

# A-Level Decision Mathematics Tool for Graph Algorithms and the Simplex Algorithm

OCR A-Level Computer Science NEA Project

A student guide to learning, understanding, and applying the steps behind specific algorithms taught as part of the Edexcel A-Level Further Maths course.

Aditya Verma

Analysis-----	1
Introduction to the Problem-----	1
Stakeholders-----	1
Existing Systems-----	1
MyMaths ( <a href="https://www.mymaths.co.uk/">https://www.mymaths.co.uk/</a> )-----	1
GeoGebra ( <a href="https://www.geogebra.org/m/newwqrmg">https://www.geogebra.org/m/newwqrmg</a> )-----	1
Pearson ActiveLearn ( <a href="https://www.pearsonactivelearn.com/app/library">https://www.pearsonactivelearn.com/app/library</a> )-----	1
Visualizations of Graph Algorithms ( <a href="https://algorithms.discrete.ma.tum.de/">https://algorithms.discrete.ma.tum.de/</a> )---	1
Primary Research-----	1
Interviews-----	1
Edexcel A-Level Further Maths Specification-----	3
Success Criteria-----	4
Computational Methods-----	5
Stakeholder Uses/Needs-----	5
Software and Hardware Requirements-----	5
Software Requirements to Implement the System-----	5
Software Requirements to Run the System-----	6
Hardware Requirements to Implement the System-----	7
Hardware Requirements to Run the System-----	8
Design-----	9
Iteration 1: GUI Mockup-----	9
Iteration 2: Simplex Algorithm-----	15
Iteration 3: Kruskal's Algorithm-----	20

Iteration 4: Simplex Algorithm GUI Implementation-----	26
Iteration 5: Kruskal's Algorithm with Visual Graphs-----	35
Development-----	42
Iteration 1: GUI Mockup-----	42
Iteration 2: Simplex Algorithm-----	42
Iteration 3: Kruskal's Algorithm-----	42
Iteration 4: Simplex Algorithm with GUI Implementation-----	42
Iteration 5: Creating Visual Graphs in the GUI-----	43
Testing-----	44
Iteration 1: GUI Mockup-----	44
Iteration 2: Simplex Algorithm-----	46
Iteration 3: Kruskal's Algorithm-----	49
Iteration 4: Simplex Algorithm with GUI Implementation-----	52
Iteration 5: Creating Visual Graphs in the GUI-----	55
Final Testing/Evaluation-----	58
Function and Robustness-----	58
Usability Testing/Stakeholder Testing-----	60
Maintenance Review-----	62
Future Improvements-----	62
Appendix-----	62

# Analysis

## Introduction to the Problem

Decision Mathematics focuses on the use of various mathematical models and algorithmic techniques that aid decision-making in complex scenarios that often involve optimisation, resource allocation and strategic planning, for example using minimum spanning trees for network systems. However, without clear understanding of the logical steps and intricate details associated with these processes, visualising the practical applications and real-world relevance of these algorithms may be difficult, especially for students learning how to manually use them, who will need a deep understanding for their exams as a part of the Further Maths A-Level. Decision Mathematics is a module taught as part of the Edexcel Further Maths A-Level at [School Name] ([SCHOOL NAME ABBREVIATION]). While the step-by-step processes involved in certain algorithms may be generally comprehensible, a well-rounded, thorough understanding of the logical and detail behind each step of the process allows for increase problem recognition which will aid them when facing more abstract questions that require them to apply their knowledge in scenarios they have not faced when revising, for example, explaining the reasoning behind each step when applying a graph algorithm.

A major existing issue is the passive learning style offered by most guides and textbooks at school, where students eventually memorise a single method that is generally inflexible when problems are offered in a different format. A better form of teaching these algorithms would be an interactive walk-through with students able to understand the process at each stage of the algorithm in relation to the offered problem and continue at their own pace since some algorithms such as the

Simplex method consist of some parts that are often difficult to learn quickly. This issue would be best overcome using visualisations, with a system that allows users to work-through their own questions with guiding explanations.

Decision maths varies from pure maths in the way it is taught and learnt, with some students clearly able to visualise the abstract concepts presented, while others may struggle to develop this way of thinking. For those who are not able to grasp these conceptual ideas, such as the logic behind iterations of the Simplex algorithm, existing systems may not be able to answer all the questions they have that pose barriers to their understanding. For [SCHOOL NAME ABBREVIATION] students, a new system to guide them to properly understand the methodology of various algorithms would be valuable for those who may not fully learn from examples (which generally only cover the basic requirements to complete an exam question) and their other current forms of revision.

Another variation between decision maths and pure maths is the increased likelihood of small errors when answering an exam question, which will have a major impact on the final solution. An example of this is a minimum spanning tree algorithm such as Prim's algorithm. One requirement for students is to perform Prim's algorithm on a distance matrix. The steps are defined below (Pearson ActiveLearn D1 Textbook):

- **The distance matrix form of Prim's algorithm is:**

- 1 Choose any vertex to start the tree.
- 2 Delete the **row** in the matrix for the chosen vertex.
- 3 Number the **column** in the matrix for the chosen vertex.
- 4 Put a ring round the lowest undeleted entry in the numbered columns. (If there is an equal choice, choose randomly.)
- 5 The ringed entry becomes the next arc to be added to the tree.
- 6 Repeat steps **2, 3, 4** and **5** until all rows have been deleted.

With the time pressure of exams, students can often end up skimming through the

distance matrix to find the smallest values. This leaves them susceptible to incorrectly identifying the correct value to choose which may result in further problems. For example, the result of Prim's algorithm may be used in other parts of an exam question, and an incorrect initial answer can result in the student struggling to answer subsequent parts.

To allow [SCHOOL NAME ABBREVIATION] students to correctly answer decision maths questions that demand attention to detail, a new system that provides students with guidance throughout the question, as well as visual examples that emulate real exam questions would be beneficial. This system would cater to various types of learners and accommodate students with differing levels of initial understanding, helping them build on their knowledge effectively.

## Stakeholders

The predominant stakeholders of this system would be [SCHOOL NAME ABBREVIATION] students learning decision maths as a part of their Edexcel Further Maths course. Students need to understand which algorithms to use in context, the reasoning behind the steps, and how to implement them quickly as a major factor in the difficulty of the exam is the time constraint. Therefore, to prepare for exams, these students would want to improve their speed at manually working through algorithm-based questions and simultaneously develop a clear understanding of the process step-by-step.

Students are taught using PowerPoints, which involve some worked examples with teacher explanation, and questions for students to attempt.

## Formulating linear programming problems



You need to be able to formulate real life problems as linear programming problems. In the previous chapter you did this with two-variable situations.

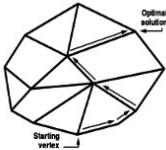
In this chapter you will consider more complicated linear programming problems.

### To formulate a linear programming problem:

- Define your decision variables
- Write down the objective
- Write down the constraints



The simplex method involves defining the system of linear inequalities as before, then 'moving' along each line, visiting vertices trying to improve your objective function on each trip.



Don't be lulled into a false sense of security – the Simplex Algorithm isn't so named because of the lack of complexity, it is named after a geometric construction known as a [simplex](#), which is a triangle generalised into n-dimensions.

[https://en.wikipedia.org/wiki/Simplex\\_algorithm](https://en.wikipedia.org/wiki/Simplex_algorithm)

## Test your understanding – Edexcel SAMS



Dale is planning a production run of three types of desk. The three types are lectern desk, roll top desk and writing desk.

In total, Dale has  $400\text{ m}^2$  of wood available; each lectern desk requires  $3\text{ m}^2$ , each roll top desk requires  $5\text{ m}^2$ , and each writing desk requires  $8\text{ m}^2$ .

In total, Dale has 350 hours available; each lectern desk requires 3 hours, each roll top desk requires 6 hours, and each writing desk requires 10 hours.

Once complete, the desks need to be stored in a warehouse. The warehouse has  $75\text{ m}^3$  of storage space available; each lectern desk requires  $1\text{ m}^3$ , each roll top desk requires  $1.5\text{ m}^3$  and each writing desk requires  $1.25\text{ m}^3$ .

The profit on each lectern desk sold is £40, the profit on each roll top desk sold is £50 and the profit on each writing desk sold is £65.

Dale wants to maximise his profit.

Let  $x$ ,  $y$  and  $z$  be the number of lectern desks, roll top desks and writing desks made respectively during the production run.

(a) Formulate this situation as a linear programming problem, giving your constraints as inequalities. (4)

(b) Complete the initial tableau in the answer book for this linear programming problem. (2)

(c) Taking the most negative number in the profit row to indicate the pivot column, perform one complete iteration of the Simplex algorithm. Give an explanation of the method by clearly stating the row operations you use. (4)

While this is an effective method at building on exam practice, it may not suit all subgroups in the overall group of students. Visual learners may benefit from seeing the algorithm performed using visualisations to build their understanding and help them answer questions, while read/write learners may learn from reading about how the algorithm works, internalising the steps, and then working through exam questions. Students are also provided with a physical copy of the Edexcel A-Level Decision Maths 1 textbook created by Pearson Edexcel (exam board); however, this once again results in a passive learning style. Visual learners are more likely to retain knowledge of a specific topic when they are provided with

interactivity options and visualisations that allow them to better picture an algorithm for example.

A major issue faced by students is that they do not have access to constant teacher explanation of questions when revising for exams. This means that although they may be able to follow the steps to answer a question, their overall understanding prior to the exam may suffer, especially when the questions are presented in a different format to what has been practiced. Another common issue faced by students is the limited time they have during the exam, posing trouble to those who have less understanding of the algorithm and may not be able to visualise the process.

Decision Mathematics predominantly consists of Advanced Subsidiary (AS-level) content and for this reason it is taught during Year 12 at [SCHOOL NAME ABBREVIATION]. This also creates further problems as students are often expected to keep up with previously taught content throughout Year 13, without frequent, in-depth lessons to remind students of topics they may have forgotten. This can prove to be problematic, with students having to cover the new content taught to them whilst keeping up with decision maths simultaneously, impacting overall grades, especially if the initial understanding of key algorithms was not to a firm level.

The A-Level Decision Mathematics exam also requires students to be able to use various algorithms effectively often relying on students knowing what process to use as shown by the past exam question from Edexcel A-Level D1 June 2020 paper:

The weights on the arcs in Figure 4 represent distances. The weight on arc EF is  $x$  where  $12 < x < 26$  and the weight on arc DG is  $y$  where  $0 < y < 10$

An inspection route of minimum length that traverses each arc at least once is found. The inspection route starts and finishes at A and has a length of 409

It is also given that the length of the shortest route from F to G via A is 140

(b) Using appropriate algorithms, find the value of  $x$  and the value of  $y$ .

(9)

However, existing systems such as online textbooks and teaching websites like MyMaths tend to initially teach the topic by focusing on examples that do not extend into other areas of the subject. An improved system would teach the foundational concepts of each topic instead of immediately applying them to practical examples, as well as explaining any key details that existing systems may miss out. Students would benefit from this as they will be less likely to associate a topic with a singular type of question and instead be able to apply their general understanding to a problem presented to them, whilst also developing a more rounded grasp of the subject. Another factor to consider is the differing levels of initial understanding that various stakeholders will have. For those who have a weaker understanding of the algorithms and tend to take longer when answering exam questions, it will be vital to ensure that my system can provide them with a base level of understanding upon which they can develop. By considering a range of initial skill levels in decision maths, I can ensure that my solution is suitable for all stakeholders and isn't limited to those who already have a sound initial understanding of the topic.

Due to the different ways that algorithms are presented in exam context, an improved system would ensure that students are not fixed to solving problems only when they are directly presented and instead can use their general understanding of a topic when the problem may not be clear.

To ensure that the system can be used by those with differing initial levels of understanding, the fixed examples will be a key factor to allow them to see how to approach an exam question. Adding visual elements will also increase the suitability of this system to those with different learning styles, extending past just those who are able to learn by reading the steps and following examples. Overall, the primary stakeholders would benefit from a system that suits various types of learners, making it more versatile and suitable for students to effectively revise. This would be best achieved through a combination of different features, such as visual elements, interactivity, and fixed examples to learn from.

## Existing Systems

MyMaths (<https://www.mymaths.co.uk/>)

One of the main existing systems is MyMaths, which covers many areas/levels of mathematics but also contains lessons on decision maths. By selecting the 'Lesson' option of a specific topic, this system offers a step-by-step explanation of the chosen topic.

### Pros:

A major benefit of MyMaths is the introduction to the topic and explaining how an algorithm works in clear steps. This will be included in my system to provide students with a 'reference point' where they can look back to see all steps of an algorithm. This system also contains examples that apply individual algorithms to reach a final solution, another beneficial feature for students.

Kruskal's algorithm      1 Introducing the minimum spanning tree

1  
2  
3 Kruskal's algorithm gives us a method for solving this problem by finding a minimum spanning tree.  
4  
5  
6  
7

1  
2  
3  
4  
5  
6  
7

Next      Oxford University Press 2024      Menu      Next

Another benefit of the MyMaths system is the intuitive method of splitting the topic up into pages, first introducing, then explaining and finally showing examples. This allows users to return to different pages if they want to revisit the

steps of the algorithm or check a previous example. MyMaths also provides users with explanations which are also user friendly, not overcomplicating their descriptions, as shown below:

The screenshot shows a digital worksheet for 'Kruskal's algorithm' under the 'Greediness and complexity' section. The interface includes a menu bar at the top with 'Kruskal's algorithm' and 'Greediness and complexity'. Below the menu, there are seven numbered steps:

- 1 What is the complexity of the algorithm?
- 2 In a complete graph with  $n$  vertices the number of edges is  $\frac{n(n - 1)}{2}$
- 3 To find the shortest edge the maximum possible number of comparisons required will be  $\frac{n(n - 1)}{2} - 1$
- 4 To find the next shortest edge from those remaining the maximum possible number of comparisons required will be  $\frac{n(n - 1)}{2} - 2$
- 5 Continuing in this way until  $(n - 1)$  edges have been chosen the final number of comparisons will be  $\frac{n(n - 1)}{2} - (n - 1)$
- 6 The total number of comparisons involved in applying Kruskal's algorithm in the worst-case scenario is  $\frac{n(n - 1)}{2} - 1 + \frac{n(n - 1)}{2} - 2 + \dots + \frac{n(n - 1)}{2} - (n - 1)$
- 7

A cartoon character icon is positioned above the notes section, which contains numbered circles 1 through 7. At the bottom of the page are navigation buttons: 'Next', 'Oxford University Press 2024', 'Menu' (highlighted in dark blue), and another 'Next' button.

With the working separated from the explanation, users can easily follow each of them individually and note down what they deem necessary. Instead of a page setup, my system will instead present users with a menu bar that allows them to select the various algorithms, as well as space for them to create graphs or enter their own Simplex tableaus to be solved.

One addition to this system that students may find helpful is a summary section at the end of the walkthrough like those included in the textbook. This would be a brief synopsis of the content students are required to know.

### Cons:

One drawback of MyMaths is the cramped layout with a minimal amount of content per page. The slider function creates the effect of progression throughout the process; however, it often results in information having to be removed from the page to allow new information to appear, making it difficult for students as they will develop a weaker understanding of content split up over various pages since they are unable to picture the steps in a cohesive manner. This is shown below:

Kruskal's algorithm

Alternative solutions

Notes

Some of the edges in our last example had the same length.

Would it have made any difference to our solution if we had listed these the other way round?

Let's switch round these entries in our table and see what happens.

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

The edges are listed in order of weight (Step 1) so we can go on to Step 2.

An alternative solution is:

Step 4: Repeat Step 3 until all the vertices have been included.

We have included all the vertices in our minimum spanning tree so Step 4 tells us we can stop.

with a total weight of:

$3 + 6 + 8 + 9 + 12 + 14 = 52 \text{ km}$

[Next](#)

Oxford University Press 2024

[Menu](#)[Next](#)

Oxford University Press 2024

[Menu](#)[Next](#)

A further drawback is the fixed examples that MyMaths uses, as this is not accompanied with a tool for those who struggle to learn from following examples. My system should allow users to enter their own graphs or Simplex tableaux to be solved and walk them through the steps involved in solving the question, as well as having fixed examples used for teaching.

### Essential Features:

My system must include an introduction of the general topic as is present in MyMaths, which would also explain the uses and purpose of the specific algorithm, with a brief description of how it works as this can allow students to think about applying the algorithm to different contexts they may face in the exam. It must also include example questions with accompanying explanation of what is happening at each step of the algorithm, so students are able to learn by following a full exam question. These questions must imitate exam-style questions to allow for an accurate representation of what students may be asked in an exam. Both of these features are well-made by MyMaths. Another key feature of my system will be clearly outlined working for the fixed examples, which can show students which parts of the method they must display to earn marks in an exam. A further aspect to consider outside of the system's content is the layout. As mentioned in the 'cons' section, MyMaths has a cramped layout with minimal information displayed on a single page and the user having to scroll to access new information. My system will have a clear, well laid out user interface that ensures information is displayed with clarity to ensure it is not missed by my stakeholders.

## [GeoGebra \(<https://www.geogebra.org/m/newwqrmg>\)](https://www.geogebra.org/m/newwqrmg)

Another commonly used interactive system is GeoGebra, which (similarly to MyMaths) contains a wide variety of maths content, but also has specific content created by users related to Decision Maths.

### **Pros:**

One advantage of this system is the feature that allows users to generate random values for a graph and helps them through the algorithm. GeoGebra is generally focused on the user working through the question, with guidance from the system, whereas MyMaths teaches the user by example, then providing them a couple of questions to attempt. Students may benefit from a combination of these teaching methods, with set examples to learn, then an option for users to create their own graphs for instance.

Another benefit of GeoGebra is the method of explaining the steps of the algorithm as the user progresses through the worked example. This allows them to understand what they need to look for at each step of the algorithm without providing an overwhelming amount of information that may be difficult to follow.

### **Cons:**

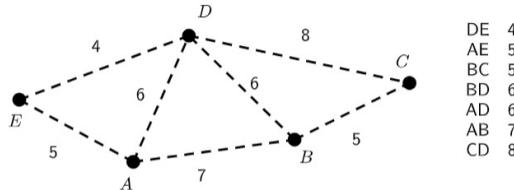
A disadvantage of the decision maths content on GeoGebra is the lack of surrounding information outside just an algorithm's process. This part of the system primarily focuses on users practicing manually working through a question but does not provide much theory/clarification relating to a topic, resulting in a weak overall understanding of the algorithm.

Looking at decision maths content from other users, GeoGebra seems to follow a similar pattern of visualisation over explanation. While this may be beneficial for those trying to image what is happening, the lack of description to accompany the visuals is unlikely to result in a full understanding of the topic, as students may be unable to articulate what they learn from the visuals without annotations to guide them.

Below is a screenshot from GeoGebra:

Select the arc of least weight to start the tree.

Then select the next arc of least weight that does not form a cycle.



DE	4
AE	5
BC	5
BD	6
AD	6
AB	7
CD	8

[Reset to example on p. 53](#)

[Generate random values](#)

[Reset this network](#)

Minimum spanning tree :  
Lengths :



This illustrates the focus placed on visual demonstration, prompting users to simply work through the algorithm. While this may form useful exam practice, such as from practicing on many randomly generated graphs, it is not as practical as other systems such as MyMaths in building foundational knowledge of the topic.

Another drawback of this system is the lack of introduction to the topic, instead opting for the user attempting an example immediately. Since exam questions are often presented in context, using examples with no contextual element may not properly develop exam technique for students.

### **Essential Features:**

My system must include visualisations as well as complementary annotations to benefit visual learners who may not be able to develop their understanding by following a fixed example. This section of GeoGebra only prioritises visualisation, which can result in students not having full understanding of the steps they are performing, hence explanations accompanying the visual element will be essential to better explain the topic to those who may have a weak initial understanding. Building on GeoGebra, my system will introduce the topic instead of presenting questions straight away, once again, allowing students to develop understanding

in the context of what may be asked in their exam. A key feature of GeoGebra is the ability to generate random graphs to perform algorithms on. This allows for a wide range of practice material for students to build their speed and effectiveness in answering exam questions. I will include a similar feature, randomly generating different graphs and providing fully worked solutions, accompanied by explanations for each step.

### **Pearson ActiveLearn (<https://www.pearsonactivelearn.com/app/library>)**

Pearson ActiveLearn is the main resource used by schools to help students with maths related content. This system contains textbooks that conform to specification content from Pearson Edexcel, providing questions and examples that go into detail on every step of specific algorithms. These textbooks are replicas of the physical textbook provided to students. ActiveLearn also provides hints and has links for students to further their knowledge using GeoGebra to visualise algorithms. While this system does not have its own visualisations, these links can be useful for those who may not just understand the explanation from the textbook. (When referring to the textbook throughout the analysis, this could refer either to this system or the physical copy, hence any drawbacks mentioned regarding the physical copy will also apply to this existing system)

#### **Pros:**

This system has a clear introduction to the topic, such as what specific algorithms can be used for and what students will be expected to complete in an exam. Another key benefit of this system is the chapter summary in each textbook, with an example shown below:

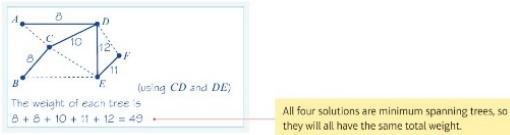
#### Summary of key points

- 1 To formulate a **linear programming** problem:
  - define your decision variables
  - write down the objective function
  - write down the constraints.
- 2 Inequalities can be transformed into **equations** using **slack variables** (so called because they represent the amount of slack between an actual quantity and the maximum possible value of that quantity).
- 3 The **simplex method** allows you to:
  - determine if a particular vertex, on the edge of the feasible region, is optimal
  - decide which adjacent vertex you should move to in order to increase the value of the objective function.
- 4 **Slack variables** are essential when using the simplex algorithm.
- 5 The simplex method always starts from a basic feasible solution, at the origin, and then progresses with each iteration to an adjacent point within the feasible region until the optimal solution is found.
- 6 To use a **simplex tableau** to solve a maximising linear programming problem, where the constraints are given as equalities:
  - Draw the tableau: you need a basic variable column on the left, one column for each variable (including the slack variables) and a value column. You need one row for each constraint and the bottom row for the objective function.
  - Create the initial tableau: enter the coefficients of the variables in the appropriate column and row.
  - Look along the objective row for the most negative entry: this indicates the pivot column.
  - Calculate  $\theta$ , for each of the constraint rows, where
$$\theta = \frac{\text{the term in the value column}}{\text{the term in the pivot column}}$$
  - Select the row with the smallest, positive  $\theta$  value to become the pivot row.
  - The element in the pivot row and pivot column is the pivot.
  - Divide all of the elements in the pivot row by the pivot, and change the basic variable at the start of the row to the variable at the top of the pivot column.
  - Use the pivot row to eliminate the pivot's variable from the other rows: this means that the pivot column now contains one 1 and zeros.
  - Repeat bullets 3 to 8 until there are no negative numbers in the objective row.
  - The tableau is now optimal and the non-zero values can be read off using the basic variable column and value column.

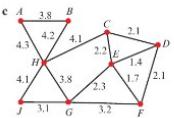
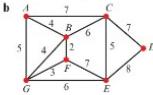
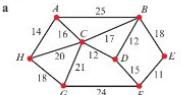
This ensures that students are fully aware of the basic content they are required to know for the exam and can refer to this summary for definitions, key points, and general explanations. In terms of practice, the textbook provides the most material out of the mentioned existing systems, with multiple exercises and summary questions frequently provided (with answers). These questions are often taken exactly from past exam papers, forming high quality revision material for students. The content is also directly taken from the exam board Edexcel; hence it is accurate for students and does not contain any unnecessary information.

#### Cons:

The main drawback of this system is that it does not allow users to input their own questions to be answered and be given an explanation step-by-step. This means that the main form of revision from this system is following the fixed examples and completing past exam questions provided which may not be suitable for those who learn more by interacting with a system and seeing an algorithm in action. A further drawback is the limited number of questions for specific topic. As mentioned before, this system provides some of the most practice material for students, however, in specific sections, students would benefit from more:

**Exercise 3A**

- 1** Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.



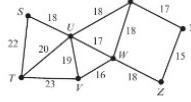
- (EP) 2 a** State what is meant by:

- i a tree  
ii a minimum spanning tree.

(1 mark)  
(1 mark)

- b Use Kruskal's algorithm to find a minimum spanning tree for this network.

(3 marks)



- c Draw the minimum spanning tree found in part b.

(1 mark)

- d State, giving a reason, whether this minimum spanning tree is unique.

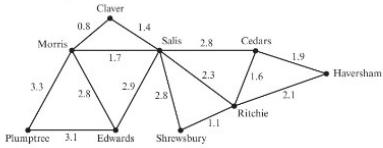
(1 mark)

56

- (P) 3** Draw a network in which:

- a the three shortest edges form part of the minimum connector (MST)  
b not all of the three shortest edges form part of the minimum connector.

- (E) 4** The diagram shows nine estates and the distances between them in kilometres. A cable TV company plans to link up the estates.



- a Find a minimum spanning tree for the network using Kruskal's algorithm. List the arcs in the order that they were added to the tree. **(4 marks)**

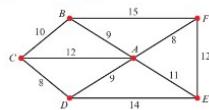
- b Use your answer to part a to find the minimum length of cable required to link all the estates together. **(1 mark)**

**3.2 Prim's algorithm**

- **Prim's algorithm can be used to find a minimum spanning tree:**

- 1 Choose any vertex to start the tree.
- 2 • Select an arc of least weight that joins a vertex already in the tree to a vertex not yet in the tree.
  - If there is a choice of arcs of equal weight, choose any of them.
- 3 Repeat step 2 until all the vertices are connected.

**Note** Prim's algorithm considers **vertices**, whereas Kruskal's algorithm considers **edges**.

**Example 3**

Use Prim's algorithm to find a minimum spanning tree for the network above. List the arcs in the order in which you add them to your tree.

**Online** Explore Prim's algorithm using GeoGebra.

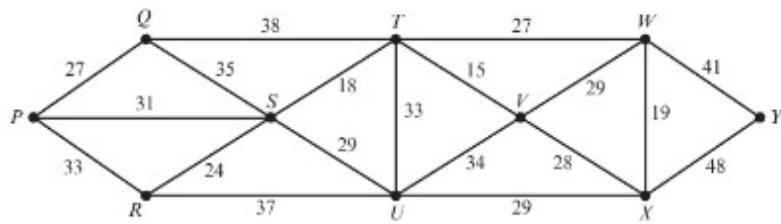


**Watch out** In your exam, you may be asked to use Prim's or Kruskal's algorithm. You must know which is which.

57

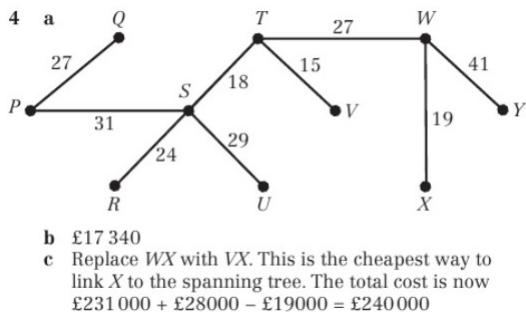
Another drawback of this system is the limited mark scheme, which generally only displays the correct answer to a question, whereas in a decision maths exam, most marks are allocated for the student showing their method. Not being able to see the exact method required by a specific question can result in students losing marks for not showing full method to reach their answer. An example question and mark scheme from the textbook are shown below:

**(E/P) 4**



The network shows ten villages and the costs, in thousands of pounds, of connecting them with a new energy supply.

- Use Prim's algorithm, starting at  $P$ , to find the energy supply network that would connect all ten villages for minimum cost. **(3 marks)**
- Draw your minimum connector and state its cost. **(2 marks)**
- Unforeseen problems with the link between villages  $W$  and  $X$  mean that the cost of connecting them rises to £34 000. Explain how this affects your minimum spanning tree. **(2 marks)**



The (E/P) indicates this is a past exam question, with the number of marks available for each part displayed too. However, the mark scheme does not explain how they have reached the answer, affecting student's understanding of the topic.

### Essential Features:

My system will utilise past exam questions as the fixed examples for students to learn the functionality of an algorithm from, exposing them to the implementation of an algorithm in an exam. The clear, detailed explanations of different algorithms will also be vital in ensuring a successful system that allows students to apply their knowledge.

An additional feature that my system will have is the ability for users to enter their own graphs to perform algorithms on as well as enter their own adjusted inequalities into an initial Simplex tableau, and provide an explanation allowing the student to understand how to solve the question, adding an element of

interactivity which will especially benefit visual learners. My system must also ensure that the content for specific topics is tailored to Edexcel A-Level Further Maths students, like the textbook, ensuring the content remains relevant for my stakeholders. For all sections of my system that present questions to the user, full working and explanations will be provided to ensure the user can learn and apply their knowledge in the future. Furthermore, my system will have a brief introduction to each topic, similar to the chapter summary mentioned in the 'pros' section.

## Visualizations of Graph Algorithms

(<https://algorithms.discrete.ma.tum.de/>)

This system is an example of a graph algorithm visualiser, presenting users with the option to learn about various graph algorithms to create minimum spanning trees, find shortest paths and various other applications. Of the options presented, the minimum spanning tree and shortest path visualisations would be the most beneficial to a decision maths student at [SCHOOL NAME ABBREVIATION].

### Pros:

A beneficial feature of this system is the brief but informative introduction to a specific topic, for example the description of spanning trees and where they are used in real-life applications, as shown below:

## Minimum Spanning Trees

A spanning tree is a set of edges that connect all nodes of a graph but does not have any superfluous edges. The problem of finding a spanning tree (usually of minimum cost) is common in communication networks where it is important that every node can communicate with all other nodes (not necessarily directly). It also occurs in distribution networks (e.g., supply houses of a block with water through a connected network of pipes, electricity networks).



**Kruskal's Algorithm**

Kruskal's Algorithm computes a minimal spanning tree on an undirected graph. Furthermore, it returns a minimal spanning forest if the graph was unconnected.

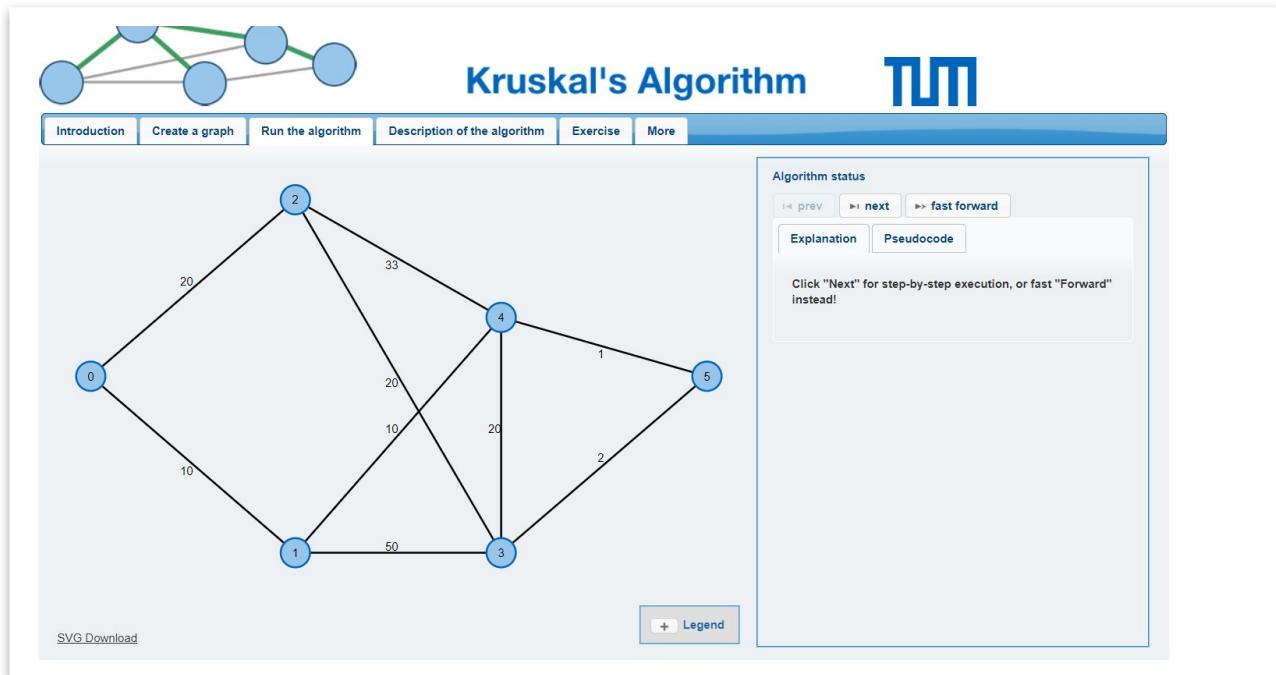


**Prim's Algorithm**

Prim's Algorithm computes a minimal spanning tree on an undirected graph.

There is a clear option for users to select from two minimum spanning tree algorithms, each with a brief description of what they do.

Upon selecting from one of the algorithms, the user is presented with a problem-solving question which solidifies user understanding of practical applications and presents the topic in relevant context. There is a menu bar with informative labels that allows users to perform different actions, ensuring that there is not an abundance of content packed onto a single page. Users are also able to select whether they want a more detailed description of the algorithm instead of presenting it initially, once again ensuring that the page is not overwhelming.



The picture above displays the menu bar, selected on 'Run the algorithm,' displaying a clear and simple window with a graph. There is a box that allows users to walk through the algorithm, explaining what each step is doing and adapting the graph simultaneously.

Presenting information in a structured manner with an intuitive layout that provides a detailed guide to the algorithm without overwhelming the user makes this system successful for developing a well-rounded understanding of specific topics.

### Cons:

The only major drawback of this system is that it is not tailored to Edexcel A-Level Further Maths students and is more general for those looking to learn discrete maths and computer science. This means that while students can learn how to perform the algorithm and what is happening at each step, they may not be able to understand what they would need to write/perform to achieve marks for an exam question. This also means that it does not contain a large proportion of information that students would be required to know, hence cannot be relied on as a sole method of revision.

### Essential Features:

This system clearly outlines the topic in context, a feature that will be necessary in my system to develop a well-rounded and contextual understanding for students. The system allows users to create their own graphs, which will be included as it is vital in ensuring that students can benefit from an element of interactivity. While this system presents users with pseudocode for each algorithm, I will not include this in my system as it is not relevant content as part of the Edexcel A-Level for [SCHOOL NAME ABBREVIATION] students using the system to revise for their decision maths exam. Another feature that I will include is the step-by-step explanation that clearly describes what each step of the algorithm is doing and informs the user when a run of the algorithm has been completed.

### **Summary of Existing Systems:**

Having researched various existing systems that provide similar functionality to the system I aim to create; I can now summarise the main benefits and drawbacks of these systems. All the systems I researched propose major benefits, however, only the 'Visualizations of Graph Algorithms (VoGA)' system seems to be well-rounded with explanation, visualisation, and examples. The only drawback I noticed with this system is the fact that it is not tailored to Edexcel A-Level Further Maths students. The most relevant system for [SCHOOL NAME ABBREVIATION] students was Pearson ActiveLearn, with textbooks created based on official Edexcel A-Level Further Maths specification content, however, the textbook lacked clear visual demonstration and did not have any interactivity for students to input their own questions. From these systems, I will utilise the effective visualisation and explanation from 'VoGA,' and tailor it more towards [SCHOOL NAME ABBREVIATION] students, using examples similar to those from the textbook. The 'VoGA' system also contains information regarding other graph algorithms which would not be relevant to [SCHOOL NAME ABBREVIATION] students, forming a limitation of my system. The main feature from this system that I will include is the detailed guide on Minimum Spanning Tree algorithms (Prim's and Kruskal's).

Both MyMaths and GeoGebra also presented high quality visualisations of various algorithms, with GeoGebra using this as their main method of teaching the topic.

However, the lack of explanation fails to make it a well-rounded system as it is difficult to develop initial understanding from just a visual representation. MyMaths combines visual representation with beneficial annotations that closely follow the Edexcel A-Level Further Maths specification. While the content is satisfactory for a student to develop their understanding, the presentation of this content, using scrollable pages is less effective as minimal information is displayed per page. Splitting up a topic is beneficial for students, however, the presentation of MyMaths with limited information per page affects students using it to learn a topic. MyMaths also has a small number of examples, which is understandable, given that the system covers such a broad range of content, however, when reviewed as a system for [SCHOOL NAME ABBREVIATION] Further Maths students, the minimal examples may not be enough to develop a full understanding of the topic. By splitting up the topic into scrollable sections, MyMaths often has to remove part of the content from a page to make room for the following steps. Students can scroll through to see this content that is removed; however, this makes it more difficult to revise, having to constantly flip between sections. My system will intuitively split up a topic into sections, each of which will have adequate room to display the necessary information for each topic to be considered well-rounded.

## Primary Research

To create a functional system that suits the need of [SCHOOL NAME ABBREVIATION] students, further research that highlights the necessary features is required. I will conduct individual interviews with current further maths students to identify features they deem essential for the system to be effective. By interviewing current students, I will be able to identify active dilemmas faced in learning decision maths, as well as the specific features these potential users of the system would benefit from. Additionally, I will reference the current Edexcel Decision Maths specification which will outline each step that students are required to know for their exam. Combining the specification's criteria alongside student input for what they would benefit from will allow me to clearly outline the features that potential stakeholders will deem necessary in a system that may form part of their revision.

## Interviews

Having conducted an interview with 3 further maths students, the key features can be extracted from the answers, indicated in the 'Key points' section below.

Question (and why the question was asked)	Student 1 Response	Student 2 Response	Student 3 Response	Key points
<p>1. What decision maths topics do you find the most challenging and what makes them challenging.</p> <p>This question was asked to identify the key topics I will need to include in my system, based on what students currently find difficult and lose marks on.</p>	<p>Simplex is the most challenging topic we have covered due to the immense attention to detail required. Learning the steps for the different types of the Simplex Algorithm are not challenging, but understanding what the algorithm does is what I</p>	<p>For me, the Simplex algorithm is the most challenging since you need a very clear understanding of its functionality to be able to answer the questions. It is also the most technical topic, with room for error at many different stages. The travelling salesman problem is also quite a difficult, broad topic with many steps involved. I</p>	<p>I find the Simplex algorithm to be the most challenging decision maths topic. The main issue for me is understanding the stages of the algorithm, however, I am fairly confident with the different forms of Simplex like Two-Stage and Big-M. Other topics are much easier to check your working for errors, however, it</p>	<p>All 3 Students find the Simplex algorithm to be the most difficult topic in decision maths, often because of small mistakes and lack of understanding rather than application of the different methods. This suggests that my system will need an effective Simplex tool to guide students and provides a limitation for my system, which will not include the different types of the Simplex algorithm. The second most difficult topic suggested by 2 of</p>

	<p>struggle with. Aside from Simplex, I also struggle with the applications of graph algorithms in different topics, such as Kruskal's and Prim's as part of the Travelling Salesman topic.</p>	<p>often lose many marks in these questions as the answers overlap between parts of the question.</p>	<p>is much more difficult to spot errors in Simplex questions, and once you notice a mistake, it takes a considerable amount of time to correct. This is because an error with one iteration of the algorithm means you must begin the entire table from the beginning again. I also sometimes lose marks on Kruskal's and Prim's algorithm due to the time pressure resulting in me rushing and not spotting the</p>	<p>the students interviewed was the Travelling Salesman Problem. In Edexcel A-Level Decision Maths, these questions require knowledge of Prim's and Kruskal's algorithms. This also aligns with Student 3's issue of rushing with Prim's and Kruskal's, hence my system will include guidance for students to gain confidence applying them. By improving student confidence, they will be quicker at answering questions and are less likely to lose marks due to time pressure.</p>
--	---	---	---	---

			smallest weight edges.	
2. Do you find timing to be an issue in Decision Maths exams compared to other Maths exams?  This question allows me to identify if students are rushing during their exams due to time pressure, resulting in more mistakes. If this is an issue, I can help improve understanding of decision maths topics so they can complete questions faster and more accurately.	Timing is the biggest challenge with decision maths. Given enough time, most students can score very good marks. What separates all the candidates is how effectively we can manage our very limited time.	I find timing to be the biggest constraint to achieving high marks. Managing time during the decision maths exam is an issue for me as I often find myself spending a lot of time on the more technical questions that aren't explicitly covered in class, such as those that test what each specific part of an algorithm is doing.	While I am generally able to finish decision maths papers within the given time limit, I often find myself rushing, which in turn results in me losing marks. This mainly occurs because I spend less time on questions than I would need to gain full marks.	Student 1 and Student 2 find timing to be the main issue, while Student 3 mentions that it isn't directly an issue but can indirectly affect his answers due to him rushing. This suggests that all 3 would benefit from being able to complete questions faster, which is best achieved through a more confident understanding of the topic in question. By including detailed explanations, students will have more knowledge surrounding the algorithm, and will have to spend less time trying to figure out what more obscure questions are asking.
3. What learning resources do	To revise for my recent	Throughout Year 12, I have utilised	While we have been provided	The answers suggest that past paper questions are

<p>you currently use for Decision Maths?</p> <p>This will allow me to outline the most common resources used by [SCHOOL NAME ABBREVIATION] students and why they are used.</p> <p>Identifying the current resources used will help me ensure that any beneficial features from these resources are included in my tool.</p>	<p>exam, I primarily used the textbook to teach myself the content and methods. After I was confident with my understanding of each topic, I did past paper questions included in the textbook which I found to be the most effective way of revision.</p>	<p>the textbook to develop an initial understanding of the topic, working through some of the examples shown. For exam practice, I complete past exam questions as they are the most accurate representation of what will be asked during exams. With more difficult topics, I sometimes look back at lesson PowerPoints, however they are less effective without teacher explanation.</p>	<p>with textbooks, I only use them to remind myself of certain subtopics I may have forgotten. My revision is based around working through as many past paper questions as I can because they are the most effective method of revising for me, as being able to quickly implement the different algorithms is the key skill for exams.</p>	<p>the most efficient revision resource for students to build confidence for their exam, accompanied by the textbook to build initial understanding.</p> <p>When using PowerPoints, Student 2 seems to find them less helpful without a teacher's explanation. The students follow the same pattern of understanding the topic, then solidifying their knowledge with practical application of past paper questions.</p>
<p>4. How effective do you find the current</p>	<p>The textbook does cover all the</p>	<p>Since the textbook is from Edexcel, it contains</p>	<p>The textbook covers the specification</p>	<p>The main resource used by all 3 students for learning topics as</p>

<p><b>resources you use in developing your understanding of Decision maths?</b></p>	<p>topics the specification requires however I believe that it does lack a solid explanation , and therefore is not an effective resource. Though sometimes there is a short step-by-step guide this doesn't help very much as being able to apply it to other questions is the important part, just following an example is insufficient, however. The textbook includes many questions</p>	<p>content from the whole specification, however, in the exam there are often questions that require critical thinking that aren't covered by the textbook.</p>	<p>Completing past exam questions is useful for topics like Prim's, Kruskal's and Dijkstra's as they are easier from repetitive practice, however exam questions aren't effective at developing overall understanding of decision maths. The main benefit of exam questions is</p>	<p>content well with some fixed example questions mimic exam questions. These examples have good in-depth explanations, however, there are only 1 or 2 examples per subtopic, which means some types of question are left unexplained . After completing past paper questions, I am more confident in the exam questions, however I still spend a</p>
				<p>mentioned above lacks clear explanation, leading to it being ineffective in various circumstances. Student 1 also finds that just following a fixed example with a short step-by-step guide is insufficient in developing understanding of a topic, so I will make sure that any fixed examples are accompanied with options for students to develop a better understanding of the topic, using a random graph generator for the graph-based algorithms, as well as student interactivity features that allow users to enter their own graphs and Simplex questions. All three students also suggested that exam questions were useful for practice, but not so much for understanding.</p>

	for most of the other topics, including the Simplex algorithm, however, the 'Algorithms on Graphs' topic doesn't have many practice questions; hence I often have to look through quite a few past papers to find more.	exposure to the more abstract questions that can be asked.	significant amount of time answering the vaguer questions that test understanding. Sometimes when I use the book for past exam questions, I find that certain topics do not have many questions to practice, such as Kruskal's algorithm, so past papers are my main source of questions.	These responses suggest that one resource is missing the content of another resource and vice versa. Combining the beneficial features of both textbook examples and practice exam questions would be the best way at develop an all-rounded understanding.
5. "Understanding the purpose of each step in	I strongly agree, I believe if I had a better	I strongly agree with the statement. Often, we are	I strongly agree with this. While I can quickly answer	This necessitates the need for logical explanations accompanying the demonstration of a

<p>an algorithm would help me apply my knowledge to more obscure questions" To what extent do you agree?</p> <p>This question will outline whether students would benefit from improved understanding of individual steps of an algorithm during their exam. If so, I can highlight explanation/descriptions as an essential feature of my system.</p>	<p>understanding of the details of the various algorithms, such as graph algorithms, I could significantly boost my marks and therefore grades, as I would be more confident in answering the obscure questions that follow the basic "perform this algorithm on this data" style questions.</p>	<p>expected to use our basic knowledge from the specification and apply it to more obscure questions for example, those testing what the iterations of the Simplex algorithm are actually doing. These questions are often worth a lot of marks and will impact my grade if I am unable to extend my understanding beyond the specification to help me with these obscure questions.</p>	<p>common exam questions which test our ability to implement an algorithm, I often struggle with the questions that test understanding of the algorithm, especially Simplex. Being able to conceptualise each step of an algorithm would definitely help me in exams.</p>	<p>question, as the students find that they lose marks on questions that test overall understanding of the algorithm instead of the basic questions that ask them to perform a specific algorithm. Student 1 suggested he would benefit from better explanation on graph algorithms, as well as Student 2 and Student 3, who identified that they would benefit from better explanations on the Simplex algorithm. This suggests that all 3 algorithms I aim to implement will require solid explanation since these are the topics that students lose marks on when unable to understand their functionality.</p>
<p>6. "When revising, understanding the purpose of each step</p>	<p>I strongly agree, as without a solid understand</p>	<p>I strongly agree with this. Being able to understand</p>	<p>I agree with the statement. Most exam questions I</p>	<p>This further shows how all 3 students often lose marks in more complex questions that test</p>

<p>in an algorithm, (e.g. understanding what each stage of simplex is actually doing) is important to me." To what extent do you agree?</p> <p>This question follows on from the previous question and allows me to identify if explanations to aid understanding would be beneficial in a revision tool for students as well as if their current resources are able to help them with understanding each process in an algorithm.</p>	<p>ing of individual steps and purposes, I cannot ensure I answer each question with confidence. This in turn loses me marks.</p>	<p>the purpose of each step in an algorithm when revising would allow me to develop the critical thinking demanded by some of the more difficult exam questions. This would allow me to gain a few extra marks that many people often lose.</p>	<p>complete for revision follow the basic structure of performing a specific algorithm on a set of data given in a context. While I can answer these without requiring much understanding, the few conceptual questions are where I struggle. My revision could be improved with better understanding.</p>	<p>knowledge that is not provided by existing systems. This suggests that my system needs to present users with more information surrounding a topic/algorithm to provide them with the knowledge they need to answer the tougher questions in the exam which are often the deciding factor between grades.</p>
7. Do you think an	I think it would be	I would find an interactive	I would benefit from	This suggests that all 3 students will

<p>interactive and visual learning tool would be beneficial for understanding decision maths algorithms? If so, please explain how.</p> <p>By asking [SCHOOL NAME ABBREVIATION] students how an interactive and visual tool would help them, I can further outline the necessary features I will include to ensure effectiveness of my system, based on what students consider to be helpful.</p>	<p>extremely helpful as then understanding the algorithms would be a lot easier. Using the textbook, I cannot picture what each algorithm is doing, but if there was a tool to demonstrate the algorithm thoroughly, for example, explaining the steps of the Simplex algorithm, I could perform much better.</p>	<p>tool highly valuable as it would present knowledge in a more comprehensible manner. Being able to physically work through an example step-by-step would help refine my understanding with more intricate details. Past exam questions do not provide an explanation, and using the textbook to try and understand a topic can often be difficult as it is harder to visualise an algorithm when all the steps are presented at once.</p>	<p>an interactive and visual tool, as it would help round off my understanding of a topic, for example, entering my own exam questions to be solved. Using a tool that illustrates an algorithm in depth, I would be able to access the knowledge for the conceptual questions I struggle with, as mentioned in previous questions. This would reduce the time I need to spend on each question</p>	<p>benefit from visual representations of algorithms, as it will also help to develop better understanding of a topic, alongside detailed annotations at each step of the algorithm. My system will require a visual aspect and a descriptive aspect to develop a strong understanding of a topic for students. With all 3 students agreeing on the benefits of a possible interactive and visual tool, I must ensure that my system combines both visualisation and interactivity. The visualisation element would help Student 1 and Student 3, who suggested they struggle picturing the algorithms and</p>
---	---	---	---	--

			during an exam, providing me with time to check my work and gain marks I would otherwise lose.	would value a tool that can illustrate an algorithm in depth. The interactive element is necessitated by Student 2, who suggested that physically working through a question would help him refine his understanding.
8. What challenges do you face with Decision maths? Select from the options below: - Understanding what you are doing at each step of the algorithm - Learning the steps of an algorithm	Learning the steps of the algorithm isn't the hard bit, but applying and understanding what each algorithm does is the difficulty. Many questions in the exams are designed to test whether we understand what each	I find learning and remembering the steps of an algorithm easy, as it is usually just a single process that can be applied to different sets of data presented in different contexts. I can gain marks for answering exam questions using an algorithm, but I	My main struggle is with understanding the details behind each step of an algorithm, as learning and remembering are not very time consuming. I do not struggle with answering questions using an algorithm, as most questions	From this response, it can be said that students aren't troubled by learning algorithms, however, to create a well-rounded system for different types of students, I will include a guide on performing the algorithm. The main challenge for all three students is using the answer obtained from performing the algorithm and identifying what they are doing at each step. This can be improved by teaching more than

<ul style="list-style-type: none"> <li>- Answering exam questions using an algorithm</li> <li>- Remembering an algorithm during an exam</li> </ul> <p>This question allows me to identify the current general weaknesses of students so I can implement features to place emphasis on correcting them e.g. through more in-depth explanations of exam questions.</p>	<p>step of the algorithm does. The first handful of marks for each question are easily earned from memorised techniques but I always lose marks on the following parts that test if I can identify what is happening at each stage or explain a specific step of the process.</p>	<p>sometimes struggle when an algorithm needs to be used in a context that I am not familiar with. This all comes down to initial understanding of the algorithm, which I would say is the main challenge that holds me back from achieving top marks. Developing understanding of the algorithm would also help me answer questions that test my knowledge of each step of an algorithm, commonly asked as the final part of a question.</p>	<p>follow the same format, just providing a different context each time. While most questions don't test my understanding of the algorithm, being able to fully understand the logic behind each step would help me for the questions that do. When these questions are asked, I don't gain the final few marks available for explanation. I would say understanding the logic behind each step of the algorithm is the biggest</p>	<p>just performing the steps of algorithms; hence my system will require thorough descriptions that exceed the bare minimum steps demanded by the specification. Since more emphasis is placed on the understanding of the Simplex algorithm, my system will just include fixed examples for the users to learn from, instead of an additional random question generator as will be used for graph algorithms. My system will also include an interactive section for users to enter their own constraints and objective which will then have the Simplex algorithm performed on them, accompanied by full working and explanations.</p>
--	---	---	---	--

			challenge for me.	
<p>9. How confident are you with your ability to mark your own work using the resources provided?</p> <p>This question allows me to identify whether my system needs to include a full marking system that mimics an Edexcel mark scheme if students are not confident in marking their work. If students are confident with marking, basic guidance as to where marks would be awarded will be the extent of a mark scheme in my system.</p>	<p>I am fairly confident as usually the mark scheme shows what parts of working out get which marks, using different types of marks, such as M1, A1, B1. The mark schemes are also very specific in what would not be awarded, so I can ensure I am not awarding marks I would not earn in an exam.</p>	<p>Using the mark schemes for past paper questions, it is easy to allocate marks and see what you needed to include to gain them. However, the textbook only provides answers to questions and does not include methods or detailed mark allocation.</p>	<p>I am confident in marking my own work. Using past paper questions and mark schemes, I can clearly see where marks are allocated and where I lost marks. By splitting up marks into M1, A1, B1 etc. it is easy to see what the question prioritises in terms of method shown, with some questions having many M1 for method and generally only 1 A1 mark for answer.</p>	<p>These answers suggest confidence in being able to mark their own work. This allows me to identify a limitation of my system, prioritising explanation and guidance for students instead of focusing on the detailed allocation of marks for a question.</p>

<p>10. How beneficial do you think an interactive and visual learning tool would be for helping you retain your knowledge through to year 13?</p> <p>This question allows me to identify whether [SCHOOL NAME] ABBREVIATION students would benefit from a system as a method of revision they can use to remind themselves of topics they struggle on through to year 13 where they will not have frequent lessons in decision maths.</p>	<p>It would be a lifesaver, having to balance the new workload of 4 subjects for my A levels alongside constant revision of Decision Maths. On top of that I will have to relearn most of the decision maths algorithms. Using the textbook to do so will be slow, however I do not have any other option. If there was a tool to help me with this, I will have more time to focus on practicing new content</p>	<p>Since we won't have frequent decision maths lessons in Year 13, I am likely to forget some of the difficult topics without proper revision. It will be difficult for me to factor in constant decision maths revision alongside all the new content we learn, especially since my main form of revision is quite time-consuming. Using a visual learning tool would help me revise specific topics that I find difficult, and the</p>	<p>With 4 subjects, I think that implementing decision maths revision into my schedule would be unsustainable. With new context covered in year 13, such as the Core Pure 2 content as well as Further Mechanics, I will struggle to maintain my knowledge of decision maths, as I am unlikely to have enough time to frequently answer past exam questions. I would benefit from a system that can remind me</p>	<p>The 3 students agree that using current resources through Year 13 to keep up with Decision maths content would be unsustainable, especially with more content from other subjects being taught simultaneously. A system that can be used to remind students of the basis of algorithms, show how they work, and explain key points that they may have forgotten regarding a certain topic will help to balance the workload of different subjects into Year 13, where they will not be having frequent lessons to cover content.</p>
---	---	--	---	---

		knowledge is more likely to remain with me through Year 13.	of various topics and cement my understanding of decision maths to revise other areas of the course through Year 13.	
11. I presented students with 2 of the researched existing systems (MyMaths and 'VoGA') and asked for their feedback using it to learn and understand Kruskal's algorithm.  This question will allow students to highlight the features they find beneficial in 2 of the existing systems I	I like how both systems introduce the topic, especially 'VoGA' as it gives real world scenarios in which the algorithm may be used. I find 'VoGA' to be more user-friendly as it splits up the topic into different options, as well as having user interactivity , but MyMaths	I think that MyMaths is a better tool for developing exam knowledge since it is tailored specifically towards A-Level Further Maths students. The content of 'VoGA' is insightful and presented in a clear layout, however, the knowledge is more generic and not as useful for me as an Edexcel further maths student.	For me, 'VoGA' was more beneficial in understanding Kruskal's algorithm, since information is presented in an intuitive and clear manner. Furthermore, the step-by-step explanation helped me to develop a solid understanding of the	Students benefitted from the introduction to the topic, which presented practical applications of the algorithm. All three students also agreed that 'VoGA' presented information in a clearer layout, helping them while learning the topic. I will ensure my system has a simple, clear layout that presents information in an effective way. Students benefit greatly from tailored content,

<p>researched and any features they find unnecessary or not useful. I selected 'VoGA' for the high-quality visualisation and explanation alongside the clearly laid out GUI, and I chose MyMaths as the content is specifically tailored to A-Level Further maths students.</p>	<p>has a cramped layout, so I struggle to take in information and occasionally missed important details as they were spread over the pages.</p>	<p>topic. I also found the 'create a graph' feature useful, as I could enter my own problem into the system and perform an algorithm on it. This helped my further develop my understanding of the topic. In some respects, such as viewing example questions, MyMaths was better, since the topic content is designed specifically for A-Level</p>	<p>which fits the required knowledge of the course and is not excessive. Student 1 and Student 3 also pointed out that the layout of MyMaths made it difficult to take in some key points.</p>
---	---	---	--

			further maths students, however the layout made it more difficult to intake information.	
--	--	--	--	--

### Summary of Limitations

Limitation	Justification for excluding this feature
An option for users to learn the different types of Simplex algorithm (Two-Stage method and Big-M method)	In my interviews, the students highlighted that learning the steps of algorithms wasn't an issue, suggested by Student 2 who said, 'I find learning and remembering the steps of an algorithm easy.' This point is reinforced by Student 3 who mentioned that 'learning and remembering are not very time consuming.'; hence I will focus on explaining the steps of the standard Simplex algorithm to place emphasis on student understanding of the topic, which can then be applied to obscure questions.
A detailed marking feature for the solutions of the fixed examples, randomly generated graph questions, user-inputted graphs and simplex tableaux that follows the Edexcel mark scheme, showing what marks are awarded for and what would not be awarded marks.	From the interviews, all 3 students were confident in marking their own work, for example, when they complete past exam questions. Instead of a full in-depth marking feature, my system will instead guide students by briefly mentioning where marks would be awarded as they walkthrough the step-by-step solution provided for each question.

A random Simplex tableau generator for users to practice applying the Simplex algorithm	As I aim to implement a random graph generator for Kruskal's algorithm and Prim's algorithm, I considered adding a random Simplex tableau generator. However, from the interview, I identified that the issue of a lack of questions was mainly found for graph algorithms, with Student 3 mentioning 'certain topics do not have many questions to practice, such as Kruskal's algorithm' and Student 1 reiterating this by saying there are 'many questions for most of the other topics, including the Simplex algorithm, however, the 'Algorithms on Graphs' topic doesn't have many practice questions'. From this I can identify that the benefit provided from a random generator would be mainly considered useful for the graph algorithms topic, hence I will exclude such a feature for the Simplex algorithm topic.
A section to explain the Travelling Salesman Problem topic to students with similar features as have been proposed for the other topics.	While the students identified the Travelling Salesman Problem (TSP) as a difficult topic, the main concept involved is the application of minimum spanning tree algorithms. For this reason, I will include a section on Kruskal's and Prim's algorithm, teaching students how to apply their knowledge to different scenarios, which will directly help students in questions about the TSP.
Content that is not required by the Edexcel A-Level Further Maths specification	Students mentioned in the interview that one of the drawbacks of the 'VoGA' existing system was the fact that it was very broad, and contained sections that students would not be required to know. This is shown by Student 2's

	<p>answer to question 11 ‘the knowledge is more generic and not as useful for me as an Edexcel further maths student.’. When researching ‘VoGA’ as an existing system, I also found that it contains many facts or concepts that A-Level students wouldn’t be tested on, such as the pseudocode for different graph algorithms. To ensure my system remains relevant for [SCHOOL NAME ABBREVIATION] students as my stakeholders, this will be a limitation of my system.</p>
--	--

### Summary from Interviews:

From these interviews, I can now identify what features current [SCHOOL NAME ABBREVIATION] Further Maths students will find beneficial in a system to help them revise decision maths. The main topics I will include are 2 minimum spanning tree algorithms (Kruskal’s algorithm and Prim’s algorithm) as these are utilised in a variety of questions, meaning that being able to fully understand their functionality will help students gain more marks. The last main topic for my system will be the standard Simplex algorithm. The three students all agreed that the Simplex algorithm is the most challenging topic, hence I will ensure that the complex details are well-explained and informative for students to revise the topic. As students do not struggle with learning algorithms and place more value on the understanding of algorithms, I will put more focus on helping students understand the logic behind the steps of different algorithms and include a summary on what steps the students need to know. A limitation of my system will be not including the different types of the Simplex algorithm, like the Two-Stage method and Big-M method, as the interviewed students do not struggle with learning and remembering the algorithms and just applying them to a given context and would instead prefer better explanation on the steps. Since the students are clear about marking their own work, this can also be considered a limitation of my system. I will not include detailed mark schemes in my system, and instead ensure my explanation contains all the marking points for an exam question which will be

briefly mentioned to the user. Students also suggested that the Travelling Salesman problem is a difficult topic, which uses graph algorithms. This forms a further limitation of my system, as I will only include the minimum spanning tree aspect of this topic to teach students how to apply the algorithms to more complex questions. I will ensure that visualisation is included in my system as the benefits were outlined by all 3 students, with Student 1 mentioning with the textbook he often ‘cannot picture’ what the algorithm is doing. Presenting the students with 2 of the existing systems that I researched allowed me to identify that a clearly laid out GUI is necessary for the user experience to ensure key information is not missed due to a cramped layout. This was mentioned by Student 3 who said, ‘the layout [of MyMaths] made it more difficult to intake information.’ This question also allowed me to outline the necessary features that are well-implemented by the current existing systems that I will also include in my system, alongside any unnecessary features that [SCHOOL NAME ABBREVIATION] students did not find helpful. The introduction to the topic was highlighted as a beneficial feature by the students, with Student 1 mentioned he liked “VoGA” as it gives real world scenarios in which the algorithm may be used.’ Hence, I will ensure there is an introductory area to each of the topics in my system. Another necessary feature will be the clearly laid-out GUI, as it benefitted students when they were presented with two of the existing systems. MyMaths’ layout made it more difficult for students to take in information, while VoGA’s intuitive method of splitting up content into different sections was deemed helpful.

This system will benefit [SCHOOL NAME ABBREVIATION] students in developing an initial understanding of a topic and revising content through year 13. Since all the interviewed students agreed that understanding the purpose behind the steps of an algorithm is the most difficult aspect, building on the initial understanding and providing a system that can remind students of concepts they may forget will be vital to ensure they retain their knowledge through year 13. The students agreed that their current revision methods would be unsustainable in maintaining a strong understanding of decision maths, given that they have new content to learn, and keep up with old topics from other subjects. By creating a system that can help students revise and answer questions they may not know how to currently answer by explaining the reasoning behind each step they have learnt as

part of an algorithm, these students will be able to achieve a high final grade at the end of year 13 when their decision maths exam takes place as part of their further maths A-Level.

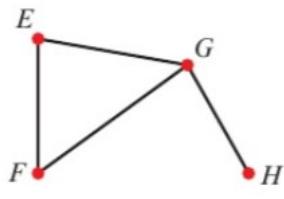
## Edexcel A-Level Further Maths Specification

The key topics that I need to research are the minimum spanning tree algorithms (Kruskal's and Prim's) which come under the 'Algorithms on graphs' section of the specification, as well as the Simplex algorithm, which is a form of linear programming (maximising an objective function subject to given inequalities known as constraints). This is a link to the specification:

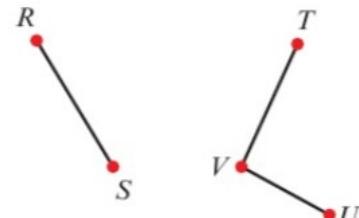
<https://qualifications.pearson.com/content/dam/pdfs/A%20Level/Mathematics/2017/specification-and-sample-assessment/a-level-l3-further-mathematics-specification.pdf>

(All images below are taken from the official Pearson Edexcel Decision 1 textbook unless otherwise specified)

A graph is defined as a collection of points (nodes or vertices) joined by lines (arcs or edges), with a connected graph having a route along edges from any vertex to any other vertex:



This is a **connected** graph.  
A path can be found between any two vertices.

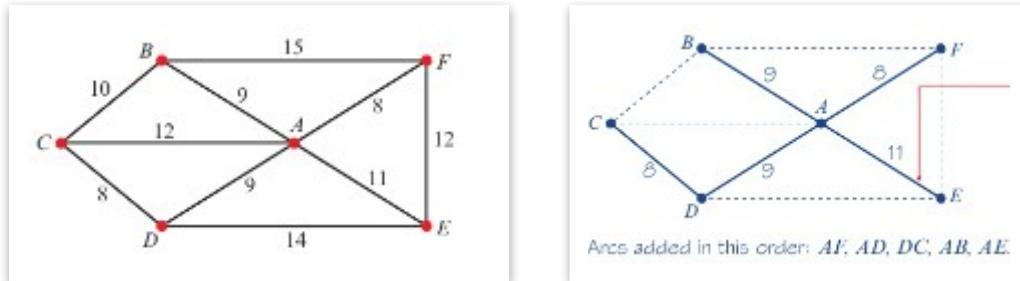


This graph is **not connected**.  
There is no path from R to V, for example.

A tree is defined as a connected graph with no cycles (A cycle is a route through a graph from one node to the next across edges where the start and ending vertex are the same and no other vertex is visited more than once). The total weight of a tree is the sum of all the arcs' weights that are included in the tree. A spanning tree is a tree that includes all the vertices of the original graph.

Below is a definition of a minimum spanning tree, as well as an example, from the Pearson ActiveLearn Decision 1 Textbook (<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>):

**1 A minimum spanning tree (MST) is a spanning tree such that the total length of its arcs (edges) is as small as possible.**



A minimum spanning tree shows the shortest way to connect all nodes in a graph, without forming any cycles. Minimum spanning trees have applications in other areas of the Edexcel A-Level Further Maths course, such as the Travelling Salesman problem, which was identified as a difficult topic during the student interviews. A node is most commonly denoted by a single letter and an arc can be denoted by the two nodes it connects, for example, the edge with weight 15 in the graph shown above would be written as BF. In a graph with  $n$  vertices, any minimum spanning tree must have  $n-1$  edges. If it had fewer than  $n-1$  edges, it would not be connected, and if it had more than  $n-1$  edges, it would contain a cycle. My system will consider a minimum spanning tree to be complete once  $n-1$  edges have been added to the tree.

Section 2.1 of the Decision Maths part of the specification outlines what students are expected to know for minimum spanning tree algorithms:

2.1 <b>The minimum spanning tree (minimum connector) problem. Prim's and Kruskal's algorithm.</b>	<b>Matrix representation for Prim's algorithm is expected. Drawing a network from a given matrix and writing down the matrix associated with a network may be required.</b>
---	---

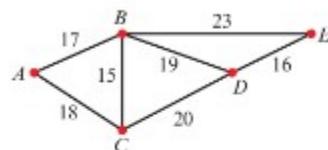
Students need to be able to perform Kruskal's and Prim's algorithm on a graph and use matrix representation, drawing a network given a distance matrix. Students are also expected to perform Prim's algorithm using just a provided distance matrix. My system will include both graphical representation and matrix representation of a network, showing how Kruskal's algorithm can be used on a graph and how Prim's algorithm can be used on both graph and matrix. Below is an image showing how a graph can be represented as a matrix.

### Example 7

Use a distance matrix to represent this network.

	A	B	C	D	E
A	—	17	18	—	—
B	17	—	15	19	23
C	18	15	—	20	—
D	—	19	20	—	16
E	—	23	—	16	—

Notice that the matrix is symmetrical about the leading diagonal (top left to bottom right). This will be the case for any non-directed network.



**Watch out** You should be able to write down the distance matrix given the network and draw the network given the distance matrix.

If the network has directed edges the distance matrix will not be symmetrical.

The matrix is read from the top row to the side column e.g. AB would be the circled value. When performing Prim's algorithm on a distance matrix, the row of the selected value is crossed out to ensure the same edge is not added twice to the MST. When deciding the next edge to add to the MST, any value from the columns of the already added vertices can be selected, if it is the smallest value available. This is shown below:

	1	2		
A	—	27	12	25
B	27	—	47	15
C	12	47	—	87
D	23	15	28	—
E	74	71	87	75

In this example, I could choose any value from the A column or C column, since those vertices are already part of the MST, however as I must choose the smallest value, I would choose AD (23) to add, then cross out the row of vertex D.

Kruskal's algorithm creates a minimum spanning tree using unconnected subtrees which eventually combine into one tree, whereas the tree in Prim's algorithm grows singularly as the algorithm is performed.

### **Kruskal's Algorithm**

I will split Kruskal's algorithm into the steps that [SCHOOL NAME ABBREVIATION] students are taught to ensure they do not skip over any part of the method that may cost them marks. These steps can be found below:

1. Sort all the edges into a list in ascending weight order.
2. Choose the edge with the lowest weight to begin the tree.
3. Check the edge with the next lowest weight, if it connects two vertices that are already included in the tree, then reject the arc as it would form a cycle. If adding the edge does not form a cycle, then add it to the tree.
4. Repeat the third step until all the vertices in the original graph are contained in the minimum spanning tree.
5. Output the final MST and its total weight alongside the arcs included.

The steps defined above will closely follow what students are taught, as shown by the image of the steps given in the Pearson Edexcel Decision Mathematics 1 textbook, hence my defined steps will be beneficial to students revising the algorithm.

- 1** Sort all the arcs (edges) into ascending order of weight.
- 2** Select the arc of least weight to start the tree.
- 3** Consider the next arc of least weight.
  - If it would form a cycle with the arcs already selected, reject it.
  - If it does not form a cycle, add it to the tree.
 If there is a choice of equal arcs, consider each in turn.
- 4** Repeat step **3** until all vertices are connected.

This algorithm finds a minimum spanning tree by adding and rejecting arcs by weight in ascending order. If adding a specific arc would create a cycle, that arc is rejected to ensure the definition of a tree is maintained. This is repeated until all the vertices are included. I will demonstrate the steps defined above on question 5 from the Edexcel June 2015 Decision Maths Question paper:

5.

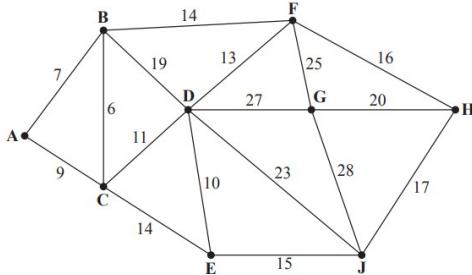


Figure 5

The numbers on the 17 arcs in the network shown in Figure 5 represent the distances, in km, between nine nodes, A, B, C, D, E, F, G, H and J.

- (a) Use Kruskal's algorithm to find a minimum spanning tree for the network. You should list the arcs in the order in which you consider them. In each case, state whether you are adding the arc to your minimum spanning tree.

(3)

First, we sort the arcs into ascending order, which I will format as a table to allow clear addition and rejection to the minimum spanning tree. The next step is to add the first edge to the MST (BC) and consider each following edge to see whether its addition would result in a valid MST. For explanation purposes, I will include a reason for each of the rejected arcs, however students are not required to justify their decisions in the question.

Arc	Add/Reject
BC	Add
AB	Add
AC	Reject (Adding this arc would form a cycle ABCA)
DE	Add
CD	Add
DF	Add
BF	Reject (Adding this arc would form a cycle BCDFB)
CE	Reject (Adding this arc would form a cycle CDEC)
EJ	Add
FH	Add
HJ	Reject (Adding this arc would form a cycle DFHJED)
BD	Reject (Adding this arc would form a cycle BDCB)
GH	Add
DJ	Reject (Adding this arc would form a cycle DJED)
FG	Reject (Adding this arc would form a cycle FGHF)
DG	Reject (Adding this arc would form a cycle DGHFD)
GJ	Reject (Adding this arc would form a cycle DFHGJED)

Something to be noted is that once the edge that connects the final vertex not currently in the MST has been added, all subsequent edges can be rejected. To implement this in my system, I will stop adding edges to the tree when  $n-1$  edges have been added. The total weight of the MST is 98km.

### Prim's Algorithm

Prim's algorithm also finds a minimum spanning tree but uses a different method by only checking arcs adjacent to vertices already included in the tree. I will also split Prim's algorithm into the steps that [SCHOOL NAME ABBREVIATION] students are taught for both graphical representation and matrix representation.

For graphical representation, my system must:

1. Select any vertex to begin the tree
2. Select the arc adjacent to a vertex already in the tree that connects it to a vertex not already contained in the tree, ensuring the selected arc has the least weight of the possible adjacent arcs (if two are equal, either can be chosen).

3. Continue adding arcs, growing the tree until all vertices are connected.
4. Output the final MST, including the total weight and all included arcs.

Below are the steps given in the official Pearson Edexcel Decision Maths 1 textbook, closely following the steps I defined above.

- 1** Choose any vertex to start the tree.
- 2**
  - Select an arc of least weight that joins a vertex already in the tree to a vertex not yet in the tree.
  - If there is a choice of arcs of equal weight, choose any of them.
- 3** Repeat step **2** until all the vertices are connected.

For matrix representation

1. Select any vertex to begin the tree and label the column with '1'
2. Cross out the row of the chosen vertex in the matrix to avoid adding the same edge twice to the MST.
3. Look through the columns of any vertices that have already been added to the MST. This step is explained above in the explanation of matrix representation. Select the smallest possible value (if 2 values are equal, choose either) and add the arc to the MST by putting a circle around the value before crossing out the rest of the row.
4. Label the respective column of the row from which the edge was chosen with the number at which it was added to the tree, e.g. the second vertex to be added is labelled '2'.
5. Repeat steps 3 and 4 until all the rows have been crossed out, suggesting that all vertices have been added to the minimum spanning tree.
6. List the edges added to the MST and output its total weight.

- 1** Choose any vertex to start the tree.
- 2** Delete the **row** in the matrix for the chosen vertex.
- 3** Number the **column** in the matrix for the chosen vertex.
- 4** Put a ring round the lowest undeleted entry in the numbered columns. (If there is an equal choice, choose randomly.)
- 5** The ringed entry becomes the next arc to be added to the tree.
- 6** Repeat steps **2, 3, 4** and **5** until all rows have been deleted.

Above are the steps from the Pearson Edexcel Decision Maths 1 textbook, closely following those I have defined, justifying my steps will be suitable for [SCHOOL NAME ABBREVIATION] students.

Using the outlined steps, I will walk through 2 questions, one performing Prim's on a graph and one on a matrix. The first question is taken from the Edexcel June 2014 Decision 1 Question Paper. The question mentions beginning from Art (A); hence this will be the first vertex added to the minimum spanning tree.

	Art	Biology	Chemistry	Drama	English	French	Graphics
Art (A)	—	61	93	73	50	48	42
Biology (B)	61	—	114	82	83	63	58
Chemistry (C)	93	114	—	59	94	77	88
Drama (D)	73	82	59	—	89	104	41
English (E)	50	83	94	89	—	91	75
French (F)	48	63	77	104	91	—	68
Graphics (G)	42	58	88	41	75	68	—

The table shows the travelling times, in seconds, to walk between seven departments in a college.

- (a) Use Prim's algorithm, starting at Art, to find the minimum spanning tree for the network represented by the table. You must clearly state the order in which you select the **edges** of your tree.

(3)

The only vertex currently in the tree is A, hence the next value must be chosen from the A column. The smallest is 42 from AG, hence the G row is crossed out and the G column is labelled with 2.



	Art	Biology	Chemistry	Drama	English	French	Graphics
Art (A)	—	61	93	73	50	48	42
Biology (B)	61	—	114	82	83	63	58
Chemistry (C)	93	114	—	59	94	77	88
Drama (D)	73	82	59	—	89	104	41
English (E)	50	83	94	89	—	91	75
French (F)	48	63	77	104	91	—	68
Graphics (G)	42	58	88	41	75	68	—

Out of the A column and G column, 41 from GD is the smallest value, hence the D row is crossed out and the D column is labelled 3. Looking through A, D and G, the smallest value is AF (48), so the F row is crossed out and the respective column is labelled 4. The minimum spanning tree currently contains AG, GD, AF.



	Art	Biology	Chemistry	Drama	English	French	Graphics
Art (A)	—	61	93	73	50	48	42
Biology (B)	61	—	114	82	83	63	58
Chemistry (C)	93	114	—	59	94	77	88
Drama (D)	73	82	59	—	89	104	41
English (E)	50	83	94	89	—	91	75
French (F)	48	63	77	104	91	—	68
Graphics (G)	42	58	88	41	75	68	—

The next smallest value from the available rows is AE, so the English row is crossed out and E is added as the next vertex included in the MST. From the available columns, the next smallest value is 58 from GB, hence Biology is added to the MST.

06 3541

	Art	Biology	Chemistry	Drama	English	French	Graphics
Art (A)	—	61	93	73	50	48	42
Biology (B)	61	—	114	82	83	63	58
Chemistry (C)	93	114	—	59	94	77	88
Drama (D)	72	82	59	—	89	104	41
English (E)	50	83	94	89	—	91	75
French (F)	48	63	77	104	91	—	68
Graphics (G)	42	58	88	41	75	68	—

The final row is Chemistry (C), and the smallest value is 59 from edge DC, hence it is added to the MST and the C column is labelled with 7.

0673541

	Art	Biology	Chemistry	Drama	English	French	Graphics
Art (A)	—	61	93	73	50	48	42
Biology (B)	61	—	114	82	83	63	58
Chemistry (C)	93	114	—	59	94	77	88
Drama (D)	72	82	59	—	89	104	41
English (E)	50	83	94	89	—	91	75
French (F)	48	63	77	104	91	—	68
Graphics (G)	42	58	88	41	75	68	—

All the vertices have now been added to the MST, and the included arcs are AG, GD, AF, AE GB, DC. The answer can be confirmed using the mark scheme shown below. Note that the mark scheme followed general convention and labelled the edges alphabetically, however for purposes of consistency, I followed the method of reading the matrix going down the column then across the row and denoted

the edges accordingly. My system will display the solution to the question in a similar way, displaying the updated distance matrix with rows crossed out, columns labelled, and values circled. My system will also ensure that the rules of Prim's algorithm are followed by checking that any arcs that could be added connects one vertex that is already in the tree to a vertex that is not.

The following question asks the student to perform Prim's algorithm on a graph and is taken from the Edexcel June 2017 Decision Maths Question Paper.

2.

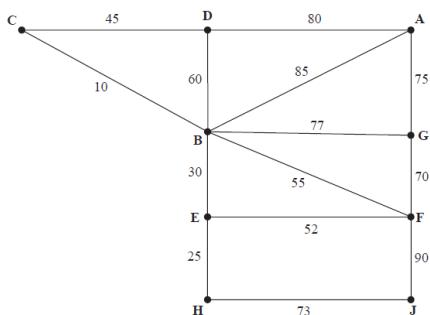


Figure 3

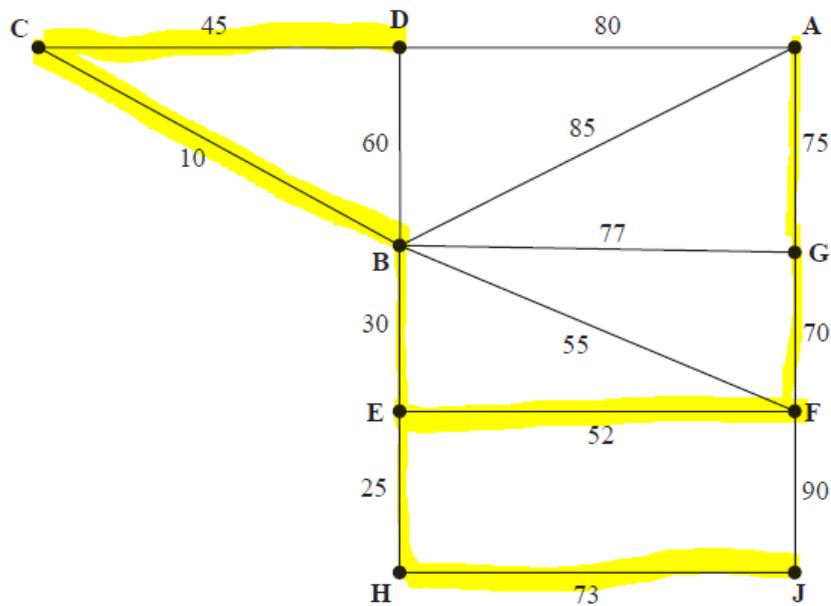
Figure 3 represents nine computer terminals, A, B, C, D, E, F, G, H and J, at Pearsonby School. The school wishes to connect them to form a single computer network. The number on each arc represents the cost, in pounds, of connecting the corresponding computer terminals.

- (a) Use Prim's algorithm, starting at B, to find the minimum spanning tree for the computer network. You must clearly state the order in which you select the arcs of your tree.

(3)

The question instructs the students to begin at vertex B, hence the first arc chosen will be BC, since it has the lowest weight of the arcs adjacent to vertex B. Looking at the arcs adjacent to vertices B and C, the next smallest is BE. The next smallest arc is EH with a weight of 25, so it is added to the MST. From the added vertices, the next arc with the lowest weight is EF, followed by GF. The final two arcs to be added are HJ (73) and AG (75), so all the vertices have now been included in the MST. Highlighted below are the added arcs forming the MST.

The list of final added arcs is BC, BE, EH, CD, EF, GF, HJ, AG with a total weight of 380



**Figure 3**

This result can be confirmed using the mark scheme shown below:

Question Number	Scheme	Marks
2. (a)	BC, BE, EH, CD; EF, FG; HJ, AG	M1; A1; A1 (3)

My system will highlight the arcs as it is selected and justify why it was added to the tree, as shown above.

Having looked at the PowerPoints used by [SCHOOL NAME ABBREVIATION] teacher to teach students; the steps provided are taken from the textbook, hence using it as a basis to outline my steps will ensure students are familiar with the steps that system will provide and allow them to consolidate their understanding.

**Kruskal's Algorithm** can be used to find a minimum spanning tree.

1. Sort all the arcs into ascending order of weight.
2. Select the **arc** of least weight to start the tree.
3. Consider the next **arc** of least weight.
  - If it would form a cycle with the arcs already selected reject it.
  - If it does not form a cycle, add it to the tree.

**Uses of minimum spanning trees**

- Cluster Analysis.
- Real-time face tracking and verification (i.e. locating human faces in a video stream).
- Protocols in computer science to avoid network cycles.
- Entropy based image registration.
- Max bottleneck paths.
- Dithering (adding white noise to a digital recording in order to reduce distortion).

Kruskal's Algorithm is sometimes called the "greedy algorithm" because it gobbles up the best (least weight) arcs first. It considers the **arcs**, not the vertices.

## Simplex Algorithm

The Simplex algorithm is used for linear programming problems (maximising or minimising an objective function subject to specific constraints/inequalities) that may not be possible to solve using graphical methods when there are more than 2 unknowns involved. The algorithm starts at one vertex of the feasible region created by plotting the inequalities and moves between vertices, increasing the objective function and a single variable at each time until it reaches the optimal solution. The variables must follow certain rules, such as non-negativity. This suggests that the value of all variables must not be negative, hence questions will often provide the student with inequalities of non-negativity e.g.  $x \geq 0, y \geq 0, z \geq 0$ . The Simplex algorithm is considered A2 (second year of A-Level studies) content meaning it is not taught for those just taking the AS (first year of A-Level) exam. However, [SCHOOL NAME ABBREVIATION] students take the full A-Level and are required to learn it. The standard Simplex method can only be used when all the constraints (excluding the non-negativity constraints) use  $\leq$  inequalities. Problems that include  $\geq$  constraints have no obvious basic feasible solutions as the origin is not included in the feasible region. These must be converted to equations using a surplus variable, which are also subject to non-negativity conditions, however, just using a surplus variable doesn't solve our issue of being able to use the Simplex method. We also need to introduce a new variable called an artificial variable that helps ensure all variables can maintain non-negativity. This is known as the two-stage Simplex method, which aims at minimising/maximising the sum of artificial variables as a way of maximising/minimising the objective function. This is the first stage, and if the sum of artificial variables is 0, we move onto the second stage

which is the same as applying the standard simplex algorithm, and if the sum is not 0, there is no feasible solution. Another method of dealing with  $\geq$  is the Big-M method, introducing M (an arbitrarily large real number). This guarantees that a term like  $M - 200$  is always positive since M represents a number much larger than 200. The Big-M method involves a different but equivalent approach, summing the artificial variables to 0. Both of these methods were identified as limitations of my system, justified by stakeholder responses in my interviews, hence I will not be including them in my system.

Before using the Simplex method, students will often have to construct their own inequalities from a given context, then convert them into equations to add to the tableau. In the standard Simplex method, this is done using a slack variable, which is added to an inequality to take up any spare capacity to ensure the equation holds true. When adding the objective function to the tableau, the variables are moved to one side e.g.  $P = 3x + 2y$  would become  $P - 3x - 2y = 0$  and the negative coefficients are entered. This allows the first step of the algorithm to be carried out. Below is an example of a simplex tableau with the constraints and objective provided:

$$\text{Maximise } P = 3x + 2y$$

subject to:

$$5x + 7y + r = 70$$

$$10x + 3y + s = 60$$

$$x, y, r, s \geq 0$$

Our initial tableau is:

Basic variable	x	y	r	s	Value
r	5	7	1	0	70
s	10	3	0	1	60
P	-3	-2	0	0	0

The first row shows the first constraint, the second row shows the second constraint and the final row shows the objective function.

The basic variables are added to the leftmost column with P (objective variable) forming the bottom row. All variables are added along the top, with the value column being the rightmost. The coefficient of each non-basic variable is added to its respective row and column e.g. the 5 from  $5x$  is added in the 'r' row and 'x' column, since the slack variable 'r' was added to its inequality, and it is the coefficient of x. Any following tableaus would contain a 'Row Operation' column to the right of the value column, in which the student is expected to write down the

operation performed during each iteration. A row operation is used to reduce the pivot value to 1 and other values in the pivot column to 0. My system will display the Simplex tableaux in this same format to ensure it replicates the way students are familiar with applying the algorithm.

For the fixed examples, I will display the full process, e.g. forming inequalities from a given context and converting them to equations, appending to the initial tableau, and iterating to the final solution. For the feature that allows users to enter their own equations into a table, forming the inequalities will not be necessary, as the inputs will be directly entered into the tableau, so they will already have been formed into equations, hence the only required steps for all features pertaining to the Simplex algorithm begin from calculating an initial pivot value. These steps can be found below with explanation, but the main part of each step has been bolded:

1. **Identify the most negative value in the objective row to identify the pivot column**
2. **Calculate the theta values** by dividing the term in the value column by the term in the pivot column for all constraint rows (all rows with values excluding the objective row). The row containing the value that gave the smallest positive theta value becomes the pivot row. Now the overlapping value between the pivot column and pivot row can be identified as the pivot.
3. The aim is to reduce the pivot value to 1 by multiplying/dividing. Add the following to a new tableau. To achieve this, **divide every term in the pivot row by the pivot value and in the 'Row Operations' column, write down the operation performed e.g. R2 / 4. Change the basic variable at the start of the row to the variable heading of the pivot column.**
4. The next goal is to reduce every other value in the pivot column to 0 and adjust all the other values in the other rows. This can be achieved by **subtracting different quantities of the '1' of the pivot value in the pivot row.** For example,  $R_3 - 4R_1$ . For the remaining values in the row, **perform the same operation using the old values for the current row and the new value from the pivot row.** Taking the example from before, to calculate the new value in a row that is not in the pivot row or column, we would do the value from the  $R_3$  in the previous tableau –  $4^*$  the value in the

current tableau in the same column as the value in row 3, but in the pivot row.

5. **Repeat steps 1 to 4 until there are no negative numbers in the objective row. Read off the value of each variable.** The variable is identified by the B.V. column and the value of this is its respective entry in the value column.

For minimisation problems, an extra variable Q is introduced, which is equal to the current variable denoting the objective multiplied by -1. For example,  $P = 3x - 2y$ , then  $Q = -P$ , hence  $Q = -3x + 2y$ . Minimising using the standard Simplex algorithm works by maximising the negative version of the objective. Minimisation follows the same steps from above, however, when filling in the initial tableau, the objective row is denoted with the new variable Q and the coefficients are added in a similar manner to above moving all variables to one side. At the end, once there are no negative values left in the objective row, the values of each variable can be read off from the value column of the tableau, however, to find the value of P, we must multiply the calculated value of Q by -1.

The steps for the standard Simplex method from the textbook are shown below, showing a similar pattern of identifying the pivot value then iterating around it until the optimal solution has been reached. The textbook's process starts from the very beginning, drawing tableaux, however, my system will only show this step for the fixed example questions.

- 1** Draw the tableaux.  
You need a basic variable column on the left, one column for each variable (including the slack variables) and a value column. You need one row for each constraint and the bottom row for the objective function.
- 2** Create the initial tableau.  
Enter the coefficients of the variables in the appropriate column and row.
- 3** Look along the objective row for the most negative entry: this indicates the pivot column.
- 4** Calculate the  $\theta$  values, for each of the constraint rows, where  
 $\theta = (\text{the term in the value column}) \div (\text{the term in the pivot column})$
- 5** Select the row with the smallest, positive  $\theta$  value to become the pivot row.
- 6** The element in the pivot row and pivot column is the pivot.
- 7** Divide the row found in step **5** by the pivot, and change the basic variable at the start of the row to the variable at the top of the pivot column. This is now the pivot row.
- 8** Use the pivot row to eliminate the pivot's variable from the other rows.  
This means that the pivot column now contains one 1 and zeros.
- 9** Repeat steps **3** to **8** until there are no more negative numbers in the objective row.
- 10** The tableau is now optimal and the non-zero values can be read off using the basic variable column and value column.

As is evident, this algorithm has many steps to follow, resulting in room for error at any stage. For this reason, my system will explain each step in an easy-to-understand way, ensuring students are confident at every stage of the algorithm. In the Edexcel A-Level specification, it is stated that students are only expected to be able to perform the Simplex algorithm on a maximum of 4 variables, hence my system will allow users to choose between 2,3 or 4 variables as part of the interactive section.

I will now walkthrough the mark scheme of both a minimisation question and a maximisation question using the standard Simplex algorithm.

- 2.** The tableau below is the initial tableau for a maximising linear programming problem.

Basic Variable	$x$	$y$	$z$	$r$	$s$	Value
$r$	2	3	4	1	0	8
$s$	3	3	1	0	1	10
$P$	-8	-9	-5	0	0	0

- (b) Solve this linear programming problem.

(8)

The most negative value is the -9 in the 'y' column, hence 'y' becomes our pivot column. Calculating the theta values, we get  $8/3$ , and  $10/3$ .  $8/3$  is the smaller of the two hence the 3 in the 'r' row is chosen. This gives us our pivot value of 3.

The first iteration is carried out. The row operation for row 1 is  $R1 * 1/3$ , bringing our pivot value to 1. The row operation for row 2 is  $R2 - 3R1$ . The pivot value is now at 1, so taking  $3*1$  away from 3 gives us the 0 we require. The row operation for the third row is  $R3 + 9R1$ , adding 9 lots of the pivot value brings the -9 to 0. All other values in each row have the same row operation applied to them, giving us the table below.

b.v	x	y	z	r	s	Value
r	2	3	4	1	0	8
s	3	3	1	0	1	10
P	-8	-9	-5	0	0	0

The pivot column header 'y' has now replaced 'r' as the pivot row label. We repeat the process, identifying 'x' as the pivot column and 's' as the pivot row. Currently, there are still negative values in the objective row, so an optimal solution has not yet been reached, hence we must continue with the iterations. Our pivot value is already at 1, hence no row operation needs to be performed on the pivot row. The first row has the row operation  $R1 - 2/3 * R2$ , which brings the  $2/3$  to 0. The row operation for the third row is  $R3 + 2R2$ , bringing the value of -2 to 0.

b.v	x	y	z	r	s	Value
y	$\frac{2}{3}$	1	$\frac{4}{3}$	$\frac{1}{3}$	0	$\frac{8}{3}$
s	1	0	-3	-1	1	2
P	-2	0	7	3	0	24

$R_1 \div 3$   
 $R_2 - 3R_1$   
 $R_3 + 9R_1$

The pivot column header 'x' has now replaced the 's' as the row label. Observing the objective row, we can see that there are no longer any negative values, hence an optimal solution has been reached. The values are as follow:  $x = 2$ ,  $y = 4/3$ ,  $z = 0$ ,  $r = 0$ ,  $s = 0$  and  $P = 28$ .

b.v	$x$	$y$	$z$	$r$	$s$	Value
$y$	0	1	$\frac{10}{3}$	1	$-\frac{2}{3}$	$\frac{4}{3}$
$x$	1	0	-3	-1	1	2
$P$	0	0	1	1	2	28

$$R_1 - \frac{2}{3}R_2$$

$$R_3 + 2R_2$$

The following minimisation question is taken from the Pearson Edexcel Decision Mathematics textbook as an example.

To minimise  $P$ , we can maximise the negative of  $P$ , hence  $Q$  is defined as  $-P$ .

Minimise  $P = 3x - y$

subject to:

$$2x + y \leq 12$$

$$x + 4y \leq 8$$

$$x \geq 0, y \geq 0$$

$$P = 3x - y$$

$$\text{Define } Q = -P = -3x + y$$

Slack variables are introduced to each of the inequalities to convert them into equations. Slack variables must follow the rules of non-negativity like the non-basic variables. This is because in a real-life situation, where the algorithm is used to find the optimal quantity of items to produce for example, there must not be a negative number of items.

$$2x + y + r = 12$$

$$x + 4y + s = 8$$

$$x, y, r, s \geq 0$$

Appending the inequalities to the initial tableau, we are given this:

Basic variable	$x$	$y$	$r$	$s$	Value
$r$	2	1	1	0	12
$s$	1	4	0	1	8
$Q$	3	-1	0	0	0

The most negative value in the objective row is the -1, hence 'y' is our pivot column. Calculating the theta values, we are left with 12 or 2, so the row giving 2 is selected and 's' is our pivot row. The pivot value is 4, hence the row operation for this row is  $R2 * \frac{1}{4}$ .

Basic variable	$x$	$y$	$r$	$s$	Value	Row operations
$r$	2	1	1	0	12	
$y$	$\frac{1}{4}$	1	0	$\frac{1}{4}$	2	$R2 \div 4$
$Q$	3	-1	0	0	0	

To bring the other values in the pivot column to 0, the row operation for the first row would be  $R1 - R2$  and the row operation for row 3 would be  $R3 + R2$ .

Performing these row operations on each value in the respective rows, we are left with the following tableau:

Basic variable	$x$	$y$	$r$	$s$	Value	Row operations
$r$	$\frac{7}{4}$	0	1	$-\frac{1}{4}$	10	$R1 - R2$
$y$	$\frac{1}{4}$	1	0	$\frac{1}{4}$	2	
$Q$	$\frac{13}{4}$	0	0	$\frac{1}{4}$	2	$R3 + R2$

There are no longer any negative values in the objective row, hence an optimal solution has been found. The values of each variable are as follows:  $x = 0$ ,  $y = 2$ ,  $r = 10$ ,  $s = 0$ . The value of  $Q$  is 2, hence the minimised value of  $P$  would be -2.

All the algorithms contain many steps that students must be able to apply, however, a key part that students aren't often taught is understanding the

purpose behind each step. My system will include the steps of the algorithm accompanied by reasoning as to why the step is necessary. This will help students answer questions that test their knowledge of the functionality of different algorithms, often the last part of a larger question, as suggested by the students in the interviews. An example of this type of question is shown below, where the knowledge required is not explicitly taught to students:

- 6 A craftworker makes three types of wooden animals for sale in wildlife parks. Each animal has to be carved and then sanded.

Each Lion takes 2 hours to carve and 25 minutes to sand.

Each Giraffe takes  $2\frac{1}{2}$  hours to carve and 20 minutes to sand.

Each Elephant takes  $1\frac{1}{2}$  hours to carve and 30 minutes to sand.

Each day the craftworker wishes to spend at most 8 hours carving and at most 2 hours sanding.

Let  $x$  be the number of Lions,  $y$  the number of Giraffes and  $z$  the number of Elephants he produces each day.

The craftworker makes a profit of £14 on each Lion, £12 on each Giraffe and £13 on each Elephant. He wishes to maximise his profit,  $P$ .

- a Model this as a linear programming problem, simplifying your expressions so that they have integer coefficients. (4 marks)

It is decided to use the simplex algorithm to solve this problem.

- b Explaining the purpose of  $r$  and  $s$ , show that the initial tableau can be written as:

Basic variable	$x$	$y$	$z$	$r$	$s$	Value
$r$	4	5	3	1	0	16
$t$	5	4	6	0	1	24
$P$	-14	-12	-13	0	0	0

(3 marks)

- c Increasing  $x$  first, work out the next complete tableau, where the  $x$  column includes two zeros. (2 marks)

- d Explain what this first iteration means in practical terms. (2 marks)

While students may be able to reach the correct answer for the first iteration, they may not understand what the process has done and therefore struggle to answer part (d) of the question. Providing the steps outlined above alongside explanations/justifications for each step, students will be able to better understand what they are doing at each stage.

## Success Criteria

Having outlined the key steps of each algorithm that I will include in my system, I can now identify the necessary success criteria to ensure that each topic is presented in a versatile and understandable manner.

Criterion	Evidence of Necessity	Benefit to Stakeholders
<p>1) My system must provide a clearly laid out user interface and allow users to select an algorithm to run with a menu bar.</p> <p>1.1) Each algorithm option must be split into specific sub-sections e.g. user creating their own graph, user viewing example questions.</p>	<p>From question 11 of the interview, students identified the benefit they gained from a well laid out GUI, with Student 3 mentioning the merit of 'VoGA' being the information presented in 'an intuitive and clear manner'. Student 1 also suggested the 'cramped layout' of MyMaths prevented him from absorbing information effectively, necessitating the division of user options into separate sub-sections. From the existing systems, I found that the most clearly laid out system was 'VoGA' which split content into multiple sections and had an introduction to the topic.</p>	<p>[SCHOOL NAME ABBREVIATION] students will develop a clearer understanding of the topic from the various options presented in a simple GUI. First, they can develop initial understanding from the examples, then input their own questions they struggle with to be fully explained. By splitting up each topic into smaller options, the user can learn using various techniques, from examples to interactive questions.</p>
<p>2) My system must have an option for users to learn the detailed steps of performing Prim's algorithm on a graph, providing explanations at each stage.</p> <p>This will include:</p> <p>2.1) Selecting a starting vertex from the graph</p>	<p>Question 8 of the interview highlighted that the main issue students face is understanding the steps of the algorithm/applying the algorithm in an obscure context. Student 1 mentioned that he 'always loses marks on</p>	<p>By providing justification and explanation for each step, students will be able to earn more marks that they may regularly lose, as further suggested by Student 3 in the interview 'I don't gain the final few marks available for explanation.' Teaching each step will</p>

<p>2.2) Choosing an adjacent arc of least weight</p> <p>2.3) Continue choosing arcs of least weight that connect a vertex already in the tree to a vertex not currently in the tree.</p> <p>2.4) Repeating the previous step until all vertices are included in the minimum spanning tree.</p> <p>2.5) Outputting the final MST and weight.</p>	<p>the following parts that test if I can identify what is happening at each stage,' justifying the addition of a feature that explains the steps of an algorithm in detail. Many marks in the decision maths exam are available for method (M1 marks), hence I will need to show each step of working for a specific question. The steps specified are necessary as they follow the process defined by the Edexcel A-Level textbook that students are expected to know. All the researched systems provide explanations for each step, suggesting it is a valuable feature for students. As well as this, the steps outline in the specification section (which follow the method students are taught) closely match the criterion, hence splitting the algorithm into these steps will benefit students by developing on their current</p>	<p>ensure confidence in answering questions about the algorithm. Students will be able to learn the functionality of the algorithm whilst also learning the steps of working they are required to show to gain all the marks.</p>
---	--	---

	understanding.	
<p>3) My system must have an option for users to learn how to perform was, providing explanations for each step.</p> <p>This includes:</p> <p>3.1) Selecting a vertex and crossing out its respective row.</p> <p>3.2) Searching in the visited columns for the smallest length then crossing out this row and clearly indicated the added/rejected arc.</p> <p>3.3) Repeating the above steps until a full minimum spanning tree has been created.</p> <p>3.4) Outputting the final list of arcs contained within the minimum spanning tree.</p>	<p>The specification states that 'Matrix representation for Prim's algorithm is expected,' as shown in the primary research. This form of question often appears in the exam, with the potential for students to gain/lose several marks depending on their knowledge.</p> <p>To ensure that students can understand what they are doing at each step, I will include a feature in my system that works through Prim's algorithm using a distance matrix to ensure confidence for students when they face questions of this format.</p> <p>Performing Prim's algorithm on a distance matrix can also appear in Travelling Salesman questions, identified as a difficult topic in question 1 of the interview by all 3 students. The steps defined in this criterion closely match those researched and displayed in the specification section of primary research, imitating the method [SCHOOL NAME]</p>	<p>Without a graph, the minimum spanning tree algorithms can be difficult to visualise, however, drawing out the respective graph is time-consuming, and students must be able to apply the algorithm to a distance matrix. The steps of this procedure correspond closely to the graph-based method; hence I will compare the steps in my explanation to help students understand better. By linking the matrix process to the graph-based process, students will have greater understanding of the steps involved. By outputting the steps of the method in the same way students are taught (crossing out rows, labelling columns and circling values), I can ensure that the content of the system remains relevant and useful.</p>

	ABBREVIATION] students are taught.	
4) My system must provide users with an option to learn the steps of Kruskal's algorithm in detail, providing justification for each step. This will include:  4.1) Sorting the arcs into ascending order of lengths  4.2) Adding and rejecting arcs based on whether they form a cycle or not, clearly indicating which arc was added/rejected.  4.3) Repeating the above steps until a minimum spanning tree has formed.	From the interviews, question 1 highlights the fact that students struggle with the more obscure applications of graph algorithms like Kruskal's and Prim's. For example, Student 1 mentioned 'I also struggle with the applications of graph algorithms... as part of the Travelling Salesman topic.' I will need to include a visual representation of the algorithm alongside descriptions which explain the process. The visual element will display which arcs are being added/rejected. These steps closely follow the outlined steps from the primary research under the specification section. Performing these steps and explaining to students what is happening at each stage will help develop understanding of the algorithm.	During the interviews, [SCHOOL NAME ABBREVIATION] students suggested that the application of graph algorithms throughout other topics in the course can be challenging, as they are generally taught in a single format. The working out that needs to be shown is very specific, with students often losing marks for missing out key points, such as adding and rejecting arcs. For this reason, I will include a comprehensive guide on performing Kruskal's algorithm on a graph.
5) My system must have a random graph generator with the following characteristics.	In question 3 of the interviews, students mentioned that past paper questions were the	Students would benefit from a well-rounded understanding of a specific topic, however,

<p>The graph generator must:</p> <p>5.1) Generate a random number of nodes (within a fixed range)</p> <p>5.2) Generate arcs connecting the nodes, with weight values assigned to each arc.</p> <p>5.3) Ensure that the graph generated follows the definition of a connected graph.</p> <p>5.4) Provide worked solutions for the selected algorithm to be performed on the graph</p> <p>These will form random questions for students to practice Prim's algorithm and Kruskal's algorithm.</p>	<p>most valuable form of revision to practice applying the algorithm, which is a key aspect of having a sound understanding of the subject. This response solidified by question 4, in which their effectiveness was highlighted. Student 3 stated that 'After completing past paper questions, I am more confident in the exam questions.' From the existing systems section, I identified that one of the beneficial features from GeoGebra was the random graph generator, as students can attempt performing an algorithm on the generated graph and have full solutions provided to compare their working to.</p>	<p>as mentioned by Student 2 in the interview, past paper questions 'aren't effective at developing overall understanding of decision maths.'</p> <p>By accompanying the randomly generated question with worked solutions and explanations, [SCHOOL NAME ABBREVIATION] students will gain exam practice, alongside a more thorough understanding of the steps, which will help them in various other types of question. Since there are limited number of textbook questions for this topic suggested by Student 1 'the 'Algorithms on Graphs' topic doesn't have many practice questions,' generating random questions will allow students to practice with questions they have not seen before.</p>
<p>6) My system must allow users to enter their own graphs to perform an algorithm on.</p> <p>6.1) There must be buttons to add/remove a</p>	<p>From question 7 of the interview, it can be said that students currently struggle to picture the steps of an algorithm with existing systems. This is mentioned in</p>	<p>This will allow students to get the solution to a question where they may not be able to access a mark scheme. For example, the limited mark scheme from the</p>

<p>node from the graph.</p> <p>6.2) There must be a button to connect nodes with an arc and assign a weight to it.</p> <p>6.3) There must then be a solution provided which explains the steps involved in applying the algorithm to the entered graph.</p> <p>6.4) The system must also allow students to clear the graph they have created and start a new one.</p>	<p>Student 1's response 'Using the textbook, I cannot picture what each algorithm is doing.' To improve student understanding, I will include an interactive graph editor, allowing students to enter their own graphs, and see the algorithm performed on it.</p> <p>In question of the interview, students found the interactive feature of the 'VoGA' existing system highly useful, with Student 3 stating 'This [interactive feature] helped my further develop my understanding of the topic.' Adding an interactive element to my system will help students develop a clearer comprehension of the topic. I identified VoGA's user interactivity element of creating a graph as an essential feature when researching existing systems due to its value to students as suggested in the 'Benefit to Students' column.</p>	<p>Pearson ActiveLearn textbook does not explain the steps to reach an answer in detail. Being able to interact with the system, visual learners will better understand the functionality of the algorithm. This feature will also allow students to enter the graph from a past question they struggled with, receiving an answer and full explanation to ensure they understand how to reach the answer. Student 3 mentioned that with an interactive tool, 'I would be able to access the knowledge for the conceptual questions I struggle with....,' suggesting an interactive feature would help students gain more marks on obscure questions.</p>
7) My system must provide users with fixed	Question 1 from the interview allows me to	Students will be able to gain more marks that

<p>examples with working out for the standard Simplex algorithm with all the steps fully explained/justified:</p> <p>7.1) Forming equations from inequalities and adding them to a table, with a maximum of 4 unknown variables</p> <p>7.2) Finding a pivot using theta values and the objective row.</p> <p>7.3) Completing the iterations until no further improvements can be made, outputting the updated table after each iteration.</p> <p>7.4) Outputting the final value of the objective and all variables.</p>	<p>clearly outline that understanding the Simplex algorithm is the most difficult topic in the Edexcel decision maths course, with Student 1 highlighting the difficulty due to the 'immense attention to detail required'. I will include a section that shows the algorithm being performed on a tableau, whilst explaining what each step is doing to improve the final solution, a concept students struggle with. This is shown by Student 2's response, 'You need a very clear understanding of its [Simplex algorithm] functionality to be able to answer the questions.' The subsections of this criterion closely follow the steps identified in the primary research, suggesting students will benefit from the Simplex method being split up this way.</p>	<p>they would otherwise lose regarding the functionality of the Simplex algorithm. By explaining each step, students will be able to answer the parts that follow the application questions, which will test how much they know about the functionality of the algorithm. The steps of the Simplex algorithm are also difficult for [SCHOOL NAME ABBREVIATION] students to visualise using existing systems such as the textbook, therefore including a feature that walks through each step will allow students to understand what each iteration is doing.</p>
8) My system must allow students to enter their	Question 11 of the interview suggested that	Allowing students to enter their own

<p>own inequalities into a table to be solved using the simplex method, with a maximum of 4 non-basic variables.</p> <p>8.1) Students must be able to view solutions to their question with clear method shown.</p> <p>8.2) The system must also allow the user to clear any inputs entered into the table.</p> <p>8.3) Identify the most negative value in the objective row to identify the pivot column</p> <p>8.4) Calculate the theta values by dividing the term in the value column by the term in the pivot column for all constraint rows</p> <p>8.5) Complete the following and add to a new tableau. Divide every term in the pivot row by the pivot value and write down the row operation. Change the basic variable at the start of the row to the variable heading of the pivot column.</p>	<p>students value user interactivity, with Student 1 mentioning 'VoGA was more user-friendly... having user interactivity'. The steps defined in the success criteria from 8.3 to 8.7 follow those outlined in the specification section of primary research. These steps also conform to those from the Pearson Edexcel Decision Maths 1 textbook; hence they will be relevant for students using the system to consolidate their knowledge of the topic. Student 2's response to question 10 of the interview also highlighted that an interactive system would be beneficial for him through to year 13. He stated that his 'main form of revision is quite time-consuming' and that 'using a visual learning tool would help me revise specific topics that I find difficult.' By implementing a feature</p>	<p>questions and providing solutions means they can practice more and receive explanations for questions they may be struggling on. Students will also benefit from an interactive element of the system, as shown by question 7 of the interview, where Student 2 stated that a visual and interactive tool would 'refine [his] understanding'. Seeing the algorithm in action on user-inputted values will allow students to retain their knowledge, as they will be aware of all the detailed steps they may forget as they learn new content for further maths and other subjects through to year 13. Presenting Student 3 with the 'VoGA' existing system in question 11 led to this response, where they mentioned that the "create a graph" feature useful, as I could enter my own problem into the system and perform an algorithm on it. This helped my further develop my understanding of the</p>
--	--	---

<p>8.6) Subtract different quantities of the '1' of the pivot value in the pivot row. Perform the same operation on the remaining values of the row, using the old values for the current row and the new value from the pivot row.</p> <p>8.7) Repeat steps 8.3 to 8.6 until there are no negative numbers in the objective row, before outputting the final values of each variable and the value of the objective function.</p>	<p>that allows users to just enter their own Simplex question and see full working and explanations, students can save time in year 13 as they will be able to frequently and effectively remind themselves of content they may have forgotten in a manner that is less time-consuming than using their current resources.</p>	<p>topic.' This suggests that interactivity element of this existing system aided Student 3's understanding and a similar feature for another challenging topic such as Simplex would have a similar effect.</p>
<p>9) My system must have an introduction each topic to the user. This will be included as a sub-section when a specific topic is selected by the user.</p> <p>9.1) The introduction must explain what the algorithm does and give an example of a real-life scenario where it would be used.</p>	<p>Question 11 of the interview highlighted the importance of introducing the topic to the user. Student 1 mentioned he likes the introduction that both the provided systems presented, 'especially 'VoGA' as it gives real world scenarios in which the algorithm may be used'. Question 6 of the interview suggested that understanding the steps algorithm is a key part of being able to perform well in exams, with</p>	<p>Students would be more aware of the possible applications of an algorithm, resulting in them being able to apply their knowledge to more obscure questions. The introduction to a topic also builds the student's understanding to help them answer questions testing their ability to explain the rationale behind each step they perform. A system like GeoGebra, which lacks an introduction, results in a one-dimensional</p>

	<p>Student 1 mentioning 'without a solid understanding... I cannot ensure I answer each question with confidence.' From the existing systems I researched, GeoGebra is the only system lacking any form of introduction to the topic, resulting in students only learning the steps of the algorithm.</p>	<p>understanding of the topic, meaning it can be harder to apply it to the more 'obscure questions' that students often struggle with, as mentioned in the interviews.</p>
10) My system must explain the steps of the various algorithms in accordance with the Edexcel A-Level Further Maths specification.	<p>The main drawback of some existing systems is that they are not tailored specifically to Edexcel Further Maths, resulting in gaps in knowledge. This is solidified by question 11 of the interview, where Student 2 mentioned the merit of MyMaths being a 'better tool for developing exam knowledge since it is tailored specifically towards A-Level Further Maths'. Of the existing systems, only 'VoGA' wasn't tailored towards A-Level Further Maths students which resulted in occasional gaps in knowledge or extra information that students deemed unnecessary. In the specification section</p>	<p>[SCHOOL NAME ABBREVIATION] students taking Edexcel Further maths can benefit from understanding the different algorithms tailored to their specification. While 'VoGA' was a more well-rounded system, the main issue students faced was the broader content that doesn't help them. By ensuring knowledge is specific for the Edexcel A-Level, students will benefit from a tailored, well-rounded system.</p>

	<p>of primary research, I have outlined specific steps and these follow those provided by the Pearson Edexcel Decision Maths 1 textbook.</p>	
<p>11) My system will utilise past exam questions from the Edexcel A-Level Further Maths exam to form the fixed examples for students to learn from. This will include:</p> <p>11.1) Displaying the past exam question, what year it is from and how many marks it is worth.</p> <p>11.2) Working through the question, providing methods and explanation for each step.</p> <p>11.3) Outputting a final answer to the question.</p>	<p>All three students agreed that past exam questions were the most successful method of revision for their decision maths exam in question 3 of the interview. When asked about the effectiveness of their current resources in question 4, Student 2 mentioned exam questions provide him with 'exposure to the more abstract questions that can be asked'. To ensure students learn the topic in accordance with the type of questions they face in exams, I will include past questions for students to learn from. A major benefit of Pearson ActiveLearn as an existing system was the fact that a large majority of the practice questions are taken from past papers, allowing students to learn and</p>	<p>Students can see how to lay out their working for an exam question that will closely imitate the questions they will encounter in their final exam. By adding in explanations for each step of the question, students will also be able to understand how the answer to the question was reached. In the Edexcel A-Level Further Maths mark schemes, explanation is often not given for how the answer was reached, which means while students can see how to get to the answer, they may not be able to apply this reasoning to other questions, a difficulty highlighted by Student 1 in question 8 of the interview 'applying and understanding what each algorithm does is the difficulty'.</p>

	apply their knowledge in very similar questions to what they may face.	
<p>12) My system's user interface must be clearly split into the different features of each subsection. This will include:</p> <p>12.1) Displaying the question e.g. a graph to perform an algorithm on</p> <p>12.2) Alongside the question, displaying fully worked solutions to the question.</p> <p>12.3) Providing an explanation/description for each step of the worked method.</p> <p>12.4) Allowing users to generate random values for a graph and select an algorithm to run, providing descriptions for each stage.</p> <p>12.5) Allowing users to enter their own graph into the system, select an algorithm and provide full solutions with explanation.</p> <p>12.6) Allowing users to enter their own values</p>	<p>[SCHOOL NAME ABBREVIATION] students often find it difficult to visualise what an algorithm is doing and find some existing resources to lack proper explanation. This is underlined in question 4 of the interview, where Student 1 responded saying the textbook he uses to revise 'does lack a solid explanation, and therefore is not an effective resource'. By combining a visual element with detailed reasoning, students will have a better understanding of a topic. From the research of existing systems, the layout of 'VoGA' was highly intuitive, splitting up different user options into separate sections to avoid information being cramped where it can be missed, as evident from Student 1's answer and Student 3's answer from question 11 of the interview.</p>	<p>Students will be able to clearly picture each step of an algorithm and understand what is happening using the descriptions provided. This will allow them to gain more marks in an exam, especially on the questions that 'test if [they] can... explain a specific step of the process.' Ensuring that content is split up intuitively and presented in an easy-to-understand manner will be beneficial for students as they are more likely to take in information effectively and are less likely to skim over/ not fully understand vital parts of an explanation that would be helpful in an exam.</p>

into a Simplex tableau to then be iterated on, with full working and justification for the steps provided.		
--	--	--

## Computational Methods

Having outlined the success criteria for my project, I can apply various computational methods that will enhance the end-user experience and benefit the development process of the system. Abstraction is the process of removing unnecessary detail from a problem to ensure that solution remains relevant to the initial, basic problem. Decomposition involves splitting up a problem into smaller manageable modules, both in the actual development of the solution e.g. splitting an algorithm up into subroutines, or in the system as a whole e.g. splitting up a topic into various user options like viewing an example, randomly generating a graph. Object oriented programming is a programming paradigm that is built on objects formed from classes which have attributes and methods. The other computational methods are predominantly different forms of computational thinking, which involve identifying preconditions, reuseable components and what decisions need to be made. I will now explain how each computational method will be applied in my system to aid the stakeholders using the system and me while developing the system.

Abstraction will be used in my system as the topics I aim to implement use simplified versions of real-life situations, such as graphs and networks using nodes to represent houses and the arcs forming road routes between them. This will simplify the different topics for users to learn, while replicating questions in a similar format to an exam. I will further use abstraction to ensure the content provided to students predominantly remains specific to the Edexcel A-Level specification, except for the introductory area, which will provide students with a basic idea of the practical applications of the algorithm. This will include removing any unnecessary details that are not relevant to the Edexcel A-Level, such as some aspects of the 'VoGA' existing system.

Decomposition will be used at various stages throughout my system as different algorithms will be split up into steps which can be intuitively developed as separate methods, which will also aid the explanation aspect of the system. I will split up each topic into various sections e.g. Description of algorithm, generating a random graph, allowing users to enter their own graph. I will further use decomposition to split up algorithms into specific steps which will make a topic easier for students to understand. For Kruskal's algorithm, I will first sort the arcs into ascending weight order, then begin adding and rejecting them to the tree, before finally outputting the final tree. By splitting up the algorithm in this manner, students can see the individual steps they need to show in an exam. I will also split up Prim's algorithm into smaller steps, first choosing a starting arc, then checking for the smallest weight arc adjacent to a vertex that already exists in the tree, connecting it to a vertex that is not currently included. After this my system will output the final minimum spanning tree. The Simplex algorithm will also be split up into its various steps to ensure students are aware of the functionality of the algorithm. This topic will be split up into generating inequalities, creating an initial tableau, finding a pivot value, and completing iterations until the optimal solution is found.

I will use Object-Oriented Programming (OOP) throughout my system, specifically in the graph generation section. This could use a main graph class with various methods that include adding/removing vertices and edges. I will also include methods to add a weight to each arc, as this will be vital to follow the style of graph used in Edexcel A-Level Further maths, and to better explain the graph algorithms. Since the graph algorithms I aim to implement are used to create minimum spanning trees, I could create a tree class which inherits from the graph class. In the tree class, I will have unique methods, such as adding the next suitable arc with the smallest weight. This will have to follow the rules of creating a tree e.g. no cycles, so I will utilise static polymorphism with method overloading using a method that additionally checks for cycles when adding an arc, as the original method would not regard cycles as an issue when creating the original graph. I will further use OOP for the Simplex algorithm to create tables, with user-defined parameters such as number of variables and inequalities. This will be beneficial in the interactive feature that allows users to enter their own

inequalities, as the tables will need to have variable sizes to account for different numbers of inequalities and unknowns that the user wants to add. Using OOP will allow the same sections of code to be reused to create multiple instances of a class.

Another computational method I will use is thinking procedurally, a key component for implementing algorithms. An example of this is in Kruskal's algorithm, where arcs must be sorted first before they can be added and rejected, then it must be checked that all vertices are contained within the spanning tree before the final MST can be outputted. This will also be used in Prim's algorithm, as choosing the next edge will first require checking that one vertex of the edge is already in the MST, and the other vertex is not. This ensures that adding the edge will expand the MST without creating a cycle. Procedural thinking will be vital for Prim's algorithm on a distance matrix as well, as the method will have to ensure that a certain row has been crossed out/ disregarded when choosing the next arc to add to the MST to avoid the formation of cycles. Another example of thinking procedurally will be following the steps of the Simplex algorithm in order. This will involve identifying the most negative value in the objective row before identifying a theta value, identifying the pivot value before beginning the iteration, checking the values of variables in the objective row to ensure they are all positive before outputting the final values. Thinking procedurally will be vital for the system as it will aid the step-by-step explanations, ensuring that each part of the process is displayed to users in the correct order.

I will also use thinking ahead in my system, managing user inputs such as number of variables and inequalities in the Simplex algorithm. Taking these user inputs will allow for easier creation of arrays as the dimensions required for the Simplex tableaux can be decided through the inputted values e.g. a table for the standard Simplex algorithm with 3 inequalities will require 4 rows. Using thinking ahead, I will also utilise reusable components in my system, specifically for the graph-based algorithms. The same section of code that is used to generate graphs can be used to perform both Kruskal's and Prim's algorithm, as the initial graph generated does not affect the steps of the algorithm. All the vertices, edges and weights generated will have the same format, hence I will be able to reuse the methods to generate these throughout the software. I will use the outputs from the method that

performs the algorithms and incorporate them into the explanation, using the weight and edge chosen from the graph as outputs and using them in the explained solution, for example “The edge ‘CD’ with weight ‘7’ was rejected as it forms a cycle”.

Visualisation will be a vital feature in my system, assisting students with their understanding as they can picture how a specific algorithm works. Visualisation can also help reduce an excessive amount of information from being displayed, which will benefit users by ensuring that the system does not have a ‘cramped layout.’ The main areas for me to use visualisation will be for the graph algorithms, as it can be difficult to picture what the algorithm is doing without seeing a visible representation of it. Using visualisation, I will be able to clarify individual steps, for example, highlighting an edge from the displayed graph when it is added to the minimum spanning tree. Being able to present information in a way such that students can see the individual steps being performed will benefit visual learners, who may struggle to understand each progression in the algorithm using existing systems such as the Edexcel A-Level Decision Maths 1 textbook. Graphs will be stored using arrays or adjacency lists (storing a node and any adjacent nodes) but presented to the user as a series of nodes connected by edges or as a distance matrix. The Simplex tableaux will be stored as 2D arrays, with dimensions which will be specified through altered user inputs but presented as a visual table on the GUI.

My system will use decisions and logic both to implement and explain the algorithms to users. When performing Kruskal’s algorithm, logic must first be used to sort each edge into ascending weight order. Next, a decision must be made when adding/rejecting an arc from the spanning tree; if the edge would form a cycle when added to the spanning tree, it must be rejected. My system will further utilise decisions/logic for the simplex algorithm. The system must choose first logically add each coefficient from the entered inequalities to a 2D array, accounting for slack variables. Next, a decision must be made to identify the most negative value in the objective row before proceeding to identify the correct pivot value around which the iterations will be performed. Lastly, decision-making must also be incorporated into the final stage of the Simplex algorithm, checking whether to output the current state of the table as complete depending on if all

the values in the objective row have become positive after performing the iterations.

## Stakeholder Uses/Needs

[SCHOOL NAME ABBREVIATION] students currently struggle in developing a fluent understanding of Edexcel A-Level Decision maths algorithms. This issue, combined with the time constraints in exams results in lower marks and grades, since the Decision maths content forms 25% of the entire A-Level grade. A further burden on students is that they do not have frequent decision maths revision lessons through year 13, having to learn the content from the second mandatory module (Core Pure 2) and the other chosen module (Further Mechanics 1). The students in the interviews identified that an interactive and visual system would help them better visualise the algorithms as well as retain knowledge through year 13.

The students will benefit from the introductory section to each topic as it will broaden their understanding of the applications of an algorithm, presenting the topic in a manner that would replicate the context provided in exam questions. The random graph generator will assist students by providing them with exam style questions to work through. The interviewed students identified that past questions were the most effective form of revision to help them during exams, however the limited number of past papers can result in students eventually memorising answers instead of using their knowledge and applying it, which is a key skill for them since the questions in their real A-Level exam will be unique.

The worked solutions/answers will allow students to see what method they need to show when answering a question to ensure they gain all the marks. This, combined with the accompanying descriptions/explanations of the steps of the algorithm will help students develop their understanding and form exam practice simultaneously. Learning the reasoning behind the steps of the algorithm while working through a question will also help students answer the more abstract questions that they identified as an issue in the interviews.

The user graph creation feature in my system will benefit students providing them with an option for learning through user interactivity, as this was a useful feature

identified by my stakeholders when they were presented with 2 of the researched existing systems. Using visualisation accompanied by step-by-step explanation/walkthroughs, students will be able to picture the algorithm being performed (especially beneficial for visual learners who may struggle using existing systems to understand decision maths topics) and improve their ability to answer questions about the algorithm. The walkthrough will also show users what they would need to write down when facing a similar exam question to ensure they gain all the marks.

Using past exam questions as fixed examples will allow the students to learn the algorithm in a context that replicates what they will encounter in their exam. By expanding on the step-by-step guide provided by the Pearson Edexcel Decision Maths 1 textbook, which was said by Student 1 to not 'help very much as being able to apply it to other questions is the important part', and incorporating visualisation into my system, students will be more adept at applying algorithms and will be able to gain more marks in an exam. Including explanations with the working will benefit students, building on the textbook used, about which Student 1 mentioned that 'just following an example is insufficient' during the interview.

My solution will further benefit my stakeholders with the interactive Simplex tableau, which will allow users to enter their own values into an initial tableau, and iterations will be performed until an optimal solution is reached. Before reaching the answer, all the stages of working will be displayed to the user, for example, explaining why a certain value was chosen as the pivot value. This will ensure the user understands what is happening at each stage of the algorithm and would be able to answer a question that test their understanding of the rationale behind the steps they have learnt.

Each feature in my system will be created to conform to requirements identified by my stakeholders from the interviews. By interviewing current [SCHOOL NAME ABBREVIATION] students, any features suggested by them to be valuable can be considered essential in creating a system that is suitable for my stakeholders in learning, understanding, and applying decision maths algorithms. My system will also consider the different levels of initial understanding that different students will have. Although the interviewed students stated that learning and remembering the different steps of algorithms isn't difficult to them, this isn't

representative of all possible stakeholders. By introducing the topic from the beginning and teaching the steps of the algorithm regardless of whether all stakeholders will need it or not, I can ensure that those with a weaker initial understanding of the topic are also able to develop their ability using my system.

## Software and Hardware Requirements

I can now outline the specific software and hardware requirements that will be used in both the development process of the system as well as the end-user experience. For the hardware, I have specified the minimum requirements to run the system that I have discovered from research of the IDE I will use. I can now also identify the specific software, like additional libraries and packages that will help the development process of my system. These will be necessary for me as the developer, as well as users of the system. Each piece of hardware and software provided below has been given justification of why it will be needed for my system.

Some specific values for hardware requirements are taken from JetBrains (the creator of IntelliJ). These can be found on this website:

(<https://www.jetbrains.com/help/idea/installation-guide.html#toolbox>). Some of the other specific values are those recommended for the use of the Java Runtime Environment, however, to account for the combination of other elements users will need, such as additional libraries, the exact minimum values haven't been taken from (<https://docs.oracle.com/javase%2F7%2Fdocs%2Fwebnotes%2Finstall%2F%2F/windows/windows-system-requirements.html#:~:text=Memory%20Requirements,-Processor%20Requirements,Pentium%202%20266%20MHz%20processor>) and rough estimates have been made instead accounting for various factors, to allow the user to smoothly run the system. The minimum hardware requirements for the IDE exceed those of Windows 10, hence the values provided will be the minimum values for the IntelliJ IDE to run.

## Software Requirements to Implement the System

Software	Explanation
Windows 10 64-bit Operating System	To create my system, I will need an operating system that provides a user interface, allowing me to download, view and use other specific application software such as an IDE. The minimum requirements to run my chosen IDE is Microsoft Windows 10 1809 64-bit or later.
Programming Language: Java IDE: IntelliJ IDEA Educational Edition	I will use Java to create my system as it is a pure object-oriented language, and many features of my system will utilise the computational method of OOP. I will use IntelliJ as I am familiar with the user interface, installing libraries and testing code. This IDE also provides many debugging features which will be vital for the iterative development process. Java is also the language I am most familiar with when creating graphical user interfaces.
Java Abstract Window Toolkit (java.awt library)  This is included in the JDK (Java Development Kit) in the IntelliJ IDE.	Java AWT will be used as the main API to develop my GUI as it contains classes and interfaces which are used to create window-based applications and allow event handling.
Java Swing (javax.swing)  This is also included in the JDK in the IntelliJ IDE	Java Swing will be used on top of Java AWT to create the different components for my GUI such as buttons, textfields and menu bars.
JGraphT library  This can be downloaded from <a href="https://jgrapht.org/download">https://jgrapht.org/download</a> and implemented into IntelliJ by adding a library in the 'From Maven' section of Project Structure settings.	The JGraphT library will be used in my system for the modelling of graphs. It is beneficial due to its support for weighted graphs which will be necessary to perform algorithms like Kruskal's and Prim's.
mxGraph library  This can also be downloaded from <a href="https://jgraph.github.io/mxgraph/">https://jgraph.github.io/mxgraph/</a> and	I will use mxGraph in conjunction with JGraphT to display and allow users to interact with graphs. This library will

added to IntelliJ by adding a new library 'From Maven' in the Project Structure settings.	allow for nodes and edges to be customised to fit specific visual requirements which will be necessary in my user graph creation feature.
Java.util package This is also included in the JDK in the IntelliJ IDE	This will be necessary for various sections of my system, such as random number generation in my random graph generator, as well as a Scanner object to take user input for the interactive features.

During the development process, I may require additional libraries/packages, however, the software outlined above are the main requirements to develop my system that I am currently aware of.

## Software Requirements to Run the System

Software	Explanation
Windows 10 64-bit Operating System	As the system is being developed using Windows 10, a similar operating system is recommended for users to ensure they can use the system in the same way as it has with various dependencies such as the external libraries used as part of my GUI.
Java Runtime Environment Can be downloaded from: <a href="https://www.java.com/en/download/manual.jsp">https://www.java.com/en/download/manual.jsp</a>	The system will be created using the JetBrains IDE IntelliJ, to which any external packages and libraries will be added to. To run the system, my stakeholders will need the Java Runtime Environment which will allow them to run the compiled executable file created.
Java Abstract Window Toolkit (java.awt library) Java Swing (javax.swing) JGraphT library mxGraph library	The libraries specified will be used to implement various aspects of the system, hence they will need to be downloaded on the users' device to ensure full functionality of the system.

Java.util package  <a href="https://jgrapht.org/download">https://jgrapht.org/download</a> <a href="https://graph.github.io/mxgraph/">https://graph.github.io/mxgraph/</a>	Java AWT, Swing and mxGraph will be needed to ensure all aspects of the GUI are visible to the user, as the program will not run in their absence. JGraphT will be needed as it is used to create graphs in my system. For the user, it will be used to store their inputted graphs and convert them into a series of nodes and arcs. By creating an executable JAR file that does not bundle dependencies, users will require the libraries on their computer. Java AWT, Swing and Util are included in the JRE, hence will not require additional downloads.
---	--

## Hardware Requirements to Implement the System

Hardware	Explanation
2GB of total system RAM	The minimum RAM required to run IntelliJ is 2GB, however, the recommended amount is 8GB.
A PC with at least x86_64 CPU architecture	The minimum requirement to run IntelliJ is given as a minimum of x86_64 architecture, with the recommendation of a multi-core CPU as the IDE allows for multithreading.
5GB Disk Space	3.5GB is the minimum requirement for disk space to store the IntelliJ IDE. I have chosen 5GB as the requirement to ensure there is sufficient space to store any libraries and to store the system.
1024 x 768 minimum screen resolution	This is the minimum monitor resolution recommended for IntelliJ; however, I will use the recommended resolution of 1920 x 1080 as this is the most common monitor resolution and will allow me to better test the GUI.

Mouse/Trackpad	This will be necessary to send inputs to the PC, such as running the program, adding breakpoints and other parts of testing
Keyboard	This is required to write the code for my system

## Hardware Requirements to Run the System

Hardware	Explanation
1GB of total system RAM	This is the recommended amount of RAM to allow the user to run the JRE and the final application with any external libraries
Minimum 1GB Disk Space	To ensure the user has enough storage for the JRE, additional libraries, and the system itself, the recommended disk space is roughly 1GB. The minimum value provided for JRE is 300MB, so to account for all necessary libraries and the final application, 1GB of disk space is recommended for users. Having enough disk space to exceed this minimum requirement will allow smooth running of the system for users.
A 2-core CPU or better	The system does not contain any heavy graphics, hence for my stakeholders to run the system, the minimum requirement will a 2-core CPU to handle the JRE, and the application with additional libraries
Minimum of 1024 x 768 screen resolution (recommended 1920 x 1080)	This will be required to allow users to interact with the GUI, ensuring that all content is displayed clearly and all window sizes fit the user's monitor/display.

Mouse	This will be required to allow users to select from the various options on my GUI, such as running an algorithm or creating a graph.
Keyboard	This will be necessary to allow users to input values into my system, such as the weights for edges on a graph, or to enter the coefficients for variables in the Simplex algorithm.

## Design

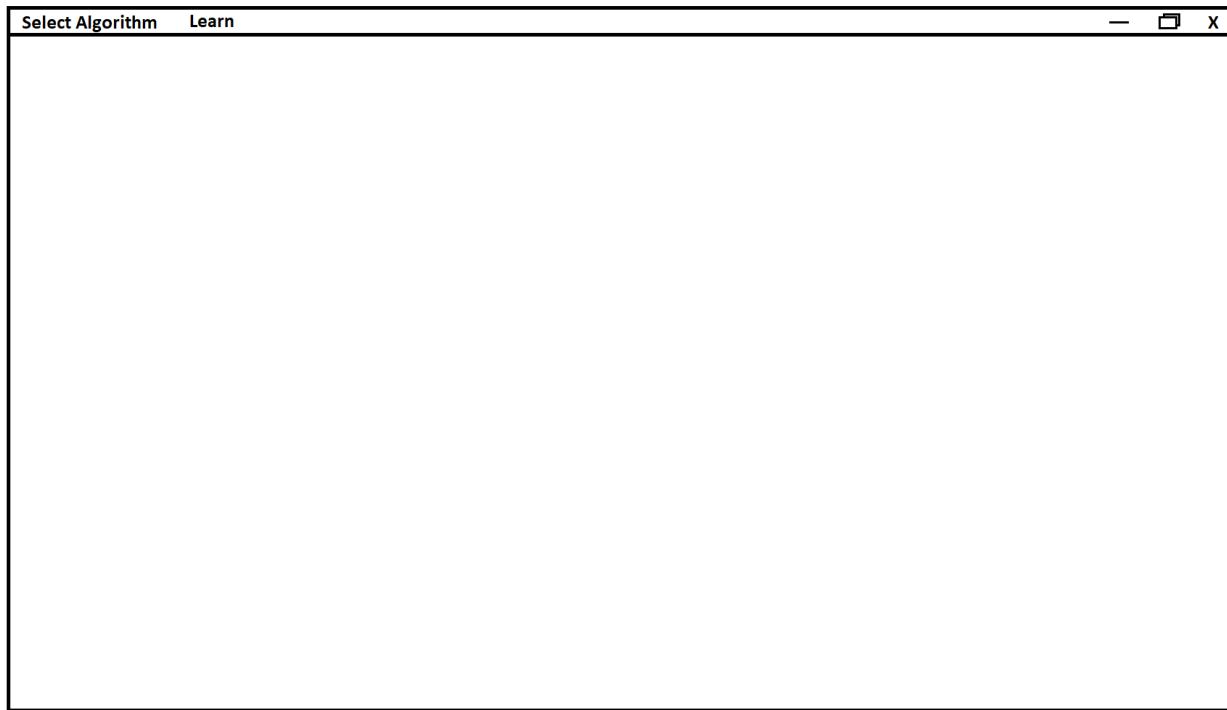
### Iteration 1: GUI Mockup

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

The first iteration of the project is a prototype GUI, forming a ‘shell’ for future functions to be added to each object, such as menu items and buttons. This achieves success criterion 1.0 and 1.1. By developing a shell for the GUI, I will be able to judge the layout for the user interface features of the algorithms when it comes to converting the algorithm in the IDE terminal to an algorithm that works in the GUI. The structure of the GUI is also likely to change throughout the development process as each algorithm and its options are added, so having an initial window to work on and alter will help ensure that changes can be easily made by altering existing features. This will also lay a foundation for success criterion 12, which outlines splitting the user interface into clear separated sections that allow the user to identify which tool they want to select and how to do so. Before I aim to implement GUI based features such as the visual graph creator, it will be important to have a basic window upon which these components can be added. This justifies my first iteration being a prototype GUI.

Since success criteria 1 and 12 highlight the importance of having the GUI split up into clear sections, I decided that having a menu bar at the top of the GUI, which can have each item split into sub-options would be the best way of ensuring the various features can be intuitively added to the user interface without taking up too much room and ensuring that the window remains clearly laid out.

Below is a rough image of what the GUI should look like:



From success criterion 1.0, adding a menu bar with clearly labelled items will ensure students gain the benefit of a well laid out GUI. This also links into criterion 1.1, making sure that each algorithm option is split into their specific sub-sections to avoid information being crammed onto one page.

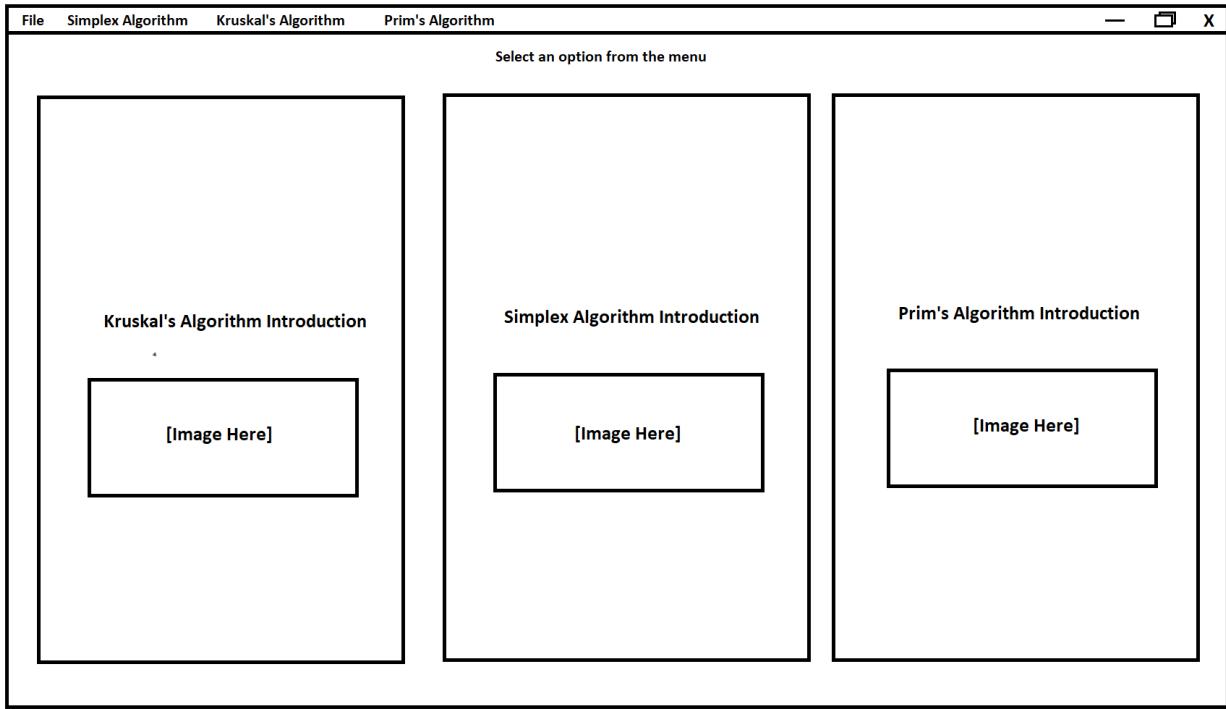
At this point, I decided to consult my stakeholders on what should be presented to users when the original window is created.

Question	Student 1	Student 2	Student 3
What features of the GUI mockup above do you think are beneficial and	I think that the menu bar at the top is a good feature since it simplifies the	I think that having the menu bar at the top is good, but since you mentioned having	I think that as a shell for the user interface this is good, but the main section could

should be maintained, and what features do you think could be improved or added?	different functions to be kept under one name and then this option can contain other sub-options. The main part of the GUI seems a bit empty and could have a feature to prompt the user about how to use the system.	different options for each algorithm I think that the 'Select Algorithm' option will get quite crowded since I'd have to first click the menu option, then select the algorithm, then select the option inside the algorithm. I think it would be better to have a menu item for each algorithm.	have another component, so it doesn't feel as empty. I also like the menu bar, but I think that the 'Select Algorithm' feature could be split up into each algorithm, then those could each have their own sub-options, so the user doesn't have to do as much to get to the desired tool.
--	---	--	--

Having consulted my stakeholders throughout the development process, they expressed that having a feature on the main 'home' screen would be beneficial in filling more of the space. So, I decided to add a button that can be pressed to introduce users to each algorithm, instead of a blank initial window, as having clear indication of the features upon opening the window would be more user-friendly, as the options are instantly presented when the window is generated. They also mentioned that the 'Select Algorithm' feature could be split up into individual algorithms, so I decided that there would be 4 menu items, File, Simplex Algorithm, Kruskal's Algorithm and Prim's Algorithm. Each of these can then contain sub-options to run the algorithm and learn the algorithm and the introduction which was highlighted as a key feature can be implemented in the introduction buttons on the user interface.

What the GUI should look like after consulting stakeholders:



The 'File' menu item contains options 'Home' (reverting to the initial window presented when the program is run), as well as 'Exit', which closes the window. Each of the algorithm menu items currently contain two options each: one to run the algorithm and one to learn the algorithm. With the interviews in analysis highlighting the importance of splitting up a topic into various options, there may be more options added for each menu item in future iterations.

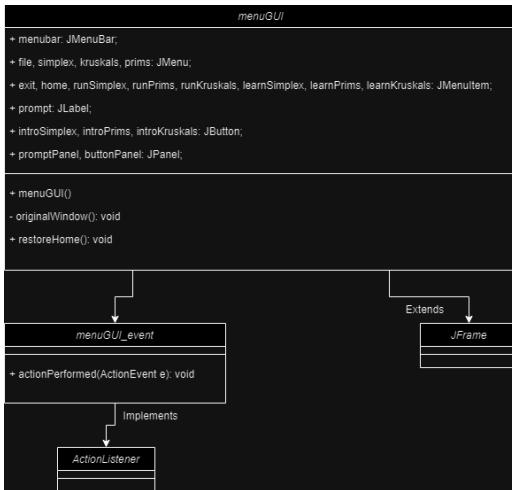
The menu bar from the initial image has been maintained; the main new feature being the three buttons that have been added to the window. The buttons have each been labelled with their purpose as well as an image representation of each algorithm, which was mentioned by the stakeholders. The spacing of each of the buttons has also been deliberately set as it is to ensure that the guidance text isn't overshadowed by the buttons as this would make it less impactful. A small gap has also been left between each button to ensure users are certain on which button is being pressed.

I presented this new mockup to my stakeholders for their opinions:

Question	Student 1	Student 2	Student 3
Do you think this new GUI mockup	I think that splitting up the	I like how the menu bar is split	I think that the introduction

<p>serves you better as a user of the system.</p>	<p>menu bar has really helped as I can now clearly see what I need to press to reach the desired function. For example, I know to press Simplex Algorithm if I want to run the algorithm or learn about it. I think having the buttons for the introduction is also a good idea. It fills the space with something informative and the GUI doesn't seem as empty any more.</p>	<p>up into separate options now. The 'File' option clearly suggests that it contains more of the general features like returning to the home page or closing the window, and having each algorithm as a menu item makes it easier for me to navigate the various tools. The buttons in the main part of the window are also good at ensuring I can instantly begin using some of the features without having to figure out where to navigate on the user interface.</p>	<p>buttons are a good addition to the system since they clearly indicate how to access the introduction to each algorithm. I also like how the menu bar is now split up into more options. Previously, it seemed like the features were too crammed under a single button that served too many purposes. Having a variety of menu items splits up this load across different buttons and more clearly indicates the purpose of each tool.</p>
---	--	---	---

Below is a UML Class Diagram for iteration 1:



This shows the relationship between the classes I will develop, as well as the classes from the libraries I aim to use. Inside each class, I have also shown the attributes and methods involved, as well as whether they are publicly or privately encapsulated. The use of this UML Class Diagram will allow for a smoother development process as the problem has been decomposed into separate modules. As shown, the main class for this iteration is menuGUI, which will contain the various methods to add components to the screen, and handle any events being processed, such as a menu item being selected

### Justification of Classes

Class Name	Justification/Purpose
menuGUI	Defines each class attribute, containing a constructor to create the original window and methods to restore the window to its original state
event	Contains the method to handle events resulting user interacting with the components of the GUI.

### Justification of Class Attributes

Attribute	Justification/Purpose
menubar	A JMenuBar at the top of the window which can have various options/menu

	items added to it
file	A JMenuItem that will manage the window functions
simplex	A JMenuItem that will contain the options for the Simplex Algorithm
kruskals	A JMenuItem containing the options for Kruskal's Algorithm
prims	A JMenuItem that contains the options for Prim's Algorithm
exit	A JMenuItem containing the option to close the window
home	A JMenuItem that will restore the original state of the window when pressed
runSimplex	A JMenuItem which will clear the window, presenting the feature to run the Simplex algorithm
runPrims	A JMenuItem that will clear the window and present the feature to run Prim's algorithm
runKruskals	A JMenuItem which will clear the window and present the feature to run Kruskal's algorithm
learnSimplex	A JMenuItem that will clear the window and present the user with the option to learn how to perform the Simplex algorithm
learnPrims	A menu item which will clear the window, allowing the user to interact with the learning feature for Prim's algorithm
learnKruskals	A JMenuItem that will clear the window and allow the user to interact with the learning feature for Kruskal's algorithm
prompt	A JLabel that will appear on the window, prompting users to select a menu option
introSimplex	A JButton that when pressed, will clear the window and present users with an

	introduction to the Simplex algorithm
introPrims	A JButton that will clear the window and present users with an introduction to Prim's algorithm when pressed.
introKruskals	A JButton that will clear the current window when pressed, presenting users with an introduction to Kruskal's algorithm.
promptPanel	A JPanel at the top of the window, below the menu bar that will contain the prompt label.
buttonPanel	A JPanel in the center of the window that will contain 3 buttons.

### Justification of Methods

Method	Justification/Purpose
menuGUI()	This is the constructor method used to create the initial window, declaring the layout, size and names of any objects to be added.
originalWindow()	This method was added to allow the 'home' button in the option for the menu bar item 'File' to return the window to its original state as is presented when the GUI is first created. This creates the panels for the center of the window and each of the buttons as well as their details like size, layout, names and icons.
restoreHome()	This method was created to restore the original state of the window when the 'home' button in the 'File' section is pressed.
actionPerformed()	This predefined method takes an event as a parameter, checking the source of the event being called before carrying

	out the necessary actions, such as closing the window down.
main()	This sets the details of the window, such as the size, visibility, as well as instantiating a menuGUI object.

## Test Plan

I have created the following test plan, showing a description of the test, and the expected outcome for iteration 1 to ensure that each component of the GUI mockup is functioning as intended. For a GUI, the main test cases are to ensure that each component appears on the GUI as intended when the program is run, as well as ensuring that any components that perform an action based on user inputs, for example pressing a button, or selecting a menu item, also correctly perform the action upon receiving the input. This test plan incorporates these two areas of testing, first ensuring that the basic functions of the window work as intended e.g. the window being the correct size and having the correct displayed components and then checking that when a menu item is selected for example, is the correct action performed. By making sure that each component works as intended, with a basic function such as clearing the window, I can lay a foundation for each menu item or button, so that they can be assigned to more complex functions in future iterations. Ensuring success with basic actions is important in creating a base upon which future functions can be developed.

Test Number	Test Type	Expected Outcome
1) The window of the GUI opening when running the program	Normal	The GUI window opening once the program is run with the correct dimensions and being visible
2) The Simplex Algorithm Introduction	Normal	When pressing the 'Simplex Introduction' button, it should clear the window for information to be presented. Since no

button on the 'home' screen should clear the window when clicked, allowing information to be presented, introducing the Simplex algorithm.		information has currently been added, the window should just be cleared.
3) The Prim's Algorithm Introduction button on the 'home' screen should clear the window when clicked, allowing information to be presented, introducing Prim's algorithm.	Normal	The Prim's introduction button should clear the window for information to be presented. Since no information has currently been added, the window should just be cleared.
4) The Kruskal's Algorithm Introduction button on the 'home' screen should clear the window when clicked, allowing information to be presented, introducing Kruskal's algorithm.	Normal	The Kruskal's introduction button should clear the window for information to be presented. Since no information has currently been added, the window should just be cleared.
5) The menu items added to each	Normal	When clicking each option in the menu bar, the added menu items should be displayed. This

menu bar option should be displayed when selected.		includes 'Home', 'Exit', 'Run Algorithm', 'Learn Algorithm'.
6) When the user clicks the menu item inside of an option, the window should clear allowing room for information to be presented. This test is applicable for: -Run Algorithm -Learn Algorithm In each of the Algorithm menu bar options.	Normal	When clicking each menu item inside of an option, the window should clear allowing room for information to be presented. Since no information has currently been added, the window should just be cleared.
7) When the user selects the 'Exit' menu item inside the 'File' option, the window should close.	Normal	When selecting the 'Exit' menu item inside the 'File' option, the window should close down.
8) When the user selects the 'Home' menu item inside the 'File' option, the window should revert to its original state from any other state	Normal	When selecting the 'Home' menu item inside the 'File' option, the window should revert to its original state from any other state. This allows the user to return to the 'Home' page after using a tool so they can navigate to another tool.

Having outlined the design choices for this iteration, how the development of various methods will take place and the suitable tests to ensure that the developed methods work as expected, the design for Iteration 1 is complete.

## Iteration 2: Simplex Algorithm

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

As the Simplex algorithm is the most complex algorithm that I aim to implement, creating a base version of the algorithm to run in the IDE terminal is important to ensure that all the maths behind the algorithm is correct, before integrating it into the GUI. This will begin success criteria 8.0 to 8.7, allowing user inputs for the Simplex algorithm. As mentioned in the Computational Methods section of analysis, I am going to use an object-oriented approach, splitting up the problem into various classes which will be outlined below.

Before programming the solution, it is important to consider how the program can ensure it is adapted to varying numbers of inputted constraints and variables. As mentioned in the analysis section, the Edexcel specification requires students to be able to perform the Simplex algorithm with up to 4 variables. Each problem can also have varying numbers of constraints, for example 2 variables can have 3 constraints, while a question with 3 variables may only have 2 constraints. All possible questions need to be accounted for, hence an array with a fixed, declared size is not reasonable for this. Having compared the size of initial tableaus to the number of variables and inequality constraints, I discovered the following:

This question has 2 variables and 2 inequalities. Having 2 inequalities means that there will be a total of 2 added slack variables, one per inequality. In a simplex tableau, each variable is given its own column and the value is given its own column as well.

The initial tableau for this problem looks like this



Solve the linear programming problem in Example 6 using simplex tableaux.

Maximise  $P = 3x + 2y$

subject to:

$$5x + 7y + r = 70$$

$$10x + 3y + s = 60$$

$$x, y, r, s \geq 0$$

**Notation** The word **tableau** is French. The plural of tableau is **tableaux**.

Basic variable	x	y	r	s	Value
r	5	7	1	0	70
s	10	3	0	1	60
P	-3	-2	0	0	0

Omitting the column headers and the basic variable column, we have a 2d array of dimensions [3][5].

The number of rows is equal to the number of inequalities + 1, because there is 1 row for each slack variable and one for the objective function. The number of columns is equal to the number of variables + the number of inequalities + 1.

This is because there is one column for each variable, one for each slack variable and one for the value.

This allows us to generalise the dimensions for the initial tableau as array[numInequalities + 1][numInequalities + numVariables + 1], which is adaptable to any combination of inequalities and variables inputted by the user.

For this prototype, the input will be registered in the terminal, by prompting the users to first enter the number of variables, then the number of constraints. Once the number of variables and constraints have been registered, the program will allow the user to enter their coefficients until they have entered n sets of coefficients where n is the number of inequalities. The number of variables is also important as it will distinguish the point between coefficients of variables and the value (example shown below for better explanation):

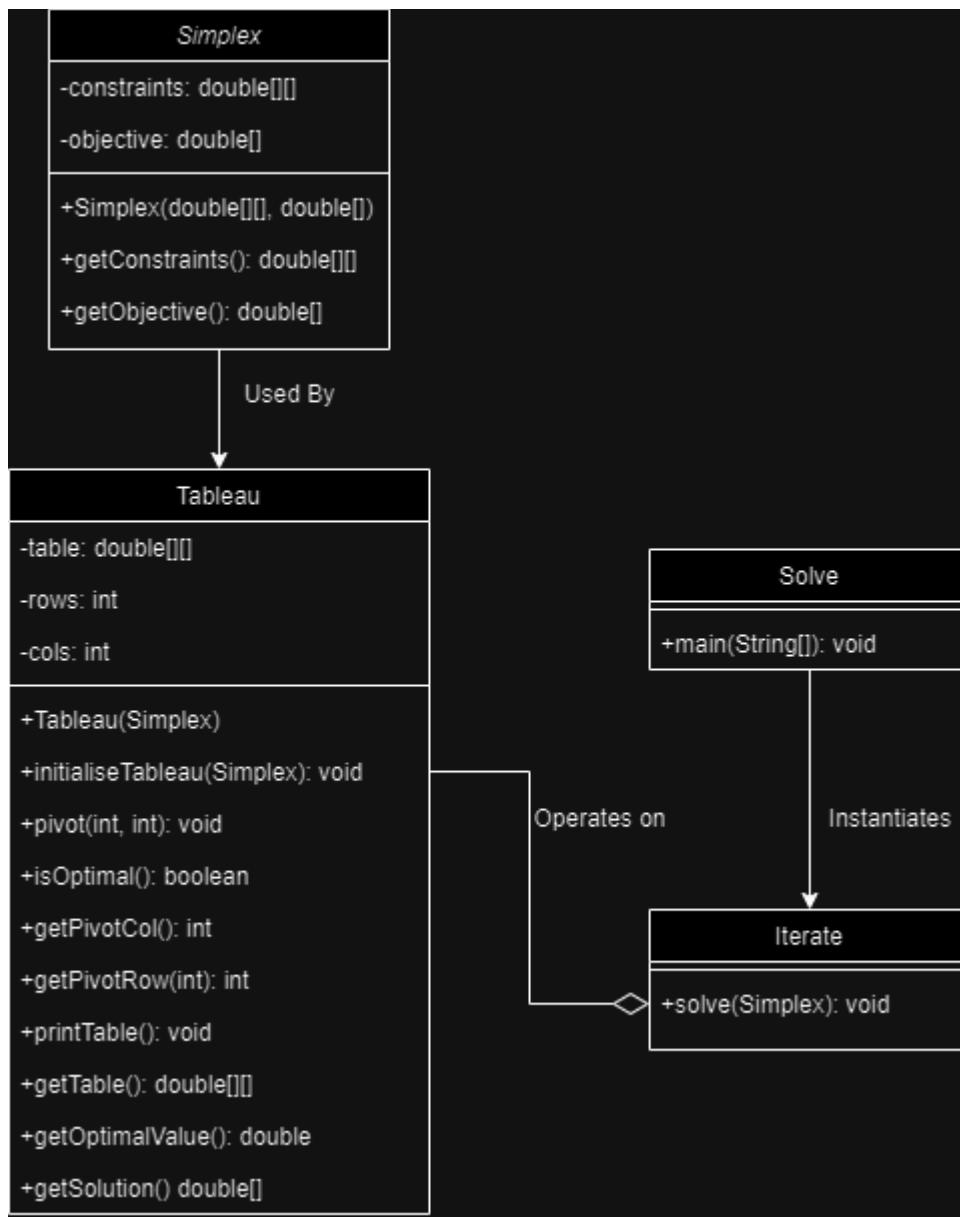
When the user enters:

2 3 5

3 5 8

and they previously entered the number of variables as 2, then the program will know that 2 and 3 are the coefficients of the variable and 5 is the value on the right-hand side. If the user had entered the number of inequalities as 2, then the program would know that after the user enters the second line, all inequalities have been entered and the prompt for the objective function can be outputted.

Below is a UML diagram for the Simplex algorithm:



Using object-oriented programming, I have split the simplex algorithm into 4 main classes. The first is a blueprint for a simplex problem, defining constraints and an objective function. The second is to create the tableau and all its functions, like finding a pivot value and iterating over the table. I created a third class with the sole purpose of repeatedly carrying out iterations of the problem until an optimal solution is found. The last class contains the main method, taking in user inputs and instantiating a new object of type Simplex. The arrows in the class diagram show the relationship between each class.

## Justification of Classes

Class Name	Justification/Purpose
Simplex	Defines the blueprint to create a simplex problem object with constraints and an objective function
Tableau	Creates the initial tableau using the constraints and objective function. Creates the methods for pivoting and finding the optimal value
Iterate	Pivots around the possible solutions until an optimal solution is found, using the methods from Tableau()
Solve	Contains the main method, creating an instance of the Simplex problem object and outputting the optimal solution

## Justification of Class Attributes

Attribute	Justification/Purpose
constraints	This is a 2d array that stores the coefficients of the variables as well as the value of the inequality
objective	Array that stores the coefficients of the variables of the objective function
table	A 2d array that stores the coefficients of the variables, slack variables, values and objective function to be iterated upon
rows	Integer that stores the number of rows in the table to be used as a parameter for the table's dimensions
cols	Integer that stores the number of columns in the table to be used as parameters for the table's dimensions

## Justification of Other Variables

Variable name	Justification/Purpose
numConstraints	Stores the number of constraints entered by the user, using the getter method from the Simplex class
numVar	Stores the number of variables entered by the user, using the getter method from the Simplex class
pivotValue	Stores the pivot value of the table which is indicated by the overlapping index in the pivot row and pivot column.
minValue	Integer that takes the value of the smallest number in the objective row, used to find the pivot column
minRatio	Finds the pivot value by taking the value of the smallest result when dividing the value column number by the pivot column number
solution	Stores the optimal solution
isBasic	Boolean variable to check whether a value in the table is a basic variable or not

## Justification of Methods

Method	Justification/Purpose
Simplex()	Constructor for the Simplex class, initialising a simplex problem with constraints and objective function
getConstraints()	Get method for the constraints which can be accessed by any other class
getObjective()	Get method for objective function which can be accessed by any other class

Tableau()	Constructor for the Tableau class, getting the number of constraints and variables and initialising an empty 2d array
initialiseTableau()	Assigns the coefficients from the constraints to the table created in the constructor. Adds the slack variables in the correct place.
pivot()	Iterates through the array, dividing each value in the pivot row by the pivot value and reducing every other value in the pivot column to 0.
isOptimal()	Checks to see if an optimal solution has been found by iterating through the objective row
getPivotCol()	Gets the pivot column by iterating through the objective row
getPivotRow()	Finds the pivot row by dividing the value column by the pivot column
printTable()	Outputs each value in the table
getTable()	Get method which returns the table array
getOptimalValue()	Returns the value in the bottom right index of the array, which is the value of the objective.
getSolution()	Checks whether a variable is a basic variable or not, if not, then it outputs the variable
main()	Prompts the user to enter their simplex problem and creates an object of type Simplex.
solve()	Iterates over the table until an optimal solution has been found

### Pseudocode for Simplex algorithm

I have written the pseudocode below to outline the rough contents of each class and method to be involved in the code for the simplex algorithm.

Class Simplex:

    Private Double[][] constraints

    Private Double[] objective

Constructor Simplex(constraintList,objectiveFunction):

    constraints -> constraintList

    objective -> objectiveFunction

Method getConstraints():

    return constraints

Method getObjective():

    return objective

Class Tableau:

    Private double[][] table

    Private int rows

    Private int cols

Method Tableau(problem): //constructor to create an empty array for an initial tableau

    numConstraints -> problem.getConstraints().length

    numVar -> problem.getObjective().length

```
rows -> numConstraints + 1  
cols -> numVar + numConstraints + 1  
table -> doubleArray[rows][cols]
```

```
initialiseTableau(problem)
```

```
Method initialiseTableau(Simplex problem)  
constraints -> problem.getConstraints()  
objective -> problem.getObjective()
```

```
for i -> 0 to constraints.length:  
    for j -> 0 to constraints[i].length -1:  
        table[i][j] -> constraints[i][j]
```

```
table[i][cols - 1] -> constraints[i][constraints[i].length -1]:
```

```
for j -> 0 to objective.length:  
    table[rows - 1][j] = -1 * objective[j]
```

```
Method pivot(pivotRow, pivotCol)  
pivotValue -> table[pivotRow][pivotCol]
```

for j -> 0 to cols:

    table[pivotRow][j] /= pivotValue

for i -> 0 to rows:

    if i NOT = pivotRow

        factor -> table[i][pivotCol]

        for j -> 0 to cols:

            table[i][j] -= factor \* table[pivotRow][j]

Method isOptimal:

for c -> 0 tools - 1:

    if table[rows-1][c] < 0

        return false

    return true

Method getPivotCol():

    pivotCol -> 0

    minValue -> table[rows - 1][0]

    for i -> 0 to cols - 1:

```
if table[rows-1][i] < minValue  
    minValue -> table[rows-1][i]  
    pivotCol -> i  
  
return pivotCol
```

Method getPivotRow(pivotCol):

```
pivotRow -> 0  
minRatio -> MAX  
for i -> 0 to rows - 1:  
    ratio -> table[i][cols - 1] / table[i][pivotCol]  
    if ratio > 0 and ratio < minRatio  
        minRatio -> ratio  
    pivotRow -> i  
  
return pivotRow
```

Method printTable():

```
for row in table:  
    for value in row:  
        print(value)
```

Method getTable():

    return table

Method getOptimalValue():

    return table[rows-1][cols-1]

Method getSolution():

    solution -> arraycols - rows]

    for i -> 0 to solution.length:

        isBasic -> false

        for j -> 0 to rows:

            if table[j][i] == 1

                isBasic -> true

                break

        if not(isBasic)

            solution[i] = 0

    return solution

Class Iterate:

Method solve(Simplex problem):

```

table -> Tableau(problem)

print("Initial Tableau:")

print("      ")

table.printTable()

while NOT table.isOptimal():

    pivotCol -> table.getPivotCol()

    pivotRow -> table.getPivotRow(pivotCol)

    table.pivot(pivotRow, pivotCol)

solution -> table.getSolution()

numDecisionVariables -> problem.getObjective().length

numSlackVariables -> solution.length - numDecisionVariables

print("The optimal solution is:")

for i -> 0 to numDecisionVariables - 1:

    print("x" + (i + 1) + " = " + solution[i])

for i -> 0 to numSlackVariables - 1:

    print("s" + (i + 1) + " = " + solution[numDecisionVariables + i])

print("Optimal value: " + table.getOptimalValue())

```

Class Solve:

Method main(args):

```

scanner -> Scanner()

print("Enter number of variables: ")

numVar -> scanner.nextInt()

print("Enter number of constraints: ")

numConstraints -> scanner.nextInt()

constraints -> doubleArray[numConstraints][numVar + 1]

print("Enter the coefficients of the constraints and include the value: ")

for i -> 0 to numConstraints - 1:

    for j -> 0 to numVar:

        constraints[i][j] -> scanner.nextDouble()

objective -> doubleArray[numVar]

print("Enter coefficients of the objective function: ")

for i -> 0 to numVar - 1:

    objective[i] -> scanner.nextDouble()

problem -> Simplex(constraints, objective)

Iterate.solve(problem)

```

To tackle the complexity of the algorithm, I decided to split up the different identifiable steps of the algorithm into different classes and methods. The main class used for this is the Tableau class, as shown in the pseudocode above. This acts as a bridge between the inputs from the user and the methods to solve the problem. Also within this class are methods I identified would be useful to create

due to their use at multiple stages. For example, I could implement the code to manually find the pivot row and pivot column inside the iterate class for each tableau, however, I decided the writing the methods `getPivotCol()` and `getPivotRow()` would make this much easier to follow. The `Iterate` class contains the `solve` method. I decided to use a while loop to ensure that iterations of the algorithm continue until an optimal solution has been reached. Using a count-controlled loop would not be suitable for this, as we do not know how many iterations of the algorithm it will take to solve the problem. The `Simplex` class is used to create an object of type `Simplex`, which contains a constraints list and an objective function. This `Simplex` problem can then be solved by the `solve()` method and the methods from the `Tableau` class.

### Test Plan

The most effective way of testing the proper functionality of the Simplex algorithm will be using past questions with answers that have already been found/created. By ensuring that the program works for various questions with differing numbers of inequalities and variables, I can then work on implementing the solution into a GUI setting.

The test questions have been taken from the Pearson ActiveLearn Decision Mathematics 1 textbook and will be shown in the full test table in the 'Testing' section.

Test Number	Test Type	Expected Outcome
1 Refer to Figure 1a below	Normal	The correct values of the final variables and maximised profit must be outputted in the terminal. These are $x_1 = 0, x_2 = 3, x_3 = 5, P = 38$
2 Refer to Figure 2a below	Normal	The right values of the final variables and maximised profit value must be outputted in the terminal. $x_1 = 0, x_2 = 40, x_3 = 10, P = 260$
3 Refer to Figure	Normal	In the terminal, the correct final values of all the variables and the maximised profit should be outputted. These are: $x_1 = 2, x_2 = 1, x_3 = 0, s_3 = 1,$

3a below		$P = 11$
4  Refer to Figure 4a below	Normal	The correct final values of the variables should be outputted in the terminal, alongside the maximised profit after all the iterations of the algorithm. These values are: $x_1 = 77/4$ , $x_2 = 0$ , $x_3 = 57/4$ , $x_4 = 0$ , $s_1 = 33$ , $s_2 = 0$ , $s_3 = 109/4$ , $P = 105.5$
5	Erroneous	The error should be recognised and instead of throwing an error, a message should be outputted in the terminal telling the user to please enter integer values only.
6	Erroneous	The error should be recognised and instead of throwing an error, a message should be outputted in the terminal telling the user to please enter the correct number of coefficients

Figure 1a

### Exercise 7B

Solve the linear programming problems in questions **1** to **6** using the simplex tableau algorithm.

- 1** Maximise  $P = 5x + 6y + 4z$   
subject to

$$\begin{aligned}x + 2y + r &= 6 \\5x + 3y + 3z + s &= 24 \\x, y, z, r, s &\geq 0\end{aligned}$$

Figure 1b

$$P = 38 \quad x = 0 \quad y = 3 \quad z = 5 \quad r = 0 \quad s = 0$$

Figure 2a

**2** Maximise  $P = 3x + 4y + 10z$   
subject to

$$x + 2y + 2z + r = 100$$

$$x + 4z + s = 40$$

$$x, y, z, r, s \geq 0$$

Figure 2b

$$P = 260 \quad x = 0 \quad y = 40 \quad z = 10 \quad r = 0 \quad s = 0$$

Figure 3a

**3** Maximise  $P = 3x + 5y + 2z$   
subject to

$$3x + 4y + 5z + r = 10$$

$$x + 3y + 10z + s = 5$$

$$x - 2y + t = 1$$

$$x, y, z, r, s, t \geq 0$$

Figure 3b

$$P = 11 \quad x = 2 \quad y = 1 \quad z = 0 \quad r = 0 \quad s = 0 \quad t = 1$$

Figure 4a

**A** 4 Maximise  $P = 4x_1 - 3x_2 + 2x_3 + 3x_4$   
 subject to

$$x_1 + 4x_2 + 3x_3 + x_4 + r = 95$$

$$2x_1 + x_2 + 2x_3 + 3x_4 + s = 67$$

$$x_1 + 3x_2 + 2x_3 + 2x_4 + t = 75$$

$$3x_1 + 2x_2 + x_3 + 2x_4 + u = 72$$

$$x_1, x_2, x_3, x_4, r, s, t, u \geq 0$$

Figure 4b

$$\text{Maximum } P = 105\frac{1}{2}, \text{ when } x_1 = 19\frac{1}{4}, x_2 = 0, x_3 = 14\frac{1}{4}, x_4 = 0, r = 33, s = 0, t = 27\frac{1}{4}, u = 0$$

### Iteration 3: Kruskal's Algorithm

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
 id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

The third iteration of the project is the implementation of Kruskal's algorithm. This will achieve success criteria 4.0 to 4.3, which involve the steps of the algorithm. Additionally, with this iteration I aim to bring in use of the GUI to take user inputs for the algorithm. This iteration aims at getting the mathematical logic behind Kruskal's algorithm working, while also bringing in an element of user input from the GUI. In further iterations, this prototype version of the algorithm in the GUI will allow for easier development of the visual element of Kruskal's algorithm using visible nodes and edges. Developing the outputs from Kruskal's algorithm and integrating them into the GUI will be the focus of a later iteration, as this initial iteration aims to focus on getting the algorithm developed correctly and just being able to check the outputs through the terminal in the IDE. Similar to the Simplex algorithm, it is important to break this problem into individual steps that also align with the Edexcel A-Level specification, ensuring that success criterion 10 is met throughout the duration of this iteration.

The first part of this iteration was to create a working prototype for Kruskal's algorithm which will run in the IDE shell. I once again used an object-oriented

approach with a separate class for edges in the graph which have their own attributes.

My idea for getting the algorithm to work on user inputted edges involves formatting the source node, destination node and weight as attributes of a single object. This can form an ‘Edge’ class. Once the series of edges have been inputted and formatted, I can then follow similar steps to the Edexcel A-Level Further Maths specification to ensure the algorithm not only works but works in accordance with the steps that [SCHOOL NAME ABBREVIATION] students are taught.

This begins by sorting the list of edges into ascending weight order, which can be easily implemented in the code and is also the first step of the algorithm taught to students. The next step is to consider each edge in the graph in turn. For this, the most suitable method would be the use of a count-controlled loop. This can iterate through the list of edges until it has iterated a specific number of times based on the number of edges there are in the graph.

## 1 Sort all the arcs (edges) into ascending order of weight.

At this stage, [SCHOOL NAME ABBREVIATION] students would perform the algorithm by inspection, considering each sorted edge in order one-by-one and referencing the graph displayed to them on the exam paper. If adding the edge would form a cycle in the graph, which would go against the definition of a tree, then the edge is rejected, otherwise it is added. This process is repeated until all edges have been considered and there is a minimum spanning tree formed in the graph. The purpose of sorting the edges at the start is to ensure that not just a spanning tree is found, but a spanning tree with the minimum total weight, so by considering the edges in ascending weight order, we add the smallest edges to the minimum spanning tree first, maintaining a lower overall weight.

## 2 Select the arc of least weight to start the tree.

## 3 Consider the next arc of least weight.

- If it would form a cycle with the arcs already selected, reject it.
- If it does not form a cycle, add it to the tree.

If there is a choice of equal arcs, consider each in turn.

To replicate this process of ‘inspection’ in the program, I need to consider what students are inspecting and how. This process involves looking at edges currently in the minimum spanning tree, looking at the edge they are currently considering and seeing if adding this edge would form a cycle.

My initial idea was to use a familiar algorithm known as a depth-first traversal. This is an algorithm that traverses along every node in a graph and can be altered to check whether two nodes of the graph have already been searched by the algorithm and therefore rejecting the edge that connects them. Two main issues arose with this solution. The first one was that traversing through all the edges in the minimum spanning tree so far for each edge would take a significant amount of time, compared to a method that can just check if one node is contained within a section of the minimum spanning tree and another isn’t. The second issue that arose is that the method of using depth-first traversal may not directly align with the Edexcel A-Level Further Maths specification, as it would remove the more natural inspection element. When performing the algorithm, the program would end up traversal each edge in the minimum spanning tree to check which nodes are present and which aren’t, however a student is more likely to just look at the nodes of the edge they are checking and see if both of them are already in the minimum spanning tree or not, instead of looking through all the edges in the minimum spanning tree.

The process of implementing a depth first traversal is shown in the image below from <https://www.javatpoint.com/depth-first-search-algorithm>

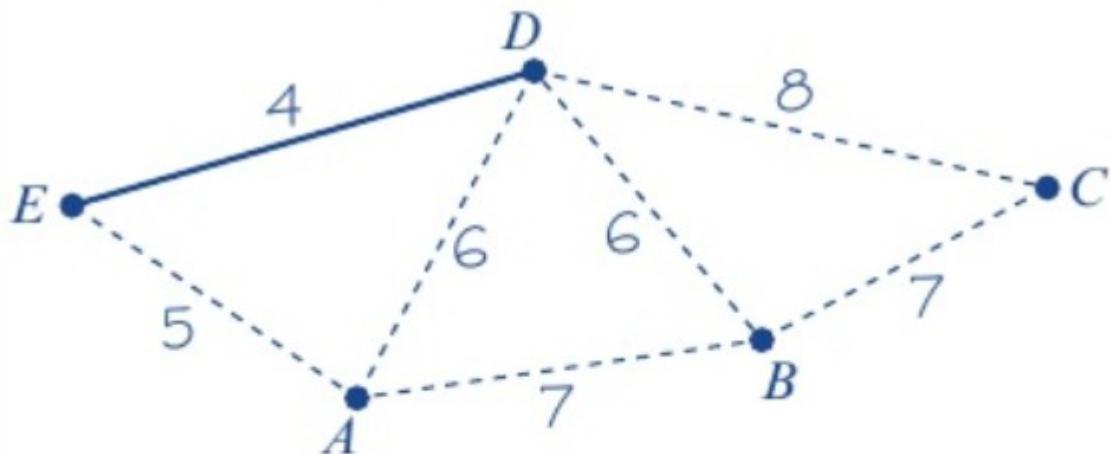
1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

This would involve using multiple data structures to keep track of the different components of the graph, which when explaining, would diverge from the explanation that adheres to the Edexcel A-Level Further Maths specification.

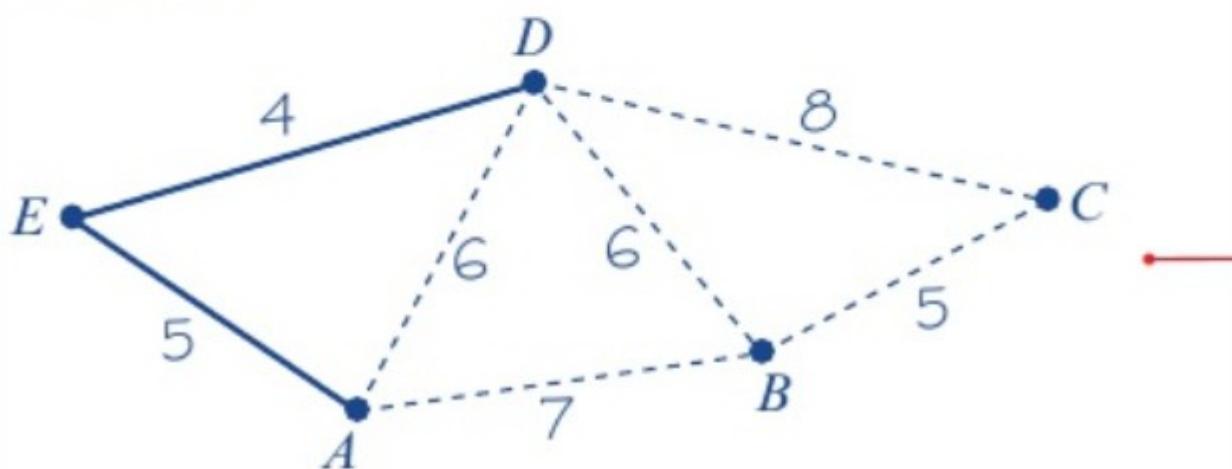
To implement the algorithm in my code, my idea is to get the source and destination of each node in the graph, and then check the component of the graph

that the destination and source are in. The component of the graph would be the minimum spanning tree currently formed, so two nodes that are part of the minimum spanning tree have the same component, but a node that is in the minimum spanning tree and a node that isn't currently in the minimum spanning tree don't have the same component. Since Kruskal's algorithm works by connecting various disconnected subtrees that eventually form the final minimum spanning tree, I will also need to check that an edge connecting two separate components is valid to be added (given that it doesn't form a cycle), so using this method of checking the component of a graph will be better at recognising separate components and therefore being able to add this edge to the graph, whereas a depth-first traversal may have to use additional data structures to store the contained nodes of different components.

Start with *DE*.



Add *AE*.



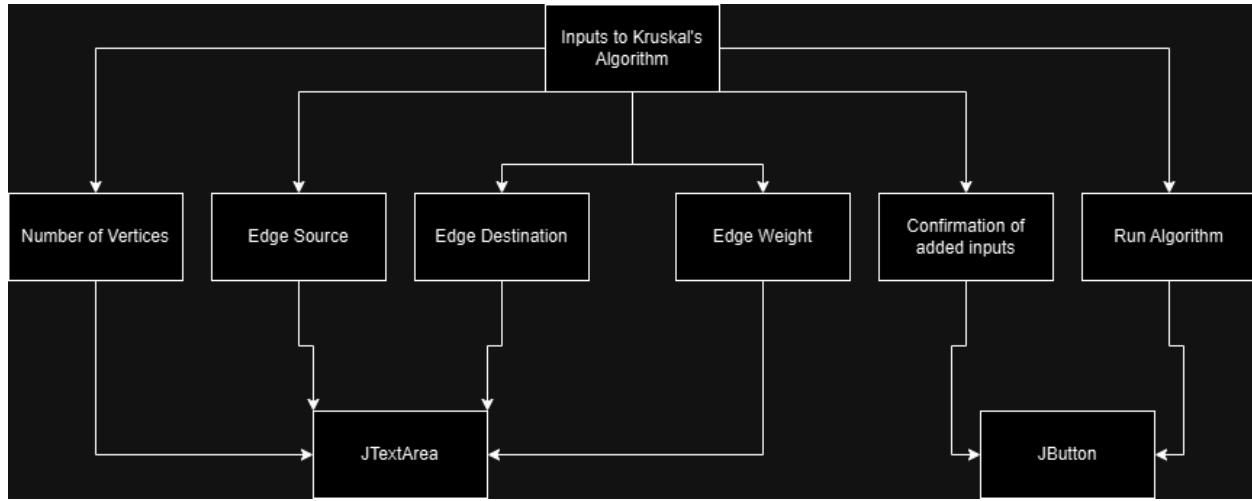
As shown by the image above taken from the Pearson ActiveLearn Decision 1 Textbook (<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>), the student would have first added 'DE', then observed the next smallest edge which is 'AE'. Using the components method, we can see that E is within the minimum spanning tree, but A isn't, so this is a valid edge to add. (Note that a slight printing error in the textbook gives the edge BC as 5 in the second graph in the image, however it should be 7). Repeating this process will yield a minimum spanning tree, that can be explained using the Further Maths steps of Kruskal's algorithm.

Below is a UML diagram which outlines the relationships between the classes used in this iteration. This also includes the interface 'Comparable' from which I used the method 'compareTo' in order to compare the weights of two edges in the graph:

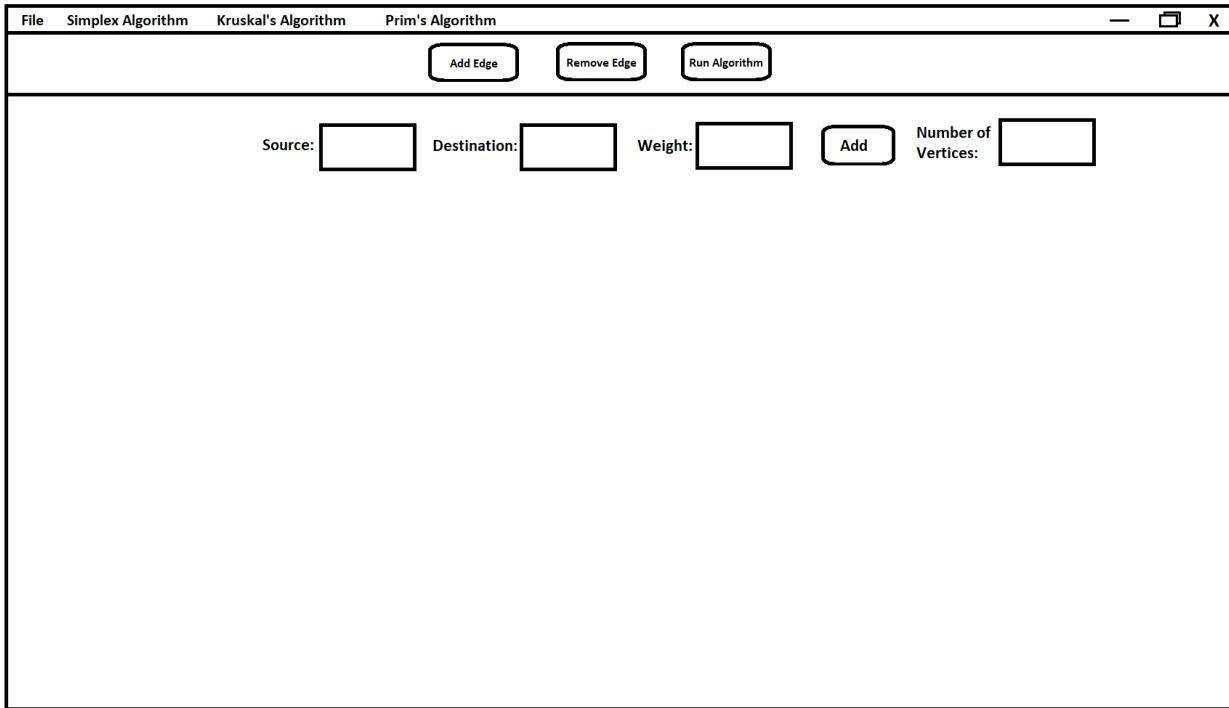


The next part of the iteration will involve using the methods created in the basic implementation of Kruskal's but taking input from the GUI instead. This feature will form part of the 'Run Kruskal's' section. The diagram below illustrates how the

inputs required for Kruskal's algorithm can be converted from user inputs in the shell to user inputs from the GUI:



The prompt to input data for Kruskal's algorithm will consist of 3 buttons. The first will be to add an edge, the second will be to remove an edge and a third will be to run the algorithm. When the 'add edge' button is pressed, the window should look similar to the following mockup:



There will be three text areas that will be labelled for the user to enter the relevant information about each edge in a graph. By taking this data and storing it in an

array, we can then add each source, destination and weight of a node to a graph object to perform the algorithm on. This iteration will focus on the 'Add Edge' and 'Run Algorithm' buttons, to lay out a foundation for taking user input from the GUI and being able to perform the algorithm. The buttons have been kept separate from the text areas to ensure success criterion 1 is still met and that the components don't get too crowded on the user interface. Laying out the text areas in a line at the top ensures that the rest of the user interface remains free for any explanation or visualisation to be added. As mentioned in the iteration introduction, this iteration will mainly focus on getting the logic of the algorithm functioning, however, making use of the computational method 'Thinking Ahead', I know that in a future iteration, the user interface will contain other components, like the displayed visual graph and the explanation of how the algorithm works, so it is best to ensure that the GUI remains consistent and ensures that any style choices made in this iteration do not interfere with the design and development in future iterations. Before moving on to developing the functions of the buttons in the GUI, I confirmed the design with my stakeholders. I asked them whether the current design was clearly laid out (adhering to success criteria 1) and if it was intuitive to use, as well as any changes they would recommend.

Question	Student 1	Student 2	Student 3
Do you think that the above design for users to enter edges into a graph for Kruskal's algorithm is clearly laid out and intuitive?	I think that the design presents the algorithm well. As a user, I can clearly see what I need to enter into each of the text boxes and there isn't too much work on my behalf in what is required, so the user interface remains clearly laid out.	I think that this GUI design works well for taking inputs into a graph. I can very clearly see what I need to input in the graph, where I need to input it and which buttons I need to press. The layout also leaves enough room in the main part of the window, I assume for explanation of	I think that the current design definitely works well at presenting the format for user inputs. I can clearly identify what I need to input, and which buttons I need to press to add the edges and then run the algorithm. The various components are also well laid out and ensure there's

		the algorithm	enough room for whatever explanation you aim to implement in the middle part of the window.
Would you recommend any improvements or additions?	No, I think that this design works well. It has all the necessary features for me as a user to input my graph, while ensuring that it isn't overcrowded.	I think that the design is fine as it is. Adding any more components may make the process less intuitive, but at the moment you have the right balance of detailed inputs, while ensuring the user interface still remains clear.	I think that your design for the inputs to the graph are fine. They clearly indicate what inputs are required and how the user can run the algorithm. To make sure it stays this way, I wouldn't recommend adding anything else.

From these questions, I can identify that my stakeholders all agree that the current design of the user interface is effective at ensuring all the necessary inputs are received from the user, that they know exactly how to input them, and that the GUI doesn't get too crowded with too many components. This allows me to continue and to highlight the classes, attributes, methods and variables that I will use to implement the processes that I have laid out.

### Justification of Classes

Class Name	Justification/Purpose
Edge	Defines the blueprint for an edge/arc in the graph, with attributes and methods which will be used by Kruskal's algorithm
Kruskals	Defines the blueprint for a graph to

	perform the Simplex algorithm on.
--	-----------------------------------

### Justification of Class Attributes

Attribute	Justification/Purpose
source	Char/Integer variable which stores the source node of the edge e.g. the edge AB has source A, as this is where the arc begins from
dest	Char/Integer variable which stores the destination node of the edge e.g. the edge AB has destination B, as this is where the arc ends
weight	Integer variable which stores the weight of the edge between 2 nodes, a vital feature for Kruskal's algorithm
edges	ArrayList to store all the edges in the graph, which can then be sorted and chosen
numVertices	Stores the number of vertices in the graph for use as a loop counter.

### Justification of Methods

Method	Justification/Purpose
Edge	This is the constructor for the Edge class that defines the source, destination and weight of an edge
compareTo	This method is part of the Edge class that compares two edges.
Kruskals	This is the constructor for the Kruskals class that defines the number of vertices and the list of edges in the graph to instantiate an object to perform the algorithm on.
addEdge	This method adds the information regarding a specific edge to the ArrayList of edges (source, destination and weight).

findComponents	Checks the vertex from the graph and finds which component of the graph it belongs to.
kruskalMST	Carries out the algorithm by first sorting the array, then checking each edge in order, adding it to the final MST if it doesn't form a cycle.
main	Instantiates an object of type Kruskals before performing the algorithm on the provided graph.
graphSettings	Adds the buttons required for the run algorithm feature to the window e.g. Add Edge, Remove Edge, Run Algorithm
edge	Adds the text areas and labels them to allow the user to enter the source, destination and weight of each edge. Also provides a text area to enter number of vertices
addToGraph	Gets the text from the text areas, stores them as strings, before converting them to characters and integers as necessary.
runKruskalsAlg	Iterates through a list of added sources, destinations and weights, incrementing by 3 each time and adding the three pieces of information as a new edge to the graph. Performs the algorithm on the graph

### Justification of Other Variables

Variable Name	Justification/Purpose
components	ArrayList to store each vertex in the graph
result	ArrayList to store the edges that are added to the minimum spanning tree
s	Stores the source of a specific edge
d	Stores the destination of a specific

	<b>edge</b>
compS	Stores the component of the graph that vertex 's' belongs to
compD	Stores the component of the graph that the vertex 'd' belongs to
total	Stores the total of the weights of the edges added to the minimum spanning tree.
addNode	Button that is pressed to present the textareas so the user can add a node
removeNode	Button that is pressed to remove a node from the graph
runAlgorithm	Button that is pressed to run the algorithm
addButton	Button that is pressed to add a node to the graph when the relevant information has been entered.
graphButtonPanel	New panel to add the buttons to
separator	Separator line under the buttons
sourceTA, destTA, weightTA, vertices	Text areas to enter source, destination, weight of an edge and number of vertices in a graph
sourceLabel, destinationLabel, weightLabel, numV	Labels to inform the user what to enter in each text area
source, dest, weight	Variables to store the contents of the text area entered by the user
sourceCH, destCH, weightInt	Variables that convert the inputs into the necessary data type so they can be passed as parameters
graphV, numVertices	Stores the number of vertices entered by the user in the text area. Converts the string input into a character, which can then be used as an integer.
graphSize	Uses the number of vertices as a parameter which is used to create the graph object with a fixed size
toAdd	ArrayList that stores all the information about edges to be added to the graph.

By splitting up each process of getting text inputs and storing them as required, before passing them into the relevant methods, I have more control over what happens when each button is pressed. For example, when the 'Add' button is pressed, a method is called that stores all the contents of the text areas. When the 'Run Algorithm' button is pressed, a separate method is called that creates a graph object and performs the algorithm on the graph.

## Pseudocode

The pseudocode below outlines how I aim to implement the logic of the algorithm that I outlined earlier. The key feature of this pseudocode is the kruskalMST method, which contains the maths behind the algorithm. For each edge in the graph, the source and destination are stored. Next, using the findComponent method that I implemented in the Kruskals class, the program stores the component of the source node and the destination node. The final check to make before deciding to add an edge to the graph is if the component of the source and destination are the same. This is done using the if-statement in the kruskalMST method.

Class Edge:

Private Integer source

Private Integer dest

Private Integer weight

Constructor Edge(nodeSource, nodeDest, nodeWeight):

source -> nodeSource - 65

dest -> nodeDest - 65

weight -> nodeWeight

Method compareTo(otherEdge):

return weight - otherEdge.weight

Class Kruskals:

Private ArrayList[Edge] edges

Private Integer numVertices

Constructor Kruskals(vertices):

numVertices -> vertices

edges -> new ArrayList

Method addEdge(source, dest, weight):

edges.add(new Edge(source, dest, weight))

Method findComponents(vertex, components):

return components[vertex]

Method kruskalMST():

edges.sort()

ArrayList[Integer] components -> new ArrayList filled with numVertices elements set to their index

for i -> 0 to numVertices - 1:

    components[i] -> i

ArrayList[Edge] result -> new ArrayList

for edge in edges:

    s -> edge.source

    d -> edge.dest

    compS -> findComponents(s, components)

    compD -> findComponents(d, components)

if compS != compD:

    result.add(edge)

    for i -> 0 to numVertices - 1:

        if components[i] == compD:

            components[i] -> compS

print("Edges included in the minimum spanning tree:")

total -> 0

for edge in result:

    total += edge.weight

    print("Edge: " + (char)(edge.source + 65) + "" + (char)(edge.dest + 65) + ",  
Weight: " + edge.weight)

```
print("Total Weight of MST is: " + total)
```

Another interesting feature of the pseudocode is the Edge class. The inputs to the algorithm are taken as a separate source, destination and weight. These can then be made attributes of the edge class, so the three inputs can be kept under one object known as an Edge. The Edge objects are used throughout the pseudocode, with a particularly important part being the for-loop in the kruskalMST method that iterates through all the edges in the list of edges.

## Test Plan

Similarly to the Simplex algorithm, the most effective way of testing the functionality of this iteration will be using past questions. An additional feature to test for this iteration will be ensuring that the components added to the GUI function correctly and that they work with the algorithm e.g. correctly extracting user inputs from text areas. As this iteration is mainly focused on the functionality of the algorithm when taking inputs from the GUI, the testing and development will not prioritise input validation. This will take place in a further iteration which will aim to display the graphs visually on the GUI window, as strict input validation will be required to create a graph that adheres to the A-Level Edexcel Specification. The main testing for this iteration will be to ensure that the algorithm I develop works as intended and can handle typical questions presented to [SCHOOL NAME ABBREVIATION] students. This justifies the use of questions from the Pearson ActiveLearn textbook to test the functionality of the algorithm.

I devised the following test plan, combining tests of the GUI components developed, as well as mathematical accuracy of the algorithm.

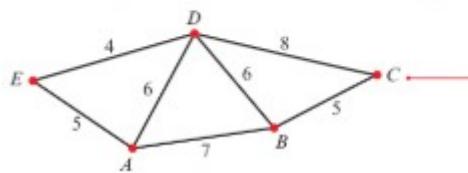
Test Number	Test Type	Expected Outcome
1	Normal	The button options should be displayed when the 'Run Kruskal's' menu item is selected
2	Normal	When pressing the 'Add Edge' button, the 4 text areas

		and additional button should appear underneath the current buttons
3	Normal	When selecting the Run Kruskal's option after another option e.g. one that displays a blank screen, the same options should be presented
4 Refer to Figure 1a	Normal	When inputting the edges from the first example in the A-Level Further Maths decision 1 textbook, the edges DE, AE, BC, and BD should be added to MST with total weight 20
5 Refer to Figure 2a	Normal	When testing example 2 from the A-Level Further Maths Decision 1 textbook, the edges AD, AC, BC, CE, EF should be added with a total weight of 49
6 Refer to Figure 3a	Normal	Inputting the graph from question 1a on exercise 3A from the Edexcel A-Level Further Maths Decision 1 textbook should provide a total weight of 98.
7 Refer to Figure 4a	Normal	Inputting the graph from question 1b on exercise 3A from the Edexcel A-Level Further Maths Decision 1 textbook should provide a total weight of 27.

Figure 1a

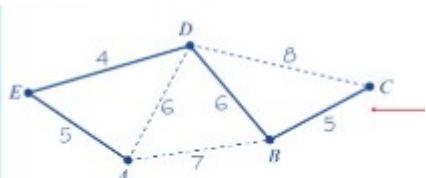
**Example 1**

Use Kruskal's algorithm to find a minimum spanning tree for this network. List the arcs in the order that you consider them. State the weight of your tree.



In a network of  $n$  vertices a spanning tree will always have  $(n - 1)$  arcs. In this case there will be 4 arcs in the spanning tree.

Figure 1b



All vertices are connected so this is a minimum spanning tree.  
Its weight is  $5 + 4 + 6 + 5 = 20$

Figure 2a

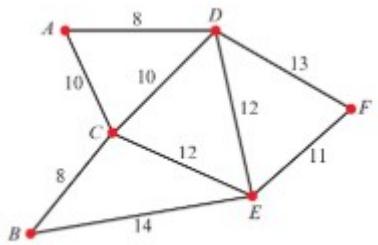
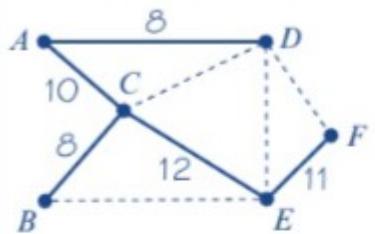


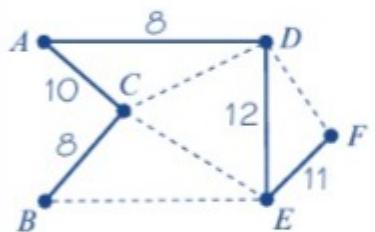
Figure 2b

One solution is

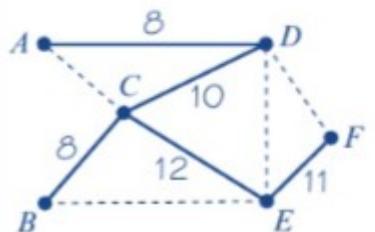


(using  $AC$  and  $CE$ )

The other three solutions are



(using  $AC$  and  $DE$ )



(using  $CD$  and  $CE$ )

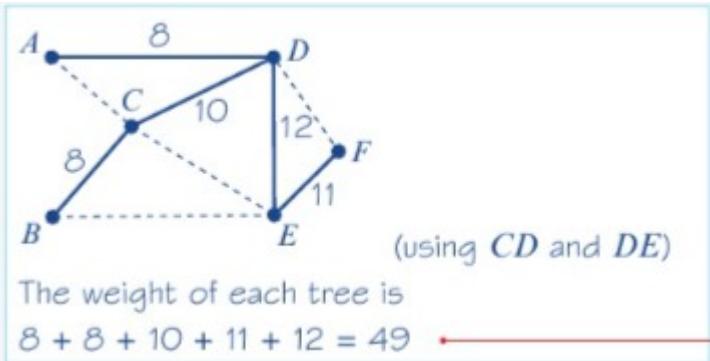


Figure 3a

**Exercise 3A**

- 1 Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

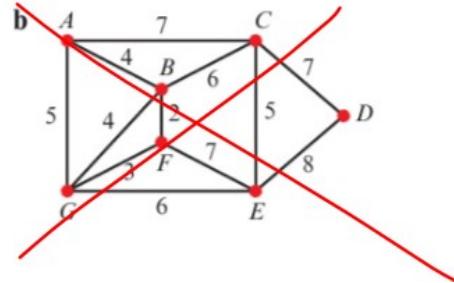
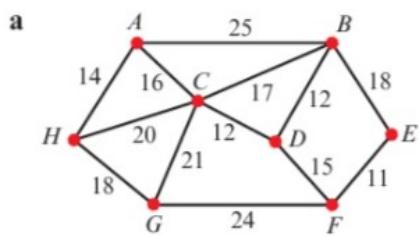


Figure 3b

- 1 a EF (11) add to tree  
BD (12) add to tree  
CD (12) add to tree  
AH (14) add to tree  
DF (15) add to tree  
AC (16) add to tree  
BC (17) reject  
GH (18) add to tree  
BE (18)  
CH (20)  
CG (21)  
FG (24)  
AB (25)
- }  
} reject all remaining arcs.

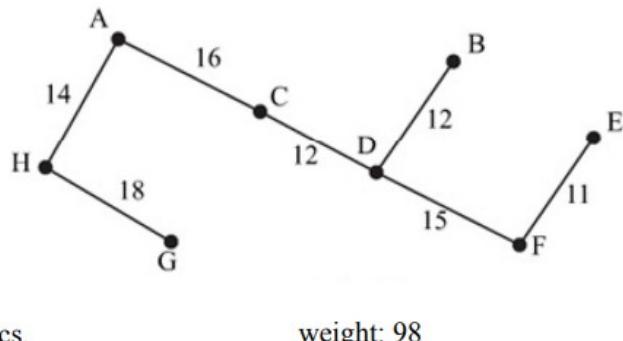


Figure 4a

### Exercise 3A

- 1 Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

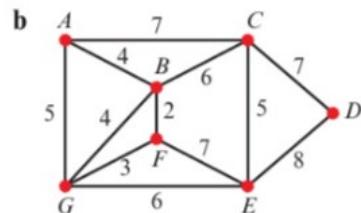
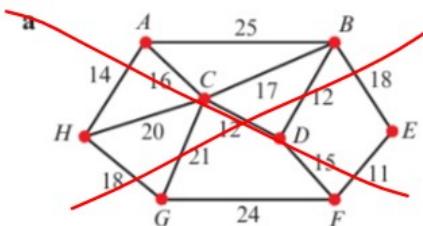
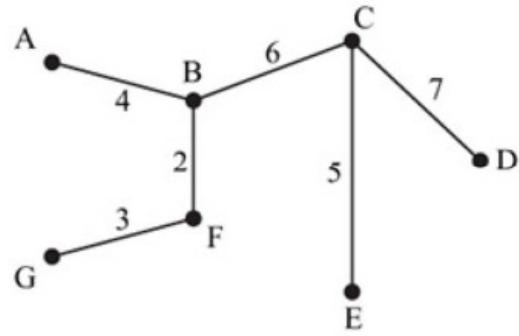


Figure 4b

- b BF (2) add to tree  
 FG (3) add to tree  
 AB (4) add to tree  
 BG (4) reject }  
 AG (5) reject }  
 CE (5) add to tree }  
 BC (6) add to tree }  
 EG (6) reject }  
 AC (7) reject }  
 CD (7) add to tree }  
 EF (7) }  
 DE (8) } reject all remaining arcs.



Having outlined the design of the algorithm, a rough basis of how I aim to implement it using the pseudocode from earlier as well as a test plan that will allow me to ensure that the developed methods in this iteration are fully functional against a variety of tests, the design for this iteration is complete.

Iteration 4: Simplex Algorithm GUI Implementation

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:

[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==)

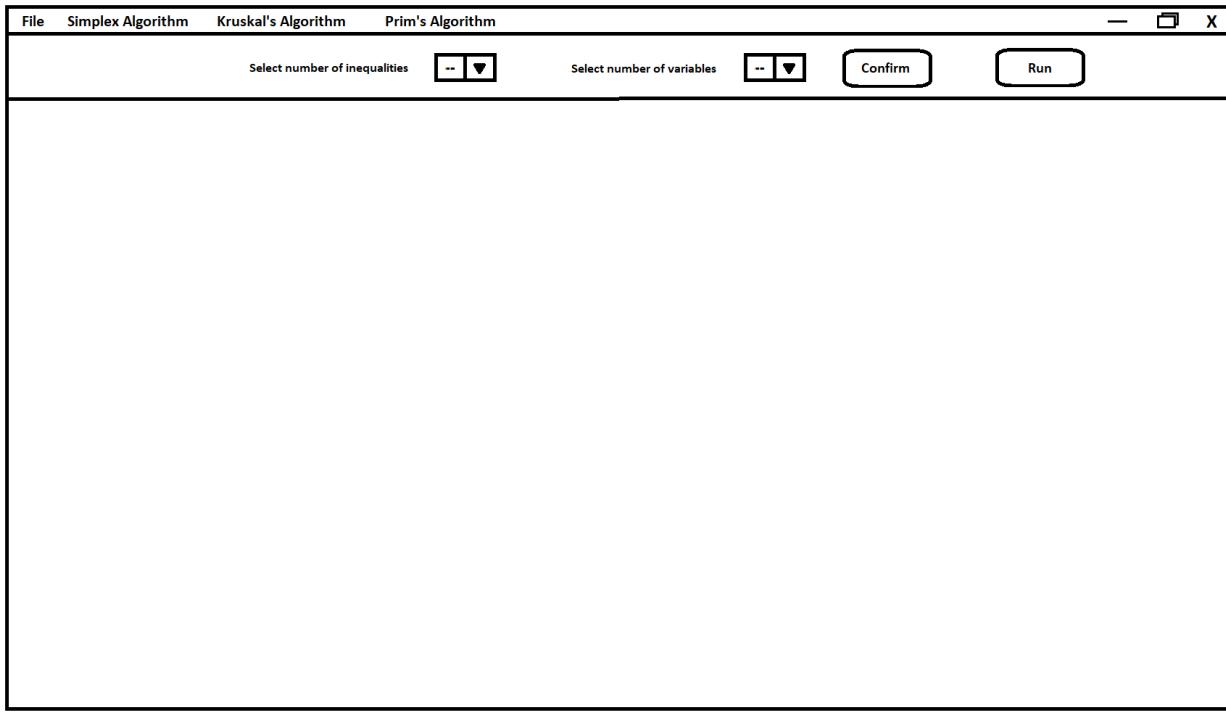
This next iteration will focus on implementing the Simplex algorithm from iteration 2 into the GUI created in iteration 1. This iteration will focus on achieving success criterion 8, which outlines allowing the user to input their own Simplex inequalities and have them explained. While success criterion 8 is the main focus, I will also have to ensure that this iteration simultaneously achieves success criteria 1 and 10. Success criterion 1 involves having a clearly laid out GUI, which is a key element of this iteration, hence the design will place emphasis on ensuring that components are laid out in an optimal, intuitive manner. This iteration also combines the simplex algorithm with the GUI, hence success criterion 10 will need to be met. This outlines explaining the algorithms in accordance with the Pearson Edexcel Further Maths specification, to ensure the explanations are valuable to [SCHOOL NAME ABBREVIATION] students. Before being able to guide users through the algorithm, I need to first ensure that user inputs into the GUI can be taken into the subroutines developed in iteration 2 and perform the algorithm on user-defined constraints. This iteration follows on well from iteration 2, as I can structure the GUI in a way that allows the user inputs to be easily formatted in the correct way for use by the methods developed for the Simplex algorithm. The first part of this iteration will aim to create the components on the window based on how many inequalities and variables the user requires. This will be vital to allow the user to enter any question proposed by the Edexcel A-Level Further Maths Decision topic of the Simplex Algorithm. The specification for the exam board mentions that students are expected to be able to perform the algorithm on a maximum of 4 variables with 4 inequalities/constraints.

<https://qualifications.pearson.com/content/dam/pdf/A%20Level/Mathematics/2017/specification-and-sample-assesment/a-level-l3-further-mathematics-specification.pdf>

Problems will be restricted to those with a maximum of four variables (excluding slack variables) and four constraints, in addition to non-negativity conditions.

This limit on the number of possible inputs allows me to use components like a drop-down box and buttons to take user inputs on the number of inequalities and number of variables required. Using buttons allows me to ensure that only valid inputs are sent by the user as the button will have set functionality that validated inputs before accepting them to be used by the algorithm. I chose this over the use of a text area/text field for the same input as it would require very detailed input validation, because the values entered for the number of inequalities and the number of variables are used throughout the methods for the algorithm.

My initial idea involved using two drop-down menus, with labels to prompt the users what each drop-down option is for. An initial mockup for this GUI is shown below:



As shown, there are two drop-down menus with labels to indicate their purpose. This is combined with two buttons, one to confirm the user's options and one to run the algorithm. There is also a separator line to ensure components displayed in the 'main' center part of the window are clearly separated from the option selection.

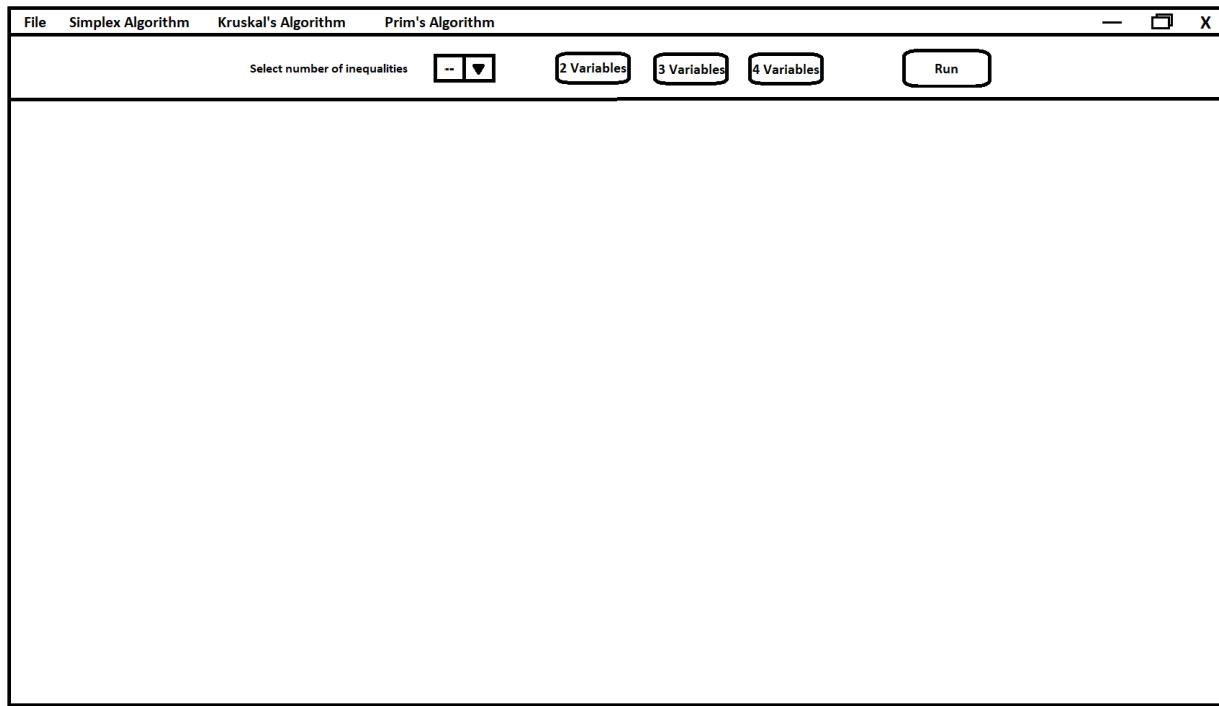
Before proceeding with adding functionality to each of these, I need to confirm with my stakeholders that this layout is suitable for the users. The questions I will ask my stakeholders are presented below. They are more open-ended questions to allow my stakeholders to

1. What do you think of the use of drop-down menus in the GUI?
2. Do you think the drop-down menus are used in the appropriate places for ease of use by potential users?
3. Are the various components laid out in an intuitive manner where you can instantly understand their function?
4. What do you think about the use of buttons in the GUI, are they used appropriately and are they labelled intuitively?
5. Is the use of a separator line suitable for this situation?

Question Number	Stakeholder 1	Stakeholder 2	Stakeholder 3
1	The drop-down menus are well labelled and clearly indicate their purpose	I think it would be better as a user to be able to see the options immediately instead of having to use a further drop-down menu	I would prefer there to be a clear indication immediately of what I need to select instead of going through multiple menus
2	I think leaving them at the top allows for clear distinction that selecting an option is a prerequisite for the algorithm to run	Leaving the menus at the top indicates clearly what I need to select.	Having all the options at the top is intuitive if they are well spaced apart.
3	There is a bit of ambiguity between the confirm and run button as it may be difficult to know when to press each button	I get a bit confused by the two buttons as they seem to have quite similar names. Removing the 'confirm' button would make it more intuitive	I think the majority of components are intuitive but the 'confirm' button seems unnecessary and a bit confusing.
4	I think that swapping one of the drop-down menus for a series of buttons would be more intuitive	I think it would be better if one of the drop-down menus was split into buttons that automatically	The 'run' button is fine, but as mentioned before, I think the 'confirm' button is unnecessary and

	as that would allow for the removal of the confirm button. The run button is clearly labelled.	confirm your decision	would prefer my option to be confirmed as soon as I select it
5	I think the line does a good job at distinguishing the main window from the options	The separator line highlights that the components at the top are options that must first be configured before running	The line helps to create a nice split between the main window and the options at the top. I would recommend keeping it.

From these answers, I can deduce that the main ambiguity comes in the form of the confirm button which doesn't serve a main purpose other than storing user inputs. I took the advice of the stakeholders and chose the split the number of variables option into 3 buttons and when one of them is selected, the user inputs are automatically stored. This allows me to just keep the run button which is more intuitive as a single function that runs the algorithm.



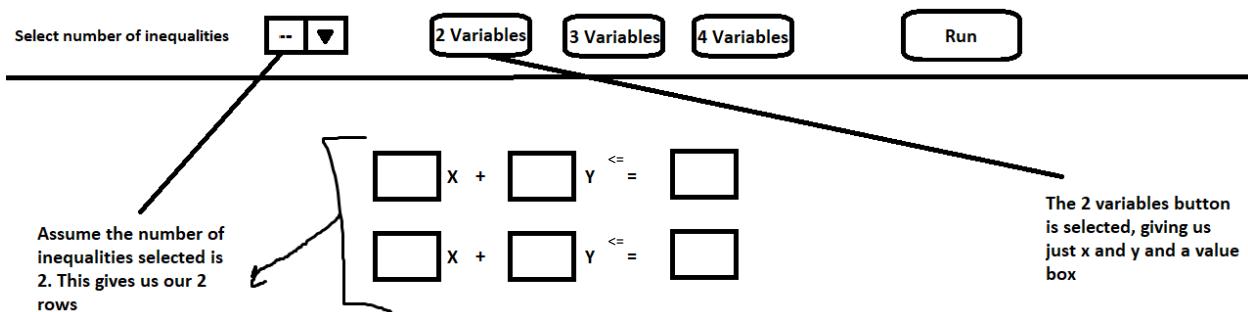
After this second mockup, I once again consulted my stakeholders about how this compares to the initial GUI.

Question	Student 1	Student 2	Student 3
Do you think this updated GUI mockup is more intuitive than the previous one?	This version of the GUI is much clearer since there's just one button that is clearly aimed at running the algorithm instead of the confirm button and the run button. I like the user of the variable buttons too as they clearly indicate their purpose, and I can	I think this version is definitely much more intuitive. Removing the confirm button made the biggest difference as I can now clearly see what to press when I'm ready to run the algorithm. The use of space is also good; the row of buttons and the drop-down doesn't take up	This GUI mockup is much better in terms of being intuitive and clear to me as the user of the system. I can now clearly see what to select and how, and I know what to do to run the algorithm, since now there's only one button instead of the 'Confirm' and 'Run'

	access them immediately.	too much space but is also prominent on the screen	button that were in the previous version.
--	--------------------------	--	---

They all agreed that the use of 3 buttons for the variables and one drop-down menu is much more intuitive. This confirms that my new mockup is effective in achieving success criterion 10, since the end-users have now stated that the GUI is intuitive. They also mentioned how removing the 'confirm' button and opting for a single 'run' button made the GUI much more user friendly. From here I can proceed with designing the functionality of these buttons and drop-down options.

Having now designed the options; I can move onto designing the appearance of the GUI and what components will appear and in what location when the options are selected. The drop-down menu will just be used to store the user's choice of number of inequalities. Each of the three variable buttons will have different functionality. The two-variable button will take the number of inequalities from the drop down and create that many rows of input fields with 2 variables. For ease of explanation this is visualised below.



Depending on the number of inequalities chosen and the number of variables selected, a different number of text areas and rows of equations will appear on the GUI. Ensuring that the users were happy with the input methods earlier is also vital

for this, since their inputs to the system will be used to create the dimensions for various data structures like the Simplex tableaux.

We also need an objective function for the user to enter. This is not dependent on the number of inequalities but is dependent on the number of variables. Taking the input from the button will make a row appear below the inequalities. In exam questions, the objective function is generally given within the question. While one option would be to have the user enter it in a section off to the side, it would be more visually intuitive and use less space if I have the text areas underneath the rows of inequalities. This is further justified as when creating an initial tableau, the objective function is entered in the bottom row of the table, as shown in the example below (screenshot is taken from the Pearson ActiveLearn Decision 1 Textbook (<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>)):

Our initial tableau is:

Basic variable	x	y	r	s	Value
r	5	7	1	0	70
s	10	3	0	1	60
P	-3	-2	0	0	0

The first row shows the first constraint, the second row shows the second constraint and the final row shows the objective function.

When the variable button is pressed, an objective row will appear below the inequalities, an example is displayed below with 2 variables selected.

Select number of inequalities     Run

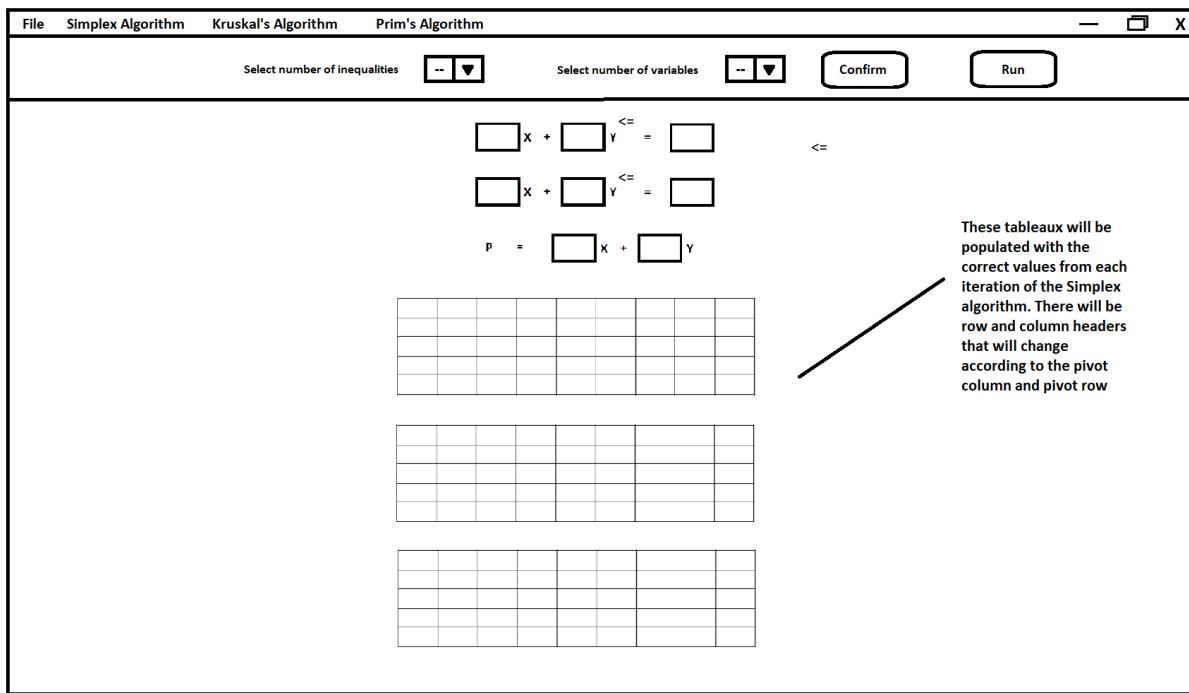
Assume the number of inequalities selected is 2. This gives us our 2 rows

The 2 variables button is selected, giving us just x and y and a value box

Objective row with 2 variables

This process will follow for all the other variable buttons. They will take the value from the drop-down menu and use it to create rows of inequalities, with the specified number of variables based on the button pressed.

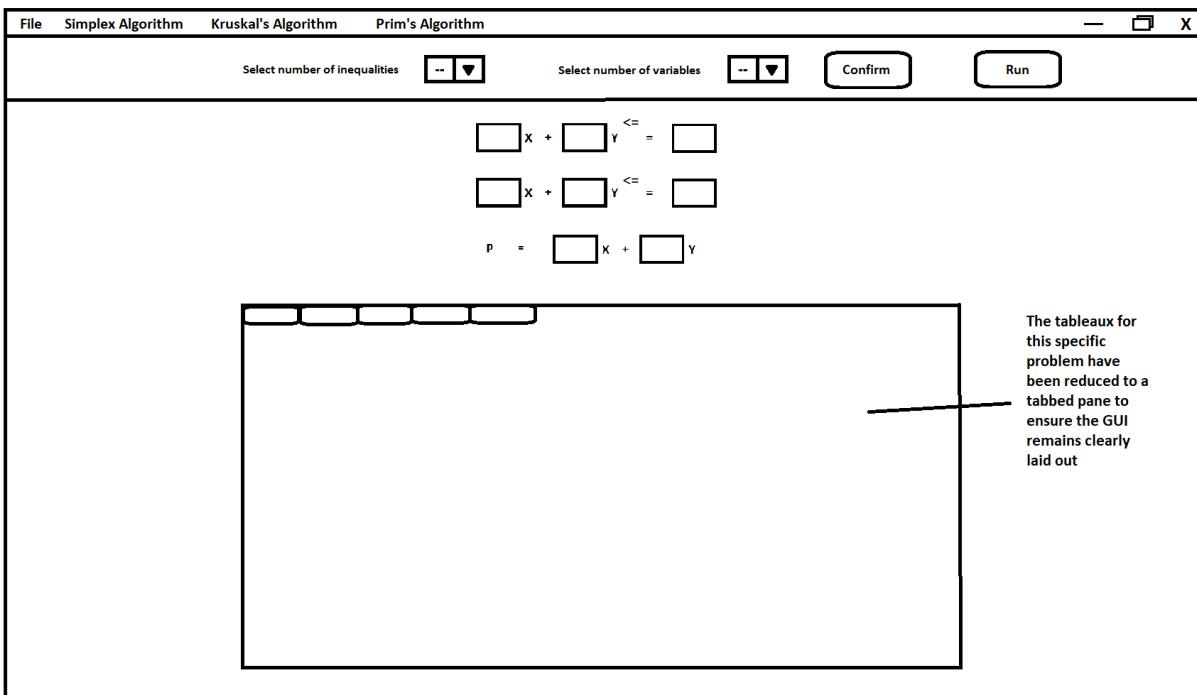
The next part of this iteration will involve displaying the tableau from each iteration of the Simplex algorithm on the GUI. My initial idea is to have the tableaux outputted one below the other in a vertical column, which will allow for explanations next to each tableau. A visualisation of this design is shown below.



After designing this, one possible issue that occurred to me is a Simplex problem that requires many iterations. This would lead to many tableaux and using a layout as suggested above would not be able to accommodate for them and display them all.

Another design I explored was the use of a JScrollPane, which would be a fixed size in the window, and would allow the user to scroll down indefinitely, for as many tableaux as were generated. However, I realised that this would not satisfy success criterion 1, which mentions a clearly laid out tableau. As discovered in the analysis, a well-laid out user interface is a key feature of a successful teaching tool. Having a scroll pane would make it difficult for the user to see each iteration of the

algorithm in turn as they would have to scroll to the correct place and keep scrolling between different sections. The next option I explored was using a JTabbedPane. This is a much better approach as I can split the tableaux into their own tab, which can then be easily selected by the user.



Using a JTabbedPane will also allow me to clearly explain the process happening on each tableau, as the explanation for each tableau can be added to separate panels with their corresponding tableau. I will make use of the feature allowing the tabs to be named in Java, and title each tab with whichever iteration of the algorithm it covers e.g. the first tab will have 'Iteration 1', the second will have 'Iteration 2' and so on.

Each tab will contain 4 components, the first will be the tableau, taking up roughly half the height of the tab, and three quarters of the width. This will ensure the tableau is clearly visible but also provides enough room for the other components. The next two components will be displayed to the right-hand side of the tableau, taking up the rest of the width of the pane. These will be a column of theta values and a column of row operations. As part of the Simplex method, theta values are calculated by dividing the number in the value column by the number in the pivot column. The smallest positive theta value is then chosen to indicate the pivot row.

The row operations are also another feature required as part of the Simplex method. The row operations show what specific calculation was performed on the values in a row to create the next tableau, all given in terms of the pivot row. For example, if Row 1 is the pivot row, an example row operation would be  $R_2 - 3R_1$ , where each value in row 2, had 3 lots of the respective value in row 1 (pivot row) taken off it. Showing the row operations counts for marks in the exam, hence I will show them next to the tableau.

The final component of each tab will be an explanation. This will take up the bottom half of the panel and span the entire width. This will be used to explain to the user what is happening at each stage, fulfilling success criteria 8 and 10. To get an idea of what to include in this explanation, I am going to ask my stakeholders what they would like to have explained in this section.

Question	Student 1	Student 2	Student 3
For an explanation feature in the Simplex algorithm feature, what do you think are key parts that should be explained to the user	I think explaining how the pivot column and row were calculated is important, although it's a minor step relative to the whole algorithm, getting it right is important, so I would value it being explained.	Explaining how to get the pivot value and the correct row operations that need to be put into the table would be useful features that I think students need a good understanding of	Once I am sure with my pivot value, I can use the Casio CG50 graphical calculator to perform the row operations, so I think the main explanations needed are how to get the pivot value, and what the row operations are.

From consulting the students, the main two areas that need explaining are the process of calculating pivot values, and the row operations involved. Students at [SCHOOL NAME ABBREVIATION] are provided with the Casio CG50 graphical calculator for their further maths exams, which have functionality of entering a tableau then manually entering row operations, which the calculator will then perform. However, the students do need to know which row to use as their pivot

row and which row operations to enter. These will be the focus of my explanations. The format of explanations will be a fixed response, which will use specific values from the tableau that is being explained. To create a more ‘natural’ sense of explanation, I am going to create multiple responses, then use a random number generator to pick between them. This should avoid the same explanation being repeated multiple times. Another factor to consider is that the explanation for different tableau throughout the iterations needs to be different. For example, the middle iterations will have to explain the row operations from the previous tableau, but for the initial tableau, there is no previous tableau, and no row operations displayed, so a modified explanation will be required. Similarly, the final tableau will not have anything performed on it, as there are no further iterations. This means I cannot use the same explanation that describes finding the pivot column and row, then calculating theta values and row operations, as this is not necessary for the final tableau.

Looking through the textbook, I also noticed how each tableau they use has the pivot column and pivot row highlighted for purposes of explanation, shown below:

Basic variable	x	y	r	s	Value	$\theta$ values
r	5	7	1	0	70	$70 \div 5 = 14$
s	10	3	0	1	60	$60 \div 10 = 6$
P	-3	-2	0	0	0	

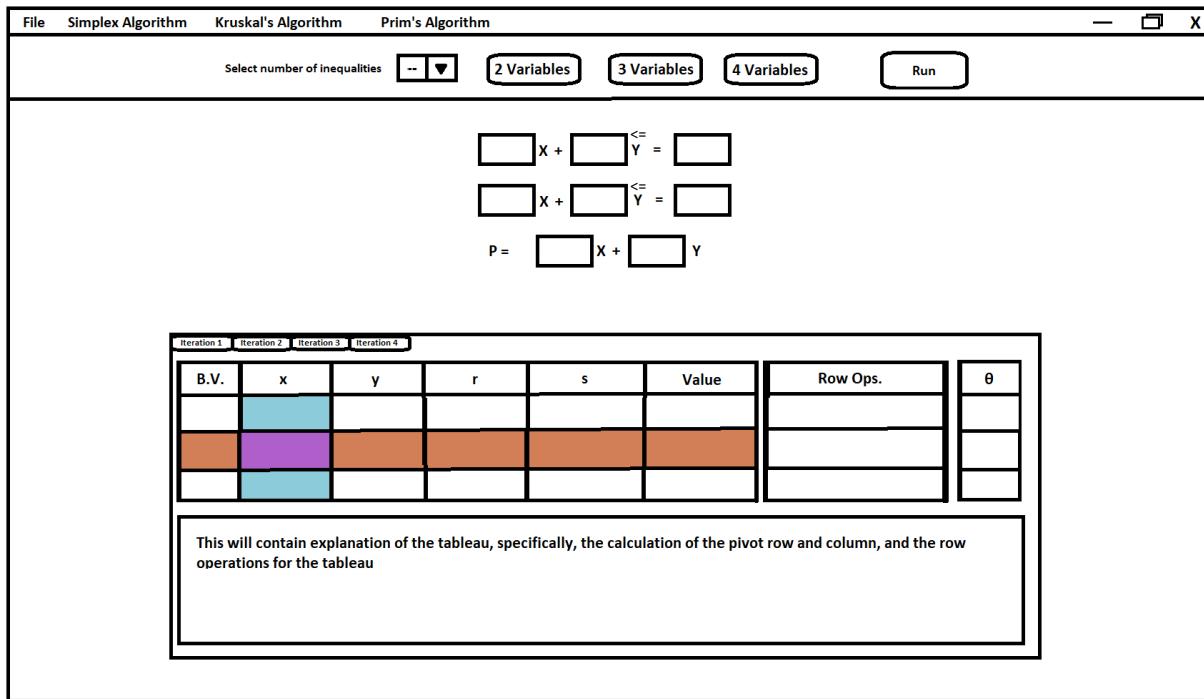
The orange is used to highlight the pivot column and the blue used to highlight the pivot row, with the green being the pivot value – intersection between pivot column and row.

Question	Student 1	Student 2	Student 3
What are your opinions on the colouring used in the tableau (showing them the textbook tableau)	I think it's useful to help students visualise how they're finding the pivot value. I think overall the colour choices are good.	I find the colouring quite useful in highlighting the pivot column and pivot row, but I feel that the green is too similar to the blue used for the pivot row	I think that the highlighting of the pivot column and row are beneficial as it clearly identifies key information, which is especially useful for a larger tableau. I do feel

		that the green is a bit too similar to the blue.
--	--	--

From this I can gather that all three students agree that the colours are beneficial. Two of them highlighted that the green used in the textbook to highlight the pivot value is too similar to the blue used for the pivot row, so for my colour scheme, I will choose a different colour that does the job of highlighting the cell, but isn't too overwhelming and detracting from the text in the table.

Combining all the features described above, an instance of the GUI after running the algorithm should look like the mockup below:



One tab has been split into the tableau, highlighted with colours, the row operations and theta values, and the explanation box. The sizing of the JTextPane will be selected to ensure it doesn't overlap with the rows of inequalities above. The furthest that the inequalities may come down in the GUI depends on the number of inequalities selected by the user. This means that the tabbed pane will be sized according to the 4-inequality option, as this will use the greatest space, and if the tabbed pane fits correctly with 4 inequalities, it will also work with

lower values. The width of the tabbed pane will be selected to ensure it is large enough to accommodate all components and takes up a sensible proportion of the available space, while not distorting the inner components, which will fill the space inside the tab.

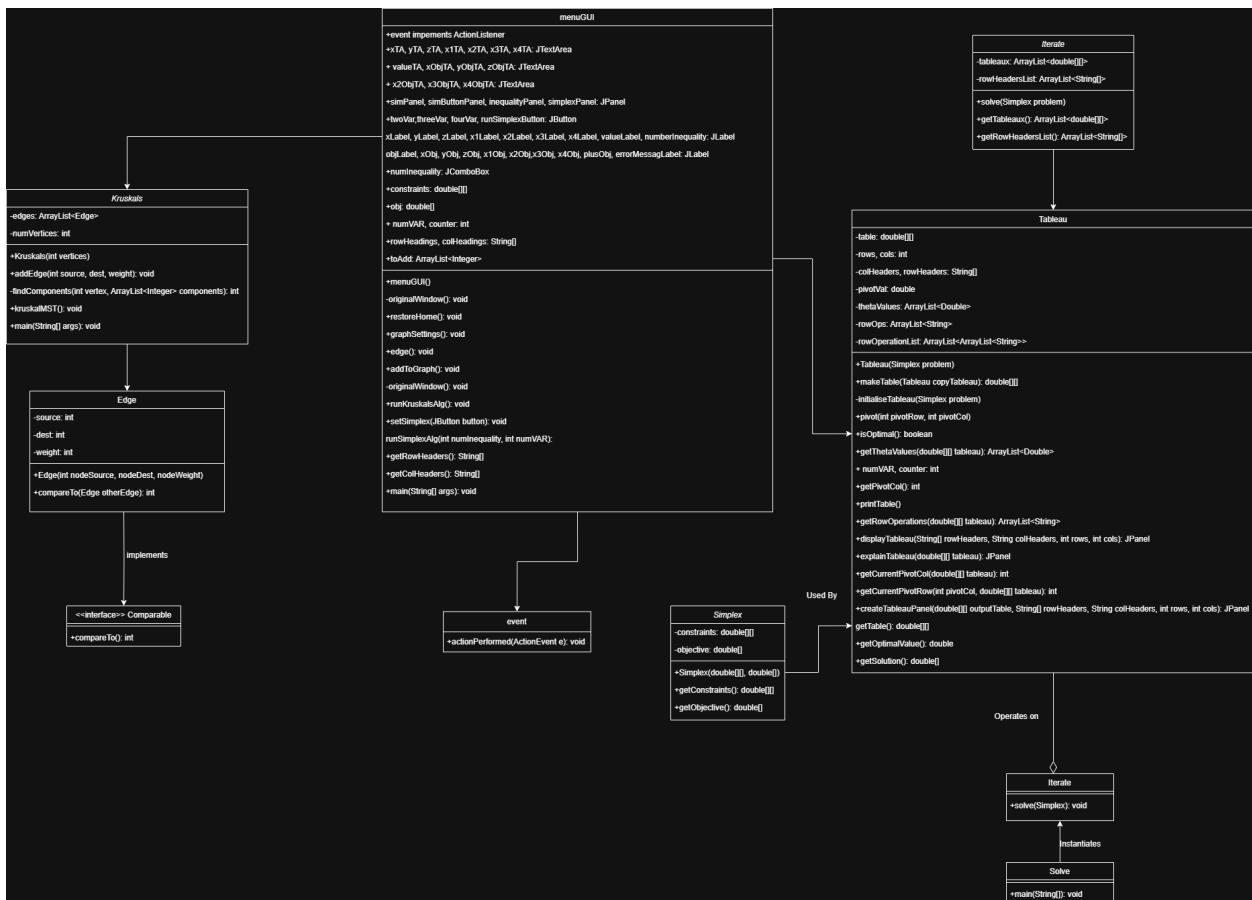
The final feature of design to consider is the input validation. Considering all inputs, I have deduced the possible errors that could be thrown, shown in the table below:

Error	Explanation of error
Selecting number of variables before number of inequalities	Since the functionality of the number of variables button will require the value from the number of inequalities drop-down, selecting the former first will result in a number format exception, since the default state of the drop-down menu is the string “--”. The function for the button will try to use “--” to define the loop counter for how many rows should be displayed, throwing an error.
Pressing run before selecting the variables, so no inputs have been collected.	This will cause an error, as the functions to retrieve inputs will try to store values from components that haven’t yet been added to the GUI
Pressing run before selecting inequalities and variables	Similar error to above, as the functions to retrieve inputs will try to store values from the components that haven’t been added to the GUI and don’t contain any values yet
Entering strings or any non integer/double value into the text areas	As the values in the text areas are stored as doubles, a number format exception will be thrown when trying to store a string. Another possible error is trying to perform calculations on strings, which will also throw an error.
Entering nothing/ leaving the text areas blank.	This is a special case of the ‘entering strings’ error, as when parsing over the

text areas, the values in the text areas will be stored as empty strings, giving the same error as above.

To deal with these errors, I will implement a try-catch loop at any point where user inputs are gathered, before displaying a label that highlights the issue to the user. I will colour the label in red to ensure that it stands out among the rest of the black text, to highlight that it needs the user attention, and the red highlighting a possible issue. Dealing with possible human error in a simple manner will ensure robustness of the system.

Below is a UML diagram showing the relations between the various classes used in this iteration. The methods developed in this iteration will be integrated into the classes created in the previous iterations, specifically the menuGUI class, Tableau class and Iterate Class.



The Tableau class will contain most new methods and attributes for this iteration, which will then be used by other classes, as displaying the tableau will make use of the methods that create the tableau, which were defined in the Tableau class. The diagram shows how the menuGUI class and Iterate class will use the methods from the Tableau class. User input will be taken from the GUI, hence they will be stored in the menuGUI class. These will then be passed into the Tableau class to perform the algorithm on. Another feature to consider is that the existing methods to perform the algorithm perform the entire algorithm on one tableau, however, to output the state of the tableau after each iteration of the algorithm, I will have to store the state of each tableau. This is where the Iterate class links into the Tableau class. In the Iterate class, the solve method has a while loop that repeatedly performs iterations on the tableau. In this while loop, I will use methods developed in the Tableau class to store the state of the tableau in that iteration of the loop. Once all the tableaux are stored, they can then be outputted in the correct order. Another factor to consider is that the changing row headers of each tableau. After an iteration of the algorithm, the row header of the pivot row changes to the value in column header of the pivot column. I will have a list of row headers and in the Iterate class, in the while loop, I will retrieve the pivot column and its value, before changing the index in the row headers list. This can then be appended to the tableau when it is displayed. Those are the main links between classes required for this iteration, highlighted by the UML diagram.

### Justification of Class Attributes

Attribute Name	Justification/Purpose
xLabel, yLabel, zLabel, x1Label, x2Label, x3Label, x4Label, valueLabel, xObj, yObj, zObj, x1Obj, x2Obj, x3Obj, x4Obj, plusObj	These are all JLabel attributes that are displayed on the screen, implying to the user what each text area represents, allowing the text area to act as a 'coefficient box' for each variable. They are all defined and added to the GUI in the same way, serving the same purpose with the only distinction being the actual text added to the label, hence they can be

	explained in one go. Based on the number of inputted inequalities and variables, different combinations of these will be outputted e.g. 4 variables will use x1Label, x2Label, x3Label, x4Label, whereas 2 variables will just use xLabel, yLabel. The attributes name with 'Obj' are added separately for the objective function, with the same purpose as described above
twoVar, threeVar, fourVar	These are 3 JButton components added to the initial screen when 'Run Simplex' is selected, allowing the user to select whether they want to enter a question with 2,3 or 4 variables.
runSimplexButton	This is a JButton component that will take the inputs entered into the GUI components, convert them into the correct format for the simplex methods, then perform the algorithm on them
simPanel	A JPanel component which acts as the main panel added to the window when Run Simplex is selected
simButtonPanel	A JPanel component added to the top of the simPanel, and will contain the option selector buttons
simplexPanel	This is the JPanel component where the text areas will appear when the user presses one of the variable buttons. A separate panel is needed for this to ensure components are added in a meaningful layout e.g. in rows of inequalities, one above the other
xTA, yTA, zTA, x1TA, x2TA, x3TA, x4TA, valueTA, xObjTA, yObjTA, zObjTA, x1ObjTA, x2ObjTA, x3ObjTA, x4ObjTA	These are all JTextArea components. They are all the same size and format, hence they can all be explained in one go. It is important that they are all the same size and format to ensure that

	when they are added to the GUI they maintain the layout of inequalities split up into separate rows, as described above
numInequality	This is a JComboBox, which presents users with a dropdown menu. I will use it to let users select the number of inequalities for their question
constraints	This is a 2d array of doubles, which will be used to format the values inputted to the GUI to perform the algorithm on.
obj	This is a 1d array of double values, which will be used to store the inputted values of the objective function, so they can be passed into the method to perform the Simplex algorithm.
numVAR	This is an integer and will store the number of variables, based on which of the variable selection buttons the user presses. This will be vital as the dimensions for the constraints array will require the number of variables to initialise it.
counter	This is an integer and will store the number of inequalities selected in the drop down menu. It is named counter, as its main use it to act as a counter variable in a for loop to initialise the row headers and the column headers
rowHeaders, colHeaders	These are both 1d String arrays that will store the row headers and column headers for the tableau, based on how many variables and inequalities are selected e.g. with more inequalities, more slack variables will need to be added to the lists. These will be displayed on the tableaux in the GUI, changing for each tableau as explained

	above.
thetaValues	This is an arraylist of doubles that will store the theta values calculated for a tableau, by dividing each number in the value column of the tableau, by the number in the pivot column of the tableau. This will then be displayed next to the tableau as part of explanation.
rowOperationList	This is an ArrayList of string arrays, which will store lists of row operations for a specific tableau. These row operations will then be formatted into labels and outputted next to the tableaux in the GUI as part of the explanation.
tableaux	This is an arraylist of 2d double arrays, which will store the various tableaux as a list of 2d arrays after each iteration of the algorithm, so they can be displayed in order on the GUI. Since the methods created in Iteration 2 were performing the algorithm on the same table, I will have to make a copy of the table during each iteration and add it to this list to store the state of the table.
rowHeadersList	This is an arraylist of a 1d array of strings, which will be used to store the row headers for the tableau after each iteration. Similarly to the list of tableaux described above, the state of the row headers during the iterations will need to be stored, to ensure they can each be displayed in the GUI.

### Justification of Methods

Method Name	Justification/Purpose
simplexSettings()	This method places the various buttons

	and drop-downs on the GUI when the Run Simplex menu item is selected.
setSimplex(JButton button)	This takes in the variable button as a parameter, and based on the number of inequalities selected and the number of variables selected by the button, the rows of inequalities and the objective row text areas are displayed on the panel.
runSimplexAlg(int numInequality, int numVAR)	This method extracts the inputs from the text areas and converts them into a tableau upon which the Simplex algorithm can be run. The parameters into this method are used to define the dimensions of the tableau.
makeTable(Tableau copyTableau)	This method is used to store a copy of an array to ensure that each iteration of the Simplex algorithm can be fully explained with the relevant tableau.
getThetaValues(double[][] tableau)	This method calculates and stores the theta values for a given tableau so they can be added to the theta panel which is provided as part of the explanation for the algorithm.
getRowOperations(double[][] tableau)	This method formats the row operations for each row in a provided table. It adds the row operations to a panel which is then provided as part of the explanation for each iteration of the algorithm.
displayTableau(String[] rowHeaders, String[] colHeaders, int rows, int cols)	This method displays the various components in the tabbed pane as part of the explanation of each iteration of the algorithm. This includes the tableau from the iteration, the row operations, theta values and the explanation of the tableau. It uses a grid bag layout to give different components customised space in the

	panel.
explainTableau(double[][] tableau, int a, int numTableau)	This method takes in a tableau as a parameter and returns an explanation for how the pivot was selected, how the row operations work and how the table reached it's current state.
getCurrentPivotCol(double[][] tableau)	This method returns the pivot column for a specific tableau, which is used for explanation of individual tableaux
getCurrentPivotRow(int pivotCol, double[][] tableau)	This method returns the pivot row for a specific tableau, which is also used as part of the explanation of individual tableaux
createTableauPanel(double[][] outputTable, String[] rowHeaders, String[] colHeaders, int rows, int cols, int tableauCount, int numberTableau)	This method displays a specific tableau that was stored using the makeTable method. It uses a panel with a grid layout to format the arraylist into a 'table' layout to be displayed on the GUI.

### Justification of Other Variables

Variable Name	Justification/Purpose
copyTable	This is a 2d double array, which is used to create a deep copy of a tableau after each iteration of the algorithm. This will be used in the makeTable method.
existingTable	This is a 2d double array that will take the value of the Tableau passed into the makeTable method as a parameter. This can then be copied to copyTable and stored.
pivotC	This is an integer that takes the value of the pivot column index of the current tableau
first	This is a double that takes each number in the value column of the tableau, used to calculate theta values

second	This is a double that takes each number in the pivot column of the tableau, used with 'first' to calculate the theta values
theta	This is a double that is assigned to the result of first/second, which is the theta value.
rowOps	This is a string arraylist that will contain each row operation for a tableau
pivotR	This is an integer that takes the value of the index of the pivot row in the current tableau
val	This is a double that takes the value of each item in the pivot column of a specific tableau
formatting	This is a string that takes the value of val as a string, then formats it to remove the '.0' from the end, if it is an integer.
toOutput	This is an arraylist of 2d double arrays that will store all of the tableaux from a simplex problem using the getTableaux method in the Iterate class
rowHeaderArray	This is an arraylist of a string array that will store the lists of changing row headers for the tableaux, using the getRowHeadersList method in the Iterate class.
tabbedPane	This is a JTabbedPane component that will display each of the tableaux in a new tab along with theta values, row operations and the explanation
fullPanel	This is a JPanel component and will be the panel which is added to which the tabbed pane is added. This panel will be added to the main GUI underneath the inequality panel and objective panel.

tableauCount	This is an integer that acts as a counter variable. This is necessary as the first tableau outputted doesn't have any row operations. This variable will be incremented for each table in the list of tableaux and if it is greater than 0, only then will row operations be added to it.
thetaList	This is an arraylist of doubles that stores the theta values for a specific table, using the getThetaValues method
thetaPanel	This is a JPanel upon which the theta values will be added and displayed in the GUI next to the tableau
theta	This is a JLabel containing the symbol " $\theta$ " and will be added to the thetaPanel as the column header so users know what the values in the column represent
rowOpPanel	This is a JPanel component, and is similar to thetaPanel, but will contain the row operations for a tableau to be displayed on the GUI
rowOp	This is a JLabel containing the text "Row Ops.". Like the theta label, it will be added as the column header of the row operations displayed on the rowOpPanel, so users know what the contents of the column represent
rowOpList	This is a string arraylist that will get the item at index of the previous tableau in the rowOperationList, as the row operations performed on a specific tableau are displayed on the next tableau, so the index must be of the previous tableau
rowOpLabel	This takes the value of the item in each index in rowOpList so it can be added to rowOpPanel.

gridPanel	This will store the various components that go in each tab e.g. tableau, row operations, theta values and explanation, using a gridbag layout to allow varying size of components
gbc	This is a gridbagconstraints object used to define layout qualities of items added to gridPanel, such as how much space they take up, where in the panel to add them.
tableauPanel	A JPanel component that stores the tableau to be displayed, calling the createTableauPanel method.
explainPanel	This is a JPanel component that will contain the text to explain a specific tableau, how the pivot row and column are calculated, how the row operations are deduced.
addPanel	The JPanel component that will be added to explainPanel, and will contain the text explanation of the tableau
option1	This is a JTextPane component that contains the text explanation of the initial tableau. This component has been chosen over the standard JLabel as this provides better control of text wrapping and restricting dimensions. This is specific to the initial tableau, in that it doesn't reference the row operations applied in the last tableau (as there is no previous tableau).
option2	This is a JTextPane component similar to option1, but with slightly different text. It conveys the same explanation, but the text has been edited to provide multiple options for which explanation can be displayed, allowing for a more natural explanation instead of the same repetitive explanation

midOption1	This is a JTextPane component containing the text explanation for any intermediate tableau (not the initial or final tableau). This component was chosen due to the better control of text wrapping and restricting dimensions. Since it is for intermediate tableaux, it references the previous tableau and how the current tableau being displayed was reached by performing row operations on the previous tableau.
midOption2	This is a JTextPane component, similar to option2, used to provide variety in the explanation options for more natural explanation, instead of repetition. This alternative option is for the intermediate tableau (not the initial or final tableau).
finalOption1	This is a JTextPane component used to explain the final tableau after the algorithm has been performed. Again, this component was chosen due to the better control of text wrapping and restricting dimensions. Since this conveys very basic information, e.g. why the algorithm is complete, then outputting the values of the variables, only one option is required, compared to the multiple possible explanations for the other tableaux.
initialResponses	This is an arraylist of JTextPane components that stores the possible explanations for the initial tableau. A random number will be generated which references an index in this array, and the contents of that index will be added to the addPanel to be returned
intermediateResponses	This is an arraylist of JTextPane

	components that stores the possible explanations for the intermediate tableaux. A random number will be generated and will reference an index in this array, and the contents of that index will be added to the addPanel to be returned
pivotCol	The integer variable in getCurrentPivotCol that will contain the index of the pivot column of a specific tableau, and is returned
pivotRow	The integer variable in getCurrentPivotRow. It will contain the index of the pivot row of a specific tableau and is returned by the function
bvLabel	A JLabel component that will contain the text 'B.V.' to indicate the column of basic variables in the tableau.
colHeaderLabel	A JLabel component that will take the value of each string in the list of colHeaders, which will then be added to the tableauPanel to be displayed on the GUI
rowHeaderLabel	A JLabel component that will take the value of each string in the rowHeaders list, so it can then be added to the tableauPanel and displayed on the GUI.
newHeader	This is a string array in the Iterate class that gets the row headers of the last tableau and then has the appropriate row header changed according to the pivot row and column.
inequalityOptions	This is a string array the contains the options for the number of inequalities that the user can select. These options are added to the drop-down menu.
separator	This is a JSeparator object which creates a line underneath the simButtonPanel to create a sense of

	separation between the buttons and the windows to ensure the GUI remains clearly laid out and ordered, and that there aren't too many components directly next to each other.
plus1	This is a JLabel that contains the string "+", and is placed between the label of one text area and the next text area, so it appears to the user as a proper inequality to enter e.g. [textarea] x + [textarea] y.
objectivePanel	This is a separate JPanel to contain the text areas and labels for the objective function. Since the format of the objective function varies from that of the inequalities ( inequalities are [value]x + [value]y <= [value], whereas the objective function is P = [value]x + [value]y ), they cannot be added in the same way, so another panel is needed for just the objective function.
rowPanel	This is a JPanel with a flow layout (components are added one after the other in a line) that is used for each inequality. The text areas and labels are added to create the effect of inequalities line by line.
plus2, plus 3	These are JLabels similar to plus1, but since the same component can't be added twice, more are needed for cases where more than one plus symbol is needed, e.g. when there is x, y and z.
plusObj	This is a JLabel similar to plus1, but for the objective panel, since the same component cannot be added multiple times to the GUI, a separate label containing '+' is needed.

plus1Obj, plus2Obj, plus3Obj	These are JLabels similar to plusObj, used for the objective panel instead, since the same component can't be added twice.
parsePanel	This is a JPanel component that is assigned to each component in inequality panel. It will be used as part of a nested for loop in which each item of the inequality panel is parsed over and checked to see if it is a text area, in which case the contents are stored
columnIndex	This is an integer introduced to ensure that the contents of a text area are added to the correct index in the 2d array of constraints. Using the predefined loop counter would mean that if the text area is in index 7 of the panel components, then trying to add the contents to this index in the 2d array may result in an index out of bounds error if the array size is smaller than the number of components in the inequality panel.
component	This is an object of type Component and is assigned to the current component being iterated over in the parsePanel. This is then checked to see if it is a text area, and if so then the contents are stored.
objective	This is an arraylist of doubles used to store the values of the objective function once they have been retrieved from the GUI text areas
comp	This is similar to 'component', an object of type Component, but is named differently as it is used in the objectivePanel instead of the inequalityPanel. It serves the same purpose of checking for text areas

	within the objectivePanel.
text	This is a string that takes the value of the text stored in a JTextArea in the objectivePanel, which can then be converted to a double and added to the objective arraylist.
problem	This is an object of type Simplex, which has the constraints and objective passed as parameters into it, before it is solved by the Iterate class.
tableauPanel	This is a JPanel that contains the panel returned when running the displayTableau method, so it will contain the visual tableau to be added to the GUI
tableauWrapper	This is a JPanel to which the tableauPanel is added to. Its purpose is to add some padding around the tableau to make it look more natural when added to other components like the JTextPane, so that it isn't pushed up/overlapping with other components
combinedPanel	This is a new JPanel which will contain the combination of the inequality panel, as well as the tableauWrapper panel, which contains the tableauPanel. This overall panel is then added to the main window in the center.

Having outlined the methods, attributes and all variables to be used in this iteration, I can move on to writing and explaining the pseudocode, providing a baseline for the development section. The pseudocode below highlights the key parts of development in this iteration, and I will explain the purpose of each method and why it will be developed in the way it is.

### Pseudocode

Method getRowOperations(tableau):

```

ArrayList[String] rowOps -> new ArrayList // create a new list for each call

Integer pivotC -> getCurrentPivotCol(tableau) // getting the pivot column of
the tableau

Integer pivotR -> getCurrentPivotRow(pivotC, tableau)

for i -> 0 to tableau.length - 1:

    Double val -> tableau[i][pivotC]

    if i != pivotR:

        if val > 0: // checking if the value in the pivot column is positive

            if val MOD 1 == 0: // checking if value in pivot column is integer

                String formatting -> Double.toString(Abs(val))

                formatting.replace(".0", "") // if integer then it removes the .0 from
the end of the double

                rowOps.add("R" + (i + 1) + " - " + formatting + "R" + pivotR) // it is
positive, so it has been converted to positive then the string '-' is put in front of it

            else:

                rowOps.add("R" + (i + 1) + " - " + String.format("%.1f", (Abs(val))) + "R"
+ pivotR) // if it's not an integer, then round the absolute value and format the row
operation

        else: // if negative then repeat the same process but add a '+' sign instead

            if val MOD 1 == 0:

                String formatting -> Double.toString(Abs(val))

                formatting.replace(".0", "")

                rowOps.add("R" + (i + 1) + " + " + formatting + "R" + pivotR)

```

```

else:

    rowOps.add("R" + (i + 1) + " + " + String.format("%.1f", (Abs(al))) + "R" +
pivotR)

else: // this case is for if the value is in the pivot row

    if val MOD 1 == 0:

        String formatting -> Double.toString(Math.abs(val))

        formatting.replace(".0", "") // remove the .0 if the value is an integer

        rowOps.add("R" + (pivotR + 1) + " / " + formatting) // format the row
operation for the pivot row

    else:

        rowOps.add("R" + (pivotR + 1) + " / " + String.format("%.1f",
(Math.abs(val)))) // no need to handle negative cases as the pivot value can never
be negative

return rowOps // return the new list

```

Method makeTable(Tableau copyTableau) -> double[][]:

```

double[][] copyTable -> new double[rows][cols] // creates a new 2d array to
store the deep copy of the table

double[][] existingTable -> copyTableau.getTable() // assigns the parameter to a
2d array

for Integer i from 0 to existingTable.length - 1:

    for Integer j from 0 to existingTable[i].length - 1:

        copyTable[i][j] -> existingTable[i][j] // copies the array

```

```

return copyTable

Method getThetaValues(double[][] tableau) -> ArrayList<Double>:

thetaValues.clear() // clears existing theta values

Integer pivotC -> getCurrentPivotCol(tableau) // get the pivot column index

for Integer i from 0 to tableau.length - 1:

    if tableau[i][pivotC] > 0:

        double first -> tableau[i][cols - 1] // the value column is at cols - 1

        double second -> tableau[i][pivotC] // the pivot column value

        double theta -> first / second // calculate theta

        thetaValues.add(theta) // add the calculated theta value to the list

    else:

        thetaValues.add(Double.POSITIVE_INFINITY) // handles cases where
division by zero would occur

    return thetaValues

```

```

Method getCurrentPivotCol(double[][] tableau) -> Integer:

Integer pivotCol -> 0

double minValue -> tableau[rows - 1][0] // sets the minimum value as the
bottom left value of the table

for Integer i from 0 to cols - 2: // iterates through objective row checking if next
value smaller (more negative) than current minValue

    if tableau[rows - 1][i] < minValue:

        minValue -> tableau[rows - 1][i] // if true, sets minValue as this new value

        pivotCol -> i // sets index of pivot column

```

```
return pivotCol
```

Method getCurrentPivotRow(Integer pivotCol, double[][] tableau) -> Integer:

```
Integer pivotRow -> 0
```

```
double minRatio -> Double.MAX_VALUE // sets value of new variable equal to  
the max value offered by java
```

```
for Integer i from 0 to rows - 2:
```

```
    double ratio -> tableau[i][cols - 1] / tableau[i][pivotCol] // divides value column  
by pivot column
```

```
    if ratio > 0 and ratio < minRatio: // if result > 0 and < current minRatio then  
sets minRatio to current value
```

```
        minRatio -> ratio
```

```
        pivotRow -> i // sets index of pivot row
```

```
    pivotVal -> tableau[pivotRow][pivotCol]
```

```
return pivotRow
```

Method runSimplexAlg(Integer numInequality, Integer numVAR):

```
constraints -> new double[numInequality][numVAR + 1] // 2d array of  
constraints
```

```
try:
```

```
    for Integer i from 0 to numInequality - 1: // iterating through each set of  
inequalities
```

```
        JPanel parsePanel -> (JPanel) inequalityPanel.getComponent(i) // creates a  
new panel object to extract components from
```

```
        Integer columnIndex -> 0 // introducing the columnIndex counter
```

```
        for Integer j from 0 to parsePanel.getComponentCount() - 1: // iterating  
through the panel
```

```

Component component -> parsePanel.getComponent(j)

if component instanceof JTextArea: // checks if the current component is
a text area

    if columnIndex < constraints[i].length:

        constraints[i][columnIndex] -> Double.parseDouble(((JTextArea)
component).getText()) // adds the value in the text area to the array

        columnIndex++


JPanel objectivePanel -> (JPanel)
inequalityPanel.getComponent(numInequality) // creates a new objective panel

ArrayList<Double> objective -> new ArrayList<>() // creating an arraylist to
store the objective function

for Integer k from 0 to objectivePanel.getComponentCount() - 1:

    Component comp -> objectivePanel.getComponent(k)

    if comp instanceof JTextArea: // checks if the current component is a
JTextArea

        String text -> ((JTextArea) comp).getText()

        objective.add(Double.parseDouble(text)) // converts the text to a double


errorMessageLabel.setText("")

for Double value in objective: // printing the objective values for debugging

    print(value)

obj -> new double[numVAR] // array to hold the objective function

```

```
for Integer l from 0 to objective.size() - 1:  
    obj[l] -> objective.get(l) // converting the arraylist into an array to be passed  
    as a parameter  
  
    Simplex problem -> new Simplex(constraints, obj) // creating a new object of  
    type Simplex  
    Iterate.solve(problem) // solving the created problem  
  
JPanel combinedPanel -> new JPanel()  
combinedPanel.setLayout(new BorderLayout()) // use BorderLayout for better  
control  
  
JPanel tableauWrapper -> new JPanel()  
tableauWrapper.setLayout(new BorderLayout())  
tableauWrapper.setBorder(BorderFactory.createEmptyBorder(10, 10, 75,  
50)) // adjust top and left padding  
  
JPanel tableauPanel -> Tableau.displayTableau(rowHeaders, colHeaders,  
numInequality + 1, numVAR + numInequality + 1)  
  
tableauWrapper.add(tableauPanel, BorderLayout.CENTER) // add tableau to  
the wrapper  
combinedPanel.add(inequalityPanel, BorderLayout.CENTER) // add the  
inequality panel  
combinedPanel.add(tableauWrapper, BorderLayout.SOUTH) // add the  
tableau wrapper to the bottom
```

```

        add(combinedPanel, BorderLayout.CENTER) // add the combined panel to the
main frame

revalidate() // refresh the GUI

repaint()

catch NumberFormatException e:

    errorMessageLabel.setText("please enter valid numbers in all the text areas.")
// displays an error message if invalid numbers are entered

```

The pseudocode covers the methods that involve a major element of logic, omitting those that are predominantly focused on adding components to the GUI and formatting those components. Having outlined the pseudocode for the logic-based methods to be developed in this iteration, I now have a baseline idea as to how each method should be structured and what needs to be included to ensure functionality. A key method from the pseudocode is the runSimplexAlg method, as this is the bridge between the GUI inputs and the inputs to the Simplex algorithm. This method will be responsible for converting the inputs taken from the text areas into the general format created in Iteration 2, so that all the information for the inequalities and objective function is contained within a single data structure, which can be used by the Simplex methods.

## **Test Plan**

To ensure that the various features of this iteration all work as intended, a variety of tests will be required to ensure proper functionality. The testing for this iteration will contain two main sections, one to test the GUI and input validation, and another to test mathematical accuracy, ensuring that the methods developed in Iteration 2 still work when inputs are taken from the GUI and formatted to be used. The questions and examples used for these tests are from the Pearson ActiveLearn Decision 1 Textbook

(<https://www.pearsonactivelearn.com/app/library/ebook?>

[id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](#)) and the answers are either taken from the book, or from the Pearson ActiveTeach Solutionbank, which provides worked solutions for all the questions from the textbook ([https://activeteach-prod.resource.pearson-intl.com/r00/r0072/r007257/r00725745/current/alevelsb\\_dm1\\_ex7b.pdf](https://activeteach-prod.resource.pearson-intl.com/r00/r0072/r007257/r00725745/current/alevelsb_dm1_ex7b.pdf))

Test Number/ Description	Test Type	Expected Outcome
1a) Testing the 2-inequality drop-down menu option with the 2-variable button  1b) Testing the 2-inequality drop-down option with the 3-variable button.  1c) Testing the 2-inequality drop-down option with the 4-variable button.	All 3 are normal test types	1a) 2 rows of inequalities and one objective row should be displayed with x and y as variables.  1b) 2 rows of inequalities and one objective row should be displayed with x, y and z as variables  1c) 2 rows of inequalities and one objective row should be displayed with x <sub>1</sub> , x <sub>2</sub> , x <sub>3</sub> and x <sub>4</sub> as variables.
2a) Testing the 3-inequality drop-down menu option with the 2-variable button  2b) Testing the 3-inequality drop-down option with the 3-variable button.  2c) Testing the 3-inequality drop-down option with the 4-variable button.	All 3 are normal test types	2a) 3 rows of inequalities and one objective row should be displayed with x and y as variables.  2b) 3 rows of inequalities and one objective row should be displayed with x, y and z as variables  2c) 3 rows of inequalities and one objective row should be

		displayed with x1, x2, x3 and x4 as variables.
3a) Testing the 4-inequality drop-down menu option with the 2-variable button  3b) Testing the 4-inequality drop-down option with the 3-variable button.  3c) Testing the 4-inequality drop-down option with the 4-variable button.	All 3 are normal test types	3a) On the main window, 4 rows of inequalities and one objective row should be displayed with x and y as variables.  3b) On the main GUI window, 4 rows of inequalities and one objective row should be displayed with x, y and z as variables  3c) On the main window, 4 rows of inequalities and one objective row should be displayed with x1, x2, x3 and x4 as variables.
4a) Trying to select the 2-variable button before the number of inequalities 4b) Trying to select the 3-variable button before the number of inequalities 4c) Trying to select the 4-variable button before the number of inequalities	All 3 of these tests are erroneous	The red error message JLabel should be displayed on the GUI indicating to the user that they must first select the number of inequalities when each of the buttons are pressed before selecting the number of inequalities
5) Selecting the 'Run' button before selecting the number of variables.	Erroneous	The red error message label should be displayed next to the buttons, indicating to

		the user that they must first select the number of variables before running.
6) Selecting the 'Run' button before selecting the number of inequalities and variables	Erroneous	The red error message label should be displayed next to the buttons, indicating to the user that they must first select the number of inequalities and variables before running.
7a) Trying to select 'Run' before entering any values into the text areas  7b) Trying to select 'Run' having only entered some values into the text areas	Erroneous	For both 7a and 7b, the error message label should appear on the GUI next to the buttons, prompting the user to enter valid values into the text areas
8a) Entering invalid characters in some of the text areas (e.g. letters)  8b) Entering invalid characters in all the text areas (e.g. letters)	Erroneous	For both 8a and 8b, the red error message label should be displayed on the GUI, informing them to enter valid numbers in the text areas. With the remedial actions from test 7a and 7b, the new message should be displayed
9) Chapter 7 Example 8 from Pearson ActiveLearn Decision 1 Textbook  Refer to Figure 1a below	Normal	The final values of variables and the total profit should be P = 26, x = 42/11 (3.82), y = 80/11 (7.23), r = 0 and s = 0.

		The final tableau should have the row headers changed to y and x respectively.
10) Chapter 7 Example 10 from Pearson ActiveLearn Decision 1 Textbook  Refer to Figure 2a below	Normal	The final values of variables and the total profit should be P = 45, x = 0, y = 2.5, z = 1.88, r = 0 and s = 0.  The final tableau should have the row headers changed to y and z respectively. There should be three iterations. The pivot column in order of iterations should be y then z and the pivot row in order of iterations should be r then s.
11) Chapter 7 Example 11 from Pearson ActiveLearn Decision 1 Textbook  Refer to Figure 3a below		The final values of variables and the total profit should be P = 144/7 (20.57), x = 0=32/7 (4.57), y = 12/7 (1.71), z = 0, r = 0, s = 0 and t = 30/7 (4.26)  The final tableau should have the row headers changed to x,y and t respectively. There should be three iterations. The pivot column in order of iterations should be y then x and the pivot row in order of iterations should be s then r.

<p>12) Chapter 7 Exercise 7B          Question 1 from Pearson ActiveLearn Decision 1 Textbook          Refer to Figure 4a below</p>	<p>Normal</p>	<p>The final values of the variables and profit should be <math>P = 38</math>, <math>x = 0</math>, <math>y = 3</math>, <math>z = 5</math>, <math>r = 0</math>, <math>s = 0</math>. There should be 3 iterations of the algorithm. The pivot row through the iterations should be <math>r</math> then <math>s</math>. The pivot column through the iterations should be <math>y</math> then <math>z</math>. The row headers should change to a final state of <math>y</math>, <math>z</math>.</p>
<p>13) Chapter 7 Exercise 7B          Question 4 from Pearson ActiveLearn Decision 1 Textbook          Refer to Figure 5a below</p>	<p>Normal</p>	<p>The final values of variables and the total profit should be <math>P = 105.5</math>, <math>x_1 = 19.25</math>, <math>x_2 = 0</math>, <math>x_3 = 14.25</math>, <math>x_4 = 0</math>, <math>r = 33</math>, <math>s = 0</math> and <math>t = 27.25</math>, <math>u = 0</math>. The final tableau should have the row headers changed to <math>r</math>, <math>x_3</math>, <math>t</math> and <math>x_1</math> respectively. There should be three iterations. The pivot column in order of iterations should be <math>x_1</math> then <math>x_3</math> and the pivot row in order of iterations should be <math>u</math> then <math>s</math>.</p>
<p>14) Chapter 7 Exercise 7B          Question 2 from Pearson ActiveLearn Decision 1 Textbook</p>	<p>Normal</p>	<p>The final values of the variables and profit should be <math>P = 260</math>, <math>x = 0</math>, <math>y = 40</math>, <math>z = 10</math>, <math>r = 0</math>, <math>s = 0</math>.</p>

Refer to Figure 6a below		There should be 3 iterations of the algorithm. The pivot row through the iterations should be s then y. The pivot column through the iterations should be y then z. The row headers should change to a final state of y, z.
15) Chapter 7 Exercise 7B Question 3 from Pearson ActiveLearn Decision 1 Textbook Refer to Figure 7a below	Normal	The final values of the variables and the maximised profit should be $P = 11$ , $x = 2$ , $y = 1$ , $z = 0$ , $r = 0$ , $s = 0$ , $t = 1$ . There should be three iterations of the algorithm. The pivot rows should be s then r, the pivot columns should be y then x and the row headers should change to a final state of x, y, t.

Figure 1a

**Example 8**

Solve the linear programming problem in Example 6 using simplex tableaux.

$$\text{Maximise } P = 3x + 2y$$

subject to:

$$5x + 7y + r = 70$$

$$10x + 3y + s = 60$$

$$x, y, r, s \geq 0$$

**Notation** The word **tableau** is French. The plural of tableau is **tableaux**.

Figure 1b

$P = 26$ ,  $y = \frac{80}{11}$  and  $x = \frac{42}{11}$  and all other variables, and slack variables, are zero.

So our full solution is

$$P = 26, x = \frac{42}{11}, y = \frac{80}{11}, r = 0, \text{ and } s = 0$$

Figure 2a

**Example 10**

- A a Use simplex tableaux to solve the linear programming problem below (from Example 7).

$$\text{Maximise } P = 10x + 12y + 8z$$

subject to:

$$2x + 2y \leq 5$$

$$5x + 3y + 4z \leq 15$$

$$x, y, z \geq 0$$

Figure 2b

The optimal solution is

$$P = 45, x = 0, y = 2\frac{1}{2}, z = 1\frac{7}{8}, r = 0, s = 0.$$

Figure 3a

**Example 11**

- a Use the simplex tableau method to solve the following linear programming problem.

$$\text{Maximise } P = 3x + 4y - 5z$$

subject to:

$$2x - 3y + 2z + r = 4$$

$$x + 2y + 4z + s = 8$$

$$y - z + t = 6$$

$$x, y, z, r, s, t \geq 0$$

Figure 3b

---

$$P = \frac{144}{7}, x = \frac{32}{7}, y = \frac{12}{7}, z = 0, r = 0, s = 0, t = \frac{30}{7}$$

Figure 4a

**Exercise 7B**

Solve the linear programming problems in questions **1** to **6** using the simplex tableau algorithm.

- 1** Maximise  $P = 5x + 6y + 4z$   
subject to

$$\begin{aligned}x + 2y + r &= 6 \\5x + 3y + 3z + s &= 24 \\x, y, z, r, s &\geq 0\end{aligned}$$

Figure 4b

## The simplex algorithm 7B

1

b.v.	x	y	z	r	s	value	θ values
r	1	(2)	0	1	0	6	3 *
s	5	3	3	0	1	24	8
P	-5	-6	-4	0	0	0	

b.v.	x	y	z	r	s	value	Row operations
y	$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	3	R1 ÷ 2
s	$\frac{7}{2}$	0	(3)	$-\frac{3}{2}$	1	15	R2 - 3R1
P	-2	0	-4	3	0	18	R3 + 6R1

b.v.	x	y	z	r	s	value	Row operations
y	$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	3	R1 (no change)
z	$\frac{7}{6}$	0	1	$-\frac{1}{2}$	$\frac{1}{3}$	5	R2 ÷ 3
P	$\frac{8}{3}$	0	0	1	$\frac{4}{3}$	38	R3 + 4R2

$$P = 38 \quad x = 0 \quad y = 3 \quad z = 5 \quad r = 0 \quad s = 0$$

Figure 5a

- A 4** Maximise  $P = 4x_1 - 3x_2 + 2x_3 + 3x_4$   
P subject to

$$\begin{aligned} x_1 + 4x_2 + 3x_3 + x_4 + r &= 95 \\ 2x_1 + x_2 + 2x_3 + 3x_4 + s &= 67 \\ x_1 + 3x_2 + 2x_3 + 2x_4 + t &= 75 \\ 3x_1 + 2x_2 + x_3 + 2x_4 + u &= 72 \\ x_1, x_2, x_3, x_4, r, s, t, u &\geq 0 \end{aligned}$$

**Hint** There are 4 decision variables and 4 slack variables. Your initial tableau will need rows for  $r, s, t$  and  $u$ , and columns for  $x_1, x_2, x_3, x_4, r, s, t$  and  $u$ .

Figure 5b

4

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	$\theta$ values
$r$	1	4	3	1	1	0	0	0	95	95
$s$	2	1	2	3	0	1	0	0	67	$33\frac{1}{2}$
$t$	1	3	2	2	0	0	1	0	75	75
$u$	3	2	1	2	0	0	0	1	72	24
$P$	-4	3	-2	-3	0	0	0	0	0	

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	Row operations
$r$	0	$3\frac{1}{3}$	$2\frac{2}{3}$	$\frac{1}{3}$	1	0	0	$-\frac{1}{3}$	71	R1 - R4
$s$	0	$-\frac{1}{3}$	$\frac{4}{3}$	$1\frac{2}{3}$	0	1	0	$-\frac{2}{3}$	19	R2 - 2R4
$t$	0	$2\frac{1}{3}$	$1\frac{2}{3}$	$1\frac{1}{3}$	0	0	1	$-\frac{1}{3}$	51	R3 - R4
$x_1$	1	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	$\frac{1}{3}$	24	R4 $\div 3$
$P$	0	$5\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{1}{3}$	0	0	0	$\frac{4}{3}$	96	R5 + 4R4

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	$\theta$ values
$r$	0	$3\frac{1}{3}$	$2\frac{2}{3}$	$\frac{1}{3}$	1	0	0	$-\frac{1}{3}$	71	$26\frac{5}{8}$
$s$	0	$-\frac{1}{3}$	$\frac{4}{3}$	$1\frac{2}{3}$	0	1	0	$-\frac{2}{3}$	19	$14\frac{1}{4}$
$t$	0	$2\frac{1}{3}$	$1\frac{2}{3}$	$1\frac{1}{3}$	0	0	1	$-\frac{1}{3}$	51	$30\frac{3}{5}$
$x_1$	1	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	$\frac{1}{3}$	24	72
$P$	0	$5\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{1}{3}$	0	0	0	$\frac{4}{3}$	96	

4 continued

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	Row operations
$r$	0	4	0	-3	1	-2	0	1	33	$R1 - 2\frac{2}{3}R2$
$x_3$	0	$-\frac{1}{4}$	1	$1\frac{1}{4}$	0	$\frac{3}{4}$	0	$-\frac{1}{2}$	$14\frac{1}{4}$	$\frac{3}{4}R2$
$t$	0	$2\frac{3}{4}$	0	$-\frac{3}{4}$	0	$-1\frac{1}{4}$	1	$\frac{1}{2}$	$27\frac{1}{4}$	$R3 - 1\frac{2}{3}R2$
$x_1$	1	$\frac{3}{4}$	0	$\frac{1}{4}$	0	$-\frac{1}{4}$	0	$\frac{1}{2}$	$19\frac{1}{4}$	$R4 - \frac{1}{3}R2$
$P$	0	$5\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$105\frac{1}{2}$	$R1 + \frac{2}{3}R2$

$$\text{Maximum } P = 105\frac{1}{2}, \text{ when } x_1 = 19\frac{1}{4}, x_2 = 0, x_3 = 14\frac{1}{4}, x_4 = 0, r = 33, s = 0, t = 27\frac{1}{4}, u = 0$$

Figure 6a

2 Maximise  $P = 3x + 4y + 10z$   
subject to

$$x + 2y + 2z + r = 100$$

$$x + 4z + s = 40$$

$$x, y, z, r, s \geq 0$$

Figure 6b (There were some slight formatting issues on the mark scheme, however all the necessary information is still clearly visible)

2

b.v.	x	y	z	r	s	value	θ values
r	1	2	2	1	0	100	50
s	1	0	4	0	1	40	10*
P	-3	-4	-10	0	0	0	

b.v.	x	y	z	r	s	value	Row operations
y	$\frac{1}{2}$	2	0	1	$-\frac{1}{2}$	80	R1 - 2R2
z	$\frac{1}{4}$	0	1	0	$\frac{1}{4}$	10	R2 ÷ 4
P	$-\frac{1}{2}$	-4	0	0	$\frac{5}{2}$	100	R3 + 10R2

b.v.	x	y	z	r	s	value	Row operations
y	$\frac{1}{4}$	1	0	$\frac{1}{2}$	$-\frac{1}{4}$	40	R1 ÷ 2
z	$\frac{1}{4}$	0	1	0	$\frac{1}{4}$	10	R2 (no change)
P	$\frac{1}{2}$	0	0	$\frac{3}{2}$	0	60	R3 + 4R1

$$P = 260 \quad x = 0 \quad y = 40 \quad z = 10 \quad r = 0 \quad s = 0$$

Figure 7a

3 Maximise  $P = 3x + 5y + 2z$   
subject to

$$\begin{aligned} 3x + 4y + 5z + r &= 10 \\ x + 3y + 10z + s &= 5 \\ x - 2y + t &= 1 \\ x, y, z, r, s, t &\geq 0 \end{aligned}$$

Figure 7b

3

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value	θ values
$r$	3	4	5	1	0	0	10	2.5
$s$	1	(3)	10	0	1	0	5	$1\frac{2}{3}*$
$t$	1	-2	0	0	0	1	1	negative pivot
$P$	-3	-5	-2	0	0	0	0	

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value	Row operations
$r$	( $\frac{5}{3}$ )	0	$-\frac{25}{3}$	1	$-\frac{4}{3}$	0	$\frac{10}{3}$	R1 - 4R2
$y$	$\frac{1}{3}$	1	$\frac{10}{3}$	0	$\frac{1}{3}$	0	$\frac{5}{3}$	R2 ÷ 3
$t$	$\frac{5}{3}$	0	$\frac{20}{3}$	0	$\frac{2}{3}$	1	$\frac{13}{3}$	R3 + 2R2
$P$	$-\frac{4}{3}$	0	$\frac{44}{3}$	0	$\frac{5}{3}$	0	$\frac{25}{3}$	R4 + 5R2

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value	Row operations
$x$	1	0	-5	$\frac{3}{5}$	$-\frac{4}{5}$	0	2	$R1 \div \frac{5}{3}$
$y$	0	1	5	$-\frac{1}{5}$	$\frac{3}{5}$	0	1	$R2 - \frac{1}{3}R1$
$t$	0	0	15	-1	2	1	1	$R3 - \frac{5}{3}R1$
$P$	0	0	8	$\frac{4}{5}$	$\frac{3}{5}$	0	11	$R4 - \frac{4}{3}R1$

$$P = 11 \quad x = 2 \quad y = 1 \quad z = 0 \quad r = 0 \quad s = 0 \quad t = 1$$

## Iteration 5: Kruskal's Algorithm with Visual Graphs

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

Having created the basis for Kruskal's algorithm, taking user inputs from the GUI and outputting an answer in the console, I can now work on displaying the graph with visualised vertices and edges to the user. This will contribute towards success criteria 6, which outlines allowing the user to create their own graph to perform an algorithm on. Combining the graph creation feature with the methods for Kruskal's algorithm in Iteration 3, we will be able to fully integrate Kruskal's algorithm into the GUI. Having first created a GUI shell, upon which components can be displayed, then creating the methods for Kruskal's algorithm, this iteration will aim to bring them together, using various techniques that I have learnt throughout the previous iterations.

The first part of this iteration will involve displaying a basic circle on the GUI which will go on to represent the nodes. For this I will develop code in a separate class on a basic JFrame, before implementing the graph display into the main GUI. After this, I will work on displaying 2 circles connected by a line. This will form an edge of the graph. Currently, the user is prompted to enter a source node, destination node and a weight. These three pieces of data will be vital for displaying the graph visually, as each node will have to be labelled, and the weight of the edge will need to be added next to the arc.

One important feature to consider is the joining of arcs in the graph display. For example, if the user enters AB, then AC, it shouldn't create 2 disconnected edges with 2 'A' nodes, instead it should join the two edges at node A. To do this, I will first take a list of edges as a parameter e.g. {"AB", "BC", "AC", "CD", "BD"}, before iterating through the list, separating the source and destination node and adding them to a list of nodes labels. Before adding the node to the list, I will first check whether it is already in the list or not, and if it is, a second copy won't be added to

ensure that any subsequent edges that are connected to this node only use the single position created. This will avoid multiple of the same node being created.

The next feature to consider is how the nodes will be formatted on the GUI. This graph display feature will allow users to move the nodes around to allow them to create the shape of a graph they may be testing the algorithm on; however, I need an initial position to add each of the nodes and edges to, ensuring there are no overlaps. The simplest layout for this is a circle, as it ensures a varying number of nodes and vertices can be added, while ensuring each node and edge is clearly visible.

To do this, I will utilise some basic maths using the unit circle and trigonometry. I will have a method to calculate the positions of each of the nodes, based on how many unique nodes there are. To do this, I will use radian measure of angles where  $2 * \pi$  radians = 360 degrees. There will be a list of all the unique nodes added to the graph. To format each node equally in a circular shape, the angle of the node will be defined as the index in the list divided by the total size of the list \*  $2\pi$ . So, for the list of nodes {A, B, C, D, E}, the angles will be as follows:

$$A = 2\pi * 0/5 = 0$$

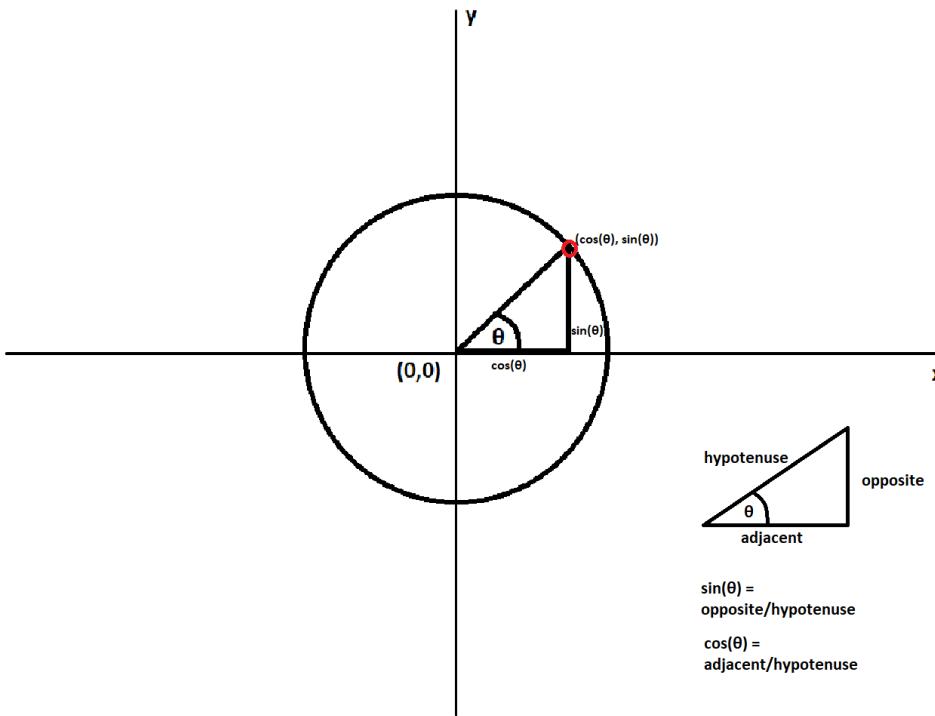
$$B = 2\pi * 1/5 = 2\pi/5$$

$$C = 2\pi * 2/5 = 4\pi/5$$

$$D = 2\pi * 3/5 = 6\pi/5$$

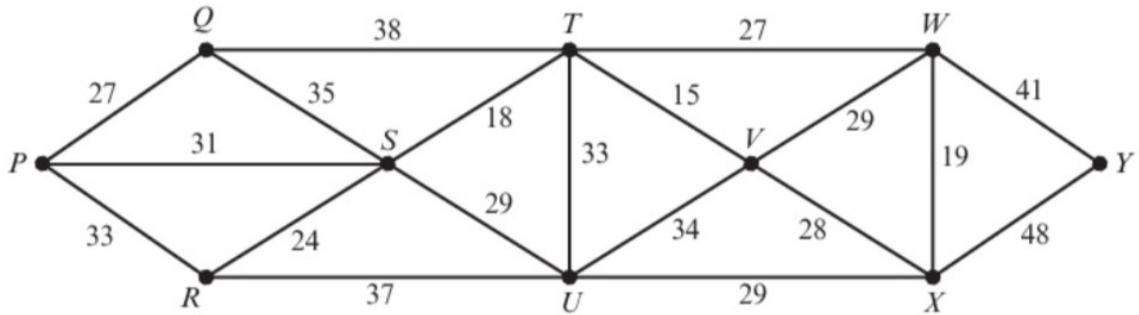
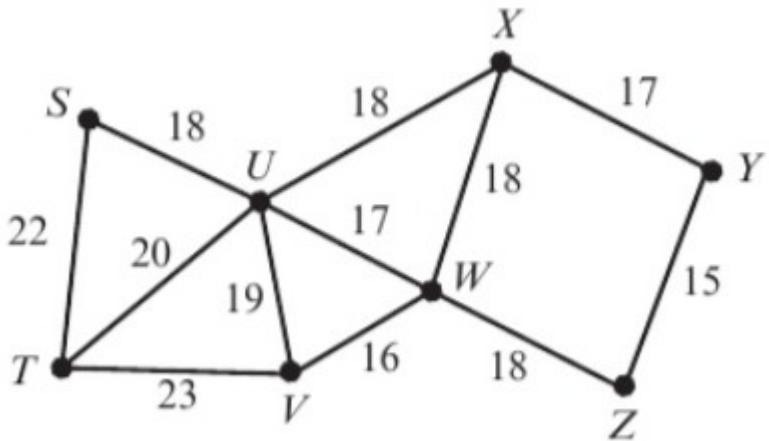
$$E = 2\pi * 4/5 = 8\pi/5$$

Using trigonometry, I can then convert the angles into coordinates with respect to the center of the frame, with the x-coordinate being  $\cos(\text{angle})$  and the y-coordinate being  $\sin(\text{angle})$ . This is demonstrated below:



Given the angle, we can now define a unique coordinate for each node, going anticlockwise as the angle  $\theta$  increases.

In the previous iterations, there hasn't been any element of drawing components on the GUI, hence I will have to learn which methods to use for each component that needs to be drawn. For the graph, four main components will need to be displayed on GUI. These are the nodes, the edges, the node label and the weight of the edge. Looking through the Pearson Edexcel ActiveLearn Textbook, the graphs are all drawn in a similar way, with circles are the nodes, lines connecting the vertices and text for the node labels and the weight of the edges.



I can therefore identify the components that will need drawing on the GUI for the graph, which will be circles, straight lines and text. Looking through the Java Documentation, I discovered how 2D graphics can be displayed:

<https://docs.oracle.com/javase/tutorial/2d/basic2d/index.html>

Looking through the section of 'Drawing Geometric Primitives', I found the methods I could use to draw these shapes, being `fillOval()`, `drawLine()` and `drawString()`. The `fillOval()` method takes in 4 parameters, the location of where to draw the component, and the width and height. Since I am trying to draw a circle, the width and height parameters will be the same. The `drawLine()` method takes in 4 parameters as well, the x and y coordinates of the start point of the line, and the x and y coordinates of the end of the line. The `drawString()` method takes in 3 parameters, the string to be drawn, and the x and y coordinates to draw it. Using these 3 methods will allow me to draw all the components I need for the graph.

Looking through the documentation again:

<https://docs.oracle.com/javase/tutorial/uiswing/painting/closer.html>, I saw that all graphics that are being drawn should be contained within one `paintComponent`

method, which takes a Graphics object as a parameter. Hence all the component drawing will be within one paintComponent method.

Another feature that isn't present in the previous iterations is the use of user input to move components on the screen. To ensure the user can move around the various nodes in the graph to make the graph look as they desire, I will need to ensure that the relevant methods to allow for components to be moved by the mouse. I separated this into 3 basic motions, the user pressing the mouse inside the drawn node circle, the user dragging the mouse once it has been pressed down and the user releasing the mouse. Referencing the Java Documentation, most methods relating to mouse movements come under the MouseListener interface within the AWT Event library.

[https://docs.oracle.com/javase/8/docs/api/java.awt.event/MouseListener.html](https://docs.oracle.com/javase/8/docs/api/java.awt/event/MouseListener.html)

From this I identified that the methods I needed were mousePressed and mouseReleased. These cover the initial click of the mouse and the release of the mouse. The one movement that none of these methods covered was the dragging motion. Another interface relating to mouse movement is the MouseMotionListener

<https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/java.awt.event/MouseMotionListener.html>, which is aimed at using the actual movement of the mouse to produce a result. Within this interface is the mouseDragged method, which is invoked when a mouse button is pressed on a component and then dragged. In these three methods, I can develop the logic to move the drawn circles to the relevant place on the user interface. Below is an outline of the logic that each method will contain:

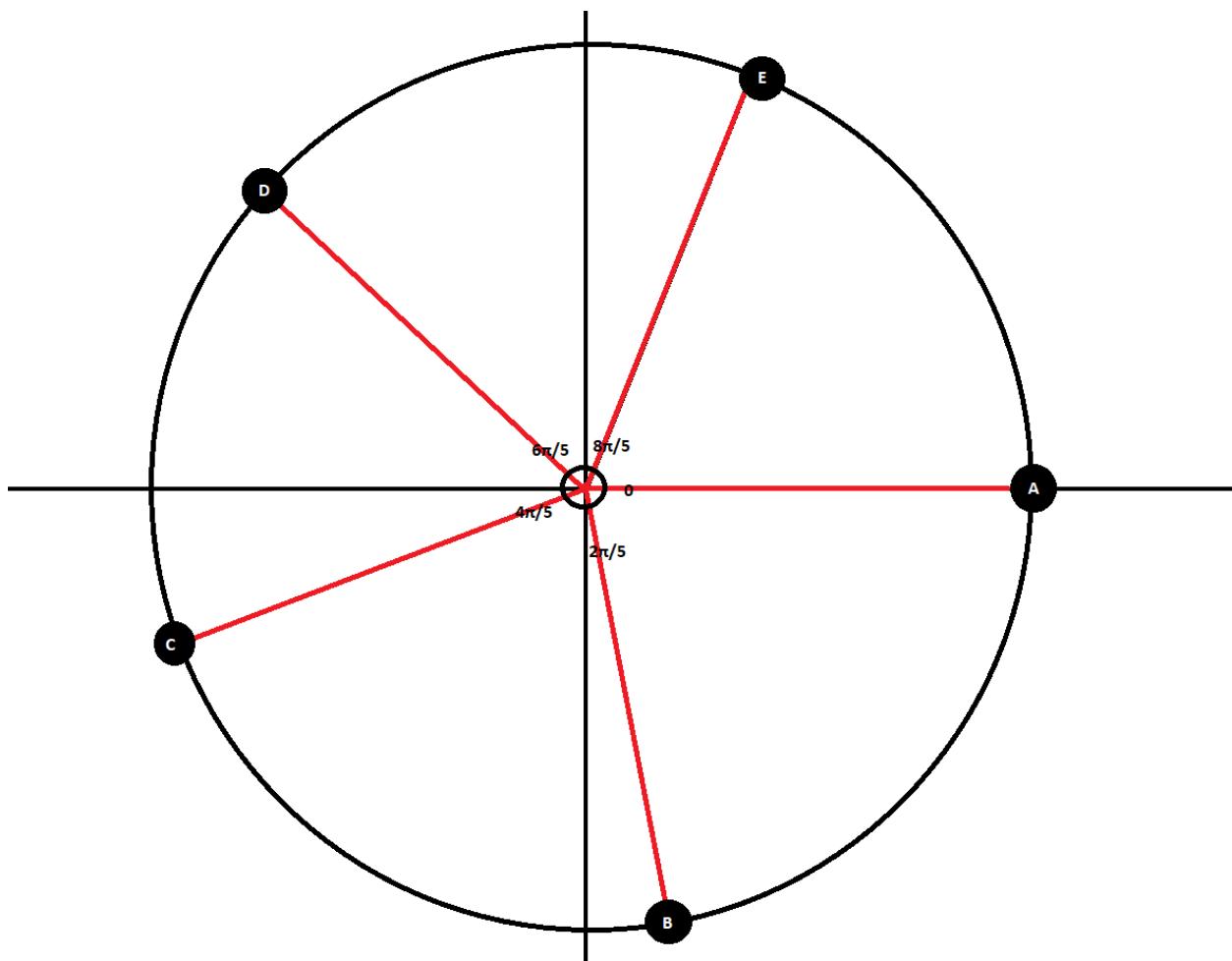
mousePressed: This will check if the user pressed the mouse inside the drawn node circle and if they have, it will assign values to where the click was, and which node was clicked based on the position.

mouseReleased: This will check the point that the user releases the mouse after dragging the node, resetting the values for the start location and the index of the node that was selected

`mouseDragged`: This will get the horizontal and vertical movement of the mouse while dragging the component, before setting the location of the component to the original location + any horizontal and vertical movement.

When reading through the Graphics documentation for Java:

<https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/java.awt/Graphics2D.html>, an interesting point I came across was that the positive y-direction in Java Graphics is considered downward, contrasting the general convention of positive being upwards. This means that when adding the nodes by increasing angle, instead of adding them in an anticlockwise order, they will be added in a clockwise order. This doesn't impact the display of the graph in any way, as the user will eventually be able to move the nodes around themselves to a desired location, however it will explain any results where the nodes are added in a clockwise manner instead of anticlockwise as the maths suggests. This can be demonstrated below:

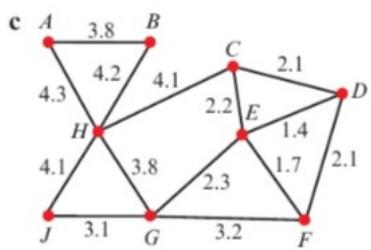
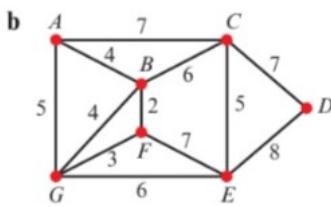
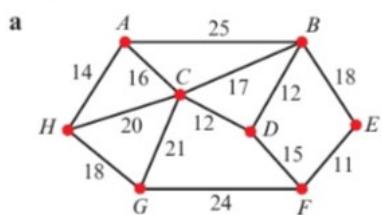


While traditionally the  $2\pi/5$  angle would be at where point 'E' is, the downward positive aspect of Java Graphics means it is  $2\pi/5$  going clockwise from the positive x-direction.

To create 'edges' between the nodes, I will use the `drawLine` method from `paintComponent`, with the start and end point as the centers of the two nodes to be connected.

This concludes the basic creation of a visual graph on the GUI taken from user inputs. Next, I need to consider where the different components of the entire explanation will be placed on the GUI. The displayed graph needs to take up a good proportion of the screen to ensure it can be seen throughout the explanation. To show working out for Kruskal's algorithm in the Edexcel A-Level Further Maths exam, [SCHOOL NAME ABBREVIATION] students are taught to lay out their work in a specific manner. This involves listing out all the edges in the graph in weight order, then writing 'Add' or 'Reject' next to the edge depending on if it is added to the minimum spanning tree, or if it is rejected from the minimum spanning tree. This gains all the marks for shown methods, as the question prompts the student to 'show the order in which they consider the arcs'. This is shown below, with a question taken from the Pearson Edexcel A-Level Decision 1 textbook:

- 1** Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.



Part of the GUI will therefore involve a table that conveys this information, an example is demonstrated below:

Arcs	Add/Reject
AB	Add
AC	Add
BD	Reject
CD	Add

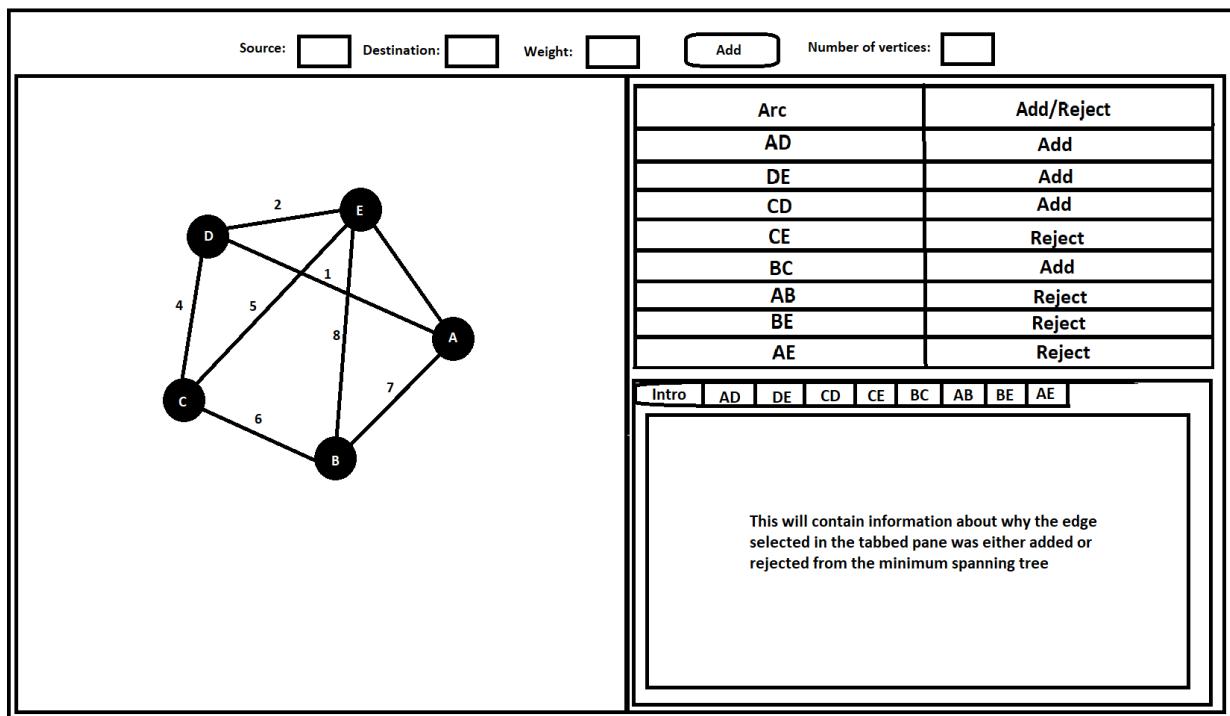
Another feature I will add to the GUI will be to allow users to click through the individual edges of the graph, providing an explanation as to why they were added/rejected from the graph, as well as highlighting the edges on the displayed graph as they click through the edges, forming the minimum spanning tree by the time they click through to the final edge. Referencing success criterion 4, the explanation for the algorithm must mention that the edges were sorted before performing the algorithm (4.1) , whether edges were added or rejected based on whether they form a cycle or not (4.2) and repeating the steps until a minimum spanning tree has formed (4.3).

To implement this I am going to use a JTabbedPane, with each tab being an edge from the graph. This method of splitting up an algorithm into steps using tabs was first implemented in Iteration 4 with the individual Simplex tableaux. A similar feature can be implemented for the edges of a graph. When the user clicks on an edge tab, it will present an explanation for why that edge was added/rejected to the graph, as well as highlighting that edge on the displayed graph. Since the edges will be added to the tabbed pane in sorted order by ascending weight, I need to ensure that if the user presses one of the middle tabs directly, then all the previous edges that were added/rejected are also highlighted on the graph, and if the user has clicked through all of them and decides to select one of the previous

edge tabs, it returns the highlights on the graph back to the state that the minimum spanning tree was once that many edges had been considered.

All edges that are added to the minimum spanning tree will be highlighted in green one-by-one to show how the graph is built up. Any edges that are rejected from the minimum spanning tree will be highlighted in red, creating a clear distinction between the final minimum spanning tree and the edges that were rejected. These colour choices will be confirmed by consulting my stakeholders, who will be able to provide any suggestions or improvements they feel would better suit students using the system.

Below is a mockup of the main section of the GUI with all the above features.



I consulted my stakeholders regarding the GUI above, as well as the colour options and whether they thought it served the purpose of explaining and performing the algorithm.

Question	Student 1	Student 2	Student 3
Do you think the GUI mockup above clearly displays all the	I think the mockup has all the key elements to explain the	The GUI mockup contains all the necessary information,	I think the GUI mockup does clearly display all the necessary

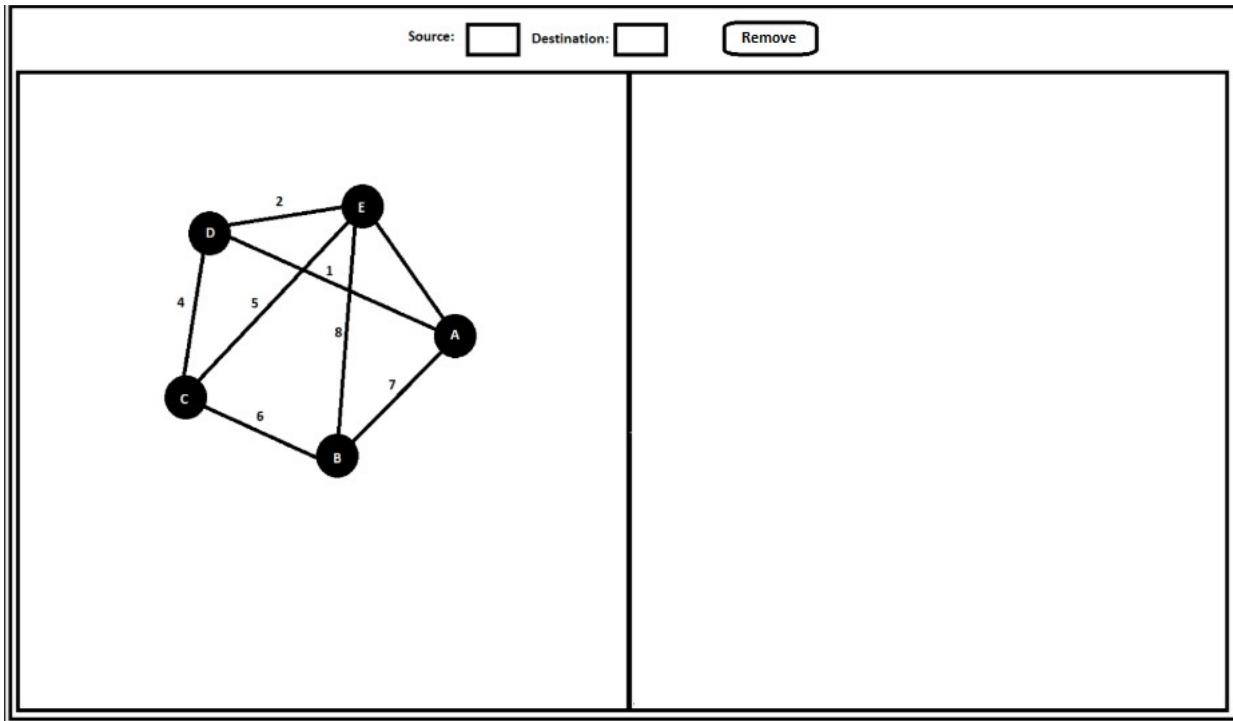
necessary information regarding Kruskal's algorithm for the Edexcel A-Level Decision maths exam	algorithm. I like how the graph is displayed next to the working out so it can be referenced while reading the working method	especially the table of added and rejected arcs as this is the key working out method that we need to show	information. The graph and table show the necessary working out for the exam and the explanation underneath will help for understanding the algorithm
When highlighting the edges of the minimum spanning tree, do you think that using a shade of green to indicate an edge was added, and a shade of red to indicate an edge was rejected is suitable and clear.	I think those colour options would definitely fit the purpose of the tool and would further help me visualise the algorithm as it is being performed on the graph.	I think the colour choices are suitable. For me, green clearly serves the purpose of indicating an edge was added and red has the same effect but for rejecting the edges	I think that the colour choices are definitely suitable. They would help highlight the graph and show the user what to focus on while performing the algorithm. It may be worth adding something to explain the colours' purpose to ensure there isn't any confusion.

As shown, all three stakeholders agree that the GUI contains all the necessary aspects to show students the method for the exam as well as explaining why each edge was added/rejected to the GUI, combined with a visual element with the graph to aid visual learners. The stakeholders also all agree that the use of green to indicate the adding of edges and the red to indicate the edges being rejected is suitable. Focusing on Student 3's response where they mention including something that indicates the colour's purpose justifies that the introductory panel of the tabbed pane can be used to first explain how to click through the edges, but

also what the highlighting colour options mean. This ensures that all students understand the use of colours, since for some students, green may not immediately indicate accepting or adding an edge to the graph, and red may not immediately suggest rejecting an edge from the graph.

Success criterion 6.1 mentions having buttons to add and remove nodes from the displayed graph, with 6.2 mentioning a separate button to connect them and assign a weight. However, given that success criterion 1 outlines ensuring a clearly laid out GUI that isn't cluttered with too many components, I made the decision to combine the functionality of these buttons into an 'Add Edge' and 'Remove Edge' button. The 'Add Edge' button was developed in Iteration 3 and allowed the user to enter a source and destination node as well as a weight into separate text areas, before adding this information to the list of edges to perform the algorithm on.

In this iteration, I will also give functionality to the 'Remove Edge' button, which will allow users to enter a source node and destination node. When clicking the 'Remove Edge' button, two text areas will be displayed, one for the source node and one for the destination node, as well as a button to confirm the removal of the edge. When clicking remove, this edge will then be removed from the visual graph display, as well as any lists in which the edges of the graph are stored. Below is a mockup of what the main section of the GUI below the separator line should look like:



As shown, most of the components have remained the same e.g. the combination panel and the graph. The text areas for adding an edge have been replaced with those for removing an edge, with the 'Add' button being replaced with a 'Remove' button as well.

An important feature to consider is when the user enters some edges and then deletes one resulting in a graph but with one node not connected to any others. Since there is a chance that the user would like to run the algorithm immediately after deleting the edge, the resulting unconnected node shouldn't remain on the graph display. To solve this issue, there will be a section of code to check that after removing an edge, is the source or destination node of that edge connected to any other nodes. If not, then it should be removed from the graph. As well as this, validation will be required to ensure that the edge entered by the user is a valid edge in the graph to be removed.

Leading on from this is the section of input validation. Iteration 3 focused on ensuring the algorithm can take inputs from the GUI and produce a correct output in the terminal. Since that iteration was mainly focused on functionality, input validation was not considered. Now that the entirety of Kruskal's algorithm is being

implemented in the GUI to be used by students, I need to ensure that all text areas are correctly checked for their inputs. These are outlined in the table below.

Validation Required	Reason
The user must enter source and destination nodes as a single capital letter	Since the ASCII values of characters are used throughout the Kruskal's algorithm methods, it will be vital that the user enters nodes in the correct format of a single capital letter.
The user must enter weight as a numerical value	The list of weights in the graph is used to determine the edges in the minimum spanning tree. Having an incorrect input such as a symbol would result in errors when trying to run the algorithm. These errors may not crash the program, however, when the input is converted to an integer data type in Java, it may take the ASCII value of the symbol which would result in incorrect solutions.
When removing an edge, the user must enter source and destination as single upper-case characters.	The logic for removing an edge from a graph, will check the inputted edge against edges existing in the graph. To ensure the edge is correctly identified in the list and then removed, there must be consistency within the inputs for edges.
When removing an edge, it must be checked that the inputted edge exists in the graph.	To ensure there are no errors in removing edges from the graph, the logic for removing an edge will ensure that the inputted edge exists in the graph before removing it from any components.
It must be checked that the user has entered a connected graph to perform the algorithm on.	A connected graph is a graph which contains a path from any node to any other by crossing the edges connecting nodes. Kruskal's algorithm can be performed on unconnected graphs, resulting in a Minimum Spanning

	Forest, however, this is not tested by the Edexcel A-Level Further Maths questions, which only require the algorithm to be performed on a connected graph.
Checking that a user hasn't entered the same edge twice in a graph.	Edexcel A-Level Further Maths questions only require students to perform Kruskal's algorithm on a connected graph where there is a single edge between two nodes, hence a check will be made to ensure the user enters unique edges.

The inputs will be validated through a series of try catch loops within the methods that add and remove the edges from the graph, as well as using boolean variables e.g. to check whether an inputted edge exists in the graph or not.

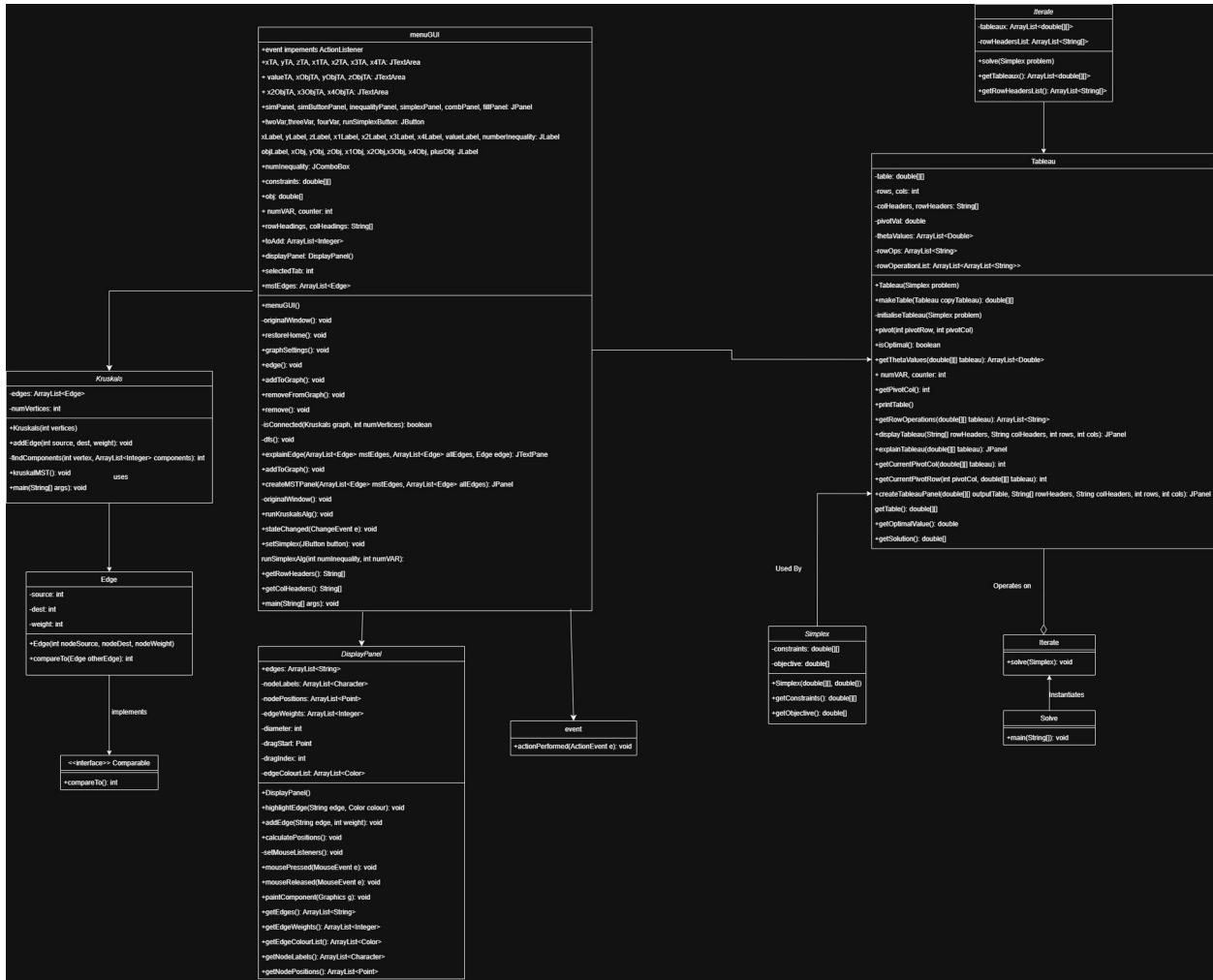
To check whether a graph is connected or not, I will implement a known method called a depth first search. This is a graph traversal algorithm that visits every node in graph. If the depth first search is complete, but some nodes have been marked as not currently visited, then we know the graph is not connected and can prompt the user to ensure they enter a connected graph.

The final component of this iteration will be the reset button, fulfilling success criterion 6.4. If the user enters the wrong graph, there needs to be a way for them to reset the graph so they can begin entering new edges, instead of manually using the remove feature to remove each of them. The Reset button will be positioned to the left of the 'Add Edge' button. Since the 'Run Algorithm' button is on the right of the row of the buttons, the 'Reset' button will be placed on the opposite side to ensure that when the user attempts to click Run Algorithm, they do not accidentally click Reset and remove the graph they have created.



Below is a UML diagram that displays the links between different classes used in this iteration. The UML diagram shows all the classes and methods developed so far.

far, illustrating the links and uses of different classes. For this iteration, the main focus is on the left half of the UML diagram, including the menuGUI class and the DisplayPanel class. These also incorporate the Kruskals class developed in Iteration 3.



As shown, the `menuGUI` class forms the connection between the `Kruskals` class and the `DisplayPanel` class. The logic behind Kruskal's algorithm takes place in the methods developed in the `Kruskals` class, but are called within the `menuGUI` class. The logic behind displaying the graph on the screen is developed in the `DisplayPanel` class, but the methods are called within the `menuGUI` class, from which it can use the results from the methods in the `Kruskals` class. The majority of

attributes in the DisplayPanel class are made private as they only need to be used by the methods within the class.

The following justification of classes, attributes, methods and variables covers the final state of the program after the development of the iteration. Any temporary variables that were used in the intermediate stages of development have been omitted from the justification as they do not appear in the final development of the iteration. Similarly, any variables or methods that were added post-testing as a remedial action have been included in the justification.

### Justification of Classes

Class Name	Justification/Purpose
DisplayPanel	This class contains all the attributes and methods to display a graph created by the user on the GUI

### Justification of Methods

Method Name	Justification/Purpose
DisplayPanel()	This is the constructor for the DisplayPanel class, it creates the list of edges, edge colours and sets the mouse listeners
addEdge(String edge, int weight)	This method adds an edge passed as a parameter to the list of edges, adds the weight to the list of weights, adds a default black colour to the list of colours and calls the calculatePositions method to create a location for it on the GUI
calculatePositions()	This method sets the position on the panel for a node to be displayed, using the maths explained earlier, to ensure all vertices are initially laid out in a circular manner.
highlightEdge(String edge, Color	This method gets the index of a specific

colour)	edge in the list of edges and then gets the item in the same index in the list of colours, highlighting the drawn edge in that colour.
setMouseListeners()	This method is used to add a mouse listener object, which will be used to allow the user to drag the nodes of the graph to their desired location
mousePressed(MouseEvent e)	This method is part of the mouse listeners, taking a mouse event as a parameter. It detects whether the user selected inside the node or not and sets the start point of the nodes motion as the point where the user clicked.
mouseReleased(MouseEvent e)	This method is also part of the mouse listeners, and resets the drag start position once the mouse is released to ensure other nodes can be moved correctly.
mouseDragged(MouseEvent e)	This gets the current location of the node, then calculates the horizontal and vertical movement of the mouse before assigning new coordinates to the point. This method also uses the panel width and height to ensure that the point remains within the boundaries of the panel.
paintComponent(Graphics g)	This is the method that draws the various components onto the panel, such as the node circle, the edge line connecting them and the weight label in the middle of the line. It creates a new 2d graphics object, before getting the positions from the list of node positions and drawing the various components in the correct location. This also gets the relevant colour from the list of edge colours and ensures

	that the edge is coloured in the correct colour.
createMSTPanel()	This method is used to create the table of edges that show whether they were added or rejected from the minimum spanning tree. It iterates through the list of edges and checks whether the edge is present in the list of minimum spanning tree edges to deduce whether it should add the 'Add' label or the 'Reject' label.
stateChanged(ChangeEvent e)	This method checks whether the tab in the JTabbedPane of edges has changed and highlights the selected edge in the correct colour based on whether it was added or rejected from the graph.
resetGraphDisplay()	This achieves success criterion 6.4 which involves allowing the user to reset the graph they have currently started, clearing all data structures storing information related to the graph.
removeFromGraph()	This method displays the relevant components on the GUI to allow the user to specify an edge in the graph they want to remove, ensuring the display panel is preserved when the text areas are changed.
remove()	This method removes the specified edge from the graph as well as any references to it throughout the lists of edges which are processed as the algorithm is run
explainEdge(ArrayList<Edge> mstEdges, ArrayList<Edge> allEdges, Edge edge)	This method takes in the current edge, edges in the minimum spanning tree and the list of all edges in the graph and returns an explanation of why the edge was added to or rejected from the graph. Different edges are given

	slightly different explanations e.g. the explanation for the final edge considered in the graph will also contain the list of edges added to the minimum spanning tree as well as the total weight.
isConnected(Kruskals graph, int numVertices)	This is a method that checks that the inputted graph is a connected graph to ensure that Kruskal's algorithm can be correctly performed on it to result in a successful output.
dfs(int node, boolean[] visited, Kruskals graph)	This is a method that performs a depth first search on the inputted graph which is used to check that the graph is connected. This method is called within the isConnected method, which returns a boolean, either true or false regarding the connectivity of the graph.
getEdgeWeights()	This is a get method inside the DisplayPanel class which returns the arraylist of edge weights so they can be used in the menuGUI class
getEdgeColourList	This is a get method in the DisplayPanel class which returns the list of colours of the various edges so they can be used in the menuGUI class
getNodeLabels	This is a get method in the DisplayPanel class which returns the list of node labels of nodes added to the graph by the user so they can be used in the menuGUI class
getNodePositions()	This is a get method in the DisplayPanel class which returns the list of positions of each node in the graph inputted by the user so they can be used in the menuGUI class.

## Justification of Class Attributes

Attribute Name	Justification/Purpose
edges	This is an arraylist of strings that stores the various edges in the graph e.g. 'AB', 'BC'. This arraylist is used multiple times e.g. when creating the panel to store the table of added and rejected arcs.
nodeLabels	This is an arraylist of characters storing the label of an individual node, for example if there is an edge 'AB' then there will be individual nodes 'A' and 'B'. This is used to ensure that the same node isn't added twice, as well as being used when drawing the node labels onto the visual graph.
nodePositions	This is an arraylist of Point objects which will store the position on the panel where each node needs to be drawn.
edgeWeights	This is an arraylist of integers that stores the weights of the edges added to the graph, so they can be drawn onto the visual graph.
diameter	This stores the fixed value of the diameter of one the node circles drawn onto the GUI.
dragStart	This is a Point object that stores the location of a node at the instant that the user starts moving it in the GUI.
dragIndex	This is an integer that stores the index of the node being dragged by the user in the GUI
edgeColourList	This is an arraylist of Color objects that stores the necessary colours of the various edges so they can be highlighted in either green for 'added' to the minimum spanning tree or red if

	they are 'rejected' from the minimum spanning tree
combPanel	This is a JPanel component that is split into two halves to store the displayed graph on the left and all the explanation and table on the right.
fillPanel	This is a JPanel component added to the right half of combPanel, storing the table of added and rejected arcs, as well as the explanation for each edge.
displayPanel	This is an instance of the DisplayPanel class which is used to create the visual graph on the screen. It is created in the menuGUI class so it can be added to the left half of combPanel in the existing GUI.
selectedTab	This is an integer that stores the index of the tab selected in the JTabbedPane and is used to determine which edge of the visual graph needs to be highlighted in its respective colour.
mstEdges	This is an arraylist of Edge objects which were created from the Edge class in Iteration 3. This stores the edges that are in the final minimum spanning tree and is used in the table of the added and rejected edges when checking if an edge should have 'Add' or 'Reject', as well as in the highlighting method to decide if it should be green or red.
kruskalsError	This is a JLabel that contains the error message to be displayed when the user enters an incorrect input e.g. a letter in the 'Weight' text area.
removeEdgeButton	This is the button displayed on the GUI next to the source and destination text areas after the user selects 'Remove Edge'. Clicking this button will remove

	the inputted edge and any data structures that store this edge and related information.
resetGraph	This is a JButton that is displayed in the row of buttons on the left of the 'Add Edge' button. Clicking this button will clear the displayed graph and any data structures the store edges and related information.

### Justification of Other Variables

Variable Name	Justification/Purpose
index	This is an integer that stores the index of a specific edge in the list of edges
startNode	This is a character that stores the first character in the edge as the start node e.g. the start node of the edge 'AB' would be 'A'
destNode	Similar to startNode, this is a character that stores the second character in the edge as the destination node e.g. the destination node of the edge 'AB' would be 'B'
panelWidth	This is an integer that stores the width of the JPanel and is used to ensure that the nodes and edges are displayed within the boundaries of the panel
panelHeight	This is an integer storing the height of the JPanel, and similarly to panelWidth it ensures the nodes and edges are displayed within the panel boundaries
centerX, centerY	These are integers that store the coordinates of the center of the JPanel
radius	This is an integer that stores the minimum value out of the panel width and height, divided by three, ensuring

	the nodes and vertices remain displayed within the center region of the panel initially.
angle	This is a double value that stores the angle of a node to be added to the graph. As explained earlier, taking the cosine and sine of this angle gives us a coordinate so the various nodes can be displayed by default in a circular format.
x, y	These are integers that store the x and y coordinates of a specific node
clickPoint	This is a Point object that stores the point that a user clicks on the GUI, which can then be checked to see if it is inside the node or not to decide whether the node should move with the user's mouse movement
distance	This is a double value that measures the distance from the click location to the center of the node.
currentPoint	This is a Point object and gets the current mouse position
nodePosition	This is a Point object and gets the position of the node that is being dragged using the dragIndex
horizontalMovement, verticalMovement	These are two integer values that store the horizontal and vertical movement of the mouse after clicking on the node
newX, newY	These are two integer values that store the new x and y coordinates that the node should be at based on the horizontal and vertical movement of the mouse
minX, minY, maxX, maxY	These store the minimum and maximum locations of the panel, the minimum being (0,0) and the maximum being (panel width, panel height). This is used to check that the

	node is dragged only within the panel
g2d	This is a Graphics2D object that can be assigned colours and has methods like drawLine and fillOval, which are used to actually draw the node circles and edges on the JPanel
edge	This is a string and stores a specific edge from the list of edges e.g. 'AB'
sourceNode, destinationNode	Similar to startNode and destNode, these are characters that store the parts of the edge in terms of start and destination node but are used in the paintComponent method so are given different names
sourceIndex, destIndex	These are integers that get the index of the source and destination nodes in the list of nodeLabels
p1, p2	These are Point objects and store the position of the source node and destination node
x1, y1, x2, y2	These are the x and y coordinates of the source node and destination node, used to determine where the nodes need to be drawn on the JPanel
position	This is a Point object and stores the position of each node in the list as part of a for loop when drawing the node labels onto the JPanel
label	This is a character that stores the label of a node at the index in a for-loop so that the node label can be drawn onto the GUI.
pair	This is a string that takes the stored source and destination labels entered into the text areas by the user and combines them into one string e.g. if they entered 'A' into the source text area and 'B' into the destination text area, then it would combine them to

	make pair = 'AB'
mstPanel	This is a JPanel component with a grid layout containing 2 columns. The left column contains the list of edges in the graph and the right column contains the text either 'Add' or 'Reject' based on whether the edge was added to the minimum spanning tree or not. This is added to the right half of combPanel as part of the working out to explain the algorithm to the user.
edgeLabel	This is a JLabel that stores the text 'Edge' as the column header for the left column of the mstPanel
actionLabel	This is a JLabel that stores the text 'Add/Reject' as the column header for the right column of the mstPanel
edgeDisplay	This is a JLabel that takes the value of each edge in the graph and is added to the mstPanel in the left column
actionDisplay	This is a JLabel that takes the value 'Add' or 'Reject' based on whether a given edge was added to the minimum spanning tree or not and is then added to the right column of the mstPanel
edgeList	This is an arraylist of Edge objects that stores each edge in the graph entered by the user into the text areas, it uses the Edge objects created in Iteration 3
explainPane	This is a JTabbedPane component that stores each edge in the graph in weight order as a new tab for the user to click through. Clicking through them highlights the edge in green or red based on whether it was added to the minimum spanning tree or not, as well as displaying the explanation for why the edge was added/rejected
kruskalsPromptPanel	This is a JPanel component added as

	the first tab of the JTabbedPane and is used to explain to the user how the tabs work and what the colour highlighting means.
kruskalsPrompt	This is a JLabel added to the kruskalsPromptPanel used to explain how the tabs of the JTabbedPane work
colourLabel	This is a JLabel added to the kruskalsPromptPanel used to explain the colouring used for the highlighting indication that green means the edge was added and red means the edge was rejected
explanation	This is a JTextPane that stores the explanation text for why an edge was added or rejected from the minimum spanning tree and is added to the tabbed pane for each edge.
title	This is a string that stores the title of each tab in the tabbed pane. The title of each tab is the edge that it explains e.g. 'AB'
sourcePane	This is a JTabbedPane object that is the current selected tab in the tabbed pane. It is part of the stateChanged method used to see which tab was selected by the user and deciding which colour to highlight the selected edge in, either green or red.
selectedTab	This is the index of the currently selected tab in the tabbed pane and is used to check how many edges have been previously added, based on how many tabs precede this one, and it highlights each of them in the correct colour.
removeSourceTA	This is a JTextArea that allows the user to enter in the source node of an edge to be removed

removeDestTA	This is a JTextArea that allows the user to enter in the destination node of an edge to be removed
removeSourceLabel	This is a JLabel indicating the purpose of the removeSourceTA to the user
removeDestLabel	This is a JLabel that indicates the purpose of the removeDestTA to the user.
index	This is an integer variable that is the index of an edge in the list of edges so it can be removed from the graph and all the arraylists that store information about it.
isValidSourceNode, isValidDestNode	These are two boolean variables with similar purposes. When the user enters a source and destination node of an edge to be removed, these take the value of true or false depending on whether the entered nodes are actually in the graph or not, as a form of input validation.
nodeIndex	This is an integer that is the index of a specific node in the list of node labels, and if this is given a value e.g. a valid node in the graph, then the edge connecting the source and destination node can be removed from the graph.
visited	This is a list of boolean values and is part of the isConnected method. The graph is checked to see if all nodes are connected in some way to one another. If a node is connected, then the index of that node in the visited list is true, otherwise it is false. This list is then iterated over and if any values in the list are false then the graph is not connected.
startNode	This is an integer value in the isConnected method and takes the

	largest possible integer value in Java. This value is then replaced by whatever the first edge added to the graph was using the Math.min() function.
explanationPane	This is a JTextPane component that stores the explanation for a specific edge in the graph. This text pane is then returned by the explainEdge method to be added to the fillPanel at the bottom as part of the JTabbedPane.
total	This is an integer value that sums the weights of the edges in the minimum spanning tree so that it can be added to the final explanation pane to indicate the total weight of the minimum spanning tree.
reversePair	This is a string that stores the reversed version of an edge e.g. if the original is 'AB' then the reverse is 'BA'. This is used to check whether the user has entered the same edge twice in a graph and throws an exception if it sees the same edge or the same edge reversed is inputted.

Below is the pseudocode for the main development of this iteration, which will cover the basis of how the methods in this iteration will be developed. This pseudocode covers how the final state of the iteration should look after any intermediate methods and variables have been removed as they do not serve purpose in the final development. The pseudocode focuses on the methods that involve a greater element of logic, hence some methods e.g. those that focus on adding components to the GUI won't be covered in the pseudocode:

Class DisplayPanel:

Private ArrayList<String> edges // stores the list of edges in the graph, each as a string

Private ArrayList<Character> nodeLabels // stores the labels of the nodes (vertices) in the graph

Private ArrayList<Point> nodePositions // stores the positions of nodes on the display panel

Private ArrayList<Integer> edgeWeights // stores the weights of the edges in the graph

Private Integer diameter // diameter of the nodes used for rendering

Private Point dragStart // starting point for dragging a node

Private Integer dragIndex // index of the node being dragged

Private String highlightedEdge // the edge currently highlighted (if any)

Private Color edgeColour // default color for edges

Private ArrayList<Color> edgeColourList // stores colors for each edge, allowing updates

Private ArrayList<ArrayList<Color>> colourStates // stores multiple states of edge colors for animations

Constructor DisplayPanel():

edges -> new ArrayList<String>() // initializes the list of edges

edgeColourList -> new ArrayList<Color>() // initializes the list of edge colors

setMouseListeners() // sets up mouse listeners for interaction

Method highlightEdge(edge, colour):

Integer index -> edges.indexOf(edge) // finds the index of the given edge

```
if index != -1:  
    edgeColourList.set(index, colour) // updates the color of the edge  
    revalidate() // updates the layout  
    repaint() // redraws the panel to reflect changes
```

Method addEdge(edge, weight):

```
edges.add(edge) // adds the new edge to the list  
edgeWeights.add(weight) // adds the weight of the new edge  
edgeColourList.add(Color.BLACK) // sets the default color for the new edge  
calculatePositions() // recalculates the positions of the nodes  
revalidate() // updates the layout  
repaint() // redraws the panel
```

Method calculatePositions():

```
for each String s in edges: // loops through each edge  
    Character startNode -> s.charAt(0) // gets the source node  
    Character destNode -> s.charAt(1) // gets the destination node  
  
    if not nodeLabels.contains(startNode): // if the source node is new:  
        nodeLabels.add(startNode) // add it to the labels list  
        nodePositions.add(new Point(0, 0)) // assign it a default position  
    if not nodeLabels.contains(destNode): // if the destination node is new:  
        nodeLabels.add(destNode) // add it to the labels list  
        nodePositions.add(new Point(0, 0)) // assign it a default position
```

```

Integer panelWidth -> getWidth() // get panel width

Integer panelHeight -> getHeight() // get panel height

Integer centerX -> panelWidth / 2 // calculate the horizontal center of the
panel

Integer centerY -> panelHeight / 2 // calculate the vertical center of the panel

Integer radius -> min(panelWidth, panelHeight) / 3 // calculate the radius for
placing nodes in a circle

for Integer i from 0 to nodeLabels.size() - 1: // loop through all nodes

    Double angle -> 2 * Math.PI * i / nodeLabels.size() // calculate the angle for
    placing the node

    Integer x -> centerX + (Integer)(radius * cos(angle)) - diameter / 2 // 
    calculate the x-coordinate

    Integer y -> centerY + (Integer)(radius * sin(angle)) - diameter / 2 // calculate
    the y-coordinate

    nodePositions.set(i, new Point(x, y)) // set the position of the node

```

Method setMouseListeners():

```

add MouseListener: // sets up listener for mouse clicks and releases

on mousePressed(e): // detects when the mouse is pressed

    Point clickPoint -> e.getPoint() // get the position of the click

    for Integer i from 0 to nodePositions.size() - 1: // loop through all nodes

        Point nodePosition -> nodePositions.get(i) // get the position of the
        current node

```

```
    Integer centerX -> nodePosition.x + diameter / 2 // calculate the center  
x of the node
```

```
    Integer centerY -> nodePosition.y + diameter / 2 // calculate the center  
y of the node
```

```
    Double distance -> clickPoint.distance(centerX, centerY) // calculate the  
distance from the click to the node center
```

```
    if distance <= diameter / 2: // if the click is within the node's radius:
```

```
        dragStart -> clickPoint // record the start of the drag
```

```
        dragIndex -> i // record the index of the node being dragged
```

```
        break // exit the loop
```

```
on mouseReleased(e): // detects when the mouse is released
```

```
    dragStart -> null // clear the drag start point
```

```
    dragIndex -> -1 // reset the drag index
```

```
add MouseMotionListener: // sets up listener for mouse movement
```

```
on mouseDragged(e): // detects when the mouse is dragged
```

```
    if dragIndex != -1 and dragStart != null: // if dragging a valid node:
```

```
        Point currentPoint -> e.getPoint() // get the current mouse position
```

```
        Point nodePosition -> nodePositions.get(dragIndex) // get the position  
of the dragged node
```

```
        Integer horizontalMovement -> currentPoint.x - dragStart.x // calculate  
horizontal movement
```

```
        Integer verticalMovement -> currentPoint.y - dragStart.y // calculate  
vertical movement
```

```

        Integer newX -> max(0, min(nodePosition.x + horizontalMovement,
getWidth() - diameter)) // bound the x-coordinate

        Integer newY -> max(0, min(nodePosition.y + verticalMovement,
getHeight() - diameter)) // bound the y-coordinate

        nodePosition.setLocation(newX, newY) // update the node's position

        dragStart -> currentPoint // update the drag start point

        repaint() // redraw the panel
    }
}

```

Method paintComponent(g):

```

super.paintComponent(g) // call the parent class's paint method

Graphics2D g2d -> (Graphics2D) g // cast to Graphics2D for better drawing
features

g2d.setColor(Color.BLACK) // set the default color

for Integer i from 0 to edges.size() - 1: // loop through all edges

    String edge -> edges.get(i) // get the edge

    Character sourceNode -> edge.charAt(0) // get the source node

    Character destinationNode -> edge.charAt(1) // get the destination node

    Integer sourceIndex -> nodeLabels.indexOf(sourceNode) // get the index of
the source node

    Integer destIndex -> nodeLabels.indexOf(destinationNode) // get the index
of the destination node

    if sourceIndex != -1 and destIndex != -1: // ensure both nodes are valid:

        Point p1 -> nodePositions.get(sourceIndex) // get the position of the
source node
}
}

```

```
    Point p2 -> nodePositions.get(destIndex) // get the position of the destination node

        Integer x1 -> p1.x + diameter / 2 // calculate the x-coordinate for the source

        Integer y1 -> p1.y + diameter / 2 // calculate the y-coordinate for the source

        Integer x2 -> p2.x + diameter / 2 // calculate the x-coordinate for the destination

        Integer y2 -> p2.y + diameter / 2 // calculate the y-coordinate for the destination

        g2d.setColor(edgeColourList.get(i)) // set the color of the edge
        g2d.setStroke(new BasicStroke(3.0)) // set the thickness of the edge
        g2d.drawLine(x1, y1, x2, y2) // draw the edge

        Integer weight -> edgeWeights.get(i) // get the weight of the edge

        Integer weightX -> (x1 + x2) / 2 // calculate the x-coordinate for the weight label

        Integer weightY -> (y1 + y2) / 2 // calculate the y-coordinate for the weight label

        g2d.setFont(new Font("Arial", Font.BOLD, 17)) // set the font for the weight label

        g2d.drawString(String.valueOf(weight), weightX, weightY - 7) // draw the weight

    for Integer i from 0 to nodeLabels.size() - 1: // loop through all nodes

        Point position -> nodePositions.get(i) // get the position of the node
```

```
Character label -> nodeLabels.get(i) // get the label of the node
```

```
g2d.setColor(Color.BLACK) // set the fill color for the node
```

```
g2d.fillOval(position.x, position.y, diameter, diameter) // draw the node
```

```
g2d.setColor(Color.WHITE) // set the color for the label
```

```
g2d.drawString(String.valueOf(label), position.x + diameter / 3, position.y + 2  
* diameter / 3) // draw the label
```

Method getEdges():

```
return edges // returns the list of edges
```

Method remove():

```
try:
```

```
String source -> sourceTA.getText().trim() // extract the source node
```

```
if not source.matches("[A-Z]"):
```

```
throw new IllegalArgumentException("Please enter a source node as a single  
capital letter.")
```

```
String dest -> destTA.getText().trim() // extract the destination node
```

```
if not dest.matches("[A-Z]"):
```

```
throw new IllegalArgumentException("Please enter a destination node as a  
single capital letter.")
```

```
String pair -> source + dest // construct the edge string
```

```
if not displayPanel.edges.contains(pair): // check if the edge exists  
    throw new IllegalArgumentException("The specified edge does not exist in the  
graph.")
```

```
Integer index -> displayPanel.edges.indexOf(pair) // get the edge index  
displayPanel.edges.remove(index) // remove the edge  
displayPanel.edgeWeights.remove(index) // remove the corresponding weight  
displayPanel.edgeColourList.remove(index) // remove the color
```

```
// update the graph visually  
displayPanel.revalidate()  
displayPanel.repaint()
```

```
// remove the edge from the `toAdd` list  
Character sourceChar -> source.charAt(0)  
Character destChar -> dest.charAt(0)
```

```
for Integer i from 0 to toAdd.size() - 1 step 3: // loop through `toAdd` list  
    if (toAdd.get(i) == (int) sourceChar and toAdd.get(i + 1) == (int) destChar) or  
        (toAdd.get(i) == (int) destChar and toAdd.get(i + 1) == (int) sourceChar):  
        toAdd.remove(i + 2) // remove weight  
        toAdd.remove(i + 1) // remove destination  
        toAdd.remove(i) // remove source
```

```
break
```

```
// check if the source node is still valid
```

```
Boolean isValidSourceNode -> false
```

```
for Integer i from 0 to toAdd.size() - 1:
```

```
    if toAdd.get(i) == sourceChar:
```

```
        isValidSourceNode -> true
```

```
if not isValidSourceNode:
```

```
    Integer nodeIndex -> displayPanel.getNodeLabels().indexOf(sourceChar) //  
    get index of source node
```

```
if nodeIndex != -1:
```

```
    displayPanel.getNodeLabels().remove(nodeIndex) // remove label
```

```
    displayPanel.getNodePositions().remove(nodeIndex) // remove position
```

```
    displayPanel.revalidate() // refresh the GUI
```

```
    displayPanel.repaint()
```

```
// check if the destination node is still valid
```

```
Boolean isValidDestNode -> false
```

```
for Integer i from 0 to toAdd.size() - 1:
```

```
    if toAdd.get(i) == destChar:
```

```
        isValidDestNode -> true
```

```

if not isValidDestNode:

    Integer nodeIndex -> displayPanel.getNodeLabels().indexOf(destChar) // get
index of destination node

    if nodeIndex != -1:

        displayPanel.getNodeLabels().remove(nodeIndex) // remove label
        displayPanel.getNodePositions().remove(nodeIndex) // remove position

        displayPanel.revalidate() // refresh the GUI
        displayPanel.repaint()

    kruskalsError.setText("Edge removed successfully.") // display success message

catch IllegalArgumentException e:

    kruskalsError.setText(e.getMessage()) // display the error message

Method isConnected(graph, numVertices):

Boolean[] visited -> new Boolean[numVertices] // track visited nodes
Integer startNode -> Integer.MAX_VALUE // initialize the starting node

for Edge edge in graph.edges: // find the smallest node to start the dfs
    startNode -> min(startNode, min(edge.source, edge.dest))

```

```
dfs(startNode, visited, graph) // perform a depth-first search
```

```
for Edge edge in graph.edges:
```

```
    if not visited[edge.source] or not visited[edge.dest]:
```

```
        return false // graph is not connected
```

```
return true // graph is connected
```

Method dfs(node, visited, graph):

```
visited[node] -> true // mark the node as visited
```

```
for Edge edge in graph.edges:
```

```
    if edge.source == node and not visited[edge.dest]: // source visited, destination  
    not
```

```
        dfs(edge.dest, visited, graph) // recursively visit destination
```

```
    else if edge.dest == node and not visited[edge.source]: // destination visited,  
    source not
```

```
        dfs(edge.source, visited, graph) // recursively visit source
```

Method stateChanged(event):

```
JTabbedPane sourcePane -> (JTabbedPane) event.getSource() // get the updated  
tabbed pane
```

```
selectedTab -> sourcePane.getSelectedIndex() - 1 // get the selected tab index
```

```
for String edge in displayPanel.getEdges():
```

```

displayPanel.highlightEdge(edge, Color.BLACK) // reset all edges to black

for Integer i from 0 to selectedTab: // iterate through tabs

    Edge edge -> graph.edges.get(i) // get the edge

    if mstEdges.contains(edge):

        String edgeString -> (char)(edge.source) + (char)(edge.dest)

        displayPanel.highlightEdge(edgeString, Color.RED) // highlight edges in the
MST

```

One thing to note is that the methods mousePressed, mouseReleased and mouseDragged are all developed inside the method setMouseListeners, as they are basic methods for event handling and don't require their own separate class.

The final element of design to consider is the test plan. To ensure that the developed methods in this iteration correctly function, they will need to be thoroughly tested to ensure they can handle a variety of user inputs. The test table below highlights how I will go about testing each feature developed in this iteration.

Test Number/Description	Test Type	Expected Outcome
1) Pressing Add Edge then inputting a source, destination and weight	Normal	An edge should be displayed on the left half of the screen, with two nodes and a line connecting them. The nodes should have labels and the line should have the weight in the center

2) Adding two edges with a common node	Normal	Both edges should be outputted in the correct locations (forming a triangle shape with the angles since there are 3 nodes) and they should share the common node. Similar to test 1, all the labels should be correctly displayed
3a) Inputting numbers into the source text area  3b) Inputting numbers into the destination text area  3c) Inputting multiple capital letters into the source and destination text areas  3d) Not inputting anything into the source and destination text areas	Erroneous	<p>3a) The error message label should be displayed next to the buttons saying "Please enter source node as a single capital letter."</p> <p>3b) The error message label should be displayed on the window next to the buttons and read, "Please enter destination node as a single capital letter."</p> <p>3c) The error message label should read the same message "Please enter source node as a single capital letter." depending on which text area the invalid input was entered into.</p> <p>3d) The error message label should be displayed and read "Please enter source node as a single capital letter." as this is the first exception thrown in the try-catch loop. The user will then be told their error one-by-one e.g. if they enter a value in the source but not the destination, it will tell them to enter a value in the destination.</p>
4a) Inputting a letter in the	Erroneous	4a) The error message label should be displayed in red text to indicate that the weight must be entered as a valid number

weight text area  4b) Inputting nothing in the weight text area		4b) The error message label should be displayed in red text next to the buttons to indicate that the weight must be inputted as a valid number
5a) Adding the same edge twice to the graph  5b) Adding the same edge to the graph but entering the source and destination the other way around	Erroneous	5a) The error message label should be displayed to indicate that the edge trying to be inputted already exists in the graph.  5b) The error message label should be displayed in red text next to the row of buttons indicating that the edge is already in the graph.
6) Clicking the 'Remove Edge' button in the GUI	Normal	Most of the components of the GUI should remain the same, the only different should be the 'Source', 'Destination' and 'Weight' text areas as well as the 'Add' button from the 'Add Edge' option should be replaced with just a 'Source' text area, 'Destination' text area and a 'Remove' button
7) Removing an edge that was previously added to the graph	Normal	When inputting a source and destination node of an edge in the graph and selecting remove, the edge should be removed from the graph display and the data structures that store information about the edge
8) Running the	Normal	When running the algorithm after removing an edge, the removed edge should not be considered in the

algorithm after removing an edge from the graph. To test, I will enter 4 edges, remove 1 and then run the algorithm		table of added and rejected values, or be added as a tab in the tabbed pane as this would potentially result in the incorrect output from running the algorithm on the desired graph (with the edge removed).
9) Entering edges into the graph then pressing the 'Reset' button	Normal	Upon pressing the 'Reset' button, the graph currently being displayed should be cleared, returning the user interface to the same state as when the user first pressed the 'Add Edge' button.
10) Moving the nodes of the graph around to change the shape of the graph.	Normal	When moving the nodes, they should remain within the black border square on the left half of the panel, the edges should follow the nodes and remain connected, the weight label should remain in the center of the edge and the node label should remain within the node circle.
11) Entering a disconnected graph to perform Kruskal's algorithm on	Erroneous	When attempting to run the algorithm after creating a disconnected graph, the error message label should appear on the GUI next to the row of buttons and indicate the inputted graph must be connected
12) Testing the highlighting	Normal	After entering a graph and running the algorithm, clicking through the edges of the graph should highlight them in either green (Add) or red (Reject),

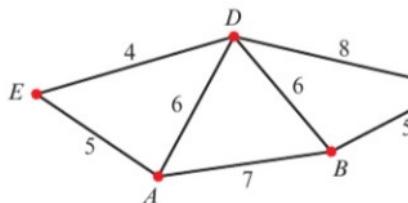
g of edges when tabs from the tabbed pane are selected		building up the minimum spanning tree.
13) Chapter 3 Example 1 from Pearson ActiveLearn Decision 1 Textbook Refer to Figure 1a below	Normal	The arcs added should be DE, AE, BC and BD and the total weight of the minimum spanning tree should be 20.
14) Chapter 3, Question 1a from Pearson ActiveLearn Decision 1 Textbook Refer to Figure 2a below	Normal	The edges added to the minimum spanning tree are EF, BD, CD, AH, DF, AC, GH and the other edges should be rejected. The minimum spanning tree should have a total weight of 98.
15) Chapter 3, Question 1b from Pearson ActiveLearn Decision 1 Textbook Refer to Figure 3a below	Normal	The edges added to the minimum spanning tree should be BF, FG, AB, CE, BC, CD and the rest should be rejected. The total weight of the minimum spanning tree should be displayed as 27.
16) Chapter 3 Question 2b from	Normal	The edges added to the minimum spanning tree should be YZ, VW, XY, UW, UX, SU, TU with a total weight of 121

Pearson ActiveLearn Decision 1 Textbook Refer to Figure 4a below		
17) Pearson Edexcel Decision 1 A-Level Exam Paper January 2005 Question 3ai Refer to Figure 5a below	Normal	The edges included in the minimum spanning tree should be displayed as FH, AD, DE, CD, BC, CF, HI, IJ, with a total weight of 188 and the other edges should be rejected
18) Pearson Edexcel Decision 1 A-Level Exam Paper May 2010 Question 2a Refer to Figure 6a below	Normal	The edges included in the minimum spanning tree for this graph are DE, GF, DC, BD, EG, AC, GH with a total weight of 174

Figure 1a)

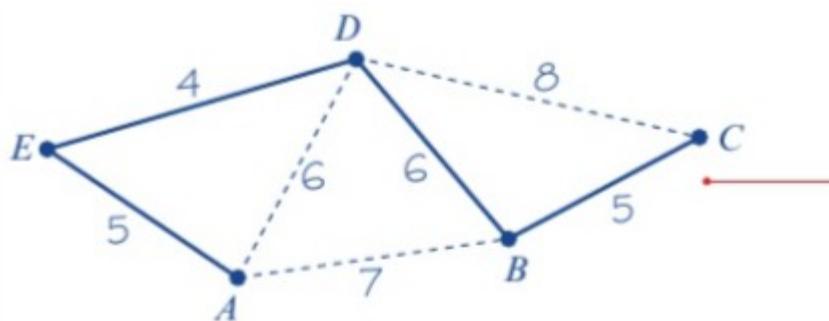
**Example 1**

Use Kruskal's algorithm to find a minimum spanning tree for this network. List the arcs in the order that you consider them. State the weight of your tree.



In a network of  $n$  vertices a spanning tree will always have  $(n - 1)$  arcs. In this case there will be 4 arcs in the spanning tree.

Figure 1b)



All vertices are connected so this is a minimum spanning tree.

Its weight is  $5 + 4 + 6 + 5 = 20$

Figure 2a)

- 1 Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

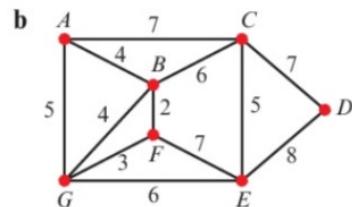


Figure 2b)

- 1 a**
- EF (11) add to tree
  - BD (12) add to tree
  - CD (12) add to tree
  - AH (14) add to tree
  - DF (15) add to tree
  - AC (16) add to tree
  - BC (17) reject
  - GH (18) add to tree
  - BE (18)
  - CH (20)
  - CG (21)
  - FG (24)
  - AB (25)
- } reject all remaining arcs.

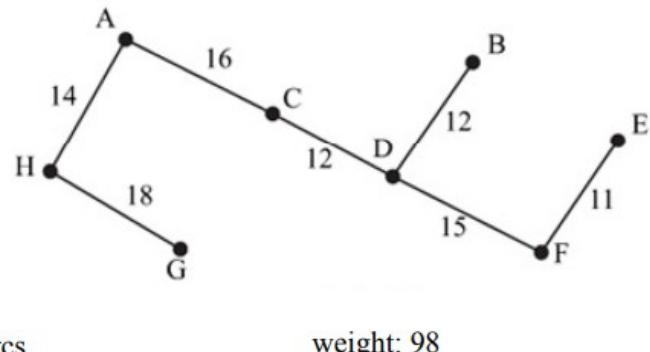


Figure 3a)

- 1** Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

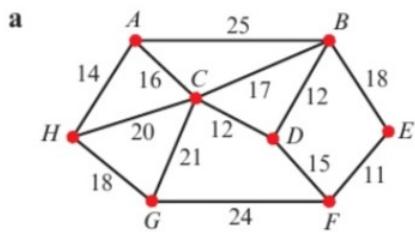


Figure 3b)

- b**
- BF (2) add to tree
  - FG (3) add to tree
  - AB (4) add to tree
  - BG (4) reject
  - AG (5) reject
  - CE (5) add to tree
  - BC (6) add to tree
  - EG (6) reject
  - AC (7) reject
  - CD (7) add to tree
  - EF (7)
  - DE (8)
- } reject all remaining arcs.

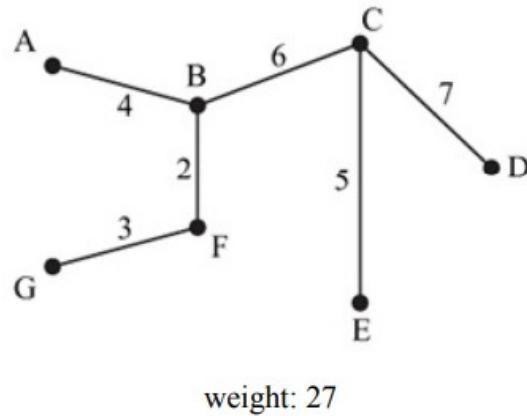


Figure 4a)

- E/P** 2 a State what is meant by:  
 i a tree  
 ii a minimum spanning tree.
- b Use Kruskal's algorithm to find a minimum spanning tree for this network.

(1 mark)

(1 mark)

(3 marks)

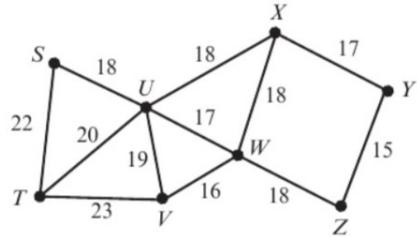


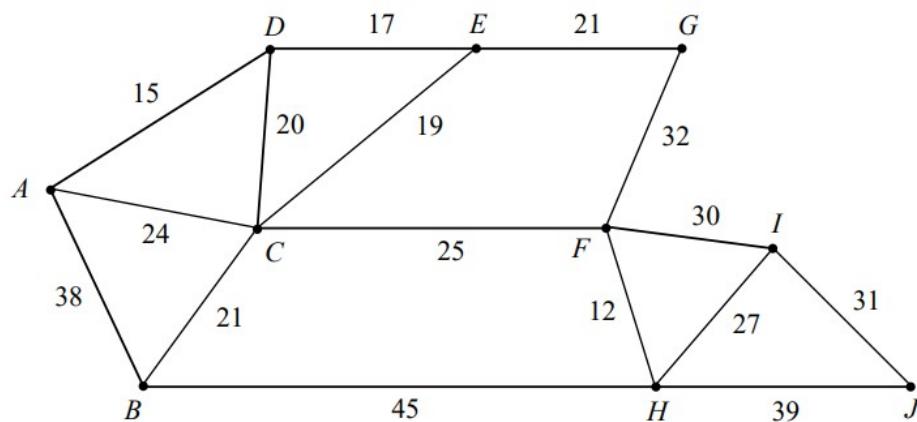
Figure 4b)

- 2 a i A tree is a connected graph with no cycles.  
 ii A minimum spanning tree is a tree of minimum total weight that connects all of the nodes.
- b By inspection the order of the arcs is  
YZ(15), VW(16), XY(17), UW(17), UX(18), WX(18), SU(18), WZ(18), UV(19), TU(20), ST(22), TV(23)  
 Underlined arcs are in the minimum spanning tree. Total weight = 121

Figure 5a)

3.

Figure 2



The network in Figure 2 shows the distances, in metres, between 10 wildlife observation points. The observation points are to be linked by footpaths, to form a network along the arcs indicated, using the least possible total length.

(a) Find a minimum spanning tree for the network in Figure 2, showing clearly the order in which you selected the arcs for your tree, using

(i) Kruskal's algorithm,

(3)

Figure 5b)

3) (a)	(i) FH, AD, DE, CE, (not DC), {BC}, (not EG), CF, HI, (not FI), IJ <sub>stop</sub>	MIAIAI (3)
--------	--	---------------

Summing these edge weights using the provided graph in the question results in a total weight of 188

Figure 6a)

2.

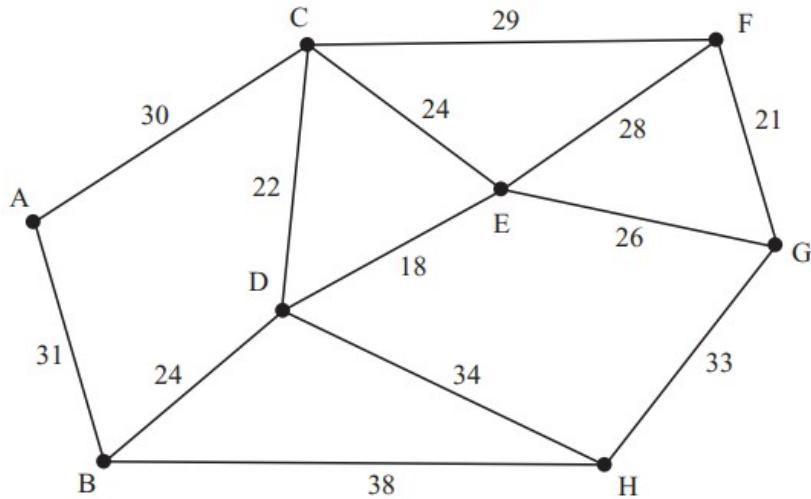


Figure 1

Figure 1 represents the distances, in metres, between eight vertices, A, B, C, D, E, F, G and H, in a network.

- (a) Use Kruskal's algorithm to find a minimum spanning tree for the network.

You should list the arcs in the order in which you consider them. In each case, state whether you are adding the arc to your minimum spanning tree.

(3)

Figure 6b)

Question Number	Scheme	Marks																																																																																	
Q2 (a)	DE GF DC $\left\{ \begin{matrix} \text{not CE} \\ \text{BD} \end{matrix} \right\}$ EG (not EF not CF) AC (not AB) GH	M1 A1 A1  3																																																																																	
(b)	<table border="1"> <tr><td>-</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td><td>H</td></tr> <tr><td>A</td><td>-</td><td>31</td><td>30</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> <tr><td>B</td><td>31</td><td>-</td><td>-</td><td>24</td><td>-</td><td>-</td><td>-</td><td>38</td></tr> <tr><td>C</td><td>30</td><td>-</td><td>-</td><td>22</td><td>24</td><td>29</td><td>-</td><td>-</td></tr> <tr><td>D</td><td>-</td><td>24</td><td>22</td><td>-</td><td>18</td><td>-</td><td>-</td><td>34</td></tr> <tr><td>E</td><td>-</td><td>-</td><td>24</td><td>18</td><td>-</td><td>28</td><td>26</td><td>-</td></tr> <tr><td>F</td><td>-</td><td>-</td><td>29</td><td>-</td><td>28</td><td>-</td><td>21</td><td>-</td></tr> <tr><td>G</td><td>-</td><td>-</td><td>-</td><td>-</td><td>26</td><td>21</td><td>-</td><td>33</td></tr> <tr><td>H</td><td>-</td><td>38</td><td>-</td><td>34</td><td>-</td><td>-</td><td>33</td><td>-</td></tr> </table>	-	A	B	C	D	E	F	G	H	A	-	31	30	-	-	-	-	-	B	31	-	-	24	-	-	-	38	C	30	-	-	22	24	29	-	-	D	-	24	22	-	18	-	-	34	E	-	-	24	18	-	28	26	-	F	-	-	29	-	28	-	21	-	G	-	-	-	-	26	21	-	33	H	-	38	-	34	-	-	33	-	B2, 1, 0  2
-	A	B	C	D	E	F	G	H																																																																											
A	-	31	30	-	-	-	-	-																																																																											
B	31	-	-	24	-	-	-	38																																																																											
C	30	-	-	22	24	29	-	-																																																																											
D	-	24	22	-	18	-	-	34																																																																											
E	-	-	24	18	-	28	26	-																																																																											
F	-	-	29	-	28	-	21	-																																																																											
G	-	-	-	-	26	21	-	33																																																																											
H	-	38	-	34	-	-	33	-																																																																											
(c)	AC CD DE BD GE GF GH	M1 A1 A1  3																																																																																	
(d)	Weight: 174	B1  1																																																																																	

# Development

## Iteration 1: GUI Mockup

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

The first part of this iteration involved creating a blank window and adding a basic menu bar which can be altered.

```
public class menuGUI extends JFrame{  
    // the following public attributes are declared with meaningful names to be used throughout the constructor and other methods  
    6 usages  
    JMenuBar menubar;
```

```

public menuGUI(){
    setLayout(new FlowLayout()); //sets the window layout as a flow layout

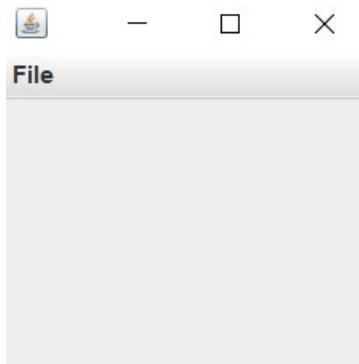
    //creating the menu bar
    menubar = new JMenuBar();
    setJMenuBar(menubar);

    //adding 'file' menu option to the menu bar |
    file = new JMenuItem("File");
    menubar.add(file);

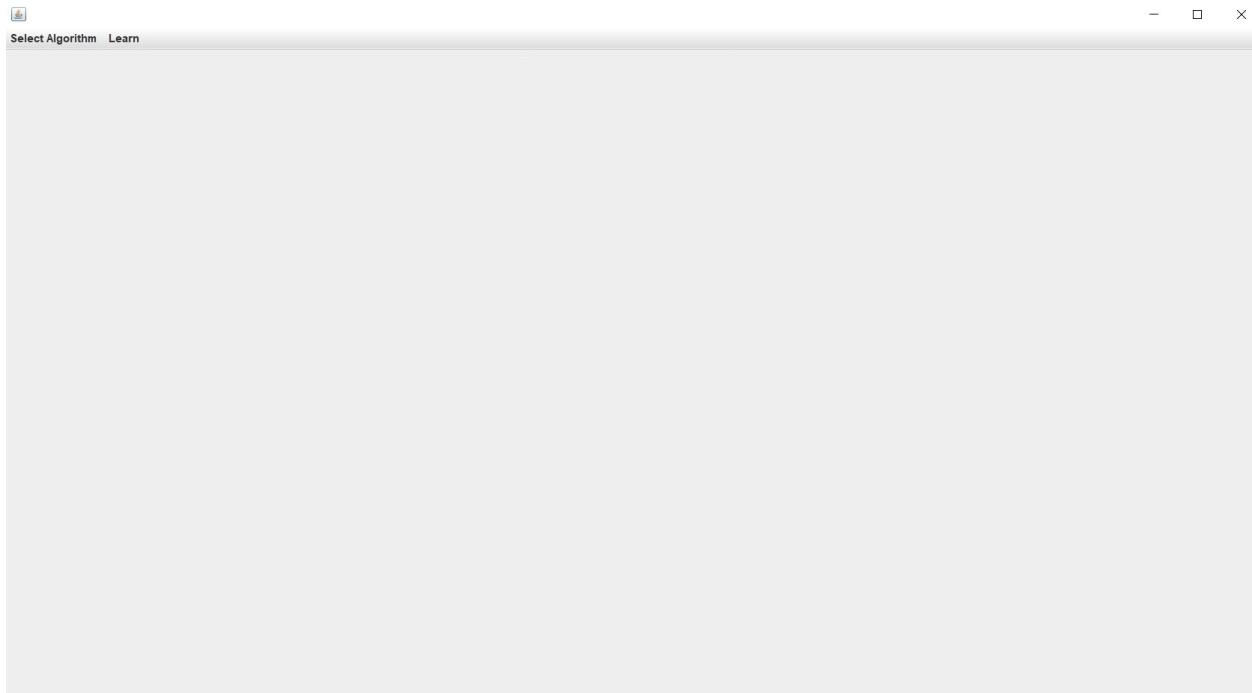
public static void main(String[] args){
    menuGUI gui = new menuGUI(); //creating an instance of the menuGUI class and setting the qualities of the window
    gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    gui.setSize(200, 200);
    gui.setVisible(true);
}

```

Result of running:



Having successfully created an initial window, I can now focus on creating the options of the menu bar and any menu items to be added within these options. As well as this, adjustments can be made to the window, resizing it to be slightly larger upon generation



Consulting my stakeholder at this point, it was recommended that the name of the menu bar options be changed to have one general 'File' section and an option for each algorithm, hence the necessary changes were made. I also added a JLabel prompting the user to select an option from the menu bar. Presenting my stakeholder with the JLabel at the top of the window, compared to the middle, it was expressed that the JLabel would be more effective at the top and having another feature in the main 'bulk' of the GUI would be more user-friendly.

```

public class menuGUI extends JFrame {
    // the following public attributes are declared with meaningful names to be used throughout the constructor and other methods
    6 usages
    JMenuBar menubar;
    4 usages
    JMenu file, simplex, kruskals, prim;
    4 usages
    JMenuItem exit, home, runSimplex, runPrims, runKruskals, learnSimplex, learnPrims, learnKruskals;
    2 usages
    JLabel prompt;

        //adding 'file' menu option to the menu bar
        file = new JMenu( s: "File");
        menubar.add(file);
        home = new JMenuItem( text: "Home");
        file.add(home);
        exit = new JMenuItem( text: "Exit");
        file.add(exit);

        //creating menu options for each algorithm
        simplex = new JMenu( s: "Simplex Algorithm");
        menubar.add(simplex);

        kruskals = new JMenu( s: "Kruskal's Algorithm");
        menubar.add(kruskals);

        prim = new JMenu( s: "Prim's Algorithm");
        menubar.add(prim);

        //creating and adding the sub-options for each algorithm to the menubar
        runSimplex = new JMenuItem( text: "Run Algorithm");
        runPrims = new JMenuItem( text: "Run Algorithm");
        runKruskals = new JMenuItem( text: "Run Algorithm");

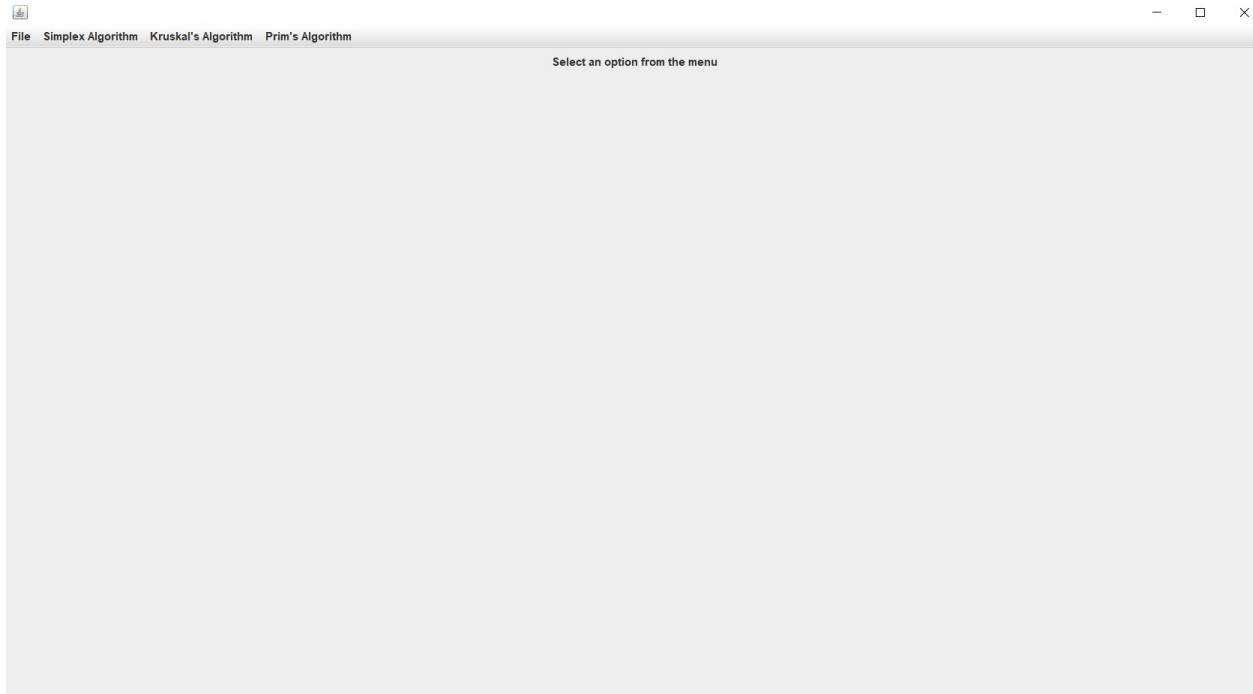
        simplex.add(runSimplex);
        kruskals.add(runKruskals);
        prim.add(runPrims);

        learnSimplex = new JMenuItem( text: "Learn Algorithm");
        learnKruskals = new JMenuItem( text: "Learn Algorithm");
        learnPrims = new JMenuItem( text: "Learn Algorithm");

        simplex.add(learnSimplex);
        kruskals.add(learnKruskals);
        prim.add(learnPrims);
    }
}

```

Result after these changes:



With the recommendation of having another feature in the center of the window, I decided to add a button for each algorithm which when clicked, would introduce each algorithm, fulfilling success criterion 9 and presenting information in a user-friendly manner as the buttons are present on the initial window without the user having to navigate the GUI to find the introductory section.

To maintain the prompt label as well as having a button for each algorithm introduction, I researched different Java Swing layouts that would allow me to create this feature. While the initial window was created using a flow layout, I decided that a more suitable layout for this feature would be a border layout, with the window split into 5 sections as illustrated below  
(<https://www.geeksforgeeks.org/java.awt.BorderLayout-class/>):



To keep the prompt label at the top, I created a new JPanel with another border layout, assigning it to the north section of the border layout, as well as creating some ‘buffer’ space to ensure components weren’t too crammed together in the window:

```
promptPanel = new JPanel(new BorderLayout());
prompt = new JLabel("Select an option from the menu", SwingConstants.CENTER);
promptPanel.add(prompt, BorderLayout.CENTER);
promptPanel.setBorder(BorderFactory.createEmptyBorder( top: 20, left: 0, bottom: 20, right: 0)); //defining the border size for the panel in the north section
add(promptPanel, BorderLayout.NORTH);
```

For the center section of the window, I created another JPanel called buttonPanel, this time utilising a grid layout with 1 row and 3 columns, one for each button to be added:

```
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout( rows: 1, cols: 3, hgap: 20, vgap: 50)); //splitting the center panel into 3 sections for 3 buttons
buttonPanel.setBorder(BorderFactory.createEmptyBorder( top: 5, left: 20, bottom: 60, right: 20)); //defining the border size for the button panel in the center section
```

The next components to be added were the buttons.

```

prompt = new JLabel( text: "Select an option from the menu", SwingConstants.CENTER);
add(prompt, BorderLayout.NORTH);
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout( rows: 1, cols: 3, hgap: 20, vgap: 50));

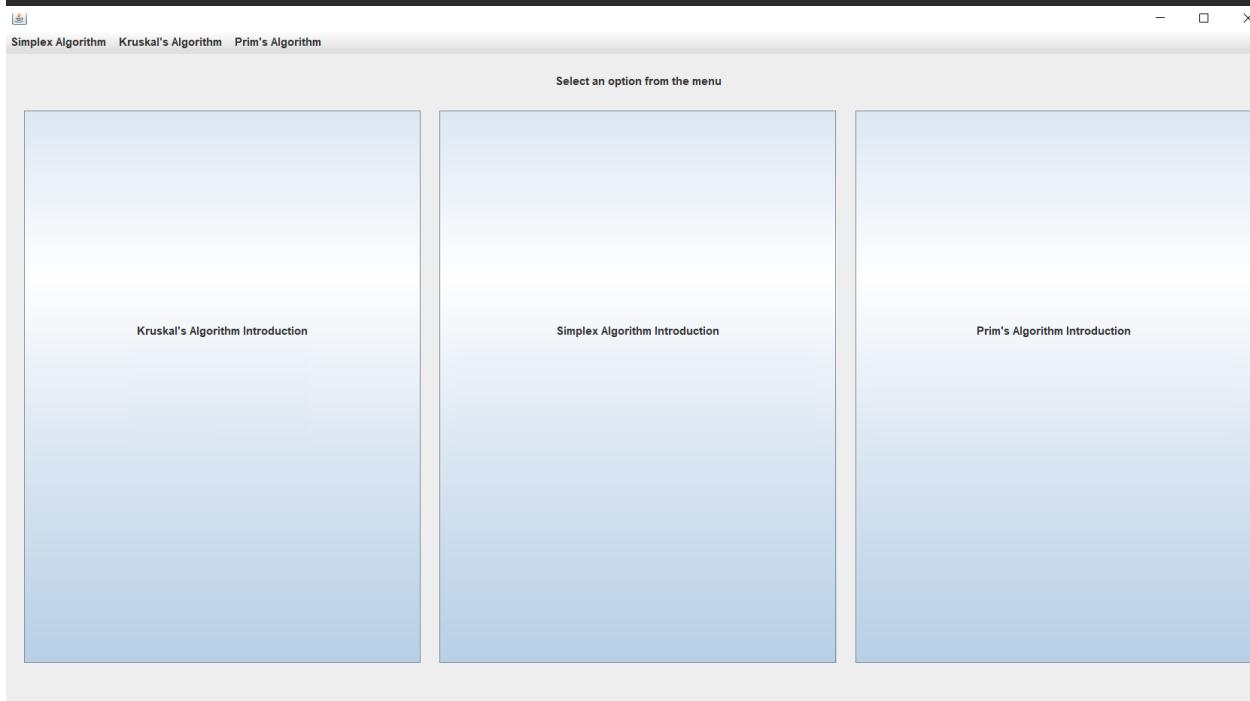
introSimplex = new JButton( text: "Simplex Algorithm Introduction");
buttonPanel.add(introSimplex);

introKruskals = new JButton( text: "Kruskal's Algorithm Introduction");
buttonPanel.add(introKruskals);

introPrims = new JButton( text: "Prim's Algorithm Introduction");
buttonPanel.add(introPrims);

add(buttonPanel, BorderLayout.CENTER);
event e = new event();
learnSimplex.addActionListener(e);

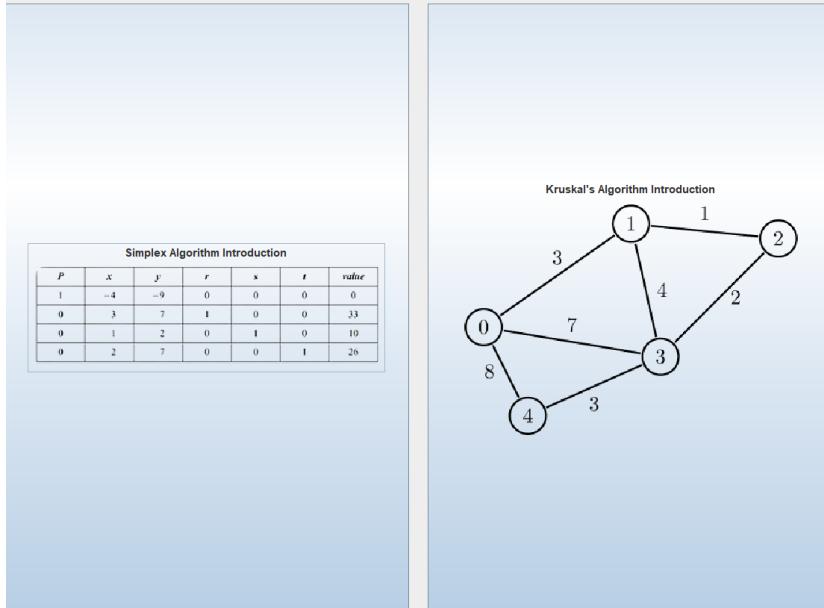
```



After creating the initial buttons, adding images proposed an issue regarding the layout of the buttons.

The main issue encountered with the buttons was aligning the label and images of the button. When the image and labels were first added, they were misaligned, and the text was positioned very close to the image. This made the text less impactful as there is a chance it could be missed by the user. Here is an image of the misaligned button contents. As shown by the image, the 'Simplex Algorithm

'Introduction' text is grouped with the image. Presenting this to my stakeholders, it was agreed upon that having some separation between the image and the text would be more user-friendly, clearly indicating the purpose of the button.



To fix this issue, I tried a variety of solutions, such as further splitting up the button section into 2 separate panels and adding the label and icon separately. This resulted in the text being separated from the button. To overcome this issue, I resized the images to have the same vertical height, before positioning the text above the image.

```

promptPanel = new JPanel(new BorderLayout());
prompt = new JLabel("Select an option from the menu", SwingConstants.CENTER);
promptPanel.add(prompt, BorderLayout.CENTER);
promptPanel.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0)); //defining the border size for the panel in the north section
add(promptPanel, BorderLayout.NORTH);

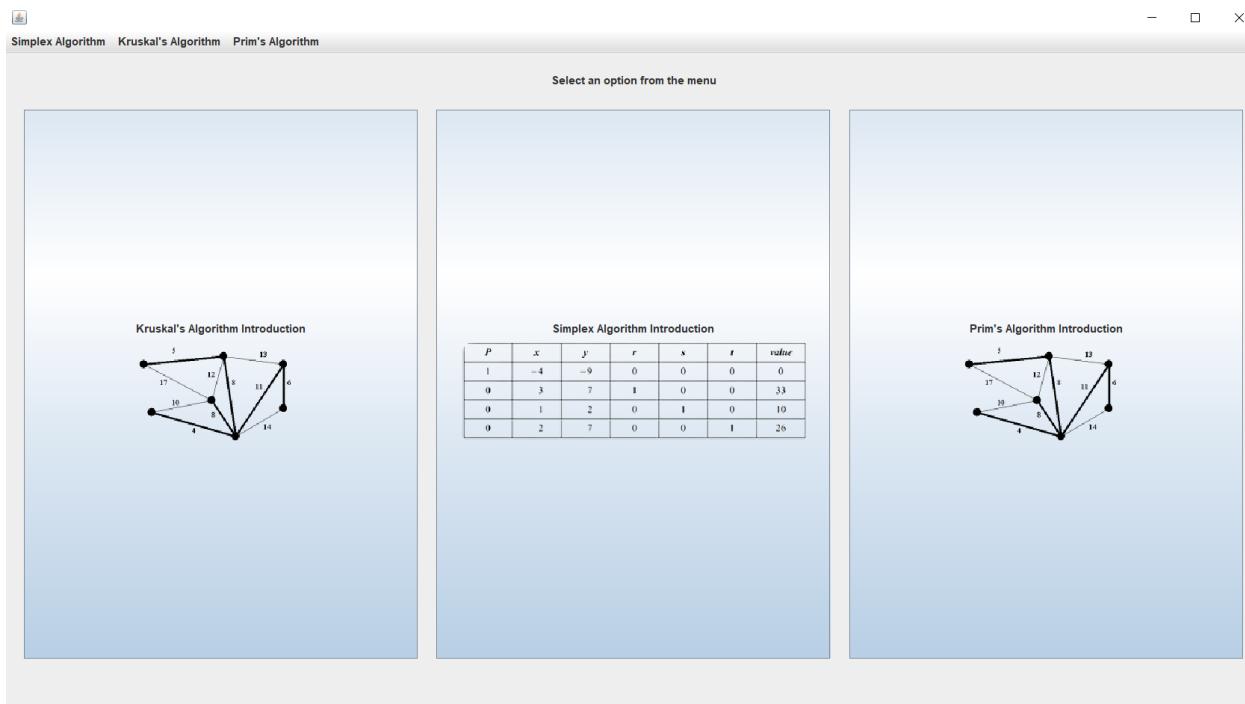
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1, 3, 20, 50)); //splitting the center panel into 3 sections for 3 buttons
buttonPanel.setBorder(BorderFactory.createEmptyBorder(5, 20, 60, 20)); //defining the border size for the button panel in the center section
add(buttonPanel, BorderLayout.NORTH);

//creating the button for Kruskal's algorithm introduction, with image, label and text positioning defined
ImageIcon kruskalsIcon = new ImageIcon("C:\\Users\\averm\\Pictures\\kruskalcon.png");
introKruskals = new JButton();
introKruskals.setText("Kruskal's Algorithm Introduction");
introKruskals.setHorizontalTextPosition(JButton.CENTER);
introKruskals.setVerticalTextPosition(JButton.TOP);
introKruskals.setFocusable(false);
introKruskals.setIcon(kruskalsIcon);
buttonPanel.add(introKruskals);

//creating the button for the simplex algorithm introduction, with image, label and text positioning defined
ImageIcon simplexIcon = new ImageIcon("C:\\Users\\averm\\Pictures\\simplex.png");
introSimplex = new JButton();
introSimplex.setText("Simplex Algorithm Introduction");
introSimplex.setHorizontalTextPosition(JButton.CENTER);
introSimplex.setVerticalTextPosition(JButton.TOP);
introSimplex.setFocusable(false);
introSimplex.setIcon(simplexIcon);
buttonPanel.add(introSimplex);

```

Download pre-built shared indexes  
Reduce the indexing time and pre-built JDK shared indexes  
 Always download [More](#)



The next step of this iteration was to add basic functions to each button and menu bar item. Since this prototype is just intended as a ‘shell’ layout, with no detailed actions being added yet, the extent of added functions in this iteration will be clearing the window of any objects for information to be presented in the future, exiting the window, and returning to the default state of the window when the ‘home’ option is selected.

Firstly, I instantiated an event object called ‘e’ and added it as an ActionListener to each component of the GUI that would need a function.

```
//adding an event action listener to each menu item within each menu bar option and each button
event e = new event();
learnSimplex.addActionListener(e);
learnPrims.addActionListener(e);
learnKruskals.addActionListener(e);
introSimplex.addActionListener(e);
introKruskals.addActionListener(e);
introPrims.addActionListener(e);
home.addActionListener(e);
exit.addActionListener(e);
```

I added the action to close the window when the ‘exit’ option inside the ‘File’ menu bar item was clicked:

```

public void actionPerformed(ActionEvent e){
    if (e.getSource() == exit) { //checks if source of event being called is 'exit' option
        System.exit( status: 0); //closes the window
    }
}

```

While my original attempt to add the 'home' button action was not successful, after testing it, I created 2 new methods called originalWindow() and restoreHome() which contained the code to create the button panel and buttons and added the buttons and panels to the window respectively. This allowed me to call originalWindow() in the constructor to create the window when the program is run, and call restoreHome() when the 'home' button is pressed to return to the original state of the window.

```

private void originalWindow(){
    promptPanel = new JPanel(new BorderLayout());
    prompt = new JLabel( text: "Select an option from the menu", SwingConstants.CENTER);
    promptPanel.add(prompt, BorderLayout.CENTER);
    promptPanel.setBorder(BorderFactory.createEmptyBorder( top: 20, left: 0, bottom: 20, right: 0));
    add(promptPanel, BorderLayout.NORTH);

    buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout( rows: 1, cols: 3, hgap: 20, vgap: 50));
    buttonPanel.setBorder(BorderFactory.createEmptyBorder( top: 5, left: 20, bottom: 60, right: 20));

    ImageIcon kruskalsIcon = new ImageIcon( filename: "C:\\\\Users\\\\averm\\\\Pictures\\\\kruskalicon.png");
    introKruskals = new JButton();
    introKruskals.setText("Kruskal's Algorithm Introduction");
    introKruskals.setHorizontalTextPosition(JButton.CENTER);
    introKruskals.setVerticalTextPosition(JButton.TOP);
    introKruskals.setFocusable(false);
    introKruskals.setIcon(kruskalsIcon);
    buttonPanel.add(introKruskals);

    ImageIcon simplexIcon = new ImageIcon( filename: "C:\\\\Users\\\\averm\\\\Pictures\\\\simplex.png");
    introSimplex = new JButton();
    introSimplex.setText("Simplex Algorithm Introduction");
    introSimplex.setHorizontalTextPosition(JButton.CENTER);
    introSimplex.setVerticalTextPosition(JButton.TOP);
    introSimplex.setFocusable(false);
    introSimplex.setIcon(simplexIcon);
    buttonPanel.add(introSimplex);
}

```

```

    ImageIcon primIcon = new ImageIcon( filename: "C:\\\\Users\\\\averm\\\\Pictures\\\\kruskalicon.png");
    introPrims = new JButton();
    introPrims.setText("Prim's Algorithm Introduction");
    introPrims.setHorizontalTextPosition(JButton.CENTER);
    introPrims.setVerticalTextPosition(JButton.TOP);
    introPrims.setFocusable(false);
    introPrims.setIcon(primIcon);
    buttonPanel.add(introPrims);
}

```

```

    public void restoreHome(){ //procedure which adds the initial components back to the window returning to the original state
        getContentPane().removeAll();
        add(promptPanel, BorderLayout.NORTH);
        add(buttonPanel, BorderLayout.CENTER);
        revalidate(); // ensuring all layout changes have been validated
        repaint(); //updating the window to display the changes to the window
    }

} else if (e.getSource() == home) { //checks if source of event is 'home' option
    restoreHome(); //restores the window to its original state
}

```

The last function to add was to clear the screen whenever one of the buttons or algorithm options are chosen. This was done using the `getContentPane().removeAll()` methods. An issue I encountered was the components not clearing when a button was clicked. This was discovered during the testing stage. To overcome this issue, I added the `revalidate()` and `repaint()` functions which validate all of the layout changes made before updating the window to present these changes.

```

else{
    getContentPane().removeAll(); //clears the window and updates any changes made
    revalidate();
    repaint();
}

```

Having added all objects currently required to the window and created all the preliminary functions, the first iteration of a GUI prototype shell is complete. I can now move onto testing the various components added to the GUI based on the test table in the design section which highlighted the necessary tests for this iteration.

## Iteration 2: Simplex Algorithm

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>)

The first part of this iteration involved creating a blueprint for a simplex problem, taking a 2d array of constraints and a one-dimensional array of objective

coefficients as parameters. I also created two public get methods for the constraints list and the objective list. Set methods are not currently required as the values of the user inputted and constraints will not need to be changed.

```
public class Simplex {
    2 usages
    private double [][] constraints;
    2 usages
    private double[] objective;

    1 usage
    public Simplex(double[][] constraintList, double[] objectiveFunction){ //constructor for a Simplex problem object
        constraints = constraintList;
        objective = objectiveFunction;
    }

    2 usages
    public double[][] getConstraints(){ //getter for constraints

        return constraints;
    }
    2 usages
    public double[] getObjective() { return objective; } //getter for objective
}
```

The next step involved creating a new class for all tableau related methods. I created a 2d array to store the initial tableau, as well as two integers, rows and cols, which will be passed as parameters for the dimensions of the initial tableau.

```
public class Tableau {
    19 usages
    private double[][][] table;
    12 usages
    private int rows;
    10 usages
    private int cols;

    1 usage
    public Tableau(Simplex problem){ //constructor to create an empty array for an initial tableau
        int numConstraints = problem.getConstraints().length; // the number of constraints is the length of the constraints array
        int numVar = problem.getObjective().length; //the number of variables is the same as the length of the objective array

        rows = numConstraints + 1;
        cols = numVar + numConstraints + 1;
        table = new double[rows][cols];
    }
}
```

As explained in the design section, the table's dimensions are slightly adapted versions of the number of inequalities and number of constraints, allowing the table to vary in size depending on user input. The tableau constructor takes an object of type Simplex as a parameter, allowing it to use the get methods to define the number of variables and constraints.

Having created the constructor method, the next method required is to take the values inputted by the user and add them to the table, hence I created a method called initialiseTableau(). The most important part of initialising the tableau is ensuring that adequate space is left to add the slack variables, otherwise the tableau will be generated incorrectly. This is demonstrated below:

Suppose we take this question again and we were to use a for loop to add each value from a list of constraints [5, 7, 70, 10, 3, 60, 3, 2]. Given the predefined parameters for the table, we would have an array that would be visualised like this:

5	7	70		
10	3	60		
3	2			

vs what it should look like

Solve the linear programming problem in Example 6 using simplex tableaux.

Maximise  $P = 3x + 2y$

subject to:

$$5x + 7y + r = 70$$

$$10x + 3y + s = 60$$

$$x, y, r, s \geq 0$$

**Notation** The word **tableau** is French. The plural of tableau is **tableaux**.

5	7	1	0	70
10	3	0	1	60
-3	-2	0	0	0

To counter this, we can use the number of variables - 1 (in other words, the length of a constraint - 1) as the upper limit counter for the for loop when adding to each column, and use the number of constraints for the upper limit counter for the for loop when adding to each row. Instead of iterating through the whole array, we can skip over n rows where n is the number of constraints = number of slack variables. To add the value to the array, we can add to the last column of the array, using the number of columns - 1 as the index for the final column.

To add the slack variables to the tableau, we can set the index [i][number of variables + i] to be 1.

```
private void initialiseTableau(Simplex problem){
    double[][] constraints = problem.getConstraints();
    double[] objective = problem.getObjective();
    int numVar = objective.length;

    for (int i = 0; i < constraints.length; i++){
        for(int j = 0; j < constraints[i].length - 1; j++){
            table[i][j] = constraints[i][j]; //adding the coefficients from constraints[][] to table[][]
        }
        table[i][numVar + i] = 1; //adding slack variables in respective indices
        table[i][cols - 1] = constraints[i][constraints[i].length - 1]; // adding the values in the value column
    }
    for(int j = 0; j < objective.length; j++){
        table[rows - 1][j] = -1 * objective[j]; //adding the values of the objective function in the bottom row * -1
    }
}
```

Having created the initial tableau, we now need 2 methods, one to locate the pivot column and one to locate the pivot row. As stated in the analysis, the pivot column is the column with the most negative value in the objective row, hence a basic for-loop can perform this action.

```

public int getPivotCol(){
    int pivotCol = 0;
    double minValue = table[rows - 1][0]; //sets the minimum value as the bottom left value of the table
    for(int i = 0; i < cols - 1; i++){ //iterates through objective row checking if next value smaller (more negative) than current minValue
        if(table[rows-1][i] < minValue){
            minValue = table[rows-1][i]; // if true, sets minValue as this new value
            pivotCol = i; //sets index of pivot column
        }
    }
    return pivotCol;
}

```

Finding the pivot value is achieved by dividing each number in the value column by the respective value in the pivot column and choosing the smallest positive result. The index providing this result becomes the pivot value.

```

public int getPivotRow(int pivotCol){
    int pivotRow = 0;
    double minRatio = Double.MAX_VALUE; //sets value of new variable equal to the max value offered by java
    for(int i = 0; i < rows - 1; i++){
        double ratio = table[i][cols - 1] / table[i][pivotCol]; //divides value column by pivot column
        if(ratio > 0 && ratio < minRatio){ // if result > 0 and < current minRatio then sets minRatio to current value
            minRatio = ratio;
            pivotRow = i; //sets index of pivot row
        }
    }
    return pivotRow;
}

```

Using the pivot value we have found, the next task is to create a method that pivots around this value, reducing it to 1 and every other value in the pivot column to 0. I realised that using a for loop to reduce each value in the pivot column to 0 would also do so to the pivot row, hence I started by reducing the pivot value to 1, then using a for loop which first checks that the current row is not the pivot row, before reducing to 0.

```

public void pivot(int pivotRow, int pivotCol){
    double pivotValue = table[pivotRow][pivotCol]; //using the pivot row and pivot column index, we can set a pivot value
    for (int j = 0; j < cols; j++){
        table[pivotRow][j] /= pivotValue; //goes through the pivot row dividing every value by the pivot value
    }
    for (int i = 0; i < rows; i++){
        if(i != pivotRow){
            double factor = table[i][pivotCol]; //goes through every other row subtracting the pivot column value from the current index value
            for (int j = 0; j < cols; j++){
                table[i][j] -= factor * table[pivotRow][j];
            }
        }
    }
}

```

After pivoting, it is required that we check if the table is optimal or not, if so, then we won't have to repeat the process again. To do this, I created a new method that iterates through the objective row and checks if any values are less than 0 (not optimal) and returns false if so, otherwise it returns true.

```

public boolean isOptimal(){
    for(int c = 0; c < cols - 1; c++){ // iterating through the objective row
        if(table[rows-1][c] < 0){ //if there are still negative values then we return false
            return false;
        }
    }
    return true;
}

```

The next method to be developed is one to extract the necessary values from the optimal table to output a solution. The main problem I encountered while developing this was trying to account for slack variables in the final output, as some questions require the student to output the value of both basic and non-basic variables. My initial method looked like this:

```

public double[] getSolution(){
    double[] solution = new double[cols - rows]; //creates a new array to store the values of the variables for a solution
    for(int i = 0; i < solution.length; i++){
        boolean isBasic = false; //iterates through the table checking if a variable is basic
        for(int j = 0; j < rows; j++){
            if(table[j][i] == 1){
                solution[i] = table[j][cols - 1]; //sets value in solution array equal to value in table
                isBasic = true; //if basic variable then breaks from the for loop
                break;
            }
        }
        if(!isBasic){
            solution[i] = 0;
        }
    }
}

```

This can correctly extract the values of the non-basic variables; however, it doesn't account for slack variables. To overcome this issue, I changed the size of the solution array to include both decision variables and slack variables. This also involved a change in the bound for the for-loop:

```

public double[] getSolution(){
    double[] solution = new double[cols - 1]; //creates a new array to store the values of the decision and slack variables for a solution
    for(int i = 0; i < cols - 1; i++){
        boolean isBasic = false; //iterates through the table checking if a variable is basic
        for(int j = 0; j < rows; j++){
            if(table[j][i] == 1){
                solution[i] = table[j][cols - 1]; //sets value in solution array equal to value in table
                isBasic = true; //if basic variable then breaks from the for loop
                break;
            }
        }
        if(!isBasic){
            solution[i] = 0;
        }
    }
    return solution;
}

```

If the value at index [j][i] is 1, indicating that this row was pivoted on, then index i in the solution array is set to the value of the respective row.

With the bulk of the functions for the algorithm now completed, the features left to add in the Tableau class were mainly for use by the main method and the actual solving process. The first of these is a method to print the table. In Java, when trying to print an array, instead of the value being printed, the output will be a string representation of the array's memory address. To output the array in a format that mimics a simplex tableau, I used a nested for-loop that outputs each value, leaving a gap between each row. I also used a format specifier to output floating point values with specified spacing and accuracy.

```
public void printTable(){
    for(double[] row: table){
        for(double value : row){
            System.out.printf("%10.2f", value); //leaves spacing of 10 and outputs with 2 decimal places
        }
        System.out.println();
    }
}
```

The next method is to get the optimised value from the table. This is always the bottom right value in the table; hence the following can be used to achieve this:

```
public double getOptimalValue(){
    return table[rows-1][cols-1]; //gets value in bottom right index of array
}
```

The final method in this class is a get method for the table:

```
public double[][] getTable(){ //get method for the table
    return table;
}
```

The next class to be implemented is to complete the iterations, finding pivots until the optimal solution has been found. Using the tableau class, we instantiate a new table with the parameter of an object of type Simplex. We then output the initial tableau:

```
public class Iterate {
    1 usage
    public static void solve(Simplex problem){
        Tableau table = new Tableau(problem); //creating a tableau object taking a parameter of a simplex problem object
        System.out.println("Initial Tableau:");
        System.out.println("");
        table.printTable();
```

Using the `isOptimal()` method from the `Tableau` class, we can then complete iterations of the algorithm until an optimal solution is found:

```
while(!table.isOptimal()){ //repeatedly finding pivot values and performing iterations while the table is not optimal
    int pivotCol = table.getPivotCol();
    int pivotRow = table.getPivotRow(pivotCol);

    table.pivot(pivotRow, pivotCol);
}
```

After an optimal solution is found, we can use the `getSolution()` method to output the values of the variables and the objective to the user. To ensure all necessary variables are outputted, I created an array to store the solutions, as well as two integers which will be used to define the parts of the array storing decision variables and separate it from the part of the array storing slack variables.

```
double[] solution = table.getSolution();
int numDecisionVariables = problem.getObjective().length; // the number of decision variables is the number of objective function coefficients
int numSlackVariables = solution.length - numDecisionVariables; // the number of slack variables is the total number of variables - number of decision variables
```

To output the variables, I will print out each decision variable as  $x_1, x_2, x_3$  etc, and each slack variable as  $s_1, s_2, s_3$  etc. For the decision variables, we print out the first section of the solution array. For the slack variables we output the rest of the array, starting from the index of `numDecisionVariables + 1` until the end of the array. The last value outputted is the optimal value:

```
double[] solution = table.getSolution();
int numDecisionVariables = problem.getObjective().length; // the number of decision variables is the number of objective function coefficients
int numSlackVariables = solution.length - numDecisionVariables; // the number of slack variables is the total number of variables - number of decision variables

System.out.println("The optimal solution is: ");
for(int i = 0; i < numDecisionVariables; i++){
    System.out.println("x" + (i+1) + " = " + solution[i]); //printing from start of solution array to index numDecisionVariables
}
for(int i = 0; i < numSlackVariables; i++){
    System.out.println("s" + (i+1) + " = " + solution[numDecisionVariables + 1]); //printing from numDecisionVariables + 1 to end of array
}
System.out.println("Optimal value: " + table.getOptimalValue()); //using getOptimalValue() method from Tableau class to output optimal value
```

The final class to develop is the class containing the main method, taking user inputs and instantiating a new object of type `Simplex`. Then using the `solve()` method from the `Iterate` class, the problem will be solved and the values outputted.

Creating a scanner object and taking user inputs which will be used at various points:

```

public static void main(String[] args){
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter number of variables: "); //prompting user inputs which will be used to create the tableau
    int numVar = scanner.nextInt();
    System.out.println("Enter number of constraints: ");
    int numConstraints = scanner.nextInt();
}

```

Creating a 2d array to store the constraints coefficients and using a for loop to allow the user to keep entering values until they reach the number of constraints they specified:

```

double[][] constraints = new double[numConstraints][numVar + 1]; //creating a 2d array to stores the coefficients of constraints
System.out.println("Enter the coefficients of the constraints and include the value: ");
for(int i = 0; i < numConstraints; i++){ //using the inputted number of constraints as upper limit for loop counter
    for(int j = 0; j < numVar + 1; j++){
        constraints[i][j] = scanner.nextDouble(); //adding entered coefficients of constraints to 2d array
    }
}

```

The last user input is the objective function, before creating an object of the Simplex class and solving it:

```

double[] objective = new double[numVar];
System.out.println("Enter coefficients of the objective function: ");
for (int i = 0; i < numVar; i++) { //using the number of variables to detect how many coefficients should be entered
    objective[i] = scanner.nextDouble();
}
Simplex problem = new Simplex(constraints, objective); //creating a new object of type Simplex
Iterate.solve(problem); //solving the created problem

```

Having now developed the methods to perform the Simplex algorithm on inputted inequalities and an objective function, the development section of this iteration is now complete, allowing me to move onto the testing section for this iteration. This iteration will require the functionality of the algorithm to be tested on various questions that would be presented to [SCHOOL NAME ABBREVIATION] students in an Edexcel A-Level Further Maths Decision exam.

## Iteration 3: Kruskal's Algorithm

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>)

The first part of this iteration I will implement is the edge class, forming a basis upon which the algorithm can be performed upon. This consists of a constructor to

create a new edge, as well as a method to compare two edges, a vital part of Kruskal's algorithm.

```
import java.util.*;
7 usages
class Edge implements Comparable<Edge>{ //creating a class which defines the attributes and method of each edge/arc
    3 usages
    int source, dest, weight;
    1 usage
    public Edge(int nodeSource, int nodeDest, int nodeWeight){ //constructor for edge class
        source = nodeSource;
        dest = nodeDest;
        weight = nodeWeight;
    }
    public int compareTo(Edge otherEdge){ //method to compare the weight of two different edges
        return weight - otherEdge.weight;
    }
}
```

Having outlined all the necessary attributes and methods for the edge class, I can now move onto developing the Kruskals class which will contain methods to perform the algorithm.

The first of these is the constructor method which takes the number of vertices in the graph as a parameter and assigns it a value, as well as creating an arraylist of edges:

```
public class Kruskals{
    4 usages
    private ArrayList<Edge> edges;
    4 usages
    private int numVertices;

    1 usage
    public Kruskals(int vertices){ //constructor for Kruskal's algorithm class
        numVertices = vertices;
        edges = new ArrayList<>();
    }
}
```

To make the implementation of the algorithm easier, I created some additional methods which will be used in the main solving method. This includes the addEdge method, which takes in 3 parameters and adds them as an edge to the arraylist of edges created by the constructor. The other method is to find the component of the graph that an edge belongs to. When deciding whether to accept or reject an edge in Kruskal's algorithm, you must check whether adding that edge will form a

cycle in the graph. This can be done by checking whether the vertices of the edge are part of the minimum spanning tree that is being formed. To make this check, I developed the findComponent method.

```
public void addEdge(int source, int dest, int weight){ //method to add edge from graph to arraylist
    edges.add(new Edge(source, dest, weight));
}
2 usages
private int findComponents(int vertex, ArrayList<Integer> components){ //finds the component of the graph that a vertex belongs to
    return components.get(vertex);
}
1 usage
```

The next method I implemented was to actually perform Kruskal's algorithm. First I sorted the array of edges using Collections.sort(), before creating a new list with the number of elements = the number of vertices in the graph. I then needed somewhere to store the edges added to the final MST, so I created a new arraylist called result.

```
public void kruskalMST(){
    Collections.sort(edges); //sorts the arraylist of edges into ascending order
    ArrayList<Integer> components = new ArrayList<>(Collections.nCopies(numVertices, 0));
    for(int i = 0; i < numVertices; i++){
        components.set(i, i);
    }
    ArrayList<Edge> result = new ArrayList<>(); //creates a new arraylist to store the edges of the MST in
```

The next part of this method involves iterating through the list of ordered edges, comparing the component of the source and destination of the edge, before adding or rejecting it from the minimum spanning tree. I did this creating 4 new variables and using a for loop to iterate through the edges.

```

for(Edge edge : edges){ //iterating through each edge in the arraylist of all edges from original graph
    int s = edge.source;
    int d = edge.dest;

    int compS = findComponents(s, components);
    int compD = findComponents(d, components);

    if(compS != compD){ //comparing the edges and adding the edge to the result arraylist if the addition of the edge doesn't form a cycle
        result.add(edge);
        for(int i = 0; i < numVertices; i++){
            if(components.get(i) == compD){
                components.set(i, s);
            }
        }
    }
}

```

The second for loop is executed only if the components of the source and destination aren't the same and this loop merges the component of the destination vertex with the component of the starting vertex.

The final part of this method is to output the results in the console, this involves each edge in the result, it's source, destination and weight, as well as the final weight of the minimum spanning tree.

```

System.out.println("Edges included in the minimum spanning tree: "); //outputs the final edges in the MST
for(Edge edge: result){
    System.out.println("Source: " + edge.source + ", Destination: " + edge.dest + ", Weight: " + edge.weight);
}

```

To check the functionality of the algorithm before moving on to the GUI implementation, I manually added some edges of a graph from the Edexcel A-Level Decision Maths textbook, and ran the program, which correctly outputted the final weight and relevant edges:

```

public static void main(String[] args){
    Kruskals graph = new Kruskals( vertices: 5); //instantiates an object of type Kruskals
    graph.addEdge( source: 0, dest: 1, weight: 7); //defining the source, destination and weight of each edge in the graph
    graph.addEdge( source: 0, dest: 3, weight: 6);
    graph.addEdge( source: 0, dest: 4, weight: 5);
    graph.addEdge( source: 1, dest: 2, weight: 5);
    graph.addEdge( source: 1, dest: 3, weight: 6);
    graph.addEdge( source: 2, dest: 3, weight: 8);
    graph.addEdge( source: 3, dest: 4, weight: 4);

    graph.kruskalMST(); //running kruskal's algorithm on the graph
}

```

While testing using the textbook questions, I noticed how the nodes/vertices in the textbook were given as letters (A, B, C etc), whereas my program currently used

numbers. Ensuring that the performance of the algorithm matches the A-Level specification as much as possible forms part of the success criteria. To fix this, I noticed that Java can deal with characters and integers similarly, however the ASCII values would vary, so in the constructor method for the edges, I subtracted a fixed amount from each value. This allowed an input of 'A' for example to be correctly converted to the number 0 for use by the algorithm.

This now allows for inputs in the following form.

```
public static void main(String[] args){  
    Kruskals graph = new Kruskals( vertices: 5); //instantiates an object of type Kruskals  
    graph.addEdge( source: 'A', dest: 'B', weight: 7); //defining the source, destination and weight of each edge in the graph  
    graph.addEdge( source: 'A', dest: 'D', weight: 6);  
    graph.addEdge( source: 'A', dest: 'E', weight: 5);  
    graph.addEdge( source: 'B', dest: 'C', weight: 5);  
    graph.addEdge( source: 'B', dest: 'D', weight: 6);  
    graph.addEdge( source: 'C', dest: 'D', weight: 8);  
    graph.addEdge( source: 'D', dest: 'E', weight: 4);  
  
    graph.kruskalMST(); //running kruskal's algorithm on the graph  
}
```

Checking this with the same graph as before, but using letters, the program produced the same correct output.

The next part of this iteration involves integrating the algorithm into the GUI, so the user can input the edges into the GUI and receive a correct output. For this iteration, the output will take place in the console, which will be built upon in further iterations, in which I am to display the result visually on a graph.

The first part involved creating a panel of buttons which will appear when the 'Run Kruskal's' menu item is selected. For this I used a flow layout so the buttons appear horizontally, as well as including a separator line to separate the button options from the text areas to come later.

```

public void graphSettings() {
    getContentPane().removeAll(); //removes all current components from the window
    promptPanel.removeAll();
    mainPanel = new JPanel(new BorderLayout()); //creates a new panel with a border layout
    graphButtonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, hgap: 10, vgap: 10)); // a new panel for the three buttons
    graphButtonPanel.add(addEdge);
    graphButtonPanel.add(removeEdge);
    graphButtonPanel.add(runAlgorithm);

    mainPanel.add(graphButtonPanel, BorderLayout.NORTH);

    JSeparator separator = new JSeparator(SwingConstants.HORIZONTAL); //this creates a separator line underneath the buttons
    mainPanel.add(separator, BorderLayout.CENTER);

    add(mainPanel, BorderLayout.NORTH);
    revalidate();
    repaint();
}

```

I added a gap in the flowlayout to give adequate spacing between the buttons to ensure the GUI doesn't become too cramped together, fulfilling part of the GUI success criteria that involves a clearly layed out user interface.

```

public void graphSettings() {
    getContentPane().removeAll(); //removes all current components from the window
    promptPanel.removeAll();
    mainPanel = new JPanel(new BorderLayout()); //creates a new panel with a border layout
    graphButtonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, hgap: 10, vgap: 10)); // a new panel for the three buttons
    graphButtonPanel.add(addEdge);
    graphButtonPanel.add(removeEdge);
    graphButtonPanel.add(runAlgorithm);

    mainPanel.add(graphButtonPanel, BorderLayout.NORTH);

    JSeparator separator = new JSeparator(SwingConstants.HORIZONTAL); //this creates a separator line underneath the buttons
    mainPanel.add(separator, BorderLayout.CENTER);

    add(mainPanel, BorderLayout.NORTH);
    revalidate();
    repaint();
}

```

I put this section of code in a method which can then be called when the Run Kruskal's option is selected by the user. Next was to add functionality to the buttons. The 'Add Edge' button is the main focus for this iteration, so I created a new method called edge() which adds the text areas and labels to the window when the button is pressed.

```

public void edge(){
    kruskalAdd = new JPanel(new FlowLayout()); //new panel underneath the button panel

    sourceTA = new JTextArea( rows: 2, columns: 5); //adding the text areas to get user input for source, destination and weight
    destTA = new JTextArea( rows: 2, columns: 5);
    weightTA = new JTextArea( rows: 2, columns: 5);
    sourceLabel = new JLabel( text: "Source:");
    destinationLabel = new JLabel( text: "Destination:");
    weightLabel = new JLabel( text: "Weight:");
    vertices = new JTextArea( rows: 2, columns: 5); // adds the text area for user to enter number of vertices
    numV = new JLabel( text: "Number of Vertices:");
}

```

For the dimensions of the buttons, I tried out different sizes until I reached a reasonable size that isn't disproportionately large but is also significant enough that the user knows where to put their inputs. These text areas and labels were all added to a new panel called kruskalAdd, which is added underneath the button panel.

```

kruskalAdd.add(sourceLabel); //the following section of code adds each of these labels and text areas to the panel
kruskalAdd.add(sourceTA);
kruskalAdd.add(Box.createVerticalStrut( height: 10)); // 10 pixels of vertical spacing
kruskalAdd.add(destinationLabel);
kruskalAdd.add(destTA);
kruskalAdd.add(Box.createVerticalStrut( height: 10)); // 10 pixels of vertical spacing
kruskalAdd.add(weightLabel);
kruskalAdd.add(weightTA);
kruskalAdd.add(addButton);
kruskalAdd.add(numV);
kruskalAdd.add(vertices);

mainPanel.add(kruskalAdd, BorderLayout.CENTER); //adding the new panel to the main panel

revalidate();
repaint();

```

The next method I created was to extract the information from the text areas so it can be passed into the methods for Kruskal's algorithm. This method is called when the 'Add' button is pressed. Since the inputs are stored as strings, I had to convert the source and destination into characters, and the weight into an integer so they are recognised as the correct data type by the algorithm.

```

public void addToGraph(){
    String source = sourceTA.getText(); //extracting the inputted text from the text areas
    char sourceCH = source.charAt(0); //converting the string inputs to characters
    String dest = destTA.getText();
    char destCH = dest.charAt(0);
    String weight = weightTA.getText();
    int weightInt = Integer.parseInt(weight); //converting the string input to an integer

    toAdd.add((int)sourceCH); //adds the source, destination and weight to an arraylist of information about edges to add
    toAdd.add((int)destCH);
    toAdd.add(weightInt);
}

```

I then added all the inputs from the text areas to an array of information that is to be added to the graph.

Having extracted the information from the text areas in the GUI and added it to an arraylist, I can now add the inputted edges to a graph and perform the algorithm using the methods from the Kruskals class. I first extracted the number of vertices inputted by the user from the text area as this is required to instantiate an object of type Kruskals. Since each item from the text area was added individually, I had to ensure that the source, destination and weight of a single edge was preserved in that order. To do this, I used a for loop that iterates through the array, using index i as the source, i+1 as the destination and i+2 as the weight. To ensure that the same data isn't used multiple times, I made the arraylist increment by 3, so it moves edge by edge through the array.

```

public void runKruskalsAlg(){
    String graphV = vertices.getText(); //extracts the text for the number of vertices from the text area
    char numVertices = graphV.charAt(0); //converts the string input to a character

    int graphSize = numVertices; //converts the character to an integer to be used as a parameter
    Kruskals graph = new Kruskals(graphSize); //creating a new object of type Kruskals
    for(int i = 0; i + 2 < toAdd.size(); i+=3){ //incrementing in 3 to ensure all 3 pieces of information for a single node are added correctly
        graph.addEdge(toAdd.get(i), toAdd.get(i+1), toAdd.get(i+2)); //adding the source destination and weight as an edge
    }
    graph.kruskalMST(); //performs the algorithm on the graph
}

```

Finally I run the algorithm on the graph created, concluding iteration 3 as a basis of Kruskal's Algorithm.

Having now developed the methods to perform Kruskal's algorithm based on the pseudocode and GUI design that was created in the Design section for this iteration, I can now move onto testing the various components with the created test plan to ensure proper functionality.

## Iteration 4: Simplex Algorithm with GUI Implementation

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

The development section of this iteration will start with adding the components to the GUI correctly, then adding functionality to each of them. The first options will appear when the Run Algorithm sub-option from the menu bar is selected in the Simplex menu item. When this button is pressed, the GUI that I designed and consulted with stakeholders should appear. This involves the drop-down menus, 3 variable buttons, run button and the separator line. I developed this in the method shown below.

```
public void simplexSettings() {  
    String[] inequalityOptions = {"--", "1", "2", "3", "4"}; //setting the options for the drop-down menu  
    numInequality = new JComboBox(inequalityOptions); //creating the drop-down menu and a label for it  
    numberInequality = new JLabel("Select number of inequalities");  
    getContentPane().removeAll(); //removes all current components from the window  
    promptPanel.removeAll();  
    simPanel = new JPanel(new BorderLayout()); //creates a new panel with a border layout  
    simButtonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10)); // a new panel for the three buttons
```

I created the buttons in the main constructor and gave them all labels:

```
twoVar = new JButton("2 Variables");  
threeVar = new JButton("3 Variables");  
fourVar = new JButton("4 Variables");  
  
runSimplexBUTTON = new JButton("Run");
```

The next part involved adding each of the buttons to the panel so they are displayed when the panel is displayed:

```

simButtonPanel.add(numberInequality); //adding the components to the top panel
simButtonPanel.add(numInequality);
simButtonPanel.add(twoVar);
simButtonPanel.add(threeVar);
simButtonPanel.add(fourVar);
simButtonPanel.add(runSimplexButton);

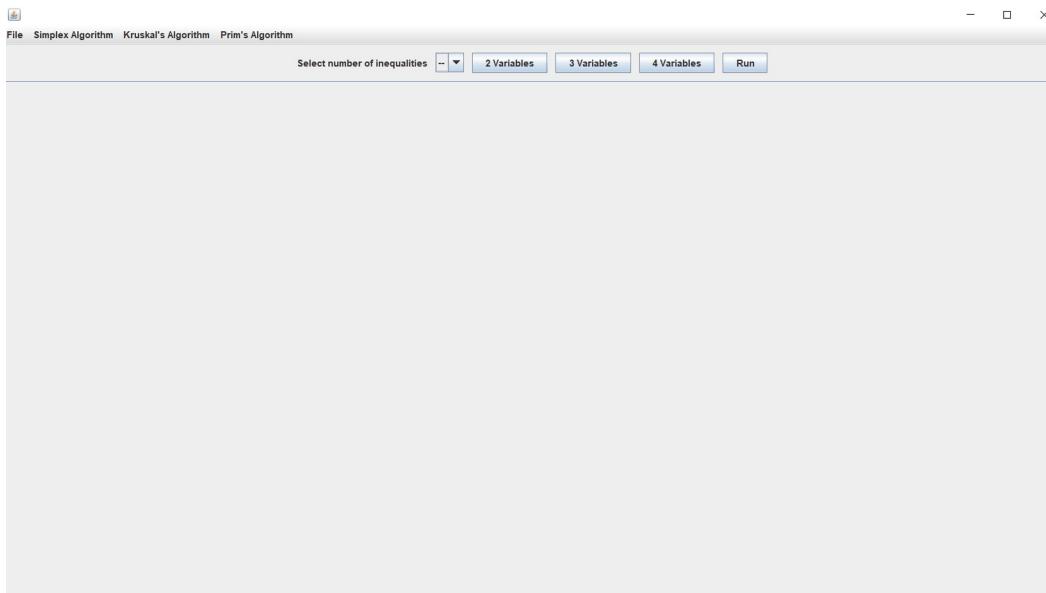
simPanel.add(simButtonPanel, BorderLayout.NORTH); //adding this to the main panel

JSeparator separator = new JSeparator(SwingConstants.HORIZONTAL); //this creates a separator line underneath the buttons
simPanel.add(separator, BorderLayout.CENTER);

add(simPanel, BorderLayout.NORTH);
revalidate(); //updating the GUI
repaint();

```

Before moving on, I ran a quick test to ensure that everything is in the correct place on the GUI:



The program successfully compiles, and all the components are displayed in the correct place respectively. I can now move on to adding functionality to each of the buttons. For the method to display the rows of inequalities, I will pass a button object as a parameter, then use selection to check the source of the button press. Depending on which button was pressed, a different number of variables will appear in each row:

```

public void setSimplex(JButton button){
    simplexPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, hgap: 5, vgap: 1)); //new panel which goes below the buttons
    inequalityPanel = new JPanel(); //panel to contain various row panels, using a box layout to stack them on top of each other
    inequalityPanel.setLayout(new BoxLayout(inequalityPanel, BoxLayout.Y_AXIS));

    int counter = Integer.parseInt(numInequality.getSelectedItem().toString()); //initialises a counter variable

```

When the two variables button is selected, the following section of code runs:

```

if (button == twoVar && (numInequality.getSelectedItem().toString() != "-")){ //checks which button pressed and ensures a value was selected from the drop-down
    numVAR = 2;
    for (int i = 0; i < counter; i++) { //for each row of inequalities
        JPanel rowPanel = new JPanel(); //new panel is created
        rowPanel.setLayout(new FlowLayout());

        JLabel plus1 = new JLabel("text: "+"); //initialising all the components that need to be added in each row
        plus1.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10)); //ensuring there is spacing between the components
        xTA = new JTextArea( rows: 2, columns: 5);
        yTA = new JTextArea( rows: 2, columns: 5);
        xlabel = new JLabel( text: "x");
        xlabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        ylabel = new JLabel( text: "y");
        ylabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        xlabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
        ylabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
        valueLabel = new JLabel( text: "=");
        valueLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        valueLabel.setBorder(new EmptyBorder( top: 0, left: 20, bottom: 0, right: 20));
        valueTA = new JTextArea( rows: 2, columns: 5);

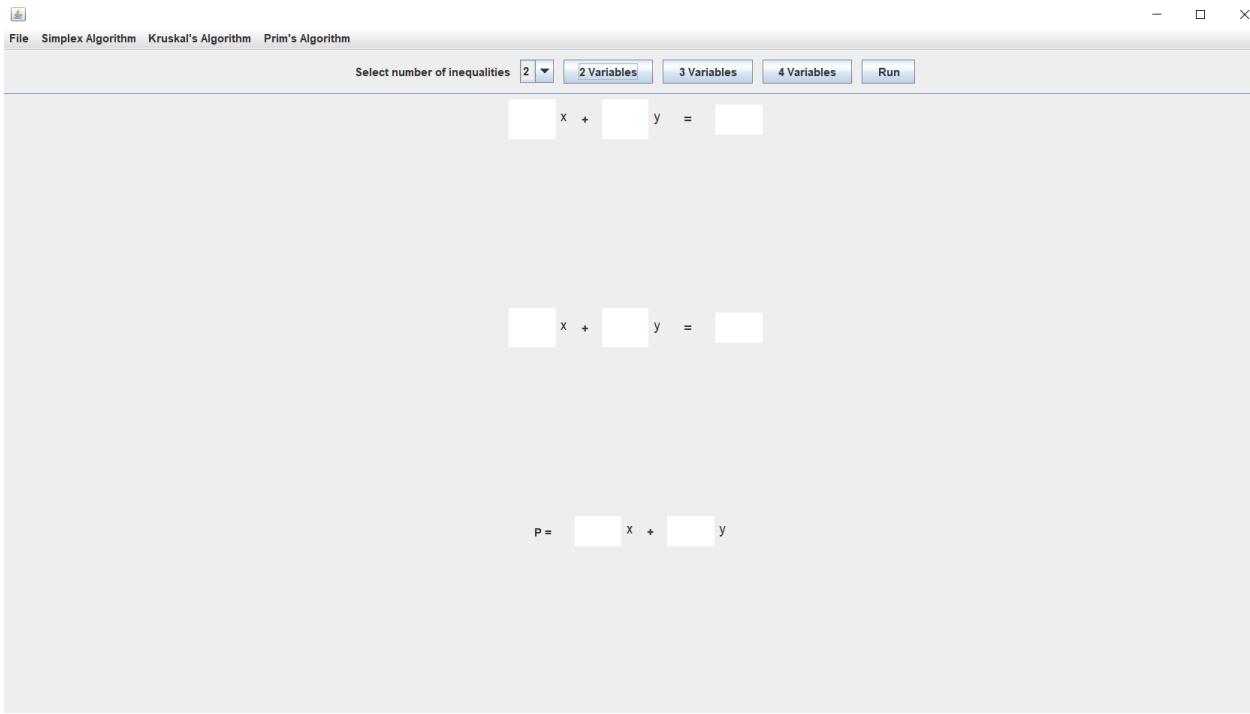
        xTA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
        yTA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));

        rowPanel.add(xTA); //adding each of the components to the row panel
        rowPanel.add(xLabel);
        rowPanel.add(plus1);
        rowPanel.add(yTA);
        rowPanel.add(yLabel);
        rowPanel.add(valueLabel);
        rowPanel.add(valueTA);

        inequalityPanel.add(rowPanel); //adding the row of components to the inequality panel
    }
}

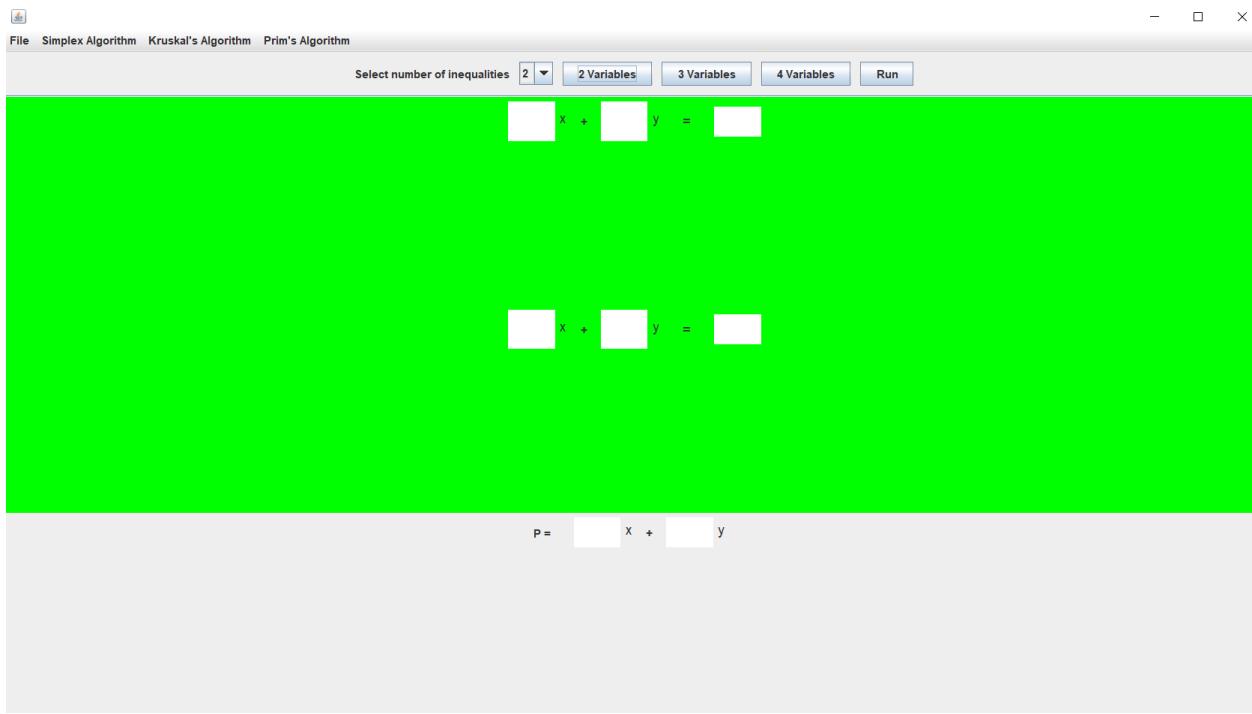
```

Since the 3 variable and 4 variable buttons will follow similar logic for adding components to the GUI when they are pressed, I will quickly check that the components are correctly added for the 2-variable button:

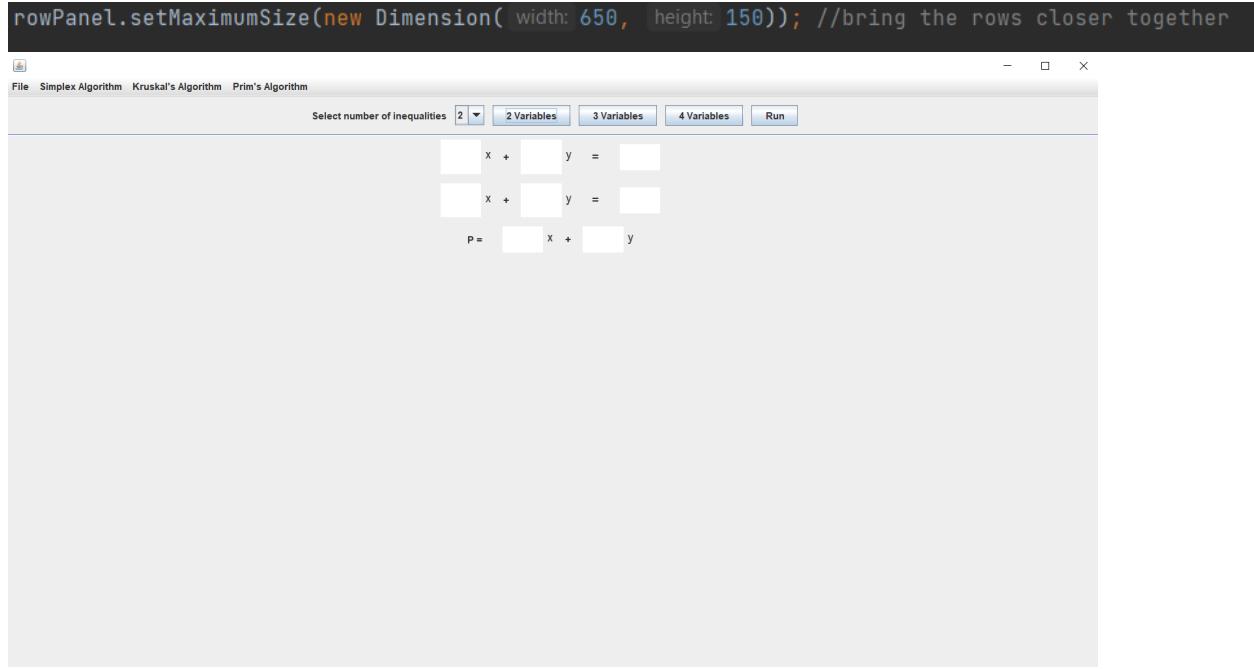


The components are added to the GUI; however, they are spaced very far apart. First, I needed to find the source of the issue. Since each of the rows are the same distance apart, I narrowed the issue down to within the for-loop. As I couldn't immediately spot the error, I added a line of code to colour the panels, which will allow me to easily identify whether it is a sizing issue or not.

```
rowPanel.setBackground(Color.green);|
```



From this, I can see that each of the row panels are too large, so despite the fact that the row of components is added to the very top of the panel, the space below the row is too large, leaving a big gap between the rows. To fix this, I used the `setMaximumSize()` function, which allows me to limit the size of a component. In this case, I am going to limit the size of the row panel, which should move the panels closer together. Trying out a few dimensions, I eventually settled on the following.



The rows are now added closer together, with some gap still left between them to allow the user to clearly distinguish between the different text areas. This allows me to develop the code for 3 variables and 4 variables:

For 3 variables, I followed a similar process, however, this time I introduced the 'z' text area and label, formatting them in the same way:

```

for (int i = 0; i < counter; i++) {
    JPanel rowPanel = new JPanel(); //creating a new panel for each row
    rowPanel.setLayout(new FlowLayout());
    rowPanel.setMaximumSize(new Dimension( width: 650, height: 150));

    JLabel plus1 = new JLabel( text: "+" ); //creating the labels to go between the text areas
    plus1.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

    JLabel plus2 = new JLabel( text: "+" );
    plus2.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

    xTA = new JTextArea( rows: 2, columns: 5); //defining the components for each row
    yTA = new JTextArea( rows: 2, columns: 5);
    zTA = new JTextArea( rows: 2, columns: 5);
    xLabel = new JLabel( text: "x");
    xLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
    yLabel = new JLabel( text: "y");
    yLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
    zLabel = new JLabel( text: "z");
    zLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
    xLabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0)); //adding adequate spacing between components
    yLabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
    zLabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
    valueLabel = new JLabel( text: "=");
    valueLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
    valueLabel.setBorder(new EmptyBorder( top: 0, left: 20, bottom: 0, right: 20));
    valueTA = new JTextArea( rows: 2, columns: 5);

    xTA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
    yTA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
    zTA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));

    rowPanel.add(xTA); //adding each of the components to the rowPanel in the order they appear due to flow layout
    rowPanel.add(xLabel);
    rowPanel.add(plus1);
    rowPanel.add(yTA);
    rowPanel.add(yLabel);
    rowPanel.add(plus2);
    rowPanel.add(zTA);
    rowPanel.add(zLabel);
    rowPanel.add(valueLabel);
    rowPanel.add(valueTA);

    inequalityPanel.add(rowPanel); //adding each row panel to the inequality panel
}

```

```

JPanel objectivePanel = new JPanel(); //creating an objective panel to go at the bottom of the inequality panel
objectivePanel.setLayout(new FlowLayout()); //the panel will have a flow layout
objectivePanel.setMaximumSize(new Dimension( width: 650, height: 150)); //limiting the size of the panel
objLabel = new JLabel( text: "P ="); //the following section defines the components that go into the objective function
objLabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 0, right: 20));
xObjTA = new JTextArea( rows: 2, columns: 5);
yObjTA = new JTextArea( rows: 2, columns: 5);
zObjTA = new JTextArea( rows: 2, columns: 5);
xObj = new JLabel( text: "x");
xObj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
yObj = new JLabel( text: "y");
yObj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
zObj = new JLabel( text: "z");
zObj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
xObj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0)); //setting borders to ensure spacing between components
yObj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
zObj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
plusObj = new JLabel( text: "+");
plusObj.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));
JLabel plus1Obj = new JLabel( text: "+");
plus1Obj.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

objectivePanel.add(objLabel); //adding all the created components to the objective panel
objectivePanel.add(xObjTA);
objectivePanel.add(xObj);
objectivePanel.add(plusObj);
objectivePanel.add(yObjTA);
objectivePanel.add(yObj);
objectivePanel.add(plus1Obj);
objectivePanel.add(zObjTA);
objectivePanel.add(zObj);
inequalityPanel.add(objectivePanel); //adding the panel to the inequality panel

```

For 4 variables, once again a similar process of creating and adding the panels and components was followed, however, when looking at the Pearson A-Level Further Maths Decision 1 textbook, I found that all problems involving 4 variables had them labelled as  $x_1, x_2, x_3, x_4$ , hence for my labels and text areas, I followed the same notation. These changes also took place in the objective panel.

```

} else if (button == fourVar && (numInequality.getSelectedItem().toString() != "--") {

    numVAR = 4;
    for (int i = 0; i < counter; i++) {
        JPanel rowPanel = new JPanel();
        rowPanel.setLayout(new FlowLayout());
        rowPanel.setMaximumSize(new Dimension( width: 650, height: 150));

        JLabel plus1 = new JLabel( text: "+" ); //this time adding 3 plus symbols as labels to the GUI
        plus1.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

        JLabel plus2 = new JLabel( text: "+" );
        plus2.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

        JLabel plus3 = new JLabel( text: "+" );
        plus3.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

        x1TA = new JTextArea( rows: 2, columns: 5); //defining the text areas and components of the GUI
        x2TA = new JTextArea( rows: 2, columns: 5);
        x3TA = new JTextArea( rows: 2, columns: 5);
        x4TA = new JTextArea( rows: 2, columns: 5);

        x1Label = new JLabel( text: "x1");
        x1Label.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        x2Label = new JLabel( text: "x2");
        x2Label.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        x3Label = new JLabel( text: "x3");
        x3Label.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        x4Label = new JLabel( text: "x4");
        x4Label.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        x1Label.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 20, right: 0)); //creating border spacing around the components
        x2Label.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 20, right: 0));
        x3Label.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 20, right: 0));
        x4Label.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 20, right: 0));
        valueLabel = new JLabel( text: "=");
        valueLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
        valueLabel.setBorder(new EmptyBorder( top: 0, left: 20, bottom: 10, right: 20));
        valueTA = new JTextArea( rows: 2, columns: 5);

        x1TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
        x2TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
        x3TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
        x4TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));

```

```
x1TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
x2TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
x3TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));
x4TA.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 10, right: 0));

rowPanel.add(x1TA); //adding each of the components to the row panel created at the start of the for loop
rowPanel.add(x1Label);
rowPanel.add(plus1);
rowPanel.add(x2TA);
rowPanel.add(x2Label);
rowPanel.add(plus2);
rowPanel.add(x3TA);
rowPanel.add(x3Label);
rowPanel.add(plus3);
rowPanel.add(x4TA);
rowPanel.add(x4Label);
rowPanel.add(valueLabel);
rowPanel.add(valueTA);

inequalityPanel.add(rowPanel);
```

```

}

JPanel objectivePanel = new JPanel(); //creating a new panel for the objective function
objectivePanel.setLayout(new FlowLayout());
objectivePanel.setMaximumSize(new Dimension( width: 650, height: 150));
objLabel = new JLabel( text: "P =" );
objLabel.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 0, right: 20));
x1ObjTA = new JTextArea( rows: 2, columns: 5); //defining each of the text areas that will appear
x2ObjTA = new JTextArea( rows: 2, columns: 5);
x3ObjTA = new JTextArea( rows: 2, columns: 5);
x4ObjTA = new JTextArea( rows: 2, columns: 5);
x1Obj = new JLabel( text: "x1");
x1Obj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15)); //altering the font style and size to fit proportional to the text areas
x2Obj = new JLabel( text: "x2");
x2Obj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
x3Obj = new JLabel( text: "x3");
x3Obj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
x4Obj = new JLabel( text: "x4");
x4Obj.setFont(new Font( name: "Arial", Font.PLAIN, size: 15));
x1Obj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0)); //creating spaces/borders between the elements
x2Obj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
x3Obj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
x4Obj.setBorder(new EmptyBorder( top: 0, left: 0, bottom: 5, right: 0));
plusObj = new JLabel( text: "+" );
plusObj.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));
JLabel plus1Obj = new JLabel( text: "+");
JLabel plus2Obj = new JLabel( text: "+");
plus1Obj.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));
plus2Obj.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 0, right: 10));

objectivePanel.add(objLabel); //adding all the created components to the objective panel
objectivePanel.add(x1ObjTA);
objectivePanel.add(x1Obj);
objectivePanel.add(plusObj);
objectivePanel.add(x2ObjTA);
objectivePanel.add(x2Obj);
objectivePanel.add(plus1Obj);
objectivePanel.add(x3ObjTA);
objectivePanel.add(x3Obj);
objectivePanel.add(plus2Obj);
objectivePanel.add(x4ObjTA);
objectivePanel.add(x4Obj);
inequalityPanel.add(objectivePanel);
}

inequalityPanel.add(simplexPanel); // add the panel to the frame and refresh the GUI
add(inequalityPanel, BorderLayout.CENTER);
revalidate();
repaint();

```

As a flow layout tends to put components directly next to each other, I had to manually create borders around the components, to ensure that the GUI did not end up too crowded and that I made use of the available space, fulfilling success criterion 1. These borders are shown by the 'setBorder()' method. The main issue with spacing arose in the horizontal direction, hence only the left and right aspects of the border have been adjusted, while the top and bottom aspects remain at 0.

I quickly ran a test of all 3 buttons to ensure they were being correctly displayed with the correct number of inequalities:

File Simplex Algorithm Kruskal's Algorithm Prim's Algorithm

Select number of inequalities **2** 2 Variables 3 Variables 4 Variables Run

$x + y =$   
 $x + y =$   
 $P = x + y$

File Simplex Algorithm Kruskal's Algorithm Prim's Algorithm

Select number of inequalities **2** 2 Variables 3 Variables 4 Variables Run

$x + y + z =$   
 $x + y + z =$   
 $P = x + y + z$

```

public void runSimplexAlg(int numInequality, int numVAR){
    int counter = ((numVAR + 1) * 2) + 1;
    double[][] constraints = new double[numInequality][numVAR + 1];
    for(int i = 0; i < numInequality; i++){
        JPanel parsePanel = (JPanel)inequalityPanel.getComponent(i);
        for(int j = 0; j < parsePanel.getComponentCount(); j++){
            Component component = parsePanel.getComponent(j);
            if(component instanceof JTextArea){
                constraints[i][counter] = Double.parseDouble(((JTextArea) parsePanel.getComponent(j)).getText());
            }
        }
    }
}

```

```

Exception in thread "AWT-EventQueue-0" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 7 out of bounds for length 3
at menuGUI.runSimplexAlg(menuGUI.java:516)
at menuGUI$event.actionPerformed(menuGUI.java:564) <4 internal lines>
at java.desktop/javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:279) <30 internal lines>

```

Since there are components within the panel that aren't text fields, it is possible that the value of counter may exceed the dimensions of the defined array when a text area is reached. For this reason, the error occurs. To fix this, I introduced a columnIndex variable, which, instead of inserting at the index [i][counter], allows us to maintain the indexing 0, 1, 2 etc despite the component index being greater.

```

public void runSimplexAlg(int numInequality, int numVAR){
    double[][] constraints = new double[numInequality][numVAR + 1]; //2d array of constraints
    for(int i = 0; i < numInequality; i++){ //iterating through each set of inequalities
        JPanel parsePanel = (JPanel)inequalityPanel.getComponent(i); //creates a new panel object to extract components from
        int columnIndex = 0; //introducing the columnIndex counter
        for(int j = 0; j < parsePanel.getComponentCount(); j++){ //iterating through the panel
            Component component = parsePanel.getComponent(j);
            if(component instanceof JTextArea){ //checks if the current component is a text area
                if(columnIndex < constraints[i].length) {
                    constraints[i][columnIndex] = Double.parseDouble(((JTextArea) parsePanel.getComponent(j)).getText()); //adds the value in the text area to the array
                    columnIndex++;
                }
            }
        }
    }
}

```

Having extracted all the values from the constraints, I now need to separately extract the values of the objective function, since they are passed separately into the simplex method. Since it is just one line of values, I created an arraylist for ease of adding elements

```

JPanel objectivePanel = (JPanel)inequalityPanel.getComponent(numInequality); //creates a new objective panel
ArrayList<Double> objective = new ArrayList<>(); //creating an arraylist to store the objective function
for(int k = 0; k < (numVAR * 2) + 2; k++){ //iterating through the panel
    if(objectivePanel.getComponent(k) instanceof JTextArea){ //checking if the current component is a text area
        objective.add((double)(Integer.parseInt(((JTextArea)objectivePanel.getComponent(k)).getText()))); //adding the value in the text area to the arraylist
    }
}
double[] obj = new double[numVAR];
for(int l = 0; l < objective.size(); l++){
    obj[l] = objective.get(l); //converting the arraylist into an array to be passed as a parameter
}

```

After all the values have been added to the arraylist, it is then essentially converted into a 1d array so it is in the correct form to be passed as a parameter into the simplex method.

Now that we have extracted all inputs from the components of the GUI and converted them into the correct form to be process, we can create a new object of type Simplex, and, using the methods from iteration 2, we can perform the algorithm on the user defined constraints.

```
Simplex problem = new Simplex(constraints, obj); //creating a new object of type Simplex
Iterate.solve(problem); //solving the created problem
```

I also made a slight change in the Solve class for the Simplex algorithm, to output the tableau after each iteration, instead of just the initial tableau. Before moving on to the next stage of development, I am going to test that the GUI works correctly, can take inputs, pass them to the Simplex method and correctly perform the algorithm.

I will try the following example from the textbook, with the tableaux and answer shown below:

$$\text{Maximise } P = 3x + 2y$$

subject to:

$$5x + 7y + r = 70$$

$$10x + 3y + s = 60$$

$$x, y, r, s \geq 0$$

Basic variable	x	y	r	s	Value	
r	5	7	1	0	70	•
s	10	3	0	1	60	•
P	-3	-2	0	0	0	•

Basic variable	x	y	r	s	Value
r	0	$\frac{11}{2}$	1	$-\frac{1}{2}$	40
x	1	$\frac{3}{10}$	0	$\frac{1}{10}$	6
P	0	$-\frac{11}{10}$	0	$\frac{3}{10}$	18

Basic variable	x	y	r	s	Value
y	0	1	$\frac{2}{11}$	$-\frac{1}{11}$	$\frac{80}{11}$
x	1	0	$-\frac{3}{55}$	$\frac{7}{55}$	$\frac{42}{11}$
P	0	0	$\frac{1}{5}$	$\frac{1}{5}$	26

$$P = 26, x = \frac{42}{11}, y = \frac{80}{11}, r = 0, \text{ and } s = 0$$

File Simplex Algorithm Kruskal's Algorithm Prim's Algorithm

Select number of inequalities **2** 2 Variables 3 Variables 4 Variables Run

5  x +  y =  70

10  x +  y =  60

P =  x +  y

The following output was produced in the terminal:

Initial Tableau:

5.00	7.00	1.00	0.00	70.00
10.00	3.00	0.00	1.00	60.00
-3.00	-2.00	0.00	0.00	0.00
0.00	5.50	1.00	-0.50	40.00
1.00	0.30	0.00	0.10	6.00
0.00	-1.10	0.00	0.30	18.00
0.00	1.00	0.18	-0.09	7.27
1.00	0.00	-0.05	0.13	3.82
0.00	0.00	0.20	0.20	26.00

The optimal solution is:

```
x1 = 3.818182
x2 = 7.272727
s1 = 0.0
s2 = 0.0
Optimal value: 26.0
```

This matches the mark scheme, deeming the newly developed methods successful in extracting inputs and running the algorithm.

The next stage of development is going to involve outputting the tableau and result in the GUI instead of the terminal.

Tableaux are currently stored in a 2d array of doubles, so I decided that to display this on the GUI, I would use a grid layout panel, which would have its dimensions defined as the same dimensions as the tableau. This would allow me to add each item in the 2d array 1 by 1, give it a border and label, giving the effect of a tableau.

```

    public static JPanel displayTableau(int rows, int cols){
        JPanel tableauPanel = new JPanel(new GridLayout(rows, cols));
        for(double[] row: table){
            for(double value: row){
                JLabel addLabel = new JLabel(Double.toString( d: Math.round(value * 100000000) / 100000000));
                Border blackline = BorderFactory.createLineBorder(Color.black);
                addLabel.setBorder(blackline);
                addLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 25));
                tableauPanel.add(addLabel);
            }
        }
        JPanel wrapperPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        wrapperPanel.add(tableauPanel);
        return wrapperPanel;
    }
}

```

Calling this method in menuGUI correctly outputted a tableau, however, this was not the initial tableau.

0.00	1.00	0.18	-0.09	7.27
1.00	0.00	-0.05	0.13	3.82
0.00	0.00	0.20	0.20	26.00

Since the methods for the simplex algorithm are being performed on the same 2d array, the displayTableau method was only outputting the final state of the tableau. To fix this, I had to find a way to store a deep copy of the table after each iteration. This led to the creation of the makeTable method.

```

public static double[][][] makeTable(Tableau copyTableau) {
    double[][][] copyTable = new double[rows][cols]; //creates a new 2d array to store the deep copy of the table
    double[][][] existingTable = copyTableau.getTable(); //assigns the parameter to a 2d array
    for (int i = 0; i < existingTable.length; i++) {
        for (int j = 0; j < existingTable[i].length; j++) {
            copyTable[i][j] = existingTable[i][j]; //copies the array
        }
    }
    return copyTable;
}

```

This takes a Tableau object as a parameter and copies it, allowing the current state of the Tableau to be stored in a list. The makeTable method is called from the Iterate class, which contains an arraylist to store the copied tableaux.

```

public class Iterate {
    4 usages
    public static ArrayList<double[][]> tableaux = new ArrayList<>();
    6 usages
    public static ArrayList<String[]> rowHeadersList = new ArrayList<>();
    1 usage
    public static void solve(Simplex problem){
        tableaux.clear(); // Clear the previous tableaux
        rowHeadersList.clear();
        Tableau table = new Tableau(problem); //creating a table object taking a parameter of a simplex problem object
        tableaux.add(Tableau.makeTable(table));
        rowHeadersList.add(menuGUI.getRowHeaders());
    }

    public static void solve(Simplex problem){
        tableaux.clear(); // Clear the previous tableaux
        Tableau table = new Tableau(problem); //creating a table object taking a parameter of a simplex problem obj
        tableaux.add(Tableau.makeTable(table)); //adding initial tableau to list of tableaux
        while (!table.isOptimal()) {
            int pivotCol = Tableau.getPivotCol(); //getting pivot column and pivot row to perform a pivot
            int pivotRow = Tableau.getPivotRow(pivotCol);
            table.pivot(pivotRow, pivotCol);
            tableaux.add(Tableau.makeTable(table));
        }
    }
}

```

The first call of makeTable is to store the initial tableau, and the second call takes place inside the while loop to store each subsequent state of the table after the iterations.

I next added the row and column headers to the tableau, since the terminal output above isn't the full tableau that would be given in an Edexcel A-Level Decision Maths exam and would require the student to add row and column headers to the table. The first task was to define the row and column headers for each size of tableau. Since the row headers are all the basic variables (in this case they are initially the slack variables) and the objective row P, and the column headers are all the variables (non-basic and basic variables e.g. x, y, z and r, s, t), they are dependent on the number of inequalities. For example, a question with 2 inequalities will only require r and s as slack variables, therefore will have r, s and P as row headers, whereas a question with 4 inequalities will require r, s, t and u as slack variables, having r, s, t, u and P as row headers.

```

counter = Integer.parseInt(numInequality.getSelectedItem().toString()); //initialises a counter variable
if (button == twoVar && (numInequality.getSelectedItem().toString() != "--")){ //checks which button pressed and ensures a value was selected from the drop-down
    numVAR = 2;
    rowHeaders = new String[counter + 1]; //creating row headers array
    for(int a = 0; a < counter; a++){
        rowHeaders[a] = Character.toString( codePoint: a+114); //adding slack variable letters starting from r, depending on number of inequalities (counter)
    }
    rowHeaders[counter] = "P"; //setting final index as P, this is always the row header of the bottom row
    colHeaders = new String[counter + 3]; //creating list of column headers
    colHeaders[0] = "x"; //for 2 variables, the first 2 column headers are the 2 variables, x and y.
    colHeaders[1] = "y";
    for(int a = 0; a < counter; a++){ //adding column headers of slack variables in a similar way to row headers using ASCII
        colHeaders[a+2] = Character.toString( codePoint: a+114);
    }
    colHeaders[counter + 2] = "Value"; //the row header of the final column is always Value
}

```

At this point, I could have manually added the slack variables into both the row header and column header array, but I instead used a for-loop in which the counter variable was converted to an ASCII value (adding 114 to start at 'r'), and adding each subsequent letter as required.

I repeated a similar process of creating row headers for 3 variables and 4 variables, shown below, but this time, first manually adding the required x,y,z or x1,x2,x3 and x4.

```

} else if (button == threeVar && (numInequality.getSelectedItem().toString() != "--")) { //checking input button
    numVAR = 3;
    rowHeaders = new String[counter + 1]; //creating list of row headers
    for(int a = 0; a < counter; a++){ //adding each slack variable beginning from r depending on number of inequalities
        rowHeaders[a] = Character.toString( codePoint: a+114);
    }
    rowHeaders[counter] = "P"; //setting final index as P, this is always the row header of the bottom row
    colHeaders = new String[counter + 4]; //creating list of column headers
    colHeaders[0] = "x"; //adding first few items to column headers'
    colHeaders[1] = "y";
    colHeaders[2] = "z";
    for(int a = 0; a < counter; a++){ //adding the slack variables to the column headers in a similar manner to above
        colHeaders[a+3] = Character.toString( codePoint: a+114);
    }
    colHeaders[counter + 3] = "Value"; //adding the final column header as 'Value', which is always the same
}
else if (button == fourVar && (numInequality.getSelectedItem().toString() != "--")) {
    numVAR = 4;
    rowHeaders = new String[counter + 1]; //creating row header array
    for(int a = 0; a < counter; a++){
        rowHeaders[a] = Character.toString( codePoint: a+114); //adding slack variables starting from r, using ASCII
    }
    rowHeaders[counter] = "P"; //adding P to the final index of the row headers, this is always the same
    colHeaders = new String[counter + 5];//creating column headers array
    colHeaders[0] = "x1"; //adding 4 variables manually to the start of column headers
    colHeaders[1] = "x2";
    colHeaders[2] = "x3";
    colHeaders[3] = "x4";
    for(int a = 0; a < counter; a++){
        colHeaders[a+4] = Character.toString( codePoint: a+114); //adding slack variables starting from r, similar to above using ASCII
    }
    colHeaders[counter + 4] = "Value"; //adding 'Value' in the final column, which is the same for all tableaux
}

```

An important factor to note here, seen in the if statement is requiring there to be an input into the drop-down for the number of inequalities. While this will be fixed in a more robust manner later, for the purpose of testing throughout

development, I added in the condition to check that a number has been selected from the drop-down.

At this point, I realised that the displayTableau method was going to get overcrowded with multiple panels and loops taking place at the same time, so I branched off by making another method called createTableauPanel. The sole purpose of this method is to create just the tableau, with row and column headers. This can then be called in the displayTableau method while also being able to add in any theta values and row operations as necessary. I created a get method in the Iterate class that can be used to retrieve the list of tableaux added to it throughout the iterations.

```
public static ArrayList<double[][]> getTableaux(){  
    return tableaux;  
}
```

In the displayTableau method, I then added a line which created a new arraylist of 2d arrays, which called the getTableaux method to get all the tableaux into one array in the menuGUI class.

```
public static JPanel displayTableau(String[] rowHeaders, String[] colHeaders, int rows, int cols) {  
    ArrayList<double[][]> toOutput = Iterate.getTableaux(); // creates a new arraylist of 2d arrays which gets all the tableaux from the iterations of the algorithm
```

In the displayTableau method, I created a for loop that iterates through each table in the toOutput list:

```
for (double[][] outputTable : toOutput) {
```

```
private static JPanel createTableauPanel(double[][] outputTable, String[] rowHeaders, String[] colHeaders, int rows, int cols) {  
    JPanel tableauPanel = new JPanel(new GridLayout(rows + 1, cols + 1)); //new panel to add the tableau values, using grid layout to create effect of a table  
    JLabel bvLabel = new JLabel("B.V."); //sets the basic variable header in the top left corner  
    bvLabel.setFont(new Font("Arial", Font.BOLD, 15));  
    bvLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));  
    tableauPanel.add(bvLabel);  
    JLabel valueLabel = null;
```

Since the tableau will now need an extra row and column for the row headers and column headers, the size of the grid layout adds 1 to each of the dimensions. This method takes the current table from the for loop, the row headers, column headers and number of rows and columns as parameters. I created a label called bvLabel, which will store the text 'B.V' and be added to the top left corner of the tableau, indicating the basic variable column. I created a new label, which will take

the value of each number in the table to be outputted, make it into a label and add it to the grid.

```

for (int i = 0; i < outputTable.length; i++) {
    for (int j = 0; j < outputTable[i].length; j++) {
        if(outputTable[i][j] % 1 == 0){
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j]), SwingConstants.CENTER)); //adding the value from the output table one by one to the tableau panel
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of the label
            valueLabel.setFont(new Font("Arial", Font.PLAIN, size 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        } else {
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j]), SwingConstants.CENTER)); //adding the value from the output table one by one to the tableau panel
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of the label
            valueLabel.setFont(new Font("Arial", Font.PLAIN, size 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        }
    }
}

```

The if-statement checks whether the value in the table is an integer (being stored as a double), if so, then it removes the '.0' that is added by default to the end of a double.

To add the row headers, I knew there was going to be a similar error as adding the tableau, since only the final state of the row headers array would be added to each of the tableau. I implemented a similar way around this, creating a row header array in the Iterate class, which has lists of row headers added to it. Each list of row headers has the necessary change made to the list, based on the pivot row and column.

```

public static ArrayList<double[][]> tableaux = new ArrayList<>();
6 usages
public static ArrayList<String[]> rowHeadersList = new ArrayList<>();
1 usage
public static void solve(Simplex problem){
    tableaux.clear(); // Clear the previous tableaux
    rowHeadersList.clear();
    Tableau table = new Tableau(problem); //creating a table object taking a parameter of a simplex problem object
    tableaux.add(Tableau.makeTable(table)); //adding initial tableau to list of tableaux
    rowHeadersList.add(menuGUI.getRowHeaders()); //adding initial row headers to list of row headers
    while (!table.isOptimal()) {
        int pivotCol = Tableau.getPivotCol(); //getting pivot column and pivot row to perform a pivot
        int pivotRow = Tableau.getPivotRow(pivotCol);
        table.pivot(pivotRow, pivotCol);
        String[] newHeader = rowHeadersList.get(rowHeadersList.size() - 1).clone(); //creating new copy of row headers
        newHeader[pivotRow] = menuGUI.getColHeaders()[pivotCol]; //changing row headers of pivot row
        rowHeadersList.add(newHeader);
        tableaux.add(Tableau.makeTable(table));
    }
}

```

Outside of the while loop, the initial row headers are added, before changing them inside the while loop and adding the string list newHeader to the list. The column headers stay the same throughout the iterations, hence no such method is needed

for them. Since components in a grid layout are added one after another, I need to consider the correct order in which the row headers and column headers will be added. The first added item will be the 'B.V' label in the top left corner. The top row will contain all the column headers; hence these will be the next items added. After this, I need to add the first row header, followed by the first row of values in the table, followed by the second row header, and so on.

```

for (String header : colHeaders) {
    JLabel colHeaderLabel = new JLabel(header, SwingConstants.CENTER); //adds each column header to the top row of the panel
    colHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting font, size, border and alignment of label
    colHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER);
    colHeaderLabel.setFont(new Font("Arial", Font.BOLD, size 15));
    colHeaderLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    tableauPanel.add(colHeaderLabel); //adding the column header label to the panel
}

for (int i = 0; i < outputTable.length; i++) {
    JLabel rowHeaderLabel = new JLabel(rowHeaders[i], SwingConstants.CENTER); //adding the row headers in the first column of the table underneath the basic variable header
    rowHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of label
    rowHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER);
    rowHeaderLabel.setFont(new Font("Arial", Font.BOLD, size 15));
    rowHeaderLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    tableauPanel.add(rowHeaderLabel); //adding the row header label to the panel

    for (int j = 0; j < outputTable.length; j++) {
        if(outputTable[i][j] % 1 == 0){
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j])), SwingConstants.CENTER); //adding the value from the output table one by one to the tableau
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of the label
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            valueLabel.setFont(new Font("Arial", Font.PLAIN, size 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        }
        else {
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j])), SwingConstants.CENTER); //adding the value from the output table one by one to the tableau
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of the label
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            valueLabel.setFont(new Font("Arial", Font.PLAIN, size 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        }
    }
}

```

When running the above code, the program successfully ran, but the output table had labels in the incorrect place, shown below:

B.V.	x	y	r	s	θ
Value	r	5.00	7.00	1.00	14
s	10.00	3.00	0.00	P	6
-3.00	-2.00	0.00			--

Looking through the code, I noticed an error with one of the loop counters. The loop that iterates through the values and assigns the numbers from the tableau to the label should loop between 0 and the number of items in a row, however, looping from 0 to outputTable.length will loop for the number of rows instead. To fix this issue, I changed to loop counter to loop from 0 to outputTable[i].length, which will loop through the items in a row.

```

for (int i = 0; i < outputTable.length; i++) {
    JLabel rowHeaderLabel = new JLabel(rowHeaders[i], SwingConstants.CENTER); //adding the row headers in the first column of the table underneath the basic variable header
    rowHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of label
    rowHeaderLabel.setFont(new Font("Arial", Font.BOLD, 15));
    rowHeaderLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    tableauPanel.add(rowHeaderLabel); //adding the row header label to the panel

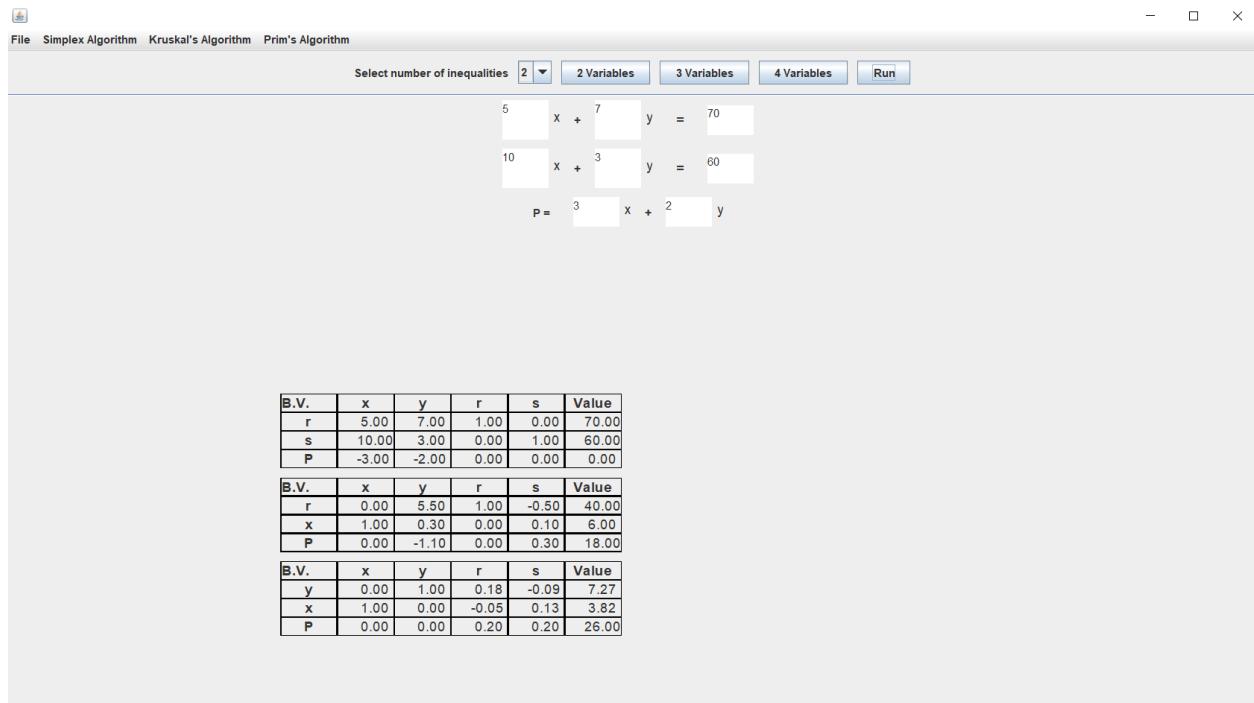
    for (int j = 0; j < outputTable[i].length; j++) {
        if(outputTable[i][j] % 1 == 0){
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j])), SwingConstants.CENTER); //adding the value from the output table one by one to the tableau
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            valueLabel.setFont(new Font("Arial", Font.PLAIN, 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        } else {
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j])), SwingConstants.CENTER); //adding the value from the output table one by one to the tableau
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            valueLabel.setFont(new Font("Arial", Font.PLAIN, 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        }
    }
}

```

Inside the for loop in the displayTableau method, I ran the following line to check whether the correct tableaux were now being displayed with the necessary changes made to each one.

```
JPanel tableauPanel = createTableauPanel(outputTable, rowHeaderArray.get(tableauCount), colHeaders, rows, cols); //creates a formatted tableau based on the current
```

Before implementing the tabbed pane, I decided to test whether the correct tableaux were being outputted with the necessary changes to the values and the row headers.



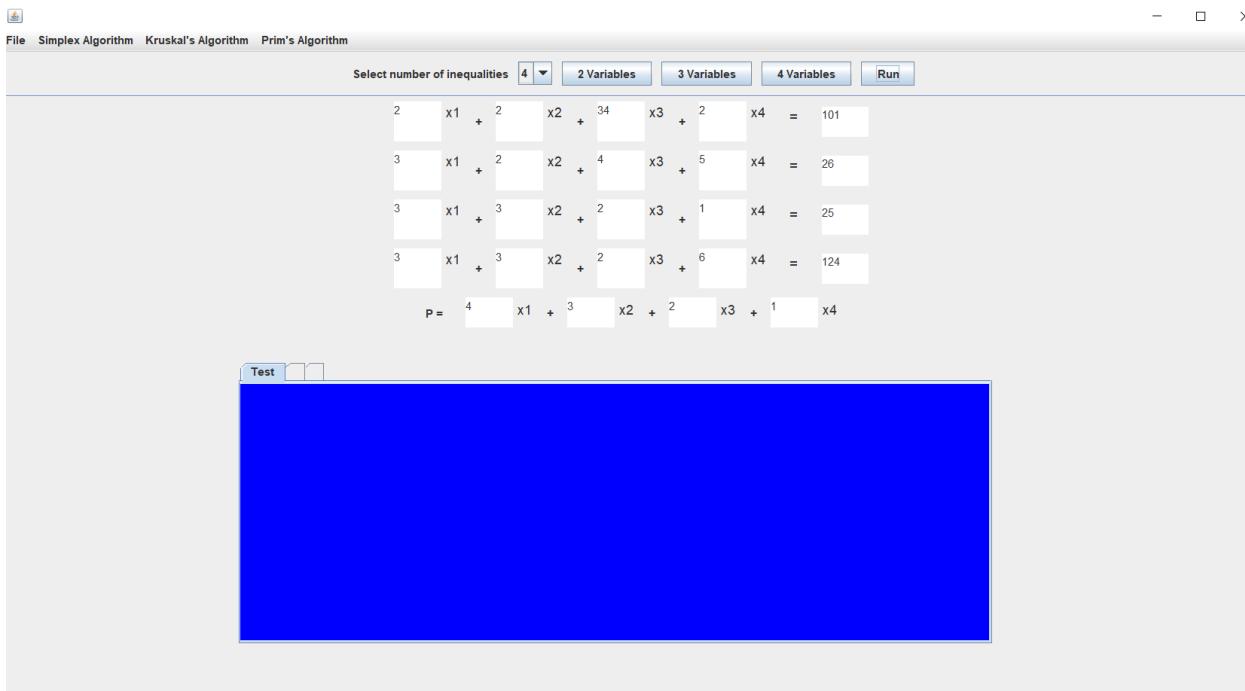
The correct tableaux are being outputted, when comparing against the correct

answer, hence I can move on to implementing the tabbed pane with row headers and column headers.

In the displayTableau method, I wrote the following lines to create the tabbed pane and the panel which contains all the components created in this method. This panel is returned by the method to menuGUI, which can then add it to the GUI in the runSimplexAlg method.

```
JTabbedPane tabbedPane = new JTabbedPane(); //creating the tabbed pane to display the tableaux
tabbedPane.setPreferredSize(new Dimension( width: 1000, height: 300)); //dimensions to ensure no overlap between the tabbed pane and the inequality panel
JPanel testPanel = new JPanel();
testPanel.setBackground(Color.BLUE);
tabbedPane.add(testPanel, constraints: "Test");
JPanel fullPanel = new JPanel(); //creating the panel that will store the tabbed pane and will be the main panel added to the gui
fullPanel.setLayout(new FlowLayout());
```

As mentioned in the design, I had to ensure that the tabbed pane and the inequality boxes wouldn't overlap. To do so, I altered the dimensions of the tabbed pane, then checked with the maximum number of inequality boxes, which had 4 rows, to ensure there was no overlap. I added a blue panel to test that there was sufficient room between the rows and the tabbed pane. As shown below, the proportions of the tabbed pane are correct, with enough room above and to the sides.



I now had to correctly add the components inside the panel in the correct place, ensuring that they take up the correct amount of space. The current panel layouts

that I have tried (box layout, grid layout, flow layout) don't offer much versatility regarding the sizing of components, hence for the panel inside the tabbed pane, I opted for a grid bag layout. This allows components to be added in a similar manner to grid layout, but instead of fixed dimension, it allows for specification on how much room horizontally and vertically each component should take. Since each tab in the pane needs to have a tableau, row operations, theta values, and an explanation, I will create individual panels inside the for loop, which iterates through each table in the array created earlier.

I first created the panels for theta values and for the row operations, giving each of them a JLabel title to indicate the purpose of the column. In order to get the theta values for a specific tableau, I developed a method called `getThetaValues`, which clears the list of theta values, before getting the current pivot column, then dividing the numbers in the Value column by their respective values in the pivot column, and adding the results to an array.

```
public static ArrayList<Double> getThetaValues(double[][] tableau) {
    thetaValues.clear();
    int pivotC = getCurrentPivotCol(tableau); //get the pivot column index

    for (int i = 0; i < tableau.length; i++) {
        if (tableau[i][pivotC] > 0) { //ensure we are not dividing by zero
            double first = tableau[i][cols - 1];
            double second = tableau[i][pivotC];
            System.out.println(first);
            System.out.println(second);
            System.out.println(" ");
            double theta = tableau[i][cols - 1] / tableau[i][pivotC]; //the value column is at cols - 1
            thetaValues.add(theta); //add the calculated theta value to the list
        } else {
            thetaValues.add(Double.POSITIVE_INFINITY); //handles cases where division by zero would occur
        }
    }
    return thetaValues;
}
```

Since the previous methods to get the pivot column and pivot rows would work on the single tableau that was being iterated on, I had to develop new methods, to get the pivot column and pivot row for a specific tableau passed in as a parameter. These methods were used in the above method to calculate theta values

```

public static int getCurrentPivotCol(double[][] tableau) {
    int pivotCol = 0;
    double minValue = tableau[rows - 1][0]; //sets the minimum value as the bottom left value of the table
    for (int i = 0; i < cols - 1; i++) { //iterates through objective row checking if next value smaller (more negative) than current minValue
        if (tableau[rows - 1][i] < minValue) {
            minValue = tableau[rows - 1][i]; // if true, sets minValue as this new value
            pivotCol = i; //sets index of pivot column
        }
    }
    return pivotCol;
}

```

```

public static int getCurrentPivotRow(int pivotCol, double[][] tableau) {
    int pivotRow = 0;
    double minRatio = Double.MAX_VALUE; //sets value of new variable equal to the max value offered by java
    for (int i = 0; i < rows - 1; i++) {
        double ratio = tableau[i][cols - 1] / tableau[i][pivotCol]; //divides value column by pivot column
        if (ratio > 0 && ratio < minRatio) { // if result > 0 and < current minRatio then sets minRatio to current value
            minRatio = ratio;
            pivotRow = i; //sets index of pivot row
        }
    }
    pivotVal = tableau[pivotRow][pivotCol];
    return pivotRow;
}

```

These methods work in a similar way to the existing methods for pivot row and column; however, they take in a specific tableau as an additional parameter and calculate based on the values in that tableau instead.

```

for (double[][] outputTable : toOutput) {
    ArrayList<Double> thetaList = getThetaValues(outputTable); //getting the theta values for each tableau in the toOutput list
    JPanel thetaPanel = new JPanel(); //creating a new panel to store the theta values and setting the layout as grid layout with 1 column to give the table format
    thetaPanel.setLayout(new GridLayout( rows: 0, cols: 1));thetaPanel.setPreferredSize(new Dimension( width: 5, thetaPanel.getPreferredSize().height));
    JLabel theta = new JLabel( text: "θ"); //label which indicates the theta column
    theta.setFont(new Font( name: "Arial", Font.BOLD, size: 16));
    theta.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //sets the font, border and alignment of the theta label
    theta.setHorizontalTextPosition(SwingConstants.CENTER);
    theta.setHorizontalAlignment(SwingConstants.CENTER);
    thetaPanel.add(theta); //adds the label to the panel
    JPanel rowOpPanel= new JPanel(); //creates a new panel to store the row operations
    rowOpPanel.setLayout(new GridLayout( rows: 0, cols: 1)); //setting the layout of the panel in a similar way to the theta panel
    rowOpPanel.setPreferredSize(new Dimension( width: 5, thetaPanel.getPreferredSize().height)); //fixing the size of the panel
    JLabel rowOp = new JLabel( text: "Row Ops."); //setting the text to be added to the top of the rowOpPanel
    rowOp.setHorizontalAlignment(SwingConstants.CENTER); //setting the font, border and alignment of the text label
    rowOp.setHorizontalTextPosition(SwingConstants.CENTER);
    rowOp.setFont(new Font( name: "Arial", Font.BOLD, size: 16));
    rowOp.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    rowOpPanel.add(rowOp); //adds label to the panel
}

```

Next, I created another for loop that iterates through the list of theta values, adding them to the panel just created. Another condition I checked is if the theta value stores infinity, then it is replaced with the text '--' as it need not be considered when calculating the pivot row. I also checked if the theta value is an integer, using the mod function, if so then it removes the '.0' automatically added to integers when they are stored as doubles. Since some values are integers and some are doubles, they are all stored as doubles. To ensure that each label is clearly visible, I adjusted the font and sizing of the text in each label and made the

text bold as well. This is also seen above when adding the row operation and theta label.

```

for(Double value: thetaList){
    if (value == Double.POSITIVE_INFINITY){ //checks if theta value is infinity (dividing by 0)
        JLabel thetaLabel = new JLabel(" - "); //sets it to -- as it need not be considered for the pivot row
        thetaLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //sets the font, border and alignment of the theta label
        thetaLabel.setHorizontalTextPosition(SwingConstants.CENTER);
        thetaLabel.setHorizontalAlignment(SwingConstants.CENTER);
        thetaLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        thetaPanel.add(thetaLabel); //adding to thetaPanel
    } else {
        if(value % 1 == 0){ //checks if the value is an integer
            String integerConversion = (Double.toString(value)).replace(".", ""); //converting value to a string and removing the .
            JLabel thetaLabel = new JLabel(integerConversion); //setting the theta label as the string created above
            thetaLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //setting the font, border and alignment of the label
            thetaLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            thetaLabel.setHorizontalAlignment(SwingConstants.CENTER);
            thetaLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            thetaPanel.add(thetaLabel); //adding label to thetaPanel
        } else {
            JLabel thetaLabel = new JLabel(String.format("%.2f",value)); //rounds the value to 2dp and adds it
            thetaLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font, border and alignment of label
            thetaLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            thetaLabel.setHorizontalAlignment(SwingConstants.CENTER);
            thetaLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            thetaPanel.add(thetaLabel); //adding label to thetaPanel
        }
    }
}

```

As mentioned in design, the initial tableau has no row operations, since there was no previous table to perform row operations on to reach this point, hence I introduced a counter variable that starts at zero and increments each time a table from the list is added. This value is then checked to ensure that the number of tableaux currently outputted is at least 1, in which case row operations can be added to the table. I once again adjusted the font and sizing of the row operation text, and made the text bold.

```

if(tableauCount >= 1){ // checking to make sure not iterating on the first tableau
    ArrayList<String> rowOpList = rowOperationList.get(tableauCount-1); //if any tableau that isn't the initial tableau, then add row operations
    for(int b = 0; b < rowOpList.size(); b++) {
        System.out.println(rowOpList.get(b));
        JLabel rowOpLabel = new JLabel(rowOpList.get(b)); //sets a label with the text of the row operation from rowOpList
        rowOpLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //sets font, border and alignment of the label
        rowOpLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        rowOpLabel.setHorizontalTextPosition(SwingConstants.CENTER);
        rowOpLabel.setHorizontalAlignment(SwingConstants.CENTER);
        rowOpPanel.add(rowOpLabel); //adds the label to the rowOpPanel
    }
}

```

Having now created each individual panel, I can now add them all to one panel with a grid bag layout. I created the new panel, as well as an object of type GridBagConstraints, which is what allows me to specify the sizing and positioning of each component in the panel.

I want the tableau to be in the top left, taking up 75 percent of the width to ensure that the tableau is clearly visible, leaving the other space in the width for the theta values and row operations. These constraints were specified using the GridBagConstraints object:

```
JPanel gridPanel = new JPanel(new GridBagLayout()); //new panel with gridbag layout for better control over spacing of components
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridx = 0; //column 0
gbc.gridy = 0; //row 0
gbc.gridwidth = 1; //span 1 column
gbc.weightx = 0.5; //take up 75% of the panel's width
gbc.fill = GridBagConstraints.BOTH; //expand both horizontally and vertically
```

I specified the column and row of the grid bag layout that the tableau should be added to, with 0 and 0 indicating the top left. I then specified the width and the weight of the width that this should take up.

I then called the createTableauPanel method, which added the returned panel to the tableau panel I created at the start of the displayTableau method. This was then added to the gridPanel with the constraints defined above.

```
JPanel tableauPanel = createTableauPanel(outputTable, rowHeaderArray.get(tableauCount), colHeaders, rows, cols); //creates a formatted tableau based on the current table
gridPanel.add(tableauPanel, gbc);
```

Having added the tableau panel with the constraints, i can use the same GridBagConstraints object and just change the sizing and positioning, before adding the next object. The next panel to be added it the row operations, horizontally adjacent to the tableau. This should only be added if we are not on the initial tableau, so I created an if statement that checks for this condition, before altering the sizing and positioning of the panel and adding it.

```
if(tableauCount >= 1) { //checks if not on initial tableau before adding row operations to the gui
    gbc.gridx = 1; //adds to the right of the tableau
    gbc.gridy = 0;
    gbc.weightx = 0.3;
    gridPanel.add(rowOpPanel, gbc); //adds the row operation panel to the grid panel
}
```

I specified the gridx as 1, which adds the panel in the next column, so the row operation panel is added directly to the right of the tableau panel.

Next, every table needs the theta values, which will be added horizontally adjacent to the row operations (or the tableau since there are no row operations for the

initial tableau). I specified the gridx as 2, to add it to the right of the existing components in the panel.

```
gbc.gridx = 2; //adds the theta panel to the gui to the right of the row operations  
gbc.gridy = 0;  
gbc.weightx = 0.1;  
gridPanel.add(thetaPanel, gbc); //adds the theta panel to the grid panel
```

The bottom half of the grid panel has been currently left blank for explanation, so I will now develop the method to explain a tableau. I split this up into 3 possible tableaux that would need different explanations, initial, intermediate (in the middle) and the final tableau. As mentioned in the design, each of these tableaux needs slightly different explanations e.g. the initial tableau explanation shouldn't be referencing the previous tableau, as there is no previous tableau, the final tableau shouldn't have any row operations being explained, as the problem is complete. I also decided that for a more natural explanation of the tableaux, I would provide multiple options for the explanation text, which will be randomly selected from when the method is called. In the design of this iteration, students mentioned the key parts they would want to have explained, which was how to calculate the pivot row and column, and how to calculate the row operations, forming the main subject of the explanation.

```
JPanel addPanel = new JPanel(); //new panel to contain explanation  
addPanel.setLayout(new BoxLayout(addPanel, BoxLayout.Y_AXIS)); //set box layout for this panel  
ArrayList<JTextPane> initialResponses = new ArrayList<>(); //list of possible explanation responses for initial tableau  
ArrayList<JTextPane> intermediateResponses = new ArrayList<>(); //list of possible explanation responses for intermediate tableau
```

I created a new panel which would be returned by the method, called addPanel. This takes a box layout, as my initial idea was to have a series of labels outputted in a list (using box layout) to explain each tableau, however, I decided to use the JTextPane components, as they offer better control over text wrapping and restricting the sizing of the component, whereas a JLabel may be cut off by the panel. Since the box layout will have no impact on the text pane (since there is only one component in the panel), I decided to keep the same layout. I created 2 lists, one to store the explanations for the initial tableau and one to store the explanations of the intermediate tableaux. I decided that since the final tableau has little explanation (just explaining why the problem is complete and reading off the values), once explanation would be sufficient, whereas for the more in-depth explanations, I will offer a variety for more natural-looking explanation. Since the

questions from the Edexcel A-Level further maths specification generally asks students to perform a maximum of around 2 or 3 iterations, I deemed that having 2 of the initial and intermediate explanation options sufficient, as it is unlikely that with few tableaux it will look very repetitive. To ensure that every problem doesn't have the exact same explanation text every time, I added in a slight variance in the explanations. I then created the first explanation option, which uses value from the specific tableau to ensure the user can refer to the table when reading through the explanation. I also added line breaks throughout the explanation, to avoid the text becoming too much of a bulky paragraph, which would be difficult to follow.

```
JTextPane option1 = new JTextPane(); //explaining the tableau, using line breaks for better flow of reading
option1.setText("looking through the objective row, the pivot column was selected as " + " " + getColHeaders()[getCurrentPivotCol(tableau)] + " " + ", highlighted in blue");
option1.setText(option1.getText() + " This gives us the pivot value as " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] +
option1.setText(option1.getText() + "\nOur aim is to reduce the pivot value to 1 and everything else in the pivot column to 0, so our first row operation is [the values in
option1.setText(option1.getText() + " We now look at each value in the pivot column by turn. We must see how we can reduce the value to 0 using the pivot value, which is no
option1.setText(option1.getText() + "\nWe take a value from the pivot column that is not the pivot value, we then subtract the value multiplied by the pivot row, so if the
option1.setText(option1.getText() + "\nWe repeat the process of applying row operations until we have completed the tableau.");
option1.setFont(new Font("Arial", Font.PLAIN, size: 15)); //setting font size and style of the explanation text
option1.setEditable(false); //ensure the user cannot edit the explanation
=
```

---

```
highlighted in blue, as this has the most negative value in the objective row." + " By dividing each number in the Value column by the respective value in the pivot column
pivotCol(tableau] + " .");
n is [the values in the pivot row] / " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + ".");
t value, which is now 1.');
pivot row, so if the value is 5, we take 5 * 1 away from it, giving us 0. This process is repeated throughout the row, using the respective values in the pivot row. We then
=
```

---

```
and selecting the smallest positive result, the pivot row was selected as " + " " + getRowHeaders()[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)] + " .");
repeat which each non-pivot value in the pivot column.");
```

I then created a second explanation, slightly rewording but ensuring that the same level of information is conveyed to the user.

```

JTextPane option2 = new JTextPane();
option2.setText("We look through the objective row for the most negative value. This gives us the pivot column as " + " " + getColHeaders()[getCurrentPivotCol(tableau)];
option2.setText(option2.getText() + " We now select the smallest, positive theta value, and the row in which this is situated is our pivot row, highlighted in orange.");
option2.setText(option2.getText() + "\nOur aim is to reduce the pivot value to 1 and everything else in the pivot column to 0, which gives us our first row operation");
option2.setText(option2.getText() + " We look at each value in the pivot column. We must see how we can reduce the value to 0 using the pivot value, which has now been multiplied by the pivot row. This process is a division operation.");
option2.setFont(new Font("Arial", Font.PLAIN, size: 15)); //setting font size and style of the explanation text
option2.setEditable(false); //ensure the user cannot edit the explanation

" + ", highlighted in blue. To find the theta values, we divide each number in the value column by its respective value in the pivot column.");
intersection between the pivot column and pivot row is our pivot value, " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + " values in the pivot row / our pivot value, " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + ".");
set to 1.");
across the row, using the respective values in the pivot row, and is repeated for each non-pivot value in the pivot column.");
```

```

pivot column.");
tableau)][getCurrentPivotCol(tableau)] + " ." );
tCol(tableau)] + ".");
```

Next, I developed the explanation of the intermediate tableau. These had to be worded slightly differently, ensuring they explain how to reach the current tableau from the previous one, and how the row operations were performed on the tableau.

```

JTextPane midOption1 = new JTextPane();
midOption1.setText("Applying the row operations to the previous tableau gives us this tableau, with those row operations displayed next to it. We repeat the process from the previous section to reach our current tableau.");
midOption1.setText(midOption1.getText() + "\nBy dividing each value in the Value column by its respective value in the pivot column, we get our theta values, shown right next to the tableau.");
midOption1.setText(midOption1.getText() + " We want to reduce the pivot value to 1 and everything else in the pivot column to 0. The first row operation we perform is a division operation.");
midOption1.setText(midOption1.getText() + " As the pivot value is 1, we just subtract whatever the value in the pivot column is * 1 from the value in the pivot column to reach 0.");
midOption1.setText(midOption1.getText() + "\n We repeat this process for each row, giving us our new tableau.");
midOption1.setFont(new Font("Arial", Font.PLAIN, size: 15)); //setting font size and style of the explanation text
midOption1.setEditable(false); //ensure the user cannot edit the explanation
```

```
it. We repeat the process from the previous tableau. Looking through the objective row, the pivot column is " + " + getColHeaders()[getCurrentPivotCol(tableau)] our theta values, shown right of the tableau. We select the smallest, positive theta value to give us our pivot row. The intersection between pivot row and pi row operation we perform is on the pivot row and involves dividing each value in the pivot row by the pivot value to reduce our pivot value to 1. We now pick t ve value in the pivot column to reduce it to 0. We must perform the same row operation on each value in the row.");
```

```
in blue. This was chosen as this column has the most negative value in the objective row.");  
ue, " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + "", highlighted in purple.");  
column that isn't in the pivot row, and reduce it to 0, using the values in the pivot row.");
```

To ensure that the code remains reasonably readable, I used the set text and get text methods of the text pane, which allowed me to develop the responses over multiple lines of code, instead of having the entire explanation in one line of code.

I now developed the second explanation for the intermediate tableau:

```
JTextPane midOption2 = new JTextPane();  
midOption2.setText("Performing the row operations on the previous tableau yields the following tableau, with the row operations shown alongside it. We repeat the st  
midOption2.setText(midOption2.getText() + " We then identify the smallest positive theta value, and the row containing this value is chosen as the pivot row, highli  
midOption2.setText(midOption2.getText() + " Our goal is to reduce the pivot value to 1 and all other entries in the pivot column to 0, resulting in our first row op  
midOption2.setText(midOption2.getText() + " Next, we examine each value in the pivot column and determine how to reduce it to 0 using the pivot value, which has alr  
midOption2.setText(midOption2.getText() + "\nFor each non-pivot value in the pivot column, we calculate the product of that value and the pivot row, then subtract i  
midOption2.setText(midOption2.getText() + "\nWe repeat this process for each row of the tableau, which will give us our tableau.");  
midOption2.setFont(new Font( name:"Arial", Font.PLAIN, size: 15)); //setting font size and style of the explanation text  
midOption2.setEditable(false); //ensure the user cannot edit the explanation
```

```
eps from the previous tableau. Examining the objective row, the pivot column is identified as " + " + getColHeaders()[getCurrentPivotCol(tableau)] + " " + ", highlighted  
lighted in orange. The intersection of the pivot column and pivot row marks the pivot value, " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentP  
peration as [the values in the pivot row] / our pivot value, " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + ".");  
ready been reduced to 1.");  
it from the corresponding entry. This is repeated for all entries in the pivot column to reduce them to 0.");
```

```
ied as " + " + getColHeaders()[getCurrentPivotCol(tableau)] + " " + ", highlighted in blue, as it corresponds to the most negative value in the objective row.");  
ue, " + tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + ".");  
tRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] + ".");  
reduce them to 0.");
```

Ensuring that each of these explanations focussed on the requested parts of the topic by the stakeholders, I moved onto the explanation of the final tableaux.

```
JTextPane finalOption1 = new JTextPane();  
ArrayList<String[]> rowHeaderArr = Iterate.getRowHeadersList();  
String[] finalRowHeaders = rowHeaderArr.get(a);  
finalOption1.setText("Looking through the objective row, there are no more negative values, meaning this tableau is final, so we can read off the final values of our  
for(in i = 0; i < finalRowHeaders.length; i ++){  
    finalOption1.setText(finalOption1.getText() + "\n");  
    finalOption1.setText(finalOption1.getText() + " " + finalRowHeaders[i] + " = " + String.format("%.2f", (tableau[i][tableau[i].length - 1])));  
}  
finalOption1.setText(finalOption1.getText() + "\n");  
finalOption1.setText(finalOption1.getText() + "All other variables have value 0.");  
finalOption1.setFont(new Font( name:"Arial", Font.PLAIN, size: 15)); //setting font size and style of the explanation text  
finalOption1.setEditable(false); //ensure the user cannot edit the explanation
```

```
; by reading along the row headers and their values.");
```

When the problem is complete, the Edexcel A-Level Decision maths questions will often ask students to list of the final values of the variables, which can often be confusing as it is not immediately obvious which values of the tableau they must read off. The students are required to read along the rows, and the values in the value column represent the value of the respective variable now in the B.V. column, which have changed as the algorithm progresses. All other variables are 0. This is explained in the final tableau explanation, and I used a for loop to create a new line and output the value each variable one-by-one, each on a new line, to ensure they can be clearly read.

Next, I added each of the initial and intermediate responses to the lists created at the start of this method. I then created a new object of type 'Random'. This creates a new random number out of 0 and 1, and the method adds the explanation text pane in that index to the addPanel. Since there is only one final tableau explanation, this is added regardless. To know which explanation to return, the tableau counter is passed as a parameter into this method. If the counter is 0, then it returns and initial explanation, if the counter is at the value of the number of tableaux in the list (e.g. it is the final tableau), it returns the final explanation, otherwise it returns an intermediate explanation.

```
public static JPanel explainTableau(double[][][] tableau, int a, int numTableau){
```

The parameters are the tableau to be explained, the counter of which tableau is being explained, and the total number of tableaux to know when to output the final tableau explanation.

```
initialResponses.add(option1);
initialResponses.add(option2);
intermediateResponses.add(midOption1);
intermediateResponses.add(midOption2);

if(a == 0){
    Random rand = new Random();
    int r = rand.nextInt( bound: 2);
    addPanel.add(initialResponses.get(r));
} else if (a == numTableau){
    addPanel.add(finalOption1);
} else{
    Random rand = new Random();
    int r = rand.nextInt( bound: 2);
    addPanel.add(intermediateResponses.get(r));
}
```

I finally returned the addPanel from the method.

```
return addPanel; //returns this panel to be added to grid panel
}
```

In the displayTableau method, I can now create a new panel which will contain the explanation, and call the createTableauPanel method on the current tableau. I specified the gridy of the GridBagConstraints to be 1, while all the other components were 0. This is to ensure that the explanation is outputted in the bottom half of the grid panel. I also set the explanation panel to span all 3 columns (tableau, row operations (if there are any) and theta values) to ensure the explanation panel takes up the whole width of the grid panel

```
JPanel explainPanel = explainTableau(outputTable, tableauCount, numTableau.toOutput.size() - 1); //creates a new panel which contains the explanation of the tableau
gbc.gridx = 0;
gbc.gridy = 1; //adds underneath the tableau
gbc.gridwidth = 3; // makes the panel span the entire width of the gridPanel
gbc.fill = GridBagConstraints.BOTH;
gridPanel.add(explainPanel, gbc); //adds the explanation to the grid panel
```

Since the tabbed pane component allows each tab at the top to be named, I decided it would be useful to name the tab based on which iteration of the algorithm it covers, so I added the following line below.

```
tabbedPane.addTab("Iteration " + (tabbedPane.getTabCount() + 1), gridPanel); //sets the title of each tab in the tabbed pane as the iteration of the algorithm
tableauCount++; //increases counter of which iteration we are on
```

This also increments the counter which keeps track of which tableau is currently being displayed. As each component of the tabbed pane has now been added, I can add it to fullPanel.

```
fullPanel.add(tabbedPane); //adds the tabbed pane to the main panel
```

The last feature to add is the highlighted pivot row, column and values for each tableau. In the createTableauPanel method, I added in 3 if statements, which checks if the current row being displayed is the pivot row, if the current column being displayed is the pivot column, and if the current value being displayed is the pivot value. If so then they are given the colours that were decided upon during the design.

```
if(i == getCurrentPivotRow(getCurrentPivotCol(outputTable), outputTable)){ //highlights the pivot row in orange for explanation
    valueLabel.setOpaque(true);
    valueLabel.setBackground(new Color(210, 126, 87));
}
if(j == getCurrentPivotCol(outputTable)){ //highlights the pivot column in blue for explanation
    valueLabel.setOpaque(true);
    valueLabel.setBackground(new Color(140, 204, 218));
}
if(i == getCurrentPivotRow(getCurrentPivotCol(outputTable), outputTable) && j == getCurrentPivotCol(outputTable)){ //highlights pivot value in a pink/purple for explanation
    valueLabel.setOpaque(true);
    valueLabel.setBackground(new Color(175, 95, 201));
}
```

Similar to the textbook, I chose a light orange and a light blue colour, however, instead of the green that the textbook used for the pivot value, which was deemed to be similar to the blue already used, I opted for a light pink/purple colour, that isn't too overwhelming, ensures the text can be read, but also clearly highlights the pivot value and remains distinct from the other colours.

I then noticed that the final tableau shouldn't have any highlighting since there are no more iterations of the algorithm to be performed.

To fix this, I added 2 new parameters for this function, which are the tableau counter from the displayTableau method, and the total number of tableau from the displayTableau method. I then checked that if the tableau counter is not equal to the total number of tableau, then the row and column can be highlighted, otherwise it is left blank.

```
if(tableauCount != numberTableau) {
    if (i == getCurrentPivotRow(getCurrentPivotCol(outputTable), outputTable)) { //highlights the pivot row in orange for explanation
        valueLabel.setOpaque(true);
        valueLabel.setBackground(new Color( r: 210, g: 126, b: 87));
    }
    if (j == getCurrentPivotCol(outputTable)) { //highlights the pivot column in blue for explanation
        valueLabel.setOpaque(true);
        valueLabel.setBackground(new Color( r: 140, g: 204, b: 218));
    }
    if (i == getCurrentPivotRow(getCurrentPivotCol(outputTable), outputTable) && j == getCurrentPivotCol(outputTable)) { //highlights pivot value in a pink/
        valueLabel.setOpaque(true);
        valueLabel.setBackground(new Color( r: 175, g: 95, b: 201));
    }
}
```

Using these methods created in the Tableau class, I can now add to the runSimplexAlg method in the menuGUI class to display the tableaux in the tabbed pane on the main window. I first created a JPanel component called combinedPanel, giving it a border layout for better control of where components are added.

```
JPanel combinedPanel = new JPanel();
combinedPanel.setLayout(new BorderLayout()); // Use BorderLayout for better control
```

To ensure there is adequate spacing around the tableau panel, I created another JPanel called tableauWrapper, to which I added an empty border. The values for the border were selected to ensure there is room below the tableau panel, so it is not pressed up against the bottom of the window, and that there is some room on all other sides to ensure success criterion 1 is met and the GUI remains clearly laid out.

```
JPanel tableauWrapper = new JPanel();
tableauWrapper.setLayout(new BorderLayout());
tableauWrapper.setBorder(BorderFactory.createEmptyBorder( top: 10, left: 10, bottom: 75, right: 50)); // Adjust top and left padding
```

I then created the tableau panel, which called the displayTableau method, passing in parameters as the row headers, column headers and the number of rows and columns, which were given in terms of number of inequalities and number of variables. As explained in the design of the array to store a tableau, the number of

rows is equal to the number of inequalities + 1, and the number of columns is equal to the number of variables + number of inequalities + 1 (without accounting for row headers and column headers, as these are managed separately in the createTableauPanel method in the Tableau class).

```
JPanel tableauPanel = Tableau.displayTableau(rowHeaders, colHeaders, rows: numInequality + 1, cols: numVAR + numInequality + 1);
```

I added the tableau panel to the center of the wrapper panel, to maintain the spacing around the panel. Finally, to the combined panel, I added the rows of inequalities created in the first part of this iteration, as well as the tableau wrapper containing the tableau panel. I specified to add the inequality panel in the center section of the border layout of the combined panel, and the tableau wrapper to the south section of the border layout, to ensure the correct formatting of the inequalities at the top of the window and the displayed tabbed pane below it.

```
tableauWrapper.add(tableauPanel, BorderLayout.CENTER); // Add tableau to the wrapper
combinedPanel.add(inequalityPanel, BorderLayout.CENTER); // Add the inequality panel
combinedPanel.add(tableauWrapper, BorderLayout.SOUTH); // Add the tableau wrapper to the bottom
```

I then added the combinedPanel to the main GUI, before revalidating and repainting the GUI to ensure it is correctly updated to add the components when the runSimplexAlg method is run.

```
add(combinedPanel, BorderLayout.CENTER); // Add the combined panel to the main frame

revalidate();
repaint();
```

The final feature to develop in this iteration is the input validation, to ensure that the program doesn't cause an error when the user does something that could cause the methods to not work as expected. These possible errors were outlined in the design section, I will paste the table of errors again below:

Error	Explanation of error
Selecting number of variables before number of inequalities	Since the functionality of the number of variables button will require the value from the number of inequalities drop-down, selecting the former first will result in a number format exception, since the default state of the

	drop-down menu is the string “--”. The function for the button will try to use “--” to define the loop counter for how many rows should be displayed, throwing an error.
Pressing run before selecting the variables, so no inputs have been collected.	This will cause an error, as the functions to retrieve inputs will try to store values from components that haven't yet been added to the GUI
Pressing run before selecting inequalities and variables	Similar error to above, as the functions to retrieve inputs will try to store values from the components that haven't been added to the GUI and don't contain any values yet
Entering strings or any non integer,double value into the text areas	As the values in the text areas are stored as doubles, a number format exception will be thrown when trying to store a string. Another possible error is trying to perform calculations on strings, which will also throw an error.
Entering nothing/ leaving the text areas blank.	This is a special case of the ‘entering strings’ error, as when parsing over the text areas, the values in the text areas will be stored as empty strings, giving the same error as above.

I created a new JLabel called errorMessageLabel, which is defined in the menuGUI constructor. The text is set to red so it stands out to the user when they enter something incorrectly into the GUI. Since the error message shouldn't always be visible on the GUI, the text is initially set to an empty string. I decided to add the label next to the buttons on the GUI on the simButtonPanel, as this is separated from the main center section of the window by the JSeparator line, so it won't get lost among the other components.

```

errorMessageLabel = new JLabel( text: "" ); // initialise the error message label
errorMessageLabel.setForeground(Color.RED); //set the error message colour to red

simButtonPanel.add(errorMessageLabel);

```

The first possible error I dealt with was the empty inputs to the text areas, or any non-number inputs to the text areas. I implemented a try-catch block around the main code in the runSimplexAlg method, as this is where the inputs from the GUI are stored. Inside the 'try' section, I set the error message label to an empty string once all the parsing of the components is complete, as this suggests there are no input errors into the text areas. This will ensure that if the user enters something incorrectly, but then enters it correctly the next time, the label won't show the error message again.

```

public void runSimplexAlg(int numInequality, int numVAR){
    constraints = new double[numInequality][numVAR + 1]; //2d array of constraints
    try {
        for (int i = 0; i < numInequality; i++) { //iterating through each set of inequalities
            JPanel parsePanel = (JPanel) inequalityPanel.getComponent(i); //creates a new panel object to extract components from
            int columnIndex = 0; //introducing the columnIndex counter
            for (int j = 0; j < parsePanel.getComponentCount(); j++) { //iterating through the panel
                Component component = parsePanel.getComponent(j);
                if (component instanceof JTextArea) { //checks if the current component is a text area
                    if (columnIndex < constraints[i].length) {
                        constraints[i][columnIndex] = Double.parseDouble(((JTextArea) parsePanel.getComponent(j)).getText()); //adds the value in the text area to the array
                        columnIndex++;
                    }
                }
            }
        }
        JPanel objectivePanel = (JPanel) inequalityPanel.getComponent(numInequality); //creates a new objective panel
        ArrayList<Double> objective = new ArrayList<>(); //creating an arraylist to store the objective function
        for (int k = 0; k < objectivePanel.getComponentCount(); k++) {
            Component comp = objectivePanel.getComponent(k);
            if (comp instanceof JTextArea) { // Check if the current component is a JTextArea
                String text = ((JTextArea) comp).getText();
                objective.add(Double.parseDouble(text)); //converts the text to a double
            }
        }
        errorMessageLabel.setText("");
    }
}

```

Inside the 'catch' section, I user a numberFormatException, and wrote a line which says that when the error is thrown, it changes the text of the label, indicating the error.

```

} catch(NumberFormatException e) {
    errorMessageLabel.setText("Please enter valid numbers only.");
}

```

The next possible user input error I dealt with was the user not selecting the number of inequalities before they try and press the button to select the number

of variables. In the actionPerformed method, I added an if statement which checks if the contents of the drop-down menu are “--” (the default option), and if so, the error message label text is once again changed to indicate the error. This was implemented for each of the three variable buttons:

```

} else if (e.getSource() == twoVar) {
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of inequalities first.");
    } else {
        errorMessageLabel.setText(""); // Clear the error message
        setSimplex(twoVar);
    }
} else if (e.getSource() == threeVar) {
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of inequalities first.");
    } else {
        errorMessageLabel.setText(""); // Clear the error message
        setSimplex(threeVar);
    }
} else if (e.getSource() == fourVar) {
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of inequalities first.");
    } else {
        errorMessageLabel.setText(""); // Clear the error message
        setSimplex(fourVar);
    }
}

```

The final error I dealt with was the user trying to press the ‘Run’ button before selecting the number of inequalities and the number of variables. I once again implemented the fix in the actionPerformed method. The first if-statement is to see if the user has selected the number of inequalities, as this is the first input required. If they haven’t the error message is displayed describing the error.

```

} else if (e.getSource() == runSimplexButton) {
    // validate if the number of inequalities and variables are selected
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of inequalities.");
    }
}

```

The next part sees if the user has selected the number of variables. For the number of inequalities, I could check if the default option was selected, but for the buttons, I didn’t have a way to check if the buttons had been pressed. Instead, I initialised the value of numVAR (the number of variables which are assigned once the button is selected) to 0. Since none of the buttons can make the value of

numVAR 0, I can make the condition of the if-statement whether the value of numVAR is 0 or not. If it is, then a button hasn't been pressed, and I can display the error message:

```
numVAR = 0; //initialise the number of variables to 0 for input validation

} else if (numVAR == 0) { //numVAR is set to 0 if no variable type is selected
    errorMessageLabel.setText("Please select the number of variables.");
} else {
    int selectedInequality = Integer.parseInt(numInequality.getSelectedItem().toString());
    runSimplexAlg(selectedInequality, numVAR);
}
```

If all the necessary user inputs have been given, then I can get the number of inequalities from the drop-down and pass the number of inequalities and the number of variables into the runSimplexAlg method and call it. This covers all the input validation required by the table above.

Having now developed all the methods for this iteration, the main technique I identified that may be useful in future iterations is being able to store a copy of an object that has changing values throughout, as this will ensure that each step of the algorithm can be displayed to the user, and will better aid the explanation of each step.

I now need to test the functionality in detail to ensure that the methods have been developed correctly and that they can handle a wide range of user inputs.

## Iteration 5: Creating Visual Graphs in the GUI

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>)

The main feature of this iteration is being able to draw a graph onto the screen based on what the user inputs in the text areas that were created in Iteration 3. Before developing this, I first need a way to display a circle on the GUI using the

paintComponent method. I will develop this in a separate class with its own basic JFrame before implementing the methods into the main GUI.

```
import javax.swing.*;
import java.awt.*;

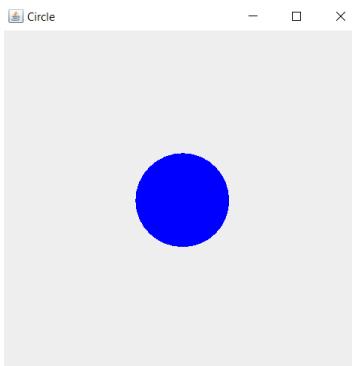
public class Display {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Circle");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);
        JPanel drawPanel = new JPanel() { // create and add a panel for drawing the circle
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);

                g.setColor(Color.BLUE);
                int diameter = 100; //to make a perfect circle, width and height must be equal
                int x = (getWidth() - diameter) / 2; // center the circle horizontally
                int y = (getHeight() - diameter) / 2; // center the circle vertically
                g.fillOval(x, y, diameter, diameter); // draw the circle
            }
        };
        frame.add(drawPanel);

        frame.setVisible(true); // make the frame visible
    }
}
```

This code uses a new class called Display, which creates a basic frame, adds a JPanel to it, before drawing on a circle. The diameter of the circle is set at 100 pixels, and the x and y coordinates of the drawing are set at the center of the panel. One thing to note at this point is the @Override before the paintComponent method. The IDE I am using (IntelliJ) automatically added in this line when creating the paintComponent method. The override is not necessary for the method and it does run correctly without the line, however, to ensure robustness of code, the override lines will be left in to avoid any runtime errors.

Running the above code generates the following output



This ran successfully and as expected created a circle on the GUI. Since the user inputs edges into the text area, I will now adapt this code to generate two circles and connect them with a line.

```
public class Display {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Circle");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);
        JPanel drawPanel = new JPanel() { // create and add a panel for drawing the circle
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);

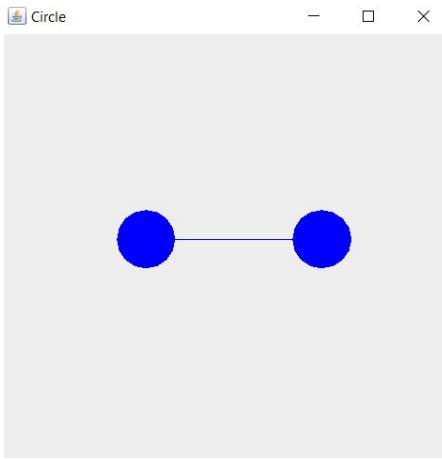
                g.setColor(Color.BLUE);
                int diameter = 50; //to make a perfect circle, width and height must be equal
                int x1 = 100; // center the circle horizontally
                int y1 = 150; // center the circle vertically
                g.fillOval(x1, y1, diameter, diameter); // draw the circle

                int x2 = 250; //setting location of second circle
                int y2 = 150;
                g.fillOval(x2, y2, diameter, diameter);
                g.setColor(Color.BLUE);

                int centerX1 = x1 + diameter/2; //to draw a line we need the center of both circles
                int centerY1 = y1 + diameter/2;
                int centerX2 = x2 + diameter/2;
                int centerY2 = y2 + diameter/2;
                g.drawLine(centerX1, centerY1, centerX2, centerY2); //connecting the center of two circles with a line
            }
        };
    }
}
```

Since I am testing a fixed number of circles, I decided to manually add new coordinates to draw a circle. The best way to draw a line between the circles is to get the coordinates of the centers of the two circles, then draw a line connecting them. This is done after drawing the two circles, as shown with the `centerX1`, `centerY1`, `centerX2` and `centerY2`.

Running this adapted code produced the following output

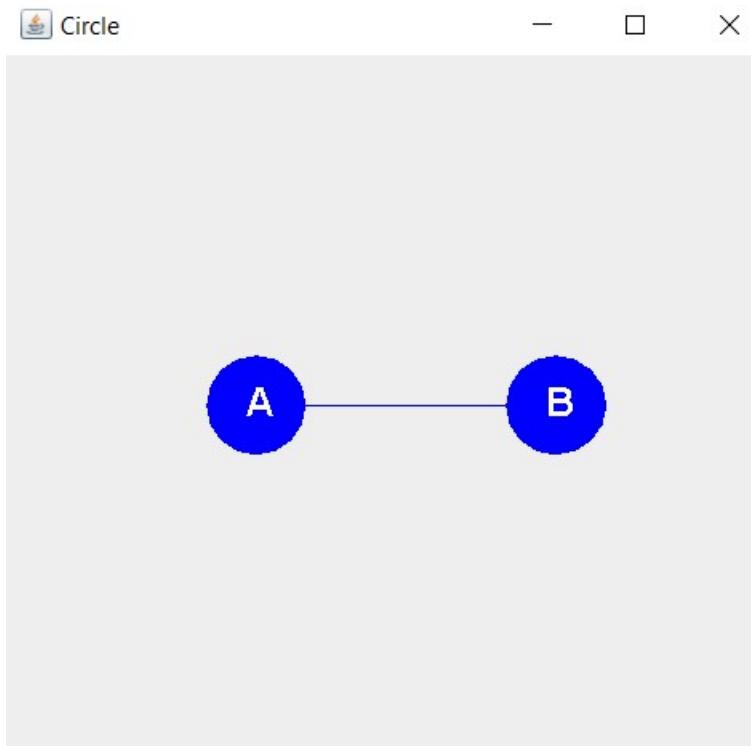


Again this ran successfully and displayed the two circles horizontally aligned, connected by a line.

Next, I am going to try and add labels to the nodes, allowing the edge to be identified e.g. with one node as A and another as B, we can note the edge 'AB'.

```
g.setColor(Color.WHITE);
g.setFont(new Font("Arial", Font.BOLD, 20)); //Labelling the circles
g.drawString("A", centerX1 - 5, centerY1 + 5); // center label 'A'
g.drawString("B", centerX2 - 5, centerY2 + 5); // center label 'B'
```

Since I am still manually defining the coordinates for the circles, I can manually add the labels by using the `drawString` method as part of `paintComponent`, adding the label roughly to the center of the node, but slightly higher up to ensure there is no overlap with the edge line connecting the nodes. Running the code at this point produced the following output:



Now being able to display two circles connected by a line on the GUI, I am going to work on developing this for a variable number of edges, instead of the manual 2 nodes I have currently been working on.

I created a new class to temporarily use in which I will get user inputs from the terminal. The two inputs I am going to take while developing this will be the number of edges and the edges themselves. The number of edges will be used just to determine how many edges the user should be able to enter into the terminal before it stops parsing over them and begins adding them to the graph.

```
public class GraphDisplay extends JFrame{  
    1 usage  
    public static JFrame frame;  
    public static void main(String[] args){  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter number of edges: "); //prompt user for number of edges  
        int numEdges = scanner.nextInt();  
        scanner.nextLine();
```

This gets the number of edges and stores them in numEdges.

```
ArrayList<String> edges = new ArrayList<>(); //new list of edges
System.out.println("Enter edges: ");
for(int i = 0; i < numEdges; i++){
    String edge = scanner.nextLine(); //adding each inputted edge to the list
    edges.add(edge);
```

This creates a new arraylist of strings to store the edges. The user is able to continually enter edges on a new line until they have entered the number of edges they previously defined.

I can now adjust the Display class, which I have now renamed to DisplayPanel as it better explains the functionality of the class, creating a frame upon which the graph is displayed. This currently has 4 parameters, the list of edges, a list of labels for each of the nodes, a list of positions for each of the nodes, and the fixed diameter of the circles to be draw. Something to note is that the nodePositions list is an arraylist of Point objects, which can store an x and y coordinate, which will be used later in setting the location for the node.

I created the constructor for this class, which takes a list of edges as a parameter and assigns it to the class attribute which is a list of edges for use in the DisplayPanel class. This constructor also calls the calculatePositions method, which I will explain below

```
public class Display extends JPanel{
    3 usages
    public ArrayList<String> edges;
    10 usages
    public ArrayList<Character> nodeLabels;
    6 usages
    public ArrayList<Point> nodePositions;
    10 usages
    public int diameter = 50;

    1 usage
    public Display(ArrayList<String> edgeList){
        edges = edgeList;
        calculatePositions();
    }
    1 usage
```

Below is the calculatePositions method. The purpose of this method is to separate the list of edges into unique nodes e.g. 'AB', 'AC', 'BC' will become 'A', 'B', 'C', so none of the nodes are repeated. First, the edge is split up into source and destination, then each of these are checked to see if they exist in the list of nodeLabels, and if not, then adding them to the list, as well as creating a placeholder in the list of positions.

```

public void calculatePositions(){
    for(String s: edges){
        char startNode = s.charAt(0);
        char destNode = s.charAt(1);

        if(!nodeLabels.contains(startNode)){
            nodeLabels.add(startNode);
            nodePositions.add(new Point(x: 0, y: 0));
        }
        if(!nodeLabels.contains(destNode)){
            nodeLabels.add(destNode);
            nodePositions.add(new Point(x: 0, y: 0));
        }
    }
}

```

After ensuring each node is added uniquely to the lists of labels and positions, I can now work on defining the position on the frame that each of these nodes should be displayed. This will use the trigonometry explained in the Design section, by splitting 360 degrees into equal sections based on the number of nodes, so with 5 nodes, 360 would be split up into 72 degrees each, but with 10 nodes, 360 would be split up into 36 degrees each. Instead of using degrees, I am using radian measure, which is another method of measuring angles, and this is used as the Math.cos() and Math.sin() methods use radians to function.

```

JFrame circleFrame = GraphDisplay.getFrame();
int centerX = circleFrame.getWidth() / 2;
int centerY = circleFrame.getHeight() / 2;
int radius = Math.min(circleFrame.getWidth(), circleFrame.getHeight()) / 3;

for(int i = 0; i < nodeLabels.size(); i++){
    double angle = 2 * Math.PI * i / nodeLabels.size();
    int x = centerX + (int)(radius * Math.cos(angle)) - diameter / 2;
    int y = centerY + (int)(radius * Math.sin(angle)) - diameter / 2;
    nodePositions.set(i, new Point(x, y));
}

```

First, I got the center of the frame, and defined a radius to ensure the edges are displayed in the center area of the frame and aren't spread apart too far. Using a

for loop, I then iterated through the list of node labels and defined the x and y coordinates of each point and set the position that each node should be displayed.

Having calculated the positions of each node and stored them in the list of positions, I can now adapt the paint component method to draw each of the nodes onto the frame.

```
@Override  
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    Graphics2D g2d = (Graphics2D) g;  
    g2d.setColor(Color.BLUE);  
  
    for(String edge: edges){  
        char startNode = edge.charAt(0);  
        char destNode = edge.charAt(1);  
  
        int startIndex = nodeLabels.indexOf(startNode);  
        int destIndex = nodeLabels.indexOf(destNode);  
  
        if(startIndex != -1 && destIndex != -1){  
            Point p1 = nodePositions.get(startIndex);  
            Point p2 = nodePositions.get(destIndex);  
  
            int x1 = p1.x + diameter /2;  
            int y1 = p1.y + diameter /2;  
            int x2 = p2.x + diameter /2;  
            int y2 = p2.y + diameter /2;  
            g2d.drawLine(x1, y1, x2, y2);  
        }  
    }  
}
```

First, I iterated through the list of edges, splitting them up onto source node and destination node. After this, I then get the index in the list of node labels where each of these nodes are. Then I check to ensure that there is a valid index, before defining the x and y coordinated of the center of the two nodes and drawing the line onto the GUI. Since the components are drawn onto the frame at the same

time, it doesn't matter whether I draw the line first or the nodes first, so I chose to draw the line first as it requires more code.

```
        }
        for(int i = 0; i < nodeLabels.size(); i++){
            Point position = nodePositions.get(i);
            char label = nodeLabels.get(i);

            g2d.setColor(Color.BLUE);
            g2d.fillOval(position.x, position.y, diameter, diameter);
            g2d.setColor(Color.WHITE);
            g2d.drawString(String.valueOf(label), position.x + diameter/3, position.y + 2 * diameter/3);
        }
    }
```

I then iterated through the list of node labels, getting the position of each node from the list of node positions, before drawing a circle at each of those points and adding the node label as well.

Running this, I was met with an error:

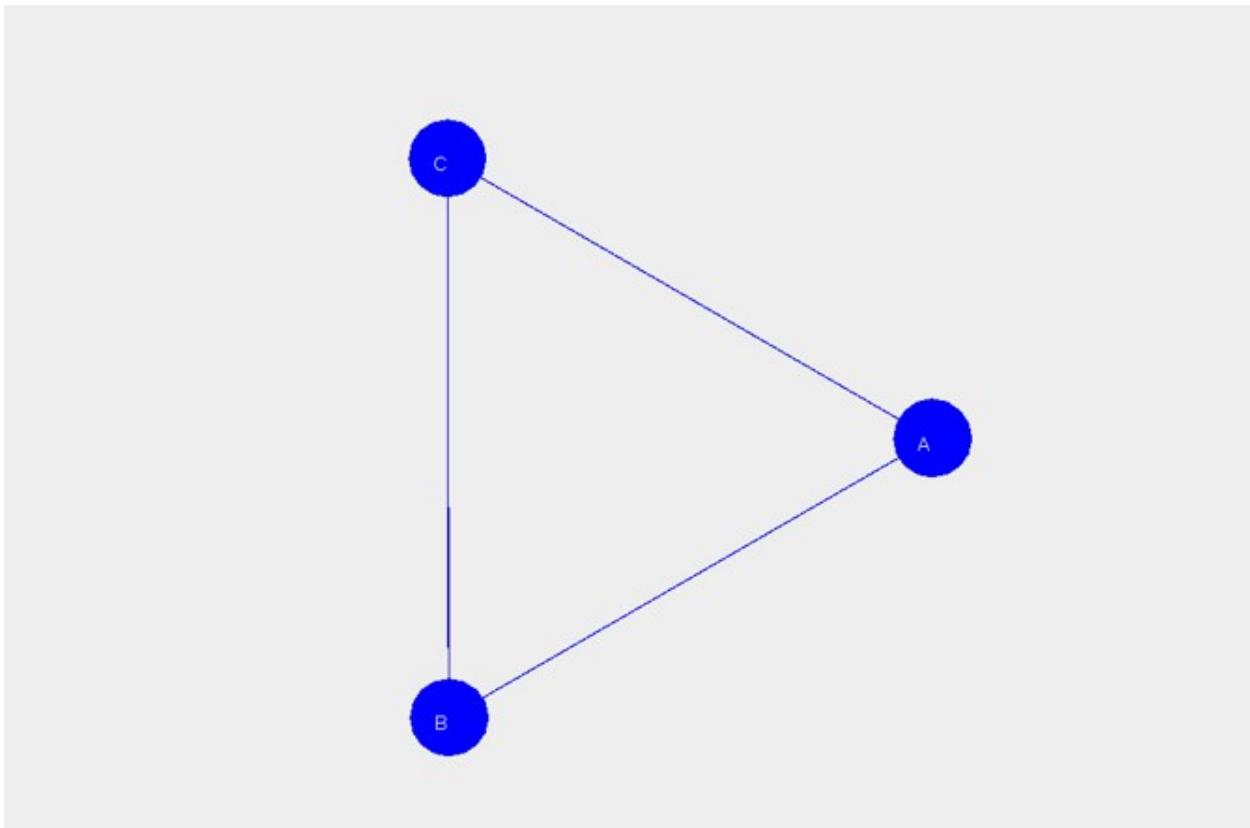
```
Exception in thread "main" java.lang.NullPointerException
at Display.calculatePositions(Display.java:28)
at Display.<init>(Display.java:13)
at GraphDisplay.main(GraphDisplay.java:28)
```

The small mistake in the code was that the list of node labels hadn't been properly defined, just added as an attribute. I realised that the same error would occur for the list of node positions, so I altered the lines to create the arraylists straight away, instead of just adding them as an attribute:

```
class Display extends JPanel{
    3 usages
    public ArrayList<String> edges; //list of edges in the graph
    10 usages
    public ArrayList<Character> nodeLabels = new ArrayList<>(); //list of labels in the graph
    6 usages
    public ArrayList<Point> nodePositions = new ArrayList<>(); //list of locations on the window for the vertices
    10 usages
    public int diameter = 50; //diameter of the vertices
    5 usages
    private JFrame circleFrame; //frame to add graph to

    1 usage
    public Display(ArrayList<String> edgeList){ //constructor which defines edges and calculates their positions
        edges = edgeList;
        calculatePositions();
    }
}
```

I then quickly ran this with 3 basic edges, and the correct layout was maintained, successfully adding the edges to the frame:



Since the graph needs to be displayed on the existing frame, I now need to alter the existing code so that the graph is displayed on a JPanel which can then be added to the main GUI in the Kruskal's section.

```

import java.util.ArrayList;
import java.util.Scanner;
import javax.swing.*;
import java.awt.*;

1 usage
public class GraphDisplay extends JFrame {
    1 usage
    public static JPanel createGraphDisplay(ArrayList<String> edgeList){ // method to return the graph panel
        DisplayPanel displayPanel = new DisplayPanel(edgeList); //creating the graph with the list of edges passed into the constructor
        return displayPanel;
    }
}

```

I changed the GraphDisplay class so instead of taking user inputs, it just contains one method to create the graph by creating a DisplayPanel object.

```

class DisplayPanel extends JPanel {
    3 usages
    private ArrayList<String> edges; // list of edges in the graph
    10 usages
    private ArrayList<Character> nodeLabels = new ArrayList<>(); // list of labels in the graph
    6 usages
    private ArrayList<Point> nodePositions = new ArrayList<>(); // list of locations on the panel for the vertices
    10 usages
    private int diameter = 50; // diameter of the vertices

    1 usage
    public DisplayPanel(ArrayList<String> edgeList) { // constructor which defines edges and calculates their positions
        edges = edgeList;
        calculatePositions();
    }
}

```

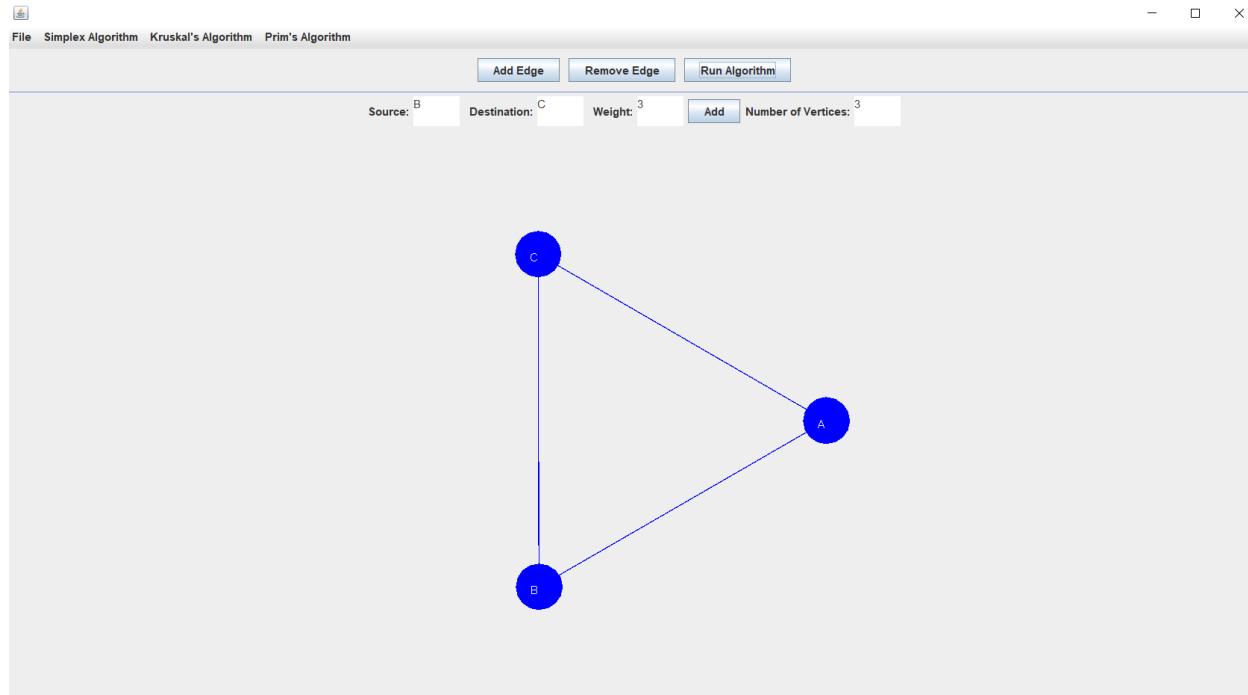
The attributes of the `DisplayPanel` class currently remain the same, with the same constructor method. The rest of the methods in the `DisplayPanel` class all remain the same to display a graph. In the `runKruskalsAlg` method in the `menuGUI` class, I then created a `JPanel` which takes the graph panel, before adding it to the GUI.

```

JPanel graphPanel = GraphDisplay.createGraphDisplay(edgePairs); //creating a panel with the graph
add(graphPanel, BorderLayout.CENTER); //adding the panel to the center of the frame
revalidate(); //refreshing the GUI
repaint();

```

Testing this with the same basic edges gave the following successful output, displaying the graph on the existing GUI



Currently, the entire graph is displayed on the GUI, but if the user wants to be able to move the nodes around before they run the algorithm, the graph needs to be displayed on the GUI before they run the algorithm. To implement this, I am going to create a method that adds the edges one at a time, each time the user presses the 'Add' button, so the graph gradually builds up on the GUI.

```
public void addEdge(String edge){  
    edges.add(edge); //adding the edge to the final list  
    calculatePositions(); //calling the method to calculate the position of the vertex  
    revalidate(); //refreshing the GUI  
    repaint();  
}
```

This method adds the current edge to the list of edges, before calling the calculatePositions() method to get the positions of all the nodes currently added to the graph.

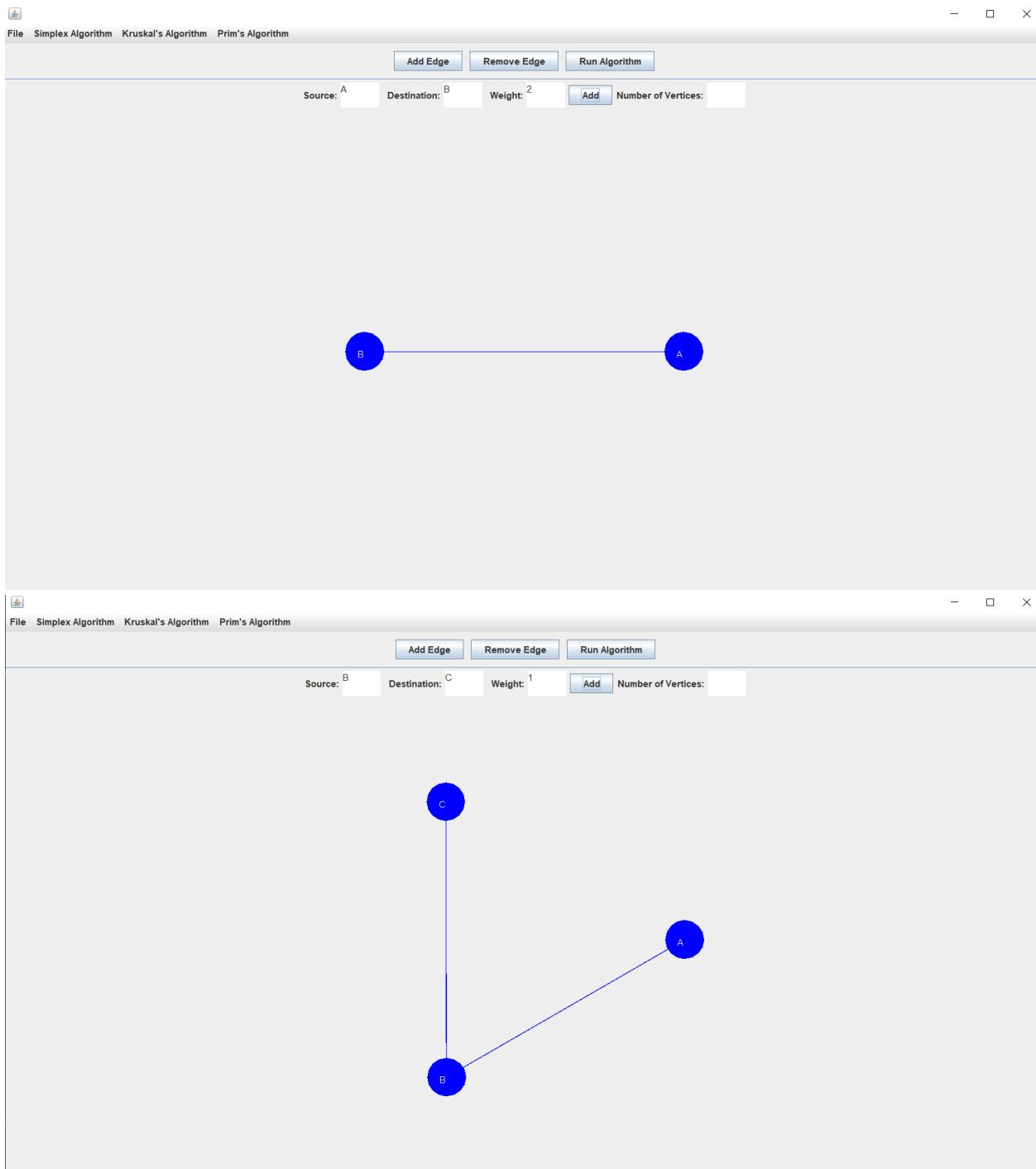
Since the DisplayPanel object now needs to be used in multiple methods in the menuGUI class, I made it an attribute declared at the start of the class

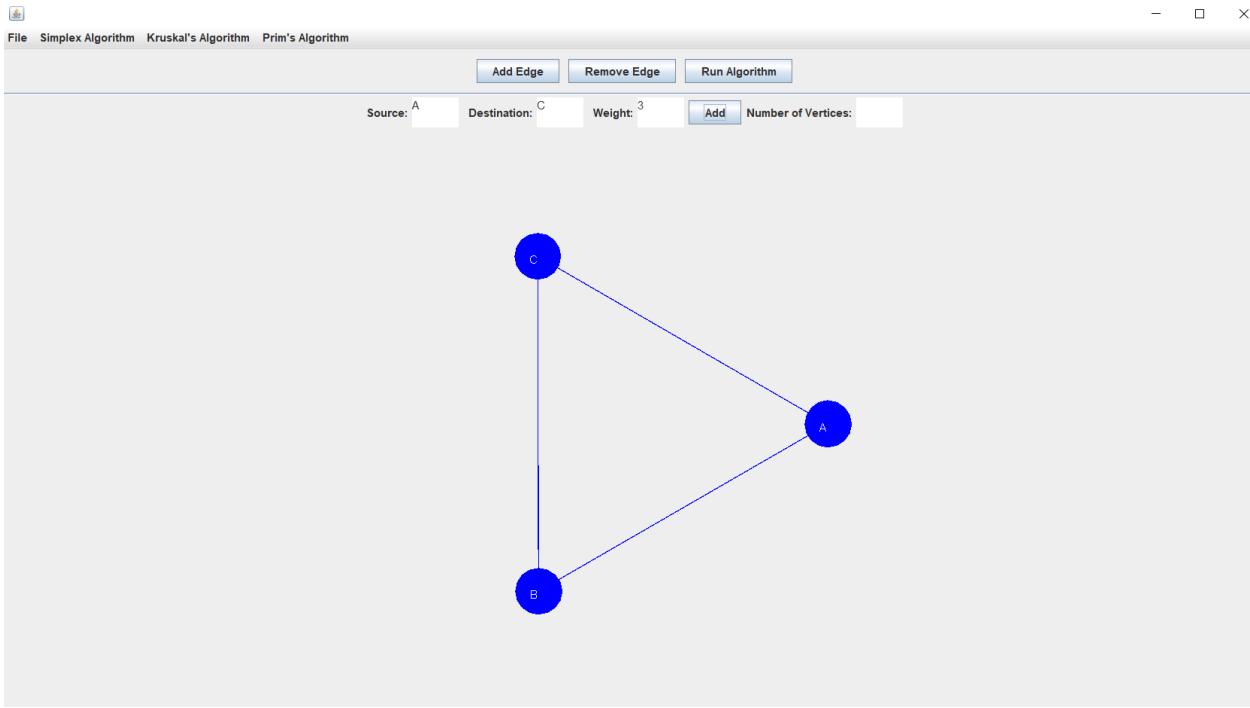
```
2 usages  
DisplayPanel displayPanel = new DisplayPanel();  
  
-  
  
String pair = source + dest; //formats the source and destination as one string  
toAdd.add((int)sourceCH); //adds the source, destination and weight to an arraylist of information about edges to add  
toAdd.add((int)destCH);  
toAdd.add(weightInt);  
displayPanel.addEdge(pair); //adds the edge to the panel  
revalidate(); //refreshing the GUI  
repaint();
```

In the method that stores the individual inputs into the text areas, I then called the addEdge method with the current edge, which will then display the graph edge by edge on the GUI.

```
add(displayPanel, BorderLayout.CENTER);  
revalidate();  
repaint();
```

Testing this with the same basic 3 edges, the following output was achieved, showing the graph being gradually built as the user enters edges





At this point, the graph displays the nodes, with edges between them and the label for each node, but currently there are no weights, an important feature of the graph.

```
private ArrayList<Integer> edgeWeights = new ArrayList<>(); // list of weights for the edges

1 usage
public DisplayPanel(){ // constructor which defines edges and calculates their positions
    edges = new ArrayList<>(); //creating the list that stores the edges
}
1 usage
public void addEdge(String edge, int weight){
    edges.add(edge); //adding the edge to the final list
    edgeWeights.add(weight); // adding the weight to the list
    calculatePositions(); //calling the method to calculate the position of the vertex
    revalidate(); //refreshing the GUI
    repaint();
}
```

I created an arraylist of integers as an attribute to store the edge weights, then in the addEdge method, I wrote a line that takes a weight passed as a parameter and adds it to the list.

```

int weight = edgeWeights.get(i); // get the weight for the current edge
int weightX = (x1 + x2) / 2; // x position for the weight
int weightY = (y1 + y2) / 2; // y position for the weight
g2d.drawString(String.valueOf(weight), weightX, weightY);

```

In the paintComponent method, I then added the above lines, which gets the weight of the current edge being outputted, before getting the position to draw the weight, which I decided would be the middle of the edge. I then used the drawString method to draw the text onto the graph panel. Since the addEdge method now takes two parameters, I adjusted the line in menuGUI to pass in the edge pair as well as the weight.

```
displayPanel.addEdge(pair, weightInt); //adds the edge to the panel
```

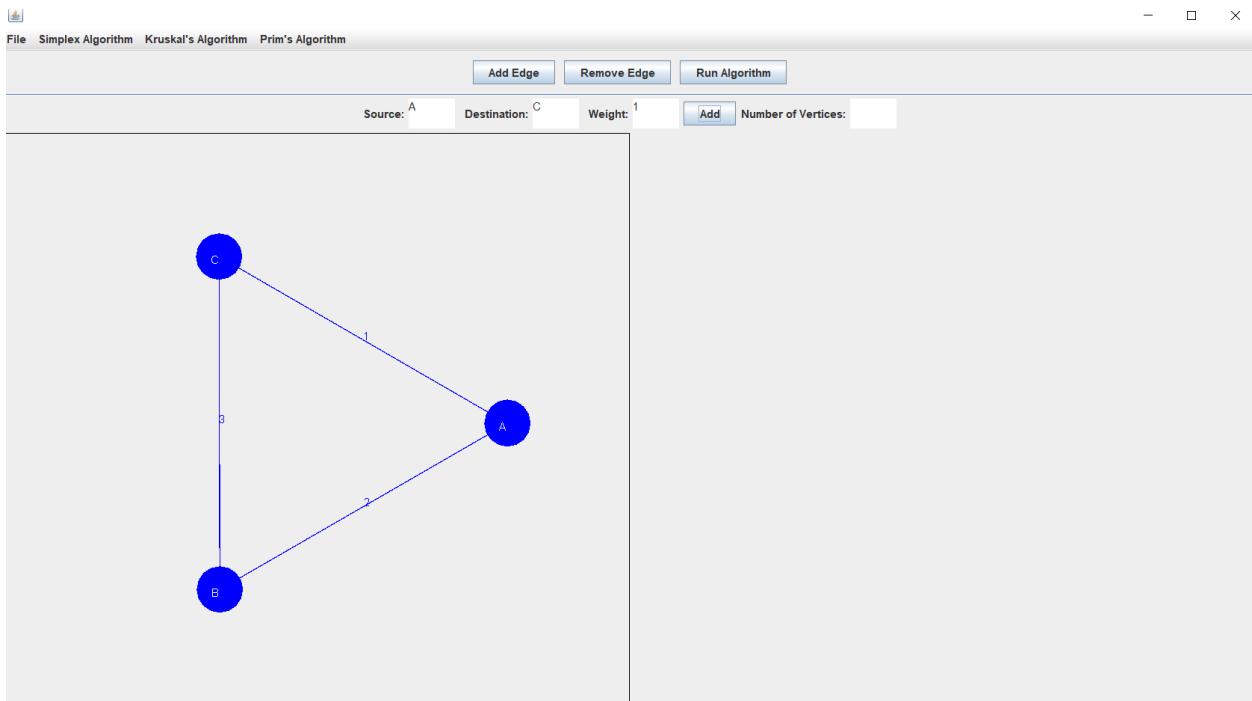
The graph is currently displayed in the center of the GUI, however this is not an optimal use of the space, since adding the explanations and the table of added and rejected edges would result in them being scattered around the edges, making them hard to follow. Using the Design section of the GUI, I decided to add the graph to the left of the frame and the explanation to the right of the frame. To do this, I created one main panel that uses a grid layout, with 2 columns. This allows me to add the graph to the first column and the rest of the explanation to the right column.

```

displayPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
combPanel = new JPanel(new GridLayout( rows: 0, cols: 2));
combPanel.add(displayPanel);
add(combPanel, BorderLayout.CENTER);
revalidate();
repaint();

```

Testing this gave the following output, correctly outputting the graph on the left half of the frame



The next part of the iteration involves allowing the user to move the nodes around to make the graph look as they intend. As outlined in the design, this will involve 3 methods, first to check when the mouse is pressed, the second to check where the mouse is let go, and a third to detect where the mouse was dragged. All these methods are created within the method `setMouseListeners`, as the few methods don't require their own entire class. As mentioned earlier, the IDE IntelliJ automatically adds in overrides to the event methods. To ensure that the methods continue to function as intended, I will leave the override methods in. I added in the drag start and drag index attributes, which will be used for the drag function of the nodes.

```
private Point dragStart = null; // starting point for dragging
private int dragIndex = -1; // index of the node being dragged
```

```

private void setMouseListeners() {
    // add mouse listener to handle mouse press events
    addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent e) {
            Point clickPoint = e.getPoint(); // get the position where the user clicked

```

To ensure the node circle is moved correctly, the mousePressed method needs to check whether the location of the click was within the node. To do this, I got the current location of the click as a Point object, before iterating through the list of node positions and making sure that the click is within the radius distance of the center of the node. Using the Point objects allowed me to separate the positions into x and y coordinates, before using the value of the diameter divided by 2 to get the radius of the circle.

```

for (int i = 0; i < nodePositions.size(); i++) {
    Point nodePosition = nodePositions.get(i);
    int centerX = nodePosition.x + diameter / 2; // find the center of the node
    int centerY = nodePosition.y + diameter / 2;
    double distance = clickPoint.distance(centerX, centerY); // measure the distance from the click to the center of the node
    if (distance <= diameter / 2) { // check if the click is within the node
        dragStart = clickPoint; // set the start point for dragging
        dragIndex = i; // store the index of the node being dragged
        break; // exit the loop once a node is found
    }
}

```

If the click is within the correct distance, it sets the drag start point as the current click, and it adds the current index in the list of node positions to the drag index, before breaking from the for loop.

The next method to develop is the mouse released method, to ensure the node stops being moved once the user released the mouse click. The main purpose of this method is to reset the values of the drag start point and the drag index, so the user can continue to move other nodes of the graph and the values from a previous node aren't maintained.

```

@Override
public void mouseReleased(MouseEvent e) {
    dragStart = null; // clear the drag start point
    dragIndex = -1; // reset the drag index
}

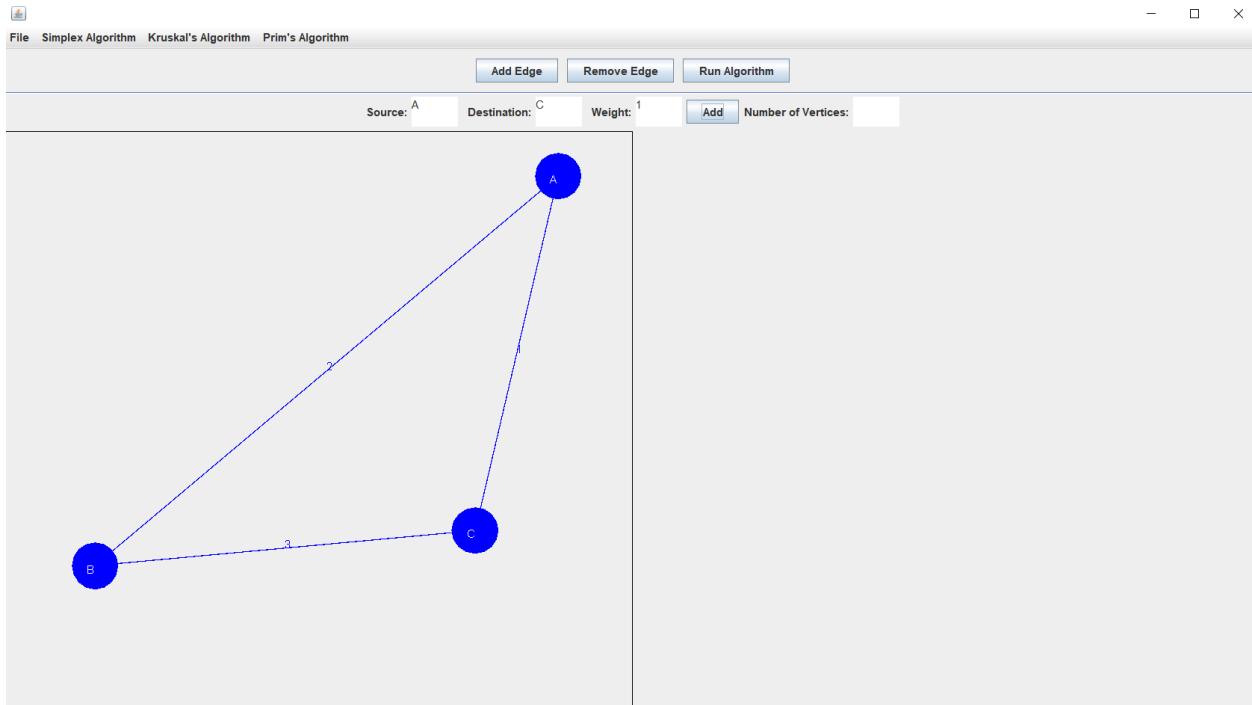
```

The third method is the mouse dragged method, which takes the current location of the mouse and gets the position at the drag index in the node positions list. Ensuring that dragStart and dragIndex have valid values, I added an if-statement to check the values.

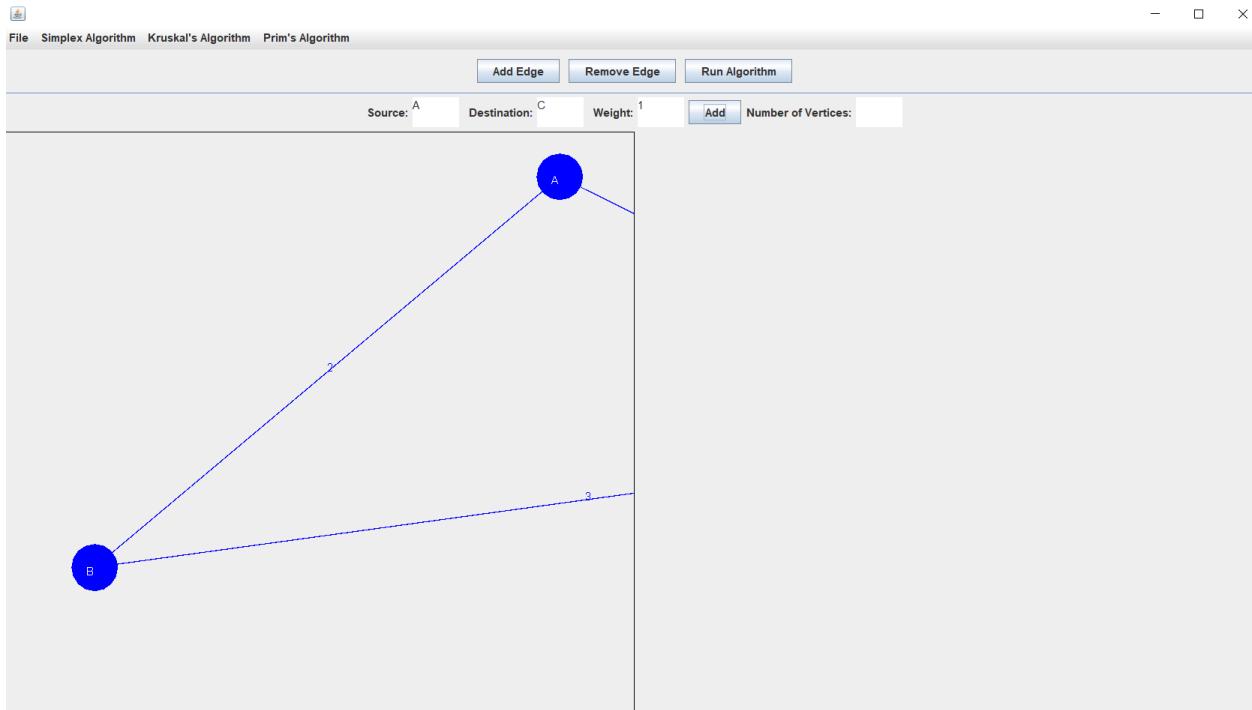
I then split the movement of the mouse into horizontal and vertical movement which subtracts the current position of the mouse from the initial location of the node. I then set the location of the node as the new positions

```
addMouseMotionListener(new MouseAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        if (dragIndex != -1 && dragStart != null) { // check if a node is being dragged
            Point currentPoint = e.getPoint(); // get the current mouse position
            Point nodePosition = nodePositions.get(dragIndex); // get the position of the node being dragged
            int horizontalMovement = currentPoint.x - dragStart.x; // calculate the horizontal movement
            int verticalMovement = currentPoint.y - dragStart.y; // calculate the vertical movement
            nodePosition.translate(horizontalMovement, verticalMovement); // update the node's position
            dragStart = currentPoint; // update the drag start point for the next movement
            repaint(); // refresh the GUI to reflect the new node position
        }
    }
});
```

Before moving on, I decided to run a quick test of the moving nodes. The initial tests were successful, as shown below.



However, when I dragged one of the nodes too far, the following occurred:



Initially, I wasn't sure why the graph was able to leave the panel, but reading over the mouse event methods, I realised that when they get the position of the mouse, it is not limited to just the panel in which the graph is being displayed. To fix this, I decided to try and get the boundaries of the panel and ensure that the node remains within them.

I created the new x and y coordinates of the node which was the horizontal and vertical movement of the graph added to the initial position of the node.

```
int newX = nodePosition.x + horizontalMovement; // calculate new x position for the node  
int newY = nodePosition.y + verticalMovement; // calculate new y position for the node
```

The maximum location in the panel is the coordinate of the bottom right corner, so I defined the maximum x and y coordinates as the width and height of the panel, which will be equal to the coordinates of the bottom right of the panel.

The minimum location in the panel is the coordinate of the top left corner, which is (0,0), so I defined the minimum x and y coordinates as x = 0 and y = 0.

```

int minX = 0; // minimum x position (left side of panel)
int minY = 0; // minimum y position (top side of panel)
int maxX = panelWidth - diameter; // maximum x position (right side of panel)
int maxY = panelHeight - diameter; // maximum y position (bottom side of panel)

```

I then set the position to add the new node as the minimum value out of the dragged position and the maximum position. This means that if the dragged position is within the panel, the node will move to the panel, but if the dragged position is outside the panel, the node won't go past the panel.

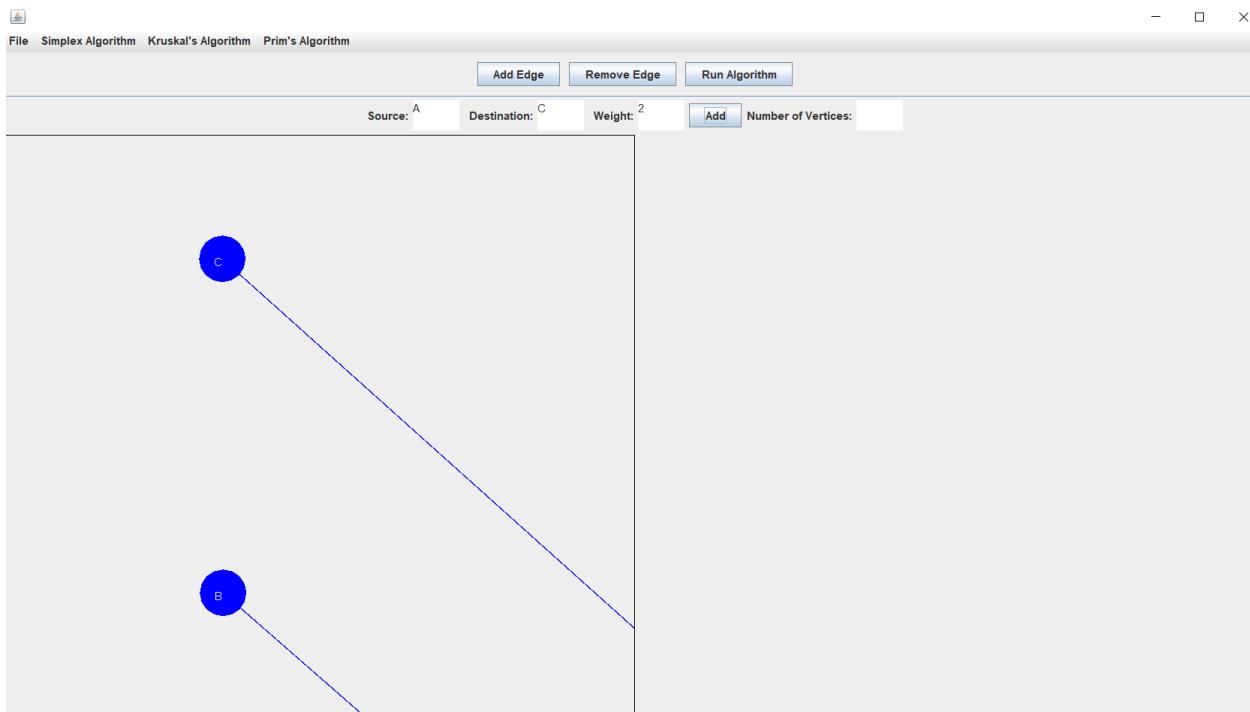
```

// keep the new position so the node stays within the panel bounds
newX = Math.max(minX, Math.min(newX, maxX));
newY = Math.max(minY, Math.min(newY, maxY));

nodePosition.translate(newX, newY); // update the node's position
dragStart = currentPoint; // update the drag start point for the next movement
repaint(); // refresh the GUI to reflect the new node position

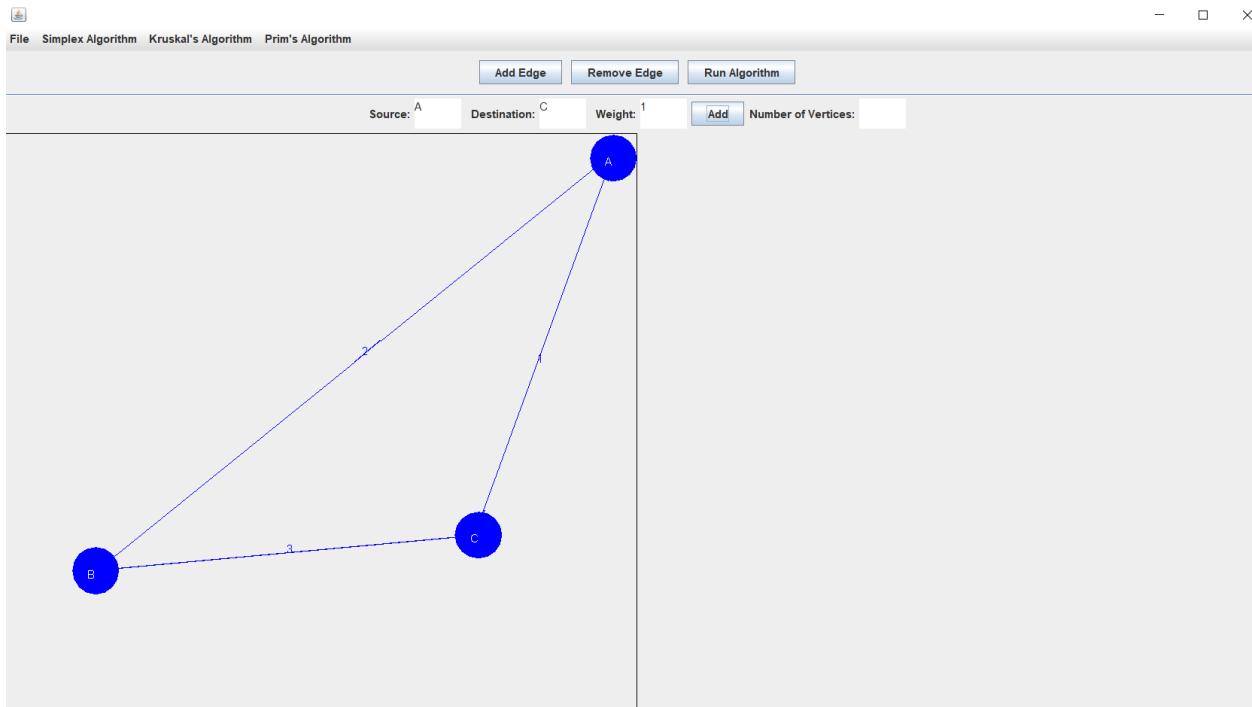
```

Running this, as soon as I clicked on one of the nodes, it would just be moved to a position in the bottom right of the panel in a position seemingly off the window



I wasn't able to find out why the node was being moved into this position, so I looked for another method to move Point objects. Instead of using the `.translate()` method, I tried using the `setLocation` method, which still takes in the x and y

coordinates as parameters. Trying this worked correctly and the node now remained within the panel.



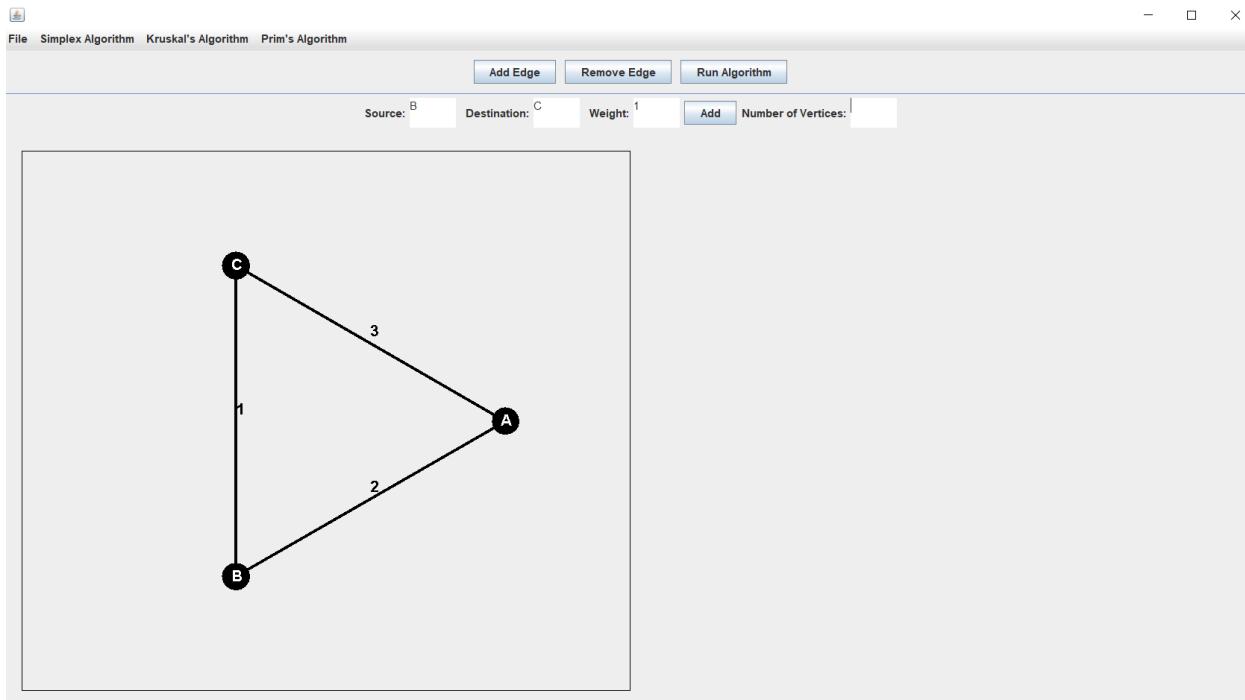
One thing I also noticed at this point is that the weight label is often covered by the edge line when it moves, so I offset the weight label slightly.

```
g2d.drawString(String.valueOf(weight), weightX, y: weightY - 7); //drawing the weight onto the panel
```

Now that the graph functionality is working, I can now format the graph to resemble something similar to the graphs that students will see in an exam question. This involves making the graph circle and lines black, as well as making the nodes smaller and the lines slightly thicker.

```
g2d.setColor(Color.BLACK); // set the color for drawing to blue
```

```
g2d.setStroke(new BasicStroke( width: 3.0f));
```



Now I can move onto working on the right column of the main panel, which will contain the table of added and rejected edges, as well as the explanation tabbed pane.

```

displayPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //setting a black border around displayPanel
fillPanel = new JPanel(new GridLayout( rows: 2, cols: 1));
fillPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
combPanel = new JPanel(new GridLayout( rows: 0, cols: 2)); //creating a new panel with 2 columns to make 2 halves which store the graph and the table
combPanel.add(displayPanel); //add the graph to the left half of the combination panel
combPanel.add(fillPanel); //add an empty panel to the right half of the combination panel
combPanel.setBorder(new EmptyBorder( top: 20, left: 20, bottom: 20, right: 20)); //create a border around the combination panel
add(combPanel, BorderLayout.CENTER); //add the combination panel to the main panel
revalidate(); //refresh the GUI
repaint();

```

I added a new panel with a grid layout to the right half of the combination panel. The dimensions for this grid are 2 rows and 1 column. The top row will contain the add/reject table, and the bottom row will contain the tabbed pane for explanations.

I created a method in menuGUI called createMSTPanel, which adds all the edges and add/reject to a table. From iteration 4, I learned that an effective way of developing a table was to use a grid layout. I created a grid layout with two columns and a variable number of rows which will adjust based on how many edges are in the graph. The left column header has the label 'Edge' and the right column header has the label 'Add/Reject' to indicate the purpose of each column.

```

public JPanel createMSTPanel(ArrayList<Edge> mstEdges, ArrayList<Edge> allEdges) {
    JPanel mstPanel = new JPanel(new GridLayout( rows: 0, cols: 2)); // Create a panel with 2 columns

    JLabel edgeLabel = new JLabel( text: "Edge"); //adding label to left column
    JLabel actionLabel = new JLabel( text: "Add/Reject"); //adding label to right column
    edgeLabel.setFont(new Font( name: "Arial",Font.PLAIN, size: 15 )); //setting font, size and style of text in label
    edgeLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //adding border around label
    actionLabel.setFont(new Font( name: "Arial",Font.PLAIN, size: 15 ));//setting font, size and style of text in label
    actionLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //adding border around label
    mstPanel.add(edgeLabel); //add the two labels to the panel
    mstPanel.add(actionLabel);
}

```

Since the list of edges is a list of Edge objects, I had to convert the integer values into letters, so the node A, which is stored as 0 has 65 added to it, then is converted to an ASCII character. I did this for the destination node as well, to get the text for an edge in the form 'AB'.

```

for (Edge edge : allEdges) {
    JLabel edgeDisplay = new JLabel( text: (char)(edge.source + 65)+ "" + (char)(edge.dest + 65)); // label for each edge
    String action = "Reject";
    if(mstEdges.contains(edge)){
        action = "Add";
    }
}

```

I then added each of the labels to the table and formatted the text to make the labels a bit larger.

```

JLabel actionDisplay = new JLabel(action); // display add or reject
edgeDisplay.setFont(new Font( name: "Arial",Font.PLAIN, size: 15 ));//setting font, size and style of text in label
edgeDisplay.setBorder(BorderFactory.createLineBorder(Color.BLACK));//adding border around label
actionDisplay.setFont(new Font( name: "Arial",Font.PLAIN, size: 15 ));//setting font, size and style of text in label
actionDisplay.setBorder(BorderFactory.createLineBorder(Color.BLACK));//adding border around label
mstPanel.add(edgeDisplay); //add the labels to the panel
mstPanel.add(actionDisplay);

return mstPanel; // return the constructed panel
}

```

I can then call this method in the runKruskalsAlg method, so when the user runs the algorithm, the table appears with the explanation

```

ArrayList<Edge> edgeList = graph.edges; //list of edges in the graph
ArrayList<Edge> mstEdges = graph.kruskalMST(); // list of edges in the minimum spanning tree
JPanel mstPanel = createMSTPanel(mstEdges, graph.edges); //panel containing the table of added/rejected arcs

```

Next I can work on the tabbed pane which will go below the table of added and rejected edges.

```

// create a container panel to hold both the graph and MST panels
combPanel.add(fillPanel); //add the empty fill panel to the right column of the combination panel
JTabbedPane explainPane = new JTabbedPane(); //creating the tabbed pane to contain the edges
Collections.sort(edgeList); //sorting the list of edges, so they can be added to the tabbed pane in sorted order
for(Edge edge: edgeList{
    JPanel edgePanel = new JPanel(); //creating a new tab for each edge in the graph
    JTextPane explanation = new JTextPane(); //adding the text pane to contain the explanation of each edge being added/rejected
    explanation.setText("Test text"); // setting some text to test the text pane
    edgePanel.add(explanation); //add the text pane to the panel for the tabbed pane
    String title = (char)(edge.source + 65) + "" + (char)(edge.dest+65); //make the edge the title of the tab e.g. 'AB'
    explainPane.addTab(edgePanel, title); //add the panel to the tabbedpane with the edge title
}

```

For each edge in the graph, a new tab is added to the tabbed pane, with the title being the name of the edge. I also added a text pane to each of them, which will contain the explanation for each edge.

Another feature to add is when the user selects one of the edges in the tabbed pane, the edge should be highlighted on the graph to show whether the graph is added or rejected. To do this, I needed a way to detect when the user selects a tab in the tabbed pane. I added a change listener object to the tabbed pane. Adding the change listener in IntelliJ automatically created an empty method called state changed, in which I will develop the logic to change the edge colours

```

explainPane.addChangeListener(new ChangeListener() { //adding a change listener to the tabbed pane to detect when the user selects a new tab
    @Override
    public void stateChanged(ChangeEvent e) {
        JTabbedPane sourcePane = (JTabbedPane) e.getSource(); //initialising a tabbed pane as the item that was updated/changed
        int selectedTab = sourcePane.getSelectedIndex(); //getting the index of the current selected tab
    }
}

```

I created a new tabbed pane which takes the source as the tab which was selected. I also needed a way to know which edge was selected, so I created a new integer which takes the value of the index of the selected tab in the tabbed pane. After checking that the user has selected one of the edge tabs, I created a new boolean variable which will be used to check whether an edge is in the minimum spanning tree or not. Iterating through the list of edges in the minimum spanning tree, the value of the boolean was set to true if the current edge from the selected tab was in the list. If the boolean is true, then the highlightEdge method is called with a green edge, otherwise the method is called and the edge colour is set to red.

```

public void highlightEdge(String edge, Color colour){ //method to colour a specific edge in the graph
    highlightedEdge = edge; //the edge to be highlighted passed as a parameter
    edgeColour = colour; //the colour that this edge should be based on whether it is in the minimum spanning tree or not
    revalidate(); //refreshing the GUI
    repaint();
}

if(selectedTab != -1){ //checking that a tab has been selected
    String selectedEdge = sourcePane.getTitleAt(selectedTab); //creating a new string which takes the value of the title of the tab e.g. 'AB'
    boolean inMST = false; //boolean variable to check if an edge is in the minimum spanning tree
    for(Edge edge: mstEdges){ //iterating through the list of edges in the minimum spanning tree
        if(edge.source == selectedEdge.charAt(0) - 65 && edge.dest == selectedEdge.charAt(1) - 65){ //checking if the source and dest match that of an edge in the MST
            inMST = true; //sets the boolean to true if the edge is in the MST
            break; //stops checking once the variable has been set to true
        }
    }
    if(inMST) {
        displayPanel.highlightEdge(selectedEdge, Color.GREEN); //makes that edge green if it is in the MST
    } else{
        displayPanel.highlightEdge(selectedEdge, Color.RED); //makes the edge red if it is not in the MST
    }
}

```

In the DisplayPanel class, in the paintComponent method, I checked that if the current edge is the edge to be highlighted, it highlights it in the correct colour

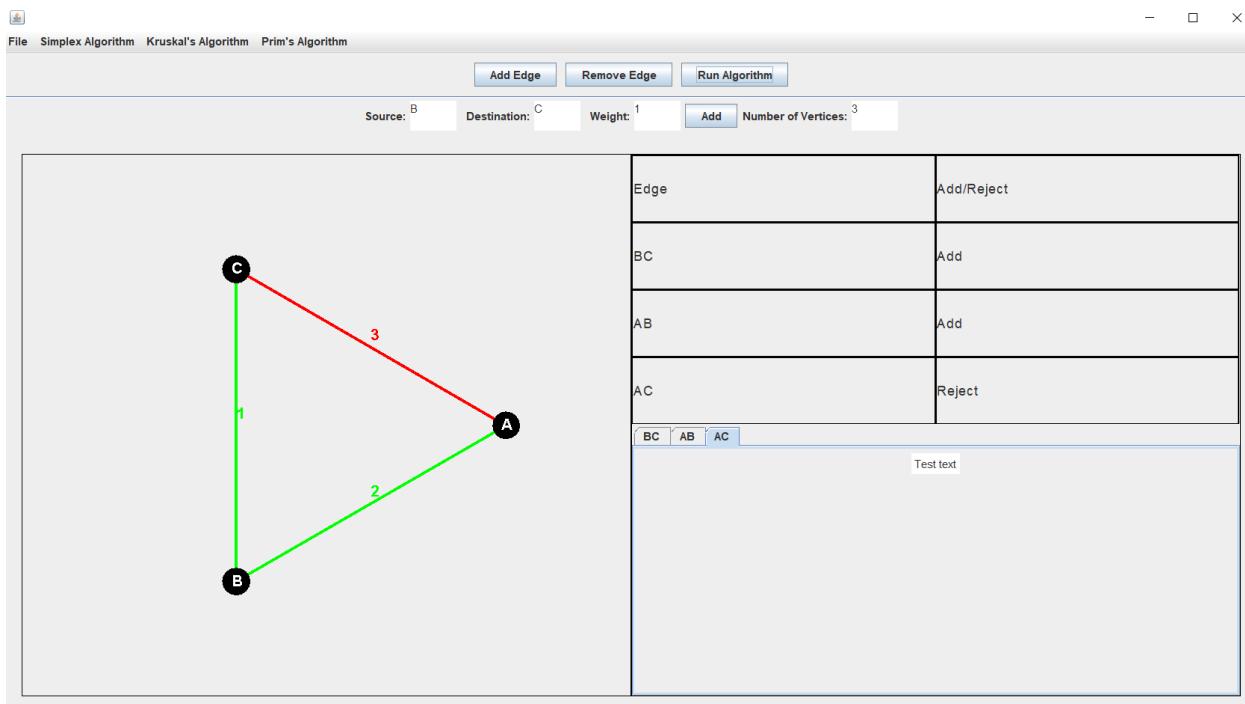
```

private String highlightedEdge = null;
2 usages
private Color edgeColour;

if (edge.equals(highlightedEdge)) {
    g2d.setColor(edgeColour); // highlighted color
} else {
    g2d.setColor(Color.BLACK); // default color
}

```

I then ran this with the same basic 3 edges to check functionality and the following output was produced



Here we can also see that the table with the added and rejected arcs is also functioning correctly.

Currently, the text in the text plane is just acting as a placeholder, now I need to add the actual explanations. Different stages of the algorithm will require explanations, which I have split up and are listed below:

- The first edge added to the minimum spanning tree
- The last edge added to the minimum spanning tree
- The last edge considered from the whole graph
- Any other edges that are added to the minimum spanning tree
- Any other edges rejected from the minimum spanning tree

```
public static JTextPane explainEdge(ArrayList<Edge> mstEdges, ArrayList<Edge> allEdges, Edge edge) {
    JTextPane explanationPane = new JTextPane();
    int total = 0;
    for (Edge mstEdge : mstEdges) {
        total += mstEdge.weight; //summing the total weight of edges in the minimum spanning tree
    }
}
```

First, I summed up the total weight of edges included in the minimum spanning tree to be outputted in the final tab.

Next I added an if statement to check whether the edge passed as a parameter was included in the minimum spanning tree, then covered the explanation for when the edge is in the minimum spanning tree. This does the explanation for the first edge in the minimum spanning tree.

```
if (mstEdges.contains(edge)) { //checking if edge is in minimum spanning tree
    if (edge == allEdges.get(0)) { //the first edge in the minimum spanning tree
        explanationPane.setText("This is the first edge with the lowest weight.");
        explanationPane.setText(explanationPane.getText() + "\nNo cycles are formed by adding this edge to the minimum spanning tree, hence we add it.");
        explanationPane.setText(explanationPane.getText() + "\n\nIn the table of added and rejected edges, this edge is listed as 'Add'.");
        explanationPane.setText(explanationPane.getText() + "\n"); //adding some spacing lines to make text more readable
        explanationPane.setText(explanationPane.getText() + "\n");
        explanationPane.setText(explanationPane.getText() + "Click the next tab to see the next edge!");
    }
}
```

This explains the final edge in the minimum spanning tree.

```
} else if (edge == mstEdges.get(mstEdges.size() - 1)) { //the final edge in the minimum spanning tree
    explanationPane.setText("Adding this edge doesn't form any cycles. Looking at the minimum spanning tree,");
    explanationPane.setText(explanationPane.getText() + "\nall the nodes are connected by edges, so our minimum spanning tree is complete.");
    explanationPane.setText(explanationPane.getText() + "\n"); //adding some spacing lines to make text more readable
    explanationPane.setText(explanationPane.getText() + "\n");
    explanationPane.setText(explanationPane.getText() + "Click the next tab to see the next edge!");
}
```

This explains the final edge in the entire graph if the final edge is included in the minimum spanning tree. Since this is the final tab in the tabbed pane, I added the line that outputs the final weight of the minimum spanning tree, as well as listing each of the edges included in the minimum spanning tree.

```
}else if(edge == allEdges.get(allEdges.size()-1)){ //the final edge in the entire graph
    explanationPane.setText("This is the final edge added to the minimum spanning tree. Looking at the minimum spanning tree,");
    explanationPane.setText(explanationPane.getText() + "\nall the nodes in the graph are connected by edges, so our minimum spanning tree is complete.");
    explanationPane.setText(explanationPane.getText() + "\nIn the table of edges, this edge is listed as 'Add'.");
    explanationPane.setText(explanationPane.getText() + "\n"); //adding some spacing lines to make text more readable
    explanationPane.setText(explanationPane.getText() + "\n");
    explanationPane.setText(explanationPane.getText() + "The final weight of the minimum spanning tree is: " + total);
    explanationPane.setText(explanationPane.getText() + "\nEdges included in the minimum spanning tree:");
    for(int i = 0; i < mstEdges.size(); i++){ //outputting the list of edges added to the minimum spanning tree
        explanationPane.setText(explanationPane.getText() + "\n" + (char)(mstEdges.get(i).source + 65) + "" + (char)(mstEdges.get(i).dest + 65));
    }
}
```

This explains any other edges that are in the minimum spanning tree that aren't the first or last edges in the graph.

```
}else { //other edges in the minimum spanning tree
    explanationPane.setText("We consider what happens to the minimum spanning tree if we add this edge.");
    explanationPane.setText(explanationPane.getText() + "\nNo cycles are formed by adding this edge, so we can add it.");
    explanationPane.setText(explanationPane.getText() + "\nLooking at the table of edges, we see this edge is given 'Add'.");
    explanationPane.setText(explanationPane.getText() + "\n"); //adding some spacing lines to make text more readable
    explanationPane.setText(explanationPane.getText() + "\n");
    explanationPane.setText(explanationPane.getText() + "Click the next tab to see the next edge!");
}
```

If the edge is not in the minimum spanning tree, then the following explanation is produced. This is for when the edge is not in the minimum spanning tree and is the final edge in the graph.

```

}else if(edge == allEdges.size()-1){ //final edge in the graph that is not in the minimum spanning tree
    explanationPane.setText("This is the final edge we consider but the minimum spanning tree is complete, so we reject it.");
    explanationPane.setText(explanationPane.getText() + "\nLooking at the table of edges, this edge is listed as 'Reject'.");
    explanationPane.setText(explanationPane.getText() + "\nThe total weight is: " + total);
    explanationPane.setText(explanationPane.getText() + "\n Edges included in the minimum spanning tree:");
    for(int i = 0; i < mstEdges.size(); i++){
        explanationPane.setText(explanationPane.getText() + "\n" + (char)(mstEdges.get(i).source + 65) + "" + (char)(mstEdges.get(i).dest + 65));
    }
}

```

This final explanation is for when the edge is not the first or last edges and is not included in the minimum spanning tree.

```

}else{ //other edges that are rejected from the minimum spanning tree
    explanationPane.setText("Adding this edge to the minimum spanning tree would form a cycle, therefore we reject it.");
    explanationPane.setText(explanationPane.getText() + "\nOn the table of added and rejected edges, we see that this edge is listed as 'Reject'.");
    explanationPane.setText(explanationPane.getText() + "\n");
    explanationPane.setText(explanationPane.getText() + "\n");
    explanationPane.setText(explanationPane.getText() + "Click the next tab to see the next edge!");
}

```

As shown throughout the code for each of the explanations, I added in some spacing lines to ensure the text isn't clumped together which would make it difficult to follow.

```

explanationPane.setFont(new Font("Arial", Font.PLAIN, 15)); //setting the font size and style for the explanation
explanationPane.setEditable(false); //making it so the user cannot edit the text of the text pane
return explanationPane; //return the text pane

```

Finally, I adjusted the font size and style of the text in the text pane and set the text pane to not be editable to ensure the user doesn't accidentally delete the explanation text. I then returned the text pane from the function.

In the runKruskalsAlg method, I can now change the line that had the placeholder text to contain the text pane with the explanation. I passed in the list of minimum spanning tree edges, list of all edges in the graph, and the current graph being considered.

```

JTextPane explanation = explainEdge(mstEdges, edgeList, edge); //adding the text pane to contain the explanation of each edge being added/rejected
edgePanel.add(explanation); //add the text pane to the panel for the tabbed pane

```

Now that the user can enter their own graph, see it built up and move it around, run the algorithm and have it explained, I just need to implement the input validation features to ensure that any unexpected inputs can be handled allowing the user to continue using the system. The first method isn't direct input validation, but it ensures that if the user enters an incorrect edge by accident, they can remove this and continue entering edges. This feature is the 'Remove Edge' button, which will take in similar inputs to the 'Add Edge' button (source and destination), however it will not take in a weight, as another form of input

validation will ensure that multiple edges with the same source and destination cannot be entered, hence there is no validation required to differentiate the edges by weight.

I first need to add functionality to the 'Remove Edge' button so it can display the source and text area. An important feature here is to ensure that all other components e.g. the displayed graph and the row of buttons all remain the same when the 'Remove Edge' button is selected. I created the removeEdgeButton in the constructor to be added to the row of buttons when the user selects 'Remove Edge'

```
removeEdgeButton = new JButton( text: "Remove");
```

```
public void removeFromGraph() {
    kruskalAdd.removeAll();
    JTextArea removeSourceTA = new JTextArea( rows: 2, columns: 5); //adding in source text area for edge to be removed
    JTextArea removeDestTA = new JTextArea( rows: 2, columns: 5); //adding in destination text area for edge to be removed
    JLabel removeSourceLabel = new JLabel( text: "Source: "); //label to indicate purpose of removeSourceTA
    JLabel removeDestLabel = new JLabel( text: "Destination: "); //label to indicate purpose of removeDestTA

    kruskalAdd.add(removeSourceLabel); //adding the various components to the kruskalAdd panel
    kruskalAdd.add(removeSourceTA);
    kruskalAdd.add(Box.createVerticalStrut( height: 10)); // 10 pixels of vertical spacing
    kruskalAdd.add(removeDestLabel);
    kruskalAdd.add(removeDestTA);
    kruskalAdd.add(removeEdgeButton); //add the remove edge button
    revalidate(); //refresh the GUI
    repaint();
}
```

This method only clears the row of text areas from the kruskalAdd panel, ensuring everything else remains the same and no components are unexpectedly removed or reset.

The remaining forms of input validation were highlighted in the Design table, however, they have been copied below for better access

Validation Required	Reason
The user must enter source and destination nodes as a single capital letter	Since the ASCII values of characters are used throughout the Kruskal's algorithm methods, it will be vital that the user enters nodes in the correct format of a single capital letter.

The user must enter weight as a numerical value	The list of weights in the graph is used to determine the edges in the minimum spanning tree. Having an incorrect input such as a symbol would result in errors when trying to run the algorithm. These errors may not crash the program, however, when the input is converted to an integer data type in Java, it may take the ASCII value of the symbol which would result in incorrect solutions.
When removing an edge, the user must enter source and destination as single upper-case characters.	The logic for removing an edge from a graph, will check the inputted edge against edges existing in the graph. To ensure the edge is correctly identified in the list and then removed, there must be consistency within the inputs for edges.
When removing an edge, it must be checked that the inputted edge exists in the graph.	To ensure there are no errors in removing edges from the graph, the logic for removing an edge will ensure that the inputted edge exists in the graph before removing it from any components.
It must be checked that the user has entered a connected graph to perform the algorithm on.	A connected graph is a graph which contains a path from any node to any other by crossing the edges connecting nodes. Kruskal's algorithm can be performed on unconnected graphs, resulting in a Minimum Spanning Forest, however, this is not tested by the Edexcel A-Level Further Maths questions, which only require the algorithm to be performed on a connected graph.
Checking that a user hasn't entered the same edge twice in a graph.	Edexcel A-Level Further Maths questions only require students to perform Kruskal's algorithm on a connected graph where there is a

	single edge between two nodes, hence a check will be made to ensure the user enters unique edges.
--	---

To ensure the user enters the source and destination node as a single capital letter when adding an edge to the graph, I implemented a try catch loop that checks if the input is within the range 'A' to 'Z'. Since there is no direct exception for capital letters, the method throws a general exception with a custom message, which is then displayed on the GUI.

```
public void addEdgeGraph() {
    try {
        String source = sourceTA.getText(); //extracting the inputted text from the text areas
        if (!source.matches("[A-Z]")) { //checking source node is a single capital letter
            throw new IllegalArgumentException("Please enter source node as a single capital letter."); //error message to indicate mistake to user
        }
        kruskalsError.setText(""); //error message set to empty string if input is valid
        char sourceCH = source.charAt(0); //converting the string inputs to characters

        String dest = destTA.getText();
        if (!dest.matches("[A-Z]")) { //checking destination node is a singel capital letter
            throw new IllegalArgumentException("Please enter destination node as a single capital letter."); //error message to indicate mistake to user
        }
        kruskalsError.setText(""); //error message set to empty string if input is valid
        char destCH = dest.charAt(0);
    }
}
```

For the input into the weight text area, I am able to use a number format exception, which expects a numerical input into the text area and throws an exception if it receives anything else.

```
String weight = weightTA.getText();
int weightInt;
try {
    weightInt = Integer.parseInt(weight);
    kruskalsError.setText(""); //error message is set to empty string if valid input entered
} catch (NumberFormatException e) { //throws exception if input is not a number
    throw new IllegalArgumentException("Weight must be a valid number."); //error message text is changed to indicate mistake to user
}
```

The exception is handled in the catch section and the text of the error label is changed to whatever the message indicated by the error is:

```
}catch(IllegalArgumentException e){
    kruskalsError.setText(e.getMessage());
}
newvalidate();
```

The next method I developed is the remove() method which is called when the removeEdgeButton is pressed. Since this takes inputs in a similar way to the

addToGraph method, I decided to implement it after developing the input validation for the add method as similar logic can be implemented in the remove() method to check that the input into the source and destination text areas are single capital letters.

```
public void remove() {
    try {
        String source = sourceTA.getText().trim(); // get the source node from the text area
        if (!source.matches("^[A-Z]$")) { //checking that input is single capital letter
            throw new IllegalArgumentException("Please enter a source node as a single capital letter.");
        }

        String dest = destTA.getText().trim(); // get the destination node from the text area
        if (!dest.matches("^[A-Z]$")) { //checking that input is single capital letter
            throw new IllegalArgumentException("Please enter a destination node as a single capital letter.");
        }
    }
```

Next, I implemented a loop to iterate through the toAdd list and check for a sequence matching that inputted edge, so if the user entered 'A' and 'B' wanting the edge 'AB' to be removed, the loop will look through the toAdd list looking for a sequence where 'A' is followed by 'B' or where 'B' is followed by 'A'. A possible error that arose when developing this is an incorrect section of the toAdd list being removed e.g. if it contains 'B', 'C', 'D', 'E', 'C', 'D' (indicating the edges BC, DE, and CD). If the user wanted to remove 'CD' it may incorrectly identify the part where the edge BC and DE meet 'B', 'C', 'D', 'E'. However, I realised that this error could not occur since the weight of each edge is added after the nodes so the toAdd list actually looks like 'B', 'C', 6, 'D', 'E', 7, 'C', 'D', 8, avoiding any overlap between edges.

```
char sourceChar = source.charAt(0); //remove the edge from the 'toAdd' list
char destChar = dest.charAt(0);

for (int i = 0; i < toAdd.size(); i += 3) {
    if ((toAdd.get(i) == (int) sourceChar && toAdd.get(i + 1) == (int) destChar) || (toAdd.get(i) == (int) destChar && toAdd.get(i + 1) == (int) sourceChar)) {
        toAdd.remove(index i + 2); //remove weight
        toAdd.remove(index i + 1); //remove destination
        toAdd.remove(i); //remove source
        break; //exit loop after removing the edge
    }
}
```

The next section of input validation is also part of the remove method and it checks whether the edge that the user entered to be removed is actually part of the graph. To do this, I formed a pair by combining the source and destination node e.g. 'A' and 'B' become 'AB', then checking whether the list of edges contains this pair. If not, then another general error is thrown with a custom message to indicate the user's mistake.

```

String pair = source + dest; // construct the edge string
if (!displayPanel.edges.contains(pair)) { //checking if inputted edge is in the graph
    throw new IllegalArgumentException("The specified edge does not exist in the graph.");
}

```

I can now finish the try-catch block by setting the text of the error message based on what the exception thrown was.

```

    kruskalsError.setText(""); // clear any previous error messages
} catch (IllegalArgumentException e) {
    kruskalsError.setText(e.getMessage()); // display an error message to the user
}

```

The next form of input validation is more complex and it involves checking that the user has entered a connected graph to perform the algorithm on, meaning that all nodes/vertices from the other can be reached from any other node/vertex by following edges between them. To do this, I implemented a well-known graph algorithm called a depth first search. This will visit each node in the graph starting from the first node entered and will set the visited status of that node to true. When iterating through the list of nodes, if any node has 'false' indicating it wasn't visited during the depth first search then the graph is not connected.

```

private void dfs(int node, boolean[] visited, Kruskals graph){ //depth first search method
    visited[node] = true;
    for(Edge edge: graph.edges){ //going through each edge in the graph
        if(edge.source == node && !visited[edge.dest]){ //checking whether the source has been visited and the destination hasn't
            dfs(edge.dest, visited, graph); //recursively doing depth first search starting from the destination
        } else if(edge.dest == node && !visited[edge.source]){ //checking whether the destination has been visited and the source hasn't
            dfs(edge.source, visited, graph); //recursively doing depth first search starting from the source
        }
    }
}

```

The depth first search begins at the first node entered in the graph and recursively visits each subtree.

```

private boolean isConnected(Kruskals graph, int numVertices) {
    boolean[] visited = new boolean[numVertices];
    int startNode = Integer.MAX_VALUE; //assigns the largest value to start node
    for (Edge edge : graph.edges) {
        startNode = Math.min(startNode, Math.min(edge.source, edge.dest)); //getting the first node in the graph to begin the dfs
    }
    dfs(startNode, visited, graph); //doing a depth first search starting from the first node in the graph

    for (Edge edge : graph.edges) {
        if (!visited[edge.source] || !visited[edge.dest]) { //checking if the source and node of each edge has been visited
            return false; //if not then the graph is not connected
        }
    }
}

return true;
}

```

This method checks that the graph is connected by adding either true or false to the visited[] list. It then iterates over this list and if the source and destination node of an edge haven't been visited, then it suggests the graph is not connected

I then called this method in the runKruskalsAlg method to ensure that if the graph is not connected, it is noticed before the algorithm is run and all the panels of explanation are displayed.

```

if (!isConnected(graph, graphSize)) { //checking if the graph is connected or not
    kruskalsError.setText("Please ensure the graph is connected."); //setting the error message text to indicate to the user
    return; // exit the method if the graph is not connected
}

```

The final form of input validation ensures that the user doesn't try to enter the same edge twice into the graph. To do this, before adding the edge to the graph, I checked whether the edge or the reverse of the edge e.g. 'AB' and 'BA' are in the graph already. If they are, then a suitable error message is displayed to the user.

```

String pair = source + dest; //formats the source and destination as one string
String reversePair = dest + source; //making the reverse of the edge e.g. 'AB' becomes 'BA'
if (displayPanel.edges.contains(pair) || displayPanel.edges.contains(reversePair)) { //checking if the user has already entered the inputted edge into the graph
    throw new IllegalArgumentException("This edge already exists in the graph."); //setting the error message text saying that they have already entered the edge
}

```

The last method to develop in this iteration is the reset graph feature. This allows the user to clear the graph they have started making if they decide they want to start with a new graph. As mentioned in the Design, the button to reset the graph will be placed on the left of the 'Add Edge' button to ensure that the user doesn't accidentally click the 'Reset' button when attempting to press the 'Run' button which is on the right-hand side of the button panel. This involves slightly altering the order in which components are added to the flow layout panel graphButtonPanel, ensuring the reset button is the first to be added.

```

resetGraph = new JButton( text: "Reset");

public void graphSettings() {
    getContentPane().removeAll(); //removes all current components from the window
    promptPanel.removeAll();
    mainPanel = new JPanel(new BorderLayout()); //creates a new panel with a border layout
    graphButtonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, hgap: 10, vgap: 10)); // a new panel for the three buttons
    graphButtonPanel.add(resetGraph); //adding the row of buttons with the reset graph button first
    graphButtonPanel.add(addEdge);
    graphButtonPanel.add(removeEdge);
    graphButtonPanel.add(runAlgorithm);
    graphButtonPanel.add(kruskalsError);
}

```

I then developed the method called `resetGraphDisplay`, with a simple function of clearing all the arraylists from the `DisplayPanel` class, as well as the list `toAdd` which contained information about each of the edges added to the graph before refreshing the GUI

```

public void resetGraphDisplay(){
    displayPanel.getEdges().clear(); //clear the arraylists storing information about the edges
    displayPanel.getEdgeWeights().clear();
    displayPanel.getEdgeColourList().clear();
    displayPanel.getNodeLabels().clear(); // clear the nodes and their positions in the DisplayPanel
    displayPanel.getNodePositions().clear();

    toAdd.clear(); // clear the edges stored in 'toAdd'

    displayPanel.revalidate(); // refresh the GUI to remove the displayed graph
    displayPanel.repaint();

    kruskalsError.setText(""); // clear any error messages
}

```

With the creation of the ‘Reset’ method, the development for Kruskal’s algorithm being performed on a graph designed by the user with full explanation in the user interface is now complete. To ensure that the various components developed in this iteration are fully functional, I can now move onto the testing section, in which the components and algorithm will be tested against the various test cases outlined in the design test table.

# Testing

## Iteration 1: GUI Mockup

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:

[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==)

To test full functionality of the initial GUI created as my first iteration, the following test plan has been created. The evidence of each outcome is outlined underneath the test table.

Test Number	Test Type	Expected Outcome	Actual Outcome	Remedial Actions	Second test (after remedial actions)
1) The window of the GUI opening when running the program	Normal	The GUI window opening once the program is run	The window successfully opened presenting all user options.	No remedial actions required	Testing this function again, the window successfully opened
2) The Simplex Algorithm Introduction button on the 'home' screen should clear the window when clicked, allowing information to be	Normal	The Simplex introduction button should clear the window for information to be presented.	Pressing the 'Simplex Algorithm Introduction' button did not clear the window, instead the same screen remained.	Added two lines to update the changes, recalculating the layout and adding the changes to the window. These changes were made to the other	When pressing the introduction button, all objects were successfully removed from the window.

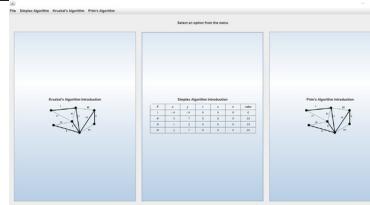
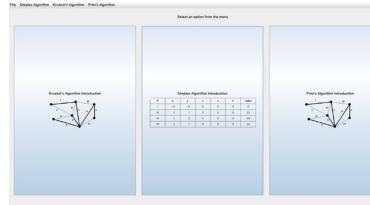
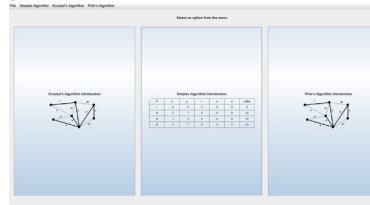
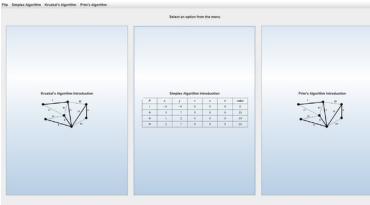
presented, introducing the Simplex algorithm.				two buttons as well	
3) The Prim's Algorithm Introduction button on the 'home' screen should clear the window when clicked, allowing information to be presented, introducing Prim's algorithm.	Normal	The Prim's introduction button should clear the window for information to be presented.	Having added the revalidate() and repaint() lines to each button, the window was successfully cleared of all components.	No remedial actions required	Testing again, this function still worked, clearing the window.
4) The Kruskal's Algorithm Introduction button on the 'home' screen should clear the window when clicked, allowing information	Normal	The Kruskal's introduction button should clear the window for information to be presented.	Completing the remedial actions for the Simplex button and applying them to the others ensure that this function worked as intended	No remedial actions needed	Testing this the second time gave the same successful result.

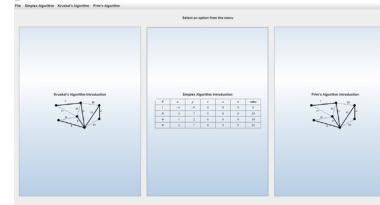
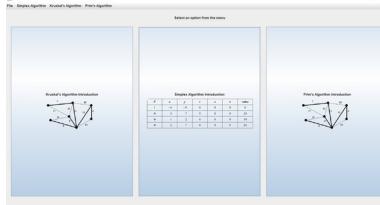
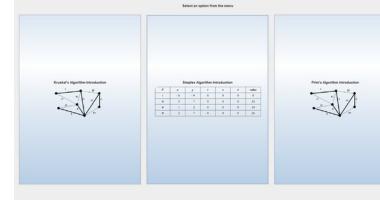
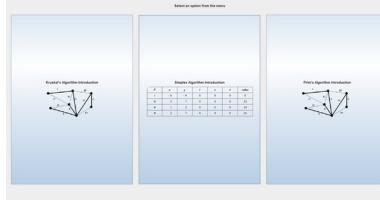
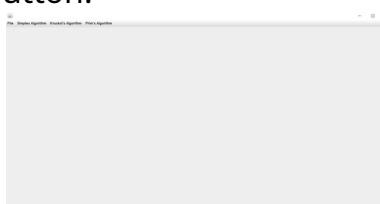
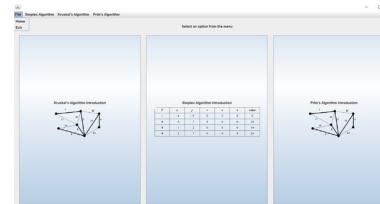
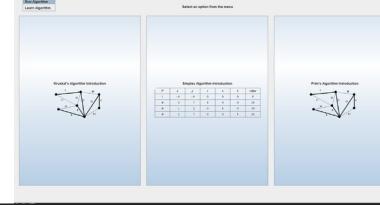
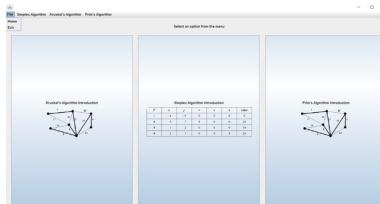
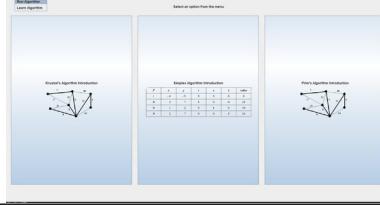
to be presented, introducing Kruskal's algorithm.			and cleared the window.		
5) The menu items added to each menu bar option should be displayed when selected.	Normal	When clicking each option in the menu bar, the added menu items should be displayed	Each menu item is successfully displayed when the respective option is clicked	No remedial actions required	Testing each menu option again, every menu item appeared again, proving the test successful.
6) When the user clicks the menu item inside of an option, the window should clear allowing room for information to be presented. This test is applicable for: -Run Algorithm -Learn Algorithm In each of	Normal	When clicking each menu item inside of an option, the window should clear allowing room for information to be presented	The window did not clear, all objects in the window remained after the menu items were clicked.	Like test number 2, adding in the revalidate() and repaint() methods ensured the changes to the window were being acknowledged	When pressing each menu item, the rest of the window now clears allowing information to be presented.

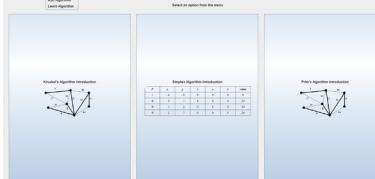
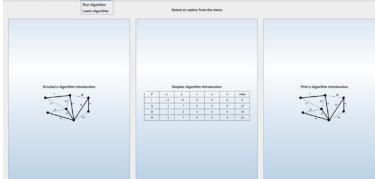
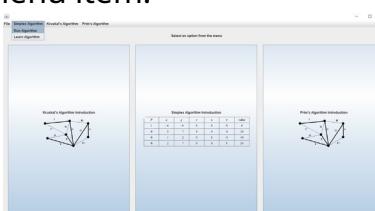
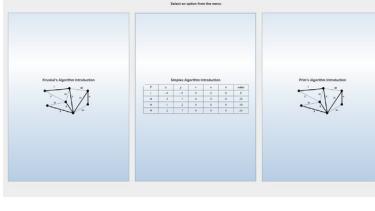
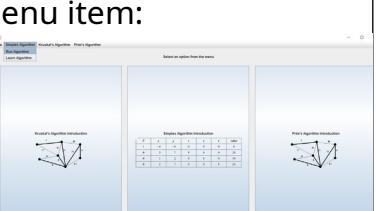
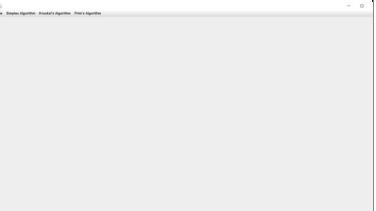
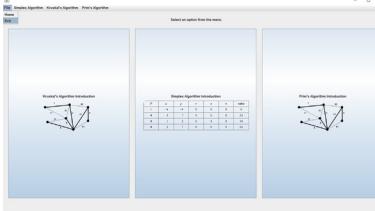
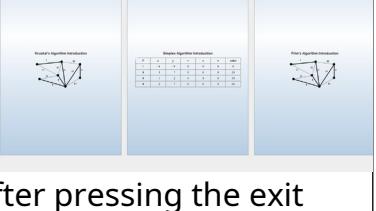
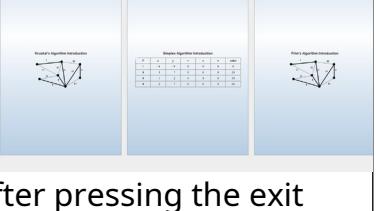
the Algorithm menu bar options.					
7) When the user selects the 'Exit' menu item inside the 'File' option, the window should close.	Normal	When selecting the 'Exit' menu item inside the 'File' option, the window should close down.	The window successfully closed down when the exit button was pressed	No remedial actions required	When testing the 'exit' function again, the window still closed.
8) When the user selects the 'Home' menu item inside the 'File' option, the window should revert to its original state from any other state	Normal	When selecting the 'Home' menu item inside the 'File' option, the window should revert to its original state from any other state	By first selecting the Prim's introduction button to clear the screen, then the home button, the test failed, as the screen remained clear from pressing the Prim's introduction button.	I created a new procedure called 'originalWin dow()' that contains the code to create the button panel and each of the introduction buttons. This procedure was then called in the constructor for the original window	When selecting the 'Home' option from a screen that is not in the original state, the original objects (panel and algorithm introduction buttons) reappear.

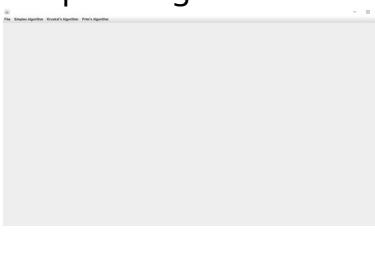
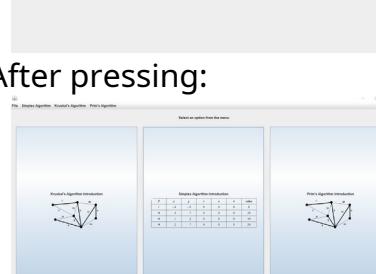
				and added as part of the event for pressing home.	
--	--	--	--	---	--

## Evidence of Test results

Test Number	First Outcome	Second Outcome (Second test or after remedial actions)
1		
2	<p>Before clicking Simplex button</p>  <p>After clicking Simplex button:</p> 	<p>Before clicking Simplex button</p>  <p>After clicking Simplex button:</p> 
3	<p>Before clicking Prim's button</p>	<p>Before clicking Prim's button</p>

	 <p>After clicking Prim's button:</p>	 <p>After clicking Prim's button:</p>
4	<p>Before clicking Kruskal's button</p>  <p>After clicking Kruskal's button:</p> 	<p>Before clicking Kruskal's button</p>  <p>After clicking Kruskal's button:</p> 
5	<p>Each menu item displayed its options when clicked:</p>  	<p>Testing each menu option again produced the same results</p>  

	 	 
6	<p>Each menu item produced the same result:</p> <p>Before clicking each menu item:</p>  <p>After clicking each menu item:</p> 	<p>After adding the revalidate() and repaint() methods:</p> <p>Before clicking each menu item:</p>  <p>After clicking each menu item:</p> 
7	<p>When pressing the exit button, the window as expected:</p> <p>Before pressing the exit button:</p>  <p>After pressing the exit button:</p> 	<p>Testing again to ensure functionality:</p> <p>Before pressing the exit button:</p>  <p>After pressing the exit button:</p> 

8	<p>Initially, the home button didn't have functionality when pressed  Before pressing:</p>  <p>After pressing:</p> 	<p>When I added the originalWindow() method, the window restored back to its original state:  Before pressing:</p>  <p>After pressing:</p> 

## Remedial Actions evidence

Test 2/Test 6

The only remedial action needed for this error was to just refresh the GUI after making any changes. This ensures that the various panels are updated to show the accurate information based on the changes made, such as clicking a button

```
revalidate(); //refreshing the GUI
repaint();
```

Testing this was successful, as shown in the second test column of the outcome table.

Test 8

The fix for this error involved creating a new method that can be called whenever the original state of the window needs to be brought back. This places all the

components like the 3 introductory buttons and the prompt label on the screen. This method is called when first running the program, and when the user presses the home button in the 'File' menu item:

```

private void originalWindow(){ // procedure called in the constructor to generate initial panels
    promptPanel = new JPanel(new BorderLayout());
    prompt = new JLabel( text: "Select an option from the menu", SwingConstants.CENTER);
    promptPanel.add(prompt, BorderLayout.CENTER);
    promptPanel.setBorder(BorderFactory.createEmptyBorder( top: 20, left: 0, bottom: 20, right: 0)); //defining the border size for the panel in the north section

    buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout( rows: 1, cols: 3, hgap: 20, vgap: 50)); //splitting the center panel into 3 sections for 3 buttons
    buttonPanel.setBorder(BorderFactory.createEmptyBorder( top: 5, left: 20, bottom: 60, right: 20)); //defining the border size for the button panel in the center section

    //creating the button for Kruskal's algorithm introduction, with image, label and text positioning defined
    ImageIcon kruskalsIcon = new ImageIcon( filename: "C:\\\\Users\\\\averm\\\\Pictures\\\\kruskalsimage.png");
    introKruskals = new JButton();
    introKruskals.setText("Kruskal's Algorithm Introduction");
    introKruskals.setHorizontalTextPosition(JButton.CENTER);
    introKruskals.setVerticalTextPosition(JButton.TOP);
    introKruskals.setFocusable(false);
    introKruskals.setIcon(kruskalsIcon);
    buttonPanel.add(introKruskals);

    //creating the button for the simplex algorithm introduction, with image, label and text positioning defined
    ImageIcon simplexIcon = new ImageIcon( filename: "C:\\\\Users\\\\averm\\\\Pictures\\\\simplex.png");
    introSimplex = new JButton();
    introSimplex.setText("Simplex Algorithm Introduction");
    introSimplex.setHorizontalTextPosition(JButton.CENTER);
    introSimplex.setVerticalTextPosition(JButton.TOP);
    introSimplex.setFocusable(false);
    introSimplex.setIcon(simplexIcon);
    buttonPanel.add(introSimplex);

    //creating the button for prim's algorithm introduction, with image, label and text positioning defined
    ImageIcon primsIcon = new ImageIcon( filename: "C:\\\\Users\\\\averm\\\\Pictures\\\\kruskalsimage.png");
    introPrims = new JButton();
    introPrims.setText("Prim's Algorithm Introduction");
    introPrims.setHorizontalTextPosition(JButton.CENTER);
    introPrims.setVerticalTextPosition(JButton.TOP);
    introPrims.setFocusable(false);
    introPrims.setIcon(primsIcon);
    buttonPanel.add(introPrims);
}

```

Testing these remedial actions proved successful in returning the window to its original state once the home option was selected, shown in the second test column of the output table.

Having tested each component according to the outlined test plan and carrying out remedial actions for the 3 failed tests, the first iteration of the GUI now has full functionality as expected for the GUI 'shell'.

## Iteration 2: Simplex Algorithm

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

It is necessary to ensure that the code I have developed is functional for a variety of inputs. As outlined by the test plan, this will involve using past questions to ensure functionality, as well as various erroneous inputs, such as integers or incorrect number of inputs. Being able to manage erroneous inputs at this stage will be a necessary feature when later integrating the Simplex algorithm feature into the GUI, ensuring that the system is able to handle a variety of unexpected inputs from the user. This justifies testing basic forms of input validation in the terminal so that in a future iteration, they can be adjusted to work within the user interface and handle more complex erroneous inputs. Another necessary test is that the same question inputted provides the same output. For this initial version of the Simplex Algorithm, all outputs are given in the form  $x_1, x_2, x_3$  etc and all slack variables are outputted as  $s_1, s_2, s_3$  etc, however when implementing this into the GUI in a future iteration, I will ensure that the correct variable identifiers are used to make the system as accurate as possible. For this iteration, just to test mathematical accuracy, the variables have just been differentiated using numbers e.g.  $x_1, x_2, x_3$ .

Test Number	Test Type	Expected Outcome	Actual Outcome	Remedial Actions	Second Test (After remedial actions or double checking to ensure functionality)
1	Normal	The correct	The correct	No	Testing

Refer to Figure 1a below		values of the final variables and maximised profit must be outputted in the terminal. These are $x_1 = 0, x_2 = 3, x_3 = 5, P = 38$	values were outputted in the terminal upon entering the inequalities and objective function. These values match the mark scheme, which can be found below the evidence table	remedial actions required	again with the same inputted inequalities, the same correct output was produced in the terminal with the correct final values of the variables and the maximised profit value, deeming this test successful
2 Refer to Figure 2a below	Normal	The right values of the final variables and maximised profit value must be outputted in the terminal. $x_1 = 0, x_2 = 40, x_3 = 10, P = 260$	The correct final values of the variables and the final maximised profit were outputted in the terminal as expected. These values match the mark	No remedial actions required	Once again, the same correct final values of each variable and the final maximised profit were outputted in the terminal, deeming this test successful

			scheme, which can be found below the evidence table		
3 Refer to Figure 3a below	Normal	In the terminal, the correct final values of all the variables and the maximised profit should be outputted. These are: $x_1 = 2, x_2 = 1, x_3 = 0, s_3 = 1, P = 11$	The outputted numbers were very close to, but not exactly the expected output. This is shown in the evidence table. The outputted values didn't quite match the mark scheme for the question, shown below the evidence table.	To overcome this issue, I first multiplied each number by 100000, rounded the number to the nearest integer before dividing by 100000. If I instead rounding straight away, solutions like 1.9 would be incorrectly rounded up	Testing with the same inequalities after implementing the remedial actions, the correct values were now outputted in the terminal, deeming this test now successful
4 Refer to Figure 4a	Normal	The correct final values of the variables should be	The correct values were outputted in the terminal	No remedial actions required	Testing again with the same inequalities produced

		<p>outputted in the terminal, alongside the maximised profit after all the iterations of the algorithm. These values are:</p> $x_1 = 77/4$ , $x_2 = 0$ , $x_3 = 57/4$ , $x_4 = 0$ , $s_1 = 33$ , $s_2 = 0$ , $s_3 = 109/4$ , $P = 105.5$	<p>when entering the inequalities and objective function and pressing enter. These values match the mark scheme, which can be found below the evidence table</p>		<p>the same result with the correct value of the variables and the final maximised profit value, matching the mark scheme, hence this test was successful.</p>
5	Erroneous	<p>The error should be recognised and instead of throwing an error, a message should be outputted in the terminal telling the user to please enter integer values only.</p>	<p>An exception was thrown when trying to input non-integer values into the terminal to run the algorithm one. This caused an error with an input mismatch</p>	<p>The error emerged when entering 'a' as the number of variables, trying to perform any calculations with this value would not work, throwing the input</p>	<p>I tested again after implementing the remedial actions As evident in the outcome table, when the input is a number, the user is made aware of their incorrect</p>

			exception. I need to be able to catch this error and respond to the user with a message indicating to them that they need to enter integers only.	mismatch exception. To fix this, I added input validation to each subroutine in the Solve class.	input and prompted to make the input again.
6	Erroneous	The error should be recognised and instead of throwing an error, a message should be outputted in the terminal telling the user to please enter the correct number of coefficients	While an optimal value was outputted, this was for a different table. When entering more than the necessary number of coefficients , the prompt to enter coefficients cut short thinking I had already entered all the	To fix this error, I had to find a way to count the number of entered values in a specific line. This involved splitting up the line e.g. '1 2 3 4 5' into '1', '2', '3', '4', '5' so that the number of constraints could be counted. I used the line.trim()	I repeated this for the objectives input as well:  The 'Invalid number of coefficients' line would be outputted directly after the 'enter coefficients line' without letting the user enter anything. To fix this, I added

			<p>required numbers, forming a table with them that was incorrect.</p>	<p>method splitting whenever a space occurred to count the number of coefficients and compare it to the number of variables entered. I swapped the input validation for the constraints to a try catch loop and introduced a new array part, to store the coefficients .</p>	<p>another scanner.nextLine() to clear the input:</p> <p>The output now works as intended, shown in the evidence table</p>
--	--	--	--	--	--

### **Outcome Table**

Test Number	Actual Outcome	Remedial Actions	Second Test (After remedial actions or double checking to ensure functionality)

1)	<pre> Enter number of variables: 3 Enter number of constraints: 2 Enter the coefficients of the constraints and include the value: 1 2 0 6 0 1 3 24 Enter coefficients of the objective function: 3 0 4 Initial Tableau:   1.00   2.00   0.00   1.00   0.00   6.00  5.00   3.00   3.00   0.00   1.00  24.00  -5.00  -6.00  -4.00   0.00   0.00   0.00  The optimal solution is: x1 = 0.0 x2 = 3.0 x3 = 5.0 s1 = 0.0 s2 = 0.0 Optimal value: 38.0 </pre>	No remedial actions required	<pre> Enter number of variables: 3 Enter number of constraints: 2 Enter the coefficients of the constraints and include the value: 1 2 0 6 0 1 3 24 Enter coefficients of the objective function: 3 0 4 Initial Tableau:   1.00   2.00   0.00   1.00   0.00   6.00  5.00   3.00   3.00   0.00   1.00  24.00  -5.00  -6.00  -4.00   0.00   0.00   0.00  The optimal solution is: x1 = 0.0 x2 = 3.0 x3 = 5.0 s1 = 0.0 s2 = 0.0 Optimal value: 38.0 </pre>
2)	<pre> Enter number of variables: 3 Enter number of constraints: 2 Enter the coefficients of the constraints and include the value: 1 2 2 100 1 0 4 48 Enter coefficients of the objective function: 3 4 10 Initial Tableau:   1.00   2.00   2.00   1.00   0.00   100.00  1.00   0.00   4.00   0.00   1.00   40.00  -3.00  -4.00  -10.00   0.00   0.00   0.00  The optimal solution is: x1 = 0.0 x2 = 40.0 x3 = 10.0 s1 = 0.0 s2 = 0.0 Optimal value: 260.0 </pre>	No remedial actions required	<pre> Enter number of variables: 3 Enter number of constraints: 2 Enter the coefficients of the constraints and include the value: 1 2 2 100 1 0 4 48 Enter coefficients of the objective function: 3 4 10 Initial Tableau:   1.00   2.00   2.00   1.00   0.00   100.00  1.00   0.00   4.00   0.00   1.00   40.00  -3.00  -4.00  -10.00   0.00   0.00   0.00  The optimal solution is: x1 = 0.0 x2 = 40.0 x3 = 10.0 s1 = 0.0 s2 = 0.0 Optimal value: 260.0 </pre>
3)	<pre> Enter number of variables: 3 Enter number of constraints: 3 Enter the coefficients of the constraints and include the value: 3 4 5 10 1 3 10 5 1 5 0 1 Enter coefficients of the objective function: 3 5 2 Initial Tableau:   3.00   4.00   5.00   1.00   0.00   0.00   0.00   10.00  1.00   3.00   10.00   0.00   1.00   0.00   5.00  1.00  -2.00   0.00   0.00   1.00   1.00  -3.00  -5.00  -2.00   0.00   0.00   0.00   0.00  The optimal solution is: x1 = 1.999999999999998 x2 = 1.0000000000000002 x3 = 0.0 s1 = 0.0 s2 = 0.0 s3 = 0.0 Optimal value: 11.0 </pre>	Evidenced below outcome table	<pre> Enter number of variables: 3 Enter number of constraints: 3 Enter the coefficients of the constraints and include the value: 3 4 5 10 1 3 10 5 1 5 0 1 Enter coefficients of the objective function: 3 5 2 Initial Tableau:   3.00   4.00   5.00   1.00   0.00   0.00   0.00   10.00  1.00   3.00   10.00   0.00   1.00   0.00   5.00  1.00  -2.00   0.00   1.00   0.00   1.00  -3.00  -5.00  -2.00   0.00   0.00   0.00   0.00  The optimal solution is: x1 = 2.0 x2 = 1.0 x3 = 0.0 s1 = 0.0 s2 = 0.0 s3 = 1.0 Optimal value: 11.0 </pre>
4)	<pre> Enter number of variables: 4 Enter number of constraints: 4 Enter the coefficients of the constraints and include the value: 1 3 1 45 2 1 2 3 47 1 1 2 3 47 1 1 2 2 27 Enter coefficients of the objective function: 4 1 2 3 Initial Tableau:   1.00   4.00   1.00   1.00   0.00   0.00   0.00   95.00  2.00   1.00   2.00   3.00  47.00   1.00   0.00   67.00  1.00   3.00   2.00   2.00  75.00   0.00   1.00   75.00  3.00   2.00   1.00   2.00  72.00   0.00   1.00   72.00  -4.00   3.00  -2.00  -3.00   0.00   0.00   0.00   0.00 </pre> <p>The optimal solution is:</p> <p>x1 = 19.25  x2 = 0.0  x3 = 14.25  x4 = 0.0  s1 = 0.8  s2 = 0.0  s3 = 27.25  s4 = 105.5  Optimal value: 105.5</p>	No remedial actions required	<pre> Enter number of variables: 4 Enter number of constraints: 4 Enter the coefficients of the constraints and include the value: 1 3 1 45 2 1 2 3 47 1 1 2 3 47 1 1 2 2 27 Enter coefficients of the objective function: 4 1 2 3 Initial Tableau:   1.00   4.00   1.00   1.00   0.00   0.00   0.00   95.00  2.00   1.00   2.00   3.00  47.00   1.00   0.00   67.00  1.00   3.00   2.00   2.00  75.00   0.00   1.00   75.00  3.00   2.00   1.00   2.00  72.00   0.00   1.00   72.00  -4.00   3.00  -2.00  -3.00   0.00   0.00   0.00   0.00  The optimal solution is: x1 = 19.25 x2 = 0.0 x3 = 14.25 x4 = 0.0 s1 = 0.8 s2 = 0.0 s3 = 27.25 s4 = 105.5 Optimal value: 105.5 </pre>

5)	<pre> Enter number of variables: Exception in thread "main" java.util.InputMismatchException: Create breakpoint     at java.base/java.util.Scanner.throwFor(Scanner.java:939)     at java.base/java.util.Scanner.next(Scanner.java:1524)     at java.base/java.util.Scanner.nextInt(Scanner.java:2258)     at java.base/java.util.Scanner.nextInt(Scanner.java:2212)     at Solve.main(Solve.java:7)  Process finished with exit code 1 </pre>	<b>Evidenced below outcome table</b>	<pre> Invalid input. Please enter only numbers. Enter number of variables:   Invalid input. Please enter only numbers. Enter number of variables:   Enter number of constraints:   Invalid input. Please enter only numbers. Enter number of constraints:   Enter the coefficients of the constraints and include the value:   Invalid input. Please enter only numbers.  </pre>
6)	<pre> Enter number of variables: Enter number of constraints:   Enter the coefficients of the constraints and include the value:   1 2 3 4 1 2 3 1 2 3 2 3 Enter coefficients of the objective function:   1 2 3 Initial Tableau:    1.00   3.00   2.00   1.00   0.00   0.00   4.00   1.00   2.00   3.00   1.00   1.00   0.00   1.00   3.00   8.00   2.00   3.00   0.00   1.00   3.00  -1.00  -2.00  -3.00   0.00   0.00   0.00   0.00  The optimal solution is: x1 = 0.0 x2 = 0.0 x3 = 0.333333 s1 = 1.0 s2 = 1.0 s3 = 2.333333 Optimal value: 1.0  Process finished with exit code 0 </pre>	<b>Evidenced below outcome table</b>	<pre> Enter number of variables:   Enter number of constraints:   Enter the coefficients of the constraints and include the value:   Invalid number of coefficients entered. Please enter again starting from the beginning.  </pre>

## Remedial actions evidence

### Test 3)

```

System.out.println("The optimal solution is:");
for(int i = 0; i < numDecisionVariables; i++){
    System.out.println("x" + (i+1) + " = " + Math.round(solution[i] * 1000000.0) / 1000000.0); //printing from start of solution array to index numDecisionVariables
}
for(int i = 0; i < numSlackVariables; i++){
    System.out.println("s" + (i+1) + " = " + Math.round(solution[numDecisionVariables + i] * 1000000.0)/1000000.0); //printing from numDecisionVariables + 1 to end of array
}
System.out.println("Optimal value: " + Math.round(table.getOptimalValue() * 1000000.0) / 1000000.0); //using getOptimalValue() method from Tableau class to output optimal value

```

### Test 5)

I implemented an if-statement that checks if the input read by the scanner object is an integer or not. If not, then a message is outputted to the user indicating that they should enter a positive number of variables. This also allows for the error message to be displayed based on if the input is a non-integer character like a symbol.

```

Scanner scanner = new Scanner(System.in);
int numVar;
int numConstraints;
while (true){
    System.out.println("Enter number of variables: ");
    if (scanner.hasNextInt()) {
        numVar = scanner.nextInt();
        if (numVar > 0){ // Ensure a positive number of variables entered
            break;
        }else{
            System.out.println("Please enter a positive number.");
        }
    }else{
        System.out.println("Invalid input. Please enter only numbers.");
        scanner.next(); // Clear the invalid input
    }
}
while (true){
    System.out.println("Enter number of constraints: ");
    if (scanner.hasNextInt()){
        numConstraints = scanner.nextInt();
        if (numConstraints > 0){// Ensure that a positive number of constraints are entered
            break;
        }else{
            System.out.println("Please enter a positive number.");
        }
    }else{
        System.out.println("Invalid input. Please enter only numbers.");
        scanner.next(); // Clear the invalid input
    }
}

```

This logic is then repeated for the number of constraints to ensure that this also contains an input of positive integers only and can inform the user if they input anything other than the accepted inputs.

## Test 6)

```

for (int i = 0; i < numConstraints; i++){
    while (true){
        String line = scanner.nextLine();
        String[] parts = line.trim().split( regex "\\s+" );
        if(parts.length == numVar + 1) { // checks if the number of inputs matches numVar + 1
            for (int j = 0; j < numVar + 1; j++) {
                try {
                    constraints[i][j] = Double.parseDouble(parts[j]); // adds entered coefficients of constraints to 2D array
                } catch (NumberFormatException e) {
                    System.out.println("Invalid input. Please enter only numbers.");
                    break;
                }
            }
            break; //exit the loop if valid input
        } else {
            System.out.println("Invalid number of coefficients entered.");
        }
    }
}

```

This remedial action follows similar input validation to ensure that only numbers are inputted for the coefficients as opposed to other characters like symbols. This then displays a message to the user to indicate what they need to do correctly.

The developed code also checks that the correct number of coefficients was entered to ensure there are no errors when defining the dimensions of the tableau, as the value for the number of variables is used when creating the tableau, so an incorrect number of inputs would lead to the tableau being misaligned / not populated correctly.

The same logic is also implemented for the objective function, ensuring that the user inputs numbers and that they enter the right number of coefficients.

```
double[] objective = new double[numVar];

System.out.println("Enter coefficients of the objective function:");
while (true) {
    String line = scanner.nextLine();
    String[] parts = line.trim().split( regex: "\\\s+" );

    if (parts.length == numVar) { // check if the number of inputs matches numVar
        for (int i = 0; i < numVar; i++) {
            try {
                objective[i] = Double.parseDouble(parts[i]);
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter only numbers.");
                break;
            }
        }
        break;
    } else {
        System.out.println("Invalid number of coefficients.");
    }
}
```

## Questions and Mark Schemes

Note that the mark schemes and questions give variables in terms of  $x$ ,  $y$ ,  $z$  mostly. Since this Iteration was focused on ensuring mathematical accuracy, the answers produced by my system are given in terms of  $x_1$ ,  $x_2$ ,  $x_3$  and  $x_4$ , with the slack variables in terms of  $s_1$ ,  $s_2$ ,  $s_3$ . When integrating the mathematical logic into the GUI in further iterations, the correct variable identifiers will be used, however, currently they have just been differentiated by a number (e.g.  $x_1$ ,  $x_2$ ) to ensure that the answers are still accurate and that the correct variables are given the correct values. The questions used were taken from the Pearson ActiveLearn Decision 1 Textbook (<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==>) (<https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8LTF8MHww>) with the answers taken from the solution bank, a collection of worked solutions to the answers from the aforementioned textbook ([https://activeteach-prod.resource.pearson-intl.com/r00/r0072/r007257/r00725745/current/alevelsb\\_dm1\\_ex7b.pdf](https://activeteach-prod.resource.pearson-intl.com/r00/r0072/r007257/r00725745/current/alevelsb_dm1_ex7b.pdf))

Figure 1a

**Exercise 7B**

Solve the linear programming problems in questions **1** to **6** using the simplex tableau algorithm.

- 1** Maximise  $P = 5x + 6y + 4z$   
subject to

$$\begin{aligned}x + 2y + r &= 6 \\5x + 3y + 3z + s &= 24 \\x, y, z, r, s &\geq 0\end{aligned}$$

Figure 1b

$$P = 38 \quad x = 0 \quad y = 3 \quad z = 5 \quad r = 0 \quad s = 0$$

Figure 2a

- 2** Maximise  $P = 3x + 4y + 10z$   
subject to

$$\begin{aligned}x + 2y + 2z + r &= 100 \\x + 4z + s &= 40 \\x, y, z, r, s &\geq 0\end{aligned}$$

Figure 2b

$$P = 260 \quad x = 0 \quad y = 40 \quad z = 10 \quad r = 0 \quad s = 0$$

Figure 3a

3 Maximise  $P = 3x + 5y + 2z$   
subject to

$$\begin{aligned}3x + 4y + 5z + r &= 10 \\x + 3y + 10z + s &= 5 \\x - 2y + t &= 1 \\x, y, z, r, s, t &\geq 0\end{aligned}$$

Figure 3b

$$P = 11 \quad x = 2 \quad y = 1 \quad z = 0 \quad r = 0 \quad s = 0 \quad t = 1$$

Figure 4a

A 4 Maximise  $P = 4x_1 - 3x_2 + 2x_3 + 3x_4$   
subject to



$$\begin{aligned}x_1 + 4x_2 + 3x_3 + x_4 + r &= 95 \\2x_1 + x_2 + 2x_3 + 3x_4 + s &= 67 \\x_1 + 3x_2 + 2x_3 + 2x_4 + t &= 75 \\3x_1 + 2x_2 + x_3 + 2x_4 + u &= 72 \\x_1, x_2, x_3, x_4, r, s, t, u &\geq 0\end{aligned}$$

Figure 4b

$$\text{Maximum } P = 105 \frac{1}{2}, \text{ when } x_1 = 19 \frac{1}{4}, x_2 = 0, x_3 = 14 \frac{1}{4}, x_4 = 0, r = 33, s = 0, t = 27 \frac{1}{4}, u = 0$$

Having carried out all tests, with a variety of pre-defined questions with correct answers, combined with some erroneous inputs to test input validation, I have also implemented any remedial actions that were highlighted during the testing phase when the tests were initially unsuccessful. These remedial actions have also been tested for accuracy in the 'Second Test' column in the above tables to ensure that the system is now correctly handling any errors that were caused previously. This concludes the testing of Iteration 2.

## Iteration Summary and Review

This iteration aimed at implementing the most complex of the algorithms that I aim to implement, that was identified as a particular challenge in the Edexcel A-Level Further Maths course by my stakeholders. Being able to take inputs and format them in a way that they can be used by the developed methods is a valuable feature that I learned from the development stage. Formatting the inputted inequalities and objective function into a 2d array to be used by the methods for the Simplex algorithm may also have uses in further iterations where I will aim to display the tableaux on the GUI, which will require specific formatting and careful thought regarding the dimensions of a tableau. I presented the methods developed from this iteration to my stakeholders to gain some feedback about what they think is effective about this iteration and what they think could be improved upon.

Question	Student 1	Student 2	Student 3
Which features of this initial version of the Simplex algorithm do you think work well and are beneficial as an end-user of the system?	I think that being able to enter my own inequalities into the program and have it output the final answer is definitely beneficial and would like to see it built into some sort of user interface.	I think it would be really useful to have a way to enter my own inequalities into the program and get the final answer. Any questions that I have that I can't find the mark scheme to can be inputted into this to get the answer.	I think that being able to input the inequalities and have the values of variables outputted at the end as well as the maximised profit works well and is useful for getting the answer to a Simplex question quickly.
What improvements or features would you like to see implemented?	Like I mentioned in the previous question, I think that the developed tool so far would be more effective if they were	I think it would be much better if this tool was implemented into some sort of user interface instead of just entering the inequalities	I think making the variable names slightly different would be better because in the exam questions they have x, y and z and they only

	<p>implemented into some sort of user interface such as the one you designed and got our feedback on, and if the process was explained to the user step by step, indicating how each step of the algorithm works.</p>	<p>and objective function into the terminal as this would also allow for better explanation of what is going on in the algorithm.</p>	<p>use <math>x_1, x_2, x_3, x_4</math> when they reach four variables, so I would like to see a feature that uses <math>x, y</math> and <math>z</math> until it needs four variables. A similar feature for the slack variables would also be useful, like <math>r, s, t, u</math> instead of <math>s_1, s_2, s_3, s_4</math> to make it more like the real questions.</p>
--	---	---	--

From this interview, it was highlighted that the students do value the Simplex tool, however two key points to consider in future iterations were raised. The first of this is implementing the tool in the GUI, which I aim to do, as well as accompanying the tableaux and answer with proper explanation as to how the algorithm is performed. This would include how to get the pivot value, how to deduce the row operations and how they are performed on the tableau. The second point was raised by student 3 who mentions implementing a feature to use variable names that are more accurate to the Edexcel A-Level Further Maths specification, which would use  $x, y$  and  $z$  for problems with up to 3 variables,  $x_1, x_2, x_3$  and  $x_4$  for problems with 4 variables, and use the slack variables  $r, s, t$  and  $u$  instead of the current  $s_1, s_2, s_3, s_4$ . These points would help ensure success criterion 10 is met and that the tool remains relevant to [SCHOOL NAME ABBREVIATION] students.

Having now gained feedback on the iteration, with an idea of how to proceed in further iterations that relate to the Simplex algorithm, this iteration is now complete.

## Iteration 3: Kruskal's Algorithm

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

As mentioned in the design section, testing of the algorithm against past questions that students are likely to be asked when revising or questions very similar to those that will come up in their actual exam is necessary to ensure that the development from this iteration aligns with success criterion 10 which outlines adhering to the Edexcel A-Level Further Maths specification.

The following test cases regarding past questions will be taken from the Edexcel A-Level Further Maths Decision 1 textbook which have reliable worked solutions. The first few tests are aimed at solely checking the functionality of the GUI components, while the last few tests will check both the GUI and the algorithm implemented together through various questions.

Test Number	Test Type	Expected Outcome	Actual Outcome	Remedial Actions (if required)	Second Test (to check remedial actions or for thorough check of working features)
1	Normal	The button options should be displayed when the 'Run'	The 3 buttons were correctly displayed as well as	No remedial actions required	Returning to the home page then selecting the Run

		Kruskal's' menu item is selected	the separator line		Kruskal's menu item again produced the same successful result
2	Normal	When pressing the 'Add Edge' button, the 4 text areas and additional button should appear underneath the current buttons	The text areas were correctly displayed as well as the 'Add' button, however, the added button was displayed very close to the textboxes resulting in a slightly cramped layout	I adjusted the spacing between the button and the text areas to ensure that adequate spacing was left between the added button and the text areas	Testing this again, all components were displayed correctly and each had adequate spacing surrounding it, deeming this test now successful
3	Normal	When selecting the Run Kruskal's option after another option e.g. one that displays a blank screen, the same options should be	The same three buttons and separator line were displayed when I tried selecting Run Kruskal's from the Simplex	No remedial actions required	Testing this again with a different page (I tried Learn Prims) produced the same successful result

		presented as if the user had pressed Run Kruskal's as their first action	Introduction page		
4 Refer to Figure 1a below	Normal	When inputting the edges from the first example in the A-Level Further Maths decision 1 textbook, the edges DE, AE, BC, and BD should be added to MST with total weight 20	The correct values were outputted in the console when all the edges from the graph in the example were added using the text areas in the GUI	No remedial actions required	Testing the same inputted edges again, the program yielded the same output of 20, with the correct 4 edges added to the minimum spanning tree, deeming this test successful
5 Refer to Figure 2a below	Normal	When testing example 2 from the A-Level Further Maths Decision 1 textbook, the edges AD, AC, BC,	The correct edges were added to the graph, however once all required edges were added, some additional	I changed the section of code which adds a vertex to the component of the minimum spanning tree, which	This change led to a correct output of 49 for the total weight with the correct edges included, hence the

		CE, EF should be added with a total weight of 49	were also added, giving a total weight of 88, an incorrect solution.	was previously incorrect.	remedial actions were effective, and this test is now deemed successful
6 Refer to Figure 3a below	Normal	Inputting the graph from question 1a on exercise 3A from the Edexcel A-Level Further Maths Decision 1 textbook should provide a total weight of 98.	The program provided the correct output of 98 with the correct edges of the minimum spanning tree outputted	No remedial actions required	Testing again with the same graph, the same correct output was produced with the same edges and total weight, deeming this test successful.
7 Refer to Figure 4a below	Normal	Inputting the graph from question 1b on exercise 3A from the Edexcel A-Level Further Maths Decision 1 textbook should provide a	The program provided the correct output of 27 as well as all correct edges of the minimum spanning tree	No remedial actions required	Running the same test again produced the same correct result with the correct edges of the minimum spanning tree and the correct

		total weight of 27.			weight, deeming this test successful
--	--	---------------------	--	--	--------------------------------------

Outcome Table

Test Number	Outcome	Remedial Actions	Second Test (either after remedial actions or to double check functionality)
1		No remedial actions required	
2		Evidenced below the outcome table	
3	<p>Before selecting the 'Run Kruskal's' option:</p> <p>After selecting the 'Run Kruskal's' Option:</p>	No remedial actions required	<p>Before selecting the 'Run Kruskal's' option:</p> <p>After selecting the 'Run Kruskal's' Option:</p>

4	Refer to Figure 1b below	<pre>Edges included in the minimum spanning tree: Edge: DE, Weight: 4 Edge: AE, Weight: 5 Edge: BC, Weight: 5 Edge: BD, Weight: 6 Total Weight of MST is: 20</pre>	No remedial actions required	<pre>Edges included in the minimum spanning tree: Edge: DE, Weight: 4 Edge: AE, Weight: 5 Edge: BC, Weight: 5 Edge: BD, Weight: 6 Total Weight of MST is: 20</pre>
5	Refer to Figure 2b below	<pre>Edges included in the minimum spanning tree: Edge: AD, Weight: 8 Edge: BC, Weight: 8 Edge: AC, Weight: 10 Edge: EF, Weight: 11 Edge: CE, Weight: 12 Edge: DE, Weight: 12 Edge: DF, Weight: 13 Edge: BE, Weight: 14 Total Weight of MST is: 88</pre> <p>Remedial action:</p> <pre>for (int i = 0; i &lt; numVertices; i++) {     if (components.get(i) == compD) {         components.set(i, compS);     } }</pre>	Evidenced below outcome table	<pre>Edges included in the minimum spanning tree: Edge: AD, Weight: 8 Edge: BC, Weight: 8 Edge: AC, Weight: 10 Edge: EF, Weight: 11 Edge: CE, Weight: 12 Total Weight of MST is: 49</pre>
6	Refer to Figure 3b below	<pre>Edges included in the minimum spanning tree: Edge: EF, Weight: 11 Edge: BD, Weight: 12 Edge: CD, Weight: 12 Edge: AH, Weight: 14 Edge: DF, Weight: 15 Edge: AC, Weight: 16 Edge: GH, Weight: 18 Total Weight of MST is: 98</pre>	No remedial actions required	<pre>Edges included in the minimum spanning tree: Edge: EF, Weight: 11 Edge: BD, Weight: 12 Edge: CD, Weight: 12 Edge: AH, Weight: 14 Edge: DF, Weight: 15 Edge: AC, Weight: 16 Edge: GH, Weight: 18 Total Weight of MST is: 98</pre>
7	Refer to Figure 4b below	<pre>Edges included in the minimum spanning tree: Edge: BF, Weight: 2 Edge: FG, Weight: 3 Edge: AB, Weight: 4 Edge: CE, Weight: 5 Edge: BC, Weight: 6 Edge: CD, Weight: 7 Total Weight of MST is: 27</pre>	No remedial actions required	<pre>Edges included in the minimum spanning tree: Edge: BF, Weight: 2 Edge: FG, Weight: 3 Edge: AB, Weight: 4 Edge: CE, Weight: 5 Edge: BC, Weight: 6 Edge: CD, Weight: 7 Total Weight of MST is: 27</pre>

## Remedial Actions evidence

## Test 2)

To ensure there was adequate spacing between the weight text area and the 'Add' button, I added a line to the code that creates a horizontal strut i.e. some space between the components.

```
kruskalAdd.add(weightTA);
kruskalAdd.add(Box.createHorizontalStrut( width: 20)); //adding space between the weight text area and the add button
kruskalAdd.add(addButton);
```

Testing this proved successful, with the result shown in the Second Test column of the outcome table above. This remedial action ensure that the GUI isn't too cramped and that components are well spaced apart.

## Test 5)

This was a major error in the logic of the kruskalMST method, since the component of the node was not being set correctly. The component was being set to the value s, which was the source node of the edge, however it should have been set to compS, which is the component of the source node. This was resulting in the check of components not being carried out correctly, which was resulting in additional edges being added to the graph, providing the answer which has more edges and a much higher weight than expected. This error was initially difficult to spot, especially given that test 4 had passed, however it is likely due to the simplicity of the graph in the example from test 4 that meant it managed to coincidentally pass the test case with the correct result.

```
for (int i = 0; i < numVertices; i++) {
    if (components.get(i) == compD) {
        components.set(i, compS);
    }
}
```

This fix in the logic of setting the component resulted in the correct output being generated when I tested the same question again. This result is shown in the Second Test column of the outcome table.

After making this change, I also decided to double check that test 4 still worked after this change was made. Since this remedial action implemented the correct logic for the algorithm to work, test case 4 still passed once again successfully.

Having completed all outlined tests and solved the problems that arose during the testing process, the testing of iteration 3 is now complete. I can now move onto an iteration review, which will involve feedback from my stakeholders on this iteration.

## Questions and Mark Schemes

Figure 1a

### Example 1

Use Kruskal's algorithm to find a minimum spanning tree for this network. List the arcs in the order that you consider them. State the weight of your tree.

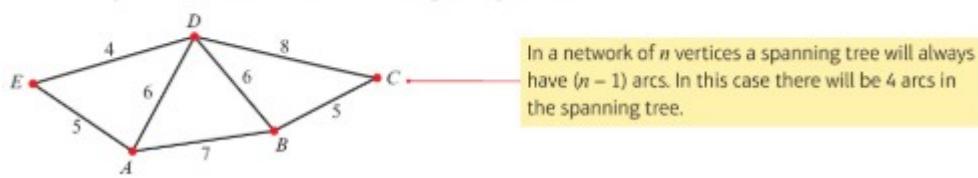
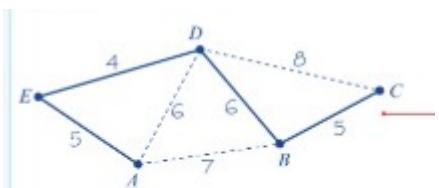


Figure 1b



All vertices are connected so this is a minimum spanning tree.

Its weight is  $5 + 4 + 6 + 5 = 20$

Figure 2a

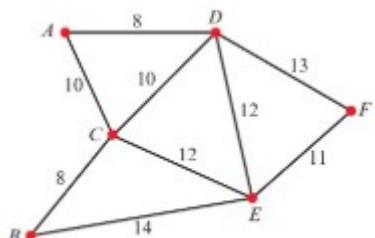
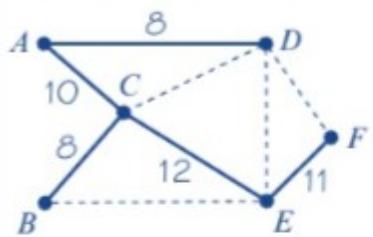


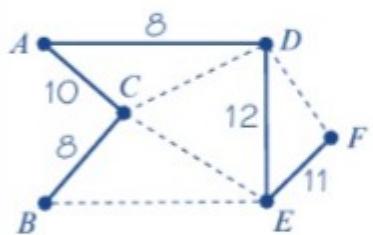
Figure 2b

One solution is

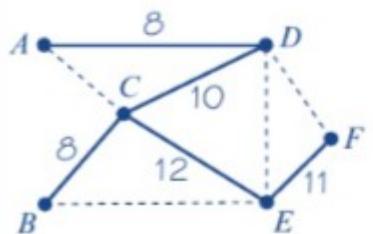


(using  $AC$  and  $CE$ )

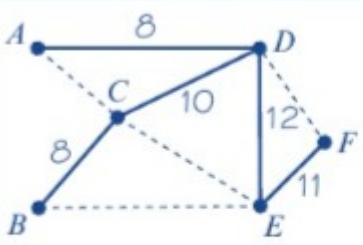
The other three solutions are



(using  $AC$  and  $DE$ )



(using  $CD$  and  $CE$ )



(using  $CD$  and  $DE$ )

The weight of each tree is

$$8 + 8 + 10 + 11 + 12 = 49$$

Figure 3a

### Exercise 3A

- 1 Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

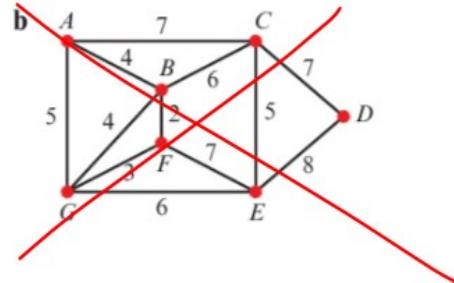
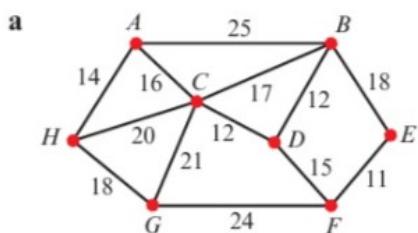


Figure 3b

- 1 a
- |         |             |
|---------|-------------|
| EF (11) | add to tree |
| BD (12) | add to tree |
| CD (12) | add to tree |
| AH (14) | add to tree |
| DF (15) | add to tree |
| AC (16) | add to tree |
| BC (17) | reject      |
| GH (18) | add to tree |
| BE (18) |             |
| CH (20) |             |
| CG (21) |             |
| FG (24) |             |
| AB (25) |             |
- } reject all remaining arcs.

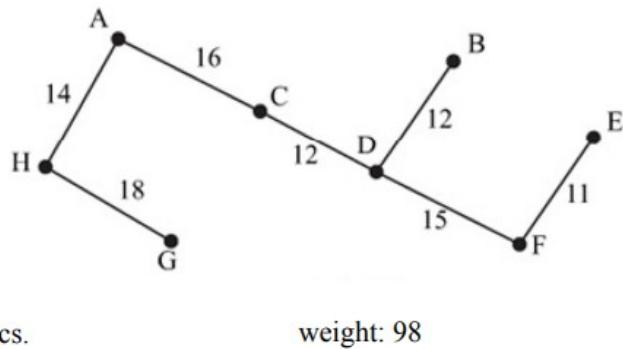


Figure 4a

### Exercise 3A

- 1 Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

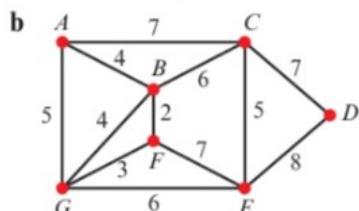
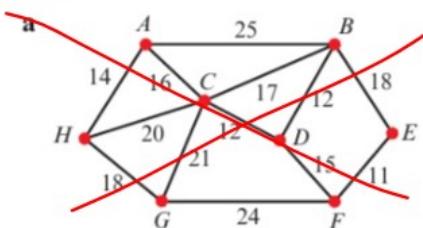
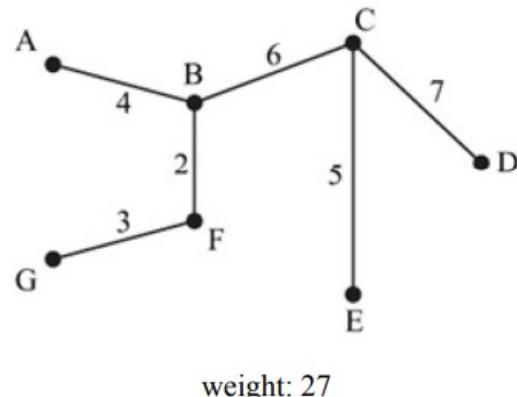


Figure 4b

- b BF (2) add to tree
- FG (3) add to tree
- AB (4) add to tree
- BG (4) reject
- AG (5) reject
- CE (5) add to tree
- BC (6) add to tree
- EG (6) reject
- AC (7) reject
- CD (7) add to tree
- EF (7)
- DE (8) } reject all remaining arcs.



## Iteration Summary and Review

This iteration worked on another complex algorithm, and I was able to develop it in a way that can be followed coherently by [SCHOOL NAME ABBREVIATION] students. While I was more familiar with the implementation of a depth first search, the various drawbacks that were highlighted, such as the divergence from the Edexcel A-Level Further Maths specification led me away from using that technique. However, I think that the method used in this iteration of checking components will also prove useful in further iterations, further utilising the computational method of 'Thinking Ahead'. Using the component checking method will allow for better explanation in a further iteration that involves making the algorithm more suitable for use by my end-users, who I will now consult regarding the developed methods.

I will ask them open-ended questions that will allow them to explain which features they think work well and are beneficial, as well as any features they think would be useful or could be improved upon in further iterations.

Question	Student 1	Student 2	Student 3
Which features of the Kruskal's Algorithm tool do you think are beneficial and work well?	I like being able to enter the edges into the user interface as it makes the system seem a bit more	I think that the user interface for this iteration is developed well. Being able to input edges into	I like the feature of being able to add edges using the graphical user interface, since it is simpler than

	<p>robust, since when you showed the Simplex algorithm development, the inputs were taken from the terminal. I think taking inputs from the user interface makes the system much better. I also like how the final edges are outputted as well as the weight since this is what we have to do in the exam questions.</p>	<p>the GUI is a good start and I also like how each edge that was added to the graph was displayed in order.</p>	<p>trying to format them in a way that would be recognised in the terminal. I like how the list of edges included in the minimum spanning tree is outputted, as well as the final weight, with indication as to what is being printed.</p>
What improvements or suggestions do you have for this part of the system in future iterations?	<p>The extraction of user inputs from the GUI is good, but I would also like to see the output take place in the GUI. Also, to ensure that the tool is useful for students, it would require explanation. I also think that adding a visual element would be useful, since students in the exam are provided with a picture of the</p>	<p>I think that to improve the system, the outputs should also be taking place in the user interface. I would also say that the algorithm requires some explanations, since currently the edges are just outputted in a list, which is useful, but they need explanation too. I think that adding a visual element</p>	<p>I think that to improve the system, the outputs should be firstly displayed on the GUI but also explained with the further maths steps, so this would include having the table of added and rejected edges that we're taught, and possibly drawing the inputted graph on the screen in some way since</p>

	graph to perform the algorithm on.	would also be very beneficial to the system, such as something that draws the graph on the screen for the user to follow.	we're also provided with the visual graph in exam questions.
--	------------------------------------	---	--

From these questions, I can identify that the key features for future iterations would be bringing the outputs to the GUI and explaining them. The students all agreed that the method of taking inputs for edges was beneficial, both in the design aspect but also in this interview, hence, this feature will remain part of the system through future iterations. All three of the students also brought up including a visual graph display element. This would serve as explanation for the algorithm but also ensure that students can follow the algorithm easily and that the explanation doesn't just end up becoming streams of text. This would help further achieve success criterion 1. Since the displayed graph is also part of the Edexcel A-Level further maths questions, implementing this feature would help better align with the specification and work towards success criterion 10.

Having now consulted the stakeholders about the features they found beneficial and what they would like to see, I now have an idea of what future iterations of this algorithm would achieve. With the design, development, testing and review of the Kruskal's Algorithm iteration now complete, I can move onto the next iteration.

## Iteration 4: Simplex Algorithm with GUI Implementation

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

The developed features from this iteration can be split into 2 main sections to be tested. The first set of features to test regard the functionality of the GUI. This will involve ensuring that various inputs to the GUI result in the correct components will be displayed e.g. ensuring that selecting 2 variables displays 2 variables. This will also allow me to test that the input validation developed in the GUI correctly guides the user into entering valid inputs. The second set of features to test will be ensuring that all the methods developed in Iteration 2 correctly work to perform the algorithm on entered questions, producing the correct answer. This part of testing will also ensure that the tableaux are displayed correctly, with the correct row operations and theta values, as well as informative explanation. The test table below outlines the description of the test, expected outcomes, actual outcomes and any remedial actions taken.

Test Number/ Description	Test Type	Expected Outcome	Actual Outcome	Remedial Actions (if required)	Second Test (either after remedial actions, or if none required then to double check functionality)
1a) Testing the 2-inequality drop-down menu option with the 2-variable button	All 3 are normal test types	1a) 2 rows of inequalities and one objective row should be displayed with x and y as variables.	The correct number of inequality rows (2) and the correct variables were displayed for each of the 3 tests. All the displayed	None required	Testing each of the 3 button options again, the same, correct components were displayed in the correct place, with
1b) Testing the 2-inequality		1b) 2 rows			

<p>drop-down option with the 3-variable button.</p> <p>1c) Testing the 2-inequality drop-down option with the 4-variable button.</p>		<p>of inequalities and one objective row should be displayed with x, y and z as variables</p> <p>1c) 2 rows of inequalities and one objective row should be displayed with x1, x2, x3 and x4 as variables.</p>	<p>components were spaced evenly in proportion to one another and the whole GUI window</p>		<p>correct proportion.</p>
--	--	--	--	--	----------------------------

2a) Testing the 3-inequality drop-down menu option with the 2-variable button	All 3 are normal test types	2a) 3 rows of inequalities and one objective row should be displayed with x and y as variables.	The correct number of inequality rows (3), the objective function and the correct variables were displayed for each of the 3 tests.	None required	To double-check the functionality, I tested each of the 3 button options again with the 3-inequality option and the same correct components were displayed in the correct place, with correct proportion.
2b) Testing the 3-inequality drop-down option with the 3-variable button.		2b) 3 rows of inequalities and one objective row should be displayed with x, y and z as variables	All the displayed components were spaced evenly in proportion to one another and the whole GUI window		
2c) Testing the 3-inequality drop-down option with the 4-variable button.		2c) 3 rows of inequalities and one objective row should be displayed with x1, x2, x3 and x4 as variables.			
3a) Testing the 4-	All 3 are normal test	3a) On the main	The correct number of	None required	Double-checking

<p>inequality drop-down menu option with the 2-variable button</p> <p>3b) Testing the 4-inequality drop-down option with the 3-variable button.</p> <p>3c) Testing the 4-inequality drop-down option with the 4-variable button.</p>	<p>types</p>	<p>window, 4 rows of inequalities and one objective row should be displayed with x and y as variables.</p> <p>3b) On the main GUI window, 4 rows of inequalities and one objective row should be displayed for each of the 3 tests.</p> <p>3c) On the main window, 4 rows of inequalities and one objective row should be displayed with x, y and z as variables</p>	<p>inequality rows (4), the objective function and the correct variables (this time <math>x_1, x_2, x_3</math> and <math>x_4</math>) were displayed.</p> <p>All the displayed components were spaced evenly in proportion to one another and the whole GUI window</p>	<p>the functionality, I tested each of the 3 button options again with the 4-inequality option and the same correct components were displayed on the GUI in the correct place, with correct proportion.</p>
--	--------------	--	---	---

		variables.			
4a) Trying to select the 2-variable button before the number of inequalities 4b) Trying to select the 3-variable button before the number of inequalities 4c) Trying to select the 4-variable button before the number of inequalities	All 3 of these tests are erroneous	The red error message JLabel should be displayed on the GUI indicating to the user that they must first select the number of inequalities when each of the buttons are pressed before selecting the number of inequalities	Testing with each of the buttons, the red error message label displayed the correct message 'Please select the number of inequalities first' next to the buttons.	No remedial actions required	Testing each of the buttons again before having selected an option from the drop-down menu, the same correct result occurred, with the error message label being displayed on the GUI next to the panels.
5) Selecting the 'Run' button before selecting the number of variables.	Erroneous	The red error message label should be displayed next to the buttons, indicating to the user that they must first select the	The error message label was correctly displayed in red, telling the user to 'Please select the number of variables.' This label	No remedial actions required	I performed this erroneous test again, and the same output was produced correctly, with the label again displaying

		number of variables before running.	was displayed in the correct location, next to the buttons on the window		the correct message and being in the right location on the GUI.
6) Selecting the 'Run' button before selecting the number of inequalities and variables	Erroneous	The red error message label should be displayed next to the buttons, indicating to the user that they must first select the number of inequalities and variables before running.	The error message label was correctly displayed in red, telling the user to 'Please select the number of inequalities .' The label was displayed next to the buttons on the window, which is the correct location.	No remedial actions required	I tested this once again, and the correct message was displayed on the GUI, in the red text and in the correct location next to the buttons.
7a) Trying to select 'Run' before entering any values into the text areas	Erroneous	For both 7a and 7b, the error message label should appear on the GUI	For both 7a and 7b, the error message label was correctly displayed in the right	I slightly altered the error message label text to also prompt users to	Testing again, the message displays in the correct place, with the red text, which

7b) Trying to select 'Run' having only entered some values into the text areas		next to the buttons, prompting the user to enter valid values into the text areas	location next to the buttons. While not an 'error' with the code, I noticed that the contents of the error message 'Please enter valid numbers only' was a bit ambiguous, as it didn't directly tell the user to fill in each of the text areas.	ensure all the text areas have inputs	is now clearer in implying to the user that they need to enter valid number in all the text areas.
8a) Entering invalid characters in some of the text areas (e.g. letters)  8b) Entering invalid characters in all the text areas	Erroneous	For both 8a and 8b, the red error message label should be displayed on the GUI, informing them to enter valid numbers in the text areas. With the	8a) The error message label was correctly displayed in red indicating that valid numbers must be entered in all the text areas.	No remedial actions required	Testing 8a again, the same correct output was produced with the error message label indicating that there needs to be valid numbers

(e.g. letters)		remedial actions from test 7a and 7b, the new message should be displayed	8b) The error message label was correctly displayed once again, indicating that there must be valid numerical inputs into all of the text areas.		inputted into each of the text areas.  A second test of 8b also yielded the same, correct result. The error message label was displayed in red text next to the row of buttons indicating that there must be valid number inputs into all the text areas.
9) Chapter 7 Example 8 from Pearson ActiveLearn Decision 1 Textbook ( <a href="https://www.pearsonactivelearn.com/app/library/ebook">https://www.pearsonactivelearn.com/app/library/ebook</a>	Normal	The final values of variables and the total profit should be $P = 26$ , $x = 42/11$ (3.82), $y = 80/11$ (7.23), $r = 0$ and $s = 0$ .	The correct final values of variables were outputted in the explanation pane by the program. Each iteration was	No remedial actions required	Testing the same inequalities and objective function from Example 8 yielded the same correct result. The

?		The final tableau should have the row headers changed to y and x respectively .	properly explained, with justification as to how the pivot row and column are deduced, how the row operations are calculated and how the current tableau was reached.		iterations were correctly split up into separate tabs, with each tab containing the tableau, row operations, theta values and explanation . The final values of variables were also correctly deduced and outputted.
10) Chapter 7 Example 10 from Pearson ActiveLearn Decision 1 Textbook ( <a href="https://www.pearsonactivelearn.com/app/library/ebook?">https://www.pearsonactivelearn.com/app/library/ebook?</a> id=NzAzNz M1fGJvb2t8 MTk5fDB8	Normal	The final values of variables and the total profit should be P = 45, x = 0, y = 2.5, z = 1.88, r = 0 and s = 0. The final tableau should have the row headers	The correct final variables were outputted with the correct final value of profit. There were three iterations as expected and the pivot column and	No remedial actions required	Testing the same inequalities again, the same output was produced, with 3 tabs in the explanation tabbed pane, the correct pivot row and column for each

MA==)		changed to y and z respectively . There should be three iterations. The pivot column in order of iterations should be y then z and the pivot row in order of iterations should be r then s.	pivot row were highlighted correctly in each of the iterations that involve calculating pivot row and column. The row headers were also correctly changed through the iterations.		tableau and the correct row headers being changed through the iterations.
11) Chapter 7 Example 11 from Pearson ActiveLearn Decision 1 Textbook ( <a href="https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==">https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==</a> )		The final values of variables and the total profit should be P = 144/7 (20.57), x = 0=32/7 (4.57), y = 12/7 (1.71), z = 0, r = 0, s = 0 and t = 30/7 (4.26) The final tableau should have the row	The correct final value of variables and total profit were outputted in the tab of the final iteration. The pivot row and column were correctly highlighted through the iterations and this was	No remedial actions required	Testing the same set of inequalities again, the same final output was produced correctly. Each tableau was explained with how the pivot row and column were deduced as well as the row

		<p>headers changed to x,y and t respectively . There should be three iterations. The pivot column in order of iterations should be y then x and the pivot row in order of iterations should be s then r.</p>	<p>explained underneath . The row headers also changed one-by-one through the iterations and were in the correct final state.</p>		<p>operations. The pivot row and column were also highlighted correctly in each iteration. Finally, the row headers also changed correctly with each iteration, reflecting the correct final state in the last tableau.</p>
12) Chapter 7 Exercise 7B Question 1 from Pearson ActiveLearn Decision 1 Textbook ( <a href="https://www.pearsonactivelearn.com/app/library/ebook/?id=NzAzNzM1fGJvb2t8">https://www.pearsonactivelearn.com/app/library/ebook/?id=NzAzNzM1fGJvb2t8</a>	Normal	<p>The final values of the variables and profit should be P = 38, x = 0, y = 3, z = 5, r = 0, s = 0. There should be 3 iterations of the algorithm. The pivot row through</p>	<p>The correct final variables were outputted, as well as the maximised profit value. The three iterations were correctly added as individual tabs in the</p>		<p>Testing the same inequalities again, the correct final variables were once again outputted with the maximised profit in the final tableau explanation . Each</p>

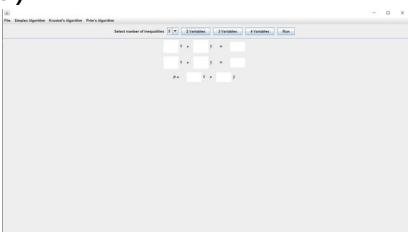
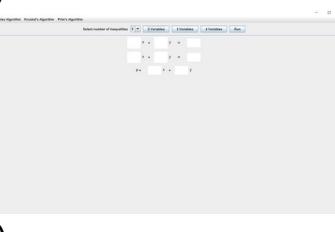
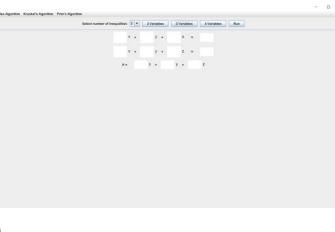
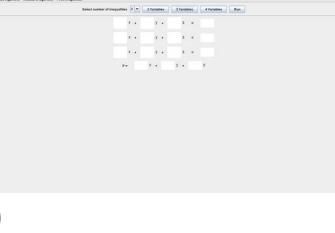
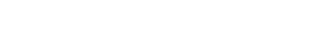
MTk5fDB8 MA==)		the iterations should be r then s. The pivot column through the iterations should be y then z. The row headers should change to a final state of y, z.	tabbed pane and each tableau had the pivot row and column correctly identified. The row headers also changed through each table, eventually taking the final correct state.		tableau had its row operations, pivot row and pivot column explained and were correctly highlighted through the tableaux, deeming this test successful.
13) Chapter 7 Exercise 7B Question 4 from Pearson ActiveLearn Decision 1 Textbook ( <a href="https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==">https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==</a> )	Normal	The final values of variables and the total profit should be P = 105.5, x1 = 19.25, x2 = 0, x3 = 14.25, x4 = 0 r = 33, s = 0 and t = 27.25, u = 0 The final tableau should have the row headers	The correct final values of variables and the total profit were outputted in the final tab of the explanation pane. There were 3 iterations as expected, and the row headers changed	To fix the issue of the lengthy decimal in the explanation, I ensured that in the explainTableau method, the numbers added to the explanation that are taken from the table	I tested the same inequality inputs again and the program once again yielded the same correct output of variables and maximised profit. The three iterations were

		<p>changed to r, x3, t and x1 respectively. There should be three iterations. The pivot column in order of iterations should be x1 then x3 and the pivot row in order of iterations should be u then s.</p>	<p>one-by-one until they reached the correct final state in the final tableau. The pivot columns were also correctly identified and highlighted as well as the pivot rows for each of the iterations of the algorithm. One slight error I noticed was in the explanation , where the pivot value wasn't being rounded, so the text contained a long decimal</p>	<p>are rounded to 2 decimal places before being outputted. This ensures that the value can still be accurately outputted, but any lengthy decimal numbers don't take up too much of the explanation panel.</p>	<p>correctly added to the tabbed pane and each of the tabs correctly contained the tableau, row operations (except for initial tableau which doesn't require any) and theta values. This time the explnaation contained the correctly rounded value, which was still identifiable as the pivot value, but wasn't given to an excessive number of decimal places,</p>
--	--	---	---	--	--

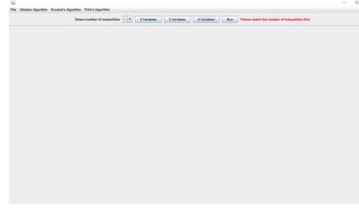
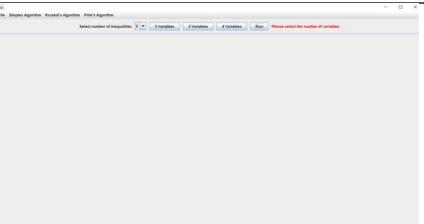
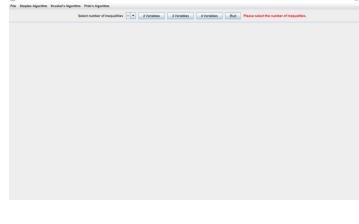
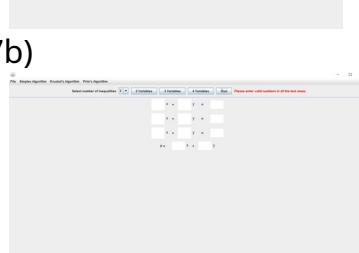
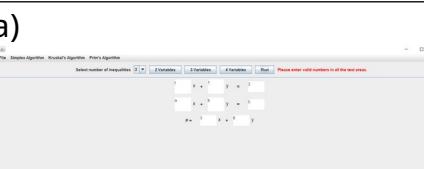
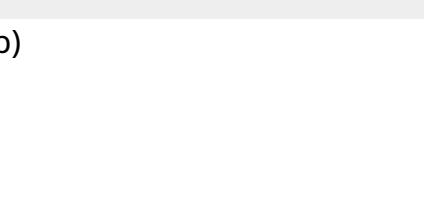
					deeming this test now successful.
14) Chapter 7 Exercise 7B Question 2 from Pearson ActiveLearn Decision 1 Textbook ( <a href="https://www.pearsonactivelearn.com/app/library/ebook/?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==">https://www.pearsonactivelearn.com/app/library/ebook/?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==</a> )	Normal	The final values of the variables and profit should be $P = 260$ , $x = 0$ , $y = 40$ , $z = 10$ , $r = 0$ , $s = 0$ . There should be 3 iterations of the algorithm. The pivot row through the iterations should be $s$ then $y$ . The pivot column through the iterations should be $y$ then $z$ . The row headers should change to a final state of $y$ , $z$ .	The correct final values of the variables and maximised profit was displayed in the final tab of the iteration tabbed pane. The pivot rows and columns were correctly identified and highlighted in the tableaux. The row headers changed correctly to the final state in the final tableaux.	No remedial actions required	Testing with the same inequalities again yielded the same correct output. The final values of the variables and maximised profit were correctly displayed in the final tab. Each tableau had proper explanation about how the pivot value was identified, what the row operations were and how the state of the tableau was reached.

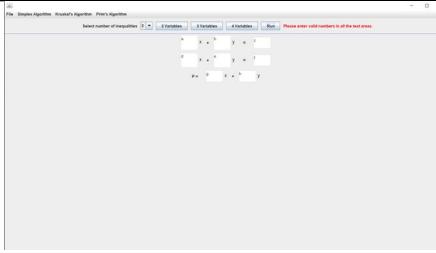
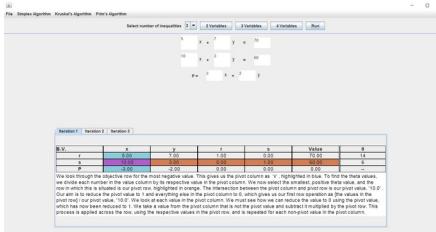
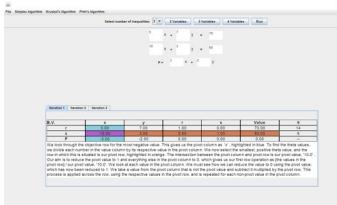
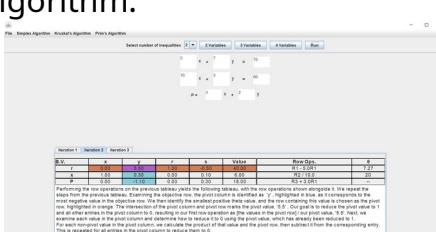
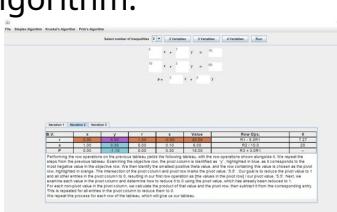
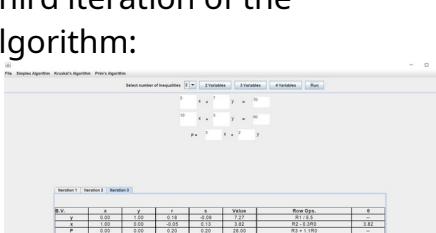
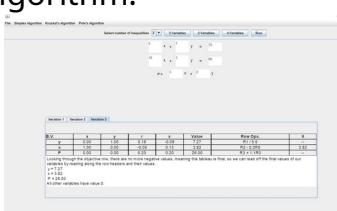
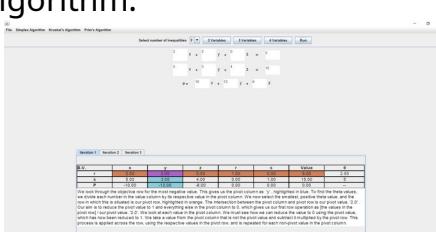
<p>15) Chapter 7 Exercise 7B Question 3 from Pearson ActiveLearn Decision 1 Textbook (<a href="https://www.pearsonactivelearn.com/app/library/ebook/?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==">https://www.pearsonactivelearn.com/app/library/ebook/?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==</a>)</p>	<p>Normal</p>	<p>The final values of the variables and the maximised profit should be <math>P = 11</math>, <math>x = 2</math>, <math>y = 1</math>, <math>z = 0</math>, <math>r = 0</math>, <math>s = 0</math>, <math>t = 1</math>. There should be three iterations of the algorithm. The pivot rows should be <math>s</math> then <math>r</math>, the pivot columns should be <math>y</math> then <math>x</math> and the row headers should change to a final state of <math>x</math>, <math>y</math>, <math>t</math>.</p>	<p>The correct output was produced with the final values of the variables and maximised profit displayed in the final tab of the tableau pane. The pivot rows and column were correctly calculated and highlighted and the row headers correctly changed one-by-one to the final state.</p>	<p>No remedial actions required</p>	<p>Testing the same inequalities again to be tested produced the same correct output with all the variables and maximised profit value displayed on the final tableau tab. Each tab of the tableau pane also contained the correct explanation for each tableau, with correct highlights of pivot row and column as well as showing the changing row headers.</p>
--	---------------	---	---	-------------------------------------	---

## **Outcome Table**

Test number	Outcome	Remedial Actions	Second Test (either with remedial actions, or to double-check functionality)
1a) 1b) 1c)	1a)  1b)  1c) 	No remedial actions required	1a)  1b)  1c) 
2a) 2b) 2c)	2a)  2b)  2c) 	No remedial actions required	2a)  2b)  2c) 

3a) 3b) 3c)	3a)   3b)   3c) 	No remedial actions required	3a)   3b)   3c) 
4a) 4b) 4c)	4a)   4b)   4c) 	No remedial actions required	4a)   4b)   4c) 

			
5)		No remedial actions required	
6)		No remedial actions required	
7a) 7b)	 	Evidenced below the outcome table	 
8a) 8b)	  	No remedial actions required	  

			
9)	<p>First iteration of the algorithm:</p>  <p>No remedial actions required</p>	<p>First iteration of the algorithm:</p> 	
	<p>Second iteration of the algorithm:</p> 	<p>Second iteration of the algorithm:</p> 	
	<p>Third iteration of the algorithm:</p> 	<p>Third iteration of the algorithm:</p> 	
	<p>Refer to Figure 1b for mark scheme</p>		
10)	<p>First iteration of the algorithm:</p> 	<p>No remedial actions required</p>	<p>First iteration of the algorithm:</p> 

Second iteration of the algorithm:

Third iteration of the algorithm:

Refer to Figure 2b for mark scheme

11) First iteration of the algorithm:

Second iteration of the algorithm:

### Third iteration of the algorithm:

Second iteration of the algorithm:

Third iteration of the algorithm:

First iteration of the algorithm:

Second iteration of the algorithm:

Third iteration of the algorithm:



	<p><b>Second iteration of the algorithm:</b></p>	<p><b>Second iteration of the algorithm:</b></p>
	<p><b>Third iteration of the algorithm:</b></p>	<p><b>Third iteration of the algorithm:</b></p>
	<p>Refer to Figure 5b for mark scheme</p>	
14)	<p><b>First iteration of the algorithm:</b></p>	<p>No remedial actions required</p>
	<p><b>Second iteration of the algorithm:</b></p>	<p><b>First iteration of the algorithm:</b></p>
	<p><b>Third iteration of the algorithm:</b></p>	<p><b>Second iteration of the algorithm:</b></p>
	<p><b>Third iteration of the algorithm:</b></p>	<p><b>Third iteration of the algorithm:</b></p>

	<p>Refer to Figure 6b for mark scheme</p>		<p>First iteration of the algorithm:</p>
15)	<p>First iteration of the algorithm:</p> <p>Second iteration of the algorithm:</p> <p>Third iteration of the algorithm:</p> <p>Refer to Figure 7b for mark scheme</p>	<p>No remedial actions required</p>	<p>First iteration of the algorithm:</p> <p>Second iteration of the algorithm:</p> <p>Third iteration of the algorithm:</p>

## Remedial Actions evidence

Test 7a/b)

To avoid ambiguity with the input validation error message, I altered the displayed message so that it informed users to input valid numbers into all the text areas, instead of just informing them to input valid numbers.

```
} catch(NumberFormatException e) {
    errorMessageLabel.setText("Please enter valid numbers in all the text areas.");
}
```

Evidence of this working was shown in the second test after remedial actions for tests 7a and 7b.

Test 13)

The outcome of test 13 wasn't a complete error, however, to avoid the numerical values taking up too much space in the explanation, they needed to be rounded. This is important as there was the risk of important information explaining the tableau being pushed off the text pane since the decimal places were taking up so much space.

To fix this issue, I went through the explainTableau method and changed each section where there are numerical values outputted to first round the number to 2 decimal places before outputting it. An example section is shown below:

```
each number in the value column by its respective value in the pivot column.");
lue, "' + Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] * 100.0) / 100.0 + '' .");
eau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] * 100.0) / 100.0 + '' .'");

is repeated for each non-pivot value in the pivot column.";

column is " + " "' + getColHeaders()[getCurrentPivotCol(tableau)] + "' " + ", highlighted in blue. This was chosen as this column has the most negative value in the object;
us our pivot row. The intersection between pivot row and pivot column, is our pivot value, "' + Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] * 100.0) / 100.0 + '' .");
he pivot value to reduce our pivot value to 1. We now pick the first value in the pivot column that isn't in the pivot row, and reduce it to 0, using the values in the
in the row.";

mn is identified as " + " "' + getColHeaders()[getCurrentPivotCol(tableau)] + "' " + ", highlighted in blue, as it corresponds to the most negative value in the object;
the pivot value, "' + Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] * 100.0) / 100.0 + '' .");
d(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)][getCurrentPivotCol(tableau)] * 100.0) / 100.0 + '' .'");

int column to reduce them to 0.";
```

This works by first multiplying the number by 100, rounding it to the nearest integer, then dividing by 100. For example,  $1.33333333333 * 100 =$

133.333333333, then we round it to 133 and finally divide by 100 to get 1.33, which is given to 2 decimal places.

Evidence of this remedial action being successful was shown in the second test after remedial actions in the outcome table.

## **Questions and Mark Schemes Figures**

The example questions did not contain definitive tableaux for each iteration of the algorithm. Instead, they would split a single tableau up into many tableaux, e.g. one tableau for each row operation. Hence, for the example questions, only the final values of variables have been provided. The mark scheme to the exam questions did contain definitive tableaux for each iteration, hence they have been provided.

Figure 1a

### **Example 8**

Solve the linear programming problem in Example 6 using simplex tableaux.

$$\text{Maximise } P = 3x + 2y$$

subject to:

$$5x + 7y + r = 70$$

$$10x + 3y + s = 60$$

$$x, y, r, s \geq 0$$

**Notation** The word **tableau** is French. The plural of tableau is **tableaux**.

Figure 1b

$P = 26$ ,  $y = \frac{80}{11}$  and  $x = \frac{42}{11}$  and all other variables, and slack variables, are zero.

So our full solution is

$$P = 26, x = \frac{42}{11}, y = \frac{80}{11}, r = 0, \text{ and } s = 0$$

Figure 2a

**Example 10****A**

- a Use simplex tableaux to solve the linear programming problem below (from Example 7).

$$\text{Maximise } P = 10x + 12y + 8z$$

subject to:

$$2x + 2y \leq 5$$

$$5x + 3y + 4z \leq 15$$

$$x, y, z \geq 0$$

Figure 2b

The optimal solution is

$$P = 45, x = 0, y = 2\frac{1}{2}, z = 1\frac{7}{8}, r = 0, s = 0.$$

Figure 3a

**Example 11**

- a Use the simplex tableau method to solve the following linear programming problem.

$$\text{Maximise } P = 3x + 4y - 5z$$

subject to:

$$2x - 3y + 2z + r = 4$$

$$x + 2y + 4z + s = 8$$

$$y - z + t = 6$$

$$x, y, z, r, s, t \geq 0$$

Figure 3b

---

$$P = \frac{144}{7}, x = \frac{32}{7}, y = \frac{12}{7}, z = 0, r = 0, s = 0, t = \frac{30}{7}$$

Figure 4a

### Exercise 7B

Solve the linear programming problems in questions 1 to 6 using the simplex tableau algorithm.

- 1 Maximise  $P = 5x + 6y + 4z$   
subject to

$$\begin{aligned}x + 2y + r &= 6 \\5x + 3y + 3z + s &= 24 \\x, y, z, r, s &\geq 0\end{aligned}$$

Figure 4b

### The simplex algorithm 7B

1

b.v.	x	y	z	r	s	value	θ values
r	1	(2)	0	1	0	6	3 *
s	5	3	3	0	1	24	8
P	-5	-6	-4	0	0	0	

b.v.	x	y	z	r	s	value	Row operations
y	$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	3	R1 ÷ 2
s	$\frac{7}{2}$	0	(3)	$-\frac{3}{2}$	1	15	R2 - 3R1
P	-2	0	-4	3	0	18	R3 + 6R1

b.v.	x	y	z	r	s	value	Row operations
y	$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	3	R1 (no change)
z	$\frac{7}{6}$	0	1	$-\frac{1}{2}$	$\frac{1}{3}$	5	R2 ÷ 3
P	$\frac{8}{3}$	0	0	1	$\frac{4}{3}$	38	R3 + 4R2

$$P = 38 \quad x = 0 \quad y = 3 \quad z = 5 \quad r = 0 \quad s = 0$$

Figure 5a

- A 4** Maximise  $P = 4x_1 - 3x_2 + 2x_3 + 3x_4$   
subject to

$$\begin{aligned}x_1 + 4x_2 + 3x_3 + x_4 + r &= 95 \\2x_1 + x_2 + 2x_3 + 3x_4 + s &= 67 \\x_1 + 3x_2 + 2x_3 + 2x_4 + t &= 75 \\3x_1 + 2x_2 + x_3 + 2x_4 + u &= 72 \\x_1, x_2, x_3, x_4, r, s, t, u &\geq 0\end{aligned}$$

**Hint** There are 4 decision variables and 4 slack variables. Your initial tableau will need rows for  $r, s, t$  and  $u$ , and columns for  $x_1, x_2, x_3, x_4, r, s, t$  and  $u$ .

Figure 5b

4

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	$\theta$ values
$r$	1	4	3	1	1	0	0	0	95	95
$s$	2	1	2	3	0	1	0	0	67	$33\frac{1}{2}$
$t$	1	3	2	2	0	0	1	0	75	75
$u$	3	2	1	2	0	0	0	1	72	24
$P$	-4	3	-2	-3	0	0	0	0	0	

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	Row operations
$r$	0	$3\frac{1}{3}$	$2\frac{2}{3}$	$\frac{1}{3}$	1	0	0	$-\frac{1}{3}$	71	R1 - R4
$s$	0	$-\frac{1}{3}$	$\frac{4}{3}$	$1\frac{2}{3}$	0	1	0	$-\frac{2}{3}$	19	R2 - 2R4
$t$	0	$2\frac{1}{3}$	$1\frac{2}{3}$	$1\frac{1}{3}$	0	0	1	$-\frac{1}{3}$	51	R3 - R4
$x_1$	1	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	$\frac{1}{3}$	24	R4 $\div 3$
$P$	0	$5\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{1}{3}$	0	0	0	$\frac{4}{3}$	96	R5 + 4R4

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	$\theta$ values
$r$	0	$3\frac{1}{3}$	$2\frac{2}{3}$	$\frac{1}{3}$	1	0	0	$-\frac{1}{3}$	71	$26\frac{5}{8}$
$s$	0	$-\frac{1}{3}$	$\frac{4}{3}$	$1\frac{2}{3}$	0	1	0	$-\frac{2}{3}$	19	$14\frac{1}{4}$
$t$	0	$2\frac{1}{3}$	$1\frac{2}{3}$	$1\frac{1}{3}$	0	0	1	$-\frac{1}{3}$	51	$30\frac{3}{5}$
$x_1$	1	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	0	0	0	$\frac{1}{3}$	24	72
$P$	0	$5\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{1}{3}$	0	0	0	$\frac{4}{3}$	96	

4 continued

Basic variable	$x_1$	$x_2$	$x_3$	$x_4$	$r$	$s$	$t$	$u$	Value	Row operations
$r$	0	4	0	-3	1	-2	0	1	33	$R1 - 2\frac{2}{3}R2$
$x_3$	0	$-\frac{1}{4}$	1	$1\frac{1}{4}$	0	$\frac{3}{4}$	0	$-\frac{1}{2}$	$14\frac{1}{4}$	$\frac{3}{4}R2$
$t$	0	$2\frac{3}{4}$	0	$-\frac{3}{4}$	0	$-1\frac{1}{4}$	1	$\frac{1}{2}$	$27\frac{1}{4}$	$R3 - 1\frac{2}{3}R2$
$x_1$	1	$\frac{3}{4}$	0	$\frac{1}{4}$	0	$-\frac{1}{4}$	0	$\frac{1}{2}$	$19\frac{1}{4}$	$R4 - \frac{1}{3}R2$
$P$	0	$5\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$105\frac{1}{2}$	$R1 + \frac{2}{3}R2$

$$\text{Maximum } P = 105\frac{1}{2}, \text{ when } x_1 = 19\frac{1}{4}, x_2 = 0, x_3 = 14\frac{1}{4}, x_4 = 0, r = 33, s = 0, t = 27\frac{1}{4}, u = 0$$

Figure 6a

2 Maximise  $P = 3x + 4y + 10z$   
subject to

$$x + 2y + 2z + r = 100$$

$$x + 4z + s = 40$$

$$x, y, z, r, s \geq 0$$

Figure 6b (There were some slight formatting issues on the mark scheme, however all the necessary information is still clearly visible)

2

b.v.	$x$	$y$	$z$	$r$	$s$	value	θ values
$r$	1	2	2	1	0	100	50
$s$	1	0	4	0	1	40	10*
$P$	-3	-4	-10	0	0	0	

b.v.	$x$	$y$	$z$	$r$	$s$	value	Row operations
$y$	$\frac{1}{2}$	2	0	1	$-\frac{1}{2}$	80	R1 - 2R2
$z$	$\frac{1}{4}$	0	1	0	$\frac{1}{4}$	10	R2 ÷ 4
$P$	$-\frac{1}{2}$	-4	0	0	$\frac{5}{2}$	100	R3 + 10R2

b.v.	$x$	$y$	$z$	$r$	$s$	value	Row operations
$y$	$\frac{1}{4}$	1	0	$\frac{1}{2}$	$-\frac{1}{4}$	40	R1 ÷ 2
$z$	$\frac{1}{4}$	0	1	0	$\frac{1}{4}$	10	R2 (no change)
$P$	$\frac{1}{2}$	0	0	$\frac{3}{2}$	0	60	R3 + 4R1

$$P = 260 \quad x = 0 \quad y = 40 \quad z = 10 \quad r = 0 \quad s = 0$$

Figure 7a

3 Maximise  $P = 3x + 5y + 2z$   
subject to

$$3x + 4y + 5z + r = 10$$

$$x + 3y + 10z + s = 5$$

$$x - 2y + t = 1$$

$$x, y, z, r, s, t \geq 0$$

Figure 7b

3

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value	θ values
$r$	3	4	5	1	0	0	10	2.5
$s$	1	(3)	10	0	1	0	5	$1\frac{2}{3}^*$
$t$	1	-2	0	0	0	1	1	negative pivot
$P$	-3	-5	-2	0	0	0	0	

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value	Row operations
$r$	( $\frac{5}{3}$ )	0	$-\frac{25}{3}$	1	$-\frac{4}{3}$	0	$\frac{10}{3}$	R1 - 4R2
$y$	$\frac{1}{3}$	1	$\frac{10}{3}$	0	$\frac{1}{3}$	0	$\frac{5}{3}$	R2 ÷ 3
$t$	$\frac{5}{3}$	0	$\frac{20}{3}$	0	$\frac{2}{3}$	1	$\frac{13}{3}$	R3 + 2R2
$P$	$-\frac{4}{3}$	0	$\frac{44}{3}$	0	$\frac{5}{3}$	0	$\frac{25}{3}$	R4 + 5R2

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value	Row operations
$x$	1	0	-5	$\frac{3}{5}$	$-\frac{4}{5}$	0	2	$R1 \div \frac{5}{3}$
$y$	0	1	5	$-\frac{1}{5}$	$\frac{3}{5}$	0	1	$R2 - \frac{1}{3}R1$
$t$	0	0	15	-1	2	1	1	$R3 - \frac{5}{3}R1$
$P$	0	0	8	$\frac{4}{5}$	$\frac{3}{5}$	0	11	$R4 - \frac{4}{3}R1$

$$P = 11 \quad x = 2 \quad y = 1 \quad z = 0 \quad r = 0 \quad s = 0 \quad t = 1$$

I have now carried out a series of tests to test both the robustness of the user interface when handling a combination of valid and erroneous tests, as well as ensuring that accurate answers are produced when testing the examples and past exam questions. Any errors that arose while testing have been dealt with as remedial actions which were documented above. After any remedial actions were taken, they were then tested to ensure that the program now functions as

intended. With the system passing all the outlined tests, this iteration is now complete.

## **Iteration Summary and Review**

The development in this iteration focused on bringing the GUI developed in Iteration 1 together with the Simplex algorithm developed in Iteration 2. This involved a variety of methods to be able to take inputs from the GUI, format them in a way that can be processed by the methods from iteration 2, before outputting them on the GUI. To ensure success criterion 1 was met throughout this iteration, a heavy element of Design took place to ensure the GUI was formatted in a way that suited the users.

Having now tested the system against a variety of tests, it is shown that firstly the GUI is able to handle a variety of unexpected inputs that could be entered by end-users. This is important to ensure that there is no ambiguity when the user makes a mistake when providing inputs and that they can seamlessly continue, knowing what to do. Secondly, this iteration worked on success criterion 10 as well, ensuring that the Simplex algorithm was explained in a way that [SCHOOL NAME ABBREVIATION] students can both understand and use in an exam. Since this was the first iteration that involved bringing in explanation to the user interface, it was important that the topics of explanation were clearly identified as pre-requisites, which was done in the design.

After completing development, I presented this newly developed part of the tool to my stakeholders, to gain feedback on what works well and any improvements they would make. These were presented as open-ended questions, and the responses are shown below

Question	Student 1	Student 2	Student 3
Which features of the Simplex Algorithm do you think work well and are beneficial for you as a user	I think that the layout in which the tableaux are presented is unique and definitely works	I think that the whole explanation section is designed well and works well. Being able to see all the	I think that the method of taking inputs is well designed and replicates the mockup you

of the system?	well. Being able to click through each iteration makes it seem like I am going through the process of the algorithm. I also like the highlighting of the pivot row and column. I found them useful when revising from the textbook and I think they are beneficial in the system as well.	components in one go like the tableau, row operations, theta values and the explanation for the tableau makes it easier to follow each iteration. I also like the user of highlighting the pivot row and column and how the specific values are incorporated into the explanation.	Showed earlier that I thought was good. Being able to enter in my own inequalities and then have the various tableaux outputted and explained is very beneficial and I particularly like the use of colours to show the pivot row, pivot column and the pivot value
What features do you think need changing or improving and how?	Most of the system works really well together. The components are well placed and take up enough space that I can clearly see what's going on. If I had to make a suggestion it would be to maybe have a summary panel that can summarise the process either at the start or at the end just to solidify understanding.	The one suggestion I would make is potentially explaining the dash that appears in the theta values column. I assume some students will know that it means either the theta value is negative or infinity and doesn't need to be considered, but for those learning the algorithm for the first time, it may be useful to point that out.	One improvement I would make is just explaining some parts of the user interface that may not be clear for people that haven't seen the system before. For example telling them to click through the iterations, or why the theta values might have a dash in them. This would help round off the system to make it easier to understand for all [SCHOOL NAME ABBREVIATION]

			students.
--	--	--	-----------

From these questions, a particular feature that was well received was the use of colours to highlight the pivot row, column and value. The students felt that this helped enhance the explanation, so I will note that the use of colours in explanation is valued by my stakeholders. The main improvement to the system that was suggested is just rounding off some explanation about how the system works, like clicking through the tabs and how the theta values work with the dash. In future iterations, I will ensure that any ambiguous features are explained to the user to ensure the system remains easy to use and understand and fully achieves success criterion 1.

Having now designed, developed and tested the iteration, progress has been made on the success criteria, with criteria 8 being met allowing users to input their own inequalities and have them solved, as well as success criterion 12, with this feature added to the section labelled 'Run Simplex Algorithm', ensuring that the GUI remains clearly split up into separate sections that can be identified by the user and selected based on what part of the tool they wish to use. Hence, this iteration is now complete.

## Iteration 5: Creating Visual Graphs in the GUI

(Any reference to the Pearson ActiveLearn Further Maths Decision 1 Textbook throughout this iteration refers to this textbook provided by the exam board:  
[https://www.pearsonactivelearn.com/app/library/ebook?  
id=NzAzNzM1fGJvb2t8MTk5fDB8MA==](https://www.pearsonactivelearn.com/app/library/ebook?id=NzAzNzM1fGJvb2t8MTk5fDB8MA==))

This iteration combines elements of a graphical user interface as well as logic-based algorithms, hence will require thorough testing. Similar to previous iterations, testing will be split up into first ensuring functionality of the GUI and its components followed by testing the logic of the algorithms, making sure they function within the user interface. The first part of testing that focuses on the GUI will allow me to ensure that the input validation developed at the end of the

iteration is functioning as expected and that the program is able to recognise and handle unexpected inputs into the input fields. The second part of the testing will focus on the actual algorithm, which will involve me creating graphs from past exam questions before running the algorithm and checking the output against the official marking scheme for the question. This will ensure that the outputs produced by the program remain accurate and are suitable for use by a [SCHOOL NAME ABBREVIATION] Further Maths student to see how the algorithm works and have it explained to them by the system. This section will have a table covering the description of the test as well as the outcome and any remedial actions taken, as well as a second table to evidence proper functionality, provide updates on the code following remedial actions and displaying correct functionality of the system after any remedial actions have taken place. While some intermediary testing did take place during the development process to ensure I could move onto the next stage of development, most features will be tested once again after the full stage of development to ensure they still function as expected.

Test Number/Description	Test Type	Expected Outcome	Actual Outcome	Remedial Actions	Second Test (either to double check functionality or to check success of remedial actions)
1) Pressing Add Edge then inputting a source, destination and weight	Normal	An edge should be displayed on the left half of the screen, with two nodes and a line connecting	The edge was correctly displayed with two circles as the nodes, labelled with the inputted	No remedial actions required	Testing this again, the same correct output was produced. Two nodes were drawn on the GUI with a line

		them. The nodes should have labels and the line should have the weight in the center	text and the line connecting them was correctly labelled with the inputted weight.		connected them as the edge. All labels were also correctly added.
2) Adding two edges with a common node	Normal	Both edges should be outputted in the correct locations (forming a triangle shape with the angles since there are 3 nodes) and they should share the common node. Similar to test 1, all the labels should be correctly displayed	The two edges were correctly displayed in the left half of the panel in the triangular shape as expected.	No remedial actions required	I tested this again by inputting two edges that both shared a common node, and the program once again produced the successful output with the two edges and 3 nodes connected and displayed in the left half of the panel
3a) Inputting numbers into the source text area	Erroneous	3a) The error message label should be displayed	3a) The error message label was correctly displayed,	No remedial actions required	3a) Testing again, the same error label was displayed with the

		next to the buttons saying "Please enter source node as a single capital letter."	highlighted in red indicating the user's error with the correct message.		correct text, deeming this test successful
3b) Inputting numbers into the destination text area		3b) The error message label should be displayed on the window next to the buttons and read, "Please enter destination node as a single capital letter."	3) The Kruskal's error label was displayed on the graph as expected, highlighted in red and indicated that the user needs to enter a capital letter in the destination node text area.	3b) I tested again by inputting numbers into the destination text area while having valid inputs in all the rest and the correct output was once again produced with the same error label displayed.	
3c) Inputting multiple capital letters into the source and destination text areas		3c) The error message label should read the same message "Please	3c) The error message label was once again correctly displayed, indicating that the user needs to enter a	3c) I tested this once again inputting capital letters into the text area and the same output was produced with the	
3d) Not inputting anything into the source and destination text areas					

		<p>enter source node as a single capital letter.” depending on which text area the invalid input was entered into.</p> <p>3d) The error message label should be displayed and read “Please enter source node as a single capital letter.” as this is the first exception thrown in the try-catch loop. The user will then be told their error one-by-one e.g.</p>	<p>single capital letter into the text area.</p> <p>3d) The Kruskal’s error message label was correctly displayed next to the buttons and explained that the source node has to be entered as a single capital letter.</p>		<p>correct error message displayed</p> <p>3d) I tested this once again, this time inputting a value into the source text area but nothing in the destination text area. The correct output was produced with the message adapting to say, “Please enter source node as a single capital letter.”, deeming this test successful.</p>
--	--	---	--	--	---

		if they enter a value in the source but not the destination, it will tell them to enter a value in the destination.			
4a) Inputting a letter in the weight text area  4b) Inputting nothing in the weight text area	Erroneous	4a) The error message label should be displayed in red text to indicate that the weight must be entered as a valid number  4b) The error message label should be displayed in red text next to the buttons to indicate that the weight must be	The error message label was correctly displayed and had the correct message to indicate that the weight must be a valid number  Again the Kruskal's error message label was correctly displayed and had the correct message to indicate that weight should be	No remedial actions required	4a) Testing this again by inputting a letter in the weight text area produced the same correct output with the error message label  4b) I tested this again, inputting nothing in the weight text area while having valid inputs in the source and destination text areas

		inputted as a valid number	entered as a valid number		and the correct error message was displayed saying to enter the weight as a valid number.
5a) Adding the same edge twice to the graph  5b) Adding the same edge to the graph but entering the source and destination the other way around	Erroneous	5a) The error message label should be displayed to indicate that the edge trying to be inputted already exists in the graph.  5b) The error message label should be displayed in red text next to the row of buttons indicating that the edge is	5a) The Kruskal's error message label was correctly displayed on the GUI next to the buttons in red text and indicated that the edge already exists in the graph.  5b) The error message label was correctly displayed in red text next to the buttons	No remedial actions required	5a) I tested this again, this time entering the same edge with a different weight and the correct output was produced with the error message label indicating that the edge already exists.  5b) This time, I tested again entering the source and

		already in the graph.	and indicated that the edge I was trying to input was already in the graph.		destination the other way around and entering a different weight for the edge and the same correct result was produced with the error message label displayed next to the buttons.
6) Clicking the 'Remove Edge' button in the GUI	Normal	Most of the components of the GUI should remain the same, the only different should be the 'Source', 'Destination' and 'Weight' text areas as well as the 'Add' button from the	The correct output was produced, all the components were maintained , and the text areas were correctly replaced with that of the 'Remove Edge' option and the 'Remove'	No remedial actions required	Testing this again, the same correct output was produced with the graph being maintained on the panel and just the text areas changing to show the correct input fields for

		'Add Edge' option should be replaced with just a 'Source' text area, 'Destination' text area and a 'Remove' button	button was also correctly added to the kruskalAdd panel.		removing an edge, and the button to confirm the removal of the edge.
7) Removing an edge that was previously added to the graph	Normal	When inputting a source and destination node of an edge in the graph and selecting remove, the edge should be removed from the graph display and the data structures that store information about the edge	I inputted the edges AB, AC, BC and AD into the graph, before removing AD. The edge between A and D was removed however the node D was still present. This is generally the desired effect e.g. if I chose to remove BC, the node C should remain since it's connected	In the remove method I added the code to check that the inputted source and destination node are connected to other nodes. This ensures that if the deletion of an edge results in a node with no edges connected to it, the singular node can be deleted. If the user	Testing again with the same inputted edges and removed edge, the singular node that was left disconnected from the rest of the graph was removed from the panel and when I tried adding an edge that included the previously removed node, it was

			<p>to AC as well, but since D has no other edges connected to it, it should be removed</p>	<p>then decides to add another edge later which contains this node, it can be brought back, however, in the case that the removing the edge is the final action the user performs, it shouldn't be the case that the singular node is left on the GUI if the user doesn't intend for it to be there. as this may result in confusion with the input to the number of vertices in the graph as the</p>	<p>correctly brought back and displayed connected to the edges as intended, deeming the remedial actions successful in producing the expected outcome.</p>
--	--	--	--	---	--

				graph they want the algorithm to be performed on may only have $n-1$ vertices, but the actual number of nodes displayed and stored is $n$ , with the extra node not having any edges connected to it.	
8) Running the algorithm after removing an edge from the graph. To test, I will enter 4 edges, remove 1 and then run the algorithm	Normal	When running the algorithm after removing an edge, the removed edge should not be considered in the table of added and rejected	After pressing run algorithm, neither the table of added and rejected values, nor the tabbed pane considered the edge when explaining the	No remedial actions required	When testing again with a different set of edges, the same correct output was produced where only the edges still in the graph were included in the table of

		values, or be added as a tab in the tabbed pane as this would potentially result in the incorrect output from running the algorithm on the desired graph (with the edge removed).	algorithm, resulting in the correct output from the algorithm.		added and rejected edges and the tabs of edges in the tabbed pane for the explanation of the algorithm on the edges, deeming this test successful.
9) Entering edges into the graph then pressing the 'Reset' button	Normal	Upon pressing the 'Reset' button, the graph currently being displayed should be cleared, returning the user interface to the same state as when the user first pressed the 'Add Edge' button.	After entering a few edges into the graph, which were displayed on the GUI, I then pressed the reset button, which cleared the displayed graph leaving the GUI in the same state as it was	No remedial actions required	Testing this again, I entered another set of edges, and the same correct output was produced with the edges and nodes all being cleared from the GUI panel and the window being

			before I added any edges.		returned to the state it was in when I first pressed 'Add Edge'.
10) Moving the nodes of the graph around to change the shape of the graph.	Normal	When moving the nodes, they should remain within the black border square on the left half of the panel, the edges should follow the nodes and remain connected, the weight label should remain in the center of the edge and the node label should remain within the node circle.	The nodes successfully moved around and remained connected by the edges. The weight label and node label remained in the correct places and when dragging the nodes to the edge of the panel, they remained in the panel and the entire node circle was still visible on the edge of the panel.	No remedial actions required	I tested this again, moving the nodes to the corners of the panel and they all remained within the boundaries. The edge lines connected the nodes followed correctly and all the labels remained in the right place, deeming this test successful.
11) Entering a disconnect	Erroneous	When attempting to run the	After inputting a graph that	No remedial actions	This time I tested running the

ed graph to perform Kruskal's algorithm on		algorithm after creating a disconnected graph, the error message label should appear on the GUI next to the row of buttons and indicate the inputted graph must be connected	wasn't connected, I attempted to run the algorithm, and was met with the correct error message label indicating that the graph needs to be connected before running.	required	algorithm on a graph that was disconnected (outputting the correct error message label) and then adding an edge to make the graph connected. This resulted in the algorithm successfully running, the expected output.
12) Testing the highlighting of edges when tabs from the tabbed pane are selected	Normal	After entering a graph and running the algorithm, clicking through the edges of the graph should highlight them in either	When clicking through the edges, they were successfully highlighted in the correct colours, resulting in a minimum spanning tree,	In the stateChanged method, I altered the logic so that each time the tab is changed, it first sets all the edges to be coloured in black and	Testing this after the remedial actions produced the correct output. I was still able to click through the edges to see the graph being built

		<p>green (Add) or red (Reject), building up the minimum spanning tree.</p>	<p>however, when I reached the last edge, clicking back to a previous edge didn't return the highlighted state to that of the edges considered up to then and instead all of the edges remained highlighted .</p>	<p>then loops from the first edge to the current edge, colouring them in the correct colour (green or red). This ensures that only the edges that have been considered up to the point of the selected edge are highlighted and the user can return to an earlier edge and the graph changes to reflect the highlighting at this point. Setting the edges to black isn't noticed at all as the</p>	<p>up, but when I clicked back to a previous edge, it successfully returned the state of highlighted edges to that of only the edges considered up to the selected edge.</p>
--	--	--	---	--	--

				colours are all processed almost instantly. To the user it just appears as if they are building up in green and red.	
13) Chapter 3 Example 1 from Pearson ActiveLearn Decision 1 Textbook	Normal	The arcs added should be DE, AE, BC and BD and the total weight of the minimum spanning tree should be 20.	After inputting the graph, I adjusted the shape to look like the graph from the textbook before running the algorithm, which then produced a correct output, and the minimum spanning was highlighted to mimic the result from the textbook	No remedial actions required	Testing this again, the same correct output was produced, with the correct arcs and the total weight being 20. Clicking through the edge tabs also highlighted the correct edges in green and the rejected edges in red as expected.
14) Chapter 3, Question	Normal	The edges added to	Inputting the graph, I	No remedial	Testing the same

1a from Pearson ActiveLearn Decision 1 Textbook		the minimum spanning tree are EF, BD, CD, AH, DF, AC, GH and the other edges should be rejected. The minimum spanning tree should have a total weight of 98.	moved the vertices to make the graph look like the textbook question and ran the algorithm. The correct edges were added and rejected from the minimum spanning tree, shown by the add/reject table and the final output of included edges and total weight matched the mark scheme	actions required	graph again, the same output was correctly produced with all the correct edges included in the minimum spanning tree and the others being rejected. Clicking through the edge tabs also showed the minimum spanning tree being correctly built up with green highlighted edges for those added and red highlighted edges for those rejected.
15) Chapter 3, Question	Normal	The edges added to	Once again, I inputted	No remedial	I tested the same

1b from Pearson ActiveLearn Decision 1 Textbook		<p>the minimum spanning tree should be BF, FG, AB, CE, BC, CD and the rest should be rejected. The total weight of the minimum spanning tree should be displayed as 27.</p>	<p>the graph and made it look like the graph from the textbook question, before running the algorithm. Looking through the add/reject table, all the correct edges were selected to be added and the others rejected. The edges and total weight were correctly displayed</p>	actions required	<p>graph again and the same correct output was produced upon running the algorithm. When clicking through the edges in the tabbed pane, each of the correct added edges was highlighted in green and the rejected edges in red. I was also able to click back to a previous edge and have the highlighted edges return to the state of the edges considered up to that</p>
---	--	---	---	------------------	--

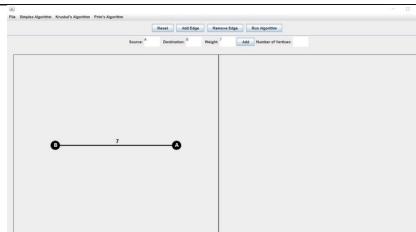
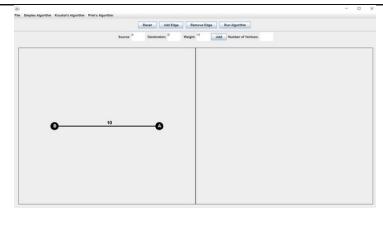
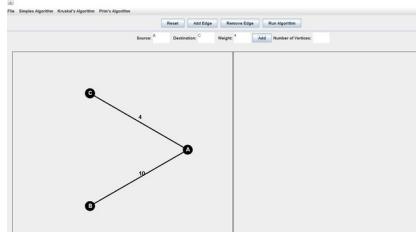
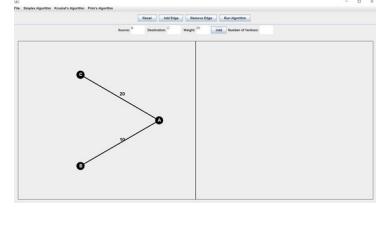
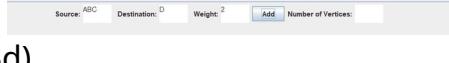
					point.
16) Chapter 3 Question 2b from Pearson ActiveLearn Decision 1 Textbook	Normal	The edges added to the minimum spanning tree should be YZ, VW, XY, UW, UX, SU, TU with a total weight of 121	Entering this graph and running the algorithm, the correct edges were highlighted in green one-by-one as I clicked through the edge tabs. The edges rejected from the graph were also correctly highlighted in red, and the final edges and weight were correctly displayed	No remedial actions required	Testing the same graph again, all the correct edges were added to the minimum spanning tree, and this was reflected by the table of added and rejected edges. Clicking through the edge tabs once again showed the minimum spanning tree being built up correctly and the final edges and weight were correctly outputted again.
17) Pearson Edexcel Decision 1	Normal	The edges included in the	Creating this graph then	No remedial actions	Testing with this graph

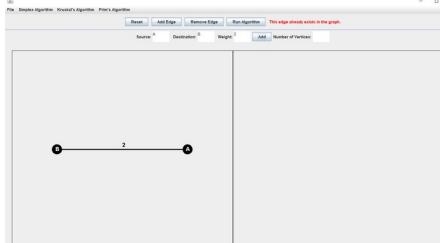
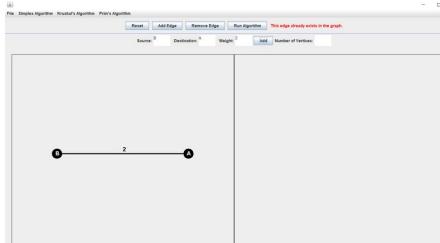
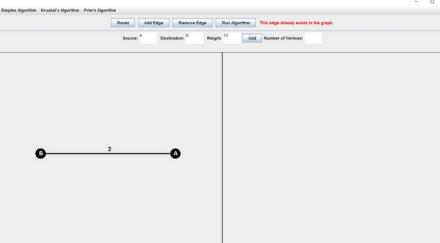
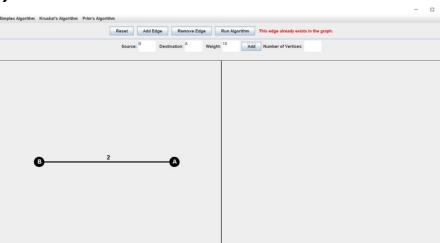
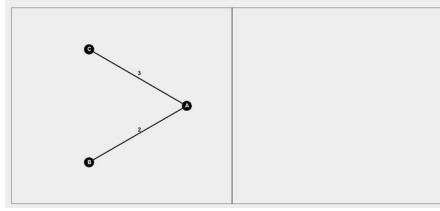
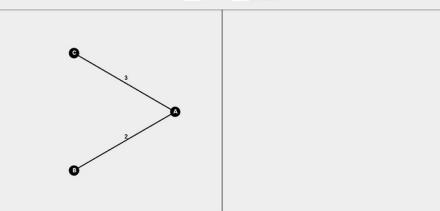
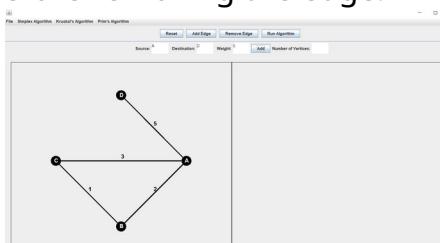
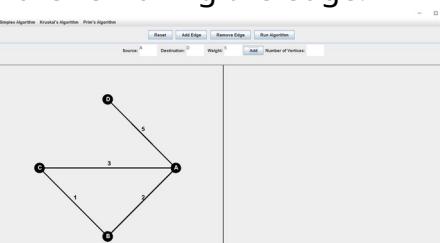
A-Level Exam Paper January 2005 Question 3ai		minimum spanning tree should be displayed as FH, AD, DE, CD, BC, CF, HI, IJ, with a total weight of 188 and the other edges should be rejected	running the algorithm, the correct output was produced with FH, AD, DE, CD, BC, CF, HI, IJ included in the minimum spanning tree and the others being rejected. The total weight was also correctly outputted as 188.	required	again, the same correct output was produced with the correct edges being added and rejected. Clicking through the edge tabs showed the minimum spanning tree being built up correctly with rejected edges highlighted in red. The add/reject table also correctly reflected which edges were added and which were rejected.
18) Pearson Edexcel Decision 1 A-Level	Normal	The edges included in the minimum	Entering the edges of this graph and	No remedial actions required	Testing the same graph once again

Exam Paper May 2010 Question 2a	spanning tree for this graph are DE, GF, DC, BD, EG, AC, GH with a total weight of 174	selecting 'Run Algorithm' produced the correct table of added/rejected edges and the correct result of included edges in the minimum spanning tree and the total weight.	produced the correct results with the edges DE, GF, DC, BD, EG, AC and GH being added to the minimum spanning tree and the rest rejected as well as the total weight of 174. Clicking through the edge tabs also showed the minimum spanning tree being correctly built up with the right edges being highlighted in green and the same for those rejected being
--	--	--	--

					highlighted in red.
--	--	--	--	--	---------------------

Outcome Table

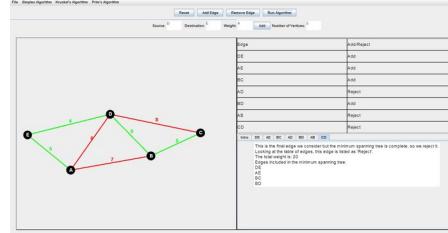
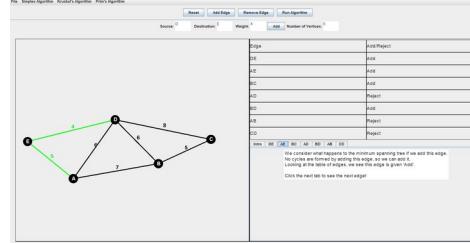
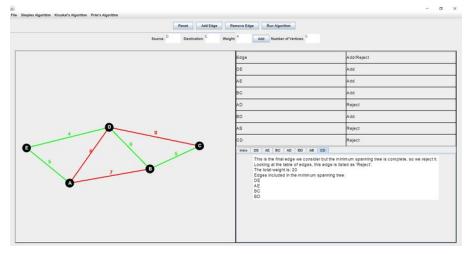
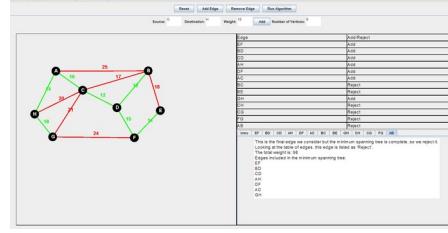
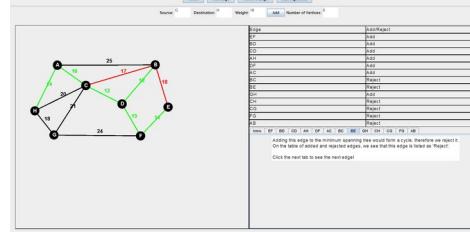
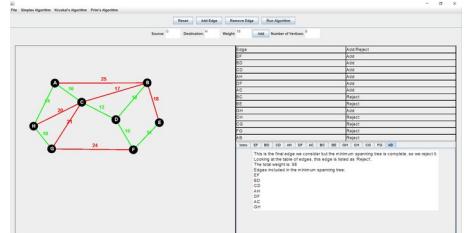
Test Number	Actual Outcome	Remedial Actions	Second Test (either to double check functionality or to check success of remedial actions)
1)		None required	
2)			
3a) 3b) 3c) 3d)	<p>3a)</p>  <p>3b)</p>  <p>3c)</p>  <p>3d)</p> 	No remedial actions required	<p>3a)</p>  <p>3b)</p>  <p>3c)</p>  <p>3d)</p> 

4a) 4b)	<p>4a)</p>  <p>4b)</p> 		<p>4a)</p>  <p>4b)</p> 
5a) 5b)	<p>5a)</p>  <p>5b)</p> 	<p>No remedial actions required</p>	<p>5a)</p>  <p>5b)</p> 
6)		<p>No remedial actions required</p>	
7)	<p>Before removing the edge:</p>  <p>After removing the edge:</p>	<p>Evidence d below this table</p>	<p>Before removing the edge:</p>  <p>After removing the edge:</p>

		<p>Introducing a new edge that contains the previously removed node:</p>								
8)	<p>Before removing the edge:</p> <p>After removing the edge:</p> <p>After running the algorithm</p> <table border="1"> <thead> <tr> <th>Edge</th> <th>Add/Reject</th> </tr> </thead> <tbody> <tr> <td>BC</td> <td>Add</td> </tr> <tr> <td>AB</td> <td>Add</td> </tr> <tr> <td>AC</td> <td>Reject</td> </tr> </tbody> </table>	Edge	Add/Reject	BC	Add	AB	Add	AC	Reject	<p>No remedial actions required</p>
Edge	Add/Reject									
BC	Add									
AB	Add									
AC	Reject									
9)	<p>Before clicking the 'Reset' button:</p>	<p>Before clicking the 'Reset' button:</p>								

	<p>After clicking the 'Reset' button:</p>	<p>After clicking the 'Reset' button:</p>
10)	<p>Before moving the nodes:</p> <p>After moving the nodes:</p>	<p>Before moving the nodes:</p> <p>After moving the nodes:</p>
11)	<p>Before connecting the graph:</p> <p>After connecting the graph:</p>	

12)	<p><b>Clicking through the edges one-by-one</b></p> <p><b>Evidence below outcome table</b></p>	<p><b>Evidence below outcome table</b></p> <p>The edges are first sorted into ascending weight order. This is the edge with the lowest weight. No cycles are formed by adding this edge to the minimum spanning tree, hence we add it. On the tabs of added and rejected edges, this edge is listed as 'Add'.</p> <p>Click the next tab to see the next edge!</p> <p><b>Outcome Table:</b></p> <table border="1"> <thead> <tr> <th>Edge</th> <th>Add/Reject</th> </tr> </thead> <tbody> <tr> <td>BC</td> <td>Add</td> </tr> <tr> <td>AB</td> <td>Add</td> </tr> <tr> <td>AC</td> <td>Add</td> </tr> <tr> <td>CD</td> <td>Reject</td> </tr> <tr> <td>BD</td> <td>Add</td> </tr> </tbody> </table> <p><b>Click the next tab to see the next edge!</b></p> <p><b>Outcome Table:</b></p> <table border="1"> <thead> <tr> <th>Edge</th> <th>Add/Reject</th> </tr> </thead> <tbody> <tr> <td>BC</td> <td>Add</td> </tr> <tr> <td>AB</td> <td>Add</td> </tr> <tr> <td>AC</td> <td>Add</td> </tr> <tr> <td>CD</td> <td>Reject</td> </tr> <tr> <td>BD</td> <td>Add</td> </tr> </tbody> </table> <p><b>Click the next tab to see the next edge!</b></p> <p><b>Outcome Table:</b></p> <table border="1"> <thead> <tr> <th>Edge</th> <th>Add/Reject</th> </tr> </thead> <tbody> <tr> <td>BC</td> <td>Add</td> </tr> <tr> <td>AB</td> <td>Add</td> </tr> <tr> <td>AC</td> <td>Add</td> </tr> <tr> <td>CD</td> <td>Reject</td> </tr> <tr> <td>BD</td> <td>Add</td> </tr> </tbody> </table> <p><b>Click the next tab to see the next edge!</b></p> <p><b>Outcome Table:</b></p> <table border="1"> <thead> <tr> <th>Edge</th> <th>Add/Reject</th> </tr> </thead> <tbody> <tr> <td>BC</td> <td>Add</td> </tr> <tr> <td>AB</td> <td>Add</td> </tr> <tr> <td>AC</td> <td>Add</td> </tr> <tr> <td>CD</td> <td>Reject</td> </tr> <tr> <td>BD</td> <td>Add</td> </tr> </tbody> </table> <p><b>Click the next tab to see the next edge!</b></p> <p><b>Outcome Table:</b></p> <table border="1"> <thead> <tr> <th>Edge</th> <th>Add/Reject</th> </tr> </thead> <tbody> <tr> <td>BC</td> <td>Add</td> </tr> <tr> <td>AB</td> <td>Add</td> </tr> <tr> <td>AC</td> <td>Add</td> </tr> <tr> <td>CD</td> <td>Reject</td> </tr> <tr> <td>BD</td> <td>Add</td> </tr> </tbody> </table> <p><b>Click the next tab to see the next edge!</b></p>	Edge	Add/Reject	BC	Add	AB	Add	AC	Add	CD	Reject	BD	Add	Edge	Add/Reject	BC	Add	AB	Add	AC	Add	CD	Reject	BD	Add	Edge	Add/Reject	BC	Add	AB	Add	AC	Add	CD	Reject	BD	Add	Edge	Add/Reject	BC	Add	AB	Add	AC	Add	CD	Reject	BD	Add	Edge	Add/Reject	BC	Add	AB	Add	AC	Add	CD	Reject	BD	Add
Edge	Add/Reject																																																													
BC	Add																																																													
AB	Add																																																													
AC	Add																																																													
CD	Reject																																																													
BD	Add																																																													
Edge	Add/Reject																																																													
BC	Add																																																													
AB	Add																																																													
AC	Add																																																													
CD	Reject																																																													
BD	Add																																																													
Edge	Add/Reject																																																													
BC	Add																																																													
AB	Add																																																													
AC	Add																																																													
CD	Reject																																																													
BD	Add																																																													
Edge	Add/Reject																																																													
BC	Add																																																													
AB	Add																																																													
AC	Add																																																													
CD	Reject																																																													
BD	Add																																																													
Edge	Add/Reject																																																													
BC	Add																																																													
AB	Add																																																													
AC	Add																																																													
CD	Reject																																																													
BD	Add																																																													
	<p><b>Returning to a previous edge, but the highlighted edges remain the same</b></p>	<p><b>Returning to a previous edge, and now the highlights update to reflect the state of the minimum spanning tree when only the edges up to the current edge have been considered.</b></p>																																																												

13)	<p>As shown in the explanation from the final tab, the correct edges are outputted, and the total weight is 20.</p>  <p>Refer to Figure 1b below to see the mark scheme answer</p>	<p>No remedial actions required</p>	<p>Example of the graph building up, showing one of the middle edges and the explanation</p>  <p>Final result:</p> 
14)	<p>As shown, the correct edges are outputted with the correct total weight</p>  <p>Refer to Figure 2b below to see the mark scheme</p>		<p>Showing an intermediary step, during the process of building up the minimum spanning tree:</p>  <p>Final answer:</p> 
15)	<p>The correct edges are outputted in the final tab and highlighted in green, with the correct total weight of 27:</p>	<p>No remedial actions required</p>	<p>Example of one of the tabs showing the tree being built up with explanation as to why the edge was rejected:</p>

	<p>Refer to Figure 3b below to see the mark scheme</p>	<p>Final answer:</p>
16)	<p>The correct edges are outputted and highlighted in green, with the total weight of 121 being correctly displayed on the final tab of the explanation edges:</p> <p>Refer to Figure 4b below for the mark scheme.</p>	<p>No remedial actions required</p> <p>Showing an intermediary explanation step, during the process of building up the minimum spanning tree:</p> <p>Final answer:</p>
17)	<p>The edges included in the minimum spanning tree are correctly listed out and highlighted in green, with the correct total weight of 188:</p>	<p>Showing an intermediate step, considering one of the middle edges which shows the currently considered edges highlighted and explains why the current edge was added:</p>

	<p>This is the first edge we consider for the minimum spanning tree. Looking at the table of edges, this edge is listed as 'Rejected'. This edge is included in the minimum spanning tree.</p> <p>AD AB BD CD DF EF IJ</p>	<p>The consideration happens to be the minimum spanning tree if we add this edge. The edge is included in the minimum spanning tree. Looking at the table of edges, this edge is listed as 'Rejected'. This edge is included in the minimum spanning tree.</p> <p>PA AD AC CD BC AC PF IJ CD</p>
	<p>Refer to Figure 5b below for the mark scheme</p>	<p>Final Answer:</p>
18)	<p>The correct edges included in the minimum spanning tree were outputted and highlighted in green, with the correct weight of 174 also outputted in the final edge explanation tab:</p> <p>This is the first edge we consider for the minimum spanning tree. Looking at the table of edges, this edge is listed as 'Rejected'. This edge is included in the minimum spanning tree.</p> <p>AD AB BD CD DF EF IJ</p>	<p>Showing an intermediate step, considering one of the middle edges. The edge is highlighted in red, and the explanation justifies why it was rejected. The add/reject table also matches with the final minimum spanning tree, stating which edges were added/rejected:</p> <p>Adding this edge to the minimum spanning tree would form a cycle, therefore we reject it. On the table of added and rejected edges, we see that the edge is listed as 'Rejected'. Click the next tab to see the next edge.</p>
	<p>Refer to Figure 6b for the mark scheme</p>	<p>Final Answer:</p>

## Remedial actions evidence

Test 7)

To fix the issue of a singular node remaining on the GUI after a connecting edge is removed, I implemented 2 boolean variables called isValidSourceNode and isValidDestNode which are initially set to false, but if the node is found in the list of edges, it means it is still connected to another node and is therefore valid and can remain in the GUI.

```
boolean isValidSourceNode = false;
for(int i = 0; i < toAdd.size(); i++){ //checking that the source node entered exists in the graph
    if(toAdd.get(i) == sourceChar){ //if the current node is the source node then it sets the boolean to true
        isValidSourceNode = true;
    }
}
```

If the node isn't valid i.e. it isn't connected to any others, it is then removed from the list of node labels and positions

```
if (!isValidSourceNode) {
    int nodeIndex = displayPanel.getNodeLabels().indexOf(sourceChar); // find the index of the source node in the nodeLabels list

    if (nodeIndex != -1) {
        displayPanel.getNodeLabels().remove(nodeIndex); //remove the node label and its corresponding position
        displayPanel.getNodePositions().remove(nodeIndex);

        displayPanel.revalidate(); // refresh the GUI
        displayPanel.repaint();
    }
}
```

The user interface is then refreshed to reflect the changes made to the GUI.

The same logic needs to be implemented for the destination node, as the singular node could be either the source or destination of the removed node. For example if 'D' is the singular node remaining, the user could have entered 'AD' and removed it or 'DA' and removed it.

```
boolean isValidDestNode = false;
for(int i = 0; i < toAdd.size(); i++){ //checking that the destination node entered exists in the graph
    if(toAdd.get(i) == destChar){ //if the current node is the destination node then it sets the boolean to true
        isValidDestNode = true;
    }
}
```

After setting the value of isValidDestNode, the same logic gets the node index in the list of node labels before removing it from the list of labels and positions.

```

if (!isValidDestNode) {
    int nodeIndex = displayPanel.getNodeLabels().indexOf(destChar); //find the index of the destination node in the nodeLabels list

    if (nodeIndex != -1) {
        displayPanel.getNodeLabels().remove(nodeIndex); // remove the node label and its corresponding position
        displayPanel.getNodePositions().remove(nodeIndex);

        displayPanel.revalidate(); // refresh the GUI to reflect the changes
        displayPanel.repaint();
    }
}

```

## Test 12)

To ensure that the highlighted edges are returned to the correct state after selecting a previous tab, I altered the logic inside the stateChanged method. I initially set all the edges to black, then iterated from the first edge to the current edge, only colouring those in the correct colour.

```

public void stateChanged(ChangeEvent e) {
    JTabbedPane sourcePane = (JTabbedPane) e.getSource(); //initialising a tabbed pane as the item that was updated/changed
    selectedTab = sourcePane.getSelectedIndex() - 1; //getting the index of the current selected tab

    for(String edge: displayPanel.getEdges()){ //iterating through list of edges
        displayPanel.highlightEdge(edge, Color.BLACK); //setting all edges to be highlighted in black
    }
    for(int i = 0; i < selectedTab + 1; i++){ //iterating through the edges
        Edge edge = graph.edges.get(i); //setting an edge object to the current edge
        if(mstEdges.contains(edge)){ //checking if the current edge is in the minimum spanning tree
            String edgeString = (char)(edge.source + 65) + "" + (char)(edge.dest + 65); //setting the string of the edge to the source + destination e.g. 'AB'
            displayPanel.highlightEdge(edgeString, Color.GREEN); //sets the edge colour to green
        } else{
            String edgeString = (char)(edge.source + 65) + "" + (char)(edge.dest + 65); //setting the string of the edge to the source + destination e.g. 'AB'
            displayPanel.highlightEdge(edgeString, Color.RED); //sets the edge colour to red if the edge is not in the MST
        }
    }
}

```

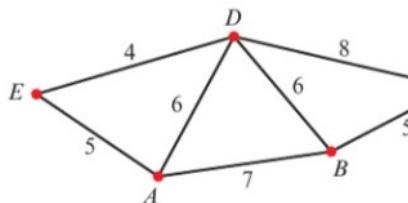
This allows me to click through all the edges, highlighting the entire graph, but then return to a previous edge and see the highlights change to how they looked when only the edges up to the current edge had been considered. Changing the edges to black and then the correct colour is not noticeable as this happens instantly, so the result is an instant changing of highlights to the correct stage.

## Questions and Mark Schemes Figures

Figure 1a)

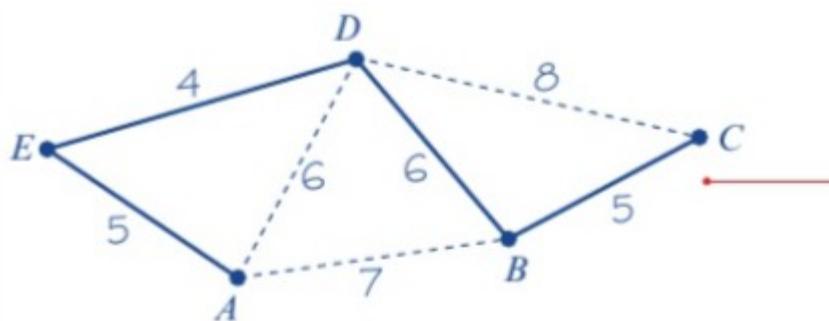
**Example 1**

Use Kruskal's algorithm to find a minimum spanning tree for this network. List the arcs in the order that you consider them. State the weight of your tree.



In a network of  $n$  vertices a spanning tree will always have  $(n - 1)$  arcs. In this case there will be 4 arcs in the spanning tree.

Figure 1b)



All vertices are connected so this is a minimum spanning tree.

Its weight is  $5 + 4 + 6 + 5 = 20$

Figure 2a)

- 1 Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

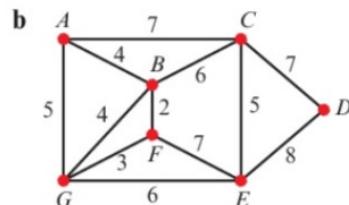


Figure 2b)

- 1 a**
- EF (11) add to tree
  - BD (12) add to tree
  - CD (12) add to tree
  - AH (14) add to tree
  - DF (15) add to tree
  - AC (16) add to tree
  - BC (17) reject
  - GH (18) add to tree
  - BE (18)
  - CH (20)
  - CG (21)
  - FG (24)
  - AB (25)
- } reject all remaining arcs.

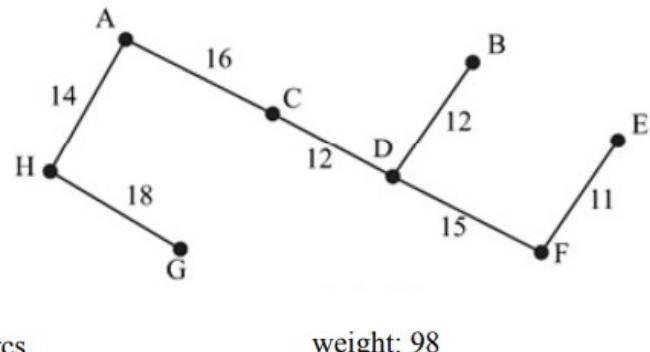


Figure 3a)

- 1** Use Kruskal's algorithm to find minimum spanning trees for each of these networks. State the weight of each tree. You must list the arcs in the order in which you consider them.

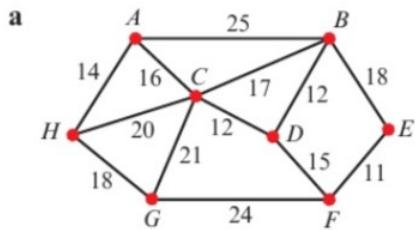


Figure 3b)

- b**
- BF (2) add to tree
  - FG (3) add to tree
  - AB (4) add to tree
  - BG (4) reject
  - AG (5) reject
  - CE (5) add to tree
  - BC (6) add to tree
  - EG (6) reject
  - AC (7) reject
  - CD (7) add to tree
  - EF (7)
  - DE (8)
- } reject all remaining arcs.

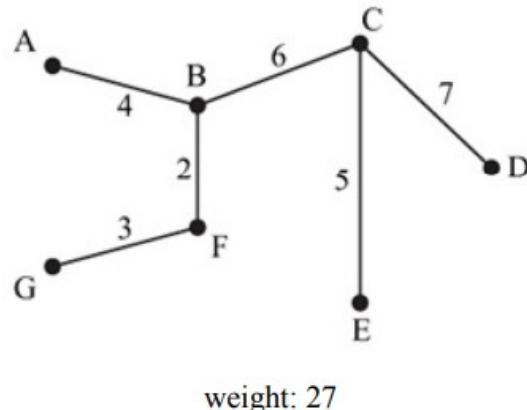


Figure 4a)

- E/P** 2 a State what is meant by:  
 i a tree  
 ii a minimum spanning tree.
- b Use Kruskal's algorithm to find a minimum spanning tree for this network.

(1 mark)

(1 mark)

(3 marks)

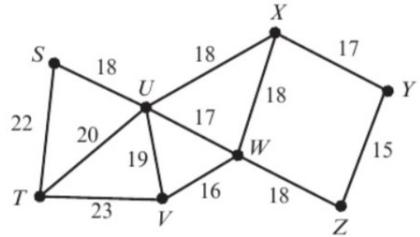


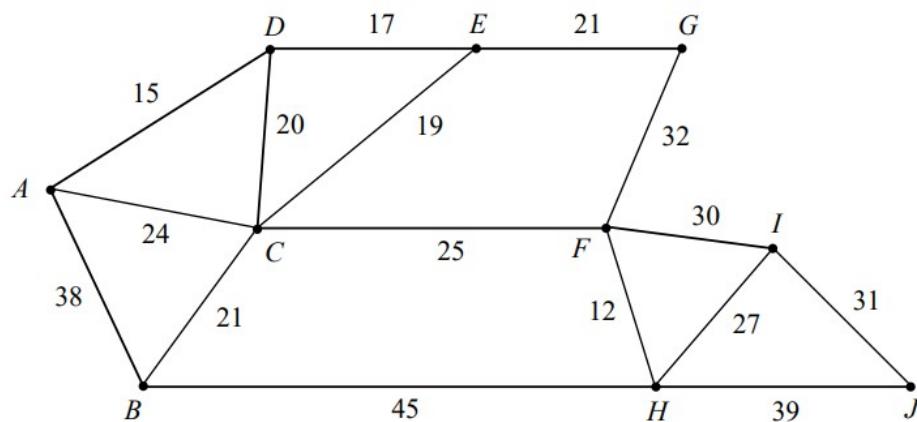
Figure 4b)

- 2 a i A tree is a connected graph with no cycles.  
 ii A minimum spanning tree is a tree of minimum total weight that connects all of the nodes.
- b By inspection the order of the arcs is  
YZ(15), VW(16), XY(17), UW(17), UX(18), WX(18), SU(18), WZ(18), UV(19), TU(20), ST(22), TV(23)  
 Underlined arcs are in the minimum spanning tree. Total weight = 121

Figure 5a)

3.

Figure 2



The network in Figure 2 shows the distances, in metres, between 10 wildlife observation points. The observation points are to be linked by footpaths, to form a network along the arcs indicated, using the least possible total length.

- (a) Find a minimum spanning tree for the network in Figure 2, showing clearly the order in which you selected the arcs for your tree, using

(i) Kruskal's algorithm,

(3)

Figure 5b)

3) (a)	(i) FH, AD, DE, CE, (not DC), {BC}, {EG}, (not AC), CF, HI, (not FI), IJ <sub>stop</sub>	MIAIAI (3)
--------	--	---------------

Summing these edge weights using the provided graph in the question results in a total weight of 188

Figure 6a)

2.

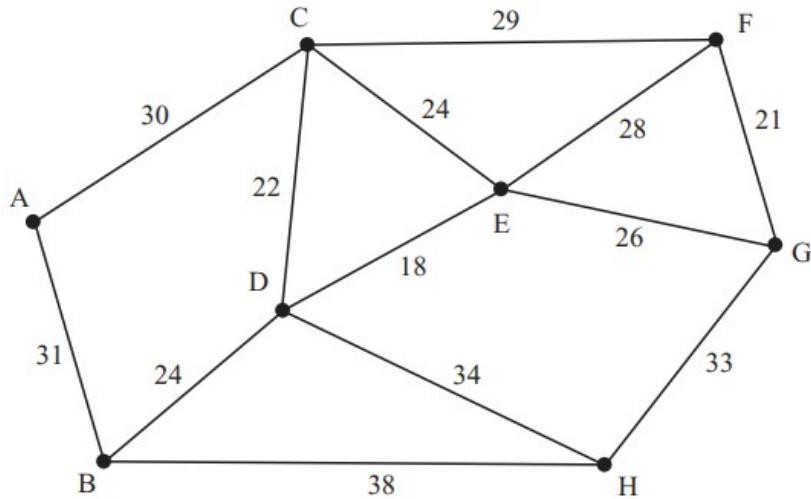


Figure 1

Figure 1 represents the distances, in metres, between eight vertices, A, B, C, D, E, F, G and H, in a network.

- (a) Use Kruskal's algorithm to find a minimum spanning tree for the network.

You should list the arcs in the order in which you consider them. In each case, state whether you are adding the arc to your minimum spanning tree.

(3)

Figure 6b)

Question Number	Scheme	Marks																																																																								
Q2 (a)	DE GF DC $\left\{ \begin{matrix} \text{not CE} \\ \text{BD} \end{matrix} \right\}$ EG (not EF not CF) AC (not AB) GH	M1 A1 A1  3																																																																								
(b)	<table border="1"> <tr><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td><td>H</td></tr> <tr><td>31</td><td>-</td><td>30</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> <tr><td>B</td><td>31</td><td>-</td><td>-</td><td>24</td><td>-</td><td>-</td><td>38</td></tr> <tr><td>C</td><td>30</td><td>-</td><td>-</td><td>22</td><td>24</td><td>29</td><td>-</td></tr> <tr><td>D</td><td>-</td><td>24</td><td>22</td><td>-</td><td>18</td><td>-</td><td>34</td></tr> <tr><td>E</td><td>-</td><td>-</td><td>24</td><td>18</td><td>-</td><td>28</td><td>26</td></tr> <tr><td>F</td><td>-</td><td>-</td><td>29</td><td>-</td><td>28</td><td>-</td><td>21</td></tr> <tr><td>G</td><td>-</td><td>-</td><td>-</td><td>26</td><td>21</td><td>-</td><td>33</td></tr> <tr><td>H</td><td>-</td><td>38</td><td>-</td><td>34</td><td>-</td><td>-</td><td>33</td></tr> </table>	A	B	C	D	E	F	G	H	31	-	30	-	-	-	-	-	B	31	-	-	24	-	-	38	C	30	-	-	22	24	29	-	D	-	24	22	-	18	-	34	E	-	-	24	18	-	28	26	F	-	-	29	-	28	-	21	G	-	-	-	26	21	-	33	H	-	38	-	34	-	-	33	B2, 1, 0  2
A	B	C	D	E	F	G	H																																																																			
31	-	30	-	-	-	-	-																																																																			
B	31	-	-	24	-	-	38																																																																			
C	30	-	-	22	24	29	-																																																																			
D	-	24	22	-	18	-	34																																																																			
E	-	-	24	18	-	28	26																																																																			
F	-	-	29	-	28	-	21																																																																			
G	-	-	-	26	21	-	33																																																																			
H	-	38	-	34	-	-	33																																																																			
(c)	AC CD DE BD GE GF GH	M1 A1 A1  3																																																																								
(d)	Weight: 174	B1  1																																																																								

Part (a) of the question didn't ask for the total weight, however this is a common follow up question, justifying why the weight is provided as part of the answer in the explanation text pane. The weight of the minimum spanning tree can be seen highlighted as part of the answer to (d).

I have now carried out a variety of tests both to test the robustness of the user interface when handling a combination of valid and erroneous tests, as well as ensuring mathematical accuracy with the various exam questions. Any errors that arose while testing have been dealt with as remedial actions, and they have been checked for functionality after to ensure the program works as intended. With the system passing all the outlined tests, this iteration is now complete.

## **Iteration Summary and Review**

This iteration focused on bringing in elements of the GUI with user interactivity and performing Kruskal's algorithm. The development that took place in this iteration acted as a bridge between Iterations 1 and 3, which worked in developing the user interface and performing Kruskal's algorithm respectively. This iteration completed success criterion 6, which outlined allowing users to enter their own graph upon which to perform a graph algorithm on. One key factor to reflect on at the end of this iteration is the slight alterations made to success criterion 6.1. This initially outlined having buttons to add and remove nodes from the graph, however, to ensure the system could simultaneously achieve success criterion 1 (focusing on a well-laid out, intuitive GUI), I decided to combine 6.1 and 6.2 into 2 buttons, one to add an edge and one to remove an edge. This still gives the user complete control of which nodes are in the graph, but reduces the possible additional burden on the user of constantly pressing different buttons to create nodes, then connect them and add a weight. The development of this iteration helped reduce this burden by automatically connecting the inputted nodes with an edge and adding the inputted weight to it.

Ensuring the GUI remained clearly laid out and intuitive was a key feature of this iteration, which constant consultation with the stakeholder to fully confirm any style choices regarding layout and colour schemes. The use of colour schemes was pivotal in this iteration, as it enhanced the explanation of the algorithm by visually displaying the build-up of the minimum spanning tree, hence confirming the choices of green and red was important.

This was the second iteration that proper explanation of the algorithm, which links into success criterion 10, explaining the algorithm in accordance with the Pearson Edexcel Further Maths specification. By outlining the steps of Kruskal's algorithm and incorporating explanation into the GUI (using the table of added and rejected edges), I was able to ensure that the user can gain valuable knowledge for their exam. When testing with past questions, all the elements of the GUI worked to ensure the question was completed and explained properly step-by-step, with successful final answers.

Improving this Iteration would involve elements from Iteration 4 of making the explanations seem more natural. Currently, the explanations for edges are split up into a few basic categories, regarding when the edge is considered and if it is added or rejected. If I were to build on this, I would implement a random number generator, the output of which would correspond to a differently worded explanation. This would help ensure that the explanation doesn't become too repetitive, as with graphs that contain many edges, the edges in the middle often have the same explanation as to why they were added or rejected.

Having now completed this iteration, I once again consulted my stakeholders to get their opinions on what they think is good and what could be improved upon. This was presented in the form of two open-ended question, and the responses are shown below:

Question	Student 1	Student 2	Student 3
Having seen the finished development of the Kruskal's algorithm tool, which features	I think that the graph being displayed on the GUI is effective and works well with being able to	I think the add and reject table is effective, since it's the main working-out method we're taught in lessons,	I think the graph display feature and the explanation feature work well in the tool. Being

<p>particularly stand out to you as beneficial?</p>	<p>move around the points to make it look how I want it. The highlighting of edges also works correctly in displaying the tree being built up as you go along the edges.</p>	<p>so being able to enter my own graph and then see how the working-out should look is beneficial. I also like the colouring of the edges as it adds a visual touch to the explanation.</p>	<p>able to click through the different edges and see them highlighted and then explained is good for learning the algorithm. I also like how the different vertices can be moved around to make the graph more customised. Finally, the add/reject table is useful in showing the specific working-out method to get the marks for the question.</p>
<p>Having seen the development of the Kruskal's algorithm tool, what features would you like to see improved and in what way?</p>	<p>Most features are good and work well to perform the algorithm. If I had to suggest an improvement, I would say that working on the explanation would be a nice change, perhaps making it seem more like a gradual process since currently, the majority of the edges have the same response.</p>	<p>The only improvement I would suggest is improving on the explanation for the different edges. I think it's good that different edges have different explanations, like the first edge always mentions sorting the edges in ascending weight order, and the final one</p>	<p>Pretty much all of the features align with my needs for this part of the tool. It works well at demonstrating the algorithm and showing the tree being built up. A recommendation would be to improve the explanation, particularly for some of the intermediate edges that are</p>

		contains the answer, but the middle ones could have some variation in response.	repetitive at the moment.
--	--	---	---------------------------

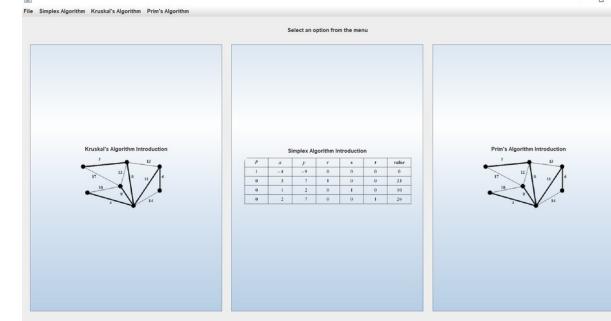
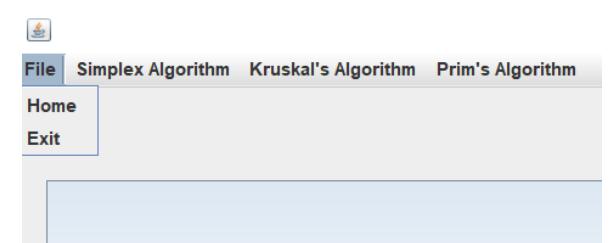
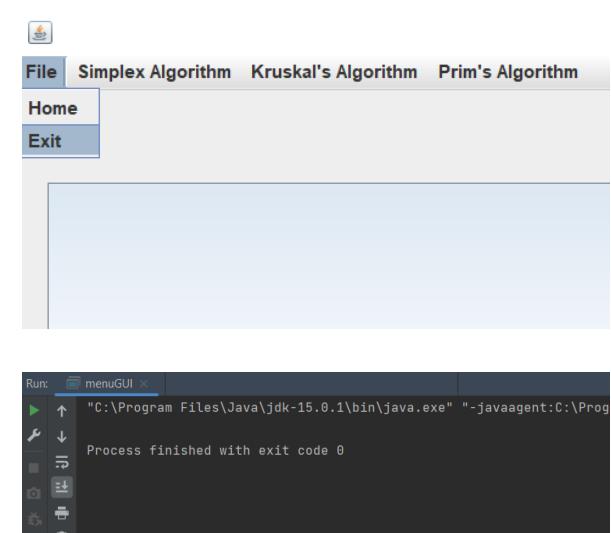
These responses highlight that most of the features in this development are effective in meeting their needs for performing the algorithm on a custom graph made by them, however all three students highlighted the explanation could use some work, mainly in the middle edges, by providing a bit more variation. This was also highlighted by me as a possible improvement in future iterations.

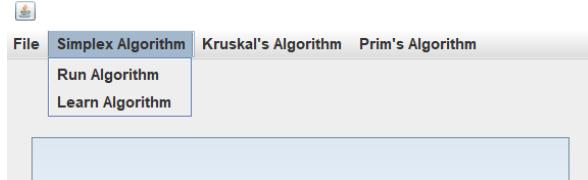
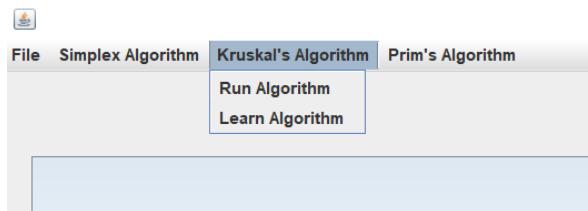
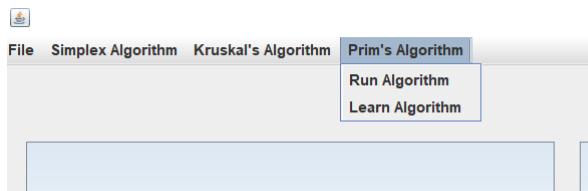
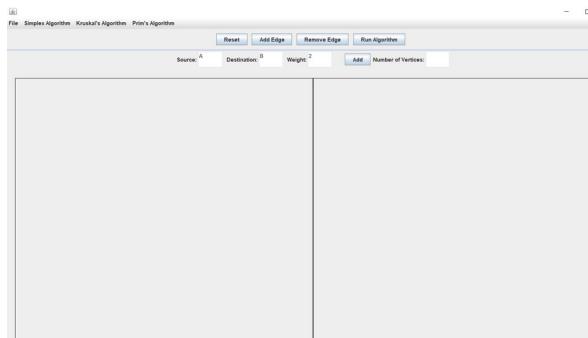
Having fully developed the feature to allow users to enter custom graphs to perform the algorithm on and have them explained step by step, as well as ensuring the system is able to handle any unexpected inputs, and finally consulting my stakeholders about what they like/would like to see improvement upon, iteration 5 is now complete.

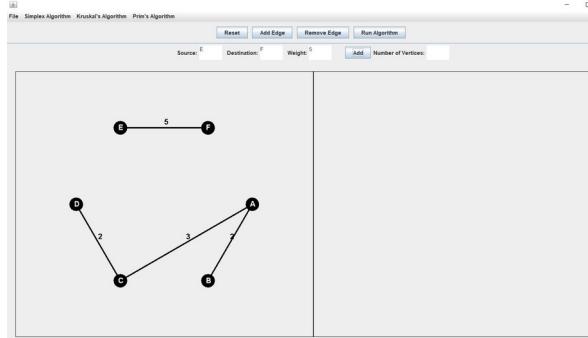
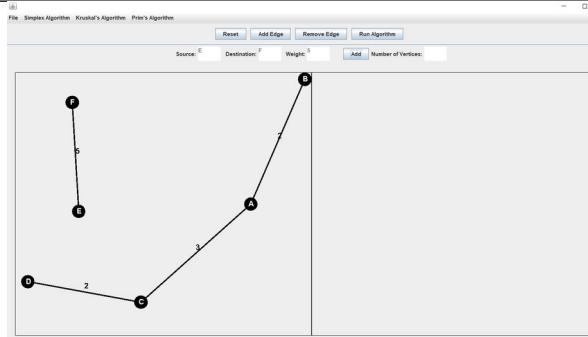
## Final Testing/Evaluation

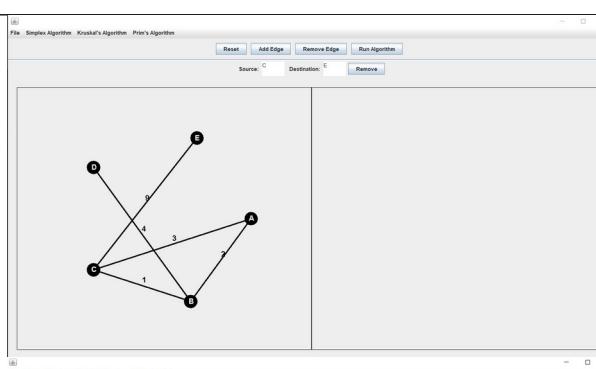
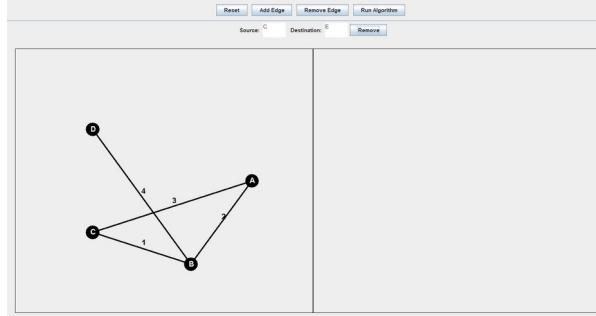
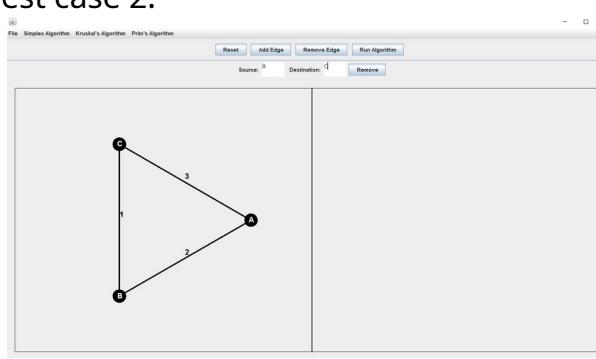
### Function and Robustness

The tests for function and robust will be categorised into normal, boundary and erroneous tests where applicable aimed at ensuring that the developed features are able to handle a variety of user inputs, imitating user interaction with the system.

Test Description	Expected Outcome	Outcome
1	Running the program should open a window to the 'Home' screen, with the 3 introductory buttons displayed and the menu bar correctly displayed.	 <p>Successful</p>
2	Clicking the 'File' option in the menu bar should show the two menu sub-options 'Home' and 'Exit'	 <p>Successful</p>
3	Clicking the 'Exit' option in the 'File' menu item should close the window	 <p>Successful</p>
4	Clicking the 'Simplex' option in the menu bar	

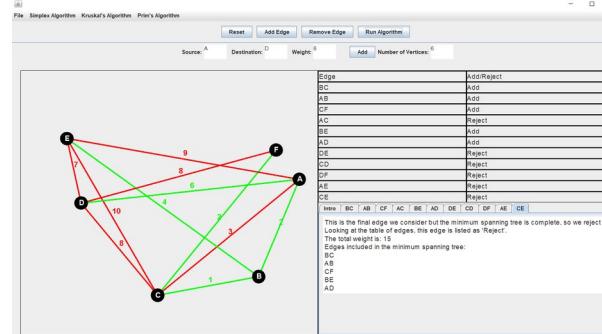
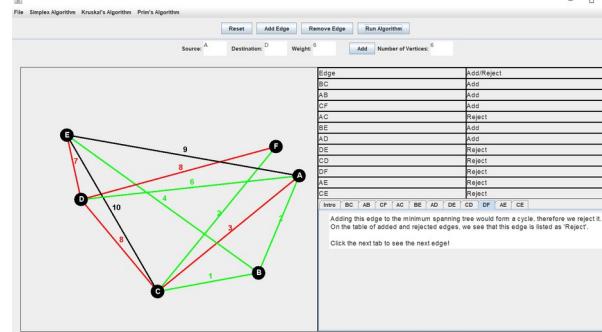
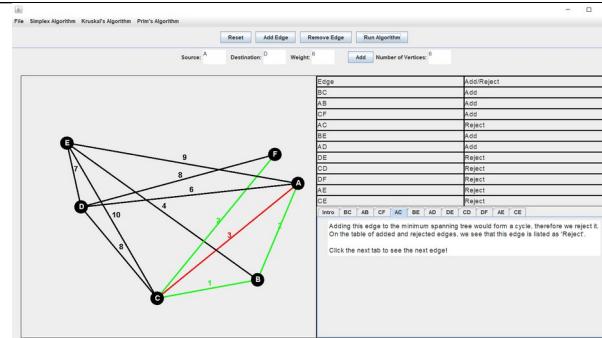
	should show the two menu sub-options 'Run Algorithm' and 'Learn Algorithm'	 Successful
5	Clicking the 'Kruskal's' option in the menu bar should show the two menu sub-options 'Run Algorithm' and 'Learn Algorithm'	 Successful
6	Clicking 'Prim's Algorithm' option in the menu bar should show the two menu sub-options 'Run Algorithm' and 'Learn Algorithm'.	 Successful
7	In the 'Run Algorithm' section of the Kruskal's Algorithm menu item, the user should be able to add an edge to the graph with the following process: -Selecting the 'Add Edge' button -Entering a source and destination	 Successful

	<p>node into the text areas</p> <p>-Selecting 'Add' This should result in two circles, labelled as the nodes, connected by a line, labelled with the edge weights, displayed in the left half of the screen.</p>	
8	<p>Adding more nodes to the graph should display the correct node labels and weights on the GUI.</p> <p>Adding the node AC should connect AB and AC at the node A, instead of creating a duplicate node 'A'. Testing with nodes 'AC (3)', 'CD (2)', 'EF (5)'.</p>	 <p>Successful</p>
9	<p>Having created a graph, I should be able to move around any of the vertices within the bounds indicated by the black border. The edges between the moving node and the connected</p>	 <p>Successful</p>

	nodes should remain consistent with the changing position of the node, as well as the node labels and the weight labels.	
10	Selecting the 'Remove Edge' button, entering a source and destination then pressing 'Remove' should remove the entered edge from the graph. This test has two separate cases, one involves removing an edge that leaves a node disconnected from the rest of the graph, and the other involves removing a node that is still connected to other nodes. In the former case, the independent node should be removed from the graph, and in the latter case, it should remain, since it still has connections to	  <p>Successful</p> <p>Test case 2:</p> 

	other nodes	
11	Clicking the 'Run Algorithm' button after entering a connected graph should preserve the displayed graph, as well as display the table of added and rejected values from the minimum spanning tree, below which should be a tabbed pane which contains each edge of the graph as a separate edge.	<p>Successful</p>
12	After running the algorithm on the created graph, clicking through the edge tabs of the tabbed pane should begin to highlight the edges in the drawn graph,	

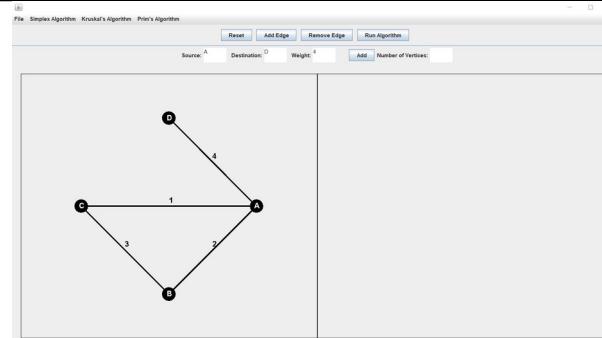
either green if it was added to the minimum spanning tree or red if it was rejected. The colouring should align with the reasoning displayed in the tab as well as the table showing the added/rejected edges above.

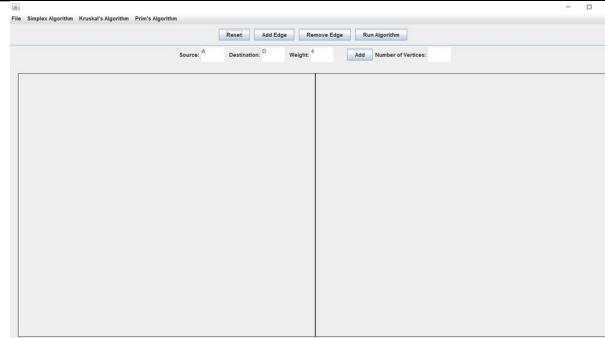
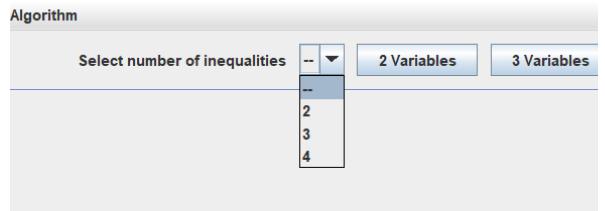
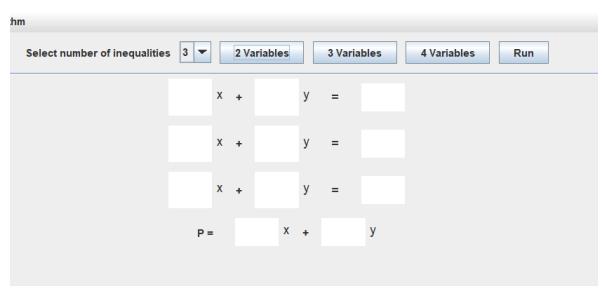
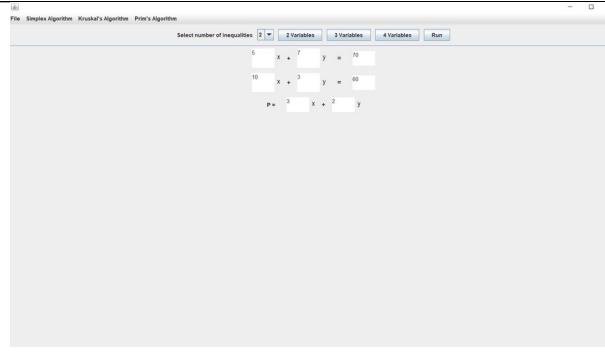


Successful

13

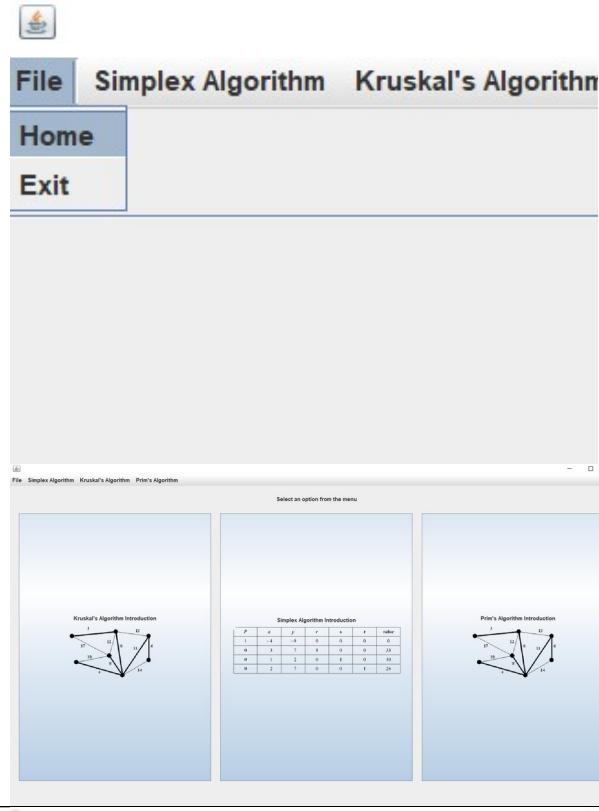
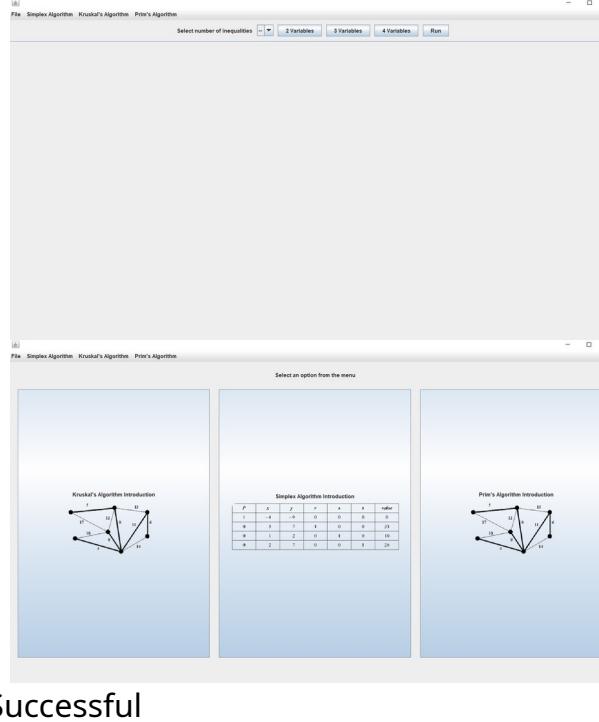
When drawing the graph, pressing the 'Reset' button should remove all the edges of the graph from the GUI, allowing the user to begin entering them again.

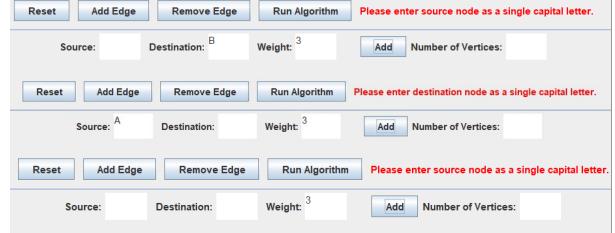
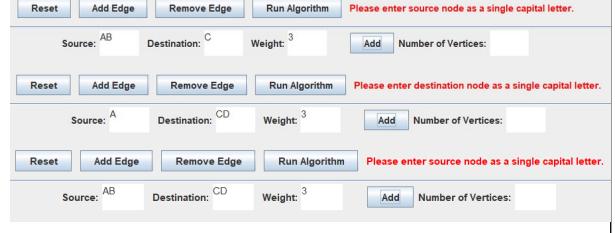


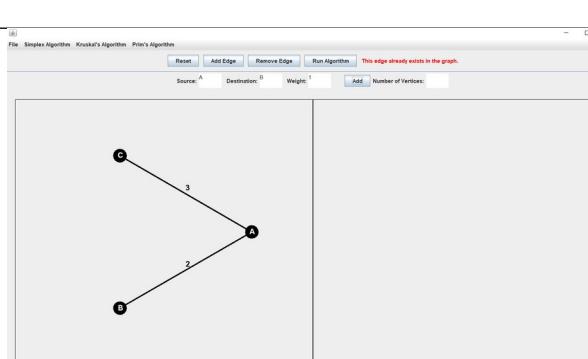
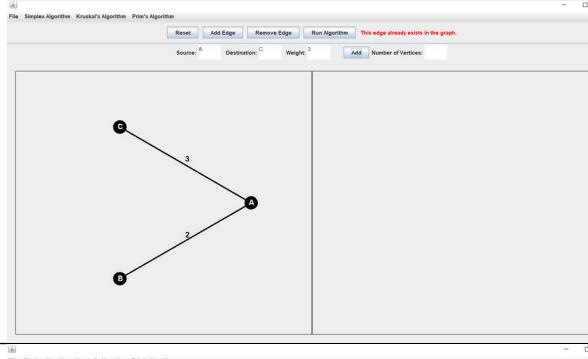
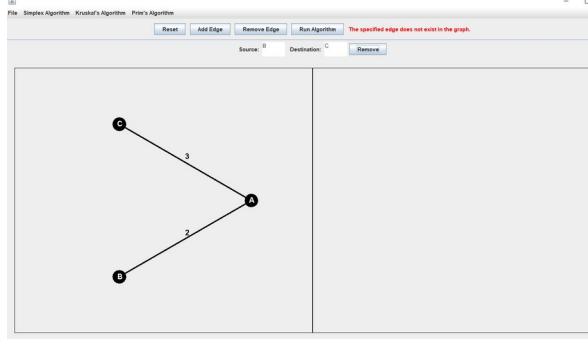
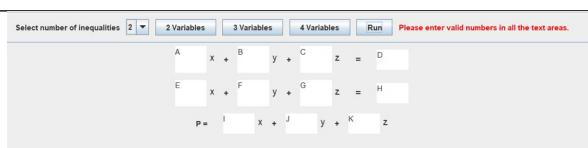
		
		Successful
14	After selecting the 'Run Algorithm' sub-option in the Simplex Algorithm menu item, I should be able to select a specific number of inequalities using the drop-down menu, select a number of variables using one of the 3 buttons, and the outcome should be the specified number of rows and variables as text areas to allow for inputs.	  Successful
15	Inputting variables into the various text areas for the inequalities and the objective function and then pressing the 'Run' button should display the tabbed	

	<p>pane in the lower middle section of the window. The first tab should contain the initial tableau filled in based on the inputted inequalities, the theta values and the explanation below the table.</p>	
16	<p>Clicking through each tab representing each iteration should change the tableau to reflect the row operations carried out. The subsequent iterations should have the tableau, row operations, theta value and</p>	<p><b>Successful</b></p> <p><b>With 2 inequalities:</b></p> <p><b>Successful</b></p> <p><b>With 3 inequalities:</b></p>

	<p>explanation for how the tableau was reached, and what needs to be performed next on the tableau. The pivot column, pivot row and pivot value should be highlighted in each tableau apart from the final tableau, which will not require any further iterations</p>	
17	<p>Following on from test 16, the final tableau of the iteration should have the final tableau, without any highlighted values, and should display the final values of each variable as part of the explanation for the tableau, as well as the final maximised profit.</p>	<p>With 2 inequalities</p> <p>Successful</p> <p>With 3 inequalities</p> <p>Successful</p>

18	<p>Having selected the 'Run Algorithm' sub-option in the Kruskal's Algorithm menu item, the home option in 'File' should correctly return the the GUI to the initial state 'home screen' that was displayed when the window was first opened.</p>	
19	<p>Having Selected the 'Run Algorithm' sub option in the Simplex Algorithm menu item, the home option in 'File' should correctly return the GUI to the initial state 'home screen' that was displayed when the window was first opened.</p>	
20	<p>Inputting numbers into the source and destination text</p>	

	areas for the nodes in the Kruskal's algorithm should display the red error message label on the GUI, mentioning that all inputs for source and destination node must be a single capital letter.	Successful
21	Inputting nothing into the source and destination text areas for the nodes in the Kruskal's algorithm should display the red error message label on the GUI, mentioning that all inputs for source and destination node must be a single capital letter.	
22	Inputting multiple capital letters into the source and destination nodes should result in the red error message being shown indicating only a single capital letter	

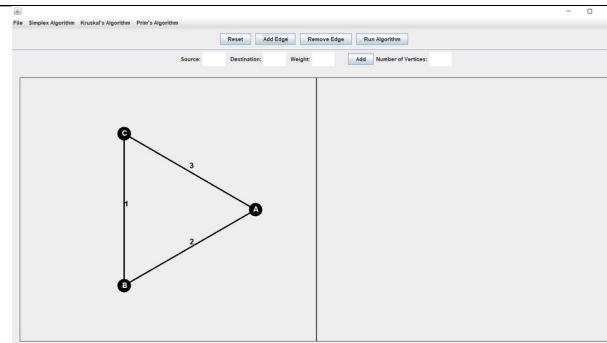
	should be outputted	
23	Inputting an edge that has already been added to the graph should result in the error message being shown indicating the edge already exists in the graph.	 
24	Removing an edge that does not exist in the graph should result in the error message label being shown indicating that the edge that is trying to be removed is not in the graph.	
25	Inputting letters into the text areas for the Simplex algorithm should result in the red error message label being shown to indicate that the user must enter valid numbers only.	

26	<p>Inputting nothing into the text areas for the Simplex algorithm should result in the red error message label being shown to indicate that the user must enter valid numbers only.</p>	<p>The screenshot shows a software interface for solving linear programming problems using the Simplex method. At the top, there's a dropdown menu labeled "Select number of inequalities" with the value set to 2. Below it are buttons for "2 Variables", "3 Variables", "4 Variables", and "Run". A red error message "Please enter valid numbers in all the text areas." is displayed prominently. The main area contains three rows of text input fields for equations. The first row has fields for x, y, z, and a constant, with the constraint sign being a plus sign. The second row has similar fields with a plus sign. The third row shows a partial equation "P = x + y + z" where the first two terms have plus signs and the last term has a plus sign. All fields are currently empty.</p>
27	<p>Inputting valid numbers into some of the text areas, and nothing in the rest should result in the red error message label being shown to indicate that the user must enter valid numbers only.</p>	<p>The screenshot shows a software interface for solving linear programming problems using the Simplex method. At the top, there's a dropdown menu labeled "Select number of inequalities" with the value set to 2. Below it are buttons for "2 Variables", "3 Variables", "4 Variables", and "Run". A red error message "Please enter valid numbers in all the text areas." is displayed prominently. The main area contains three rows of text input fields for equations. The first row has a coefficient 1, followed by plus signs for x, y, and z, and a constant 2. The second row has a coefficient 3, followed by plus signs for x, y, and z, and a constant 4. The third row shows a partial equation "P = 5 x + y + 6 z" where the first term has a coefficient 5, the second term has a coefficient 1, and the third term has a coefficient 6. The constraint signs for the first two terms are plus signs, and for the third term is a plus sign. Some fields contain invalid input like '5' or '6'.</p>
28	<p>Inputting valid numbers into some of the text areas, and letters in the rest should result in the red error message label being shown to indicate that the user must enter valid numbers only.</p>	<p>The screenshot shows a software interface for solving linear programming problems using the Simplex method. At the top, there's a dropdown menu labeled "Select number of inequalities" with the value set to 2. Below it are buttons for "2 Variables", "3 Variables", "4 Variables", and "Run". A red error message "Please enter valid numbers in all the text areas." is displayed prominently. The main area contains three rows of text input fields for equations. The first row has a coefficient 1, followed by plus signs for x, y, and z, and a constant 2. The second row has a coefficient C, followed by plus signs for x, y, and z, and a constant 4. The third row shows a partial equation "P = 5 x + E y + 6 z" where the first term has a coefficient 5, the second term has a letter E, and the third term has a coefficient 6. The constraint signs for the first two terms are plus signs, and for the third term is a plus sign. Some fields contain invalid input like 'C' or 'E'.</p>
29	<p>Selecting the '2 Variables' button before selecting</p>	<p>The screenshot shows a software interface for solving linear programming problems using the Simplex method. At the top, there's a dropdown menu labeled "Select number of inequalities" with the value set to 2. Below it are buttons for "2 Variables", "3 Variables", "4 Variables", and "Run". A red error message "Please select the number of inequalities first." is displayed prominently. The main area is empty.</p>

	the number of inequalities should display the red error message label indicating to select the number of inequalities first.	
30	Selecting the '3 Variables' button before selecting the number of inequalities should display the red error message label indicating to select the number of inequalities first.	<p>Select number of inequalities <input type="button" value="2"/> <input type="button" value="3"/> <input type="button" value="4"/> Run Please select the number of inequalities first.</p>
31	Selecting the '4 Variables' button before selecting the number of inequalities should display the red error message label indicating to select the number of inequalities first.	<p>Select number of inequalities <input type="button" value="2"/> <input type="button" value="3"/> <input type="button" value="4"/> Run Please select the number of inequalities first.</p>
32	Trying to run without having selected the number of inequalities or the number of variables should	<p>Select number of inequalities <input type="button" value="2"/> <input type="button" value="3"/> <input type="button" value="4"/> Run Please select the number of inequalities.</p>

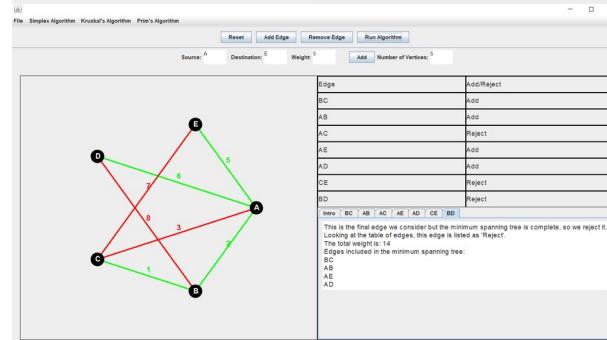
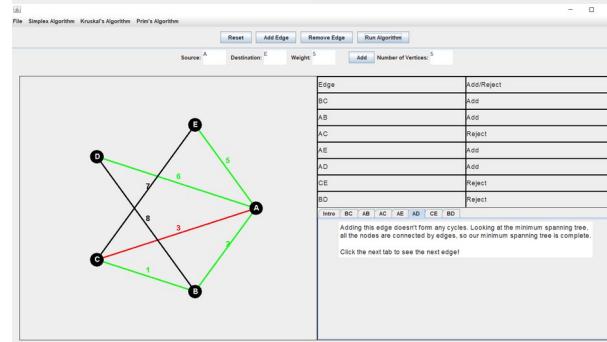
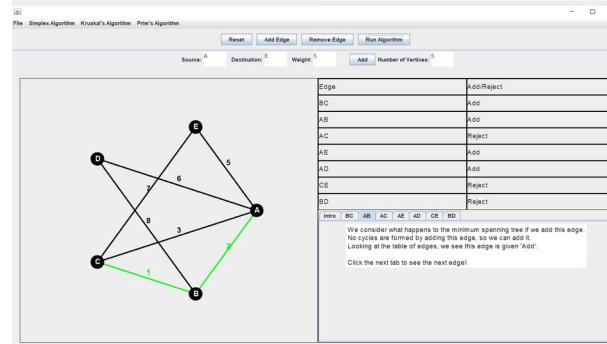
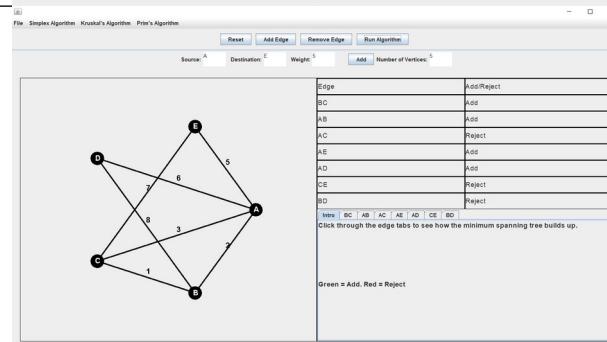
	display the red error message label indicating that the user must first input the number of inequalities.	
33	Trying to run after only selecting the number of inequalities and not the number of variables should display the red error message label indicating that the user must select the number of variables.	<p>Select number of inequalities 2 2 Variables 3 Variables 4 Variables Run Please select the number of variables.</p>
34	Selecting the 'Run Algorithm' option for Kruskal's algorithm, adding an edge and then selecting 'Run Algorithm' in the Simplex Algorithm menu item, before returning to the Kruskal's algorithm option should preserve the graph already created. (Generally, going off and back on to the Run Algorithm page should preserve the	<p>Source: B Destination: C Weight: 1 Add Number of Vertices:</p> <p>Select number of inequalities 2 2 Variables 3 Variables 4 Variables Run</p> $\begin{array}{l} x + y = \\ x + y = \\ p = x + y \end{array}$

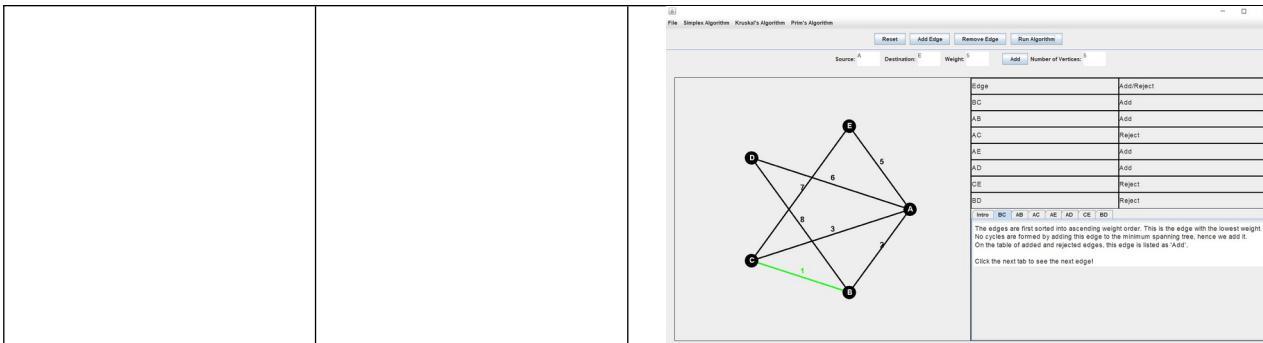
inputted data).



35

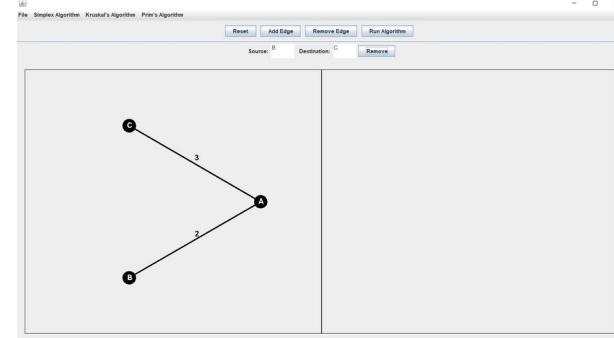
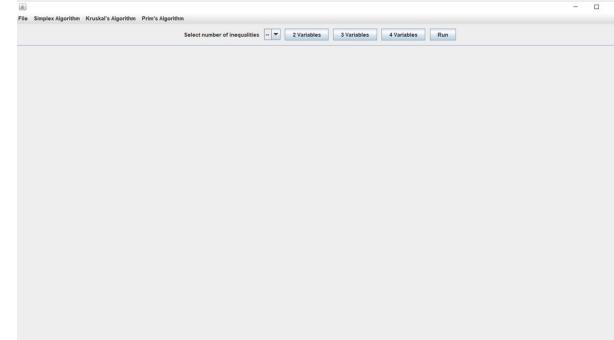
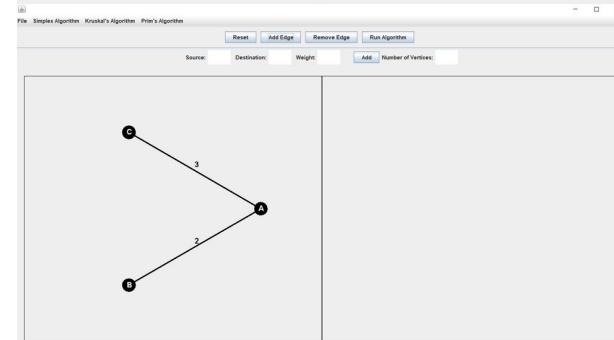
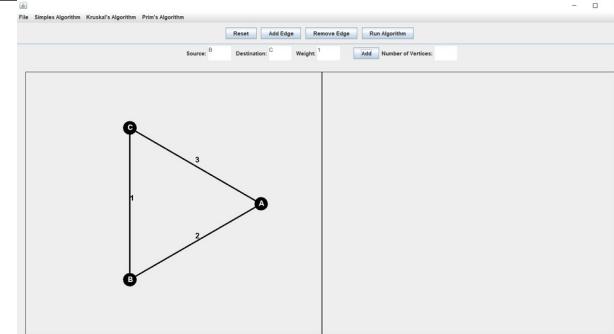
After running Kruskal's algorithm on an inputted graph, clicking through the tabs for each edge should highlight the edges in order and after clicking to the nth edge, I should be able to click to the mth edge where  $m < n$  and the state of highlighted edges should return to that of when only  $m$  edges had been considered.





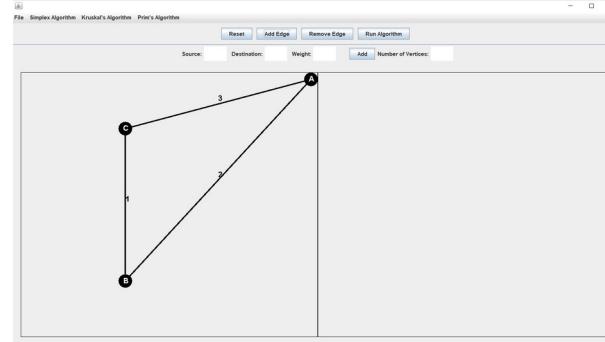
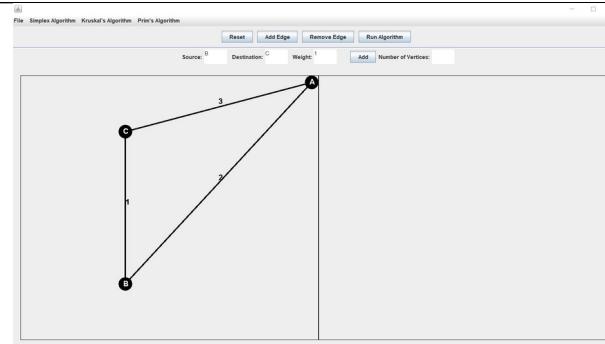
36

Drawing a graph in the Kruskal's algorithm feature and then removing an edge, before clicking off and back onto the option like in Test 34, should preserve the graph with the edge removed.



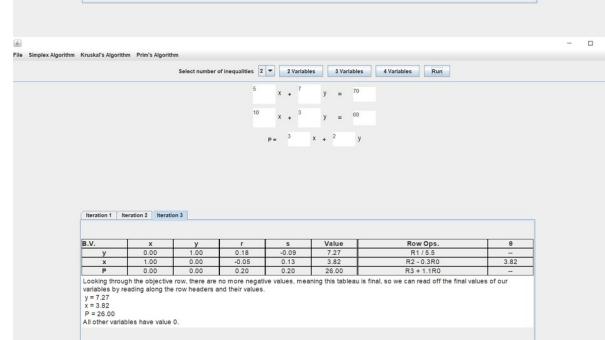
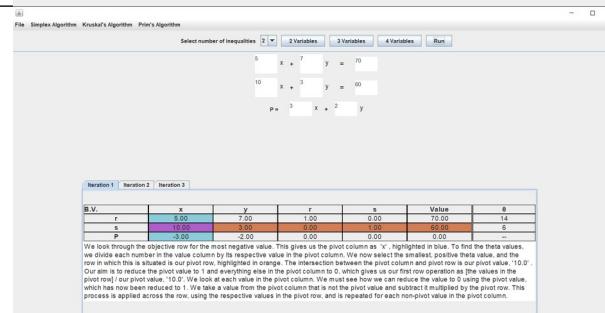
37

Drawing a graph in the Kruskal's algorithm feature and then moving a node around by dragging it, before clicking off and back onto the option (as in test 34), should preserve the position of the graph with the moved node(s).



38

After inputting inequalities and an objective function into the ‘Run Algorithm’ option for Simplex and then running to display the tableau, I should be able to click through the tabs to show each iteration and should be able to click back to a



	previous iteration and have the correct tableau, theta values, row operations (if applicable) and explanation for the given tableau.	
39	Having run the Simplex algorithm on inputted inequalities and an objective function, then clicking off the Simplex Algorithm option and back onto it, should clear the previously completed problem, allowing the user to start with a new problem.	
40	In the explanation panel for each iteration of the Simplex algorithm, selecting the explanation text	

	and trying to alter it e.g. deleting it should not let the user change the contents of the explanation	
41	In the explanation tab for each iteration of the Kruskal's algorithm, selecting the explanation text and trying to alter it e.g. deleting it should not let the user change the contents of the explanation	
42	Creating a graph in the 'Run Algorithm' feature of Kruskal's algorithm, before selecting the 'Run Algorithm' feature for the Simplex algorithm and completing a Simplex problem before returning to the Kruskal's graph should preserve the graph as created before selecting the Simplex Algorithm option.	

43	Running the algorithm on a created graph, before clicking the 'Reset' button should clear the graph and the method working to return the Kruskal's Run Algorithm feature back to its original format before a graph a created.	 

Testing functionality of algorithms:

Test	Outcome	Answer/Mark Scheme
------	---------	--------------------

1

8. The tableau below is the initial tableau for a maximising linear programming problem.

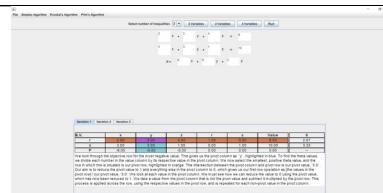
Basic Variable	$x$	$y$	$z$	$r$	$s$	Value
$r$	2	3	4	1	0	8
$s$	3	3	1	0	1	10
$P$	-8	-9	-5	0	0	0

(a) For this problem  $x \geq 0$ ,  $y \geq 0$ ,  $z \geq 0$ . Write down the other two inequalities in the objective function.

(b) Solve this linear programming problem.

(c) State the final value of  $P$ , the objective function, and of each of the variables.

## Edexcel Decision Maths 1 January 2003 Question 8



b	x	y	z	r	s	Value
$r$	2	(3)	4	1	0	8
$x$	3	3	1	0	1	10
$P$	-8	-9	-5	0	0	0

M1

A1

2

6. The tableau below is the initial tableau for a maximising linear programming problem.

Basic variable	$x$	$y$	$z$	$r$	$s$	$t$	Value
$r$	0	10	1	1	0	0	3600
$s$	6	9	12	0	1	0	3600
$t$	2	3	4	0	0	1	2400
$P$	-35	-55	-60	0	0	0	0

(a) Use the simplex method to find the optimal solution. M1

- (b) Taking the most negative number in the profit row to indicate the pivot column at each stage, solve this linear programming problem. State the row operations that you use. A1

- (c) State the values of the objective function and each variable. A1

(3)

## Edexcel Decision Maths 1 June 2006 Question 6

The Simplex Algorithm - Initial's Simplex - Profit Optimisation							
Initial tableau of inequalities							
	$x$	$y$	$z$	$r$	$s$	$t$	
$r$	0	10	1	1	0	0	3600
$s$	6	9	12	0	1	0	3600
$t$	2	3	4	0	0	1	2400
$P$	-35	-55	-60	0	0	0	0

The Simplex Algorithm - Initial's Simplex - Profit Optimisation							
Initial tableau of inequalities							
	$x$	$y$	$z$	$r$	$s$	$t$	
$r$	0	10	1	1	0	0	3600
$s$	6	9	12	0	1	0	3600
$t$	2	3	4	0	0	1	2400
$P$	-35	-55	-60	0	0	0	0

The Simplex Algorithm - Initial's Simplex - Profit Optimisation							
Initial tableau of inequalities							
	$x$	$y$	$z$	$r$	$s$	$t$	
$r$	0	10	1	1	0	0	3600
$s$	6	9	12	0	1	0	3600
$t$	2	3	4	0	0	1	2400
$P$	-35	-55	-60	0	0	0	0

(b)							
$b.v$	$x$	$y$	$z$	$r$	$s$	$t$	Value
$r$	2	$\frac{1}{10}y_1$	0	1	$\frac{1}{6}y_2$	0	600
$s$	$2y_1$	$3y_2$	0	0	$\frac{1}{6}y_2$	$\frac{1}{2}y_1$	$R_1 + 10R_2$
$t$	0	0	0	0	$\frac{1}{6}y_2$	1	$R_2 + 12R_1$
$P$	-5	-10	0	1	$\frac{1}{6}y_2$	$\frac{1}{2}y_1$	$R_1 + 4R_2$

$m_1$   
 $A_1$   
 $m_1$   
 $A_1 \wedge$   
 $B_1$   
 $(c)$

$$(c) P = 20400 \quad x = 0 \quad y = 240 \quad z = 120 \quad r = 0 \quad s = 0 \quad t = 1200$$

$m_1$   
 $A_1 \wedge$   
 $m_1$   
 $A_1$   
 $M_1$   
 $A_1 \wedge$   
 $A_1$   
 $(c)$

The mark scheme starts awarding marks at the second tableau, since the first is already given in the question, however for illustration purposes, the first tableau has also been evidenced in the 'actual outcome'. Looking at the first tableau in the mark scheme, this corresponds to the second tableau in the outcome, matching each entry with correct row operations, M1, A1, M1, A1, B1.

Looking at the second tableau in the mark scheme, which corresponds with the third tableau in the outcome, each entry and the row operations all match, another M1, A1, M1, A1.

Finally, looking at the final values of each of the variables, which match that of the mark scheme, providing a further M1, A2

3		13/13																																																																																				
<p>4. A three-variable linear programming problem in <math>x</math>, <math>y</math> and <math>z</math> is to be solved. The objective is to maximise the profit <math>P</math>. The following initial tableau was obtained.</p> <table border="1"> <thead> <tr> <th>Basic variable</th> <th><math>x</math></th> <th><math>y</math></th> <th><math>z</math></th> <th><math>r</math></th> <th><math>s</math></th> <th><math>t</math></th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><math>r</math></td> <td>2</td> <td>0</td> <td>4</td> <td>1</td> <td>0</td> <td>0</td> <td>80</td> </tr> <tr> <td><math>s</math></td> <td>1</td> <td>4</td> <td>2</td> <td>0</td> <td>1</td> <td>0</td> <td>160</td> </tr> <tr> <td><math>P</math></td> <td>-2</td> <td>-8</td> <td>-20</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> <p>(a) Taking the most negative number in the profit row to indicate the pivot column, perform one complete iteration of the simplex algorithm, to obtain tableau <math>T</math>. State the row operations that you use. (5)</p> <p>(b) Write down the final simplex tableau. (4)</p> <p>(c) State whether tableau <math>T</math> is optimal. Give a reason for your answer. (4)</p>	Basic variable	$x$	$y$	$z$	$r$	$s$	$t$	Value	$r$	2	0	4	1	0	0	80	$s$	1	4	2	0	1	0	160	$P$	-2	-8	-20	0	0	0	0																																																						
Basic variable	$x$	$y$	$z$	$r$	$s$	$t$	Value																																																																															
$r$	2	0	4	1	0	0	80																																																																															
$s$	1	4	2	0	1	0	160																																																																															
$P$	-2	-8	-20	0	0	0	0																																																																															
<p><b>Edexcel Decision Maths 1</b> <b>January 2007 Question 4</b></p>		<p>This question only required one iteration of the algorithm. Looking through the rows of the tableau, we see that each value from the mark scheme tableau matches my second tableau which represents one iteration being performed, giving us M1, A1 for the first row, M1, A1 for the second row and A1 for the final row.</p>																																																																																				
<p>6. The tableau below is the initial tableau for a maximising linear programming problem in <math>x</math>, <math>y</math> and <math>z</math>.</p> <table border="1"> <thead> <tr> <th>Basic variable</th> <th><math>x</math></th> <th><math>y</math></th> <th><math>z</math></th> <th><math>r</math></th> <th><math>s</math></th> <th><math>t</math></th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><math>r</math></td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> <td>0</td> <td>64</td> </tr> <tr> <td><math>s</math></td> <td>1</td> <td>3</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>16</td> </tr> <tr> <td><math>t</math></td> <td>4</td> <td>2</td> <td>2</td> <td>0</td> <td>0</td> <td>1</td> <td>60</td> </tr> <tr> <td><math>P</math></td> <td>-5</td> <td>-7</td> <td>-2</td> <td>-4</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> <p>(a) Taking the most negative number in the profit row to indicate the pivot column at each stage, perform two complete iterations of the simplex algorithm. State the row operations you use. (9)</p> <p>(b) Explain how you know that your solution is optimal. (4)</p> <p>(Total 10 marks)</p>	Basic variable	$x$	$y$	$z$	$r$	$s$	$t$	Value	$r$	4	3	2	1	0	0	64	$s$	1	3	0	0	1	0	16	$t$	4	2	2	0	0	1	60	$P$	-5	-7	-2	-4	0	0	0		<p>5/5</p> <table border="1"> <thead> <tr> <th>b.v.</th> <th><math>x</math></th> <th><math>y</math></th> <th><math>z</math></th> <th><math>R</math></th> <th><math>s</math></th> <th><math>t</math></th> <th>value</th> </tr> </thead> <tbody> <tr> <td><math>r</math></td> <td>4</td> <td>5</td> <td>2</td> <td>5</td> <td>1</td> <td>0</td> <td>0</td> <td>64</td> </tr> <tr> <td><math>s</math></td> <td>1</td> <td>3</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>16</td> </tr> <tr> <td><math>t</math></td> <td>4</td> <td>2</td> <td>2</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>60</td> </tr> <tr> <td><math>P</math></td> <td>-5</td> <td>-7</td> <td>-2</td> <td>-4</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> <p>M1 A1 M1 A1ft A1 M1 A1ft M1 A1</p>	b.v.	$x$	$y$	$z$	$R$	$s$	$t$	value	$r$	4	5	2	5	1	0	0	64	$s$	1	3	0	0	0	1	0	16	$t$	4	2	2	0	0	0	1	60	$P$	-5	-7	-2	-4	0	0	0	0
Basic variable	$x$	$y$	$z$	$r$	$s$	$t$	Value																																																																															
$r$	4	3	2	1	0	0	64																																																																															
$s$	1	3	0	0	1	0	16																																																																															
$t$	4	2	2	0	0	1	60																																																																															
$P$	-5	-7	-2	-4	0	0	0																																																																															
b.v.	$x$	$y$	$z$	$R$	$s$	$t$	value																																																																															
$r$	4	5	2	5	1	0	0	64																																																																														
$s$	1	3	0	0	0	1	0	16																																																																														
$t$	4	2	2	0	0	0	1	60																																																																														
$P$	-5	-7	-2	-4	0	0	0	0																																																																														
<p><b>Edexcel Decision Maths 1</b> <b>June 2008 Question 6</b></p>		<p>This question also only asked for two iterations of the algorithm. No marks are awarded for the initial tableau as this is provided in the question. Looking through the second tableau, which represents the first iteration of the</p>																																																																																				

algorithm, each row matches the mark scheme, awarding M1, A1, M1, A1ft (follow through), A1. Looking through the tableau for the second iteration, each row matches the mark scheme, giving a further M1, A1ft, M1, A1.

9/9

4

7. The tableau below is the initial tableau for a linear programming problem in  $x$ ,  $y$  and  $z$ . The objective is to maximise the profit,  $P$ .

basic variable	$x$	$y$	$z$	$r$	$t$	Value
$r$	12	4	5	0	0	246
$s$	9	6	3	1	0	153
$t$	5	2	-2	0	1	171
$P$	-2	-4	-3	0	0	0

Using the information in the tableau, write down

- (a) the three constraints required for integer optimisation. (9)

Taking the most negative number in the profit row to indicate the pivot column at each stage,

- (c) solve this linear programming problem. Make your method clear by stating the row operations you use. (9)

- (d) State the final values of the objective function and each variable. (3)

## Edexcel Decision Maths 1 June 2007 Question 7

Initial tableau - Maximise Profit						
basic variable	$x$	$y$	$z$	$r$	$t$	Value
$r$	12	4	5	0	0	246
$s$	9	6	3	1	0	153
$t$	5	2	-2	0	1	171
$P$	-2	-4	-3	0	0	0

Method 1: Row Operations 1

The first row has the most negative value in the profit row. This gives us the pivot column as highlighted in blue. To find the pivot element, we divide the first row by 12. The value in the first column of the first row is 12, so we divide the entire row by 12. This gives us the new first row. We then take the value from the pivot column that is on the new first row and multiply it by every other row. This gives us the new second and third rows. We then take the value from the pivot column that is on the new second row and multiply it by every other row. This gives us the new third row. We then take the value from the pivot column that is on the new third row and multiply it by every other row. This gives us the new fourth row.

Iteration 1 - Maximise Profit						
basic variable	$x$	$y$	$z$	$r$	$t$	Value
$r$	1	0	5/12	0	0	246
$s$	9/12	1	0	0	0	153
$t$	5/12	0	-2	0	1	171
$P$	-2/12	-4/12	-3/12	0	0	0

Method 2: Row Operations 1

The first row has the most negative value in the profit row. This gives us the pivot column as highlighted in blue. To find the pivot element, we divide the first row by 12. The value in the first column of the first row is 12, so we divide the entire row by 12. This gives us the new first row. We then take the value from the pivot column that is on the new first row and multiply it by every other row. This gives us the new second and third rows. We then take the value from the pivot column that is on the new second row and multiply it by every other row. This gives us the new third row. We then take the value from the pivot column that is on the new third row and multiply it by every other row. This gives us the new fourth row.

Iteration 1 - Maximise Profit						
basic variable	$x$	$y$	$z$	$r$	$t$	Value
$r$	1	0	5/12	0	0	246
$s$	9/12	1	0	0	0	153
$t$	5/12	0	-2	0	1	171
$P$	-2/12	-4/12	-3/12	0	0	0

Method 3: Row Operations 1

The first row has the most negative value in the profit row. This gives us the pivot column as highlighted in blue. To find the pivot element, we divide the first row by 12. The value in the first column of the first row is 12, so we divide the entire row by 12. This gives us the new first row. We then take the value from the pivot column that is on the new first row and multiply it by every other row. This gives us the new second and third rows. We then take the value from the pivot column that is on the new second row and multiply it by every other row. This gives us the new third row. We then take the value from the pivot column that is on the new third row and multiply it by every other row. This gives us the new fourth row.

Iteration 1 - Maximise Profit						
basic variable	$x$	$y$	$z$	$r$	$t$	Value
$r$	1	0	5/12	0	0	246
$s$	9/12	1	0	0	0	153
$t$	5/12	0	-2	0	1	171
$P$	-2/12	-4/12	-3/12	0	0	0

Marks for this question are awarded from the second tableau in the outcome column.

Looking through each row of the tableau, all the values match hence M1, A1, M1, A1 and B1 are awarded. Next, we look through the third tableau, where each of the values match that of the mark scheme, awarding a further M1, A1, M1, A1. This gives 9/9 for part (c). Part (d) asked for the final values of each variable, which is shown in the final tableau in the outcome

(a)  $P=150$      $x=0$      $y=1.5$      $z=4.8$      $r=246$      $t=0$      $\text{ai}^2$      $\text{ai}^2$      $\text{ai}^2$

(b)  $\text{ai}^1$      $\text{ai}^1$      $\text{ai}^1$

(c)  $\text{ai}^1$      $\text{ai}^1$

(d)  $\text{ai}^1$      $\text{ai}^1$      $\text{ai}^1$

5

7. This question is to be answered on the sheet provided in the answer booklet.

A chemical company makes 3 products X, Y and Z. It wishes to maximise its profit  $P$ . The manager considers the limitations on the raw materials available and models the situation with the following Linear Programming problem.

$$\text{Maximise } P = 3x + 6y + 4z,$$

$$\text{subject to } x + z \leq 4,$$

$$x + 4y + 2z \leq 6,$$

$$x \geq 0, y \geq 0, z \geq 0,$$

where  $x$ ,  $y$  and  $z$  are the weights, in kg, of products X, Y and Z respectively.

A possible initial tableau is

Basic variable	$x$	$y$	$z$	$r$	$s$	$t$	Value
$r$	1	0	1	1	0	0	4
$s$	1	4	2	0	1	0	6
$t$	1	1	2	0	0	1	12
$P$	-3	-6	-4	0	0	0	0

(a) Explain

$$(x+1)^{-1} = \frac{1}{x+1}$$

(b) Use your row operations.

(b) Solve this Linear Programming problem by using the Simplex algorithm. Increase  $y$  for your first iteration and then increase  $x$  for your second iteration. (10)

## Edexcel Decision Maths 1 June 2001 Question 7

Iteration 1: Initial Tableau							
b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value
$r$	1	0	1	1	0	0	4
$s$	1	4	2	0	1	0	6
$t$	1	1	2	0	0	1	12
$P$	-3	-6	-4	0	0	0	0

Iteration 1: Iteration 1							
b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value
$r$	1	0	1	1	0	0	4
$s$	$\frac{1}{4}$	1	$\frac{1}{2}$	0	$\frac{1}{4}$	0	$\frac{1}{2}$
$t$	1	1	2	0	0	1	10 $\frac{1}{2}$
$P$	- $\frac{11}{4}$	0	-1	0	$\frac{1}{4}$	0	9

Iteration 1: Iteration 2							
b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value
$r$	1	0	1	1	0	0	4
$s$	$\frac{1}{4}$	1	$\frac{1}{2}$	0	$\frac{1}{4}$	0	$\frac{1}{2}$
$t$	0	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$	1	$10\frac{1}{2}$
$P$	0	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$	0	15

column, matching the mark scheme, so we get 3/3, giving 12/12 total.

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value
$r$	1	0	1	1	0	0	4
$s$	1	$\frac{1}{4}$	2	0	1	0	6
$t$	1	1	2	0	0	1	12
$P$	-3	-6	-4	0	0	0	0

b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value
$r$	1	0	1	1	0	0	4
$s$	$\frac{1}{4}$	1	$\frac{1}{2}$	0	$\frac{1}{4}$	0	$\frac{1}{2}$
$t$	0	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$	1	$10\frac{1}{2}$
$P$	- $\frac{11}{4}$	0	-1	0	$\frac{1}{4}$	0	9

The first mark is awarded for finding the pivot value, which is done correctly by the system, highlighting it in purple. The second mark is awarded for achieving the second tableau after performing an iteration, which matches the tableau from 'Iteration 2' in the outcome column. Further M1, A1, A1, A1 marks are awarded for the correct row operations and identifying the pivot, all of which are correctly done by the system. The final M1, A1, A1, A1 are awarded for the third tableau, getting the correct row operations and correct entries in each cell of the table. The final iteration from the system matches the mark scheme, hence these

Iteration 1: Iteration 2							
b.v.	$x$	$y$	$z$	$r$	$s$	$t$	value
$r$	1	0	1	1	0	0	4
$s$	$\frac{1}{4}$	1	$\frac{1}{2}$	0	$\frac{1}{4}$	0	$\frac{1}{2}$
$t$	0	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$	1	$10\frac{1}{2}$
$P$	0	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$	0	15

marks are awarded.

10/10

6

An initial Simplex tableau for the above situation is

Basic variable	x	y	z	r	s	t	Value
r	3	2	4	1	0	0	35
s	1	3	2	0	1	0	20
t	2	4	3	0	0	1	24
P	-4	-5	-3	0	0	0	0

(b) Solve this linear programming problem using the Simplex algorithm. Take the most negative number in the profit row to indicate the pivot column at each stage.

(II)

## Edexcel Decision Maths 1 November 2002 Question 8

Basic Variable	x	y	z	r	s	t	Value
r	2	0	$\frac{5}{4}$	1	0	$-\frac{1}{2}$	23
s	$-\frac{1}{2}$	0	$-\frac{5}{4}$	0	1	$-\frac{1}{2}$	2
y	$\frac{1}{2}$	1	$\frac{3}{4}$	0	0	$\frac{1}{4}$	6
P	$-\frac{5}{2}$	0	$\frac{1}{4}$	0	0	$\frac{5}{4}$	30

$R_1 - 2R_2$  M1  
 $R_2 - 3R_3$  A1  
 $R_3 + 4R_4$  M1  
 $R_4 + 5R_3$  A1

Basic Variable	x	y	z	r	s	t	Value
x	1	0	$\frac{5}{4}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	$\frac{21}{4}$
s	0	0	$\frac{5}{4}$	$\frac{1}{2}$	1	$-\frac{7}{8}$	$\frac{31}{4}$
y	0	1	$\frac{1}{4}$	$-\frac{1}{2}$	0	$\frac{3}{8}$	$\frac{1}{4}$
P	0	0	$\frac{1}{4}$	$\frac{21}{4}$	0	$\frac{1}{2}$	$\frac{105}{4}$

$R_1 \div 2$  M1  
 $R_2 + \frac{1}{2}R_1$  A1  
 $R_3 - \frac{1}{2}R_1$  M1  
 $R_4 + \frac{1}{2}R_1$  A1

Marks are awarded from the second tableau, for each row being correct with the relevant row operations. Looking at 'Iteration 2', we see each row entry matches the first mark scheme table, awarding M1, A1, M1, A1. The next iteration from the system corresponds to the second table in the mark scheme, again matching each row entry with correct row operations. This awards us M1, A1, M1, A1. Finally, the mark scheme awards 3 marks for the final values of each variable, which are presented in the explanation of the last tableau by the system. These match that of the mark scheme hence awarding M1, A1, A1.

11/11

7

Another constraint and the objective function give the following Simplex tableau. The profit  $P$  is stated in euros.

Basic variable	$x$	$y$	$z$	$r$	$s$	Value
$r$	3	5	6	1	0	50
$s$	1	2	4	0	1	24
$P$	-1	-3	-4	0	0	0

(c) Write down the profit on each small lamp.

(3)

(d) Use the Simplex algorithm to solve this linear programming problem.

(9)

## Edexcel Decision Maths 1 November 2003 Question 8



b.v	x	y	z	r	s	value
$r$	$\frac{3}{2}$	2	0	1	$-\frac{3}{2}$	14
$s$	$\frac{1}{4}$	$\frac{1}{2}$	1	0	$\frac{1}{4}$	6
$P$	0	-1	0	1	0	24

b.v	x	y	z	r	s	value
$y$	$\frac{3}{4}$	1	0	$\frac{1}{2}$	$-\frac{3}{4}$	7
$z$	$\frac{1}{4}$	0	1	$-\frac{1}{4}$	$\frac{5}{8}$	$\frac{5}{8}$

Profit = 31 Euros  
medium large  
 $y = 7$   $z = 2.5$   $x = r = s = 0$

(3)

The first M1, A1, A1 is awarded from the second tableau produced by the system, which matches the mark scheme in terms of row entries and row operations. The next set of marks are awarded for the third tableau produced by the system, which has the same row operations and entries in each cell of the tableau, awarding M1, A1, A1.

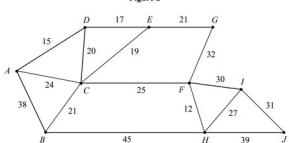
Lastly, marks are awarded for outputting the final values of each variable, as shown in the explanation for the final tableau. These values match the mark scheme, awarding M1, A1, A1.

9/9

8

3.

Figure 2



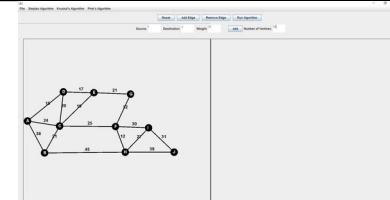
The network in Figure 2 shows the distances, in metres, between 10 wildlife observation points. The observation points are to be linked by footpaths, to form a network along the arcs indicated, using the least possible total length.

(a) Find a minimum spanning tree for the network in Figure 2, showing clearly the order in which you selected the arcs for your tree, using

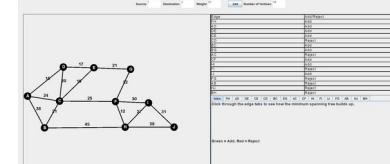
Kruskal's algorithm.

(3)

## Edexcel Decision Maths 1 January 2005 Question 3



3) (a) (i)  $AB, DE, EF, FG, EH, HI, IJ, HJ$  M1 A1 A1 (3)



The first screenshot provides evidence that the graph presented in the question can be successfully drawn using the system.

This question didn't require the minimum spanning tree to be

drawn in any way, but for explanation purposes, the system provides that feature regardless. The main part for marking is the table of added and rejected edges, showing FH, AD, DE, CE, BC, EG, CF, HI, IJ as the added edges. This matches the edges listed in the mark scheme, following the same order. While not relevant for the marking, the coloured edges follow the relevant edges and produce a valid minimum spanning tree.

3/3

9

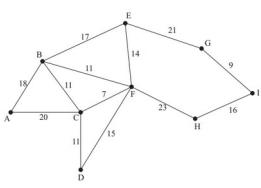


Figure 3

Figure 3 represents a network of paths in a park. The number on each arc represents the length of the path in metres.

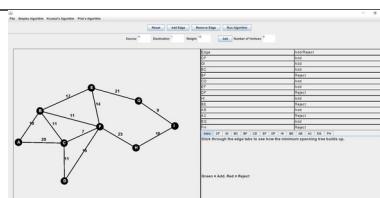
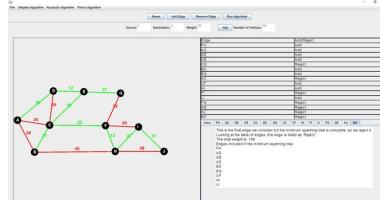
(b) Using your answer to part (a) and Kruskal's algorithm, find a minimum spanning tree for the network in Figure 3. You should list the arcs in the order in which you consider them and state whether you are adding it to your minimum spanning tree.

(4)

(c) Find the total weight of the minimum spanning tree.

(1)

### Edexcel Decision Maths 1 January 2008 Question 2



(b)

CF ✓  
GI ✓  
(BC or BF - accept one, reject one)  
CD ✗  
EF ✗  
DF ✗  
HI ✗  
BE ✗  
AB ✗  
AC ✗  
EG ✗  
Tree complete

m 1

A 1

A 1

A 1

4

(c)

107 m

B 1

<1

Entering this graph into the system was successful, and running the algorithm performed the list of added and rejected edges. Looking through, we see CF, GI, BC, CD, EF, HI, AB, EG. These match up with the mark scheme, which allowed either one of BC or BF, given that the other was rejected, which the system did by

choosing BC and rejecting BF. This gives M1, A1, A1, A1. Part (c) asked for the weight of the minimum spanning tree, given in the last edge tab as 107, matching the mark scheme, awarding a further B1 mark.

5/5

10

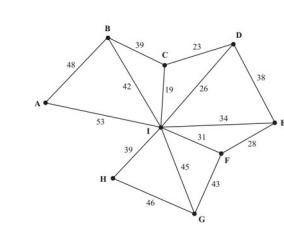
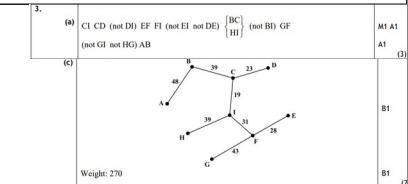
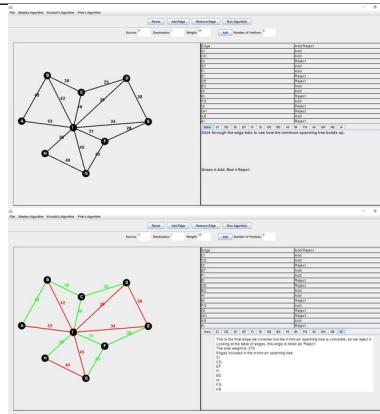


Figure 2

- (a) Use Kruskal's algorithm to find a minimum spanning tree for the network shown in Figure 2. You should list the arcs in the order in which you consider them. In each case, state whether you are adding the arc to your minimum spanning tree. (3)
- (b) Starting at A, use Prim's algorithm to find a minimum spanning tree for the network in Figure 2. You must clearly state the order in which you include the vertices in your tree. (3)
- (c) Draw a minimum spanning tree for the network in Figure 2 using the vertices given in Diagram 1 of the answer book. State the weight of the minimum spanning tree. (2)

### Edexcel Decision Maths 1 January 2011 Question 3



Upon creating the graph and running the algorithm, the table of added and rejected edges produced: CI, CD, EF, FI, BC, HI, FG, AB as the added edges and the rest rejected. As shown by the mark scheme, with BC and HI permitted to be added in either order. This awards the marks M1, A1, A1. While the system doesn't draw a separate minimum spanning tree, we can use the green edges to see if the correct edges have been added. Comparing the tree to the mark scheme, both have the same shape, with the same

added edges awarding a further B1 mark. The final mark is awarded for outputting the correct weight of 270, giving another B1 mark.

5/5

11

### Edexcel Decision Maths 1 January 2012 Question 1

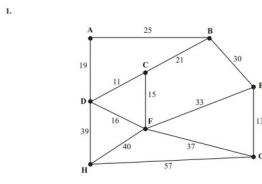


Figure 1

Figure 1 represents the distances, in km, between eight vertices, A, B, C, D, E, F, G and H in a network.

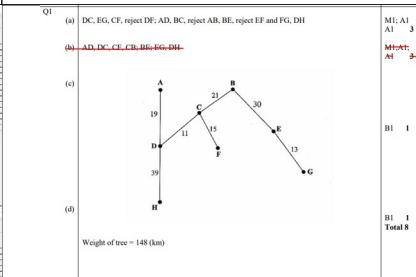
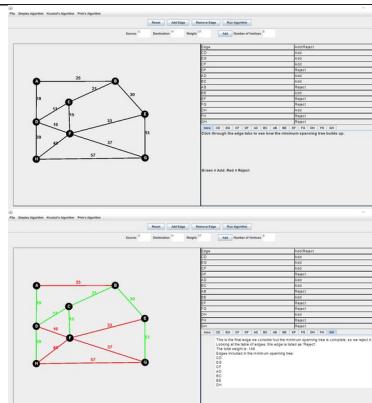
(a) Use Kruskal's algorithm to find the minimum spanning tree for the network. You should list the arcs in the order in which you consider them. In each case, state whether you are adding the arc to your minimum spanning tree.

(b) Starting at A, use Prim's algorithm to find the minimum spanning tree. You must clearly state the order in which you selected the arcs of your tree.

(c) Draw the minimum spanning tree using the vertices given in Diagram 1 in the answer book.

(d) State the weight of the tree.

(Total 8 marks)



Looking through the table of added and rejected edges, the system produced: CD, EG, CF, AD, BC, BE, DH as the added edges, which matches the mark scheme awarding M1, A1, A1. Part (c) involved drawing the minimum spanning tree, which the system does in the form of highlighting the added edges in green. This matches the mark scheme, awarding B1. For part (d), the output produced was 148, which matches the mark scheme again, awarding another B1 mark.

5/5

12

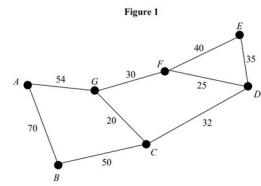
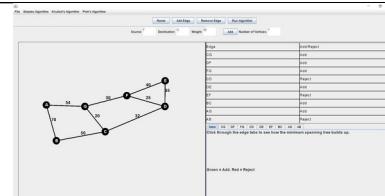


Figure 1 shows 7 locations, A, B, C, D, E, F and G which are to be connected by pipelines. The arcs show the possible routes. The number on each arc gives the cost, in thousands of pounds, of laying that particular section.

(a) Use Kruskal's algorithm to obtain a minimum spanning tree for the network, giving the order in which you selected the arcs. (4)

### Edexcel Decision Maths 1 June 2001 Question 2



Q2 (a) GC, FO, FG ; DE, BC, GA

After entering the graph and running the algorithm, the added edges in the table were: CG, DF, FG, DE, BC, AG, with the others being rejected. As shown, this matches the above mark scheme (note: GC is the same as CG, more generally, in an undirected graph, nodes can be labelled either way around), awarding M1, A1, M1, A1.

M1 A1 / M1 A1

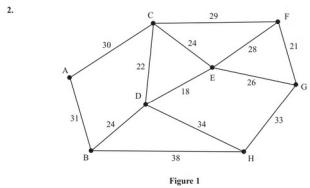
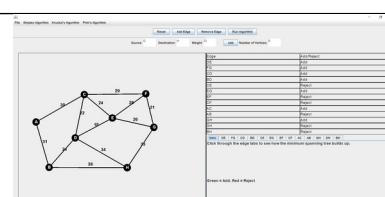


Figure 1 represents the distances, in metres, between eight vertices, A, B, C, D, E, F, G and H, in a network.

(a) Use Kruskal's algorithm to find a minimum spanning tree for the network. You should list the arcs in the order in which you consider them. In each case, state whether you are adding the arc to your minimum spanning tree. (3)

13)

### Edexcel Decision Maths 1 June 2010 Question 2



Q2 (a) DE GF DC {not CE} BD EG (not EF not CF) AC (not AB) GH

M1 A1 / M1 A1

Running the algorithm on the drawn graph produced the table of added and rejected edges, with the added edges being: DE, FG, CD, BD, EG, AC, GH. This matches with the edges in the mark scheme, with the relevant edges being rejected too. This awards the marks M1, A1, A1.

3/3

## Usability Testing/Stakeholder Testing

The most important part of testing this system is ensuring that it is not only functional according to its purpose, but also that it is usable by the stakeholders

who the system is directed towards. To receive feedback on how the system is deemed by my stakeholders, I presented the original 3 current [SCHOOL NAME ABBREVIATION] further maths students, Student 1, Student 2 and Student 3 with the whole system. This testing is split up into two stages, the first was more open-ended where I allowed the students to use the system on their own for the first time, try out features and see how usable it is. Secondly, I gave them some brief prompts and then asked for their feedback on these specific features of the system that would be important in ensuring were considered throughout development to ensure usability. It is important that the stakeholders are allowed to use the system on their own first, before giving them any prompts, to ensure that their responses to how easy-to-use the system is hasn't been affected by previous exposure to the system via my prompts.

After allowing the 3 students to use the system on their own, without any guidance on what to do, I interviewed them with the following questions. Their results are summarised below:

Question	Student 1 (Student 1) Response	Student 2 (Student 3) Response	Student 3 (Student 2) Response
Having been able to use the system, first outline which features you used, and how usable each of them were	I first used the Simplex algorithm tool by clicking on the option in the menu bar. Each item is labelled well, so I knew which button menu option to click if I wanted to run the algorithm by entering my own inequalities.  Although I did enter the number of inequalities before the number of variables with the left-to-right layout, it might not be clear to all users which one to do first, so I	I first tried out the Kruskal's algorithm feature where I can draw my own graph. I think this feature is split up well into various functions like adding an edge, removing an edge and running the algorithm. The table of added and rejected edges is useful but sometimes the formatting is a bit odd, and the colours seem to blend in with the	Since my weakest topic in decision maths is the Simplex algorithm, I first tried out the feature to let me run the algorithm. I tried a question from the textbook and navigating the GUI was easy, with clearly labelled buttons etc. When running the algorithm on the inequalities I entered, I checked the tableaux

	<p>tried it again the other way around and received the red error message, which was helpful in case I didn't know which inputs I needed to make first.</p> <p>Entering the inequalities was intuitive as well as running it, and I liked the use of the tabs to be able to click through the iterations. All the necessary information is presented in each tab. One thing I would prefer is if the box with the tabs was a bit larger, especially in the scenarios where the inequalities don't take up as much room, which would make the explanations a bit easier to read.</p> <p>I also tried the Kruskal's algorithm feature where I can draw my own graph. I think the method for drawing a graph is helpful as it was easy for me to translate a graph from the textbook into the required inputs for the visual graph and I like</p>	<p>background. I think the idea of clicking through the edge tabs is a good idea and I like how the edges are coloured in green and red; it helps with understanding how the edges are added. I think it's good that you explain what the colours mean by saying that green is adding the edge and red is rejecting the edge.</p> <p>I next tried using the learn feature for the Simplex algorithm, but as you mentioned this hadn't been developed due to priority placed on the getting the algorithms with user input working. Since I'm already very familiar with how the algorithms work, personally I don't think it matters if it's missing or not. While I think it would add to the completeness of the</p>	<p>against the mark scheme from the textbook and they all matched and were explained fully which was very helpful in my understanding. While the tabs did contain all the necessary information, I feel like it could take up more room on the whole window instead of being a fixed rectangle near the bottom.</p> <p>I tried the Simplex learn feature next but as you mentioned it wasn't completed. I think it would be a useful feature to make the system more suited for people learning the algorithm from scratch, but for me and other students who have already learnt the purpose of the algorithm in class, I think that having the feature to run my own</p>
--	--	--	--

	<p>that I can move around the different nodes and that the edges follow.</p> <p>When I ran the algorithm on the graph I made, the table was presented clearly and I liked how I could click through the edges and they would be coloured in different colours based on whether they're added or rejected.</p>	<p>system, allowing users to be introduced to the algorithm, learn how it works and then try it out, I still think the most important part is being able to run the algorithm on my own inequalities e.g. for a question I want to see the working for.</p>	<p>inequalities sort of encompasses those features too, since it also explains each part of the algorithm.</p> <p>I also tried the Kruskal's graph drawing feature, which was very intuitive and laid out well. I like the use of different colours to highlight the different edges being added or rejected and it's good that there is a colour key to inform me as the user what each colour represents. Like the Simplex algorithm feature, this also does a good job at explaining the steps of the algorithm, meaning that I personally don't think the learning feature being incomplete is an issue.</p>
Do you think the	The main uses of colour that I spotted were in	I think that colours are used well	I particularly liked the use of colour

<p>colour choices throughout that system are insightful, necessary and unambiguous?</p>	<p>the buttons and in some parts of the different algorithm features, for example, highlighting the pivot row, pivot column and pivot value. During the development I remember you asked about the colour choices from the textbook, and it's clear to see that you've taken them into account and ensured that the colours are clearly differentiable and serve their purpose well. I see that the colour choices are explained in the explanation of each tableau on the GUI, but I think that some sort of preset that explains the colours off to the side would be useful, instead of having to read through the explanation to find out the meaning of the colours.</p> <p>For the Kruskal's tool, the colours serve their purpose well, and the red and green are well explained making them unambiguous, aligning</p>	<p>throughout the system. In the Kruskal's feature, I like how on the graph there is white text on the black node, to make it clearly visible what the node label is. I also find the edge highlighting feature useful with my understanding of the topic, I think it adds a nice touch to the system.</p> <p>I think there is a good use of colour in the Simplex feature too, none of the colours stand out excessively, they serve their purpose well. I think there could be some more use of colours in different areas of the program, for example, making the Kruskal's table of added and rejected edges stand out, since the background of the table seems to blend in with the rest of the GUI. Overall I think the</p>	<p>in the Kruskal's algorithm feature, I found it insightful, and unambiguous. While not necessary for a student learning what they need to do to get marks in an exam question, I think it's valuable in acting as part of the explanation but tapping into the visual side to encompass a larger set of learning styles.</p> <p>I liked the use of colour on the home screen with the buttons, which make a clear indicator for someone first opening the system.</p> <p>Another good use of colour is in the Simplex algorithm feature, which made it easier to follow the iterations of the problem, seeing the changing pivot row, column and</p>
---	---	---	--

	<p>with the meaning of colours to different people. They add to the explanation and help with my visual learning and are clearly explained, so this is a good use of colour.</p> <p>Another good use of colour is in the error message for when I put in an incorrect input. Showing up in a different colour to the rest of the text is useful in highlighting immediately that something needs to change.</p> <p>I think that some other places in the system could use some colours, if not to the highlight a point, then just to differentiate a part of the GUI from another. An example of this could be making the 'Run' algorithm button a different colour to the others.</p>	<p>colour choices are good, serve a purpose and are well defined, but the system as a whole could use a bit more colour outside of the standard options from the Java libraries used.</p>	<p>value. The colour choices are good and are explained well in the tableau explanation, ensuring that their purpose is unambiguous. Again, whilst not necessary, it rounds off the feature with depth in different learning styles which will be valuable to a range of students. Throughout the system, colours serve a valuable purpose and are insightful, but I feel there is some more room for colour in certain places.</p>
Are any inputs to the system presented in an easy-to-use way,	The first inputs to the system were selecting the menu items, which were well-labelled and easy to navigate. I think they are split up well	The main inputs to the system were for the Kruskal's algorithm feature and the Simplex algorithm feature.	I first used the Simplex algorithm feature, and the inputs were all very intuitive. I followed from left

and are their functions intuitive?	<p>into the different sub-options for different algorithms.</p> <p>I first tried the Simplex algorithm feature, and the inputs were very intuitive for me. Following left to right from the drop-down menu to the buttons ensured that I selected all the necessary options in the right order. For me, this was intuitive, but to ensure it remains accessible for all users, I would recommend having some sort of text explaining to follow from left to right, or to make the options appear in chronological order e.g. have the variable buttons only appear when I have selected the number of inequalities.</p> <p>The next inputs to the system were the inequalities for the Simplex algorithm. Only the necessary number of boxes were presented, which made it much easier to know which inputs I needed.</p>	<p>For the Kruskal's algorithm feature, I found every input to be well-labelled and intuitive. All the input boxes and buttons were of a good size and there wasn't too much effort from me as the stakeholder to place my inputs for the system. One thing I would say is that when I'm writing in the textbox, my text isn't centred and the text is slightly small, however, it's still clearly legible and the text positioning doesn't impact the functionality or usability at all, so this is a minor issue.</p> <p>The next inputs were for the Simplex Algorithm feature. I found the drop-down and the buttons easy-to-use and the GUI was not too cluttered even with the variety of options. When entering the inequalities into the</p>	<p>to right from the drop-down menu to the buttons and that meant I selected all the necessary options in the correct order, however I understand why this may be a cause of ambiguity, so instead of just relying on the error message to show up when the incorrect order is followed, I think having some text to prompt the user of which inputs should be entered in which order would be useful. For the actual inequalities, each box was already labelled with which input is necessary and the input boxes were also presenting in a sort of grid to mimic the exam questions.</p> <p>The next inputs were for Kruskal's algorithm, which I found to be</p>
------------------------------------	---	--	--

	<p>Furthermore, each box was already labelled with which input is necessary, and the box was a decent size to remain noticeable on the GUI, but also not taking up too much space.</p> <p>The other inputs to the system were for the Kruskal's algorithm feature. The buttons for different functions were clearly labelled so I knew exactly what to press e.g. to add an edge or to remove an edge. The textboxes presented were also labelled intuitively. I entered an example graph from the textbook, and found that the process of adding each edge did take quite some time, however, any method I can think of outside of pasting an image of a graph and having it transformed into a graph that I can move around, every method to take inputs will take time so this is not an issue. Similarly, removing an edge was</p>	<p>boxes, the labels made it easy to see which input needed to go into each textbox. Another subtle feature that I liked was when entering a question with 4 variables, the variable names changed to x1, x2, x3 and x4, just like how the exam questions present them. This just made it easier for me as the stakeholder to not confuse which inputs need to go in which box.</p> <p>I would also consider being able to move the edges around as an input. This feature works very smoothly and having the boundary for the area that the different nodes and edges can move within ensures the whole graph remains on the screen but is also easily customisable to different shapes.</p>	<p>intuitive, however a bit tedious having to click between the boxes for source node, destination node and weight of the edge especially for a graph with 10+ edges. Removing an edge has an easy process of similarly entering source and destination, with clearly labelled input boxes.</p> <p>I also liked the feature of being able to move around the different edges of the graph; they all moved smoothly and allowed me to make the graph look like the textbook. The edges on the GUI followed well and remained within the boundary, whilst also tracking my inputs well.</p> <p>I think all the</p>
--	--	---	--

	<p>just as intuitive, presenting the necessary and clearly labelled text areas.</p> <p>Overall, the inputs to this system are all well-defined, with my only critique being to have some explanation for the order of inputs for the Simplex algorithm feature.</p>	<p>One thing I would say about this feature is that there is no prompt to move around the edges on the GUI, so if I had not just tried moving them around, I wouldn't know this is a feature of the system, so I would recommend having some sort of prompt or guidance informing the users that the graph can move around.</p>	<p>inputs to this system are well-defined and usable.</p>
For this question, I prompted my stakeholders to enter unexpected inputs to the system to test if the input validation can guide users into entering the correct values	<p>The error message label appears in a suitable place on the GUI, where it is easy to see something has changed and needs attention. I tried a combination of invalid inputs e.g. first selecting the number of variables before the number of inequalities. The relevant error message showed up informing me of my error. When I fixed the error, this message disappeared, having served its purpose.</p> <p>I also tested this with</p>	<p>I first tested invalid inputs for the Kruskal's algorithm feature by entering nothing in the source and destination text boxes. Trying to add this edge to the graph caused the error message label to show up on the GUI. This was placed in a good location, where I was able to notice that the input was not valid, especially since the text was in red.</p> <p>I then tried entering</p>	<p>With your prompt, I first tried running the Simplex algorithm without having selected any number of inequalities or variables. This resulted in a red error message popping up next to the buttons, informing me of what inputs I needed to provide.</p> <p>I also tried leaving some of the input boxes for the Simplex</p>

	<p>the Kruskal's algorithm feature, first trying to add the same edge to the graph and then trying to remove an edge from the graph that doesn't exist. In each scenario, the necessary error message label showed up, with the rest of the GUI being preserved without being impacted by the incorrect input.</p> <p>The input validation in your system is definitely robust in handling a variety of cases of unexpected inputs.</p>	<p>invalid inputs for the Simplex algorithm feature, by entering letters instead of numbers for some of the text areas. Again, the error message label with the correct description of the error was displayed in red and I was able to re-enter my inputs correctly and the program functioned as expected. Overall, the system does a very good job at managing unexpected inputs and correcting them to the user in a subtle but informative way</p>	<p>inequalities empty and running the algorithm, which also prompted the error message label to tell me to enter valid numbers in all the text areas.</p> <p>I also tested the input validation for the Kruskal's algorithm feature, by not entering any numbers in the text boxes and trying to add as an edge, which once again displayed the error label with an informative message to tell me what to correct. I think that the input validation in this system is well designed and can handle a variety of user inputs, guiding them towards how to correct enter inputs.</p>
For this question, I asked my	Upon opening the system, I first used the Simplex algorithm	I first started by drawing a graph using the Kruskal's	Upon opening the feature, I used the Simplex algorithm

<p>stakeholders to use various features of the system, one after another, to test how easy it is to move between features and use the system as a whole.</p>	<p>feature, entering inequalities and selecting the 'Run Algorithm' problem. After this, I selected the Kruskal's algorithm feature. The GUI cleared and loaded the screen to allow me to add edges to the graph. After adding edges to the graph, I ran Kruskal's algorithm on it, which worked as expected, with no impact from having completed the Simplex algorithm prior to this.</p> <p>I then used the menu bar to return to the home screen before going back onto the Simplex algorithm feature, which allowed me to input a new problem from the beginning, selecting the number of inequalities and the number of variables.</p> <p>Having used a series of features from the system one after another, I think it manages whole system robustness quite well.</p>	<p>algorithm feature, then before running the algorithm, I returned to the home screen and then used the menu bar to navigate to the Simplex algorithm feature. I completed a Simplex problem with 2 inequalities and 2 variables, and then returned to the Kruskal's algorithm feature using the menu bar.</p> <p>After having clicked off the Kruskal's feature and used the Simplex feature before returning to the Kruskal's algorithm section, the graph that I had initially drawn was still preserved, allowing me to continue where I left off.</p> <p>Moving between features on the system is seamless and made easier for the user e.g. by preserving inputs to the graph.</p>	<p>feature first, entering a few inequalities but not running the algorithm. I then clicked on the Kruskal's algorithm feature, before drawing a graph and removing an edge from it using the 'Remove Edge' button. I then returned to the Simplex algorithm feature, but the inputs I had previously made had been reset once I had clicked off it. From here, I navigated back to the Kruskal's algorithm feature, which had preserved my graph, and the removed edge was still removed, allowing me to continue by running the algorithm on the graph I created and read through the explanation.</p> <p>I think the</p>
--	--	---	---

			Kruskal's feature generally does a good job at maintaining user inputs, however the 'reset' function for the Simplex algorithm is defined by the user clicking off the feature, meaning if they input something and return to it, the inputs aren't preserved. I think having a separate reset feature would be more practical.
Of the missing features, could you explain which ones you think would be most important if continued in development, and which ones have a lower priority.	The missing features I noticed were the Prim's algorithm feature, the random graph feature, the algorithm introduction feature and the learn algorithm feature.  From these I think that Prim's algorithm would be the most important to continue developing, however since my weakest area in Decision maths is the Simplex algorithm, for me the system works fine without it.	Since I found entering the graphs to be quite a tedious process, I think that the most important feature that you could implement next would be the random graph generator, as this would take away a lot of the manual effort and just allow for easier access to solutions.  For me the Prim's algorithm feature isn't as important,	For me, the main feature that needed implementing was the Simplex algorithm, and this has been done well, so personally I think that not having the random graph feature or the Prim's algorithm feature isn't too much of an issue, however, I can understand why they would be necessary, since

	<p>The random graph generator feature seems like a good idea, however I don't think it's inherently necessary for the functionality of the system, since it's difficult to judge how accurate the generated graphs would be in comparison to the exam questions.</p> <p>For me the introduction and learn feature seem like more minor features, because I'm already familiar with how the questions will be presented and what the algorithm is used for. I can also easily enter example questions from the textbook myself if I don't understand them. The main difficulty for me is actually applying the algorithm, which the system has a feature for, hence this is less of an issue to be missed out, and more of an additional helpful feature.</p>	<p>since I'm more familiar with it than the other two algorithms. Prim's algorithm seems more systematic to me, but Kruskal's required more observation and more working out to be shown. Simplex is also agreeably the most difficult topic amongst our class, so I'm glad that has been implemented well and is useful for understanding the algorithm.</p> <p>I think that the learn feature has been well incorporated into the run algorithm features, so I don't think a separate feature is necessary. Since you mentioned the learn feature would have used fixed examples, now seeing how the run Simplex algorithm feature turned out, I think it does a better job at explaining the algorithm with any question the user</p>	<p>Prim's algorithm is often asked more often through a variety of topics e.g. the Travelling Salesman problem.</p> <p>For me, the explanation and working out incorporated into the 'Run' feature encompasses everything that a separate 'Learn' feature would do, hence this is a lower priority feature that I would want to be implemented.</p> <p>I don't think the introduction feature is particularly necessary. The only reason to really implement this would be just to round off the system, making it more accessible in terms of learning for people trying to learn the algorithms with no prior</p>
--	--	--	---

		<p>has, as opposed to the fixed example system.</p>	<p>knowledge. However, our class generally has a good understanding of what each algorithm is used for, how it's presented in the exam, with the main difficulty being performing it. Your system tackles the most important aspect of each algorithm; hence I think most of the missing features would just be 'nice additions' as opposed to necessary for functionality.</p>
Any further suggestions or improvements for the user experience e.g. layouts, quality of explanation.	I think the layout of the system is well-developed, with a clear menu bar at the top and a large middle area to actually have the features running. The only suggestion I could make in terms of layout would be to have the tableau for the Simplex feature take up slightly more room. I understand that with a certain number of	I think that the system is laid out well and split up clearly into different sections. Within each feature the layout is also consistent, using similar sized buttons, recognisable input boxes with prompts, which overall contributes to familiarity with	The explanations are of good quality, but I feel like they are more in depth for the Simplex algorithm feature than for the Kruskal's algorithm feature. I understand that each Simplex tableau has much more to explain than a single edge in the graph, but

	<p>inequalities, the vertical room is limited, but when that's not the case, the whole explanation box could use up more room both vertically and horizontally, just to make the system clearer.</p>	<p>features throughout the system. I think the explanations are created well, but at times can become quite lengthy e.g. for the individual Simplex tableaux which is just a paragraph of text, all the same size. The difficulty is that all the text is necessary, so to make it more readable, maybe you could make some key parts bold e.g. the pivot value or how to perform the row operations. Other than that, it's all laid out well.</p>	<p>perhaps adding a bit more e.g. 'The next smallest edge is [insert edge] with weight [insert weight]. Adding this to the graph would [create a cycle/not create a cycle], hence we [reject/add] it [from/to] the minimum spanning tree'. The current explanation seems a bit basic.</p> <p>The layout is created well, with each component having a good amount of room such that it doesn't take up too much space but is also large enough to be clearly seen. If you wanted a bit more space e.g. for the simplex tableau, you could try and make a 'hide taskbar' style feature, where the buttons at the top disappear when not in use. Other</p>
--	--	--	--

			than that, it's all well laid out and an easy system to use/navigate.
--	--	--	---

### **Summary of Stakeholder Usability Testing**

Having received feedback from my stakeholders on the various key elements of my system, I can now review their responses and identify and key patterns among them as to what features were found to be helpful, which ones were less helpful and what features they would like to see added.

Question 1:

All the students agreed that the system is split up well and laid out in an intuitive manner. They mentioned that it is easy to navigate to their desired location using the menu bar and the individual options within it. One notable feature brought up by the students was that the order of inputs required by the system, while relatively intuitive, could use some explanation in the system just to create certainty among those using the system. An example of this could be prompting users to first enter number of inequalities instead of having that message show up once the user makes an incorrect input.

Question 2:

Overall, the students agreed that the colours used in the system were effective and chosen well, with Student 1 mentioning 'the colours are clearly differentiable and serve their purpose well'. I generally tried to avoid excessive use of colours in unnecessary areas to ensure that any use of colour was purposeful and aided the explanation of an algorithm. However, the students also expressed that some areas of the system lacked enough colour and that there were more areas where it would remain purposeful e.g. distinguishing the table of added and rejected edges in the Kruskal's algorithm feature from the rest of the background. Adding more colour in meaningful areas is something that would be considered high priority for me to implement in the system, should I develop it further.

### Question 3:

There were some mixed responses regarding the usability of inputs to the system, with general agreement that they were intuitively presented, however, with some caveats, particularly in the Kruskal's algorithm feature. Student 2 mentioned that the inputs to the system for the edges in a graph were 'a bit tedious' having to click through the various boxes for a single edge. Some recommendations were made about alternative methods of taking user inputs to the system e.g. scanning a graph, however, this would be much more complex to implement, as acknowledged by Student 2 in his response. Generally, the Simplex algorithm feature had intuitive inputs, with one noted piece of feedback being that a simple line of text to inform the users on the order in which to input to the system instead of relying on just an error message that pops up after the user gives their inputs in the wrong order.

### Question 4:

Input validation and robustness was noted as one of the features that the stakeholders thought were well developed, with Student 1 mentioning that the ability to handle unexpected inputs was 'definitely robust'. In a system with a lot of human input required, mistakes or unexpected inputs are inevitable, so being able to handle any combination of incorrect inputs was seen as high priority in the development stage. The students also generally agreed that the response to an incorrect input was well-developed, with a simple error message appearing in red text informing the user of their mistaken input. Student 3 described this response to invalid inputs as a 'subtle but informative way' to guide the user to correct inputs.

### Question 5:

This question was aimed at testing functionality of the system, as opposed to the singular features that my stakeholders had currently tested. This involved having

the students use part of a feature, before selecting another feature to use and returning to the original feature, simulating a realistic use case of the system where students are likely to be switching between different features. Student 3 mentioned that switching between features was 'seamless' and how preserving inputs for the Kruskal's algorithm 'made [it] easier' as a user. One feature Student 2 picked up on was that there was no feature to preserve inputs to the Simplex algorithm feature, and that clicking off it meant resetting inputs. As an improvement (and as suggested by Student 2), functionality of the system as a whole could be made better by having the inputs preserved when clicking off the Simplex algorithm feature, and having a separate reset button, like in the Kruskal's algorithm feature. The main difficulty is that with the Kruskal's algorithm feature, inputs are preserved in the form of a graph, which can be drawn onto the GUI each time the feature is selected, however for the Simplex algorithm, preservation of inputs would involve potentially preserving inputs in textboxes that haven't been saved anywhere. Overall functionality was deemed successful, with some minor improvements suggested by the stakeholders.

#### Question 6:

This was an important question to ask, given that some features of the system remained incomplete. Identifying which of these features were important to the stakeholders was important to know what future development would focus on as well as what could be now deemed as unnecessary/low priority after evaluating the current system with stakeholders. The main missing feature identified by the stakeholders was Prim's algorithm, left undeveloped due to the unexpected complexity of general GUI management throughout the development stage, adding to time constraints. Furthermore, from the analysis, it was identified that students mostly struggled on the Simplex algorithm, hence my priority was placed on ensuring it was implemented into the GUI first. The students also justified that the main features they struggle with have been successfully developed and that features like Prim's algorithm and the random graph generator would be nice additions for completeness of the system, but in their opinion, it remains effective at serving its purpose, with some representation of graph algorithms through the Kruskal's algorithm feature. The introduction and learn features were also

undeveloped in this system, again, due to time constraints as well as much lower priority placed on having them within the system. Student 1 mentioned that he is 'already familiar with how the questions will be presented and what the algorithm is used for' and Student 2 mentioned that 'the explanation and working out incorporated into the 'Run' feature encompasses everything that a separate 'Learn' feature would do'. Having placed my focus on ensuring that the algorithms are explained well as they are performed, the students agreed that this integrated feature serves the same purpose as a separate feature, deeming it lower priority. Of the missing features, I would rank them in the following order from highest priority needing to be implemented in the future to lowest priority: Prim's algorithm, Random Graph Generator, Learn Algorithm, Introduction to Algorithm. Having implemented a separate 'Graph' class that contains all the methods for displaying the graph on the GUI and moving around different edges, the system is already well set up for continued development of the other graph features and is not just limited to Kruskal's algorithm.

#### Question 7:

Having allowed the stakeholders to use the system independently and then following up with specific prompts, I asked for some final general feedback they have from any of the tests they performed on the system, particularly relating to usability and layout. Student 3 mentioned that the layout is 'consistent' with 'recognisable input boxes' to create a sense of familiarity when navigating the system. The students all mentioned that the system serves its purpose well but could use a few improvements to the layout. Student 1 mentioned that the Simplex feature where the tableaux are displayed could 'take up slightly more room', especially in scenarios where there is a lot of unused space between the top of the tabbed pane and the bottom input row for the objective function. Using more horizontal room is another improvement that could be made. When designing the method for inputs, I wanted to replicate the 'grid-like' layout of inequalities, which also happens to mimic the exam questions' presentation where the inequalities are given line by line. This led me to follow a similar layout when creating the tableau pane, keeping it all centred, and while adjustments were made to the horizontal spacing, having received feedback, there is still scope for further use of

horizontal space that is currently empty and isn't used dynamically (there's no case where the horizontal spacing on the side may be used, whereas for the vertical spacing, the available height is dependent on the number of inequalities selected). Through this student/stakeholder feedback, the consensus is that the system is effective with a variety of features that have been developed well, but all three students agree that there is room for improvement with specific features that could use some additional guidance to the user, or present options in a different manner, as well as possibility for features that are currently undeveloped to be implemented in the future.

### **Success Criteria Evaluation**

Criterion	Achieved/Not Achieved	Evidence
<p>1) My system must provide a clearly laid out user interface and allow users to select an algorithm to run with a menu bar.</p> <p>1.1) Each algorithm option must be split into specific sub-sections e.g. user creating their own graph, user viewing example questions.</p>	<p>This criterion has been achieved throughout the system, ensuring that the different features are separated and clearly identifiable.</p> <p>The system contains a menu bar at the top, which contains options for each algorithm, which are further decomposed into the specific feature. Within each individual feature, components like input boxes and buttons are kept consistent in size and labelling, resulting in a familiar/consistent user interface. This was a main priority of the</p>	<p>Tests 1 – 7 provide evidence for this success criteria being met, showing the initial creation of the menu bar when the GUI is opened, as well as the sub-options inside each menu item, with consistent naming throughout.</p> <p>Question 1 and parts of Question 7 from the stakeholder interviews regarding the system also evidence that potential users of the system agree that it is split up in an intuitive manner, with Student 1 mentioning that 'Each</p>

	<p>system, forming Iteration 1 of developments as a foundation for the rest of the system to be developed upon.</p>	<p>item is labelled well' and Student 3 mentioning that the features are 'split up well'. The various pieces of evidence justify that this success criteria has been met.</p>
<p>2) My system must have an option for users to learn the detailed steps of performing Prim's algorithm on a graph, providing explanations at each stage.</p> <p>This will include:</p> <p>2.1) Selecting a starting vertex from the graph</p> <p>2.2) Choosing an adjacent arc of least weight</p> <p>2.3) Continue choosing arcs of least weight that connect a vertex already in the tree to a vertex not currently in the tree.</p> <p>2.4) Repeating the previous step until all vertices are included in the minimum spanning tree.</p> <p>2.5) Outputting the final MST and weight.</p>	<p>As mentioned previously, Prim's algorithm remained undeveloped after the final Development section of the system. The main reason for this is the time constraints arising from the unexpected depth/complexity of getting various elements of the GUI to function effectively, namely the graph drawing tool and the Simplex tableau panel. Were I to continue development, I think the current state of the system would allow me to implement Prim's algorithm using existing features, for example the existing class for creating graphs, as well as using similar techniques as already implemented to inspect the edges of the graph to find a minimum</p>	<p>As this success criterion remains unsuccessful, no tests were attempted, hence there is no evidence of this being met. As the GUI was developed as a base for the whole system, there are menu items and options that would lead to the Prim's algorithm feature, meaning the system is set up in a way to support further development of this feature. My stakeholders mentioned that whilst having Prim's algorithm in the system would be a high priority feature, they still think the system is functional and effective, especially as it focuses on the difficulties they face, namely, the Simplex algorithm.</p>

	spanning tree.	
3) My system must have an option for users to learn how to perform Prim's algorithm on a distance matrix, providing explanations for each step. This includes:  3.1) Selecting a vertex and crossing out its respective row.  3.2) Searching in the visited columns for the smallest length then crossing out this row and clearly indicated the added/rejected arc.  3.3) Repeating the above steps until a full minimum spanning tree has been created.  3.4) Outputting the final list of arcs contained within the minimum spanning tree.	As mentioned previously, Prim's algorithm as a whole remained undeveloped after the final Development section of the system. Again, this was due to the time constraints of development and more focus placed on the GUI implementation of the other algorithms. While the implementation of Prim's algorithm on a graph may have taken less time, implementing the distance matrix version would have required a new class and further additions to the GUI. As Prim's algorithm is more commonly asked on a distance matrix, I felt that implementing one part of the Prim's algorithm feature whilst omitting the other wouldn't be as beneficial to my stakeholders, for example, if they were looking to run Prim's algorithm on a matrix and the only available option was the graph-based version.	As this feature remains undeveloped, it was not tested during the final testing section, however it was brought up during the student interviews in regard to its priority as a feature. Of the missing features, Prim's algorithm was identified as the most important if I were to continue development, however, as my stakeholders mentioned, the highest priority for them was on the Simplex algorithm feature, being the most difficult topic for them, which was successfully developed. The main context for Prim's algorithm which is identified as the difficulty for my stakeholders is within the Travelling Salesman problem chapter, however, looking through the frequency of questions being examined, the Travelling Salesman problem ranks 5 <sup>th</sup> out of 8 chapters, with only 87 marks worth of questions ever examined since the new 2017 specification:

		<table border="1"> <thead> <tr> <th>Decision</th><th>Marks</th></tr> </thead> <tbody> <tr> <td>FD1ch3 Algorithms on graphs</td><td>185</td></tr> <tr> <td>FD1ch7 The simplex algorithm</td><td>163</td></tr> <tr> <td>FD1ch8 Critical path analysis</td><td>123</td></tr> <tr> <td>FD1ch1 Algorithms</td><td>98</td></tr> <tr> <td>FD1ch5 The travelling salesman problem</td><td>87</td></tr> <tr> <td>FD1ch4 Route inspection</td><td>82</td></tr> <tr> <td>FD1ch6 Linear programming</td><td>61</td></tr> <tr> <td>FD1ch2 Graphs and networks</td><td>46</td></tr> </tbody> </table>	Decision	Marks	FD1ch3 Algorithms on graphs	185	FD1ch7 The simplex algorithm	163	FD1ch8 Critical path analysis	123	FD1ch1 Algorithms	98	FD1ch5 The travelling salesman problem	87	FD1ch4 Route inspection	82	FD1ch6 Linear programming	61	FD1ch2 Graphs and networks	46
Decision	Marks																			
FD1ch3 Algorithms on graphs	185																			
FD1ch7 The simplex algorithm	163																			
FD1ch8 Critical path analysis	123																			
FD1ch1 Algorithms	98																			
FD1ch5 The travelling salesman problem	87																			
FD1ch4 Route inspection	82																			
FD1ch6 Linear programming	61																			
FD1ch2 Graphs and networks	46																			
		<p>Algorithms on graphs tops the list, but this consists of 4 main subtopics being Kruskal's Algorithm, Dijkstra's Algorithm, Floyd's Algorithm and Prim's Algorithm.</p>																		
<p>4) My system must provide users with an option to learn the steps of Kruskal's algorithm in detail, providing justification for each step. This will include:</p> <p>4.1) Sorting the arcs into ascending order of lengths</p> <p>4.2) Adding and rejecting arcs based on whether they form a cycle or not, clearly indicating which arc was added/rejected.</p> <p>4.3) Repeating the above steps until a minimum spanning tree has formed.</p>	<p>This criterion has been partially achieved, through explanation within the Run Algorithm feature. The initial intent was to have a separate feature under the 'Learn Algorithm' sub-option for Kruskal's algorithm, but during the Development of the Run Algorithm feature, I found a lot of explanation going into each step of the algorithm as it was being run, hence this was a lower priority success criterion to have implemented as a standalone feature.</p>	<p>My stakeholders agreed that the 'Run Algorithm' feature did a good job at also serving as a Learn feature, with Student 3 mentioning 'the learn feature has been well incorporated into the run algorithm features, so I don't think a separate feature is necessary', however, this success criterion has been given partially completed status, firstly because there isn't a direct 'Learn Algorithm' feature, and because Student 2 noted that he though the explanations were 'a bit basic'.</p>																		
<p>5) My system must have a random graph generator with the following characteristics.</p>	<p>This criterion has unfortunately not been achieved, however, all the necessary methods</p>	<p>As this criterion was not achieved, no tests were performed to provide evidence for it. The main</p>																		

<p>The graph generator must:</p> <p>5.1) Generate a random number of nodes (within a fixed range)</p> <p>5.2) Generate arcs connecting the nodes, with weight values assigned to each arc.</p> <p>5.3) Ensure that the graph generated follows the definition of a connected graph.</p> <p>5.4) Provide worked solutions for the selected algorithm to be performed on the graph</p> <p>These will form random questions for students to practice Prim's algorithm and Kruskal's algorithm.</p>	<p>to create a random graph already exist in the system. To implement this, I would just need a basic function to generate a random number to represent the number of vertices and a series of random numbers to represent the edge weights of the graph. All the logic for displaying the graph on the screen has been written as part of the graph class, hence this would be a relatively simple feature to implement.</p>	<p>reason this went undeveloped is because the feature was decided to have options for Kruskal's algorithm and Prim's algorithm to be run on the graph, and since Prim's algorithm was also undeveloped towards the development stage of the project, this feature was left out. Having spoken to the students in the stakeholder review regarding this feature, it was only ranked as the most important feature that was missing by Student 3, but for more ease-of-use purposes than for pure functionality of the system.</p>
<p>6) My system must allow users to enter their own graphs to perform an algorithm on.</p> <p>6.1) There must be buttons to add/remove a node from the graph.</p> <p>6.2) There must be a button to connect nodes with an arc and assign a weight to it.</p>	<p>This success criterion was completely achieved during the development of the system. As mentioned during the design phase for Iteration 5, I decided to slightly alter the sub-criteria 6.1 and 6.2 to make the system easier to use for the user. This involved replacing add/remove node</p>	<p>Tests 7-13, 23, 24 and 35-37 from the function/robustness testing show the various GUI elements of this success criterion being achieved, with each button serving its described functionality. Tests 9 – 14 from the algorithms testing section show this feature being tested against a</p>

<p>6.3) There must then be a solution provided which explains the steps involved in applying the algorithm to the entered graph.</p> <p>6.4) The system must also allow students to clear the graph they have created and start a new one.</p>	<p>buttons with an add/remove edge button, which asked for source destination and edge weight, essentially combining criteria 6.1 and 6.2 into one feature that required the user to press less buttons to achieve the same task.</p>	<p>variety of questions from past exams that asked students to perform Kruskal's algorithm on a given graph. The stakeholders also tried out this feature, with Student 3 mentioning that he thinks 'this feature is split up well into various functions like adding an edge, removing an edge and running the algorithm'. Hence, this success criterion has been fully achieved.</p>
<p>7) My system must provide users with fixed examples with working out for the standard Simplex algorithm with all the steps fully explained/justified:</p> <p>7.1) Forming equations from inequalities and adding them to a table, with a maximum of 4 unknown variables</p> <p>7.2) Finding a pivot using theta values and the objective row.</p> <p>7.3) Completing the iterations until no further improvements can be made, outputting the</p>	<p>With more emphasis placed on getting the algorithms working for varying user inputs, this success criterion was unfortunately not achieved. The GUI is still set up in a way such that the feature could be added with relative ease. To do this, I could use the same logic to perform the algorithm on user-inputted inequalities but have hard-coded values that are used every time from example questions. These can be placed in the tabbed pane like how</p>	<p>As this criterion was not achieved, no specific tests were carried out to validate it being met/not met. When consulting the stakeholders about the system and its missing features, Student 1 and Student 3 both picked up on the fact that the fixed examples learn feature was missing from the system, however, Student 3 mentioned now seeing 'how the run Simplex algorithm feature turned out, I think it does a better job at explaining the algorithm with any question the user has, as opposed to the fixed</p>

<p>updated table after each iteration.</p> <p>7.4) Outputting the final value of the objective and all variables.</p>	<p>existing questions are solved, with the scope for extra explanation and possibly an image of the mark scheme to support the working out.</p>	<p>example system' and Student 1 supported this response, mentioning that he 'can also easily enter example questions from the textbook myself if I don't understand them'. While it would be nice to round the system off with fixed examples to introduce users to the topic, my stakeholders justified that the system still works effectively without it, and that the existing features can be used to achieve a similar outcome.</p>
<p>8) My system must allow students to enter their own inequalities into a table to be solved using the simplex method, with a maximum of 4 non-basic variables.</p> <p>8.1) Students must be able to view solutions to their question with clear method shown.</p> <p>8.2) The system must also allow the user to clear any inputs entered into the table.</p> <p>8.3) Identify the most negative value in the</p>	<p>This criterion was achieved, through the 'Run Algorithm' feature of Simplex Algorithm menu item, which allows users to input a number of inequalities using a drop-down menu, select how many variables they would like to add in their question, which then prompts them with the necessary number of input text boxes to input the coefficients of their inequalities and objective function. When they</p>	<p>Tests 14-17, 20-22, 25-33, 38, 39 and 40 from the function and robustness testing evidence that this criterion has been met, showing that each button and option shows the correct number of text boxes/components on the GUI. Each input box has also had input validation performed, to ensure it can handle a variety of input cases, as verified by the stakeholders, with Student 1 mentioning that the system is 'definitely robust in handling a variety of</p>

<p>objective row to identify the pivot column</p> <p>8.4) Calculate the theta values by dividing the term in the value column by the term in the pivot column for all constraint rows</p> <p>8.5) Complete the following and add to a new tableau. Divide every term in the pivot row by the pivot value and write down the row operation. Change the basic variable at the start of the row to the variable heading of the pivot column.</p> <p>8.6) Subtract different quantities of the '1' of the pivot value in the pivot row. Perform the same operation on the remaining values of the row, using the old values for the current row and the new value from the pivot row.</p> <p>8.7) Repeat steps 8.3 to 8.6 until there are no negative numbers in the objective row, before outputting the final values of each variable</p>	<p>press the 'Run' button, each iteration of the Simplex algorithm is shown as a new tab in the tabbed pane, with highlighting to indicate pivot row, column, and value. Each tab contains a tableau, explanation, row operations (where necessary) and theta values, with the final tab containing the final values of each variable, a common follow-up question in exams asked after the students performs the Simplex algorithm.</p>	<p>cases of unexpected inputs'. Tests 1-8 from the algorithms testing section show the system working correctly against past exam questions, with the answers and methods verified against the official mark schemes. The system was able to obtain full marks when performing the algorithm on the variety of questions, deeming it successful in being able to tackle a variety of questions that the user may have, hence this success criterion is fully completed.</p>
---	--	---

and the value of the objective function.		
<p>9) My system must have an introduction each topic to the user. This will be included as a subsection when a specific topic is selected by the user.</p> <p>9.1) The introduction must explain what the algorithm does and give an example of a real-life scenario where it would be used.</p>	<p>Unfortunately, due to priority being placed on implementing the working algorithms into the GUI, this criterion was not achieved. The GUI is created in a way that would allow each topic to have an introduction to the user, with the 3 main buttons on the GUI linking to the introduction for their named feature. Since this feature would not require any algorithmic logic, implementing it would be predominantly focused on GUI development, using text areas and labels to convey information to the user.</p>	<p>No tests were performed to try and evidence this success criterion. When consulting my stakeholders about this criterion not being achieved, they mentioned that for them it was considered to be a low priority feature in the system, since they are already familiar with what the algorithms are used for, but mentioned it would be a nice extra feature to round off the system. Student 2 mentioned that the introductions to the algorithms 'would just be 'nice additions' as opposed to necessary for functionality', and all three students agreed that the system is able to serve its purpose well without them.</p>
10) My system must explain the steps of the various algorithms in accordance with the Edexcel A-Level Further Maths specification.	This success criterion has been achieved for both the Simplex algorithm feature and the Kruskal's algorithm feature. For the Simplex algorithm feature, I ensured that the explanation followed a similar step-by-step	Tests 1-8 show the step-by-step explanation of the Simplex algorithm and process of displaying the tableau after each iteration, comparing the output from the system to the mark scheme. The outputs matched the

	<p>system to the textbook in identifying pivot row, pivot column and pivot value, explaining the theta values and row operations, before performing them on the tableau. For the Kruskal's algorithm feature, as mentioned in the Design section of Iteration 5, I chose a method of implementing the algorithm that most closely followed the way that [SCHOOL NAME ABBREVIATION] students are taught, as opposed to the more conventional method, which would be harder to explain. I also ensured the table of added/rejected edges was included, as this is the main method that [SCHOOL NAME ABBREVIATION] students are taught</p>	<p>sections for which marks were awarded in the mark scheme, suggesting the system is successful in following the Edexcel A-Level Further Maths specification. Similarly, tests 9-13 show the process for Kruskal's algorithm, from running the algorithm to having the table of added and rejected edges listed in weight order, as [SCHOOL NAME ABBREVIATION] students are taught, as well as having a list of all edges included in the minimum spanning tree and the final weight of the minimum spanning tree in the last edge tab for the graph. As each feature closely follows the specification and how students from [SCHOOL NAME ABBREVIATION] are taught, this success criterion has been fully achieved.</p>
11) My system will utilise past exam questions from the Edexcel A-Level Further Maths exam to form the fixed examples for students to learn from. This will include:	<p>As a standalone feature, similar to success criterion 7, this criterion was unfortunately not achieved. Again, the system GUI is designed in a way that would allow</p>	<p>While no tests were performed for this criterion as it was not achieved, tests 1-13 from the algorithms testing section show that the system would be successful in completing</p>

<p>11.1) Displaying the past exam question, what year it is from and how many marks it is worth.</p> <p>11.2) Working through the question, providing methods and explanation for each step.</p> <p>11.3) Outputting a final answer to the question.</p>	<p>for this feature to be implemented without major changes required to the system, and all the relevant methods to complete this criterion have already been created. However, due to time restrictions and more emphasis placed on getting the user input algorithms working, it was not feasible for me to implement a variety of past exam questions as fixed examples.</p>	<p>past exam questions, should they have been used to form fixed examples in the system. Checking the outputs from the system against the mark scheme showed that the system can tackle past exam questions in the way that students are expected to for their exam.</p>
<p>12) My system's user interface must be clearly split into the different features of each subsection. This will include:</p> <p>12.1) Displaying the question e.g. a graph to perform an algorithm on</p> <p>12.2) Alongside the question, displaying fully worked solutions to the question.</p> <p>12.3) Providing an explanation/description for each step of the worked method.</p> <p>12.4) Allowing users to</p>	<p>This success criterion was completed (apart from criterion 12.4 as mentioned part of success criterion 5) throughout the iterations of the system. Iteration 1 focused on creating a base for the GUI upon which other features could be built. Whilst implementing each feature e.g. the Kruskal's algorithm feature, I ensured that adequate sizing and spacing was used, for example, using half the window for the graph to be displayed and the other half to be split up into two smaller</p>	<p>Tests 1-6 show how the system has been decomposed into various sections to make user navigation of the system easier, with clear labelling of each feature. Tests 39, 41 and 42 show how the different features are kept distinct, but work consecutively, allowing the system to be used in various ways as opposed to a single function per use. Test 18 shows how the system has been designed around the 'Home Screen' with the different features revolving around the main window,</p>

<p>generate random values for a graph and select an algorithm to run, providing descriptions for each stage.</p> <p>12.5) Allowing users to enter their own graph into the system, select and algorithm and provide full solutions with explanation.</p> <p>12.6) Allowing users to enter their own values into a Simplex tableau to then be iterated on, with full working and justification for the steps provided.</p>	<p>sections, one to contain the table of added/rejected edges, and another to contain the explanation for the edges. Similar planning took place for the Simplex algorithm, ensuring that the text boxes for user inputs were presented in an easy-to-understand way that would be familiar to users of the system.</p>	<p>from which users are presented with all options of the system. The menu bar is available to access from any feature, allowing users to quickly move between different sections of the program with one action. Consulting the stakeholders in the interviews, Student 3 mentioned that the 'layout is also consistent, using similar sized buttons, recognisable input boxes with prompts, which overall contributes to familiarity with features throughout the system', with the other students agreeing that the system as a whole has been laid out well, hence this success criterion was successful.</p>
---	---	---

Some features that were not explicitly listed as success criteria to be achieved by the development but were necessary for its functionality include:

- 1) Use of various features of the system one after another ensuring functionality between features
- 2) Colour choices throughout the system

As these features were not included in the success criteria, but did form part of the testing for function and robustness, as well as a topic of discussion in the user interviews, the results of each of these extra prerequisites are summarised below.

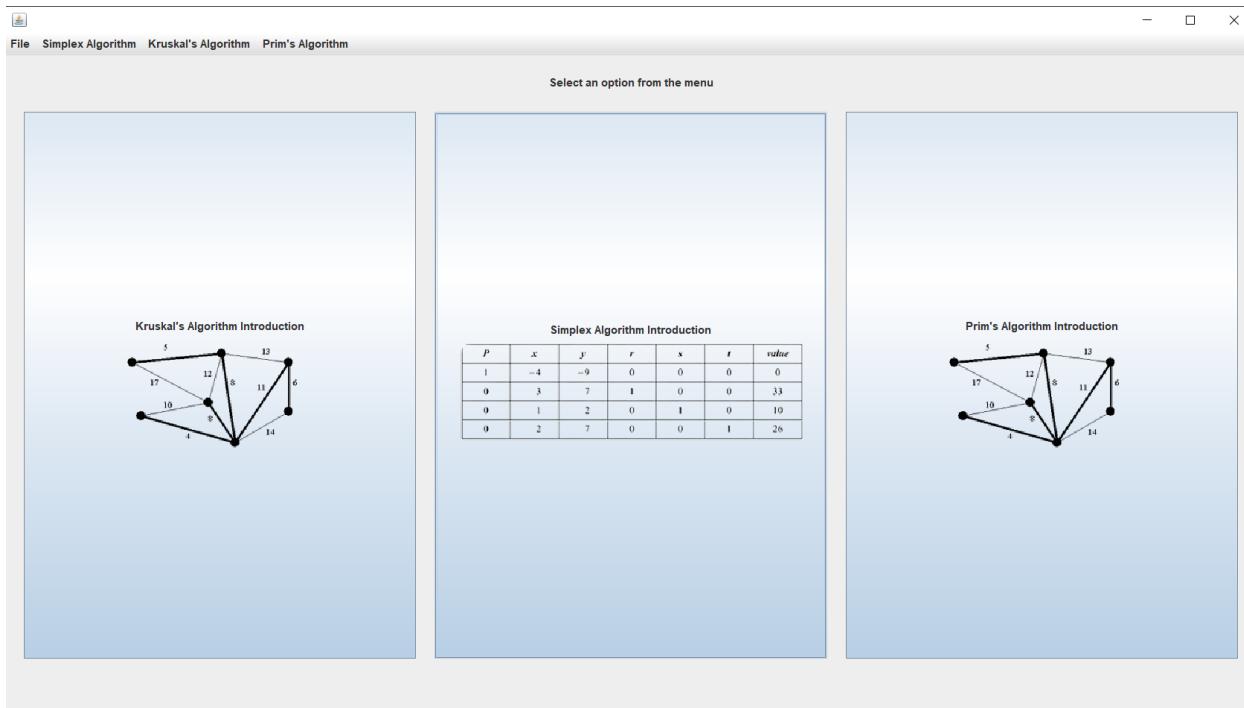
- 1) Ensuring that the system functions well when multiple features are used one after another is also a key for usability in the system, as students are more likely to be switching between different features instead of using the system once for a single question and then closing it. This was achieved and is evidenced through tests 34-39 as well as test 42, which showed multiple features being used partially and consecutively. During the stakeholder interviews, Student 1 mentioned that the system 'manages whole system robustness quite well'. Student 3 described the movement between features as 'seamless'. One thing Student 2 picked up on is that the 'reset' feature for the Simplex algorithm could be more well-defined, as currently going off the feature resets the question: 'having a separate reset feature would be more practical', which would be something to bear in mind if I were to continue developing the system
- 2) Colour choices throughout the system were generally limited to features where I saw them as necessary to enhance the learning method for users of the system. This formed part of the testing indirectly, by ensuring that algorithms were performed as expected, and that colours were displayed as expected. This is evidenced through tests 15, 16, 35, 38 from the robustness/functionality testing, and tests 1-13 from the algorithms testing, which indirectly showed the correct use of colour following the implemented logic. I also asked the students about the use of colour throughout the system during the interviews. The general consensus of colour usage was positive, with Student 1 mentioning the 'colours are clearly differentiable and serve their purpose well'. Student 2 supported this with his response, saying that the colours 'made it easier to follow the iterations of the problem, seeing the changing pivot row, column and value'. However, the students agreed that overall use of colour was minimal, with Student 3 mentioning 'could be some more use of colours in different areas of the program' and Student 2 also saying 'I feel there is some more room for colour in certain places'. Hence, this target was partially achieved, with scope for improvements.

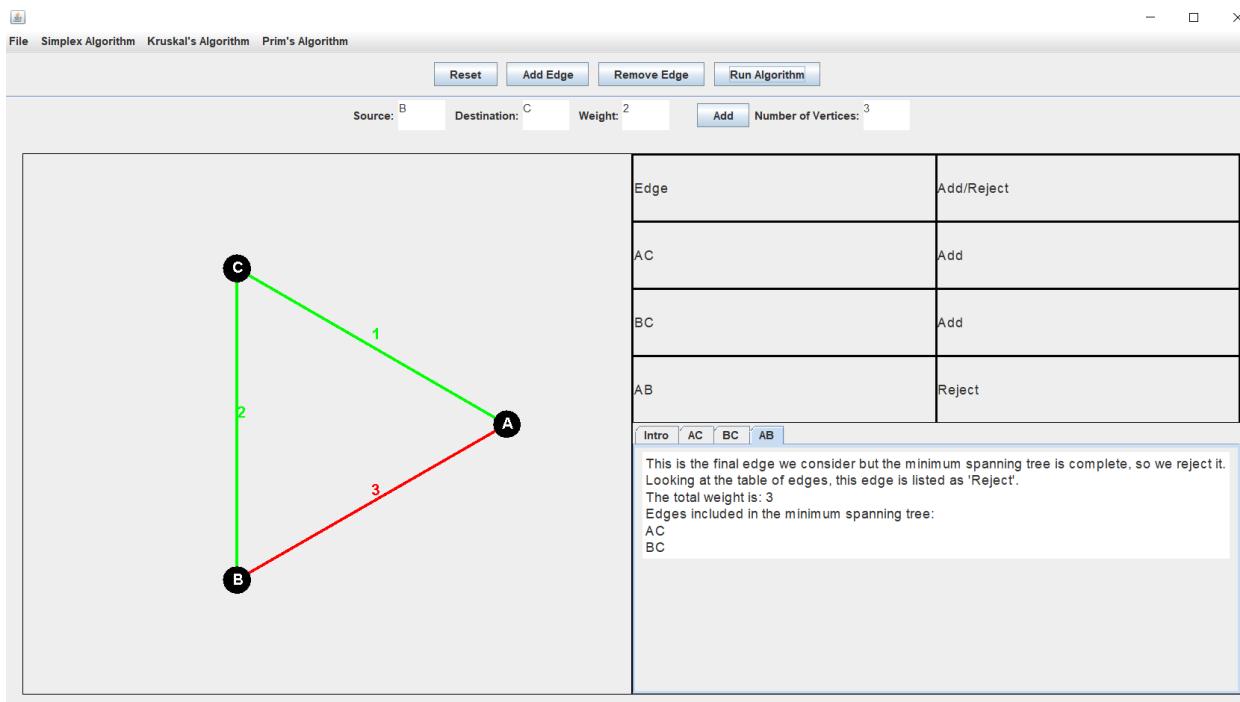
## Usability Review/Evaluation

Usability of the system can be split up into four main areas: layout, ease of input, use of colour, consistency, and I will evaluate each of these sections in relation to my system.

Layout:

The main aim for layout of the system was to create an app in a style that users would be familiar with navigating. When evaluating the existing systems, one of the main features I found to be helpful when navigating between features was to have general options available from all subsections of the program. This led me to implement a menu bar style, which is visible and accessible from all sections of the program.





I also considered layout within the individual features. Each algorithm is quite complex, and students need to be aware of a number of specific steps and points, so I had to ensure all of this can be conveyed on the GUI without it becoming too cluttered (adhering to success criterion 1). I opted to have the hard coded option in a line at the top, for example, when using the Simplex algorithm feature, users will always have to select a number of inequalities and variables, so these options will need to be at the top, however, the number of input boxes that appear will vary based on the prior inputs, so these are designated to the center of the window. I also wanted to present the inequality input boxes in a way that mimics the shape of the tableaux, and the way they are often presented in exam questions, one on top of another. Since it is necessary that the input boxes are seen by the user, I opted to have them in the center of the GUI as opposed to anchoring them to the left or right.

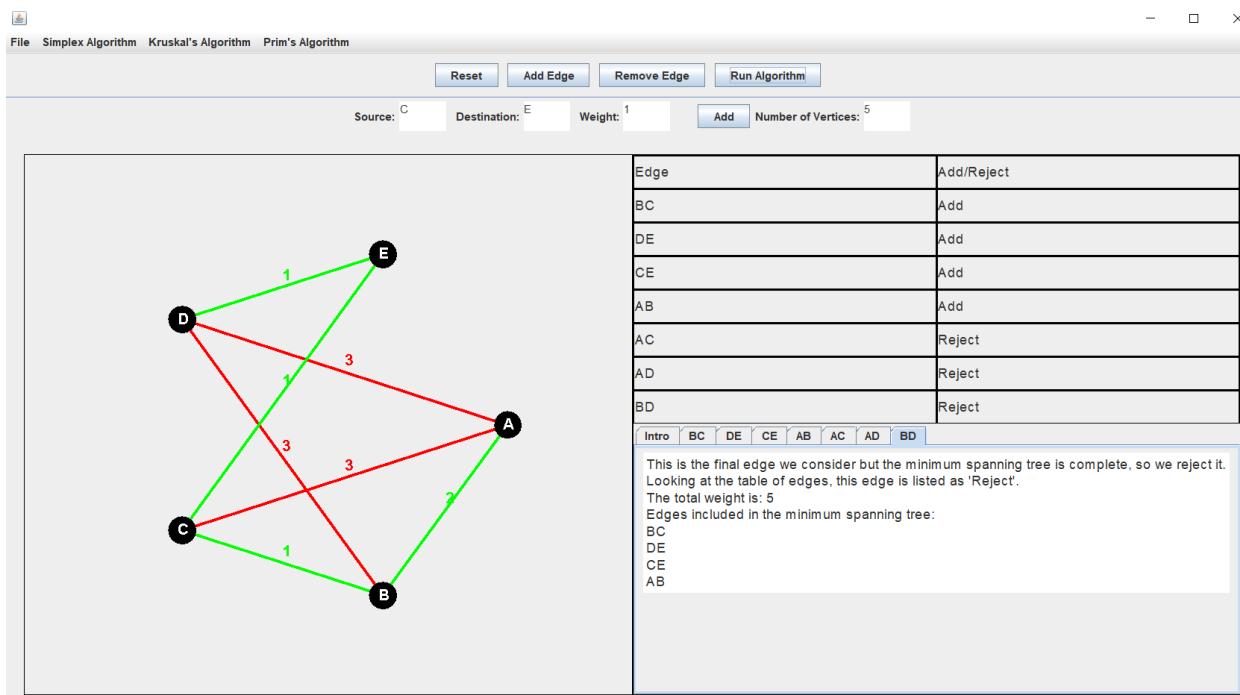
Since the input boxes are in the center, I also opted to have the tableaux/working out also in the center. Using a tabbed pane allowed me to ensure that no matter the number of tableaux a question required, all of them can fit on the GUI window and are visible.

B.V.	x	y	r	s	Value	$\theta$
r	5.00	7.00	1.00	0.00	79.00	15.80
s	10.00	3.00	0.00	1.00	60.00	6
P	-3.00	-2.00	0.00	0.00	0.00	-

Looking through the objective row, the pivot column was selected as 'x', highlighted in blue, as this has the most negative value in the objective row. By dividing each number in the Value column by the respective value in the pivot column and selecting the smallest positive result, the pivot row was selected as 's'. This gives us the pivot value as '10'. Our aim is to reduce the pivot value to 1 and everything else in the pivot column to 0, so our first row operation is [the values in the pivot row] / '10'. We now look at each value in the pivot column by turn. We must see how we can reduce the value to 0 using the pivot value, which is now 1. We take a value from the pivot column that is not the pivot value, we then subtract the value multiplied by the pivot row, so if the value is 5, we take 5 \* 1 away from it, giving us 0. This process is repeated throughout the row, using the respective values in the pivot row. We then repeat which each non-pivot value in the pivot column. We repeat the process of applying row operations until we have completed the tableau.

When deciding on the dimensions for the tabbed pane, I was overly cautious about the gap between the top of the tabbed pane and the objective row input boxes, especially when students chose 4 inequalities, which led the input boxes to take up more vertical room. However, in fixing the dimensions of the tabbed pane, there is a lot of white space left in scenarios where there are less inequalities chosen. This was picked up on by the stakeholders, with Student 1 mentioning ‘the whole explanation box could use up more room both vertically and horizontally, just to make the system clearer’.

The Kruskal’s algorithm feature was also deliberately split up in a way such that the displayed graph could take up enough room to allow for users to enter a feasible number of nodes and edges. This was accompanied by the table of added and rejected edges, which also took up a quarter of the main section of the window, as this is a key part of the working out. I opted for a similar system using a tabbed pane for the edges, creating a ‘sequential’ effect of clicking through the edges in order, similar to how they are considered when performing the algorithm.



Like the Simplex algorithm feature, I again chose to use a bar of buttons which are present throughout the use of the feature, with the input boxes underneath in the main area of the GUI as these will change based on the user’s chosen option. For example, the ‘Add Edge’ button contains input boxes for weight and number of

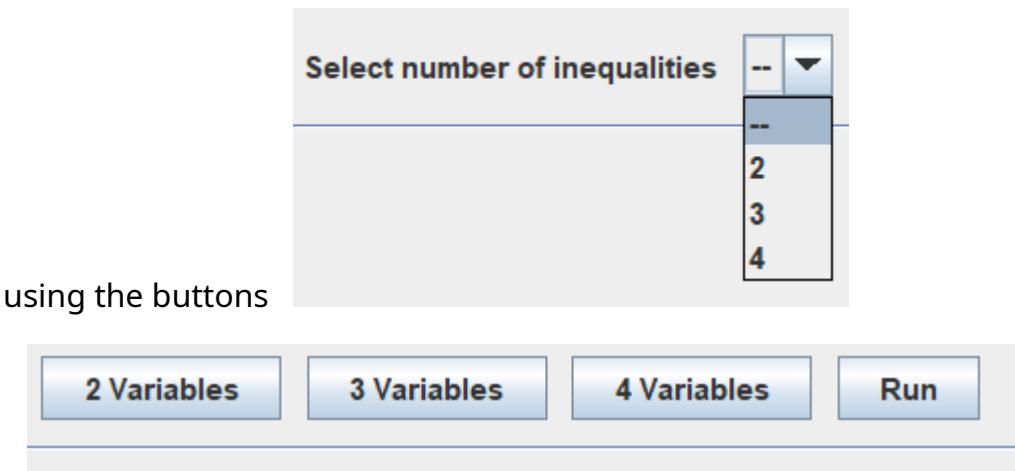
vertices, but the 'Remove Edge' button only contains input boxes for source and destination nodes. I also used a JSeparator line, seen below the row of buttons to emphasise the separation between the fixed components of the system and the changing components, providing a changing 'main section' of the GUI, with the controls out of the way to maximise room for the necessary explanation. If I were to improve this, I would place more emphasis on the explanation for each of the edges, as the students mentioned that they currently 'lack detail'. Another factor to consider is that the textpane within the tabs doesn't maximise the available space, resulting in areas of whitespace, which could be used to either add more explanation, or could be traded for more space for the graph. An alternative option to the 50: 25,25 split would be a 60: 20, 20 split, to allow for more room to display the graph, whilst still conveying the necessary information regarding the algorithm.

Overall, the layout of the system had thought put in regarding the justification of sizing and location of various components, but could do with some refinements to ensure the system is fully accessible. One point is that as part of the Simplex algorithm feature, certain inputs to the system are unclear, with Student 1 mentioning that while the inputs for him were 'intuitive', it would be better 'to ensure it remains accessible for all users' by 'having some sort of text explaining to follow from left to right'. Smaller features like this that round off the system would form the majority of improvements in the layout section of usability

#### Ease Of Input:

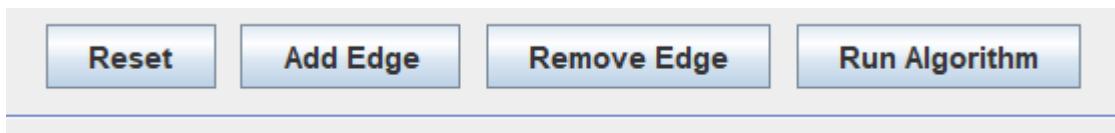
The inputs for this system can be split up into two sub-sections, prompted inputs and custom inputs. The prompted inputs come in the form of buttons and other interactive features with fixed choices, for example, selecting either 2,3 or 4

inequalities from the drop-down menu, and selecting either 2,3 or 4 variables



In this case, the order of inputs is important, with users having to enter the number of inequalities before selecting the number of variables. This is subconsciously prompted by placing the drop-down menu first, however, a key point to consider is that this is not the default option in parsing text for all people, and some [SCHOOL NAME ABBREVIATION] students may try to select the number of variables first. This was made robust by adding an input validation error message in case the user provides inputs in the wrong order, however, as mentioned by the stakeholders, a better option may be to have a short line indicating the order to them so they don't have to try, receive an error and then be prompted again.

Further prompted inputs are evident in the Kruskal's algorithm feature, however, order is not important here.



As mentioned in the Design section for Iteration 5, the 'Reset' button was deliberately placed on the opposite end of the 'Run Algorithm' button to avoid one being pressed accidentally instead of another, since the 'Add Edge' feature lets you continually add edges until you click off the feature, the user won't be continually clicking in that area unless they want to deliberately reset the graph. These buttons were also labelled clearly to ensure that each of their functions was properly understood. This was done in response to the stakeholder criticism of the

initial mockup for the Simplex algorithm GUI feature, where two buttons 'Confirm' and 'Run' were ambiguous in their purpose. After changing the labelling for the Simplex algorithm feature:

Select number of inequalities

I kept in mind that for the subsequent features developed, clear labels for any inputs would be required.

The next subset of inputs to consider is the custom inputs, where users can input their own values and are only prompted by a text area in which to enter their desired inputs. I opted for labels adjacent to each input box to indicate its purpose, making sure there was adequate spacing between the label for a specific box and the label for another box, to ensure there was no ambiguity regarding which box the label was referring to.

$x$	+		$y$	=	
$x$	+		$y$	=	
$x$	+		$y$	=	
$P =$		$x$	+		$y$

Source:  Destination:  Weight:  Add Number of Vertices:

The stakeholders brought up the use of clear labelling in the interviews as a beneficial feature of the system, as it contributed to the overall ease-of-use, with Student 3 mentioning that he 'found every input to be well-labelled and intuitive'.

Another factor Student 3 brought up was that 'the input boxes and buttons were of a good size and there wasn't too much effort from me as the stakeholder to place my inputs for the system'. It was important to consider the relative sizing of

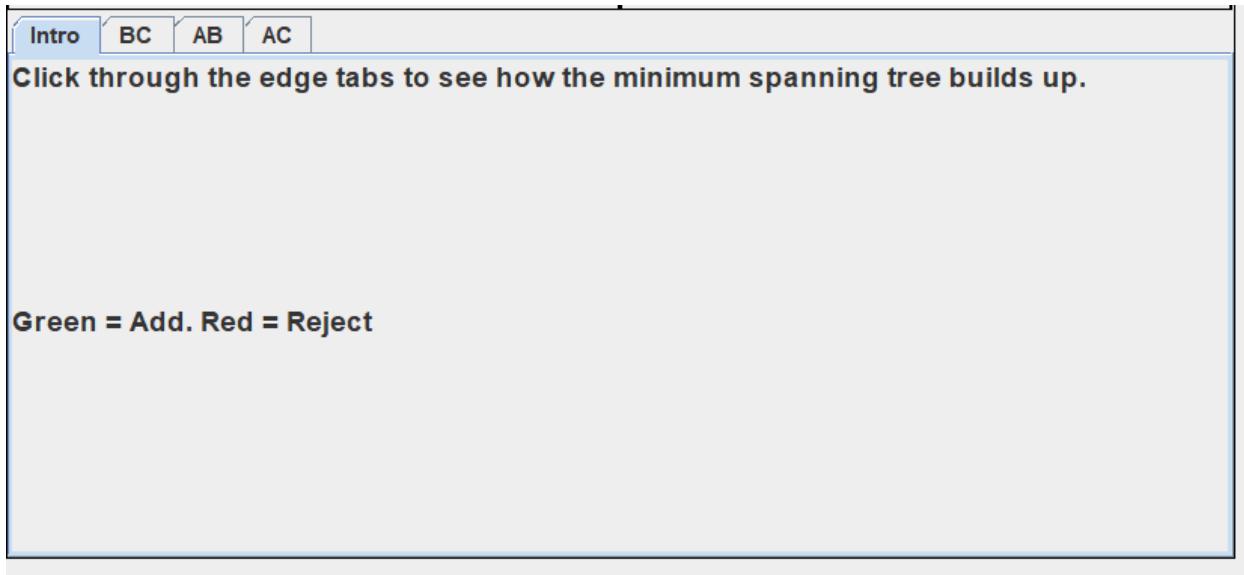
various input components, such as text areas to ensure that they were visible to the users. This was especially important for the Simplex algorithm feature, which required up to 20 input boxes if the 4 inequalities, 4 variables option were to be chosen. This meant that a balance had to be struck between having adequate room for other components and not being too small such that the size of the input boxes impedes the usability.

For the Kruskal's Algorithm feature, similar labelling for each input box was required to ensure users were aware of its purpose:

**Source:**      **Destination:**      **Weight:**      **Add**      **Number of Vertices:**

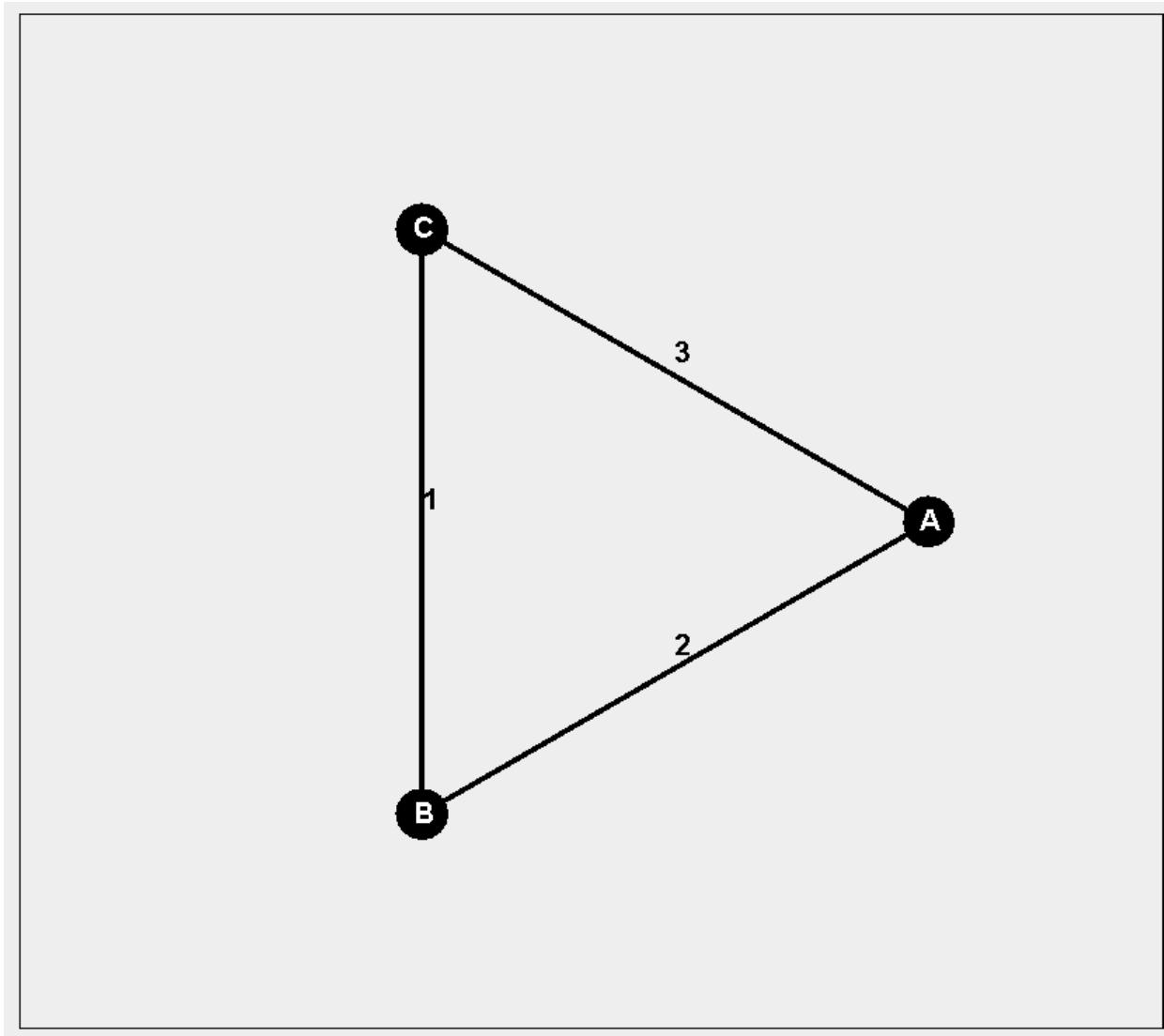
During the design and development stages, this seemed to be the most straightforward and practical method of taking user input, however, it was brought up by Student 2 during the interviews that he found it 'a bit tedious having to click between the boxes for source node, destination node and weight of the edge especially for a graph with 10+ edges'. Having input boxes was an easy-to-understand way for users to input a graph, however this sacrificed part of the usability of the system, forming an important feature to consider as a further improvement to the system. This could potentially be achieved through the use of hotkeys, allowing for quick switching between the input boxes using the keyboard, or as Student 2 suggested for a more advanced version, having a tool to scan a graph from an image and paste it into the correct format for the system to perform the algorithm on.

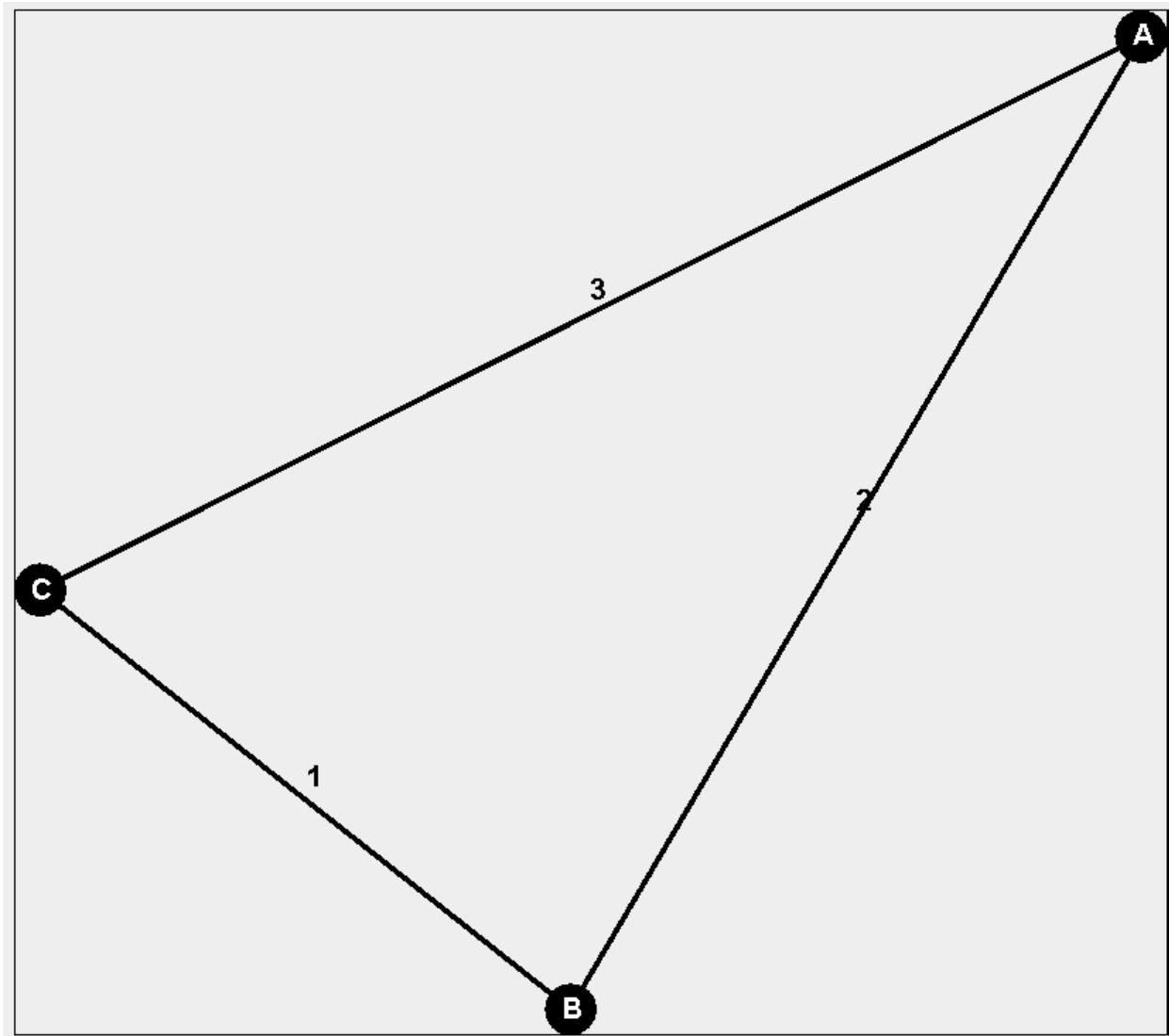
Another input to the system is the user interacting with the tabbed pane, either for the iterations of the Simplex algorithm, or for the individual edges of the graph as part of Kruskal's algorithm.



Since these inputs are less obvious than more conventional components like buttons, I felt it was necessary to add a small prompt to the users, indicating the purpose of the tabs. Java's JTabbedPane also does a good job at subtly hinting to the user that they can interact with it, highlighting the currently selected tab in blue while the others remain the default colour. Combining this with the prompt message ensures that users are aware of this feature and can access it easily.

A further input to the graph is the feature that allows users to move the vertices of the graph within the Kruskal's algorithm feature. The stakeholders in the interviews mentioned that whilst this feature it was useful, it could go unused, with Student 3 mentioning 'there is no prompt to move around the edges on the GUI, so if I had not just tried moving them around, I wouldn't know this is a feature of the system'. A common theme with features around the system is that they are often missing a prompt to the user, which impacts the usability. Many of these features could do with an extra JLabel to the JPanel containing the drawn graph, indicating to the user how they can interact with the graph in this way. In general, the stakeholders agreed that while it was a useful feature, as it allows them to format their inputted graph in a completely customised way, however, they mentioned it could be introduced in a more understandable way, rather than relying on students just happening to try and move the nodes of the graph.



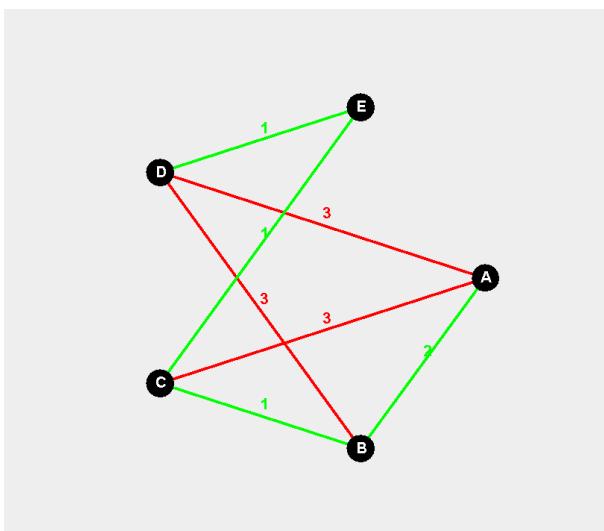


#### Use of Colour:

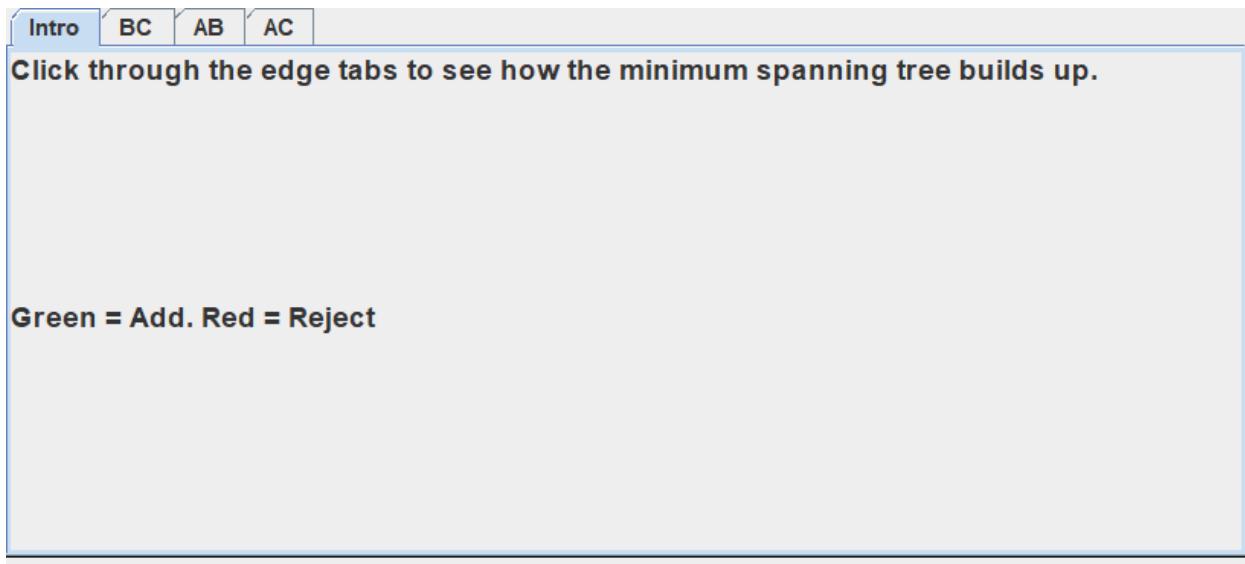
Colour in the system has been deliberately used in 2 key areas, being the Simplex algorithm tableau (highlighting the pivot row and column) and the Kruskal's algorithm graph (highlighting added edges in green and rejected edges in red. As mentioned during the Analysis section, adding a variety of learning methods to the same tool was seen as beneficial to the stakeholders e.g. the explanation to benefit those who learn by reading, and having colours for those who learn better through visual example.

B.V.	x	y	r	s	Value	$\theta$
r	5.00	7.00	1.00	0.00	79.00	15.80
s	10.00	3.00	0.00	1.00	60.00	6
P	-3.00	-2.00	0.00	0.00	0.00	--

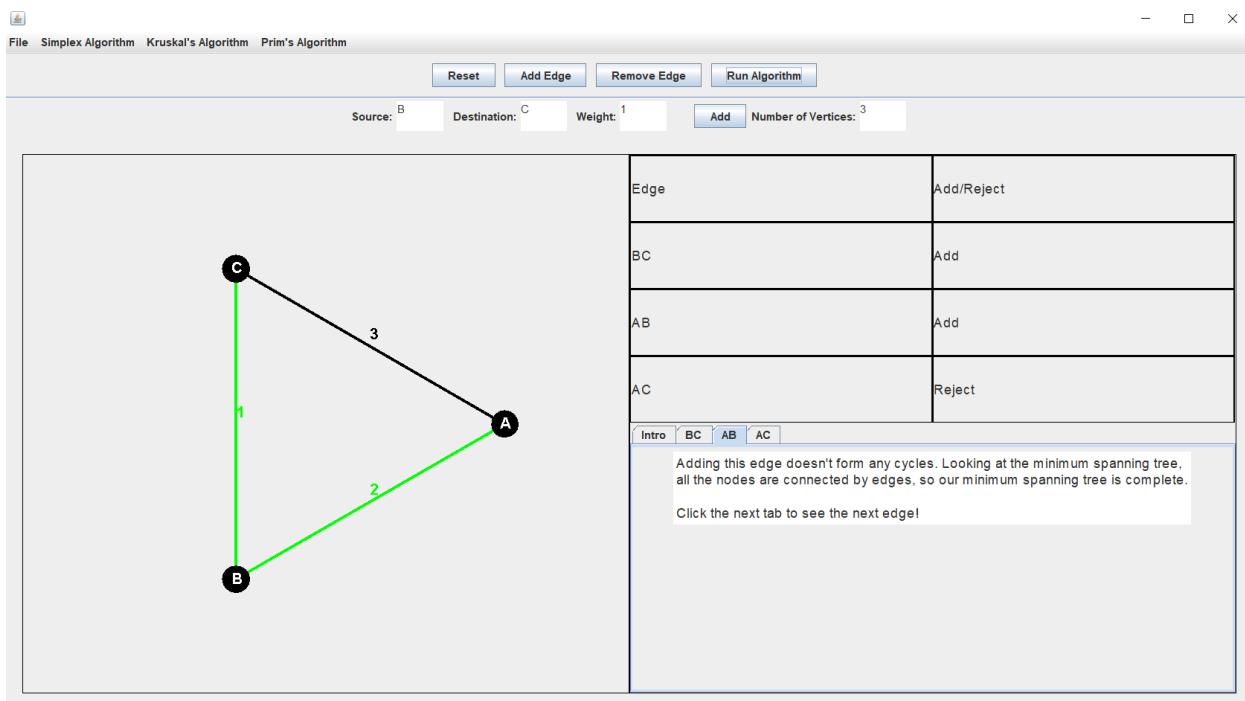
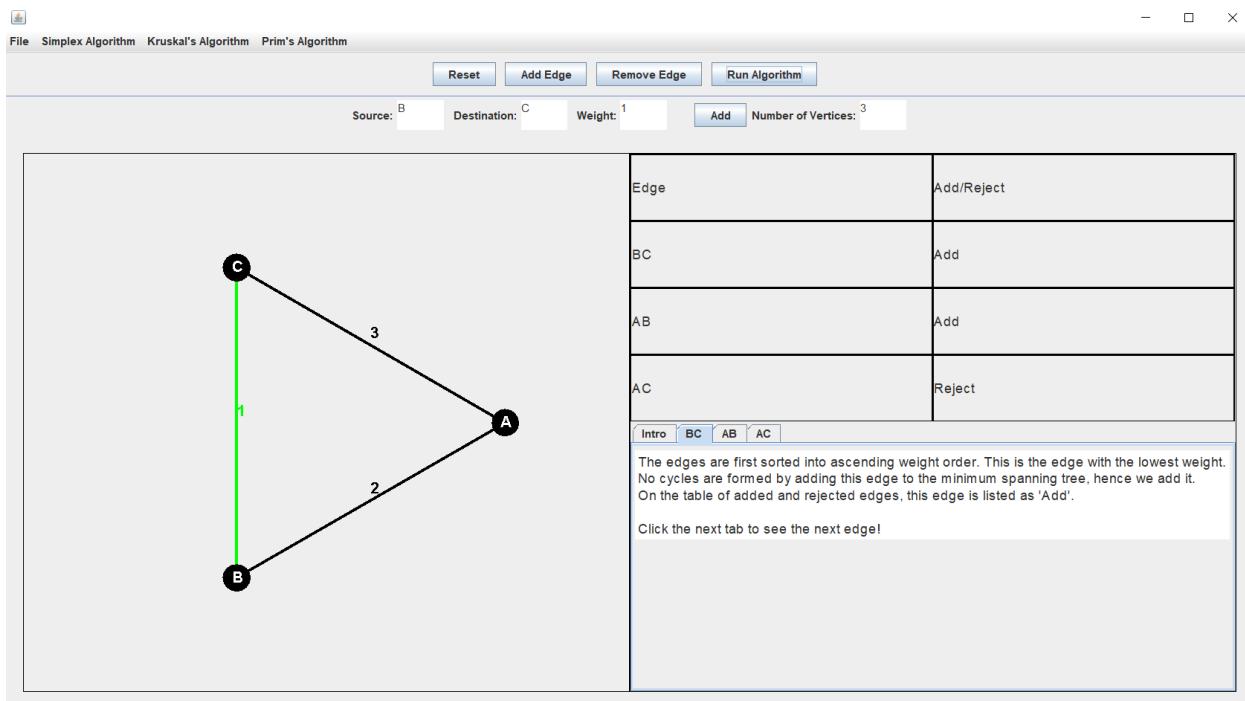
For the Simplex tableau, I took inspiration from the user of colours from the Pearson ActiveLearn Textbook, however my stakeholders brought up the fact that the green used by the textbook resembled the blue too closely. I opted for colours that stood out enough to be purposeful but were also not conflicting with each other. One thing to note is that having colour within the tableau panel helps to make up for the fact the panel remains on the smaller size, however, to make the whole feature more effective, it would be better if the whole panel were larger and had the relevant colours too. For the Simplex feature in particular, Student 1 did mention that 'some sort of preset that explains the colours off to the side would be useful' as currently the only explanation of the colour choices is in the actual explanation. This leads me onto evaluating the choice of colours in the explanation for the tableaux. One thing that the stakeholders definitely picked up on was that while the explanation was detailed and beneficial, the use of the same colour and the same style of text made it slightly difficult to take in information effectively, with Student 3 mentioning 'explanations are created well, but at times can become quite lengthy e.g. for the individual Simplex tableaux which is just a paragraph of text, all the same size'. His recommendation and something I also considered a viable option for improvement was to emphasise key parts of the explanation using bold text or making it a different colour to stand out.

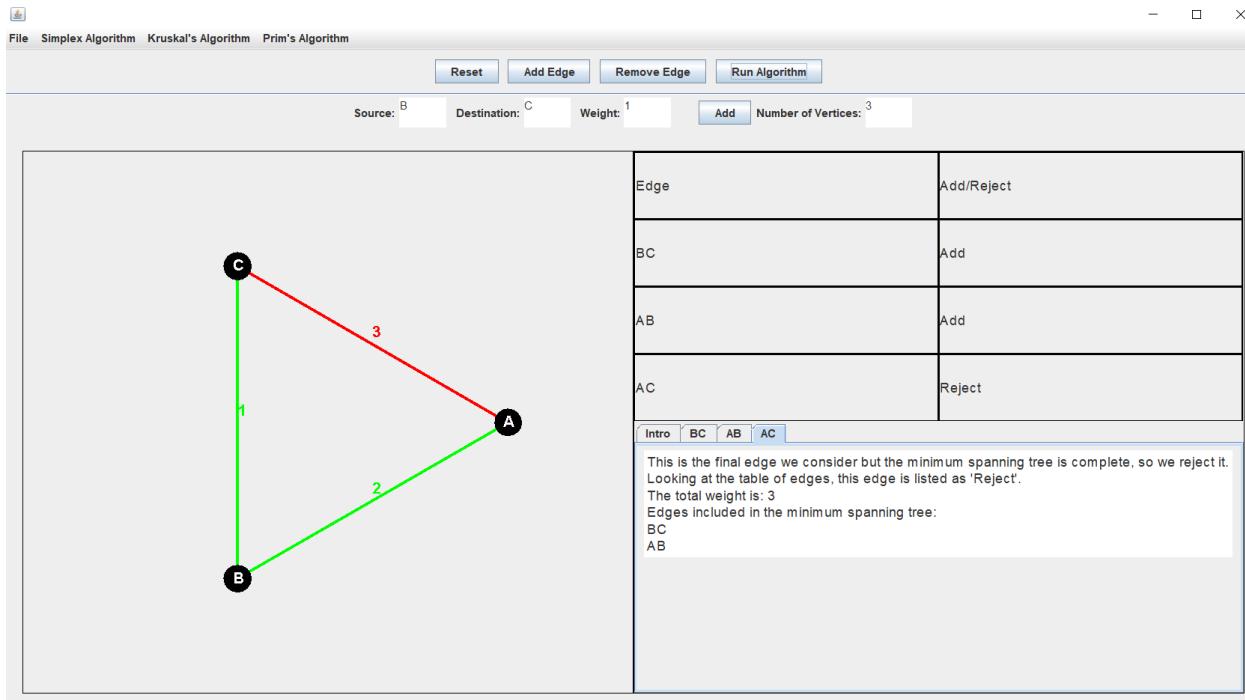


For the Kruskal's algorithm feature, the main use of colour was in the edges of the graph, which are coloured either green (added) or red (rejected). To avoid any ambiguity regarding the colour choices, I ensured that these were well explained in the introductory tab of the edge panel:

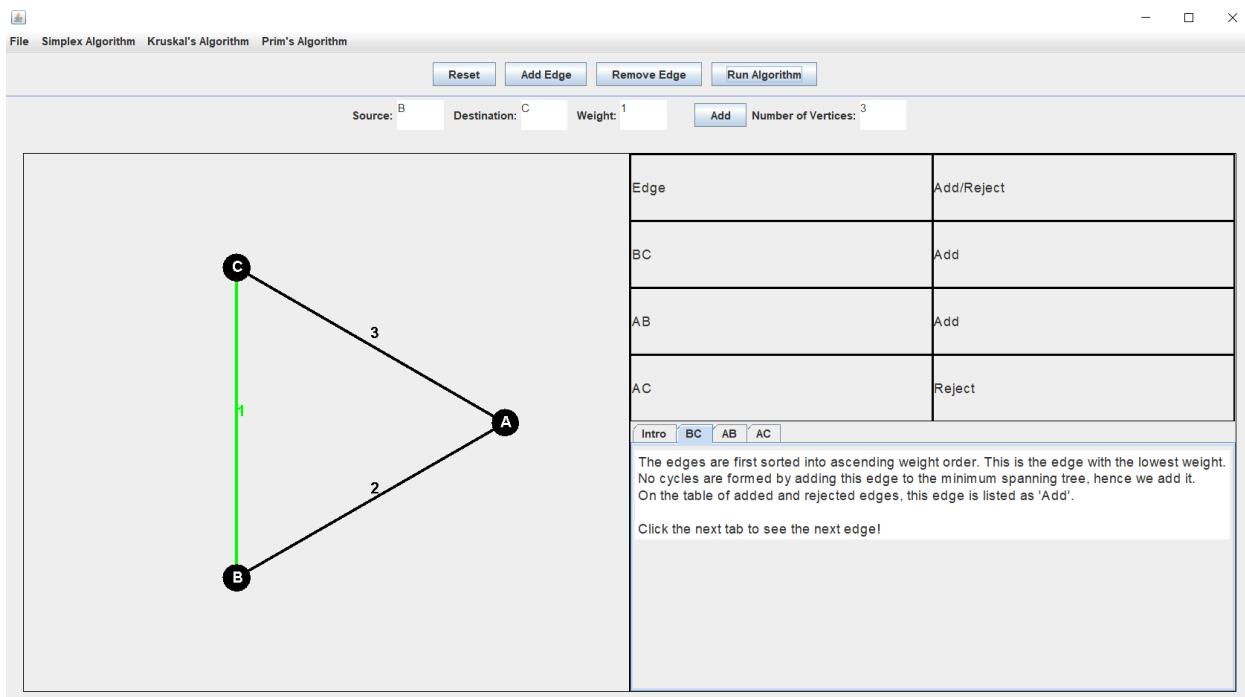


This is presented to users immediately after they select 'Run Algorithm', and is clearly presented in the center of the tab to ensure clarity for the user. By clicking through the edge tabs, the edges of the graph are coloured in one-by-one, building up the minimum spanning tree. This also allows for the user to click back to a previous edge and return the colouring of the tree to the state it was in when only that number of edges had been considered. This is demonstrated below, firstly clicking through all of the edges one by one.





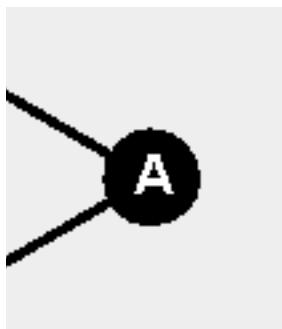
And then returning to the edge 'BC', which removed the colouring for 'AB' and 'AC', given that they hadn't been considered by the time 'BC' was considered:



Initially, when clicking through the edges, returning to a previous edge would just maintain the colouring from the final tab they selected, however, I realised that some users would want to run through the algorithm on the same graph again,

without having to re-enter it, especially for larger graphs, hence this feature was added to ensure they can keep returning to any edge they want as many times as they want. Regarding the colours in the student interview, Student 1 mentioned 'red and green are well explained making them unambiguous', hence we can say that this use of colours adds to the usability of the system.

Another subtle use of colours was in creating the actual visual graph itself, having black nodes and edges, with white text for the node labels. The nodes and vertices were chosen to be black as this is how graphs in exam questions are presented, and appears the clearest on a white background.



White text seemed to be the obvious choice to ensure the the node labels are clearly visible among the black nodes. One option I could have opted for was having the node label separate to the node, offset slightly, similar to how the edge weight label is slightly offset but adjacent to the edge. However, the main drawback of implementing this was that when moving around the nodes and edges, there may be too many components detached from the actual graph which could end up overlapping or being covered by the actual nodes, so the best option was to keep the labels within the node.

The pre-defined Java components also include some basic colour, for example the buttons and the drop-down menu, which are coloured blue as opposed to the generic white/off-white colour of the panels and background. This helps to make interactive components clearly differentiable within the system. One detail that Student 1 picked up on as a suggested improvement for the colours in the system was 'making the 'Run' algorithm button a different colour to the others'. The benefit in this would be to create a distinction not just between the buttons and the GUI but also amongst the buttons based on their purpose. The 'Run Algorithm' button is the last button the user presses before the algorithm is performed,

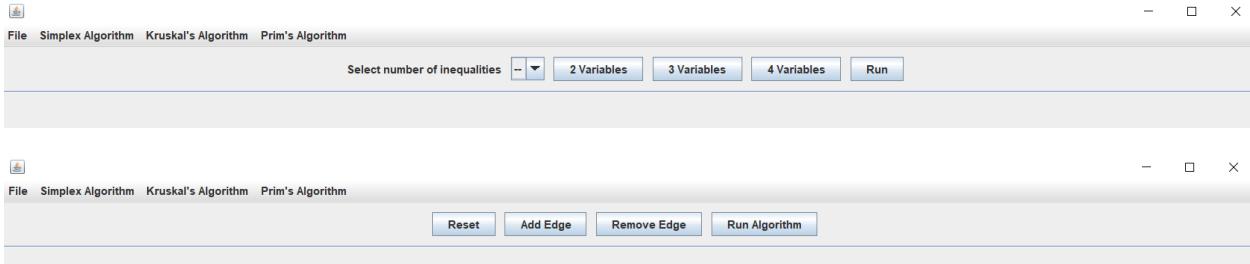
hence the system could benefit from greater use of distinguishing its purpose from more generic buttons such as the 'Add Edge' and 'Remove Edge' buttons.

Upon interviewing my stakeholders, colours seemed to be an area for improvement in the system, with the students agreeing that the current usage is implemented well, but the system could do with more. Some helpful suggestions were provided as to where the system could benefit from more colour, with Student 3 saying 'I think there could be some more use of colours in different areas of the program, for example, making the Kruskal's table of added and rejected edges stand out, since the background of the table seems to blend in with the rest of the GUI' and Student 2 reinforcing this with 'there is some more room for colour in certain places'. As the table of added and rejected edges is also considered a vital part of the Kruskal's algorithm feature, it is understandable that some deliberate changes should be made to ensure it stands out amongst the rest of the GUI. The table is the main component that students learning to perform the algorithm will be looking for, as this is the method they must recreate in the exam, hence creating a distinction between the table of added and rejected edges and the background would be beneficial and a future improvement that would be valuable to implement. The main point to extract from this is that the stakeholders do find the colours to be a beneficial feature of the system and having been presented with colour options in certain aspects, they feel that the system would be stronger if the whole system followed a similar pattern.

#### Consistency:

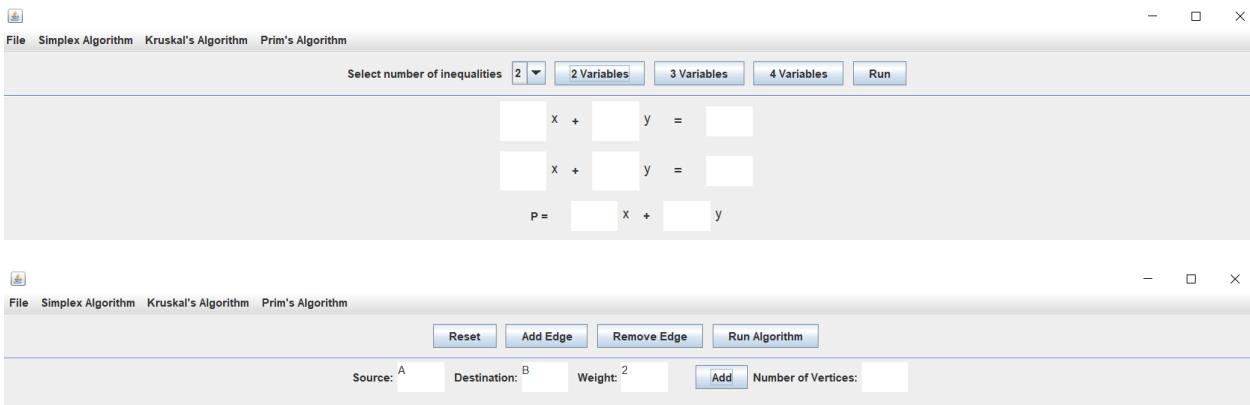
Consistency refers to the use of similar techniques and styles within the GUI between various features. Having features that are familiar between systems helps to improve the usability of the system, reducing the learning curve of a new feature when it is presented in a similar manner to a familiar feature. I considered consistency when developing my system by ensuring inputs to the system are presented in a similar way for the Kruskal's algorithm and the Simplex algorithm features.

Both of these features contain a row of buttons at the top, forming the 'hard-coded' options that must be selected in order to continue:



There is also a separator line included in both, underneath the row of buttons to distinguish the top of the main window, where the fixed options are presented, from the center of the window where the components that appear are based on specific user inputs, and these components allow for custom user input e.g. the boxes for the inequalities in the Simplex algorithm or the text boxes for the source, destination and weight for adding edges in Kruskal's algorithm. The buttons were also presented in the center section of the top of the window, making sure they are prominent and visible.

Another form of consistency is by having the input boxes appear in the same place for the Simplex algorithm and Kruskal's algorithm. While the Simplex algorithm requires many more boxes for the inequalities, these all appear in the center of the GUI, just below the row of buttons



As shown, while the inputs vary in terms of what is required, a similar style is maintained with how they are presented. The textboxes remain roughly the same size between features, made slightly smaller for the Kruskal's algorithm feature to allow for more room for the various other components that must all be displayed. The buttons are also a similar size horizontally and maintain the same vertical size for both features, adding to the consistency between features.

The last prominent feature aimed at maintaining consistency within the program is the use of tabs in a tabbed pane for both features; moving between the iterations of the Simplex algorithm and clicking through the edges for Kruskal's algorithm. Initially, the idea to use a tabbed pane was to ensure that all the Simplex tableaux could fit on the GUI, as my original idea to have them just displayed on the main window had flaws, namely, that there may not be enough room for 3+ tableau, which is an average number of iterations of the Simplex algorithm. Therefore, to contain the various tableaux, as well as creating the effect of moving through the iterations instead of having the entire result statically presented, I opted for a tabbed pane.

Iteration 1						
Iteration 2						
Iteration 3						
B.V.	x	y	r	s	Value	$\theta$
r	5.00	7.00	1.00	0.00	70.00	14
s	10.00	3.00	0.00	1.00	60.00	6
P	-3.00	-2.00	0.00	0.00	0.00	—

Looking through the objective row, the pivot column was selected as 'x', highlighted in blue, as this has the most negative value in the objective row. By dividing each number in the Value column by the respective value in the pivot column and selecting the smallest positive result, the pivot row was selected as 's'. This gives us the pivot value as '10.0'.

Our aim is to reduce the pivot value to 1 and everything else in the pivot column to 0, so our first row operation is [the values in the pivot row] / '10'. We now look at each value in the pivot column by turn. We must see how we can reduce the value to 0 using the pivot value, which is now 1. We take a value from the pivot column that is not the pivot value, we then subtract the value multiplied by the pivot row, so if the value is 5, we take  $5 * 1$  away from it, giving us 0. This process is repeated throughout the row, using the respective values in the pivot row. We then repeat which each non-pivot value in the pivot column.

We repeat the process of applying row operations until we have completed the tableau.

When considering how to implement the feature of clicking through the edges for the Kruskal's algorithm, my initial idea was to have 'forward' and 'backward' buttons to allow the user to move between edges of the graph, however, I felt it was necessary to be able to directly click to a specific edge rather than have to use the same button multiple times to reach a certain edge, which would become tedious for larger graphs. Combining this requirement with the opportunity to maintain consistency, as well as using familiar methods in the development stage of this feature, I opted for using a tabbed pane to represent the edges of the graph.

Intro	BC	AB	AC
<p>The edges are first sorted into ascending weight order. This is the edge with the lowest weight. No cycles are formed by adding this edge to the minimum spanning tree, hence we add it. On the table of added and rejected edges, this edge is listed as 'Add'.</p> <p>Click the next tab to see the next edge!</p>			

This allows me to keep the feature compact, combining the explanation for each edge with the feature that allows the user to click through the edges, as well as creating a sense of familiarity within the system. Those who first use the Simplex algorithm feature and then try this feature will be familiar with interacting with a tabbed pane and will make navigation of this feature easier. The same is true vice versa, where having seen the method of clicking through tabs as progression through the algorithm, users are familiar with this technique in the Simplex algorithm feature for clicking through the iterations of the algorithm.

Consistency within the program helped with the usability of the system, and while not a specific question, this was brought up by Student 3, who mentioned 'Within each feature the layout is also consistent, using similar sized buttons, recognisable input boxes with prompts, which overall contributes to familiarity with features throughout the system'. This suggests that considering the sizing and techniques used for various GUI aspects like input areas and buttons, the system became slightly easier to use as opposed to having completely contrasting methods of presenting components between different features.

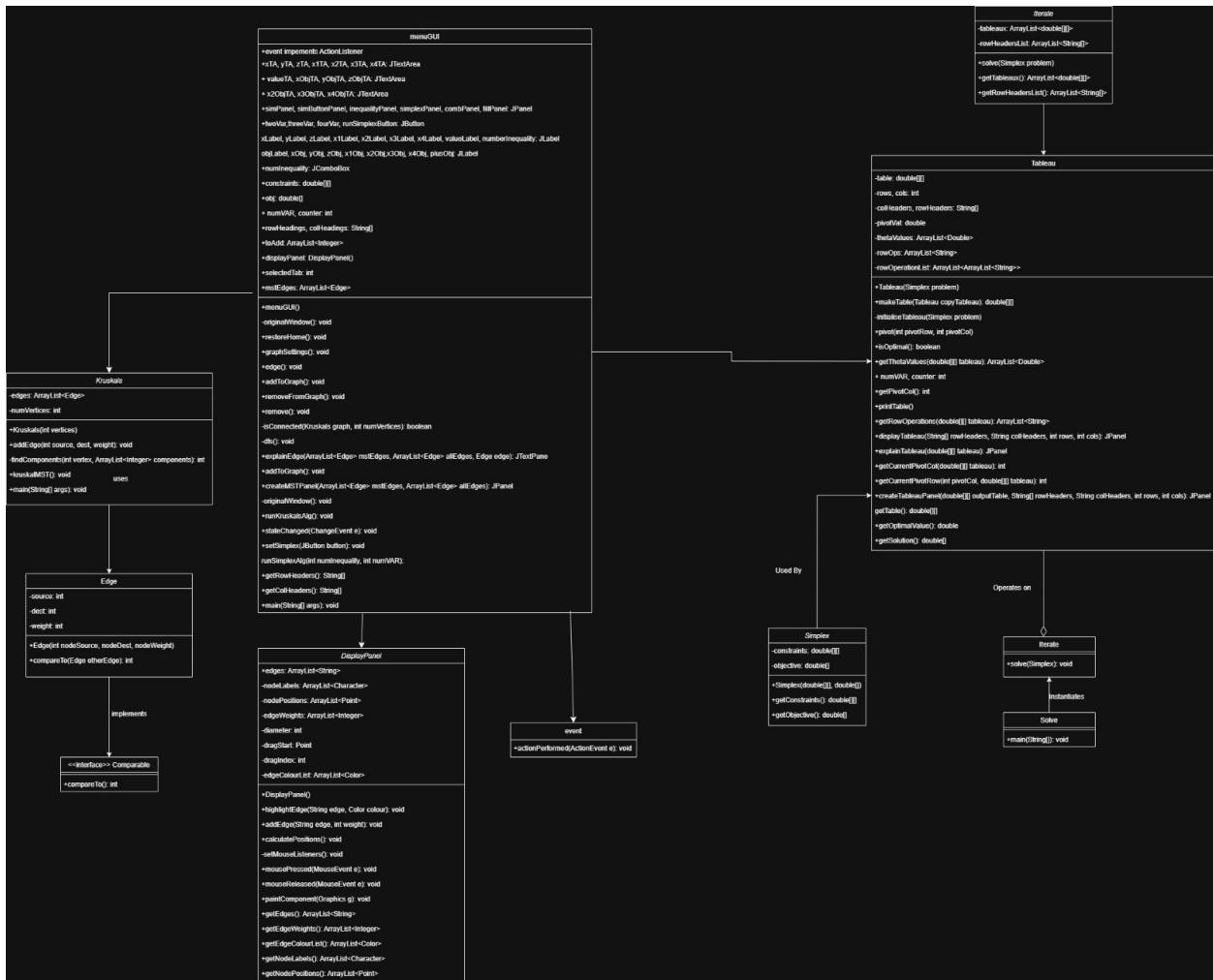
With overall usability, the factors considered seem to be successful in making the system more usable for my stakeholders, with Student 2 mentioning that he found this 'an easy system to use/navigate', however, the general consensus is that since each feature is beneficial, the system would be better overall if each of these usability factors were developed to a greater extent e.g. more use of colours and more prompts for input.

## Maintenance Review

Having evaluated the system in regard to the end-users, it is also important to consider further development of the system, and how the program developed at present has been created in a way such that a seamless transition could be made for another developer to pick up the project and continue.

The basic factors for system maintenance involve naming of classes, methods, attributes and variables, as well as the use of comments throughout the code.

Regarding the naming throughout the code, we refer to the UML diagram showing each class part of the development, on page 170 (copied below for easier access):



The largest class is **menuGUI**, knitting together the various classes for individual algorithms using the GUI as a bridge between them. From the name, we can infer that this class contains methods regarding the user interface/menu of the system, so a new developer would know that in order to add functionality to a new feature e.g. an option in the menu bar, this would take place in the **menuGUI** class.

The next largest class is **Tableau**, which performs various methods relating to the Simplex tableaux. This is named based on the function of the methods in the class, so a new programmer trying to implement a feature in the tableaux such as changing the colour choices, or see how a specific feature works e.g. how the tableaux are constructed, would know that the relevant methods are contained within that class. The **Iterate** class above the **Tableau** class is named as per its

function, performing an iteration of the Simplex algorithm on a provided tableau. One feature to note is that the `getTableaux()` method and `getRowHeadersList()` method weren't used throughout the program, but were created as generic getter methods for any future requirements e.g. if a new programmer were to access the list of tableaux or the list of row headers, the methods to access them have already been created.

The class `DisplayPanel` was created separately from the Kruskal's class. As discussed previously, the `DisplayPanel` class is used to create a generic graph on the GUI, which is currently only used by the Kruskal's Algorithm feature. However, if a new developer were to begin creating a class and methods for the Prim's algorithm feature, having a separate class that contains all the methods for creating a graph, colouring in the edges and moving around the vertices would allow them to just implement the logic for Prim's algorithm and have the same generic graph methods that already exist, as opposed to having custom methods for the different graph algorithms. This makes development of further features easier through reusable components that haven't been made too bespoke to a specific area of the program.

The two main classes for Kruskal's algorithm are `Kruskals` and `Edge`. These are clearly named so as to indicate that the '`Kruskals`' class contains the main methods and logic for performing the algorithm and aggregates the '`Edge`' class, forming a basic object to be used within the main logic. The methods within these classes are also named specifically for their function e.g. `addEdge()` to add an edge to the graph, or `kruskalMST()` to create a minimum spanning tree using Kruskal's algorithm. Having clearly named methods ensures that a developer can roughly understand the function of a specific method by a glance at the name and would know which method to implement any further code/logic into.

This system contains a large number of attributes, all of which serve a specific purpose and have been named accordingly both for ease of development for me between the iterations, as well as maintaining the system for future development by new programmers. The `menuGUI` class contains the most attributes, predominantly GUI components like text areas and buttons. I followed a specific naming strategy for these attributes to ensure that while some of them were similar in function, they were clearly distinguishable. For example, the labels for

Aditya Verma (1197)                    528

the text areas for the Simplex algorithm inequality inputs were labelled in the following format [variable name][Label], giving us xLabel, yLabel, zLabel etc. to represent the JLabels for the variables x, y and z. A similar convention was followed when naming the labels for the Simplex objective function, this time replacing 'Label' with 'Obj', giving us xObj, yObj, zObj etc. to distinguish not only the variable name, but also that these labels were for the objective function specifically and not generic for the inequalities. All attributes which have multiple components followed this method of labelling, ensuring that they weren't too long and specific, but also were clearly distinguishable, the latter being the main priority.

Other attributes that weren't repeats were named according to their specific purpose, and were longer than the repetitive components like 'xObj' and 'xLabel'. A key example of this is in the DisplayPanel class, which contains attributes such as nodePositions, nodeLabels, dragStart and dragIndex to name a few. The first two are arraylists, with the identifiers determining what is contained within them. This would help a new developer picking up from the current stage of development as they can clearly see which list contains the different labels of each node in the graph, and which list contains the positions of each node in the graph. This would be useful for the Prim's algorithm feature, which would likely use many of the existing attributes for creating graphs. The dragStart and dragIndex variables are used to provide information about a specific node being moved around by the user. I chose to name them 'dragStart' and 'dragIndex' as opposed to just 'start' and 'index' to ensure clarity that these variables relate to the movement of the nodes and to avoid them from being confused by a new developer. For example, having the 'dragStart' variable named as just 'start' leaves its purpose ambiguous, as a developer wouldn't know what it represents the start of. However, having 'dragStart' helps indicate that it stores to location that the dragging motion/movement begins.

Finally, all arrays, arraylists and general lists are labelled in one of two ways: either using a plural term such as thetaValues, to indicate that multiple values are being stored, or in the case that there are more than one list representing similar values, the name is created as [value being stored][List]. For example, the Tableau class has an arraylist called rowOps, which uses the plural terminology to indicate that the row operations for a specific tableau are being stored. There is another

arraylist, which is two-dimensional, labelled as rowOperationList, following the second convention, as this stores the rowOps list as individual items, so each element in this arraylist is an entire arraylist of row operations. The purpose of this is to store the row operations for all the iterations, and since the values being stored are very similar, a clear distinction between names was required. Seeing the data type and the name of the attribute will allow for a new programmer to distinguish between different attributes with similar purposes to avoid errors being made in future development.

Next I will consider the use of comments in the code as a means of increasing maintainability of the system. The comments included in the code were designed to be short enough that a developer wouldn't have to spend a lot of time reading through each of them defining the function of a specific line, but also informative enough that the developer would know what the line does. An example of commented code that is representative of the system is found on page 265 in the development of Iteration 5, but has been pasted below for ease of access:

```
public class Display {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Circle");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);
        JPanel drawPanel = new JPanel() { // create and add a panel for drawing the circle
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);

                g.setColor(Color.BLUE);
                int diameter = 50; //to make a perfect circle, width and height must be equal
                int x1 = 100; // center the circle horizontally
                int y1 = 150; // center the circle vertically
                g.fillOval(x1, y1, diameter, diameter); // draw the circle

                int x2 = 250; //setting location of second circle
                int y2 = 150;
                g.fillOval(x2, y2, diameter, diameter);
                g.setColor(Color.BLUE);

                int centerX1 = x1 + diameter/2; //to draw a line we need the center of both circles
                int centerY1 = y1 + diameter/2;
                int centerX2 = x2 + diameter/2;
                int centerY2 = y2 + diameter/2;
                g.drawLine(centerX1, centerY1, centerX2, centerY2); //connecting the center of two circles with a line
            }
        };
    }
}
```

As shown, comments are added every few lines, either relating to the exact purpose of the line, or to why the line is necessary for a later point in the program.

For example, the comment '//create and add a panel for drawing the circle' indicates the purpose of the line, whereas a comment such as '//center the circle horizontally' defines the reason that the variable 'x1' is set to 100, but while the line of code itself may not perform the whole action, it is a prerequisite for later on, as shown in the later line with the comment '//to draw a line we need the center of both circles', in which x1 is used. Comments were also a useful indicator as to whether a specific line was doing too much in one go, specifically for mathematical/logic-based lines of code.

Another example of comment use is to define the purpose of different attributes in the classes, as evident on page 271, also pasted below:

```
public ArrayList<String> edges; //list of edges in the graph
10 usages
public ArrayList<Character> nodeLabels = new ArrayList<>(); //list of labels in the graph
6 usages
public ArrayList<Point> nodePositions = new ArrayList<>(); //list of locations on the window for the vertices
10 usages
public int diameter = 50; //diameter of the vertices
5 usages
private JFrame circleFrame; //frame to add graph to
```

The comments used here are particularly useful for the JFrame circleFrame component, as this contains a variety of other components, so having not only a clear name, but a short definition of its purpose will help future development in knowing which JFrame component to add a specific feature to.

Comments were also used within methods where new variables are created, to help clarify the type of variable it is, as well as what purpose it will serve in the program. Below is an example from page 240, with the comment '//creates a new 2d array to store the deep copy of the table'. This clearly indicates the type of variable and what purpose it serves, combined with the distinct variable name copyTable to reinforce the specific function/purpose of the variable.

```

public static double[][][] makeTableau(Tableau copyTableau) {
    double[][][] copyTable = new double[rows][cols]; //creates a new 2d array to store the deep copy of the table
    double[][][] existingTable = copyTableau.getTable(); //assigns the parameter to a 2d array
    for (int i = 0; i < existingTable.length; i++) {
        for (int j = 0; j < existingTable[i].length; j++) {
            copyTable[i][j] = existingTable[i][j]; //copies the array
        }
    }
    return copyTable;
}

```

Another convention I followed to help maintainability of the code was the single-page rule, which suggests that methods should generally not be more than a page in length, otherwise they should be split up further. This is generally the case within my code, as evidenced below with a few examples from pages 236, 240, 248 and 249

```

public void runSimplexAlg(int numInequality, int numVAR){
    double[][] constraints = new double[numInequality][numVAR + 1]; //2d array of constraints
    for(int i = 0; i < numInequality; i++){ //iterating through each set of inequalities
        JPanel parsePanel = (JPanel)inequalityPanel.getComponent(i); //creates a new panel object to extract components from
        int columnIndex = 0; //introducing the columnIndex counter
        for(int j = 0; j < parsePanel.getComponentCount(); j++){ //iterating through the panel
            Component component = parsePanel.getComponent(j);
            if(component instanceof JTextArea){ //checks if the current component is a text area
                if(columnIndex < constraints[i].length) {
                    constraints[i][columnIndex] = Double.parseDouble(((JTextArea) parsePanel.getComponent(j)).getText()); //adds the value in the text area to the array
                    columnIndex++;
                }
            }
        }
    }
}

public static JPanel displayTableau(int rows, int cols){
    JPanel tableauPanel = new JPanel(new GridLayout(rows, cols));
    for(double[] row: table){
        for(double value: row){
            JLabel addLabel = new JLabel(Double.toString( d: Math.round(value * 100000000) / 100000000));
            Border blackline = BorderFactory.createLineBorder(Color.black);
            addLabel.setBorder(blackline);
            addLabel.setFont(new Font( name: "Arial", Font.PLAIN, size: 25));
            tableauPanel.add(addLabel);
        }
    }
    JPanel wrapperPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
    wrapperPanel.add(tableauPanel);
    return wrapperPanel;
}

```

```

public static ArrayList<Double> getThetaValues(double[][] tableau) {
    thetaValues.clear();
    int pivotC = getCurrentPivotCol(tableau); //get the pivot column index

    for (int i = 0; i < tableau.length; i++) {
        if (tableau[i][pivotC] > 0) { //ensure we are not dividing by zero
            double first = tableau[i][cols - 1];
            double second = tableau[i][pivotC];
            System.out.println(first);
            System.out.println(second);
            System.out.println(" ");
            double theta = tableau[i][cols - 1] / tableau[i][pivotC]; //the value column is at cols - 1
            thetaValues.add(theta); //add the calculated theta value to the list
        } else {
            thetaValues.add(Double.POSITIVE_INFINITY); //handles cases where division by zero would occur
        }
    }
    return thetaValues;
}

```

```

public static int getCurrentPivotRow(int pivotCol, double[][] tableau) {
    int pivotRow = 0;
    double minRatio = Double.MAX_VALUE; //sets value of new variable equal to the max value offered by java
    for (int i = 0; i < rows - 1; i++) {
        double ratio = tableau[i][cols - 1] / tableau[i][pivotCol]; //divides value column by pivot column
        if (ratio > 0 && ratio < minRatio) { // if result > 0 and < current minRatio then sets minRatio to current value
            minRatio = ratio;
            pivotRow = i; //sets index of pivot row
        }
    }
    pivotVal = tableau[pivotRow][pivotCol];
    return pivotRow;
}

```

These all show functions contained within roughly 15-20 lines of code, which indicates that the methods have been decomposed to an adequate level, such that a single function isn't overly complex with multiple pages of logic. This helps make it easier for new developers when reading over the code seeing the process split up into smaller functions. For example, having the Simplex algorithm split up into, getting the pivot row, getting the pivot column, performing the first row operation and storing it, and then performing the row operations on each other row in the tableau, each developed as their own separate methods is much easier to follow than having a single method called 'performSimplex' combining the function of all the steps above. This would also help a new developer with debugging, isolating a bug to a single method with no more than 15-20 lines of code which is much easier to spot than having a bug somewhere embedded in 50 lines of mathematical logic.

There are a few functions which break the single page rule, the reason being that GUI functionality often requires multiple lines to achieve a single outcome, hence the method `displayTableau` is one of the longest, as shown on page 246:

```

for (int i = 0; i < outputTable.length; i++) {
    JLabel rowHeaderLabel = new JLabel(rowHeaders[i], SwingConstants.CENTER); //adding the row headers in the first column of the table underneath the basic variable header
    rowHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER); //setting the font, size, border and alignment of label
    rowHeaderLabel.setFont(new Font("Arial", Font.BOLD, size: 15));
    rowHeaderLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    tableauPanel.add(rowHeaderLabel); //adding the row header label to the panel

    for (int j = 0; j < outputTable[i].length; j++) {
        if(outputTable[i][j] % 1 == 0){
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j])), SwingConstants.CENTER); //adding the value from the output table one by one to the tableau
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            valueLabel.setFont(new Font("Arial", Font.PLAIN, size: 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        } else {
            valueLabel = new JLabel(String.format("%.2f", (outputTable[i][j])), SwingConstants.CENTER); //adding the value from the output table one by one to the tableau
            valueLabel.setHorizontalTextPosition(SwingConstants.CENTER);
            valueLabel.setFont(new Font("Arial", Font.PLAIN, size: 15));
            valueLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            tableauPanel.add(valueLabel); //adds the value label to the tableau panel
        }
    }
}

```

This is an extract of the method, just demonstrating that even each label in the panel required 4-5 lines of code just to set various attributes like its alignment and font size. Hence these methods were justified to break the single-page rule.

The general convention followed was having any mathematical logic broken down into specific steps and decomposed into smaller methods, which each performed part of the method. Hence there is no single method which contains all the logic for an algorithm, and instead a method like kruskalMST() relies on calling various other methods such as findComponent() to perform the algorithm. This will aid future development of the program, allowing for new programmers to understand how logic has been currently implemented, following the separate methods. This combines the necessity of clear labelling of methods, to ensure they can be followed in a meaningful order to reach the desire outcome, and can be built upon if future features were to be implemented

## Future Improvements

There were a variety of features left incomplete during the development of the system, due to time constraints and priority being placed on ensuring that the features most demanded by my stakeholders were implemented. Having consulted the stakeholders regarding the missing features and their relative priority, I have summarised each of the missing success criteria and how I would go about implementing it in the future:

### Success Criterion 2 (Prim's algorithm on a graph):

This was the missing feature that was ranked as highest priority by my stakeholders to implement in the system, with Student 1 mentioning 'I think that Prim's algorithm would be the most important to continue developing' and Student 2 mentioning 'Prim's algorithm is often asked more often through a variety of topics e.g. the Travelling Salesman problem'. To go about implementing this algorithm, I would follow a similar process to the other graph algorithm, Kruskal's. This would involve having a similar method to `findComponent`, however restricting the search for edges to only those adjacent to the edges already included, as per the steps of Prim's algorithm. From iteration 1, the GUI is already created in a way such that there is a 'shell' for Prim's algorithm to be developed, and the main GUI features would make use of the `DisplayPanel` class. As mentioned previously, keeping this as a generic class was very valuable, allowing for more versatility with future work on Prim's, instead of just restricting the graph display features to Kruskal's algorithm.

Instead of having a table of added and rejected edges, I would allow for more room in the explanation to fully justify which edge is being inspected and why, while incorporating the same tab system to click through the edges and have them coloured in their relevant colour -green or red.

### Success Criterion 3 (Prim's algorithm on a distance matrix)

There were no methods implemented to perform an algorithm on a matrix, however, this development would likely take place following the development of the basic Prim's algorithm on a graph. This would most likely be achieved through the use of a two-dimensional array containing the weights of edges between nodes, and using specific indexing to limit searches to the relevant columns of the table.

To implement this into the GUI, I would create a new menu option within the Prim's Algorithm menu item label 'Prim's Algorithm Matrix', to distinguish it from the graph-based version. To display the matrix on the GUI, I would follow a similar procedure that I used to create the tableau on the GUI, using a grid layout and

iterating through the indices to add specific values. I would also add more colour into this feature, for example, illustrating which columns are valid to be inspecting for the next edge to add by highlighting them in a specific colour. Colour could also be used in this way to highlight the specific cells in the matrix of values that have been entered. The main difficulty of implementing this would be trying to put a strike-through in the row of the matrix, as this is the method we were taught, since as far as I am aware, this is not possible through the basic grid layout features. To maintain consistency, I would also use a tab feature, where each tab represents a new node being considered as the entire matrix is inspected step-by-step to reach the final outcome. As this is no graph element, I could have a separate explanation feature to go into more detail regarding the specific inspection of the column and which columns to not be looking through at the specific stage in the algorithm. I would also take into account stakeholder feedback, making certain parts of the explanation bold, or in different colours to emphasise specific sections and to make it easier to read.

Success Criteria 4 and 9 (Implementing a Learn algorithm feature and and Introduction to Algorithm feature):

These were defined as lower priority success criteria by my stakeholders, who mentioned they are already familiar with the surrounding knowledge and that that difficulty is in 'performing the algorithm'. If I were to implement these features, I would maintain consistency by using the same styles of components that already exist, such as the Simplex tableau and the table of added and rejected edges. This would contribute to the overall familiarity of the system and would allow for better cross-referencing as a user moves from the learn feature to the run algorithm feature.

These two criteria are predominantly focused on the GUI, so their development would mainly require the use of JLabels and JTextAreas, which would be formatted using different types of panel. One type of panel that would be useful for a learn feature/introduction feature would be a gridBagLayout, as part of the Java JPanel styles, which follows a similar idea to the gridLayout, however, it allows for customised sizing of the cells. This would be implemented into the system e.g.

having more room for a Simplex tableau, and slightly less room to show the row operations.

This feature would be a lot lighter to implement than the more mathematical and logic-based algorithm work that the existing development focused on, but would take concepts learnt from the GUI development to make the system well-rounded.

Success Criteria 7 and 11 (Using past exam questions in the system as fixed examples)

Having examples is a useful feature for students, as it is something they can easily reference and work through multiple times to fully understand a topic. Since the analysis identified that past exam questions were one of the most valuable forms of revision, these would form the main examples in my system.

This would involve having screenshots of questions and mark schemes and using JLabels and JTextAreas to explain them. This is another GUI based improvement, but I would incorporate the algorithm work that has already been implemented to help the development of this feature e.g. as mentioned previously, providing the system's worked solutions to an exam question could take place by using the existing solving features, but providing fixed values to them to run when the user selects the fixed examples tool. Again, the GUI has been developed in a way that would make future development of this easier, with existing menu items and buttons labelled in regard to having learning features with examples.

Success Criteria 5 (Random graph generator)

As mentioned previously, to generate random graphs, I would use the existing DisplayPanel class, and use the Util library in Java, with the random number generator. Firstly, I would generate a random number of nodes in the graph, followed by random weights to represent the edges. Since the existing methods for this feature are all existent in the graph, creating a method to call the correct methods is all that would be required to implement this feature. The next part of this feature would be running the chosen algorithm on it. Currently, only Kruskal's algorithm has been developed, so after developing Prim's algorithm (the top priority), this would follow in second.

Overall, while the system was successful in implementing two of the three key algorithms I aimed at implementing, it was generally agreed that a variety of improvements could be made. Most of these were small features like prompts and colour choices, which came down to the time constraints on the development of the project and the wide scope of each feature to be developed. Having evaluated the existing features for their effectiveness in serving their purpose, as well as the missing features in how they would be implemented, this project has now been fully evaluated and tested.

## **Appendix**

Questions Used From:

<https://www.physicsandmathstutor.com/a-level-maths-papers/d1-edexcel/>

Mark Schemes Used From:

<https://www.physicsandmathstutor.com/a-level-maths-papers/d1-edexcel/>

Textbook For A-Level Decision Maths

<https://www.pearsonactivelearn.com/app/login?redirect=/app/library>

Java Documentation For Graphics

<https://docs.oracle.com/javase/tutorial/2d/geometry/index.html>

## General Java Documentation

<https://docs.oracle.com/en/java/>

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.border.EmptyBorder;
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.lang.reflect.Array;
import java.util.*;

public class menuGUI extends JFrame{
    // the following public attributes are declared with meaningful names to be used
    // throughout the constructor and other methods
    JMenuBar menubar;
    JMenu file, simplex, kruskals, prims;
    JMenuItem exit, home, runSimplex, runPrims, runKruskals, learnSimplex, learnPrims,
    learnKruskals;
    JLabel prompt, numV, sourceLabel, destinationLabel, weightLabel, xLabel, yLabel,
    zLabel, x1Label, x2Label, x3Label, x4Label, valueLabel, numberInequality, reminder,
    objLabel, x0bj, y0bj, z0bj, x10bj, x20bj, x30bj, x40bj, plus0bj, errorMessageLabel,
    kruskalsError;
    JButton introSimplex, introPrims, introKruskals, addEdge, removeEdge, runAlgorithm,
    addButton, twoVar, threeVar, fourVar, runSimplexButton, removeEdgeButton, resetGraph;
    JPanel promptPanel, buttonPanel, graphButtonPanel, introPanel, mainPanel, kruskalAdd,
    simPanel, simButtonPanel, inequalityPanel, simplexPanel, tableauPanel, tableauToAdd,
    combPanel, fillPanel;
    JTextArea sourceTA, destTA, weightTA, vertices, xTA, yTA, zTA, x1TA, x2TA, x3TA,
    x4TA, valueTA, x0bjTA, y0bjTA, z0bjTA, x10bjTA, x20bjTA, x30bjTA, x40bjTA;
    JComboBox numInequality;
    double[][] constraints;
    double[] obj;

    int numVAR, counter;
    public static String[] rowHeaders, colHeaders;
    ArrayList<Integer> toAdd = new ArrayList<>();
    DisplayPanel displayPanel = new DisplayPanel();
    static int selectedTab;
    static ArrayList<Edge> mstEdges;

    public menuGUI(){
        setLayout(new BorderLayout()); //sets the window layout as a border layout
        //creating the menu bar
```

```

menubar = new JMenuBar();
setJMenuBar(menubar);

errorMessageLabel = new JLabel(""); // initialise the error message label
errorMessageLabel.setForeground(Color.RED); //set the error message colour to red
numVAR = 0; //initialise the number of variables to 0 for input validation
kruskalsError = new JLabel("");
kruskalsError.setForeground(Color.RED);

kruskalAdd = new JPanel(new FlowLayout()); //new panel underneath the button panel
combPanel = new JPanel(new GridLayout(0,2)); //creating a new panel with 2 columns to
make 2 halves which store the graph and the table
resetGraph = new JButton("Reset");

//adding 'file' menu option to the menu bar
file = new JMenu("File");
menubar.add(file);
home = new JMenuItem("Home");
file.add(home);
exit = new JMenuItem("Exit");
file.add(exit);

//creating menu options for each algorithm
simplex = new JMenu("Simplex Algorithm");
menubar.add(simplex);

kruskals = new JMenu("Kruskal's Algorithm");
menubar.add(kruskals);

prims = new JMenu("Prim's Algorithm");
menubar.add(prims);

//creating and adding the sub-options for each algorithm to the menubar
runSimplex = new JMenuItem("Run Algorithm");
runPrims = new JMenuItem("Run Algorithm");
runKruskals = new JMenuItem("Run Algorithm");

simplex.add(runSimplex);
kruskals.add(runKruskals);
prims.add(runPrims);

learnSimplex = new JMenuItem("Learn Algorithm");
learnKruskals = new JMenuItem("Learn Algorithm");

```

```

learnPrims = new JMenuItem("Learn Algorithm");

simplex.add(learnSimplex);
kruskals.add(learnKruskals);
prims.add(learnPrims);

addEdge = new JButton("Add Edge");
removeEdge = new JButton("Remove Edge");
runAlgorithm = new JButton("Run Algorithm");

twoVar = new JButton("2 Variables");
threeVar = new JButton("3 Variables");
fourVar = new JButton("4 Variables");

runSimplexButton = new JButton("Run");

 addButton = new JButton("Add");

removeEdgeButton = new JButton("Remove");

originalWindow(); //calls the originalWindow method to add the buttons and label to
the window
add(promptPanel, BorderLayout.NORTH);
add(buttonPanel, BorderLayout.CENTER);
//adding an event action listener to each menu item within each menu bar option
event e = new event();
runKruskals.addActionListener(e);
runSimplex.addActionListener(e);
learnSimplex.addActionListener(e);
learnPrims.addActionListener(e);
learnKruskals.addActionListener(e);
introSimplex.addActionListener(e);
introKruskals.addActionListener(e);
introPrims.addActionListener(e);
runPrims.addActionListener(e);

home.addActionListener(e);
exit.addActionListener(e);
addEdge.addActionListener(e);
removeEdge.addActionListener(e);
addButton.addActionListener(e);
runAlgorithm.addActionListener(e);
twoVar.addActionListener(e);
threeVar.addActionListener(e);
fourVar.addActionListener(e);
runSimplexButton.addActionListener(e);

```

```

removeEdgeButton.addActionListener(e);
resetGraph.addActionListener(e);

}

private void originalWindow(){ // procedure called in the constructor to generate initial
panels
    promptPanel = new JPanel(new BorderLayout());
    prompt = new JLabel("Select an option from the menu", SwingConstants.CENTER);
    promptPanel.add(prompt, BorderLayout.CENTER);
    promptPanel.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0)); //defining the
border size for the panel in the north section

    buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(1, 3, 20, 50)); //splitting the center panel
into 3 sections for 3 buttons
    buttonPanel.setBorder(BorderFactory.createEmptyBorder(5, 20, 60, 20)); //defining the
border size for the button panel in the center section

    //creating the button for kruskal's algorithm introduction, with image, label and
text positioning defined
    ImageIcon kruskalsIcon = new ImageIcon("C:\\Users\\averm\\Pictures\\
kruskalsimage.png");
    introKruskals = new JButton();
    introKruskals.setText("Kruskal's Algorithm Introduction");
    introKruskals.setHorizontalTextPosition(JButton.CENTER);
    introKruskals.setVerticalTextPosition(JButton.TOP);
    introKruskals.setFocusable(false);
    introKruskals.setIcon(kruskalsIcon);
    buttonPanel.add(introKruskals);

    //creating the button for the simplex algorithm introduction, with image, label and
text positioning defined
    ImageIcon simplexIcon = new ImageIcon("C:\\Users\\averm\\Pictures\\simplex.png");
    introSimplex = new JButton();
    introSimplex.setText("Simplex Algorithm Introduction");
    introSimplex.setHorizontalTextPosition(JButton.CENTER);
    introSimplex.setVerticalTextPosition(JButton.TOP);
    introSimplex.setFocusable(false);
    introSimplex.setIcon(simplexIcon);
    buttonPanel.add(introSimplex);
}

```

```

    //creating the button for prim's algorithm introduction, with image, label and text
    //positioning defined
    ImageIcon primsIcon = new ImageIcon("C:\\Users\\averm\\Pictures\\kruskalsimage.png");
    introPrims = new JButton();
    introPrims.setText("Prim's Algorithm Introduction");
    introPrims.setHorizontalTextPosition(JButton.CENTER);
    introPrims.setVerticalTextPosition(JButton.TOP);
    introPrims.setFocusable(false);
    introPrims.setIcon(primsIcon);
    buttonPanel.add(introPrims);
}
public void restoreHome(){ //procedure which adds the initial components back to the
window returning to the original state
    getContentPane().removeAll();
    add(promptPanel, BorderLayout.NORTH);
    add(buttonPanel, BorderLayout.CENTER);
    revalidate(); // ensuring all layout changes have been validated
    repaint(); //updating the window to display the changes to the window
}
public void graphSettings() {
    getContentPane().removeAll(); //removes all current components from the window
    promptPanel.removeAll();
    mainPanel = new JPanel(new BorderLayout()); //creates a new panel with a border
layout
    graphButtonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10)); // a new
panel for the three buttons
    graphButtonPanel.add(resetGraph); //adding the row of buttons with the reset graph
button first
    graphButtonPanel.add(addEdge);
    graphButtonPanel.add(removeEdge);
    graphButtonPanel.add(runAlgorithm);
    graphButtonPanel.add(kruskalsError);

    mainPanel.add(graphButtonPanel, BorderLayout.NORTH);

    JSeparator separator = new JSeparator(SwingConstants.HORIZONTAL); //this creates a
separator line underneath the buttons
    mainPanel.add(separator, BorderLayout.CENTER);

    add(mainPanel, BorderLayout.NORTH);
    revalidate();
    repaint();
}

public void kruskalsIntro() {
    getContentPane().removeAll();
    promptPanel.removeAll();

```

```

        introPanel = new JPanel(new BorderLayout ());
        JLabel kruskalsTitle = new JLabel("Kruskal's Algorithm", SwingConstants.CENTER);
        kruskalsTitle.setFont(new Font("Arial", Font.BOLD, 36));
        kruskalsTitle.setBorder(new EmptyBorder(20, 0, 10, 0));
        introPanel.add(kruskalsTitle, BorderLayout.NORTH);
        add(introPanel, BorderLayout.CENTER);
        revalidate(); //refreshing the GUI
        repaint();
    }
    public void resetGraphDisplay(){
        displayPanel.getEdges().clear(); //clear the arraylists storing information about the
edges
        displayPanel.getEdgeWeights().clear();
        displayPanel.getEdgeColourList().clear();
        displayPanel.getNodeLabels().clear(); // clear the nodes and their positions in the
DisplayPanel
        displayPanel.getNodePositions().clear();

        toAdd.clear(); // clear the edges stored in `toAdd`

        displayPanel.revalidate(); // refresh the GUI to remove the displayed graph
displayPanel.repaint();

        kruskalsError.setText(""); // clear any error messages
    }
}

```

```

public void edge(){
    kruskalAdd.removeAll();
    combPanel.removeAll();

    sourceTA = new JTextArea(2, 5); //adding the text areas to get user input for source,
destination and weight
    destTA = new JTextArea(2, 5);
    weightTA = new JTextArea(2, 5);
    sourceLabel = new JLabel("Source:"); //adding labels to prompt the user for which
input
    destinationLabel = new JLabel("Destination:");
    weightLabel = new JLabel("Weight:");
    vertices = new JTextArea(2, 5); // adds the text area for user to enter number of
vertices
    numV = new JLabel("Number of Vertices:");
}

```

```

kruskalAdd.add(sourceLabel); //the following section of code adds each of these
labels and text areas to the panel
kruskalAdd.add(sourceTA);
kruskalAdd.add(Box.createVerticalStrut(10)); // 10 pixels of vertical spacing
kruskalAdd.add(destinationLabel);
kruskalAdd.add(destTA);
kruskalAdd.add(Box.createVerticalStrut(10)); // 10 pixels of vertical spacing
kruskalAdd.add(weightLabel);
kruskalAdd.add(weightTA);
kruskalAdd.add(Box.createHorizontalStrut(20)); //adding space between the weight text
area and the add button
kruskalAdd.add(addButton);
kruskalAdd.add(numV);
kruskalAdd.add(vertices);

mainPanel.add(kruskalAdd, BorderLayout.CENTER); //adding the new panel to the main
panel
displayPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //setting a
black border around displayPanel
fillPanel = new JPanel(new GridLayout(2,1));
fillPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
//combPanel = new JPanel(new GridLayout(0,2)); //creating a new panel with 2 columns
to make 2 halves which store the graph and the table
combPanel.add(displayPanel); //add the graph to the left half of the combination
panel
combPanel.add(fillPanel); //add an empty panel to the right half of the combination
panel
combPanel.setBorder(new EmptyBorder(20, 20, 20, 20)); //create a border around the
combination panel
add(combPanel, BorderLayout.CENTER); //add the combination panel to the main panel
revalidate(); //refresh the GUI
repaint();
}

public void addToGraph() {
try {
String source = sourceTA.getText(); //extracting the inputted text from the text
areas
if (!source.matches("[A-Z]")) { //checking source node is a single capital letter
throw new IllegalArgumentException("Please enter source node as a single
capital letter."); //error message to indicate mistake to user
}
kruskalsError.setText(""); //error message set to empty string if input is valid
char sourceCH = source.charAt(0); //converting the string inputs to characters

String dest = destTA.getText();

```

```

        if (!dest.matches("[A-Z]")) { //checking destination node is a singel capital
letter
            throw new IllegalArgumentException("Please enter destination node as a single
capital letter."); //error message to indicate mistake to user
        }
        kruskalsError.setText(""); //error message set to empty string if input is valid
        char destCH = dest.charAt(0);

        String weight = weightTA.getText();
        int weightInt;
        try {
            weightInt = Integer.parseInt(weight);
            kruskalsError.setText(""); //error message is set to empty string if valid input
entered
        } catch (NumberFormatException e) { //throws exception if input is not a number
            throw new IllegalArgumentException("Weight must be a valid number."); //error
message text is changed to indicate mistake to user
        }

        String pair = source + dest; //formats the source and destination as one string
        String reversePair = dest + source; //making the reverse of the edge e.g. 'AB'
becomes 'BA'
        if (displayPanel.edges.contains(pair) || displayPanel.edges.contains(reversePair)) {
//checking if the user has already entered the inputted edge into the graph
            throw new IllegalArgumentException("This edge already exists in the graph.");
//setting the error message text saying that they have already entered the edge
        }
        toAdd.add((int) sourceCH); //adds the source, destination and weight to an arraylist
of information about edges to add
        toAdd.add((int) destCH);
        toAdd.add(weightInt);
        displayPanel.addEdge(pair, weightInt); //adds the edge to the panel
        revalidate(); //refreshing the GUI
        repaint();
    }catch(IllegalArgumentException e){
        kruskalsError.setText(e.getMessage());
    }
    revalidate();
    repaint();
}

public void removeFromGraph() {
    kruskalAdd.removeAll();
    JTextArea removeSourceTA = new JTextArea(2,5); //adding in source text area for edge
to be removed
    JTextArea removeDestTA = new JTextArea(2,5); //adding in destination text area for
edge to be removed
    JLabel removeSourceLabel = new JLabel("Source: "); //label to indicate purpose of
}

```

```

removeSourceTA
    JLabel removeDestLabel = new JLabel("Destination: "); //label to indicate purpose of
removeDestTA

    kruskalAdd.add(removeSourceLabel); //adding the various components to the kruskalAdd
panel
    kruskalAdd.add(removeSourceTA);
    kruskalAdd.add(Box.createVerticalStrut(10)); // 10 pixels of vertical spacing
    kruskalAdd.add(removeDestLabel);
    kruskalAdd.add(removeDestTA);
    kruskalAdd.add(removeEdgeButton); //add the remove edge button
    revalidate(); //refresh the GUI
    repaint();

}

public void remove() {
    try {
        String source = sourceTA.getText().trim(); // get the source node from the text
area
        if (!source.matches("[A-Z]")) { //checking that input is single capital letter
            throw new IllegalArgumentException("Please enter a source node as a single
capital letter.");
        }

        String dest = destTA.getText().trim(); // get the destination node from the text
area
        if (!dest.matches("[A-Z]")) { //checking that input is single capital letter
            throw new IllegalArgumentException("Please enter a destination node as a
single capital letter.");
        }

        String pair = source + dest; // construct the edge string
        if (!displayPanel.edges.contains(pair)) { //checking if inputted edge is in the
graph
            throw new IllegalArgumentException("The specified edge does not exist in the
graph.");
        }

        int index = displayPanel.edges.indexOf(pair); // get the edge index
        displayPanel.edges.remove(index); // remove the edge
        displayPanel.edgeWeights.remove(index); // remove the corresponding weight
        displayPanel.edgeColourList.remove(index); // remove the color

        displayPanel.revalidate(); //update the graph
        displayPanel.repaint();
    }
}

```

```

char sourceChar = source.charAt(0); //remove the edge from the `toAdd` list
char destChar = dest.charAt(0);

for (int i = 0; i < toAdd.size(); i += 3) {
    if ((toAdd.get(i) == (int) sourceChar && toAdd.get(i + 1) == (int) destChar)
    || (toAdd.get(i) == (int) destChar && toAdd.get(i + 1) == (int) sourceChar)) {
        toAdd.remove(i + 2); //remove weight
        toAdd.remove(i + 1); //remove destination
        toAdd.remove(i); //remove source
        break; //exit loop after removing the edge
    }
}
boolean isValidSourceNode = false;
for(int i = 0; i < toAdd.size(); i ++){ //checking that the source node entered
exists in the graph
    if(toAdd.get(i) == sourceChar){ //if the current node is the source node then
it sets the boolean to true
        isValidSourceNode = true;
    }
}
if (!isValidSourceNode) {
    int nodeIndex = displayPanel.getNodeLabels().indexOf(sourceChar); // find the
index of the source node in the nodeLabels list

    if (nodeIndex != -1) {
        displayPanel.getNodeLabels().remove(nodeIndex); //remove the node label
and its corresponding position
        displayPanel.getNodePositions().remove(nodeIndex);

        displayPanel.revalidate(); // refresh the GUI
        displayPanel.repaint();
    }
}
boolean isValidDestNode = false;
for(int i = 0; i < toAdd.size(); i ++){ //checking that the destination node
entered exists in the graph
    if(toAdd.get(i) == destChar){ //if the current node is the destination node
then it sets the boolean to true
        isValidDestNode = true;
    }
}
if (!isValidDestNode) {
    int nodeIndex = displayPanel.getNodeLabels().indexOf(destChar); //find the
index of the destination node in the nodeLabels list

    if (nodeIndex != -1) {
        displayPanel.getNodeLabels().remove(nodeIndex); // remove the node label

```

```

and its corresponding position
    displayPanel.getNodePositions().remove(nodeIndex);

        displayPanel.revalidate(); // refresh the GUI to reflect the changes
        displayPanel.repaint();
    }
}

kruskalsError.setText(""); // clear any previous error messages
} catch (IllegalArgumentException e) {
    kruskalsError.setText(e.getMessage()); // display an error message to the user
}
}

public JPanel createMSTPanel(ArrayList<Edge> mstEdges, ArrayList<Edge> allEdges) {
    JPanel mstPanel = new JPanel(new GridLayout(0, 2)); // Create a panel with 2 columns

    JLabel edgeLabel = new JLabel("Edge"); //adding label to left column
    JLabel actionLabel = new JLabel("Add/Reject"); //adding label to right column
    edgeLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font, size and style
    of text in label
    edgeLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //adding border
    around label
    actionLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font, size and style
    of text in label
    actionLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //adding border
    around label
    mstPanel.add(edgeLabel); //add the two labels to the panel
    mstPanel.add(actionLabel);

    //sort all edges by weight
    Collections.sort(allEdges);

    //iterate through all edges and check if they are in the MST
    for (Edge edge : allEdges) {
        JLabel edgeDisplay = new JLabel((char)(edge.source + 65)+ "" + (char)(edge.dest +
65)); // label for each edge
        String action = "Reject";
        if(mstEdges.contains(edge)){
            action = "Add";
        }
        JLabel actionDisplay = new JLabel(action); // display add or reject
        edgeDisplay.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font, size and
        style of text in label
    }
}

```

```

        edgeDisplay.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //adding
border around label
        actionDisplay.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font, size and
style of text in label
        actionDisplay.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //adding
border around label
        mstPanel.add(edgeDisplay); //add the labels to the panel
        mstPanel.add(actionDisplay);
    }

    return mstPanel; // return the constructed panel
}

```

```

public void runKruskalsAlg(){
    String graphV = vertices.getText(); //extracts the text for the number of vertices
from the text area
    char numVertices = graphV.charAt(0); //converts the string input to a character

    int graphSize = numVertices; //converts the character to an integer to be used as a
parameter
    Kruskals graph = new Kruskals(graphSize); //creating a new object of type Kruskals
    for(int i = 0; i + 2 < toAdd.size(); i+=3){ //incrementing in 3 to ensure all 3
pieces of information for a single node are added correctly
        graph.addEdge(toAdd.get(i), toAdd.get(i+1), toAdd.get(i+2)); //adding the source
destination and weight as an edge
    }
    if (!isConnected(graph, graphSize)) { //checking if the graph is connected or not
        kruskalsError.setText("Please ensure the graph is connected."); //setting the
error message text to indicate to the user
        return; // exit the method if the graph is not connected
    }
    ArrayList<Edge> edgeList = graph.edges; //list of edges in the graph
    mstEdges = graph.kruskalMST(); // list of edges in the minimum spanning tree
    JPanel mstPanel = createMSTPanel(mstEdges, graph.edges); //panel containing the table
of added/rejected arcs
}

```

```

// create a container panel to hold both the graph and MST panels
combPanel.add(fillPanel); //add the empty fill panel to the right column of the
combination panel
JTabbedPane explainPane = new JTabbedPane(); //creating the tabbed pane to contain
the edges
 JPanel kruskalsPromptPanel = new JPanel(new BorderLayout()); //creating a new panel
to store the intro information for the tabbed pane
 JLabel kruskalsPrompt = new JLabel("Click through the edge tabs to see how the
minimum spanning tree builds up."); //prompting the user to click through the edges
 kruskalsPrompt.setFont(new Font("Arial", Font.BOLD, 15)); //setting font size and
style for text
 JLabel colourLabel = new JLabel("Green = Add. Red = Reject"); //informing the user
about the use of colours in the highlighted edges
 colourLabel.setFont(new Font("Arial", Font.BOLD, 15)); //setting font size and style
for text
 kruskalsPromptPanel.add(kruskalsPrompt, BorderLayout.NORTH); //adding prompt to panel
 kruskalsPromptPanel.add(colourLabel, BorderLayout.CENTER); //adding colour
information to panel
 explainPane.add(kruskalsPromptPanel, "Intro"); //adding the panel to the tabbed pane
with the title 'Intro'
 Collections.sort(edgeList); //sorting the list of edges, so they can be added to the
tabbed pane in sorted order
 for(Edge edge: edgeList){
     JPanel edgePanel = new JPanel(); //creating a new tab for each edge in the graph
     JTextPane explanation = explainEdge(mstEdges, edgeList, edge); //adding the text
pane to contain the explanation of each edge being added/rejected
     edgePanel.add(explanation); //add the text pane to the panel for the tabbed pane
     String title = (char)(edge.source + 65) + "" + (char)(edge.dest+65); //make the
edge the title of the tab e.g. 'AB'
     explainPane.add(edgePanel, title); //add the panel to the tabbedpane with the
edge title
 }
 explainPane.addChangeListener(new ChangeListener() { //adding a change listener to
the tabbed pane to detect when the user selects a new tab
    @Override
    public void stateChanged(ChangeEvent e) {
        JTabbedPane sourcePane = (JTabbedPane) e.getSource(); //initialising a tabbed
pane as the item that was updated/changed
        selectedTab = sourcePane.getSelectedIndex() - 1; //getting the index of the
current selected tab

        for(String edge: displayPanel.getEdges()){ //iterating through list of edges
            displayPanel.highlightEdge(edge, Color.BLACK); //setting all edges to be
highlighted in black
        }
        for(int i = 0; i < selectedTab + 1; i++){ //iterating through the edges
            Edge edge = graph.edges.get(i); //setting an edge object to the current

```

```

edge
        if(mstEdges.contains(edge)){ //checking if the current edge is in the
minimum spanning tree
            String edgeString = (char)(edge.source + 65) + "" + (char)(edge.dest
+ 65); //setting the string of the edge to the source + destination e.g. 'AB'
            displayPanel.highlightEdge(edgeString, Color.GREEN); //sets the edge
colour to green
        } else{
            String edgeString = (char)(edge.source + 65) + "" + (char)(edge.dest
+ 65); //setting the string of the edge to the source + destination e.g. 'AB'
            displayPanel.highlightEdge(edgeString, Color.RED); //sets the edge
colour to red if the edge is not in the MST
        }
    }
}
fillPanel.add(mstPanel); //adds the add/reject table to the top half of the right
side of the panel
fillPanel.add(explainPane); //adds the tabbed pane to the bottom half of the right
side of the panel
combPanel.add(fillPanel); //adds the updated fillPanel to the overall combination
panel
    revalidate(); //refreshing the GUI
    repaint();
}
private boolean isConnected(Kruskals graph, int numVertices) {
    boolean[] visited = new boolean[numVertices];
    int startNode = Integer.MAX_VALUE; //assigns the largest value to start node
    for (Edge edge : graph.edges) {
        startNode = Math.min(startNode, Math.min(edge.source, edge.dest)); //getting the
first node in the graph to begin the dfs
    }
    dfs(startNode, visited, graph); //doing a depth first search starting from the first
node in the graph

    for (Edge edge : graph.edges) {
        if (!visited[edge.source] || !visited[edge.dest]) { //checking if the source and
node of each edge has been visited
            return false; //if not then the graph is not connected
        }
    }
}

return true;
}

private void dfs(int node, boolean[] visited, Kruskals graph){ //depth first search
method
    visited[node] = true;

```

```

        for(Edge edge: graph.edges){ //going through each edge in the graph
            if(edge.source == node && !visited[edge.dest]){ //checking whether the source has
been visited and the destination hasn't
                dfs(edge.dest, visited, graph); //recursively doing depth first search
starting from the destination
            } else if(edge.dest == node && !visited[edge.source]){ //checking whether the
destination has been visited and the source hasn't
                dfs(edge.source, visited, graph); //recursively doing depth first search
starting from the source
            }
        }
    }

public static JTextPane explainEdge(ArrayList<Edge> mstEdges, ArrayList<Edge> allEdges,
Edge edge) {
    JTextPane explanationPane = new JTextPane();
    int total = 0;
    for (Edge mstEdge : mstEdges) {
        total += mstEdge.weight; //summing the total weight of edges in the minimum
spanning tree
    }
    if (mstEdges.contains(edge)) { //checking if edge is in minimum spanning tree
        if (edge == allEdges.get(0)) { //the first edge in the minimum spanning tree
            explanationPane.setText("The edges are first sorted into ascending weight
order. This is the edge with the lowest weight.");
            explanationPane.setText(explanationPane.getText() + "\nNo cycles are formed
by adding this edge to the minimum spanning tree, hence we add it.");
            explanationPane.setText(explanationPane.getText() + "\nOn the table of added
and rejected edges, this edge is listed as 'Add'.");
            explanationPane.setText(explanationPane.getText() + "\n"); //adding some
spacing lines to make text more readable
            explanationPane.setText(explanationPane.getText() + "\n");
            explanationPane.setText(explanationPane.getText() + "Click the next tab to
see the next edge!");
        } else if (edge == mstEdges.get(mstEdges.size() - 1)) { //the final edge in the
minimum spanning tree
            explanationPane.setText("Adding this edge doesn't form any cycles. Looking at
the minimum spanning tree,");
            explanationPane.setText(explanationPane.getText() + "\nall the nodes are
connected by edges, so our minimum spanning tree is complete.");
            explanationPane.setText(explanationPane.getText() + "\n"); //adding some
spacing lines to make text more readable
            explanationPane.setText(explanationPane.getText() + "\n");
            explanationPane.setText(explanationPane.getText() + "Click the next tab to
see the next edge!");
        }else if(edge == allEdges.get(allEdges.size()-1)){ //the final edge in the entire
graph
            explanationPane.setText("This is the final edge added to the minimum spanning
tree. Looking at the minimum spanning tree,");
            explanationPane.setText(explanationPane.getText() + "\nall the nodes in the
graph are connected by edges, so our minimum spanning tree is complete.");
        }
    }
}

```

```

        explanationPane.setText(explanationPane.getText() + "\nIn the table of edges,
this edge is listed as 'Add'");
        explanationPane.setText(explanationPane.getText() + "\n");//adding some
spacing lines to make text more readable
        explanationPane.setText(explanationPane.getText() + "\n");
        explanationPane.setText(explanationPane.getText() + "The final weight of the
minimum spanning tree is: " + total);
        explanationPane.setText(explanationPane.getText() + "\nEdges included in the
minimum spanning tree:");
        for(int i = 0; i < mstEdges.size(); i++){ //outputting the list of edges
added to the minimum spanning tree
            explanationPane.setText(explanationPane.getText() + "\n" + (char)
(mstEdges.get(i).source + 65) + "" + (char)(mstEdges.get(i).dest + 65));
        }
    }else { //other edges in the minimum spanning tree
        explanationPane.setText("We consider what happens to the minimum spanning
tree if we add this edge.");
        explanationPane.setText(explanationPane.getText() + "\nNo cycles are formed
by adding this edge, so we can add it.");
        explanationPane.setText(explanationPane.getText() + "\nLooking at the table
of edges, we see this edge is given 'Add'.");
        explanationPane.setText(explanationPane.getText() + "\n");//adding some
spacing lines to make text more readable
        explanationPane.setText(explanationPane.getText() + "\n");
        explanationPane.setText(explanationPane.getText() + "Click the next tab to
see the next edge!");
    }
}else if(edge == allEdges.get(allEdges.size()-1)){ //final edge in the graph that is
not in the minimum spanning tree
    explanationPane.setText("This is the final edge we consider but the minimum
spanning tree is complete, so we reject it.");
    explanationPane.setText(explanationPane.getText() + "\nLooking at the table of
edges, this edge is listed as 'Reject'.");
    explanationPane.setText(explanationPane.getText() + "\nThe total weight is: " +
total);
    explanationPane.setText(explanationPane.getText() + "\nEdges included in the
minimum spanning tree:");
    for(int i = 0; i < mstEdges.size(); i++){ //outputting the list of edges added to
the minimum spanning tree
        explanationPane.setText(explanationPane.getText() + "\n" + (char)
(mstEdges.get(i).source + 65) + "" + (char)(mstEdges.get(i).dest + 65));
    }
}else{ //other edges that are rejected from the minimum spanning tree
    explanationPane.setText("Adding this edge to the minimum spanning tree would form
a cycle, therefore we reject it.");
    explanationPane.setText(explanationPane.getText() + "\nOn the table of added and
rejected edges, we see that this edge is listed as 'Reject'.");
    explanationPane.setText(explanationPane.getText() + "\n");//adding some spacing
lines to make text more readable
    explanationPane.setText(explanationPane.getText() + "\n");
}

```

```

        explanationPane.setText(explanationPane.getText() + "Click the next tab to see
the next edge!");
    }
    explanationPane.setFont(new Font("Arial", Font.PLAIN, 15)); //setting the font size
and style for the explanation
    explanationPane.setEditable(false); //making it so the user cannot edit the text of
the text pane
    return explanationPane; //return the text pane
}
public static int getSelectedTab(){
    return selectedTab;
}
public static ArrayList<Edge> getMSTEdges(){
    return mstEdges;
}
public void simplexSettings() {
    String[] inequalityOptions = {"--", "1","2", "3", "4"}; //setting the options for the
drop-down menu
    numInequality = new JComboBox(inequalityOptions); //creating the drop-down menu and a
label for it
    numberInequality = new JLabel("Select number of inequalities");
    getContentPane().removeAll(); //removes all current components from the window
    promptPanel.removeAll();
    simPanel = new JPanel(new BorderLayout()); //creates a new panel with a border layout
    simButtonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10)); // a new
panel for the three buttons
    simButtonPanel.add(numberInequality); //adding the components to the top panel
    simButtonPanel.add(numInequality);
    simButtonPanel.add(twoVar);
    simButtonPanel.add(threeVar);
    simButtonPanel.add(fourVar);
    simButtonPanel.add(runSimplexButton);
    simButtonPanel.add(errorMessageLabel);

    simPanel.add(simButtonPanel, BorderLayout.NORTH); //adding this to the main panel

    JSeparator separator = new JSeparator(SwingConstants.HORIZONTAL); //this creates a
separator line underneath the buttons
    simPanel.add(separator, BorderLayout.CENTER);

    add(simPanel, BorderLayout.NORTH);
    revalidate(); //updating the GUI
    repaint();
}

```

```

public void setSimplex(JButton button){
    simplexPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 1)); //new panel which
goes below the buttons
    inequalityPanel = new JPanel(); //panel to contain various row panels, using a box
layout to stack them on top of each other
    inequalityPanel.setLayout(new BoxLayout(inequalityPanel, BoxLayout.Y_AXIS));

    counter = Integer.parseInt(numInequality.getSelectedItem().toString()); //initialises
a counter variable
    if (button == twoVar && (numInequality.getSelectedItem().toString() != "--")){
//checks which button pressed and ensures a value was selected from the drop-down
        numVAR = 2;
        rowHeaders = new String[counter + 1]; //creating row headers array
        for(int a = 0; a < counter; a++){
            rowHeaders[a] = Character.toString(a+114); //adding slack variable letters
starting from r, depending on number of inequalities (counter)
        }
        rowHeaders[counter] = "P"; //setting final index as P, this is always the row
header of the bottom row
        colHeaders = new String[counter + 3]; //creating list of column headers
        colHeaders[0] = "x"; //for 2 variables, the first 2 column headers are the 2
variables, x and y.
        colHeaders[1] = "y";
        for(int a = 0; a < counter; a++){ //adding column headers of slack variables in
a similar way to row headers using ASCII
            colHeaders[a+2] = Character.toString(a+114);
        }
        colHeaders[counter + 2] = "Value"; //the row header of the final column is always
Value
        for (int i = 0; i < counter; i++) { //for each row of inequalities
            JPanel rowPanel = new JPanel(); //new panel is created
            rowPanel.setLayout(new FlowLayout());
            rowPanel.setMaximumSize(new Dimension(650, 150)); //bring the rows closer
together
            JLabel plus1 = new JLabel("+"); //initialising all the components that need
to be added in each row
            plus1.setBorder(new EmptyBorder(0, 10, 0, 10)); //ensuring there is spacing
        }
    }
}

```

```

between the components

    xTA = new JTextArea(2, 5);
    yTA = new JTextArea(2, 5);
    xLabel = new JLabel("x");
    xLabel.setFont(new Font("Arial", Font.PLAIN, 15));
    yLabel = new JLabel("y");
    yLabel.setFont(new Font("Arial", Font.PLAIN, 15));
    xLabel.setBorder(new EmptyBorder(0, 0, 5, 0));
    yLabel.setBorder(new EmptyBorder(0, 0, 5, 0));
    valueLabel = new JLabel("=");
    valueLabel.setFont(new Font("Arial", Font.PLAIN, 15));
    valueLabel.setBorder(new EmptyBorder(0, 20, 0, 20));
    valueTA = new JTextArea(2, 5);

    xTA.setBorderStyle(new EmptyBorder(0, 0, 10, 0));
    yTA.setBorderStyle(new EmptyBorder(0, 0, 10, 0));

    rowPanel.add(xTA); //adding each of the components to the row panel
    rowPanel.add(xLabel);
    rowPanel.add(plus1);
    rowPanel.add(yTA);
    rowPanel.add(yLabel);
    rowPanel.add(valueLabel);
    rowPanel.add(valueTA);

    inequalityPanel.add(rowPanel); //adding the row of components to the
inequality panel
}

JPanel objectivePanel = new JPanel(); //creating a new objective panel which will
go at the bottom of the inequality panel
objectivePanel.setLayout(new FlowLayout());
objectivePanel.setMaximumSize(new Dimension(650, 150)); //once again limiting the
size it can be
objLabel = new JLabel("P ="); //defining all the components with adequate spacing
objLabel.setBorder(new EmptyBorder(0, 0, 0, 20));
xObjTA = new JTextArea(2, 5);
yObjTA = new JTextArea(2, 5);
xObj = new JLabel("x");
xObj.setFont(new Font("Arial", Font.PLAIN, 15));
yObj = new JLabel("y");
yObj.setFont(new Font("Arial", Font.PLAIN, 15));
xObj.setBorder(new EmptyBorder(0, 0, 5, 0));
yObj.setBorder(new EmptyBorder(0, 0, 5, 0));
plusObj = new JLabel("+");
plusObj.setBorder(new EmptyBorder(0, 10, 0, 10));

objectivePanel.add(objLabel); //adding the components to the objective panel
objectivePanel.add(xObjTA);

```

```

        objectivePanel.add(xObj);
        objectivePanel.add(plusObj);
        objectivePanel.add(yObjTA);
        objectivePanel.add(yObj);
        inequalityPanel.add(objectivePanel); //adding the objective panel to the
inequality panel

    } else if (button == threeVar && (numInequality.getSelectedItem().toString() !=
"--")) { //checking input button
    numVAR = 3;
    rowHeaders = new String[counter + 1]; //creating list of row headers
    for(int a = 0; a < counter; a++){ //adding each slack variable beginning from r
depending on number of inequalities
        rowHeaders[a] = Character.toString(a+114);
    }
    rowHeaders[counter] = "P"; //setting final index as P, this is always the row
header of the bottom row
    colHeaders = new String[counter + 4]; //creating list of column headers
    colHeaders[0] = "x"; //adding first few items to column headers'
    colHeaders[1] = "y";
    colHeaders[2] = "z";
    for(int a = 0; a < counter; a++){
        colHeaders[a+3] = Character.toString(a+114); //adding the slack variables to
the column headers in a similar manner to above
    }
    colHeaders[counter + 3] = "Value"; //adding the final column header as 'Value',
which is always the same
    for (int i = 0; i < counter; i++) {
        JPanel rowPanel = new JPanel(); //creating a new panel for each row
        rowPanel.setLayout(new FlowLayout());
        rowPanel.setMaximumSize(new Dimension(650, 150));

        JLabel plus1 = new JLabel("+"); //creating the labels to go between the text
areas
        plus1.setBorder(new EmptyBorder(0, 10, 0, 10));

        JLabel plus2 = new JLabel("+");
        plus2.setBorder(new EmptyBorder(0, 10, 0, 10));

        xTA = new JTextArea(2, 5); //defining the components for each row
        yTA = new JTextArea(2, 5);
        zTA = new JTextArea(2, 5);
        xLabel = new JLabel("x");
        xLabel.setFont(new Font("Arial", Font.PLAIN, 15));
        yLabel = new JLabel("y");
        yLabel.setFont(new Font("Arial", Font.PLAIN, 15));
        zLabel = new JLabel("z");
        zLabel.setFont(new Font("Arial", Font.PLAIN, 15));
    }
}

```

```

        xLabel.setBorder(new EmptyBorder(0,0, 5, 0)); //adding adequate spacing
between components
        yLabel.setBorder(new EmptyBorder(0,0, 5, 0));
        zLabel.setBorder(new EmptyBorder(0,0, 5, 0));
        valueLabel = new JLabel("=");
        valueLabel.setFont(new Font("Arial", Font.PLAIN, 15));
        valueLabel.setBorder(new EmptyBorder(0, 20, 0, 20));
        valueTA = new JTextArea(2, 5);

        xTA.setBorder(new EmptyBorder(0,0, 10, 0));
        yTA.setBorder(new EmptyBorder(0,0, 10, 0));
        zTA.setBorder(new EmptyBorder(0,0, 10, 0));

        rowPanel.add(xTA); //adding each of the components to the rowPanel in the
order they appear due to flow layout
        rowPanel.add(xLabel);
        rowPanel.add(plus1);
        rowPanel.add(yTA);
        rowPanel.add(yLabel);
        rowPanel.add(plus2);
        rowPanel.add(zTA);
        rowPanel.add(zLabel);
        rowPanel.add(valueLabel);
        rowPanel.add(valueTA);

        inequalityPanel.add(rowPanel); //adding each row panel to the inequality
panel
    }
    JPanel objectivePanel = new JPanel(); //creating an objective panel to go at the
bottom of the inequality panel
    objectivePanel.setLayout(new FlowLayout()); //the panel will have a flow layout
    objectivePanel.setMaximumSize(new Dimension(650, 150)); //limiting the size of
the panel
    objLabel = new JLabel("P ="); //the following section defines the components that
go into the objective function
    objLabel.setBorder(new EmptyBorder(0, 0, 0, 20));
    xObjTA = new JTextArea(2, 5);
    yObjTA = new JTextArea(2, 5);
    zObjTA = new JTextArea(2,5);
    xObj = new JLabel("x");
    xObj.setFont(new Font("Arial", Font.PLAIN, 15));
    yObj = new JLabel("y");
    yObj.setFont(new Font("Arial", Font.PLAIN, 15));
    zObj = new JLabel("z");
    zObj.setFont(new Font("Arial", Font.PLAIN, 15));
    xObj.setBorder(new EmptyBorder(0,0, 5, 0)); //setting borders to ensure spacing
between components
    yObj.setBorder(new EmptyBorder(0,0, 5, 0));

```

```

zObj.setBorder(new EmptyBorder(0,0, 5, 0));
plusObj = new JLabel("+");
plusObj.setBorder(new EmptyBorder(0, 10, 0, 10));
JLabel plus1Obj = new JLabel("+");
plus1Obj.setBorder(new EmptyBorder(0, 10, 0, 10));

objectivePanel.add(objLabel); //adding all the created components to the
objective panel
objectivePanel.add(xObjTA);
objectivePanel.add(xObj);
objectivePanel.add(plusObj);
objectivePanel.add(yObjTA);
objectivePanel.add(yObj);
objectivePanel.add(plus1Obj);
objectivePanel.add(zObjTA);
objectivePanel.add(zObj);
inequalityPanel.add(objectivePanel); //adding the panel to the inequality panel

} else if (button == fourVar && (numInequality.getSelectedItem().toString() != "--"))
{
    numVAR = 4;
    rowHeaders = new String[counter + 1]; //creating row header array
    for(int a = 0; a < counter; a++){
        rowHeaders[a] = Character.toString(a+114); //adding slack variables starting
from r, using ASCII
    }
    rowHeaders[counter] = "P"; //adding P to the final index of the row headers, this
is always the same
    colHeaders = new String[counter + 5];//creating column headers array
    colHeaders[0] = "x1"; //adding 4 variables manually to the start of column
headers
    colHeaders[1] = "x2";
    colHeaders[2] = "x3";
    colHeaders[3] = "x4";
    for(int a = 0; a < counter; a++){
        colHeaders[a+4] = Character.toString(a+114); //adding slack variables
starting from r, similar to above using ASCII
    }
    colHeaders[counter + 4] = "Value"; //adding 'Value' in the final column, which is
the same for all tableaux
    for (int i = 0; i < counter; i++) {
        JPanel rowPanel = new JPanel();
        rowPanel.setLayout(new FlowLayout());
        rowPanel.setMaximumSize(new Dimension(650, 150));

        JLabel plus1 = new JLabel("+"); //this time adding 3 plus symbols as labels
to the GUI
        plus1.setBorder(new EmptyBorder(0, 10, 0, 10));

```

```

JLabel plus2 = new JLabel("+");
plus2.setBorder(new EmptyBorder(0, 10, 0, 10));

JLabel plus3 = new JLabel("+");
plus3.setBorder(new EmptyBorder(0, 10, 0, 10));

x1TA = new JTextArea(2, 5); //defining the text areas and components of the
GUI
x2TA = new JTextArea(2, 5);
x3TA = new JTextArea(2, 5);
x4TA = new JTextArea(2, 5);
x1Label = new JLabel("x1");
x1Label.setFont(new Font("Arial", Font.PLAIN, 15));
x2Label = new JLabel("x2");
x2Label.setFont(new Font("Arial", Font.PLAIN, 15));
x3Label = new JLabel("x3");
x3Label.setFont(new Font("Arial", Font.PLAIN, 15));
x4Label = new JLabel("x4");
x4Label.setFont(new Font("Arial", Font.PLAIN, 15));
x1Label.setBorder(new EmptyBorder(0,0, 20, 0)); //creating border spacing
around the components
x2Label.setBorder(new EmptyBorder(0,0, 20, 0));
x3Label.setBorder(new EmptyBorder(0,0, 20, 0));
x4Label.setBorder(new EmptyBorder(0,0, 20, 0));
valueLabel = new JLabel "=";
valueLabel.setFont(new Font("Arial", Font.PLAIN, 15));
valueLabel.setBorder(new EmptyBorder(0, 20, 10, 20));
valueTA = new JTextArea(2, 5);

x1TA.setBorder(new EmptyBorder(0,0, 10, 0));
x2TA.setBorder(new EmptyBorder(0,0, 10, 0));
x3TA.setBorder(new EmptyBorder(0,0, 10, 0));
x4TA.setBorder(new EmptyBorder(0,0, 10, 0));

rowPanel.add(x1TA); //adding each of the components to the row panel created
at the start of the for loop
rowPanel.add(x1Label);
rowPanel.add(plus1);
rowPanel.add(x2TA);
rowPanel.add(x2Label);
rowPanel.add(plus2);
rowPanel.add(x3TA);
rowPanel.add(x3Label);
rowPanel.add(plus3);
rowPanel.add(x4TA);
rowPanel.add(x4Label);
rowPanel.add(valueLabel);

```

```

        rowPanel.add(valueTA);

        inequalityPanel.add(rowPanel);
    }
    JPanel objectivePanel = new JPanel(); //creating a new panel for the objective
function
    objectivePanel.setLayout(new FlowLayout());
    objectivePanel.setMaximumSize(new Dimension(650, 150));
    objLabel = new JLabel("P =");
    objLabel.setBorder(new EmptyBorder(0, 0, 0, 20));
    x1ObjTA = new JTextArea(2, 5); //defining each of the text areas that will appear
    x2ObjTA = new JTextArea(2, 5);
    x3ObjTA = new JTextArea(2,5);
    x4ObjTA = new JTextArea(2,5);
    x1Obj = new JLabel("x1");
    x1Obj.setFont(new Font("Arial", Font.PLAIN, 15)); //altering the font style and
size to fit proportional to the text areas
    x2Obj = new JLabel("x2");
    x2Obj.setFont(new Font("Arial", Font.PLAIN, 15));
    x3Obj = new JLabel("x3");
    x3Obj.setFont(new Font("Arial", Font.PLAIN, 15));
    x4Obj = new JLabel("x4");
    x4Obj.setFont(new Font("Arial", Font.PLAIN, 15));
    x1Obj.setBorder(new EmptyBorder(0,0, 5, 0)); //creating spaces/borders between
the elements
    x2Obj.setBorder(new EmptyBorder(0,0, 5, 0));
    x3Obj.setBorder(new EmptyBorder(0,0, 5, 0));
    x4Obj.setBorder(new EmptyBorder(0,0, 5, 0));
    plusObj = new JLabel("+");
    plusObj.setBorder(new EmptyBorder(0, 10, 0, 10));
    JLabel plus1Obj = new JLabel("+");
    JLabel plus2Obj = new JLabel("+");
    plus1Obj.setBorder(new EmptyBorder(0, 10, 0, 10));
    plus2Obj.setBorder(new EmptyBorder(0, 10, 0, 10));

    objectivePanel.add(objLabel); //adding all the created components to the
objective panel
    objectivePanel.add(x1ObjTA);
    objectivePanel.add(x1Obj);
    objectivePanel.add(plusObj);
    objectivePanel.add(x2ObjTA);
    objectivePanel.add(x2Obj);
    objectivePanel.add(plus1Obj);
    objectivePanel.add(x3ObjTA);
    objectivePanel.add(x3Obj);
    objectivePanel.add(plus2Obj);
    objectivePanel.add(x4ObjTA);
    objectivePanel.add(x4Obj);

```

```

        inequalityPanel.add(objectivePanel); //adding the objective panel to the
inequality panel
    }

inequalityPanel.add(simplexPanel); // Add the panel to the frame and refresh the GUI
add(inequalityPanel, BorderLayout.CENTER);
revalidate();
repaint();
}

public void runSimplexAlg(int numInequality, int numVAR){
    constraints = new double[numInequality][numVAR + 1]; //2d array of constraints
    try {
        for (int i = 0; i < numInequality; i++) { //iterating through each set of
inequalities
            JPanel parsePanel = (JPanel) inequalityPanel.getComponent(i); //creates a
new panel object to extract components from
            int columnIndex = 0; //introducing the columnIndex counter
            for (int j = 0; j < parsePanel.getComponentCount(); j++) { //iterating
through the panel
                Component component = parsePanel.getComponent(j);
                if (component instanceof JTextArea) { //checks if the current
component is a text area
                    if (columnIndex < constraints[i].length) {
                        constraints[i][columnIndex] = Double.parseDouble(((JTextArea)
parsePanel.getComponent(j)).getText()); //adds the value in the text area to the array
                        columnIndex++;
                    }
                }
            }
        }
        JPanel objectivePanel = (JPanel) inequalityPanel.getComponent(numInequality);
//creates a new objective panel
        ArrayList<Double> objective = new ArrayList<>(); //creating an arraylist to
store the objective function
        for (int k = 0; k < objectivePanel.getComponentCount(); k++) {
            Component comp = objectivePanel.getComponent(k);
            if (comp instanceof JTextArea) { // Check if the current component is a
JTextArea
                String text = ((JTextArea) comp).getText();
                objective.add(Double.parseDouble(text)); //converts the text to a
double
            }
        }
    }
}

```

```

        }
    }
    errorMessageLabel.setText("");
    for (int u = 0; u < objective.size(); u++) {
        System.out.println(objective.get(u));
    }
    obj = new double[numVAR];
    for (int l = 0; l < objective.size(); l++) {
        obj[l] = objective.get(l); //converting the arraylist into an array to be
passed as a parameter
    }
    Simplex problem = new Simplex(constraints, obj); //creating a new object of
type Simplex
    Iterate.solve(problem); //solving the created problem

    JPanel combinedPanel = new JPanel();
    combinedPanel.setLayout(new BorderLayout()); // Use BorderLayout for better
control

    JPanel tableauWrapper = new JPanel();
    tableauWrapper.setLayout(new BorderLayout());
    tableauWrapper.setBorder(BorderFactory.createEmptyBorder(10, 10, 75, 50)); //
Adjust top and left padding

    JPanel tableauPanel = Tableau.displayTableau(rowHeaders, colHeaders,
numInequality + 1, numVAR + numInequality + 1);

    tableauWrapper.add(tableauPanel, BorderLayout.CENTER); // Add tableau to the
wrapper
    combinedPanel.add(inequalityPanel, BorderLayout.CENTER); // Add the
inequality panel
    combinedPanel.add(tableauWrapper, BorderLayout.SOUTH); // Add the tableau
wrapper to the bottom
    add(combinedPanel, BorderLayout.CENTER); // Add the combined panel to the
main frame

    revalidate();
    repaint();
} catch(NumberFormatException e) {
    errorMessageLabel.setText("Please enter valid numbers in all the text
areas.");
}
}

public static String[] getRowHeaders(){
    return rowHeaders;
}

```

```

public static String[] getColHeaders(){
    return colHeaders;
}

}

public class event implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if (e.getSource() == exit) { //checks if source of event being called is
'exit' option
            System.exit(0); //closes the window

        } else if (e.getSource() == home) { //checks if source of event is 'home'
option
            restoreHome(); //restores the window to its original state
        }else if (e.getSource() == runPrims){
            getContentPane().removeAll();
            graphSettings();
        } else if (e.getSource() == runKruskals){
            getContentPane().removeAll();
            graphSettings(); //displays the graph buttons on screen if the run
kruskal's option is selected
        } else if (e.getSource() == introKruskals){
            kruskalsIntro();
        } else if (e.getSource() == addEdge){
            edge(); //displays the text areas for input if the add edge button is
pressed
        } else if (e.getSource() == addButton){
            addToGraph(); //adds the edge to an arraylist of edges to be added if the
add button is pressed
        } else if(e.getSource() == removeEdge) {
            removeFromGraph();
        }else if(e.getSource() == removeEdgeButton){
            remove();
        } else if(e.getSource() == resetGraph){
            resetGraphDisplay();
        } else if (e.getSource() == runAlgorithm){
            runKruskalsAlg(); //creates the graph, adding all the edges from the
arraylist and runs the algorithm if run algorithm button is pressed
        } else if (e.getSource() == runSimplex){
            getContentPane().removeAll();
            simplexSettings();
        } else if (e.getSource() == twoVar) {
            if (numInequality.getSelectedItem().toString().equals("--")) {
                errorMessageLabel.setText("Please select the number of inequalities
first.");
            } else {
                errorMessageLabel.setText(""); // clear the error message
            }
        }
    }
}

```

```

        setSimplex(twoVar);
    }
} else if (e.getSource() == threeVar) {
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of inequalities
first.");
    } else {
        errorMessageLabel.setText(""); // clear the error message
        setSimplex(threeVar);
    }
} else if (e.getSource() == fourVar) {
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of inequalities
first.");
    } else {
        errorMessageLabel.setText(""); // clear the error message
        setSimplex(fourVar);
    }
} else if (e.getSource() == runSimplexButton) {
    // validate if the number of inequalities and variables are selected
    if (numInequality.getSelectedItem().toString().equals("--")) {
        errorMessageLabel.setText("Please select the number of
inequalities.");
    } else if (numVAR == 0) { //numVAR is set to 0 if no variable type is
selected
        errorMessageLabel.setText("Please select the number of variables.");
    } else {
        int selectedInequality =
        Integer.parseInt(numInequality.getSelectedItem().toString());
        runSimplexAlg(selectedInequality, numVAR);
    }
} else{
    getContentPane().removeAll(); //clears the window and updates any changes
made
    revalidate();
    repaint();
}
}

}

public static void main(String[] args){
    menuGUI gui = new menuGUI(); //creating an instance of the menuGUI class and
setting the qualities of the window
    gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    gui.setSize(1360, 768);
    gui.setVisible(true);
}

```

```

}

import javax.swing.BorderFactory;
import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.math.*;
import java.util.Random;

public class Tableau extends menuGUI {
    private static double[][] table;
    private static int rows;
    private static int cols;
    private static double pivotVal;
    private static ArrayList<Double> thetaValues = new ArrayList<>();
    private static ArrayList<ArrayList<String>> rowOperationList = new ArrayList<>();
    public Tableau(Simplex problem) { //constructor to create an empty array for an
initial tableau
        int numConstraints = problem.getConstraints().length; // the number of
constraints is the length of the constraints array
        int numVar = problem.getObjective().length; //the number of variables is the same
as the length of the objective array

        rows = numConstraints + 1;
        cols = numVar + numConstraints + 1;
        table = new double[rows][cols];

        initialiseTableau(problem);
    }

    public static double[][] makeTable(Tableau copyTableau) {
        double[][] copyTable = new double[rows][cols]; //creates a new 2d array to store
the deep copy of the table
        double[][] existingTable = copyTableau.getTable(); //assigns the parameter to a
2d array
        for (int i = 0; i < existingTable.length; i++) {
            for (int j = 0; j < existingTable[i].length; j++) {
                copyTable[i][j] = existingTable[i][j]; //copies the array
            }
        }
    }
}

```

```

        return copyTable;
    }

private void initialiseTableau(Simplex problem) {
    double[][] constraints = problem.getConstraints();
    double[] objective = problem.getObjective();
    int numVar = objective.length;

    for (int i = 0; i < constraints.length; i++) {
        for (int j = 0; j < constraints[i].length - 1; j++) {
            table[i][j] = constraints[i][j]; //adding the coefficients from
constraints[][] to table[][]
        }
        table[i][numVar + i] = 1; //adding slack variables in respective indices
        table[i][cols - 1] = constraints[i][constraints[i].length - 1]; // adding the
values in the value column
    }
    for (int j = 0; j < objective.length; j++) {
        table[rows - 1][j] = -1 * objective[j];//adding the values of the objective
function in the bottom row * -1
    }
}

public void pivot(int pivotRow, int pivotCol) {
    double pivotValue = table[pivotRow][pivotCol]; //using the pivot row and pivot
column index, we can set a pivot value
    for (int j = 0; j < cols; j++) {
        table[pivotRow][j] /= pivotValue; //goes through the pivot row dividing every
value by the pivot value
    }
    for (int i = 0; i < rows; i++) {
        if (i != pivotRow) {
            double factor = table[i][pivotCol]; //goes through every other row
subtracting the pivot column value from the current index value
            for (int j = 0; j < cols; j++) {
                table[i][j] -= factor * table[pivotRow][j];
            }
        }
    }
}

public boolean isOptimal() {
    for (int c = 0; c < cols - 1; c++) { // iterating through the objective row
        if (table[rows - 1][c] < 0) { //if there are still negative values then we
return false
            return false;
    }
}

```

```

        }
        return true;
    }
    public static ArrayList<Double> getThetaValues(double[][] tableau) {
        thetaValues.clear();
        int pivotC = getCurrentPivotCol(tableau); //get the pivot column index

        for (int i = 0; i < tableau.length; i++) {
            if (tableau[i][pivotC] > 0) { //ensure we are not dividing by zero
                double first = tableau[i][cols - 1];
                double second = tableau[i][pivotC];
                System.out.println(first);
                System.out.println(second);
                System.out.println(" ");
                double theta = tableau[i][cols - 1] / tableau[i][pivotC]; //the value
                column is at cols - 1
                thetaValues.add(theta); //add the calculated theta value to the list
            } else {
                thetaValues.add(Double.POSITIVE_INFINITY); //handles cases where division
                by zero would occur
            }
        }
        return thetaValues;
    }
    public static int getPivotCol() {
        int pivotCol = 0;
        double minValue = table[rows - 1][0]; //sets the minimum value as the bottom left
        value of the table
        for (int i = 0; i < cols - 1; i++) { //iterates through objective row checking if
        next value smaller (more negative) than current minValue
            if (table[rows - 1][i] < minValue) {
                minValue = table[rows - 1][i]; // if true, sets minValue as this new
                value
                pivotCol = i; //sets index of pivot column
            }
        }
        return pivotCol;
    }

    public static int getPivotRow(int pivotCol) {
        int pivotRow = 0;
        double minRatio = Double.MAX_VALUE; //sets value of new variable equal to the max
        value offered by java
        for (int i = 0; i < rows - 1; i++) {
            double ratio = table[i][cols - 1] / table[i][pivotCol]; //divides value
            column by pivot column
            if (ratio > 0 && ratio < minRatio) { // if result > 0 and < current minRatio
            then sets minRatio to current value
                minRatio = ratio;
            }
        }
    }
}

```

```

        pivotRow = i; //sets index of pivot row
    }
}
pivotVal = table[pivotRow][pivotCol];
return pivotRow;
}

public void printTable() {
    for (double[] row : table) {
        for (double value : row) {
            System.out.printf("%10.2f", value); //leaves spacing of 10 and outputs
with 2 decimal places
        }
        System.out.println();
    }
}

public static ArrayList<String> getRowOperations(double[][] tableau) {
    ArrayList<String> rowOps = new ArrayList<>(); //create a new list for each call
    int pivotC = getCurrentPivotCol(tableau); //getting the pivot row and pivot
column of the tableau
    int pivotR = getCurrentPivotRow(pivotC, tableau);
    for(int i = 0; i < tableau.length; i++){
        double val = tableau[i][pivotC];
        if(i != pivotR){
            if(val > 0) { //checking if the value in the pivot column is negative
                if(val % 1 == 0){ //checking if value in pivot column is integer
                    String formatting = Double.toString(Math.abs(val)); //new string
which takes the positive version of the
                    formatting.replace(".0", ""); //if integer then it removes the .0
from the end of the double
                    rowOps.add("R" + (i + 1) + " - " + formatting + "R" + pivotR);
//it is negative, so it has been converted to positive then the string '-' is put in
front of it
                }else{
                    rowOps.add("R" + (i + 1) + " - " + String.format("%.1f",
(Math.abs(val))) + "R" + pivotR); //if it's not an integer, then round the absolute value
and format the row operation
                }
            }
        } else{ //if positive then repeat the same process but add a '+' sign
instead
            if(val % 1 == 0){
                String formatting = Double.toString(Math.abs(val)); //if integer
then remove the .0 and add to rowOps
                formatting.replace(".0", "");
                rowOps.add("R" + (i + 1) + " + " + formatting+ "R" + pivotR);
            }
        }
    }
}

```

```

        } else{
            rowOps.add("R" + (i + 1) + " + " + String.format("%.1f",
(Math.abs(val))) + "R" + pivotR);
        }
    }
} else{ //this case is for if the value is in the pivot row
    if(val % 1 == 0) {
        String formatting = Double.toString(Math.abs(val));
        formatting.replace(".0", ""); //remove the .0 if the value is an
integer
        rowOps.add("R" + (pivotR + 1) + " / " + formatting); //format the row
operation for the pivot row
    } else{
        rowOps.add("R" + (pivotR + 1) + " / " + String.format("%.1f",
(Math.abs(val)))); //no need to handle negative cases as the pivot value can never be
negative
    }
}
return rowOps; //return the new list
}

```

```

public static JPanel displayTableau(String[] rowHeaders, String[] colHeaders, int
rows, int cols) {
    ArrayList<double[][]> toOutput = Iterate.getTableaux(); // creates a new
arraylist of 2d arrays which gets all the tableaux from the iterations of the algorithm
    ArrayList<String[]> rowHeaderArray = Iterate.getRowHeadersList(); //gets the list
of rowheaders from the Iterate class and assigns them to an arraylist of string arrays
    JTabbedPane tabbedPane = new JTabbedPane(); //creating the tabbed pane to display
the tableaux
    tabbedPane.setPreferredSize(new Dimension(1000, 300)); //dimensions to ensure no
overlap between the tabbed pane and the inequality panel
    JPanel fullPanel = new JPanel(); //creating the panel that will store the tabbed
pane and will be the main panel added to the gui
    fullPanel.setLayout(new FlowLayout());
    int tableauCount = 0;
    for(int k = 0; k < toOutput.size() - 1; k++){ //getting the row operations for
each tableau in the list above
        rowOperationList.add(getRowOperations(toOutput.get(k)));
    }
    for (double[][] outputTable : toOutput) {
        ArrayList<Double> thetaList = getThetaValues(outputTable); //getting the
theta values for each tableau in the toOutput list
        JPanel thetaPanel = new JPanel(); //creating a new panel to store the theta
values and setting the layout as grid layout with 1 column to give the table format
        thetaPanel.setLayout(new GridLayout(0, 1));thetaPanel.setPreferredSize(new
Dimension(5, thetaPanel.getPreferredSize().height));
        JLabel theta = new JLabel("θ"); //label which indicates the theta column
    }
}

```

```

        theta.setFont(new Font("Arial", Font.BOLD, 15));
        theta.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //sets the
font, border and alignment of the theta label
        theta.setHorizontalAlignment(SwingConstants.CENTER);
        theta.setHorizontalTextPosition(SwingConstants.CENTER);
        thetaPanel.add(theta); //adds the label to the panel
        JPanel rowOpPanel= new JPanel(); //creates a new panel to store the row
operations
        rowOpPanel.setLayout(new GridLayout(0, 1)); //setting the layout of the panel
in a similar way to the theta panel
        rowOpPanel.setPreferredSize(new Dimension(5,
thetaPanel.getPreferredSize().height)); //fixing the size of the panel
        JLabel rowOp = new JLabel("Row Ops."); //setting the text to be added to the
top of the rowOpPanel
        rowOp.setHorizontalAlignment(SwingConstants.CENTER); //setting the font,
border and alignment of the text label
        rowOp.setHorizontalTextPosition(SwingConstants.CENTER);
        rowOp.setFont(new Font("Arial", Font.BOLD, 15));
        rowOp.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        rowOpPanel.add(rowOp); //adds label to the panel
        for(Double value: thetalist){
            if (value == Double.POSITIVE_INFINITY){ //checks if theta value is
infinity (dividing by 0)
                JLabel thetaLabel = new JLabel("--"); //sets it to -- as it need not
be considered for the pivot row
                thetaLabel.setFont(new Font("Arial",Font.PLAIN, 15)); //sets the
font, border and alignment of the theta label
                thetaLabel.setHorizontalAlignment(SwingConstants.CENTER);
                thetaLabel.setHorizontalTextPosition(SwingConstants.CENTER);
                thetaLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
                thetaPanel.add(thetaLabel); //adding to thetaPanel
            } else {
                if(value % 1 == 0){ //checks if the value is an integer
                    String integerConversion = (Double.toString(value)).replace(".0",
""); //converting value to a string and removing the .
                    JLabel thetaLabel = new JLabel(integerConversion); //setting the
theta label as the string created above
                    thetaLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //setting
the font, border and alignment of the label
                    thetaLabel.setHorizontalTextPosition(SwingConstants.CENTER);
                    thetaLabel.setHorizontalAlignment(SwingConstants.CENTER);

                    thetaLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
                    thetaPanel.add(thetaLabel); //adding label to thetaPanel
                } else {
                    JLabel thetaLabel = new JLabel(String.format("%.2f",value));
//rounds the value to 2dp and adds it
                    thetaLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //setting
font, border and alignment of label
                    thetaLabel.setHorizontalTextPosition(SwingConstants.CENTER);

```

```

        thetaLabel.setHorizontalAlignment(SwingConstants.CENTER);

thetaLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    thetaPanel.add(thetaLabel); //adding label to thetaPanel
}
}
}

if(tableauCount >= 1){ // checking to make sure not iterating on the first
tableau
    ArrayList<String> rowOpList = rowOperationList.get(tableauCount-1); //if
any tableau that isn't the initial tableau, then add row operations
    for(int b = 0; b < rowOpList.size(); b++) {
        System.out.println(rowOpList.get(b));
        JLabel rowOpLabel = new JLabel(rowOpList.get(b)); //sets a label with
the text of the row operation from rowOpList
        rowOpLabel.setFont(new Font("Arial", Font.PLAIN, 15)); //sets font,
border and alignment of the label
        rowOpLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        rowOpLabel.setHorizontalAlignment(SwingConstants.CENTER);
        rowOpLabel.setHorizontalTextPosition(SwingConstants.CENTER);
        rowOpPanel.add(rowOpLabel); //adds the label to the rowOpPanel
    }
}
JPanel gridPanel = new JPanel(new GridBagLayout()); //new panel with gridbag
layout for better control over spacing of components
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridx = 0; //column 0
gbc.gridy = 0; //row 0
gbc.gridwidth = 1; //span 1 column
gbc.weightx = 0.5; //take up 75% of the panel's width
gbc.fill = GridBagConstraints.BOTH; //expand both horizontally and vertically

JPanel tableauPanel = createTableauPanel(outputTable,
rowHeaderArray.get(tableauCount), colHeaders, rows, cols, tableauCount,toOutput.size()-
1); //creates a formatted tableau based on the current table

gridPanel.add(tableauPanel, gbc);
if(tableauCount >= 1) { //checks if not on initial tableau before adding row
operations to the gui
    gbc.gridx = 1; //adds to the right of the tableau
    gbc.gridy = 0;
    gbc.weightx = 0.3;
    gridPanel.add(rowOpPanel, gbc); //adds the row operation panel to the
grid panel
}
gbc.gridx = 2; //adds the theta panel to the gui to the right of the row
operations

```

```

        gbc.gridx = 0;
        gbc.gridy = 0;
        gridPanel.add(thetaPanel, gbc); //adds the theta panel to the grid panel

        JPanel explainPanel = explainTableau(outputTable, tableauCount,
toOutput.size() - 1); //creates a new panel which contains the explanation of the tableau
        gbc.gridx = 0;
        gbc.gridy = 1; //adds underneath the tableau
        gbc.gridwidth = 3; // makes the panel span the entire width of the gridPanel
        gbc.fill = GridBagConstraints.BOTH;
        gridPanel.add(explainPanel, gbc); //adds the explanation to the grid panel

        tabbedPane.add("Iteration " + (tabbedPane.getTabCount() + 1), gridPanel);
//sets the title of each tab in the tabbed pane as the iteration of the algorithm
        tableauCount++; //increases counter of which iteration we are on
    }
    fullPanel.add(tabbedPane); //adds the tabbed pane to the main panel

    return fullPanel; //returns fullPanel as the main panel to be added to the GUI
}

public static JPanel explainTableau(double[][][] tableau, int a, int numTableau){
    JPanel addPanel = new JPanel(); //new panel to contain explanation
    addPanel.setLayout(new BoxLayout(addPanel, BoxLayout.Y_AXIS)); //set box layout
for this panel
    ArrayList<JTextPane> initialResponses = new ArrayList<>(); //list of possible
explanation responses for initial tableau
    ArrayList<JTextPane> intermediateResponses = new ArrayList<>(); //list of
possible explanation responses for intermediate tableau
    JTextPane option1 = new JTextPane(); //explaining the tableau, using line breaks
for better flow of reading
        option1.setText("Looking through the objective row, the pivot column was selected
as " + " " + getColHeaders()[getCurrentPivotCol(tableau)] + ' ' + ", highlighted in
blue, as this has the most negative value in the objective row." + " By dividing each
number in the Value column by the respective value in the pivot column and selecting the
smallest positive result, the pivot row was selected as " + " " +getRowHeaders()
[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)] + '.');
        option1.setText(option1.getText() + " This gives us the pivot value as '" +
Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]
[getCurrentPivotCol(tableau)] * 100.0) / 100.0 + ' .' );
        option1.setText(option1.getText() + "\nOur aim is to reduce the pivot value to 1
and everything else in the pivot column to 0, so our first row operation is [the values
in the pivot row] / '" +
Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]
[getCurrentPivotCol(tableau)] * 100) / 100 + '.');
        option1.setText(option1.getText() + " We now look at each value in the pivot
column by turn. We must see how we can reduce the value to 0 using the pivot value, which

```

```

is now 1.");
    option1.setText(option1.getText() + "\nWe take a value from the pivot column that
is not the pivot value, we then subtract the value multiplied by the pivot row, so if the
value is 5, we take 5 * 1 away from it, giving us 0. This process is repeated throughout
the row, using the respective values in the pivot row. We then repeat which each non-
pivot value in the pivot column.");
    option1.setText(option1.getText() + "\nWe repeat the process of applying row
operations until we have completed the tableau.");
    option1.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font size and style
of the explanation text
    option1.setEditable(false); //ensure the user cannot edit the explanation

    JTextPane option2 = new JTextPane();
    option2.setText("We look through the objective row for the most negative value.
This gives us the pivot column as " + " " + getColHeaders()[getCurrentPivotCol(tableau)]
+ " " + ", highlighted in blue. To find the theta values, we divide each number in the
value column by its respective value in the pivot column.");
    option2.setText(option2.getText() + " We now select the smallest, positive theta
value, and the row in which this is situated is our pivot row, highlighted in orange. The
intersection between the pivot column and pivot row is our pivot value, '" +
Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]
[getCurrentPivotCol(tableau)] * 100.0) / 100.0 + "' .");
    option2.setText(option2.getText() + "\nOur aim is to reduce the pivot value to 1
and everything else in the pivot column to 0, which gives us our first row operation as
[the values in the pivot row] / our pivot value, '" +
Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]
[getCurrentPivotCol(tableau)] * 100.0) / 100.0 + "' .");
    option2.setText(option2.getText() + " We look at each value in the pivot column.
We must see how we can reduce the value to 0 using the pivot value, which has now been
reduced to 1.");
    option2.setText(option2.getText() + " We take a value from the pivot column that
is not the pivot value and subtract it multiplied by the pivot row. This process is
applied across the row, using the respective values in the pivot row, and is repeated for
each non-pivot value in the pivot column.");
    option2.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font size and style
of the explanation text
    option2.setEditable(false); //ensure the user cannot edit the explanation

    JTextPane midOption1 = new JTextPane();
    midOption1.setText("Applying the row operations to the previous tableau gives us
this tableau, with those row operations displayed next to it. We repeat the process from
the previous tableau. Looking through the objective row, the pivot column is " + " " +
getColHeaders()[getCurrentPivotCol(tableau)] + " " + ", highlighted in blue. This was
chosen as this column has the most negative value in the objective row.");
    midOption1.setText(midOption1.getText() + "\nBy dividing each value in the Value
column by its respective value in the pivot column, we get our theta values, shown right
of the tableau. We select the smallest, positive theta value to give us our pivot row.
The intersection between pivot row and pivot column, is our pivot value, '" +
Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]

```

```

[getCurrentPivotCol(tableau)] * 100.0) / 100.0 + "", highlighted in purple.");
    midOption1.setText(midOption1.getText() + " We want to reduce the pivot value to
1 and everything else in the pivot column to 0. The first row operation we perform is on
the pivot row and involves dividing each value in the pivot row by the pivot value to
reduce our pivot value to 1. We now pick the first value in the pivot column that isn't
in the pivot row, and reduce it to 0, using the values in the pivot row.");
    midOption1.setText(midOption1.getText() + " As the pivot value is 1, we just
subtract whatever the value in the pivot column is * 1 from the value in the pivot column
to reduce it to 0. We must perform the same row operation on each value in the row.");
    midOption1.setText(midOption1.getText() + "\n We repeat this process for each
row, giving us our new tableau.");
    midOption1.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font size and
style of the explanation text
    midOption1.setEditable(false); //ensure the user cannot edit the explanation

    JTextPane midOption2 = new JTextPane();
    midOption2.setText("Performing the row operations on the previous tableau yields
the following tableau, with the row operations shown alongside it. We repeat the steps
from the previous tableau. Examining the objective row, the pivot column is identified as
" + " " + getColHeaders()[getCurrentPivotCol(tableau)] + " " + ", highlighted in blue,
as it corresponds to the most negative value in the objective row.");
    midOption2.setText(midOption2.getText() + " We then identify the smallest
positive theta value, and the row containing this value is chosen as the pivot row,
highlighted in orange. The intersection of the pivot column and pivot row marks the pivot
value, " + Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]
[getCurrentPivotCol(tableau)] * 100.0) / 100.0 + ' .");
    midOption2.setText(midOption2.getText() + " Our goal is to reduce the pivot value
to 1 and all other entries in the pivot column to 0, resulting in our first row operation
as [the values in the pivot row] / our pivot value, " +
Math.round(tableau[getCurrentPivotRow(getCurrentPivotCol(tableau), tableau)]
[getCurrentPivotCol(tableau)] * 100.0) / 100.0 + ' .");
    midOption2.setText(midOption2.getText() + " Next, we examine each value in the
pivot column and determine how to reduce it to 0 using the pivot value, which has already
been reduced to 1.");
    midOption2.setText(midOption2.getText() + "\nFor each non-pivot value in the
pivot column, we calculate the product of that value and the pivot row, then subtract it
from the corresponding entry. This is repeated for all entries in the pivot column to
reduce them to 0.");
    midOption2.setText(midOption2.getText() + "\nWe repeat this process for each row
of the tableau, which will give us our tableau.");
    midOption2.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font size and
style of the explanation text
    midOption2.setEditable(false); //ensure the user cannot edit the explanation

    JTextPane finalOption1 = new JTextPane();
    ArrayList<String[]> rowHeaderArr = Iterate.getRowHeadersList();
    String[] finalRowHeaders = rowHeaderArr.get(a);
    finalOption1.setText("Looking through the objective row, there are no more
negative values, meaning this tableau is final, so we can read off the final values of

```

```

our variables by reading along the row headers and their values.");
    for(int i = 0; i < finalRowHeaders.length; i ++){
        finalOption1.setText(finalOption1.getText() + "\n");
        finalOption1.setText(finalOption1.getText() + " " + finalRowHeaders[i] + " =
" + String.format("%.2f", (tableau[i][tableau[i].length - 1])));
    }
    finalOption1.setText(finalOption1.getText() + "\n");
    finalOption1.setText(finalOption1.getText() + "All other variables have value
0.");
    finalOption1.setFont(new Font("Arial", Font.PLAIN, 15)); //setting font size and
style of the explanation text
    finalOption1.setEditable(false); //ensure the user cannot edit the explanation

    initialResponses.add(option1);
    initialResponses.add(option2);
    intermediateResponses.add(midOption1);
    intermediateResponses.add(midOption2);

    if(a == 0){
        Random rand = new Random();
        int r = rand.nextInt(2);
        addPanel.add(initialResponses.get(r));
    } else if (a == numTableau){
        addPanel.add(finalOption1);
    } else{
        Random rand = new Random();
        int r = rand.nextInt(2);
        addPanel.add(intermediateResponses.get(r));
    }

    return addPanel; //returns this panel to be added to grid panel
}

public static int getCurrentPivotCol(double[][] tableau) {
    int pivotCol = 0;
    double minValue = tableau[rows - 1][0]; //sets the minimum value as the bottom
left value of the table
    for (int i = 0; i < cols - 1; i++) { //iterates through objective row checking if
next value smaller (more negative) than current minValue
        if (tableau[rows - 1][i] < minValue) {
            minValue = tableau[rows - 1][i]; // if true, sets minValue as this new
value
            pivotCol = i; //sets index of pivot column
        }
    }
    return pivotCol;
}

```

```

public static int getCurrentPivotRow(int pivotCol, double[][] tableau) {
    int pivotRow = 0;
    double minRatio = Double.MAX_VALUE; //sets value of new variable equal to the max
value offered by java
    for (int i = 0; i < rows - 1; i++) {
        double ratio = tableau[i][cols - 1] / tableau[i][pivotCol]; //divides value
column by pivot column
        if (ratio > 0 && ratio < minRatio) { // if result > 0 and < current minRatio
then sets minRatio to current value
            minRatio = ratio;
            pivotRow = i; //sets index of pivot row
        }
    }
    pivotVal = tableau[pivotRow][pivotCol];
    return pivotRow;
}

private static JPanel createTableauPanel(double[][] outputTable, String[] rowHeaders,
String[] colHeaders, int rows, int cols, int tableauCount,int numberTableau) {
    JPanel tableauPanel = new JPanel(new GridLayout(rows + 1, cols + 1)); //new panel
to add the tableau values, using grid layout to create effect of a table
    JLabel bvLabel = new JLabel("B.V."); //sets the basic variable header in the top
left corner
    bvLabel.setFont(new Font("Arial", Font.BOLD, 15));
    bvLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    tableauPanel.add(bvLabel);
    JLabel valueLabel = null;

    for (String header : colHeaders) {
        JLabel colHeaderLabel = new JLabel(header, SwingConstants.CENTER); //adds
each column header to the top row of the panel
        colHeaderLabel.setHorizontalAlignment(SwingConstants.CENTER); //setting font,
size, border and alignment of label
        colHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER);
        colHeaderLabel.setFont(new Font("Arial", Font.BOLD, 15));
        colHeaderLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        tableauPanel.add(colHeaderLabel); //adding the column header label to the
panel
    }

    for (int i = 0; i < outputTable.length; i++) {
        JLabel rowHeaderLabel = new JLabel(rowHeaders[i], SwingConstants.CENTER);
//adding the row headers in the first column of the table underneath the basic variable
header
        rowHeaderLabel.setHorizontalAlignment(SwingConstants.CENTER); //setting the
font, size, border and alignment of label
        rowHeaderLabel.setHorizontalTextPosition(SwingConstants.CENTER);
    }
}

```



```

        return tableauPanel;
    }

    public double[][] getTable(){ //get method for the table
        return table;
    }
    public double getOptimalValue(){
        return table[rows-1][cols-1]; //gets value in bottom right index of array
    }
    public double[] getSolution(){
        double[] solution = new double[cols - 1]; //creates a new array to store the
values of the decision and slack variables for a solution
        for(int i = 0; i < cols - 1; i++){
            boolean isBasic = false; //iterates through the table checking if a variable
is basic
            for(int j = 0; j < rows; j++){
                if(table[j][i] == 1){
                    solution[i] = table[j][cols - 1]; //sets value in solution array
equal to value in table
                    isBasic = true; //if basic variable then breaks from the for loop
                    break;
                }
            }
            if(!isBasic){
                solution[i] = 0;
            }
        }
        return solution;
    }
}

```

```

import java.util.*;

public class Solve {
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        int numVar;
        int numConstraints;
        while (true){
            System.out.println("Enter number of variables: ");
            if (scanner.hasNextInt()) {
                numVar = scanner.nextInt();
                if (numVar > 0){ // Ensure a positive number of variables entered
                    break;
                }else{
                    System.out.println("Please enter a positive number.");
                }
            }
        }
    }
}

```

```

    }else{
        System.out.println("Invalid input. Please enter only numbers.");
        scanner.next(); // Clear the invalid input
    }
}

while (true){
    System.out.println("Enter number of constraints: ");
    if (scanner.hasNextInt()){
        numConstraints = scanner.nextInt();
        if (numConstraints > 0){ // Ensure that a positive number of constraints
are entered
            break;
        }else{
            System.out.println("Please enter a positive number.");
        }
    }else{
        System.out.println("Invalid input. Please enter only numbers.");
        scanner.next(); // Clear the invalid input
    }
}
scanner.nextLine();

double[][] constraints = new double[numConstraints][numVar + 1]; //creating a 2d
array to stores the coefficients of constraints
System.out.println("Enter the coefficients of the constraints and include the
value: ");
for (int i = 0; i < numConstraints; i++){
    while (true){
        String line = scanner.nextLine();
        String[] parts = line.trim().split("\\s+");
        if(parts.length == numVar + 1) { // checks if the number of inputs
matches numVar + 1
            for (int j = 0; j < numVar + 1; j++) {
                try {
                    constraints[i][j] = Double.parseDouble(parts[j]); // adds
entered coefficients of constraints to 2D array
                } catch (NumberFormatException e) {
                    System.out.println("Invalid input. Please enter only
numbers.");
                }
            }
            break; //exit the loop if valid input
        } else {
            System.out.println("Invalid number of coefficients entered. Please
enter again starting from the beginning.");
        }
    }
}

```

```

    }
    double[] objective = new double[numVar];

    System.out.println("Enter coefficients of the objective function:");
    while (true) {
        String line = scanner.nextLine();
        String[] parts = line.trim().split("\\s+");

        if (parts.length == numVar) { // check if the number of inputs matches numVar
            for (int i = 0; i < numVar; i++) {
                try {
                    objective[i] = Double.parseDouble(parts[i]);
                } catch (NumberFormatException e) {
                    System.out.println("Invalid input. Please enter only numbers.");
                    break;
                }
            }
            break;
        } else {
            System.out.println("Invalid number of coefficients.");
        }
    }
    Simplex problem = new Simplex(constraints, objective); //creating a new object of
    type Simplex
}

import java.util.*;

public class Simplex {
    private double [][] constraints;
    private double[] objective;

    public Simplex(double[][] constraintList, double[] objectiveFunction){ //constructor
for a Simplex problem object
        constraints = constraintList;
        objective = objectiveFunction;
    }

    public double[][] getConstraints(){ //getter for constraints

        return constraints;
    }
    public double[] getObjective(){
        return objective;
    } //getter for objective
}

```

```

import java.lang.reflect.Array;
import java.util.*;
public class Iterate {
    public static ArrayList<double[][]> tableaux = new ArrayList<>();
    public static ArrayList<String[]> rowHeadersList = new ArrayList<>();
    public static void solve(Simplex problem){
        tableaux.clear(); // Clear the previous tableaux
        rowHeadersList.clear();
        Tableau table = new Tableau(problem); //creating a table object taking a
parameter of a simplex problem object
        tableaux.add(Tableau.makeTable(table)); //adding initial tableau to list of
tableaux
        rowHeadersList.add(menuGUI.getRowHeaders()); //adding initial row headers to list
of row headers
        while (!table.isOptimal()) {
            int pivotCol = Tableau.getPivotCol(); //getting pivot column and pivot row to
perform a pivot
            int pivotRow = Tableau.getPivotRow(pivotCol);
            table.pivot(pivotRow, pivotCol);
            String[] newHeader = rowHeadersList.get(rowHeadersList.size() - 1).clone();
//creating new copy of row headers
            newHeader[pivotRow] = menuGUI.getColHeaders()[pivotCol]; //changing row
headers of pivot row
            rowHeadersList.add(newHeader);
            tableaux.add(Tableau.makeTable(table));
        }
        double[] solution = table.getSolution();
        int numDecisionVariables = problem.getObjective().length; // the number of
decision variables is the number of objective function coefficients
        int numSlackVariables = solution.length - numDecisionVariables; // the number of
slack variables is the total number of variables - number of decision variables

        System.out.println("The optimal solution is: ");
        for(int i = 0; i < numDecisionVariables; i++){
            System.out.println("x" + (i+1) + " = " + Math.round(solution[i] *1000000.0) /
1000000.0); //printing from start of solution array to index numDecisionVariables
        }
        for(int i = 0; i < numSlackVariables; i++){
            System.out.println("s" + (i+1) + " = " +
Math.round(solution[numDecisionVariables + i] * 1000000.0)/1000000.0); //printing from
numDecisionVariables + 1 to end of array
        }
        System.out.println("Optimal value: " + Math.round(table.getOptimalValue() *
1000000.0) / 1000000.0); //using getOptimalValue() method from Tableau class to output
optimal value
    }
    public static ArrayList<double[][]> getTableaux(){
        return tableaux;
    }
}

```

```

    }
    public static ArrayList<String[]> getRowHeadersList(){
        return rowHeadersList;
    }

}

import java.lang.reflect.Array;
import java.util.*;
class Edge implements Comparable<Edge>{ //creating a class which defines the attributes
and method of each edge/arc
    int source, dest, weight;
    public Edge(int nodeSource, int nodeDest, int nodeWeight){ //constructor for edge
class
        source = nodeSource - 65;
        dest = nodeDest - 65;
        weight = nodeWeight;
    }
    public int compareTo(Edge otherEdge){ //method to compare the weight of two different
edges
        return weight - otherEdge.weight;
    }
}
public class Kruskals{
    public ArrayList<Edge> edges;
    private int numVertices;

    public Kruskals(int vertices){ //constructor for Kruskal's algorithm class
        numVertices = vertices;
        edges = new ArrayList<>();
    }
    public void addEdge(int source, int dest, int weight){ //method to add edge from
graph to arraylist
        edges.add(new Edge(source, dest, weight));
    }
    private int findComponents(int vertex, ArrayList<Integer> components){ //finds the
component of the graph that a vertex belongs to
        return components.get(vertex);
    }
    public ArrayList<Edge> kruskalMST(){
        Collections.sort(edges); //sorts the arraylist of edges into ascending order
        ArrayList<Integer> components = new ArrayList<>(Collections.nCopies(numVertices,
0));
        for(int i = 0; i < numVertices; i++){
            components.set(i, i);
        }
        ArrayList<Edge> result = new ArrayList<>(); //creates a new arraylist to store
the edges of the MST in
        for (Edge edge : edges) { //iterating through each edge in the arraylist of all
edges from original graph

```

```

        int s = edge.source;
        int d = edge.dest;

        int compS = findComponents(s, components);
        int compD = findComponents(d, components);
        if (compS != compD) { //comparing the edges and adding the edge to the result
arraylist if the addition of the edge doesn't form a cycle
            result.add(edge);
            for (int i = 0; i < numVertices; i++) {
                if (components.get(i) == compD) {
                    components.set(i, compS);
                }
            }
        }
    }
    System.out.println("Edges included in the minimum spanning tree: "); //outputs
the final edges in the MST
    int total = 0;
    for(Edge edge: result){
        total += edge.weight;
        System.out.println("Edge: " + (char)(edge.source + 65) + "" + (char)
(edge.dest + 65) + ", Weight: " + edge.weight);
    }
    System.out.println("Total Weight of MST is: " + total);
    return result;
}
public static void main(String[] args){
}
}

```

```

import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.Array;
import java.util.ArrayList;
import javax.swing.*;

class DisplayPanel extends JPanel {
    public ArrayList<String> edges; // list of edges in the graph
    public ArrayList<Character> nodeLabels = new ArrayList<>(); // list of labels in the
graph
    public ArrayList<Point> nodePositions = new ArrayList<>(); // list of locations on
the panel for the vertices
    public ArrayList<Integer> edgeWeights = new ArrayList<>(); // list of weights for the
edges
    private int diameter = 30; // diameter of the vertices
    private Point dragStart = null; // starting point for dragging

```

```

private int dragIndex = -1; // index of the node being dragged
private String highlightedEdge = null;
private Color edgeColour;
public ArrayList<Color> edgeColourList;
private ArrayList<ArrayList<Color>> colourStates;

public DisplayPanel() { // constructor
    edges = new ArrayList<>();
    edgeColourList = new ArrayList<>();
    setMouseListeners();
}
public void highlightEdge(String edge, Color colour){ //method to colour a specific
edge in the graph
    int index = edges.indexOf(edge);
    if(index != -1){
        edgeColourList.set(index, colour);
    }
    //need to create method that does all the colours automatically at the start
    //highlightedEdge = edge; //the edge to be highlighted passed as a parameter
    //edgeColour = colour; //the colour that this edge should be based on whether it
is in the minimum spanning tree or not
    revalidate(); //refreshing the GUI
    repaint();
}

public void addEdge(String edge, int weight) {
    edges.add(edge); // adding the edge to the final list
    edgeWeights.add(weight); // adding the weight to the list
    edgeColourList.add(Color.BLACK); //adding default colour black to list
    calculatePositions(); // calling the method to calculate the position of the
vertex
    revalidate(); // refreshing the GUI
    repaint();
}

public void calculatePositions() {
    for (String s : edges) {
        char startNode = s.charAt(0); // Separating the edge into start node and end
node
        char destNode = s.charAt(1);

        if (!nodeLabels.contains(startNode)) { // Check if the start node is already
in the graph
            nodeLabels.add(startNode); // If not already in graph then add to the
labels
            nodePositions.add(new Point(0, 0)); // Create a temporary position in the
list of positions
        }
    }
}

```

```

        if (!nodeLabels.contains(destNode)) { // Check if the destination node is
already in the graph
            nodeLabels.add(destNode); // If not already in graph then add to the
labels
            nodePositions.add(new Point(0, 0)); // Create a temporary position in the
list of positions
        }
    }
    int panelWidth = getWidth();
    int panelHeight = getHeight();
    int centerX = panelWidth / 2; // Define the x-coordinate of the center of the
panel
    int centerY = panelHeight / 2; // Define the y-coordinate of the center of the
panel
    int radius = Math.min(panelWidth, panelHeight) / 3; // Make the radius of the
graph fit inside the panel

    for (int i = 0; i < nodeLabels.size(); i++) {
        double angle = 2 * Math.PI * i / nodeLabels.size(); // Split the nodes into
equal angles in a 360-degree circle
        int x = centerX + (int) (radius * Math.cos(angle)) - diameter / 2; // X-
coordinate on the circle
        int y = centerY + (int) (radius * Math.sin(angle)) - diameter / 2; // Y-
coordinate on the circle
        nodePositions.set(i, new Point(x, y)); // Set the position of each node going
around in a circle
    }
}

private void setMouseListeners() {
    // add mouse listener to handle mouse press events
    addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent e) {
            Point clickPoint = e.getPoint(); // get the position where the user
clicked
            for (int i = 0; i < nodePositions.size(); i++) {
                Point nodePosition = nodePositions.get(i);
                int centerX = nodePosition.x + diameter / 2; // find the center of
the node
                int centerY = nodePosition.y + diameter / 2;
                double distance = clickPoint.distance(centerX, centerY); // measure
the distance from the click to the center of the node
                if (distance <= diameter / 2) { // check if the click is within the
node
                    dragStart = clickPoint; // set the start point for dragging
                    dragIndex = i; // store the index of the node being dragged
                    break; // exit the loop once a node is found
                }
            }
        }
    });
}

```

```

        }

    }

    // reset drag state when mouse is released
    @Override
    public void mouseReleased(MouseEvent e) {
        dragStart = null; // clear the drag start point
        dragIndex = -1; // reset the drag index
    }
});
```

// add mouse motion listener to handle dragging of the node

```

addMouseMotionListener(new MouseAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        if (dragIndex != -1 && dragStart != null) { // check if a node is being
dragged
            Point currentPoint = e.getPoint(); // get the current mouse position
            Point nodePosition = nodePositions.get(dragIndex); // get the
position of the node being dragged
            int horizontalMovement = currentPoint.x - dragStart.x; // calculate
the horizontal movement
            int verticalMovement = currentPoint.y - dragStart.y; // calculate the
vertical movement

            int newX = nodePosition.x + horizontalMovement; // calculate new x
position for the node
            int newY = nodePosition.y + verticalMovement; // calculate new y
position for the node

            int panelWidth = getWidth(); // get the width of the panel
            int panelHeight = getHeight(); // get the height of the panel

            int minX = 0; // minimum x position (left side of panel)
            int minY = 0; // minimum y position (top side of panel)
            int maxX = panelWidth - diameter; // maximum x position (right side
of panel)
            int maxY = panelHeight - diameter; // maximum y position (bottom side
of panel)

            // keep the new position so the node stays within the panel bounds
            newX = Math.max(minX, Math.min(newX, maxX));
            newY = Math.max(minY, Math.min(newY, maxY));

            nodePosition.setLocation(newX, newY); // update the node's position
            dragStart = currentPoint; // update the drag start point for the next
movement
        }
    }
});
```

```

                repaint(); // refresh the GUI to reflect the new node position
            }
        }
    });
}

@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g; // create a new 2D graphics object
    g2d.setColor(Color.BLACK); // set the color for drawing to black
    int tab = menuGUI.getSelectedTab();
    for (int i = 0; i < edges.size(); i++) {
        String edge = edges.get(i);
        char sourceNode = edge.charAt(0); //get the source node from the edge string
        char destinationNode = edge.charAt(1); //get the destination node from the
        edge string

        int sourceIndex = nodeLabels.indexOf(sourceNode); //find the label for the
        source node
        int destIndex = nodeLabels.indexOf(destinationNode); //find the label for the
        destination node

        if (sourceIndex != -1 && destIndex != -1) { //checking that there is a
        position for the source and destination node in the list of positions
            Point p1 = nodePositions.get(sourceIndex); //getting the position of the
            start node
            Point p2 = nodePositions.get(destIndex); //getting the position of the
            end node

            int x1 = p1.x + diameter / 2; //getting the x-coordinate of the center of
            the source node
            int y1 = p1.y + diameter / 2; //getting the y-coordinate of the center of
            the source node
            int x2 = p2.x + diameter / 2; //getting the x-coordinate of the center of
            the destination node
            int y2 = p2.y + diameter / 2; //getting the y-coordinate of the center of
            the destination node

            g2d.setColor(edgeColourList.get(i));
            g2d.setStroke(new BasicStroke(3.0f)); //making the graphics object have a
            thicker stroke to make the edge line thicker
            g2d.drawLine(x1, y1, x2, y2); //drawing the line between the centers of
            the points

            int weight = edgeWeights.get(i); //getting the weight of the arc from the
            list of edge weights

```

```

        int weightX = (x1 + x2) / 2; //setting the x-coordinate of the weight to
be added in the middle, between the two nodes
        int weightY = (y1 + y2) / 2; //setting the y-coordinate of the weight to
be added in the middle, between the two nodes
        g2d.setFont(new Font("Arial", Font.BOLD, 17)); //making the graphics
object have a bold and slightly larger font so the weight is more prominent
        g2d.drawString(String.valueOf(weight), weightX, weightY - 7); //drawing
the weight onto the panel
    }
}

for (int i = 0; i < nodeLabels.size(); i++) {
    Point position = nodePositions.get(i); //getting the position of each node
    char label = nodeLabels.get(i); //getting the label for each node

    g2d.setColor(Color.BLACK); //drawing the node
    g2d.fillOval(position.x, position.y, diameter, diameter); //drawing the node
in the retrieved position
    g2d.setColor(Color.WHITE); //changing the colour to white for the text
    g2d.drawString(String.valueOf(label), position.x + diameter / 3, position.y +
2 * diameter / 3); //adding the text in the node

}
}

public ArrayList<String> getEdges(){
    return edges;
}

public ArrayList<Integer> getEdgeWeights() {
    return edgeWeights;
}

public ArrayList<Color> getEdgeColourList() {
    return edgeColourList;
}

public ArrayList<Character> getNodeLabels() {
    return nodeLabels;
}

public ArrayList<Point> getNodePositions(){
    return nodePositions;
}

}

```





