COMP 272

Homework 10

Strongly Connected Components

The algorithm for finding strongly connected components (SCCs) you will implement was originally discovered by Kosaraju Rao. It uses two depth first traversals (DFT). Here is the algorithmic steps.

Let G=(V,E) be a directed graph. Let Reverse(G) be the graph with edges in G reversed in their direction.

Let the vertex set V in G be numbered 0,1,2,…, n-1, where n is the number of vertices.

Algorithm SCC (with some hints on implementation)

1. Run the post-Order DFT on Reverse(G) and obtain the finishingNumbers of each vertex. Finishing Numbers go from 0 to n-1 as each vertex in the post-order DFT gets printed in the code I have provided. This can be stored in an array of int called **finishing**, where **finishing[k]=v means that k is the finishing number of vertex v.** Post-order DFT on Reverse(G) can be obtained by just running Post-Order DFT on G using the inList.
2. Using the finishing number as the new labels of the vertex set, run a regular DFT. This part needs to be run starting at the highest finishing number to the lowest by decrementing by 1 in the outer loop. In the outer loop of DFT each time a vertex p is unmarked, it would be considered the *leader* of the strongly connected component it will be part of.
3. You will print out just two values after executing Kosaraju's algorithm. (i) number of strongly connected components, and (ii) max size among all strongly connected components. The answers are respectively 10559, and 71307. Everyone should get these correctly. Also, the data file has edges like (0,0) or (82166,82166). No need to add these edges. I have provided a modified Directedgraph.java file that does not add these edges. Use the new one.
4. After you have obtained the leaders, construct a new Reduced graph, Reduced (G) (using the Directed Graph class) with all leaders as the vertex set V. Essentially, you will use a single vertex v for each SCC of G, where v is the leader. ~~You may map the leaders to integers 0 through k if G has k+1 SCCs. For instance, in the above situation, there will be four vertices in Reduced (G) 0,1,2,3 where 0 corresponds to leader 0, 1 corresponds to leader 9, 2 corresponds to leader 6, and 3 corresponds to leader 8.~~ You may use a special HashMap<Integer, DirectedNodeList> so you could use the leaders as representatives for their respective components. Here the *key* will be the leader value *k* and the *value* will be the DirectedNodeList structure that gives the inList and OutList of other leaders that are connected to *k.* Build edges such that a directed edge from v1 to v2 exists in Reduced(G) if and only if there is any directed edge from a vertex in the SCC corresponding to v1 to a vertex in the SCC corresponding to v2. The reduced graph's vertex set size should be the same as number of SCCs i.e., 10559. You may represent the SCC's as a HashMap<Integer, ArrayList<Integer>> where *k* is the leader and *value* is the list of vertices in the strongly connected component represented by the leader *k.*
5. Use the dataset provided. The number of vertices is 82,168, the number of directed edges is 948,464. If you get stack overflow (like I did on my Mac) due to recursion, please use command line option -Xss as in *java -Xss100m Classfile* to overcome stack overflow.