‘Modelling with Algorithms’ Helper Tool

Computer Science NEA

Name: Leonardo Matteucci
Candidate Number: —
Centre Name: Barton Peveril College
Centre Number: —–

August 13, 2025

Contents

Chapter 1

Analysis

1.1 Statement of Problem

Algorithms are a fundamental concept in further maths, above all the modelling with algorithms topic, but also other parts of the specification that are used everywhere in the real-world including, but not limited to: mapping systems, optimisations, and further maths exams. However, when students are first introduced to this topic, they may struggle to fully understand how an algorithm works and the concepts behind it and instead may just resort to memorising the facts. It is also often hard to find worked solutions to certain example problems and this could lead to them not having a good understanding of how to solve a problem. If a teacher wants to check a student's work it can often be very time consuming and is often prone to errors.

1.2 Background

This problem can be seen in many A level further maths students where they can be seen not quite understanding an algorithm or just having no idea where they went wrong when performing said algorithm. It concerns the Modelling with Algorithms paper found in an OCR MEI B further maths exam. The topic includes many algorithms found in computer science (but not a level computer science). Algorithms may include:

- Simplex: An optimisation algorithm that finds the maximum value of an objective, subject to constraints.
- Dijkstra's algorithm: A minimum path graph traversal algorithm.
- Prim's and Kruskal's algorithms: Finds a minimum spanning tree (a connected graph in which it has the lowest possible total weight) of a weighted graph.
- Quicksort: An efficient sorting algorithm.
- Bin packing: A problem on how to efficiently fit items of certain sizes into the fewest bins (of a specific size).

Students need to perform these algorithms very accurately in the exam as an inaccuracy early in the question tends to lose the candidate a lot of marks. The project may also expand to include other algorithms of A level further maths such as complex numbers or matrices.

Simplex is a linear programming [8] algorithm such that it optimises a linear function given linear constraints. It was developed by George Dantzig who worked for the US Army Air Force during WW2. He helped with planning methods and figured out that there was a way to represent the constraints through linear inequalities. He added an objective function to this and worked out a method to optimise the objective given constraints. [12]

Dijkstra developed this program to demonstrate the capabilities of a new computer called ARMAC. The algorithm isn't overly complicated and he demonstrated it using 64 cities within the Netherlands, finding the shortest path from A to B. [6]

"*What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually, that algorithm became to my great amazement, one of the cornerstones of my fame.*"
- Edsger Dijkstra on developing his algorithm.

1.3 End Users

The program is being created for A level further maths students and teachers. The specific users I have selected are: Serkan Karakus, an A level further maths teacher and 2 students - Garam Lee (student a) and Nathan Richards (student b). The students will help me tailor the design to their needs as they are the primary end users. Serkan will provide an unbiased opinion on what further maths students really need as he teaches the subject and so knows what students want. He will also be able to use the program as a useful tool when teaching the subject or simply when performing certain algorithms.

1.4 Initial Research

1.4.1 Existing, similar programs

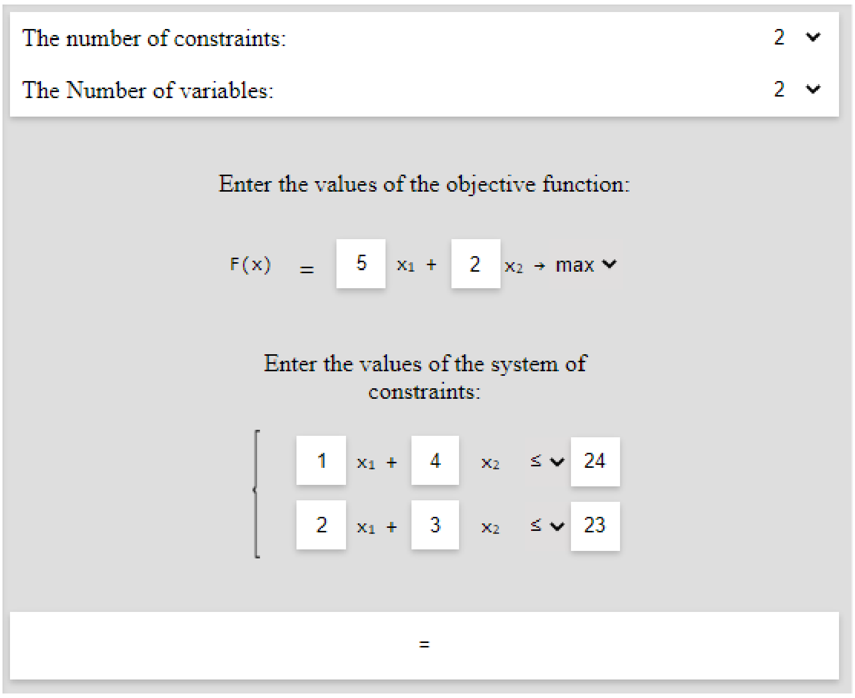
1.4.1.1 Simplex Method (HMLA)

A similar simplex program to the one I am trying to create already exists here:
https://linprog.com/en/main

This website allows a user to easily enter in a Simplex problem using any number of constraints and variables. It then outputs the final answer along with each iteration of the tableau. An expandable tab can be opened that shows detailed workings of how each element in a tableau is calculated.

$$\begin{gathered}\text{Maximise: } 5x + 2y \\ \text{Subject to: } x + 4y \leqslant 24 \\ 2x + 3y \leqslant 23\end{gathered}$$

Figure 1.1: An example simplex problem. (integralmaths.org)



The number of constraints: 2

The Number of variables: 2

Enter the values of the objective function:

F(x) = 5 x₁ + 2 x₂ → max

Enter the values of the system of constraints:

1 x₁ + 4 x₂ ≤ 24

2 x₁ + 3 x₂ ≤ 23

=

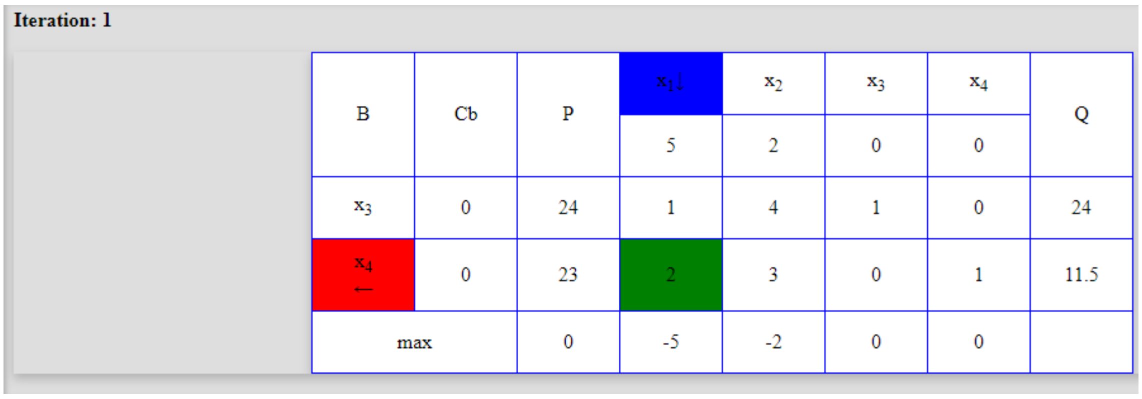
Figure 1.2: The above simplex problem entered into the website.

Pros:

- It has a user friendly interface that allows a linear program to be entered easily.
- The program reaches the correct solution each time, showing not only the result, but also the final value of each variable and the workings needed to achieve this.
  - Each tableau iteration is shown and the pivots used are highlighted in colour. (Figure 1.3)
  - It shows how each element in every tableau is calculated. (Figure 1.4)

Cons:

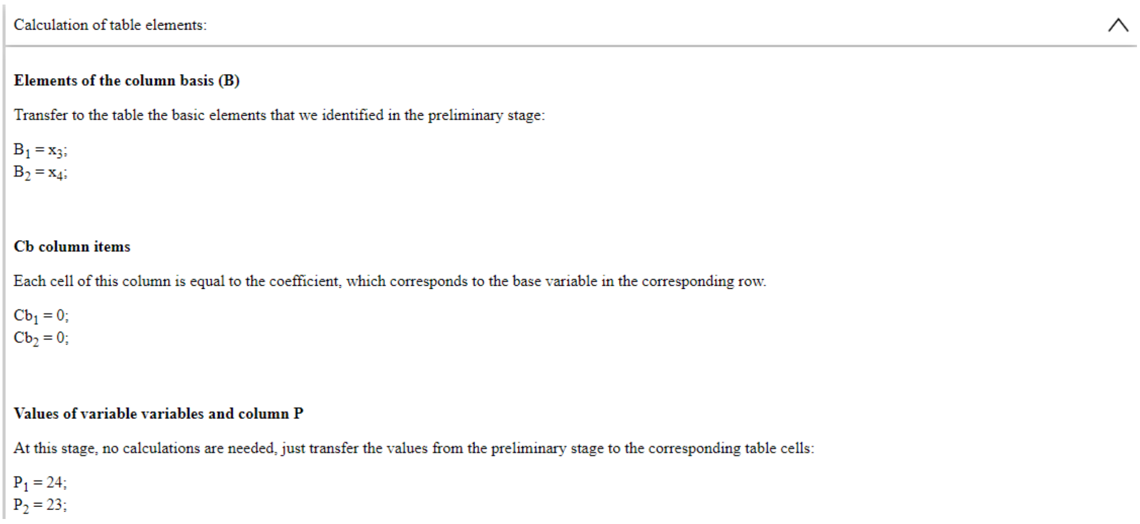
- The tableau is formatted slightly differently to what is expected in a Further maths exam. It should, instead, look like Figure 1.5
- The workings shown for calculating the elements in the tableau appear to be needlessly complicated, beyond the specification and difficult to understand.
- The website only has one algorithm featured in the Modelling with Algorithms Further Maths OCR MEI paper (being Simplex).
- The service can only be accessed online and the site goes down frequently.



Iteration: 1

B	Cb	P	x_1↓	x_2	x_3	x_4	Q
			5	2	0	0	
x_3	0	24	1	4	1	0	24
x_4 ←	0	23	2	3	0	1	11.5
max		0	-5	-2	0	0	

Figure 1.3: The website's tableau of one of the iterations of the problem entered in Figure 1.2



Calculation of table elements:

Elements of the column basis (B)

Transfer to the table the basic elements that we identified in the preliminary stage:

$B_1 = x_3$;
$B_2 = x_4$;

Cb column items

Each cell of this column is equal to the coefficient, which corresponds to the base variable in the corresponding row.

$Cb_1 = 0$;
$Cb_2 = 0$;

Values of variable variables and column P

At this stage, no calculations are needed, just transfer the values from the preliminary stage to the corresponding table cells:

$P_1 = 24$;
$P_2 = 23$;

Figure 1.4: A snapshot of the workings and explanation the website gives for the working out

P	x	y	s_1	s_2	RHS
1	-5	-2	0	0	0
0	1	4	1	0	24
0	2	3	0	1	23

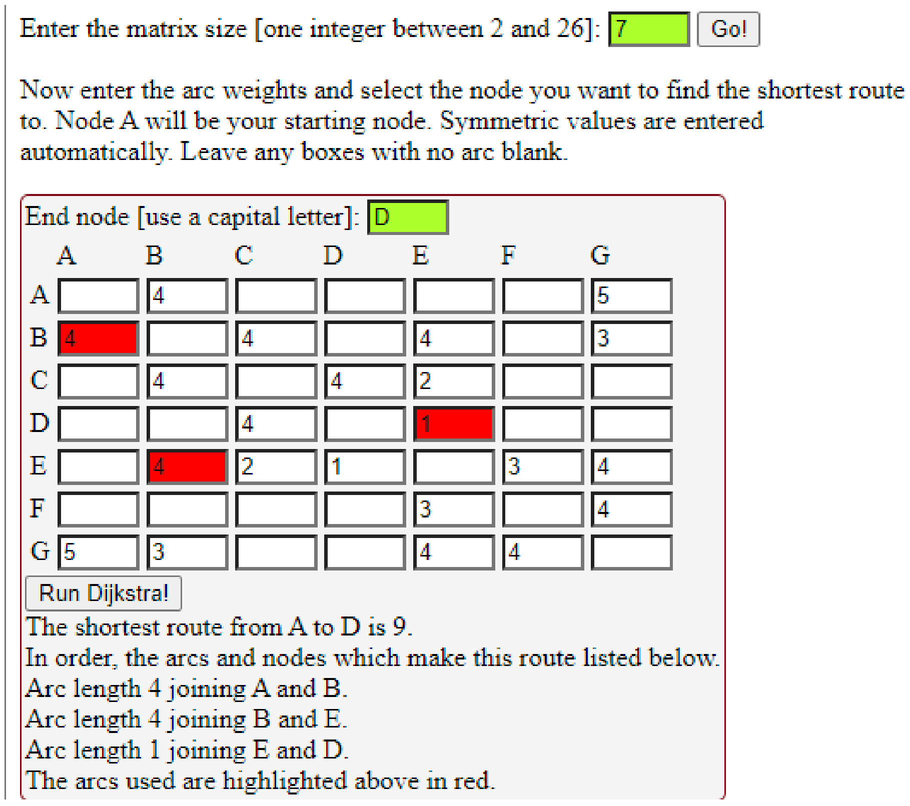
Figure 1.5: Initial tableau of the problem presented in Figure 1.1

What I could apply in my solution:
Although this program performs Simplex slightly differently to what is expected at A level, I can still implement an interface that will be easy to enter in the linear programming problem. Another thing I could add is showing the workings and tableaux at each iteration (if the user desires) and use colour to highlight important parts such as pivot.

1.4.1.2 Prim's and Dijkstra's algorithms (jakebakermaths)

Can be found at:
http://jakebakermaths.org.uk/maths/index.php

This website contains lots of mathematical algorithms but only 2 of which concern the further maths exam: Dijkstra's and Prim's algorithms.



Enter the matrix size [one integer between 2 and 26]: 7 Go!

Now enter the arc weights and select the node you want to find the shortest route to. Node A will be your starting node. Symmetric values are entered automatically. Leave any boxes with no arc blank.

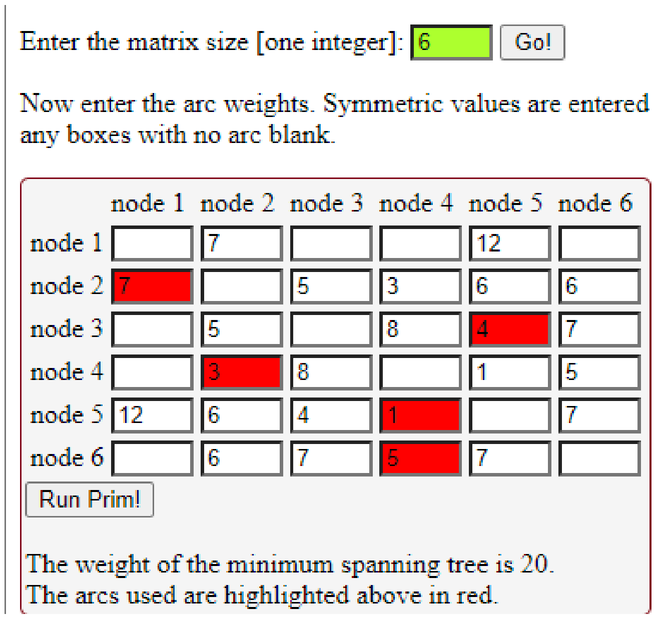
End node [use a capital letter]: D

	A	B	C	D	E	F	G
A		4					5
B	🟥 4		4		4		3
C		4		4	2		
D			4		🟥 1		
E		🟥 4	2	1		3	4
F					3		4
G	5	3			4	4	

Run Dijkstra!

The shortest route from A to D is 9.
In order, the arcs and nodes which make this route listed below.
Arc length 4 joining A and B.
Arc length 4 joining B and E.
Arc length 1 joining E and D.
The arcs used are highlighted above in red.

Figure 1.6: The website running Dijkstra's algorithm



Enter the matrix size [one integer]: 6 Go!

Now enter the arc weights. Symmetric values are entered automatically. Leave any boxes with no arc blank.

	node 1	node 2	node 3	node 4	node 5	node 6
node 1		7			12	
node 2	🟥 7		5	3	6	6
node 3		5		8	🟥 4	7
node 4		🟥 3	8		1	5
node 5	12	6	4	🟥 1		7
node 6		6	7	🟥 5	7	

Run Prim!

The weight of the minimum spanning tree is 20.
The arcs used are highlighted above in red.

Figure 1.7: The website running Prim's algorithm

The website allows a user to enter in the size of an adjacency matrix (how many nodes are in the graph) and then allows the user to easily enter in the distance between nodes.

Pros:

- The boxes are easy to fill in and automatically apply symmetry to the matrix such that an arc doesn't need to be entered twice.
- Arcs used are colour coded and the result is output.

Cons:

- The service can only be accessed online.
- Doesn't show any workings which are essential in an exam:
  - For Prim's algorithm the website doesn't state the order in which arcs are added to the minimum spanning tree.
  - For Dijkstra's algorithm the website doesn't show the workings boxes that appear in an exam:

Order of becoming permanent	*Permanent label*
Temporary labels	

    Which would end up looking like this:

7	35
40 39 36 35	

What I could apply in my solution:
An interface which makes entering in adjacency matrices easy e.g. snapping to cells and symmetrical input. Colour coding of important numbers. A way to specify how big the adjacency matrix is.

1.4.2 Potential abstract data types / algorithms

1.4.2.1 Simplex Algorithm

The first potential algorithm that I could use would be the Simplex Algorithm [12]. The Simplex Algorithm is an algorithm in which a function of variables (e.g. P = 3x + 2y - z) is to be maximised or minimised given constraints (e.g. $x + y \leqslant 7$). The Simplex Algorithm can only handle $\leqslant$ constraints (where each of the variables are $\geqslant$ 0) so a 2 stage Simplex Algorithm could be implemented as well. This would be able to deal with $\geqslant$ constraints (other than $\geqslant$ 0) that the original Simplex Algorithm cannot deal with. This would introduce a second objective function to be minimised as well as two other types of variables (surplus and artificial). It is assumed that all variables are $\geqslant$ 0.
For example, given the following linear programming problem:

$$\begin{gathered}\text{Maximise: } P = 5x + 7y \\ \text{Subject to: } 2x + 3y \leqslant 30 \\ x + y \leqslant 12 \\ x \geqslant 2\end{gathered}$$

The Simplex algorithm would involve reformulating it as so:

$$\begin{gathered}\text{Objective function: } P - 5x - 7y = 0 \\ \text{Subject to: } 2x + 3y + s_1 = 30 \\ x + y + s_2 = 12 \\ x - s_3 + a_1 = 2\end{gathered}$$

Where s_1 and s_2 are slack, s_3 is surplus and a_1 is artificial. As above, the addition of a slack variable removes the $\leqslant$ inequality and the subtraction of a surplus variable with the addition of an artificial variable removes the $\geqslant$. Note: a "=" constraint would result in splitting the equation into 2 inequalities, one with a $\leqslant$ and one with a $\geqslant$ which would then be reformulated into augmented form.
As it is a 2 stage Simplex problem due to the $\geqslant$ inequality, a second objective function must be minimised first. This second objective function can be described as $A = \Sigma a$, where Σa is the sum of all artificial variables. In this case it would be written as:

$$A = -x + s_3 + 2$$

Which would then be rewritten as:

$$A + x - s_3 = 2$$

As an initial tableau it would be presented as:

A	P	x	y	s_1	s_2	s_3	a_1	RHS
1	0	1	0	0	0	-1	0	2
0	1	-5	-7	0	0	0	0	0
0	0	2	3	1	0	0	0	30
0	0	1	1	0	1	0	0	12
0	0	1	0	0	0	-1	1	2

For a 2 stage problem such as the one above (the following paragraph does not need to be, and cannot be executed if it is only a 1 stage problem):
The objective function containing the A must first be minimised. This is to satisfy all the $\geqslant$ inequalities first. An iteration is done by choosing the *most* ***positive*** value from the objective function (top row) as the pivot column, ignoring both objective columns and the RHS. After this (ignoring both objective function rows), a series of tests are done where the RHS column is divided by the corresponding value in the pivot column. In this case (where x is the pivot column), the first test would be $\frac{30}{2}$ which results in 15, the second test would be $\frac{12}{1}$ which equals 12 and the final test would be $\frac{2}{1}$, equalling 2. If a divide by zero error occurs, it is simply ignored. The smallest non-negative value (including 0) is chosen to be the pivot row.
The rest of the step involves making the pivot column variable (in this case x) into a *Basic Variable*. The table is then iterated upon by using this pivot row to make a new row of each type. The first thing that must happen is to create a new pivot row. This can be achieved by finding the pivot value (the intersection of the pivot column and row) and dividing it by itself to make 1. The rest of the pivot row is then also divided by the pivot value. In this case, it is 1 so there is no change to the row as everything is effectively divided by 1. The rest of the values in the pivot column must all turn to 0 by doing:

$$(\text{Value to turn to 0}) + n(\text{Pivot value}) = 0$$

and finding n. The rest of the row is also then transformed by using each value and adding n lots of the corresponding value in the pivot row to it. For example, the second to bottom row is transformed by finding n: $1 + n \times 1 = 0 \implies n = -1$. Now lets look at the RHS column: the new RHS column would be calculated by doing $12 + n \times 2 = 12 - 2 = 10$. This same process would happen for the rest of the row. The entire step would happen again for each and every row, resulting in the second iteration looking as follows:

A	P	x	y	s_1	s_2	s_3	a_1	RHS
1	0	0	0	0	0	0	-1	0
0	1	0	-7	0	0	-5	5	10
0	0	0	3	1	0	2	-2	26
0	0	0	1	0	1	1	-1	10
0	0	1	0	0	0	-1	1	2

As seen in the tableau above, x is now a Basic variable as there is one occurrence of the number 1 and the rest of the values in the column are 0. As there are now no longer any positive values in the top row (objective function) apart from A, the table can be reduced by removing all columns and rows to do with artificial variables - in this case it is the top row and the columns of A and a_1. The reduced tableau looks like this:

P	x	y	s_1	s_2	s_3	RHS
1	0	-7	0	0	-5	10
0	0	3	1	0	2	26
0	0	1	0	1	1	10
0	1	0	0	0	-1	2

Now the 2 stage problem has been converted into a 1 stage problem, and the 1 stage Simplex Algorithm can be performed.

Instructions for a 1 stage problem:
After the problem has been converted into tabular form, the tableau is iterated upon until there are no longer any negative values in the objective row. Now (unlike the 2 stage problem) the objective function must be maximised. This involves performing iterations until there are no longer any negative values in the objective function. An iteration is performed exactly the same as described in the 2 stage Simplex instructions but the key difference is that the pivot column is the *most* ***negative*** value in the objective function row.
For the example case, the next iteration would have y as the pivot column and the second row as the pivot row. The next tableau would look as follows:

P	x	y	s_1	s_2	s_3	RHS
1	0	0	$\frac{7}{3}$	0	$-\frac{1}{3}$	$\frac{212}{3}$
0	0	1	$\frac{1}{3}$	0	$\frac{2}{3}$	$\frac{26}{3}$
0	0	0	$-\frac{1}{3}$	1	$\frac{1}{3}$	$\frac{4}{3}$
0	1	0	0	0	-1	2

The next and final iteration looks as follows:

P	x	y	s_1	s_2	s_3	RHS
1	0	0	2	1	0	72
0	0	1	1	-2	0	6
0	0	0	-1	3	1	4
0	1	0	-1	3	0	6

Now that the algorithm is finished, we need to identify the Basic and Non-Basic variables. Basic variables are where there is an instance of a 1 in the column and the rest of the column are 0s. In the above example the basic variables are:

- $P = 72$

- $x = 2$
- $y = 6$
- $s_3 = 4$

The Non-Basic variables are: s_1 and s_2. As they are Non-Basic, they value 0.
This all means that P is maximised at 72, where $x = 2$ and $y = 6$.

1.4.2.2 Dijkstra's Algorithm

Another potential algorithm would be the graph traversal algorithm: Dijkstra's Algorithm [3]. This algorithm involves a weighted graph which needs to be traversed in the shortest possible path from A to B. This could be implemented using the abstract data type of a graph. The algorithm is a greedy algorithm [4] such that it selects the best option at the moment and doesn't reconsider previous nodes. This has the knock on effect that it won't work with negative weights. Each node could have the information that would be displayed in Dijkstra's Algorithm. Dijkstra's Algorithm works by finding the shortest distance to each of the vertices from the start. When considering a node, it looks at all the other nodes that connect to it (that have already had their shortest path found) in order and adds the shortest path to it. The algorithm essentially makes a tree where the start is the root node.

Dijkstra's Algorithm works by using a Priority queue that stores nodes. The node with the shortest distance from the start is dequeued each time the algorithm loops. A list of visited nodes is also stored. The algorithm then works as follows:

1. The distance from the source node to all other nodes is set to infinity. The distance to the source node is set to 0.
2. The source node is then Enqueued into the priority queue.
3. While the queue is not empty:
    (a) Dequeue the node with the current shortest distance to the source node.
    (b) If the node has been visited, go to the next iteration (continue).
    (c) Add the node to the list of visited nodes.
    (d) For all neighbours of the node:
        i. Calculate the distance to the neighbour through the current node.
        ii. If it is smaller, update the node and add it to the priority queue.

I would store the priority queue as an abstract data type:
The priority queue would work as a Min Heap Binary Tree, where the lowest value is always stored at the root. This property holds true for all sub-trees. The tree is also space efficient, where it is an almost complete tree and only the last 2 rows of the tree can have missing children (< 2). The i^{th} node would be indexed by arr[i], and the other nodes associated to it by:

Parent Node	arr[(i - 1)/2]
Left Child Node	arr[2 * i + 1]
Right Child Node	arr[2 * i + 2]

As it is functioning as a priority queue, there is only the need to be able to add an element and the ability to delete (and get) the first element. To insert (enqueue) an element:

1. Insert the element at the bottom of the tree
2. Until the parent element is less than or equal to the current element:
    (a) Swap the element with the parent element (sift up)

To get and delete (dequeue) the minimum element:

1. Store the first element
2. Replace the first element with the last element
3. Delete the last element
4. The min-heap property is broken so the first element must be put into the correct place
5. Heapify(): - essentially the opposite of the insert function; sift down
    (a) While the element is larger than both its children:
        i. If the element is smaller than both the children, stop
        ii. Otherwise choose the smallest of the two children and swap with it

I would store the graph as an abstract data type:
The graph abstract data type would contain edges stored in an adjacency matrix (2D array). These would then be converted by Dijkstra's Algorithm into nodes (not within the graph ADT as other algorithms also utilise a graph). To randomly generate a problem, a graph would need to be able to be randomly generated, this can be done by creating a random walk, using a uniform spanning tree algorithm [9]. This works by choosing a random node to be in the tree. Then another random node is chosen and if it isn't in the tree, it is connected to the tree by adding a connection to another node in the tree. There are no loops as it is a tree.

1.4.2.3 Other Potential Algorithms

Some other potential algorithms that I could implement are the minimum spanning tree (MST) algorithms of Kruskal's and Prim's. These involve finding the combination of edges so that it gets the lowest possible weight of a graph while connecting all the nodes in a graph.

- Prim's algorithm [11] starts at a node and initialises a tree. It adds the shortest edge not within already within the tree (that connects to the tree) to the tree - not creating a cycle (as this would no longer be a tree). This step is repeated until all nodes are connected.
- Kruskal's algorithm [7] involves choosing the smallest edge until the MST is complete. If the next smallest edge creates a cycle, the algorithm simply ignores it.

Another potential algorithm is the quick sort algorithm [2]. The algorithm works by choosing a pivot (generally the start element) and placing it in the correct place in the list, without sorting other elements. This splits the list into two either side of the pivot and each of these sub-lists would have the above step applied to it - a divide and conquer algorithm.
The algorithm is typically implemented recursively. However, this would present the problem such that it would be difficult to display how the exam board wants it. Therefore it may be a good idea to implement it iteratively.

1.4.3 First interview

1.4.3.1 Interview with teacher:

1. What parts of the MwA topic do you find students struggle most with?
   "Simplex. They make very simplex mistakes like sign errors and using the wrong pivots etc. And if you have long Simplex algorithm which needs you to do 2-3 iterations, thats where students make a lot of the mistakes."

2. So how would you want a program such as this to accept inputs?
   "Yeah, so inequalities. I want that program to change it to the equations with the variables inside and then and then in the initial table and in that initial table you need to be identifying the pivot points, pivot rows, etc."

3. Would a graphical representation of a 2 dimensional problem be helpful at all?
   "Yes, definitely. We've we've done questions with graphical ones where we use the objective lines like a ruler and then move on to find the maximum with the with the coordinates which we sub into the equation."

4. So it would be useful to show each point tested?
   "Yes, it will be very useful, especially for a 2 dimensional problem."

5. I could make it such that the program is able to output some LaTeX code that you would be able to convert yourself. This would be able to neatly display the Simplex tableaux.
   "You can get the graph of the problem as well. And you can also animate in latex, which you might try and animate. It will look nice, right? It's like the objective line, animate it moving up. Or shading regions animated 1 by 1. [...] I use latex myself. I think it will be very useful especially students able students who are going to do go to a university which who are going to do masters or or or PhD. They would need to type in their research into LaTeX. And then it will convert into PDF."

6. How would you feel about an extra additional part of the program that will have like example questions and shows how the program works? Would that be useful?
   "That would be very useful. When students use it they can see one example and when they get it they can put in their own one."

7. Are there any other specific features or functionalities you would want to see in software?
   "I want students to know where Simplex is used in real life like maximum profit or minimum number of workers. Also visualisations where possible would help students understand better."

8. Are there any other Modelling with Algorithms topics that students struggle with?
   "I think Dijkstra's is another one students find difficult. If you make a simple mistake, the rest becomes terrible."

From this interview it seems like visualisations are of fundamental importance such as showing the points tested in a problem. It would be good to show how values are achieved in each step of a Simplex problem and highlighting how pivots are chosen. The program should be able to take in constraints as inequalities. I will likely include an output to LaTeX format and potentially animate a problem. Additionally, showing the paths used to achieve values in Dijkstra's algorithm may be of use.

1.4.3.2 Interview with students (*a* and *b*):

1. What parts of the MwA topic do you find students struggle most with?
   (a) "I find the multi step aspect of many high mark MwA questions to be very difficult - despite knowing how to apply the algorithm, the amount of steps makes it difficult to flawlessly execute. This includes 2 stage simplex, graphical linear programming and critical path analysis on large graphs. Additionally, novel problems based on known algorithms are also quite difficult for me - such as trying to figure out the graph from a given Dijkstra application (with edge weights redacted)"
   (b) "I generally struggle with simplex, especially 2-stage simplex, since there are many steps to solving each problem. Dijkstra's algorithm can also be challenging from time to time depending on the question, notably performing Dijkstra's without a graph."
2. You [both] mentioned Simplex. What parts of Simplex workings are most important to you?
   (a) "Simplex is a very difficult algorithm, much due to its opaque nature, the method involved seemingly unconnected from the problem, and therefore I'd like to see a wide variety of steps. I think showing the method of turning constraints into the initial tableau is a must - it is very easy to get lost in simplex if one does not fully understand even how to start. Additionally choosing the pivot column row would be helpful - especially to demonstrate more unusual cases such as when the RHS has entries of 0 in it on the selected pivot row. In 2 stage simplex the transformation of the first stage's final tableau to the second stage's first tableau would be helpful, as I often confuse which columns/rows to delete."
   (b) "The majority of simplex workings are important. Notably, ratio testing to find the pivot, and row operations. As I tend be fine with formulating the constraints of the simplex problem, and converting them to augmented form, setting up the simplex is typically not much of an issue for me."
3. How would you want a Simplex program to accept inputs?
   (a) [Student *a* has answered this in the previous question]

   (b) “Inputting directly into the initial tableau would be very important to have. Inputting the constraints would also be convenient to have, but it is not as necessary.”

4. Would a graphical representation of a 2 dimensional problem be helpful at all or do the constraints present themselves in an easy enough to understand manner? Would it be useful to display what points were tested in the process?

   (a) “For a 2 dimensional problem, a graphical representation would definitely be helpful, especially since 2D graphical linear programming is a common question in of itself. Displaying the testing of each point in turn would be useful - especially since it can be confusing to figure out which points of the region give the optimum values. Additionally I would like to see the graphical representation demonstrate integer linear programming too - showing what happens when the optimum values are not integers, especially since this can make it confusing what points need to be tested.”

   (b) “Yes. While not essential, it would most likely be helpful to have the option to display the problem graphically. Maybe there could be an option to visualise what has happened graphically after each iteration of simplex? That would probably make what is actually happening when doing simplex easier to understand.”

5. Would the option to also have a text file with LaTeX code to display this be necessary?

   (a) **“surely if you’re doing latex you use some C# library to display the graphs to the user”**

   (b) “Yes.”

6. How would you feel about an additional part of the program to be able to have an extra tutorial-like set of problems to show how the program works?

   (a) “This definitely is a good idea! A modelling with algorithms program is a very novel one, and I feel like I would not be able to fully pick up how to use it, however well made it is, at the start - a set of simple tutorial problems that I could always refer to when I can’t figure out how to use a certain feature of the program would solve that.”

   (b) “I think that having tutorial-like sections with predefined problems would be helpful. When I first use the program, it would be quite nice to be guided through how everything works, and this would be especially useful if the topic is not very familiar.”

7. Are there any specific features or functionalities you would like to see in the software?

   (a) “I would definitely like to see randomly generated questions - need to ramp up my practice on the longer algorithm questions - it would be useful if this could come with a difficulty select function, so I could start with some easy questions and then slowly work up to harder and harder questions. Additionally I would like to see a feature that allows me to customise the experience of the further maths helper, such as menu colours and font colours, so that I can make the further maths helper feel just right (for me).”

(b) "Maybe having an option to display a visualisation of simplex after each iteration would be quite useful, since it may help deepen understanding as it provides context to what is actually happening during the simplex process. It would be preferred if there a some randomly generated problems, which can be used for practise. It would also be nice if the user has a large amount of control over these randomly generated problems, for example the user being able to control the amount of nodes of a shortest path problem."

These interviews appear to struggle the most with Simplex and Dijkstra's algorithm. It appears that the many steps involved in each algorithm tends to make it much more difficult to earn all the marks. Student *a* would like to see Simplex constraints as an input format but student *b* wants to be able to enter a tableau and so being able to enter both may be an important part of the product. Both students would like to see the testing of each point of a 2D problem and so a graph could be shown for every iteration of Simplex. Student *a* also said that a graphical representation of integer linear programming could be added. While not essential, it could potentially be added to the program. Converting this to LaTeX code seems like an option that Student *b* would like. However, Student *a* mentioned something interesting that hadn't previously occurred to me - could the program produce an image alongside the console. This will be further researched later, taking into account the opinions of my other end users.
Student *b* talks about how they would like the program to be able to output randomly generated problems. This could be implemented into my solution as the program should already be able to solve the problems they're asking for and it would act as further practise for someone taking the exam.

1.4.3.3 Questionnaire

To help support these answers I conducted a 30 person questionnaire:

1. The first question involved ranking algorithms (Simplex / 2 Stage Simplex, Dijkstra's Algorithm (table), Prim's or Kruskal's (table), Quicksort (letters/numbers), Bin Packing) where 5 was the most difficult to execute and 1 is the least difficult.

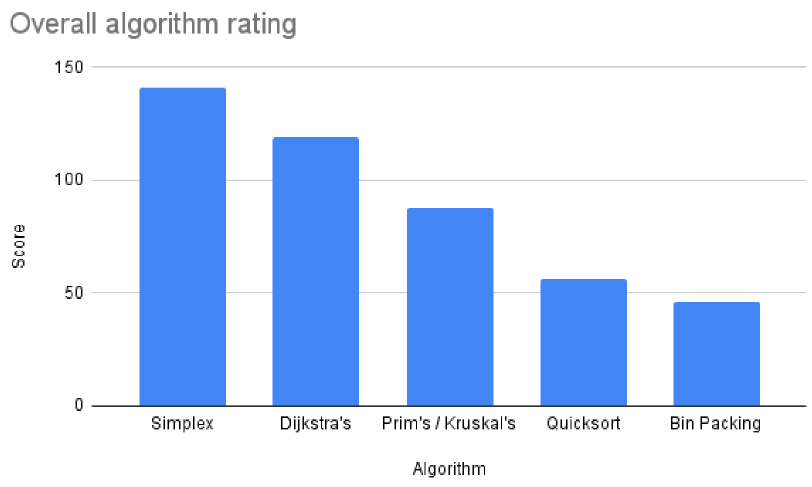


Figure 1.8: Accumulative rating of each algorithm

Figure 1.8 clearly shows that people on average find Simplex the most difficult where it almost reaches the maximum possible score of 150. The graph also shows how people tend to find tabular algorithms more difficult than ones not involving tables.

2. The second question asked about other potential algorithms that a user may want.
A popular response to this was to include other sorting algorithms such as bubble sort and shuttle sort. However, although these can come up in an exam, they are extremely rare and will have the instructions to perform them. This makes these algorithms much less relevant and important than quick sort.

3. Another question that I posed was what workings were most important to people when performing Simplex (Figure 1.9).

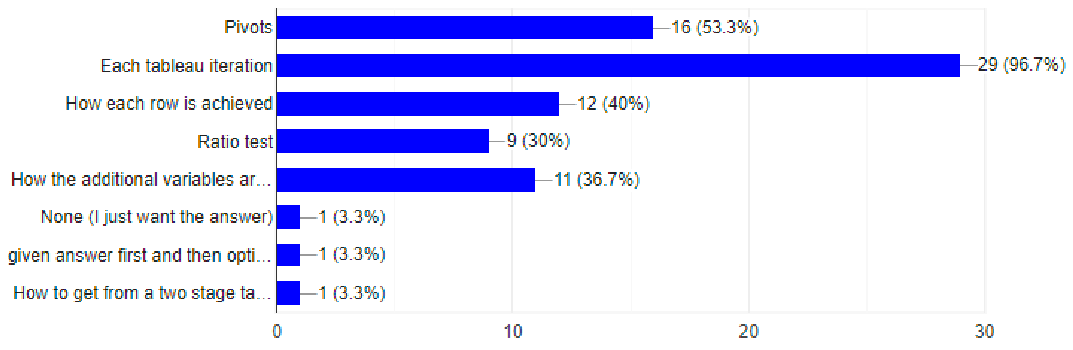


Figure 1.9: Poll on what are the most important workings are for Simplex

From this it is clear that it is vital to include each tableaux iteration along with the pivots and the workings on how to achieve each row. It is also apparent that a selection of students find it difficult to reformulate the problem into an initial tableau so seeing how additional variables (slack, artificial, surplus) are created is of importance.
There was also a suggestion of hiding the workings and then having the option to show them which would be ideal to add if I have time.

4. Some other suggestions that I received from the questionnaire include:

   (a) Displaying the minimum spanning tree for Prim's/Kruskal's.

   (b) A way to look back at previous questions you have done recently.

   (c) Using colour-coding to highlight the most important steps.

   (d) Randomly generated questions with "nice" numbers/solutions

This questionnaire was sent out and completed by my entire further maths class, along with some students completing it from other further maths classes. This may have produced a slightly biased result as it is an example of Opportunity Sampling and so some of the results may lean towards what my class tends to find difficult. If I were to do this again I would attempt to do a random sample of some sort with the whole cohort.

1.4.4 Key Components

Following the interviews, I have decided on the following key requirements:

1. Questions will be able to be randomly generated.
2. Simplex questions can be solved correctly.
3. 2D questions will be able to displayed graphically.
4. Dijkstra's Algorithm questions can be solved correctly.

1.5 Further Research

After some further investigation prompted by student a's response to an output to LaTeX format, I discovered a C# library called "ScottPlot". This would make it so that, instead of parsing the equations and tableaux into LaTeX format, I'd need to turn each equation into a C# function instead. The library works such that I can plot 2D functions in a windows form. I can find the basic variables at each iteration of the tableau and these represent the coordinates that would be plotted. To highlight the feasible region I could find the intersections of all lines and test each intersection to see whether or not they satisfy all the constraints. The coordinates which do satisfy the constraints would then make up the polygon that represents the feasible region.

1.5.1 Prototype

The code for the prototype can be found in Appendix A.
For the prototype I created a program which can perform the Simplex algorithm, given the initial tableau in a 2D array. For example, given the following linear programming problem:

$$\begin{gathered}\text{Maximise: } P = 5x + 7y \\ \text{Subject to: } 2x + 3y \leqslant 30 \\ x + y \leqslant 12 \\ x \geqslant 2\end{gathered}$$

Note: this problem has been reformulated into augmented form in section 1.4.2.1.
As an initial tableau it would be presented as:

A	P	x	y	s_1	s_2	s_3	a_1	RHS
1	0	1	0	0	0	-1	0	2
0	1	-5	-7	0	0	0	0	0
0	0	2	3	1	0	0	0	30
0	0	1	1	0	1	0	0	12
0	0	1	0	0	0	-1	1	2

The top line is the second objective function that was constructed and the second line is the primary objective function. The other lines are the constraints.
This would then be input into my code as:

```
variables = new string[] { "A", "P", "x1", "x2", "s1", "s2", "s3"
    , "a1", "RHS" };
values = new double[,] {
{ 1, 0, 1, 0, 0, 0, -1, 0, 2, },
{ 0, 1, -5, -7, 0, 0, 0, 0, 0 },
{ 0, 0, 2, 3, 1, 0, 0, 0, 30 },
{ 0, 0, 1, 1, 0, 1, 0, 0, 12 },
{ 0, 0, 1, 0, 0, 0, -1, 1, 2 } };
```

Where "variables" is a parallel array to the 2D array "value" which stores the tableau.
In the prototype code there is also the option to run a (less complicated) 1 stage simplex problem, the same one as found in Figure 1.1.

The prototype went well as it could perform full 2 stage Simplex which at the time I thought would be the hardest part of the project. However, it made me realise that parsing an input could be the most difficult part of getting a user to use the program. This is because there are so many different things that I need to conisder: Are they using a tableau or equations? How big is the tableau? How many equations? What kind of equations? etc.
On top of this I'd need to consider how to parse an input from a user in an easy manner (e.g. using arrow keys to navigate an equation or table and then being able to type in numbers accordingly).

However, a core part of the program (and the most difficult algorithm) was implemented in the prototype. As I was able to achieve this, it is evident that the project is manageable within the available time.

1.5.2 Second Interview

I conducted a short second interview over the internet, attaching Figure 1.10 for context:
I have discovered that it may be possible to plot a 2D problem using windows forms after the program has run. Would you like this to be a potential feature in the code?

Teacher: Yes, I would love the visual explanations 2D would be perfect.

Student b: Hark! You've done it! Yes, utilising windows forms to display the graph while program is running would be a useful and convenient feature!

The teacher added that he would still "love to see LaTeX format" but that it is optional. As both the students found that the interactive graph would be much preferred, it seems unlikely that I will add this functionality.

I currently have the precision set to display to 2 d.p. What kind of precision do you think would be the most useful when displaying information?

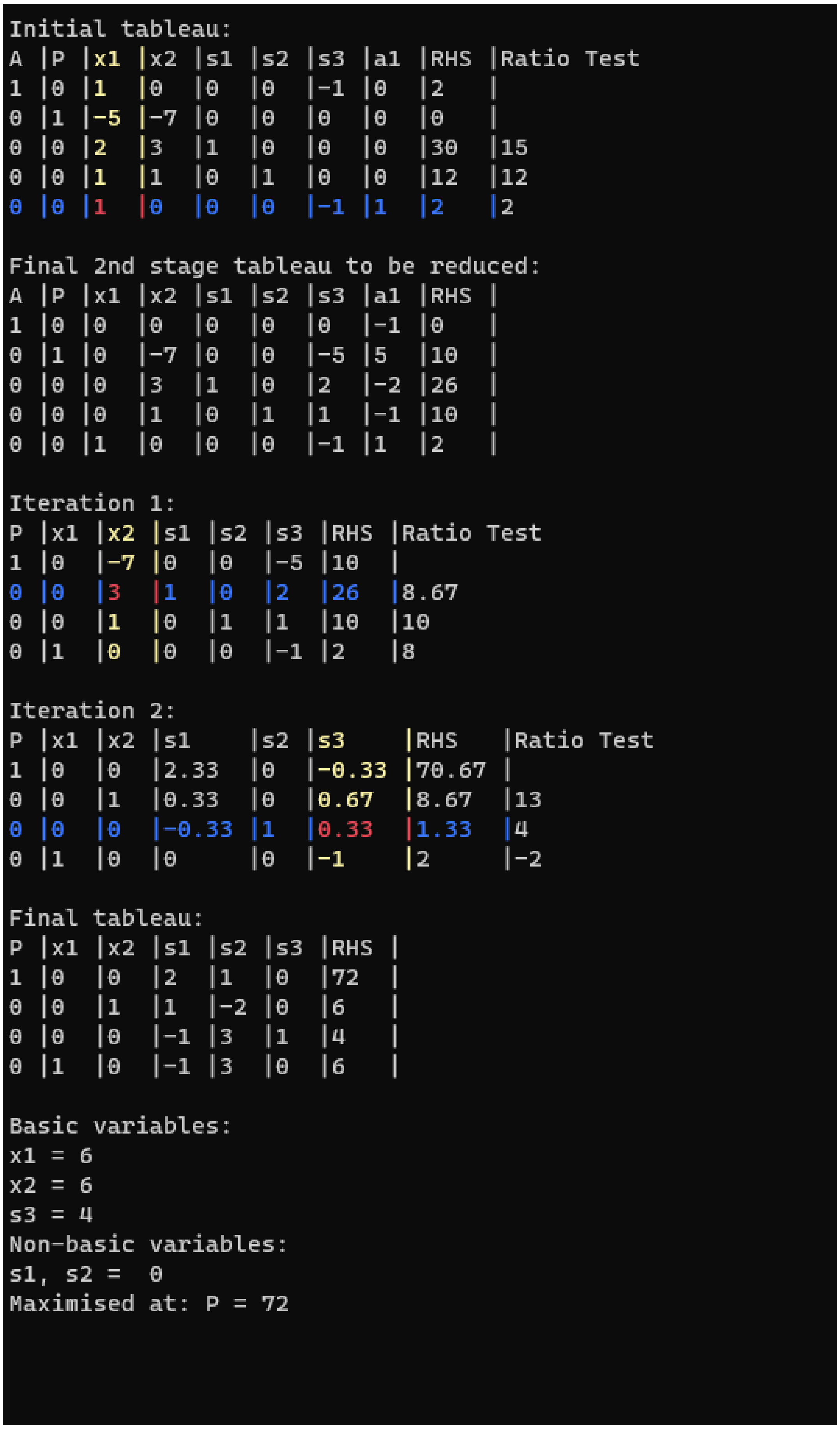
Teacher: 3 s.f. would be perfect.

Student a: Probably 3 sig fig. Fractions might be nicer?? Although that might display badly.

Student b: 12. For maximum precision.

The consensus for this seems to be to display the value to 3 significant figures. 12 decimal points would be unnecessarily precise and could be wrong sometimes due to how computers handle numbers (precision error). Handling fractions might be a good idea but it would be difficult to include all the same functionality that a C# double can store and, as *Student a* mentioned, it wouldn't display very nicely.

On another note, the teacher also wanted to say that he would like to know how each tableau is made on every iteration.



```
Initial tableau:
A |P |x1 |x2 |s1 |s2 |s3 |a1 |RHS |Ratio Test
1 |0 |1  |0  |0  |0  |-1 |0  |2   |
0 |1 |-5 |-7 |0  |0  |0  |0  |0   |
0 |0 |2  |3  |1  |0  |0  |0  |30  |15
0 |0 |1  |1  |0  |1  |0  |0  |12  |12
0 |0 |1  |0  |0  |0  |-1 |1  |2   |2

Final 2nd stage tableau to be reduced:
A |P |x1 |x2 |s1 |s2 |s3 |a1 |RHS |
1 |0 |0  |0  |0  |0  |0  |-1 |0   |
0 |1 |0  |-7 |0  |0  |-5 |5  |10  |
0 |0 |0  |3  |1  |0  |2  |-2 |26  |
0 |0 |0  |1  |0  |1  |1  |-1 |10  |
0 |0 |1  |0  |0  |0  |-1 |1  |2   |

Iteration 1:
P |x1 |x2 |s1 |s2 |s3 |RHS |Ratio Test
1 |0  |-7 |0  |0  |-5 |10  |
0 |0  |3  |1  |0  |2  |26  |8.67
0 |0  |1  |0  |1  |1  |10  |10
0 |1  |0  |0  |0  |-1 |2   |8

Iteration 2:
P |x1 |x2 |s1    |s2 |s3    |RHS   |Ratio Test
1 |0  |0  |2.33  |0  |-0.33 |70.67 |
0 |0  |1  |0.33  |0  |0.67  |8.67  |13
0 |0  |0  |-0.33 |1  |0.33  |1.33  |4
0 |1  |0  |0     |0  |-1    |2     |-2

Final tableau:
P |x1 |x2 |s1 |s2 |s3 |RHS |
1 |0  |0  |2  |1  |0  |72  |
0 |0  |1  |1  |-2 |0  |6   |
0 |0  |0  |-1 |3  |1  |4   |
0 |1  |0  |-1 |3  |0  |6   |

Basic variables:
x1 = 6
x2 = 6
s3 = 4
Non-basic variables:
s1, s2 =  0
Maximised at: P = 72
```

Figure 1.10: The prototype running a 2D, 2 stage Simplex problem.

1.6 Objectives

1.6.1 Main Objectives

1. The program should be able to generate practice questions of any of the types of problems within the program
   - 1.1 The questions should be randomly generated
   - 1.2 The questions should have various difficulties
   - 1.3 The questions should be able to be solved by the program
2. Simplex Algorithm
   - 2.1 Given any maximising Simplex Algorithm problem, the program will produce the correct optimised result
   - 2.2 Given a 2 stage Simplex Algorithm problem, the program will produce the correct optimised result
   - 2.3 The program is able to accept all constraints ($\leqslant, \geqslant, =$)
     - 2.3.1 $\leqslant$ is reformulated to an equation with 1 slack variable
     - 2.3.2 $\geqslant$ is reformulated to an equation with 1 artificial and 1 surplus variable
     - 2.3.3 $=$ is reformulated to 2 equations: 1 which is $\leqslant$ and then reformulated and the other which is $\geqslant$ and then reformulated (as above)
   - 2.4 The program is able to accept an initial tableau
   - 2.5 The program shows workings
     - 2.5.1 Colour codes pivots (row, column)
     - 2.5.2 Shows the ratio tests when calculating pivot rows
     - 2.5.3 Basic / non-basic variables have an option to be shown
     - 2.5.4 An option to display how a row is calculated in each tableau
     - 2.5.5 A 2D problem (entered through constraints or randomly generated; **not** entered through a tableau) can be represented visually
       - 2.5.5.1 The feasible region will be highlighted
       - 2.5.5.2 The points where each iteration "reached" should be visible
   - 2.6 Non-bounded regions will be discovered.
   - 2.7 Values in tableaux are rounded to 3 significant figures.
3. Dijkstra's Algorithm
   - 3.1 Given an adjacency matrix, the program should be able to produce the shortest distance from a node of the user's choosing to all other nodes.
   - 3.2 The user should be able to input a matrix using their arrow keys.
     - 3.2.1 The user should be able to input whether the graph is directed or not.
     - 3.2.2 If the graph is undirected, automatically apply symmetry to the input matrix
   - 3.3 The program will display the boxes with workings, as displayed in the exam.

4. Add a tutorial-like feature that explains how the program works.

5. Add a menu to navigate the application

1.6.2 "If I have time" Objectives

These are objectives that aren't necessary for the program but would benefit it greatly.

1. Add an option to use Prim's and Kruskal's Algorithms,

   1.1 The input would be handled the same as Dijkstra's Algorithm but graphs can only be undirected

   1.2 For Prim's Algorithm, the order that edges are chosen is shown.

   1.3 For Kruskal's Algorithm, the order that edges are chosen is shown, along with whether or not the edge has been added to the Minimum Spanning Tree.

2. A minimisation option for Simplex Algorithm

3. An integer linear programming option for Simplex Algorithm

4. Add an option to use quick sort

5. Add an option to store the Simplex 2D graph as an image locally.

1.7 Modelling

1.7.1 Flow chart

Figure 1.11 is the flowchart of the main program and how it will function. It doesn't contain the algorithms that I will use (e.g. Dijkstra's) and is just a layout of how the classes will be used. I intend to have each of the algorithm question generators implement the same interface so that they can be accessed by the same subroutine.

1.7.2 Class Diagram

Figure 1.12 is the initial layout for the classes. It is very basic but it allows me to add/remove things very easily when I create the program if needs be as it can be added to the diagram. Note that, although I model the program with the Minimum Spanning Tree algorithms, they are not necessary to meet my objectives.

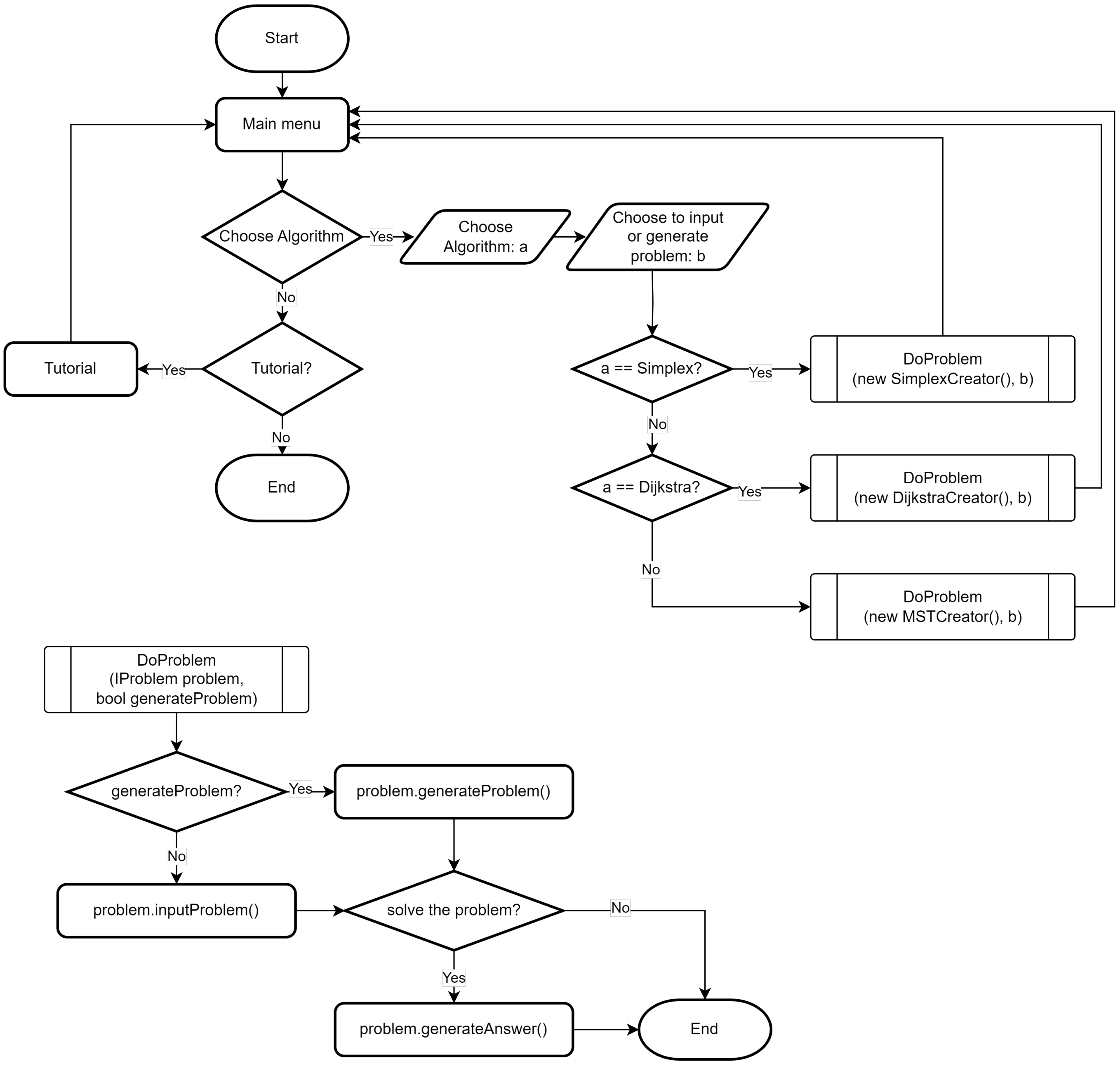

Start
Main menu
Choose Algorithm
Yes
Choose Algorithm: a
Choose to input or generate problem: b
No
Tutorial
Yes
Tutorial?
No
End
a == Simplex?
Yes
DoProblem (new SimplexCreator(), b)
No
a == Dijkstra?
Yes
DoProblem (new DijkstraCreator(), b)
No
DoProblem (new MSTCreator(), b)
DoProblem (IProblem problem, bool generateProblem)
generateProblem?
Yes
problem.generateProblem()
No
problem.inputProblem()
solve the problem?
No
Yes
problem.generateAnswer()
End

Figure 1.11: Flowchart of the main program

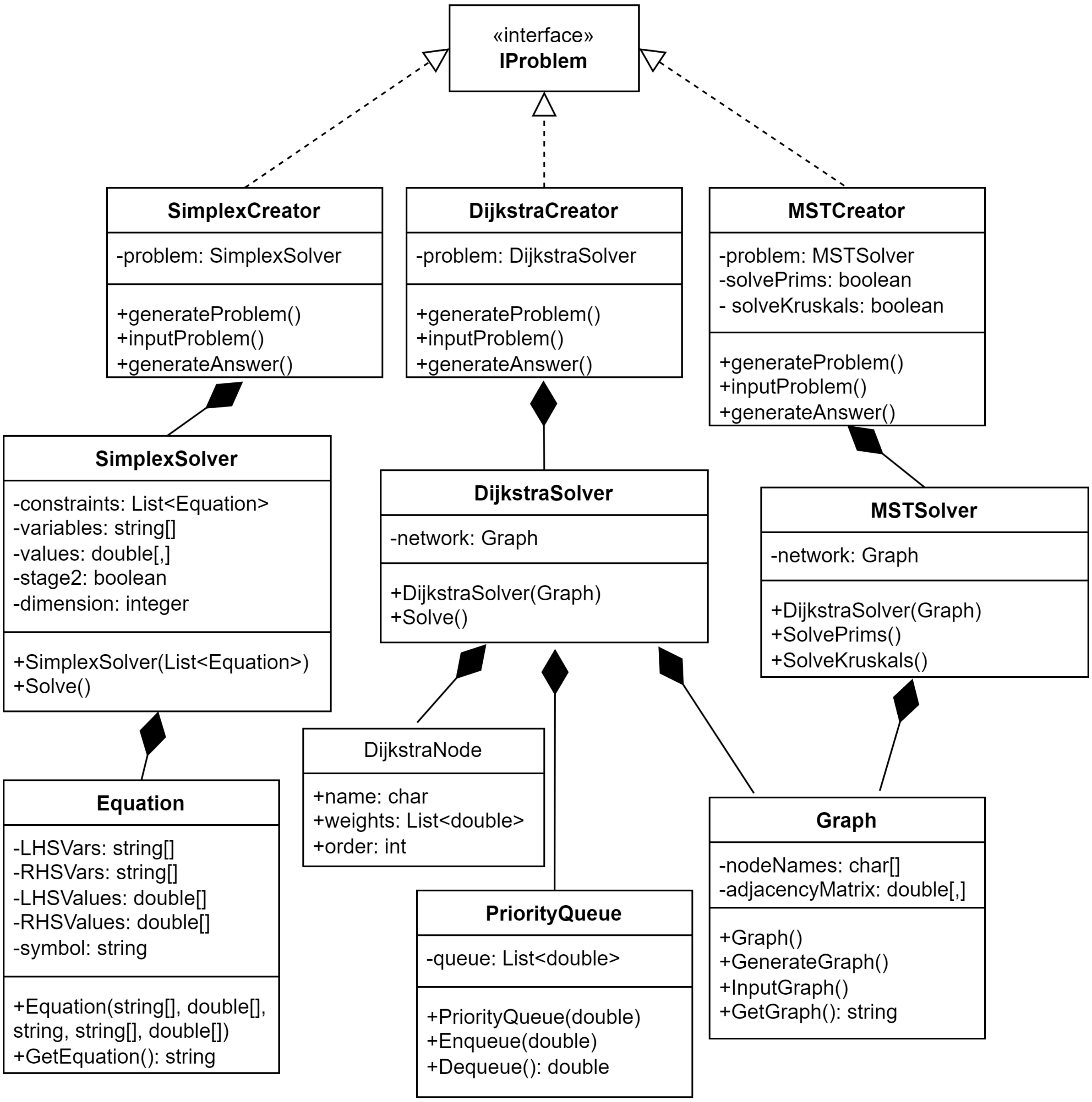


Figure 1.12: Initial idea for the layout of classes

Chapter 2

Design

2.1 High level overview

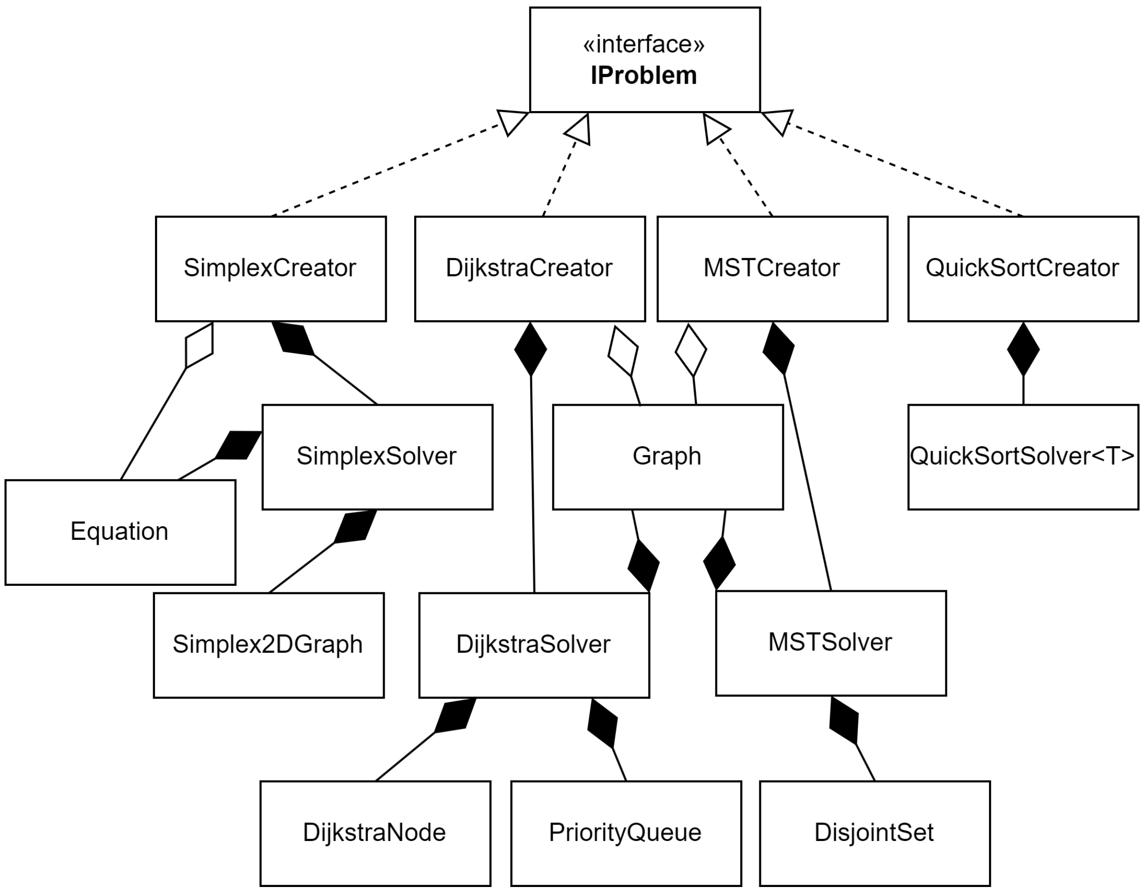


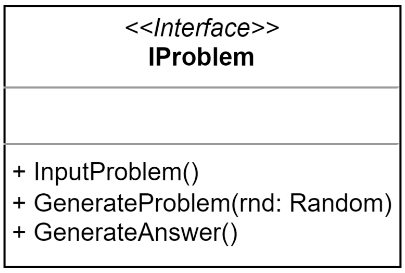
Figure 2.1: How I would use OOP to layout my program.

Alongside these classes I will also implement:

- A menu class that will be utilised in the main program as well as within some option settings when creating a problem. (Covered in Section 2.4.1)
- A sort of menu which allow you to scroll up and down between different options. (Covered in Section 2.4.1)
- A matrix entering class which will be used to enter a graph or an adjacency matrix depending on the problem. (Covered in section 2.4.2.2)

- An exception class for throwing an exception when the Simplex feasible region is not bounded.
- A tutorial class which contains the tutorial. (Covered in Section 2.3)
- Several other static classes which perform functions such as: console helper, maths class.

In the diagram above, it can be seen that all the problems implement an interface:



This is so that in the main program the problem class just needs to be instantiated and then it can just be interacted with through an interface.
The Random object is passed throughout the program otherwise there is an increased chance of generating similar seeds (and so similar outputs).

The problem classes in the diagram are in two parts: the creator and the solver. Now this isn't strictly necessary but I think that the abstraction it provides is useful. The creator class puts together a problem to be solved in the solver class. For example: it creates a Graph object and then this is passed into the solver object.

The quick sort solver class has a generic type as it can be used for either doubles or characters (see Section 2.9).

Outside from the diagram above, there is an Exception class I can implement. It would be fairly simple (as it just needs to be a type of Exception with the correct name) and it just calls the base Exception class.

There would also be a console helper class. This would be fairly useful when creating the tutorial and menus as I would have it to print out lines of text slowly, wait for key presses, and delete lines of characters from the console.

There would also be a math class which would help me with small little maths functions that can be used anywhere in the program e.g. significant figures, precision checks, subscript integer converter, etc.

2.2 Choice of Programming Language

To do this project I will use C#. C# is an object-oriented language which is also imperative. Although I am more proficient in Python, Python is mainly imperative and is not great with the OOP model. This means that I have significantly more experience with using OOP in C# which is the paradigm I intend to use. This should allow me to spend less time coding (or learning how to code) and more time on implementation.

C# also has the library ScottPlot which allows me to graph in 2D for Simplex. This is gone over in more detail in Section 2.5.2.

2.3 Tutorial

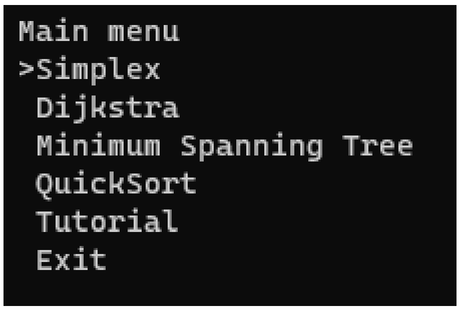
The tutorial should be designed to help the user navigate the program more easily. The tutorial should include an explanation of how to use each aspect of the program. This could have certain elements highlighted in colour such as red for warnings and green for subheadings.
Another thing could be to incorporate interactive versions of the elements found in Section 2.4.2. This would allow the user to get a feel of how they work which would help them make less mistakes when entering maths.

2.4 UI Design

2.4.1 Menus

Menus would be used to help navigate to different sections of the program. They would be navigated using arrow keys and the enter key only, where all other keys do nothing. This makes it so that no other key can be used and exceptions won't need to be thrown to handle this. There are 2 types of menu I could implement:

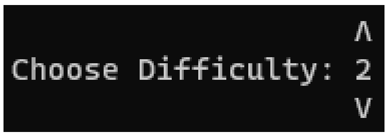
The first would be a list of items to choose from. An arrow (>) could be displayed to the left of the options and it would move up and down to select different options. This would be useful for displaying a list of unique options. See Figure 2.2.



```
Main menu
>Simplex
 Dijkstra
 Minimum Spanning Tree
 QuickSort
 Tutorial
 Exit
```

Figure 2.2: An example menu

Another type of "menu" I could add is one that changes between different options as you click up and down arrows. This would be particularly useful when selecting a number from a range of numbers or a node from a range of nodes (letters). The option of whether you can increase or decrease would be shown by up (Λ) and down (V) "arrows" above and beneath the number. See Figure 2.3.



```
                    Λ
Choose Difficulty: 2
                    V
```

Figure 2.3: How it looks to choose a number wtihtin a set range

2.4.2 Entering Maths

The program needs to be able to have an easy maths input. You need to be able to enter equations for Simplex, and tables in some form for all of them.

2.4.2.1 Equations

An equation could be entered by displaying the variables within the equation, and having the user enter in coefficients. Blank values (initially set where the user can enter values) could be represented by underscores (_) and these would be parsed as 0. For example, the program may display the equation:

$$\_x_1 + \_x_2 + \_x_3 \leq \_$$

Which could have values entered like this:

$$-4x_1 + \_x_2 + 1x_3 \leq 234$$

And would be displayed as:

$$-4x_1 + x_3 \leq 234$$

A defined set of operations could be used to navigate an equation:

Key	Operation
Right Arrow	Move cursor one position right.
Left Arrow	Move cursor one position left.
Tab	Move to next item.
Enter	Move to next item unless it is the last item, then finish entering the equation.
Backspace	Delete character preceding the cursor.
Escape (Esc)	Finish entering the equation.
Any Number	Number is added at the cursor's location.
Period/Full Stop/.	"." is added at the cursor's location.
Dash/-	If there is no dash preceding the element, one will be added.
All other keys	Nothing.

It should be noted that the left and right arrows operate within an element, such that it moves between the different characters. If the cursor happens to be at the start or end of the current element, it would move onto the next element in the equation.

The equation could then be stored in a class, with arrays holding the LHS and RHS of the equation, being parallel for coefficients and variables. The class would also store the symbol used in the equation, be it "=", "<=" or ">=". The symbols $\leqslant$ and $\geqslant$ cannot be displayed in all consoles, so they will not be implemented. For example, the problem above could have the arrays:
LHSvariables = ["x_1", "x_2", "x_3"]
LHScoefficients = [-4, 0, 3]
RHSvariables = [""]
RHScoefficients = [234]

2.4.2.2 Tables/Matrices

The table could be displayed with columns/rows of variables/nodes and each grid can have a value entered. This could be entered in the same fashion as equations, with a slightly modified set of operations. New/modified operations could be:

Key	Operation
Up Arrow	Move cursor up one row.
Down Arrow	Move cursor down one row.
Enter	Move to next element below unless it is the last item, then finish entering the table.
Dash/-	If there is no dash preceding the element and it is not a Dijkstra's Algorithm problem, a dash will be added to the beginning of the element.

As the table should be able to function as both a Simplex tableau and an adjacency matrix, it must be able to be in the correct format for both. This means that when it is not an adjacency matrix it shouldn't display the variables down the side of the tableau but it should when it is a graph. It also means that when entering a graph, boxes at the intersection of the same node (e.g. B and B) should be blocked out by a "/", and the cursor should automatically skip over these boxes. This is because loops are not crucial to any of the potential graph problems and should be ignored. When entering an undirected graph, the program should automatically fill out lines of symmetry which can be achieved by indexing the opposite values in a 2D array e.g. table[i, j] would be symmetrically completed in table[j, i]. As highlighted above, Dijkstra's Algorithm cannot be performed with negative edge weights so they shouldn't be able to be entered when entering a Dijkstra's Algorithm problem.

The table would then be parsed, where a 0 or an underscore (_) would represent no connection. This table would then be stored in a graph class or be input directly into a Simplex class, depending on the problem.

2.5 Simplex

2.5.1 The Algorithm

The Simplex algorithm works as described in Section 1.4.2.1. The program should ask whether to enter constraints or a tableau and also if it should display row operations and basic variables after each iteration. More examples can be found between Sections 3.2.1 - 3.2.4.

2.5.1.1 Accepting Constraints

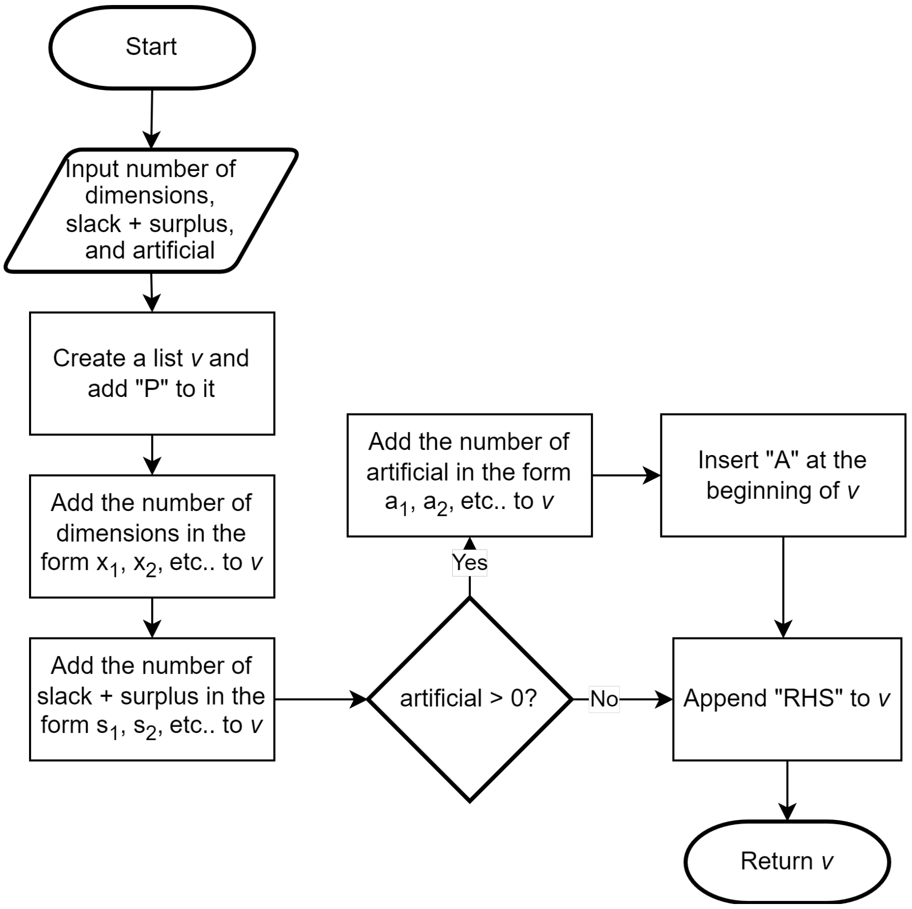
The program should be able to accept constraints. It would first ask the user how many variables there are and then have scrolling menu for number of each type of constraint (the max amount would be how many equations there are minus how many have been assigned). The user would then be presented with a list type menu of the different types of equations they have chosen. They could then pick and edit these equations at will.

At this stage I would also check if all dimensions are bounded and all equations are used before proceeding. To check that they are all bounded just check that there is at least one coefficient that isn't 0 with each dimension in the $\leqslant$ equations.

2.5.1.2 Accepting a Tableau

The program should ask how many variables there are (dimensions), the number of slack + surplus variables and the number of artificial variables. The slack and surplus variables can be grouped together as after they've been parsed from constraints they act in the exact same fashion. Artificial variables would later be deleted so these are separate.

The program could then take all this information and create a list of variables to put in a table:



Start
Input number of dimensions, slack + surplus, and artificial
Create a list *v* and add "P" to it
Add the number of dimensions in the form x_1, x_2, etc.. to *v*
Add the number of slack + surplus in the form s_1, s_2, etc.. to *v*
artificial > 0?
Yes
Add the number of artificial in the form a_1, a_2, etc.. to *v*
Insert "A" at the beginning of *v*
No
Append "RHS" to *v*
Return *v*

This list of variables can then be used in a table that the user would enter their values into.

2.5.1.3 Generating a Problem (Constraints)

There are various difficulty settings I can implement here:

	Dimensions					Constraints		
Difficulty	2	3	4	5	6	$\leqslant$	$\geqslant$	$=$
1	✓					✓		
2	✓	✓				✓	✓	
3		✓	✓			✓	✓	
4			✓	✓		✓	✓	✓
5				✓	✓	✓	✓	✓

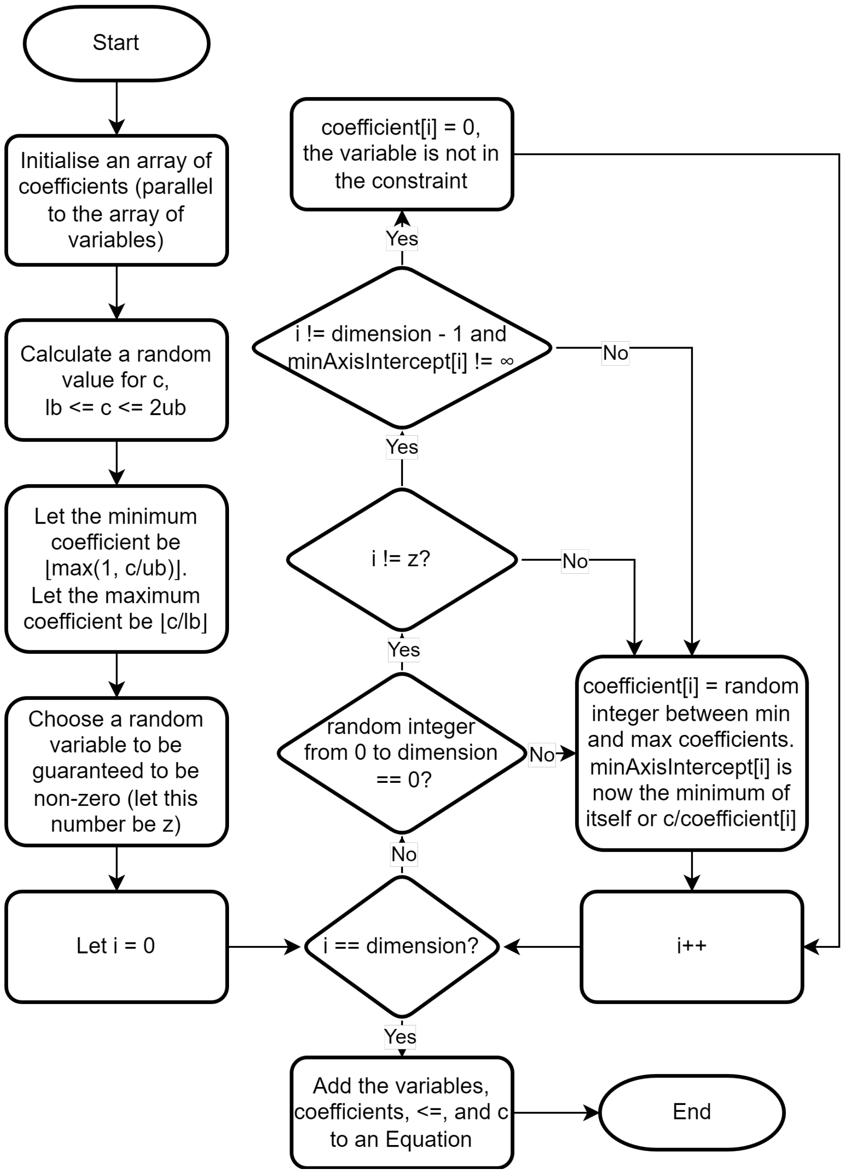
A 2 stage problem is only guaranteed with difficulty level 5, however there is an extremely high chance of one with difficulties 3 & 4 and a 50% chance of one with difficulty level 2.

To generate the constraints I would first set up a random lower bound for the constraints and then use an equation (lower bound = lb, upper bound = ub):

$$ub = lb + 4\lfloor (lb)^{\frac{1}{2}} \rfloor$$

This keeps the whole problem within a sensible range (which also happens to look good on the 2D graph).

To generate 2-3 "$\leqslant$" constraints:
Each constraint will be in the form $x_1 + x_2 + ... + x_n \leqslant c$, where $lb \leqslant c \leqslant 2ub$. The program instantiates an array (minAxisIntercepts) of doubles to infinity and this will store the minimum axis intercepts (parallel to the array of variables and so has length equivalent to the number of dimensions). This is to make sure that every region is bounded (which can be seen when each item in the array is non-infinite). Then for each equation:



Start
Initialise an array of coefficients (parallel to the array of variables)
Calculate a random value for c, lb <= c <= 2ub
Let the minimum coefficient be ⌊max(1, c/ub)⌋. Let the maximum coefficient be ⌊c/lb⌋
Choose a random variable to be guaranteed to be non-zero (let this number be z)
Let i = 0
i == dimension?
Yes
Add the variables, coefficients, <=, and c to an Equation
End
No
random integer from 0 to dimension == 0?
No
Yes
i != z?
No
Yes
i != dimension - 1 and minAxisIntercept[i] != ∞
No
Yes
coefficient[i] = 0, the variable is not in the constraint
coefficient[i] = random integer between min and max coefficients. minAxisIntercept[i] is now the minimum of itself or c/coefficient[i]
i++

On each iteration there is a random variable that must be guaranteed to have a non-zero coefficient. This is so that each equation actually bounds one or more variables, otherwise you would have a rather ridiculous looking inequality in the form: "$\leqslant c$", where there is nothing preceding the inequality.
There is also a clause where if it is the last equation and minimum axis intercept of that variable is infinity, it must have a coefficient assigned that iteration as it would otherwise be unbounded.

To generate "$\geqslant$" constraints it is exactly the same as generating "$\leqslant$" constraints with a one key difference: the variable guaranteed to be non-zero is also guaranteed to intersect its axis at a value less than the lower bound. This guarantees a feasible region. The rare "=" constraint is the exact same, but can only occur once.

2.5.1.4 Performing the Algorithm

All equations would be reformulated the exact same way as described in Section 1.4.2.1. Then I can take all unique variables and put them into a tableau, where each row represents a constraint / objective function (this step doesn't apply if it's already a tableau).
After this has happened, the algorithm could then be performed as in Figure 2.4. It may be noted that the 2 loops in the flowchart are very similar. However, the subtle but key differences in conditions mean they have to be separate.

After each iteration and after the reduction of a 2 stage tableau, the tableau must be displayed. To ensure that each column is of the correct width to be tabular, the program must first find the longest length item in each column before displaying. Pivot rows and pivot columns, and the intersection which is the pivot value would be colour coded accordingly.

To make sure that the problem is solvable with Simplex, after each iteration the program will check whether te tableau has been generated perviously as this would create an infinite loop. If it has been generated before, it is likely that the region is not bounded and so an exception is thrown.

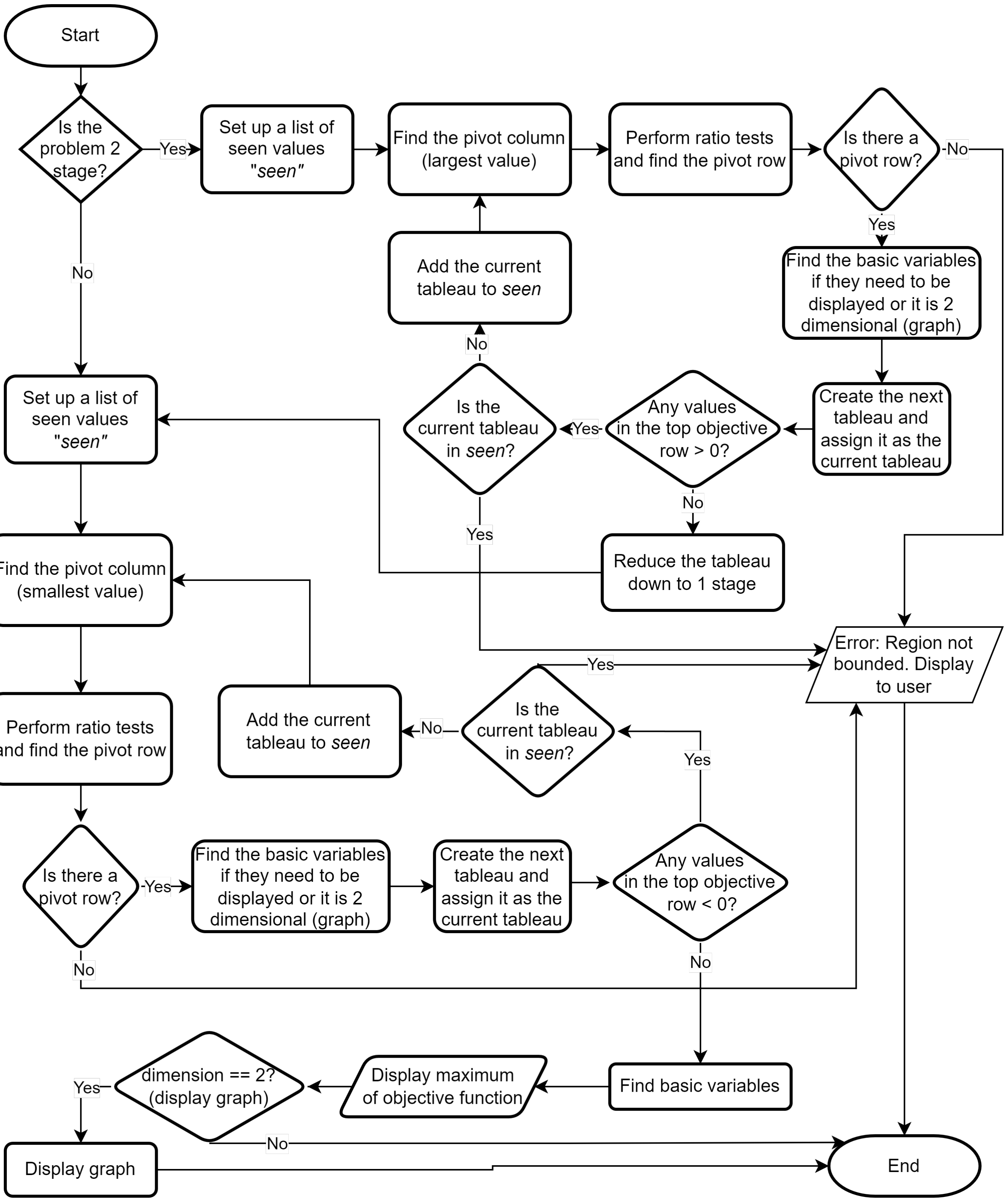
To find the pivot row, the program would just do a ratio test on each of the columns. A division by 0 would just be ignored and would read as "undefined" in the output.

To find basic variables and their values the program must find all variables in which there is one "1" in the column and the rest are 0's. The corresponding value in the RHS will then be found. There is a rare case where more than one variable corresponds to a value in the RHS and this can just be handled as the sum of the two variables.

C# (as with all programming languages) has precision errors. 0 is an extremely important value when it comes to simplex so we say that if a number is within $\varepsilon = 1\text{e}{-12}$ of 0, then it must be 0. This should be a good enough work around for a precision error and any reasonable Simplex problem.
To round items in tableau to 3.s.f, there is an idea with using logarithms but after considering precision errors, it doesn't always work. However, an easy alternative method is to use the in-built scientific notation converter which rounds automati-

cally, and then convert it back so the number goes:
double → string → double, where the string is the rounded number but in scientific notation.

The rest of the algorithm (creating the next tableau) operates as described in Section 1.4.2.1.



Start
Is the problem 2 stage?
Yes
No
Set up a list of seen values "*seen*"
Find the pivot column (largest value)
Perform ratio tests and find the pivot row
Is there a pivot row?
No
Yes
Find the basic variables if they need to be displayed or it is 2 dimensional (graph)
Create the next tableau and assign it as the current tableau
Any values in the top objective row > 0?
Yes
No
Is the current tableau in *seen*?
No
Yes
Add the current tableau to *seen*
Reduce the tableau down to 1 stage
Set up a list of seen values "*seen*"
Find the pivot column (smallest value)
Perform ratio tests and find the pivot row
Is there a pivot row?
Yes
No
Find the basic variables if they need to be displayed or it is 2 dimensional (graph)
Create the next tableau and assign it as the current tableau
Any values in the top objective row < 0?
Yes
No
Is the current tableau in *seen*?
Yes
No
Add the current tableau to *seen*
Error: Region not bounded. Display to user
Find basic variables
Display maximum of objective function
dimension == 2? (display graph)
Yes
No
Display graph
End

Figure 2.4: How Simplex would work in my program

2.5.2 Graphing the problem (libraries)

This concerns all 2-dimensional Simplex problems (that aren't entered through a tableau).

The library I will be using to plot the graphs is ScottPlot (`https://scottplot.net/`). It is a library mainly used for statistical modelling and plotting points. However, there is a small feature within ScottPlot that should allow me to graph functions in an interactive window. As this is not the main purpose of ScottPlot, there is a little bit of setting up I need to do to get it working for Simplex.

Some simply cosmetic changes have been requested by student a post-interview such as dark mode and the font comic sans. These will be implemented in some form e.g. using a plain black image to create a black background.

First off, the view limits of the graph need to be set so that they can't go negative so: $x, y \in [0, \inf)$.

After this, we can concern ourselves with the constraints which must be plotted. I must also find the polygon which is the feasible region so I need to keep track of all intersections between each line with each other line and the axes.
To plot each constraint, each constraint needs to be changed into a function in terms of x. Take the inequality $ax + by \leqslant c$. To plot it as a function (it will have to be an equality) of x, it is simply:

$$f(x) = -\frac{a}{b}x + \frac{c}{b}$$

However, there is a small problem if the coefficient of y is 0. Luckily, vertical lines can be plotted to handle this. The vertical line or function is then added to the plot. I will also add this function that is in slope-intercept form ($y = mx + c$) to a list of tuples which stores (slope or gradient, y-intercept) to be used later.
At this step we can also very easily add the line intersections with the axes to the list of intersections (which are stored as tuples):

```
if $a > 0$ then
    AddItem(intersections, $(\frac{c}{a}, 0)$)
end if
if $b > 0$ then
    AddItem(intersections, $(0, \frac{c}{b})$)
end if
```

At this step I keep track of the maximum value of the intersections with each respective axis. This is so that I can set out the graph in a nice range when it displays (as this is what would be saved as an image if the user chose that option).

Now that the lines have been plotted on the graph, the next step is to find all intersections between lines. Using the list of slope-intercept forms I can find the intersection between any two lines. To do this I can iterate through all unique combinations of 2 values in the list by using a nested loop:

```
$n \leftarrow size(slopeIntercept)$
**for** $i \leftarrow 0$ to $n$ **do**
    **for** $j \leftarrow i + 1$ to $n$ **do**
        FindIntersection(slopeIntercept[i], slopeIntercept[j])
    **end for**
**end for**
```

It's useful to ignore all intersections where $x < 0$ or $y < 0$ as all intersections with the axes have already been found and the region is always bounded by $x \geqslant 0$, $y \geqslant 0$. To find the intersection between 2 sets of slope-intercept forms: (m_1, c_1), (m_2, c_2), I can use the following algorithm:

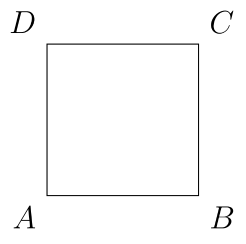
```
**function** FindIntersection($m_1$, $c_1$, $m_2$, $c_2$)
    **if** $m_1 = m_2$ **then**
        **return** $(-1, -1)$
    **end if**

    **if** $m_1 = \inf$ **then**
        **return** $(c_1, m_2 \times c_1 + c_2)$
    **end if**
    **if** $m_2 = \inf$ **then**
        **return** $(c_2, m_1 \times c_2 + c_1)$
    **end if**

    $x = (c_2 - c_1)/(m_1 - m_2)$
    $y = m_1 \times x + c_1$
    **return** $(x, y)$
**end function**
```

It can be seen that the above function is split into 3 parts:

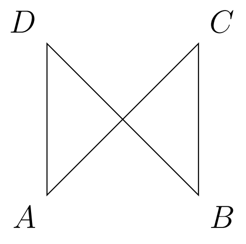
- The first case $m_1 = m_2$ checks whether the gradients of the two slopes are the same. If this is the case, the lines must be parallel and never intersect and so the function returns (-1, -1) which will be ignored as $x < 0$ or $y < 0$.
- The second case is where either of the gradients of the two lines are infinite. This means that the equation is in the form $ax = c$. If the first equation is in this form, the point of intersection must be $(c_1, f_2(c_1))$, where $f_2(x)$ is the function for the 2nd equation and vice versa.
- The third case is where there are two "normal" lines which are in the form $y = mx + c$ but where m can be any value including 0. Setting the two equations equal to each other and rearranging for x, you get that $x = \frac{c_2 - c_1}{m_1 - m_2}$. Then the value of y can be either of the functions: $f_1(x)$ or $f_2(x)$ which should produce an identical result.

The next step is to iterate through the list of coordinates (where $x \geqslant 0$, $y \geqslant 0$) and find all the points that satisfy all the inequalities. It is fairly simple to just plug the coordinates into each constraint and check if the symbol satisfies the result.

After this has been done, we now have the coordinates of all the points that lie on the polygon that is the feasible region. ScottPlot plots the polygon by tracing a line from coordinate to coordinate and then filling in the region bounded by the line. However, this could result in the following problem:
Take a polygon ABCD:



It looks like this as it traces the edge from A→B→C→D→A. However, now consider the case where the corners are not in some cycle of ABCD:



This is what happens when the corners are not in a cycle of ABCD and instead have been traced in the order: A→C→B→D→A. This is no longer a polygon and doesn't represent the feasible region.

To find a route around the polygon I think the best method is to use complex loci. This should be fine as the polygon must always be convex. This is due to the nature of the constraints being lines that bound a region.
The first thing I would do would be to find the mean of x and y of the coordinates of the polygon. This is to achieve some sort of centre point. The value of the complex number associated with each coordinate (x , y) is:

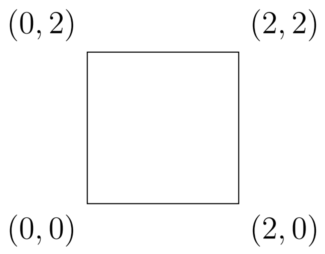
$$(x - \overline{x}) + (y - \overline{y})i$$

I would then sort the coordinates by their argument, which can be calculated using the following algorithm (where complex numbers are in the form a+bi):

```
function ARGUMENT(a, b)
    if $a = 0$ & $b > 0$ then
        return $\frac{\pi}{2}$
    end if
    if $a = 0$ & $b < 0$ then
        return $-\frac{\pi}{2}$
    end if

    if $a < 0$ & $b \geqslant 0$ then
        return atan($\frac{b}{a}$)+$\pi$
    end if
    if $a < 0$ & $b < 0$ then
```

```
        return atan($\frac{b}{a}$)$-\pi$
    end if
    return atan($\frac{b}{a}$)
end function
```

This is a fairly simple algorithm to find the principal argument of any complex number, where atan() is equivalent to arctan() or $\tan^{-1}()$. The reason the argument should find a path around the polygon edge is that each coordinate should have a unique argument (as the polygon is convex) which, when ordered, goes around the polygon in an anticlockwise manner. I can just sort these using the in-built C# sort() function. I would then add these ordered coordinates to a ScottPlot polygon which can then be drawn.
Let's go through an example: Take the polygon below with the coordinates as shown:



It is evident that $(\overline{x},\ \overline{y}) = (1,\ 1)$ as they are both calculated by: $\overline{x} = \frac{0+2+2+0}{4} = 1$. Now take a jumbled list of these elements: [(0, 2), (2, 0), (2, 2), (0, 0)]. Now for each of these we will map them to their respective complex number and then find the argument:

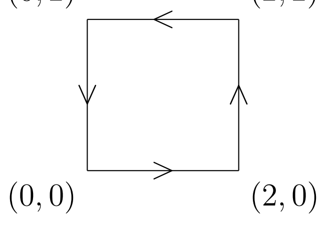
$$(0,2) \to -1+i \to \arg(-1+i) = \frac{3\pi}{4}$$

$$(2,0) \to 1-i \to \arg(1-i) = -\frac{\pi}{4}$$

$$(2,2) \to 1+i \to \arg(1+i) = \frac{\pi}{4}$$

$$(0,0) \to -1-i \to \arg(-1-i) = -\frac{3\pi}{4}$$

Now we rearrange the list so that it is sorted by the argument: [(0, 0), (2, 0), (2, 2), (0, 2)]. If you go through this list (and loop back round to the start) no lines are being intersected and it forms the polygon as intended:



The polygon is now drawn and the program can proceed to the next step.

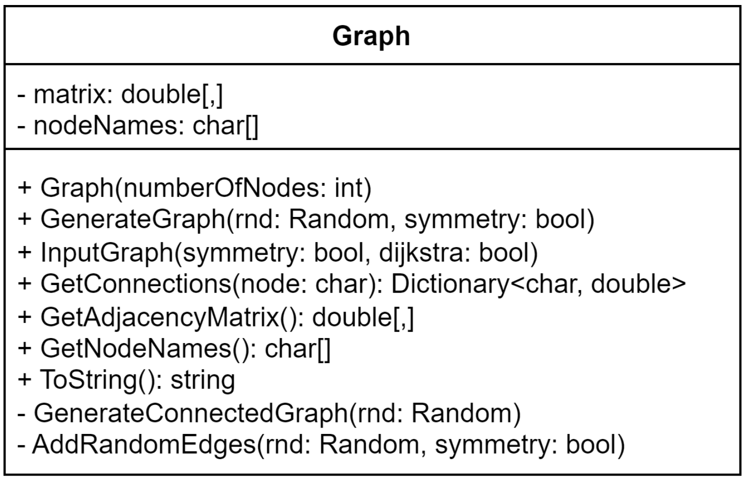
While the Simplex Algorithm is being performed, I want to add the coordinates of locations where the algorithm has "visited". This is where the Basic Variables in Simplex come in. Any basic variable will have a corresponding value, otherwise the coordinate at that point is 0. Say there are Basic Variables of: $x = 2$, $s_1 = 5$

and all other variables: $y, s_2, s_3, ...$ are non-basic. Then the coordinate to plot the point at must be (2, 0). This can be done by adding a ScottPlot marker at that stage.

Finally, when the Simplex Algorithm has finished executing, the program can open a new windows form to display this graph. To do this I would start up a new thread and open this window. The reason for multi-threading is so that the user can interact with both the console and the plot at the same time. At this stage the user should be able to save and name the image of the plot to their device in a sub-folder "images" (controlled through the console).

2.6 Graphs

The Graph class will be used for all graphing algorithms (Dijkstra's, Prim's and Kruskal's). It needs to be able to take in a user's graph and generate graphs stored in an adjacency matrix. The class could be laid out something like this:



```
Graph
- matrix: double[,]
- nodeNames: char[]
+ Graph(numberOfNodes: int)
+ GenerateGraph(rnd: Random, symmetry: bool)
+ InputGraph(symmetry: bool, dijkstra: bool)
+ GetConnections(node: char): Dictionary<char, double>
+ GetAdjacencyMatrix(): double[,]
+ GetNodeNames(): char[]
+ ToString(): string
- GenerateConnectedGraph(rnd: Random)
- AddRandomEdges(rnd: Random, symmetry: bool)
```

A graph will always start at node A and go up to another node up to Z and this will be decided when creating the object with the number of nodes. GetConnections() takes in a node and returns a dictionary of the names of the nodes it's connected to along with the length between them.

There is the option to input a graph which would be done through entering a matrix/table as outlined in Section 2.4.2.2. If there is symmetry then both connections (e.g. $A \to B$ and $B \to A$) would be filled in at the same time. The Dijkstra boolean makes it so that negatives can't be input into the graph (as Dijkstra's algorithm cannot work with negative values). Unfilled boxes in the input will just be parsed as 0 which means no connection. It doesn't make sense to have 0 be allowed as a connection as that just means the two nodes with the 0 connection are the same node.

The other option is to generate a random graph with edge weights within a bound (this uses same formula as in Section 2.5.1.3). To do this the program can generate a spanning tree (undirected) and then add random edges after that (directed or undirected). This is represented by GenerateConnectedGraph() followed by AddRandomEdges() in the diagram. To make a spanning tree I would use a loop-erased random walk described in Section 1.4.2.2. I could implement this in my program as in Figure 2.5:

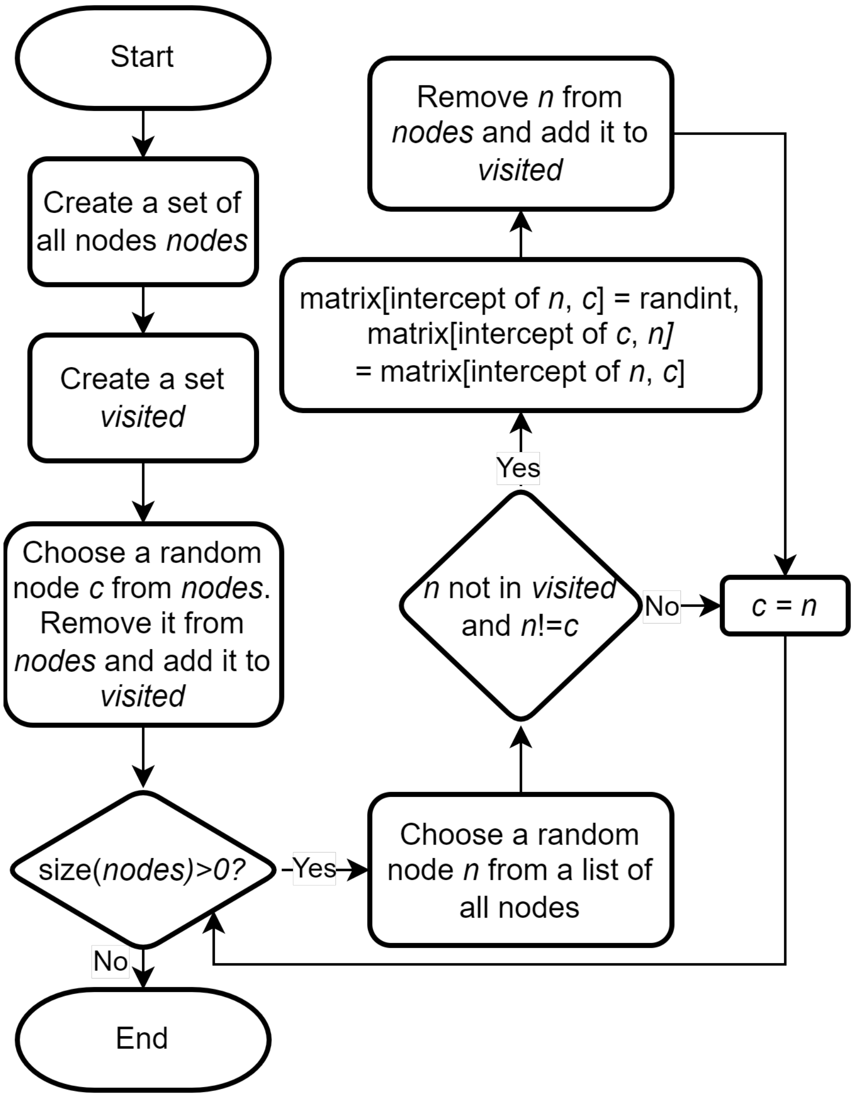

Start
Create a set of all nodes *nodes*
Create a set *visited*
Choose a random node *c* from *nodes*. Remove it from *nodes* and add it to *visited*
size(*nodes*)>0?
Yes
No
End
Choose a random node *n* from a list of all nodes
n not in *visited* and *n*!=*c*
Yes
No
matrix[intercept of *n*, *c*] = randint, matrix[intercept of *c*, *n]* = matrix[intercept of *n*, *c*]
Remove *n* from *nodes* and add it to *visited*
c = *n*

Figure 2.5: How a loop-erased random walk could work.

The random walk starts at a random node. It then keeps track of the current node c. While not all nodes have been visited, the algorithm chooses a random neighbour node n to c. If n isn't in the tree, it is moved from the set of available nodes and into the set of visited nodes. A random integer weight is added into the adjacency matrix between the two nodes in both directions (as to be undirected). Regardless of whether or not n was in the tree or not, c now becomes n. This repeats until all nodes have been visited.
The algorithm essentially walks between nodes and creates connections if the next node hasn't been visited before.

After the spanning tree has been created, random edges can now be added; either directed or undirected depending on the problem.

2.7 Dijkstra's Algorithm

2.7.1 The Algorithm

Dijkstra's Algorithm works the exact same way as described in Section 1.4.2.2. It could be imlpemented in my program as follows:

```
nodeDict $\leftarrow$ Dictionary(char $\rightarrow$ DijkstraNode)
q $\leftarrow$ PriorityQueue
Enqueue(q, nodeDict[startNode])
order $\leftarrow$ 1
while length(q) $>$ 0 do
    cn = Dequeue(q)
    if cn is visited then
        continue
    end if
    cn.visited $\leftarrow$ true
    cn.order $\leftarrow$ order
    order $\leftarrow$ order + 1
    for node, distance in connections(network, cn) do
        if cn.weight + distance $<$ nodeDict[node].weight then
            nodeDict[node].weight $\leftarrow$ cn.weight + distance
            nodeDict[node].route $\leftarrow$ cn.route + node
            Enqueue(q, nodeDict[node])
        end if
    end for
end while
```

Where:

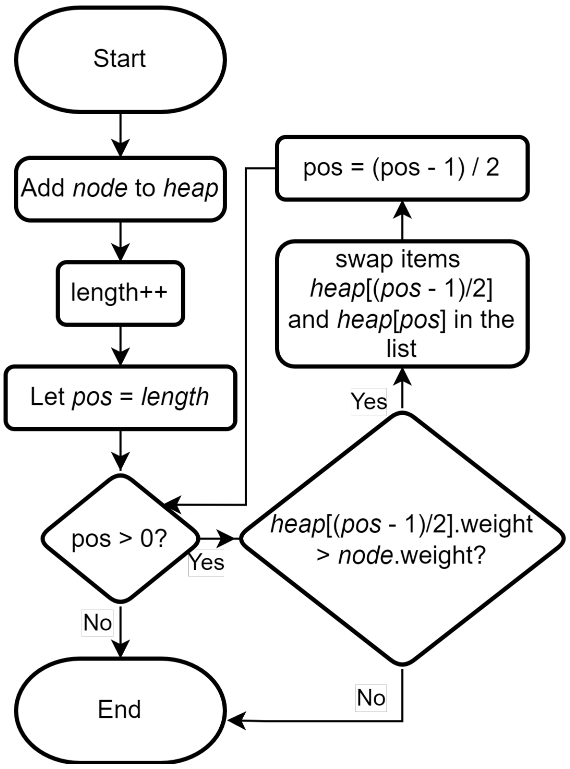
- **cn** is the current node (this has been condensed to make the pseudocode fit on one line)
- **nodeDict** is a dictionary that maps a node's name to its DijkstraNode object defined in Section 2.7.2.2. Each DijkstraNode object is instantiated in this step.
- **q** is a priority queue defined in Section 2.7.2.1
- **network** is a Graph defined in Section 2.6

After this has run, the program can just display all the nodes with their respective boxes.

2.7.2 Data structures

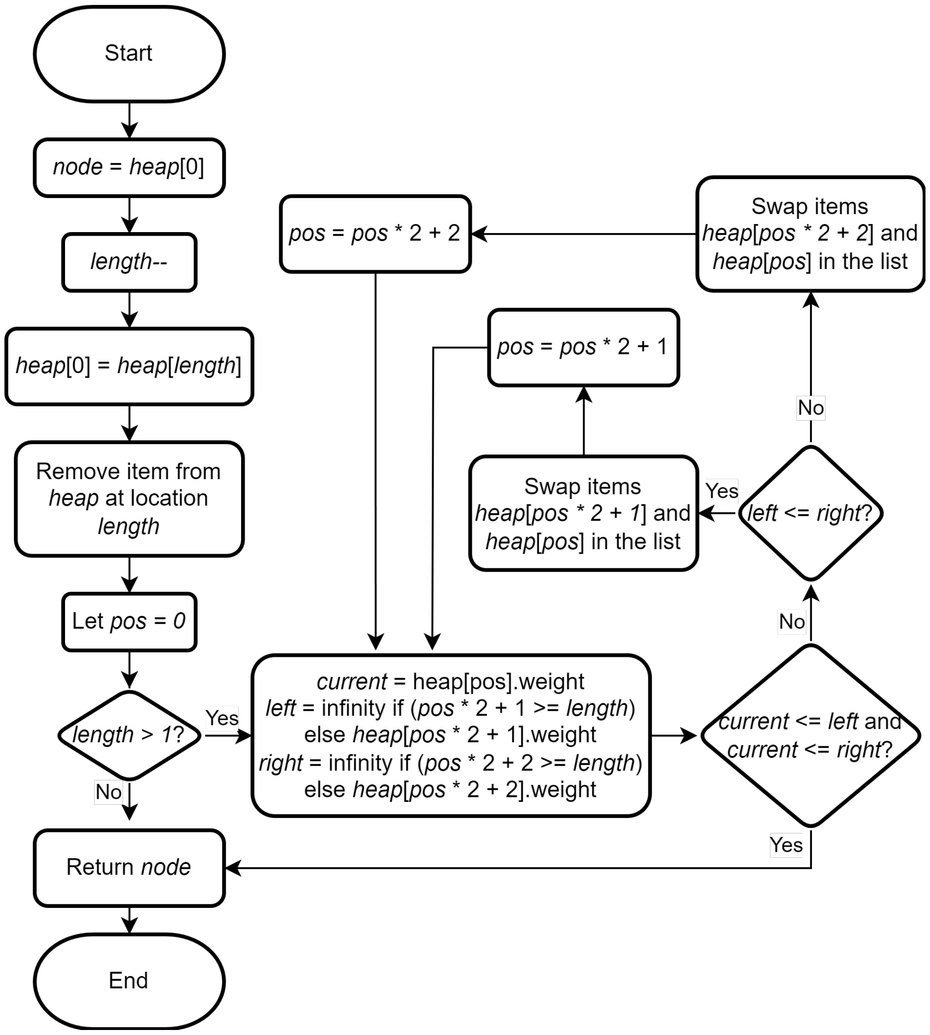
2.7.2.1 Priority Queue

To implement the priority queue I will use a Minimum Heap Binary Tree (described in Section 1.4.2.2). I could implement it as a list composed of DijkstraNodes *heap* and keep track of the length as an integer *length.* Then I just implement the Enqueue() with the sift up and Dequeue() with the sift down.



Start
Add *node* to *heap*
length++
Let *pos* = *length*
pos > 0?
Yes
No
heap[(*pos* - 1)/2].weight > *node*.weight?
Yes
No
swap items *heap*[(*pos* - 1)/2] and *heap*[*pos*] in the list
pos = (pos - 1) / 2
End

Figure 2.6: Enqueue() function (with sift-up)



Start
node = *heap*[0]
length--
heap[0] = *heap*[*length*]
Remove item from *heap* at location *length*
Let *pos* = *0*
length > *1*?
Yes
No
current = heap[pos].weight
left = infinity if (*pos* * 2 + 1 >= *length*) else *heap*[*pos* * 2 + 1].weight
right = infinity if (*pos* * 2 + 2 >= *length*) else *heap*[*pos* * 2 + 2].weight
current <= *left* and *current* <= *right*?
Yes
No
left <= *right*?
Yes
No
Swap items *heap*[*pos* * 2 + *1*] and *heap*[*pos*] in the list
pos = *pos* * 2 + 1
Swap items *heap*[*pos* * 2 + 2] and *heap*[*pos*] in the list
pos = *pos* * 2 + 2
Return *node*
End

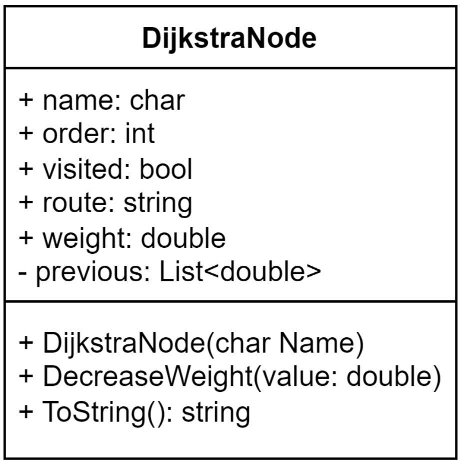
Figure 2.7: Dequeue() function (with sift-down)

The Enqueue() function works by adding an item to the end of the list and sifting it up the tree if it is smaller than it's parent.

The Dequeue() function works by storing the first node. It then moves the last node in the list and replaces the first node with it. This is then sifted down into its correct position and this should simultaneously sift the smallest node to the top (which should be one of the children to the root node). *left* and *right* are the children to *current*. If they don't exist, they set to infinity.

2.7.2.2 Node

This is a node that stores lots of information to do with the execution of Dijkstra's Algorithm - a DijkstraNode. The class diagram looks like:



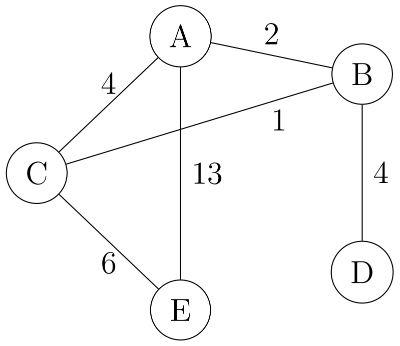
Variable	Description
name	The name of the node (from A-Z).
order	The placement of this node in relation to other nodes of when it has been visited.
visited	Has the node been visited yet (and therefore the shortest path achieved)?
route	The route associated with the current weight.
weight	The current shortest distance to get to the node.
previous	A list of all previous weights (not including infinity).

This is essentially a struct but with a few small methods:

- The constructor DijkstraNode() takes in the node name and instantiates all the variables. It sets *weight* to infinity.
- DecreaseWeight() takes in a value, adds it to *previous* and sets *weight* equal to it.
- ToString() will override C#'s ToString() method. This will display the Dijkstra box (unless the weight is infinite, meaning that the node hasn't been visited).

2.7.3 Example

Let's use the graph below:



Of which the adjacency matrix looks like:

	A	*B*	*C*	*D*	*E*
A	–	2	4	–	13
B	2	–	1	4	–
C	4	1	–	–	6
D	–	4	–	–	–
E	13	–	6	–	–

We will now attempt to perform Dijstkra's Algorithm from node A:

First set up the dictionary *nodeDict* to map each character that is a node to a DijkstraNode object. The DijkstraNode object for node A is set so that its weight is 0 and its route is "A".
Next set up the priority queue q and enqueue the *nodeDict*['A'].
Then, *order* $\leftarrow$ 1

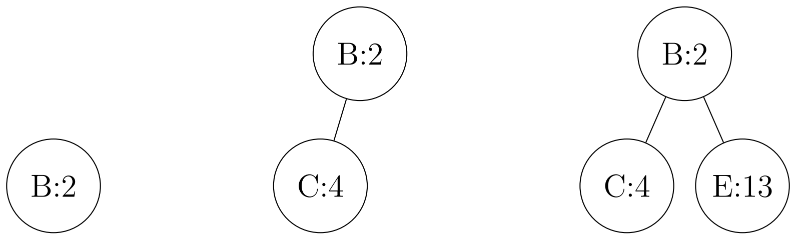
Dequeue q and we get the DijkstraNode associated with node A *cn*. *cn*.visited is false so we carry on. *cn*.visited is now set to true (node A can no longer be visited). *cn*.order is assigned *order*. *order* is incremented. Now we visit all of A's connections:

- (B, 2)
  cn.weight + 2 = 2 $< \infty$.
  nodeDict['B'].weight $\leftarrow$ *cn*.weight + 2
  nodeDict['B'] $\leftarrow$ "A" + 'B'.
  Enqueue *nodeDict*['B'] to q.

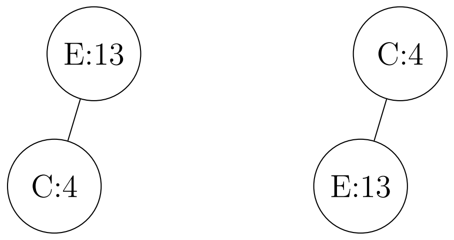
- (C, 4)
  cn.weight + 4 = 4 $< \infty$.
  nodeDict['C'].weight $\leftarrow$ *cn*.weight + 2
  nodeDict['C'] $\leftarrow$ "A" + 'C'.
  Enqueue *nodeDict*['C'] to q.

- (E, 13)
  cn.weight + 13 = 13 $< \infty$.
  nodeDict['E'].weight $\leftarrow$ *cn*.weight + 13
  nodeDict['E'] $\leftarrow$ "A" + 'E'.
  Enqueue *nodeDict*['E'] to q.

After each step, the heap that is the priority queue looks like these (node: weight):



Now we Dequeue from q. To do this: we store B. Then move E:13 to the top of the tree (as it is currently the last node). Then check its children: C:4 and nothing. So we get *left* $\leftarrow 4$ and *right* $\leftarrow \infty$. $4 < 13$ and $4 < \infty$ so we get that E:13 swaps with its left child C:4:



We then return the DijkstraNode associated with node B with weight 2 and call this *cn*. *cn*.visited is false so we carry on. *cn*.visited is now set to true. *cn*.order is assigned *order* = 2. *order* is incremented. Now we visit all of B's connections:

- (A, 2)
  cn.weight + 2 = 4 > 4. Continue
- (C, 3)
  cn.weight + 1 = 3 < 4.
  nodeDict['C'].weight $\leftarrow$ *cn*.weight + 1
  nodeDict['C'] $\leftarrow$ "AB" + 'C'.
  Enqueue *nodeDict*['C'] to *q*.
- (D, 4)
  cn.weight + 4 = 6 < ∞.
  nodeDict['D'].weight $\leftarrow$ *cn*.weight + 4
  nodeDict['D'] $\leftarrow$ "AB" + 'D'.
  Enqueue *nodeDict*['D'] to *q*.

To perform the first Enqueue(), C:3 is added to the end of the tree. This is then compared with C:4 which is larger and so it swaps places with C:4:

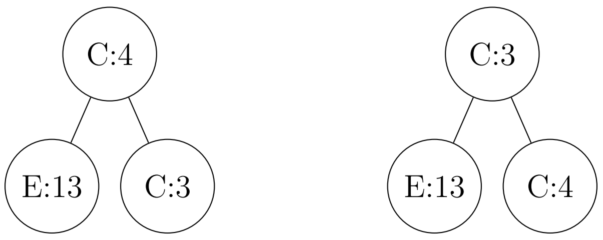


Figure 2.8: When C:3 is added to the tree followed by after it has been sifted up

A similar process will be applied to D:4. This will be added to the end of the tree (left child to E:13) and then will switch places with it after being sifted up.
C:3 is now dequeued from the list, resulting in the new heap:

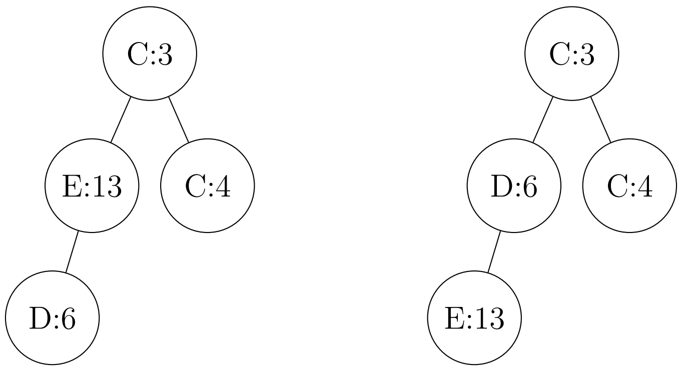
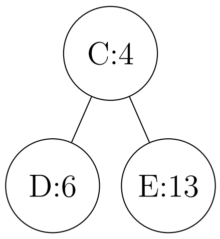


Figure 2.9: When D:4 is added to the tree followed by after it has been sifted up



Now we set this node as *cn*. *cn*.visited is now set to true. *cn*.order is assigned *order* = 3. *order* is incremented. Now we visit all of C's connections:

- (A, 4)
  cn.weight + 4 = 7 > 0. Continue
- (B, 1)
  cn.weight + 1 = 4 > 2. Continue
- (E, 6)
  cn.weight + 6 = 9 < 13.
  nodeDict['E'].weight ← *cn*.weight + 6
  nodeDict['E'] ← "ABC" + 'E'.
  Enqueue *nodeDict*['E'] to *q*.

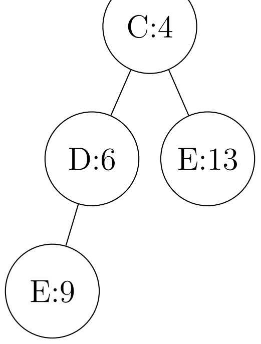
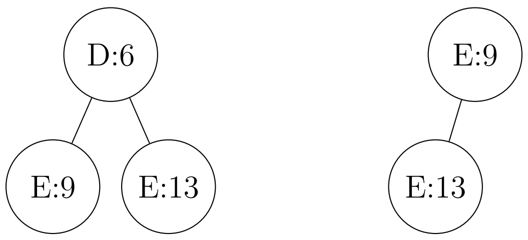


Figure 2.10: When E:9 is added to the tree (it doesn't sift-up at all)

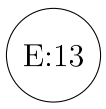
We now dequeue the next node: C:4. Node C has been visited so this is ignored. We then visit the next node D:6. The tree after each Dequeue() step looks like:



Now we set this node (D) as *cn*. *cn*.visited is now set to true. *cn*.order is assigned *order* = 4. *order* is incremented. Now we visit all of D's connections:

- (B, 1)
  cn.weight + 4 = 10 > 2. Continue

We now Dequeue the next node: E:9. The new tree is simply:



Now we set this node (E) as *cn*. *cn*.visited is now set to true. *cn*.order is assigned *order* = 5. *order* is incremented. Now we visit all of E's connections:

- (A, 13)
  cn.weight + 13 = 22 > 0. Continue
- (C, 6)
  cn.weight + 6 = 15 > 3. Continue

Finally we dequeue the last item of E:13. E has been already visited so this is ignored.

The algorithm is now complete and the boxes for each node look like:

A: A

1	0
0	

B: A,B

2	2
2	

C: A,B,C

3	3
4, 0	

D: A,B,D

4	6
6	

E: A,B,C,E

5	9
13, 9	

2.8 Minimum Spanning Trees

Both minimum spanning tree algorithms use the Graph class. This is how they find connections between nodes.

There is no input validation for this step as it is input using an undirected adjacency matrix. A disconnected graph error would arise when running the algorithms and this would be discovered when there are no more edges for an algorithm to use.

Generating a problem uses the Graph classes random graph generation for an undirected matrix. The difficulty varies the size of the adjacency matrix.

2.8.1 Prim's Algorithm

Prim's Algorithm works as briefly described in Section 1.4.2.3. Prim's algorithm will always start at node A as the exam board wants it. A flowchart on Prim's algorithm can be seen in Figure 2.11. This flowchart is slightly unconventional in that 2 for loops (specifically foreach) are encased in a process box but I think it still gets the point across.
The list *tree* stores a list of edges (in the form nodenode) and the weight of that edge. This would be used when displaying the tree later. *Visited* stores all nodes that have been visited. These form the current tree and so anything connecting to this would be added to this tree (no disjoint sets).

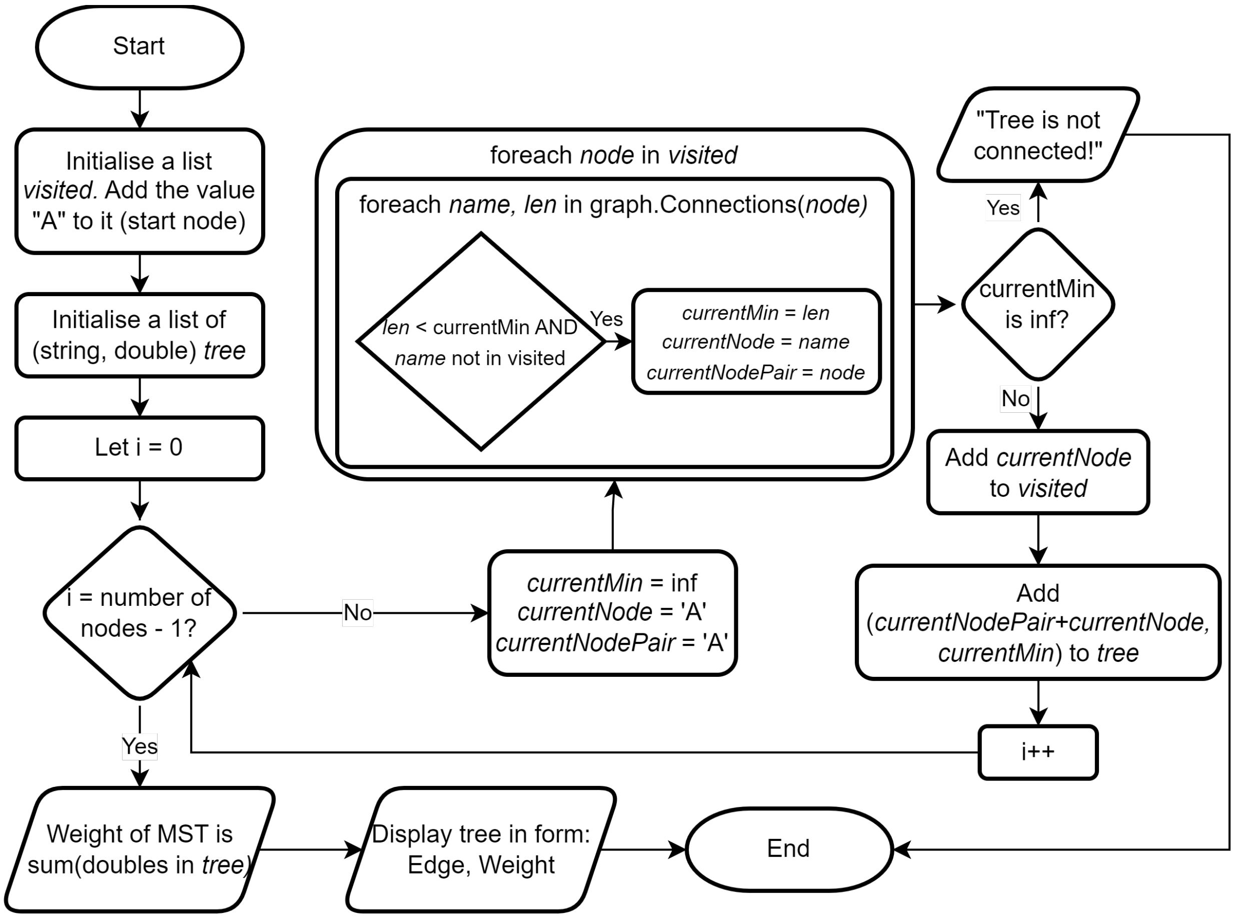

Start
Initialise a list visited. Add the value "A" to it (start node)
Initialise a list of (string, double) tree
Let i = 0
i = number of nodes - 1?
No
Yes
currentMin = inf
currentNode = 'A'
currentNodePair = 'A'
foreach node in visited
foreach name, len in graph.Connections(node)
len < currentMin AND name not in visited
Yes
currentMin = len
currentNode = name
currentNodePair = node
currentMin is inf?
Yes
"Tree is not connected!"
No
Add currentNode to visited
Add (currentNodePair+currentNode, currentMin) to tree
i++
Weight of MST is sum(doubles in tree)
Display tree in form: Edge, Weight
End

Figure 2.11: Slightly unconventional flowchart showing how Prim's algorithm would work in my program.

2.8.2 Kruskal's Algorithm

Kruskal's Algorithm works as described in Section 1.4.2.3. However, it is a bit more complex than this as now we have to consider disjoint sets.

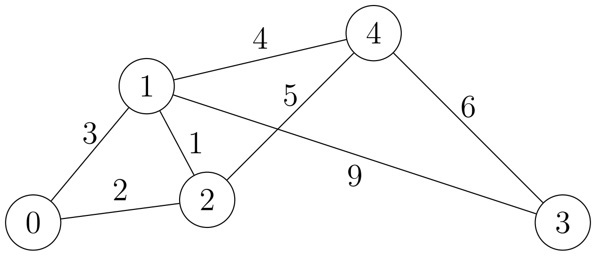
2.8.2.1 Disjoint Set Data Structure

The disjoint set data structure works by storing sets of nodes. The first thing to do is set up a parallel array of pointers (*set*) to the nodes. At the start, each node is in its own set (and so points to itself).
To add an edge to the disjoint sets (as to Unify() 2 sets), the program must first check whether or not the edge creates a cycle. This can be done by checking if the root node (the most parent node) of the set that the nodes are currently in are equal. If it is, they must be in the same set. To find this node we can use the recursive algorithm:

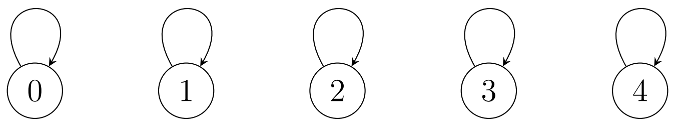
```
function FIND(node: int)
    if set[node] != node then
        return Find(set[node])
    end if
    return node
end function
```

The two nodes' root node can be compared and if they're the same, Unify() will return **false**. Otherwise, the new root of one of the edges points to the other's root and the function returns **true**.

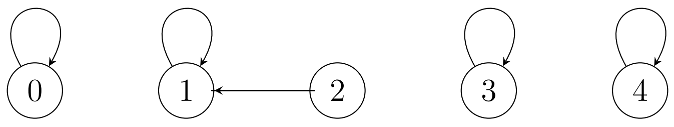
Here's an example using the following network:



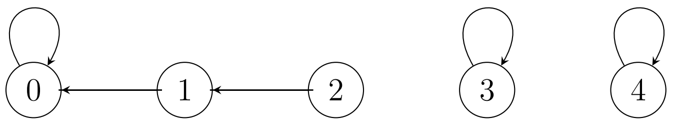
Take the nodes 0-4. The parallel array *set* = [0, 1, 2, 3, 4] has all nodes being their own parent so that they are all in separate sets (the diagram represents the pointers):



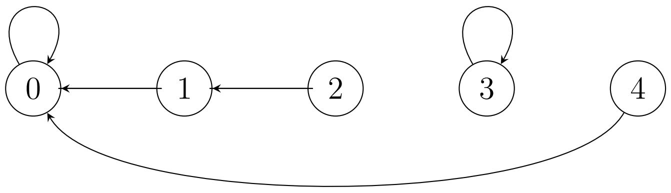
Now we want to add the edge 1-2. We first find the parents of both the nodes: Find(1) = 1, Find(2) = 2. Now that we've verified that they're different, we can unify the two sets and add a connection between 1 and 2. In this case we let 1 be the parent. Now *set* = [0, 1, 1, 3, 4]:



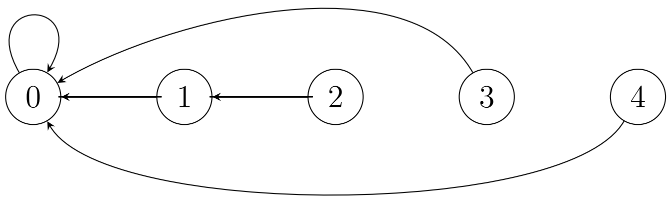
Now we want to add the edge 0-2. Find(0) = 0, Find(2) = 1. They're different, so add a connection between 0 and 2. In this case we let 0 be the parent. Now *set* = [0, 0, 1, 3, 4] as we make *set*[Find(2)] = Find(0):



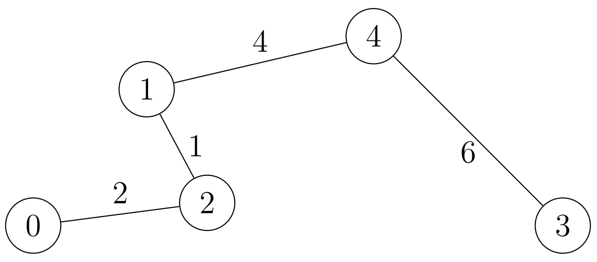
Now we want to add the edge 0-1. Find(0) = 0, Find(1) = 0. They're the same, so we return **false**.
Next we want to add the edge 1-4. Find(1) = 0, Find(4) = 4. We can add a connection between 1 and 4 now. *set*[Find(4)] = Find(1) so *set* = [0, 0, 1, 3, 0]:



Now we want to add the edge 2-4. Find(2) → Find(1) = 0, Find(4) = 0. They're the same, so we return **false**.
Next we want to add the edge 3-4. Find(3) = 3, Find(4) = 0. We can add a connection between 3 and 4 now. *set*[Find(3)] = Find(4) so *set* = [0, 0, 1, 0, 0]:



There are now $n-1$ pointers not pointing to themselves (all nodes are now in 1 set) or on the MST $n-1$ edges added. This means that the Minimum Spanning Tree is complete and Kruskal's can terminate (if there are not enough arcs for a minimum spanning tree, the algorithm terminates after using all abailable arcs). The minimum spanning tree using the example above may look something like:



2.8.2.2 The Algorithm

This works as described earlier. A flowchart (Figure 2.12) can be used to represent how the algorithm may be implemented in my program:

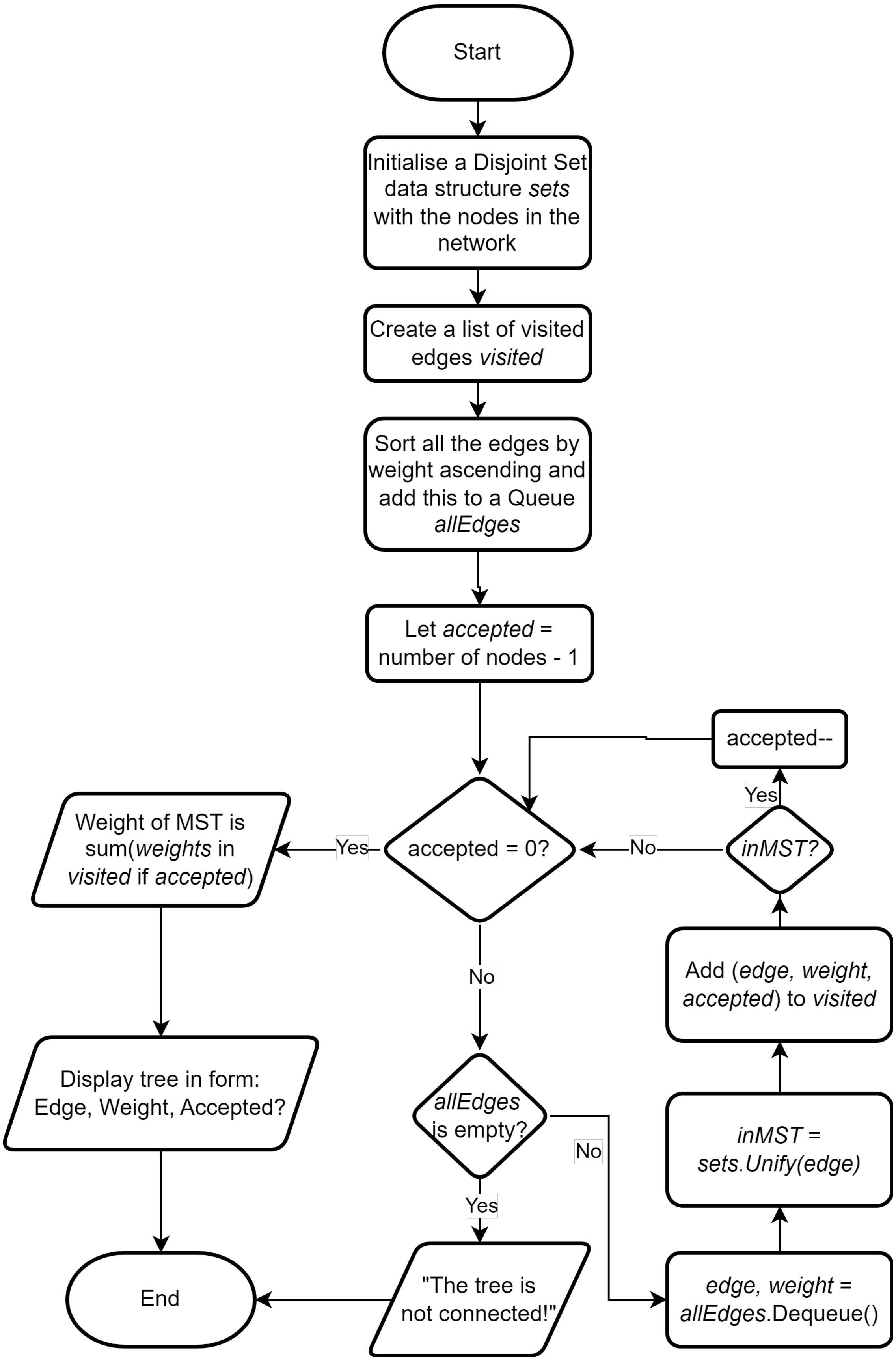

Start
Initialise a Disjoint Set data structure *sets* with the nodes in the network
Create a list of visited edges *visited*
Sort all the edges by weight ascending and add this to a Queue *allEdges*
Let *accepted* = number of nodes - 1
accepted--
Yes
Weight of MST is sum(*weights* in *visited* if *accepted*)
Yes
accepted = 0?
No
inMST?
No
Add (*edge, weight, accepted*) to *visited*
Display tree in form: Edge, Weight, Accepted?
allEdges is empty?
No
inMST = *sets.Unify(edge)*
Yes
End
"The tree is not connected!"
edge, weight = *allEdges*.Dequeue()

Figure 2.12: Flowchart for how I could implement Kruskal's Algorithm in my program

2.9 Quick Sort

The quick sort algorithm can be used to sort numbers or letters into ascending/descending order. The user could enter a list of numbers or characters separated by commas and then the program would choose the type of solver object to create, based off of the first item of the list's type.

The algorithm (as touched upon in Section 1.4.2.3) would have to be implemented iteratively rather than recursively. This is due to the nature of having to display the algorithm the way the exam board wants it. This means that the entire list is shown at each iteration of the algorithm. Otherwise, this wouldn't be possible. Pivots, solved and unsolved items would be colour coded accordingly.
To implement the algorithm iteratively:

1. The program finds each sub-list within the main list, separated by sorted values.
2. The first value of each sub-list is the pivot.
3. Each sub-list is sorted into lists of items less than or equal to the pivot or greater than the pivot. (The type of the element shouldn't matter as they should be able to be compared regardless as characters have values.)
4. If sorting ascending, they are concatenated as: "$\leqslant$" list, pivot, "$>$" list. Otherwise, the opposite way around.
5. Repeat these steps until there are no more sub-lists/all items are sorted.

Chapter 3

Testing

3.1 Tests

The testing video can be found at: `https://tinyurl.com/4xcatks9`

3.1.1 Main Objective Tests

ID	**Description** (Objective number)	**Input Data**	**Expected Outcome**	**Test Outcome**	**Pass/ Fail**	**Evidence**
1	5	Run the program	A menu should display when run	A menu is displayed when run	Pass	Video - 0:00
2	4	Select the tutorial option in the menu	An interactive tutorial is played	An interactive tutorial is played where equations and tables can be entered.	Pass	Video - 0:05
4	2.3.1	The problem in section 3.2.1	$\leqslant$ is reformulated as found in section 3.2.1	The problem is reformulated correctly.	Pass	Video - 2:27
5	2.5.1	The problem in section 3.2.1	Pivot row and columns are colour coded for each iteration.	Pivot row and columns are colour coded for each iteration.	Pass	Video - 2:27

6	2.5.2	The problem in section 3.2.1	Shows the ratio tests when calculating pivot rows as found in section 3.2.1	Shows the ratio tests when calculating pivot rows correctly	Pass	Video - 2:27
7	2.5.3	The problem in section 3.2.1 and selecting the option to show basic/non-basic variables	Shows the basic/non-basic variables after each iteration as found in section 3.2.1	Shows the basic/non-basic variables after each iteration as found in section	Pass	Video - 2:27
8	2.5.4	The problem in section 3.2.1 and selecting the option to show how each row is calculated	Shows how the next tableau is calculated as found in section 3.2.1	Shows how the next tableau is calculated using equations	Pass	Video - 2:27
9	2.5.5.1	The problem in section 3.2.1	Shows the feasible region highlighted as in Figure 3.3	Highlights the feasible region	Pass	Video - 2:27
10	2.5.5.2	The problem in section 3.2.1	Shows the the points where each iteration “reached” as in Figure 3.3	Shows the the points where each iteration “reached”	Pass	Video - 2:27

11	2.2	The problem in section 3.2.2	Shows each iteration of the tableau and the reaches the correct optimised result of $P = 72$ as shown in Section 3.2.2	Shows each iteration and reaches the correct optimised result	Pass	Video - 4:36
12	2.3.2	The problem in Section 3.2.2	$\geqslant$ is reformulated as found in section 3.2.2	$\geqslant$ is reformulated correctly	Pass	Video - 4:36
13	2.6	The incorrect problem in Section 3.2.3	The problem is found to be unsolvable and an exception is thrown.	The program states that the region is not bounded.	Pass	Video - 6:19
14	2.3.3	The correct problem in Section 3.2.3	$=$ is reformulated as found in section 3.2.3	The problem is solved correctly	Pass	Video - 7:02
15	2.6	The incorrect tableau in Section 3.2.4	The problem is found to be unsolvable and an exception is thrown	An exception is thrown, shown by displaying that the region is not bounded.	Pass	Video - 8:50
16	2.4	The correct tableau in Section 3.2.4	This should result in the same thing result as in Section 3.2.2	The correct values are shown as in 3.2.2	Pass	Video - 9:33

17	2.1, 1.1, 1.2, 1.3	A randomly generated Simplex problem of difficulty level 1	A 2 dimensional problem with solely $\leqslant$ constraints. This should result in the same as my workings found in the video.	A 2 dimensional problem with solely $\leqslant$ constraints is generated. My workings match those of the program's	Pass	Video - 11:00
18	1.1, 1.2	A randomly generated Simplex problem of difficulty level 5	This should randomly generate a problem of greater difficulty than in Test 17. This should have more dimensions and potentially be a 2 stage problem.	A 5 dimensional problem with both $\leqslant$ and $\geqslant$ constraints is generated.	Pass	Video - 12:00
19	2.7	All tests including a fully solved problem using tableaux.	All items in tableaux are rounded to 3.s.f.	All items held within tableaux are rounded to 3.s.f	Pass	Video - 2:27, 4:38, 8:00, 10:50, 11:20, 12:10

20	3.1, 3.3	The problem in Section 3.2.6, from node A	This should result in the workings found in Section 3.2.6, with the boxes for workings actually being boxes rather than in a table. The boxes should correspond to the values wanted by the exam board as detailed in Section 1.4.1.2	The start node of A is able to be chosen. Correct boxes are displayed.	Pass	Video - 12:20
21	3.1 Path check	The problem in Section 3.2.6	The correct shortest path to H is found as in Section 3.2.6	The correct path ABCFGH is found.	Pass	Video - 13:30
22	3.2.1	Entering the problem in Section 3.2.6	There should be an option to select whether the graph is directed. In this case it is undirected.	Option chosen.	Pass	Video - 12:33

23	3.2.2	Entering the problem in Section 3.2.6	Symmetry should be applied when entering this matrix such that values for an edge don’t need to be entered twice.	Symmetry is applied when entering the matrix.	Pass	Video - 12:36
24	3.1	The adjacency matrix in Section 3.2.5	The program should attempt to use Dijkstra’s Algorithm (from node A) and will not end up visiting C.	Node C is not visited when performing the algorithm.	Pass	Video - 14:07
25	3.2	When entering adjacency matrix in Section 3.2.5	The values connecting a node to itself can’t be accessed (through a “\”) to stop loops from being entered.	Values can’t be accessed. When navigating, the cursor skips over the “\”.	Pass	Video - 13:50
26	3.1, 1.1, 1.2, 1.3	A randomly generated Dijkstra’s problem of difficulty level 1	A reasonably small undirected adjacency matrix is generated. My workings in the video should match the output of the program’s.	A small undirected adjacency matrix is generated. My workings match those of the program’s.	Pass	Video - 14:14

27	1.1, 1.2	A randomly generated Dijkstra's problem of difficulty level 5/6	A larger adjacency matrix than that of Test 23 which may or may not be directed.	A larger matrix is generated. However, this matrix is undirected and so I generated another problem of difficulty level 6 which was directed to fully test the capabilities of the program.	Pass	Video - 15:05, 15:40

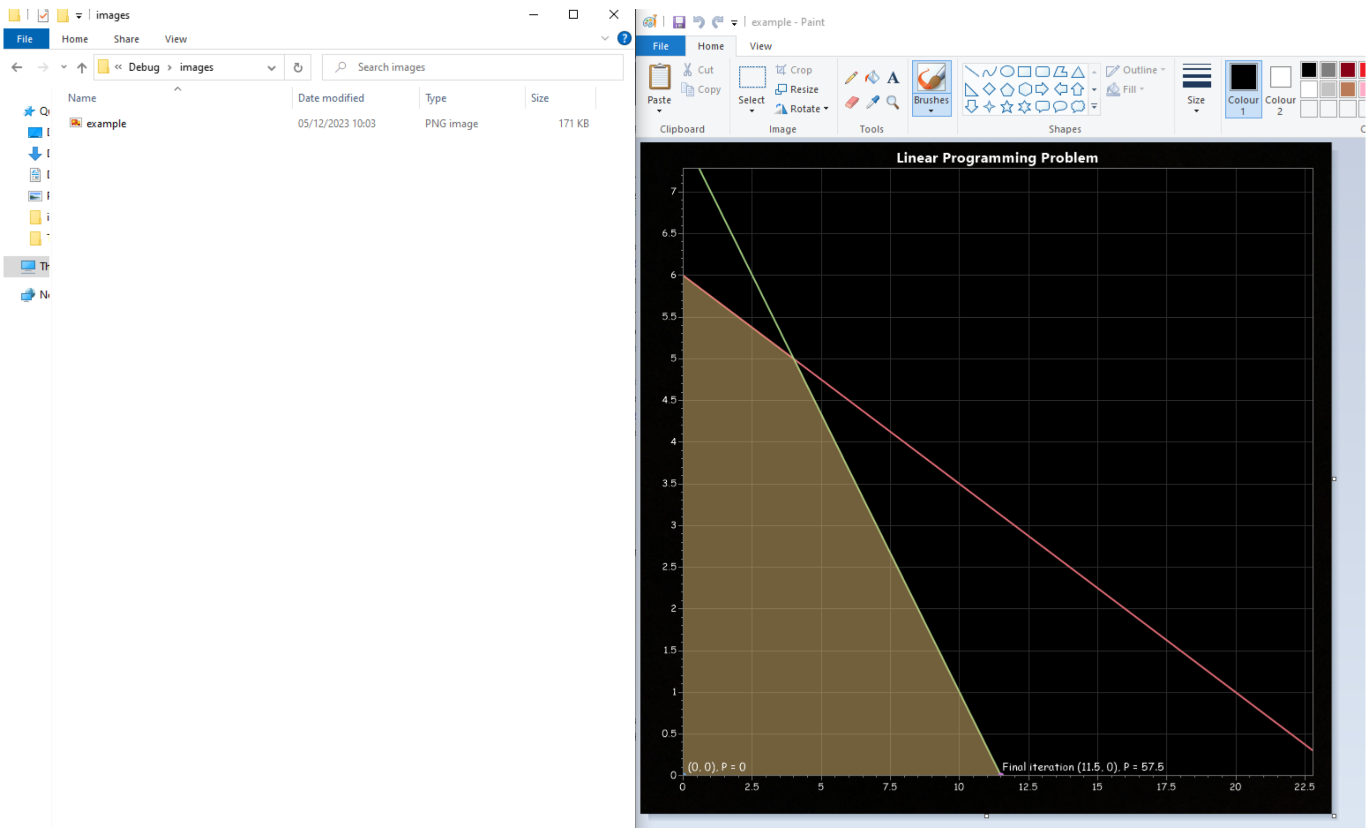
3.1.2 "If I have time" Objective Tests

ID	**Description** (Bonus objective number)	**Input Data**	**Expected Outcome**	**Test Outcome**	**Pass/ Fail**	**Evidence**
28	1.1	Entering the problem found in Section 3.2.7	Entering the adjacency matrix should be symmetrical.	The matrix is symmetrical when entered.	Pass	Video - 16:30

29	1.2	The problem found in Section 3.2.7	End up with the same workings for Prim's algorithm as in Section 3.2.7. The order in which the edges have been chosen should be shown.	Correct workings for Prim's algorithm.	Pass	Video - 17:38
30	1.3	The problem found in Section 3.2.7	End up with the same workings for Kruskal's algorithm as in Section 3.2.7. The order in which the edges are chosen should be shown, along with whether or not the edge has been accepted.	Correct workings for Kruskal's algorithm. Accepted edges are shown by a smiley face (☻) as not all consoles support check marks (✓)	Pass	Video - 17:38
31	1.1	The adjacency matrix found in Section 3.2.5	The error should be detected when running both Kruskal's and Prim's.	Error message "The tree is not connected!" is displayed.	Pass	Video - 18:18

32	Original objectives: 1.1, 1.2, 1.3	A randomly generated Prim's/ Kruskal's problem of low difficulty.	This should match my workings in the video.	A Kruskal's problem is generated and the workings match mine.	Pass	Video - 19:10
33	Original objectives: 1.1, 1.2, 1.3	A randomly generated Prim's/ Kruskal's problem of higher difficulty than of Test 28 (a larger adjacency matrix). This should be of the other algorithm than the one generated in Test 32.	This should match my workings in the video.	A Prim's problem with a large adjacency matrix is generated and the workings match mine.	Pass	Video - 20:00
34	4: Quick sort is applied to a list of numbers, descending.	The problem found in Section 3.2.8	This should match the workings in Section 3.2.8.	Workings match the program's	Pass	Video - 20:39
35	4: Quick sort is applied to a list of letters, ascending.	The problem found in Section 3.2.9	This should match the workings in Section 3.2.9.	Workings match the program's.	Pass	Video - 21:01
36	Original objectives: 1.1, 1.2, 1.3	A randomly generated Quick sort problem of low difficulty.	This should match my workings in the video.	The workings match.	Pass	Video - 21:30

37	Original objectives: 1.1, 1.2, 1.3	A randomly generated Quick sort problem of high difficulty.	This should be a longer list of elements than in Test 36. This should match my workings in the video.	The workings match.	Pass	Video - 21:47
38	5	Selecting "Yes" after a graph is displayed for the problem in Section 3.2.1.	An image of the graph displayed should be stored in an "images" subfolder.	An image of the graph displayed is stored in an "images" subfolder .	Pass	Figure 3.1. Video - 2:45



Name	Date modified	Type	Size
example	05/12/2023 10:03	PNG image	171 KB

Figure 3.1: The graph stored as an image in a subfolder and opened in paint.

3.1.3 Thoughts

All tests were passed.

Something I found when calculating the large Prim's algorithm problem (Test 33)

is that I made a mistake when performing Prim's algorithm myself on several attempts. These were cut from the video but the program helpfully made me realise my mistakes.
Test 32 was interesting as although the program correctly found an optimal solution, it didn't find all optimal solutions. This means that the program may not be the most helpful in this specific situation. A way to find all possible solutions could be to run a depth-first or breadth-first search on different possible solutions (by branching where there are equivalent lengths) and then take unique solutions.

3.2 Test Workings

3.2.1 Simplex Algorithm Problem

$$\begin{gathered}\text{Maximise: } 5x + 2y \\ \text{Subject to: } x + 4y \leqslant 24 \\ 2x + 3y \leqslant 23\end{gathered}$$

The program should be able to maximise a Simplex Algorithm problem (as above) and give the correct optimised result of P = 57.5 as seen below. The program should accept the constraints and reformulate them as below:

$$\begin{gathered}\text{Objective function: } P - 5x - 2y = 0 \\ \text{Subject to: } x + 4y + s_1 = 24 \\ 2x + 3y + s_2 = 23\end{gathered}$$

It should then be able to put these constraints into an initial tableau and perform iterations:

P	x	y	s_1	s_2	RHS
1	-5	-2	0	0	0
0	1	4	1	0	24
0	2	3	0	1	23

P	x	y	s_1	s_2	RHS
1	0	5.5	0	2.5	57.5
0	0	2.5	1	-1.5	12.5
0	1	1.5	0	1.5	11.5

Figure 3.2: Tableaux for test 1

The pivot column and row as in the initial tableau (column x, last row) should be highlighted in colour. Ratio tests should be shown next to the tableau (should be 24 for the 2nd row and 11.5 for the 3rd). To test objectives 2.5.3 and 2.5.4 they will both be selected. The row operations to get from the initial tableau should be (in order for each row):

$$\begin{gathered}\text{new row 1} = (\text{previous row 1}) + 5(\text{pivot row}) \\ \text{new row 2} = (\text{previous row 2}) - (\text{pivot row}) \\ \text{new row 3} = (\text{previous row 3})/2\end{gathered}$$

The basic variables in the initial tableau are: $P = 0$, $s_1 = 24$, $s_2 = 23$. x and y are non-basic.
For the final tableau, the basic variables are: $P = 57.5$, $x = 11.5$, $s_1 = 12.5$. y and s_2 are non-basic.
As this is a 2D problem the program should also produce a graph where points are highlighted as where an iteration "reached". The feasible region should also be highlighted.

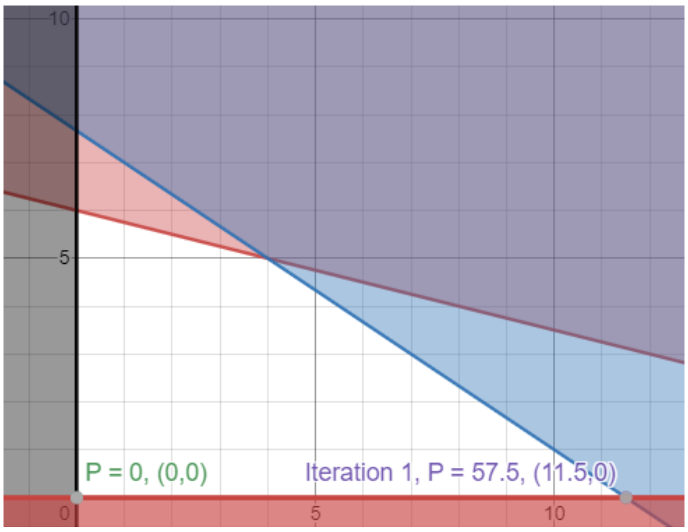


Figure 3.3: The problem represented graphically, where the feasible region is in white, and the points where each iteration "reached" are labelled. (Desmos)

3.2.2 2 Stage Simplex $\geqslant$ Problem

The program should also be able to maximise a 2 stage linear programming problem (objective 2.2). The problem below is the same as that found in section 1.4.2.1.

$$\begin{gathered}\text{Maximise: } P = 5x + 7y \\ \text{Subject to: } 2x + 3y \leqslant 30 \\ x + y \leqslant 12 \\ x \geqslant 2\end{gathered}$$

The Simplex algorithm would involve reformulating it as so:

$$\begin{gathered}\text{Objective function: } P - 5x - 7y = 0 \\ \text{Subject to: } 2x + 3y + s_1 = 30 \\ x + y + s_2 = 12 \\ x - s_3 + a_1 = 2 \\ \text{Second objective: } A + x - s_3 = 2\end{gathered}$$

Initial tableau:

A	P	x	y	s_1	s_2	s_3	a_1	RHS
1	0	1	0	0	0	-1	0	2
0	1	-5	-7	0	0	0	0	0
0	0	2	3	1	0	0	0	30
0	0	1	1	0	1	0	0	12
0	0	1	0	0	0	-1	1	2

Iteration 1:

A	P	x	y	s_1	s_2	s_3	a_1	RHS
1	0	0	0	0	0	0	-1	0
0	1	0	-7	0	0	-5	5	10
0	0	0	3	1	0	2	-2	26
0	0	0	1	0	1	1	-1	10
0	0	1	0	0	0	-1	1	2

Reduced tableau:

P	x	y	s_1	s_2	s_3	RHS
1	0	-7	0	0	-5	10
0	0	3	1	0	2	26
0	0	1	0	1	1	10
0	1	0	0	0	-1	2

Iteration 2:

P	x	y	s_1	s_2	s_3	RHS
1	0	0	$\frac{7}{3}$	0	$-\frac{1}{3}$	$\frac{212}{3}$
0	0	1	$\frac{1}{3}$	0	$\frac{2}{3}$	$\frac{26}{3}$
0	0	0	$-\frac{1}{3}$	1	$\frac{1}{3}$	$\frac{4}{3}$
0	1	0	0	0	-1	2

Iteration 3/ Final iteration:

P	x	y	s_1	s_2	s_3	RHS
1	0	0	2	1	0	72
0	0	1	1	-2	0	6
0	0	0	-1	3	1	4
0	1	0	-1	3	0	6

P is maximised at 72, when $x = 6$ and $y = 6$.

3.2.3 2 Stage Simplex = Problem

Another Simplex test for a problem involving an "=". The first thing to test is checking a region not being bounded (objective 2.6). e.g given the following problem:

$$\text{Maximise: } P = x + y$$
$$\text{Subject to: } x + y \leqslant 1$$
$$x + y = 2$$

It is evident that both constraints cannot be satisfied at once, thus this problem has no solution. The Simplex Algorithm will iterate and produce the same tableau over and over again and this should be picked up by the program and stopped.
The next test will be with a valid problem as below:

$$\text{Maximise: } P = 50x_1 + 40x_2 + 40x_3 + 30x_4$$
$$\text{Subject to: } x_1 + x_2 \leqslant 20$$
$$x_1 + x_2 + x_3 + x_4 = 40$$

These would be reformulated as:

$$\text{Objective function: } P - 50x_1 - 40x_2 - 40x_3 - 30x_4 = 0$$
$$\text{Subject to: } x_1 + x_2 + s_1 = 20$$
$$x_1 + x_2 + x_3 + x_4 + s_2 = 40$$
$$x_1 + x_2 + x_3 + x_4 + a_1 - s_3 = 40$$
$$\text{Second objective: } A + x_1 + x_2 + x_3 + x_4 - s_3 = 40$$

Initial tableau:

A	P	x_1	x_2	x_3	x_4	s_1	s_2	s_3	a_1	RHS
1	0	1	1	1	1	0	0	−1	0	40
0	1	−50	−40	−40	−30	0	0	0	0	0
0	0	1	1	0	0	1	0	0	0	20
0	0	1	1	1	1	0	1	0	0	40
0	0	1	1	1	1	0	0	−1	1	40

Iteration 1:

A	P	x_1	x_2	x_3	x_4	s_1	s_2	s_3	a_1	RHS
1	0	0	0	1	1	−1	0	−1	0	20
0	1	0	10	−40	−30	50	0	0	0	1000
0	0	1	1	0	0	1	0	0	0	20
0	0	0	0	1	1	−1	1	0	0	20
0	0	0	0	1	1	−1	0	−1	1	20

Iteration 2:

A	P	x_1	x_2	x_3	x_4	s_1	s_2	s_3	a_1	RHS
1	0	0	0	0	0	0	0	0	−1	0
0	1	0	10	0	10	10	0	−40	40	1800
0	0	1	1	0	0	1	0	0	0	20
0	0	0	0	0	0	0	1	1	−1	0
0	0	0	0	1	1	−1	0	−1	1	20

Reduced tableau:

P	x_1	x_2	x_3	x_4	s_1	s_2	s_3	RHS
1	0	10	0	10	10	0	−40	1800
0	1	1	0	0	1	0	0	20
0	0	0	0	0	0	1	1	0
0	0	0	1	1	−1	0	−1	20

Final tableau:

P	x_1	x_2	x_3	x_4	s_1	s_2	s_3	RHS
1	0	10	0	10	10	40	0	1800
0	1	1	0	0	1	0	0	20
0	0	0	0	0	0	1	1	0
0	0	0	1	1	−1	1	0	20

P is maximised at 1800, where $x_1 = 20$ and $x_3 = 20$.

Test	Pass/Fail
Incorrect constraints	
Correct constraints	

3.2.4 Simplex Tableau Problem

This is to test the entering of a tableau (objective 2.4). The first test is a tableau which shouldn't work and the program should say that the region is not bounded e.g. the tableau in Figure 3.4. [video timestamp]

P	x	y	s_1	s_2	RHS
1	0	0	0	2.5	8
0	0	0	0	0	9
0	0	0	0	0	10

Figure 3.4: Incorrect tableau

Then test a correct tableau such as the reduced tableau after the first iteration of the problem in Test 3 [video timestamp]:

P	x	y	s_1	s_2	s_3	RHS
1	0	-7	0	0	-5	10
0	0	3	1	0	2	26
0	0	1	0	1	1	10
0	1	0	0	0	-1	2

A tableau from one of the iterations during Section 3.2.2. This should then have the same output as 3.2.2.

3.2.5 Incomplete Adjacency Matrix

Below is an example of an incomplete adjacency matrix:

	A	B	C	D
A	$-$	2	$-$	$-$
B	2	$-$	$-$	4
C	$-$	$-$	$-$	$-$
D	$-$	4	$-$	$-$

When performing Dijkstra's from vertex A, vertex C won't be visited.
When performing an MST algorithm, the tree won't be connected so the algorithms cannot complete.

3.2.6 Dijkstra's Algorithm Problem

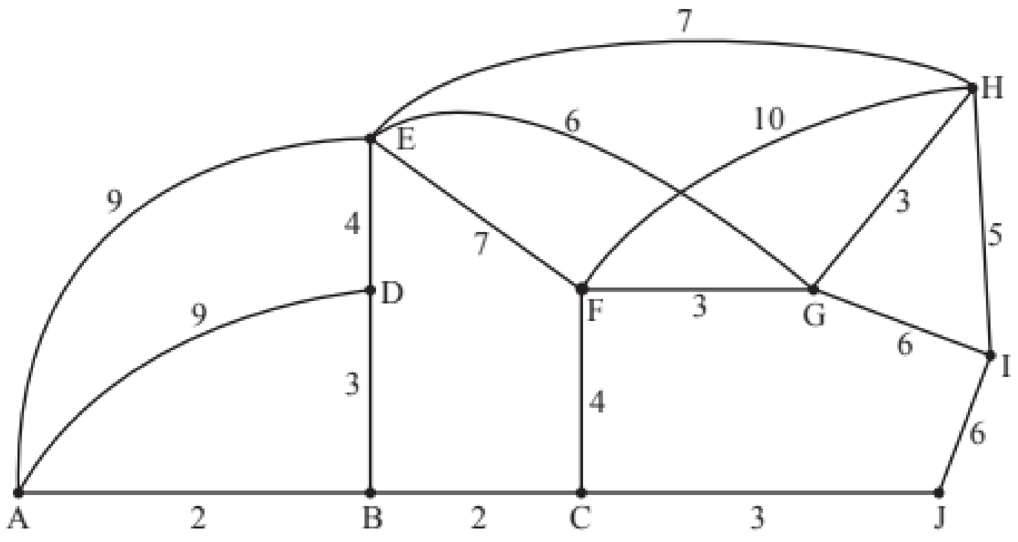
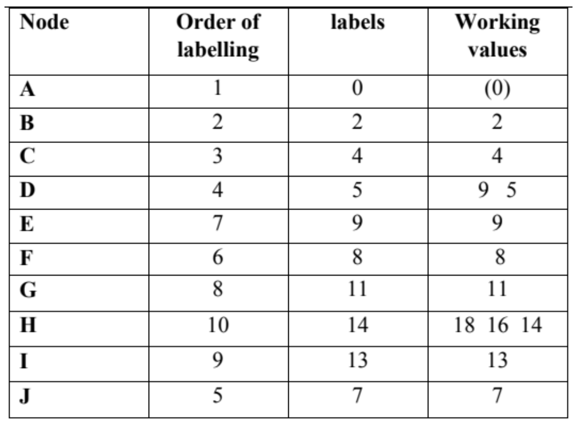


Figure 3.5: Example Dijkstra's problem (OCR MEI)

The graph in Figure 3.5 can be represented in an adjacency matrix like so:

	A	B	C	D	E	F	G	H	I	J
A	–	2	–	9	9	–	–	–	–	–
B	2	–	2	3	–	–	–	–	–	–
C	–	2	–	–	–	4	–	–	–	3
D	9	3	–	–	4	–	–	–	–	–
E	9	–	–	4	–	7	6	7	–	–
F	–	–	4	–	7	–	3	10	–	–
G	–	–	–	–	6	3	–	3	6	–
H	–	–	–	–	7	10	3	–	5	–
I	–	–	–	–	–	–	6	5	–	6
J	–	–	3	–	–	–	–	–	6	–

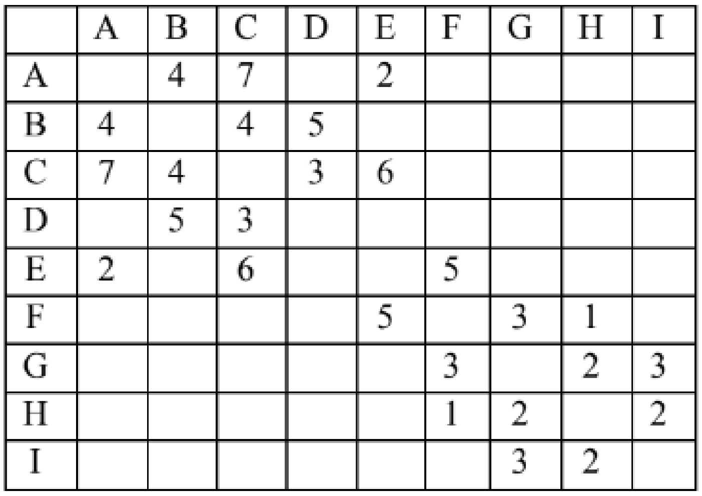
After Dijkstra's Algorithm has been applied from vertex A, it results in Figure 3.6. This question in particular from OCR MEI asks for the shortest path from A to H which is: ABCFGH.



Node	Order of labelling	labels	Working values
A	1	0	(0)
B	2	2	2
C	3	4	4
D	4	5	9 5
E	7	9	9
F	6	8	8
G	8	11	11
H	10	14	18 16 14
I	9	13	13
J	5	7	7

Figure 3.6: The boxes for Dijkstra's algorithm in a tabular format (OCR MEI)

3.2.7 MST Problem



	A	B	C	D	E	F	G	H	I
A		4	7		2				
B	4		4	5					
C	7	4		3	6				
D		5	3						
E	2		6			5			
F					5		3	1	
G						3		2	3
H						1	2		2
I							3	2	

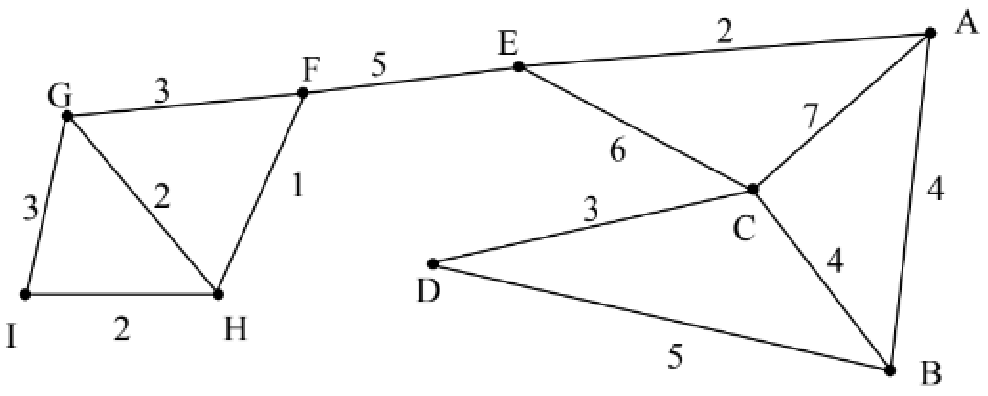


Figure 3.7: An example Minimum Spanning Tree Problem (OCR MEI)

Kruskal's Algorithm would find the MST as follows (note that the order in which arcs of equal weight are chosen does not matter (if there are rejected arcs with the same weight as the last checked arc, they might not appear as Kruskal's stops after findin the tree), nor which way round the letters of each arc are):

Arc	Weight	Accepted
FH	1	✓
AE	2	✓
GH	2	✓
HI	2	✓
CD	3	✓
FG	3	X
GI	3	X
AB	4	✓
BC	4	✓
BD	5	X
EF	5	✓

Starting from node A, Prim's algorithm would execute as follows:

Arc	Weight
AE	2
AB	4
BC	4
CD	3
EF	5
FH	1
HG	2
HI	2

These would form the MST of:

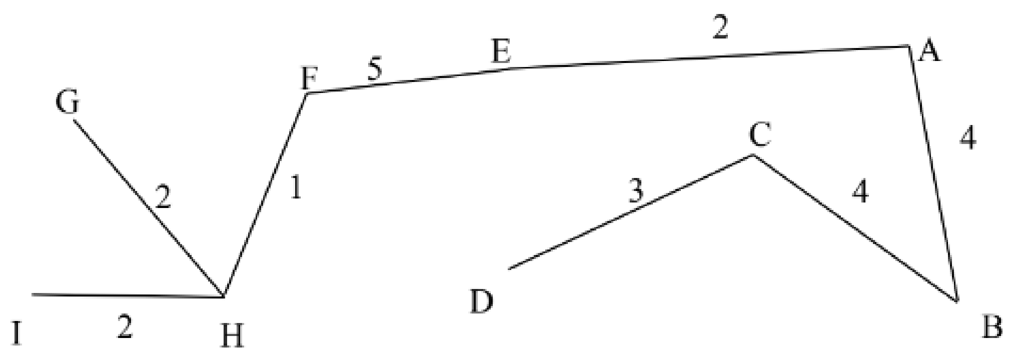


Figure 3.8: The MST of the above problem.

Which has total weight of 23.

3.2.8 Quick sort Numbers Descending

Take the following list of numbers: 24, 17, 9, 25, 18, 11, 23, 19, 30, 15
and sort it into descending order using the quick sort algorithm.

This would be sorted as follows (where each line is a new pass, pivots are in *italics* and sorted items are in **bold**):
24 17 9 25 18 11 23 19 30 15
25 30 **24** *17* 9 18 11 23 19 15
30 **25 24** *18* 23 19 **17** *9* 11 15
30 25 24 *23* 19 **18 17** *11* 15 **9**
30 25 24 23 *19* **18 17** *15* **11 9**
30 25 24 23 19 18 17 15 11 9

3.2.9 Quick sort Letters Ascending

Take the following list of letters: B, H, Z, Y, A, P, X
and sort it into ascending order using the Quick sort algorithm.
This would be sorted as follows (using the same style as in Section 3.2.8):
B H Z Y A P X
A **B** *H* Z Y P X
A B H *Z* Y P X
A B H *Y* P X **Z**
A B H *P* X **Y Z**
A B H P *X* **Y Z**
A B H P X Y Z

Chapter 4

Evaluation

4.1 Overall Effectiveness of the System

The original problem was that there is a lack of worked example solutions to MwA problems and mark schemes can be difficult to understand and so students don't actually understand how it works. Each type of problem in my solution can be randomly generated with varying difficulty. This is so that a user could start with a small problem and work their way up gradually to larger problems as their understanding increases. It also makes it so that there is an unlimited supply of problems. My solution focuses on the algorithm which causes students the most distress: Simplex. To help with understanding, Simplex problems display: pivots, ratio tests, row operations and basic variables. They can also show graphs for a 2D problem with a highlighted feasible region and where the algorithm "reaches" at each step. Other than Simplex, the program can also execute Dijkstra's, Kruskal's, and Prim's algorithms which are featured in the exam. The quick sort algorithm is also implemented with highlighted pivots and solved values to help with understanding.
Overall, the program is fairly decent at explaining MwA topics but there could be improvements made especially to non-Simplex algorithms.

4.2 Evaluation of Objectives

4.2.1 Main Objective Evaluation

1. The program should be able to generate practice questions of any of the types of problems within the program.

   1.1 The questions should be randomly generated.

   This has been achieved as each time a problem is generated it makes use of the Random object to generate some part of the problem.

   1.2 The questions should have various difficulties

   Each question has various different things which increase/decrease difficulty such as Simplex dimensions, size of a matrix, or length of a list.

   1.3 The questions should be able to be solved by the program

   All valid questions can be solved.

2. Simplex Algorithm

   2.1 Given any maximising Simplex Algorithm problem, the program will produce the correct optimised result

   1 stage problems can be solved correctly.

   2.2 Given a 2 stage Simplex Algorithm problem, the program will produce the correct optimised result

   2 stage problems can be solved correctly.

   2.3 The program is able to accept all constraints ($\leqslant, \geqslant, =$)

      2.3.1 $\leqslant$ is reformulated to an equation with 1 slack variable

      2.3.2 $\geqslant$ is reformulated to an equation with 1 artificial and 1 surplus variable

      2.3.3 $=$ is reformulated to 2 equations: 1 which is $\leqslant$ and then reformulated and the other which is $\geqslant$ and then reformulated (as above)

   These are all reformulated correctly and an equation can be entered easily.

   2.4 The program is able to accept an initial tableau

   The program can accept this through a matrix which is then converted into a tableau.

   2.5 The program shows workings

      2.5.1 Colour codes pivots (row, column)

      2.5.2 Shows the ratio tests when calculating pivot rows

      2.5.3 Basic / non-basic variables have an option to be shown

      2.5.4 An option to display how a row is calculated in each tableau

      2.5.5 A 2D problem (entered through constraints or randomly generated; **not** entered through a tableau) can be represented visually

         2.5.5.1 The feasible region will be highlighted

         2.5.5.2 The points where each iteration “reached” should be visible

   All of these objectives have been met.

   2.6 Non-bounded regions will be discovered.

   Constraints can be checked if they bound a region. If there are no constraints, a non-bounded region will be discovered through a repeated table.

   2.7 Values in tableaux are rounded to 3 significant figures.

   This uses the in-built scientific notation converter to round to 3.s.f.

3. Dijkstra’s Algorithm

   3.1 Given an adjacency matrix, the program should be able to produce the shortest distance from a node of the user’s choosing to all other nodes.

   The shortest distance is always found to each node (unless the network is not connected) and Dijkstra’s Algorithm is run correctly. However, as discussed in Section 3.1.3, there may always be more than 1 optimal solution.

3.2 The user should be able to input a matrix using their arrow keys.

3.2.1 The user should be able to input whether the graph is directed or not.

3.2.2 If the graph is undirected, automatically apply symmetry to the input matrix

Adjacency matrix can be entered easily with symmetry and this will become a graph for Dijkstra's to be performed on.

3.3 The program will display the boxes with workings, as displayed in the exam. The boxes with workings are displayed, along with the route which may be useful either for a question or for understanding.

4. Add a tutorial-like feature that explains how the program works.
   A tutorial has been added with interactive parts for the equations and tables/-matrices.

5. Add a menu to navigate the application
   A menu is able to be navigated with > as a pointer.

All of these objectives were met to get the program to meet its basic functionality.

4.2.2 "If I have time" Objective Evaluation

These are objectives that aren't necessary for the program but would benefit it greatly.

1. Add an option to use Prim's and Kruskal's Algorithms,

   1.1 The input would be handled the same as Dijkstra's Algorithm but graphs can only be undirected.
   This has been implemented.

   1.2 For Prim's Algorithm, the order that edges are chosen is shown.
   The order of the edges is shown along with the weights and total weight.

   1.3 For Kruskal's Algorithm, the order that edges are chosen is shown, along with whether or not the edge has been added to the Minimum Spanning Tree.
   The order of the edges is shown along with the weights and total weight and whether or not the graph was accepted (☻, X).

2. A minimisation option for Simplex Algorithm
   Due to time constraints, this has not been implemented. However, it is not crucial to the exam and therefore problem.

3. An integer linear programming option for Simplex Algorithm
   Due to time constraints, this has not been implemented. However, it is not crucial to the exam and therefore problem.

4. Add an option to use quick sort

   This has been implemented iteratively and can handle both numbers and letters (as seen in the exam).

5. Add an option to store the Simplex 2D graph as an image locally.

   This has been added and it is stored in images subfolder. This could be useful for later use of a problem and it was fairly easy to implement as it is just a feature in the library (ScottPlot).

It is not ideal that not all of them have been implemented but the ones I have implemented are well done I'd think.

4.3 End User Feedback

I sent each user a .zip file of the project and they just needed to extract and run the executable (.exe) inside.

The teacher commented on the project:
"The program works well. I like the different difficulty levels and different colors to highlight important sections. There is an excellent level of explanation of the stages of obtaining the answers with colors for Simplex and the feasible region popped up with a nice graph. The Simplex was done as we discussed. Well done!"

This is good as it shows that my program meets a teacher's expectations for explanation of the Simplex Algorithm. However, the teacher also commented on Dijkstra's Algorithm and quick sort which didn't quite have enough explanations. These weren't discussed in the first interviews but these could certainly be improved upon.

Students *a* and *b* both left feedback on each individual algorithm:

Regarding Simplex, Student *a* commented:
"The Simplex mode is very powerful - I find that it explains 2 step simplex very well, going through each step of simplex separately. I like the added touch of a graphical representation of the problem when possible - especially as this gives some nice intuition as to what the algorithm is actually doing. I would have liked the ability to be able to solve equality constraints via substitution however, but the program is still able to deal with them alternatively."

Student *b* said that it "works good" and was pleased with the options of showing row operations and basic variables, the graph (and its colours!), and being able to save the graph. Some improvements he added were to possibly be able to go back and edit an input that caused an error and also to maybe include the constraints when saving the graph as an image.

For Dijkstra's Algorithm student *a* found it "easy to use" and the path to each node helpful. As an added touch he would've also liked to see a step-by-step guide on using Dijkstra's Algorithm. Student *b* again said that it "works good" and also raised the point of displaying it graphically but wasn't concerned that the program didn't whatsoever.

There wasn't much to say about minimum spanning tree algorithms. Student *b* again for the second time said that it "works good" but had no idea what the smiley face meant (☻). This can be easily remedied by adding 1 line of text in the tutorial. Student *a* said:
"The minimum spanning tree mode is perfect! - the step by step nature of both Prim's algorithm and Kruskal's Algorithm is very easy to follow. The way in which the interface automatically ensures the graph is undirected is also very helpful - it makes typing in the adjacency matrix without mistakes far easier."

The quick sort mode received was as follows: Student *b* said (for the 4th time!) that it "works good" and enjoyed the colour coding. He did provide an improvement of having less freedom in the input as he didn't like being able to put in numbers and letters together (and so sort by ASCII value) but I don't think that this is particularly relevant as the program handles just numbers or just letters fine (which is all of the possible inputs the exam board can give) and this is just additional functionality. Student *a* found it easy to use and as an extra feature would've liked to be able to set restrictions on the random generation as he found lists of letters particularly challenging and wanted longer lists (harder difficulties) to not include them.

Both students also left general feedback as well:
"In general, the program is very well made and easy to use. The user interface makes it very easy to enter in any kind of information needed, be it menu options, numbers, inequalities or adjacency matrices: all features (while intuitive anyway) are explained in the tutorial - I especially like the interactive aspects of the tutorial as they make clear what the the tutorial means, while engaging the user's interest. Further Maths Modelling With Algorithms-wise, the program seems to be very feature packed. While it is not a completely comprehensive tool for the module (e.g. I would have liked to see some bin packing and maximum flow problems), it does every feature it offers very well. The combination of problem generation and the problem solvers works very well - allowing for easy walkthrough of any possible problem, while also helping create new ones for practice." - Student *a*

Student *b* really liked the colours for readability (e.g. when displaying a table each row always alternates between grey and cyan), the interface design in menus, and entering the data through equations or a table. One bit of feedback he left was that "pressing a key (e.g. enter) multiple times while another process is happening (such as the text is displaying) puts the keypresses into a queue, and so immediately skips to the next section when the process is complete." This should be preventable through the use of C#'s Console.KeyAvailable which would stop a keypress from registering while things are happening. The last criticism was that he didn't like to repeat the tutorial each time he wanted to review a certain feature.

"Overall, the program has fit its purpose well, with all necessary functionality. However, in my opinion, there are decent few quality of life improvements that would make the experience a bit nicer." - Student *b*.
I agree with his statement that it has fit its purpose but both the teacher and student *a* have asked for some slightly more useful improvements:

4.4 System Improvements

As student *a* mentioned there are several quality of life improvements I could make:

- Saving an incorrect input:
  This could be achieved by having a sort of store of the last few problems you have done which save the type of problem along with the input. The input could then be edited until the program closes.

- Splitting up the tutorial:
  Abstracting each different part of the tutorial into separate subroutines and then having a menu to select certain parts would work. I could then also have an option to play the whole tutorial which would sequentially execute each subroutine.

- Pushing a key while another process is happening:
  Console.KeyAvailable is a Boolean and so this can be achieved by simply using it as the condition of a while loop so that keys can't be registered during the while loop.

- Adding constraints to a saved graph:
  Student *b* suggested either displaying the equations on the image or having them in a separate text file. I don't think that putting them in a separate text file is necessary as it isn't too difficult to place a list of equations in the top right corner of the image (where lines are far less likely to lie). Another way to handle this would be to display the equations of the line on the line but I think that this could get confusing with other lines being in the way.

- Adding restrictions to randomness:
  This was suggested by student *a*. This could be implemented by having a menu of options (e.g. number of $\leqslant$ constraints) in each topic and being able to change the amounts using a scroller menu. This would be separate from the difficulty settings as the difficulty would be determined by the constraints the user sets.

The main improvement raised was to implement a graph for Dijkstra's Algorithm.

A way to approach this problem could be to use the Microsoft Automatic Graph Layout (MSAGL [1]) library in C#. This would launch an interactive windows form (just like Scottplot) and would display a network. The library is able to show directed edges and nodes are fully customisable (such as a Dijkstra box). The library however comes with it's own Graph class and so I would have to figure out a way to integrate that with my graph class (and eventually rename my class). This library would also be particularly useful for displaying the MST of a network as well as its complete self.

Student *a* would've also liked to potentially see an additional 2 topics: bin packing [5] and network flow.

Bin packing was the least voted for option in Figure 1.8 but it is something that I could implement. There are 3 main algorithms that are needed in the exam: first-fit, first-fit decreasing, first-fit ascending. These are all the same algorithm but the decreasing and ascending versions sort the list first. Bin packing goes hand-in-hand with quick sort due to this and a typical exam question is to sort a list of numbers and then perform first-fit on it. First-fit is an online algorithm such that it doesn't matter what the next item it, it just focuses on the current item to be placed. First-fit works as follows:

1. Take a list of numbers and an infinite amount of bins of size n. Starting from the first item:
2. Place the item into the first available bin such that the total in the bin is $\leqslant n$.
3. Increase the current capacity of the bin by the number.
4. Move onto the next item and go back to step 2.

There is a rare exam question in which students are asked to find an optimal packing. Usually in the exam, the optimal solution will be easy to spot. This problem cannot be solved in polynomial time (as it is NP-complete [10]) and so for large(ish) inputs it doesn't make sense to implement. This is why heuristic algorithms such as first-fit are used. There are various different algorithms that can be implemented to solve this as found in: `https://en.wikipedia.org/wiki/Bin_packing_problem#Exact_algorithms`.
In all, I find that bin-packing would not be a bad idea to implement into the project as it is still useful to students (even if less-so than other algorithms already in the program).

The other type of algorithm suggest by student a is network flow. This problem involves finding the maximum flow through a network given one or more source nodes and one or more sink nodes connecting a directed graph. If there are multiple source and sink nodes, they will be combined to make a supersource and supersink which act as one source/sink for the network. I could use my current Graph class to implement this into my program. There are many algorithms that can be used to find the maximum flow but in the exam we aren't really taught to use a specific one (a tiny bit like trial and improvement but a lot of spotting).
Another thing with network flow is finding cuts. A cut separates source nodes from sink nodes by "cutting" straight through edges on a network. This separates certain nodes from other nodes. A cut's weight is just the sum of the flow through the cut and out of the source node partition. A useful thing to implement would be to find the minimum cut (which is typical in an exam) and then by max-flow min-cut theorem it is also the maximum flow. In an exam however, cuts are found by spotting and algorithms aren't used.
Overall, a network flow problem could be implemented into the solution as it is in the exam, but I'm not sold on the effectiveness that my program would bring to help students out.

Chapter 5

Technical Solution

5.1 Directory of technical skills

This is for the most complex bits of code.

Skill	Location
Interface	Section 5.2.1 (IProblem Interface)
Randomly generating Simplex problems	5.2.4 (creator class)
Simplex Algorithm (1 and 2 stage)	Section 5.2.4 (solver class)
Simple custom exception	Section 5.2.4 (RegionNotBoundedException)
Graphing a 2D problem (Simplex)	Section 5.2.4 (2D graph class)
Graph/Network implementation	Section 5.2.3 (Graph class)
Implementation of Dijkstra's Algorithm	Section 5.2.5 (solver class)
Priority Queue	Section 5.2.5 (Priority Queue)
Implementation of Kruskal's Algorithm	Section 5.2.6 (solver class: lines 7–51)
Disjoint Sets	Section 5.2.6 (Disjoint Set class)
Implementation of Prim's Algorithm	Section 5.2.6 (solver class: lines 52-91)
Iterative implementation of quick sort	Section 5.2.7 (solver class)

5.2 Code

5.2.1 Main Program

```
private static void DoProblem(Random rnd, IProblem problem, bool
    userInput)
{
    if (userInput) problem.InputQuestion();
    else problem.GenerateQuestion(rnd);

    if (new Menu(new List<string>() { "Solve!", "Exit"}).
        SelectOption() == 0)
    {
        problem.GenerateAnswer();
    }
}
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.UTF8;
    Random rnd = new Random();
    List<string> types = new List<string>() { "Simplex", "
        Dijkstra", "Minimum Spanning Tree", "QuickSort", "
        Tutorial", "Exit" };

    do
    {
        Console.WriteLine("Main menu");
        int type = new Menu(types).SelectOption();
        if (type == 5) break;
        ConsoleHelper.ClearLine(Console.CursorTop - 1);
        if (type == 4) Tutorial.StartTutorial();
        else
        {
            int enter = new Menu(new List<string>() { "Generate
               new problem", "Enter your own problem" }, back:
               true).SelectOption();
            if (enter == -1) continue;
            switch (type)
            {
                case 0:
                    DoProblem(rnd, new SimplexCreator(), enter ==
                         1);
                    break;
                case 1:
                    DoProblem(rnd, new DijkstraCreator(), enter
                        == 1);
                    break;
                case 2:
                    DoProblem(rnd, new MSTCreator(), enter == 1);
                    break;
                case 3:
                    DoProblem(rnd, new QuickSortCreator(), enter
                        == 1);
                    break;
            }
        }
        // Leave a gap in the console between this and last
           question, making sure the whole last question stays
           in the console history
        ConsoleHelper.ClearLine(Console.CursorTop - 1);
        Console.WriteLine("--------------------");

        // The first option is compatible with the school console
           , the other option is compatible with all other
           consoles
        try
        {
            Console.SetWindowPosition(0, Console.CursorTop);
        }
        catch (ArgumentOutOfRangeException e)
        {
            for (int i = 0; i < Console.BufferHeight; i++)
            {
                Console.WriteLine();
            }
            Console.CursorTop = 0;
        }
    } while (true);
}
```

IProblem Interface:

```
public interface IProblem
{
    void GenerateQuestion(Random rnd);
    void GenerateAnswer();
    void InputQuestion();
}
```

5.2.2 Static classes

Maths class:

```
public static class myMath
{
    public static readonly char[] subscriptNums = new char[] { '₀', '₁', '₂', '₃', '₄', '₅', '₆', '₇', '₈', '₉' };
    public const int sigFigPrecision = 3;
    private const double EPSILON = 1e-12;

    public static string GetSubscript(int num)
    {
        string sub = "";
        foreach(char digit in num.ToString())
        {
            sub += subscriptNums[digit - '0'];
        }
        return sub;
    }
    public static bool WithinPrecision(double value1, double value2) => Math.Abs(value1 - value2) < EPSILON;
    public static double SigFig(double num, int significantFigures = sigFigPrecision)
    {
        if (num >= 0 - EPSILON && num <= 0 + EPSILON) return 0;

        return double.Parse(num.ToString($"G{significantFigures}"));
    }
    public static double Argument(double a, double b)
    {
        double pi = Math.PI;

        // Imaginary axis
        if (a == 0 && b > 0) return pi / 2;
        if (a == 0 && b < 0) return -pi / 2;

        // top-left quadrant, bottom-left qudarant, right 2 quadrants
        if (a < 0 && b >= 0) return Math.Atan(b / a) + pi;
        if (a < 0 && b < 0) return Math.Atan(b / a) - pi;
        return Math.Atan(b / a);
    }
}
```

Console Helper:

```
public static class ConsoleHelper
{
    public static void ClearLine(int height, int left = 0)
    {
        if (height < 0) height = 0;
        Console.SetCursorPosition(left, height);
        Console.WriteLine(new string(' ', Console.BufferWidth - left));
        Console.SetCursorPosition(left, height);
    }
    public static void ClearLines(int top, int bottom = -1)
    {
        if (bottom == -1) bottom = Console.BufferHeight - 1;
        for (int i = top; i < bottom; i++) ClearLine(i);
        Console.SetCursorPosition(0, top);
    }
    public static void ColourText(ConsoleColor colour, ConsoleColor originalColour, string text, bool slowText = false)
    {
        Console.ForegroundColor = colour;
        if (slowText) SlowText(text);
        else Console.Write(text);
        Console.ForegroundColor = originalColour;
    }
    public static void SlowText(string text)
    {
        for (int i = 0; i < text.Length; i++)
        {
            Console.Write(text[i]);
            Thread.Sleep(15);
        }
        Console.WriteLine();
    }
    public static void WaitForKey(string message = "Press any key to continue")
    {
        Console.WriteLine(message);
        Console.ReadKey(true);
    }
}
```

Menu:

```
public class Menu
{
    public List<string> items;
    private char pointerType;
    bool back;

    public Menu(List<string> items, char pointer = '>', bool back = false) => (this.items, pointerType, this.back) = (items, pointer, back);

    public void ChangePointer(char newPointer) => pointerType = newPointer;

    public string Select(bool returnItemPos = false)
    {
        Console.CursorVisible = false;

        foreach (string item in items)
        {
            Console.WriteLine($" {item}");
        }

        int maxOption = items.Count();
        int top = Console.CursorTop;
        int option = 0;
        Console.CursorTop = option + top - maxOption;
        Console.CursorLeft = 0;
        Console.Write(pointerType);
        do
        {
            ConsoleKeyInfo choice = Console.ReadKey(true);

            if (choice.Key == ConsoleKey.DownArrow && option <
                maxOption - 1)
            {
                Console.CursorTop = option + top - maxOption;
                Console.CursorLeft = 0;
                Console.Write(" ");
                option++;
                Console.CursorTop = option + top - maxOption;
                Console.CursorLeft = 0;
                Console.Write(pointerType);
            }
            else if (choice.Key == ConsoleKey.UpArrow && option >
                0)
            {
                Console.CursorTop = option + top - maxOption;
                Console.CursorLeft = 0;
                Console.Write(" ");
                option--;
                Console.CursorTop = option + top - maxOption;
                Console.CursorLeft = 0;
                Console.Write(pointerType);
            }
            else if (choice.Key == ConsoleKey.Enter)
            {
                // erase the menu
                ConsoleHelper.ClearLines(top - maxOption, top);
                Console.CursorVisible = true;

                // return the choice
                return returnItemPos? option.ToString() : items[
                    option];
            }
            else if (back && (choice.Key == ConsoleKey.Escape ||
                choice.Key == ConsoleKey.LeftArrow))
            {
                ConsoleHelper.ClearLines(top - maxOption, top);
                Console.CursorVisible = true;
                return "-1";
            }
        } while (true);
    }
    public int SelectOption()
    {
        return int.Parse(Select(true));
    }
}
```

Type of menu that scrolls through options:

```
public class OptionScroller
{
    private string text;
    private int low, high;
    private bool ascii;

    public OptionScroller(string text, int lowestValue = int.MinValue, int highestValue = int.MaxValue, bool ascii = false) => (this.text, low, high, this.ascii) = (text, lowestValue, highestValue, ascii);

    public int Select()
    {
        Console.CursorVisible = false;
        int top = Console.CursorTop + 1;
        int currentOption = low == int.MinValue? 0 : low;
        do
        {
            Console.CursorTop = top;
            Console.Write($"{text}{(ascii ? ((char)(currentOption + 65)).ToString() : currentOption.ToString())}");
            if (currentOption != high)
            {
                Console.CursorTop = top - 1;
                Console.CursorLeft = text.Length + (int)Math.Round((double)currentOption.ToString().Length / 2);
                Console.Write("Λ");
            }
            if (currentOption != low)
            {
                Console.CursorTop = top + 1;
                Console.CursorLeft = text.Length + (int)Math.Round((double)currentOption.ToString().Length / 2);
                Console.Write("V");
            }

            ConsoleKeyInfo key = Console.ReadKey(true);
            if (key.Key == ConsoleKey.UpArrow && currentOption < high) currentOption++;
            else if (key.Key == ConsoleKey.DownArrow && currentOption > low) currentOption--;
            else if (key.Key == ConsoleKey.Enter)
            {
                ConsoleHelper.ClearLines(top - 1, top + 2);
                Console.CursorVisible = true;
                return currentOption;
            }
            ConsoleHelper.ClearLines(top - 1, top + 2);
        } while (true);
    }
}
```

Tutorial:

```
public static class Tutorial
{
    public static void StartTutorial()
    {
        ConsoleHelper.SlowText("A Level further maths ‘Modelling with Algorithms’ tool tutorial");
        ConsoleHelper.WaitForKey();

        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "\nEntering Equations:", true);
        ConsoleHelper.SlowText("To enter an equation you will see something like this:\n_x₁ + _x₂ <= _");
        ConsoleHelper.SlowText("You can enter in numbers where there are underscores \"_\"");
        ConsoleHelper.ColourText(ConsoleColor.Red, ConsoleColor.Gray, "Note that: any underscores left in the equation will automatically be parsed as '0'", true);
        ConsoleHelper.SlowText("You can move around in the equation by pressing tab/enter/right arrow to move right and left arrow to move left.");
        ConsoleHelper.SlowText("If you press enter while you're editing the last variable or press \'esc\' at any point, it will stop editing the equation.");
        ConsoleHelper.ColourText(ConsoleColor.Magenta, ConsoleColor.Gray, "Have a go now!", true);

        string[] tutorialArray = new string[] {"x", "y"};
        double[] tutorialValues = new EquationEnter(new string[] { "x + ", "y <=", "" }).Select();
        Console.WriteLine(new Equation(tutorialArray, tutorialValues.Take(2).ToArray(), "<=", tutorialValues[2]));
        ConsoleHelper.WaitForKey();

        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "\nEntering tables", true);
        ConsoleHelper.SlowText("To enter a table (such as a Simplex Tableau or an adjacency matrix) you will see something like this:");
        ConsoleHelper.SlowText(@"
|A|B |C|D|
|_| _|_|_|
|_| _|_|_|
Enter");
        ConsoleHelper.SlowText("The same logic applies as when entering an equation.");
        ConsoleHelper.SlowText("The only difference is that pressing enter moves you down a column instead.");
        ConsoleHelper.SlowText("To exit you can either hit escape or navigate down to the \"Exit\" button and press enter.");
        ConsoleHelper.ColourText(ConsoleColor.Red, ConsoleColor.Gray, "If you see a \"/\" this means that box cannot be filled in (such as to prevent loops in a graph)\n");
        ConsoleHelper.ColourText(ConsoleColor.Magenta, ConsoleColor.Gray, "Have a go now!", true);

        Graph tutorialGraph = new Graph(5);
        tutorialGraph.InputGraph();
        Console.WriteLine(tutorialGraph);
        ConsoleHelper.WaitForKey();

        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "\nThe Simplex Algorithm", true);
        ConsoleHelper.ColourText(ConsoleColor.Magenta, ConsoleColor.Gray, "Pivot rows are displayed in
magenta.\n");
        ConsoleHelper.ColourText(ConsoleColor.Yellow, ConsoleColor.Gray, "Pivot columns are displayed in yellow.\n");
        ConsoleHelper.SlowText("Basic variables and row operations can be turned on/off before running.");
        ConsoleHelper.SlowText("The program accepts both constraints (entered through an equation) and a tableau (entered through a table)");
        ConsoleHelper.WaitForKey();

        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "\nDijkstra's Algorithm", true);
        ConsoleHelper.SlowText("You can generate a question or add a graph through an adjacency matrix.");
        ConsoleHelper.SlowText("You can select whether or not the graph is undirected or directed.");
        ConsoleHelper.SlowText("The program will display the boxes you find in the exam as well as the route to each node.");
        ConsoleHelper.WaitForKey();

        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "\nMST algorithms", true);
        ConsoleHelper.SlowText("The same as Dijkstra's algorithm but the graph has to be undirected and you can enter negative values.");
        ConsoleHelper.SlowText("You can choose to compute Prim's or Kruskal's (or both!)");
        ConsoleHelper.WaitForKey();

        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "\nQuickSort Algorithm", true);
        ConsoleHelper.ColourText(ConsoleColor.Red, ConsoleColor.Gray, "Pivots are displayed in red.\n");
        ConsoleHelper.ColourText(ConsoleColor.Green, ConsoleColor.Gray, "Sorted values are displayed in green.\n");
        ConsoleHelper.SlowText("You can sort using characters or numbers.");
        ConsoleHelper.WaitForKey("Press any key to exit.");
    }
}
```

5.2.3 Graphs/Matrices

Entering a matrix

```
public class MatrixEnter
{
    private double[,] matrix;
    private string[] columns;
    bool graph, symmetry, negative;
    public MatrixEnter(double[,] matrix, string[] columns, bool isGraph = false, bool symmetrical = false, bool negative = true) =>
        (this.matrix, this.columns, graph, symmetry, this.negative) = (matrix, columns, isGraph, symmetrical, negative);

```

```csharp

    private int WriteOut(int top, int col, int row, string[,] currentValues, int buffer, bool setup = false)
    {
        int startLeft = 0;
        List<string> lines = new List<string>();

        // find largest size in each column
        int[] largestSize = new int[columns.Length];
        for (int i = 0; i < largestSize.Length; i++)
        {
            largestSize[i] = columns[i].Length;
            for (int j = 0; j < matrix.GetLength(0); j++)
            {
                largestSize[i] = Math.Max(largestSize[i], currentValues[j, i].Length);
                if(i == col && j == row) largestSize[i] = Math.Max(largestSize[i], currentValues[j, i].Length + 2);
            }
        }
        lines.Add("|".PadLeft(graph? 2 : 1));

        if (graph) foreach (string column in columns) lines.Add($"{column}|");
        else for (int i = 0; i < matrix.GetLength(0); i++) lines.Add("|");

        for (int i = 0; i < matrix.GetLength(1); i++)
        {
            lines[0] += $"{columns[i]}".PadLeft(largestSize[i]) + "|";
            for (int j = 0; j < matrix.GetLength(0); j++)
            {
                if (j == row && i == col)
                {
                    lines[j + 1] += $" {$"{currentValues[j, i]}".PadLeft(largestSize[i] - 2)} |";
                    startLeft = lines[j + 1].Length - 2 - buffer;
                }
                else lines[j + 1] += $"{(currentValues[j, i])}".PadLeft(largestSize[i]) + "|";
            }
        }
        Console.CursorTop = top;
        Console.CursorLeft = 0;
        for (int i = 0; i < lines.Count; i++)
        {
            Console.WriteLine(lines[i]);

            // colouring for visibilty
            if (i % 2 == 0) Console.ForegroundColor = ConsoleColor.Cyan;
            else Console.ForegroundColor = ConsoleColor.Gray;
        }
        Console.WriteLine($"{(row == matrix.GetLength(0)? ">" : " ")}Enter");
        Console.ForegroundColor = ConsoleColor.Gray;

        // On some consoles, this gets around moving the matrix
           down a line each time when the matrix goes out of the
            buffer height
        if (setup && top != Console.CursorTop - (matrix.GetLength
           (0) + 2))
        {
            top = Console.CursorTop - (matrix.GetLength(0) + 2) -
                 1;
            ConsoleHelper.ClearLine(Console.CursorTop);
        }

        Console.SetCursorPosition(startLeft, row + top + 1);
        return top;
    }
    private string[,] SetUpCurrentValues()
    {
        string[,] currentValues = new string[matrix.GetLength(0),
            matrix.GetLength(1)];
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = 0; j < matrix.GetLength(1); j++)
            {
                if (graph && i == j) currentValues[i, j] = "/";
                else if (matrix[i, j] == 0) currentValues[i, j] =
                     "_";
                else currentValues[i, j] = matrix[i, j].ToString
                   ();
            }
        }
        return currentValues;
    }
    private void ParseForReturn(string[,] currentValues)
    {
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = 0; j < matrix.GetLength(1); j++)
            {
                if (double.TryParse(currentValues[i, j], out
                   double x)) matrix[i, j] = x;
                else matrix[i, j] = 0;
            }
        }
    }
    public double[,] Select()
    {
        int maxWidth = (Console.BufferWidth - (graph ? 2 : 1)) /
           columns.Length - 1;
        string[,] currentValues = SetUpCurrentValues();
        int top = Console.CursorTop;
        int col = graph? 1 : 0;
        int row = 0;
        int align = 0;

        top = WriteOut(top, col, row, currentValues, align, true)
           ;

        bool exit = false;
        do
        {
            WriteOut(top, col, row, currentValues, align);
            ConsoleKeyInfo key = Console.ReadKey(true);

            if (row != matrix.GetLength(0))
            {
                bool isTooBig = currentValues[row, col].Length == maxWidth;
                if (!isTooBig && (int.TryParse(key.KeyChar.ToString(), out int num) || (key.KeyChar == '.' && !currentValues[row, col].Contains('.'))))
                {
                    if (currentValues[row, col] == "_") currentValues[row, col] = key.KeyChar.ToString();
                    else currentValues[row, col] = currentValues[row, col].Insert(currentValues[row, col].Length - align, key.KeyChar.ToString());

                    if (symmetry) currentValues[col, row] = currentValues[row, col];
                }
                else if (!isTooBig && key.KeyChar == '-' && negative && !currentValues[row, col].Contains('-'))
                {
                    if (currentValues[row, col] == "_") currentValues[row, col] = "-";
                    else currentValues[row, col] = $"-{currentValues[row, col]}";
                }
                else if (key.Key == ConsoleKey.Backspace && currentValues[row, col].Length > 0)
                {
                    currentValues[row, col] = currentValues[row, col].Remove(currentValues[row, col].Length - 1 - align, 1);
                    if (symmetry) currentValues[col, row] = currentValues[row, col];
                }
                else if (key.Key == ConsoleKey.RightArrow || key.Key == ConsoleKey.Tab)
                {
                    if (graph && col + 1 == row && col + 2 < columns.Length) col++;
                    if (align == 0 && col < columns.Length - 1 && !(graph && col + 1 == row))
                    {
                        col++;
                        align = 0;
                    }
                    else if (align > 0) align--;
                }
                else if (key.Key == ConsoleKey.LeftArrow)
                {
                    if (align == currentValues[row, col].Length && col > 0)
                    {
                        if (graph && col - 1 == row && col - 2 >= 0) col--;
                        if (col > 0 && !(graph && col - 1 == row))
                        {
                            align = 0;
                            col--;
                        }
                    }
                    else if (align < currentValues[row, col].Length) align++;
                }
                else if (key.Key == ConsoleKey.DownArrow || key.Key == ConsoleKey.Enter)
                {
                    if (graph && col == row + 1 && row + 2 <= matrix.GetLength(0)) row++;
                    if (row < matrix.GetLength(0) - 1 && !(graph && col == row + 1))
                    {
                        row++;
                        align = Math.Min(currentValues[row, col].Length, align);
                    }
                    else if (row == matrix.GetLength(0) - 1)
                    {
                        col = 0;
                        align = 0;
                        row++;
                    }
                }
                else if (key.Key == ConsoleKey.Escape)
                {
                    exit = true;
                }
            }
            else if (key.Key == ConsoleKey.Enter)
            {
                exit = true;
            }
            if (key.Key == ConsoleKey.UpArrow && row - 1 >= 0)
            {
                if (graph && col == row - 1 && row - 2 >= 0) row--;
                if (row - 1 >= 0 && (!graph || (graph && col != row - 1)))
                {
                    row--;
                    align = Math.Min(currentValues[row, col].Length, align);
                }
            }
            ConsoleHelper.ClearLines(top, top + matrix.GetLength(0) + 3);
        } while (!exit);
        Console.SetCursorPosition(0, top);
        ParseForReturn(currentValues);
        return matrix;
    }
}
```

Graph

```
public class Graph
{
    // A 0 represents no connection
    private double[,] matrix;
    private char[] nodeNames;

    public Graph(int numberOfNodes)
    {
        CreateMatrix(numberOfNodes);
    }
    private void CreateMatrix(int numberOfNodes)
    {
        matrix = new double[numberOfNodes, numberOfNodes];
        nodeNames = new char[numberOfNodes];
        for (int i = 0; i < numberOfNodes; i++)
        {
            nodeNames[i] = (char)(i + 65);
        }
    }
    private void GenerateConnectedGraph(Random rnd, int lowerBound, int upperBound)
    {
        // Random Walk, uniform spanning tree
        HashSet<char> nodes = new HashSet<char>(nodeNames);
        HashSet<char> visited = new HashSet<char>();

        char currentNode = nodeNames[rnd.Next(nodeNames.Length)];
        nodes.Remove(currentNode);
        visited.Add(currentNode);

        while (nodes.Count > 0)
        {
            char neighbourNode = nodeNames[rnd.Next(nodeNames.Length)];
            if (!visited.Contains(neighbourNode) && currentNode != neighbourNode)
            {
                matrix[currentNode - ‘A’, neighbourNode - ‘A’] = rnd.Next(lowerBound, upperBound);
                matrix[neighbourNode - ‘A’, currentNode - ‘A’] = matrix[currentNode - ‘A’, neighbourNode - ‘A’];

                nodes.Remove(neighbourNode);
                visited.Add(neighbourNode);
            }
            currentNode = neighbourNode;
        }
    }
    private void AddRandomEdges(int numOfEdges, Random rnd, int lowerBound, int upperBound, bool symmetry)
    {
        HashSet<(int, int)> available = new HashSet<(int, int)>();
        // If symmetrical (undirected), look for locations where both sides are empty
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = symmetry? i + 1 : 0; j < matrix.
                GetLength(0); j++)
            {
                if (matrix[i, j] == 0 && i != j) available.Add((i
                    , j));
            }
        }
        for (int i = 0; i < numOfEdges; i++)
        {
            (int row, int col) = available.ToArray()[rnd.Next(
                available.Count)];
            available.Remove((row, col));
            matrix[row, col] = rnd.Next(lowerBound, upperBound);
            if (symmetry) matrix[col, row] = matrix[row, col];
        }
    }
    public void GenerateGraph(Random rnd, bool symmetry = true)
    {
        int numberOfExtraEdges = rnd.Next(nodeNames.Length - 1,
            nodeNames.Length * (nodeNames.Length - 1) / 2 + 1) -
            (nodeNames.Length - 1);

        // Randomly generates a range that the lengths can be
        int lowerBound = rnd.Next(1, 20);
        int upperBound = lowerBound + 4 * (int)Math.Round(Math.
            Pow(lowerBound, 0.5));

        // First thing is to create a connected graph.
        GenerateConnectedGraph(rnd, lowerBound, upperBound);

        // Then add random extra edges
        AddRandomEdges(numberOfExtraEdges, rnd, lowerBound,
            upperBound, symmetry);
    }

    public void InputGraph(bool symmetry = true, bool dijkstra =
        false)
    {
        Console.WriteLine("Enter adjacency matrix (anything not a
             number is automatically 0, LHS is To and top row is
            From)");
        matrix = new MatrixEnter(matrix, nodeNames.Select(x => x.
            ToString()).ToArray(), true, symmetry, !dijkstra).
            Select();
        ConsoleHelper.ClearLine(Console.CursorTop - 1);
    }

    public double[,] GetAdjacencyMatrix() => matrix;
    public char[] GetNodeNames() => nodeNames;

    public Dictionary<char, double> GetConnections(char node)
    {
        if (node - 65 >= nodeNames.Length) return null;
        Dictionary<char, double> connections = new Dictionary<
            char, double>();
        for (int i = 0; i < nodeNames.Length; i++)
        {
            if (matrix[node - 65, i] != 0) connections.Add((char)
                (i + 65), matrix[node - 65, i]);
        }
        return connections;
    }

    public override string ToString()
    {
        if (matrix is null) return "";
        List<string> lines = new List<string>();
        int[] largestSize = new int[nodeNames.Length];
        for (int i = 0; i < largestSize.Length; i++)
        {
            for (int j = 0; j < matrix.GetLength(0); j++)
            {
                largestSize[i] = Math.Max(largestSize[i], matrix[
                    j, i].ToString().Length);
            }
        }
        lines.Add(" |");
        foreach (char node in nodeNames) lines.Add($"{node}|");
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            lines[0] += $"{nodeNames[i]}".PadLeft(largestSize[i])
                +"|";
            for (int j = 0; j < matrix.GetLength(0); j++)
            {
                lines[j + 1] += $"{(matrix[j, i] == 0? "-":
                    matrix[j, i].ToString())}".PadLeft(
                    largestSize[i])+"|";
            }
        }
        return String.Join("\n", lines.ToArray());
    }
}
```

5.2.4 Simplex

Equation

```
public class EquationEnter
{
    // works in a similar fashion to MatrixEnter but for
        equations instead
    private string[] columns;
    private double[] values;
    private bool negative;

    public EquationEnter(string[] columnSeparators, bool negative
         = true) => (columns, values, this.negative) = (
        columnSeparators, new double[columnSeparators.Length],
        negative);
    public EquationEnter(string[] columnSeparators, double[]
        columnValues, bool negative = true) => (columns, values,
        this.negative) = (columnSeparators, columnValues,
        negative);

    private void WriteOut(int left, int height, int currentOption
        , string[] currentValues, int align)
    {
        ConsoleHelper.ClearLines(height + 1, height + 1 + (
            columns.Sum(x => x.Length) + currentValues.Sum(x => x
            .Length)) / Console.BufferWidth);
        ConsoleHelper.ClearLine(height, left);
        Console.Write(String.Join(" ", Enumerable.Range(0, currentOption).Select(x => $"{currentValues[x]}{columns[x]}")));
        int startLeft = Console.CursorLeft + currentValues[currentOption].Length - align + 1;
        int startHeight = Console.CursorTop + startLeft / Console.BufferWidth;
        Console.Write($" {currentValues[currentOption]} {columns[currentOption]} ");
        Console.Write(String.Join("", Enumerable.Range(currentOption + 1, values.Length - currentOption - 1).Select(x => $"{currentValues[x]}{columns[x]}")));
        Console.SetCursorPosition(startLeft % Console.BufferWidth, startHeight);
    }
    public double[] Select()
    {
        string[] currentValues = values.Select(x => x == 0 ? "_" : x.ToString()).ToArray();
        int top = Console.CursorTop;
        int left = Console.CursorLeft;
        int currentOption = 0;
        int buffer = 0;

        bool exit = false;
        do
        {
            WriteOut(left, top, currentOption, currentValues, buffer);
            ConsoleKeyInfo key = Console.ReadKey(true);

            if (int.TryParse(key.KeyChar.ToString(), out int num) || (key.KeyChar == ‘.’ && !currentValues[currentOption].Contains(‘.’)))
            {
                if (currentValues[currentOption] == "_") currentValues[currentOption] = key.KeyChar.ToString();
                else currentValues[currentOption] = currentValues[currentOption].Insert(currentValues[currentOption].Length - buffer, key.KeyChar.ToString());
            }
            else if (key.KeyChar == ‘-’ && negative && !currentValues[currentOption].Contains(‘-’))
            {
                if (currentValues[currentOption] == "_") currentValues[currentOption] = "-";
                else currentValues[currentOption] = $"-{currentValues[currentOption]}";
            }
            else if (key.Key == ConsoleKey.Backspace && currentValues[currentOption].Length > 0 && currentValues[currentOption].Length - 1 - buffer >= 0)
            {
                currentValues[currentOption] = currentValues[currentOption].Remove(currentValues[currentOption].Length - 1 - buffer, 1);
            }
            else if (key.Key == ConsoleKey.RightArrow)
            {
                if (buffer == 0 && currentOption < values.Length - 1) currentOption++;
                else if (buffer > 0) buffer--;
            }
            else if (key.Key == ConsoleKey.LeftArrow)
            {
                if (buffer == currentValues[currentOption].Length && currentOption > 0)
                {
                    buffer = 0;
                    currentOption--;
                }
                else if (buffer < currentValues[currentOption].Length) buffer++;
            }
            else if ((key.Key == ConsoleKey.Tab || key.Key == ConsoleKey.Enter) && currentOption < values.Length - 1)
            {
                buffer = 0;
                currentOption++;
            }
            else if (key.Key == ConsoleKey.Escape || (key.Key == ConsoleKey.Enter && currentOption == values.Length - 1))
            {
                ConsoleHelper.ClearLines(top, top + (columns.Sum(x => x.Length) + currentValues.Sum(x => x.Length)) / Console.BufferWidth + 1);
                exit = true;
            }
        } while (!exit);
        return currentValues.Select(x => double.TryParse(x, out double y) ? y : 0).ToArray();
    }
}
public class Equation
{
    private string[] LHSvars, RHSvars;
    private double[] LHS, RHS;
    public string symbol;
    public Equation(string[] LHSvariables, double[] LHScoefficient, string equality, string[] RHSvariables, double[] RHScoefficient) =>
        (symbol, LHSvars, LHS, RHSvars, RHS) = (equality, LHSvariables, LHScoefficient, RHSvariables, RHScoefficient);
    public Equation(string[] LHSvariables, double[] LHScoefficient, string equality, double RHSvalue) =>
        (symbol, LHSvars, LHS, RHSvars, RHS) = (equality, LHSvariables, LHScoefficient, new string[1], new double[] { RHSvalue });

    private string JoinCoefficients(string[] vars, double[] values)
    {
        string currentSide = "";
        for (int i = 0; i < vars.Length; i++)
        {
            string toAdd;
            if (values[i] == 0 && !(vars[i] is null)) continue;

            if (values[i] == 1 && !(vars[i] is null)) toAdd = $"+
                {vars[i]} ";
            else if (values[i] == -1) toAdd = $"- {vars[i]} ";
            else if (values[i] < 0) toAdd = $"- {Math.Abs(values[
               i])}{vars[i]} ";
            else toAdd = $"+ {values[i]}{vars[i]} ";

            currentSide += currentSide == "" ? toAdd.TrimStart
               ('+') : toAdd;
        }
        return currentSide.Trim();
    }
    public string[] GetLHSvariables() => LHSvars;
    public double[] GetLHSValues() => LHS;
    public double[] GetRHSValues() => RHS;
    public string[] GetRHSvariables() => RHSvars;

    public override string ToString() => $"{JoinCoefficients(
       LHSvars, LHS)} {symbol} {JoinCoefficients(RHSvars, RHS)}"
       ;
}
```

Simple exception class for unbounded regions:

```
public class RegionNotBoundedException : Exception
{
    public RegionNotBoundedException() { }
    public RegionNotBoundedException(string message) : base(
       message) { }
}
```

Simplex creator class:

```
public class SimplexCreator : IProblem
{
    private SimplexSolver problem;

    private void GenerateLEQ(Random rnd, int dimension, int
       noConstraints, int LB, int UB, List<Equation> constraints
       , string[] variableNames)
    {
        // 2-3 <= constraints
        double[] minAxisIntercepts = new double[dimension].Select
           (x => double.PositiveInfinity).ToArray();

        for (int i = 0; i < noConstraints; i++)
        {
            double[] tempVariableValues = new double[dimension];
            int c = rnd.Next(LB, UB * 2 + 1);
            int LBCoefficient = Math.Max(1, c / UB);
            int UBCoefficient = c / LB;
            int guaranteed = rnd.Next(dimension);

            for (int j = 0; j < dimension; j++)
            {
                // One in (Dimension) chance that a coefficient is 0.
                // If it is a 0 but that dimension is not yet bounded on the final pass, it is skipped.
                if (rnd.Next(0, dimension) == 0 && !(double.IsPositiveInfinity(minAxisIntercepts[j]) && i == dimension - 1) && j != guaranteed) tempVariableValues[j] = 0;
                else
                {
                    tempVariableValues[j] = rnd.Next(LBCoefficient, UBCoefficient + 1);
                    minAxisIntercepts[j] = Math.Min(minAxisIntercepts[j], c / tempVariableValues[j]);
                }
            }
            constraints.Add(new Equation(variableNames, tempVariableValues, "<=", c));
        }
    }
    private bool GenerateGEQ(Random rnd, int dimension, int difficulty, int LB, List<Equation> constraints, string[] variableNames)
    {
        // 0-2 >= constraints. 0 if difficulty is 1, up to 1 if difficulty is 2
        // There should always be a feasible region as long as one of the axis intercepts in the equation is less than the smallest axis intercept of any of the <= equations
        int noConstraints;
        bool artificial = false;

        if (difficulty == 1) noConstraints = 0;
        else if (difficulty == 2) noConstraints = rnd.Next(0, 2);
        else noConstraints = rnd.Next(0, 3);

        for (int i = 0; i < noConstraints; i++)
        {
            artificial = true;
            int c = rnd.Next(1, LB * 2 + 1);
            double[] tempVariableValues = new double[dimension];
            int guaranteed = rnd.Next(dimension);

            for (int j = 0; j < dimension; j++)
            {
                if (j == guaranteed) tempVariableValues[j] = rnd.Next(1, c / LB + 1);
                else if (rnd.Next(0, dimension) == 0) tempVariableValues[j] = 0;
                else tempVariableValues[j] = rnd.Next(1, LB);
            }
            constraints.Add(new Equation(variableNames, tempVariableValues, ">=", c));
        }
        return artificial;
    }
    private void GenerateEQ(Random rnd, int dimension, int LB, List<Equation> constraints, string[] variableNames)
    {
        int c = rnd.Next(1, LB * 2 + 1);
        double[] tempVariableValues = new double[dimension];
        int guaranteed = rnd.Next(dimension);

        for (int j = 0; j < dimension; j++)
        {
            if (j == guaranteed) tempVariableValues[j] = rnd.Next
                (1, c / LB + 1);
            else if (rnd.Next(0, dimension) == 0)
                tempVariableValues[j] = 0;
            else tempVariableValues[j] = rnd.Next(1, LB);
        }
        constraints.Add(new Equation(variableNames,
            tempVariableValues, "=", c));
    }
    public void GenerateQuestion(Random rnd)
    {
        Equation objective;
        List<Equation> constraints = new List<Equation>();

        int difficulty = new OptionScroller("Choose Difficulty: "
            , 1, 5).Select();

        // 2-5 dimensions
        int dimension = rnd.Next(0, 2) + difficulty;
        if (difficulty == 1) dimension = 2;

        string[] variableNames = new string[dimension];
        double[] tempVariableValues = new double[dimension];
        for (int i = 0; i < dimension; i++)
        {
            variableNames[i] = dimension == 2 ? i == 0 ? "x" : "y
                " : $"x{myMath.GetSubscript(i + 1)}";
            tempVariableValues[i] = rnd.Next(1, 11) * (rnd.Next
                (0, 5) == 0 ? -1 : 1);
        }

        // Create objective function
        objective = new Equation(new string[] { "P" }, new double
            [] { 1 }, "=", variableNames, tempVariableValues);

        //Create constraints
        int noConstraints = rnd.Next(2, 4);
        int lowerBound = rnd.Next(8, 80);
        int upperBound = lowerBound + 4 * (int)Math.Round(Math.
            Pow(lowerBound, 0.5));

        GenerateLEQ(rnd, dimension, rnd.Next(2, 4), lowerBound,
            upperBound, constraints, variableNames);
        bool GEQConstraints = GenerateGEQ(rnd, dimension,
            difficulty, lowerBound, constraints, variableNames);
        // If no >= constraint, chance of there being a =
            constraint as long as the difficulty is greater than
            2
        if ((!GEQConstraints && rnd.Next(0, 2) == 0 && difficulty
             > 3) || (!GEQConstraints && difficulty == 5))
            GenerateEQ(rnd, dimension, lowerBound, constraints,
            variableNames);

        // Display the created problem
        Console.WriteLine($"Maximise: {objective}\nSubject to:");
        foreach (var constraint in constraints)
        {
            Console.WriteLine(constraint);
        }
        Console.WriteLine(String.Join(", ", variableNames.Select(x => $"{x} >= 0")));
        Console.WriteLine();
        problem = new SimplexSolver(objective, constraints, dimension, BasicVarsMenu(), IterationStepMenu());
    }

    private bool BasicVarsMenu()
        => new Menu(new List<string>() { "Display basic variables at each iteration", "Don't display basic variables at each iteration" }).SelectOption() == 0;
    private bool IterationStepMenu()
        => new Menu(new List<string>() { "Show how to calculate each row", "Don't show how to calculate each row" }).SelectOption() == 0;

    private Equation InputObjective(int dimension, string[] variableNames, string[] variableConcat)
    {
        Console.Write("Enter objective function: \nP = ");
        Equation objective = new Equation(new string[] { "P" }, new double[] { 1 }, "=", variableNames, new EquationEnter(variableConcat).Select());
        ConsoleHelper.ClearLine(Console.CursorTop - 1);
        Console.WriteLine($"Objective: {objective}");
        return objective;
    }
    private void InputEquations(Equation[] constraints, int leqConstraints, int geqConstraints, string[] variableNames, string[] variableConcat)
    {
        while (true)
        {
            int option = new Menu(Enumerable.Range(1, constraints.Length).Select(x => $"Equation {x} ({(x <= leqConstraints ? "<=" : x - leqConstraints <= geqConstraints ? ">=" : "=")}): {(!(constraints[x - 1] is null) ? constraints[x - 1].ToString() : "")}").Concat(new string[] { "Finish!" }).ToList()).SelectOption();
            if (option == constraints.Length) break;

            string symbol = option < leqConstraints ? "<=" : option - leqConstraints < geqConstraints ? ">=" : "=";
            double[] values;
            if (constraints[option] is null)
            {
                values = new EquationEnter(Enumerable.Range(0, variableConcat.Length)
                    .Select(x => x == variableConcat.Length - 1 ? $"{variableConcat[x]} {symbol}" : variableConcat[x])
                    .Concat(new string[] { "" }).ToArray()).
```

```
                        Select();
            }
            else
            {
                values = new EquationEnter(Enumerable.Range(0,
                    variableConcat.Length)
                    .Select(x => x == variableConcat.Length - 1 ?
                        $"{variableConcat[x]} {symbol}" :
                        variableConcat[x])
                    .Concat(new string[] { "" }).ToArray(),
                        constraints[option].GetLHSValues().Concat
                        (constraints[option].GetRHSValues()).
                        ToArray()).Select();

            }
            constraints[option] = new Equation(variableNames,
                values.Take(values.Length - 1).ToArray(), symbol,
                 values[values.Length - 1]);
        }
    }
    private bool CheckBounded(int dimension, Equation[]
        constraints)
    {
        // Validate input, check if all dimensions are restricted
        bool[] validate = new bool[dimension].Select(x => false).
            ToArray();
        foreach (Equation constraint in constraints)
        {
            if (constraint is null) return false;
            if (constraint.symbol == ">=") continue;
            double[] LHS = constraint.GetLHSValues();
            for (int i = 0; i < dimension; i++)
            {
                if (LHS[i] != 0)
                {
                    validate[i] = true;
                }
            }
        }
        // Returns true if the region is bounded
        return !validate.Contains(false);
    }
    private void InputConstraints(int dimension)
    {
        Equation objective;
        Equation[] constraints = new Equation[new OptionScroller(
            "Select number of constraints: ", 1).Select()];
        Console.WriteLine($"Number of constraints: {constraints.
            Length}");
        int leqConstraints = new OptionScroller("Number of '<='
            constraints: ", 0, constraints.Length).Select();
        Console.WriteLine($"Number of '<=' constraints: {
            leqConstraints}");
        int geqConstraints = 0;
        if (constraints.Length - leqConstraints > 0)
            geqConstraints = new OptionScroller("Number of '>='
            constraints: ", 0, constraints.Length -
            leqConstraints).Select();
        Console.WriteLine($"Number of '>=' constraints: {
            geqConstraints}");
        Console.WriteLine($"Number of ‘=’ constraints: {
            constraints.Length - leqConstraints - geqConstraints}
            ");

        // Create the variable names array and the same array but
            connected with + to enter in the equations
        string[] variableNames = new string[dimension];
        for (int i = 0; i < dimension; i++)
        {
            variableNames[i] = dimension == 2 ? i == 0 ? "x" : "y
                " : $"x{myMath.GetSubscript(i + 1)}";
        }
        string[] variableConcat = String.Join("+ ,",
            variableNames).Split(‘,’).ToArray();

        objective = InputObjective(dimension, variableNames,
            variableConcat);
        InputEquations(constraints, leqConstraints,
            geqConstraints, variableNames, variableConcat);

        // If dimension not restricted, throw error
        try
        {
            if (!CheckBounded(dimension, constraints)) throw new
                RegionNotBoundedException();

            // Display problem and stop
            Console.WriteLine("Subject to:");
            foreach (var constraint in constraints)
            {
                Console.WriteLine(constraint);
            }
            Console.WriteLine();
            problem = new SimplexSolver(objective, constraints.
                ToList(), dimension, BasicVarsMenu(),
                IterationStepMenu());
        }
        catch (RegionNotBoundedException)
        {
            Console.WriteLine("Region not bounded or not all
                constraints filled!");
            Console.ReadKey();
            Console.Clear();
        }
    }
    private bool CheckBasic(double[,] matrix, int col)
    {
        int ones = 0;
        int zeroes = 0;
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            if (matrix[i, col] == 0) zeroes++;
            else if (matrix[i, col] == 1) ones++;
        }
        return ones == 1 && ones + zeroes == matrix.GetLength(0);
    }
    private string[] SetUpTableauVariables(int dimension, int
        noOfS, int noOfA)
    {
        if (noOfA > 0) return new string[] { "A", "P" }
        .Concat(Enumerable.Range(1, dimension).Select(x => $"x{myMath.GetSubscript(x)}").ToArray())
        .Concat(Enumerable.Range(1, noOfS).Select(x => $"s{myMath.GetSubscript(x)}").ToArray())
        .Concat(Enumerable.Range(1, noOfA).Select(x => $"a{myMath.GetSubscript(x)}").ToArray())
        .Concat(new string[] { "RHS" }).ToArray();

        return new string[] { "P" }
        .Concat(Enumerable.Range(1, dimension).Select(x => $"x{myMath.GetSubscript(x)}").ToArray())
        .Concat(Enumerable.Range(1, noOfS).Select(x => $"s{myMath.GetSubscript(x)}").ToArray())
        .Concat(new string[] { "RHS" }).ToArray();
    }
    private void InputTableau(int dimension)
    {
        // Input number of constraints etc.
        int noOfConstraints = new OptionScroller("Select number of constraints: ", 1).Select();
        Console.WriteLine($"Number of constraints: {noOfConstraints}");
        int noOfS = new OptionScroller("Select number of slack/surplus variables (\"s\"): ", 1).Select();
        Console.WriteLine($"Number of slack/surplus: {noOfS}");
        int noOfA = new OptionScroller("Select number of artificial variables (\"a\"): ", 0).Select();
        Console.WriteLine($"Number of artificial: {noOfA}");

        string[] variables = SetUpTableauVariables(dimension, noOfS, noOfA);

        int artificialOffset = noOfA > 0 ? 1 : 0;
        double[,] values = new double[noOfConstraints + 1 + artificialOffset, variables.Length];
        if (noOfA > 0) values[0, 0] = 1;
        values[0 + artificialOffset, 0 + artificialOffset] = 1;

        values = new MatrixEnter(values, variables).Select();

        // There is no input validation for a tableau apart from checking if A and P are basic
        try
        {
            if (CheckBasic(values, 0) && (noOfA == 0 || ((noOfA > 0) && CheckBasic(values, 1))))
            {
                problem = new SimplexSolver(values, variables, noOfA > 0, BasicVarsMenu(), IterationStepMenu());
            }
            else throw new FormatException("");
        }
        catch (FormatException)
        {
            Console.WriteLine("Input in incorrect format :(");
            Console.ReadKey();
            Console.Clear();
        }
    }
    public void InputQuestion()
    {
        int dimension = new OptionScroller("Select dimension: ",
           2).Select();
        Console.WriteLine($"Dimension: {dimension}");

        Console.WriteLine("Select format:");
        if (new Menu(new List<string>() { "Constraints", "Tableau
           "}).SelectOption() == 0)  // Constraints
        {
            ConsoleHelper.ClearLine(Console.CursorTop - 1);
            InputConstraints(dimension);
        }
        else  // Tableau
        {
            ConsoleHelper.ClearLine(Console.CursorTop - 1);
            InputTableau(dimension);
        }
    }
    public void GenerateAnswer()
    {
        if (!(problem is null)) problem.Solve();
        else Console.WriteLine("There is no problem to solve!");
    }
}
```

Simplex solver class:

```
public class SimplexSolver
{
    // Unformatted equations
    private Equation objective;
    private List<Equation> constraints;

    private string[] variables;
    private double[,] values;
    private bool stage2, displayBasic, displayRowOperation,
        skipReformulation;

    // variables for 2D graphing
    private Simplex2DGraph plot;
    private int dimension;

    // constraints
    public SimplexSolver(Equation objective, List<Equation>
        constraints, int dimension, bool displayBasicVars, bool
        displayRowCalc)
         => (this.objective, this.constraints, this.dimension,
           displayBasic, displayRowOperation, skipReformulation)
            = (objective, constraints, dimension,
           displayBasicVars, displayRowCalc, false);

    // tableaux
    public SimplexSolver(double[,] values, string[] variables,
        bool stage2, bool displayBasicVars, bool displayRowCalc)
         => (this.values, this.variables, this.stage2,
           displayBasic, displayRowOperation, dimension,
           skipReformulation) = (values, variables, stage2,
           displayBasicVars, displayRowCalc, 0, true);


    private int[] FindLargestWidth(int pivotCol)
    {
        //The following for loop finds out how much space each
           variable should take up when being displayed on the
           console
        int[] largestSizes = new int[variables.Length];
        for (int i = 0; i < variables.Length; i++)
        {
            int largestSize = variables[i].Length;
            for (int j = 0; j < values.GetLength(0); j++)
            {
                //increase the corresponding largest size if the
                   variable in the column is wider
                if (myMath.SigFig(values[j, i]).ToString().Length
                    > largestSize) largestSize = myMath.SigFig(
                   values[j, i]).ToString().Length;
            }
            largestSize++;
            largestSizes[i] = largestSize;

            if (i == pivotCol) ConsoleHelper.ColourText(
                ConsoleColor.Yellow, ConsoleColor.Gray, $"{
                variables[i]}".PadRight(largestSize) + "|");
            else Console.Write($"{variables[i]}".PadRight(
                largestSize) + "|");
        }
        return largestSizes;
    }
    private void DisplayRatio(double[] ratios, int pivotCol, int
        i)
    {
        if (ratios.Length == 0) Console.WriteLine();
        else if ((stage2 && i >= 2 && values[i, pivotCol] == 0)
           || (!stage2 && i >= 1 && values[i, pivotCol] == 0))
           Console.WriteLine("undefined");
        else if (stage2 && i >= 2) Console.WriteLine(myMath.
           SigFig(ratios[i - 2]));
        else if (!stage2 && i >= 1) Console.WriteLine(myMath.
           SigFig(ratios[i - 1]));
        else Console.WriteLine();
    }
    private void DisplayRow(int pivotRow, int pivotCol, int[]
        largestSizes, int i)
    {
        //checks if it is a pivot, then changes colour
           accordingly
        if (i == pivotRow) Console.ForegroundColor = ConsoleColor
           .Magenta;
        for (int j = 0; j < values.GetLength(1); j++)
        {
            string text = myMath.SigFig(values[i, j]).ToString().
                PadRight(largestSizes[j]) + "|";
            if (i == pivotRow && j == pivotCol) ConsoleHelper.
                ColourText(ConsoleColor.Red, ConsoleColor.Magenta
                , text);
            else if (j == pivotCol) ConsoleHelper.ColourText(
                ConsoleColor.Yellow, ConsoleColor.Gray, text);
            else Console.Write(text);
        }
        Console.ForegroundColor = ConsoleColor.Gray;
    }
    private void DisplayTableau(string message)
    {
        //The pivot column and row are off the grid so they can't
            be highlighted
        //There are no ratio tests to be shown
        DisplayTableau(-1, -1, message, Array.Empty<double>());
    }
    private void DisplayTableau(int pivotRow, int pivotCol,
        string iterationMessage, double[] ratios)
    {
        // Display the current iteration of the tableau. The
            final tableau is set to have iteration = -1.
        Console.WriteLine(iterationMessage);

        int[] largestSizes = FindLargestWidth(pivotCol);
        Console.WriteLine(ratios.Length > 0 ? "Ratio Test" : "");

        for (int i = 0; i < values.GetLength(0); i++)
        {
            DisplayRow(pivotRow, pivotCol, largestSizes, i);
            DisplayRatio(ratios, pivotCol, i);
        }
        Console.WriteLine();
    }
    private int ChooseLargestColumn()
    {
        double largest = values[0, 2];
        int column = 2;
        for (int i = 1; i < values.GetLength(1) - 1; i++)
        {
            if (values[0, i] > largest)
            {
                largest = values[0, i];
                column = i;
            }
        }
        return column;
    }
    private int ChooseSmallestColumn()
    {
        double smallest = values[0, 1];
        int column = 1;
        for (int i = 1; i < values.GetLength(1) - 1; i++)
        {
            if (values[0, i] < smallest)
            {
                smallest = values[0, i];
                column = i;
            }
        }
        return column;
    }
    private double[] RatioTest(int pivotColumn)
    {
        int buffer = stage2 ? 2 : 1;
        double[] tests = new double[values.GetLength(0) - buffer
            ];
        for (int i = buffer; i < values.GetLength(0); i++)
        {
            if (values[i, pivotColumn] != 0) tests[i - buffer] = values[i, values.GetLength(1) - 1] / values[i, pivotColumn];
            else
            {
                // -1 is a temporary value. In the DisplayTableau() procedure it will display as "undefined"
                tests[i - buffer] = -1;
            }
        }
        return tests;
    }
    private int FindPivotRow(double[] tests)
    {
        double min = tests.Max();
        try
        {
            // This can only happen if it is a user input through a tableau
            if (min < 0) throw new RegionNotBoundedException();
        }
        catch (RegionNotBoundedException)
        {
            ConsoleHelper.ColourText(ConsoleColor.Red, ConsoleColor.Gray, "Region is not bounded!!\n");
            return -1;
        }
        int row = 0;
        for (int i = 0; i < tests.Length; i++)
        {
            if (tests[i] >= 0 && tests[i] <= min)
            {
                min = tests[i];
                row = i + 1;
            }
        }
        return stage2 ? row + 1 : row;
    }
    private double[,] CreateNextTableau(int pivotRow, int pivotCol)
    {
        if (displayRowOperation) Console.WriteLine("To calculate the next table:");
        // Perform the iteration using the pivot row and column
        double[,] newTableau = new double[values.GetLength(0), values.GetLength(1)];
        double pivot = values[pivotRow, pivotCol];

        for (int i = 0; i < values.GetLength(1); i++)
        {
            newTableau[pivotRow, i] = values[pivotRow, i] / pivot;
        }

        double multiplier;
        for (int i = 0; i < values.GetLength(0); i++)
        {
            if (i == pivotRow)
            {
                if (displayRowOperation) Console.WriteLine($"
                   new_row_{i + 1} = pivot_row / {myMath.SigFig(
                   pivot, 5)}");
                continue;
            }
            multiplier = values[i, pivotCol] / pivot;
            if (displayRowOperation) Console.WriteLine($"new_row_
               {i + 1} = previous_row - pivot_row * {myMath.
               SigFig(multiplier, 3)}");
            for (int j = 0; j < values.GetLength(1); j++)
            {
                newTableau[i, j] = values[i, j] - multiplier *
                   values[pivotRow, j];
                newTableau[i, j] = myMath.WithinPrecision(Math.
                   Round(newTableau[i, j]), newTableau[i, j])?
                   Math.Round(newTableau[i, j]) : newTableau[i,
                   j];
            }
        }

        if (displayRowOperation) Console.WriteLine();
        return newTableau;
    }

    private void FindBasicVars(bool display, string iteration,
       bool reductionState = false)
    {
        int[] rowOfBasic = new int[variables.Length - 1];

        // 2 stage coordinate plotting
        (double x, double y) coordinates = (0, 0);
        bool plotted = false;

        // identify if each variable is basic and the row it
           corresponds to
        for (int i = 0; i < variables.Length - 1; i++)
        {
            bool basic = true;
            int zeroes = 0;
            int ones = 0;
            int oneLoc = 0;
            for (int j = 0; j < values.GetLength(0); j++)
            {
                if (values[j, i] == 0) zeroes++;
                else if (values[j, i] == 1)
                {
                    ones++;
                    oneLoc = j;
                }
                else
                {
                    basic = false;
                    break;
                }
            }
            if (basic && ones == 1) rowOfBasic[i] = oneLoc;
            else rowOfBasic[i] = 0;
        }

        List<string> nonBasic = new List<string>();
        if (display) Console.WriteLine("Basic variables: ");
        for (int i = stage2 ? 2 : 1; i < variables.Length - 1; i
            ++)
        {
            if (rowOfBasic[i] == -1) continue;
            if (rowOfBasic[i] == 0)
            {
                nonBasic.Add(variables[i]);
            }
            else
            {
                // gather all other variables on the same row.
                int row = rowOfBasic[i];
                int[] occurences = rowOfBasic.Select((x, j) => x
                    == row ? j : -1).Where(j => j != -1).ToArray
                    ();
                List<string> namedOccurences = new List<string>()
                    ;

                for (int j = 0; j < occurences.Length; j++)
                {
                    namedOccurences.Add(variables[occurences[j]])
                        ;
                    rowOfBasic[occurences[j]] = -1;
                }
                if (display) Console.WriteLine(String.Join(" + ",
                     namedOccurences) + " = " + myMath.SigFig(
                    values[row, variables.Length - 1]));

                // Add to plot marker
                if (dimension == 2 && (namedOccurences[0][0] == '
                    x' || namedOccurences[0][0] == 'y'))
                {
                    // if they both lie on the same line
                    if (occurences.Length == 2 && namedOccurences
                        .Contains("x") && namedOccurences.
                        Contains("y"))
                    {
                        plot.AddLinePointToPlot(row, iteration,
                            values, variables);
                        plotted = true;
                    }
                    else if (namedOccurences[0][0] == 'x')
                    {
                        coordinates.x = values[row, variables.
                            Length - 1];
                    }
                    else if (namedOccurences[0][0] == 'y')
                    {
                        coordinates.y = values[row, variables.
                            Length - 1];
                    }
                }
            }
        }
        if (display)
        {
            Console.WriteLine("Non-basic variables: ");
            Console.WriteLine(String.Join(", ", nonBasic) + " =
                0\n");
        }

        // Plot the marker if not a line
        if (dimension == 2 && !plotted) plot.PlotMarker(iteration
            , coordinates.x, coordinates.y, values);
    }

    private bool DetermineStage2()
    {
        for (int i = 2; i < values.GetLength(1); i++)
        {
            if (values[0, i] > 0) return true;
        }
        return false;
    }

    private bool DetermineSolved(List<double[,]> seen)
    {
        SeenState(seen);

        for (int i = 0; i < values.GetLength(1) - 1; i++)
        {
            if (values[0, i] < 0) return false;
        }
        return true;
    }

    private void SeenState(List<double[,]> seen)
    {
        // Check for instances if the current array is in 'seen'.
             This means that the state has been seen before, and
            the problem is unsolvable.
        foreach (double[,] state in seen)
        {
            bool copy = true;
            for (int i = 0; i < values.GetLength(0); i++)
            {
                for (int j = 0; j < values.GetLength(1); j++)
                {
                    if (values[i, j] != state[i, j])
                    {
                        copy = false;
                        break;
                    }
                }
                if (!copy) break;
            }
            if (copy) throw new RegionNotBoundedException();
        }
    }

    private Equation ReformulateSlack(Equation constraint, int
        slackCount) =>
        new Equation(constraint.GetLHSvariables().Concat(new
            string[] { $"s{myMath.subscriptNums[slackCount]}" }).
            ToArray(),
             constraint.GetLHSValues().Concat(new double[] { 1 }).
                ToArray(), "=",
             constraint.GetRHSValues()[0]);

    private Equation ReformulateArtificial(Equation constraint,
        int slackCount, int artificialCount) =>
         new Equation(constraint.GetLHSvariables().Concat(new
            string[] { $"a{myMath.subscriptNums[artificialCount]}"
            , $"s{myMath.subscriptNums[slackCount]}" }).ToArray(),
          constraint.GetLHSValues().Concat(new double[] { 1, -1
              }).ToArray(), "=",
          constraint.GetRHSValues()[0]);

    private void ReformulateObjective(List<Equation> equations)
    {
        Equation newObjective = new Equation(objective.
            GetLHSvariables().Concat(objective.GetRHSvariables())
            .ToArray(),
          objective.GetLHSValues().Concat(objective.
              GetRHSValues().Select(x => -x)).ToArray(), "=",
              0);
        equations.Add(newObjective);
        Console.WriteLine($"The objective function becomes:\n{
            newObjective}");
    }
    private (int, int, List<int>) ReformulateConstraints(List<
        Equation> equations)
    {
        // Reformulate the constraints
        int slackSurplus = 0;
        int artificial = 0;
        List<int> rowsOfArtificial = new List<int>();

        Console.WriteLine("Subject to: ");
        foreach (Equation constraint in constraints)
        {
            Equation newConstraint;
            if (constraint.symbol == "=")
            {
                newConstraint = ReformulateSlack(constraint, ++
                    slackSurplus);
                equations.Add(newConstraint);
                Console.WriteLine(newConstraint);
                newConstraint = ReformulateArtificial(constraint,
                     ++slackSurplus, ++artificial);
                rowsOfArtificial.Add(equations.Count() + 1);
            }
            else if (constraint.symbol == "<=") newConstraint =
                ReformulateSlack(constraint, ++slackSurplus);
            else
            {
                newConstraint = ReformulateArtificial(constraint,
                     ++slackSurplus, ++artificial);
                rowsOfArtificial.Add(equations.Count() + 1);
            }

            equations.Add(newConstraint);
            Console.WriteLine(newConstraint);
        }
        return (slackSurplus, artificial, rowsOfArtificial);
    }
    private void SetUpVariables(int slackSurplus, int artificial)
    {
        if (artificial > 0) variables = new string[] { "A", "P"
            }.Concat(objective.GetRHSvariables())
```

```
                .Concat(Enumerable.Range(1, slackSurplus).Select(x => $"s{myMath.subscriptNums[x]}").ToArray())
                .Concat(Enumerable.Range(1, artificial).Select(x => $"a{myMath.subscriptNums[x]}").ToArray())
                .Concat(new string[] { "RHS" }).ToArray();
        else variables = new string[] { "P" }.Concat(objective.GetRHSvariables())
                .Concat(Enumerable.Range(1, slackSurplus).Select(x => $"s{myMath.subscriptNums[x]}").ToArray())
                .Concat(new string[] { "RHS" }).ToArray();
    }
    private void AddEquationsToTableau(int artificial, List<Equation> equations)
    {
        int artificialOffset = artificial > 0 ? 1 : 0;
        values = new double[equations.Count() + artificialOffset, variables.Length];

        for (int i = 0; i < equations.Count(); i++)
        {
            string[] currentNames = equations[i].GetLHSvariables();
            double[] currentValues = equations[i].GetLHSValues();

            // Add LHS
            for (int j = 0; j < variables.Length - 1; j++)
            {
                if (currentNames.Contains(variables[j])) values[i + artificialOffset, j] = currentValues[Array.IndexOf(currentNames, variables[j])];
                else values[i + artificialOffset, j] = 0;
            }
            // Add RHS
            values[i + artificialOffset, variables.Length - 1] = equations[i].GetRHSValues()[0];
        }
    }
    private void CreateSecondObjective(List<int> rowsOfArtificial)
    {
        values[0, 0] = 1;
        for (int i = 1; i < variables.Length; i++)
        {
            values[0, i] = variables[i][0] != 'a' ? rowsOfArtificial.Sum(row => values[row, i]) : 0;
        }
        Equation secondObjective = new Equation(
            variables.Take(variables.Length - 1).ToArray(),
            Enumerable.Range(0, variables.Length - 1).Select(x => values[0, x]).ToArray(), "=",
            values[0, variables.Length - 1]);
        Console.WriteLine($"As there are artificial variables (>= constraints) a second objective is needed (Σa):\n{secondObjective}");
    }
    private void CreateTableau(int slackSurplus, int artificial, List<int> rowsOfArtificial, List<Equation> equations)
    {
        SetUpVariables(slackSurplus, artificial);
        AddEquationsToTableau(artificial, equations);
        // Create the next objective function if applicable
        if (artificial > 0) CreateSecondObjective(rowsOfArtificial);
        stage2 = artificial > 0;
    }
    private void Reformulate()
    {
        List<Equation> equations = new List<Equation>();

        Console.WriteLine("The problem can then be put into augmented form as follows: ");

        // Reformulate the objective function
        ReformulateObjective(equations);
        (int slackSurplus, int artificial, List<int> rowsOfArtificial) = ReformulateConstraints(equations);

        // Create the initial tableau
        CreateTableau(slackSurplus, artificial, rowsOfArtificial, equations);
    }

    private int LoopStage2()
    {
        int iteration = 0;
        List<double[,]> seen = new List<double[,]>();
        // minimise the 2nd objective
        while (stage2)
        {
            int pivotColumn = ChooseLargestColumn();
            double[] ratios = RatioTest(pivotColumn);
            int pivotRow = FindPivotRow(ratios);
            if (pivotRow == -1) throw new RegionNotBoundedException();
            DisplayTableau(pivotRow, pivotColumn, iteration == 0 ? "Initial tableau:" : $"Iteration {iteration}:", ratios);

            if (dimension == 2 || displayBasic)
            {
                FindBasicVars(displayBasic, iteration == 0 ? "" : $"Iteration {iteration}");
            }

            iteration++;
            values = CreateNextTableau(pivotRow, pivotColumn);
            stage2 = DetermineStage2();
            SeenState(seen);
            seen.Add(values.Clone() as double[,]);
        }
        return iteration;
    }
    private void Reduce2Stage()
    {
        List<int> indexToRemove = variables.Select((x, i) => x.ToLower()[0] == ‘a’ ? i : -1).Where(i => i != -1).ToList();
        variables = variables.Select((x, i) => indexToRemove.
            Contains(i) ? "*" : x).Where(x => x != "*").ToArray()
            ;
        double[,] tempValues = new double[values.GetLength(0) -
            1, variables.Length];
        for (int i = 1; i < values.GetLength(0); i++)
        {
            int counter = 0;
            for (int j = 0; j < values.GetLength(1); j++)
            {
                if (indexToRemove.Contains(j)) continue;
                tempValues[i - 1, counter] = values[i, j];
                counter++;
            }
        }
        values = tempValues;
    }
    private void LoopStage1(bool reduced, int iteration)
    {
        bool solved = false;

        List<double[,]> seen = new List<double[,]>();
        while (!solved)
        {
            int pivotColumn = ChooseSmallestColumn();
            double[] ratios = RatioTest(pivotColumn);
            int pivotRow = FindPivotRow(ratios);
            if (pivotRow == -1) break;
            DisplayTableau(pivotRow, pivotColumn, iteration == 0
                ? "Initial tableau:" : reduced ? "Reduced tableau
                :" : $"Iteration {iteration}:", ratios);

            if (dimension == 2 || displayBasic)
            {
                FindBasicVars(displayBasic, iteration == 0 ? "" :
                    $"Iteration {iteration}", reduced);
            }

            iteration++;
            values = CreateNextTableau(pivotRow, pivotColumn);
            solved = DetermineSolved(seen);
            seen.Add(values.Clone() as double[,]);
            reduced = false;
        }
    }
    public void Solve()
    {
        if (!skipReformulation) Reformulate();
        Console.WriteLine();

        int iteration = 0;
        bool reduced = false;

        if (dimension == 2) plot = new Simplex2DGraph(stage2,
            values, constraints);

        try
        {
            // identify if 2 stage
            if (stage2)
            {
                    // Do the iterations to minimise A
                    iteration = LoopStage2();
                    DisplayTableau($"Iteration {iteration}:");

                    // reduce to a 1 stage problem
                    Reduce2Stage();
                    reduced = true;
                }

                // solve 1 stage simplex iteratively.
                LoopStage1(reduced, iteration);

                DisplayTableau("Final tableau:");
                FindBasicVars(true, "Final iteration");
                Console.WriteLine($"Maximised at: {variables[0]} = {
                    values[0, values.GetLength(1) - 1]}");

                if (dimension == 2) plot.DisplayPlot();
                else ConsoleHelper.WaitForKey();
            }
            catch (RegionNotBoundedException)
            {
                Console.WriteLine("Region was not bounded!");
                ConsoleHelper.WaitForKey();
            }
        }
}
```

Simplex class for displaying the 2D graph:

```
public class Simplex2DGraph
{
    private ScottPlot.Plot plot;
    public Simplex2DGraph(bool stage2, double[,] values, List<
        Equation> constraints)
    {
        // Set up formatting
        plot = new ScottPlot.Plot();
        plot.Palette = ScottPlot.Palette.OneHalfDark;
        plot.Title($"Linear Programming Problem");
        plot.Style(ScottPlot.Style.Black);
        Bitmap image = new Bitmap("A_black_image.jpg");
        plot.Style(figureBackgroundImage: image);
        plot.XAxis.TickLabelStyle(color: Color.WhiteSmoke,
            fontName: "comic sans ms");
        plot.YAxis.TickLabelStyle(color: Color.WhiteSmoke,
            fontName: "comic sans ms");

        FindPolygon(stage2, values, constraints);
    }
    private (double, double) FindIntersection(double m1, double
        c1, double m2, double c2)
    {
        // parallel
        if (m1 == m2) return (-1, -1);

        // handles equations in the form x = c
        if (double.IsInfinity(m1)) return (c1, m2 * c1 + c2);
        if (double.IsInfinity(m2)) return (c2, m1 * c2 + c1);

        double x = (c2 - c1) / (m1 - m2);
        double y = m1 * x + c1;
        return (x, y);
    }

    private List<(double, double)> FindAllIntersections(List<(
        double, double)> slopeIntercept)
    {
        List<(double, double)> lineIntersections = new List<(
            double, double)>();
        for (int i = 0; i < slopeIntercept.Count(); i++)
        {
            for (int j = i + 1; j < slopeIntercept.Count(); j++)
            {
                (double x, double y) = FindIntersection(
                    slopeIntercept[i].Item1, slopeIntercept[i].
                    Item2, slopeIntercept[j].Item1,
                    slopeIntercept[j].Item2);
                if (x < 0 || y < 0) continue;
                lineIntersections.Add((x, y));
            }
        }
        return lineIntersections;
    }

    private List<(double, double)> ParseConstraintsToGraph(int
        stage2Offset, double[,] values)
    {
        List<Func<double, double?>> functions = new List<Func<
            double, double?>>();
        double xMax = 0;
        double yMax = 0;

        List<(double, double)> lineIntersections = new List<(
            double, double)>() { (0, 0) };
        List<(double, double)> slopeIntercept = new List<(double,
             double)>();

        // create all the necessary functions
        for (int i = stage2Offset; i < values.GetLength(0); i++)
        {
            double xcoefficient = values[i, stage2Offset];
            double ycoefficient = values[i, stage2Offset + 1];
            double yintercept = values[i, values.GetLength(1) -
                1];

            // Checks if the y coefficient is 0 - this would
                result in a divide by 0 error.
            // Adds the gradient (m) along with the y intercept (
                c) to the list "gradientIntercepts" which is used
                 to find all intersections
            if (ycoefficient == 0)
            {
                plot.AddVerticalLine(yintercept / xcoefficient);
                slopeIntercept.Add((double.PositiveInfinity,
                    yintercept / xcoefficient));
            }
            else
            {
                functions.Add(new Func<double, double?>((x) => (
                    xcoefficient / ycoefficient) * -x +
                   yintercept / ycoefficient));
                slopeIntercept.Add((-xcoefficient / ycoefficient,
                    yintercept / ycoefficient));
            }

            // Find axis and add to list of intersection coords
            if (xcoefficient > 0)
            {
                lineIntersections.Add((yintercept / xcoefficient,
                    0));
                xMax = Math.Max(yintercept / xcoefficient, xMax);
            }
            if (ycoefficient > 0)
            {
                lineIntersections.Add((0, yintercept /
                   ycoefficient));
                yMax = Math.Max(yintercept / ycoefficient, yMax);
            }
        }
        foreach (Func<double, double?> func in functions)
        {
            plot.AddFunction(func, lineWidth: 2);
        }
        plot.SetAxisLimits(0, xMax + xMax / 5, 0, yMax + yMax /
           5);

        return lineIntersections.Concat(FindAllIntersections(
           slopeIntercept)).ToList();
    }
    private List<(double, double)> ValidateCoords(List<(double,
       double)> lineIntersections, List<Equation> constraints)
    {
        // Find all the vertices of the feasible region
        // Loop through and see if the intersection satisfies all
            the constraints
        List<(double, double)> validatedCoords = new List<(double
           , double)>();
        foreach ((double x, double y) in lineIntersections)
        {
            bool isInFR = true;
            foreach (Equation constraint in constraints)
            {
                double LHS = constraint.GetLHSValues()[0] * x +
                   constraint.GetLHSValues()[1] * y;
                if (!((constraint.symbol == "<=" && LHS <=
                   constraint.GetRHSValues()[0])
                    || (constraint.symbol == ">=" && LHS >=
                       constraint.GetRHSValues()[0])
                    || (constraint.symbol == "=" && LHS ==
                       constraint.GetRHSValues()[0])
                    || myMath.WithinPrecision(LHS, constraint.
                       GetRHSValues()[0])))
                {
                    isInFR = false;
                    break;
                }
            }
            if (isInFR)
            {
                validatedCoords.Add((x, y));
            }
        }
        return validatedCoords;
    }
    private (double[], double[]) FindPolygonRoute(List<(double, double)> coords)
    {
        (double x, double y) mean = (coords.Sum(x => x.Item1) / coords.Count, coords.Sum(x => x.Item2) / coords.Count);
        coords.Sort((x1, x2) => myMath.Argument(x1.Item1 - mean.x, x1.Item2 - mean.y).CompareTo(myMath.Argument(x2.Item1 - mean.x, x2.Item2 - mean.y)));
        return (coords.Select(x => x.Item1).ToArray(), coords.Select(x => x.Item2).ToArray());
    }
    private void FindPolygon(bool stage2, double[,] values, List<Equation> constraints)
    {
        int stage2Offset = stage2 ? 2 : 1;
        // Find the polygon that is the feasible region and add it to the graph
        List<(double, double)> lineIntersections = ParseConstraintsToGraph(stage2Offset, values);
        List<(double, double)> validatedCoords = ValidateCoords(lineIntersections, constraints);
        (double[] xCoords, double[] yCoords) = FindPolygonRoute(validatedCoords);
        plot.AddPolygon(xCoords.ToArray(), yCoords.ToArray(), plot.GetNextColor(0.5));
    }


    public void AddLinePointToPlot(int row, string iteration, double[,] values, string[] variables)
    {
        plot.AddFunction(new Func<double, double?>((x) => -1 * x + values[row, variables.Length - 1]), color: Color.DarkBlue);
        plot.AddText($"{iteration} (x + y = {values[row, variables.Length - 1]})", 0, 0, size: 10, color: Color.DarkBlue);
    }
    public void PlotMarker(string iteration, double x, double y, double[,] values)
    {
        var marker = plot.AddMarker(x, y, size: 7);
        marker.Text = $"{iteration} ({myMath.SigFig(x)}, {myMath.SigFig(y)}), P = {myMath.SigFig(values[0, values.GetLength(1) - 1])}";
        marker.TextFont.Color = Color.White;
        marker.TextFont.Alignment = Alignment.LowerLeft;
        marker.TextFont.Size = 12;
        marker.TextFont.Name = "Comic Sans MS";
    }

    private void LaunchInteractiveWindow()
    {
        new ScottPlot.FormsPlotViewer(plot, windowWidth: 800, windowHeight: 800, windowTitle: "Simplex Problem").
```

```
            ShowDialog();
    }
    public void DisplayPlot()
    {
        plot.SetViewLimits(0, double.PositiveInfinity, 0, double.PositiveInfinity);
        Thread interactiveWindow = new Thread(LaunchInteractiveWindow);
        interactiveWindow.Start();
        Console.WriteLine("Would you like to save it as an image?");
        string response = new Menu(new List<string>() { "Yes", "No" }, '>').Select();

        if (response == "Yes")
        {
            Console.Write("Enter a name to save the file as: ");
            try
            {
                plot.SaveFig($"images\\{Console.ReadLine()}.png");
            }
            catch (System.Runtime.InteropServices.ExternalException)
            {
                Console.WriteLine("Unfortunately you couldn't name it that.");
            }
        }
        ConsoleHelper.WaitForKey();

        Console.WriteLine("Please wait for the interactive window to close (you can speed this up by closing it manually)..");
        while (interactiveWindow.IsAlive) interactiveWindow.Abort();
        ConsoleHelper.ClearLine(Console.CursorTop - 1);
    }
}
```

5.2.5 Dijkstra's Algorithm

Priority Queue

```
public class PriorityQueue
{
    private List<DijkstraNode> heap;
    private int length;

    public PriorityQueue()
    {
        heap = new List<DijkstraNode>();
        length = 0;
    }

    public void Enqueue(DijkstraNode node)
    {
        heap.Add(node);
        int pos = length++;

        // sift-up
        while (pos > 0)
        {
            if (heap[(pos - 1) / 2].weight > node.weight)
            {
                (heap[(pos - 1) / 2], heap[pos]) = (heap[pos], heap[(pos - 1) / 2]);
                pos = (pos - 1) / 2;
            }
            else break;
        }
    }
    public DijkstraNode Dequeue()
    {
        length--;
        DijkstraNode node = heap[0];
        heap[0] = heap[length];
        heap.RemoveAt(length);
        int pos = 0;

        if (length > 1)
        {
            // sift-down
            while (true)
            {
                double current = heap[pos].weight;
                double left = (pos * 2 + 1 >= length) ? double.PositiveInfinity : heap[pos * 2 + 1].weight;
                double right = (pos * 2 + 2 >= length) ? double.PositiveInfinity : heap[pos * 2 + 2].weight;

                if (current <= left && current <= right) break;

                if (left <= right)
                {
                    (heap[pos], heap[pos * 2 + 1]) = (heap[pos * 2 + 1], heap[pos]);
                    pos = pos * 2 + 1;
                }
                else
                {
                    (heap[pos], heap[pos * 2 + 2]) = (heap[pos * 2 + 2], heap[pos]);
                    pos = pos * 2 + 2;
                }
            }
        }

        return node;
    }
    public int GetLength() => length;
}
```

Dijkstra Node

```
public class DijkstraNode
{
    public char name;
    public int order;
    public bool visited;
    public string route;
    private List<double> previous;
    public double weight;

    public DijkstraNode(char name, bool start)
    {
        this.name = name;
        visited = false;
        previous = new List<double>();
        route = "";

        if (start)
        {
            weight = 0;
            previous.Add(0);
            route += name;
        }
        else weight = double.PositiveInfinity;
    }

    public void DecreaseWeight(double value)
    {
        previous.Add(value);
        weight = value;
    }

    public override string ToString()
    {
        if (double.IsPositiveInfinity(weight)) return $"{name}:
            Not Visited";
        string bottomLine = $"|{(previous.Count == 1 ? $"{
            previous[0]}".PadRight(weight.ToString().Length) :
            String.Join(",", previous))}";
        string topLine = "|" + $"{order}".PadRight(bottomLine.
            Length / 2 - 1) + "|" + $"{weight}|".PadLeft(
            bottomLine.Length / 2 - (bottomLine.Length+1)%2);
        bottomLine += "|".PadLeft(topLine.Length - bottomLine.
            Length);
        return $"{name}: {String.Join(",", route.ToCharArray())}\
            n{topLine}\n{bottomLine}";
    }
}
```

Dijkstra creator class:

```
public class DijkstraCreator : IProblem
{
    DijkstraSolver problem;
    private char startNode;
    public void GenerateQuestion(Random rnd)
    {
        int difficulty = new OptionScroller("Choose Difficulty: "
            , 1, 6).Select();
        Graph graph = new Graph(rnd.Next(0, 2) + (difficulty - 1)
             * 2 + 4);
        graph.GenerateGraph(rnd, difficulty <= 2 || rnd.Next(0,
            2) != 0);
        problem = new DijkstraSolver(graph);
        startNode = graph.GetNodeNames()[rnd.Next(graph.
            GetNodeNames().Length)];

        Console.WriteLine($"Perform Dijkstra's algorithm from {
            startNode} on the following network:\n{graph}");
    }

    public void InputQuestion()
    {
        int vertices = new OptionScroller("Number of vertices: ",
             3, 26).Select();
        Console.WriteLine($"Number of vertices: {vertices}");
        Graph graph = new Graph(vertices);
        startNode = graph.GetNodeNames()[new OptionScroller("
            Start node: ", 0, vertices - 1, true).Select()];
        Console.WriteLine($"Start node: {startNode}");
        graph.InputGraph(new Menu(new List<string>() { "
            Undirected (symmetrical) Graph", "Directed Graph"}).
            SelectOption() == 0, true);
        problem = new DijkstraSolver(graph);
        Console.WriteLine(graph + Environment.NewLine);
    }

    public void GenerateAnswer()
    {
        if (!(problem is null)) problem.Solve(startNode);
        else Console.WriteLine("There is no problem to solve!!");
        ConsoleHelper.WaitForKey();
    }
}
```

Dijkstra solver class:

```
public class DijkstraSolver
{
    Graph network;

    public DijkstraSolver(Graph network) => this.network =
        network;

    private Dictionary<char, DijkstraNode> SetUpDict(char
        startNode)
    {
        Dictionary<char, DijkstraNode> nodeDict = new Dictionary<
            char, DijkstraNode>();
        foreach (char c in network.GetNodeNames()) nodeDict.Add(c
            , new DijkstraNode(c, startNode == c));
        return nodeDict;
    }
    public void Solve(char startNode)
    {
        Dictionary<char, DijkstraNode> nodeDict = SetUpDict(
            startNode);
        PriorityQueue q = new PriorityQueue();
        q.Enqueue(nodeDict[startNode]);
        int order = 1;

        while (q.GetLength() > 0)
        {
            DijkstraNode currentNode = q.Dequeue();
            if (currentNode.visited) continue;
            currentNode.visited = true;
            currentNode.order = order++;

            foreach(KeyValuePair<char, double> node in network.
                GetConnections(currentNode.name))
            {
                if (currentNode.weight + node.Value < nodeDict[
                    node.Key].weight)
                {
                    nodeDict[node.Key].DecreaseWeight(currentNode
                        .weight + node.Value);
                    nodeDict[node.Key].route = currentNode.route
                        + node.Key;
                    q.Enqueue(nodeDict[node.Key]);
                }
            }
        }
        foreach (DijkstraNode node in nodeDict.Values) Console.
            WriteLine(node);
    }
}
```

5.2.6 Minimum Spanning Trees

Disjoint set

```
public class DisjointSet
{
    private int[] set;
    private Dictionary<char, int> map;
    public DisjointSet(char[] nodes)
    {
        // Creates a unique set for each node
        set = new int[nodes.Length];
        map = new Dictionary<char, int>();

        for (int i = 0; i < nodes.Length; i++)
        {
            set[i] = i;
            map.Add(nodes[i], i);
        }
    }

    private int Find(int node)
    {
        if (set[node] != node) return Find(set[node]);
        return node;
    }

    public bool Unify(string edge)
    {
        int p1 = Find(map[edge[0]]);
        int p2 = Find(map[edge[1]]);

        if (p1 == p2) return false;
        set[p2] = p1;
        return true;
    }
}
```

MST creator class

```
public class MSTCreator : IProblem
```

```
{
    MSTSolver problem;
    bool prims;
    bool kruskals;
    public void GenerateQuestion(Random rnd)
    {
        int difficulty = new OptionScroller("Choose Difficulty: "
            , 1, 6).Select();
        Graph graph = new Graph(rnd.Next(0, 2) + (difficulty - 1)
             * 2 + 4);
        graph.GenerateGraph(rnd);
        problem = new MSTSolver(graph);
        prims = rnd.Next(0, 2) == 0;
        kruskals = !prims;

        Console.WriteLine($"Perform {(prims ? "Prim" : "Kruskal")
            }'s algorithm on the following network:\n{graph}\n");
    }
    private void EvaluateChoice(int choice)
    {
        if (choice == 0)
        {
            prims = true;
            kruskals = false;
        }
        else if (choice == 1)
        {
            prims = false;
            kruskals = true;
        }
        else
        {
            prims = true;
            kruskals = true;
        }
    }
    public void InputQuestion()
    {
        int vertices = new OptionScroller("Number of vertices: ",
             3, 26).Select();
        Console.WriteLine($"Number of vertices: {vertices}");
        Graph graph = new Graph(vertices);

        EvaluateChoice(new Menu(new List<string>() { "Solve Prim'
            s", "Solve Kruskal's", "Both" }).SelectOption());
        if (prims) Console.WriteLine("Solve using Prim's");
        if (kruskals) Console.WriteLine("Solve using Kruskal's");

        graph.InputGraph();
        Console.WriteLine(graph + Environment.NewLine);
        problem = new MSTSolver(graph);
    }
    public void GenerateAnswer()
    {
        if (problem is null) Console.WriteLine("There is no
            problem to solve!!");
        else
        {
            if (prims) problem.SolvePrims();
            if (kruskals) problem.SolveKruskals();
        }
        ConsoleHelper.WaitForKey();
    }
}
```

MST solver class

```
public class MSTSolver
{
    private Graph network;

    public MSTSolver(Graph network) => this.network = network;

    private Queue<(string, double)> SortAllEdges()
    {
        List<(string, double)> allEdges = new List<(string,
            double)>();

        // Get all edges
        double[,] matrix = network.GetAdjacencyMatrix();
        for (int i = 0; i < matrix.GetLength(0) - 1; i++)
        {
            for (int j = i + 1; j < matrix.GetLength(1); j++)
            {
                if (matrix[i, j] != 0) allEdges.Add(($"{(char)(i
                    + 65)}{(char)(j + 65)}", matrix[i, j]));
            }
        }

        // sort by weight
        return new Queue<(string, double)>(allEdges.OrderBy(x =>
            x.Item2));
    }
    public void SolveKruskals()
    {
        DisjointSet sets = new DisjointSet(network.GetNodeNames()
            );

        int accepted = network.GetNodeNames().Length - 1;
        // edge name, weight, in MST?
        List<(string, double, bool)> edges = new List<(string,
            double, bool)>();
        Queue<(string, double)> allEdges = SortAllEdges();

        while (accepted > 0)
        {
            if (allEdges.Count == 0)
            {
                ConsoleHelper.ColourText(ConsoleColor.Red,
                    ConsoleColor.Gray, "The tree is not connected
                    !\n");
                break;
            }
            (string edge, double weight) = allEdges.Dequeue();
            bool inMST = sets.Unify(edge);
            edges.Add((edge, weight, inMST));
            if (inMST) accepted--;
        }

        Console.WriteLine($"Kruskal's Algorithm: \nMST of weight
            {edges.Sum(x => x.Item3? x.Item2 : 0)}");
        for (int i = 0; i < edges.Count; i++)
        {
            Console.WriteLine($"{i + 1}: {edges[i].Item1} - {edges[i].Item2} {(edges[i].Item3? "Θ" : "X")}");
        }
    }
    public void SolvePrims()
    {
        List<char> visited = new List<char>() { ‘A’ };
        List<(string, double)> tree = new List<(string, double)>();

        for (int i = 0; i < network.GetNodeNames().Length - 1; i++)
        {
            double currentMin = double.PositiveInfinity;
            char currentNode = ‘A’;
            char currentNodePair = ‘A’;

            foreach(char node in visited)
            {
                foreach(KeyValuePair<char, double> connection in network.GetConnections(node))
                {
                    if (connection.Value < currentMin && !visited.Contains(connection.Key))
                    {
                        currentMin = connection.Value;
                        currentNode = connection.Key;
                        currentNodePair = node;
                    }
                }
            }

            if (double.IsInfinity(currentMin))
            {
                ConsoleHelper.ColourText(ConsoleColor.Red, ConsoleColor.Gray, "The tree is not connected!\n");
                break;
            }

            visited.Add(currentNode);
            tree.Add(($"{currentNodePair}{currentNode}", currentMin));
        }
        Console.WriteLine($"Prim’s Algorithm: \nMST of weight {tree.Sum(x => double.IsInfinity(x.Item2) ? 0 : x.Item2)}");
        for (int i = 0; i < tree.Count; i++)
        {
            Console.WriteLine($"{i + 1}: {tree[i].Item1} - {tree[i].Item2}");
        }
        Console.WriteLine();
    }
}
```

5.2.7 Quick Sort

Quick sort creator class

```
public class QuickSortCreator : IProblem
{
    QuickSortSolver<char> cProblem;
    QuickSortSolver<double> dProblem;

    public void GenerateQuestion(Random rnd)
    {
        int difficulty = new OptionScroller("Choose Difficulty: "
            , 1, 5).Select();
        bool alphabet = rnd.Next(0, 2) == 1;
        bool ascending = rnd.Next(0, 2) == 1;

        if (!alphabet) // numbers
        {
            double[] dValues = new double[rnd.Next(4, 7) + (
               difficulty - 1) * 2];
            for (int i = 0; i < dValues.Length; i++)
            {
                if (difficulty <= 2) dValues[i] = rnd.Next(0,
                    200);
                else dValues[i] = rnd.Next(-99, 200);
            }
            dProblem = new QuickSortSolver<double>(dValues,
               ascending);
            Console.WriteLine($"Sort by {(ascending ? "ascending"
                : "descending")}: {String.Join(", ", dValues)}")
               ;
        }
        else // letters
        {
            char[] cValues = new char[rnd.Next(4, 7) + (
               difficulty - 1) * 2];
            for (int i = 0; i < cValues.Length; i++)
            {
                cValues[i] = (char)(rnd.Next(0, 26) + 65);
            }
            cProblem = new QuickSortSolver<char>(cValues,
               ascending);
            Console.WriteLine($"Sort by {(ascending ? "ascending"
                : "descending")}: {String.Join(", ", cValues)}")
               ;
        }
    }

    public void InputQuestion()
    {
        Console.WriteLine("Enter a list of letters or numbers
            separated by commas:");
        List<string> input = Console.ReadLine().Split(',').Select
            (x => x.Trim()).ToList();
        bool ascending = new Menu(new List<string>() { "Sort
            ascending", "Sort descending" }).SelectOption() == 0;
        Console.WriteLine("Bad values are omitted.");

        if (double.TryParse(input[0], out double d))
        {
            List<double> dValues = new List<double>();
            foreach (string s in input)
            {
                if (double.TryParse(s, out double num)) dValues.
                    Add(num);
            }
            dProblem = new QuickSortSolver<double>(dValues.
                ToArray(), ascending);
        }
        else
        {
            List<char> cValues = new List<char>();
            foreach (string s in input)
            {
                if (char.TryParse(s, out char c)) cValues.Add(c);
            }
            cProblem = new QuickSortSolver<char>(cValues.ToArray
                (), ascending);
        }
    }
    public void GenerateAnswer()
    {
        if (!(cProblem is null)) cProblem.Solve();
        else if (!(dProblem is null)) dProblem.Solve();
        else Console.WriteLine("There is no problem to solve!");
    }
}
```

Quick sort solver class

```
public class QuickSortSolver<T>
{
    private bool asc;
    private T[] values;
    private bool[] sorted;

    public QuickSortSolver(T[] values, bool ascending) => (this.
        values, asc, sorted) = (values, ascending, new bool[
        values.Length]);

    private List<int> FindPivots()
    {
        List<int> pivots = new List<int>();
        for (int i = 0; i < sorted.Length; i++)
        {
            if (sorted[i] == false && (i == 0 || sorted[i - 1] ==
                 true)) pivots.Add(i);
        }
        return pivots;
    }

    private void DisplayFinal()
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine(String.Join(", ", values));
        Console.ForegroundColor = ConsoleColor.Gray;
    }
    private void DisplayIteration(List<int> pivots)
    {
        for (int i = 0; i < sorted.Length; i++)
        {
            if (sorted[i]) Console.ForegroundColor = ConsoleColor
```

```
                .Green;
            else if (pivots.Contains(i)) Console.ForegroundColor
                = ConsoleColor.Red;
            Console.Write($"{values[i]}{(i != sorted.Length - 1?
                ", " : "")}");
            Console.ForegroundColor = ConsoleColor.Gray;
        }
        Console.WriteLine();
    }

    private double CMP(T x, T y) => (dynamic)x - (dynamic)y;

    private List<T> AddSorted(int start)
    {
        List<T> append = new List<T>();
        for (int i = start; i < sorted.Length; i++)
        {
            if (!sorted[i]) break;
            append.Add(values[i]);
        }
        return append;
    }
    private T[] PerformSwaps(List<int> pivots)
    {
        List<T> swaps = new List<T>();

        // Append all sorted items before the first pivot
        swaps = swaps.Concat(AddSorted(0)).ToList();
        foreach (int loc in pivots)
        {
            // Sort the parts into 2 categories: <= (leq) or > (
                gt) pivot
            List<T> leq = new List<T>();
            List<T> gt = new List<T>();
            int append = sorted.Length;

            for (int i = loc + 1; i < sorted.Length; i++)
            {
                if (sorted[i])
                {
                    append = i;
                    break;
                }
                if (CMP(values[i], values[loc]) <= 0) leq.Add(
                    values[i]);
                else gt.Add(values[i]);
            }

            // Add the categories in the correct order
            swaps = swaps.Concat(asc ? leq : gt).ToList();
            sorted[swaps.Count()] = true;
            swaps.Add(values[loc]);
            swaps = swaps.Concat(asc ? gt : leq).ToList();

            //Append all sorted items before the next pivot
            swaps = swaps.Concat(AddSorted(append)).ToList();
        }
        return swaps.ToArray();
    }

    public void Solve()
    {
        Console.WriteLine();
        while (!sorted.All(x => x == true))
        {
            List<int> pivots = FindPivots();
            DisplayIteration(pivots);
            values = PerformSwaps(pivots);
        }
        DisplayFinal();
        ConsoleHelper.WaitForKey();
    }
}
```

Appendix A

Prototype code

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NEA_prototype
{
    internal class Program
    {
        static void Main(string[] args)
        {
            SimplexSolver solver = new SimplexSolver();
            solver.solve();
            Console.ReadKey();
        }
    }

    public class SimplexSolver
    {
        private const int decimalPrecision = 2;
        private string[] variables;
        private double[,] values;
        private bool maximising, stage2, dimension2;

        //this will be the sole input function after testing.
        public SimplexSolver(string objective, List<string> constraints, bool maximisation)
        {
            //accept a problem to be able to be solved.
            maximising = maximisation;

            //create custom exception to return if there is no feasible region.
        }

        //for initial testing
        public SimplexSolver(bool test2Stage = true)
        {
            if (test2Stage)
            {
                //two stage test
                variables = new string[] { "A", "P", "x1", "x2", "s1", "s2", "s3", "a1", "RHS" };
                values = new double[,] {
```

```
                { 1, 0, 1, 0, 0, 0, -1, 0, 2, },
                { 0, 1, -5, -7, 0, 0, 0, 0, 0 },
                { 0, 0, 2, 3, 1, 0, 0, 0, 30 },
                { 0, 0, 1, 1, 0, 1, 0, 0, 12 },
                { 0, 0, 1, 0, 0, 0, -1, 1, 2 } };
                maximising = true;
                stage2 = true;
            }
            else
            {
                //one stage test
                variables = new string[] { "P", "x", "y", "s1", "
                    s2", "RHS" };
                values = new double[,] {
                    { 1, -5, -2, 0, 0, 0 },
                    { 0, 1, 4, 1, 0, 24 },
                    { 0, 2, 3, 0, 1, 23 } };
                maximising = true;
                stage2 = false;
            }
        }

        private void colourText(ConsoleColor colour, ConsoleColor
            originalColour, string text)
        {
            Console.ForegroundColor = colour;
            Console.Write(text);
            Console.ForegroundColor = originalColour;
        }
        private void displayTableau(string message)
        {
            //The pivot column and row are off the grid so they
                can't be highlighted
            //There are no ratio tests to be shown
            displayTableau(-1, -1, message, Array.Empty<double>()
                );
        }
        private void displayTableau(int pivotRow, int pivotCol,
            string iterationMessage, double[] ratios)
        {
            // Display the current iteration of the tableau. The
                final tableau is set to have iteration = -1.
            Console.WriteLine(iterationMessage);

            //The following for loop finds out how much space
                each variable should take up when being displayed
                 on the console
            int[] largestSizes = new int[variables.Length];
            for (int i = 0; i < variables.Length; i++)
            {
                int largestSize = variables[i].Length;
                for (int j = 0; j < values.GetLength(0); j++)
                {
                    //increase the corresponding largest size if
                        the variable in the column is wider
                    if (Math.Round(values[j, i], decimalPrecision
                        ).ToString().Length > largestSize)
                        largestSize = Math.Round(values[j, i],
                        decimalPrecision).ToString().Length;
                }
                largestSize++;
                largestSizes[i] = largestSize;

                if (i == pivotCol) colourText(ConsoleColor.Yellow, ConsoleColor.Gray, $"{variables[i]}".PadRight(largestSize) + "|");
                else Console.Write($"{variables[i]}".PadRight(largestSize) + "|");
            }
            Console.WriteLine(ratios.Length > 0? "Ratio Test" : "");

            for (int i = 0; i < values.GetLength(0); i++)
            {
                //checks if it is a pivot, then changes colour accordingly
                if (i == pivotRow) Console.ForegroundColor = ConsoleColor.Magenta;
                for (int j = 0; j < values.GetLength(1); j++)
                {
                    string text = Math.Round(values[i, j], decimalPrecision).ToString().PadRight(largestSizes[j]) + "|";
                    if (i == pivotRow && j == pivotCol) colourText(ConsoleColor.Red, ConsoleColor.Magenta, text);
                    else if (j == pivotCol) colourText(ConsoleColor.Yellow, ConsoleColor.Gray, text);
                    else Console.Write(text);
                }
                Console.ForegroundColor = ConsoleColor.Gray;

                // display ratio test
                if (ratios.Length == 0) Console.WriteLine();
                else if ((stage2 && i >= 2 && values[i, pivotCol] == 0) || (!stage2 && i >= 1 && values[i, pivotCol] == 0)) Console.WriteLine("undefined");
                else if (stage2 && i >= 2) Console.WriteLine(Math.Round(ratios[i - 2], decimalPrecision));
                else if (!stage2 && i >= 1) Console.WriteLine(Math.Round(ratios[i - 1], decimalPrecision));
                else Console.WriteLine();
            }
            Console.WriteLine();
        }
        private int chooseLargestColumn()
        {
            double largest = values[0, 2];
            int column = 2;
            for (int i = 1; i < values.GetLength(1) - 1; i++)
            {
                if (values[0, i] > largest)
                {
                    largest = values[0, i];
                    column = i;
                }
            }
            return column;
        }
        private int chooseSmallestColumn()
        {
            double smallest = values[0, 1];
            int column = 1;
            for (int i = 1; i < values.GetLength(1) - 1; i++)
            {
                if (values[0, i] < smallest)
                {
                    smallest = values[0, i];
                    column = i;
                }
            }
            return column;
        }
        private double[] ratioTest(int pivotColumn)
        {
            int buffer = stage2 ? 2 : 1;
            double[] tests = new double[values.GetLength(0) -
                buffer];
            for (int i = buffer; i < values.GetLength(0); i++)
            {
                if (values[i, pivotColumn] != 0) tests[i - buffer
                    ] = values[i, values.GetLength(1) - 1] /
                    values[i, pivotColumn];
                else
                {
                    // -1 is a temporary value. In the
                        DisplayTableau() procedure it will
                        display as "undefined"
                    tests[i - buffer] = -1;
                }
            }
            return tests;
        }
        private int findPivotRow(double[] tests)
        {
            double min = tests.Max();
            if (min < 0) throw new regionNotBoundedException();
            int row = 0;
            for (int i = 0; i < tests.Length; i++)
            {
                if (tests[i] >= 0 && tests[i] < min)
                {
                    min = tests[i];
                    row = i + 1;
                }
            }
            return stage2? row + 1 : row;
        }
        private double[,] createNewTable(int pivotRow, int
            pivotCol)
        {
            // Perform the iteration using the pivot row and
                column
            double[,] newTableau = new double[values.GetLength(0)
                , values.GetLength(1)];
            double pivot = values[pivotRow, pivotCol];

            for (int i = 0; i < values.GetLength(1); i++)
            {
                newTableau[pivotRow, i] = values[pivotRow, i] /
                    pivot;
            }

            double multiplier;
            for (int i = 0; i < values.GetLength(0); i++)
            {
                if (i == pivotRow) continue;
                multiplier = values[i, pivotCol] / pivot;
                for (int j = 0; j < values.GetLength(1); j++)
                {
                    newTableau[i, j] = values[i, j] - multiplier
                        * values[pivotRow, j];
                }
            }

            return newTableau;
        }

        private void displayBasicVars()
        {
            int[] rowOfBasic = new int[variables.Length - 1];

            // identify if each variable is basic and the row it
                corresponds to
            for (int i = 0; i < variables.Length - 1; i++)
            {
                bool basic = true;
                int zeroes = 0;
                int ones = 0;
                int oneLoc = 0;
                for (int j = 1; j < values.GetLength(0); j++)
                {
                    if (values[j, i] == 0) zeroes++;
                    else if (values[j, i] == 1)
                    {
                        ones++;
                        oneLoc = j;
                    }
                    else
                    {
                        basic = false;
                        break;
                    }
                }
                if (basic && ones == 1) rowOfBasic[i] = oneLoc;
                else rowOfBasic[i] = 0;
            }

            List<string> nonBasic = new List<string>();
            Console.WriteLine("Basic variables: ");
            for (int i = 1; i < variables.Length - 1; i++)
            {
                if (rowOfBasic[i] == -1) continue;
                if(rowOfBasic[i] == 0)
                {
                    nonBasic.Add(variables[i]);
                }
                else
                {
                    // gather all other variables on the same row .
                    int row = rowOfBasic[i];
                    int[] occurences = rowOfBasic.Select((x, j) => x == row? j : -1).Where(j => j != -1).ToArray();
                    List<string> namedOccurences = new List<string>();

                    for (int j = 0; j < occurences.Length; j++)
                    {
                        namedOccurences.Add(variables[occurences[j]]);
                        rowOfBasic[occurences[j]] = -1;
                    }
                    Console.WriteLine(String.Join(" + ", namedOccurences) + " = " + values[row, variables.Length - 1]);
                }
            }
            Console.WriteLine("Non-basic variables: ");
            Console.WriteLine(String.Join(", ", nonBasic) + " = 0");
        }

        private bool determineStage2()
        {
            for (int i = 0; i < values.GetLength(1); i++)
            {
                if (values[0, i] > 0) return false;
            }
            return true;
        }
        private bool determineSolved()
        {
            for (int i = 0; i < values.GetLength(1); i++)
            {
                if (values[0, i] < 0) return false;
            }
            return true;
        }

        public void solve()
        {
            int iteration = 0;
            bool solved = false;
            bool reduced = false;

            //identify if 2 stage
            if (stage2)
            {
                // minimise the 2nd objective
                while (stage2)
                {
                    int pivotColumn = chooseLargestColumn();
                    double[] ratios = ratioTest(pivotColumn);
                    int pivotRow = findPivotRow(ratios);
                    displayTableau(pivotRow, pivotColumn, iteration == 0? "Initial tableau:" : $"
                        Iteration {iteration}:", ratios);
                    iteration++;
                    values = createNewTable(pivotRow, pivotColumn
                        );
                    stage2 = determineStage2();
                }

                displayTableau($"Iteration {iteration}:");
                // reduce to a 1 stage problem
                List<int> indexToRemove = variables.Select((x, i)
                     => x.ToLower()[0] == 'a' ? i : -1).Where(i
                    => i != -1).ToList();
                variables = variables.Select((x, i) =>
                    indexToRemove.Contains(i)? "*" : x).Where(x
                    => x != "*").ToArray();
                double[,] tempValues = new double[values.
                    GetLength(0) - 1, variables.Length];
                for (int i = 1; i < values.GetLength(0); i++)
                {
                    int counter = 0;
                    for (int j = 0; j < values.GetLength(1); j++)
                    {
                        if (indexToRemove.Contains(j)) continue;
                        tempValues[i - 1, counter] = values[i, j
                            ];
                        counter++;
                    }
                }
                values = tempValues;
                reduced = true;
            }

            //solve 1 stage simplex iteratively.
            while (!solved)
            {
                int pivotColumn = chooseSmallestColumn();
                double[] ratios = ratioTest(pivotColumn);
                int pivotRow = findPivotRow(ratios);
                displayTableau(pivotRow, pivotColumn, iteration
                    == 0 ? "Initial tableau:" : reduced? "Reduced
                     tableau:" : $"Iteration {iteration}:",
                    ratios);
                iteration++;
                values = createNewTable(pivotRow, pivotColumn);
                solved = determineSolved();
                reduced = false;
            }
            displayTableau("Final tableau:");
            displayBasicVars();
            Console.WriteLine($"{(maximising? "Maximised":"
                Minimised")} at: {variables[0]} = {values[0,
                values.GetLength(1) - 1]}");
        }
    }

    //Custom exception class
    public class regionNotBoundedException : Exception
    {
        public regionNotBoundedException() { }
        public regionNotBoundedException(string message) : base(
                message) { }
        }
}
```

Bibliography

[1] Microsoft automatic graph layout, December 2007.
Available at: `https://tinyurl.com/3268844a` (Accessed: January 2024).

[2] GeeksforGeeks. Quicksort – data structure and algorithm tutorials, May 2023.
Available at: `https://www.geeksforgeeks.org/quick-sort/` (Accessed: June 2023).

[3] Jesús Lagares. How does dijkstra's algorithm work?, Sep 2022.
Available at: `https://tinyurl.com/udkzfxv5` (Accessed: June 2023).

[4] Programiz. Greedy algorithm.
Available at: `https://tinyurl.com/5n8tve4w` (Accessed: June 2023).

[5] Wikipedia. Bin packing problem.
Available at: `https://en.wikipedia.org/wiki/Bin_packing_problem` (Accessed: January 2024).

[6] Wikipedia. Dijkstra's algorithm.
Available at: `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm` (Accessed: June 2023).

[7] Wikipedia. Kruskal's algorithm.
Available at: `https://en.wikipedia.org/wiki/Kruskal%27s_algorithm` (Accessed: June 2023).

[8] Wikipedia. Linear programming.
Available at: `https://en.wikipedia.org/wiki/Linear_programming` (Accessed: June 2023).

[9] Wikipedia. Loop-erased random walk: Uniform spanning tree.
Available at: `https://en.wikipedia.org/wiki/Loop-erased_random_walk#Uniform_spanning_tree` (Accessed: August 2023).

[10] Wikipedia. Np-completeness.
Available at: `https://en.wikipedia.org/wiki/NP-completeness` (Accessed: January 2024).

[11] Wikipedia. Prim's algorithm.
Available at: `https://en.wikipedia.org/wiki/Prim%27s_algorithm` (Accessed: June 2023).

[12] Wikipedia. Simplex algorithm, May 2023.
Available at: `https://en.wikipedia.org/wiki/Simplex_algorithm` (Accessed: June 2023).