Duration: 3 lab sessions. **WARNING:** You will find it difficult to complete this exercise if you do not work for all three weeks.

For this lab exercise you should do all your work in your COMP26120/ex10 directory.

For this assignment, you can only get full credit if you do not use code from outside sources. You can get partial credit by using properly attributed code from outside sources (see marking scheme).

**Learning Objectives**

By the end of this lab you should be able to:

- Explain the all-pairs shortest path problem in graphs and describe two algorithms for computing it and their properties
- Implement a priority queue data structure and explain the complexity of its main operations
- Implement Dijkstra’s algorithm, and use it to test the small world hypothesis on social network data.
- Implement a heuristic path finding algorithm using local information, apply it to the same data and then compare the results.
- Use complexity arguments to show which of these two is more efficient for a particular set of data.

**Introduction**

**The “small world” hypothesis**

We are all parts of social networks. These are the networks of people we interact with socially: acquaintances, friends, colleagues, teachers, family, etc. We can think of these networks as graphs. Each individual is a node, and is connected to the people that individual personally knows.

One property these social networks are thought to have is the “small-world property” that any pair of nodes in this very large network is connected by a short path. It is often said that any two humans are connected by a path involving no more than five other people, so the small-world property is also referred to as “six degrees of separation”. 


The six degrees of separation notion comes from a set of experiments carried out by Stanley Milgram in the late 1960s, which attempted to investigate chains of acquaintances in the United States. In the experiment, Milgram sent several packages to 160 random people living in Omaha, Nebraska, asking them to give the package to a friend or acquaintance whom they thought would bring the package closer to a set final individual, a stockbroker living in Boston, Massachusetts, 1450 miles away. Many were not delivered, but those that were passed through on average about six hands from sender to receiver. The experiment itself has since been criticised, but the idea that social networks possess this special small-world property has persisted.

If Milgram’s experiments are to be believed, social networks graphs must possess two properties.

1. Between any two nodes there must be at least one short path (the small-world property).

2. Assuming that not all paths are short, there must be some information available locally at each node which suggests a good next step, i.e one that is part of a short path.

This lab comes in three parts. Parts 1 and 2 test the first property: are the paths short in particular social networks. Part 3 tests the second property: is there a simple way to find the next node in a short path.

The social network data

The social network data is found in /opt/info/courses/COMP26120/problems/ex10/. These are the networks of Facebook friends from two US universities, Caltech and University of Oklahoma. These are represented as graphs in the same format as the graphs from lab exercise 9, and are in the files Caltech.gx and Oklahoma.gx. The Caltech data contains 769 nodes and 33,312 edges (e.g. approximately 43 friends in the same university each). The Oklahoma data contains 17,425 nodes and 1,785,056 edges (approximately 102 friends each). In both cases, only the friends from within the same university are included. The data was collected on a particular date in September 2005.

It should be noted that a Facebook friends network may be very different from a real social network. Unfortunately, it is very difficult to get data on real social networks, because this information is not recorded in any one place.

Description

Part 1: Background Investigation

This part is worth 5 marks in total

Copy into your ex10 directory the file LabReport.tex from the usual COMP26120/problems/ex10 directory. (Don’t copy the data files. They are about 50M and may crunch your quota.) Write in the first section of the LabReport.tex file your statement of the small-world hypothesis and how you are going to test it (without describing the algorithms).

We need a way to find the shortest path between the nodes of a graph, and we are interested in the shortest paths between all of the nodes. This is called the all-pairs shortest path problem. There are (at least) two algorithms to solve this: Dijkstra’s algorithm and Floyd’s algorithm (also called Floyd-Warshall’s algorithm).

For these graphs, Dijkstra’s algorithm is more efficient. We would like you to learn about the complexity of these two algorithms and find out why Dijkstra’s algorithm is more efficient for these graphs. Write the complexities of the two algorithms in the LabReport.tex document and the argument showing that Dijkstra’s algorithm is more efficient for these graphs.
Part 2: Finding the shortest paths

This part is worth 15 marks in total

The graphs should be represented in the same format as in Lab exercise 9, so you can use the same code to read in the data files.

Part 2a: Implementation of the shortest-path algorithm

You will need to learn about the Dijkstra’s algorithm and implement it. Dijkstra’s algorithm requires a priority-first search. It contains a graph traversal algorithm which is similar to those you may have implemented in lab exercise 9, except you don’t pull from your search list the most recently added (depth-first search) or the one which has been longest in the list (breadth-first search), but the one which is closest in distance to the starting node. Thus, you need to implement a priority queue.

Updated: The remainder of Part 2a has been updated since the lab was first published.
The social network given in this lab is not a weighted graph. You should process length of each edge in your algorithm as one equal unit. It has been commented by some students that in an unweighted graph there is no need for a priority queue. Therefore, there are two options for the next part:

1. Implement a priority queue. You can implement a priority queue inefficiently, by using a linear search to find the best item each time you pull from the queue. You will get more points if you implement this in one of the efficient ways. For instance, it can be implemented as a heap. The relevant section in the textbook, is section 2.4, and particularly 2.4.3. Finding shortest paths in graphs is discussed in Chapter 7 of the textbook. (If you use code from outside sources for this, you can only get partial credit for this part. See marking scheme for details.)

2. Implement a queue. You can use a simple queue implementation if you also provide a justification for why this is correct for unweighted graphs in your LabReport.tex. The explanation should argue in terms of the workings of Dijkstra’s algorithm. You will be asked to explain this justification during marking, but you must have written out a justification in your report.

After implementing the (priority) queue, you can complete the implementation of Dijkstra’s algorithm and put it all in a file called part1.c. You should use and extend graph.h and graph_functions.c from exercise 9.

Part 2b: Test for the small-world property

Use your algorithm to measure properties of the paths in the two social networks. Write down

- execution time;
- total amount of reachable nodes;
- average length of shortest paths between any two reachable nodes.

and your conclusions from that in the LabReport.tex file. Each of the statistics takes up 1 mark. Are these small-world networks?

Part 3: Approximate path finding and heuristics

This part is worth 10 marks in total

Once an all-pairs shortest path algorithm is run, it is possible to give each node a look-up table of the next node in the shortest path to any target node (a routing table). However, sometimes it is not feasible to run an algorithm which explores the entire graph to generate this table.
Certainly the participants in Milgram’s experiment could only know about their friends and acquaintances, and had to choose which of those to give it to based on some quantity used to determine which of those was most likely to be the next step on the shortest path, or at least a short path. We will refer to such a quantity as a heuristic. The heuristic they might have used may have been geographical (give it to someone who has some connection to somewhere close to Boston), or social (give it to someone who knows a lot of people).

Using heuristics to find approximations to shortest paths may be useful when the graph is too big to implement Dijkstra or Floyd, or when it is changing too rapidly for there to be up-to-date information about the entire network at any node. In this part, you will investigate whether such heuristics find short paths in these networks.

We will call this approach “heuristic path following”. It works like this. You choose some heuristic. To find a path to a target, each node chooses the next node from its outlist based on this heuristic. For instance, for the Facebook data, the heuristic could be, choose the node among its friends with the most friends (the friendship relation is mutual, so out-degree is the same as in-degree in these graphs). Using this heuristic, the algorithm works as follows, to find the path from SOURCE to TARGET:

```
CURRENT ← SOURCE;
while (TARGET not in CURRENT.OUTLIST) and (CURRENT.OUTLIST not empty)
    add CURRENT to PATH
    Let MAXOUT be the unvisited node in CURRENT.OUTLIST with largest out-degree
    CURRENT ← MAXOUT;
endwhile
add CURRENT to PATH
```

Figure 1: heuristic path following

Here CURRENT is the current node, and PATH stores the entire path. This approach is clearly not guaranteed to find the shortest paths (why not?).

Your task is to implement this algorithm and test to what extent it finds shortest paths. Put the code in a file called `part2.c`.

Compare the
- execution time;
- total amount of reachable nodes;
- average length of shortest paths between any two reachable nodes.

of your heuristic method with the results of Part 2 (actual shortest paths) on the two data sets. Write the results and conclusions in the `LabReport.tex` document.

**Submission and Marking Scheme**

You should submit `LabReport.tex`, `graph.h`, `graph_functions.c`, `part1.c` and `part2.c`.

Marking scheme to appear soon.