24/03/2020: This lab has been revised in light of the Covid-19 shutdown. The required part of the lab now consists of the implementation of an APQ using an array-based binary heap, and of Dijkstra's algorithm using that APQ, tested on a simple graph. The second part of the lab, extending to real-world route maps, is now optional. The deadline has been extended to Friday 3rd April.

This lab is part of the formal continuous assessment for CS2516. It focuses on Dijkstra's algorithm for computing the shortest paths in graphs, using an array-based heap implementation of an adaptable priproty queue. It will count for up to 10% of the total marks available for this module.

Revision

- What is the definition of the shortest path between two vertices in a weighted graph?
- What are the main steps in Dijkstra's algorithm?
- What is an Adaptable Priority Queue? Where would you use it in Dijkstra's algorithm? What APQ implementation would be best for use in Dijkstra applied to standard road maps?

Implementing and testing Dijkstra's Algorithm

The assignment comes in two parts. The first part of the assignment, described here, is required, and the full 10 marks are available for perfect solutions to this part. Nothing has changed in the specification of this part, apart from allowing an unsorted list APQ in case you are unable to get the binary heap working.

Implement Dijkstra's algorithm for finding the shortest paths from a given source vertex to all other reachable vertices in a weighted graph. Your version of Dijkstra must use an Adaptable Priority Queue, and you should implement the APQ using an array-based Binary Heap. *If you can't get the Binary Heap working, then you can instead implement the APQ as an unsorted list instead (but for a maximum of 6 marks for the assignment).*

Test your implementation on the two graphs listed below.

- simplegraph1.txt, which has 5 vertices and 7 edges. The shortest path from v1 to v4 should have length 8, and the preceding vertex on its path should be v3.
- simplegraph2.txt, which has 28 vertices, and represents the graph from Lecture 15, with the names of the vertices changed from a,b,c ... to 1,2,3, ... The shortest path from 14 to 5 should have length 16, and the preceding vertex on its path should be 8.

You will need to read the graphs in from the textfiles. You can write the code to read the textfiles yourself, or you can use the method supplied in graphreader.txt. This method assumes you

have an implementation of the Graph ADT with the same method signatures as given in the solutions to previous labs. You will need to find the internal reference to a vertex - you can write your own method, or you can use the get_vertex_by_label() method from the Graph implementation. You will also need to print out the shortest path results in some sensible format - the simplest thing to do is print out every vertex followed by two pieces of data: the cost of the shortest path to this vertex from the source, and preceding vertex in that path.

Submission instructions

The submission deadline will be Friday 3rd April, at 1pm.

- You must submit a single python file, which must be named assign2-XYZ.py, where XYZ is replaced by your student ID number. This file must contain all of your Python code for implementing the graphs, for implementing Dijkstra's algorithm, for reading in from the text files, and for any other data structures that you have implemented that your code requires to run. Do not include the text from the text files, and do not submit the text files.
- To submit, send an email to k.brown@cs.ucc.ie with the subject "CS2516 submission" and with the python file attached, by the submission deadline. If you are emailing with questions about the assignment, please use a different subject line.
- Note that you can submit multiple times - I will take the last submission.
- Make sure that you compile and test your code before submitting, and that you submit exactly the version that you tested. Between your last successful test and your submission of the code, do not add any new comments or do any other editing.

Plagiarism

This assignment is part of the formal assessment for CS2516, and so you must submit your own work. You must comply with the UCC Policy on Plagiarism. In particular, do not get anyone else to write your code for you, do not copy algorithm implementations from websites, and do not submit code that is copied from anyone else (either from this year's class or from previous years). We are required to check for plagiarised submissions.

If you do use somebody else's code, you must give clear and explicit credit for that code (and you will lose marks if that code is part of what is being assessed).

However, you are free to use any code supplied by lecturing staff in lecture notes or in lab solutions in CS2516 (or CS2515) in academic year 2019-2