EVALUATION

TESTING

In this section, I will test the functionality, robustness and the usability of the system in order to evaluate how well the system functions and performs overall.

FUNCTION

Testing for function is concerned with checking that all the parts of the system work as expected and provide correct results. During development, I tested each Prototype against the Success Criteria in order to check that the requirements have been met or take note of the criteria that were only partially met due to limitations in my code, such as in the Graph Input section. However, in this evaluation stage of the Software Development Life Cycle, I will test the system using the Black Box Testing table I created in the DESIGN section, which is based off the Success Criteria but doesn't check against each one as specifically as the testing at the end of each Prototype did.

Furthermore, there are many elements in this system that can't be easily displayed as screenshots due to the animations and visual nature of the code. Therefore, I will make reference to times in the video evidence given in order to show clearly how the different sections of the system function.

SUCCESS CRITERIA	INPUT	EXPECTED RESULT	JUSTIFICATION	REAL RESULT	CODE
[1][a,b,c, e]	Open the program.	The login page should load.	This test ensures that the program starts properly and displays the correct window on startup.	Timing: 00.00.00 As expected	1.1.b 1.1.c 1.1.d 1.1.e 1.1.f 1.1.g
[2][a,b,c,	Attempt to register	Once the Register	This ensures that the	For a student account, the timings were: 00.00.02 -	1.1.c 1.1.e

d,e,f,g,h]	successfully.	button has been pressed, if all inputs are valid, the home page will be opened. Otherwise, messages will display what has been incorrectly input.	user can register correctly and gain access to the rest of the system.	00.01.20 For a teacher account , the timings were: 00.13.16 - 00.14.06 As expected. Additionally, this evidence shows how the registration checks for an error in inputting the passwords, which worked as expected, and that the user is registered in the database.	1.1.f 1.2.k 1.2.l 1.2.m 1.2.n 1.2.o 1.2.q 1.2.r 1.2.s 1.3.b 1.3.c 1.3.d
[1][d] [3][c,d]	Attempt to login successfully.	Once the Login button has been pressed, the home page will be opened if the inputs were correct. On the home page, there will be a quiz statistics chart and a button to the quiz page for a student account. Otherwise, a message will display if an input is incorrect.	This ensures that a student user can log in successfully and gain access to the rest of the system if they have already registered. It also ensures that the correct type of home page is displayed.	For inputting incorrect details, the timings are: 00.12.12 - 00.12.41 For correctly logging in, the timings are: 00.12.41 - 00.12.55	1.1.c 1.1.d 1.1.f 1.2.a 1.2.b 1.2.d 1.2.i 1.3.b
[3][e]	Attempt to log out.	The login page will be displayed.	This ensures that a user can exit the system properly.	Timing: 00.12.05 As expected.	2.1.a 2.1.b 1.1.d 1.1e 1.1.1 1.3.b 1.3.d

[3][b] [4][a,b,c, d,e,k]	Open Bubble Sort Visualisaiton page.	The Bubble Sort Visualisation page will open, with fields to enter the dataset into and a bar chart, on which the dataset will be displayed.	This ensures that the Bubble Sort page can be accessed successfully, therefore enabling the visualisation of the algorithm.	Timing: 00.01.24 - 00.01.26 As expected.	2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.E 2.1.G 2.1.H 2.1.I 2.3.G 2.3.I
[4][d,f,g, h,i,j]	Input a dataset and visualise the Bubble Sort algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the sorting of the dataset will occur in both the text version and on the bar chart.	This ensures that the visualisation functionality and the validation for the Bubble Sort algorithm execute properly.	Timing: 00.01.28 - 00.02.08 As expected and shows how the system deals with an invalid dataset input. From 00.02.09 - 00.02.14, the visualisation of the sorting of the dataset is shown.	2.3.G 2.3.I 2.2.d 2.3.B 2.3.C 2.3.H 2.3.J
[4][l] [9][a,b,c, d,e]	Use the menu to navigate through the visualisation of Bubble Sort.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted	This ensures that the menu functions correctly for a Bubble Sort visualisation.	Timing: 00.02.15 - 00.02.37 From 00.02.20 - 00.02.28, the right arrow key is being pressed. From 00.02.29 - 00.02.35, the left arrow key is being pressed. This is all as expected except the Pause button glitches a bit when pressed as the visualisation may begin again but the button still says ‘Play’.	2.3.H 2.3.I 2.3.b 2.3.h 2.3.l 2.3.c 2.3.I 2.3.J

		dataset. Pressing the arrow keys changes the visualisation to the previous or next step.			
[3][b] [5][a,b,c, d,e,j]	Open the Prim's Algorithm Visualisaiton page.	The Prim's Algorithm Visualisation page will open, with fields to enter a graph and a start vertex.	This ensures that the Prim's Algorithm page can be accessed successfully, therefore enabling the visualisation of the algorithm.	Timing: 00.02.42 - 00.02.44 Once the graph has been input, the timings for the visualisation are: 00.04.19 - 00.04.32 This is as expected.	2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.E 2.1.G 2.1.H 2.1.I 3.2.b 3.2.n 3.2.s 3.2.Q 3.2.U
[5][f,g,h, i] [10][a,b, c,d,e]	Input a graph and visualise Prim's algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the visualisation of finding the MST is executed on the graph and in text.	This ensures that the visualisation functionality and the validation for Prim's algorithm execute properly.	Timing: 00.02.45 - 00.04.19 At 00.03.21, the delete key is pressed after clicking on vertex 'B'. However, vertex 'A' was deleted. This is not as expected. The edge created at 00.04.07 is not displayed very clearly on the NetworkX graph on the right side of the screen because the weight is covered up. Otherwise, everything is as expected.	3.2.b 3.2.u 3.2.v 3.2.C 3.2.D 3.2.W 3.2.F 3.2.G 3.1.l 3.1.m 3.1.n 3.1.h 3.1.c 3.1.i 3.1.j 3.1.p 3.1.J
[5][k] [9][a,b,c, d,e]	Use the menu to navigate through the visualisation of Prim's	Pressing Pause stops the visualisation at the current state.	This ensures that the menu functions correctly for a Prim's	Timing: 00.04.34 - 00.04.45 For pressing the left arrow key the timings are: 00.04.40 - 00.04.41. For pressing the right arrow key,	2.3.H 2.3.I 2.3.b 2.3.h 2.3.l 2.3.c

	algorithm.	Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the arrow keys changes the visualisation to the previous or next step.	algorithm visualisation.	the timings are: 00.04.42 - 00.04.43. This is all as expected.	2.3.I 2.3.J
[3][b] [6][a,b,c,d,e,f,k]	Open the Dijkstra's Algorithm Visualisation page.	The Dijkstra's Algorithm Visualisation page will open, with fields to enter a graph, start and end vertices and a blank vertex table.	This ensures that the Dijkstra's Algorithm page can be accessed successfully, therefore enabling the visualisation of the algorithm.	Timing: 00.04.49 - 00.04.53 Once the graph has been input, the page and the visualisation can be seen during timings: 00.06.25 - 00.06.49 This is all as expected.	2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.E 2.1.G 2.1.H 2.1.I 3.3.K 3.3.f 3.3.L 3.3.i 3.3.k 3.3.l 3.3.e 3.3.m 3.3.d
[6][g,h,i,j] [10][a,b,	Input a graph and visualise Dijkstra's algorithm.	An invalid dataset input will result in an error message being	This ensures that the visualisation functionality and the	Timing: 00.04.54 - 00.06.24 Trying to submit a graph of less than 4 vertices is shown at 00.05.24 and an unconnected graph is	3.3.e 3.3.f 3.3.K 3.3.m 3.3.C 3.3.D

c,d,e]		output. Otherwise, the visualisation of finding the shortest path between the start and end vertices is executed on the graph and in text.	validation for Dijkstra's algorithm execute properly.	shown at 00.05.46. The system responds well to both. Unlike in the Prim's graph input, when vertex ' ' is clicked and delete is pressed, it is correctly deleted. This can be seen at 00.05.14. All as expected.	3.3.E 3.3.M 3.3.N 3.3.q 3.1.l 3.1.m 3.1.n 3.1.h 3.1.c 3.1.i 3.1.j 3.1.p 3.1.J
[6][l] [9][a,b,c, d,e]	Use the menu to navigate through the visualisation of Dijkstra's algorithm.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the arrow keys changes the visualisation to the previous or next step.	This ensures that the menu functions correctly for a Dijkstra's Algorithm visualisation.	Timing: 00.06.51 - 00.07.07 Using the left arrow key is shown from 00.07.00 - 00.07.03 and using the right arrow key is shown from 00.07.03 - 00.07.06. This is all as expected.	2.3.H 2.3.I 2.3.b 2.3.h 2.3.l 2.3.c 2.3.I 2.3.J
[3][b] [7][a,b,c, d,e,j]	Open the Simplex Algorithm Visualisation page.	The Simplex Algorithm Visualisation page will open, with fields to enter coefficients	This ensures that the Simplex Algorithm page can be accessed successfully, therefore enabling	Timing: 00.07.11 - These timings are until the visualisation has finished. All is as expected.	2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.q 2.1.s 2.1.t 2.1.z 2.1.A

		for constraints and an objective function.	the visualisation of the algorithm.		2.1.B 2.1.E 2.1.G 2.1.H 2.1.I 4.1.e 4.1.f 4.1.g 4.1.a
[7][f,g,h,i]	Input coefficients for the constraints and visualise the Simplex algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the visualisation of finding the optimal value for the constraints and objective function entered is executed on the axes.	This ensures that the visualisation functionality and the validation for the Simplex algorithm execute properly.	Timing: 00.07.12 - 00.08.10 How the system responds to inputting invalid values is shown at 00.07.40	4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G
[7][k] [9][a,b,c,d,e]	Use the menu to navigate through the visualisation of the Simplex Algorithm.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the	This ensures that the menu functions correctly for a Simplex Algorithm visualisation.	Timing: 00.08.12 - 00.08.26 The video doesn't show the changes very clearly however the highlighted red line does move between points in the feasible region correctly.	2.3.H 2.3.I 2.3.b 2.3.h 2.3.l 2.3.c 2.3.I 2.3.J

		arrow keys changes the visualisation to the previous or next step.			
[8][a,b,m]	Open the Quiz page.	The quiz page will open, with a title and buttons for each of the algorithm types chosen by the user when registering.	This ensures that the user can correctly access the Quiz page so that they can test themselves as part of learning.	Timing: 00.08.29 - 00.08.31	4.3.o 4.3.p 4.2.c
[8][c,d,e]	Complete a quiz question for the Bubble Sort algorithm.	An unordered dataset is displayed as text and as options in dropdown menus. There is also a field to enter the number of passes. When the user submits an answer the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	This ensures that the user can complete a question for the Bubble Sort algorithm type, therefore allowing users to practice this algorithm.	Timing: 00.08.32 - 00.09.39 For correctly answering, the timings are: 00.08.32 - 00.09.17 For incorrectly answering, the timings are: 00.09.18 - 00.09.39 Everything is as expected.	4.2.j 4.2.h 4.2.i 4.2.q
[8][f,g]	Complete a quiz question for Prim's Algorithm.	A graph is randomised and displayed. Next to it	This ensures that the user can complete a question for the	Timing: 00.09.40 - 00.10.39 For correctly answering, the timings are: 00.09.44 - 00.10.27	4.3.a 4.3.b 4.3.c 4.3.d

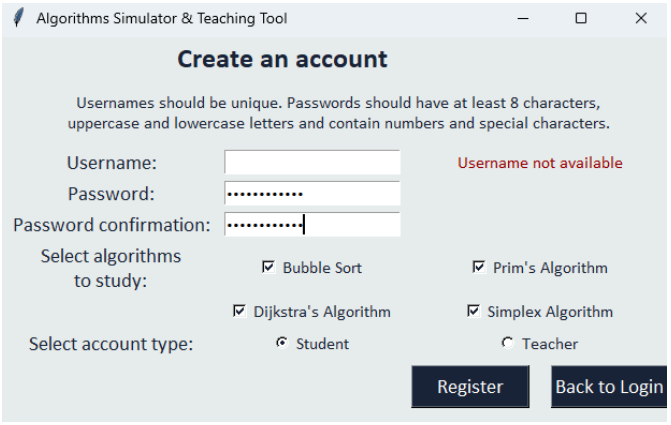
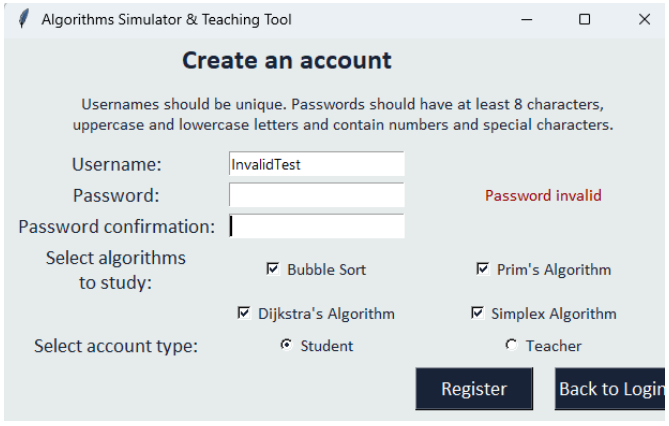
		are text boxes to enter the MST edges and MST weight into. When the user submits their answers, the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	Prim's algorithm type, therefore allowing users to practice this algorithm.	For incorrectly answering, the timings are: 00.10.28 - 00.10.38 Everything is as expected.	
[8][h,i]	Complete a quiz question for Dijkstra's Algorithm.	A graph is randomised and displayed. Next to it are text boxes to enter the shortest path and the shortest path's weight into. When the user submits their answers, the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	This ensures that the user can complete a question for the Dijkstra's algorithm type, therefore allowing users to practice this algorithm.	Timing: 00.10.40 - 00.11.14 For correctly answering, the timings are: 00.10.41 - 00.10.59 For incorrectly answering, the timings are: 00.11.00 - 00.11.13 Everything is as expected.	4.3.a 4.3.b 4.3.c 4.3.d
[8][j,k]	Complete a quiz question for the	3 constraints and an objective function	This ensures that the user can complete a	Timing: 00.11.15 For correctly answering, the timings are: 00.11.18 -	4.2.j 4.2.h 4.2.i 4.2.q

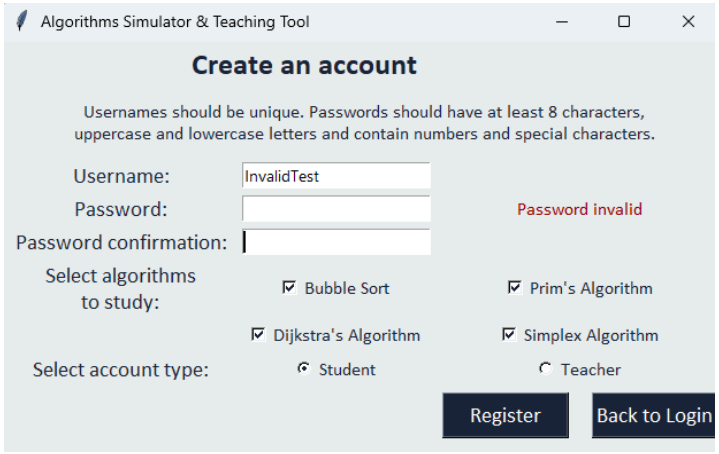
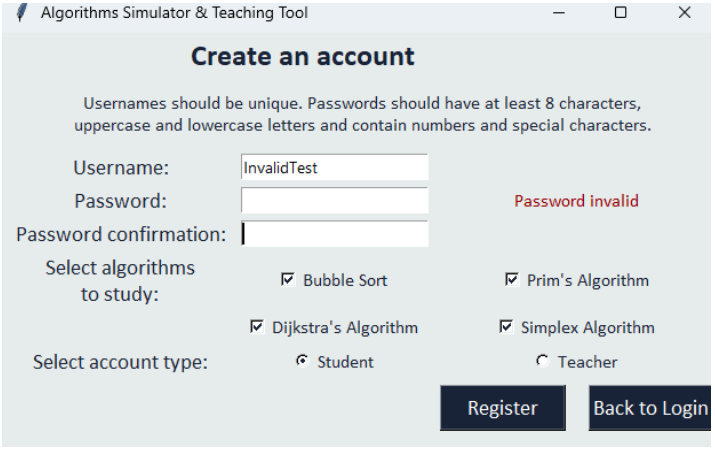
	Simplex Algorithm.	are randomised and the constraints are displayed on the graph. There are text boxes to enter the optimal x y and P values into. When the user submits their answers, the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	question for the Simplex algorithm type, therefore allowing users to practice this algorithm.	00.11.34 For inputting invalid answers, the timings are: 00.11.35 - 00.11.48 For incorrectly answering, the timings are: 00.11.50 - 00.11.56 All is as expected.	4.2.y 4.2.v

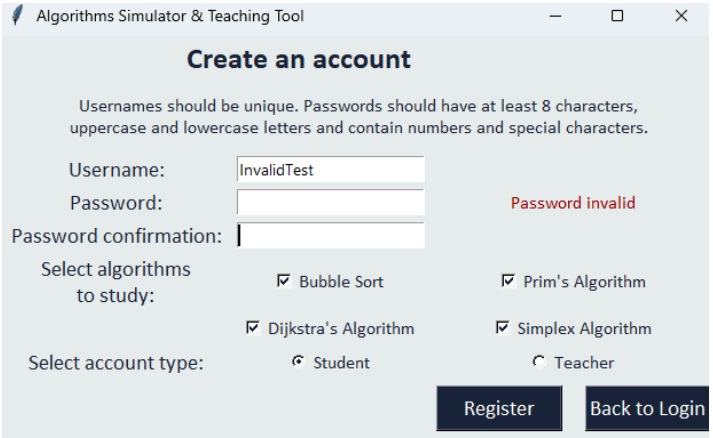
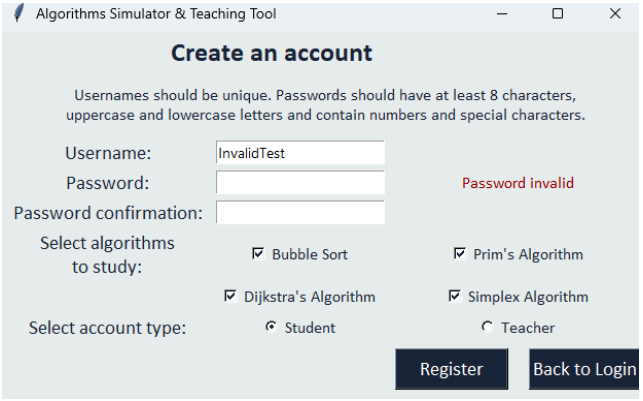
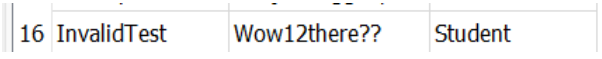
ROBUSTNESS

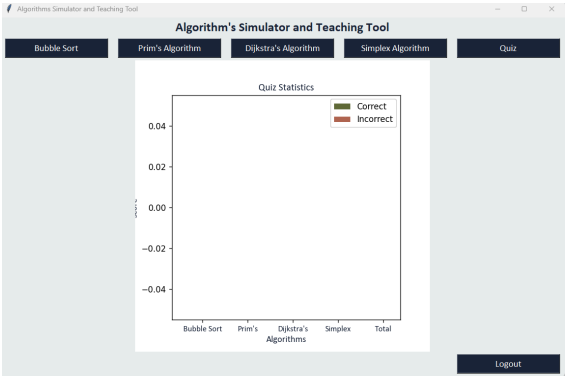
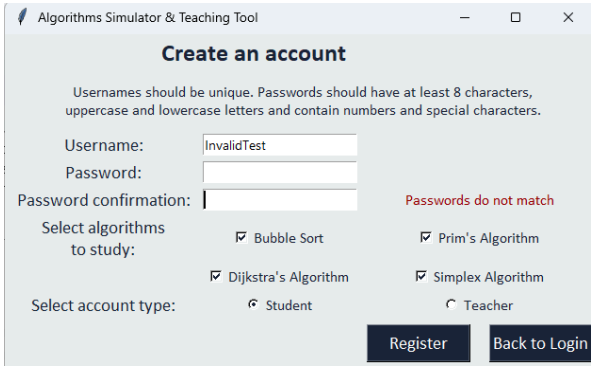
Robustness testing is concerned with how the code responds to erroneous, boundary and valid inputs. It focuses on checking how the validation in my code works in order to make sure that the system does not crash or loop indefinitely. For this, I will test the specific parts of the system that require user inputs to see how they respond to the different inputs:

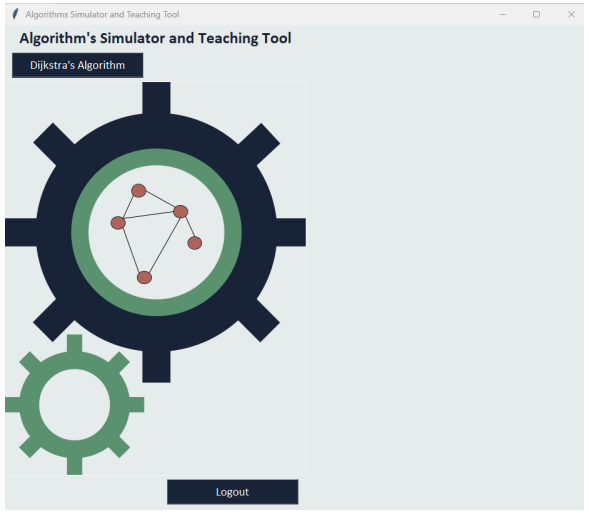
SECTION	INPUT	EXPECTED RESULT	JUSTIFICATION	REAL RESULT	CODE

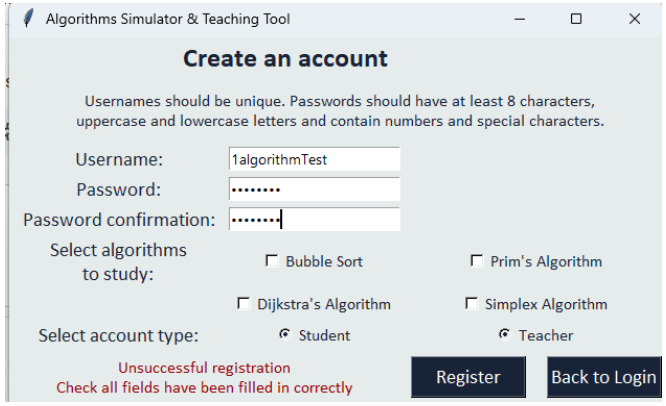
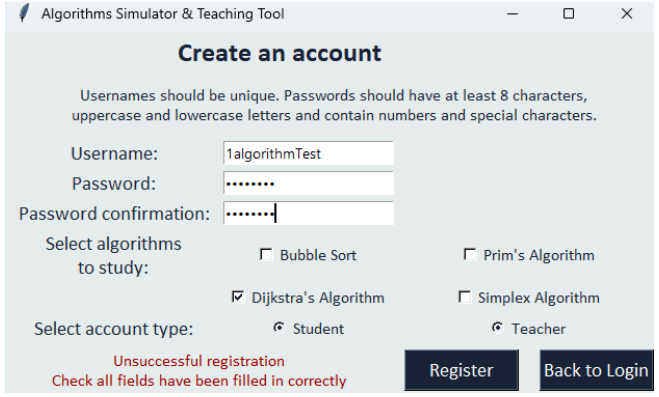
Register	Test_accS	'Username not available'	This checks that the usernames entered must be unique. The username Test_accS has already been used so this is an erroneous data test.	 As expected	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q … [refined gui]
Register	Wow12&h	'Password invalid'	This checks that the usernames entered must be greater than 8 characters long so this is a boundary test.	 As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q

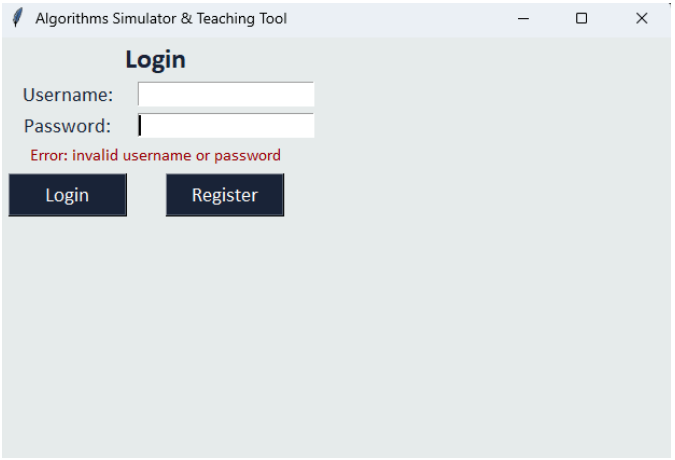
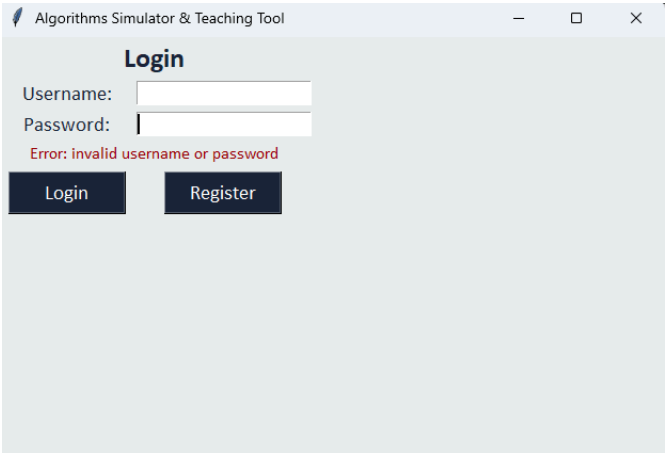
Register	WOW12&H?	'Password invalid'	This checks that there must be both uppercase and lowercase characters in the password so this is an erroneous data test.	 As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q
Register	wow12&h?	'Password invalid'	This checks that there must be both uppercase and lowercase characters in the password so this is an erroneous data test.	 As expected	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q

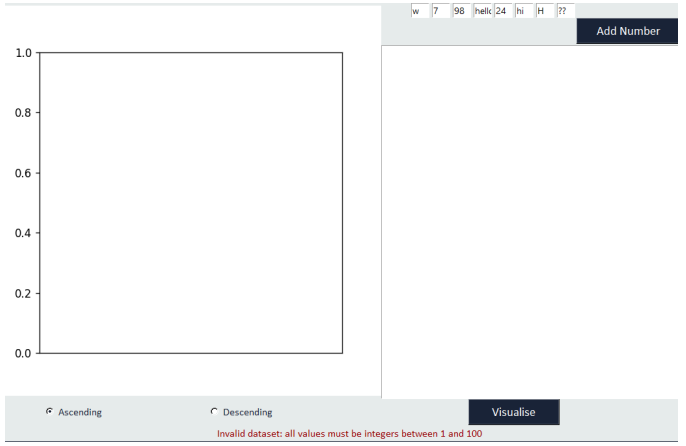
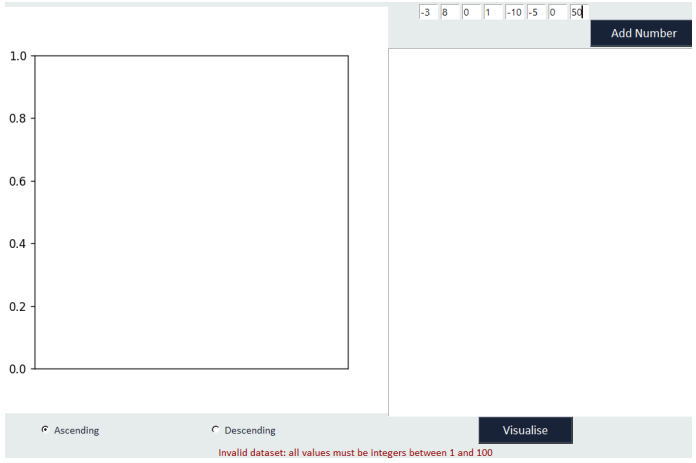
Register	wow&there	'Password invalid'	This checks that there must be numbers in the password so this is an erroneous data test.	 As expected	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q
Register	wow12there	'Password invalid'	This checks that there must be special characters in the password so this is an erroneous data test.	 As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q
Register	Wow12there??	Registration window closed and	This checks that all criteria have been		1.1.c 1.1.e

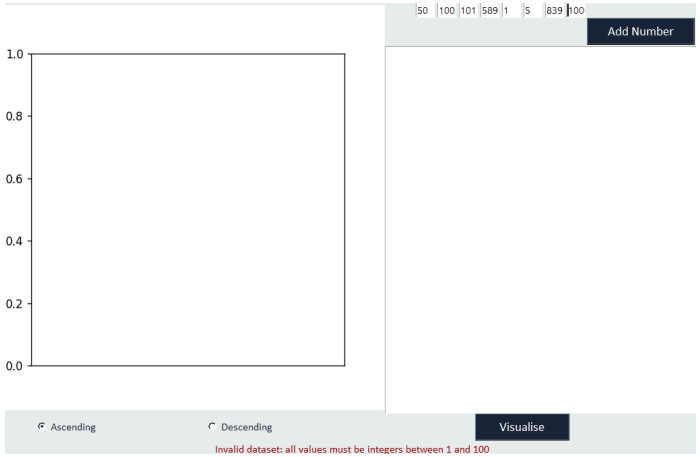
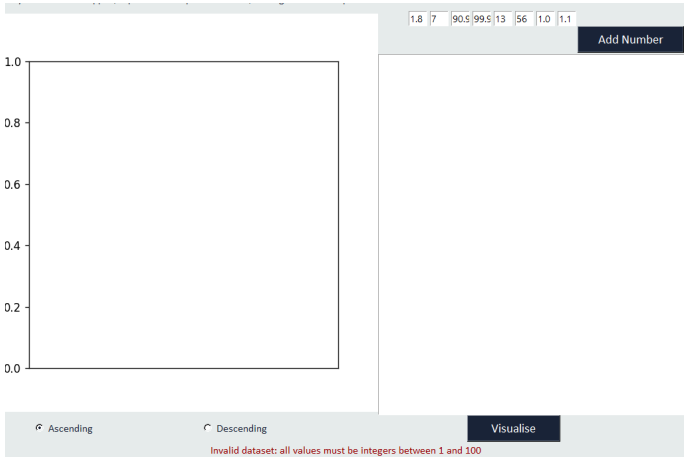
		the Home page is opened. The account is created in the database.	met so this is a normal data test.	 24 InvalidTest Bubble Sort 0 0 25 InvalidTest Prim's 0 0 26 InvalidTest Dijkstra's 0 0 27 InvalidTest Simplex 0 0 As expected.	1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q 1.2.g
Register	Password: Wow12there?? Password confirmation: wow12	The password fields are cleared and a message says 'Passwords don't match'	This checks that the password fields match correctly. So, this is an erroneous data test.	 As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q

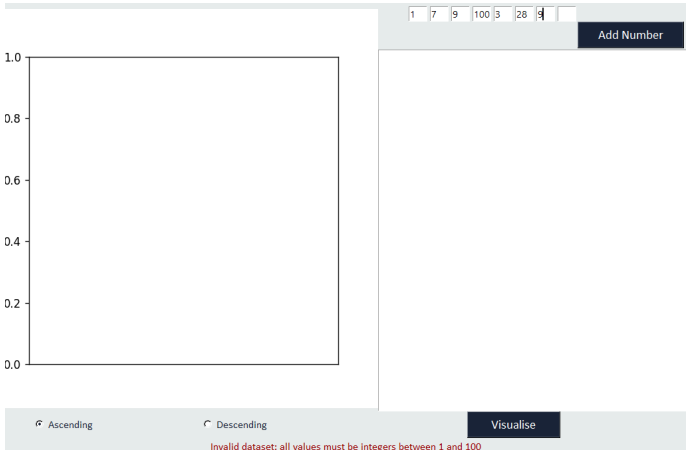
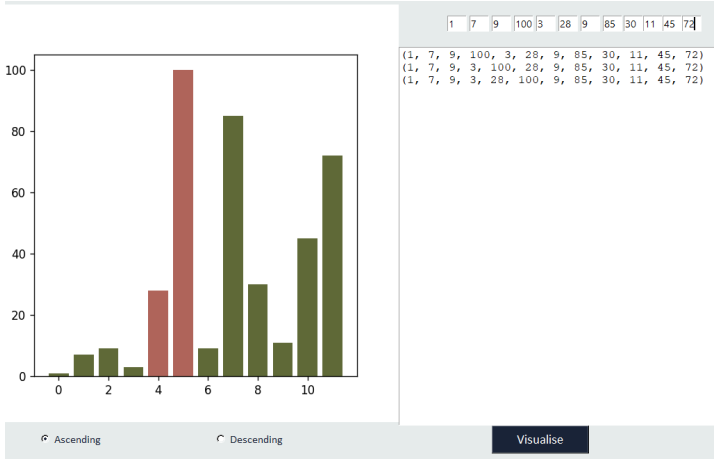
Register	Dijkstra's	An account is created.	This checks that at least 1 algorithm has been chosen so this is a normal boundary test.	 This is the same account I used for the no algorithm and no account type tests below: 17 1algorithmTest AyoHit%1 Teacher 17 17 1algorithmTest Dijkstra's As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q
Register	Bubble Sort Prim's Dijkstra's Simplex	An account is created	This is a normal data test for if all algorithms are chosen by the user.	This is for the InvalidTest account that I created: 24 24 InvalidTest Bubble Sort 0 0 25 25 InvalidTest Prim's 0 0 26 26 InvalidTest Dijkstra's 0 0 27 27 InvalidTest Simplex 0 0 16 InvalidTest Wow12there?? Student As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p

					1.2.q
Register	No algorithms selected	'Please ensure all fields have been filled out and at least 1 algorithm has been selected.'	This is a boundary test for if no algorithm are chosen, therefore ensuring that the user must select at least 1 algorithm	 As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q
Register	No account type chosen	'Please ensure all fields have been filled out and at least 1 algorithm has been selected.'	This is a boundary test for if no account type has been chosen.	 As expected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q

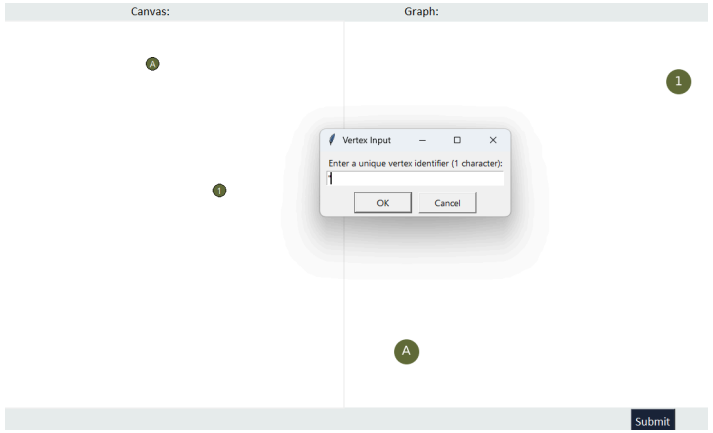
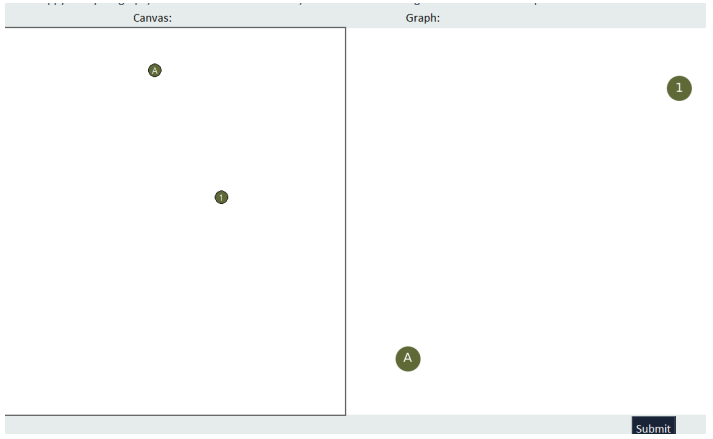
Login	Username: StudentTest Password: Testing1234>	'Incorrect username' and the username and password fields are cleared.	This is an erroneous data test for if an incorrect username is entered.	 As expected.	1.1.c 1.1.d 1.2.a 1.2.d 1.2.i
Login	Username: FullTestT Password: Ring34%	'Incorrect password' and the username and password fields are cleared.	This is an erroneous data test for if an incorrect password for a correct username is entered.	 As expected.	1.1.c 1.1.d 1.2.a 1.2.d 1.2.i

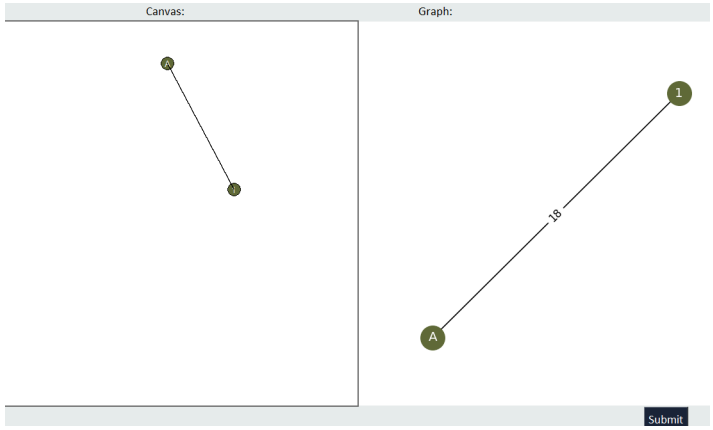
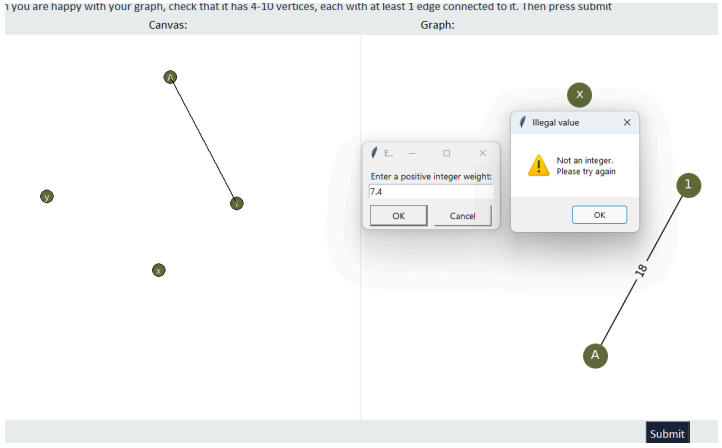
Bubble Sort	w , 7 , 98 , hello , 24 , hi , H , ??	‘Invalid dataset: all values must be integers between 1 and 100’	This is an erroneous data test for if letters or other characters are input.	 As expected.	2.2.a 2.2.b 2.2.d
Bubble Sort	-3 , 8 , 0 , 1 , -10 , -5 , 0 , 50	‘Invalid dataset: all values must be integers between 1 and 100’	This is a boundary test for whether all the numbers are greater than or equal to 1.	 As expected.	2.2.a 2.2.b 2.2.d

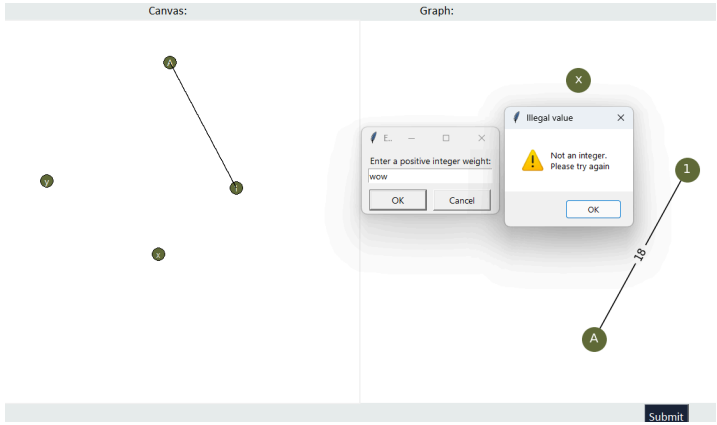
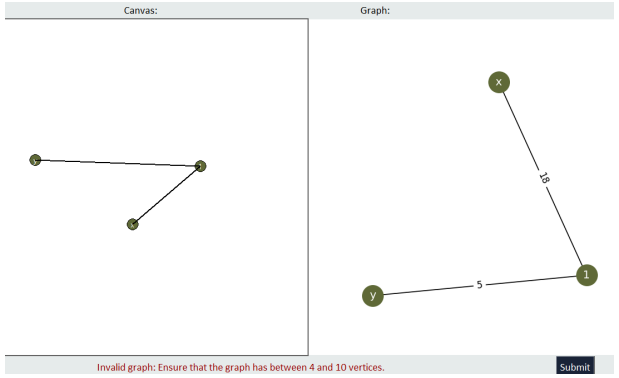
Bubble Sort	50 , 100 , 101 , 589 , 1 , 5 , 839 , 1000	'Invalid dataset: all values must be integers between 1 and 100'	Enter numbers greater than 100 - boundary This is a boundary test for whether all the numbers are less than 100	 As expected.	2.2.a 2.2.b 2.2.d
Bubble Sort	1.8 , 7 , 90.9 , 99.9 , 13 , 56 , 1.0 , 1.1	'Invalid dataset: all values must be integers between 1 and 100'	Enter decimals This is an erroneous data test for whether all the input numbers are integers.	 As expected.	2.2.a 2.2.b 2.2.d

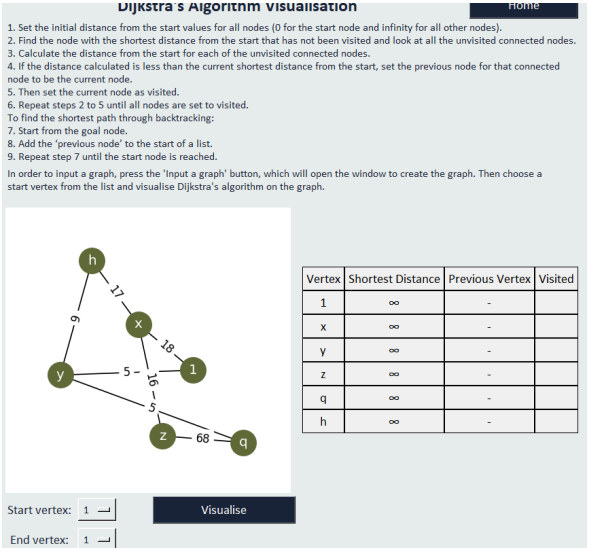
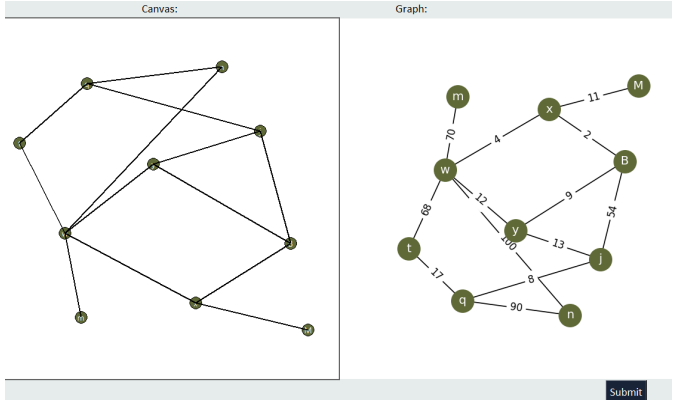
Bubble Sort	1 , 7 , 9 , 100 , 3 , 28 , 9	'Invalid dataset: the dataset must consist of between 8 and 12 numbers (inclusive)'	This is a boundary test for the number of integers that must be entered by the user.	 As expected.	2.2.a 2.2.b 2.2.d
Bubble Sort	1 , 7 , 9 , 100 , 3 , 28 , 9 , 85 , 30 , 11 , 45 , 72	The Bubble Sort visualisation on the dataset begins correctly.	This is a normal boundary test for the number of integers that must be entered by the user.	 As expected.	2.2.a 2.2.b 2.2.d 2.2.g 2.2.h

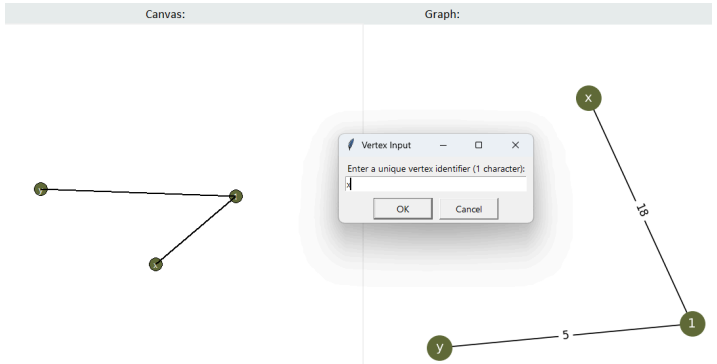
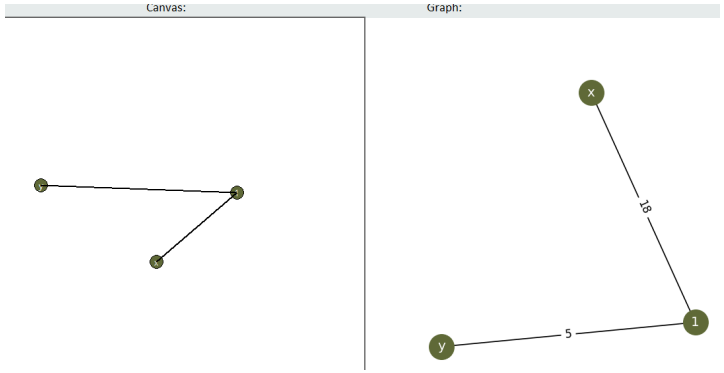
Graph input	A	The vertex is created and displayed on the graph and canvas.	This is a normal data test for inputting a character for a vertex id	 As expected.	3.1.h 3.1.l 3.1.c 3.1.i 3.1.j
Graph input	1	The vertex is created and displayed on the graph and canvas.	This is a normal data test for inputting a character for a vertex id	 As expected.	3.1.h 3.1.l 3.1.c 3.1.i 3.1.j

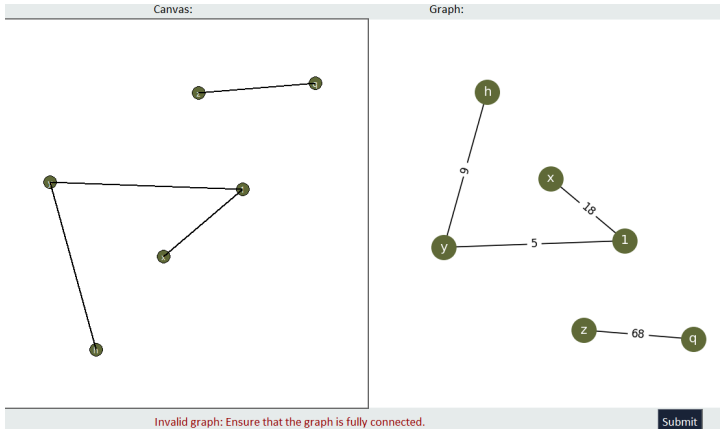
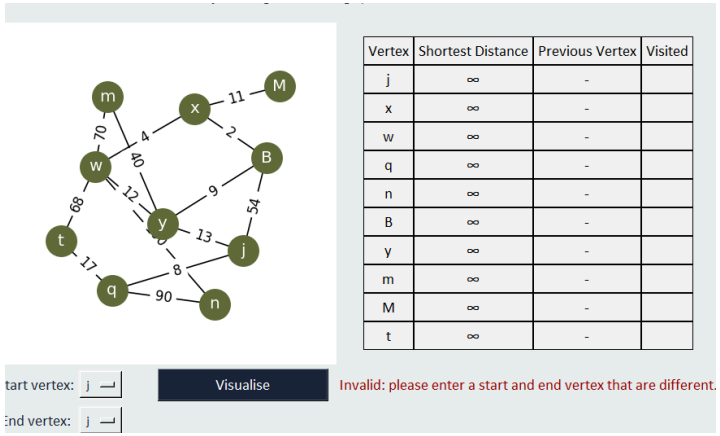
Graph input	“	The vertex is not created and is not displayed on the graph.	This is an erroneous data test for inputting a vertex id.	  As expected.	3.1.h 3.1.l 3.1.c

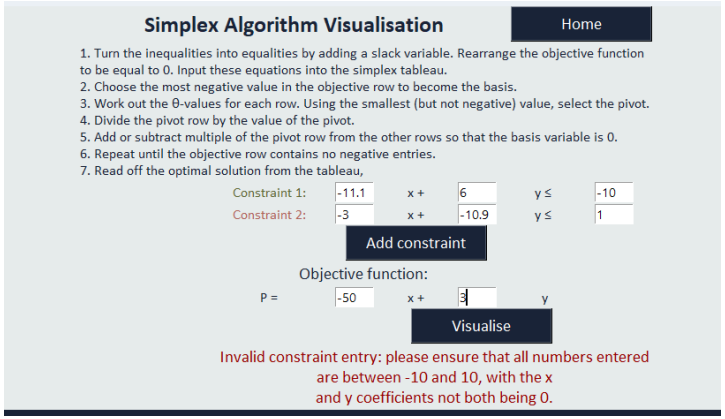
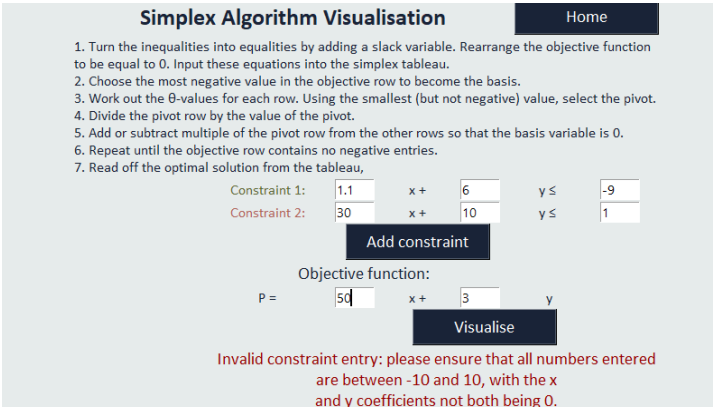
Graph input	18	The edge is created and displayed on the graph.	This is a normal data test for the weight of an edge for if an integer is input.	 As expected.	3.1.h 3.1.l 3.1.d 3.1.i 3.1.j
Graph input	7.4	The edge is created and displayed on the graph.	This is a normal data test for the weight of an edge for if a decimal/float is input.	 As expected.	3.1.h 3.1.l 3.1.d

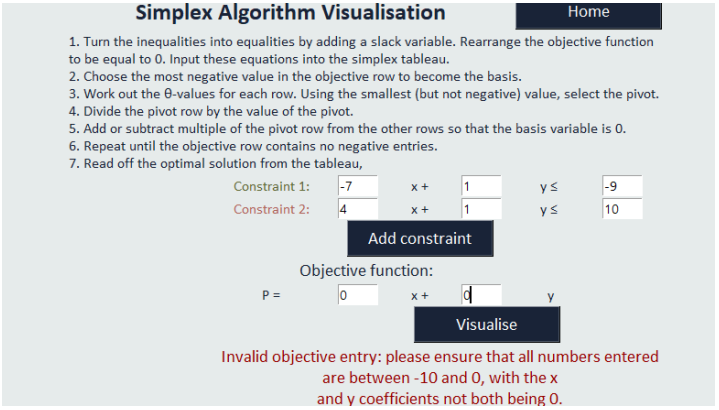
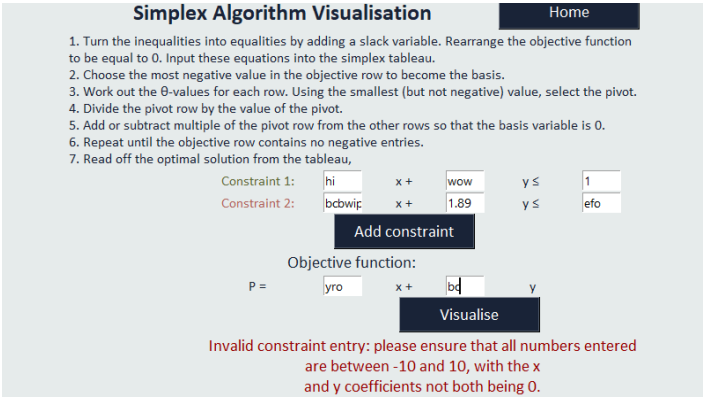
Graph input	wow	The ‘Not and integer’ error pops up.	This is an erroneous data test for inputting the edge weight.	 As expected.	3.1.h 3.1.l 3.1.d
Graph input	Press submit with 3 connected vertices	The ‘Invalid: Ensure that the graph has between 4 and 10 vertices.’ message is displayed.	This is a boundary test for the number of vertices that the graph must have.	 As expected.	3.1.p 3.1.l

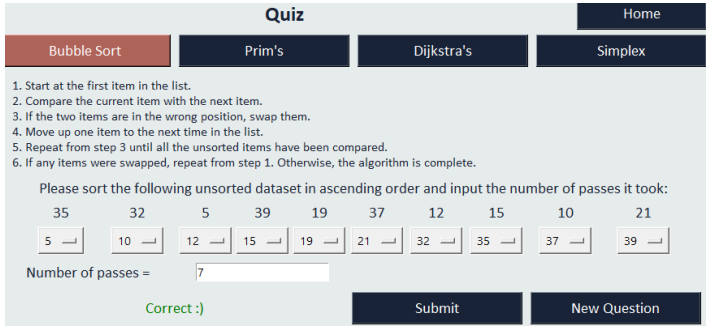
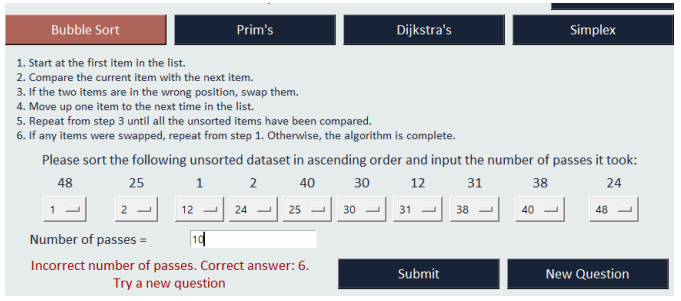
Graph input	Press submit with 6 connected vertices	The graph is created in the Prim's or Dijkstra's page.	This is a normal boundary test for if a valid number of vertices is input.	 As expected.	3.3.n 3.3.o 3.3.l 3.3.x 3.3.C 3.3.D 3.3.E 3.3.K 3.3.L 3.3.M 3.3.N
Graph input	Try create an 11th vertex	The vertex does not appear on the canvas and graph.	This is a boundary test, ensuring that more than 10 vertices can't be input.	 When double clicking, the popup to enter the vertex id does	3.1.c 3.1.l

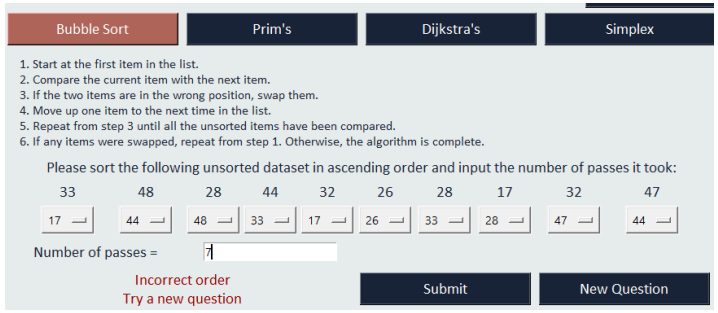
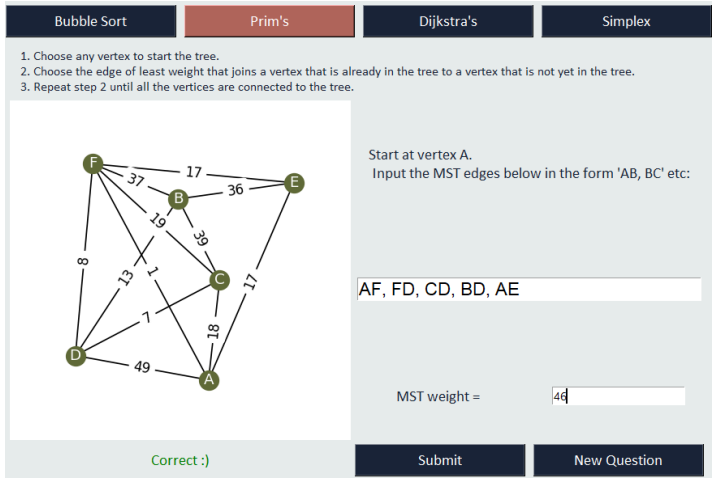
				not appear. This is as expected.	
Graph input	Try inputting a vertex id that has already been used.	The vertex does not appear on the canvas and graph.	This is an erroneous data test to ensure that all vertices have unique ids.	  As expected.	3.1.h 3.1.l 3.1.c

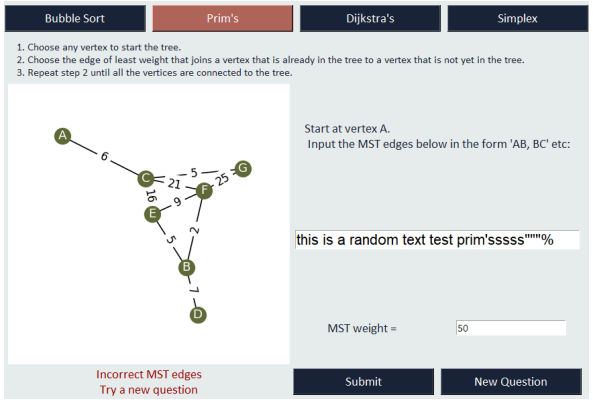
Graph input	Try submitting a graph of 6 vertices that is not fully connected.	The 'Not fully connected' message is displayed.	This is a test to ensure that the input graph is fully connected as Prim's and Dijkstra's algorithms can't function otherwise.	 As expected.	3.3.J
Dijkstra's	Enter the same start and end vertices	The 'Invalid: start and end vertices must be different' message is displayed.	This is an erroneous data test to ensure that the vertices input are different, therefore allowing Dijkstra's algorithm to execute.	 As expected.	3.3.m

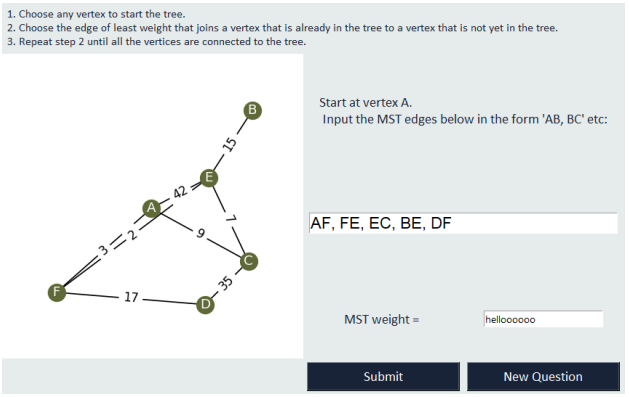
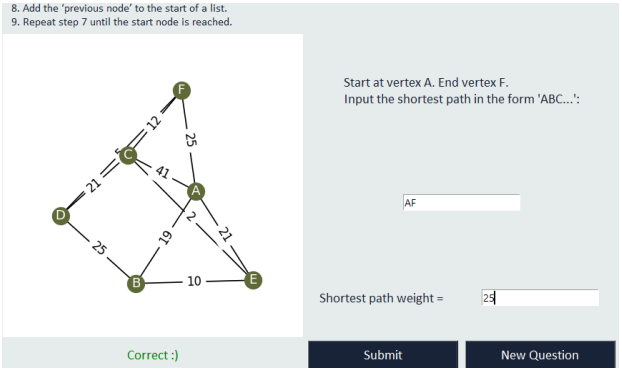
Simplex	Enter numbers less than -10 for coefficients	The 'Invalid: please ensure all numbers are between -10 and 10' message is displayed.	This is an erroneous data test to ensure that all coefficients are greater than -10.	 As expected.	4.1.e 4.1.g
Simplex	Enter numbers greater than 10 for coefficients	The 'Invalid: please ensure all numbers are between -10 and 10' message is displayed.	This is an erroneous data test to ensure that all coefficients are less than 10.	 As expected.	4.1.e 4.1.g

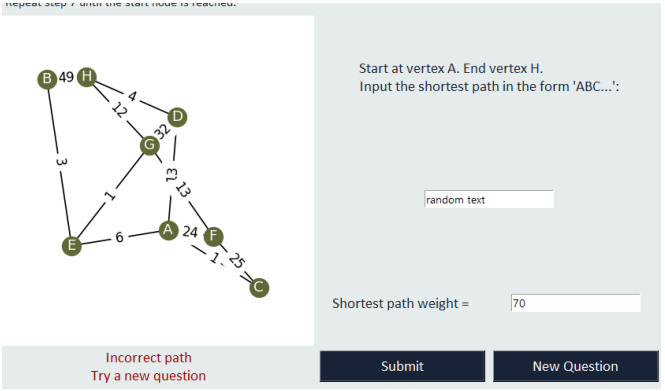
Simplex	Enter 0 for x and y	The 'Invalid: please ensure x and y coefficients are not both 0' message is displayed.	This is an erroneous data test to ensure that the coefficients are real lines.	 As expected.	4.1.e 4.1.g
Simplex	Enter text for coefficients	The 'Invalid: please ensure x and y coefficients are not both 0' message is displayed.	This is an erroneous data test to ensure that numbers are input by the user.	 As expected.	4.1.e 4.1.g

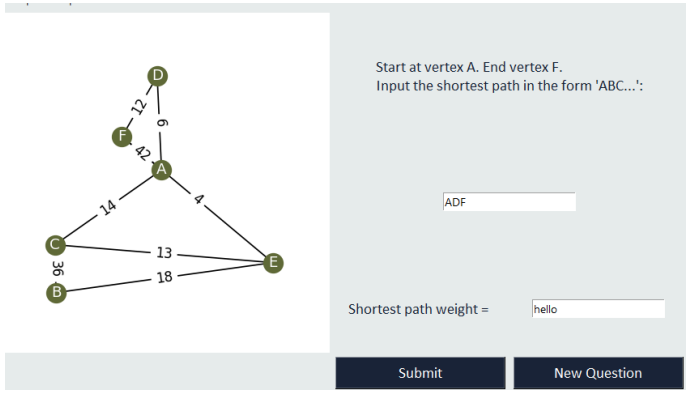
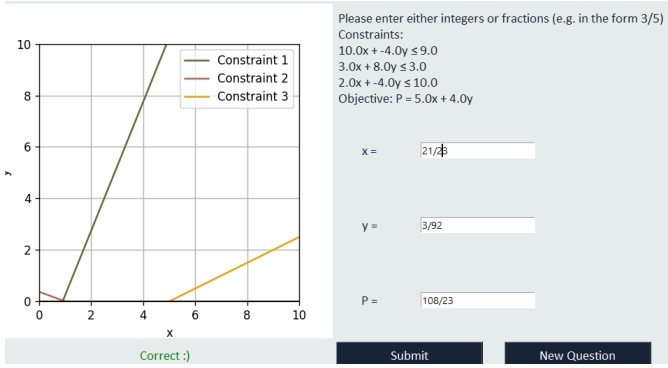
Quiz	Enter the correct order and weight for a Bubble Sort question.	'Correct' is output and the database is updated.	This is a normal data test to ensure that answers can be validated correctly.	 The correct score was 6 before this. 4 HelloWord!!! Bubble Sort 7 11 5 HelloWord!!! Prim's 8 9 6 HelloWord!!! Dijkstra's 7 5 7 HelloWord!!! Simplex 2 7 As expected.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q 4.2.l
Quiz	Enter correct order and text for the number of passes for a Bubble Sort question.	'Incorrect number of passes' is output and the database is updated.	This is an erroneous data test to ensure that the number of passes is validated correctly.	 4 HelloWord!!! Bubble Sort 7 12 5 HelloWord!!! Prim's 8 9 6 HelloWord!!! Dijkstra's 7 5 7 HelloWord!!! Simplex 2 7 As expected.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q 4.2.l

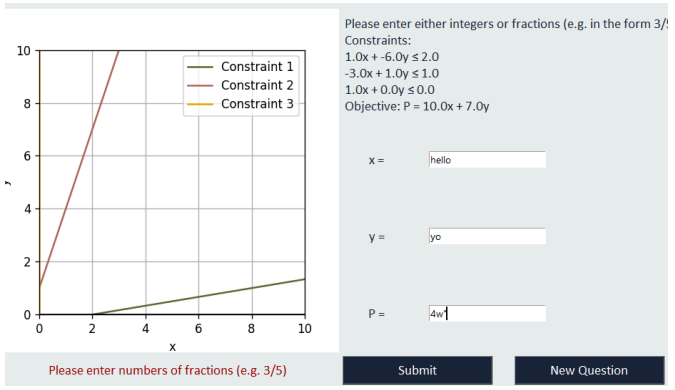
Quiz	Enter an incorrect order and correct weight for a Bubble Sort question.	'Incorrect order' is output and the database is updated.	This is an erroneous data test to ensure that the order is validated correctly.	 4 HelloWord!!! Bubble Sort 7 13 5 HelloWord!!! Prim's 8 9 6 HelloWord!!! Dijkstra's 7 5 7 HelloWord!!! Simplex 2 7 As expected.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q 4.2.l
Quiz	Enter the correct MST edges and weight for a Prim's algorithm question.	'Correct' is output and the database is updated.	This is a normal data test to ensure that the answers can be validated correctly.		4.3.b 4.3.c 4.3.f 4.3.e 4.3.n

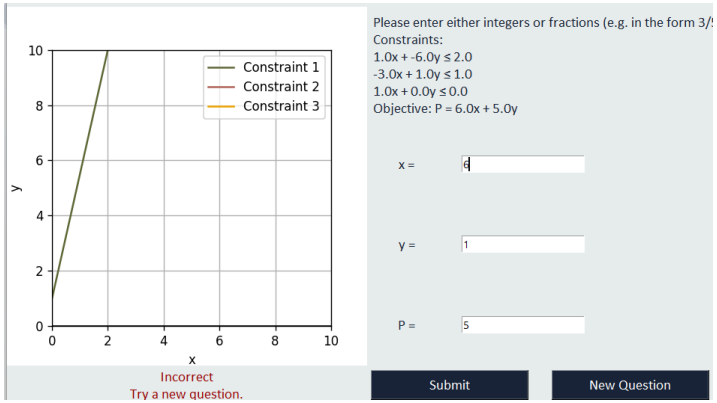
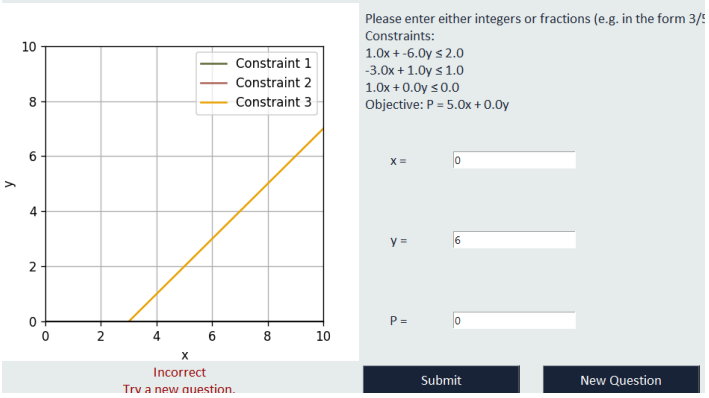
				<table><tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>7</td><td>13</td></tr><tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>9</td><td>9</td></tr><tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>7</td><td>5</td></tr><tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>2</td><td>7</td></tr></table> As expected.	
Quiz	Enter random text for MST edges for a Prim's algorithm question	'Incorrect MST edges' is output and the database is updated.	This is an erroneous data test to ensure that the MST edges are validated correctly.	 <table><tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>7</td><td>13</td></tr><tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>9</td><td>10</td></tr><tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>7</td><td>5</td></tr><tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>2</td><td>7</td></tr></table> As expected.	4.3.b 4.3.c 4.3.f 4.3.e 4.3.n

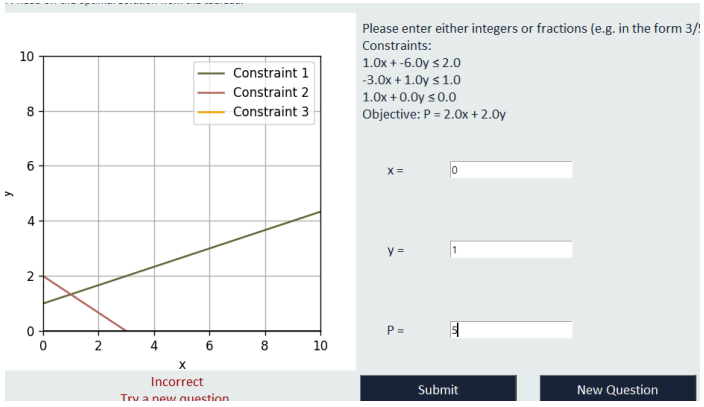
Quiz	Enter text for the weight for a Prim's algorithm question	'Incorrect weight' is output and the database is updated.	This is an erroneous data test to ensure that the weight is validated correctly.	 When submit is pressed, nothing is output on the interface. The database remains the same. In the Shell, the following is output: `_tkinter.TclError: expected floating-point number but got "hellooooooo"` Could be improved by giving a message to input an integer for the weight.	4.3.b 4.3.c 4.3.f 4.3.e 4.3.n
Quiz	Enter correct shortest path and weight for a Dijkstra's question	'Correct' is output and the database is updated.	This is a normal data test to ensure that the answers can be validated correctly.		4.3.k 4.3.l 4.3.n

				<table><tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>7</td><td>13</td></tr><tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>9</td><td>10</td></tr><tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>8</td><td>5</td></tr><tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>2</td><td>7</td></tr></table> As expected.	
Quiz	Enter random text for the shortest path for a Dijkstra's question	'Incorrect path' is output and the database is updated.	This is an erroneous data test to ensure that the shortest path is validated correctly.	 <table><tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>7</td><td>13</td></tr><tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>9</td><td>10</td></tr><tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>8</td><td>6</td></tr><tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>2</td><td>7</td></tr></table> As expected.	4.3.k 4.3.l 4.3.n

Quiz	Enter random text for the weight for a Dijkstra's question	'Incorrect weight' is output and the database is updated.	This is an erroneous data test to ensure that the weight is validated correctly.	 When submit is pressed, nothing is output on the interface. The database remains the same. In the Shell, the following is output: `_tkinter.TclError: expected floating-point number but got "hello"` Could be improved by giving a message to input an integer for the weight.	4.3.k 4.3.l 4.3.n
Quiz	Enter correct x y and P values for a Simplex question	'Correct' is output and the database is updated.	This is a normal data test to ensure that the weight is validated correctly.	 An incorrect answer for a Simplex question had been input	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C

				before but the new state of the database is: 4 HelloWord!!! Bubble Sort 7 13 5 HelloWord!!! Prim's 9 10 6 HelloWord!!! Dijkstra's 8 6 7 HelloWord!!! Simplex 3 8 As expected.	
Quiz	x = hello, y = yo, P = 4w" for a Simplex question	'Incorrect' is output and the database is updated.	This is an erroneous data test to ensure that the weight is validated correctly.	 As expected.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C

Quiz	x is incorrect and y and P are correct for a Simplex question	'Incorrect' is output and the database is updated.	This is an erroneous data test to ensure that the x value is validated correctly.	 4 HelloWord!!! Bubble Sort 7 13 5 HelloWord!!! Prim's 9 10 6 HelloWord!!! Dijkstra's 8 6 7 HelloWord!!! Simplex 3 9 As expected.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C
Quiz	y is incorrect and x and P are correct for a Simplex question	'Incorrect' is output and the database is updated.	This is an erroneous data test to ensure that the y value is validated correctly.		4.2.j 4.2.h 4.2.i 4.2.q 4.2.C

				<table><tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>7</td><td>13</td></tr><tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>9</td><td>10</td></tr><tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>8</td><td>6</td></tr><tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>3</td><td>10</td></tr></table> As expected.	
Quiz	P is incorrect and x and y are correct for a Simplex question	'Incorrect' is output and the database is updated.	This is an erroneous data test to ensure that the P value is validated correctly.	 <table><tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>7</td><td>13</td></tr><tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>9</td><td>10</td></tr><tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>8</td><td>6</td></tr><tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>3</td><td>11</td></tr></table> As expected.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C

USABILITY

Usability testing is concerned with how the aspects of the interface function. This involves aspects like how the system appears and how keyboard or mouse inputs cause the system to function. For this, I am going to give evidence of the usability features of this system using screenshots and video testing evidence. Additionally, I am going to give a questionnaire to two of my stakeholders about the usability of this system in order to test it.

USABILITY FEATURES

Video evidence is from the NEA final testing video and the Simplex visualisation final testing video. Unless specified otherwise, timings are given for the NEA final testing video.

SUCCESS CRITERIA	VIDEO EVIDENCE TIMING	CODE	JUSTIFICATION
[1][a,b,c,e]	Login page: 00.00.00	1.1.c 1.1.d 1.1.f 1.1.g 1.3.b 1.3.c	The title of the page and input fields for the username and password are clearly labelled so the user knows what page they are on and what they need to input. The buttons are labelled with a font colour that contrasts the background well, making it clear to read.
[2][a,b,d,e,f,g,i]	Register page 00.00.03	1.1.c 1.1.e 1.1.f 1.2.k 1.2.l 1.2.m 1.2.n 1.2.p 1.2.q 1.2.r 1.2.s 1.3.b 1.3.c 1.3.d	The title of the page and input fields are clearly labelled. Checkboxes and radiobuttons make it easy to know what they are for and how the user should enter their choices. The buttons are labelled with a font colour that contrasts the background well, making it clear to read.
[3][a,b,d,e]	Home page for a Student account: 00.01.07 Quiz statistics: 00.11.59	2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.h 2.1.l 2.1.p 2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.E 2.1.G 2.1.H 2.1.I	The title of the page is clear so the user knows what page they are on. The buttons are labelled with a font colour that contrasts the background well, making it clear to read. The bar chart has contrasting colours that make it clear how many questions of each algorithm they have answered correctly and incorrectly.
[3][a,b,e]	Home page for a Teacher account: 00.14.09	1.1.e 1.1.f 1.3.b 1.3.d 2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.q 2.1.s 2.1.t	The title of the page is clear so the user knows what page they are on. The buttons are labelled with a font colour that contrasts the background well, making it clear to read. The logo adds visual interest to the page.

		2.1.z 2.1.A 2.1.B 2.1.E 2.1.G 2.1.H 2.1.I	
[4][a,b,c,d,e,k]	Bubble Sort visualisation page: 00.01.27	2.3.G 2.3.I	The title of the page is clear so the user knows which algorithm visualisation page they are on. The buttons are labelled with a font colour that contrasts the background well, making it clear to read. The textboxes are in a row so it is clear that the user needs to input the dataset into them.
[4][f,k]	Bubble Sort visualisation: 00.02.09 - 00.02.14	2.2.d 2.3.G 2.3.I	The database is clearly displayed on the bar chart and the highlighting makes it clear which items are being compared and swapped at each stage.
[4][l] [9][a,b,c,d,e]	Bubble Sort menu: 00.02.15 - 00.02.36	2.3.b 2.3.c 2.3.h 2.3.H 2.3.I 2.3.J	The menu functions well to allow the user to control which part of the visualisation they want to look at or use. The arrow keys allow users to easily go through the visualisation at their own pace if the normal speed is too fast.
[10][a,b,e]	Graph input: 00.02.46 - 00.04.18 00.04.54 - 00.06.23	3.1.c 3.1.e 3.1.h 3.1.i 3.1.j 3.1.l 3.1.m 3.1.n	The graph input page has clear instructions for how the user can input the graph and the page has a title, which makes it easy to know what the page is for. Keyboard and mouse inputs are quite straightforward, although they glitch a bit, especially the deleting of vertices.
[5][a,b,c,e,j]	Prim's visualisation page: 00.02.44 With the graph input: 00.04.19	3.2.b 3.2.q 3.2.Q 3.2.U	This page has a clear title so the user knows which algorithm visualisation they are on. The buttons are labelled with a font colour that contrasts the background well, making it clear to read. The graph is easy to understand because of the colours. A dropdown menu is easy to navigate for choosing the start vertex.
[5][f,g,h,i]	Prim's visualisation: 00.04.26 - 00.04.33	3.2.b 3.2.u 3.2.v 3.2.C 3.2.D 3.2.F 3.2.W	The edges being added to the MST are highlighted in the same colour as the nodes and at the end are highlighted in a pink colour so they stand out, making it simple to understand what the MST for the input graph is. The text is easy to follow thanks to the clear font.
[5][k] [9][a,b,c,d,e]	Prim's menu: 00.04.34 -	2.3.b 2.3.c 2.3.d 2.3.h	The menu functions well to allow the user to control which part of the visualisation they

	00.04.45	2.3.I 2.3.H 2.3.J	want to look at or use. The arrow keys allow users to easily go through the visualisation at their own pace if the normal speed is too fast.
[6][a,b,c,e,f,k]	Dijkstra's visualisation page: 00.04.50 With the graph input: 00.06.26	3.3.d 3.3.e 3.3.f 3.3.i 3.3.k 3.3.l 3.3.m 3.3.K	This page has a clear title so the user knows which algorithm visualisation they are on. The buttons are labelled with a font colour that contrasts the background well, making it clear to read. The graph is easy to understand because of the colours. Dropdown menus make it easy to navigate choosing the start and end vertices.
[6][g,h,i,j]	Dijkstra's visualisation: 00.06.43 - 00.06.49	3.3.e 3.3.f 3.3.m 3.3.q 3.3.C 3.3.D 3.3.E 3.3.K 3.3.N 3.3.M	The vertex table is filled in clearly thanks to the tick indicating which vertices have been visited. The yellow highlighting indicates which vertices have been visited and the pink indicates the current vertex, therefore making it easy to see how the vertex table is being filled out.
[6][l] [9][a,b,c,d,e]	Dijkstra's menu: 00.06.51 - 00.07.07	2.3.b 2.3.c 2.3.d 2.3.h 2.3.I 2.3.H 2.3.J	The menu functions well to allow the user to control which part of the visualisation they want to look at or use. The arrow keys allow users to easily go through the visualisation at their own pace if the normal speed is too fast.
[7][a,b,c,d,e,f,g, j]	Simplex visualisation page: 00.07.12 When the constraints have been input: 00.08.05	4.1.a 4.1.e 4.1.f 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G	This page has a clear title so the user knows which algorithm visualisation they are on. The buttons are labelled with a font colour that contrasts the background well, making it clear to read. Depending on the values input for the constraints, the axes can either clearly or not display the constraints well. However, the highlighting makes it easier to distinguish between the plotted constraints.
[7][h,i]	Simplex visualisation: 00.08.05 - 00.08.09 In the Simplex visualisation video evidence: 00.00.36 - 00.00.39	4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G	Depending on the constraints entered, the visualisation can either be very clear or not. For the NEA final testing video, the constraints gave a very small feasible region so it was difficult to see what was going on. However, for the Simplex visualisation video evidence, the feasible region was larger so the user could see the red highlighting showing the path around the feasible region clearer.
[7][k] [9][a,b,c,d,e]	Simplex menu: 00.08.11 -	2.3.b 2.3.c 2.3.d 2.3.h	The menu functions well to allow the user to control which part of the visualisation they

	00.08.26 In the Simplex visualisation video evidence: 00.00.41 - 00.00.56	2.3.I 2.3.H 2.3.J	want to look at or use. The arrow keys allow users to easily go through the visualisation at their own pace if the normal speed is too fast.
[8][a,b,l,m]	Initial Quiz page: 00.08.29	4.2.c 4.2.h 4.2.i 4.2.j 4.2.o 4.2.q 4.2.C 4.2.J 4.3.b 4.3.e 4.3.g 4.3.k 4.3.l 4.3.m 4.3.n 4.3.o 4.3.p	This page has a clear title so the user knows that they are on the quiz page. The buttons for each of the question types are labelled with a font colour that contrasts with the background, making them clear to read. The logo adds visual interest to the page.
[8][d,e]	Bubble Sort question: 00.08.32 - 00.09.41	4.2.h 4.2.i 4.2.j 4.2.q	The dropdown boxes make it easy to ensure that the user does not input values that aren't in the dataset, although it adds extra time for the user to search through the menu to find the value they are looking for. Labels make it clear what the user should be inputting for each of the sections.
[8][f,g]	Prim's algorithm question: 00.09.43 - 00.10.39	4.3.a 4.3.b 4.3.c 4.3.d	The labels make it clear what the user needs to input for each of the textboxes. One issue is that edges may cover up other edges so the weights can't be read off, which makes some question not possible for the user to solve. However, the graph is usually easy to read off thanks to the colour coding.
[8][h,i]	Dijkstra's question: 00.10.43 - 00.11.13	4.3.a 4.3.b 4.3.c 4.3.d	The labels make it clear what the user needs to input for each of the textboxes. One issue is that edges may cover up other edges so the weights can't be read off, which makes some question not possible for the user to solve. However, the graph is usually easy to read off thanks to the colour coding.
[8][j,k]	Simplex question: 00.11.15 - 00.11.57	4.2.h 4.2.i 4.2.j 4.2.q 4.2.v 4.2.y	The labels make it clear what the user needs to input for each of the x y and P textboxes. The constraints are usually displayed clearly on the axes thanks to the highlighting but issues may occur depending on the values randomised. The ability to enter fractions makes it easier for the user, rather than having to work out the

			values as decimals, which can cause rounding errors.
General	Full system: 00.00.00 - 00.14.25	N/A	All fonts and colour schemes match, making the system seem more professional, cohesive and user friendly.

STAKEHOLDER FEEDBACK

I asked two stakeholders the following questions to test the usability of this system:

1. How did you find the registering for an account?
2. How did you find the login process?
3. What did you think of the home page? Were there features you particularly liked or disliked?
4. What did you think of the Bubble Sort Algorithm Visualisation? Was it easy to follow how the dataset changed at each point in the visualisation?
5. How did you find inputting a graph? Was it easy to create and delete nodes and edges?
6. What did you think of the Prim's Algorithm Visualisation? Was the highlighting on the graph easy to follow?
7. What did you think of the Dijkstra's Algorithm Visualisation? Was the highlighting on the graph and filling in the vertex table easy to follow?
8. What did you think of the Simplex Algorithm Visualisation? Was it clear to follow?
9. How did you find the quiz page? Were questions displayed clearly?

This is in order to obtain the stakeholders' final feedback after all iterations have been completed and each prototype has completed a Stakeholder Review.

QUESTION	STAKEHOLDER 1 FEEDBACK	STAKEHOLDER 2 FEEDBACK
1	The registration process was pretty standard and easy to complete.	It was pretty easy but I think that having the ability to view passwords would make it better.
2	Login was again easy to do.	Login was quick but I think that being able to view the password would make it better.
3	The home page was good and it was obvious what everything on the screen was for.	The home page was quite professional looking but I think that it could be a bit more visually interesting.
4	The visualisation was good because the colour changes made it really easy to understand.	Quite good and easy to follow because of the layout.
5	Inputting a graph was okay but I think it could have been made a bit easier because sometimes clicking to add edges didn't work.	It was not very easy to create and delete nodes but it was okay otherwise.

6	The Prim's visualisation was not too hard to follow. If I were to use it for revision, I would say that it saying the final answer and displaying it makes it quite useful.	It was easy to follow but I think that the screen was a bit bare.
7	From a Further Maths student's perspective, I think that it would have been more useful to have the boxes next to the vertices to fill in. Still, displaying the answer at the end is helpful.	It was not the easiest to follow when filling in the vertex table at the same time as the graph being highlighted during the visualisation, but this was easier to understand when using the menu.
8	It was clear to follow because of the contrasting colours. But I think that having the tableaus output could be more useful for me as the graph doesn't always show it in the clearest way.	Constraints sometimes didn't display well so it was harder to understand where the points were. However, it worked well otherwise.
9	The quiz section was quite useful and I think it's good for revising the algorithms.	Actually quite clear, and I liked being able to do new questions. Graphs sometimes were hard to read. I think that displaying the final answers could have been more useful.

REVIEW AND EVALUATION

Throughout development, I have tested each Prototype against Success Criteria related to the modules developed within that Prototype. Through this, I determined that all criteria were either fully or partially met. Criteria [8], [5][d] and [6][d] were not fully met due to changes made to the approach I took to coding the system during development. Additionally minor changes had to be made to the way the Dijkstra's visualisation was carried out. These changes had to be made due to limitations in the libraries that I used, specifically Tkinter and NetworkX.

All testing carried out in this section after finishing development passed successfully. Specifically, I tested the functionality, robustness and usability of the system. In functionality testing, I determined that the only functionality in the system that does not always work as expected was the deleting of vertices in the Graph Input section, which I think is due to the way Tkinter determines the position of the place the user clicked not always matching with where the vertex is. In robustness testing, I determined that the only area for improvement would be outputting a message to tell the user to input an integer for the weight. Otherwise, the system responds correctly/well to all boundary/ erroneous data input. In usability testing, I gave evidence of all the usability features in the system and gave a questionnaire to 2 of my stakeholders. The feedback given was overall positive, saying that the visualisations are clear to follow, but they also pointed out the issues already discussed with the graph input section and randomised constraints and graphs not always being very clearly displayed. An area of improvement also given was making some of the visualisations more specific to the way they should be displayed in A-Level Further Maths. When designing the system, I

determined that this would be too complicated in the given time frame but that it can be a task for post-development.

Overall, I would say that the system functions well and, although there are specific issues that can be fixed, development went successfully and the final product is a good tool for revision and teaching.

LIMITATIONS AND MAINTENANCE

Whilst I faced many difficulties in the development process, all success criteria were either fully or partially met (if a different approach was used in development), therefore making my system fully functional and user friendly, as shown through the feedback from my stakeholders. However, for this user-friendly interface, I have had to make extensive use of Python libraries like Tkinter, matplotlib and NetworkX. All of these must be installed by the user for the code to be able to run. A list of all the libraries required will be provided at the end of the FINAL CODE section. These libraries may go through many updates. Therefore, it would be better if the code were compiled into a single, executable file, which would ensure that the program would function even if the libraries were changed because the libraries and the specific modules used could be loaded into the file.

One limitation of my code is that I have not implemented any security features like encryption into the program due to the limited time that I had to develop this. In the future, I would like to be able to add functionality for hashing when storing the passwords and usernames, therefore making it a more reliable system because hashing functions are one-way functions so the password or username could not be worked out easily once the hash function has been applied. However, this is a feature that I would not implement unless I was releasing this to the public.

Additionally, as discussed throughout the evaluation and testing of the system, the time constraints for developing this system did not allow me to fix the errors that sometimes occur with deleting vertices in the graph section, which I think is due to the way Tkinter determines the position of the place the user clicked not always matching with where the vertex is. Additionally, the randomisation of the questions involving graphs and constraints plotted on axes made some questions very difficult to read and understand due to limitations in the plotting mechanisms of the NetworkX library and the values that could create questions that can be solved for Simplex algorithm questions. These issues can be fixed in post-development.

In terms of maintenance, my program is modular in nature due to me designing and developing it to be written mostly in OOP. Additionally, I have commented my code and put commented print statements in to ensure that the functionality and specifics of the code are clear, along with the ability to follow the program flow more easily if the print statements were uncommented. Therefore, maintaining the code in the future if any errors were to arise would be simpler. Adding further functionality like more algorithm visualisations would also be easier because of the pre-existing structure that I have used for the algorithm visualisations and the modularity of the code.

POST DEVELOPMENT

My first focus in the post-development stage would be to fix the bugs with deleting vertices, which was found during the Success Criteria testing in Prototype 3, and the graphs and constraints randomised for the Prim's, Dijkstra's and Simplext questions in the Quiz section. Additionally, completing the functionality to highlight as backtracking happens at the end of the Dijkstra's visualisation is another task to complete, which I discussed in success criteria [6][i] but was not fully met.

Then, in terms of features of a good secure system, implementing a hashing function and hash table to store the usernames and passwords in the database would be the next task in post development . As discussed in the LIMITATIONS AND MAINTENANCE section, this would add security to the system because the value of the password could not be worked out from the hash value stored. Furthermore, hash values allow for fast and efficient retrieval of data, which would be useful if the system were to grow to have many users.

Then, the enhancements suggested by my stakeholders in the stakeholder reviews after each prototype could be the next stage in post-development. The suggested improvements were the following:

- Functionality to view the passwords in the Login and Register page password fields so that checking that their entry is correct could be easier for the user.
- Allowing users to enter decimal (float or real) numbers in the Bubble Sort visualisation.
- Functionality to reset the visualisations, which would clear the data entered and allow the user to start again rather than have to go back to the home page and then re-enter the visualisation page.
- Having the ability to display the solution to the Dijkstra's algorithm similar to the way done in the A-Level Further Maths Decision 1[32] module instead of with the vertex table, which is used in Computer Science A-Level.

A focus on implementing more algorithm visualisations would be the next stage in post development. As discussed in the PROJECT DEFINITION section, this system would be very useful to students studying A-Level Computer Science or the Decision Maths module in A-Level Further Maths. Therefore, some algorithms that could be implemented in the future are:

- The Planarity algorithm
- The Route Inspection algorithm
- Bin Packing algorithms
- Critical Path algorithm
- Insertion Sort algorithm
- Merge Sort algorithm
- Quick Sort algorithm
- A* Pathfinding algorithm

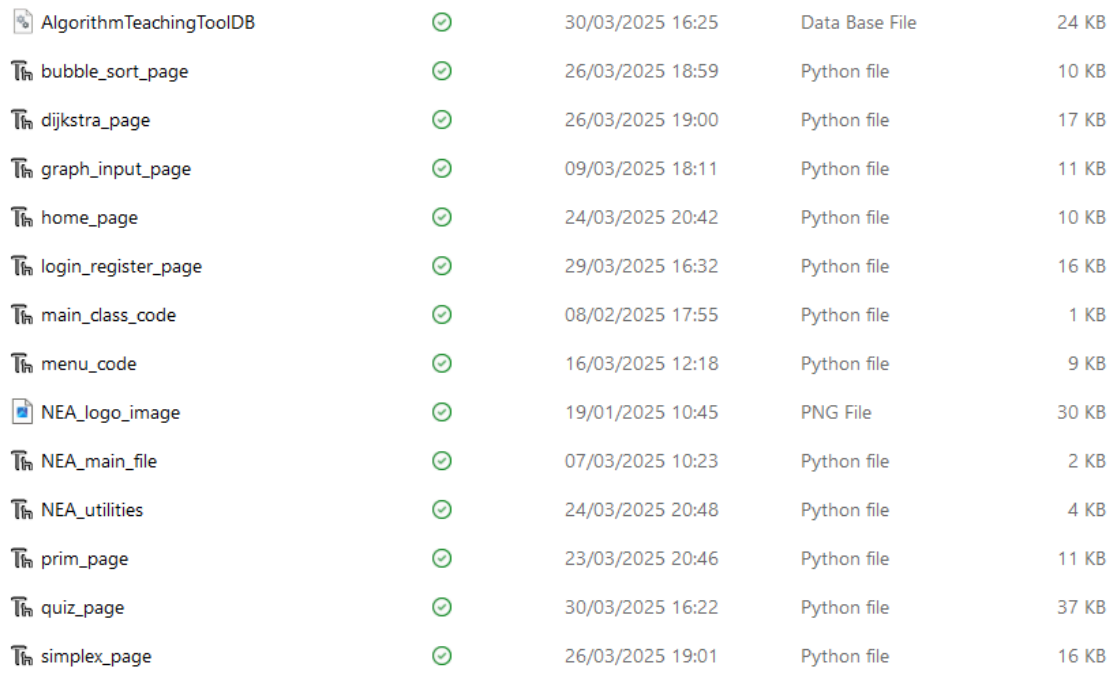
A third focus in post development could be adding functionality for users to join classes or study groups to give users more reasons to utilise this software. The classes could be managed by a teacher and allow the teacher to set specific questions or algorithms to practice for the students. The study groups could be made up of multiple users and their quiz statistics could be shared to try and

motivate them with their revision of the algorithms as this system is mostly designed with students in mind, although it can be used by teachers.

FINAL CODE

In order to display the final code for the system clearly, I will show screenshots collected in the files that I split the project into. Print statements that I have used that may be useful in checking program flow in post-development have been left in the program but commented out.

Firstly, below is the file structure for everything required for the system:



AlgorithmTeachingToolDB	30/03/2025 16:25	Data Base File	24 KB
bubble_sort_page	26/03/2025 18:59	Python file	10 KB
dijkstra_page	26/03/2025 19:00	Python file	17 KB
graph_input_page	09/03/2025 18:11	Python file	11 KB
home_page	24/03/2025 20:42	Python file	10 KB
login_register_page	29/03/2025 16:32	Python file	16 KB
main_class_code	08/02/2025 17:55	Python file	1 KB
menu_code	16/03/2025 12:18	Python file	9 KB
NEA_logo_image	19/01/2025 10:45	PNG File	30 KB
NEA_main_file	07/03/2025 10:23	Python file	2 KB
NEA_utilities	24/03/2025 20:48	Python file	4 KB
prim_page	23/03/2025 20:46	Python file	11 KB
quiz_page	30/03/2025 16:22	Python file	37 KB
simplex_page	26/03/2025 19:01	Python file	16 KB

NEA_main_file.py :

```python
# NEA_main_file: file for starting the system by taking the user to the login page

# imports specific to the initial main class
from tkinter import *
import tkinter as tk
import sqlite3
from login_register_page import LoginRegister


conn1 = sqlite3.connect("AlgorithmTeachingToolDB.db")
#print("open successful")

conn1.execute('''CREATE TABLE IF NOT EXISTS Users
            (Username       TEXT    NOT NULL    PRIMARY KEY,
            Password        TEXT    NOT NULL,
            Account_type    TEXT    NOT NULL ) ;''')

conn1.execute('''CREATE TABLE IF NOT EXISTS StudentEnrolment
            (StudentEnrolID  INTEGER    PRIMARY KEY  AUTOINCREMENT,
            Username        TEXT    NOT NULL,
            Algorithm       TEXT    NOT NULL,
            CorrectScore    INT     NOT NULL,
            IncorrectScore  INT     NOT NULL,
            FOREIGN KEY (Username) REFERENCES Users(Username) ) ; ''')

conn1.execute('''CREATE TABLE IF NOT EXISTS TeacherEnrolment
            (TeacherEnrolID  INTEGER    PRIMARY KEY  AUTOINCREMENT,
            Username    TEXT    NOT NULL,
            Algorithm   TEXT    NOT NULL,
            FOREIGN KEY (Username) REFERENCES Users(Username) ) ; ''')
conn1.commit()
conn1.close()



if __name__ == "__main__":
    loginRoot = tk.Tk()
    loginRoot.geometry('625x400')
    loginRoot.title("Algorithms Simulator & Teaching Tool")
    running1 = LoginRegister(loginRoot)
    loginRoot.mainloop()
```

main_class_code.py :

```python
# imports specific to the Main class
import sqlite3

class Main:
    def __init__(self, master):
        self.master = master # this is the window
        self.title = None
        # database connection:
        self.conn = sqlite3.connect("AlgorithmTeachingToolDB.db") # actual database
        self.cursor = self.conn.cursor() # for selecting

    def closeConn (self):
        self.conn.close()
```

NEA_utilities.py :

```python
# NEA_utilities: file for closing one page and opening another, contains the fonts

# tkinter import:
from tkinter import *
import tkinter as tk


def closeOpen(root, newType, username=None):
    #print("in closeOpen")
    root.destroy()
    if newType == "home":
        try:
            from home_page import HomeMain
            homeRoot = Tk()
            homeRoot.geometry('1150x740') # to change
            homeRoot.title("Algorithms Simulator and Teaching Tool")
            running = HomeMain(homeRoot, username)
            homeRoot.mainloop()
        except ImportError as e:
            print(f"Failed import HomeMain: {e}")
            return

    elif newType == "login":
        try:
            from login_register_page import LoginRegister
            loginRoot = Tk()
            loginRoot.geometry('625x400')
            loginRoot.title("Algorithms Simulator & Teaching Tool")
            running = LoginRegister(loginRoot)
            loginRoot.mainloop()
        except ImportError as e:
            print(f"Failed import LoginRegister: {e}")
            return
```

```python

     elif newType == "bubble sort":
         try:
             from bubble_sort_page import BubbleSort
             bubbleSortRoot = Tk()
             bubbleSortRoot.geometry('400x1000') # to change
             bubbleSortRoot.title("Algorithms Simulator and Teaching Tool")
             running = BubbleSort(bubbleSortRoot, username)
             bubbleSortRoot.mainloop()
         except ImportError as e:
             print(f"Failed import BubbleSort: {e}")
             return
```

```
46
47      elif newType == "prim":
48          try:
49              from prim_page import Prim
50              primRoot = Tk()
51              primRoot.geometry('500x350') # to change
52              primRoot.title("Algorithms Simulator and Teaching Tool")
53              running = Prim(primRoot, username)
54              primRoot.mainloop()
55          except ImportError as e:
56              print(f"Failed import Prim: {e}")
57              return
58
59      elif newType == "dijkstra":
60          try:
61              from dijkstra_page import Dijkstra
62              dijkstraRoot = Tk()
63              dijkstraRoot.geometry('500x350') # to change
64              dijkstraRoot.title("Algorithms Simulator and Teaching Tool")
65              running = Dijkstra(dijkstraRoot, username)
66              dijkstraRoot.mainloop()
67          except ImportError as e:
68              print(f"Failed import Dijkstra: {e}")
69              return
70
71      elif newType == "simplex":
72          try:
73              from simplex_page import Simplex
74              simplexRoot = Tk()
75              simplexRoot.geometry('500x350') # to change
76              simplexRoot.title("Algorithms Simulator and Teaching Tool")
77              running = Simplex(simplexRoot, username)
78              simplexRoot.mainloop()
79          except ImportError as e:
80              print(f"Failed import Simplex: {e}")
81              return
```

```
83      elif newType == "quiz":
84          try:
85              from quiz_page import Quiz
86              quizRoot = Tk()
87              quizRoot.geometry('500x350') # to change
88              quizRoot.title("Algorithms Simulator and Teaching Tool")
89              running = Quiz(quizRoot, username)
90              quizRoot.mainloop()
91          except ImportError as e:
92              print(f"Failed import Quiz: {e}")
93              return
94
95  # fonts:
96  fontLarge = ("Calibri", 16, "bold")
97  fontMedium = ("Calibri", 12)
98  fontSmall = ("Calibri", 10)
```

login_register_page.py :

```python
# imports required specific to the LoginRegister class
from tkinter import *
import tkinter as tk
import sqlite3
import string
from main_class_code import Main
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall


class LoginRegister(Main):
    def __init__(self, master):
        super().__init__(master) # inherits all attribtues of Main class

        # login frame variables
        self.loginFrame = None
        self.username = StringVar()
        self.password = StringVar()
        self.loginInvalid = None

        # register frame variables
        self.regFrame = None
        self.newUsername = StringVar()
        self.newPassword = StringVar()
        self.newConfirmPassword = StringVar()

        # invalid message variables
        self.usernameInvalid = None
        self.passwordInvalid = None
        self.passwordsMatchInvalid = None
        self.invalidMessageReg = None

        # algorithm selection variables - 1 is onvalue, 0 is offvalue
        self.bubbleSortSelect = IntVar()
        self.primSelect = IntVar()
        self.dijkstraSelect = IntVar()
        self.simplexSelect = IntVar()
        self.algorithmsChosen = {"Bubble Sort": self.bubbleSortSelect,
                                 "Prim's": self.primSelect,
                                 "Dijkstra's": self.dijkstraSelect,
                                 "Simplex": self.simplexSelect}
        self.accountType = StringVar() # account selection
        self.loginRegisterWidgets()
        self.master.configure(bg="#eaebed")
```

```
    def loginRegisterWidgets(self):
        self.title = Label(self.master, text="Login", font=fontLarge, fg="#1b263b", bg="#eaebed")
        self.title.grid(row=0, column=0, columnspan=5)

        # login frame widgets
        self.loginFrame = Frame(self.master, bg="#eaebed")
        Label(self.loginFrame, text="Username:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=2, column=0, columnspan=2)
        Label(self.loginFrame, text="Password:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=3, column=0, columnspan=2)
        Entry(self.loginFrame, textvariable=self.username).grid(row=2, column=2, columnspan=3)
        Entry(self.loginFrame, textvariable=self.password, show="•").grid(row=3, column=2, columnspan=3)
        self.loginInvalid = Label(self.loginFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed") # unsuccessful login text
        self.loginInvalid.grid(row=5, column=0, columnspan=5)
        Button(self.loginFrame, text="Login", command=self.loginCheck, activebackground="white", activeforeground="#1b263b",
               bg="#1b263b", fg="white", font=fontMedium, width=10).grid(row=6, column=1, padx=10, pady=5)
        Button(self.loginFrame, text="Register", command=self.newRegistrationPage, activebackground="white", activeforeground="#1b263b",
               bg="#1b263b", fg="white", font=fontMedium, width=10).grid(row=6, column=3, padx=10, pady=5)

        self.loginFrame.grid(row=1, column=0, columnspan=5) # first frame accessed

        # register frame widgets
        self.regFrame = Frame(self.master, bg="#eaebed")
        Label(self.regFrame, text="Usernames should be unique. Passwords should have at least 8 characters,\nuppercase and lowercase letters and contain numbers and special characters.",
              font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=1, column=0, columnspan=6, pady=10)
        Label(self.regFrame, text="Username:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=2, column=0, columnspan=2)
        Label(self.regFrame, text="Password:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=3, column=0, columnspan=2)
        Label(self.regFrame, text="Password confirmation:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=4, column=0, columnspan=2, padx=10)
        Label(self.regFrame, text="Select algorithms\nto study:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=5, column=0, columnspan=2)
        Label(self.regFrame, text="Select account type:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=7, column=0, columnspan=2)
        # entry fields:
        Entry(self.regFrame, textvariable=self.newUsername).grid(row=2, column=2, columnspan=2)
        Entry(self.regFrame, textvariable=self.newPassword, show="•").grid(row=3, column=2, columnspan=2)
        Entry(self.regFrame, textvariable=self.newConfirmPassword, show="•").grid(row=4, column=2, columnspan=2)
```

```
        # invalid messages:
        self.usernameInvalid = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
        self.usernameInvalid.grid(row=2, column=4, columnspan=2)
        self.passwordInvalid = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
        self.passwordInvalid.grid(row=3, column=4, columnspan=2)
        self.passwordsMatchInvalid = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
        self.passwordsMatchInvalid.grid(row=4, column=4, columnspan=2)
        self.invalidMessageReg = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
        self.invalidMessageReg.grid(row=8, column=0, columnspan=4)
        # algorithm checkboxes:
        Checkbutton(self.regFrame, text="Bubble Sort", variable=self.bubbleSortSelect, onvalue=1, offvalue=0,
                    font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=5, column=2, columnspan=2)
        Checkbutton(self.regFrame, text="Prim's Algorithm", variable=self.primSelect, onvalue=1, offvalue=0,
                    font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=5, column=4, columnspan=2)
        Checkbutton(self.regFrame, text="Dijkstra's Algorithm", variable=self.dijkstraSelect, onvalue=1, offvalue=0,
                    font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=6, column=2, columnspan=2)
        Checkbutton(self.regFrame, text="Simplex Algorithm", variable=self.simplexSelect, onvalue=1, offvalue=0, font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=6, column=4, columnspan=2)
        # account selection radiobutton:
        Radiobutton(self.regFrame, text="Student", variable=self.accountType, value="Student", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=7, column=2, columnspan=2)
        Radiobutton(self.regFrame, text="Teacher", variable=self.accountType, value="Teacher", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=7, column=4, columnspan=2)
        # buttons
        Button(self.regFrame, text="Register", command=self.registerCheck, activebackground="white", activeforeground="#1b263b",
               bg="#1b263b", fg="white", font=fontMedium, width=10).grid(row=8, column=4, padx=10, pady=5)
        Button(self.regFrame, text="Back to Login", command=self.newLoginPage, activebackground="white", activeforeground="#1b263b",
               bg="#1b263b", fg="white", font=fontMedium, width=10).grid(row=8, column=5, padx=10, pady=5)
```

```
    def loginCheck(self):
        # if pass - go to closeLoginOpenHome
        #print("in login check")
        usernameHolder = self.username.get()
        if self.validateLoginInput(): # is True
            #print("login validated")
            # clear input fields:
            self.loginInvalid["text"] = ""
            self.username.set("")
            self.password.set("")
            # open home page:
            closeOpen(self.master, "home", usernameHolder)
        #else:
            #print("login failed")
```

```
119     def validateLoginInput(self):
120         #print("in validate login input")
121         username = self.username.get()
122         password = self.password.get()
123         # check for no username / password entered
124         if not username or not password:
125             self.loginInvalid["text"] = "Error: invalid username or password"#
126             self.username.set("")
127             self.password.set("")
128             return False
129         # select from database for entered username and password
130         try:
131             self.cursor.execute("SELECT * FROM Users WHERE Username = ? and Password = ?", (username, password))
132             user = self.cursor.fetchone() # returns values or None
133             if user: # not None
134                 #print("Login successful")
135                 return True
136             else:
137                 #print("login unsuccessful: no values found")
138                 self.loginInvalid["text"] = "Error: invalid username or password"
139                 self.username.set("")
140                 self.password.set("")
141                 return False
142         except sqlite3.Error as e:
143             #print(f"Database error: {e}")
144             self.loginInvalid["text"] = "Error: invalid username or password"
145             self.username.set("")
146             self.password.set("")
147             return False
```

```
150     def registerCheck(self):
151         '''
152         check username is unique - if not, self.usernameInvalid["text"] = "Username unavailable"
153         check passwords match - if not, self.passwordsMatchInvalid["text"] = "Passwords do not match"
154         check password valid - if not, self.passwordInvalid["text"] = "Password invalid"
155         if pass - go to closeLoginOpenHome
156         '''
157         #print("in register check")
158         usernameHolder = self.newUsername.get()
159         passwordHolder = self.newPassword.get()
160         confirmPasswordHolder = self.newConfirmPassword.get()
161         accountHolder = self.accountType.get()
162         valid = False
163 
164         if not usernameHolder or not passwordHolder or not confirmPasswordHolder:
165             self.invalidMessageReg["text"] = "Unsuccessful registration\nCheck all fields have been filled in correctly"
166             return
167         else:
168             self.invalidMessageReg["text"] = ""
169 
170         if not self.validateUsernameReg():
171             self.usernameInvalid["text"] = "Username not available"
172             self.newUsername.set("")
173             return
174         else:
175             self.usernameInvalid["text"] = ""
176 
177         # validate password
178         if not self.validatePasswordReg():
179             self.passwordInvalid["text"] = "Password invalid"
180             self.newPassword.set("")
181             self.newConfirmPassword.set("")
182             return
183         else:
184             self.passwordInvalid["text"] = ""
```

```
186         if passwordHolder != confirmPasswordHolder:
187             self.passwordsMatchInvalid["text"] = "Passwords do not match"
188             self.newPassword.set("")
189             self.newConfirmPassword.set("")
190             return
191         else:
192             self.passwordsMatchInvalid["text"] = ""
193 
194         # validate algorithm choice
195         algorithmValues = [var.get() for var in self.algorithmsChosen.values()]
196         #print(algorithmValues)
197         if not any(value == 1 for value in algorithmValues):
198             self.invalidMessageReg["text"] = "Unsuccessful registration\nCheck all fields have been filled in correctly"
199             #print("algorithms fails")
200             return
201         else:
202             self.invalidMessageReg["text"] = ""
203 
204         # validate account type selection
205         if not accountHolder: # not None
206             self.invalidMessageReg["text"] = "Unsuccessful registration\nCheck all fields have been filled in correctly"
207             #print("account fails")
208             return
209         else:
210             self.invalidMessageReg["text"] = ""
```

```
        # enter values into database
        try:
            self.cursor.execute("INSERT INTO Users (Username, Password, Account_type) VALUES (?, ?, ?)", (usernameHolder, passwordHolder, accountHolder))
            self.conn.commit()
            #print("inserted into Users")

            if accountHolder == "Student":
                for name, numValue in self.algorithmsChosen.items():
                    if numValue.get() == 1:
                        self.cursor.execute("INSERT INTO StudentEnrolment (Username, Algorithm, CorrectScore, IncorrectScore) VALUES (?, ?, ?, ?)", (usernameHolder, name, 0, 0))
                        self.conn.commit()
                #print("inserted into StudentEnrolment")

            elif accountHolder == "Teacher":
                for name, numValue in self.algorithmsChosen.items():
                    if numValue.get() == 1:
                        self.cursor.execute("INSERT INTO TeacherEnrolment (Username, Algorithm) VALUES (?, ?)", (usernameHolder, name))
                        self.conn.commit()
                 #print("inserted into TeacherEnrolment")

            #print("Registration successful")
            # clear inputs after successful registration:

            self.newUsername.set("")
            self.newPassword.set("")
            self.newConfirmPassword.set("")
            for value in self.algorithmsChosen.values():
                value.set(0)
            self.accountType.set("")
            self.usernameInvalid["text"] = ""
            self.passwordInvalid["text"] = ""
            self.passwordsMatchInvalid["text"] = ""
            self.invalidMessageReg["text"] = ""
            closeOpen(self.master, "home", usernameHolder)
        except sqlite3.Error as e:
            #print(f"Database error: {e}")
            self.invalidMessageReg["text"] = "Unsuccessful registration\nCheck all fields have been filled in correctly"
```

```
251     def validateUsernameReg(self):
252         #print("in validate username")
253         usernameHolder = self.newUsername.get()
254         self.cursor.execute("SELECT Username FROM Users WHERE Username = ?", (usernameHolder,)) # comma to make tuple
255         usernameFound = self.cursor.fetchone()
256         if usernameFound:
257             return False
258         else:
259             return True
260 
261     def validatePasswordReg(self):
262         #print("in valdiate password")
263         passwordHolder = self.newPassword.get()
264          # longer than 8 characters (inclusive)
265         if len(passwordHolder) >= 8:
266             # contains both uppercase and lowercase characters
267             if any(character.isupper() for character in passwordHolder) == True and any(character.islower() for character in passwordHolder) == True:
268                 # contains special chars - held in string.punctuation
269                 if any(character in string.punctuation for character in passwordHolder) == True:
270                     # contains number
271                     if any(character.isdigit() for character in passwordHolder) == True:
272                         return True # all tests passed
273         return False # at least 1 test failed
```

```
276     def newRegistrationPage(self):
277         self.title["text"] = "Create an account"
278         self.newUsername.set("")
279         self.newPassword.set("")
280         self.newConfirmPassword.set("")
281         for value in self.algorithmsChosen.values():
282             value.set(0)
283         self.accountType.set("")
284         self.usernameInvalid["text"] = ""
285         self.passwordInvalid["text"] = ""
286         self.passwordsMatchInvalid["text"] = ""
287         self.invalidMessageReg["text"] = ""
288         print("in new registration page")
289         self.loginFrame.grid_forget()
290         self.regFrame.grid(row=1, column=0, columnspan=6)
```

```
    def newLoginPage(self):
        self.title["text"] = "Login"
        self.loginInvalid["text"] = ""
        self.username.set("")
        self.password.set("")
        print("in new login page")
        self.regFrame.grid_forget()
        self.loginFrame.grid(row=1, column=0, columnspan=6)
```

home_page.py :

```
# imports required specific to the HomeMain class
from tkinter import *
import tkinter as tk
import sqlite3
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg # allows matplotlib graph to be put in tkinter window
from PIL import Image, ImageTk # for logo image editing
from main_class_code import Main
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall


class HomeMain(Main):
    def __init__(self, master, pUsername):
        #print(f"inside HomeMain __init__")
        super().__init__(master)
        #print("Back inside HomeMain __init__")
        self.homeFrame = None

        # current account username:
        self.currentAccUsername = pUsername
        self.accountType = self.accountTypeGet()

        if self.accountType == "Student":
            # holds algorithm names to go as labels on the x axis
            self.algorithmNames = self.algorithmAxisValues("Name")
            # hold the correct and incorrect scores to be plot as bars on the bar chart
            self.correctScores = self.algorithmAxisValues("Correct")
            self.incorrectScores = self.algorithmAxisValues("Incorrect")
            self.master.geometry("1150x740")

        elif self.accountType == "Teacher":
            self.algorithmNames = self.teacherAlgorithmsGet()
            self.master.geometry("935x775")

        self.logoImg = None
        self.master.configure(bg="#eaebed")
        HomeMain.pageStarter(self)
```

```
    def pageStarter(self):
        self.homeWidgets()
        if self.accountType == "Student":
            self.plotStatistics()
        elif self.accountType == "Teacher":
            self.displayLogo()


    def accountTypeGet(self):
        usernameHolder = self.currentAccUsername
        #print(usernameHolder)
        self.cursor.execute("SELECT Account_type FROM Users WHERE Username=?", (usernameHolder,))
        result = self.cursor.fetchone()
        print(f"result: {result}")
        accountType = result[0]
        #print(accountType)
        return accountType

    def teacherAlgorithmsGet(self):
        usernameHolder = self.currentAccUsername
        algorithmList = []
        self.cursor.execute("SELECT Algorithm FROM TeacherEnrolment WHERE Username=?", (usernameHolder,))
        result = self.cursor.fetchall()
        #print(result)
        algorithmList = [row[0] for row in result]
        #print(algorithmList)
        return algorithmList
```

```
67      def homeWidgets(self):
68          self.title = Label(self.master, text="Algorithm's Simulator and Teaching Tool", font=fontLarge, fg="#1b263b", bg="#eaebed")
69          self.title.grid(row=0, column=0, columnspan=5)
70
71          self.homeFrame = Frame(self.master, bg="#eaebed")
72
73          buttonLocations = self.locations()
74          #print(buttonLocations)
75          # the index of each item in the list is the column of it in the gui
76          if "Bubble Sort" in buttonLocations:
77              Button(self.homeFrame, text="Bubble Sort", command=self.openBubbleSort, activebackground="white", activeforeground="#1b263b",
78                     bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=1, column=buttonLocations.index("Bubble Sort"), padx=10, pady=5) #bubble sort button
79          if "Prim's" in buttonLocations:
80              Button(self.homeFrame, text="Prim's Algorithm", command=self.openPrim, activebackground="white", activeforeground="#1b263b",
81                     bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=1, column=buttonLocations.index("Prim's"), padx=10, pady=5) # prim's button
82          if "Dijkstra's" in buttonLocations:
83              Button(self.homeFrame, text="Dijkstra's Algorithm", command=self.openDijkstra, activebackground="white", activeforeground="#1b263b",
84                     bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=1, column=buttonLocations.index("Dijkstra's"), padx=10, pady=5) # dijkstra button
85          if "Simplex" in buttonLocations:
86              Button(self.homeFrame, text="Simplex Algorithm", command=self.openSimplex, activebackground="white", activeforeground="#1b263b",
87                     bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=1, column=buttonLocations.index("Simplex"), padx=10, pady=5) # simplex button
88          if "Quiz" in buttonLocations:
89              Button(self.homeFrame, text="Quiz", command=self.openQuiz, activebackground="white", activeforeground="#1b263b", bg="#1b263b",
90                     fg="white", font=fontMedium, width=20).grid(row=1, column=buttonLocations.index("Quiz"), padx=10, pady=5) # quiz button
91
92          if self.accountType == "Student":
93              Button(self.homeFrame, text="Logout", command=self.logout, activebackground="white", activeforeground="#1b263b", bg="#1b263b",
94                     fg="white", font=fontMedium, width=20).grid(row=4, column=4, padx=10, pady=5) # logout button Student
95          elif self.accountType == "Teacher":
96              Button(self.homeFrame, text="Logout", command=self.logout, activebackground="white", activeforeground="#1b263b", bg="#1b263b",
97                     fg="white", font=fontMedium, width=20).grid(row=7, column=3, padx=10, pady=5) # logout button Teacher
98         self.homeFrame.grid(row=1, column=0, columnspan=5)
```

```
101     def locations(self):
102         try:
103             buttonLocations = []
104             if self.accountType == "Student":
105                 for counter in range (len(self.algorithmNames)-1):
106                     buttonLocations.append(self.algorithmNames[counter])
107                 buttonLocations.append("Quiz")
108             elif self.accountType == "Teacher":
109                 buttonLocations = self.algorithmNames
110             return buttonLocations
111         except sqlite3.Error as e:
112             print(f"Database error: {e}")
```

```
    def algorithmAxisValues(self, listType):
        valuesList = []
        try:
            self.cursor.execute("SELECT Algorithm, CorrectScore, IncorrectScore FROM StudentEnrolment WHERE Username = ?", (self.currentAccUsername,))
            results = self.cursor.fetchall()
            #print(results)
            if listType == "Name":
                for row in results:
                    valuesList.append(row[0])
                valuesList.append("Total")
                #self.algorithmNames = valuesList
            elif listType == "Correct":
                totalCorrect = 0
                for row in results:
                    valuesList.append(row[1])
                    totalCorrect += row[1]
                valuesList.append(totalCorrect)
                #self.correctScores = valuesList
            elif listType == "Incorrect":
                totalIncorrect = 0
                for row in results:
                    valuesList.append(row[2])
                    totalIncorrect += row[2]
                valuesList.append(totalIncorrect)
                #self.incorrectScores = valuesList
        except sqlite3.Error as e:
            print(f"Database error: {e}")
        return valuesList
```

```
143     def plotStatistics(self):
144         # create figure and axes for the bar chart
145         # fig = entire chart (figure) area
146         # ax represents the axes where the bars are plotted
147         #print(f"Correct Scores: {self.correctScores}")
148         #print(f"Incorrect Scores: {self.incorrectScores}")
149         fig, ax = plt.subplots(figsize=(5,5))
150         x = range(len(self.algorithmNames))
151 
152         # plot the bars
153         ax.bar(x, self.correctScores, width=0.3, label="Correct", align="center", color="#606c38")
154         ax.bar([p + 0.3 for p in x], self.incorrectScores, width=0.3, label = "Incorrect", align = "center", color = "#b26754")
155 
156         # labels, title, legend
157         ax.set_xlabel("Algorithms", fontsize=11, fontfamily="Calibri", color="#1b263b")
158         ax.set_ylabel("Score", fontsize=11, fontfamily="Calibri", color="#1b263b")
159         ax.set_title("Quiz Statistics", fontsize=12, fontfamily="Calibri", color="#1b263b")
160         ax.set_xticks([p + 0.3 for p in x])
161         ax.set_xticklabels(self.algorithmNames, fontsize=10, fontfamily="Calibri", color="#1b263b")
162         ax.legend()
163 
164         # embed into Tkinter
165         canvas = FigureCanvasTkAgg(fig, self.homeFrame)
166         canvasWidget = canvas.get_tk_widget()
167         canvasWidget.grid(row=2, column=1, columnspan=3)     # rowspan = 4
168         canvasWidget.config(bg="#eaebed")
```

```
170     def displayLogo(self):
171         # open the image
172         image = Image.open("NEA_logo_image.png")
173         image = image.resize((481,626))
174         self.logoImg = ImageTk.PhotoImage(image) # creates tkinter widget
175 
176         # create a label and set the image
177         imageLabel = Label(self.homeFrame, image=self.logoImg)
178         imageLabel.grid(row=2, column=0, columnspan=4, rowspan=5)
```

```
    def openBubbleSort(self):
        usernameHolder = self.currentAccUsername
        closeOpen(self.master, "bubble sort", usernameHolder)

    def openPrim(self):
        usernameHolder = self.currentAccUsername
        closeOpen(self.master, "prim", usernameHolder)

    def openDijkstra(self):
        usernameHolder = self.currentAccUsername
        closeOpen(self.master, "dijkstra", usernameHolder)

    def openSimplex(self):
        usernameHolder = self.currentAccUsername
        closeOpen(self.master, "simplex", usernameHolder)

    def openQuiz(self):
        usernameHolder = self.currentAccUsername
        closeOpen(self.master, "quiz", usernameHolder)

    def logout(self):
        closeOpen(self.master, "login")

    def hideHomeWidgets(self):
        for widget in self.homeFrame.winfo_children():
            widget.grid_forget()
        self.title.grid_forget()
```

bubble_sort_page.py :

```
# imports specific to the BubbleSort class
from tkinter import *
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg # allows matplotlib graph to be put in tkinter window
from menu_code import Menu
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
```

```
class BubbleSort(Menu):
    def __init__(self, master, pUsername, show_menu=True):
        Menu.__init__(self, master, 12, 19)
        self.numbers = []
        self.userEntries = []

        self.username = pUsername

        self.master.configure(bg="#eaebed")
        self.master.geometry("1105x905") # required here as goes back into HomeMain first

        self.sortOrder = StringVar(value="Ascending")

        self.invalidMessage = None

        self.fig = None
        self.ax = None
        self.canvas = None

        self.visualiseText = None

        self.numberPasses = 0

        self.bubbleSortWidgets()
        if show_menu:
            self.menuWidgets()
```

```
    def bubbleSortWidgets(self):
        # BUBBLE SORT FRAME:
        self.bubbleSortFrame = Frame(self.master, bg="#eaebed")
        self.bubbleSortFrame.grid(row=0, column=0, columnspan=19, rowspan=12)

        # TITLE / TOP SECTION:
        self.title = Label(self.bubbleSortFrame, text="Bubble Sort Algorithm Visualisation", font=fontLarge, fg="#1b263b", bg="#eaebed")
        self.title.grid(row=0, column=0, columnspan=17)

        Button(self.bubbleSortFrame, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=17, columnspan=2)

        Label(self.bubbleSortFrame,
              text= "1. Start at the first item in the list.\n"
                    "2. Compare the current item with the next item.\n"
                    "3. If the two items are in the wrong position, swap them.\n"
                    "4. Move up one item to the next time in the list.\n"
                    "5. Repeat from step 3 until all the unsorted items have been compared.\n"
                    "6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=1, column=0, columnspan=19, sticky="w", padx=10, pady=5) # justify aligns the text to the left within the label

        # create 8 entry fields for user to enter numbers into
        self.entryFrame = Frame(self.bubbleSortFrame, bg="#eaebed")
        self.entryFrame.grid(row=4, column=7, columnspan=12, rowspan=2)
        for i in range(8):
            entry = Entry(self.entryFrame, width=3)
            entry.grid(row=0 , column=(i+7), padx=2)
            #print(f"userEntries: {self.userEntries}")
            self.userEntries.append(entry)
        #print(f"userEntries: {self.userEntries}")
```

```
        self.addButton = Button(self.entryFrame, text="Add Number", command=self.addEntry, bg="#1b263b", fg="white", font=fontMedium, width=15)
        self.addButton.grid(row=1, column=15, columnspan=4)

        # ascending descending choice:
        Radiobutton(self.bubbleSortFrame, text="Ascending", variable=self.sortOrder, value="Ascending", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=10, column=0, columnspan=3)
        Radiobutton(self.bubbleSortFrame, text="Descending", variable=self.sortOrder, value="Descending", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=10, column=3, columnspan=3)

        Button(self.bubbleSortFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=10, column=11, columnspan=4)

        self.invalidMessage = Label(self.bubbleSortFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
        self.invalidMessage.grid(row=11, columnspan=19)

        # dataset plot
        self.fig, self.ax = plt.subplots(figsize=(5,5))
        self.ax.set_xticks([])
        self.canvas = FigureCanvasTkAgg(self.fig, master=self.bubbleSortFrame)
        self.canvas.get_tk_widget().grid(row=4, column=0, columnspan=7, rowspan=5, pady=5)

        self.visualiseText = Text(self.bubbleSortFrame, width=50, height=30)
        self.visualiseText.grid(row=6, column=7, columnspan=12, rowspan=4)
```

```
    def addEntry(self):
        #print("in add entry")
        if len(self.userEntries) < 11:
            entry = Entry(self.entryFrame, width=3)
            entry.grid(row=0, column=(7+len(self.userEntries)))
            self.userEntries.append(entry)
        elif len(self.userEntries) == 11:
            entry = Entry(self.entryFrame, width=3)
            entry.grid(row=0, column=(7+len(self.userEntries)))
            self.userEntries.append(entry)
            self.addButton.grid_forget()

    def updateChart(self, numbers, swapIndices=None):
        # this is for animating the swapping of items on the bar chart
        #print("in update chart")
        self.ax.clear()
        barColours = ["#606c38"] * len(numbers) # all bars blue
        if swapIndices: # not None
            for index in swapIndices:
                barColours[index] = "#b2675e" # swapping items in red
        self.ax.bar(range(len(numbers)), numbers, color=barColours)
        self.canvas.draw()
        self.bubbleSortFrame.update()
        #time.sleep(0.3) # smooth animation
```

```
def validate(self):
    #print("in validate")
    for entry in self.userEntries:
        try:
            num = int(entry.get())
            if 1 <= num <= 100:
                self.numbers.append(num)
            else:
                raise ValueError
        except ValueError:
            self.invalidMessage["text"] = "Invalid dataset: all values must be integers between 1 and 100"
            self.numbers = []
            return

    if len(self.numbers) < 8 or len(self.numbers) > 12:
        #print("in length error")
        self.invalidMessage["text"] = "Invalid dataset: the dataset must consist of between 8 and 12 numbers (inclusive)"
        self.numbers = []
        return

    self.invalidMessage["text"] = ""
    self.bubbleSortSteps()
    #print(self.steps)
    #print(self.stepsDictionary)
    self.bubbleSortAnimate()
```

```
def bubbleSortSteps(self):
    #print("in bubble sort steps")
    n = len(self.numbers)
    swapped = True
    stepsDict = {}
    stepsDict[tuple(self.numbers)] = [0]

    if self.sortOrder.get() == "Ascending":
        while swapped:
            swapped = False
            for j in range(n-1):
                if self.numbers[j] > self.numbers[j+1]:
                    temp= self.numbers[j]
                    self.numbers[j] = self.numbers[j+1]
                    self.numbers[j+1] = temp
                    swapped = True
                    #self.steps.append(self.numbers[:])
                    # note must be tuple because lists are mutable so they are unhashable
                    stepsDict[tuple(self.numbers)] = [j, j+1]
```

```
                if swapped:
                    self.numberPasses += 1
        elif self.sortOrder.get() == "Descending":
            while swapped:
                swapped = False
                for j in range(n-1):
                    if self.numbers[j] < self.numbers[j+1]:
                        temp= self.numbers[j]
                        self.numbers[j] = self.numbers[j+1]
                        self.numbers[j+1] = temp
                        swapped = True
                        #self.steps.append(self.numbers[:])
                        # note must be tuple because lists are mutable so they are unhashable
                        stepsDict[tuple(self.numbers)] = [j, j+1]
                if swapped:
                    self.numberPasses += 1

        self.stepsDictionary = stepsDict
        self.steps = list(self.stepsDictionary.keys())
```

```
    def bubbleSortAnimate(self):
        #print("in bubble sort animate")
        if not self.isPaused and self.currentStep < len(self.steps)-1:
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
            self.visualiseText.insert(tk.END, f"{numbers}\n")
            self.visualiseText.see(tk.END)

            self.bubbleSortFrame.update()
            self.currentStep += 1
            self.master.after(750, self.bubbleSortAnimate)

        elif not self.isPaused and self.currentStep == len(self.steps)-1:
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
            self.visualiseText.insert(tk.END, f"{numbers}\n")
            self.visualiseText.see(tk.END)
            #print("sort complete")
            self.invalidMessage["text"] = "Sort complete"
            self.bubbleSortFrame.update()


    def openHome(self):
        closeOpen(self.master, "home", self.username)

# for testing ------------------------------------------------------------------------------
if __name__ == "__main__":
    root = tk.Tk()
    root.title("BubbleSort test")
    #root.geometry("800x595")
    graph_input = BubbleSort(root, "HelloWord!!!") #HellowWord!!! username for testing purposes
    root.mainloop()
```

prim_page.py :

```
# imports specific to the Prim class
from tkinter import *
import tkinter as tk
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
from graph_input_page import GraphInput
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import networkx as nx
import heapq # for a queue for prim's algorithm
from menu_code import Menu
```

```
class Prim(GraphInput, Menu): #HomeMain
    def __init__(self, master, pusername, show_menu=True):
        self.master = master
        #print("in prim __init__ 1")
        GraphInput.__init__(self, master)
        #print("in prim __init__ after GraphInput")
        Menu.__init__(self, master, 13, 12)
        #print("in prim__init__ after Menu")
        self.username = pusername
        self.primWidgets()
        if show_menu:
            self.menuWidgets()
            self.menuText["text"] = "Use right/left arrow keys to step forward/back\nthrough the visualisation when paused."
```

```
    def primWidgets(self):
        self.master.configure(bg="#eaebed")
        self.master.geometry("845x640")

        self.primFrame = Frame(self.master, bg="#eaebed")
        self.primFrame.grid(row=0, column=0, columnspan=12, rowspan=13)

        self.title = Label(self.primFrame, text="Prim's Algorithm Visualisation", fg="#1b263b", bg="#eaebed", font=fontLarge)
        self.title.grid(row=0, column=0, columnspan=10) # change columnspan

        Button(self.primFrame, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=10, columnspan=2)

        Label(self.primFrame,
              text= "1. Choose any vertex to start the tree.\n"
                    "2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.\n"
                    "3. Repeat step 2 until all the vertices are connected to the tree.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=1, column=0, columnspan=12, rowspan=3, sticky="w", padx=10, pady=5)

        Label(self.primFrame,
              text="In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.\n"
                   "Then choose a start vertex from the list and visualise Prim's algorithm on the graph.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=4, column=0, columnspan=12, rowspan=2, sticky="w", padx=10, pady=5)

        self.graphInputButton = Button(self.primFrame, text="Input a graph", bg="#1b263b", fg="white",
                                       font=fontMedium, width=20, command=self.graphInputWindow)
        self.graphInputButton.grid(row=6, column=1, columnspan=2)
```

```
        Label(self.primFrame, text="Start vertex:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=6, column=8, columnspan=2)

        Label(self.primFrame, text="MST edges:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=7, column=8, columnspan=4)
        self.visualiseText = Text(self.primFrame, width=30, height=16)
        self.visualiseText.grid(row=8, column=8, columnspan=4, rowspan=4, padx=5)

        Button(self.primFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=12, column=0, columnspan=3, padx=10, pady=5)

        self.invalidMessage = Label(self.primFrame, text="", font=fontMedium, bg="#eaebed", fg="#990000")
        self.invalidMessage.grid(row=12, column=3, columnspan=9)

        self.startVertexOptions = ["Select"] # initially Select - changes to the list of vertices once graph input
        self.startVertexChoice = StringVar(self.primFrame)
        self.startVertexChoice.set("Select")

        self.startVertexDropdown = OptionMenu(self.primFrame, self.startVertexChoice, *self.startVertexOptions)
        self.startVertexDropdown.grid(row=6, column=10, columnspan=2)
        self.startVertexDropdown.config(font=fontSmall, bg="#eaebed", fg="#1b263b")
        self.startVertexDropdown["menu"].config(font=fontSmall, bg="#eaebed", fg="#1b263b")


    def openHome(self):
        closeOpen(self.master, "home", self.username)
```

```
    def graphInputWindow(self):
        # open graph input window (but don't close current window
        #print("in graph input window method")
        # hide widgets on prim page
        self.hidePrimWidgets()
        # hide menu widgets
        self.hideMenuWidgets()
        # create graph page to open
        self.graphInputPageWidgets()
```

```
    def primExtraWidgets(self):
        # show menu widgets
        self.menuWidgets()
        self.menuText["text"] = "Use right/left arrow keys to step forward/back\nthrough the visualisation when paused."

        # vertex dropdown menu
        self.startVertexOptions = list(self.graph.nodes)
        self.startVertexChoice.set(self.startVertexOptions[0] if self.startVertexOptions else "Select")
        self.startVertexDropdown["menu"].delete(0, "end")
        for vertex in self.startVertexOptions:
            self.startVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.startVertexChoice.set(v))

        # destroy button so axes can go there
        self.graphInputButton.destroy()
        # change size of window
        self.master.geometry("845x697")

        # create axes to plot graph on
        self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
        self.primGraphCanvas = FigureCanvasTkAgg(self.fig, self.primFrame)
        self.primGraphCanvas.get_tk_widget().grid(row=6, column=0, rowspan=6, columnspan=8)
        # draw graph on axes
        pos = nx.get_node_attributes(self.graph, 'pos')
        nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
        edge_labels = nx.get_edge_attributes(self.graph, 'weight')
        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
        self.primGraphCanvas.draw()
```

```
    def updatePrimGraph(self, step):
        #print("in update graph")
        # will change colours of current vertex and joined edges
        self.ax.clear()
        pos = nx.get_node_attributes(self.graph, "pos")

        nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
        edge_labels = nx.get_edge_attributes(self.graph, "weight")
        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)

        for u,v in step:
            colour = "#b2675e" if step == self.steps[-1] else "#606c38"
            nx.draw_networkx_edges(self.graph, pos, edgelist=[(u,v)], edge_color = colour, width=2, ax=self.ax)

        self.primGraphCanvas.draw()
```

```
124     def hidePrimWidgets(self):
125         for widget in self.primFrame.winfo_children():
126             widget.grid_forget()
127         self.title.grid_forget()
128 
129     def validate(self):
130         #print("in validate prim's")
131         # note that graph has already been validated
132         # checks that a start vertex has been input, then starts finding steps
133         #print(f"start vertex: {self.startVertexChoice}")
134         if self.startVertexChoice == "Select": # select is default option where there is no vertex selected
135             #print("invalid")
136             self.invalidMessage["text"] = "Invalid: please enter a start vertex."
137             # start vertex not chosen
138             return
139         self.invalidMessage["text"] = ""
140         self.primSteps()
141         self.primAnimate()
```

```
161     def primSteps(self, start=None):
162         #print("in prim steps")
163 
164         self.steps = []
165         self.mstEdges = []
166         if start:
167             startVertex = start
168         else:
169             startVertex = self.startVertexChoice.get()
170         visited = set() # stores mst edges in an unordered, unchangeable and unindexed format
171         minHeap = [] # this is the queue for the edges connected to the current vertex
172 
173         visited.add(startVertex)
174 
175         for neighbour, edgeData in self.graph[startVertex].items():
176             weight = edgeData["weight"]
177             # enqueue all connected nodes and edges onto minHeap
178             heapq.heappush(minHeap, (weight, startVertex, neighbour))
179 
180         self.steps.append(tuple(self.mstEdges))
181 
182         while len(visited) < len(self.graph.nodes):
183             if not minHeap:
184                 self.invalidMessage["text"] = "The graph is not fully connected so there is no MST"
185                 return
186             # heappop returns the smallest edge value element in minHeap
187             weight, u, v = heapq.heappop(minHeap)
188             if v not in visited:
189                 visited.add(v)
190                 self.mstEdges.append((u,v))
191                 self.steps.append(tuple(self.mstEdges))
192                 for neighbour, edgeData in self.graph[v].items():
193                     weight = edgeData["weight"]
194                     if neighbour not in visited:
195                         heapq.heappush(minHeap, (weight, v, neighbour))
```

```
    def primAnimate(self):
        # animates at start when not paused
        #print("in primAnimate")
        # test steps:
        if not self.steps:
            #print("Error: no steps recorded in self.steps")
            return

        if not self.isPaused and self.currentStep < len(self.steps):
            mstStep = self.steps[self.currentStep] # current step#s edges
            self.updatePrimGraph(mstStep)
            # show all edges currently in the mst in the Text section
            self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}{v}," for u,v in mstStep]) + "\n")
            self.visualiseText.see(tk.END)

            self.currentStep += 1
            self.master.after(750, self.primAnimate)

        elif not self.isPaused and self.currentStep == len(self.steps):
            mstStep = self.steps[self.currentStep - 1]
            print(mstStep)
            print(str(len(mstStep)))
            if len(mstStep) == len(self.graph)-1:
                self.updatePrimGraph(mstStep)
                self.invalidMessage["text"] = "MST found"
            else:
                self.invalidMessage["text"] = "The graph is not fully connected so there is no MST"

# for testing ------------------------------------------------------------------------
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Prim's test")
    #root.geometry("800x595")
    graph_input = Prim(root, "HelloWord!!!") #HellowWord!!! username for testing purposes
    root.mainloop()
```

dijkstra_page :

```
# imports specific to the Dijkstra's page
from tkinter import *
import tkinter as tk
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
from graph_input_page import GraphInput
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import networkx as nx
from menu_code import Menu
import heapq

class Dijkstra(GraphInput, Menu):
    def __init__(self, master, pUsername, show_menu=True):
        self.master = master
        #print("in prim __init__ 1")
        GraphInput.__init__(self, master)
        #print("in prim __init__ after GraphInput")
        Menu.__init__(self, master, 14, 16)

        self.username = pUsername

        self.dijkstraWidgets()
        if show_menu:
            self.menuWidgets()
            self.menuText["text"] = "Use right/left arrow keys to step forward/back through\nthe visualisation when paused."
```

```
    def dijkstraWidgets(self):
        self.master.configure(bg="#eaebed")
        self.master.geometry("905x505")

        self.dijkstraFrame = Frame(self.master, bg="#eaebed")
        self.dijkstraFrame.grid(row=0, column=0, columnspan=16, rowspan=14)

        self.title = Label(self.dijkstraFrame, text="Dijkstra's Algorithm Visualisation", font=fontLarge, bg="#eaebed", fg="#1b263b")
        self.title.grid(row=0, column=0, columnspan=14)

        Button(self.dijkstraFrame, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=14, columnspan=2)

        Label(self.dijkstraFrame, text=
              "1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).\n"
              "2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.\n"
              "3. Calculate the distance from the start for each of the unvisited connected nodes. \n"
              "4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected\nnode to be the current node. \n"
              "5. Then set the current node as visited.\n"
              "6. Repeat steps 2 to 5 until all nodes are set to visited.\n"
              "To find the shortest path through backtracking:\n"
              "7. Start from the goal node.\n"
              "8. Add the ‘previous node’ to the start of a list.\n"
              "9. Repeat step 7 until the start node is reached.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=1, column=0, rowspan=3, columnspan=16, sticky="w")

        Label(self.dijkstraFrame,
              text='''In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.
Then choose a\nstart vertex from the list and visualise Dijkstra's algorithm on the graph.\n''',
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=4, column=0, rowspan=2, columnspan=16, sticky="w")
```

```
        self.graphInputButton = Button(self.dijkstraFrame, text="Input a graph", bg="#1b263b", fg="white", font=fontMedium, width=20, command=self.graphInputWindow)
        self.graphInputButton.grid(row=6, column=1, columnspan=2)

        # start vertex:
        Label(self.dijkstraFrame, text="Start vertex:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=12, column=0, columnspan=2)

        self.startVertexOptions = ["Select"] # initially Select - changes to the list of vertices once graph input
        self.startVertexChoice = StringVar(self.dijkstraFrame)
        self.startVertexChoice.set("Select")

        self.startVertexDropdown = OptionMenu(self.dijkstraFrame, self.startVertexChoice, *self.startVertexOptions)
        self.startVertexDropdown.grid(row=12, column=2, columnspan=2)
        self.startVertexDropdown.config(font=fontSmall, bg="#eaebed", fg="#1b263b")
        self.startVertexDropdown["menu"].config(font=fontSmall, bg="#eaebed", fg="#1b263b")

        #end vertex:
        Label(self.dijkstraFrame, text="End vertex:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=13, column=0, columnspan=2)

        self.endVertexOptions = ["Select"] # initially Select - changes to the list of vertices once graph input
        self.endVertexChoice = StringVar(self.dijkstraFrame)
        self.endVertexChoice.set("Select")

        self.endVertexDropdown = OptionMenu(self.dijkstraFrame, self.endVertexChoice, *self.endVertexOptions)
        self.endVertexDropdown.grid(row=13, column=2, columnspan=2)
        self.endVertexDropdown.config(font=fontSmall, bg="#eaebed", fg="#1b263b")
        self.endVertexDropdown["menu"].config(font=fontSmall, bg="#eaebed", fg="#1b263b")

        Button(self.dijkstraFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=12, column=5, columnspan=3, padx=10, pady=5)

        self.invalidMessage = Label(self.dijkstraFrame, text="", font=fontMedium, bg="#eaebed", fg="#990000")
        self.invalidMessage.grid(row=12, column=8, columnspan=8)
```

```
    def graphInputWindow(self):
        print("in start graph input window")
        self.hideDijkstraWidgets()
        self.hideMenuWidgets()
        self.graphInputPageWidgets()

    def hideDijkstraWidgets(self):
        print("in hide dijkstra widgets")
        for widget in self.dijkstraFrame.winfo_children():
            widget.grid_forget()
        self.title.grid_forget()
```

```
    def validate(self):
        print("in validate")
        # checks start and end vertex entered
        if self.startVertexChoice.get() == "Select": # select is default option where there is no vertex selected
            self.invalidMessage["text"] = "Invalid: please enter a start vertex."
            # start vertex not chosen
            #print("invalid start")
            return
        if self.endVertexChoice.get() == "Select":
            #print("invalid end")
            self.invalidMessage["text"] = "Invalid: please enter an end vertex."
            return
        if self.startVertexChoice.get() == self.endVertexChoice.get():
            #print("invalid same")
            self.invalidMessage["text"] = "Invalid: please enter a start and end vertex that are different."
            return
        #print("valid")
        self.invalidMessage["text"] = ""
        self.dijkstraSteps()
        self.dijkstraAnimate()
```

```
    def dijkstraExtraWidgets(self):
        print("in dijkstra extra widgets")
        self.master.geometry("905x880")

        self.menuWidgets()
        self.menuText["text"] = "Use right/left arrow keys to step forward/back through\nthe visualisation when paused."
        # destroy Graph Input button
        self.graphInputButton.destroy()

        # start vertex choices:
        self.startVertexOptions = list(self.graph.nodes)
        self.startVertexChoice.set(self.startVertexOptions[0] if self.startVertexOptions else "Select")
        self.startVertexDropdown["menu"].delete(0, "end")
        for vertex in self.startVertexOptions:
            self.startVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.startVertexChoice.set(v))

        # end vertex choices:
        self.endVertexOptions = list(self.graph.nodes)
        self.endVertexChoice.set(self.startVertexOptions[0] if self.startVertexOptions else "Select")
        self.endVertexDropdown["menu"].delete(0, "end")
        for vertex in self.endVertexOptions:
            self.endVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.endVertexChoice.set(v))


        # create axes to plot graph on
        self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
        self.dijkstraGraphCanvas = FigureCanvasTkAgg(self.fig, self.dijkstraFrame)
        self.dijkstraGraphCanvas.get_tk_widget().grid(row=6, column=0, rowspan=6, columnspan=6)
        # draw graph on axes
        pos = nx.get_node_attributes(self.graph, 'pos')
        nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
        edge_labels = nx.get_edge_attributes(self.graph, 'weight')
        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
        self.dijkstraGraphCanvas.draw()
```

```
        # create frame to hold the table
        self.tableFrame = Frame(self.dijkstraFrame, bg="#eaebed")
        self.tableFrame.grid(row=6, column=8, rowspan=6, columnspan=8)

        headers = ["Vertex", "Shortest Distance", "Previous Vertex", "Visited"]
        for column, text in enumerate(headers): # enumerate returns index and value
            label = Label(self.tableFrame, text=text, font=fontMedium, borderwidth=1, relief="solid", padx=5, pady=5)
            if text == "Vertex":
                label.grid(row=6, column=8, columnspan=2)
            else:
                label.grid(row=6, column=(8+(2*column)), sticky="nsew", columnspan=2)

        # fill in vertex values in distance table
        self.tableData = {}
        for row, vertex in enumerate(self.graph.nodes(), start=1):
            self.tableData[vertex] = {}
            # vertex name:
            vLabel = Label(self.tableFrame, text=vertex, font=fontMedium, borderwidth=1, relief="solid", padx=5, pady=5)
            vLabel.grid(row=(6+row), column=8, sticky="nsew", columnspan=2)
            self.tableData[vertex]["vertex"] = vLabel
            # shortest distance: initialise infinity except start:
            sdLabel = Label(self.tableFrame, text="∞", font=fontMedium, borderwidth=1, relief="solid", padx=5, pady=5)
            sdLabel.grid(row=(6+row), column=10, sticky="nsew", columnspan=2)
            self.tableData[vertex]["shortest distance"] = sdLabel
            # previous vertex: initialise to -
            pvLabel = Label(self.tableFrame, text="-", font=fontMedium, borderwidth=1, relief="solid", padx=5, pady=5)
            pvLabel.grid(row=(6+row), column=12, sticky="nsew", columnspan=2)
            self.tableData[vertex]["previous vertex"] = pvLabel
            # visited: iniitalise empty
            visitedLabel = Label(self.tableFrame, text="", font=fontMedium, borderwidth=1, relief="solid", padx=5, pady=5)
            visitedLabel.grid(row=(6+row), column=14, sticky="nsew", columnspan=2)
            self.tableData[vertex]["visited"] = visitedLabel
```

```
    def updateDijkstraTable(self, step):
        visitedNodes, currentNode, checkingEdges, distances, previous = step
        for vertex in self.graph.nodes():
            # update shortest distance:
            # if the vertex already has a shortest distance, change to that, otherwis infinity
            self.tableData[vertex]["shortest distance"]["text"] = (str(distances[vertex]) if distances[vertex] != float("inf") else "∞")
            # update previous vertex
            # if the vertex already has a previous vertex, change to that, otherwise -
            self.tableData[vertex]["previous vertex"]["text"] = (previous[vertex] if previous[vertex] else "-")
            # visited ndoes stay marked
            self.tableData[vertex]["visited"]["text"] = ("✓" if vertex in visitedNodes else "")
```

```
200     def openHome(self):
201         closeOpen(self.master, "home", self.username)
202 
203 
204     def updateDijkstraGraph(self, visitedNodes, currentNode=None, checkingEdges=[]):
205         print("in udpate dijkstra graph")
206         self.ax.clear()
207         pos = nx.get_node_attributes(self.graph, "pos")
208 
209         # draw base graph:
210         nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
211         edge_labels = nx.get_edge_attributes(self.graph, "weight")
212         nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
213 
214         # highlighted visited nodes:
215         nx.draw_networkx_nodes(self.graph, pos, nodelist=visitedNodes, node_color="#eca400", ax=self.ax)
216 
217         # highlight current node:
218         if currentNode:
219             nx.draw_networkx_nodes(self.graph, pos, nodelist=[currentNode], node_color="#b2675e", ax=self.ax)
220 
221         # highlight edges being checked:
222         nx.draw_networkx_edges(self.graph, pos, edgelist=checkingEdges, edge_color="#b2675e", width=2, ax=self.ax)
223 
224         self.dijkstraGraphCanvas.draw()
```

```
239     def dijkstraSteps(self, startV=None, endV=None):
240         print("in dijkstra steps")
241         if startV:
242             start = startV
243         else:
244             start = self.startVertexChoice.get()
245 
246         if endV:
247             end = endV
248         else:
249             end = self.endVertexChoice.get()
250 
251         self.distances = {node: float("inf") for node in self.graph.nodes}
252         self.previous = {node: None for node in self.graph.nodes}
253         self.distances[start] = 0
254 
255         self.steps = []
256         queue = [(0, start)] # this is a priority queue for storing (distance, vertex)
257         visited = set()
```

```
        while queue:
            currentDistance, currentNode = heapq.heappop(queue)
            if currentNode in visited:
                # continue = go to process next vertex by going to the next iteration of while queue
                # this maintains Dijkstra's O((v+E)logV) complexity
                continue
            visited.add(currentNode)

            # store step for animation:
            checkingEdges = [(currentNode, neighbour) for neighbour in self.graph.neighbors(currentNode) if neighbour not in visited]
            self.steps.append((visited.copy(), currentNode, checkingEdges.copy(), dict(self.distances), dict(self.previous)))

            # check for shortest connected edge/node
            for neighbour in self.graph.neighbors(currentNode):
                if neighbour in visited:
                    continue
                edgeWeight = self.graph[currentNode][neighbour]["weight"]
                newDistance = currentDistance + edgeWeight
                if newDistance < self.distances[neighbour]:
                    self.distances[neighbour] = newDistance
                    self.previous[neighbour] = currentNode
                    heapq.heappush(queue, (newDistance, neighbour))

        # backtrack for shortest path:
        path = []
        current = end
        while current:
            path.append(current)
            current = self.previous[current]
        path.reverse()
        self.steps.append((("shortest path"), path))
```

```
    def dijkstraAnimate(self):
        print("in dijkstra animate")
        step = self.steps[self.currentStep]
        if not self.isPaused and step[0]=="shortest path":
            # step[1] contains the shortest path as a list of vertices
            # therefore adding the combos of edges into the list path_edges
            # e.g. if step[1] = [A, B, C] then path_edges = [(A,B), (B,C)]
            pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1]) -1)]
            self.updateDijkstraGraph([], checkingEdges=pathEdges)
            shortestPathText = " -> ".join(step[1])
            self.invalidMessage["text"] = f'''Shortest path between {self.startVertexChoice.get()} and
{self.endVertexChoice.get()} is {shortestPathText} \nWeight: {self.distances[self.endVertexChoice.get()]}'''

        elif not self.isPaused and self.currentStep < len(self.steps):
            visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
            self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
            # update table
            self.updateDijkstraTable(step)
            self.currentStep += 1
            self.master.after(750, self.dijkstraAnimate)
```

```
# for testing ------------------------------------------------------------------------------
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Dijkstra's test")
    #root.geometry("800x595")
    graph_input = Dijkstra(root, "HelloWord!!!") #HellowWord!!! username for testing purposes
    root.mainloop()
```

simplex_page :

```
# imports specific to the Simplex page
from tkinter import *
import tkinter as tk
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
from menu_code import Menu
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from fractions import Fraction

class Simplex(Menu):
    def __init__(self, master, pUsername, show_menu=True):
        Menu.__init__(self, master, 12, 14)
        self.username = pUsername
        self.textConstraints = [] # for widgets
        self.valueConstraints = []
        self.valueObjective = []
        self.tableau = np.array([], dtype=float) # empty numpy array
        self.canvas = None
        self.currentStep = 0
        self.simplexPath = [(0,0)]
        self.simplexWidgets()
        if show_menu:
            self.menuWidgets()
            self.menuText["text"] = "Use right/left arrow keys to step forward/back\nthrough the visualisation when paused."
        self.invalid = False
```

```
    def addConstraint(self, row):
        #print("in add constraint")
        if row == 6:
            self.constraintButton.destroy()
            Label(self.simplexFrame, text="Constraint 3: ", font=fontSmall, bg="#eaebed", fg="#eca400").grid(row=6, column=8)

        xEntry = Entry(self.simplexFrame, width=5)
        xEntry.grid(row=row, column=9)
        Label(self.simplexFrame, text=" x + ", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=row, column=10)
        yEntry = Entry(self.simplexFrame, width=5)
        yEntry.grid(row=row, column=11)
        Label(self.simplexFrame, text=" y ≤ ", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=row, column=12)
        rhsEntry = Entry(self.simplexFrame, width=5)
        rhsEntry.grid(row=row, column=13)

        self.textConstraints.append({
            "x": xEntry,
            "y": yEntry,
            "rhs": rhsEntry})
```

```
    def simplexWidgets(self):
        self.master.configure(bg="#eaebed")
        self.master.geometry("845x470")
        self.simplexFrame = Frame(self.master, bg="#eaebed")
        self.simplexFrame.grid(row=0, column=0, columnspan=14, rowspan=12)

        self.title = Label(self.simplexFrame, text="Simplex Algorithm Visualisation", font=fontLarge, bg="#eaebed", fg="#1b263b")
        self.title.grid(row=0, column=0, columnspan=12) # change columnspan
        Button(self.simplexFrame, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=12, columnspan=2)

        Label(self.simplexFrame, text=
              "1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function\nto be equal to 0. Input these equations into the simplex tableau.\n"
              "2. Choose the most negative value in the objective row to become the basis.\n"
              "3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot.\n"
              "4. Divide the pivot row by the value of the pivot.\n"
              "5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.\n"
              "6. Repeat until the objective row contains no negative entries.\n"
              "7. Read off the optimal solution from the tableau,",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=1, column=0, rowspan=3, columnspan=14, sticky="w")

        # constraint 1:
        Label(self.simplexFrame, text="Constraint 1:", font=fontSmall, bg="#eaebed", fg="#606c38").grid(row=4, column=8)
        self.addConstraint(4)
        # constraint 2:
        Label(self.simplexFrame, text="Constraint 2:", font=fontSmall, bg="#eaebed", fg="#b2675e").grid(row=5, column=8)
        self.addConstraint(5)
        # constraint 3 button:
        self.constraintButton = Button(self.simplexFrame, text="Add constraint", command=lambda: self.addConstraint(6), bg="#1b263b", fg="white", font=fontMedium, width=15)
        self.constraintButton.grid(row=6, column=9, columnspan=3)
```

```
         # objective:
         Label(self.simplexFrame, text="Objective function:", font=fontMedium, bg="#eaebed", fg="#1b263b").grid(row=7, column=8, columnspan=4)
         Label(self.simplexFrame, text="P =", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=8, column=8)
         xEntry = Entry(self.simplexFrame, width=5)
         xEntry.grid(row=8, column=9)
         Label(self.simplexFrame, text=" x + ", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=8, column=10)
         yEntry = Entry(self.simplexFrame, width=5)
         yEntry.grid(row=8, column=11)
         Label(self.simplexFrame, text=" y", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=8, column=12)
         self.objectiveText = {"x": xEntry,
                               "y": yEntry}

         # visualise:
         Button(self.simplexFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=9, column=10, columnspan=3)
         # invalid:
         self.invalidMessage = Label(self.simplexFrame, text="", font=fontMedium, bg="#eaebed", fg="#990000")
         self.invalidMessage.grid(row=10, column=8, columnspan=6, rowspan=2)

     def openHome(self):
         closeOpen(self.master, "home", self.username)
```

```
    def validate(self):
        #print("in validate")
        # validate constriants
        # reset lists before revalidating
        self.valueConstraints = []
        self.valueObjective = []
        for constraint in self.textConstraints:
            try:
                x = float(constraint["x"].get())
                y = float(constraint["y"].get())
                rhs = float(constraint["rhs"].get())
                if x == 0.0 and y == 0.0:
                    raise ValueError
                if x < -10.0 or x > 10.0:
                    raise ValueError
                if y < -10.0 or y > 10.0:
                    raise ValueError
                if rhs < -10.0 or rhs > 10.0:
                    raise ValueError
                self.valueConstraints.append([x,y,rhs])
                print(x, y, rhs)
            except ValueError:
                self.invalidMessage["text"] = '''Invalid constraint entry: please ensure that all numbers entered\n
are between -10 and 10, with the x\nand y coefficients not both being 0.'''
                self.valueConstraints = []
                return
        #print("constraints valid")
        # validate objective
        try:
            x = float(self.objectiveText["x"].get())
            y = float(self.objectiveText["y"].get())
            if x == 0.0 and y == 0.0:
                raise ValueError
            if x < -10.0 or x > 10.0:
                raise ValueError
            if y < -10.0 or y > 10.0:
                raise ValueError
            #print("objective valid")
            self.valueObjective.append(x)
            self.valueObjective.append(y)
            #print(x, y)
```

```
        except ValueError:
            self.invalidMessage["text"] = '''Invalid objective entry: please ensure that all numbers entered\n
are between -10 and 0, with the x\nand y coefficients not both being 0.'''
            self.valueObjective = []
            return
        # print("fully valid")

        '''
        if self.checkFeasible() == False:
            # no feasible region
            self.invalidMessage["text"] = "Invalid: no feasible region."
            return'''

        # all validation passed
        self.invalidMessage["text"] = ""
        self.createTableau()
        self.plotConstraints()
        self.master.geometry("1005x858")
        self.simplexSteps()
        if self.invalid == False:
            self.simplexAnimate()
```

```
    def createTableau(self):
        #print("in create tableau")
        numSlackVars = len(self.valueConstraints)
        for i, constraint in enumerate(self.valueConstraints):
            row = constraint[:-1] # take all coefficients except rhs
            row += [1 if j==i else 0 for j in range(numSlackVars)] # add slack variable
            row.append(constraint[-1]) # append rhs value
            if self.tableau.size == 0:
                self.tableau = np.array([row])
            else:
                self.tableau = np.vstack([self.tableau, row])
        # add objective with -ve coefficients as maximising
        objectiveRow = [-c for c in self.valueObjective] + [0]*(numSlackVars +1)
        self.tableau = np.vstack([self.tableau, objectiveRow])
        # check
        #print("initial tableau:")
        #print(self.tableau)
```

```
    def plotConstraints(self):
        colours = ["#606c38", "#b2675e", "#eca400"]

        self.fig, self.ax = plt.subplots(figsize=(5,5))
        for constraint in self.valueConstraints:
            a, b, c = constraint
            #print(a, b, c)
            i = self.valueConstraints.index(constraint)
            xVals = np.linspace(0, 10, 200) # used to plot constriant lines on graph
            if b != 0: # is a line in the form ax + by = c
                yVals = (c - a*xVals) / b
            else: # in the form ax = c
                xVals = np.full_like(xVals, c / a)
                yVals = np.linspace(0, 10, 200)
            self.ax.plot(xVals, yVals, label=f"Constraint {i+1}", color=colours[i])

        self.ax.set_xlim(0,10)
        self.ax.set_ylim(0,10)
        self.ax.axhline(0, color="black", linewidth=1)
        self.ax.axvline(0, color="black", linewidth=1)
        self.ax.set_xlabel("x")
        self.ax.set_ylabel("y")
        self.ax.legend()
        self.ax.grid()

        self.canvas = FigureCanvasTkAgg(self.fig, master=self.simplexFrame)
        self.canvas.get_tk_widget().grid(row=4, column=0, rowspan=8, columnspan=8)
        self.canvas.draw()
```

```
    def isOptimal(self):
        #print("in is optimal")
        # checks all values in objective row except rhs are non-negative
        return np.all(self.tableau[-1, :-1] >= -1e-9)


    def findPivotColumn(self):
        # find most negative coef in objective row
        return np.argmin(self.tableau[-1, :-1])


    def findPivotRow(self, pivotColumn):
        # find row by doing rhs/pivot column value - least positive
        ratios = []
        for i in range(len(self.valueConstraints)):
            if self.tableau[i, pivotColumn] > 0:
                ratios.append(self.tableau[i, -1] / self.tableau[i, pivotColumn])
            else:
                ratios.append(float("inf")) # ignores negative or 0 values

        pivotRow = np.argmin(ratios)
        if ratios[pivotRow] == float("inf"):
            #print("No valid pivot row found. solution unbounded.")
            return -1
        return pivotRow
```

```
    def pivot(self, row, column):
        # perform row op to change values by Gaussian elimination style row reduction
        pivotValue = self.tableau[row, column]

        # 1: make pivot element 1
        self.tableau[row] = self.tableau[row] / pivotValue

        # 2: row reduction
        for i in range(len(self.tableau)):
            if i != row:
                multiplier = self.tableau[i, column]
                self.tableau[i] -= multiplier * self.tableau[row]
        #print("Updated tableau after pivot:")
        #print("Tableau after pivot")
        #print(self.tableau)


    def getVariableValue(self, varIndex):
        # find row where variable is basic
        column = self.tableau[:, varIndex] # selects all rows in varIndex column
        basicRow = -1

        for i in range(len(column) -1): # ignores last row as this is the objective function
            if abs(column[i]-1) < 1e-9: # avoids rounding error of 0.999999999
                # ensure all other coefficients in row are 0
                if np.count_nonzero(column)==1:
                    basicRow = i
                    break
        #print("basic row is:", basicRow)
        # variable is basic returns value in rightmost column, variable not basic returns 0
        return self.tableau[basicRow, -1] if basicRow != -1 else 0
```

```
     def updateSimplexGraph(self):
         #print("in update simplex graph")
         self.ax.clear()

         # replot constraints
         self.plotConstraints()

         # draw path of visited points
         if self.currentStep > 0:
             xPath, yPath = zip(*self.simplexPath[:self.currentStep +1])
             self.ax.plot(xPath, yPath, "ro-", linewidth=3, markersize=5)
         self.canvas.draw()
```

```
    def simplexSteps(self):
        #print("in simplex steps")
        self.steps = [(0,0,0)]
        x,y,P = 0,0,0
        while not self.isOptimal():
            #print("in new step loop")
            pivotColumn = self.findPivotColumn()
            #print("pivot column:", pivotColumn)
            pivotRow = self.findPivotRow(pivotColumn)
            #print("pivot row:", pivotRow)

            # check for unbounded:
            if self.tableau[pivotRow, pivotColumn] <= 0 :
                self.invalidMessage["text"] = "Unbounded solution."
                self.invalid = True
                return
            self.invalid = False

            #print("Tableau before pivot:")
            #print(self.tableau)

            self.pivot(pivotRow, pivotColumn)

            # steps append ...
            x = Fraction(self.getVariableValue(0)).limit_denominator() # x is in tableau column 0
            y = Fraction(self.getVariableValue(1)).limit_denominator() # y is in column index 1
            P = Fraction(self.tableau[-1][-1]).limit_denominator()
            #print("At this step:")
            #print("x, y, P:", x, y, P)
            self.steps.append((x,y,P))
            self.simplexPath.append((x,y))

        #print("tableau is optimal")
        self.steps.append((x,y,P))
        #print("steps:")
        #print(self.steps)
```

```
    def simplexAnimate(self):
        #print("in simplex animate")
        if self.currentStep < len(self.steps)-1 and not self.isPaused:
            self.updateSimplexGraph()
            self.currentStep += 1
            self.master.after(1000, self.simplexAnimate) # change time for smoothness
        elif self.currentStep == len(self.steps)-1:
            self.updateSimplexGraph()
            self.invalidMessage["text"] = f"Optimal solution P = {self.steps[-1][2]}\nx = {self.steps[-1][0]}, y = {self.steps[-1][1]}"

# for testing ------------------------------------------------------------------------
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Simplex test")
    #root.geometry("800x595")
    graph_input = Simplex(root, "HelloWord!!!") #HellowWord!!! username for testing purposes
    root.mainloop()
```

graph_input.py :

```
from tkinter import *
import tkinter as tk
from tkinter import simpledialog
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from NEA_utilities import fontLarge, fontMedium, fontSmall


class GraphInput:
    def __init__(self, master):
        self.master = master
        self.graph = nx.Graph()
        self.validated = False
        self.selected_vertex = None
        #self.target_vertex = None
        self.selected_edge = None
```

```
    def graphInputPageWidgets(self):
        #self.graphInputFrame = Frame(self.master, bg="#eaebed")
        self.master.configure(bg="#eaebed")
        self.master.geometry("1200x815")


        Label(self.master, text="Input a Graph", font=fontLarge, bg="#eaebed", fg="#1b263b").grid(row=0, column=0, columnspan=10)

        Label(self.master,
              text="  • Double click on the canvas to add a vertex. Enter a unique identifier.\n"
                   "  • Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.\n"
                   "  • Click on an element once and press delete to remove it.\n"
                   " When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit",
              font = fontMedium,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=1, column=0, columnspan=10, sticky="w")

        Label(self.master, text="Canvas:", font=fontMedium, bg="#eaebed").grid(row=2, column=0, columnspan=2)
        Label(self.master, text="Graph:", font=fontMedium, bg="#eaebed").grid(row=2, column=5, columnspan=2)

        self.canvasFrame = Frame(self.master, bg="#eaebed")
        self.canvasFrame.grid(row=3, column=0)

        self.canvas = Canvas(self.canvasFrame, width=600, height=600, bg="white")
        self.canvas.grid(row=3, column=0, columnspan=5, rowspan=5)

        self.fig, self.ax = plt.subplots(figsize=(5,5))
        self.graphCanvas = FigureCanvasTkAgg(self.fig, self.master)
        self.graphCanvas.get_tk_widget().grid(row=3, column=5, columnspan=5, rowspan=5)

        self.invalidMessage = Label(self.master, text="", font=fontMedium, bg="#eaebed", fg="#990000")
        self.invalidMessage.grid(row=8, column=0, columnspan=8)
```

```
54          Button(self.master, text="Submit", font=fontMedium, fg="white", bg="#1b263b", command=self.validateGraph).grid(row=8, column=8, columnspan=2)
55
56          self.canvas.focus_set()
57          # set up the keyboard/user clicks/inputs:
58          self.canvas.bind("<Double-Button-1>", self.add_vertex)
59          self.canvas.bind("<Button-1>", self.select_or_create_edge)
60          self.canvas.bind("<Delete>", self.delete_element)
61
62
63      def validateGraph(self):
64          #print("in validate graph")
65          # check for 4 to 10 vertices:
66          if len(self.graph.nodes) < 4 or len(self.graph.nodes) > 10:
67              self.invalidMessage["text"] = "Invalid graph: Ensure that the graph has between 4 and 10 vertices."
68              self.validated = False
69              return
70
71          # check each vertex has at least 1 edge to it
72          for vertex in self.graph.nodes:
73              if len(list(self.graph.neighbors(vertex))) == 0:
74                  self.invalidMessage["text"] = "Invalid graph: Ensure that each vertex has at least 1 edge connected to it."
75                  self.validated = False
76                  return
77
78          # check weight edges greater than 0
79          for start, end, data in self.graph.edges(data=True):
80              if data.get("weight", 0) <= 0:
81                  self.invalidMessage["text"] = "Invalid graph: Ensure that the weight of each edge is greater than 0."
82                  self.validated = False
83                  return
84
85          # check connected:
86          if not nx.is_connected(self.graph):
87              self.invalidMessage["text"] = "Invalid graph: Ensure that the graph is fully connected."
88              self.validated = False
89              return
90
91          self.invalidMessage["text"] = " "
92          self.validated = True
93          self.backToPage()
```

```
    def backToPage(self):
        if self.validated == True:
            self.graphInputHideWidgets()
            if hasattr(self, "primWidgets"):
                self.primWidgets()
                self.primExtraWidgets()
            elif hasattr(self, "dijkstraWidgets"):
                self.dijkstraWidgets()
                self.dijkstraExtraWidgets()


    def add_vertex(self, event):
        #print("in add vertex")
        if len(self.graph.nodes) < 10:
            vertex_id = simpledialog.askstring("Vertex Input", "Enter a unique vertex identifier (1 character):")
            if vertex_id and len(vertex_id) == 1 and vertex_id not in self.graph.nodes and vertex_id != '"':
                x, y = event.x, event.y
                self.graph.add_node(vertex_id, pos=(x, y))

                self.canvas.create_oval(x-10, y-10, x+10, y+10, fill='blue', outline='black')
                self.canvas.create_text(x, y, text=vertex_id, fill='white')

                #print(f"Added vertex: {vertex_id} at position ({x}, {y})")
                self.update_visualization()
```

```
    def select_or_create_edge(self, event):
        #print("in select or create edge")
        for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
            if abs(event.x - x) < 10 and abs(event.y - y) < 10:
                if self.selected_vertex is None:
                    self.selected_vertex = node
                    #print(f"selected vertex: {node}")
                elif self.selected_vertex != node:
                    self.target_vertex = node
                    weight = simpledialog.askinteger("Edge Weight", "Enter a positive integer weight:")
                    if weight and weight > 0:
                        self.graph.add_edge(self.selected_vertex, self.target_vertex, weight=weight)
                        # Draw the edge visually on the canvas
                        self.canvas.create_line(
                            self.graph.nodes[self.selected_vertex]['pos'][0],
                            self.graph.nodes[self.selected_vertex]['pos'][1],
                            self.graph.nodes[self.target_vertex]['pos'][0],
                            self.graph.nodes[self.target_vertex]['pos'][1],
                            fill="black", width=2, tags=(self.selected_vertex, self.target_vertex)
                        )
                        #print(f"Created edge between {self.selected_vertex} and {self.target_vertex} with weight {weight}")
                        self.update_visualization()
                    self.selected_vertex = None
                    self.target_vertex = None
                return
        # Check if the click is on an edge
        for edge in self.graph.edges:
            x1, y1 = self.graph.nodes[edge[0]]['pos']
            x2, y2 = self.graph.nodes[edge[1]]['pos']
            if min(x1, x2) <= event.x <= max(x1, x2) and min(y1, y2) <= event.y <= max(y1, y2):
                self.selected_edge = edge
                #print(f"Selected edge: {self.selected_edge} for deletion")
                return

        self.selected_vertex = None
        self.target_vertex = None
```

```
    def delete_element(self, event):
        #print("in delete element")
        if self.selected_vertex is not None:
            #print(f"Selected vertex: {self.selected_vertex} for deletion")
            for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
                #print("in for")
                #print(f"node to delete: {node}")
                #print(f"node position : {(x,y)}")
                #print(f"event position: ({event.x},{event.y})")
                #print(f"Difference: ({abs(event.x - x)},{abs(event.y - y)})")
                if abs(event.x - x) < 500 and abs(event.y - y) < 500:
                    #print("in if")
                    self.graph.remove_node(node)
                    # Remove the vertex from the canvas
                    self.canvas.delete(node)

                    #print(f"Deleted vertex: {node}")
                    self.update_visualization()
                    self.selected_vertex = None
                    return
        if self.selected_edge is not None:
            #print(f"Selected edge: {self.selected_edge} for deletion")
            for edge in list(self.graph.edges):
                x1, y1 = self.graph.nodes[edge[0]]['pos']
                x2, y2 = self.graph.nodes[edge[1]]['pos']

                if min(x1, x2) <= event.x <= max(x1, x2) and min(y1, y2) <= event.y <= max(y1, y2):
                    self.graph.remove_edge(*self.selected_edge)
                    # Remove the edge from the canvas
                    edge_tag = self.selected_edge
                    self.canvas.delete(edge_tag)
                    #self.canvas.delete(edge)
                    #print(f"Deleted edge between {edge[0]} and {edge[1]}")
                    self.update_visualization()
                    self.selected_edge = None
                    return
```

```
    def update_visualization(self):
        #print("in update visualisation")
        self.ax.clear()
        pos = nx.get_node_attributes(self.graph, 'pos')
        nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
        edge_labels = nx.get_edge_attributes(self.graph, 'weight')
        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
        self.graphCanvas.draw()
        self.redraw_canvas()


    def redraw_canvas(self):
        #print("Redrawing canvas...")
        self.canvas.delete("all")  # Clear the canvas

        # Redraw vertices
        for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
            self.canvas.create_oval(x-10, y-10, x+10, y+10, fill="#606c38", outline='black', tags=node)
            self.canvas.create_text(x, y, text=node, fill='white', tags=node)

        # Redraw edges
        for edge in self.graph.edges:
            x1, y1 = self.graph.nodes[edge[0]]['pos']
            x2, y2 = self.graph.nodes[edge[1]]['pos']
            edge_tag = (edge[0], edge[1])
            self.canvas.create_line(x1, y1, x2, y2, fill="black", width=2, tags=edge_tag)


    def graphInputHideWidgets(self):
        for widget in self.master.winfo_children():
            widget.grid_forget()
```

menu_code.py :

```
# imports specific to the Menu class
from tkinter import *
import tkinter as tk
from NEA_utilities import fontLarge, fontMedium, fontSmall


class Menu:
    def __init__(self, master, rowCoord, columnsLength):
        self.rowStart = rowCoord # this is the y coordinate of where the menu starts for the current page
        self.menuLength = columnsLength # this is the rowspan of the menu based on the current page
        self.menuFrame = None

        self.master = master
        self.currentStep = 0
        self.isPaused = False
```

```
    def menuWidgets(self):
        #print("in menuWidgets")

        self.menuFrame = Frame(self.master, bg="#1b263b")
        #self.menuFrame.grid(row=self.rowStart, column = 0, columnspan = self.menuLength, rowspan=3)
        self.menuFrame.grid(row=self.rowStart, column=0, columnspan=self.menuLength, rowspan=3, sticky="ew")

        Button(self.menuFrame, text="Skip Back", command=self.skipBack, fg="#1b263b", bg="#eaebed",
               font=fontMedium, width=15).grid(row=1, column= 1, padx=10, pady=10)
        self.playPauseButton = Button(self.menuFrame, text="Pause", command=self.togglePlayPause,
                                      fg="#1b263b", bg="#eaebed", font=fontMedium, width=15)
        self.playPauseButton.grid(row=1, column=3, padx=5, pady=5)
        Button(self.menuFrame, text="Skip Forward", command=self.skipForward, bg="#eaebed",
               fg="#1b263b", font=fontMedium, width=15).grid(row=1, column=5, padx=10, pady=10)
        # add text describing arrow stuff
        self.menuText = Label(self.menuFrame, text="Use right/left arrow keys to step forward/back through the visualisation when paused.",
                              font=fontSmall, fg="#eaebed", bg="#1b263b")
        self.menuText.grid(row=1, column=7, columnspan=12, padx=5)

        # these are for the keyboard inputs:
        self.master.bind("<Left>", lambda e: self.stepBack())
        self.master.bind("<Right>", lambda e: self.stepForward())
```

```
    def skipForward(self):
        self.currentStep = len(self.steps) - 1 # gets the last step index
        self.isPaused = True
        if hasattr(self, "updateChart"): # note that methods count as attributes
            self.updateChart(self.steps[self.currentStep])

        elif hasattr(self, "updatePrimGraph"):
            self.updatePrimGraph(self.steps[self.currentStep])

        elif hasattr(self, "updateDijkstraGraph"):
            print("in dijkstra skip forward")
            step = self.steps[self.currentStep]
            pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1]) -1)]
            self.updateDijkstraGraph([], checkingEdges=pathEdges)
            shortestPathText = " -> ".join(step[1])
            self.invalidMessage["text"] = f'''Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice.get()}
is {shortestPathText} \nWeight: {self.distances[self.endVertexChoice.get()]}'''
            step1 = self.steps[self.currentStep-1] # so that the vertex table is actually filled in
            self.updateDijkstraTable(step1)

        elif hasattr(self, "updateSimplexGraph"):
            self.updateSimplexGraph()
            self.invalidMessage["text"] = f"Optimal solution P = {self.steps[-1][2]}\nx = {self.steps[-1][0]}, y = {self.steps[-1][1]}"
```

```
40      def skipBack(self):
41          self.currentStep = 0
42          self.isPaused = True
43
44          if hasattr(self, "updateChart"): # note that methods count as attributes
45              self.updateChart(self.steps[self.currentStep])
46
47          elif hasattr(self, "updatePrimGraph"):
48              self.updatePrimGraph(self.steps[self.currentStep])
49
50          elif hasattr(self, "updateDijkstraGraph"):
51              print("in dijkstra skip back")
52              step = self.steps[self.currentStep]
53              visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
54              self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
55              self.updateDijkstraTable(step)
56
57          elif hasattr(self, "updateSimplexGraph"):
58              self.updateSimplexGraph()
59
60          self.invalidMessage["text"] = ""
```

```
87      def togglePlayPause(self):
88          print("in toggle play pause")
89          self.isPaused = not self.isPaused # switches the state
90          self.playPauseButton.config(text="Pause" if not self.isPaused else "Play") # switches the text
91          if not self.isPaused: # ie displays Pause but the visualisation is playing
92              print("in not self.isPaused")
93              #self.playAnimation()
94              if hasattr(self, "bubbleSortAnimate"):
95                  self.bubbleSortAnimate()
96
97              elif hasattr(self, "primAnimate"):
98                  self.primAnimate()
99
100             elif hasattr(self, "dijkstraAnimate"):
101                 #print("in dijkstra toggle play pause")
102                 self.dijkstraAnimate()
103
104             elif hasattr(self, "simplexAnimate"):
105                 self.simplexAnimate()
```

```
108     def stepBack(self):
109         if self.isPaused and self.currentStep > 0:
110             self.currentStep -= 1
111             if hasattr(self, "updateChart"): # note that methods count as attributes
112                 numbers = self.steps[self.currentStep]
113                 indices = self.stepsDictionary[numbers]
114                 self.updateChart(numbers, indices)
115
116             elif hasattr(self, "updatePrimGraph"):
117                 mstStep = self.steps[self.currentStep]
118                 self.updatePrimGraph(mstStep)
119                 self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}{v}," for u,v in mstStep]) + "\n")
120                 self.visualiseText.see(tk.END)
121
122             elif hasattr(self, "updateDijkstraGraph"):
123                 print("in dijkstra step back")
124                 step = self.steps[self.currentStep]
125                 if isinstance(step[0], set): # check not shortest path
126                     visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
127                     self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
128                     self.updateDijkstraTable(step)
129
130             elif hasattr(self, "updateSimplexGraph"):
131                 self.updateSimplexGraph()
132                 self.invalidMessage["text"] = ""
```

```
    def stepForward(self):
        if self.isPaused and self.currentStep < len(self.steps)-1:
            self.currentStep += 1
            if hasattr(self, "updateChart"): # note that method count as attributes
                numbers = self.steps[self.currentStep]
                indices = self.stepsDictionary[numbers]
                self.updateChart(numbers, indices)

            elif hasattr(self, "updatePrimGraph"):
                mstStep = self.steps[self.currentStep]
                self.updatePrimGraph(mstStep)
                self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}{v}," for u,v in mstStep]) + "\n")
                self.visualiseText.see(tk.END)

            elif hasattr(self, "updateDijkstraGraph"):
                print("in dijkstra step forward")
                step = self.steps[self.currentStep]
                if step[0] == "shortest path":
                    pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1]) -1)]
                    self.updateDijkstraGraph([], checkingEdges=pathEdges)
                    shortestPathText = " -> ".join(step[1])
                    self.invalidMessage["text"] = f'''Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice.get()}
is {shortestPathText} \nWeight: {self.distances[self.endVertexChoice.get()]}'''
                else:
                    visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
                    self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
                    self.updateDijkstraTable(step)

            elif hasattr(self, "updateSimplexGraph"):
                step = self.steps[self.currentStep]
                self.updateSimplexGraph()
                if step == self.steps[-1]:
                    self.invalidMessage["text"] = f"Optimal solution P = {self.steps[-1][2]}\nx = {self.steps[-1][0]}, y = {self.steps[-1][1]}"
                else:
                    self.invalidMessage["text"] = ""
```

```
171     def hideMenuWidgets(self):
172         for widget in self.menuFrame.winfo_children():
173             widget.grid_forget()
174         self.title.grid_forget()
175         self.menuFrame.configure(bg="#eaebed")
```

quiz_page.py :

```
1  # imports specific to the Quiz class
2  from tkinter import *
3  import tkinter as tk
4  import sqlite3 # for writing scores to the database
5  from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
6  from main_class_code import Main
7  from PIL import Image, ImageTk # for logo image editing
8  from random import randint, shuffle
9  import numpy as np
10 import matplotlib.pyplot as plt
11 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
12 from bubble_sort_page import BubbleSort
13 from prim_page import Prim
14 from dijkstra_page import Dijkstra
15 from simplex_page import Simplex
16 from fractions import Fraction
17 import networkx as nx
18 import heapq
```

```
21 class Quiz(Main):
22     def __init__(self, master, pUsername):
23         super().__init__(master)
24         self.master = master
25         self.username = pUsername
26         self.currentQType = None
27 
28         # instances of algorithm classes - composition:
29         self.bubbleSort = BubbleSort(master, pUsername, show_menu=False)
30         self.prim = Prim(master, pUsername, show_menu=False)
31         self.dijkstra = Dijkstra(master, pUsername, show_menu=False)
32         self.simplex = Simplex(master, pUsername, show_menu=False)
33 
34         # hide algorithm frames to avoid interference
35         self.bubbleSort.bubbleSortFrame.grid_forget()
36         self.prim.primFrame.grid_forget()
37         self.dijkstra.dijkstraFrame.grid_forget()
38         self.simplex.simplexFrame.grid_forget()
39 
40         self.userAlgorithmLocations = self.accountAlgorithms()
41 
42         self.initialQuizWidgets()
```

```
    def initialQuizWidgets(self):
        self.master.configure(bg="#eaebed")
        self.master.geometry("930x730")

        self.title = Label(self.master, text="Quiz", font=fontLarge, bg="#eaebed", fg="#1b263b")
        self.title.grid(row=0, column=0, columnspan=10)
        Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=10, columnspan=2, padx=5, pady=1)

        if "Bubble Sort" in self.userAlgorithmLocations:
            self.bubbleSortButton = Button(self.master, text="Bubble Sort", command=lambda: self.newType("Bubble Sort"), bg="#1b263b", fg="white", font=fontMedium, width=20)
            self.bubbleSortButton.grid(row=1, column=0, columnspan=3, padx=5, pady=5)

        if "Prim's" in self.userAlgorithmLocations:
            primColumn = self.userAlgorithmLocations.index("Prim's") * 3
            self.primButton = Button(self.master, text="Prim's", command=lambda: self.newType("Prim's"), bg="#1b263b", fg="white", font=fontMedium, width=20)
            self.primButton.grid(row=1, column=primColumn, columnspan=3, padx=5, pady=5)

        if "Dijkstra's" in self.userAlgorithmLocations:
            dijkstraColumn = self.userAlgorithmLocations.index("Dijkstra's") * 3
            self.dijkstraButton = Button(self.master, text="Dijkstra's", command=lambda: self.newType("Dijkstra's"), bg="#1b263b", fg="white", font=fontMedium, width=20)
            self.dijkstraButton.grid(row=1, column=dijkstraColumn, columnspan=3, padx=5, pady=5)

        if "Simplex" in self.userAlgorithmLocations:
            simplexColumn = self.userAlgorithmLocations.index("Simplex") * 3
            self.simplexButton = Button(self.master, text="Simplex", command=lambda: self.newType("Simplex"), bg="#1b263b", fg="white", font=fontMedium, width=20)
            self.simplexButton.grid(row=1, column=simplexColumn, columnspan=3, padx=5, pady=5)


        image = Image.open("NEA_logo_image.png")
        image = image.resize((481,626))
        self.logoImg = ImageTk.PhotoImage(image) # creates tkinter widget
        self.imageLabel = Label(self.master, image=self.logoImg)
        self.imageLabel.grid(row=2, column=3, columnspan=6, rowspan=8)
```

```
    def bubbleSortQuizWidgets(self):
        # widgets for bubbleSort frame
        #print("in bubbleSort quiz widgets")

        self.bubbleSortButton["bg"] = "#b2675e"

        self.bubbleSortQuizFrame = Frame(self.master, bg="#eaebed")
        self.bubbleSortQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)

        Label(self.bubbleSortQuizFrame,
              text= "1. Start at the first item in the list.\n"
                    "2. Compare the current item with the next item.\n"
                    "3. If the two items are in the wrong position, swap them.\n"
                    "4. Move up one item to the next time in the list.\n"
                    "5. Repeat from step 3 until all the unsorted items have been compared.\n"
                    "6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=2, column=0, columnspan=12, sticky="w", padx=10, pady=5)

        Label(self.bubbleSortQuizFrame, text="Please sort the following unsorted dataset in ascending order and input the number of passes it took:",
              font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=4, column=0, columnspan=12, padx=2)
```

```
        self.numLabels = []
        self.numChoices = []
        self.dropdowns = []
        iCounter = 1 # column index counter
        for number in self.quizNumbers:
            numText= str(number)
            label = Label(self.bubbleSortQuizFrame, text=numText, font=fontMedium, fg="#1b263b", bg="#eaebed")
            label.grid(row=5, column=iCounter, padx=1, pady=1)
            self.numLabels.append(label)

            var = IntVar()
            var.set(0) # default value
            self.numChoices.append(var)

            dropdown = OptionMenu(self.bubbleSortQuizFrame, var, *self.quizNumbers)
            dropdown.grid(row=6, column=iCounter, padx=1, pady=1)
            self.dropdowns.append(dropdown)

            iCounter += 1

        Label(self.bubbleSortQuizFrame, text="Number of passes = ", font=fontMedium, fg="#1b263b", bg="#eaebed", width=20).grid(row=7, column=0, columnspan=3, padx=5, pady=5)

        self.passesInput = IntVar()
        Entry(self.bubbleSortQuizFrame, textvariable=self.passesInput).grid(row=7, column=3, columnspan=3, padx=1, pady=1)

        self.answerText = Label(self.bubbleSortQuizFrame, text="", font=fontMedium, fg="#990000", bg="#eaebed")
        self.answerText.grid(row=9, column=0, columnspan=6)

        Button(self.bubbleSortQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
        Button(self.bubbleSortQuizFrame, text="New Question", command=lambda: self.newQuestion("Bubble Sort"), bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)
```

```
    def primQuizWidgets(self):
        # widgets for prim frame
        #print("in prim quiz widgets")

        self.primButton["bg"] = "#b2675e"

        self.primQuizFrame = Frame(self.master, bg="#eaebed")
        self.primQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)

        Label(self.primQuizFrame,
              text= "1. Choose any vertex to start the tree.\n"
                    "2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.\n"
                    "3. Repeat step 2 until all the vertices are connected to the tree.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=2, column=0, columnspan=12, sticky="w", padx=10, pady=5)

        Label(self.primQuizFrame,
              text="Start at vertex A.\n Input the MST edges below in the form 'AB, BC' etc:",
              font=fontMedium,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=4, column=6, columnspan=6, rowspan=2)

        self.mstEntry = StringVar()
        Entry(self.primQuizFrame, textvariable=self.mstEntry, width=35, font=("Arial", 14)).grid(row=6, column=6, columnspan=6, rowspan=2)

        Label(self.primQuizFrame, text="MST weight = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=8, column=6, columnspan=3)
        self.weightEntry = IntVar()
        Entry(self.primQuizFrame, textvariable=self.weightEntry).grid(row=8, column=9, columnspan=3)

        self.answerText = Label(self.primQuizFrame, text="", font=fontMedium, fg="#990000", bg="#eaebed")
        self.answerText.grid(row=9, column=0, columnspan=6)

        Button(self.primQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
        Button(self.primQuizFrame, text="New Question", command=lambda: self.newQuestion("Prim's"), bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)
```

```
        #print(f"graph: {self.quizGraph}")
        # create axes to plot graph on
        self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
        self.primQuizGraphCanvas = FigureCanvasTkAgg(self.fig, self.primQuizFrame)
        self.primQuizGraphCanvas.get_tk_widget().grid(row=4, column=0, rowspan=5, columnspan=6)
        # draw graph on axes
        pos = nx.spring_layout(self.quizGraph)
        nx.draw(self.quizGraph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=200, font_color='white', font_size=11, ax=self.ax
        edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
        nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, font_size=10, ax=self.ax)
        self.primQuizGraphCanvas.draw()
```

```
192     def dijkstraQuizWidgets(self):
193         # widgets for dijkstra frame
194         #print("in dijkstra quiz widgets")
195         self.dijkstraButton["bg"] = "#b2675e"
196 
197         self.dijkstraQuizFrame = Frame(self.master, bg="#eaebed")
198         self.dijkstraQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)
199 
200         Label(self.dijkstraQuizFrame, text=
201               "1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).\n"
202               "2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.\n"
203               "3. Calculate the distance from the start for each of the unvisited connected nodes. \n"
204               "4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected\nnode to be the current node. \n"
205               "5. Then set the current node as visited.\n"
206               "6. Repeat steps 2 to 5 until all nodes are set to visited.\n"
207               "To find the shortest path through backtracking:\n"
208               "7. Start from the goal node.\n"
209               "8. Add the 'previous node' to the start of a list.\n"
210               "9. Repeat step 7 until the start node is reached.",
211               font=fontSmall,
212               fg="#1b263b",
213               bg="#eaebed",
214               justify="left").grid(row=2, column=0, columnspan=12, sticky="w", padx=10, pady=5)
215 
216         self.startEndText = Label(self.dijkstraQuizFrame,
217               text=f"Start at vertex A. End vertex {self.endVertex}.\nInput the shortest path in the form 'ABC...':",
218               font=fontMedium,
219               fg="#1b263b",
220               bg="#eaebed",
221               justify="left")
222         self.startEndText.grid(row=4, column=6, columnspan=6, rowspan=2)
223 
224         self.shortestPathEntry = StringVar()
225         Entry(self.dijkstraQuizFrame, textvariable=self.shortestPathEntry).grid(row=6, column=6, columnspan=6, rowspan=2)
226 
227         Label(self.dijkstraQuizFrame, text="Shortest path weight = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=8, column=6, columnspan=3)
228         self.weightEntry = IntVar()
229         Entry(self.dijkstraQuizFrame, textvariable=self.weightEntry).grid(row=8, column=9, columnspan=3)
```

```
        self.answerText = Label(self.dijkstraQuizFrame, text="", font=fontMedium, fg="#990000", bg="#eaebed")
        self.answerText.grid(row=9, column=0, columnspan=6)

        Button(self.dijkstraQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
        Button(self.dijkstraQuizFrame, text="New Question", command=lambda: self.newQuestion("Dijkstra's"), bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)

        #print(f"graph: {self.graph}")
        # create axes to plot graph on
        self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
        self.dijkstraQuizGraphCanvas = FigureCanvasTkAgg(self.fig, self.dijkstraQuizFrame)
        self.dijkstraQuizGraphCanvas.get_tk_widget().grid(row=4, column=0, rowspan=5, columnspan=6)
        # draw graph on axes
        pos = nx.spring_layout(self.quizGraph)
        nx.draw(self.quizGraph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=200, font_color='white', font_size=11, ax=self.ax)
        edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
        nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)
        self.dijkstraQuizGraphCanvas.draw()
```

```
    def simplexQuizWidgets(self):
        # widgets for simplex frame
        #print("in simplex quiz widgets")
        self.simplexButton["bg"] = "#b2675e"
        self.simplexQuizFrame = Frame(self.master, bg="#eaebed")
        self.simplexQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)


        Label(self.simplexQuizFrame, text=
              "1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function\nto be equal to 0. Input these equations into the simplex tableau.\n"
              "2. Choose the most negative value in the objective row to become the basis.\n"
              "3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot.\n"
              "4. Divide the pivot row by the value of the pivot.\n"
              "5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.\n"
              "6. Repeat until the objective row contains no negative entries.\n"
              "7. Read off the optimal solution from the tableau.",
              font=fontSmall,
              fg="#1b263b",
              bg="#eaebed",
              justify="left").grid(row=2, column=0, rowspan=2, columnspan=12, sticky="w", padx=5, pady=5)

        self.constraintTexts = []
        for constraint in self.quizConstraints:
            a,b,c = constraint
            text = f"{a}x + {b}y ≤ {c}"
            self.constraintTexts.append(text)

        a,b, = self.quizObjective
        self.objectiveText = f"P = {a}x + {b}y"
```

```
        self.textSayingConstraints = Label(self.simplexQuizFrame, text=
              "Please enter either integers or fractions (e.g. in the form 3/5)\n"+
              "Constraints:\n" + "\n".join(self.constraintTexts) + "\n"
              "Objective: " + self.objectiveText,
              font=fontMedium,
              fg="#1b263b",
              bg="#eaebed",
              justify="left")
        self.textSayingConstraints.grid(row=4, column=7, rowspan=2, columnspan=5, sticky="w", padx=5, pady=5)

        Label(self.simplexQuizFrame, text="x = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=6, column=6, columnspan=2, padx=5)
        self.xInput = StringVar()
        Entry(self.simplexQuizFrame, textvariable=self.xInput).grid(row=6, column=8, columnspan=2)

        Label(self.simplexQuizFrame, text="y = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=7, column=6, columnspan=2, padx=5)
        self.yInput = StringVar()
        Entry(self.simplexQuizFrame, textvariable=self.yInput).grid(row=7, column=8, columnspan=2)

        Label(self.simplexQuizFrame, text="P = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=8, column=6, columnspan=2, padx=5)
        self.pInput = StringVar()
        Entry(self.simplexQuizFrame, textvariable=self.pInput).grid(row=8, column=8, columnspan=2)


        self.answerText = Label(self.simplexQuizFrame, text="", font=fontMedium, fg="#990000", bg="#eaebed")
        self.answerText.grid(row=9, column=0, columnspan=6)

        Button(self.simplexQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
        Button(self.simplexQuizFrame, text="New Question", command=lambda: self.newQuestion("Simplex"), bg="#1b263b", fg="white",
               font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)
        # plot on graph:
        self.colours = ["#606c38", "#b2675e", "#eca400"]
```

```
        self.fig, self.ax = plt.subplots(figsize=(4,4))
        for constraint in self.quizConstraints:
            a, b, c = constraint
            #print(a, b, c)
            i = self.quizConstraints.index(constraint)
            xVals = np.linspace(0, 10, 200) # used to plot constriant lines on graph
            if b != 0: # is a line in the form ax + by = c
                yVals = (c - a*xVals) / b
            else: # in the form ax = c
                xVals = np.full_like(xVals, c / a)
                yVals = np.linspace(0, 10, 200)
            self.ax.plot(xVals, yVals, label=f"Constraint {i+1}", color=self.colours[i])

        self.ax.set_xlim(0,10)
        self.ax.set_ylim(0,10)
        self.ax.axhline(0, color="black", linewidth=1)
        self.ax.axvline(0, color="black", linewidth=1)
        self.ax.set_xlabel("x")
        self.ax.set_ylabel("y")
        self.ax.legend()
        self.ax.grid()

        self.canvas = FigureCanvasTkAgg(self.fig, master=self.simplexQuizFrame)
        self.canvas.get_tk_widget().grid(row=4, column=0, rowspan=5, columnspan=6)
        self.canvas.draw()
```

```
    def bubbleSortDataset(self):
        # 10 random integers between 1 and 50 inclusive
        #print("in create bubbleSort dataset")
        self.quizNumbers = []
        for i in range(10):
            number = randint(1, 50)
            self.quizNumbers.append(number)


    def randomGraph(self):
        # connected graph of 6-8 vertices labelled A, B, C...
        # weights between 1 and 50
        #print("in create random graph")
        numVertices = randint(6,8)
        vertices = [chr(65+i) for i in range(numVertices)]
        edges = []
        startVertex = vertices[0]
        visited = set([startVertex])
        queue = []

        # alll edges from start vertex to queue
        for v in vertices:
            if v != startVertex:
                weight = randint(1,50)
                heapq.heappush(queue, (weight, startVertex, v))
```

```
        while queue:
            weight, u, v = heapq.heappop(queue)
            # has been visited - add to MST
            if v not in visited:
                visited.add(v)
                edges.append((u, v, weight))
                for w in vertices:
                    if w != v and w not in visited:
                        weight = randint(1,50)
                        heapq.heappush(queue, (weight, v, w))
            if len(visited) == numVertices:
                break

        # add random extra edges:
        extraEdges = randint(4, 10)
        allPossEdges = [(u,v) for u in vertices for v in vertices if u<v]
        shuffle(allPossEdges)
        totalEdges = extraEdges + len(edges)
        for u, v in allPossEdges:
            if len(edges) >= totalEdges:
                break
            if (u,v) not in edges and (v,u) not in edges:
                weight = randint(1,50)
                edges.append((u,v,weight))

        #print(f"Vertices generated: {vertices}")
        #print(f"Edges generated: {edges}")
        self.quizGraph = nx.Graph()
        self.quizGraph.add_weighted_edges_from(edges)
        #print(f"self.graph nodes: {self.graph.nodes}")
        #print(f"self.graph edges: {self.graph.edges}")

        self.endVertex = vertices[-1] # this is needed for dijkstra's
```

```
    def simplexConstraints(self):
        # 3 constraints and objective function
        # integers coefficients between -10 and 10, x and y not both 0
        #print("in create simplex constraints")
        while True:
            self.quizConstraints = []
            while len(self.quizConstraints) < 3:
                xCoef = randint(-10, 10)
                yCoef = randint(-10, 10)
                rhs = randint(0, 10)
                if yCoef !=0 or xCoef != 0: # allows for one to be 0 using or
                    self.quizConstraints.append([float(xCoef), float(yCoef), float(rhs)])

            self.quizObjective = []
            while len(self.quizObjective) < 1:
                xCoef = randint(0, 10)
                yCoef = randint(0, 10)
                if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
                    self.quizObjective.append(float(xCoef))
                    self.quizObjective.append(float(yCoef))


            # check if feasible:
            self.simplex.valueConstraints = self.quizConstraints[:]
            self.simplex.valueObjective = self.quizObjective[:]
            #print(f"simplex valueConstraints: {self.simplex.valueConstraints}")
            #print(f"simplex valueObjective: {self.simplex.valueObjective}")

            self.simplex.createTableau()
            self.simplex.simplexSteps()
            '''print("steps:")
            for step in self.simplex.steps:
                print(step)'''
```

```
            #print(f"state of invalid: {self.simplex.invalid}")
            if self.simplex.invalid == False:
                break
            else:
                #print("Generated infeasible problem")
                self.simplex.steps = [(0,0,0)]
                self.simplex.tableau = np.array([], dtype=float) # empty numpy array'''
                self.simplex.valueConstraints = []
                self.simplex.valueObjective = []
        correctX, correctY, correctP = self.simplex.steps[-1]
        #print(f"correct: x={correctX}, y={correctY}, P={correctP}")
```

```
    def check(self):
        # calls updateScore
        #print("in check")
        if self.currentQType == "Bubble Sort":
            self.bubbleSort.numbers = self.quizNumbers[:] # copy
            self.bubbleSort.bubbleSortSteps()
            correctOrder = list(self.bubbleSort.steps[-1]) # final sorted order
            correctPasses = self.bubbleSort.numberPasses

            userOrder = [var.get() for var in self.numChoices]
            try:
                userPasses = int(self.passesInput.get())
            except ValueError:
                userPasses = -1

            if userOrder == correctOrder and userPasses == correctPasses:
                self.answerText["text"] = "Correct :)"
                self.answerText["fg"] = "#008000"
                self.updateScore("Correct")
            elif userOrder != correctOrder:
                self.answerText["text"] = "Incorrect order\nTry a new question"
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
            else: # passes wrong
                self.answerText["text"] = f"Incorrect number of passes. Correct answer: {correctPasses}.\nTry a new question"
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
```

```
        elif self.currentQType == "Prim's":
            self.prim.graph = self.quizGraph.copy()
            #self.prim.startVertexChoice = "A"
            self.prim.primSteps(start="A")
            # edges in alphabetical roder AB instead of BA:
            correctMST = set(tuple(sorted(edge)) for edge in self.prim.steps[-1])
            correctWeight = sum(self.quizGraph[u][v]["weight"] for u,v in correctMST)

            #print(f"Correct MST: {correctMST}")
            #print(f"Correct weight: {correctWeight}")
            # remove spaces, split by commas
            userMSTInput = self.mstEntry.get().replace(" ", "").split(",")
            userMST = set()
            for edge in userMSTInput:
                if len(edge) == 2:
                    userMST.add(tuple(sorted(edge))) # store as sorted tuple - AB same as BA
            try:
                userWeight = int(self.weightEntry.get())
            except ValueError:
                userWeight = -1 # invalid input

            #print(f"User MST: {userMST}")
            #print(f"User weight: {userWeight}")
            # check correctness:
            if userMST == correctMST and userWeight == correctWeight:
                self.answerText["text"] = "Correct :)"
                self.answerText["fg"] = "#008000"
                self.updateScore("Correct")
            elif userMST != correctMST:
                self.answerText["text"] = "Incorrect MST edges\nTry a new question"
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
            else:
                self.answerText["text"] = f"Incorrect MST weight. Correct answer: {correctWeight}"
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
```

```
        elif self.currentQType == "Dijkstra's":
            self.dijkstra.graph = self.quizGraph.copy()
            self.dijkstra.dijkstraSteps(startV="A", endV=self.endVertex)
            correctPath = "".join(self.dijkstra.steps[-1][1]) # last tuple, 2nd part
            correctWeight = self.dijkstra.distances[self.endVertex]

            userPath = self.shortestPathEntry.get().strip().upper()
            try:
                userWeight = int(self.weightEntry.get())
            except ValueError:
                userWeight = -1

            # check correctness:
            if userPath == correctPath and userWeight == correctWeight:
                self.answerText["text"] = "Correct :)"
                self.answerText["fg"] = "#008000"
                self.updateScore("Correct")
            elif userPath != correctPath:
                self.answerText["text"] = "Incorrect path\nTry a new question"
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
            else:
                self.answerText["text"] = f"Incorrect path weight. Correct answer: {correctWeight}"
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
```

```
        elif self.currentQType == "Simplex":
            correctX = Fraction(self.simplex.steps[-1][0]).limit_denominator()
            correctY = Fraction(self.simplex.steps[-1][1]).limit_denominator()
            correctP = Fraction(self.simplex.steps[-1][2]).limit_denominator()

            #correctX, correctY, correctP = self.simplex.steps[-1]
            #print(f"correct: x={correctX}, y={correctY}, P={correctP}")
            try:
                userX = Fraction(self.xInput.get())
                userY = Fraction(self.yInput.get())
                userP = Fraction(self.pInput.get())
            except ValueError:
                self.answerText["text"] = "Please enter numbers of fractions (e.g. 3/5)"
                self.answerText["fg"] = "#990000"
                return

            if userX == correctX and userY == correctY and userP == correctP:
                self.answerText["text"] = "Correct :)"
                self.answerText["fg"] = "#008000"
                self.updateScore("Correct")
            else:
                self.answerText["text"] = "Incorrect\nTry a new question."
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
```

```
572     def newQuestion(self, qType):
573         # forget current widgets, create new ones.for qType
574         #print(f"in new question for: {qType}")
575         if qType == "Bubble Sort":
576             self.bubbleSortDataset()
577             # reset text labels:
578             for label, number in zip(self.numLabels, self.quizNumbers):
579                 label.config(text=str(number))
580             # reset dropdowns:
581             for var, dropdown in zip(self.numChoices, self.dropdowns):
582                 var.set(0)
583                 dropdown["menu"].delete(0, "end")
584                 for num in self.quizNumbers:
585                     dropdown["menu"].add_command(label=num, command=lambda value=num, var=var: var.set(value))
586             # clear text box:
587             self.passesInput.set("")
588             self.answerText["text"] = ""
589             self.bubbleSort.numberPasses = 0
590 
591 
592         elif qType == "Prim's":
593             self.randomGraph()
594 
595             self.ax.clear()
596             pos = nx.spring_layout(self.quizGraph)
597             nx.draw(self.quizGraph, pos, with_labels=True, node_color='#606c38', edge_color='black',
598                     node_size=200, font_color='white', font_size=11, ax=self.ax)
599             edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
600             nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)
601             self.primQuizGraphCanvas.draw()
602 
603             self.mstEntry.set("")
604             self.weightEntry.set(0)
605             self.answerText["text"] = ""
```

```
        elif qType == "Dijkstra's":
            self.randomGraph()

            self.ax.clear()
            pos = nx.spring_layout(self.quizGraph)
            nx.draw(self.quizGraph, pos, with_labels=True, node_color='#606c38', edge_color='black',
                    node_size=200, font_color='white', font_size=11, ax=self.ax)
            edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
            nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)
            self.dijkstraQuizGraphCanvas.draw()

            self.shortestPathEntry.set("")
            self.weightEntry.set(0)
            self.answerText["text"] = ""

            # add code to update the text saying the end vertex
            self.startEndText["text"] = f"Start at vertex A. End vertex {self.endVertex}.\nInput the shortest path in the form 'ABC...':"
```

```
        elif qType == "Simplex":
            self.simplexConstraints()
            self.answerText["text"] = ""

            self.constriantsTexts = [ f"{a}x + {b}y ≤ {c}" for a,b,c in self.quizConstraints]
            self.objectiveText = f"P = {self.quizObjective[0]}x + {self.quizObjective[1]}y"
            self.textSayingConstraints["text"] = "Please enter either integers or fractions (e.g. in the form 3/5)\n" + "Constraints:\n" + "\n".join(self.constraintTexts) + "\n" + "Objective: " + self.objectiveText

            self.xInput.set("")
            self.yInput.set("")
            self.pInput.set("")

            self.ax.clear()
            for constraint in self.quizConstraints:
                a, b, c = constraint
                #print(a, b, c)
                i = self.quizConstraints.index(constraint)
                xVals = np.linspace(0, 10, 200) # used to plot constriant lines on graph
                if b != 0: # is a line in the form ax + by = c
                    yVals = (c - a*xVals) / b
                else: # in the form ax = c
                    xVals = np.full_like(xVals, c / a)
                    yVals = np.linspace(0, 10, 200)
                self.ax.plot(xVals, yVals, label=f"Constraint {i+1}", color=self.colours[i])

            self.ax.set_xlim(0,10)
            self.ax.set_ylim(0,10)
            self.ax.axhline(0, color="black", linewidth=1)
            self.ax.axvline(0, color="black", linewidth=1)
            self.ax.set_xlabel("x")
            self.ax.set_ylabel("y")
            self.ax.legend()
            self.ax.grid()

            self.canvas.draw()
            self.answerText["text"] = ""
```

```
668     def newType(self, newQType):
669         # qType = question type ---> bubble Sort, prim, dijkstra, simplex
670         # forget currentQType ---> None initially
671         #print(f"current question type: {self.currentQType}, new question type: {newQType}")
672         self.imageLabel.grid_forget()
673 
674         # change current button colour to blue, forget current question widgets
675         if self.currentQType == "Bubble Sort":
676             self.bubbleSortButton["bg"] = "#1b263b"
677             for widget in self.bubbleSortQuizFrame.winfo_children():
678                 widget.grid_forget()
679 
680         elif self.currentQType == "Prim's":
681             self.primButton["bg"] = "#1b263b"
682             for widget in self.primQuizFrame.winfo_children():
683                 widget.grid_forget()
684 
685         elif self.currentQType == "Dijkstra's":
686             self.dijkstraButton["bg"] = "#1b263b"
687             for widget in self.dijkstraQuizFrame.winfo_children():
688                 widget.grid_forget()
689 
690         elif self.currentQType == "Simplex":
691             self.simplexButton["bg"] = "#1b263b"
692             for widget in self.simplexQuizFrame.winfo_children():
693                 widget.grid_forget()
```

```
695         # new question widgets:
696         if newQType == "Bubble Sort":
697             self.currentQType = "Bubble Sort"
698             self.bubbleSortDataset()
699             self.bubbleSortQuizWidgets()
700             self.master.geometry("880x440")
701 
702         elif newQType == "Prim's":
703             self.currentQType = "Prim's"
704             self.randomGraph()
705             self.primQuizWidgets()
706             self.master.geometry("880x650")
707 
708         elif newQType == "Dijkstra's":
709             self.currentQType = "Dijkstra's"
710             self.randomGraph()
711             self.dijkstraQuizWidgets()
712             self.master.geometry("880x795")
713 
714         elif newQType == "Simplex":
715             self.currentQType = "Simplex"
716             self.simplexConstraints()
717             self.simplexQuizWidgets()
718             self.master.geometry("965x795")
```

```
    def updateScore(self, answer):
        # answer = Correct or Incorrect
        #print("in update score")
        usernameHolder = self.username
        algorithmHolder = self.currentQType
        if answer == "Correct":
            self.cursor.execute("UPDATE StudentEnrolment SET CorrectScore = CorrectScore+1 WHERE Username = ? AND Algorithm=?;", (usernameHolder, algorithmHolder))
            self.conn.commit()
        elif answer == "Incorrect":
            self.cursor.execute("UPDATE StudentEnrolment SET IncorrectScore = IncorrectScore+1 WHERE Username = ? AND Algorithm=?;", (usernameHolder, algorithmHolder))
            self.conn.commit()
        #print("score updated")
```

```
    def accountAlgorithms(self):
        try:
            usernameHolder = self.username
            self.cursor.execute("SELECT Algorithm FROM StudentEnrolment WHERE Username=?;", (usernameHolder,))
            result = self.cursor.fetchall()
            #print(f"sql result: {result}")
            algorithmsList = [row[0] for row in result]
            #print(f"algorithms list: {algorithmsList}")
            return algorithmsList
        except sqlite3.Error as e:
            print(f"Database error: {e}")

    def openHome(self):
        closeOpen(self.master, "home", self.username)

# fro testing ------------------------------------------------------------------------
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Quiz test")
    #root.geometry("800x595")
    graph_input = Quiz(root, "HelloWord!!!") #HellowWord!!! username for testing purposes
    root.mainloop()
```

Finally, for the software requirements for the user, the libraries required to be installed are:

- tkinter
- sqlite3
- string
- matplotlib
- PIL
- networkx
- heapq
- numpy
- fractions

REFERENCES

1. https://qualifications.pearson.com/content/dam/pdf/A%20Level/Mathematics/2013/Specification%20and%2 0sample%20assessments/UA035243_GCE_Lin_Maths_Issue_3.pdf (14/09/2024) — page 102
2. https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-prims-algorithm/ (08/10/24)
3. https://www.geeksforgeeks.org/time-and-space-complexity-of-dijkstras-algorithm/ (08/10/24)
4. https://open.icm.edu.pl/server/api/core/bitstreams/66100a73-8231-4c0f-b1d3-2127b7f24f93/content#:~:te x t=Visualizations%20are%20associated%20with%20cognitive,representation%22%20(Duval%201999). (11/10/24)
5. https://www.jcq.org.uk/examination-results/2024-a-level-results/ (25/09/24)
6. https://education-today.co.uk/record-numbers-of-students-choose-computer-science-a-level-in-2022/ (25/09/24)
7. https://futurescot.com/new-census-data-shows-computing-teacher-numbers-at-record-low/ (25/09/24)
8. https://www.sciencedirect.com/science/article/pii/S1110866518302603 (07/10/2024)
9. https://www.cs.usfca.edu/~galles/visualization/Algorithms.html (14/09/2024)
10. https://gilp.henryrobbins.com/en/latest/index.html (08/10/24)
11. https://www.davbyjan.com/ (08/10/24)
12. https://lordslibrary.parliament.uk/research-briefings/lln-2024-0040/ (30/09/24)
13. https://www.gov.uk/data-protection (30/09/24)
14. https://www.geeksforgeeks.org/what-are-the-minimum-hardware-requirements-for-python-progr amming/ (20/11/24)
15. https://thonny.org/ (20/11/24)
16. https://gs.statcounter.com/os-market-share/desktop/worldwide/#daily-20240921-20241020 (20/11/24)
17. https://docs.python.org/3/library/tkinter.html (20/11/24)
18. https://matplotlib.org/ (20/11/24)
19. https://networkx.org/documentation/stable/install.html (20/11/24)
20. https://www.geeksforgeeks.org/python-sqlite/ (20/11/24)
21. https://www.supercolor.com/blog/the-meaning-of-the-color-blue/#:~:text=The%20color%20blue%20repre sents%20both,stability%2C%20faith%2C%20and%20intelligence. (17/11/24)
22. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/ (01/03/25)
23. https://www.tutorialspoint.com/sqlite/index.htm (02.01.24)
24. https://www.geeksforgeeks.org/bar-plot-in-matplotlib/ (12.01.25)
25. https://www.geeksforgeeks.org/how-to-embed-matplotlib-charts-in-tkinter-gui/ (12.01.25)
26. https://adacomputerscience.org/concepts/oop_polymorphism?topic=object_oriented_programming (12.01.25)
27. https://www.geeksforgeeks.org/properties-of-minimum-spanning-tree-mst/ (05/03/25)
28. https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.algorithms.compone nts (09/03/25)
29. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shorte st_paths.generic.has_path.html (09/03/25)
30. https://www.geeksforgeeks.org/inheritance-and-composition-in-python/ (22/03/25)
31. https://www.uky.edu/~dsianita/300/online/LP.pdf (22/03/25) — page 18
32. https://maths.shelswell.org.uk/dijkstras-algorithm (30/03/25)