskal’s Algorithm” button	Proceed to the Kruskal’s Minimum Spanning Tree Algorithm Topic Overview window	Passed
		Click “Dijkstra’s Algorithm” button	Proceed to the Dijkstra’s Shortest Path Algorithm Topic Overview window	Passed

Module 2.1.*.1 – Topic Overview

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Show the objectives and prerequisites for learning the selected algorithm	GUI testing	No input or operation required	Objectives, prerequisites and buttons for step-by-step demonstrations on example graphs are correctly shown	Passed
Go to the step-by-step demonstration window for the example graph	Button testing	Click any one of the “Example” buttons	Proceed to the Step-by-Step Demonstration window for the correct algorithm and on a correct example graph	Passed
			For the tabular version of Module 2.1.1.1 – Prim’s Algorithm Topic Overview, A correct table of the example graph are shown	Passed

Module 2.1.*.2 – Step-by-Step Demonstrations

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Provide a full algorithm description, separated by steps	GUI testing	No input or operation required	Correct algorithm description are shown, separated by steps	Passed
Show the selected example graph on the window	GUI testing	No input or operation required	Correct example graph are shown	Passed
Step forward	Button and GUI testing	Click “Next” button	All demonstrations (both textual and graphical) are functional	Passed
Illustrations on graph	GUI testing	No input or operation required	Correct vertices and edges are highlighted in correspondence with the current state of the graph on each step	Passed
User operations on graph	Button and GUI testing	Click on the nodes/edges when required	- The "Next" button is disabled until the operations on graph is done by users - Correct nodes/edges are recorded to perform the algorithm in the following steps - Correct textual explanations are shown in correspondence to the operations on graph - Correct nodes/edges are highlighted on the graph in correspondence to the operations	Passed
		For the tabular version of Module 2.1.1.2 – Prim's Algorithm Step-by-Step Demonstrations, click on the columns/ rows/entries on the table when required		Passed
		For Module 2.1.2.2 – Kruskal's Algorithm Step-by-Step Demonstrations click on the edges on the ordered list		Passed
Finishing-up	GUI testing	No input or operation required	- The "Next" button is disabled - Correct textual explanations and results are shown, with the graph correctly illustrated on the final state	Passed
		For Module 2.1.3.2 – Dijkstra's Algorithm Step-by-Step Demonstrations click on a vertex as the finishing vertex	Correct shortest path are shown on the graph	Passed

Module 2.2 – Task Setting Window (Teacher accounts only)

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Create a graph via adjacency matrix	Button testing	Click "Adjacency Matrix" button	Proceed to the Edit Adjacency Matrix window	Passed
Create a graph via adjacency list	Button testing	Click "Adjacency List" button	Proceed to the Edit Adjacency List window	Passed
Create a graph via the Sketch Board	Button testing	Click "Sketch Board" button	Proceed to the Sketch Board window	Passed
Validation	Attempt to add a question with no graph	Question name: Test question 0 Description: Test Graph: [No graph] Task 1: Find the Minimum Spanning Tree of the graph using Kruskal's Algorithm	Error message: "Please enter a graph!"	Passed
Validation	Attempt to add a question with no subtask	Question name: Test question 1 Description: Test Graph: [A vaild graph] Task 1: [Empty]	Error message: "Please enter at least a task!"	Passed
		Question name: Test question 2 Description: Test Graph: [A vaild graph] Task 1: [Empty] Task 2: [Empty] … Task 10: [Empty]	Error message: "Please enter at least a task!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Attempt to add a question with at least two repetitive subtasks	Question name: Test question 3 Description: Test Graph: [A vaild graph] Task 1: Find the Minimum Spanning Tree of the graph using Kruskal's Algorithm Task 2: Find the Minimum Spanning Tree of the graph using Kruskal's Algorithm	Error message: "Repeated task content at task 2!"	Passed
		Question name: Test question 4 Description: Test Graph: [A vaild graph] Task 1: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: A Task 2: Write the graph in adjacency matrix representation Task 3: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: B Task 4: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: A Task 5: Write the graph in adjacency matrix representation	Error message: "Repeated task content at task 4!" Error message: "Repeated task content at task 5!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Improper task: attempt to find the Minimum Spanning Tree for a directed graph	Question name: Test question 5 Description: Test Graph: [A vaild directed graph] Task 1: Find the Minimum Spanning Tree of the graph using Kruskal's Algorithm Task 2: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: A	Error message: "Task 1 is improper: Cannot find a Minimum Spanning Tree for a directed graph!" Error message: "Task 2 is improper: Cannot find a Minimum Spanning Tree for a directed graph!"	Passed
Validation	Improper task: starting/ finishing vertex not stated in the subtask when required	Question name: Test question 6 Description: Test Graph: [A vaild graph] Task 1: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: [Empty] Task 2: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: A - Finishing vertex: [Empty] Task 3: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: [Empty] - Finishing vertex: [Empty]	Error message: "Task 1 is improper: Starting vertex is empty!" Error message: "Task 2 is improper: Finishing vertex is empty!" Error message: "Task 3 is improper: Starting vertex is empty!" Error message: "Task 3 is improper: Finishing vertex is empty!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Improper task: starting/ finishing vertex not exist in the graph	Question name: Test question 7 Description: Test Graph: [A vaild graph containing vertices A, B, C, D, E] Task 1: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: W Task 2: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: A - Finishing vertex: X Task 3: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: Y - Finishing vertex: Z	Error message: "Task 1 is improper: Starting vertex does not exist in the graph!" Error message: "Task 2 is improper: Finishing vertex does not exist in the graph!" Error message: "Task 3 is improper: Starting vertex does not exist in the graph!" Error message: "Task 3 is improper: Finishing vertex does not exist in the graph!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	A combination of all the errors above	Question name: Test question 8 Description: Test Graph: [A vaild directed graph containing vertices A, B, C, D, E] Task 1: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: [Empty] Task 2: Find the Minimum Spanning Tree of the graph using Kruskal's Algorithm Task 3: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: X - Finishing vertex: Y Task 4: Write the graph in adjacency matrix representation Task 5: Write the graph in adjacency matrix representation	Error message: "Task 1 is improper: Cannot find a Minimum Spanning Tree for a directed graph!" Error message: "Task 1 is improper: Starting vertex is empty!" Error message: "Task 2 is improper: Cannot find a Minimum Spanning Tree for a directed graph!" Error message: "Task 3 is improper: Starting vertex does not exist in the graph!" Error message: "Task 3 is improper: Finishing vertex does not exist in the graph!" Error message: "Repeated task content at task 5!"	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Correct task setting (normal)	Question name: Test question 9|10|11 Description: Test Graph: [A vaild graph from Adjacency Matrix|Adjacency List|Sketch Board] Task 1: Find the Minimum Spanning Tree of the graph using Prim's Algorithm - Starting vertex: A Task 2: Find the Minimum Spanning Tree of the graph using Kruskal's Algorithm Task 3: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: B - Finishing vertex: C Task 4: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: B - Finishing vertex: D Task 5: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: C - Finishing vertex: D Task 6: Draw the graph corresponding to the adjacency list/matrix representation Task 7: Write the graph in adjacency matrix representation Task 8: Write the graph in adjacency list representation	No error message displayed	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Correct task setting (Question name, or description, or both are empty)	Question name: [Empty] Description: Test Graph: [A vaild graph] Task 1: Write the graph in adjacency matrix representation	No error message displayed Unnamed question is saved with a system default name in the format of "New Question + [Current date/time]" Description is allowed empty	Passed
		Question name: Test question 12 Description: [Empty] Graph: [A vaild graph] Task 1: Write the graph in adjacency matrix representation		Passed
		Question name: [Empty] Description: [Empty] Graph: [A vaild graph] Task 1: Write the graph in adjacency matrix representation		Passed
Validation	Correct task setting (Some of the input boxes for tasks are empty but at least one is valid)	Question name: Test question 13 Description: [Empty] Graph: [A vaild graph] Task 1-5: [Empty] Task 6: Write the graph in adjacency matrix representation Task 7-10: [Empty]	No error message displayed All the input boxes for empty tasks are removed	Passed
Validation	Correct task setting (SQL attempts)	Question name: dummy', 'dummy'); Description: DROP TABLE QUESTIONBANK;-- Graph: [A vaild graph] Task 1: Write the graph in adjacency matrix representation	No error message displayed SQL injection commands are regarded as string parameters	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Correct task setting (Very long question name and description)	Question name: Test question 14 #######... (Until maximum character limit (140 characters) is reached) Description: TestTestTest… (Until maximum character limit (1500 characters) is reached) Graph: [A vaild graph] Task 1: Write the graph in adjacency matrix representation	No error message displayed	Passed
Validation	Correct task setting (Dense graph)	Question name: Test question 15|16|17 Description: Test Graph: [A vaild dense graph from Adjacency Matrix|Adjacency List|Sketch Board] Task 1: Find the Shortest Path using Dijkstra's Algorithm - Starting vertex: A - Finishing vertex: Z	No error message displayed	Passed
Save the question	Database testing	No input or operation required	Problem descriptions, graphs and subtasks are correctly saved in the database	Passed

Module 2.2.1 – Edit Adjacency Matrix

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Attempt to enter a invalid weight	Edge BC: InvalidData Edge DA: -123.4	Error message: "Invalid input at row: B, column: C!" Error message: "Negative weight at row: D, column: A!"	Passed
	Attempt to enter a negative weight			
Validation	Correct inputs	[Correct weight entries]	No error message displayed	Passed
Save the graph	Button and GUI testing	With correct inputs, click "Submit" button	Proceed to the Task Setting window, with the adjacency matrix correctly shown	Passed

Module 2.2.2 – Edit Adjacency List

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Attempt to enter a invalid string	Adjacent Edges of A: InvalidData Adjacent Edges of B: A,1.5,C,D,2.5,2,E,0.5 Adjacent Edges of C: A,B,2,C,3,D,4 Adjacent Edges of D: A,-123.4	Error message: "Invalid input at vertex A!" Error message: "Invalid input at vertex B!" Error message: "Self loop at vertex C!" Error message: "Negative edge weight at edge DA!"	Passed
	Attempt to enter a weight without a preceding vertex name			
	Attempt to create a self loop			
	Attempt to enter a negative weight			

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Validation	Correct inputs	[Correct weight entries]	No error message displayed	Passed
Save the graph	Button and GUI testing	With correct inputs, click "Submit" button	Proceed to the Task Setting window, with the adjacency list correctly shown	Passed

Module 2.2.3 – Sketch Board

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Create a vertex	Button and GUI testing	Click on the "Vertex" button, then single click on the Sketch Board panel (Normal)	A new vertex with default vertex name is created on the single click position	Passed
Create a vertex	Button and GUI testing	Click on the "Vertex" button, create 26 vertices on the Sketch Board panel, and then attempt to create another new vertex by single clicking on the Sketch Board panel (Erroneous)	Message box displayed: "Maximum number of vertices has been reached! (You can create up to 26 vertices)"	Passed
Create an edge	Button and GUI testing	Click on the "Edge" button, single click on a vertex on the Sketch Board panel, and then single click on another vertex on the Sketch Board panel (Normal)	An undirected edge with weight 1 between the two clicked vertices is drawn on the Sketch Board panel	Passed
Create an edge	Button and GUI testing	Click on the "Edge" button, single click on a vertex on the Sketch Board panel, and then single click on a blank place on the Sketch Board panel (the number of vertices does not exceed the maximum limit) (Normal)	A new vertex with default vertex name is created on the single click position; An undirected edge with weight 1 between the clicked vertex and the newly created vertex is drawn on the Sketch Board panel	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Create an edge	Button and GUI testing	Click on the "Edge" button, single click on a vertex on the Sketch Board panel, and then single click on a blank place on the Sketch Board panel (the number of vertices is already at the maximum limit) (Erroneous)	Message box displayed: "Maximum number of vertices has been reached! (You can create up to 26 vertices)"; No new edge is created	Passed
Create an edge	Button and GUI testing	Click on the "Edge" button, single click on a blank place on the Sketch Board panel, and then single click on a vertex on the Sketch Board panel (Normal)	A new vertex with default vertex name is created on the single click position; An undirected edge with weight 1 between the newly created vertex and the clicked vertex is drawn on the Sketch Board panel	Passed
Create an edge	Button and GUI testing	Long press the "Edge" button, wait for the "Directed edge" button and the "Undirected edge" button to appear, click on the "Directed edge button", then repeat the previous 4 "Create an edge" testing operations	The "Directed edge" button (shown as ↗) and the "Undirected edge" button (shown as ／) are shown successfully, and disappear after one of them is clicked;	Passed
			A directed edge with weight 1 from the first clicked vertex to the second clicked vertex is drawn on the Sketch Board panel in all the above 4 "Create an edge" testing operations	Passed
Edit a vertex	Button and GUI testing	Click on the "Vertex" button, and attempt to drag a vertex (Normal)	The vertex is successfully dragged, along with all its connecting edges	Passed
Edit a vertex	Button and GUI testing	Click on the "Tag" button, and double click on a vertex (Normal)	A vertex tag window with the name of the vertex and its connecting edges is shown	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Edit a vertex/edge	Attempt to enter an invalid name for the vertex	On the vertex tag window: enter vertex name: "@"	Error message "Invalid name!"	Passed
Edit a vertex/edge	Attempt to enter a valid name for the vertex	On the vertex tag window: enter vertex name: [A valid vertex name different from the original vertex name]	No error message is shown The vertex tag window is closed, and the name of the vertex is changed on the Sketch Board panel	Passed
Edit a vertex/edge	Attempt to change the weight of an already existing edge	On the vertex tag window: Change the weight of an already existing edge (Normal)	The weight of the edge is changed *in one direction* from this vertex to the destination vertex	Passed
Edit a vertex/edge	Button and GUI testing	On the vertex tag window: Uncheck an edge (Normal)	The edge is removed *in one direction* from this vertex to the destination vertex	Passed
Edit a vertex/edge	Attempt to create a new directed edge from the vertex	On the vertex tag window: Check a destination vertex where there does not exist an edge from this vertex to the destination vertex, and set a weight for the edge (Normal)	A new directed edge with the specified weight is formed from this vertex to the destination vertex	Passed
Edit a vertex/edge		On the vertex tag window: Check a destination vertex where there does not exist an edge from this vertex to the destination vertex, and leave the weight for the edge blank (Erroneous)	Error message "Invalid input for weight *XY*: Please input a positive real number!" ***NB*** *X* represents the current vertex, and *Y* represents the destination vertex	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Edit a vertex/edge	Attempt to create a new directed edge from the vertex *(cont.)*	On the vertex tag window: Check a destination vertex where there does not exist an edge from this vertex to the destination vertex, and enter an invalid string in the weight textbox (Erroneous)	Error message "Invalid input for weight *XY*: Please input a positive real number!" ***NB*** *X* represents the current vertex, and *Y* represents the destination vertex	Passed
Edit a vertex/edge		On the vertex tag window: Check a destination vertex where there does not exist an edge from this vertex to the destination vertex, and set a negative weight for the edge (Erroneous)	Error message "Invalid input for weight *XY*: Please input a positive real number!" ***NB*** *X* represents the current vertex, and *Y* represents the destination vertex	Passed
Edit a vertex/edge		On the vertex tag window: Check a destination vertex where there does not exist an edge from this vertex to the destination vertex, and enter 0 for the weight of the edge (Extreme erroneous)	Error message "Invalid input for weight *XY*: Please input a positive real number!" ***NB*** *X* represents the current vertex, and *Y* represents the destination vertex	Passed
Clear the panel	Button and GUI testing	Click the "Clear" button	The Sketch Board panel is cleared	Passed
Save the graph	Button and GUI testing	Click the "Submit" button	Proceed to the Task Setting window, with the graph image correctly shown	Passed

Module 2.3 – Question Bank Section: List of Questions

Module 2.3.1 – Add Questions

Module 2.3.2 – Edit Questions

Module 2.3.3 – Delete Questions

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Query the questions from the database	Database and GUI testing	No operation needed	The list correctly shows the question name and date modified of all the questions, consistent to the value stored in the QUESTIONBANK table in the database	Passed
Sort the questions	GUI testing	Click on the column headers	The questions are sorted in ascending or decending order with respect to data modified or question name	Passed
Add questions	Button and GUI testing	Requires teacher account Click the "Add Question" button	A new Task Setting window is shown	Passed
Add questions	Database and GUI testing	Requires teacher account After the new question is edited (validatation is also passed), click the "Submit" button on the Task Setting window	The questions, graphs, subtasks are stored in the database in the correct tables;	Passed
			The answer to each subtask are correctly calculated and saved in the database;	Passed
			Go back to the List of Questions window, with the newly added question shown on the list.	Passed
Edit questions	Button, GUI and database testing	Requires teacher account Select a question from the list, then click the "Edit Question" button	A Task Setting window is shown with the current contents of the question loaded correctly in place, consistent to the value stored in the database.	Passed
		Requires teacher account Double click on a question from the list		

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Edit questions	Database and GUI testing	Requires teacher account After the question is edited (validatation is also passed), click the "Submit" button on the Task Setting window	The questions, graphs, subtasks are updated in the database in the correct tables;	Passed
			The answer to each subtask are correctly calculated and saved in the database;	Passed
			Go back to the List of Questions window, with the information of the edited question updated on the list.	Passed
Delete questions	Button, GUI and database testing	Requires teacher account Select a question from the list, then click the "Delete Question" button	The questions, graphs, subtasks and answers are correctly deleted from the database;	Passed
			The deleted question is also removed from the list of questions.	Passed
Proceed to the Do Question window	Button and GUI testing	Select a question from the list, then click the "Do Question" button	Proceed to the Do Question window	Passed
		On a student account: double click on a question from the list		

Module 2.3.4 – Do Questions

Module 2.3.4.1 – Mark Questions

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Query the content of the question	Database and GUI testing	No operation needed	The Do Question window correctly shows the question name, problem description, graph, and subtasks of the selected questions, consistent to the data stored in the database	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Do questions	Attempt to enter a graph in adjacency matrix as required (Button and GUI testing)	1. Click the "Adjacency Matrix" button	A new Edit Adjacency Matrix window is shown.	Passed
		2. Enter several valid entries in the Edit Adjacency Matrix window, then click the "Save" button	Go back to the Do Question window.	Passed
		3. Click the same "Adjacency Matrix" button again	The Edit Adjacency Matrix window is shown, along with the correct previous workings.	Passed
Do questions	Attempt to enter a graph in adjacency list as required (Button and GUI testing)	1. Click the "Adjacency List" button	A new Edit Adjacency List window is shown.	Passed
		2. Enter several valid entries in the Edit Adjacency List window, then click the "Save" button	Go back to the Do Question window.	Passed
		3. Click the same "Adjacency List" button again	The Edit Adjacency List window is shown, along with the correct previous workings.	Passed
Do questions	Attempt to draw a graph using the Sketch Board as required	1. Click the "Sketch Board" button	A new Sketch Board window is shown.	Passed
		2. On the Sketch Board window, click the "View Task" button	A temporary readonly window of the task graph in the correct representation format is shown	Passed
		3. Draw a valid graph on the Sketch Board window, then click the "Submit" button	Go back to the Do Question window.	Passed
Mark the question	Button and GUI testing	Click the "Mark it!" button with the presence of both correct answers and wrong answers	Correct answers are marked with green √, and wrong answers are marked with red ×	Passed
			Total marks given are correctly shown in the "Your score" label, in red	Passed
		Click the "Mark it!" button with all answers correct	Full marks are correctly shown in the "Your score" label, in green	Passed

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Show answer	Button, GUI and database testing	Click the "Answer" button for a subtask with numerical answer	Correct answer value is shown, consistent to the values stored in the database	Passed
			The "Answer" button is now named "Hide"	Passed
			For the correctly attempted subtasks, the answer labels are green, and for the incorrectly attempted subtasks, the answer labels are red	Passed
Show answer	Button, GUI and database testing	Click the "Answer" button for a subtask with graphical answer in adjacency matrix	A readonly Adjacency Matrix window is shown, with the correct entries of the answer matrix, consistent to the values stored in the database	Passed
Show answer	Button, GUI and database testing	Click the "Answer" button for a subtask with graphical answer in adjacency list	A readonly Adjacency List window is shown, with the correct entries of the answer list, consistent to the values stored in the database	Passed
Hide answer	Button and GUI testing	Click the "Hide" button for a subtask with numerical answer	The label of answer value is hidden; The "Hide" button is now named "Answer".	Passed
Hide answer	Button and GUI testing	Close the readonly Adjacency Matrix/List window	Go back to the Do Question window.	Passed

Module 3 – User Accounts

Module 3.1 – Account Setting

Module 3.2 – Quit

Requirement	Description	Inputs or Operation	Expected Outcome	Result
Show account operation buttons	Button and GUI testing	Click the  button on the User Account menu	A "Settings" button and a "Quit" button is shown; The  button now becomes  .	Passed
Hide account operation buttons	Button and GUI testing	Click the  button on the User Account menu	The "Settings" and "Quit" buttons are hidden; The  button now becomes  .	Passed
Query the account information	Button, database and GUI testing	Click the  button on the User Account menu, then click the "Settings" button	A Sign up window is shown, along with the correct account information of the current account loaded in place, consistent to the value stored in the database	Passed
Update account information	Button, database and GUI testing	After the account information is edited (validation is also passed), click the "Submit" button	The account information is updated correctly in the database;	Passed
			Go back to the previous window where the Account Setting request is called.	Passed
Quit the system	Button and GUI testing	Click the  button on the User Account menu, then click the "Quit" button	Quit the entire system.	Passed

Evaluation

Feedbacks from users

The final version of the system has been tested by the following users:

- Mr John Cowley (JHC): Head of Mathematics Department of Ellesmere College;
- Mr Peter Hayes (PJH): Teacher of Mathematics of Ellesmere College, in charge of teaching Decision 1 for A-Level further mathematics students;
- Mr Thomas Hurst (TDH): Teacher of Design and Technology of Ellesmere College.

Feedback emails from the above mentioned users have been obtained and analysed:

Mr John Cowley

Feedback from JHC is copied below:
(Please refer to **Appendix 4 - Original feedback emails from users.pdf** for the original email)

Victor,

The programs now work ok.

As a learner I would still appreciate the opportunity to go back a step to consolidate my learning. Your program only allows me to go forwards through it.
I know I can reload the tool and start again which is useful but a little time-consuming.

JHC

Suggestion	Comments	Analysis
Enable stepping backwards in Step-by-Step Demonstration	This is possible to be implemented	The naïve implementation of this functionality may require temporarily storing the details of the algorithm and front-end GUI states for every step, so efficiency will be a challenge to overcome.

Mr Peter Hayes

Feedback from PJH is copied below:

(Please refer to **Appendix 4 - Original feedback emails from users.pdf** for the original email and attachment)

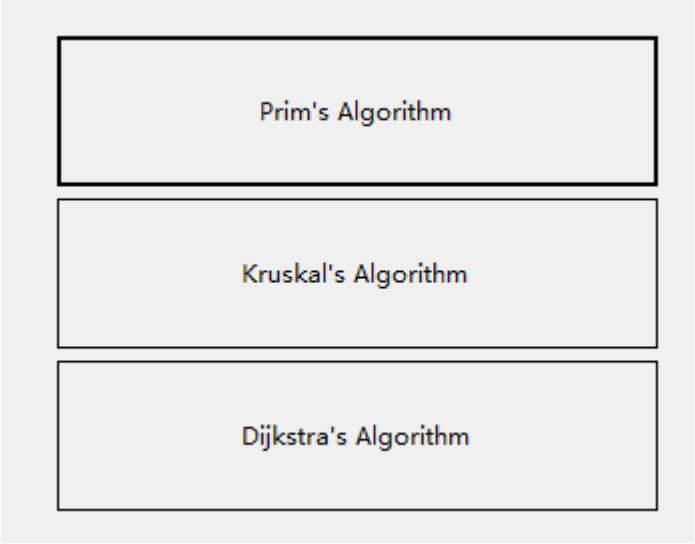
Victor – graph teaching app

Sign-up

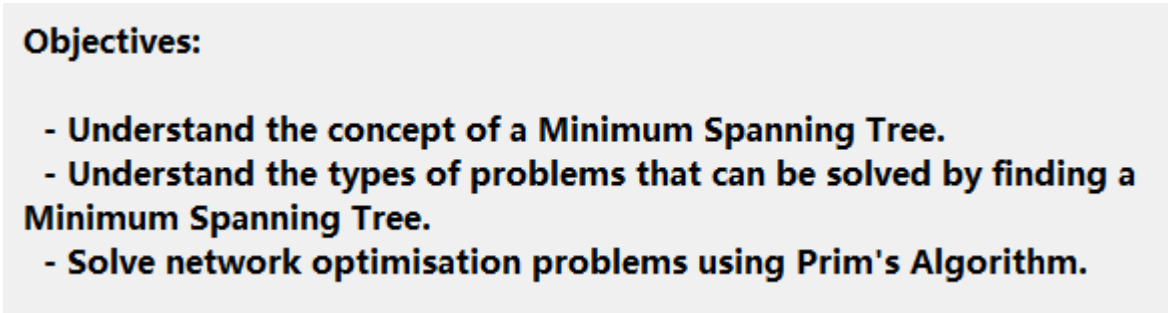
- *Date of birth only moves month by month*
- *When it came to log-in a different user name appeared and invalid name error came up*

Teaching section

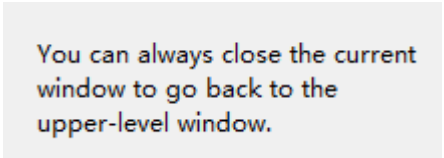
- *Titles in boxes very small font – rather uninteresting page*



Prim's

- 

  No explanation of these objectives before starting examples so I do not know what a minimum spanning tree is

- 

Would prefer a box to choose

- **Prim's Minimum Spanning Tree Algorithm: (Graphical version)**

  STEP 1 Choose an arbitrary vertex of the graph.

  STEP 2 Add an edge of minimum weight joining a vertex already included to a vertex not already included.

  STEP 3 If a spanning tree is obtained then STOP; otherwise return to STEP 2.

Better to have only one step showing – otherwise learner may not bother to read.

- **STEP 3** If a spanning tree is obtained then STOP; otherwise return to STEP 2.

  We have not yet formed a Minimum Spanning Tree, so go back to STEP 2.

If I do not know what a minimum spanning tree is how do I know I need to return to step 2?

- **Now we have picked 6 edges and has formed a Minimum Spanning Tree.**
  Therefore Prim's algorithm has finished.

Explain why 6 edges tells me the minimum spanning tree is formed.
Better to end by drawing a separate diagram with minimum spanning tree only.

Tabular version

- **Prim's Minimum Spanning Tree Algorithm: (Tabular version)**

  STEP 1 Cross through the entries in an arbitrary row, and mark the corresponding column.

  STEP 2 Choose a minimum entry from the uncircled entries in the marked column(s).

  STEP 3 If no such entry exists then STOP; otherwise go to STEP 4.

  STEP 4 Circle the weight w(i, j) found in STEP 2; mark column j; cross through row i.

  STEP 5 Return to STEP 2.

Again too much at once.

Step 4 – difficult to understand – where are I, j, defined?

- *Found it difficult to choose vertex*
- **Edge CA has the minimum weight from the uncircled entries in the marked column(s) (3), therefore it has been chosen.**
  Now that we have found an entry, we should go to STEP 4.

  The chosen value goes bold and remains uncircled!
- **Edges CB, AB have the same weight (8). Please pick one of your choice:**
  Please click on the weights in the tableau.

  Why can't I choose an edge on the graph? Not easy to find in table for new learner.
	A 2	B 3	C 1	D 4	E 7	F 6	G 5
A	–	~~8~~	3	–	–	–	~~12~~
B	~~8~~	–	8	~~9~~	–	–	–
~~C~~	~~3~~	~~8~~	–	~~16~~	–	–	~~15~~
D	–	9	~~16~~	–	~~14~~	~~14~~	~~18~~
E	–	–	–	14	–	~~15~~	–
F	–	–	–	~~14~~	~~15~~	–	6
G	12	–	~~15~~	~~18~~	–	~~6~~	–

  Algorithm complete – no circled values.
- *Font size again very small. I would prefer to see values centred in box and line crossing out values to be longer.*

Kruskal's

- **Edges BC, AB have the same weight. Please pick one of your choice.**
  Please click on the edge in the list (NOT on the graph).

  I would prefer to click on graph or even to have a choice.

Dijkstra's

-

Dijkstra's Shortest Path Algorithm:

STEP 1 Make the given start vertex permanent by giving it permanent label 0 and order label 1.

STEP 2 For each vertex that is not permanent and is connected by an arc to the vertex that has just been made permanent (with permanent label = P), add the arc weight to P.
If this is smaller than the best temporary label at the vertex, write this value as the new best temporary label.

STEP 3 Choose the vertex that is not yet permanent which has the smallest best temporary label. If there is more than one such vertex, choose any one of them.
Make this vertex permanent and assign it the next order label.

STEP 4 If every vertex is now permanent, or if the target vertex is permanent, use 'trace back' to find the routes or route, then STOP;
otherwise return to STEP 2.

Too much information at once.

-

You have chosen vertex C.
It has been made permanent by being given permanent label 0 and order label 1.

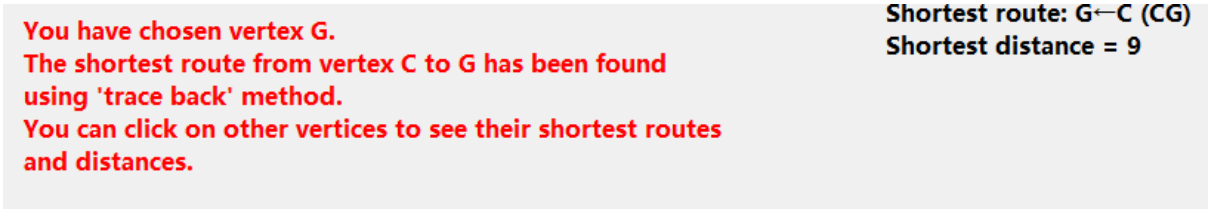
Need to explain meaning of these numbers.

-

Vertices A, B, D, G have been given a new temporary label.

Too fast – do one at a time.

- *"smallest best" - ? English*
- *Graph keeps flashing when Next is pressed.*

- 

  I chose vertex G which is not very interesting. There seems to be no way to go back and choose a different vertex.
- *"trace back" method needs to be explained.*

Suggestion	Comments	Analysis
Date of birth only moves month by month	This relate to the intuitiveness of the project, and is possible to be implemented	Actually the users can click on the header of the calendar to go to the years, decades and centuries. However, due to lack of instruction, users cannot know that instinctively. This can be solved by enabling the users to type in the date of birth textbox. Validation is required to ensure that the syntax of the date of birth is correct.
When it came to log-in a different user name appeared and invalid name error came up	This relate to the detail of the project, and is easy to be implemented	Every time the system goes back to the log-in window, the contents in the username textbox and the password textbox should be cleared.
Titles in boxes with small font, and the page is uninteresting	This relates to the GUI design of the project, and is easy to be implemented	Increase the font size of the text, and add aesthetic design to the pages.
No explanation of these objectives before starting examples	This relates to the explanation of the teaching tool, and is easy to be implemented	Add explanation to the objectives (for example, define the minimum spanning tree in the objectives)
Prefer a box to choose to go back to the upper-level window	This is easy to be implemented	Add a "Return to previous page" button that closes this window when clicked, so it will trigger the same event of closing the window, and go back to the upper-level window.

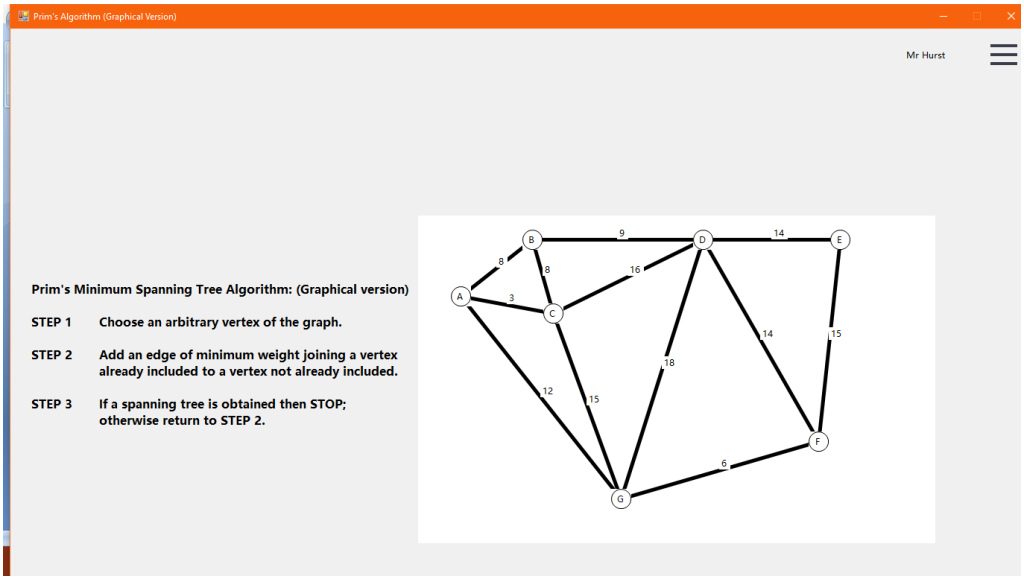
Suggestion	Comments	Analysis
To much content at once Better to have only one step showing – otherwise learner may not bother to read	This is easy to be implemented	Extend the steps before starting the Step-by-Step Demonstration of the algorithm, in the way that when the users click the "Next" button, only one more step is shown on the system, until all steps have been shown.
Better to end by drawing a separate diagram with minimum spanning tree only.	This relates to the GUI design of the project, and is possible to be implemented	When the minimum spanning tree algorithm is finished, add another panel to the Step-by-Step Demonstration window to show the minimum spanning tree only.
Step 4 in Prim's algorithm (tabular version): i, j are not defined	This relates to the explanation of the teaching tool, and is easy to be implemented	Define i, j in Step 4.
In Prim's algorithm (tabular version) and Kruskal's algorithm: cannot choose an edge on the graph – Not easy to find in table for new learner	This is possible to be implemented	Add events to the edges on the graph so that users can choose an edge on the graph. This would be the same event in Prim's algorithm (graphical version).
In Prim's algorithm (tabular version): the chosen value goes bold and remains uncircled	This relates to the GUI design of the project, and is hard to be implemented	Making the chosen value bold is a compensation to the GUI as there isn't a way to circle the label. This is definitely possible to be implemented, but will have to seek other approach to circle the values.
Need to explain the meanings of the permanent label and the temporary label in Dijkstra's algorithm	This relates to the explanation of the teaching tool, and is easy to be implemented	Explain the manings of the pamernent label and the temporary label in the steps.
In Dijkstra's algorithm, temporary labels of multiple vertices are updated too fast	This is easy to be implemented	When the users click the "Next" button, the temporary labels of vertices are updated one at a time.
English wording problem in Dijkstra's algorithm: "smallest best"	The wording is adapted from the reformed linear A-Level mathematics formula booklet It is easy to be altered	Change the wording.

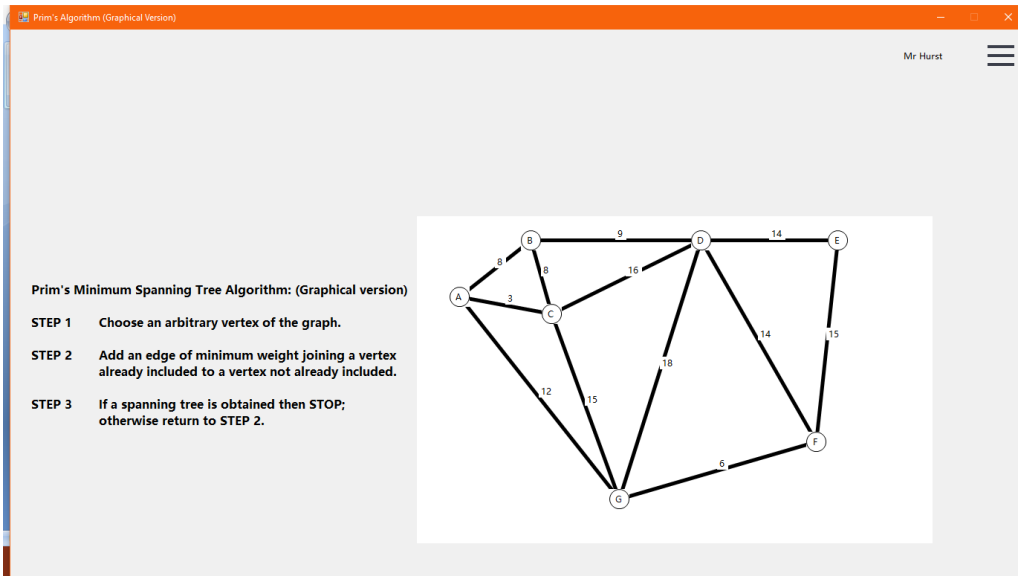
Suggestion	Comments	Analysis
In Dijkstra's algorithm: graph keeps flashing when the "Next" button is pressed	This is due to the entire graph is redrawn when the "Next" button is clicked. It is possible to optimise the processing	When the "Next" button is clicked, only redrawn the affected edges.
In Dijkstra's algorithm: there seems to be no way to go back and choose a different vertex	This relate to the intuitiveness of the project, and is easy to be implemented	Actually the users can just click on another vertex on the graph to choose a different vertex. However, the hint "You can click on other vertices to see their shortest routes and distances" is not obvious enough to be seen immediately. This can be solved by making the hint more obvious.
In Dijkstra's algorithm: “Trace back” method needs to be explained	This relates to the explanation of the teaching tool, and is easy to be implemented	Explain the "trace back" method in the steps.

Mr Thomas Hurst

Feedback from TDH is copied below:

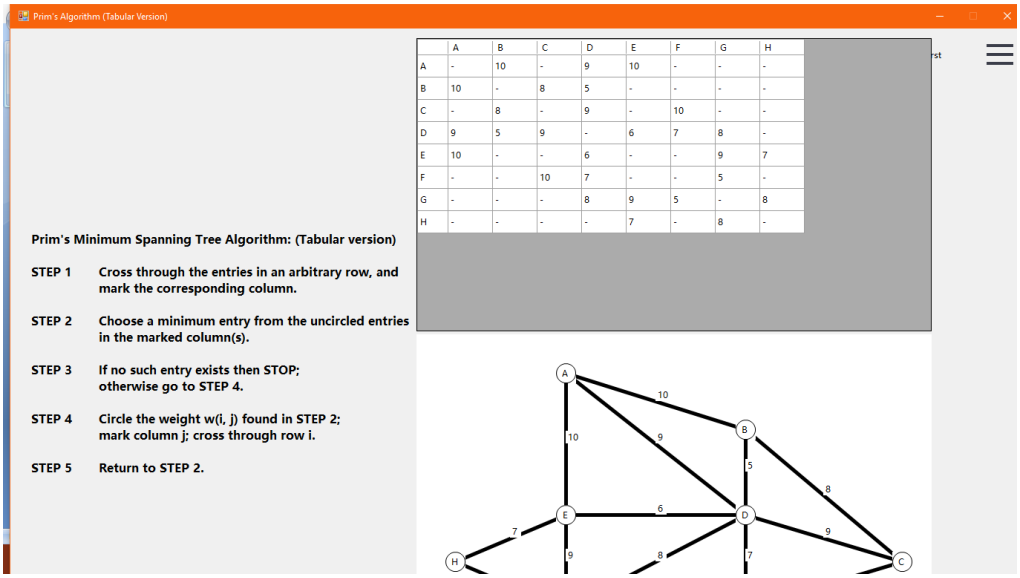
(Please refer to **Appendix 4 - Original feedback emails from users.pdf** for the original email and attachment)

- *Ability to simply type in Date of Birth rather than have to use the calendar function to either scroll back month by month or move up levels through years and then decades to find the right date.*
- *The only way to close the calendar on the date field is to press the X button. Clicking on the next field or using tab to move through the fields works but the calendar still obscures the next field*
- *Not sure if it is due to the resolution on my laptop but any new window sits off centre of the screen for some reason.*
- 

  It would be good to have both forward and backwards buttons to navigate between the examples of graphical and tabular methods.
- *Seems to have scaling issues of the images*
- 

  There is no way I can see vertex G on the map.

It is even worse on example 2



The question bank and a number of other frames as shown in the images above obscure the user name.

- *From a design/educational point of view the overall programme looks very dull and dreary, I am not sure what your objectives were at the outset but I think that the aesthetic needs to be worked on to make it suitable for a teaching tool.*

Suggestion	Comments	Analysis
Enable simply typing in date of birth	This is possible to be implemented	Validation is required to ensure that the syntax of the date of birth is correct.
Calendar obscuring the next fields	This relates to the GUI design of the project, and is easy to be implemented	When the users click on the textboxes of the obscured fields, the calendar can be send to the back so that the textboxes of the fields are shown.
Any new window sits off centre of the screen	This is not a problem as all the new windows are deliberately set to be located on the centre of the screen.	No further action needed.
Have both forward and backwards buttons in the Step-by-Step Demonstrations	This is possible to be implemented	The naïve implementation of this functionality may require temporarily storing the details of the algorithm and front-end GUI states for every step, so efficiency will be a challenge to

Suggestion	Comments	Analysis
		overcome.
Scaling issues of the images	The reason of this is most likely to be the resolution of TDH's laptop, as this has never happened on any of the other testing machines	No further action needed for this specific case. In the future if the system are to be created full screen or online, automatic scaling will be required to avoid such issues.
Aesthetic design	This relates to the GUI design of the project, and is possible to be implemented	Add aesthetic design to the pages. Need to be mindful that the aesthetic design should not be distractive.

Possible extensions

The functionalities of this system can be extended, including but not limited to the following:

1. Create an Administrator module for an administrator to manipulate (i.e., add/edit/delete) the user accounts and Question Bank data directly through the front-end, without the need of speciality in database operations for the administrator.
   This would take up to 1 month to be fully implemented. Only one more module of code (approximately 10% of the entire code) needs to be written.
2. An "Exam Board" attribute can be added to the questions, and can be used to catagorise or filter out the questions in the Question Bank.
   This would require only little programming work, and would expect to take up to 2 weeks to be fully implemented. However, population of the question data from various exam boards to the database will require a lot of time and repetitive work.
3. A Graph Generator module can be added, which can generate a graph visually based on the number of vertices and edges given (subjected to the maximum limit), or the same graph in adjacency matrix or adjacency list represenations. This module can be used in:
   - The automatic generation of visual graph to an adjacency matrix/list
   - Randomly generate a suitable graph and perform Step-by-Step Demonstration on it

   This would take approximately 3-4 months to be fully implemented, and approximately 30%-50% more code needs to be written. The implementation of this module would require complex optimisation algorithms for graphics and space arranging.
4. Currently the example graphs for the Step-by-Step Demonstrations are hard-coded, and the system should enable Step-by-Step Demonstrations on user-designed graphs (through the Sketch Board).
   This would take approximately 2-3 months to be fully implemented, and approximately 20%-40% more code needs to be written. The implementation of this functionality should set limit to the scales, features of graphs, and the numbers of vertices/edges, for the user-designed graphs.
5. Currently the workings on the Sketch Board can only be saved as a PNG file, and the system should enable the graph drawn on the Sketch Board to be dynamically saved so that users can continue working later.
   This would take approximately 3 months to be fully implemented, and approximately 30%-40% more code needs to be written. The naïve implementation of this functionality may require saving all the details (such as location) of every vertices and edges as temporary objects, and the efficiency is a challenge to overcome.
6. Currently in the Do Questions module, when the users edit the adjacency matrices/lists for the subtasks, the Edit Adjacency Matrix/List windows take up the full size of the system that obscure the question page, and users have to save their workings and close the Edit Adjacency Matrix/List windows to go back and see the questions. The system should enable the users to see the questions at the same time when they edit the adjacency matrices/lists.
   This would take approximately 3-4 months to be fully implemented, and approximately 30%-50% more code needs to be written. The implementation of this functionality would require redesigning of the Edit Adjacency Matrix/List windows for the Do Questions module.

7. New graph algorithms, such as finding the Hamiltonian cycle, can be added to this project. This would take approximately 2-3 months to be fully implemented, and approximately 25%-30% more code needs to be written.
8. A Student Tracking System can be added to the system to track individual student's progress on each topic. This includes:
    - Visualising the history of attempts of a student on each question
    - Visualising the progress of a student for each chapter
    - A League Table for the teachers to get visualised access to all their students based on their progresses on each chapter

    This would take approximately 5-6 months to be fully implemented, and approximately 60%-70% more code needs to be written. The implementation of this system would require:
    - Redesigning of the current database structures
    - Complex algorithms to quantify and calculate the progress
    - Complex dynamic GUI designing
9. The entire system can be put online:
    - The database can be updated online based on a server;
    - Users can have access to the system with their data updated at anywhere with Internet access.

    This would take approximately 8-10 months to be fully implemented, and approximately 80% to more than 100% more code needs to be written. The implementation of the online system would require at least:
    - Rewriting the code for most of the modules to make them online-supporting
    - Adding new modules for online client-server actions
    - Redesigning the GUIs to make them fit in the webpage
    - Adding server-side extensions for manipulating the data from the database
    - Solve the potential risks of concurrency issues

Client meeting log

NB People involved in the client meeting log:

- Mr John Cowley (JHC): Head of Mathematics Department of Ellesmere College;
- Mr Peter Hayes (PJH): Teacher of Mathematics of Ellesmere College, in charge of teaching Decision 1 for A-Level further mathematics students;
- Dr Sarah Shakibi (HSS): Head of Computer Science Department and Teacher of Mathematics of Ellesmere College;
- Mr Thomas Hurst (TDH): Teacher of Design and Technology of Ellesmere College.

Date	People involved	Points discussed	Actions to be taken
June 2017	JHC	Please refer to the record of interviews above	
June 2017	PJH	Please refer to the record of interviews above	
June 2017	HSS	Please refer to the record of interviews above	
13 September 2017	HSS	The Sketch Board (Module 2.2.3) is presented: - Vertex, Edge and Tag buttons are all functional. HSS would very much prefer to see the following functionality to be created as soon as possible: - Teachers/learners will be able to produce new tasks/questions by creating an adjacency matrix, an adjacency list or a graph; - The two different options of adjacent matrix and list should also appear as buttons on the interface, so that students can get equal practice with both.	Work on the remaining functionalities.

Date	People involved	Points discussed	Actions to be taken
10 October 2017	HSS	The updated Sketch Board (Module 2.2.3) and Task Setting Section (Module 2.2) are presented: - The Sketch Board: the creation/deletion/renaming of vertices, the creation of edge, and the clear button are all functional. - Remaining functionality on the Sketch Board: edge properties (directions and weights) and the save change functionalities. Currently, the active window sketch is 'saved' as hard code, and will be overwritten if another sketch follows. The envisaged destination for the submitted sketch will be the back-end database in 2 formats: adjacency matrix and PNG image. - The Task Setting Section: adding tasks is semi-functional: currently teachers can add, edit, or remove a task from the active window, but the set tasks are not stored in the back-end database (this will be fixed when the Question Bank database is completed.)	Work on the remaining functionalities. Structure of tables in the database should be designed on paper and sent to HSS as soon as possible.

Date	People involved	Points discussed	Actions to be taken
10 October 2017	HSS	HSS suggests: - Teacher should be able to see (via queries) a full list of tasks that have been created, and these should be fully editable by teacher later; - The Login window is required, with hashing of passwords; - The creation of adjacency lists/matrices can be done using Microsoft Excel and then be exported to the system; - A robust back-end database is now very much in need.	
30 October 2017	HSS	The Login window (Module 0), the Sign up window (Module 1), the design of database structure and its DDL are presented: - The Login window with hashing of passwords is now complete, but it needs testing that the data from the front end is read correctly into the back end. - The DDL for Accounts database is complete: 1) The functionality of the Foreign Keys as well as 'clean' data entry into the tables needs testing; 2) Cross-table queries and population of tables needs testing.	Work on the remaining functionalities. Ensure that all the tables have been populated with 'dummy' data, in order to show a full working demonstration for the system: - Two working student logins - Two working teacher logins

Date	People involved	Points discussed	Actions to be taken
30 October 2017	HSS	HSS suggests: - For the Accounts and Question Bank databases (mutually independent), create two disjoint subsets within the same database, rather than two separate databases. - Population of the tables is now crucially important.	
14 November 2017	HSS	The working demonstrations for student/teacher logins, sign ups and account settings, and the finalised design of the database are presented: all successful.	Work on the remaining functionalities. The Question Bank Section (Module 2.3) should be completed as soon as possible. Create a sample question which is quite basic in structure, but utilises the functionality of the tables in the database to test if it works.
6 December 2017	HSS	The finalised Task Setting Section (Module 2.2) and Question Bank Section (Module 2.3) are presented: - The edition of a graph in the form of adjacency matrix/list are both functional, with full validation. - Questions and corresponding graphs can be successfully written into or read from the back-end database, with full validation.	Progress so far: User accounts: completed This includes: - Login, Sign up, and Log out - Account settings - Communications between the front-end system and the back-end database - Validation

Date	People involved	Points discussed	Actions to be taken
6 December 2017	HSS	- Users are able to see (via queries) a full list of tasks that have been created in the Question Bank section. - Teachers are able to add a new question into the Question Bank Section, and can also edit or delete an existing question in the Question Bank Section. - Remaining functionalities of the Question Bank Section: Do Question window and marking the input answers.	Teaching section: to be completed - The graph algorithms are ready - The step-by-step demonstrations of those algorithms are not yet implemented Task setting section: completed. This includes: - Creating a graph via adjacency matrix, adjacency list, or the Sketch Board - Adding, editing, or deleting a question corresponding to the graph - Communications between the front-end system and the back-end database - Validation Question Bank: to be completed - Listing all the questions in the Question Bank: completed - Adding, editing, or deleting a question in the Question Bank: completed - Communications between the front-end system and the back-end database: completed - Validation: completed - Do Questions window and marking the input answers: not yet implemented Deadlines for all implementation to be completed: 8 January 2018

Date	People involved	Points discussed	Actions to be taken
8 January 2018	HSS	All implementation has been completed. Bugs found in the following modules: - Module 1 – Sign up: Sign up request passed even when a date of birth from the future is entered - Module 1 – Sign up: Sign up request passed even when an invalid syntaxed email address is entered - Module 1 – Sign up: Sign up request rejected when firstly sign up with a username, change it, and then change other account settings (It outputs "This username has already been taken!") - Module 2.2 – Task Setting: System crashes when the problem description is too long - Module 2.3 – Question Bank: System crashes when there is no question in the question bank and the "Do Question" button is clicked	Fix the bugs as soon as possible
19 March 2018	JHC	Completed project is presented to JHC for testing. Errors found in Module 2.1.2.2 – Krukskal's Algorithm Step-by-Step Demonstration: For example it said "BD and BD have the same weight" whereas it should have said AC and BC. There are also issues due to lack of explanation in various areas.	Fix the bugs and add explanations as soon as possible, and send the finalised project to the users.

Date	People involved	Points discussed	Actions to be taken
19 March 2018	TDH	Finalised project is presented to TDH for testing. Feedback email from TDH is received.	Analyse the suggestions made in TDH's feedback.
20 March 2018	JHC	Finalised project is presented to JHC for testing.	Wait for JHC's reply email for feedback.
21 March 2018	PJH	Finalised project is presented to PJH for testing.	Wait for PJH's reply email for feedback.
22 March 2018	JHC	Feedback email from JHC is received.	Analyse the suggestions made in JHC's feedback.
22 March 2018	PJH	Feedback email from PJH is received.	Analyse the suggestions made in PJH's feedback.

References

[1] MyMaths: Bringing maths alive, https://www.mymaths.co.uk/

[2] VisuAlgo: Visualising data structures and algorithms through animation, https://visualgo.net/en

[3] MySQL Workbench: https://www.mysql.com/products/workbench/

[4] OCR A Level Further Mathematics A (H245) Formulae Booklet, http://ocr.org.uk/Images/308765-a-level-further-mathematics-a-formulae-booklet.pdf

[5] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. *Introduction to Algorithms third edition*. MIT Press. 2009. ISBN 978-0-262-03384-8.

[6] The MD5 cryptographic hash function, Ius mentis, http://www.iusmentis.com/technology/hashfunctions/md5/

[7] MD5, Wikipedia, https://en.wikipedia.org/wiki/MD5

[8] By Matt_Crypto - original illustration for Wikipedia, created in Dia., Public Domain, https://commons.wikimedia.org/w/index.php?curid=214963