1.204 Computer Algorithms in Systems Engineering

Spring 2010

Problem Set 3

Due: 12 noon, Monday March 8, 2010

Problem 1: Arlington network

a. Problem statement

You are given a database table with TIGER (US Census and US Geological Survey) data for the town of Arlington, Massachusetts. Please download it, unzip it and attach it to your SQL Server instance. It is under Homework->Homework3 (ArlingtonNetwork.zip).

The database consists of a single table, ArlingtonNetwork. It is not normalized, since it repeats latitude and longitude data. You will SELECT the data you need from this table. The data you will use will be normalized—you're only selecting 3 columns from the table. The columns in the table and their descriptions are given below:

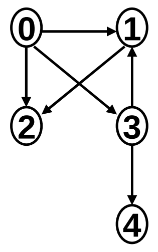
1. ID: Arc or street segment number.
2. StreetName
3. StartLatitude
4. StartLongitude
5. StartNode: node number at start of arc
6. EndLatitude
7. EndLongitude:
8. EndNode: node number at end of arc
9. StartAddress-L: the starting street address of the street on the left side
10. EndAddress-L: the ending street address of the street on the left side
11. StartAddress-R: similar to column 9, on the right side
12. EndAddress-R: similar to column 10, on the right side
13. ZipCode-L: zip code on the left side
14. ZipCode-R: zip code on the right side
15. Distance: length of the arc in meters

You only need to use fields 5, 8 and 15 to create a network on which you can find shortest paths. These fields define from node (field 5), to node (field 8) and distance (field 15). Optional: also use street name (field 2), and display it when you output the network (in the steps below). You may ignore the other fields. Assign your own arc number instead of field 1.

You will read this data from the SQL Server database table and create a modified Graph data structure. You will extend the adjacency array data structure to allow a program to not only easily find all the arcs out of a node, but also to find all the arcs into a node. An example is:

If we read the arcs from input and sort by ‘from’ node, we get:

From	To	Cost	(Arc number)
0	1	43	0
0	2	52	1
0	3	94	2
1	2	22	3
3	4	71	4
3	1	37	5



– **The ‘from’ node repeats when out-degree > 1**

We recast this structure as arrays H, To, Cost (and T):

(Node)	H	(Arc)	To	Cost	Index	T	(Node)
0	0	0	1	43	0	0	0
1	3	1	2	52	5	0	1
2	4	2	3	94	1	2	2
3	4	3	2	22	3	4	3
4	6	4	4	71	2	5	4
5	6 (sentinel)	5	1	37	4	6(sent)	5

In the example, we use an ‘index’ array to put the arcs in order of their destination. If we read down ‘index’, it says that the arcs must be accessed in the order 0, 5, 1, 3, 2, 4 to be sorted by ‘to’ node rather than ‘from’ node:

- There are no arcs into node 0
- Arcs 0 and 5 go into node 1
- Arcs 1 and 3 go into node 2
- Arc 2 goes into node 3
- Arc 4 goes into node 4
- There is a sentinel arc, arc 6, from a sentinel node, node 5, to terminate the array, used in the same way as the sentinel in the ‘from’ node direction

If we look at the ‘T’ array, it is analogous to the ‘H’ array: it tells us the first arc into a node, but it must use the ‘index’ array as an intermediate step to get the arcs in the correct ordering. Array ‘index’ is a level of indirection, similar to a ‘pointer sort’ or ‘index sort’, if you’ve seen or used one of those before.

Use Java’s Arrays.sort() method to help you. For example, make a copy of the array of arcs, sort them by ‘to’ node instead of ‘from’ node, and then place the arc numbers from that sort into the ‘index’ array. Or, use JDBC and SQL Server: SELECT the data from the table a second time with ORDER BY the destination node, get the second ResultSet and place the arc numbers in the 'index' array, as described below.

This representation, which allows easy access to all arcs into or out of a node, is useful in many transportation and logistics problems (including the branch and bound example we'll do near the end of this term). For example, we may have a network with many vendors who ship to a few warehouses. We solve a vehicle routing problem to pick up goods from the vendors and deliver them to the warehouse using the forward representation (arcs out of each node), as usual. However, we select warehouse locations using an algorithm that minimizes distances between the warehouse and all vendors; this is best done with the backward representation (arcs into each node), which lets us trace paths <u>to</u> a warehouse from all its customers. If we have, for example, 100,000 customers and 20 warehouses, it's the difference between running 20 and 100,000 shortest path tree computations.

b. Assignment

The assignment is to read a set of arcs from a SQL Server database and place it into an adjacency array data structure that allows us to easily find the arcs into and out of each node. You are free to do this any way you wish; a suggested method is given below.

a. Use the Graph.java code from lecture as the basis of the solution.

b. Modify the Graph constructor to read all the arcs from the database instead of a file. Don't do anything in the database or JDBC except read the table data into your Java program. Your constructor does not need arguments; you can hardwire the database connection into your code.

c. Modify the Graph constructor to create a 'backward' (arcs into a node) as well as a 'forward' (arcs out of a node) representation of the graph. The backward representation has a 'tail' or 'T' array, analogous to the 'head' or 'H' array, except that it holds all the arcs that come into a node (rather than come out of a node). Since the arcs are sorted in the order that works for the forward representation, you will need to have an 'index' array to help you create the backward representation, as described in the example above.

d. Write a traverseReverse() method to traverse the network using the backward star form.

e. Modify the main() method to:
- Create your modified Graph object, which will have both forward and backward adjacency representations
- Call traverse() to print out the network based on the forward representation
- Call traverseReverse() to print out the network based on the backward representation

Hints:

- You only need to connect to the database once. Use the later code examples from lecture 4 as a guide in which the connection is in the first 'try' block and the SQL statements are in the second 'try' block.

- You can get the number of nodes and number of arcs from the database by 'SELECT … MAX(…). You can assume that the nodes and arcs are numbered consecutively, without gaps, if you wish (though your code will probably work with gaps too.) You may want to do these queries before reading all the arcs from the database, so that you know how large your H, to, from and other arrays will be. You may also just dimension your arrays large enough to hold 2000 entries, which is simpler and fine for this homework.

- After you've read the arcs from the database, and know how many arcs and nodes there are, you can then use the Graph.java code to create the H, to and dist arrays. It should require no changes.

- If you sort the array of arcs using Java's Arrays.sort() method, first make a copy of the array of arcs using Java's Arrays.arraycopy() method. You'll need to keep the original version sorted by 'from' node.

- You can then use the same logic to create the T array from the copy of the graph that is sorted by 'to' node. Just replace H by T, origin by dest, etc.

- You then need to write the code to create the index array.

f. In a separate file (Word document, text file or any typical format), please briefly answer the following questions:

1. If your network has two or more arcs from a 'from' node to a 'to' node, do your traverse methods traverse these arcs correctly? Why or why not? (We'll address how this affects shortest path and minimum spanning tree algorithms later; don't worry about those issues here. Address only whether traverse and reverseTraverse work correctly.)

2. If your network has 'loop' arcs that have the same 'from' node and 'to' node, do your traverse methods traverse correctly? Why or why not?

3. If you had to add an arc to your graph, after having built it, what is the time complexity of adding it to the adjacency array representation? Use O() or Ω() or Θ() notation, whichever is most descriptive. Briefly support your answer.

Problem 2: k-ary heaps

We covered binary heaps in lecture. An extension to binary heaps is k-ary heaps. A k-ary heap satisfies the heap property and each internal node in a k-ary heap has k children. (For k = 2, the k-ary heap is a binary heap.) Assume that there are n elements in the heap. Assume this is a max k-ary heap, with the largest element at the top.

Please answer the following questions. Briefly support each answer.

1. If you represent a k-ary heap using an array, and the root is element 0:
    a. For an element at index i in the array, what are the indices of its k children?

    b. What is the index of the parent of the element at index i? Assume that element i is not the root.
2. What is the height (or depth) of a k-ary heap as a function of n and k?
3. Analyze the run time for an insert operation in a k-ary heap. Use O() or Ω() or Θ() notation, whichever is most descriptive.
4. Which operation is simpler in a k-ary heap (k>2), inserting an element or deleting an element? Explain briefly, using O() or Ω() or Θ() notation.

Optional question:

The insert operation is faster in a k-ary heap than in a binary heap, but the delete operation is generally slower. Building a heap (heapify) is independent of k. If the operations occur with the same frequency, what is the optimal value of k? An approximate answer is fine.

Turn In

1. Place a comment with your full name, Stellar username, and assignment number at the beginning of all `.java` files in your solution.
2. Place all of the files in your solution in a single zip file.
    a. Do not turn in electronic or printed copies of compiled byte code (`.class` files) or backup source code (`.java~` files)
    b. Do not turn in printed copies of your solution.
3. Submit this single zip file on the 1.204 Web site under the appropriate problem set number. For directions see **How To: Submit Homework** on the 1.204 Web site.
4. Your solution is due at noon. Your uploaded files should have a timestamp of no later than noon on the due date.
5. After you submit your solution, please recheck that you submitted your .java file. If you submitted your .class file, you will receive **zero credit.**

Penalties

- 30 points off if you turn in your problem set after Monday (March 8) noon but before noon on the Wednesday (March 10). You have two no penalty two-day (48 hours) late submissions per term.
- No credit if you turn in your problem set after Wednesday (March 10).