Shortest Path Algorithm with Real World Data

Brad Peterson - Weber State University

Goal: To work with Dijkstra's shortest path algorithm with realistic data. In the book, the algorithm uses a 2D array to store our graph data. This works for small amounts of nodes, but once we hit thousands of nodes, we're wasting massive amounts of memory with unusable data. This assignment has you modify your data structure so that it can handle thousands of nodes.

Instructions: This assignment comes in two parts. The test part where you will do almost all of your work. After, you should attempt your algorithm on directed road network of the city of Rome, Italy, from 1999. (Courtesy of DIMACS from here: http://www.dis.uniroma1.it/challenge9/download.shtml)

In order to make this work, we need to convert our matrix to CSR format. This allows us to leave out missing data. For example, the path for a node from itself to itself is zero. But we don't have to specify that number in the graph data structure. And if there isn't a path from a node to another node, we can assume that weight is a large number. And we also don't have to specify that number in the graph data structure.

Please see the lecture videos and the homework assignment notes on how to do this. A short summary is given with Figure 12-8 in the book. The CSR arrays should be:

```
weights: 16 2 3 5 3 12 7 10 4 5
columns: 1  3 4 2 1 1  4 1  2 3
row:     0 3 4 5 7 10
```

How this works is if you want to get an edge weight, such as from node 3 to node 4, you would do the following. Take the starting node (3), and go to that index in the row array (row[3] = 5). Add one to your starting node, and go to that index in the row array, (row[3+1] = 7). Now you are going to look at indexes 5 <= indexes < 7 in your columns array. At column[5] = 1. That 1 is no good, we are trying to get to node 4. So lets try column[6] = 4. That's perfect. We are trying to get to node 4, and we found the destination node. So go to weights[6] = 7, and we have our weight. To go from node 3 to node 4, the cost is 7.

If we tried to go from node 4 to node 3, we would see that row[4] = 7 and row[4+1] = 10. So go between 7 <= index < 10. column[7] = 1, that doesn't work. column[8] = 2, that doesn't work. column[9] = 3, found it. So go to weights[9] = 5. 5 is the weight from node 3 to node 4.

It we tried to go from node 0 to node 2, we would see that row[0] = 0, and row[0+1] = 3. So go between 0 <= index < 3. column[0] = 1, that's not it. column[1] = 3. That's not it. column[2] = 4. That's not it. We're out of options. That means the weight doesn't exist. In our program, we are assuming that if the edge doesn't exist, we will just return a really big number.

Your goal is to get all TODOs in the assignment completed. Specifically you need to:

- Replace the `weights[][]` 2D array with a function call that uses those two arguments. The function should use the passed in source and destination nodes, look through the CSR arrays, find the weight if it exists, and return it. If the weight doesn't exist, return LARGE_NUMBER.
- Declare four global arrays called graphWeights, columns, row, and smallestWeight. Allocate them appropriately in `createCsrArrays()`.
- Iterate through the edges collection and populate the three CSR arrays.

You will notice that I read all edge data into an STL vector. An STL vector is simply a collection of objects. It's almost exactly the same as the ArrayList in Java. You can add in objects using the .push_back() method. To retrieve objects, you can access them using [] indexing. You will also see that I sort the vector twice, this is to help get everything in order so your upcoming CSR matrix will make more sense.

Once you convert the edges in the vector to the CSR matrix, you will not need that STL vector any longer. You will work with the CSR matrix exclusively.

In Visual Studio, to have the program read the .gr file, you need to put the .gr file in the base directory of your project (not necessarily in the same directory as your .cpp file).

When it's functional, you should see this for a test run.

```
Would you like to do:
1. Test run
2. Full run
Enter your selection: 1
Halfway done sorting...
Finished sorting...
Test run debugging.  Your CSR arrays are:
weights: 16 2 3 5 3 12 7 10 4 5
columns: 1 3 4 2 1 1 4 1 2 3
row: 0 3 4 5 7 10
Which node would you like to search from: 0
Looking up shortest path for 0 of 4 amount of nodes
Looking up the shortest path for the last node
Which node do you want to see weights for (-1 to exit): 1
For node 1 the cost is 10
Which node do you want to see weights for (-1 to exit): 2
For node 2 the cost is 7
Which node do you want to see weights for (-1 to exit): 3
For node 3 the cost is 2
Which node do you want to see weights for (-1 to exit): 4
For node 4 the cost is 3
Which node do you want to see weights for (-1 to exit): 5
Error: That's no node with that ID!
Which node do you want to see weights for (-1 to exit): -1
```

For the actual data run, you should see the following. Make sure you start searching from node 1, as node zero in the Rome data doesn't point to or from any other node. This should take under 10 seconds total processing time.

```
Would you like to do:
1. Test run
2. Full run
Enter your selection: 2
Reading edge # 0
Finished reading 3354 nodes and 8871 edges.
Halfway done sorting...
Finished sorting...
```

```
Which node would you like to search from: 1
Looking up shortest path for 0 of 3353 amount of nodes
Looking up shortest path for 100 of 3353 amount of nodes
Looking up shortest path for 200 of 3353 amount of nodes
Looking up shortest path for 300 of 3353 amount of nodes
Looking up shortest path for 400 of 3353 amount of nodes
Looking up shortest path for 500 of 3353 amount of nodes
Looking up shortest path for 600 of 3353 amount of nodes
Looking up shortest path for 700 of 3353 amount of nodes
Looking up shortest path for 800 of 3353 amount of nodes
Looking up shortest path for 900 of 3353 amount of nodes
Looking up shortest path for 1000 of 3353 amount of nodes
Looking up shortest path for 1100 of 3353 amount of nodes
Looking up shortest path for 1200 of 3353 amount of nodes
Looking up shortest path for 1300 of 3353 amount of nodes
Looking up shortest path for 1400 of 3353 amount of nodes
Looking up shortest path for 1500 of 3353 amount of nodes
Looking up shortest path for 1600 of 3353 amount of nodes
Looking up shortest path for 1700 of 3353 amount of nodes
Looking up shortest path for 1800 of 3353 amount of nodes
Looking up shortest path for 1900 of 3353 amount of nodes
Looking up shortest path for 2000 of 3353 amount of nodes
Looking up shortest path for 2100 of 3353 amount of nodes
Looking up shortest path for 2200 of 3353 amount of nodes
Looking up shortest path for 2300 of 3353 amount of nodes
Looking up shortest path for 2400 of 3353 amount of nodes
Looking up shortest path for 2500 of 3353 amount of nodes
Looking up shortest path for 2600 of 3353 amount of nodes
Looking up shortest path for 2700 of 3353 amount of nodes
Looking up shortest path for 2800 of 3353 amount of nodes
Looking up shortest path for 2900 of 3353 amount of nodes
Looking up shortest path for 3000 of 3353 amount of nodes
Looking up shortest path for 3100 of 3353 amount of nodes
Looking up shortest path for 3200 of 3353 amount of nodes
Looking up shortest path for 3300 of 3353 amount of nodes
Looking up the shortest path for the last node
Which node do you want to see weights for (-1 to exit): 0
For node 0 the cost is 99999999
Which node do you want to see weights for (-1 to exit): 1
For node 1 the cost is 0
Which node do you want to see weights for (-1 to exit): 2
For node 2 the cost is 193
Which node do you want to see weights for (-1 to exit): 3
For node 3 the cost is 381
Which node do you want to see weights for (-1 to exit): 4
For node 4 the cost is 596
Which node do you want to see weights for (-1 to exit): 5
For node 5 the cost is 2891
Which node do you want to see weights for (-1 to exit): 6
For node 6 the cost is 5458
Which node do you want to see weights for (-1 to exit): 7
For node 7 the cost is 5584
Which node do you want to see weights for (-1 to exit): 8
For node 8 the cost is 4698
Which node do you want to see weights for (-1 to exit): 9
For node 9 the cost is 8839
```

```
Which node do you want to see weights for (-1 to exit): 10
For node 10 the cost is 8935
Which node do you want to see weights for (-1 to exit): -1
```