Assignment Four

Objectives

- Understand how to use graphs to solve problems in real life.
- Understand how to represent graphs using adjacency lists
- Understand how to traverse graphs
- Understand how to find a shortest path
- Consolidate your skills in time complexity analysis

Admin

Marks 10 marks. Marking is based on the correctness and efficiency of your code. Your code must be well commented. Notice that 10 marks will be scaled up to 25 marks in calculating your final result.
Group? This assignment is completed individually.
Due Time 23:59:59pm Sunday 3 May 2020.
Late Submissions Late submissions will not be accepted!

In this assignment, you will use a digraph to represent a bus network and implement several functions for the bus network.

The bus network of a city X consists of a set of bus stops and a set of buses. Each bus stop has a unique name which consists of one or two English words and the name contains no more than 20 characters. If a bus stop name is two English words such as Parramatta Road, there is a white space character between the two words. Between two adjacent stops of a bus route, there is a fixed distance. Each bus has a bus name which is a string of digits with no more than three digits and the first digit is not 0. The route of each bus is a loop which starts at a stop and ends at the same stop. The bus network is represented by four text files: BusStops.txt, BusNames.txt, BusRoutes.txt and Distance.txt. BusStops.txt contains the names of all the bus stops. BusNames.txt contains the names of all the buses. BusRoutes.txt contains the route of each bus. Distance.txt contains the distance between each pair of two adjacent bus stops. File names can be different from these file names.

The format of BusStops.txt

- All the bus stop names are stored in arbitrary order.
- Each bus stop name has a unique serial number which is a non-negative integer and starts with 0.
- Each line contains the serial number of the name of a bus stop. The serial number and the name are separated by a colon (:)

A sample BusStops.txt is here.

The format of BusNames.txt

- All the bus names are stored in arbitrary order.
- Each line stores one bus name.

A sample BusNames.txt is here.

The format of BusRoutes.txt

- All the bus routes are stored sequentially one after another in arbitrary order, and two adjacent routes are separated by a #.

- The route of each bus consists of a sequence of the serial numbers of bus stops with the following format:
  Bus name: the serial number of bus stop name 1, the serial number of bus stop names 2, …, the serial number of bus stop name m, the serial number of bus stop name 1#, where m is dependent on bus name.
  For each bus, it starts at bus stop names 1 and travels to bus stop names 2, …, bus stop name m, in this order, and comes back to bus stop name 1.

A sample BusRoute.txt is here.

The format of Distance.txt

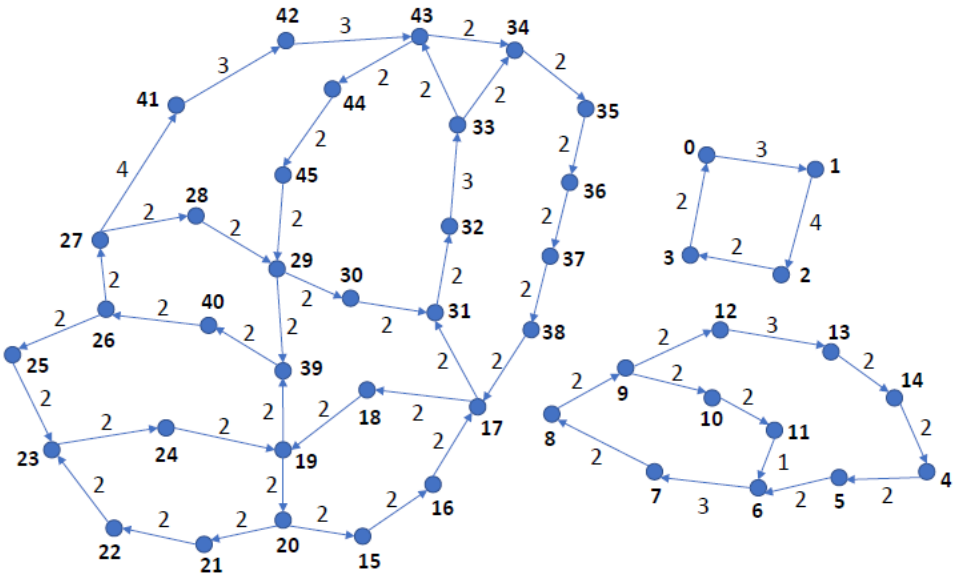
- Each line stores the distance between two adjacent bus stops in the following format: current stop-next stop:distance between current stop and next stop

A sample Distance.txt is here.

We need to solve the following problems:

1. Strong connectivity of a bus network. A bus network is strongly connected if for each bus stop, a passenger can take one or more buses to reach any other bus stop without walking during the travel (if a passenger gets off at a bus stop, he or she gets on to a different bus at the same bus stop during the travel). Given a bus network, we want to determine if a bus network is strongly connected.
2. The maximal strongly connected components of a bus network. A strongly connected component of a bus network is a subnetwork such that for each bus stop in the subnetwork, a passenger can take one or more buses to reach any other bus stops of the subnetwork without walking during the travel (if a passenger gets off at a bus stop, he or she gets on to a different bus at the same bus stop during the travel). A maximal strongly connected component is the largest strongly connected component. Given a bus network, if it is not strongly connected, we want to find the number of maximal strongly connected components.,
3. Reachability test. For each bus stop, we want to know all the bus stops reachable from this bus stop. A bus stop B is reachable from a bus stop A if a passenger can take one or more buses from A to reach B without walking during the travel (if a passenger gets off at a bus stop, he or she gets on to a different bus at the same bus stop during the travel).
4. Travel route. Given any source bus stop and a destination bus stop, we want to find a route with the minimum length such that the passenger can reach the destination stop from the source stop via one or more buses without walking during the travel (if a passenger gets off at a bus stop, he or she gets on to a different bus at the same bus stop during the travel). The length of a route is the sum of the distance of each pair of two adjacent bus stops on the route.

In order to solve the above problems, we need to create a directed edge-weighted graph to represent the bus network, where each vertex denotes a bus stop and each edge (A, B) denotes that there is a bus stopping at A and then at B, and each edge weight is the distance between the two adjacent bus stops. Given the sample bus network represented by BusStops.txt, BusNames.txt, BusRoutes.txt and Distance.txt, the digraph is shown in Figure 1.



Legend:

0: Albion Street
1: Anzac Parade
2: Appian Way
3: Barrenjoey
4: Bathurst Street
5: Bayswater Road
6: Blacktown
7: Bondi Road
8: Bridge Street
9: Broadway
10: Castlereagh
11: City Road
12: Cleveland
13: College Street
14: Crown Street
15: Cumberland
16: Darling Point
17: Eddy Avenue
18: Elizabeth Street
19: Epping
20: Five Ways
21: General Holmes
22: George Street
23: Glebe Point
24: Gore Hill
25: Goulburn
26: Great North
27: Great Western
28: Grosvenor
29: Harris Street
30: Heathcote
31: Henry Lawson
32: Hereward
33: Hunter Street
34: Lane Cove
35: Lennox Bridge
36: Lime Street
37: Macquarie Street
38: Market Street
39: Martin Place
40: Old South Head
41: Old Windsor
42: Park Street
43: Parramatta Road
44: Pennant Hills
45: Phillip Street

Figure 1: The digraph of a sample bus network

After constructing an edge-weighted digraph for a bus network, Problem 1 reduces to the problem of determining if the digraph for the bus network is strongly connected strong (refer to slides 35-36, Graph (II)). Problem 2 is equivalent to the problem of finding all the maximal strongly connected components of the digraph for the bus network. There are many algorithms for finding maximal strongly connected components. You may refer to https://en.wikipedia.org/wiki/Strongly_connected_component. However, the maximal strongly connected component problem in this assignment is a special case of the general maximal strongly connected component problem and can be solved by using simple graph traversal. Problem 3 can be solved by performing a graph traversal. For Problem 4, we can use Dijkstra's shortest path algorithm to solve it.

In order to solve the above problems, you need to write a C program that implements the following C functions:

1. int StronglyConnectivity(const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance). In this function, the four parameters giving the names of the file BusStops.txt, the file BusNames.txt, the file BusRoutes.txt and the file Distance.txt, respectively, in this order, collectively define a bus network. If the bus network given by the parameters of this function is strongly connected, this function returns 1. Otherwise, it returns 0.
2. void maximalStronlyComponents(const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance). The four parameters are the same as in int StronglyConnectivity(const char *busStops, const char *BusNames const char *BusRoutes, const char *Distance). This function prints on the screen all the strongly connected components of the bus network defined by the four parameters with the following format:

   Strongly connected component 1: bus stop1 of this strongly connected component, bus stop 2 of this strongly connected component, …

   Strongly connected component 2: bus stop1 of this strongly connected component, bus stop 2 of this strongly connected component, …

   …

   For each strongly connected component, this function prints the names of its constituent bus stops in arbitrary order. You can determine any reasonable line length for each line displayed on the screen.
3. void reachableStops(const char *sourceStop, const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance). The first parameter const char *sourceStop of this function is the name of a bus stop. The other parameters which collectively define a bus network, are the same as in int StronglyConnectivity(const char *busStops, const char *BusNames const char *BusRoutes, const char *Distance). This function prints the names of all the bus stops reachable from the bus stop given by const char *sourceStop in the bus network defined by the last four parameters. The names of all the bus stops must be printed in **breadth-first-search order**.
4. void TravelRoute(const char *sourceStop, const char *destinationStop, const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance). The first two parameters const char *sourceStop and const char *destinationStop denote the name of a source bus stop and the name of a destination bus stop, respectively. The other parameters which collectively define a bus network, are the same as in int StronglyConnectivity(const char *busStops, const char *BusNames const char *BusRoutes, const char *Distance). This function finds a route with the minimum distance from the source bus stop to the destination bus stop given by the first parameter and the second parameter, respectively, and prints the route on the screen with the following format:

   Travel route from A to B: bus name1(start bus stop, end bus stop), …, bus name k(start bus stop, end bus stop)

   Where A and B are the names of the source bus stop and the destination bus stop, respectively. A travel route consists of one or more parts. Each part has a start stop and an end stop. The start stop of the first part is the source bus stop of the route and the end stop of the last part is the destination bus stop. Each part represents a partial route by taking one bus and is denoted by three attributes: bus name, the first bus stop and the last bus stop of this part in the following format:

bus name(first bus stop, last bus stop)
If there is no route from A to B, this function prints “No route exists from A to B”, where A and B are the names of the source bus stop and the destination bus stop, respectively.
Two adjacent parts of a travel route is separated by a coma (,). You can determine any reasonable length of each line displayed on the screen.

You need to define all the data structures used by your program on your own and write a main function for testing your program. In addition to the above functions, you can define any other helper functions.

Time Complexity Requirements

For each function of your program, you need to analyse its worst-case time complexity in big-O notation.
For the above three functions, there are specific time complexity requirements as follows:

1. int StronglyConnectivity(const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **O(n+m)**
2. void maximalStronlyComponents(const char *busStops, const char *BusNames, const char *BusRoutesy, const char *Distance) **O(n+m)**
3. void reachableStops(const char *sourceStop, const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **O(n+m)**
4. void TravelRoute(const char *sourceStop, const char *destinationStop, const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **O((m+n)log n)**

where n and m are the number of vertices and the number of edges of the digraph for the bus network. For each of the above functions, if your function does not meet the above time complexity requirement, penalty will be applied depending on how much higher your time complexity is.

Design Manual

In this assignment, you need to submit a design manual. In the design manual, you describe how you develop your program. The design manual should contain the following components:

1. Descriptions of major data structures used in your program, including the representation of the edge-weighted graph, bus stops, and bus names, as well as other data structures, including heap, linked lists.
2. Algorithms. Describe all the major algorithms used. For example, Dijkstra’s shortest path algorithm for digraph.
3. Time complexity analyses of all the functions of your program.

After reading your design manual, a fellow student should understand how you developed your program.

Put your design manual in a file named myDesignManual.pdf.

Platform

We strongly recommend you test your program on a CSE machine by using VLAB in order to make marking faster.

Main function

Your program needs to include a main function for testing all the four functions. You have freedom to design your main. However, you need to describe how a user uses your main to test your program with different files defining a bus network in MEADME.pdf.

README.pdf

You are required to prepare a file named README.pdf. README.pdf should contain the following components:

1. A paragraph that gives a simple description of the command for compiling your program and the platform you used to run your program.
2. Detailed descriptions about how to test your program using your main function.
3. Any other useful comments for marker.

Assignment Submission Checklist:

1. Design manual
2. All the C files and header files
3. README file

How to submit your assignment?

a. Go to the Assignment 4 section
b. Click on Assignment Specification
c. Click on Make Submission
d. Put all your files in a single folder and compress the folder with all the files into a single file using tar or zip and submit the compressed file.

Marking Scheme

1. int StronglyConnectivity(const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **2 marks**
2. void maximalStronlyComponents(const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **2 marks**
3. void reachableStops(const char *sourceStop, const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **1 mark**
4. void TravelRoute(const char *sourceStop, const char *destinationStop, const char *busStops, const char *BusNames, const char *BusRoutes, const char *Distance) **4 marks**
5. Deign Manual: **1 mark**

Plagiarism

This is an individual assignment. Each student will have to develop his/her own solution without help from other people. In particular, it is not permitted to exchange code or pseudocode. You are not allowed to use code developed by persons other than yourself. All work submitted for assessment must be entirely your own work. We regard unacknowledged copying of material, in whole or part, as an extremely serious offence. For further information, see the Course Outline.