|  |  |  |
| --- | --- | --- |
|  | Faculty of Computing, Engineering and Science |  |

**Assessment Cover Sheet and Feedback Form** 2018-19

|  |  |  |  |
| --- | --- | --- | --- |
| Module Code:  CS3S661 | Module Title:  Computer Networks | | Module Team:  Emlyn Everitt |
| Assessment Title and Tasks:  **“Finding the shortest available paths and calculating Max flow of a Network”** | | | Assessment No.  1 |
| Date Set:  24-Sep-2018 09:00 | | Submission Date:  11-Jan-2019 23:59 | Return Date:  08-Feb-2019 23:59 |

**IT IS YOUR RESPONSIBILITY TO KEEP RECORDS OF ALL WORK SUBMITTED**

|  |
| --- |
| **Marking and Assessment** |
| This assignment will be marked out of 100%  This assignment contributes to 50% of the total module marks. |
| **Learning Outcomes to be assessed** (as specified in the validated module descriptor [https://icis.southwales.ac.uk/](https://icis.southwales.ac.uk/studentmodules/14926/studentmodulespecifications) ):  1) Compare and analyse the theoretical and practical issues of network infrastructure. |
| *Provisional mark only: subject to change and / or confirmation by the Assessment Board* |

## Source (100 marks)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | | **Professional**  **(1st Class)** | **Senior**  **(Upper 2nd Class)** | **Intermediate**  **(Lower 2nd Class)** | **Amateur**  **(3rd Class)** | **Novice**  **(Narrow Fail)** | **Beginner**  **(Fail)** |
| **All Source Code (100 marks)** | **Design,**  **Structure & Efficiency** | Program is designed in a logical manner. Control structures are used effectively and correctly. Data structures, algorithms and patterns are implemented efficiently and appropriately. | Program is generally designed in logical manner. Control structures are used correctly. Data structures, algorithms and patterns are implemented effectively. | Program design is mostly logical. Control structures are used correctly, although more appropriate structures may have been selected. Reasonable data structures, algorithms and patterns are generally implemented. | Program isn’t as clear or logical as it should be. Control structures are frequently used incorrectly. Steps that are obviously inefficient are used. | Program isn’t as clear or logical as it should be. Control structures are occasionally used incorrectly. Steps that are clearly inefficient are used. | No meaningful evidence of logical, efficient or effective decision making in terms of selected data structures, algorithms and patterns. |
| **Comments** | Initial comments are complete. Internal documentation is complete and well suited to the program. Comments clarify meaning where needed. | Initial comments are nearly complete. Internal documentation is nearly complete and generally well suited to the program Comments generally clarify meaning where needed. | Initial comments are complete but internal documentation is in some small fashion inadequate. Comments usually clarify meaning. Unhelpful comments may exist. | Initial comments are incomplete or internal documentation is inadequate. Comments exist, but are frequently unhelpful or occasionally misleading. | Initial comments are incomplete and internal documentation is inadequate. Comments exist, but are frequently unhelpful or misleading. | Code is generally poorly commented with little regard to expectations. |

# Assessment – Detailed Requirements

For detailed coursework deliverables, see “Coursework Requirements” below.

***“Finding the shortest path and calculating Max flow of a Network”***

Optimising networks for maximum data throughput is crucial for the delivery of high quality network services; and Ford-Fulkerson algorithm will find the optimal max flow configuration for any given network based on available network capacity.

**Your task is to implement a solution based on the pathing algorithm developed in tutorials. No other solution will be accepted.**

**First Deliverable: Dijkstra (commented) code [60%]**

Base your solution on the following code/ data structures:

infinity = 1000000

invalid\_node = -1

class Node:

previous = invalid\_node

distfromsource = infinity

visited = False

class Dijkstra:

def \_\_init\_\_(self):

'''initialise class'''

self.startnode = 0

self.endnode = 0

self.network = []

self.network\_populated = False

self.nodetable = []

self.nodetable\_populated = False

self.route = []

self.route\_populated = False

self.currentnode = 0

def populate\_network(self, filename):

'''populate network data structure'''

def populate\_node\_table(self):

'''populate node table'''

def parse\_route(self, filename):

'''load in route file'''

def return\_near\_neighbour(self):

'''determine nearest neighbours of current node'''

def calculate\_tentative(self):

'''calculate tentative distances of nearest neighbours'''

def determine\_next\_node(self):

'''determine next node to examine'''

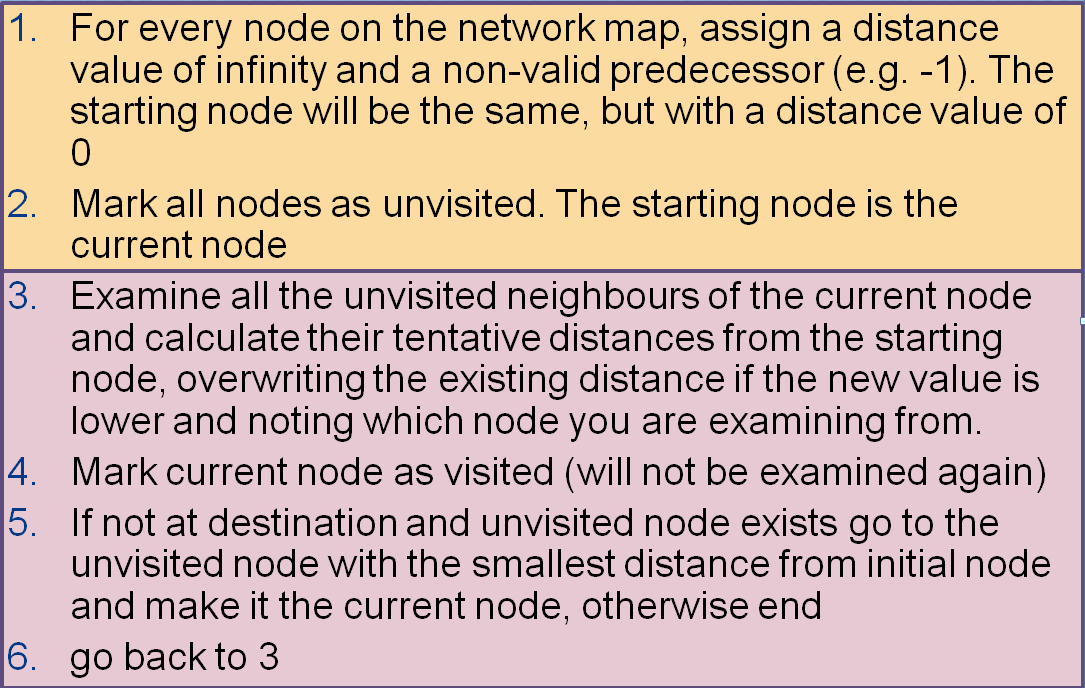
def calculate\_shortest\_path(self):

'''calculate shortest path across network'''

def return\_shortest\_path(self):

'''return shortest path as list (start->end), and total distance'''

and the following pseudo code:



Code your own version of Dijkstra’s algorithm, capable of discovering the shortest path across a network (graph).

Towards this end, your algorithm should be able to:

* Read in a file called “network.txt” – a multi-line file with node, nearest neighbour and distance metric information
* Read in a file called “route.txt” – a single line file with start and destination information, where the line is of the form “B>F” (any start / end combination is possible)
* Print out to the screen the shortest path and total distance

Your code should be constructed using best programming practice, including the appropriate use of functions and data structures. Your code should also be suitably commented to ensure readability and to demonstrate your understanding of how it works.

**Second Deliverable: Max Flow (commented) [40%]**

Using your Dijkstra’s algorithm you created in part 1 as a starting point, create a solution to the Max Flow calculation. This should be done in a fashion similar to the one demonstrated in lectures.

Towards this end, your algorithm should be able to:

* Read in a file called “network.txt” – a multi-line file with node, nearest neighbour and distance metric information
* Read in a file called “route.txt” – a single line file with start and destination information, where the line is of the form “B>F” (any start / end combination is possible)
* Print out to the screen every path discovered during the calculation, along with its associated bottleneck flow value
* Then print out the total max flow for the network

Base your solution on the following code/ data structures:

class MaxFlow(Dijkstra): #inherits from Dijkstra class

def \_\_init\_\_(self):

'''initialise class'''

Dijkstra.\_\_init\_\_(self)

self.original\_network = []

def populate\_network(self, filename):

'''Dijkstra method + need to make a copy of original network (hint)'''

def return\_near\_neighbour(self):

'''similar to Dijkstra method, but takes into account the existence and impact of flows'''

def return\_bottleneck\_flow(self):

'''determine the bottleneck flow of a given path'''

def remove\_flow\_capacity(self):

'''remove flow from network and return both the path and the amount removed'''

def return\_max\_flow(self):

'''calculate max flow across network, from start to end, and return both the max flow value and all the relevant paths'''

## Submission

When submitting your work, please place your code in a **single ‘.py file’** to Blackboard. This .py file should execute Dijkstra’s algorithm first and then the Max Flow algorithm. During this process, both ‘route.txt’ and ‘network.txt’ should remain unchanged.

## Code Comments

It is of particular importance that your code is well commented. The comments you make will form your documentation for this assignment and the quality of those comments will contribute significantly to your overall mark. Comments should attempt to describe the functionality of each section of code and how it fits into the overall behaviour of the program rather than just superficially describing what each line of code does in isolation.

e.g.

**Right**

x++; // x is the iterator used to keep track of the position in the data structure

//which is incremented each time the code loops

**Wrong**

x++ //x is incremented

It should also be noted that the marks awarded for comments will be weighted in favour of the feature complexity they are attempting to describe; with comments successfully describing the more advanced features resulting in the most marks being earned.