ASSIGNMENT 4: SEARCHING WITH GRAPHS

CS3D5A, Trinity College Dublin

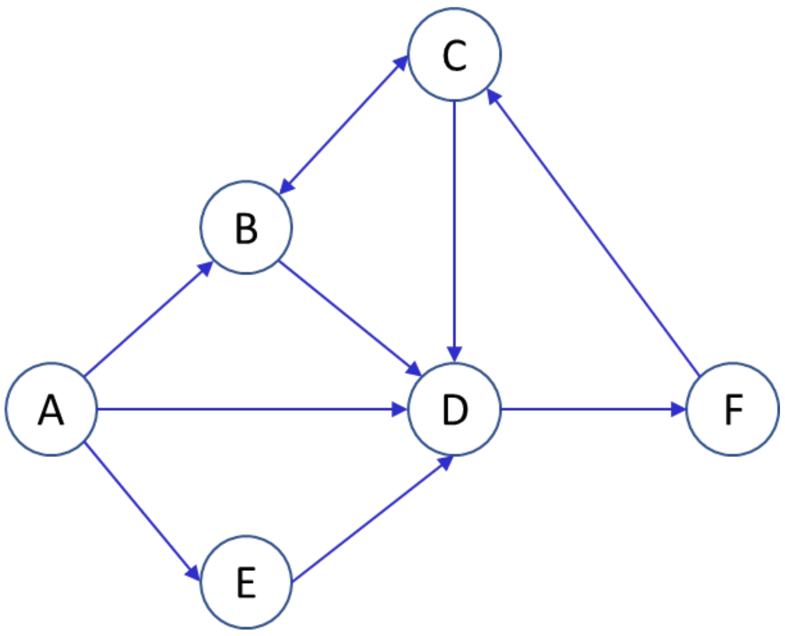
Deadline: 18:00 30/11/2018
Grading: The assignment will be graded during the lab hours on 30/11/2018
Questions: You will able to ask questions during the lab hours on 23/11/2018
Submission: Submit via blackboard. Include a separate .c file for each task, and the short assignment report in pdf, word, or text file.
The report should indicate for task 1 and 2 the graph representation that you used and justify this choice, as well as the outputs you obtained. For task 3 it should document your approach and results.

Goals:

- Learn how to implement a graph, weighted and unweighted, directed and undirected
- Become familiar and learn how to implement graph traversals
- Learn how to implement Dijkstra and use it on a real-world example

Task 1 – BFS and DFS

Choose a suitable graph representation, implement it and represent the graph below. Perform both a Depth First Search and a Breath First search with A as the start vertex, printing the nodes in the order you visit them. When several node choices are available, use alphabetical order to choose a node to process.



Sample output:

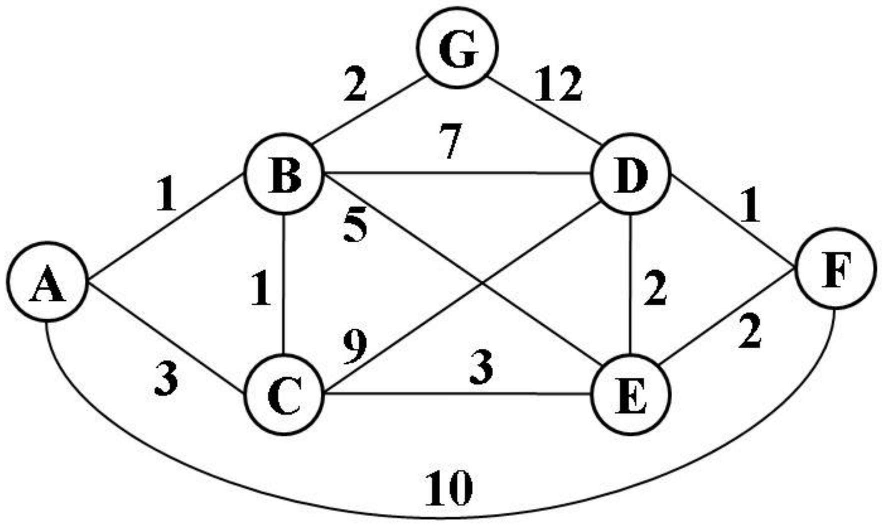
```
DFS: A B C D F E

BFS: A B D E C F
```

Task 1 mark allocations	
Suitable graph representation used	1
Building the graph above	1
Correct Depth First Search implementation	1

Correct Breadth First Search implementation	1

Task 2 – Dijkstra

Choose a suitable graph representation, implement it and represent the graph below (from Tutorial 8). Use Dijkstra's algorithm to calculate the shortest path from A to each of the other nodes. Your algorithm should output the list of the nodes in the order in which they were made permanent, and the shortest distance from A to each of the other nodes.



Task 2 mark allocations	
Suitable graph representation used	1
Building the graph above	1
Correct Dijkstra implementation	2

Task 3 – On the buses

Dublin Bus now provides real-time updates on the location and expected time of arrival for their buses. Google Maps use this information to advise you on the best sequence of buses/trains to take in order to reach your desired destination in the shortest possible time. This is achieved by viewing Dublin as being comprised of a number of nodes in a graph (locations) and edges between those nodes (roads/bus routes/train tracks). Each edge has a weight which depends on how long it will take you to travel towards your intended destination via that route. Given nodes, edges and weights, Dijkstra's algorithm can be used to determine the optimal route to get you from where you are to where you want to be.

For this assignment you have been provided with two files – one contains a list of all bus stops in Dublin (nodes) and the other contains a list of routes between those bus stops (edges). The weights on each of the edges is the distance in metres between each bus stop.

(This is real data which is publically available via a live API. You can grab more complete information from here if you are interested on extending this problem: https://data.dublinked.ie/)

Your task is to load the data from both files and use them to build a graph which models the public transport system of Dublin city. Your program should take as input a starting bus stop ID and a destination bus stop ID. Using Dijkstra's algorithm on the graph, print the optimal sequence of bus stops from source to destination.

The sample output below shows the route to get from stop 300 (Eden Quay) to stop 253 (Beaumont Hospital). Stops are provided with latitude and longitude information so you can actually check your route on Google Maps if you wish.

Note that we are taking a very simplistic, unrealistic view of how the bus service in Dublin works. We don't account for how long you will need to wait at a stop before a bus arrives. We don't account for traffic. We assume that a bus follows the exact same route in both directions. Don't overcomplicate this for yourself. It can be implemented very naturally based on the code you have written for task 2.

Keep in mind that this is an **undirected** graph. So when you load an edge from the edges file, you must ensure that both nodes contain a reference to each other.

Task 3 mark allocations	
Loading data from files (it's in CSV format so just use the parser we've already written) and able to choose locations based on stop ID	1
Printing optimal route between two given bus stops	1

Sample output from stop 300 (Eden Quay) to stop 253 (Beaumont Hospital)

```
Loaded 4806 vertices
Loaded 6179 edges
 300 Eden Quay          53.348269 -6.255763
 497 Amiens Street      53.350503 -6.250701
 515 Amiens Street      53.353504 -6.248089
 516 North Strand Rd    53.355680 -6.245662
4384 North Strand Rd    53.357671 -6.242686
 519 North Strand Rd    53.360302 -6.239553
 521 Annesley Bridge    53.361625 -6.237989
 522 Marino Mart        53.363272 -6.235341
 523 Marino Mart        53.364281 -6.231608
 669 Malahide Road      53.366311 -6.228657
 670 Malahide Road      53.368950 -6.226009
 671 Malahide Road      53.370719 -6.224138
 672 Malahide Road      53.373465 -6.221061
4382 Malahide Road      53.374900 -6.219600
1185 Collins Ave        53.376371 -6.221506
1186 Collins Ave        53.377642 -6.226322
1187 Collins Ave        53.378606 -6.231340
1188 Collins Ave        53.380014 -6.235577
1189 Collins Ave        53.380722 -6.237977
 216 Beaumont Road      53.382329 -6.238176
 217 Beaumont Road      53.384324 -6.236780
 242 Beaumont Road      53.385650 -6.231992
 243 Beaumont Road      53.385779 -6.229525
 253 Beaumont Hospital  53.389942 -6.224379
```