Route Finding

PROGRAMMING PROJECT 2 REPORT

LUKE HUTTON [SC17LJTH]

Contents

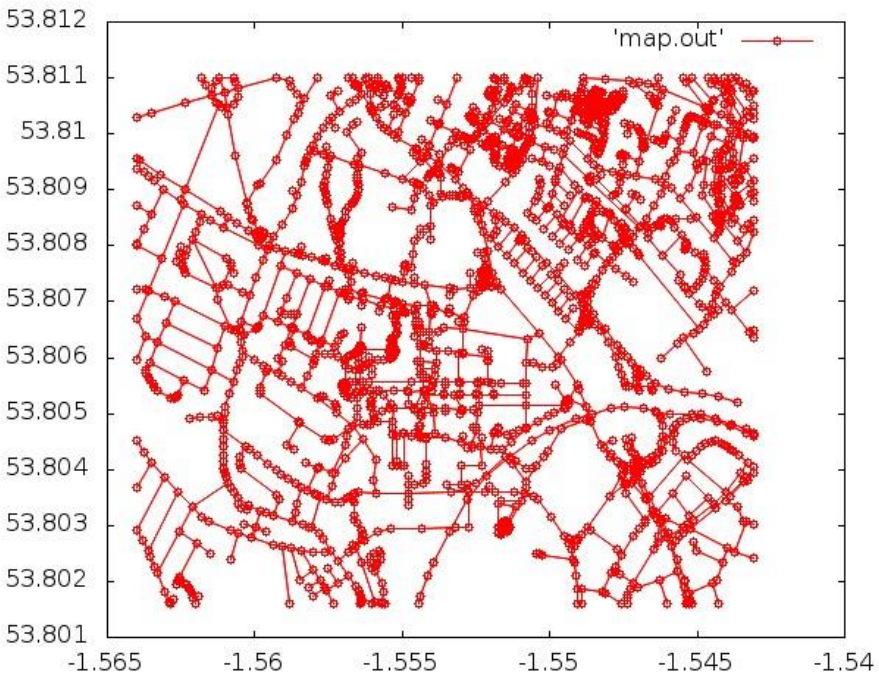
Project Summary

Chosen project - Route finding

A graph or network is collection of points (nodes) connected by a series of lines (edges). Graphs are used in an area of maths called discrete maths to model optimisation problems that involves finding an efficient solution to a problem. Solving an optimisation problem means finding the best solution out of many feasible solutions.

This project will involve creating an application that will find the shortest path between two nodes on a network. The network will be given in the form of a file; the application will then read this and output the shortest path. In order to create such an application, the following things need to be taken into consideration: importing the data, the data structure used to store the network and the algorithm used to determine the shortest path.

The aim of this application will be to display, to the user, a graphical shortest route between any two points on the university's campus (or any map input). It will also output the shortest distance of the path (this measurement will be in the units given in the data). The data output should look something like this:



(only with a path coloured in another colour between two points of nodes)

Design Plan

Large scale design plan

Aim of the project

The ultimate aim of the project is to find the shortest path between two input nodes. This can be done using Dijkstra's algorithm in O((E + N) log N) time using a priority queue with a

binary heap (where E is the number of edges and N is the number of nodes). This will help to implement Dijkstra's algorithm efficiently, which is critical for such a large data set.

INPUT: The program must be able to accept an XML input from a file. This data will contain information about the nodes and edges in the graph. For the nodes the id, latitude and longitude will be stored and for the edges the id, source node, destination node and length will be stored. Some of this data will be used and other parts won't. A plugin called libxml2 can be used to read and parse XML files I will therefore use this in my project to read the data file.

Upon further inspection I decided against using libxml2 and instead using the built in sscanf() and fgets() functions instead. It was simpler to read the file this way as libxml2 required a complex setup and contained many additional features the program wouldn't need. However, this alternative method has limitations. The structure of the XML file must remain consistent and use the same <link> and <node> tags to denote an edge and node.

Another input from the user will be the 2 points to find the shortest path between. The user will enter these as a parameter when running the program through the terminal.

OUTPUT: The program will output an image which will display a map with the shortest path between two nodes displayed on it. The path will be a different colour to the standard edges in the graph. The program will also output the path in terms of node ID's and what the size of the shortest path is. This information will be output in the terminal.

I will do this using GNUplot as we have already used it before and it is an easy to use application. With gnuplot you can change the line colour of elements. Therefore, it will be easy to display the shortest path calculated by the program.

PROBLEM TO BE SOLVED: The main problem in this program is how to find the shortest path given a network of nodes and edges. This can be found using Dijkstra's algorithm. To make this as efficient as possible I will implement a priority queue with a binary heap allowing a node of minimum length to be found in O(1) time given that the heap is maintained.

I will break the development of the project down into 3 main sections:

- Reading the data from an input file and creating the network
- Finding the shortest path between a pair of nodes
- Outputting the path both in text and graphical form

Medium scale design plan

Design Iterations

The simplest application I will design is one which will take input from an XML file and output the shortest path in the terminal as a series of node ID's. The next iteration of the program is to optimise the way that the shortest path algorithm calculates the shortest path and add a way to view the network and path graphically.

Code modules

The program will consist of the following files:

- main.c – the main file where the program is run
- readFile.c – takes the input XML file and reads it, adding the nodes and edges
- buildNetwork.c – contains functions to 'build' the network such as addNodes() and createNetwork()
- networkUtils.c – contains standard functions to use on the network such as getNode()
- binaryHeap.c – implements a binary min heap to use with the priority queue
- priorityQueue.c – implements a priority queue to use with djikstras algorithm
- djikstrasAlgorithm.c – implements dkikstra's algorithm to find the shortest path between two nodes
- outFile.c – takes the resulting graph and outputs it to a file so it can be read by gnu plot
- handelError.c – this will be a small module designed to ouput an error message (which will be decided in other parts of the program) and exit the program all together.

And the following header files:

- networkStructure.h – defines the structure of the network, nodes and edges
- Header files for each of the '.c' files above, declaring the functions the .c files use.

Dijkstra's algorithm

Djikstra's algorithm can be used to calculate the shortest path between two nodes on a network. This algorithm originally had a runtime of $O(N^2)$ where N is the number of nodes in the network. However once implemented with a min-priority queue with a binary heap, the efficiency is increased giving it a run time of O((E + N)log(N)) where E is the number of edges and N is the number of nodes in the network.[1] This is the pseudo code I will base my implementation off of in c:

```
"Find shortest path"
FOR x IN numberOfNodes
  distance(x) ← infinity
  processed(x) ← false
  parent (x) ← null
ENDFOR

distance(SourceNode) ← 0

WHILE (there are still nodes left to process)
```

```
  Let 'sNode' be a node which hasn't been processed that has the
smallest distance(node)
ENDWHILE

  IF (sNode ← destinationNode) THEN
    Exit While Loop
  ENDIF

  processed(sNode) ← true

  FOR Each unprocessed parent node, dNode
    IF (distance(node) + weight(sNode, dNode) < distance(dNode))
THEN
      distance(dNode) ← distance(sNode) + weight(sNode, dNode)
      parent(dNode) ← sNode
    ENDIF
  ENDFOR

# Find path
node ← destinationNode

WHILE node != sourceNode
  Append node to the beginning of the path list
  node ← parent(dNode)
ENDWHILE

Append sourceNode to the beginning of the path list
```

Priority queue implementation

To implement the min-priority queue for use with djikstra's algorithm, we will use a min-binary heap. This is a data structure that guarantees the item with the lowest priority will be at the top of the structure. This means that finding the item with the minimum priority can be found in O(1) time and found and removed in a worst case run time of O(log n).[2]

Whilst a more efficient implementation with better time complexity is possible using a Fibonacci heap, this is often inefficient in practice due to the complexity of the algorithm. They are only typically efficient on sparse networks with less edges between nodes.

A min-binary tree is ordered in such a way so that the first item on the tree has the lowest priority. For any node N, if P is the parent of N, then the priority of P is less than or equal to the priority of N[4].

Hashing algorithm to efficiently find nodes

Dijkstra's algorithm requires nodes to be 'found' amongst all nodes that were added. Instead of searching through each node and checking its ID (which would take at most O(n) time), a technique called hashing can be used. This has an average time complexity of O(1) meaning it is much more efficient, especially on a large dataset like the one we have.

In order to form a hash of the data, I will use a library called uthash[5]. This provides a way to make a hash table and handle hash clashes that occur. Uthash allows any structed in C to

become hashable. An instance of this structure can then be stored in a hash table array and found very quickly. This makes finding nodes given an Id very quick.

The two functions I will use from the uthash library are:

- HASH_FIND_INT – this function finds an item in the hash table array given an integer as a key.
- HASH_ADD_INT – this function adds an item to the hash table given an integer as a key. The function takes the key and the complementary structure, develops a hash and adds the pointer to the node to the hash table at the hash index.
- HASH_DEL – this function safely removes an item from the hash table.

Test Plan

The development of the application can be divided into 3 main theoretical sections:

- Input from the file
- Running Dijkstra's algorithm on the data input
- Outputting the shortest path to the user

Medium scale tests

In each of these sections I will complete various low-level tests to make sure that the output is as expected before moving onto the next section. This will include checking each function works as intended and checking inputs with standard and erroneous data.

I will also check for memory leaks using valgrind [6], an application that allows you to find data that hasn't been freed when the program exits, causing memory leaks to occur.

Large scale tests

Once each section has been developed I will produce a number of tests to ensure the program works as intended. This table shows the tests I will implement in a bash script to demonstrate the program works as intended. They cover all areas in which erroneous data could be input and produce an error as well as various graph sizes which should be output correctly.

Test	Expected output:	Actual output:
Test 1 – testing wrong filename input	An error stating that the file at the file path entered could not be found.	
Test 2 – running the program with no parameters provided for the file path and start and end node ID's	An error stating that too few parameters were provided.	
Test 3 – running Dijkstra's algorithm on a disconnected graph	An error stating that a path could not be constructed because a node in the path is disconnected.	
Test 4 – testing an input with node ID's that don't exist	An error stating that the node ID's could not be found.	
Test 5 – testing Dijkstra's algorithm on a simple map	Expected path: 1-2-5-4 Expected distance: 7	
Test 6 – testing Dijkstra's algorithm on another simple graph.	Expected path: 6-1-2 Expected distance: 3	
Test 7 – wrong type of node ID entered i.e. text instead of a number	An error stating that the node ID's entered are invalid.	
Test 8 – the same start and end nodes are entered	An error stating that the start node and end node cannot be the same.	
Test 9 – a path that should contain a single edge between two adjacent nodes	Expected path: 1967343264-1615401915 Expected distance: 49.025177	

Test 10 -a medium length path between two nodes	Expected Path: -2562,-2560, 1187324666, 1187324679, 54060643, 54060637, 985096817, 985096821, 985096825, -2416, -2380 Expected distance: 139.481734	
Test 11 – A large path with many edges	*	

* Expected path: -2524, -2540, 247958669, 247958668, 301673229, 247958643, 247958642, 247958641, 247958640, 247958645, 247958646, 247958647, 247958648, 247958649, 247958650, 247958651, 247958653, 247958654, 301673241, 247958598, 247958611, 247958610, 247958608, 247958607, 247958606, 247958605, 247958604, 301673248, 247958601, 247958600, 247958599, 247293219, -2502, 247293220, 247293215, 985096764, 985096813, 985096817, 54060637, 54060643, 1187324679, 1187324666, -2560, 1187324670, 1187324682, 1187324692, 1187324676, 1187324678, 1187324665, 1187324690, 984231425, 54060524, 984231603, 54060515, 54060547, 54060551, 1725582974, 54060555, 54060557, 1668111641, 1668111640
Expected distance: 464.077136

At each iteration of design

Schedule

This project is to be completed within a realistic time frame of 6 weeks. This will include time spent over the Easter break. The table below shows what I will work towards in each week of the project being completed.

Week number	Week dates	What will be achieved?
1	12th March to 18th March	In this week I will write the summary of the project, start the design of the application and begin to determine the structure the application will take. I will research numerous data structures (such as an adjacency list and adjacency matrix) and determine the structures best suited to the project.
2	19th March to 25th March	In this week I will begin to write the part of the program that accepts a file as an input and adds the relevant data to the graph. I will also work on the design plan section of the report, justifying why I used particular data structures.
3	26th March to 1st April	In this week I will write the majority of code to work towards outputting the shortest path. I will implement a priority queue and Dijkstra's algorithm and ensure the shortest path output is correct.
4	2nd April to 8th April	In this week I will add the code to output the shortest path graphically using GNUplot. Once this is completed the majority of the program will be complete.
5	9th April to 15th April	I will use this week to make any additional changes to the application, write tests that can be used to make sure the program works with no errors and begin to write a test plan.
6	16th April to 22nd April	I will use this week to finalise the project. I will finish writing the test section of the report and ensure there are no additional changes to the code that need to be made.

Test Outcomes

Testing the file data input functions

The program first needed to be able to take a file as an input and obtain node and edge data to be used throughout the rest of the program. In order to test this, I started out with a small trial dataset of 3 nodes and 3 edges to see if the data was obtained correctly and added to the struts correctly. Then I used the large dataset to test for a much larger set of nodes and edges.

```
----Nodes----
ID: -8847, X: -1.562377, Y: 53.807817
ID: -8849, X: -1.562578, Y: 53.807866
ID: -2560, X: -1.554711, Y: 53.807613
ID: -2562, X: -1.554926, Y: 53.807634
ID: -2558, X: -1.554915, Y: 53.807660
ID: 1970536624, X: -1.555058, Y: 53.804074
ID: 1967343264, X: -1.554946, Y: 53.804074
ID: 1615401915, X: -1.554947, Y: 53.804646
ID: 319432035, X: -1.554973, Y: 53.804638
ID: 247293203, X: -1.555344, Y: 53.804849
ID: 1615404345, X: -1.555335, Y: 53.804747
ID: 454231775, X: -1.555337, Y: 53.804634
ID: 1968799687, X: -1.555334, Y: 53.804071
ID: 1970536626, X: -1.555181, Y: 53.804071
ID: -2554, X: -1.556291, Y: 53.805410
ID: -2552, X: -1.556289, Y: 53.805399
ID: 247293206, X: -1.556054, Y: 53.805110
ID: -2548, X: -1.556098, Y: 53.805114
ID: -2550, X: -1.556098, Y: 53.805232
```

This list of nodes continues, and the edges are then displayed. But it is clear to see that the nodes and edges are correctly added to the ‘adjacencyListArray’ and each node’s ‘linked adjacency list array’ respectively.

I then checked to make sure that this part of the program had no memory leaks. Including destroying the network after adding all the data:

```
luke@luke:~/Desktop/Computer_Science/Yr1 Sem 2/Programming Project/comp1921/Cour
sework 2$ make
gcc -Iinclude   -c -o src/main.o src/main.c
gcc -o out/main src/main.o src/readFile.o src/buildNetwork.o src/networkUtils.o
src/binaryHeap.o src/priorityQueue.o src/djikstrasAlgorithm.o src/writeNetwork.o
 src/handleError.o -Iinclude -lm
luke@luke:~/Desktop/Computer_Science/Yr1 Sem 2/Programming Project/comp1921/Cour
sework 2$ valgrind ./out/main
==5432== Memcheck, a memory error detector
==5432== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5432== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5432== Command: ./out/main
==5432==
==5432==
==5432== HEAP SUMMARY:
==5432==     in use at exit: 0 bytes in 0 blocks
==5432==   total heap usage: 4,700 allocs, 4,700 frees, 598,504 bytes allocated
==5432==
==5432== All heap blocks were freed -- no leaks are possible
==5432==
==5432== For counts of detected and suppressed errors, rerun with: -v
==5432== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
luke@luke:~/Desktop/Computer_Science/Yr1 Sem 2/Programming Project/comp1921/Cour
sework 2$
```

Testing the program with Dijkstra's algorithm

The next part of the program involved implementing Dijkstra's algorithm to perform on the data input from the text file. In order to test this, I came up with multiple pre-calculated paths and made sure the program returned these correctly:

The tests used can be run using the 'runTests.sh' bash script

Test	Expected output:	Actual output:	Pass / Fail
Test 1 – testing wrong filename input	An error stating that the file at the file path entered could not be found.	'Error: Error opening file, please check the file path provided'.	Pass
Test 2 – running the program with no parameters provided for the file path and start and end node ID's	An error stating that too few parameters were provided.	'Error: Missing arguments 'filePath, start node ID, end node ID''	Pass
Test 3 – running Dijkstra's	An error stating that a path could not be constructed because a	'Error: Path cannot be completed because a node is disconnected from the network'	Pass

algorithm on a disconnected graph	node in the path is disconnected.		
Test 4 – testing an input with node ID's that don't exist	An error stating that the node ID's could not be found.	'Error: Node ID could not be found, please ensure the node exists'	Pass
Test 5 – testing Dijkstra's algorithm on a simple map	Expected path: 1-2-5-4 Expected distance: 7	'Path distance: 7.000000 ---------PATH---------- ID:1, X:2.000000, Y:4.000000 ID:2, X:4.000000, Y:6.000000 ID:5, X:7.000000, Y:2.000000 ID:4, X:9.000000, Y:4.000000 ----------'	Pass
Test 6 – testing Dijkstra's algorithm on another simple graph.	Expected path: 6-1-2 Expected distance: 3	'Path distance: 3.000000 ---------PATH---------- ID:6, X:4.000000, Y:2.000000 ID:1, X:2.000000, Y:4.000000 ID:2, X:4.000000, Y:6.000000 ----------'	Pass
Test 7 – wrong type of node ID entered i.e. text instead of a number	An error stating that the node ID's entered are invalid.	'Error: Node ID could not be found, please ensure the node exists'	Pass
Test 8 – the same start and end nodes are entered	An error stating that the start node and end node cannot be the same.	'Error: Start node cannot be the same as the end node'	Pass
Test 9 – a path that should contain a single edge between two adjacent nodes	Expected path: 1967343264-1615401915 Expected distance: 49.025177	'Path distance: 49.025177 ---------PATH---------- ID:1967343264, X:-1.554946, Y:53.804074 ID:1615401915, X:-1.554947, Y:53.804646 ----------'	Pass
Test 10 -a medium length path between two nodes	Expected Path: -2562,-2560, 1187324666, 1187324679, 54060643, 54060637, 985096817,	'Path distance: 139.481734 ---------PATH---------- ID:-2562, X:-1.554926, Y:53.807634 ID:-2560, X:-1.554711, Y:53.807613	Pass

	985096821, 985096825, -2416, -2380 Expected distance: 139.481734	ID:1187324666, X:-1.554948, Y:53.807527 ID:1187324679, X:-1.554977, Y:53.807365 ID:54060643, X:-1.555217, Y:53.807386 ID:54060637, X:-1.555378, Y:53.807402 ID:985096817, X:-1.555495, Y:53.807059 ID:985096821, X:-1.555548, Y:53.807069 ID:985096825, X:-1.555564, Y:53.807037 ID:-2416, X:-1.556363, Y:53.807154 ID:-2380, X:-1.556347, Y:53.807182 ----------’	

Test 11 – A large path with many edges

Expected Result:
Expected path: -2524, -2540, 247958669, 247958668, 301673229, 247958643, 247958642, 247958641, 247958640, 247958645, 247958646, 247958647, 247958648, 247958649, 247958650, 247958651, 247958653, 247958654, 301673241, 247958598, 247958611, 247958610, 247958608, 247958607, 247958606, 247958605, 247958604, 301673248, 247958601, 247958600, 247958599, 247293219, -2502, 247293220, 247293215, 985096764, 985096813, 985096817, 54060637, 54060643, 1187324679, 1187324666, -2560, 1187324670, 1187324682, 1187324692, 1187324676, 1187324678, 1187324665, 1187324690, 984231425, 54060524, 984231603, 54060515, 54060547, 54060551, 1725582974, 54060555, 54060557, 1668111641, 1668111640
Expected distance: 464.077136

Actual Result:
‘Path distance: 464.077136
---------PATH----------
ID:-2524, X:-1.556024, Y:53.805705
ID:-2540, X:-1.556026, Y:53.805809
ID:247958669, X:-1.556134, Y:53.805909
ID:247958668, X:-1.555860, Y:53.805960
ID:301673229, X:-1.555653, Y:53.805973
ID:247958643, X:-1.555440, Y:53.805993
ID:247958642, X:-1.555420, Y:53.805998
ID:247958641, X:-1.555412, Y:53.805999
ID:247958640, X:-1.555397, Y:53.806004
ID:247958645, X:-1.555393, Y:53.806023
ID:247958646, X:-1.555395, Y:53.806042
ID:247958647, X:-1.555397, Y:53.806051
ID:247958648, X:-1.555404, Y:53.806069
ID:247958649, X:-1.555412, Y:53.806084
ID:247958650, X:-1.555418, Y:53.806096
ID:247958651, X:-1.555426, Y:53.806114
ID:247958653, X:-1.555429, Y:53.806124

ID:247958654, X:-1.555431, Y:53.806137
ID:301673241, X:-1.555433, Y:53.806157
ID:247958598, X:-1.555411, Y:53.806157
ID:247958611, X:-1.555411, Y:53.806227
ID:247958610, X:-1.555400, Y:53.806294
ID:247958608, X:-1.555371, Y:53.806336
ID:247958607, X:-1.555319, Y:53.806361
ID:247958606, X:-1.555260, Y:53.806407
ID:247958605, X:-1.555212, Y:53.806430
ID:247958604, X:-1.555167, Y:53.806458
ID:301673248, X:-1.555178, Y:53.806551
ID:247958601, X:-1.555190, Y:53.806568
ID:247958600, X:-1.555207, Y:53.806595
ID:247958599, X:-1.555217, Y:53.806633
ID:247293219, X:-1.555220, Y:53.806689
ID:-2502, X:-1.555212, Y:53.806754
ID:247293220, X:-1.555209, Y:53.806774
ID:247293215, X:-1.555187, Y:53.806972
ID:985096764, X:-1.555269, Y:53.806968
ID:985096813, X:-1.555247, Y:53.807015
ID:985096817, X:-1.555495, Y:53.807059
ID:54060637, X:-1.555378, Y:53.807402
ID:54060643, X:-1.555217, Y:53.807386
ID:1187324679, X:-1.554977, Y:53.807365
ID:1187324666, X:-1.554948, Y:53.807527
ID:-2560, X:-1.554711, Y:53.807613
ID:1187324670, X:-1.554686, Y:53.807622
ID:1187324682, X:-1.554646, Y:53.807846
ID:1187324692, X:-1.554574, Y:53.808077
ID:1187324676, X:-1.554412, Y:53.808180
ID:1187324678, X:-1.554382, Y:53.808301
ID:1187324665, X:-1.554195, Y:53.808311
ID:1187324690, X:-1.554048, Y:53.808346
ID:984231425, X:-1.554026, Y:53.808717
ID:54060524, X:-1.553937, Y:53.808757
ID:984231603, X:-1.554091, Y:53.808897
ID:54060515, X:-1.554290, Y:53.809044
ID:54060547, X:-1.554369, Y:53.809073
ID:54060551, X:-1.554168, Y:53.809208
ID:1725582974, X:-1.554076, Y:53.809261
ID:54060555, X:-1.554034, Y:53.809306
ID:54060557, X:-1.553964, Y:53.809354
ID:1668111641, X:-1.554272, Y:53.809558
ID:1668111640, X:-1.554399, Y:53.809494
----------’

Then I checked that the program wasn't leaking memory using valgrind:

```
250225342
1659456268
151917549
21545962
1659456381
664212051
21069425
249661995
21069424
247957367
244449810
247293286
--------------
==31892==
==31892== HEAP SUMMARY:
==31892==     in use at exit: 0 bytes in 0 blocks
==31892==   total heap usage: 4,713 allocs, 4,713 frees, 730,552 bytes allocated
==31892==
==31892== All heap blocks were freed -- no leaks are possible
==31892==
==31892== For counts of detected and suppressed errors, rerun with: -v
==31892== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
luke@luke:~/Desktop/Computer_Science/Yr1 Sem 2/Programming Project/comp1921/Cour
sework 2$
```

Testing the programs output to use with gnuplot

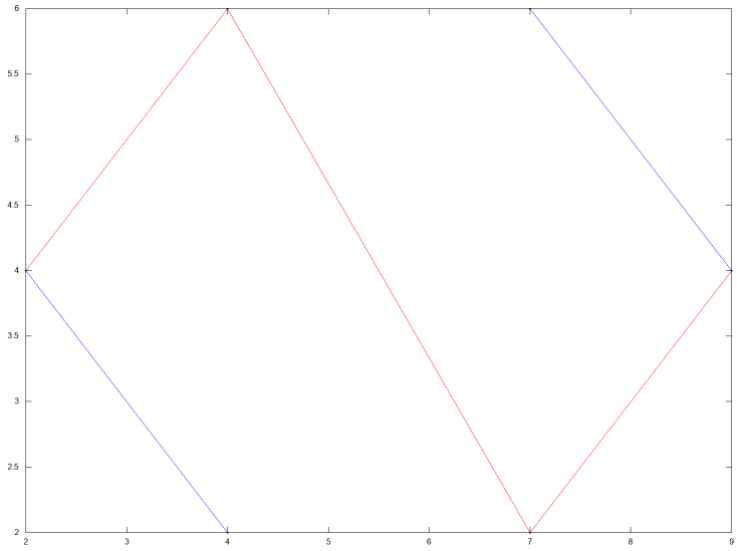
The last part of the program is creating a way to visualize the data collected. This uses an application called gnuplot which can be used to create graphs and charts. It was easy to test whether this part of the application worked correctly as the output should be a network with certain edges colours differently denoting a path. I tested this using the paths calculated above to make sure the program worked correctly.

I then checked to make sure the program contained no memory leaks for each path tested, this is an example of one such path:

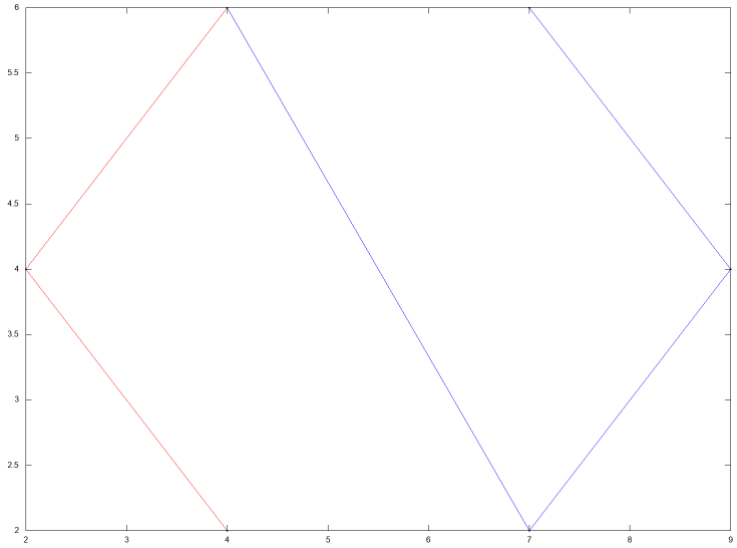
```
1659456268
151917549
21545962
1659456381
664212051
21069425
249661995
21069424
247957367
244449810
247293286
--------------
Writing network...
==31916==
==31916== HEAP SUMMARY:
==31916==     in use at exit: 0 bytes in 0 blocks
==31916==   total heap usage: 4,715 allocs, 4,715 frees, 735,200 bytes allocated
==31916==
==31916== All heap blocks were freed -- no leaks are possible
==31916==
==31916== For counts of detected and suppressed errors, rerun with: -v
==31916== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
luke@luke:~/Desktop/Computer_Science/Yr1 Sem 2/Programming Project/comp1921/Cour
sework 2$
```

The program has been designed to output a png file giving a visual representation of the network and path. In the tests implemented above here are the png files produced:

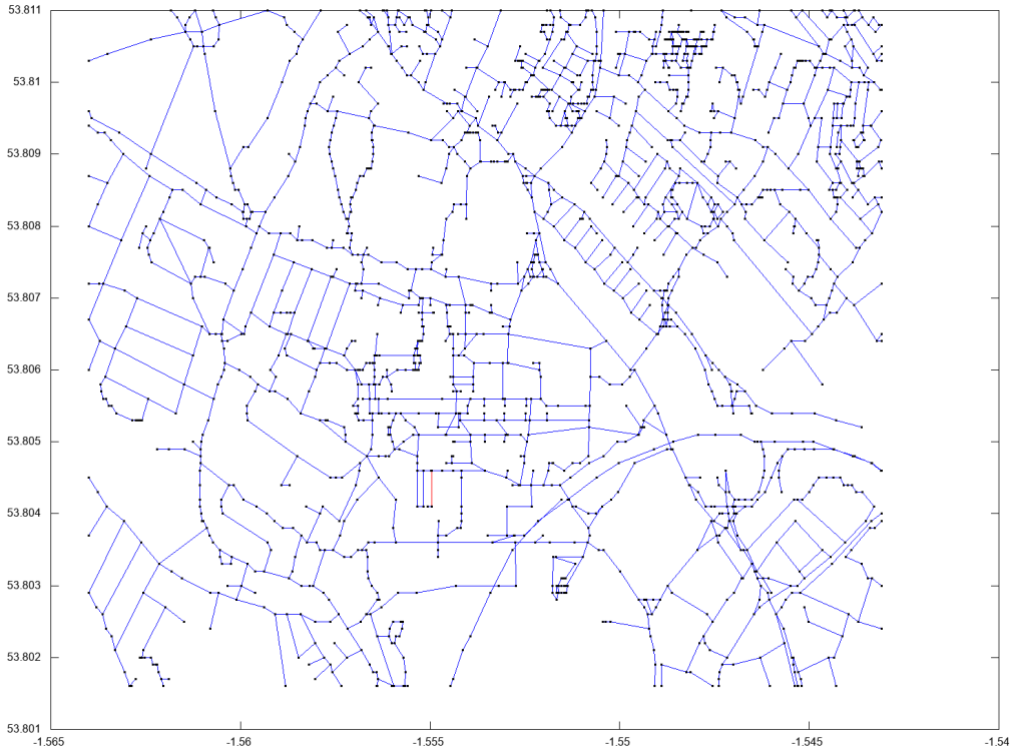
Test 5



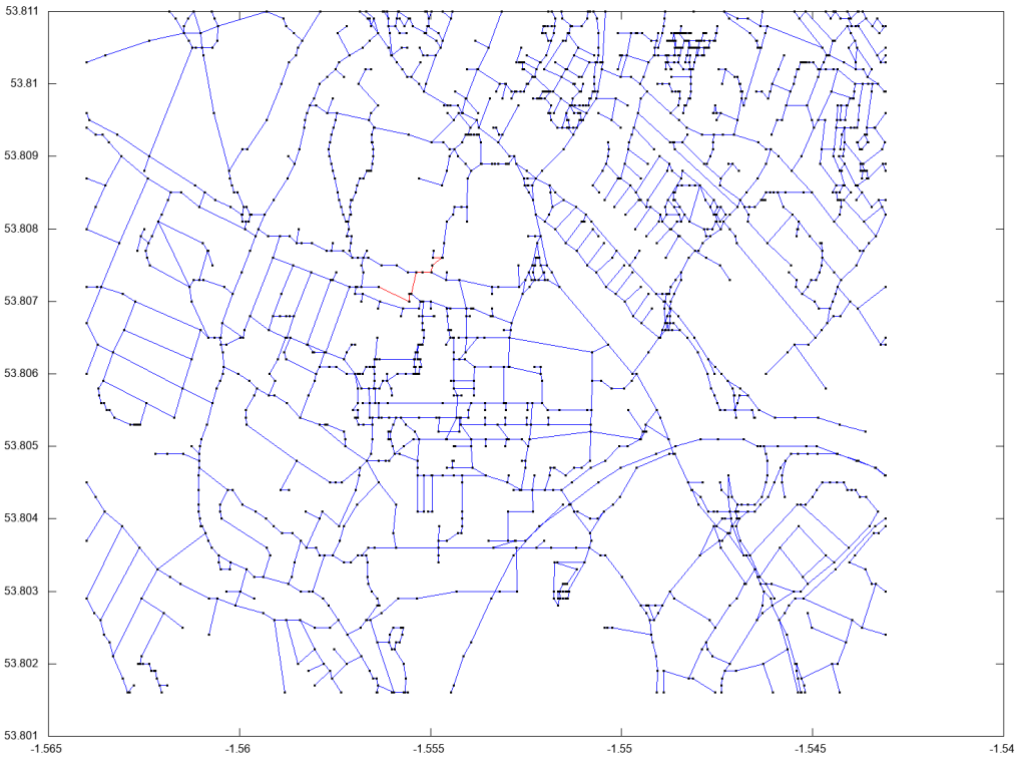
Test 6



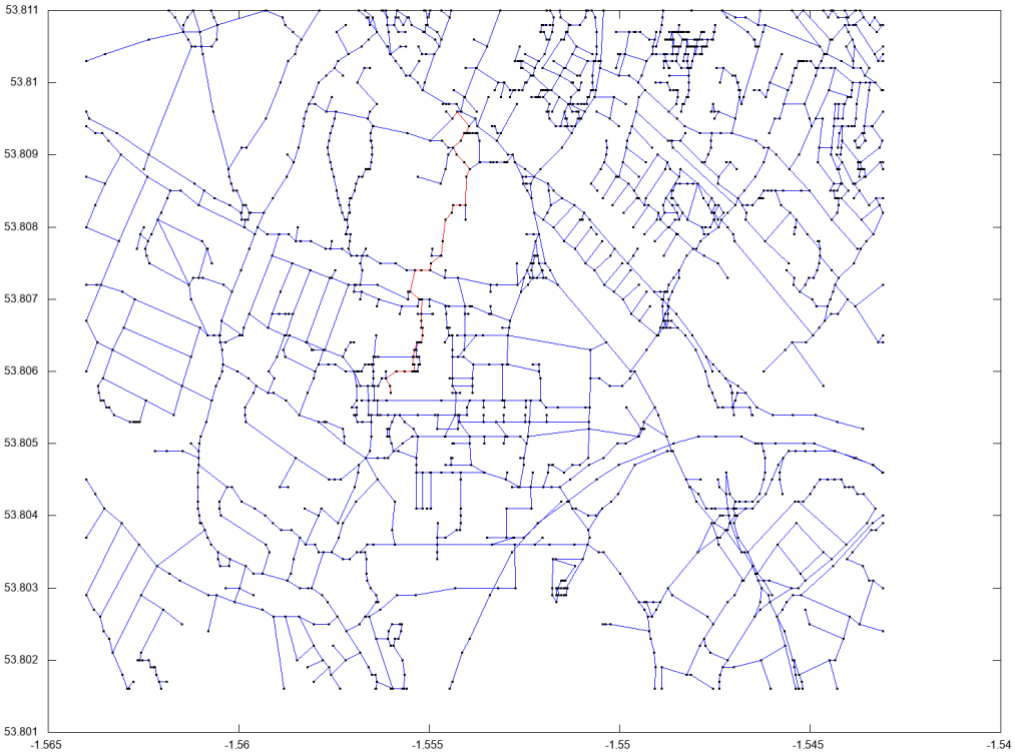
Test 9



Test 10



Test 11


53.811
53.81
53.809
53.808
53.807
53.806
53.805
53.804
53.803
53.802
53.801
-1.565
-1.56
-1.555
-1.55
-1.545
-1.54

Reflection

What have you learned from the process of completing this project?

Completing this project, I learnt many new skills. One main thing this project taught me was how to structure a large program and the importance of splitting code down into modules or separate files. This helps improve the organisation and overall quality of the application, making it easy to maintain and edit code in the future. It also helped improve my understanding of networks and the best data structures to use with them. This is important as it makes the application efficient even when the network is large containing many nodes and edges. I have also learned about the importance of design when developing a program. It is important to think about what the program needs to achieve and what data structures are going to be used before you start developing the application.

What do you feel are your strengths and weaknesses in this regard?

Strengths – I feel like the application is organised well, making it easy for future maintenance. The fact that all data related to the network is stored in a struct makes the process of getting the data that you want (e.g. a node from the array) more intuitive and it allows multiple networks to be created at once if this is a need in the future.

Weaknesses – Currently the application doesn't allow directed edges. This is because it doesn't make sense for a path for pedestrians to be one way. It also doesn't allow for negative edge weights. These changes could be implemented with adaptations to the algorithm used to find the shortest path fairly easily, however they are not needed in the context the program is needed for. However, this decreases the number of real life applications this program would be useful for.

Another improvement that could be made to the application is to introduce heuristics when finding the shortest path. This will improve the speed of finding the shortest path on larger networks.

What can you use from this in future programming work on the degree and otherwise?

The main thing I will take away from this project is to fully design the application before beginning to develop it. Making sure I have a clear idea how elements will interact with each other and what actually needs to be output. I spent a lot of time rewriting code because I changed my mind on how to piece various modules together.

References

[1] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm Retrieved 22/3/2018
[2] https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html Retrieved 22/3/2018
[3] https://en.wikipedia.org/wiki/Fibonacci_heap Retrieved 22/3/2018
[4] https://en.wikipedia.org/wiki/Heap_(data_structure) Retrieved 25/3/2018
[5] https://troydhanson.github.io/uthash/ Retrieved 21/4/2018
[6] http://valgrind.org/ Retrieved 23/4/2018