

## Analysis

### Background Information

We use google maps on a daily basis to provide us a route from one destination to another. The online software provides the most efficient route from point A to B. What if we had this kind of software for other uses and not just for routes around the world. This is why I came up with the project idea of a route finder around my school, Ernest Bevin College. I remember coming to this school and struggling to find the fastest routes around the school with the only help I received being from other students or teachers. Most students and teachers would be busy teaching or working, therefore asking them for directions may distract them, and not every student is going to know the exact and fastest routes around the school. Reported in 2012 there was approximately 1275 pupils within the school, most likely increased by a few hundred's in the past 5 years. This means the more the students the more the students are crowded in one area within the corridors or stairs.

As a member of the school since year 7 I have experienced several traffic and issues migrating around the school class rooms. Year 7 was most definitely the most chaotic year with the lack of knowledge to the shortest route around the school and the older students who would push around year 7s around the corridor. The corridors are always crowded, I mean imagine approximately 100 students trying to go up or down the stairs of one of the floors. With only 2 staircases within the main building the routes are limited. However, with more students going through one set of staircases the other would not be as packed. There are several issues with this crowd created every end/beginning of the period. The problems involve:

- Students may get hurt
- Teachers may be barged
- Crowds create a higher chance of theft
- Loss of items
- Items within bag may be damaged
- Scuffles occur between pupils
- Late to attend lessons



As you can see from the above image this is one of many hot spots around the school filled with traffic. (Taken at approximately 3:35 end of period 6) This would be much worse during the shifts of periods especially at period 1)

My project is a routing and pathing tool for around my school, Ernest Bevin College. This project will help students and teachers get around the school. The program will consist of the simplest and most efficient route from point A to point B. Let's say for example a student starts at Room 304 (on the 3<sup>rd</sup> floor) and would like to reach Room 103 (on the 1<sup>st</sup> floor), the program will provide the shortest route to get from Room 304 to Room 103 and where there would be less traffic. The pathing function will be created using Dijkstra's algorithm and taking a heuristic approach.

A lot of the routes around the school are not always accessible due to either the doors being locked or doors only being accessible by members of staff. This has proved to be a frustrating problem for those who tend to walk through certain areas of the school as they are closed off and other routes around to get there may take longer or are unknown.

## Current Systems



Our school has attempted to facilitate a system that would make it easier to navigate the school. For example, the idea of always sticking to the left while going up or down the stairs to reduce the clusters and create formal lines. This has been attempted several times before and has proven to work at times but has never fully successfully worked, this is due to the lack of communication and lack of responsive students. The system involves each student sticking to the left side at all times, from the stairs to the corridors. It is especially hard for this to work in corridors as students tend to cluster outside their lesson waiting for the teachers to allow them in. The only real solution for this is for the teachers to allow students to enter from 5 mins before the lesson begins. This allows students to enter straight in and reduce clusters and traffic outside the room. However, this may not be possible if the teacher was having a lesson previously. I will implement my program around the idea that the students follow these current systems. I will also ensure to remind the students to stick to one side of the corridor or stairs.



As you can see from this image, our school has put up one on each stair case to show the students where their lessons are. However, this does not show which direction to take other than just telling you which floor it is on. This is also not present throughout the school building and mostly in the main building.

Currently there are no computerised solutions for the problem at hand. Students would most likely ask around other students or teachers to get around the school. This is a good idea but with students constantly forgetting and asking around every day around the school

the task becomes a bit frustrating for teachers and/or other students. Teachers are also distracted by their class to know how long a student has gone for a toilet break, with a system that will calculate the distance between the class room and the closest toilet the teacher will easily be able to make sure the student returns in time to be able to continue with his/her studies and lose less. Some students around the school may not be as social or confident as other students this may be difficult for them to ask other students or teachers and therefore not as efficient.

Also, at the beginning of the year for a year 7 student, they are introduced around the school and guided to where and what the departments are. As efficient and normal this procedure is, the chances of the students remembering the routes are not that high. They are also not shown the hotspots around the school that will appear once school officially starts. This is also not highly accurate as they know where the departments are but maybe not where the exact rooms are, therefore not unique to each student's time table. This is why I am choosing to have a main focus on assisting year 7 pupils rather than the entire school.

On a larger scale however, current systems such as Google maps exist. The product is free and provides routes all around the world and the time it takes to reach from point A to B. The software was designed 2005 and has been constantly updated with newer features ever since. However, in relation to my task at hand, Google maps is more global approach to my software, providing information on what bus to take, to information on what other means of transport and the duration for those. Obviously, I will not need any of these implementations for a route-finding tool around a school.

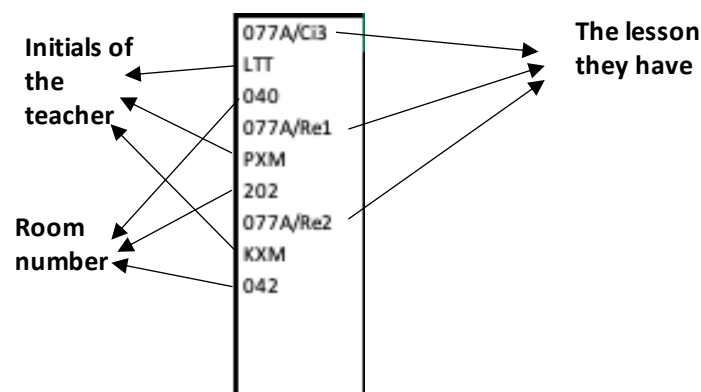
## What I would need:

For my project to be successful I would need a lot of data regarding the distances from the beginning of a corridor to another or the distance down/up stairs. All this data will be easy to obtain using a laser measurer. I would also need data of where students will be walking to depending on their timetable. With over 1200 students this requires filtering through an excel file to create paths for every student. This will ensure that I can get the best possible routes around the school with the least amount of traffic around the school. I will obtain the data from within the SIMs tool accessible by my client, Miss Iweha. Here is a screenshot of the data I will be using:

**The days and periods**

**The set group**

|     | Mon:1   | Mon:2   | Mon:3   | Mon:4   | Mon:5   | Mon:6  | Mon:7 | Tue:1  | Tue:2   | Tue:3  | Tue:4  | Tue:5  | Tue:6  |
|-----|---|---|---|---|---|--|-------|--|---|--|--|--|--|
| 07a | 077A/Ci3<br>LTT<br>040<br>077A/Re1<br>PXM<br>202<br>077A/Re2<br>KXM<br>042                          | 077A/Gg3<br>LXS<br>315<br>077A/Ma1<br>ED<br>306<br>077A/Mi2<br>ACC<br>208                           | 077A/A1<br>RMP<br>017<br>077A/Hi2<br>SW<br>314<br>077A/Mi3<br>JY<br>210 | 077A/E1<br>JH<br>207<br>077A/E2<br>AO<br>041<br>077A/E3b<br>HL<br>211   | 077A/Ma2<br>YC<br>037<br>077A/Sc3<br>JAM<br>032<br>077A/Sc1<br>CS<br>314  | 077A/Mu1<br>EA<br>014<br>077A/Sc2b<br>SSS<br>023<br>077A/Sc3<br>JAM<br>032 |       | 077A/E2<br>AO<br>041<br>077A/Ma1<br>ED<br>306<br>077A/Sc3<br>JAM<br>032  | 077A/E3a<br>LXO<br>211<br>077A/Gg1<br>FON<br>311<br>077A/Ma2<br>YC<br>037 | 077A/Ad1<br>LDM<br>101<br>077A/Te1<br>EG<br>020<br>077A/Te2<br>EAM<br>010<br>077A/Te3<br>WL<br>102 | 077A/Ad1<br>LDM<br>101<br>077A/Te1<br>EG<br>020<br>077A/Te2<br>EAM<br>010<br>077A/Te3<br>WL<br>102 | 077A/E1<br>JH<br>207<br>077A/Gg3<br>LXS<br>315<br>077A/Mi2<br>ACC<br>208   | 077A/Di2<br>JK<br>012<br>077A/Ma3<br>MBM<br>304<br>077A/Mi1<br>ACC<br>208  |
| 07b | 077B/Ad1<br>EP<br>009<br>077B/Te1<br>RMP<br>017<br>077B/Te2<br>HHD<br>102<br>077B/Te3<br>EAM<br>010 | 077B/Ad1<br>EP<br>009<br>077B/Te1<br>RMP<br>017<br>077B/Te2<br>HHD<br>102<br>077B/Te3<br>EAM<br>010 | 077B/Di1<br>JK<br>012<br>077B/E3<br>KAC<br>214<br>077B/Ma2<br>ED<br>306 | 077B/E2<br>CE<br>204<br>077B/Ma1<br>YC<br>037<br>077B/Ma3<br>MBM<br>306 | 077B/E1<br>JSJ<br>212<br>077B/Hi2<br>ZT<br>312<br>077B/Sc3b<br>ACC<br>030 | 077B/A2<br>RMP<br>017<br>077B/Hi3<br>MAS<br>312<br>077B/Mi1a<br>ACC<br>208 |       | 077B/E3<br>KAC<br>214<br>077B/Gg2<br>LXS<br>315<br>077B/Ma1<br>YC<br>037 | 077B/Di3<br>JK<br>012<br>077B/E1<br>JSJ<br>212<br>077B/Mi2<br>ACC<br>208  | 077B/Cp1<br>JLS<br>110<br>077B/E2<br>CE<br>204<br>077B/Gg3<br>JAB<br>040                           | 077B/Cp1<br>JLS<br>110<br>077B/Mi3<br>AHW<br>206<br>077B/Sc2<br>JAB<br>030                         | 077B/Cp2<br>HES<br>117<br>077B/Sc1<br>KL<br>028<br>077B/Sc3a<br>JAB<br>030 | 077B/Cp2<br>HES<br>117<br>077B/Sc1<br>KL<br>028<br>077B/Sc3a<br>JAB<br>030 |
| 07c | 077C/Cp1<br>SV<br>121   | 077C/Ci1<br>KXM<br>042  | 077C/Te1<br>EAM<br>010  | 077C/Te1<br>EAM<br>010  | 077C/Sc1a<br>SNS<br>024   | 077C/Sc1a<br>SNS<br>024  |       | 077C/Di1<br>JK<br>012  | 077C/Gg1<br>LTT<br>040  | 077C/Sc1b<br>SNS<br>024  | 077C/E1<br>CIW<br>207  | 077C/Ma1<br>GO<br>038  | 077C/Ad1<br>RMP<br>017   |



A lot of the contents within the excel file are not useful for my project, and will have to get filtered out and will be done through a simple reading function within the program. I could also filter it out myself which would be much easier but takes longer. However, I chose not to do this as it would be much more efficient to have my program sort it as the timetables would change yearly and for some students more often (due to shifts within tutor groups or

set groups for certain subjects). This would result in new routes being created every year. I will need the room at which each lesson is and what lesson is next, and due to the fact this project is mainly focused on year 7s I would also need the names of the initials of their tutors which is not provided within this file but can easily be linked to each individual rows of the timetable above. The initials of each teacher is an unneeded for my program and will need to be filtered out.

For example:

**The days and periods** – This bit of data is necessary as it represents the day in which the lessons are going on as well as which period they take place on. School begins at 8:50, Period 1 begins at 9:05, Period 2 begins at 9:55, Break begins at 10:45, Period 3 begins at 11:05, Period 4 begins at 11:55, Lunch begins at 12:45, Tutor period begins at 1:40, Period 5 begins at 2:00, Period 6 begins at 2:45, Period 7 begins at 3:30 till 4:30. These will all come in necessary to figure out from which point the students are going to move to, for example students will move from period 1 to 2 not period 3 to 5 straight. The days will be helpful to identify when the program stops reading as it has found all the routes for the day.

**The set groups** – Within our school, the years are split into 3 different sets and represent the ability of the students. There are 3 classes for set a, 3 classes for set b, and 1 class for set c. This bit of data will not come in handy for my program, and therefore will not be used.

**The lesson they have** – ‘077A/Re1’ represents both the set of the group as well as the subject in which the students are studying. For example ‘077A’ represents that the student is of set A of year 7, the “Re1” represents that the students are in the first class of RE (Religious Education)

**Room number** – This is the most crucial bit of information that this data set provides. This bit of data provides me the location of which each individual lesson is taking place. This allows me to plot in the starting node and ending node for each period. For example, I will have to find the route between room 40 and 315 as they finish period 1 in room 40 and have to get to room 315 to begin their next lesson. I will also have to take into consideration that once period 2 is over break begins and therefore students tend to go to the main hall within the school or the canteen, this also applies for after period 4 when lunch starts. Tutor time must also be taken into consideration as students will have to report back to their tutor bases.

The information that this data set does not have is the tutor bases of each individual tutor groups. For example, 7ABA is the tutor group of Mr Abdullah who resides in the room 39. Luckily however I have been provided the data of the tutor groups:

| YEAR 7 - YELLOW<br>AR / OO / CI                     |     |     |
|---|-----|-----|
| Business Studies,<br>Computing, Economics,<br>Maths |     |     |
| Adam Razaq  |     |     |
| Junaid Shaikh<br>7JRS                               | 35  | 7A1 |
| Abdiqani Abdalla<br>7ABA                            | 39  | 7A2 |
| Geoffrey Opoku<br>7GO                               | 38  | 7A3 |
| Youssef Chabibi<br>7YC                              | 37  | 7B1 |
| Jayshree Sajadah<br>7JLS                            | 110 | 7B2 |
| Ghassan Suleiman<br>7GS                             | 108 | 7B3 |
| Alexandra Newnham<br>7AN                            | 307 | 7C  |

| Timetable - 2<br>as at 20/12/2018 |          |          |          |          |           |           |
|-----------------------------------|----------|----------|----------|----------|-----------|-----------|
|                                   | Mon:1    | Mon:2    | Mon:3    | Mon:4    | Mon:5     | Mon:6     |
| 07a                               | 077A/Ci3 | 077A/Gg3 | 077A/A1  | 077A/E1  | 077A/Ma2  | 077A/Mu1  |
|                                   | LTT      | LXS      | RMP      | JH       | YC        | EA        |
|                                   | 040      | 315      | 017      | 207      | 037       | 014       |
|                                   | 077A/Re1 | 077A/Ma1 | 077A/Hi2 | 077A/E2  | 077A/Sc3  | 077A/Sc2b |
|                                   | PXM      | ED       | SW       | AO       | JAM       | SSS       |
|                                   | 202      | 306      | 314      | 041      | 032       | 023       |
|                                   | 077A/Re2 | 077A/Mi2 | 077A/Mi3 | 077A/E3b | 077A/Sk1  | 077A/Sc3  |
|                                   | KXM      | ACC      | JY       | HL       | CS        | JAM       |
|                                   | 042      | 208      | 210      | 211      | 314       | 032       |
| 07b                               | 077B/Ad1 | 077B/Ad1 | 077B/Dr1 | 077B/E2  | 077B/E1   | 077B/A2   |
|                                   | EP       | EP       | JK       | CE       | JSJ       | RMP       |
|                                   | 009      | 009      | 012      | 204      | 212       | 017       |
|                                   | 077B/Te1 | 077B/Te1 | 077B/E3  | 077B/Ma1 | 077B/Hi2  | 077B/Hi3  |
|                                   | RMP      | RMP      | KAC      | YC       | ZT        | MAS       |
|                                   | 017      | 017      | 214      | 037      | 312       | 312       |
|                                   | 077B/Te2 | 077B/Te2 | 077B/Ma2 | 077B/Ma3 | 077B/Sc3b | 077B/Mi1a |
|                                   | HHD      | HHD      | ED       | MBM      | JAB       | ACC       |
|                                   | 102      | 102      | 306      | 306      | 030       | 208       |
|                                   | 077B/Te3 | 077B/Te3 |          |          |           |           |
|                                   | EAM      | EAM      |          |          |           |           |
|                                   | 010      | 010      |          |          |           |           |
| 07c                               | 077C/Cp1 | 077C/Ci1 | 077C/Te1 | 077C/Te1 | 077C/Sc1a | 077C/Sc1a |
|                                   | SV       | KXM      | EAM      | EAM      | SNS       | SNS       |
|                                   | 121      | 042      | 010      | 010      | 024       | 024       |

As you can see the image on the left provides the list of tutor groups and where they are. I can simply imagine there is another column before period 1 for the tutor groups. This shows the start node for the route from morning registration to period 1.

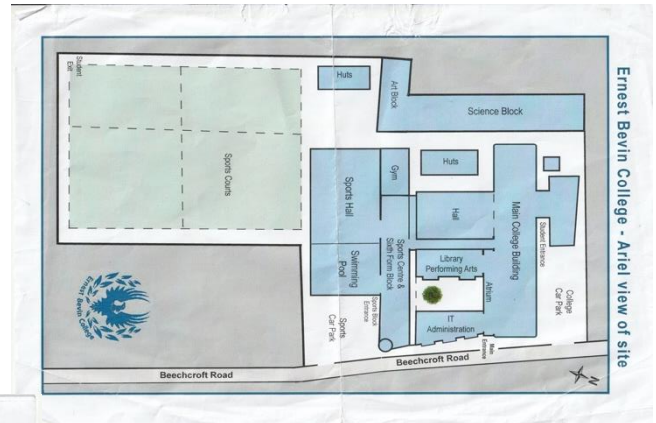
The image on the left is of all the excel file but formatted to provide an idea of which of the rows are which tutor groups, can be seen by the red writing.

I would also need the map of the school to be able to understand the ratio between each classroom and the paths that can be taken. Unfortunately, the measurements for the rooms were not provided, therefore I would need to collect this information myself by walking around with a laser measurer from room to room.

Furthermore, if I can I will attempt to further my program by creating an interface to see these maps and the routes provided with all other possible routes.



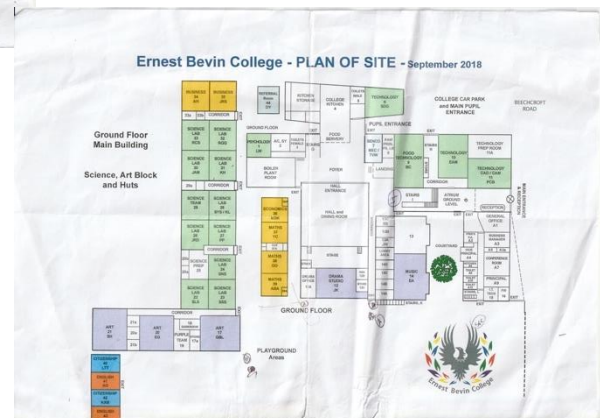
As you can see this is a blueprint view of the entire school from above, this is useful for my project as it will help indicate the entrances to the school and the many shortcuts around the school such as the courtyard.



This is a much more useful map that indicates all 3 floors of the building and provides the corridors that I would need to know about in order for the students to walk through.

This image is a map of the ground floor within the school. Our school has separate buildings for some subjects. There for the use of the 3 exit points from the building are needed so I will be able to divide the routes.

These maps are mostly useful for me as it provides me the routes around the school for me to obtain the measurements.



One issue is that my school does not have data of the distance of rooms etc. This is a major throw back as some of the distances I will collect may not be accurate. However, if the user is of another school which have this data then it will be easy for them to use the program as long as they follow the format within the text file.

Another issue is that there are not names for certain points of the map and they can be crucial for certain routes. Therefore, I have created random names for these nodes such as "I", "A", "A!" etc. The user would not understand what these nodes identify as, therefore I will change the name later when a more suitable name can be appointed.

All my data of the routes will need to be easy to access therefore I have come up with the solution to print out the information of the routes at the back of the timetables provided at the beginning of the year. It would look something like this (left is current timetable and right is of the routes).



## Dijkstra's Algorithm

My program will need to use a searching algorithm in order to find the distance from point A to B. This can be done by several search algorithms such as Dijkstra's.

Dijkstra's is an algorithm is an algorithm to find the shortest path between nodes in a graph, which may represent, road networks or a maze but in my case, my school. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. In my case it will be used to find the shortest path from a single node to a single destination node by stopping the algorithm once the shortest path to the destination has been found. In my case, the nodes in the graph represents rooms/ main points throughout Ernest Bevin College, and therefore the and the edges are the distances from each destination, Dijkstra's algorithm can be used to find the shortest route between one room and all other rooms.

The algorithm works as followed:

Step 1: Marks all nodes as unvisited and creates a set of all unvisited nodes.

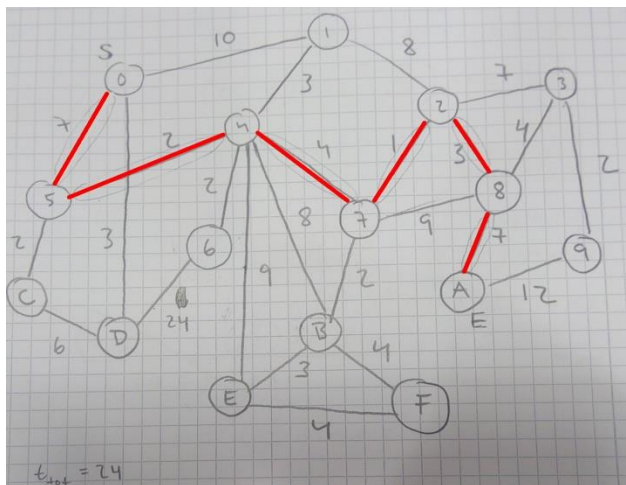
Step 2: Assigns every node an unfixed distance value (so set it to 0 for our initial node and infinity for all other nodes)

Step 3: Consider all of the unvisited neighbours and calculate their fixed distances through the current node. Compare the newly calculated distance to the current assigned value and assign the smallest value.

Step 4: After all unvisited neighbours are considered we mark the current node as visited and remove from the set of unvisited nodes. Then move on to the next unvisited node with the shortest distance that we found from step 3 and check all distances for those neighbours. This repeats until target node is met.

Step 5: Checks whether the target node has been marked as visited and if so then stop.

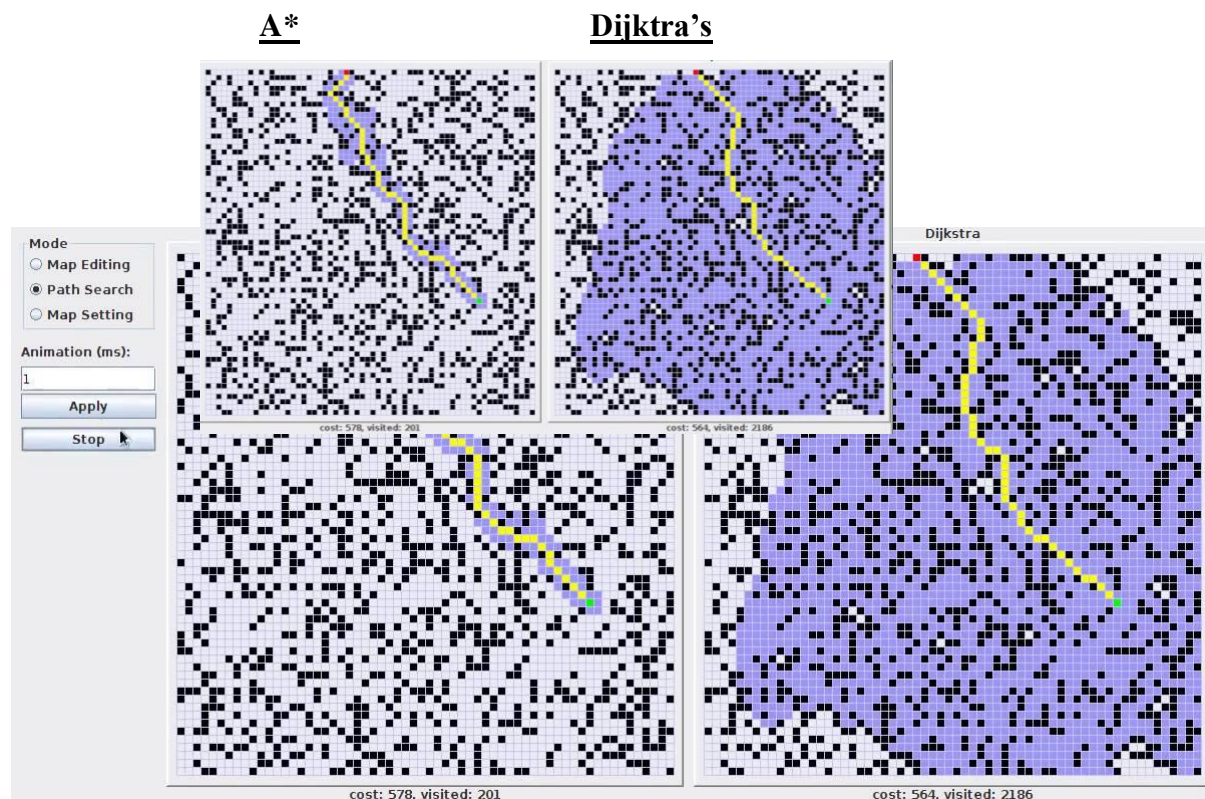
Dijkstra's original algorithm has the run time of  $O(|n|^2)$ , where n is the number of nodes.



As you can see this is a small example of how Dijkstra's works in graph form. The problem I am facing requires a lot more nodes and larger float values.

## Other possible searching algorithms and why I chose not to use them:

A\* algorithm – Uses best-first search and therefore is faster than dijkstra's, however If you have many target nodes and you don't know which one is closest to the main one, A\* is not very optimal. This is because it needs to be run several times (once per target node) in order to get to all of them. A\* expands on a node only if it seems promising. It's only focus is to reach the goal node as quickly as possible from the current node, not to try and reach every other node.



Bellman Ford's Algorithm - This algorithm depends on the relaxation principle where the shortest distance for all vertices is gradually replaced by more accurate values until eventually reaching the optimum solution. In the beginning all vertices have a distance of "Infinity", but only the distance of the source vertex = 0, then update all the connected vertices with the new distances (source vertex distance + edge weights), then apply the same concept for the new vertices with new distances and so on.

The main reason for why I have not chosen to implement these algorithms into my program is due to my lack of understanding of them and I am more familiar with Dijkstra's due to it being a part of the syllabus. Dijkstra's is also more efficient due to the fact it checks all possible routes whereas A\* is a greedy algorithm.





Here is the graph but translated into written and more understandable form similar to a relationship matrix also includes the values (distance from one point to another):

|   |   |
|---|---|
| <p>Stairs A (Rm 301-305: 5, Rm 306-311: 14, Stairs C: 9)</p> <p>Rm 301-305 (Stairs A: 5)</p> <p>Rm 306-311 (Stairs A: 14, Stairs B: 15)</p> <p>Stairs B (Rm 306-311: 14, Rm 310-315: 5, Stairs D: 9)</p> <p>Rm 310-315 (Stairs B: 5)</p> <p>Stairs C (Stairs A: 9, Rm 201-204: 5, Rm 205-211: 15, Stairs E: 9)</p> <p>Rm 201-203 (Stairs C: 5)</p> <p>Rm 205-211 (Stairs C: 14, Stairs D: 15)</p> <p>Stairs D (Rm 205-211: 14, Rm 212-217: 5, Stairs F: 9)</p> <p>Rm 212-217 (Stairs D: 5)</p> <p>Stairs E (Rm 101-104: 5.5, Rm 108-110: 7, Stairs G: 9)</p> <p>Rm 101-102 (Stairs E: 5.5)</p> <p>Rm 108-110 (Stairs E: 7, Stairs F: 9, Staff: 15.3, Stairs I: 14)</p> <p>Staff (Stairs F: 9, Rm 108-110: 9, Upper Atrium: 9.2, Rm 108-110: 8.1)</p> <p>Rm 116 (Staff: 2, Upper Atrium: 6.5)</p> <p>Stairs I (Library: 3, Upper Atrium: 19, Bottom Atrium: 9.2, Rm 108-110: 8.1)</p> <p>Library (Upper Atrium: 19, Stairs I: 3)</p> <p>Upper Atrium (Staff: 6.5, Rm 116: 6.5, Rm 117-122: 10.3)</p> <p>Rm 117-122 (Upper Atrium: 10.3)</p> <p>Stairs F (Rm 108-110: 9, Staff Room: 9, Stairs H: 9)</p> <p>Stairs H (Rm 9: 9, Rm 10-11: 12, Bottom Atrium: 12, Stairs F: 9)</p> <p>Rm 9 (Stairs H: 9, Rm 10-11: 12, Bottom Atrium: 7.9)</p> <p>Rm 10-11 (Stairs H: 12, Bottom Atrium: 9, Rm 9: 10)</p> <p>Bottom Atrium (I: 11, Reception: 5.5, Bottom Courtyard Exit: 3, Stairs H: 12, Rm 10-11: 9, Rm 9: 7.9, Offices: 5.6)</p> <p>Reception (Bottom Atrium: 5.5)</p> <p>Offices (Bottom Atrium: 5.6)</p> <p>I (Bottom Atrium: 9, Rm 13: 13, Landing: 3.8, Rm 9: 5.4, Foyer: 7.9)</p> <p>Landing (I: 3.8, Foyer: 7.4)</p> <p>Foyer (Canteen: 6.2, Landing: 7.4, I: 7.9, Hall: 5.2, Pupil: 12, Rm 1-3: 9, Stairs G: 8)</p> <p>Canteen (Stairs G: 7, Hall: 7, Pupil: 9)</p> <p>Pupil (Canteen: 5, A Toilet: 9)</p> <p>A Toilet (Pupil: 5)</p> <p>Hall (Foyer: 5.2)</p> | <p>Stairs G (Stairs E: 9, Rm 1-3: 9, Foyer: 7.9, Hall Exit: 7)</p> <p>Rm 1-3 (Stairs G: 9, Hall Exit: 7, Foyer: 7.9)</p> <p>Hall Exit (Stairs G: 9, Rm 1-3: 9, Stairs G: 9, Sci Corr 2: 1, Rm 36-37: 1)</p> <p>Rm 36-37 (Hall Exit: 1, Rm 38-39: 1)</p> <p>Back G (Stairs G: 9, Referral (A): 2.75, Sci Corr 1: 1.8)</p> <p>Rm 38-39 (A: 1, Rm 36-37: 1)</p> <p>Referral (A) (Stairs G: 9, Sci Corr 1: 1.8)</p> <p>Sci Corr 2 (Hall Exit: 1, Sci Corr 1: 2.1, Sci Corr 3: 1)</p> <p>Sci Corr 1 (Back G: 1.8, Referral (A): 2.75, Sci Corr 2: 2.1)</p> <p>Sci Corr 3 (Sci Corr 2: 1.8, Sci/Art Corr: 1.9)</p> <p>Sci/Art Corr (Sci Corr 3: 1.9, A: 1.5, B: 1)</p> <p>B (Rm 17: 17, Rm 40-43: 5.2)</p> <p>Rm 17 (B: 17, Rm 40-43: 5.2)</p> <p>Rm 40-43 (B: 5.2, Rm 17: 17)</p> <p>A (Rm 38-39: 1, Sci/Art Corr: 1.9, Gym: 2.5, A: 1.62)</p> <p>Rm 13-14 (I: 13)</p> <p>Rm 14 (Rm 12-13: 12.4, Rm 15: 1)</p> <p>Rm 12 (A: 1.5, Rm 14: 12.6)</p> <p>A (Courtyard (s): 4.8, Rm 12: 13, SFC: 1.62)</p> <p>Courtyard (s) (C: 2.33, Medical: 1.5, Sports Hall: 4.8, Concourse: 1.75, Stairs: 3.2)</p> <p>Medical (Courtyard (s): 1.5, Sports Hall: 4.8)</p> <p>Sports Hall (Medical: 4.8, Courtyard (s): 1.5)</p> <p>Concourse (SFC: 1.62, Courtyard (s): 1.75)</p> <p>Gym (Sports Hall: 4.8)</p> <p>Courtyard (s) (C: 2.33)</p> <p>C (Courtyard Exit: 2.7, Courtyard (s): 2.33, SFC: 1.62)</p> <p>Stairs S (Courtyard (s): 3.2, Concourse: 1.75, Stairs T: 9)</p> <p>Stairs S (Stairs S: 9.1, Dejo: 0.5, CH 106-107: 12, Rm 105-106: 18, R 23)</p> <p>Dejo (Stairs S: 9.1)</p> <p>CH 106-107 (Stairs S: 9.1)</p> <p>Rm 105-106 (Stairs S: 9.1)</p> <p>Stairs T (Stairs S: 9.1)</p> <p>Stairs S (Stairs S: 9.1, SFC: 1.62, C: 1.62)</p> |
|---|---|

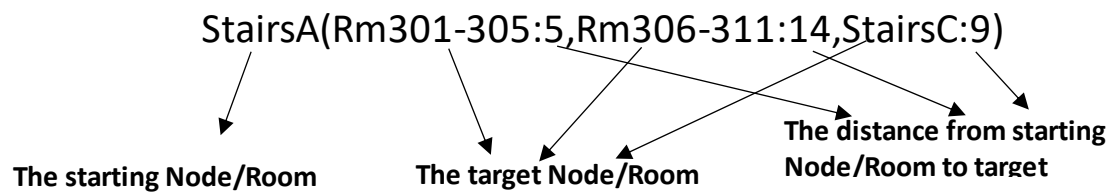
The distances were collected by me.

Below is the format in which I have typed up the data into a text file so I will be able to access all the nodes and distances for my project:

```

Rooms.txt
StairsA(Rm301-305:5,Rm306-311:14,StairsC:9)
Rm301-305(StairsA:5)
Rm306-311(StairsA:14,StairsB:15)
StairsB(Rm306-311:14,Rm310-315:5,StairsD:9)
Rm310-315(StairsB:5)
StairsC(StairsA:9,Rm201-204:5,Rm205-211:15,StairsE:9)
Rm201-203(StairsC:5)
Rm205-211(StairsC:14,StairsD:15)
StairsD(Rm205-211:14,Rm212-217:5,StairsF:9)
Rm212-217(StairsD:5)
StairsE(Rm101-104:5.5,Rm108-110:7,StairsG:9)
Rm101-102(StairsE:5.5)
Rm108-110(StairsE:7,StairsF:9,Staff:15.3,StairsI:14)
Staff(StairsF:9,Rm108-110:9,UpperAtrium:9.2,Rm108-110:8.1)
Rm116(Staff:2,UpperAtrium:6.5)
StairsI(Library:3,UpperAtrium:19,BottomAtrium:9.2,Rm108-110:8.1)
Library(UpperAtrium:19,StairsI:3)
UpperAtrium(Staff:6.5,Rm116:6.5,Rm117-122:10.3)
Rm117-122(UpperAtrium:10.3)
StairsF(Rm108-110:9,StaffRoom:9,StairsH:9)
StairsH(Rm9:9,Rm10-11:12,BottomAtrium:12,StairsF:9)
Rm9(StairsH:9,Rm10-11:12,BottomAtrium:7.9)
Rm10-11(StairsH:12,BottomAtrium:9,Rm9:10)
BottomAtrium(I:11,Reception:5.5,CourtyardExit:3,StairsH:12,Rm10-11:9,Rm9:7.9,Offices:5.6)
Reception(BottomAtrium:5.5)
Offices(BottomAtrium:5.6)
I(BottomAtrium:9,Rm13-14:13,Landing:3.8,Rm9:5.4,Foyer:7.9)
Landing(I:3.8,Foyer:7.4)
Foyer(Canteen:6.2,Landing:7.4,I:7.9,Hall:5.2,Pupil:12,Rm1-3:9,StairsG:8)
Canteen(StairsG:7,Hall:7,Pupil:9)

```



**The starting Node/Room** – This is the name of the node at which the edge begins at, also known as the start vertex. (StairsA)

**The target Node/Room** – This is the name of the node at which the edge ends at, also known as the end vertex. (Rm 301-305)

**The distance from starting Node/Room to target Node/Room** – This provides the distance in metres from starting node to target node. (5)

This will all be filtered to better suit the needs of the program. As my program takes in each individual value separately, the program will have to split each line of the text file and then split each node/distance within each of those lines.

## User/Client

Having identified the issue of the lack of knowledge for year 7s and other students, I had decided to interview a fellow student, Hathisan Kanthakumar, whom understands the difficulties of finding a route around the school. I then conducted an interview with him where we discussed what he would expect in the program.

### **As a school student how is it like travelling from one class to another after lessons?**

I believe that as the years have gone by the traffic around the school has increased. From students shouting and screaming every time they go down the stairs to students barging other fellow students. I have dealt with all this and I think it is more of a hassle to get passed the huge crowds of students especially at peak hours such as beginning of break or lunch or beginning of period 1. As a lot of students have double periods and therefore the number of people travelling to their next class is less than other times.

### **What are the main problems with the current system?**

Moving around the school when I was in year 7 was difficult as it took a while getting to know the short cuts around the school and sometimes they aren't even the shortest possible routes. For example, it is confusing to know which side of the stairs I should take to get around the school and the current system of asking around is an anxious system. Also the whole idea of stick to the left has been disrupted and no longer heavily focused on by teachers of this school.

### **What are the main features that you are looking for in a new system?**

I believe it is essential to include times for when some routes are accessible. For example during lunch times certain doors are closed and therefore are inaccessible and require a different route. Maybe be able to have this printed on the timetables of students to help them out, especially for younger students such as year 7s or 8s.

### **How do you suppose the user uses the new system?**

I would expect the system to be used on a phone of somewhat, however with the school rules declining the use of mobile phones I see that it would much more useful for a paper version. The routes could also be submitted to the tutors of each student and provided to the students if they wish or can be printed onto the back of the timetable sheet.

## **Programmed in Python by myself**

Python is a relatively simple programming language that I have been studying over the past few years. I will be needing excel to access data which python is able to read from using simple functions.



## My Proposed Solution

My Clients will be able to use my program as follows:

1. Enter their current location within the school (e.g Room 205) **or** Enter their tutor group (only for year 7s) e.g ABA
  - Will be a simple input box or straight into an exe file.
  - Will list the names of tutor groups and ask user to pick which one.
  - Or will provide the option to select a room to begin at and another room to finish at. This will then provide a list of rooms to pass in order to get to the intended room specified by the user.
2. Output the nodes that must be passed to get to the location
3. Provide traffic information depending on the time the students are walking around.  
(Most likely colour co-ordinated)

The program will lack user input due to the fact that it is focal on the Dijkstra's algorithm and therefore used more as a program that simply prints onto a word document of the routes that are needed to be taken.

| <b>Timetable - BAIG, Zayyan 07YC</b><br>as at 03/04/2019 |                              |                              |                               |                               |                               |
|--|------------------------------|------------------------------|-------------------------------|-------------------------------|-------------------------------|
|  | Mon                          | Tue                          | Wed                           | Thu                           | Fri                           |
| 1  | Tech.<br>EP<br>9. Tech       | Maths.<br>YC<br>37. Ma       | Games<br>NM<br>Gym 1          | Citizenship<br>LTT<br>40. Eng | 21st Skills<br>PXM<br>202. Re |
| 2  | Tech.<br>EP<br>9. Tech       | English<br>JYT<br>207. Eng   | Games<br>NM<br>Gym 1          | Mod.Lang.<br>CP<br>210. Mfl   | Art<br>RMP<br>17. Art         |
| 3  | Drama<br>JK<br>12. Drama     | Computing<br>JLS<br>110. Ict | Citizenship<br>LTT<br>40. Eng | Hist.<br>SW<br>314. Hum       | Maths.<br>YC<br>37. Ma        |
| 4  | Maths.<br>YC<br>37. Ma       | Computing<br>JLS<br>110. Ict | Geog.<br>PRH<br>313. Hum      | Maths.<br>YC<br>37. Ma        | Music<br>EA<br>14. Mus        |
| 5  | English<br>JSJ<br>212. Econ  | Science<br>KL<br>28. Sci     | Mod.Lang.<br>ACC<br>208. Mfl  | Geog.<br>PRH<br>313. Hum      | Hist.<br>SW<br>314. Hum       |
| 6  | Mod.Lang.<br>ACC<br>208. Mfl | Science<br>KL<br>28. Sci     | Science<br>KL<br>28. Sci      | English<br>JSJ<br>207. Eng    | English<br>JSJ<br>212. Econ   |
| 7  |                              |                              |                               |                               |                               |

|                           |                        |
|---------------------------|------------------------|
| ACC : Miss A CRUCIANI     | KL : Ms K LAKSHMAN     |
| CP : Miss C PEROY         | LTT : Ms L THOMAS      |
| EA : Miss E ATTINA        | NM : Mr N MCCARTHY     |
| EP : Mrs E POTTER         | PRH : Miss P HEAD      |
| JK : Mr J KILNER          | PXM : Miss P MIAH      |
| JLS : Mrs J SAJADAH       | RMP : Miss R-M PARSONS |
| JSJ : Mr J SHERRIFF-JOLLY | SW : Mr S WATKIN       |
| JYT : J TOPPIN            | YC : Mr Y CHABIBI      |

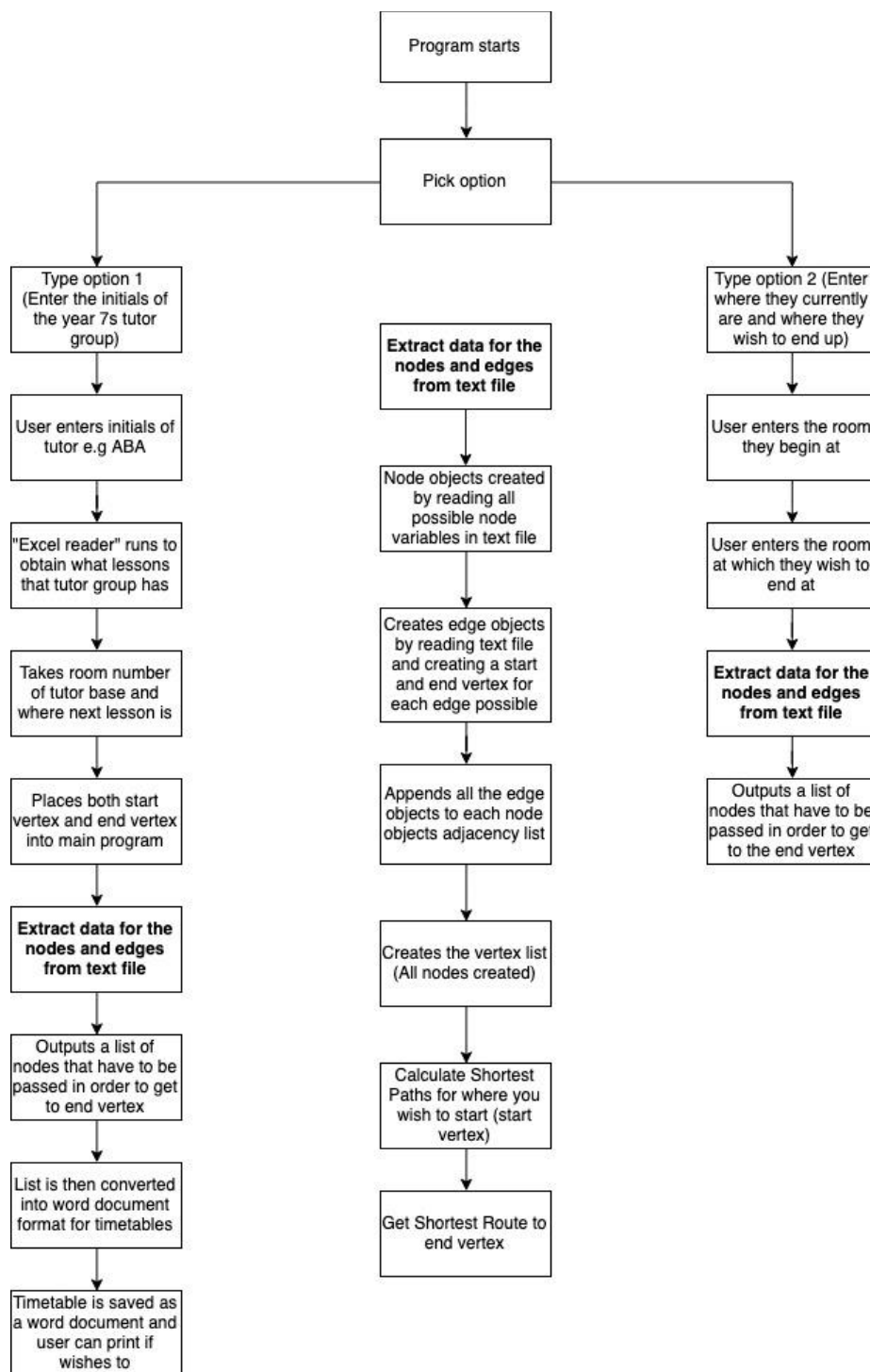
The above image is of how the current timetable of our schools are and have proved to be useful for students to know what lesson they have and which room. However, they are not advised on which routes to take and this is the solution I have come up with:

|   | Mon.   | Tue | Wed | Thu | Fri |
|---|--|-----|-----|-----|-----|
| 1 | Take left stairs, go past canteen.                             | -   | -   | -   | -   |
| 2 | -  | -   | -   | -   | -   |
| 3 | From Canteen go through left hall down to rm 12                | -   | -   | -   | -   |
| 4 | Take a right at corridor go straight until met with huts       | -   | -   | -   | -   |
| 5 | Take right stairs until floor 2, go left to middle of corridor | -   | -   | -   | -   |
| 6 | Opposite room  | -   | -   | -   | -   |
| 7 | -  | -   | -   | -   | -   |

The table above is only an illustrated example of what the back of the timetables will look like if my project is able to be printed onto the back of the timetables. It will also align with the actual timetable as most students tend to cut their timetable up and have a smaller version of the timetable. Having it equal in length and width will result in a much cleaner look for the timetable. I have only filled out the Monday periods 1 to 7 slots of what it would be like for the average student. “-” represents either that there is no lesson next or the lesson is within the same area or same room (If they have a double period).

### Acceptable limitations:

- Too many students to make the most efficient routes possible with little routes accessible.
- User must have knowledge around the behaviour of school students during times of heavy traffic.
- The idea of having weights on each edge based on the number of pupils at the time may be inaccurate due to lack of efficient and reliable data.
- The idea of a text base guide may prove to be insufficient and not as useful as a visual representation of the routes around school. However due to time constraints I will not be creating a map of the school with the routes.

Initial Model:

This diagram shows what the program does at each stage and the results of the user's inputs. The pick options is the initial simple exe file that opens once program opens. It then asks the user for input of either 1 or 2. The "Extract data for the nodes and edges from text file" is the main part of the code, this will be used to access all my classes and create new objects by reading from a text file provided in the same folder. After creating the nodes the Dijkstra's class will be used to perform the entire Dijkstra's algorithm as well as hold a list of the route. The route will then be printed accordingly.

### System Objectives:

Upon discussion with my client I have decided a user interface will be very useful.

My program will provide the following options:

- User can pick option to get the route of a year 7's route for the entire week.
- If user wishes to they may have this be saved onto a word document and the user can then print from the file.
- User can enter 2 values and the route will be outputted on the interface.

The program should give the user the ability to enter whatever room number they wish and have the correct output printed. The text should be of readable size, as well as simple instructions for point to point. In order to create the user interface I am using PyQt5 GUI toolkit.

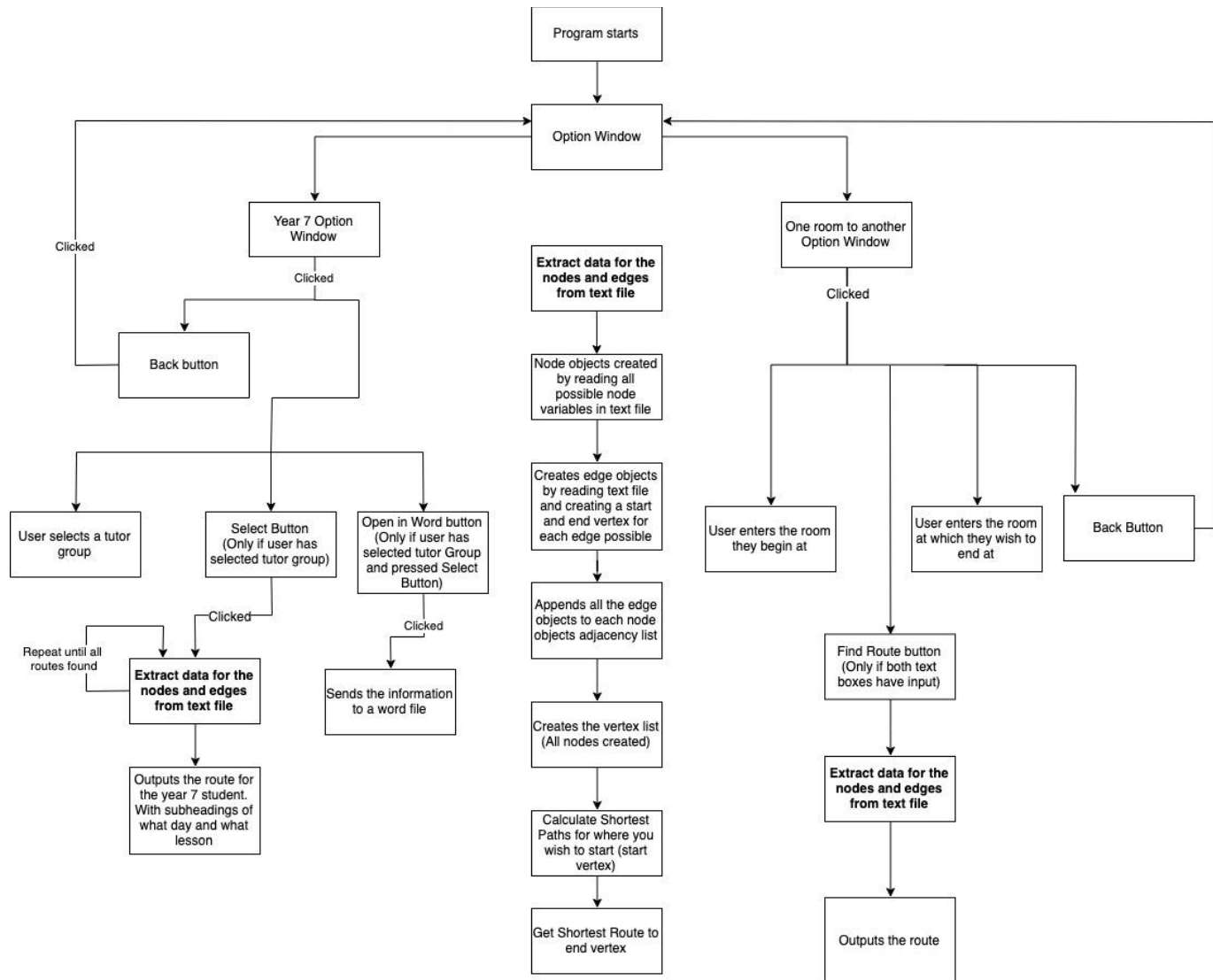
## **Documented Design**

### **Introduction**

In this section, I will be discussing the thoughts that went behind the creation of my program. As planned in the analysis stage, I have used Python as my programming language. I will be presenting various diagrams to show the creation of my program. I have also used Microsoft Excel in order to hold the data of year 7s lessons. I will also be explaining the Dijkstra's algorithm that I have used as the algorithm is very sophisticated for the system, and overall is the main support for my project. All of this will create an understanding of my program and how the different classes within my program co-exist and provide the smoothest and efficient working product.

## Updated System Walkthrough

I have made small changes to my original system as the idea of a proper UI came to mind. These changes are shown below:



The changes made have adapted to the use of a User Interface. Initially I did not have a User Interface as I was unsure which route to take. This meant that the route would be outputted on the python shell. This was not efficient or useful for the user, therefore the use of an interface was needed.

The main "Extract data for the nodes and edges from a text file" has not changed as it is the basis of the dijkstras algorithm. The back button is used to return the user to the Options window .

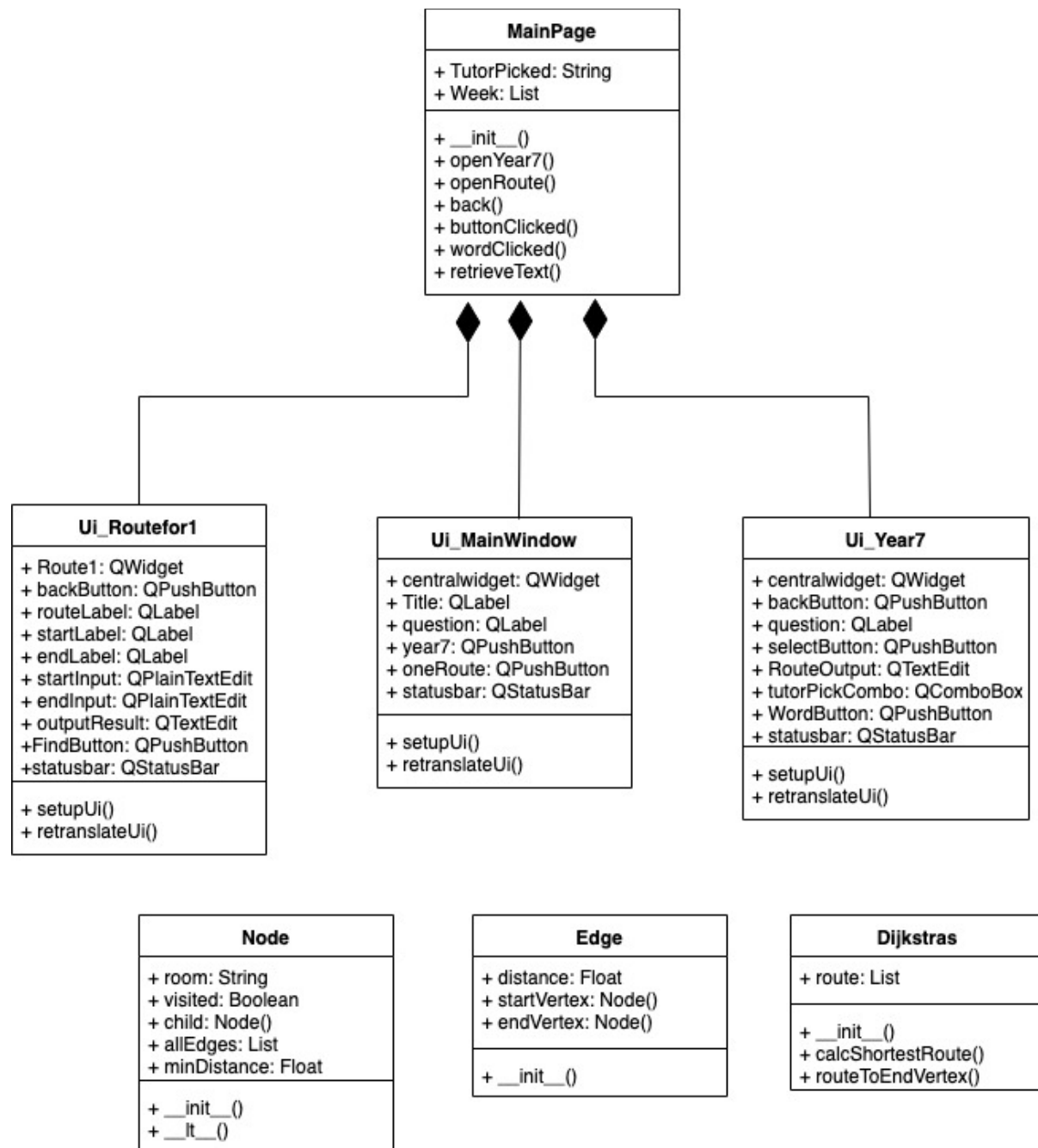
On the left side you can see the Year 7 options window. This contains one output box as well as a combo box used to display the tutor group options. This is more efficient to my previous idea as I do not need to make sure the user enters an incorrect value. There is also the select button which can be pressed whenever and goes through the dijkstras algorithm several times to get all routes. This ranges from all days of the week as well as all periods within the day. There is an also a button to



write the route onto a word document. This is because all year 7s are provided with a timetable at the beginning of the year and therefore would be easy to print the routes on the back of the sheet. This is put into a table format.

On the right side you can see the One room to another options screen. This screen is simple as it only has 1 button ,2 input text boxes and 1 output text box.

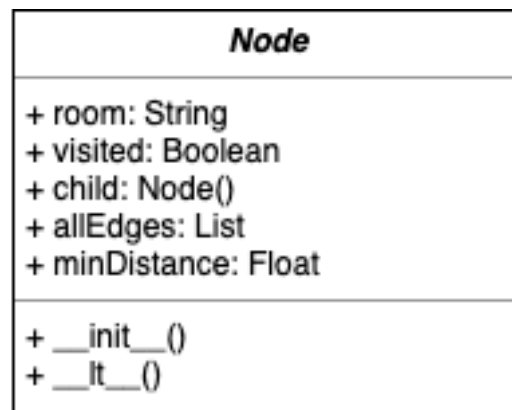
## Class Diagram



These classes represent all of the classes that I have used and the inheritance of the "MainPage" class I have used.

I will be going into further detail in the next few pages.

### Node class



This diagram shows the class that creates every Node. It holds the name of the node (room number or area).

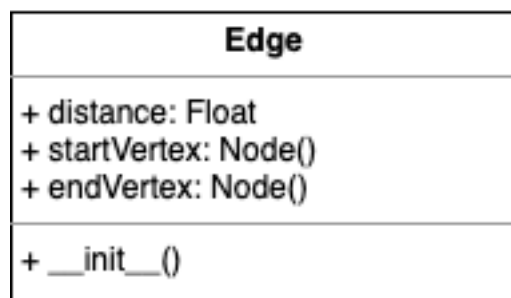
Holds whether the Node has been visited or not (True or False).

“child” is the vertex that proceeds the given vertex. Important for when we backtrack.

The “allEdges” holds all the edges that are created from the Edge class also known as an adjacency list.

minDistance holds the lowest value for when Dijkstra’s is run (constantly updated unless the next distance is greater than previous distances together).

### Edge class



This diagram shows the class used for the creation of each edge. The class is simple as it just holds both the starting node and ending nodes together with the distance between the two nodes.

The distance holds the distance in meters as a float.

The startVertex holds the Node at which the edge begins at.

The endVertex holds the Node at which the edge finishes at.

**Dijkstra's class**

| <b>Dijkstras</b>  |
|---|
| + route: List   |
| + __init__()<br>+ calcShortestRoute()<br>+ routeToEndVertex() |

This diagram shows the class used for the main algorithm, Dijkstra's. The method "calculateShortestPath" calculates all the nodes that are connected to the startVertex and all other nodes. The method "routeToEndVerex" backtracks from the specified vertex until the startVertex is met. The "route" list holds the values once backtracked so my program can output it.

**MainPage Class**

| <b>MainPage</b>  |
|--|
| + TutorPicked: String<br>+ Week: List  |
| + __init__()<br>+ openYear7()<br>+ openRoute()<br>+ back()<br>+ buttonClicked()<br>+ wordClicked()<br>+ retrieveText() |

This diagram shows the class used to bring all the windows together. Each method is connected to a button within any of the window. The "\_\_init\_\_" begins the program by providing the user with the main menu interface. The main menu interface contains 2 buttons, one for the year 7 option window and another for the one route window.

The "openYear7" method is used to pop up the interface for the year 7 option within the main menu. The method creates the list of strings within the combo box (tutor group initials), it also connects the select button to the "buttonClicked" method. It also connects the back button to the "back" function.

The "OpenRoute" method is used to pop up the interface for the route option within the main menu. This also contains a back button and is connected to the "back" function. There is also the use of the "FindButton" and is connected to the "retrieveText" function.

The "back" method is used to return the user to the main menu and allocates the same buttons mentioned previously in the "\_\_init\_\_" function.

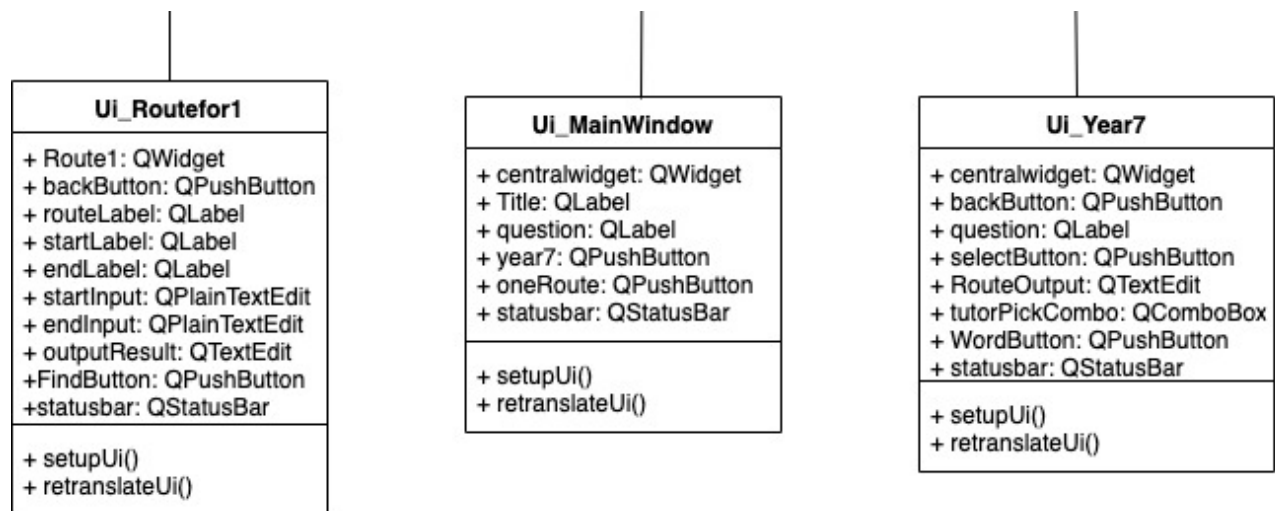
The "buttonClicked" function is used when the select button in the year 7 window is clicked. This takes the the value the user has selected in the "tuturPickCombo" (combo box) and

stores it as “TutorPicked”. The entire function outputs the full route into the “RouteOutput” text box. Then if the “WordButton” button is pressed it moves to the “wordClicked” method.

The “wordClicked” method is used when the user wishes to save the routes into a word document.

The “retrieveText” method is used to get the values stored in both “startInput” and “endInput” text boxes and then sends to Dijkstras to find the route and returns the value into the “outputResult” text box within the route window.

### Ui\_Routefor1, Ui\_MainWindow, Ui\_Year7 Classes:



I have shown these 3 classes together as they were created using the software “PyQt Designer”. Each class represents a window. The method “setupUi” was created by PyQt Designer to build the widget tree on the parent widget. For example, the “backButton” attribute is a push button. The method “retranslateUi” is used to handle the translation of the string properties of the form. These classes are all saved as ui files but I have converted them into python code to have a better understanding of what goes on.

## Class Tables and Method Descriptions

| CLASS Node  |   |            |               |   |
|-------------|---|------------|---------------|---|
| Attributes  |   | Type       | Typical Value | Description   |
| room        | + | String     | "Rm301-305"   | Holds the name of the Node object when instance passed.               |
| visited     | + | Boolean    | False         | Holds whether the Node has been visited or not.                       |
| child       | + | Node()     |               | Holds the previous Node for backtracking.                             |
| allEdges    | + | List       | []            | Holds the list of Edges that are appended to the Node.                |
| minDistance | + | Float      | 1000000000    | Initialised as a large number to simulate infinity/ the maximum size. |
| Methods     |   | Parameters | Return        | Description   |
| __lt__      |   | Node()     | Node()        | Less than comparison method   |

### Class Node

#### Initialise:

Declare room as roomname  
 Declare visited as False  
 Declare there is currently no child, None  
 Declare the empty list (allEdges)  
 Declare minDistance as a large number

#### LessThan:

Equal Priority Node to minDistance  
 Equal Other Priority to the new provided Nodes' minDistance  
 Compare Priority and Other Priority and return if Priority is less than Other

The pseudo-code above shows the what happens in the constructor method when a Node is initiated as well as the less than rich comparison method. The local variable "room" is the name of the rooms in this case, also known as the Node or the Vertexes. The local variable "visited" informs you whether the Node has already been visited for when Dijkstra's runs, stopping the Node from being read again as well as backtracking. The local variable "child" is



used to show the predecessor of the main Node, this allows Dijkstra's to back track and make the child Node the main Node and carry on back tracking until the original Node is met. The local list "allEdges" stores all possible Edge instances that are linked to the Node. For example when "edge1" is created with the StartVertex of Node1 and EndVertex of Node2, this will be appended to both the Node1's "allEdges" instance as well as Node2's (will be further explained when I explain the Edge class). The local variable "minDistance" is used to represent a large number also known as infinity, this is because in the Dijkstra's algorithm, when comparing nodes, the minimum distance will be calculated and compared to the minDistance to make sure the value is less than and possible, for example if there was no edge between 2 nodes and there is no other possible routes then the minDistance will stay 1000000000 as it represents infinity(not found yet or unfindable).

The "\_\_lt\_\_" method is a comparison method used for sorting. This is used for when you wish to compare custom objects. This is useful for my code as I need to compare Node objects to come out with my results. There were other methods of comparing values however they consisted of longer much less efficient coding. The rich comparison overrides the operator "<".

| CLASS Edge  |   |        |               |   |
|-------------|---|--------|---------------|---|
| Attributes  |   | Type   | Typical Value | Description   |
| distance    | + | Float  | 5             | Holds the distance between the StartVertex and EndVertex. |
| startVertex | + | Node() |               | Holds the node object for which the Vertex begins.        |
| endVertex   | + | Node() |               | Holds the Node object for which the Vertex ends.          |

**Class Edge:**

**Initialise:**

**Declare distance between nodes as distance**  
**Declare the beginning of an edge as a Node object**  
**Declare the end of an edge as a Node object**

This class is a very simple class that uses another object as an initialiser. The Edge class is used to create Edges that link vertexes together. The local variable "distance" stores the

weight of the Edge, for example from “Rm301-305” to “StairsA” the weight is 5 as the distance between the two Nodes is 5 metres. The “startVertex” attribute holds the Node object of where the edge starts. The “endVertex” attribute holds the Node object at which the edge ends.

| CLASS Dijkstra's  |   |                      |                            |  |
|-------------------|---|----------------------|----------------------------|--|
| Attributes        |   | Type                 | Typical Value              | Description  |
| route             | + | List                 | ["Rm301-305, StairsB,..."] | Holds the list of all the nodes that have been backtracked, also known as the route. |
| Methods           |   | Parameters           | Return                     | Description  |
| calcShortestRoute | + | graph<br>startVertex |                            | Calculates distance between startVertex and all other Nodes possible.                |
| routeToEndVertex  | + | endVertex            | string                     | Backtracks from endVertex until startVertex is met.                                  |

#### Class Dijkstra's:

##### Initialise:

Declare an empty list to append nodes when backtracking

##### calcShortestRoute:

create an empty list as queue

allocate the starting point

push first item into queue

While there are elements in queue THEN

Pop and return smallest value from queue

For the number of edges in allEdges list THEN

Get the initial vertex (startVertex) of edge

Get the target vertex (endVertex) of edge

Add the minDistance of initial vertex with the weight of edge

If the new distance is less than target vertexes minDistance THEN

Target vertex's child becomes initial vertex  
 Update minDistance  
 Push target vertex into queue

Otherwise pass

This class is the main algorithm for my project, Dijkstra's. The "calcShortestRoute" method simply updates the queue accordingly and continuously updates for the shortest path. This algorithm will be further explained within the algorithm section.

**shortestPathTo:**

prints the minDistance of target vertex which is value of the shortest path  
 equal node with target vertex

while there are child instances within the node THEN  
     append to the node to the route list  
     backtracks  
 reverse the route list  
 return route

This class is the 2<sup>nd</sup> part to Dijktras as it backtracks from the target vertex to the start vertex. Will be explained further in algorithm section.

| CLASS MainPage |   |            |                               |   |
|----------------|---|------------|-------------------------------|---|
| Attributes     |   | Type       | Typical Value                 | Description   |
| TutorPicked    | + | String     | "ABA"                         | Stores the value the user picked in the tutorComboBox after pressing the select button. |
| Week           | + | List       | [[ "Rm 301-305, ... ], [...]] | List within a list that contains all the routes for each day.                           |
| Methods        |   | Parameters | Return                        | Description   |
| __init__       | + | -          | -                             | Reads from the 'mainMenu.ui'. Returning the mainMenu window                             |

|               |   |   |   |  |
|---------------|---|---|---|--|
| openYear7     | + | - | - | Reads from the 'year7.ui' file. Returning the year 7 option window.  |
| openRoute     | + | - | - | Reads from the 'route.io' file. Returning the route option window.   |
| back          | + | - | - | Returns the user to the mainMenu window again.   |
| buttonClicked | + | - | - | Within the year 7 option window, works as the method for when the select button is clicked. Outputs the route into the "RouteOutput" text box.   |
| wordClicked   | + | - | - | Sends the routes and the the tutor name into the 'writeToWord' function.   |
| retrieveText  | + | - | - | Within the route option window, sends the users inputs (startInput and endInput text boxes) to the "retrieveRoute" function that returns the route. Then outputs into the "outputResult" text box. |

Class MainPage(QMainWindow):

Initialise:

```

super(MainPage, self).__init__()
load MainMenu ui file
when year7 button clicked go to openYear7 function
when oneRoute button clicked go to openRoute function

```

openYear7:

```

load year7 ui file
when backButton clicked go to back function
append "ABA", "YC" ... to tutorPickCombo combo box
when selectButton clicked go to buttonClicked function

```

openRoute:

```

load route ui file
when backButton clicked go to back function
when oneRoute clicked go to openRoute function

```

**back:**

load MainMenu file  
when year7 button clicked go to openYear7 function  
when oneRoute button clicked go to openRoute function

**buttonClicked:**

store value in tutorPickCombo as TutorPicked  
send TutorPicked to checkTutor function and return startVal and row  
send row and startVal to reader function and return Week  
Route is empty string  
listOfDays = ["Mon", "Tue", "Wed", "Thu", "Fri"]  
listOfPeriods = ["To Period 1: ", "To Period 2: ", ..... "To Period 6"]  
index = 0  
otherIndex = 0  
for every day in week THEN  
    Route = Route + listOfDays[index] + (next line)  
    for every period in day THEN  
        Route = Route + (next line) + listOfPeriods[otherIndex] + (next line)  
        for every route in period THEN  
            Route = Route + route + (next line)  
        add 1 to otherIndex  
    add 1 to index  
    reset otherIndex to 0  
    add a new line to Route  
remove last 2 characters from Route  
set text in RouteOutput as Route  
when WordButton is clicked go to wordClicked function

**wordClicked:**

send Week to writeToWord function

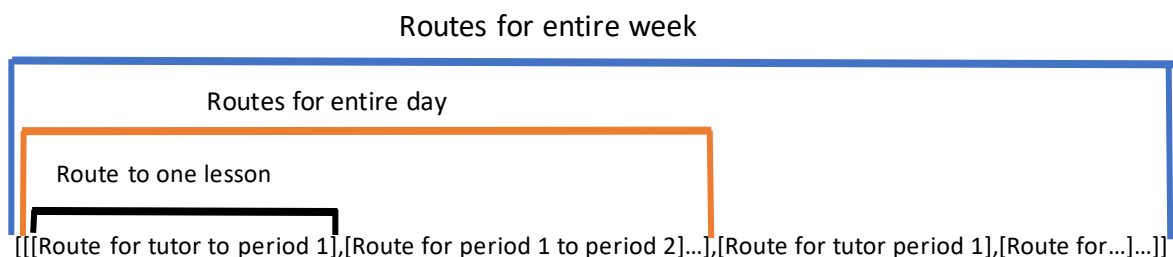
**retrieveText:**

store value in startInput as start  
store value in endInput as end  
send start and end to retrieveRoute function and return route  
Route is empty string  
for every node in route  
    Route = Route + node + (next line)  
remove last character from Route  
set text in outputResult as Route

This part of the pseudo code is used to provide a usage for the widgets in the windows. Firstly, in the initialiser method you inherit the initialiser function from the MainPage class which is the super class. I then used the PyQt5 uic module and imported "loadUi" in order to load the ui file that I created on PyQt Designer. The user is then displayed with the mainMenu window, within the mainMenu window there are 2 buttons. One is the button to open the year7 window, which contains the first option of reading a year 7 students timetable, and the other option to open the oneRoute window, which contains the simple 2 input texts for the user to get the shortest route of desired locations. Depending on the button clicked a different method is called.

If the user has selected the year7 button, then the user is introduced to the year7 interface. This window contains a back button (which when pressed returns you to the mainMenu window) as well as a combo box containing a list of all the possible tutor groups the user can enter and 2 buttons, a button to present the route onto the text box and another button which results in the creation of a word document containing the route for the year 7. "openYear7", "openRoute", "back" are all methods that send the user to a specific window. The "buttonClicked" method first finds out what value the tutor initials the user has selected by reading the current text within the "tutorPickCombo" (combo box) and saving it as TutorPicked. As the variable will need to be used elsewhere other than just this method I have used self. The value is then sent to the function "checkTutor", which is explained further in the documented design. This function returns the start value as well as the row number. The startValue is the Node at which the tutor base is located in and the row value indicates the row within the excel file that represents the Tutor group. These 2 values are then sent to the reader function which returns a list containing a list. This list contains every route for each day within the school week.

In order to be able to use this information we must make it readable and take it out the lists. In order to do this, I created the list containing the list of days within a week and a list containing what the routes are for ("To get to period 1" or "To get to period 2" etc). These lists are used for when we split the routes we received into presentable and understandable data. I created 2 variables as indexes for each list, index and otherIndex. I then iterated through every day in the day in the self.Week list, to get a better understanding imagine that the first value in self.Week is:



As you can see it is a list within a list within a list. I decided to create a long string containing the output for the RouteOutput text box. I had to split the lists in order to get them appropriate for their day. Then I appended the name of the day to the string followed by a "\n" which creates a new line in python. After this I read all the routes in an entire day and appended the name of the period followed by a "\n". I then had to actually output the routes for each lesson by appending for every value followed by a "\n". This was to ensure that the layout was easy to read. Once all the values for one route was appended, the next lesson is read and once again the routes are appended. This is repeated until all the routes within the entire day is appended. Once it has finished, then it goes to the next day of the list and repeats every step. This repeats until every value is added to the string. This will be easily understood when the output is shown in the user interface.

The user now has the choice whether they wish to have this saved onto a word document or stop using the application or change the tutor group. Assuming that the user presses the



“WordButton”, the wordClicked method is called. This method sends the self.Word variable to the writeToWord function. This function is explained later in the documented design.

If at the the “mainMenu” interface the user selected the oneRoute option, then the “retrieveText” method is called. This method obtains the start value for Dijkstras from the users input in “startInput” aswell as the target node for Dijkstras from the users input in “endVertex”. Both these values are send to the retrieveRoute function which returns the shortest path. Then the shortest path needs to be converted into a string in order to set the text of “outputResult”.

**Other Functions**

| FUNCTIONS       |            |            |  |
|-----------------|------------|------------|--|
| Functions       | Parameters | Return     | Description  |
| nodefinder()    | -          | nodes      | Creates a list of every line within the "Rooms.txt" text file.   |
| createGraph()   | graph      |            | Splits every line of the text file and creates Node and Edge objects then appends to the graph.  |
| retrieveRoute() | start, end |            | This is the calculation part of the code that compares the start and target values with the graph then performs Dijkstras. Then returns the shortest route |
| reader()        | row, start |            | This reads through the excel file and performs dijkstras on all lessons within the week for year 7 students.   |
| checkTutor()    | tutee      | start, row | Depending on which tutor group the user entered the start vertex will be set.  |
| roomNum()       | val        | start, row | Used to check the users input  |
| writeToWord()   | week       |            | Used to write into a word document. Writes a table containing the routes.  |

**nodeFinder()**

```

DEFINE nodefinder()
    global nodes
    nodes is empty list
    open "Rooms.txt" THEN
        for every line in text file THEN
            add line to nodes

```

This simple pseudo code reads through every line in the text file and appends to the list, "nodes". For example the list will look like this: [StairsA(Rm 301-305:5,Rm 306-315:14, StairsC:9) , Rm 301-305(StairsA:5) ,.....]. The nodes list will be used later in the program to through and create Nodes and Edges.

**createGraph()**

```

DEFINE createGraph()
  nodeId = 1
  edgeId = 1
  count = 0
  nodeVal = 0
  edgeVal = 0
  position = 0
  graph = []
  edgelist = []
  nodefinder()
  x = True
  while x is True:
    count starts at 0
    create variable tempnode as string
    for character in tempnode THEN
      if the character is "(" THEN
        man = 0
        mainNode = tempnode[first character to count]
        true = True
        if graph is empty THEN
          append Node + nodeId as a joint string
          make (Node + nodeId) a Node object
          add 1 to nodeId
          add 1 to count
          c = count
          add 1 to nodeVal
        If not THEN
          for every node in graph THEN
            if mainNode is in the list THEN
              true = False
            if not THEN
              if true is True THEN
                add 1 to man
              if not THEN
                pass
          If true is True THEN
            append Node + nodeId as a joint string
            make graph[nodeVal] a Node object
            add 1 to nodeId
            add 1 to count
            c = count
            add 1 to nodeVal
          If not THEN
            add 1 to count
            c = count

    if the character is ":" THEN
      true = True
      chil = 0

```

```
        child = tempnode[c to count-1]
        for every node in graph THEN
            true = True
            chil = 0
            child = tempnode[c to count-1]
            for every node in graph THEN
                if child is in the list THEN
                    true = False
                if not THEN
                    if true is True THEN
                        add 1 to chil
                    if not THEN
                        pass
            if true is True THEN
                append Node + nodeId as a joint string
                make graph[nodeVal] a Node object
                add 1 to nodeId
                add 1 to nodeVal
                c = count
            if not THEN
                c = count

    if the character is “,” THEN
        distance = tempnode[c to count-1]
        append Edge + edgeId
        make edgelist[edgeVal] an Edge object
        append edge object into node object’s “allEdges” list
        add 1 to edgeVal
        add 1 to edgeVal
        append Edge + edgeId
        make edgelist[edgeVal] an Edge object
        append edge object into node object’s “allEdges” list
        add 1 to edgeId
        add 1 to edgeVal
        c = count

    if the character is “)” THEN
        distance = tempnode[c to count-1]
        append Edge + edgeId
        make edgelist[edgeVal] an Edge object
        append edge object into node object’s “allEdges” list
        add 1 to edgeId
        make edgelist[edgeVal] an Edge object
        append edge object into node object’s “allEdges” list
        add 1 to edgeId
        add 1 to edgeVal
    if not THEN
        add 1 to count
    x = False

run(graph)
```

This part of the code simply iterates through every character in the text file and splits them into their correct positions, such as distances and nodes. As there are several nodes within the text file I had to implement within the code to check whether the node already exists, and if it did, I would need to replace the indexValue of the graph to that and use that as an edge.

In the above pseudo code you can see that the variable “tempnode” is created to hold the line at which the index value position is. Then for every character within that line, it iterates until either the character is an open bracket “(” or a colon “:” or a comma “,” or a close bracket “)”.

If the character is an open bracket the code first reads all the characters before the open bracket as the “mainNode” (this can be seen as the startVertex for an edge). The list graph is checked whether it is empty or not. This is in order to prevent a crash from happening for when the list is checked for a repeating value. If the list is empty then the string "Node1" will be appended to the list (the value “1” in the string will increase as the number of nodes being created increases). As you can see the nodeId is used as a mean of creating new string variable that will then be used as the objects’ name. However, if the list is not empty, so after the first node object is created, it then checks whether the mainNode is already in the node list. If the node is in the graph then it saves the index value of that as man, and then is used for when the node is later appended to a list. If mainNode is not already in the list then a new Node object is created.

If the character is a colon, then the process is similar to the open bracket one, however we do not need to check if the list is empty as the list has already got a value, the mainNode object. The child is compared to the values of which are already in the list and if present then the index is stored as “chil”, if not then a new Node object is created. In this if statement however we call each node a child instead of a mainNode as these are the neighbours/child of the node (what the mainNode is connected to). The mainNode does not change until the position increases and the next line is being read.

If the character is a comma then the program knows that whatever comes before the value is a number (the distance between the mainNode and the child Node). The distance is stored as distance. The same way that the Node objects are created, the Edge objects are created too. Everytime a new Edge object is created the instances are filled with the distance and the mainNode aswell as the child Node. The child Node will always differ when the comma is met, this is because after the comma the next child node will be present and will replace the “child” variable. While also creating the Edge object the if statement also appends the Edge object to the mainNode aswell as the child Node. This is repeated twice as previously mentioned the graph is a directed graph.

If the character is a closed bracket, then the process is identical to the comma if statement but adds a 1 to the position value (so the next line can be read).

Here is a small example of what the format of the nodes and edges are:

This is Node objects name

**Node1 = Node("Rm301-305")**

**Node2 = Node("StairsA")**

**Node3 = Node("Rm306-311")**

This is the name of the room or area

This is Edge objects name

**Edge1 = Edge(1, Node1, Node2)**

**Edge2 = Edge(3, Node2, Node3)**

**Edge3 = Edge(2, Node1, Node3)**

startVertex and endVertex for each edge

The distance between the nodes

**Node1.allEdges.append(edge1)**

**Node1.allEdges.append(edge2)**

**Node2.allEdges.append(edge3)**

Appending each edge to each node

This is the format of how I was going to create and append each node and edge. In order to do this, I created the function above. As you can see with the above example, it only creates a directed graph. In order to solve this issue I created double the amount of edges in order to have 2 of the same edge but different start and end vertexes (opposite). This part of the code simply creates all Node objects, Edge objects and appends each Edge to the correct node. In order for Dijkstras to work these objects need to be created and must be accurate so the wrong edges are not appended to the wrong nodes.

List of all variables and use:

| Variables     |         |  |
|---------------|---------|--|
| Variable name | Type    | Description  |
| nodeId        | integer | Holds the index value of graph   |
| edgeId        | integer | Holds the index value of edgelist  |
| count         | integer | Holds the index value for characters in tempnode   |
| nodeVal       | integer | Holds the index value of a node that already exists in graph and needs to be used to create an Edge object |
| position      | integer | Holds the index value of lines in the text file  |
| graph         | list    | Holds all the Node objects and is used when an existing node is trying to be recreated                     |
| edgelist      | list    | Holds all Edge objects   |

|          |         |  |
|----------|---------|--|
| x        | boolean | Used for while loop to make sure all nodes and edges have been created   |
| tempnode | string  | Temporarily holds the line of which index position is, for example "Rm301-305(StairsA:5)"  |
| man      | integer | Used as a method to find what the index value of the already existing node is  |
| mainNode | string  | Is the first node to be read from each line  |
| true     | boolean | Used to stop the value of man from increasing while iterating through graph to find existing node                                  |
| c        | integer | Same value as count but stops once one of the special characters are met and stores the index value (count)                        |
| chil     | integer | Similar to "man" as it is used to find the index value of an already existing node. Difference is that this is for the child nodes |
| child    | string  | Is the child nodes read from each line   |
| distance | float   | Stores the value of the distance that is then placed into an Edge object as the distance between the two nodes                     |

### checkTutor()

DEFINE checkTutor()

```

    start is an empty string
    if the tutor they picked is "ABA" THEN
        start equals to "Rm38-39"
        row = 12
    otherwise if the tutor they picked is "YC" THEN
        start equals to "Rm36-37"
        row = 18
    otherwise if the tutor they picked is "JRS" THEN
        start equals to "SciCorr1"
        row = 6
    otherwise if the tutor they picked is "AN" THEN
        start equals to "Rm301-305"
        row = 30
    otherwise if the tutor they picked is "GS" THEN
        start equals to "Rm108-110"
        row = 21
    otherwise if the tutor they picked is "GO" THEN
        start equals to "Rm117-122"
        row = 19
    otherwise if the tutor they picked is "JLS" THEN
        start equals to "Rm108-110"
        row = 24

```

if not THEN  
pass

return start and row

This function is just used to identify the room of the user's tutor group. The data for which rooms was collected from this sheet:

Then using the other sheet I was able to identify which row number within the excel file was linked to the tutor base. The row will then be carried back to the "run()" function which then uses the value to send both the start value and the row number to the "reader()" function. For example if the user enters "ABA", the program will know that his tutor place is located in "Rm38-39" and will recognise that it is in row 13 of the excel file.

| YEAR 7 - YELLOW<br>AR / OO / CI                     |     |
|---|-----|
| Business Studies,<br>Computing, Economics,<br>Maths |     |
| Adam Razaq  |     |
| 7A1 Junaid Shaikh<br>7JRS                           | 35  |
| 7A2 Abdiqani Abdalla<br>7ABA                        | 39  |
| 7A3 Geoffrey Opoku<br>7GO                           | 38  |
| 7B1 Youssef Chabibi<br>7YC                          | 37  |
| 7B2 Jayshree Sajadah<br>7JLS                        | 110 |
| 7B3 Ghassan Suleiman<br>7GS                         | 108 |
| 7C Alexandra Newnham<br>7AN                         | 307 |

| Variables     |         |  |
|---------------|---------|--|
| Variable name | Type    | Description  |
| start         | string  | Holds the name of the node for which the start vertex is |
| row           | integer | Holds the row number for when reading excel file         |
| tutee         | string  | Stores the user's tutor base that they enter themselves  |



**retrieveRoute()**

```

DEFINE retrieveRoute (start, end):
    aRoute = Dijkstras()
    return a new graph from createGraph function
    send start into roomNum function and return as startVal
    send end into roomNum function and return as endVal
    for every node in graph THEN
        if startVal is in list THEN
            aRoute.calcShortestRoute (graph, node)
        else:
            pass
    for every node in graph:
        if endVal is in list THEN
            route = aRoute.routeToEndVertex(node)
    return route

```

This function is used to retrieve the shortest route to a destination. When the user enters 2 values (start and target vertex), the 2 values are split into “start” and “end”. This function first sends the values to the roomNum function to ensure that the user has entered valid inputs. The two values are then checked by comparing every node in the graph to the values. This is done by comparing the start/end value to the name of every node in the graph until they match. If they match then the first part of the dijkstras algorithm is done. After the end value is checked with the graph. If correct then the shortest route will be stored as the variable “route”.

| Variables     |                |   |
|---------------|----------------|---|
| Variable name | Type           | Description   |
| aRoute        | class object   | Used to perform dijkstras   |
| start         | integer/string | Depending on what the user enters stores the room they specified                          |
| end           | integer/string | Depending on what the user enters stores the room they specified                          |
| startVal      | string         | A procedure used to return the name of the Node to be searched, used as the starting node |
| endVal        | string         | A procedure used to return the name of the Node to be searched, used as the target node   |
| route         | list           | The list obtained from dijkstras also known as the shortest route                         |

roomNum()

```
DEFINE roomNum (val)
  If val is an integer THEN
    Make val an integer
    If value in range if val in range(301, 306):
      val = "Rm301-305"
    otherwise if val is in the range(306, 312):
      val = "Rm306-311"
    otherwise if int(val) is in the range(311, 316):
      val = "Rm310-315"
    otherwise if int(val) is in the range(201,204):
      val = "Rm201-203"
    otherwise if int(val) in range(205, 212):
      val = "Rm205,211"
    otherwise if int(val) in range(212, 218):
      val = "Rm212-217"
    otherwise if int(val) in range (101,103):
      val = "Rm101-102"
    otherwise if int(val) in range(108, 111):
      val = "Rm108-110"
    otherwise if "staff" in val:
      val = "Staff"
    otherwise if int(val) == 116:
      val == "Rm116"
    otherwise if int(val) in range(117, 123):
      val == "Rm117-122"
    otherwise if int(val) == 9:
      val == "Rm9"
    otherwise if int(val) == 10 or 11:
      val == "Rm10-11"
    otherwise if int(val) in range(1, 4):
      val == "Rm1-3"
    otherwise if int(val) == 36 or 37:
      val == "Rm36-37"
    otherwise if int(val) == 38 or 39:
      val == "Rm38-39"
    otherwise if val == 44:
      val == "Referral"
    otherwise if int(val) in range(32, 36):
      val == "SciCorr1"
    otherwise if int(val) in range(28, 32):
      val == "SciCorr2"
    otherwise if int(val) in range(24, 28):
      val == "SciCorr3"
    otherwise if int(val) in range(17, 24):
      val == "Sci/ArtCorr"
    otherwise if int(val) == 17:
      val == "Rm17"
    otherwise if int(val) in range(40, 44):
      val == "Rm40-43"
```

```
otherwise if int(val) in range(13,15):
    val == "Rm13-14"
otherwise if int(val) == 12:
    val == "Rm12"
otherwise if int(val) == 106 or 107:
    val == "Ch106-107"
otherwise if val == 103 or 104:
    val == "Rm103-104"
otherwise if val is not an integer THEN
    if val == "reception" or "Reception":
        val == "Reception"
    otherwise if val == "Offices" or "offices":
        val == "Offices"
    otherwise if val == "Canteen" or "canteen":
        val == "Canteen"
    otherwise if val == "Pupil" or "Entrance" or "Pupil entrance":
        val == "Pupil"
    otherwise if val == "Hall" or "hall":
        val == "Hall"
    otherwise if val == "Referral":
        val == "Referral"
    otherwise if val == "Courtyard" or "courtyard":
        val == "CourtyardS"
    otherwise if val == "Medical" or "medical":
        val == "Medical"
    otherwise if val == "Sports" or "Sports hall" or "PE hall":
        val == "Sports Hall"
    otherwise if val == "Gym" or "gym":
        val == "Gym"
    otherwise if val == "Dojo" or "dojo":
        val == "Dojo"
    otherwise if val == "Study" or "study":
        val == "Study"
    otherwise if val == "SFC":
        val == "SFC"
    otherwise if val == "FLD1":
        val == "B"
    otherwise if val == "FLD2":
        val == "B"
    otherwise if val == "GYM1":
        val == "Sports Hall"
    otherwise if val == "GYM2":
        val == "Gym"
if not THEN
    val == "inc"
return val
```

This part of the code simply just checks the value the user has entered, also known as "val" and then converts it into the name of the actual node (the format used within the Nodes

object). This allows the program to read the room the user would like to start at and end at and make sure there are no errors. When val is set to "inc" then the program resets as it is an incorrect value that the user entered, as seen in the "run()" command. The val variable is returned and then placed back into the code in the "run()" function which then depending on the option sets out to compute the solution.

| Variables     |                   |  |
|---------------|-------------------|--|
| Variable name | Type              | Description  |
| val           | string or integer | Depending on the user's input the value will either be a string or an integer. The value is used as a means of equalling to the actual node that can be traceable. |

## Reader()

```

DEFINE reader(row, start)
    create new graph with createGraph function
    index = 0
    listOfRooms = []
    route = ["MONP1", "MONP2", ....]
    open csv file as f THEN
        file = csv.reader(f)
        rows = [r for r in file]

    for lines from the start to the 32ND line THEN
        remove all instances of "\xa0" from line

    for i in range (1,35) THEN
        if the row has any empty cells THEN
            append the start value to listOfRooms
        if not THEN
            append the value within the cell to listOfRooms

    add start to the beginning of listOfRooms

    while length of listOfRooms is not equal to 1 THEN
        route[index] = Dijkstras()
        for every node in graph THEN
            if the first value in listOfRooms is in list THEN
                route[index].calcShortestRoute(graph, node)
            if not THEN
                pass
        for every node in graph THEN
            if the second value in listOfRooms is in list THEN
                route[index].routeToEndVertex(node)
            if not THEN
                pass
    create new graph with createGraph function
  
```

```

    add 1 to index
    pop first value from listOfRooms
    if the length of listOfRooms is equal to 1 THEN
        break
    otherwise THEN
        if first value in listOfRooms is equal to start THEN
            pop first value from listOfRooms
    Mon = route[from 1st value to 6th]
    Tue = route[from 6th to 12th]
    Wed = route[12th to 18th]
    Thu = route[18th to 24th]
    Fri = route[24th to last value]

```

This part of the code is the main function in order for the excel file to be read and translated. The first part of the function before the while loop is used to read the excel file. I have used the csv module in order for this to work. It creates the variable rows by reading every line within the excel file. I then used the i in range 32 (as the first 32 lines are for year 7s) to remove the “\xa0” character that is hidden within the excel file but present when converted to strings within python. After stripping I then read the line, depending on the x values provided (dependant on the day the user picks), and remove all empty cells. This creates a list of the rooms I need to travel from point A to B to C... etc.

The second part of the code is used to find all the routes and store all the results. Most of the parts have already been discussed previously such as the for loops in this function. The difference however is that the “route” list contains strings that will then equal to the Dijkstras class to make objects. The index will increase every time after a dijkstras object is created and outputs the value of the route. I have decided to pop from the queue stack as this allows the while loop to carry on until the length goes down till there is only one value left within the list.

Near the end of the pseudo code you can see that I check whether the length of listOfRooms is equal to 1 or not, this is to ensure that if it is then it doesn't pop and increase the index causing the error: “not enough index” to occur.

| Variables     |         |  |
|---------------|---------|--|
| Variable name | Type    | Description  |
| index         | integer | Holds the index value for when selecting the Dijkstras object within the route list  |
| listOfRooms   | list    | Holds the list of the room numbers the user needs to travel from (Period1 to Period2 to Period3...)                            |
| route         | list    | Holds the Objects of Dijkstras, these objects contain the list of the route for each individual route, so a list within a list |
| file          | csv     | Holds the entire csv file  |
| rows          | list    | Holds every line in the excel file as a 2d array   |

|      |      |   |
|------|------|---|
| Mon  | list | Contains all the routes for Monday                              |
| Tue  | list | Contains all the routes for Tuesday                             |
| Wed  | list | Contains all the routes for Wednesday                           |
| Thu  | list | Contains all the routes for Thursday                            |
| Fri  | list | Contains all the routes for Friday                              |
| Week | list | Contains a list of all the days and the routes within the days. |

### writeToWord(week, tutor)

DEFINE writeToWord(week):

document = Document()

add heading in document of text "Time table for " + tutor

add a headings for each column

create a table with 6 rows and columns with the 'Table Grid' style

row = rable.rows[0]

row0.table.rows[0]

row0.cells[0].text = "From Tutor Base to Period 1"

row1.table.rows[1]

row1.cells[0].text = "From Period 1 to Period 1"

row2.table.rows[2]

row2.cells[0].text = "From Period 2 to Period 3"

row3.table.rows[3]

row3.cells[0].text = "From Period 3 to Period 4"

row4.table.rows[4]

row4.cells[0].text = "From Period 4 to Period 5"

row5.table.rows[5]

row5.cells[0].text = "From Period 5 to Period 6"

rows = [row0, row1, row2, row3, row4, row5]

rowIndex = 0

columnIndex = 1

position = 0

fullRoute is an empty string

while position is not equal to 5 THEN

reset rowIndex to 0

for every route in temp THEN

for every room in route THEN

fullRoute = fullRoute + route + "\n"

remove final character from fullRoute

rows[rowIndex].cells[columnIndex].text = fullRoute

fullRoute is empty string

if rowIndex is not equal to 5 THEN

add 1 to rowIndex

otherwise

pass

**add 1 to position**  
**add 1 to columnIndex**  
**save the document as "Timetable"**

This function is used to both calculate the routes as well as write into a word document. I have imported the python library "python-docx", as it allows me to create and update Microsoft Word (.docx) files.

The first line of the function creates a new document from the built-in default template. The default template is an empty word file. The next line creates the header within the text file. The "\t" within the strings represents a tab indent, this ensures that the heading is in the centre of the page. The next line creates a paragraph, however in this context I have used it to create the headings of the table. I would have implemented this within the table however I wanted to stick to the layout of the actual timetable, and it seemed impossible to get rid of some of the outline of each cell. Then I had to create the table itself, the table contained the attributes rows, cols and style. These represent how many rows columns and what the style of the grid is. If the style attribute was not there, then the basic style would have been used, however that style was not representative of the actual timetable format.

The next few lines simply creates a variable for every row within the table, as well as setting all the values in the first column. "row0.cells[0].text = "From Tutor Base to Period 1", the "0" index value is representative of which column the text is appended to. This is repeated until all of the first column is appended.

The next line creates a list of all the rows to be able to access them easier. The next step was to extract all the routes into a format that can be written into the table. I have used 4 indexes (rowIndex, columnIndex, position), each used for the index value of either the rows in the table, the columns in the table or which day of the week is being read. I have created the "fullRoute" variable, this is used to contain the string needed to be placed in each cell of the table. Firstly, I created a temp variable which takes the routes for the day of the week. For example, the first day would be Monday so the entire route for Monday will be stored as previously shown in the reader class. I then made sure to equate rowIndex to 0 to show that we are writing to the first row. The code then reads through every route in the day and splits it further by reading every room in route. Using every value of room, it appends each node to the string, making sure that each node is set in the next line to be more presentable. Once every node is read it removes the last character as it is a "\n" which is an unnecessary next line that wastes space. Then we add the string into the specified cell (you can see the rows as the x co-ordinate and the cells as the y co-ordinate). Then reset the string value. I check to make sure that the rowIndex has not got above the possible index value, to ensure there is no errors. If the index is not at the limit then it adds 1, this changes the row to the next row. When the rowIndex is equal to 5 it is passed and therefore recognised that the index is at the end. Once every room is appended to their specified rows it moves to the next day and repeats until all cells are filled. The final line is used to save the entire file as "Timetable.docx", the file is saved in the folder of the code.

| Variables     |              |                         |
|---------------|--------------|-------------------------|
| Variable name | Type         | Description             |
| document      | Class Object | Creates a new document. |

|             |         |  |
|-------------|---------|--|
| row0        | row     | Holds the 1 <sup>st</sup> row.                                 |
| row1        | row     | Holds the 2 <sup>nd</sup> row.                                 |
| row2        | row     | Holds the 3 <sup>rd</sup> row                                  |
| row3        | row     | Holds the 4 <sup>th</sup> row                                  |
| row4        | row     | Holds the 5 <sup>th</sup> row                                  |
| row5        | row     | Holds the 6 <sup>th</sup> row                                  |
| rows        | list    | Holds the list of all rows                                     |
| rowIndex    | integer | Index value for which row is being used                        |
| columnIndex | integer | Index value for which column is being used                     |
| position    | integer | Index value for which day within the “week” list is being used |
| fullRoute   | string  | Holds the string that is to be placed in the cell of the table |
| temp        | list    | Holds the list of all routes within a day                      |



## Data Structures

My program uses a queue for when Dijkstra's is performed to calculate which node has a lower weight value. I have used the `heapq` module to do this as the module provides an implementation of the `heapq` queue algorithm, also known as the priority queue.

I have used the following functions to perform this:

### **`heapq.heappush(queue, startVertex)`**

Pushes the `startVertex` into the queue, which maintains the heap invariant.

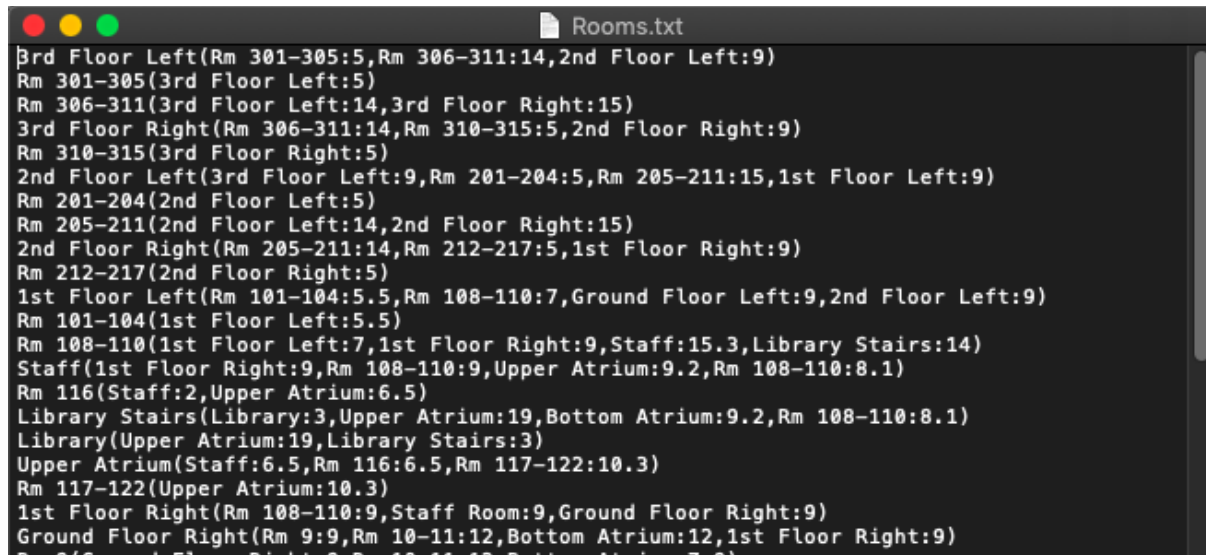
### **`heapq.heappop(queue)`**

Pops and returns the smallest node from the queue.

This module came in handy as it allowed me to easily create a queue for my nodes to be pushed and popped.

## File Structure

Data is read from a text file containing all the distances between nodes as well as the start and end vertexes. This is shown in the image below:



```

3rd Floor Left(Rm 301-305:5,Rm 306-311:14,2nd Floor Left:9)
Rm 301-305(3rd Floor Left:5)
Rm 306-311(3rd Floor Left:14,3rd Floor Right:15)
3rd Floor Right(Rm 306-311:14,Rm 310-315:5,2nd Floor Right:9)
Rm 310-315(3rd Floor Right:5)
2nd Floor Left(3rd Floor Left:9,Rm 201-204:5,Rm 205-211:15,1st Floor Left:9)
Rm 201-204(2nd Floor Left:5)
Rm 205-211(2nd Floor Left:14,2nd Floor Right:15)
2nd Floor Right(Rm 205-211:14,Rm 212-217:5,1st Floor Right:9)
Rm 212-217(2nd Floor Right:5)
1st Floor Left(Rm 101-104:5.5,Rm 108-110:7,Ground Floor Left:9,2nd Floor Left:9)
Rm 101-104(1st Floor Left:5.5)
Rm 108-110(1st Floor Left:7,1st Floor Right:9,Staff:15.3,Library Stairs:14)
Staff(1st Floor Right:9,Rm 108-110:9,Upper Atrium:9.2,Rm 108-110:8.1)
Rm 116(Staff:2,Upper Atrium:6.5)
Library Stairs(Library:3,Upper Atrium:19,Bottom Atrium:9.2,Rm 108-110:8.1)
Library(Upper Atrium:19,Library Stairs:3)
Upper Atrium(Staff:6.5,Rm 116:6.5,Rm 117-122:10.3)
Rm 117-122(Upper Atrium:10.3)
1st Floor Right(Rm 108-110:9,Staff Room:9,Ground Floor Right:9)
Ground Floor Right(Rm 9:9,Rm 10-11:12,Bottom Atrium:12,1st Floor Right:9)
Rm 9(Ground Floor Right:9,Rm 10-11:12,Bottom Atrium:12)

```

My code also has to read from an excel file containing all the rooms for a year 7 tutor group. This is shown below:

|    | A                | B         | C         | D         | E         | F         | G          | H      |
|----|------------------|-----------|-----------|-----------|-----------|-----------|------------|--------|
| 1  | Timetable - 1    |           |           |           |           |           |            |        |
| 2  | as at 20/12/2018 |           |           |           |           |           |            |        |
| 3  |                  |           |           |           |           |           |            |        |
| 4  |                  | Mon:1†    | Mon:2†    | Mon:3†    | Mon:4†    | Mon:5†    | Mon:6†     | Mon:7† |
| 5  | 07a†             | 077A/Ci3† | 077A/Gg3† | 077A/A1†  | 077A/E1†  | 077A/Ma2† | 077A/Mu1†  |        |
| 6  |                  | LTT†      | LXS†      | RMP†      | JH†       | YC†       | EA†        |        |
| 7  | JRS              | 040†      | 315†      | 017†      | 207†      | 37†       | 014†       |        |
| 8  |                  | 077A/Re1† | 077A/Ma1† | 077A/Hi2† | 077A/E2†  | 077A/Sc3† | 077A/Sc2b† |        |
| 9  |                  | PXM†      | ED†       | SW†       | AO†       | JAM†      | SSS†       |        |
| 10 | GO               | 202†      | 306†      | 314†      | 041†      | 032†      | 023†       |        |
| 11 |                  | 077A/Re2† | 077A/Mi2† | 077A/Mi3† | 077A/E3b† | 077A/Sk1† | 077A/Sc3†  |        |
| 12 |                  | KXM†      | ACC†      | JY†       | HL†       | CS†       | JAM†       |        |
| 13 | ABA              | 042†      | 208†      | 210†      | 211†      | 314†      | 032†       |        |

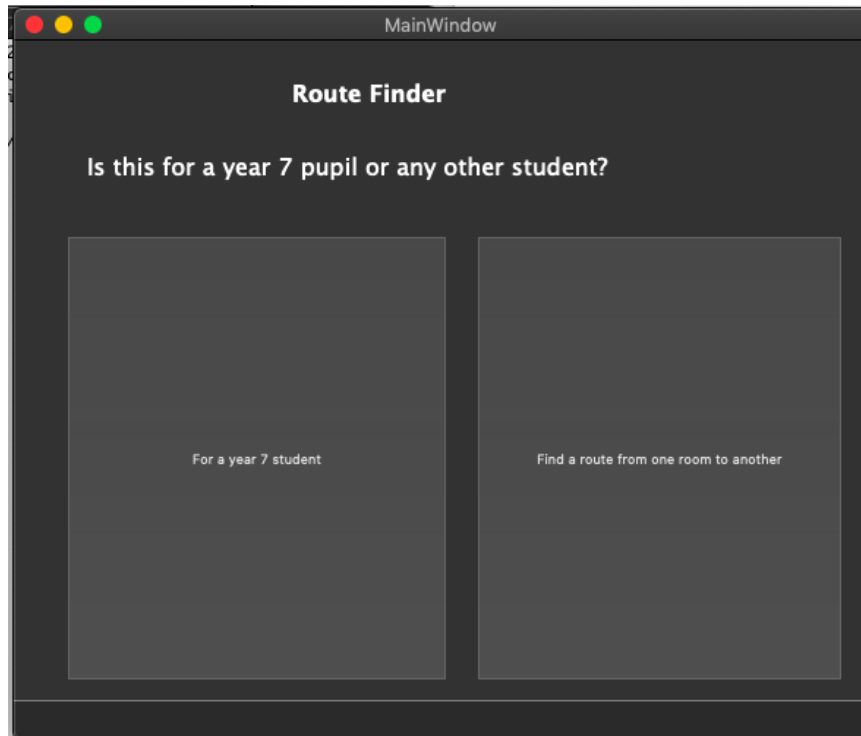
The data is stored in my program when the user selects the year 7 tutor group and wishes to write to a text file. When the “open in word” button is clicked, the “writeToWord” function is called which saves the data by writing to a word document. Each piece of data being saved is split by a new line and split between cells. This is to make the data more presentable. An example of a timetable is shown below:

## Time table for ABA

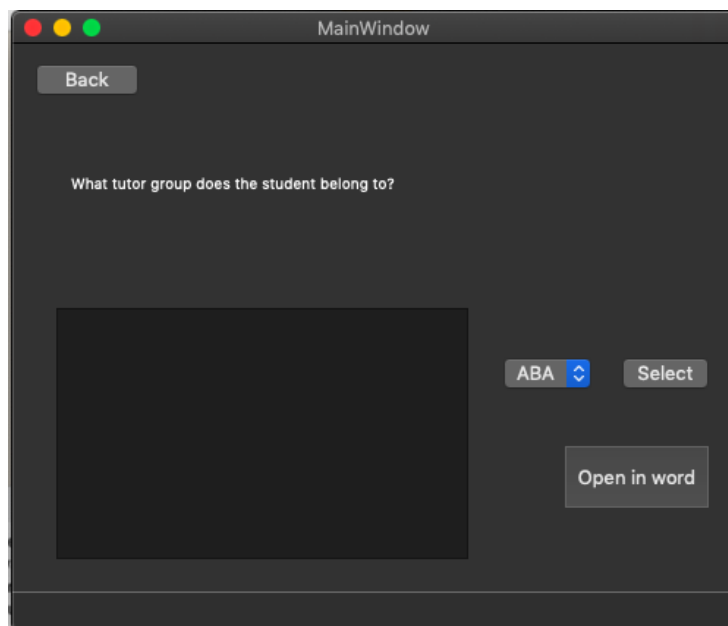
|                             | Mon   | Tue   | Wed  | Thu  | Fri  |
|-----------------------------|---|---|--|--|--|
| From Tutor Base to Period 1 | Rm36-37<br>HallExit<br>StairsG<br>Foyer<br>I<br>Rm9                               | Rm36-37<br>HallExit<br>StairsG<br>StairsE<br>Rm108-110<br>StairsF<br>StairsD<br>Rm212-217 | Rm36-37<br>Rm38-39<br>A<br>Gym   | Rm36-37<br>Rm38-39<br>A<br>Sci/ArtCorr<br>B<br>Rm40-43   | Rm36-37<br>HallExit<br>StairsG<br>StairsE<br>Rm108-110<br>Staff<br>UpperAtrium<br>Rm117-122                                |
| From Period 1 to Period 2   | Rm9   | Rm212-217<br>StairsD<br>StairsF<br>StairsH<br>Rm9<br>I<br>Rm13-14<br>A!<br>Rm12           | Gym  | Rm40-43<br>B<br>Sci/ArtCorr  | Rm117-122<br>UpperAtrium<br>Staff<br>Rm108-110<br>StairsE<br>StairsG<br>HallExit<br>Rm36-37<br>Rm38-39<br>A<br>Sci/ArtCorr |
| From Period 2 to Period 3   | Rm9<br>I<br>Rm13-14<br>A!<br>Rm12   | Rm12<br>A!<br>Rm13-14<br>I<br>BottomAtrium<br>StairsI<br>Rm108-110                        | Gym<br>A<br>Sci/ArtCorr<br>B<br>Rm40-43  | Sci/ArtCorr<br>A<br>A!<br>Rm13-14<br>I<br>Rm9<br>StairsH<br>StairsF<br>StairsD<br>StairsB<br>Rm310-315 | Sci/ArtCorr<br>A<br>A<br>Rm38-39<br>Rm36-37<br>HallExit<br>StairsG<br>StairsE<br>StairsC<br>Rm201-204                      |
| From Period 3 to Period 4   | Rm12<br>A!<br>Rm13-14<br>I<br>Foyer<br>StairsG<br>StairsE<br>StairsC<br>Rm201-204 | Rm108-110   | Rm40-43<br>B<br>Sci/ArtCorr<br>A<br>A!<br>Rm13-14<br>I<br>Rm9<br>StairsH<br>StairsF<br>StairsD | Rm310-315<br>StairsB<br>StairsD<br>Rm205-211<br>StairsC<br>Rm201-204                                   | Rm201-204<br>StairsC<br>StairsE<br>StairsG<br>HallExit<br>Rm36-37<br>Rm38-39<br>A<br>Sci/ArtCorr<br>B<br>Rm40-43           |

## User Interface

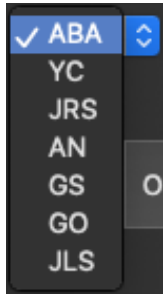
One of my system objectives for my program was for it to have a simple interface the user can use without issues. Using a rough design from prior designing within the system walkthrough stage. I have made the final interface which can be seen below.



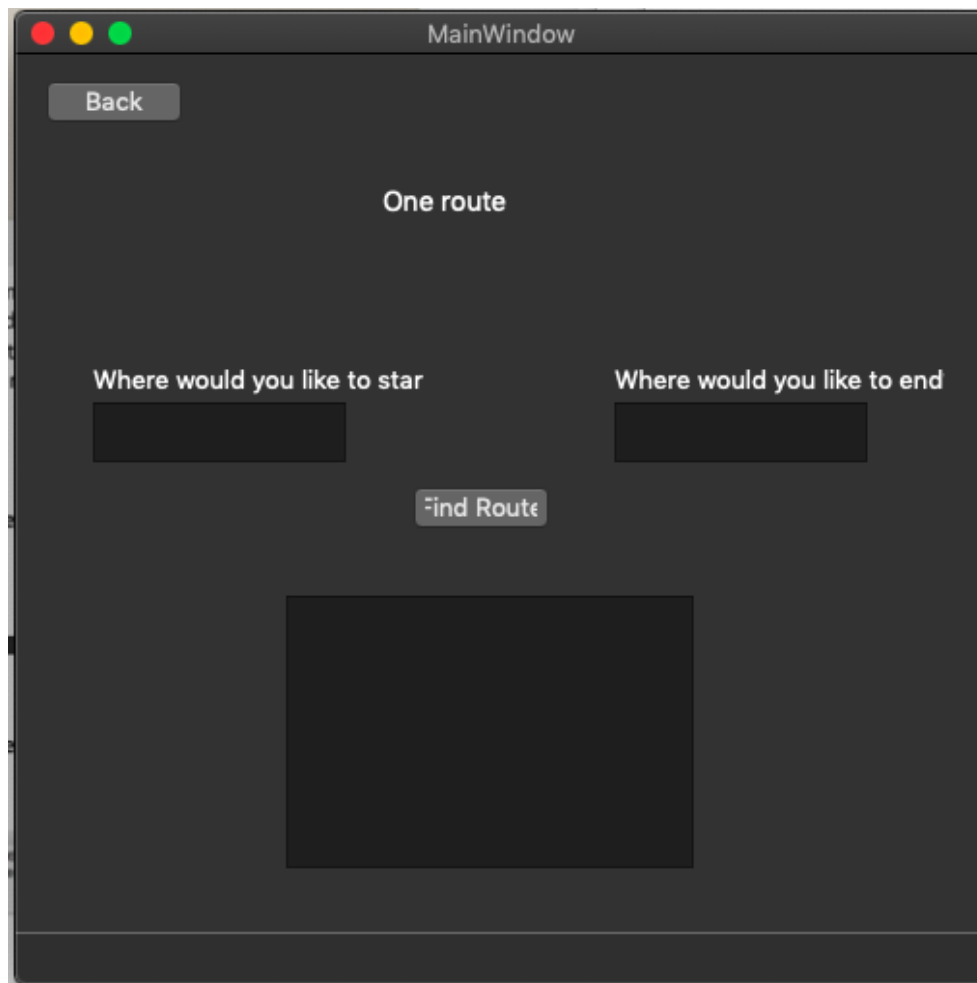
The mainWindow menu is a simple dialog with 2 simple buttons. Each serving a purpose of opening other dialogs, allowing the user to choose what they wish to do.



The year 7 window has more buttons and contains a combobox holding all the values:



There is a simple label that represent informs the user what they need to do and the back button in order to go back to the mainMenu window.



The one route window is the window that presents the main inputs and outputs. In this screenshot you are unable to see some of the labels, however this is only an issue with my laptop. There is a back button that takes the user back to the MainMenu window.

## Algorithm

Dijkstras is an algorithm for finding the shortest paths between nodes in a graph, which in my case represents all the rooms in my school.

Here is the pseudo code format of Dijkstra's I have used:

**Class Dijktras:**

**Initialise:**

**create empty list called route**

**calcShortestRoute:**

**create an empty list as queue**

**allocate the starting vertex**

**push first vertex into queue**

**While there are elements in queue THEN**

**Pop and return smallest value from queue**

**For the number of edges in allEdges list THEN**

**Get the initial vertex (startVertex) of edge**

**Get the target vertex (endVertex) of edge**

**Add the minDistance of initial vertex with the weight of edge**

**If the new distance is less than target vertexes minDistance THEN**

**Target vertex's child becomes initial vertex**

**Update minDistance**

**Push target vertex into queue**

**Otherwise pass**

This algorithm works by creating an empty list, also known as the Queue, then finding the starting vertex and pushing that starting vertex into the Queue. My program checks while there are elements within the queue to carry on popping them out. It then checks all the edges within the allEdges instance and equals the initial and target vertexes. The minimum distances of the initial vertex is added to the distance that is present on the edge of the child of the nodes.

It then checks the new distance and compares to the minimum distance of the target vertex. If it is then the target vertex's predecessor also known as the child becomes the initial vertex. Update the minimum distance and push the target vertex into the Queue. This carries on until every node has been checked and compared to. Once this has been done the 2<sup>nd</sup> part of Dijktras is performed.

**shortestPathTo:**

**prints the minDistance of target vertex which is value of the shortest path  
equal node with target vertex**

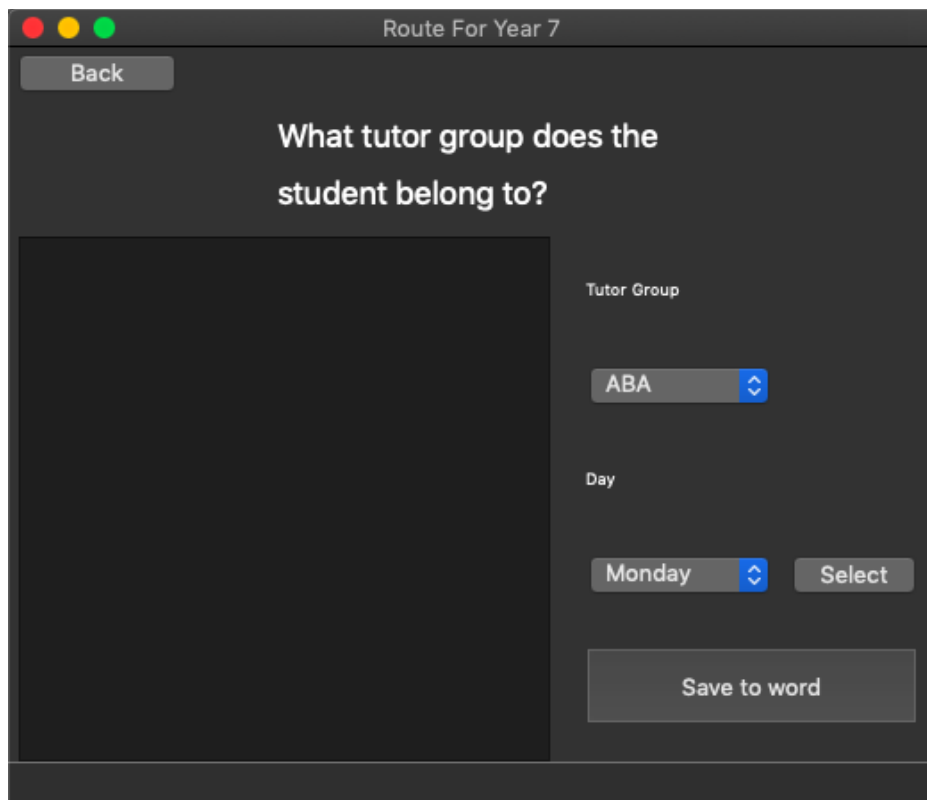
```
while there are child instances within the node THEN
    append child to route
    allocate new node as the child of the current node (backtracks)
reverse the route list
return route
```

When the target vertex is given to the dijkstras object, the 2<sup>nd</sup> part is performed. This part of the algorithm simply equals the node to be analysed with the target vertex. Once done it continuously adds the child of the node into the route and then allocates the new node as the child of the node that was previously checked. This step is repeated until it is no longer possible to back track. The list is then reversed as when backtracking it appends to the list reading backwards rather than in normal order. The route is then returned.

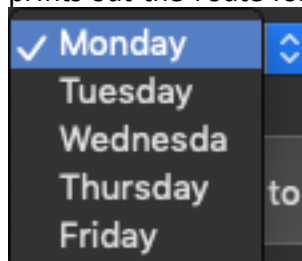
## Changes to User Interface after meeting with client

After further discussions with my client, I have found that there were some minor issues with the client. The client was unable to view the interface on their screens as the text was either too small or the GUI did not scale when increasing size. This was a minor change easily fixed by using a grid format.

The client also wished that within the year 7 window, there was an option to view specified days rather than the whole week, this was due to the fact that output box was very long and difficult to navigate. This also applied small changes within the code, further explained below.



As you can see, there is a new “Day” label as well as a button containing a list of days. This prints out the route for the specified day of the specified tutor group.



I also increased the sizes of some of the buttons to be more accessible. I changed the name of the button from “Open to word” to “Save to word” as the program only saves and not opens word. This is the changed pseudo code within the mainPage class, as follows:



Class mainPage(QMainWindow)

...

buttonClicked:

```

    store value in tutorPickCombo as TutorPicked
    store value in dayBox as DayPicked
    send TutorPicked to checkTutor function and return startVal and row
    send row and startVal to reader function and return Week
    Route is empty string

    Index = 0
    If DayPicked is Monday THEN
        dayIndex is 0
    elif DayPicked is Tuesday THEN
        dayIndex is 1
    elif DayPicked is Wednesday THEN
        dayIndex is 2
    elif DayPicked is Thursday THEN
        dayIndex is 3
    elif DayPicked is Friday THEN
        dayIndex is 4
    for every period in week[dayIndex] THEN
        Route = Route + (next line) + listOfPeriods[index] + (next line)
        for every route in period THEN
            Route = Route + route + (next line)
        add 1 to index
    set text in RouteOutput as Route
    when WordButton is clicked go to wordClicked function

```

wordClicked:

```

    send Week to writeToWord function

```

In short explanation the new variable “DayPicked” and “DayIndex” was added and “otherIndex” was removed. DayPicked stores the value the user selected within the “dayBox” combo box. This is then used to check what day the user has specified and depending on the day is provided with a “dayIndex” value. Each value corresponds to the days in the week list. For example the first value in the week is the list containing all routes for Monday, second value is for Tuesday and so on.

I also made very slight adjustments to the final line in the writeToWord function so that the file saves with the name of the tutor group:

```

    document.save("Timetable for "+ tutorGroup + '.docx')

```

Minor changes to the Main Menu was also made, such as a simple logo of my school on the top right. As followed:

INSERT IMAGE WITH LOGO

There is also very slight changes with the one route window as it also did not scale and therefore I used a grid layout to fix the issue:

One Route

Back

## One route

Where would you like to start?

Where would you like to end?

Find Route

Testing

| Test |  |            |                        |                  |
|------|--|------------|------------------------|------------------|
| 1    | Inputs on Main Menu window                 | User input | Expected outcome       | Observed outcome |
| 1.1  | Selects "For a year 7 student"             | -          | Opens Year 7 window    | Same as expected |
| 1.2  | Selects "Find a route from one to another" | -          | Opens one route window | Same as expected |

| Test  |                         |   |                  |
|-------|-------------------------|---|------------------|
| 2     | Inputs on year 7 window |   |                  |
| 2.1   | Combo Box's selected    | Expected outcome  | Observed outcome |
| 2.1.1 | ABA, Monday             | Prints the route for ABA student for a Monday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.2 | ABA, Tuesday            | Prints the route for ABA student for a Tuesday in output box with indentation and correctly spaced with period specified.   | Same as expected |
| 2.1.3 | ABA, Wednesday          | Prints the route for ABA student for a Wednesday in output box with indentation and correctly spaced with period specified. | Same as expected |
| 2.1.4 | ABA, Thursday           | Prints the route for ABA student for a Thursday in output box with indentation and correctly spaced with period specified.  | Same as expected |
| 2.1.5 | ABA, Friday             | Prints the route for ABA student for a Friday in output box with indentation and correctly spaced with period specified.    | Same as expected |

|        |                |   |                  |
|--------|----------------|---|------------------|
| 2.1.6  | YC, Monday     | Prints the route for YC student for a Monday in output box with indentation and correctly spaced with period specified.     | Same as expected |
| 2.1.7  | YC, Tuesday    | Prints the route for YC student for a Tuesday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.8  | YC, Wednesday  | Prints the route for YC student for a Wednesday in output box with indentation and correctly spaced with period specified.  | Same as expected |
| 2.1.9  | YC, Thursday   | Prints the route for YC student for a Thursday in output box with indentation and correctly spaced with period specified.   | Same as expected |
| 2.1.10 | YC, Friday     | Prints the route for YC student for a Friday in output box with indentation and correctly spaced with period specified.     | Same as expected |
| 2.1.11 | JRS, Monday    | Prints the route for JRS student for a Monday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.12 | JRS, Tuesday   | Prints the route for JRS student for a Tuesday in output box with indentation and correctly spaced with period specified.   | Same as expected |
| 2.1.13 | JRS, Wednesday | Prints the route for JRS student for a Wednesday in output box with indentation and correctly spaced with period specified. | Same as expected |
| 2.1.14 | JRS, Thursday  | Prints the route for JRS student for a Thursday in output box with  | Same as expected |

|        |               |  |                  |
|--------|---------------|--|------------------|
|        |               | indentation and correctly spaced with period specified.  |                  |
| 2.1.15 | JRS, Friday   | Prints the route for JRS student for a Friday in output box with indentation and correctly spaced with period specified.   | Same as expected |
| 2.1.16 | AN, Monday    | Prints the route for AN student for a Friday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.17 | AN, Tuesday   | Prints the route for AN student for a Tuesday in output box with indentation and correctly spaced with period specified.   | Same as expected |
| 2.1.18 | AN, Wednesday | Prints the route for AN student for a Wednesday in output box with indentation and correctly spaced with period specified. | Same as expected |
| 2.1.19 | AN, Thursday  | Prints the route for AN student for a Thursday in output box with indentation and correctly spaced with period specified.  | Same as expected |
| 2.1.20 | AN, Friday    | Prints the route for AN student for a Friday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.21 | GS, Monday    | Prints the route for GS student for a Monday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.22 | GS, Tuesday   | Prints the route for GS student for a Tuesday in output box with indentation and correctly spaced with period specified.   | Same as expected |

|        |               |  |                  |
|--------|---------------|--|------------------|
| 2.1.23 | GS, Wednesday | Prints the route for GS student for a Wednesday in output box with indentation and correctly spaced with period specified. | Same as expected |
| 2.1.24 | GS, Thursday  | Prints the route for GS student for a Thursday in output box with indentation and correctly spaced with period specified.  | Same as expected |
| 2.1.25 | GS, Friday    | Prints the route for GS student for a Friday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.26 | GO, Monday    | Prints the route for GO student for a Monday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.27 | GO, Tuesday   | Prints the route for GO student for a Tuesday in output box with indentation and correctly spaced with period specified.   | Same as expected |
| 2.1.28 | GO, Wednesday | Prints the route for GO student for a Wednesday in output box with indentation and correctly spaced with period specified. | Same as expected |
| 2.1.29 | GO, Thursday  | Prints the route for GO student for a Thursday in output box with indentation and correctly spaced with period specified.  | Same as expected |
| 2.1.30 | GO, Friday    | Prints the route for GO student for a Friday in output box with indentation and correctly spaced with period specified.    | Same as expected |
| 2.1.31 | JLS, Monday   | Prints the route for JLS student for a Monday in output box with   | Same as expected |

|        |                             |  |                  |
|--------|-----------------------------|--|------------------|
|        |                             | indentation and correctly spaced with period specified.  |                  |
| 2.1.32 | JLS, Tuesday                | Prints the route for JLS student for a Tuesday in output box with indentation and correctly spaced with period specified.                                    | Same as expected |
| 2.1.33 | JLS, Wednesday              | Prints the route for JLS student for a Wednesday in output box with indentation and correctly spaced with period specified.                                  | Same as expected |
| 2.1.34 | JLS, Thursday               | Prints the route for JLS student for a Thursday in output box with indentation and correctly spaced with period specified.                                   | Same as expected |
| 2.1.35 | JLS, Friday                 | Prints the route for JLS student for a Friday in output box with indentation and correctly spaced with period specified.                                     | Same as expected |
| 2.2    | When word button is pressed | Expected Outcome   | Observed Outcome |
| 2.2.1  | ABA, (any specified day)    | Creates document called "Timetable for ABA.docx" with heading "Time table for ABA" as well as the correct routes outputted in the correct cells of the table | Same as expected |
| 2.2.2  | YC, (any specified day)     | Creates document called "Timetable for YC.docx" with heading "Time table for YC" as well as the correct routes outputted in the correct cells of the table   | Same as expected |
| 2.2.3  | JRS, (any specified day)    | Creates document called "Timetable for JRS.docx" with heading "Time table for JRS" as well as the correct routes outputted in the correct cells of the table | Same as expected |

|       |                         |  |                  |
|-------|-------------------------|--|------------------|
| 2.2.4 | AN, (any specified day) | Creates document called "Timetable for AN.docx" with heading "Time table for AN" as well as the correct routes outputted in the correct cells of the table   | Same as expected |
| 2.2.5 | GS, ()waF               | Creates document called "Timetable for GS.docx" with heading "Timetable for GS" as well as the correct routes outputted in the correct cells of the table    | Same as expected |
| 2.2.6 | GO                      | Creates document called "Timetable for GO.docx" with heading "Time table for GO" as well as the correct routes outputted in the correct cells of the table   | Same as expected |
| 2.2.7 | JLS                     | Creates document called "Timetable for JLS.docx" with heading "Time table for ABA" as well as the correct routes outputted in the correct cells of the table | Same as expected |
| 2.3   | If Back button pressed  | Goes back to mainMenu window and refreshes year 7 window   | Same as expected |

| Test  |                            |             |                                    |                  |
|-------|----------------------------|-------------|------------------------------------|------------------|
| 3     | Inputs on one route window |             |                                    |                  |
| 3.1   | Start entered              | End entered | Expected outcome                   | Observed Outcome |
| 3.1.1 | 301                        | Reception   | Route from Rm 301-305 to Reception | Same as expected |
| 3.1.2 | 215                        | 34          | Route from Rm 212-217 to Science   | Same as expected |
| 3.1.3 | 1                          | 315         | Route from Rm 1-3 to Rm 310-315    | Same as expected |
| 3.1.4 | 36                         | Offices     | Route from Rm 36-37 to Offices     | Same as expected |
| 3.1.5 | 40                         | Gym         | Route from Rm 40-43 to Gym         | Same as expected |
| 3.1.6 | SFC                        | 102         | Route from SFC to Rm 101-102       | Same as expected |



|        |                     |           |   |                  |
|--------|---------------------|-----------|---|------------------|
| 3.1.7  | Library             | sports    | Route from Library to Sports Hall                       | Same as expected |
| 3.1.8  | 17                  | 104       | Route from Rm 17 to Rm 103-104                          | Same as expected |
| 3.1.9  | 1000                | Reception | "That is not a valid option" is outputted               | Same as expected |
| 3.1.10 | NonInt              | 301       | "That is not a valid option" is outputted               | Same as expected |
| 3.1.11 | 301                 | 305       | Prints just Rm 301-305                                  | Same as expected |
| 3.1.12 | 315                 | 301       | Route from Rm 310-315 to Rm 301-305                     | Same as expected |
| 3.1.13 | Canteen             | 9         | Route from Canteen to Rm 9                              | Same as expected |
| 3.1.14 | 217                 | 101       | Route from Rm 212-217 to Rm 101-102                     | Same as expected |
| 3.1.15 | 320                 | 430       | "That is not a valid option" is outputted               | Same as expected |
| 3.2    | Back Button Pressed |           | Goes back to mainMenu window and refreshes route window | Same as expected |

### Testing Screenshots