CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2017
Webpage: `http://www.cs.clemson.edu/~bcdean/` — TTh 12:30-1:45
Handout 16: Lab #11 — McAdams 119

1 Single-Source Shortest Paths: Diameter of the USA

Suppose you want to take a road trip from one point in the continental USA to another point, following a shortest path along the way. What is the largest distance you can travel? In graph theoretic terms, this is known as the "diameter" of the USA road network – it is the largest of all shortest path distances over all pairs of nodes. Computing the diameter of a graph is somewhat computationally expensive, since we generally need to compute the shortest path from every node to every other node. For even medium-sized graphs, this can take too long. However, we shall see in this exercise that one can approximate the diameter of a network to within a factor of 2 much faster, using only two calls to a single-source shortest path algorithm like Dijkstra's algorithm.

2 Estimating Diameter

As shown in the figure below, suppose we compute shortest paths out of and into an arbitrarily chosen node M (we'll use the node in the USA road network representing the curb outside McAdams hall). This takes two runs of Dijkstra's algorithm. Let X and Y denote the longest outgoing and incoming shortest path lengths for M.

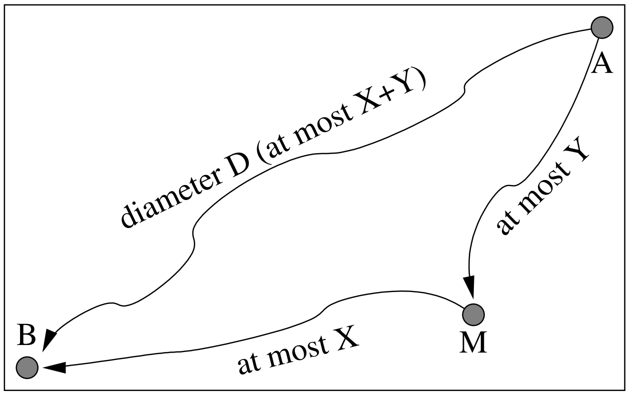

Figure 1: X is the length of the longest outgoing path from M, and Y is the length of the longest incoming path into M. The shortest path from A to B (of length D) corresponds to the actual diameter of the graph.

If the actual diameter of our graph is a path of length D from node A to node B, then we observe that the shortest path length from A to M is at most X, the shortest path length from M to B is at most Y, and therefore by the triangle inequality $D \leq X + Y$. Moreover, since D is the longest of all shortest path lengths, $D \geq X$ and $D \geq Y$, so $D \geq \max(X, Y)$. Finally, the maximum of two numbers is at least as large as their average. We can summarize all of this below in one inequality chain:

$$\frac{X+Y}{2} \leq \max(X, Y) \leq D \leq X + Y.$$

Therefore, the true diameter of our graph is bounded between $(X + Y)/2$ and $X + Y$, so we have it approximated to within a factor of 2.

3 Reading the Input

The input to this lab is a directed graph representing the entire USA road network, with 23,947,347 nodes and 58,333,344 directed edges:

```
/group/course/cpsc212/f17/lab11/usa_network.txt.gz
```

The first two numbers in the file tell you the number of nodes and edges in the network, and each remaining line describes a directed edge, specifying the node index of its source and destination, along with the number of seconds required to drive along the edge. Nodes are numbered sequentially from 0 up to $N - 1$, where N is the total number of nodes. The curb outside McAdams Hall is node 17,731,931.

Since this is a very large network, it is stored in compressed form using `gzip`, a popular compression tool. The commands `gzip` and `gunzip` can be used to compress and uncompress files with this compression scheme, and you can use `zless` or `zcat` to view the contents of a compressed file without actually expanding it on disk. Since reading a large file over the network file system can take a long time (particularly with many of your lab-mates reading the file at the same time), you should make a copy of this file in the `/tmp` directory on your local machine:

```
cp /group/course/cpsc212/f17/lab11/usa_network.txt.gz /tmp
```

Since other students may be remotely logged into your machine, it is polite to give them read permission to this file:

```
chmod a+r /tmp/usa_network.txt.gz
```

Code has been provided to you for reading this file directly in its archived state without the need to uncompress it on disk:

```
/group/course/cpsc212/f17/lab11/sp.cpp
```

If you look at this code, it uses the `popen` command to open a pipe from another process (running `zcat` on the input) as if it was an input file. Since `popen` does not support a C++ stream interface, the input here is done in C style, using `fscanf`. However, you should use C++ constructs to add the rest of your code.

The code provided to you reads all edges into a large array of structures, and then sorts the array so that edges emanating from the same node will be grouped together. That is, the array will contain all edges leaving node 0, followed by all edges leaving node 1, etc., as shown below:

You will need to write code to generate the indexing structure on the left in Figure 2, where the

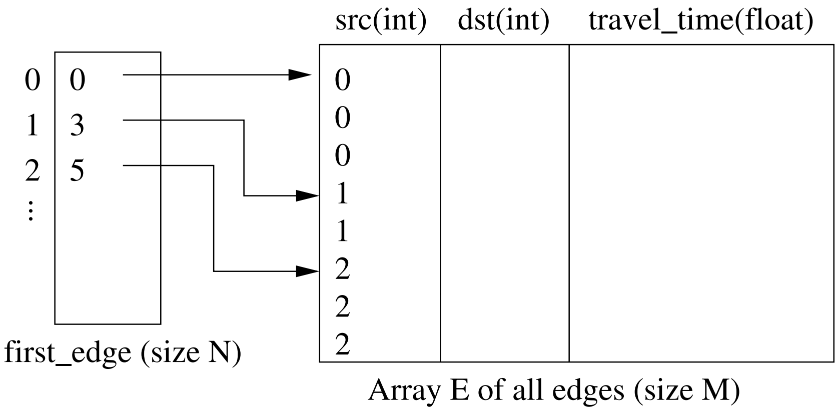

Figure 2: Data structure storing the nodes and edges in our network.

*i*th entry, `first_edge[i]` specifies the index within the edge array for all the edge records leaving node *i*. This gives a convenient way to represent a network in a space-efficient fashion: we store all the edges in a large sorted list, and then for each node we store an index into the point in this list containing the edge records leaving that node.

4 Dijkstra's Algorithm

Your main task in this lab is to write an implementation of Dijkstra's single-source shortest path algorithm. You will want to store the distance labels of nodes in an array of floats, and you should also use an STL `priority_queue` to ensure that each iteration of Dijsktra's algorithm always selects the pending node with minimum distance label.

The STL priority queue supports most of the functions that are now familiar to you from the STL queue: `empty()`, `push()`, `pop()`, and `top()` (this last function is analogous to `front()` in a queue – it returns a reference to the highest-priority element in the structure). Note that the STL priority queue is a "max" priority queue, so calling `q.top()` returns a reference to the *maximum* element in the queue, whereas Dijkstra's algorithm needs to find the minimum. To remedy this, you will want to negate elements as you place them in the queue. Moreover, since you want to associate distances with nodes in the queue, you will want to store `pair<float,int>` objects in the priority queue, where the first part of the pair is a distance label and the second part is a node index. That way, the priority queue will be ordered by distance but when you extract a record from it, you will be able to access the node associated with this record as well.

For further reference, you may want to consult the pseudocode for Dijsktra's algorithm provided in the on-line CpSc 2120 lecture slides.

5 Submission and Grading

For this lab, you will receive 8 points for correctness and 2 points for having well-organized, readable code. Zero points will be awarded for code that does not compile, so make sure your code compiles on the lab machines before submitting!

Final submissions are due by 11:59pm on the evening of Friday, December 1. No late submissions will be accepted.