1 Laboratory Description and Procedures

1.1 Learning Objectives

You should be able to do the following after completing this laboratory.

1. Understand the concept and use of a graph.
2. Understand the power and generality of graph search in solving many important problems.
3. Understand the importance of object-oriented design and code re-use.
4. Understand the importance of and application of the shortest path problem.

1.2 Key Words

You should be able to define the following key words after completing this laboratory.

1. graph
2. vertex
3. edge
4. graph search
5. shortest path

1.3 Scenario

As we’ve learned about in class, the shortest path problem arises in many applications. As an example, consider how the Internet operates. The public Internet is a collection of networks that interconnects millions of computing devices, or end systems, throughout the world. End systems are not connected directly to the Internet. Instead, end systems are connected to a local or regional network through an Internet Service Provider (ISP), which in turn is connected to the Internet through a national or international ISP, such as UUNET or Sprint. Each component network is comprised of end systems interconnected by intermediate switching devices known as routers. Routers are responsible for forwarding packets along a path through the network from the origin to the destination. Special routers called gateway routers are responsible for the packets that traverse multiple networks. To facilitate the forwarding of packets, each router stores a table that lists for each destination the next router on the path to that destination. The table is constructed and
updated based on the results obtained from a routing algorithm. Each component network, or autonomous system (AS), is managed independently, but each router within an AS must run the same protocol. OSPF (Open Shortest Path First) is a common protocol used for routing packets within an AS. The basic idea of OSPF is that each router constructs a complete topological map (a directed graph) of the entire AS by exchanging information packets. Each router then runs Dijkstra’s Algorithm to determine the shortest path to all other routers from itself. The link costs used by the algorithm must be specified by the network administrator. The algorithm is run whenever new link information is received by a router as well as periodically to ensure that the routing table contains valid next hop information. Clearly, the graphs in question here are extremely large and computational efficiency is critical. In this lab, you will implement Dijkstra’s Algorithm as it would be implemented within a simple router.

1.4 Design and Analysis

In this lab, you will use the implementation of Dijkstra’s Algorithm for finding a shortest path in a graph to test two implementations of a priority queue. The data structures that you will use are the GiMPy graph class we have used in previous labs and the priority queue class from BLImPy, along with a new priority queue class you will implement yourself.

1.5 Program Specifications

You should implement a priority queue class. You may do so based either on a “naive” implementation or one based on a heap. In either case, your class should be a drop-in replacement for the PriorityQueue class of BLImPy.

1.5.1 Algorithms

The algorithm exemplified in this lab is Dijkstra’s Algorithm for solving the shortest path problem with nonegative edge weights.

1.5.2 Data Structures

The data structures used in this lab are the graph data structure with adjacency lists, and the heap priority queue data structure.

2 Laboratory Assignments

2.1 Programming and Analysis (50 points)

1. (20 points) Implement a priority queue class with the API given in priority_queue.py, as described above. You can implement your priority queue either as a heap stored in the BinaryTree class of GiMPy or as an unordered list (the latter is the “naive” implementation in which you simply search through the list in each iteration to find the minimum.

2. (10 points) Test your priority queue implementations by using the implementation of Dijkstra’s Algorithm in the module network.py. Generate random networks and check that the length of the shortest path found is the same with your implementation as when using the PriorityQueue implementation in BLImPy.
3. (10 points) Compare the empirical running time of the algorithm for graphs ranging in size from 500 to 2000 and with different densities. Graph the running time of the algorithm both with your implementation of priority queues and with the implementation in BLImPy. Compare these running times against the theoretical worst case. How well do they match? You should run multiple random graphs to get an “average”. Please submit your code for generating these graphs so they can be reproduced.

4. (10 points) For graphs of a fixed size of 1000, produce a graph showing running times for graph densities from 0.1 to 1 by intervals of 0.1. You should run multiple graphs of each density to get an “average”. Please submit your code for generating these graphs so they can be reproduced.

2.2 Analysis and Follow-up Questions (30 points)

1. (10 points) Consider the following idea for implementing a priority queue. We keep the items in an unordered list, but we keep track of the maximum items seen so far and return that item when required. In this way, both insertion and finding the maximum are constant time. Will this idea work in general?

2. (10 points) Consider an implementation of a priority queue in which we simply sort the items each time we need to find the maximum item. To achieve this, we keep a separate list of items that have been inserted since the last sort and merge them into the sorted list whenever we need to find the maximum. What are advantages and disadvantages of this approach? When would it work well? Use an analysis of theoretical running times to justify your conclusions.

3. (10 points) If every edge has length 1, what is the running time of Dijkstra’s Algorithm?

4. (a) (5 points) Dijkstra’s algorithm is typically only used when the edge weights are non-negative. Give an example of a graph with negative edge weights for which the algorithm doesn’t work correctly.

   (b) (5 points) Explain why the algorithm will work correctly when the only edges with negative weights are edges incident to the source node.