Basic Algorithms (V22.0310); Fall 2005; Yap

HOMEWORK 3

Date Due: Oct 12

Instructions

- For Chapter 4, we want you to begin by reading sections 4.4 and 4.5 (these are about shortest paths and minimum spanning trees, respectively).

Question 1

(10 Points)

(a) Question 1, page 107 in text. How many topological orderings in a graph.

(b) Suppose G is a digraph consisting of two linear lists of nodes. These lists have $n_1 \geq 1$ and $n_2 \geq 1$ nodes respectively. Give a formula for the number of topological orderings of G.

Question 2

(10 Points)

Question 2, page 107 in text. Detecting cycles in bigraphs.

Question 3

(50 Points)

It is unfortunate that the text did not really give a good algorithm for computing topological ordering of a digraph. Consider the following solution: let the vertices of the input graph be $V = \{1, 2, \ldots, n\}$. Recall that a topological ordering is a listing

$$(v_1, v_2, \ldots, v_n) \tag{1}$$

of all the vertices such that edges have the form (v_i, v_j) where $i < j$. We say v_i has **rank** i in this topological ordering. We represent (1) by an array called $Rank[1..n]$ such that $Rank[v_i] = i$. For instance, if the topogical ordering in (1) is $(3, 1, 2, 4)$, then $Rank[1, 2, 3, 4] = [2, 3, 1, 4]$.

Here is a skeleton of a DFS-based algorithm for computing $Rank$. Initially, $Rank[v] = -1$ for each v. We use DFS(v) to assign a rank to v: but before we can assign a rank to v, we must assign ranks to all the nodes that can be reached from v. We have a global counter R that is initialized to n; to assign a rank to a vertex v, the current value of R is used. We then decrement R. We also need a driver program to call DFS.

```
Driver Program
    Input: adjacency digraph $G = (V, E)$ where $V = \{1, \ldots, n\}$.
    Output: $Rank[1..n]$
        Initialization:
            For $i = 1$ to $n$, $Rank[i] \leftarrow -1$.
            $R \leftarrow n$
        For $i = 1$ to $n$
            If $Rank[i] < 0$ then DFS($i$).
        Return($Rank$).
```

```
DFS($v$)
        For each $u$ that is adjacent to $v$,
            If $Rank[u] < 0$ then DFS($u$).
        $Rank[v] \leftarrow R$.
        $R \leftarrow R - 1$.
```

(a) Prove that this algorithm is correct if the input graph G is a DAG.

(b) Show how to modify this algorithm so that it would be correct for DAG's, but it could detect a non-DAG. HINT: It is NOT a major change to the above algorithm. Two additional lines suffices. The idea is this: if G is not a DAG, it has a cycle C. Let u be the first vertex of the cycle that is seen. Then, when we do $DFS(u)$, we will see a "back edge" from some node v in C to u. We try to detect such back edges (v, u). We suggest adding the value "$Rank[u] = 0$" to indicate that u has been seen but not yet assigned a rank.

(c) Write a Java program to implement your solution for Topological sort. To test your program, please provide two input graphs with at least 10 vertices each: one input graph is a DAG and one is not.

HINT: Unfortunately, Java does not have a standard graph package. To do your own graph class, the simplest solution is to use the adjacency matrix representation. Slightly more sophisticated is to use adjacency list, taking advantage of the list implementations in Java (LinkedList or ArrayList).

Question 4

(30 Points)

We want to run hand simulations of Prim's and Kruskal's MST algorithm on the following graph G_{12} on 12 nodes (Figure 1).

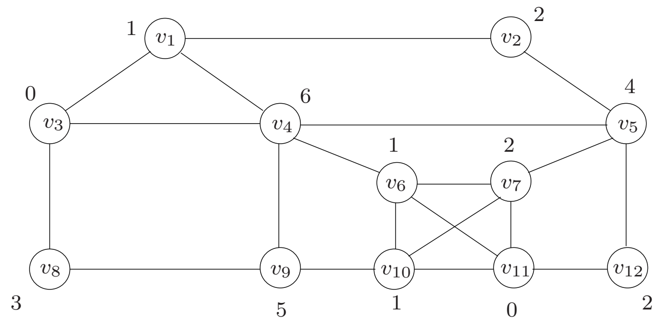

Figure 1: Graph on 12 vertices

We only assign weights $W(u)$ to the nodes u of the graph. The weight of an edge (u, v) is just the sum of the weights $W(u) + W(v)$.

Show the set of edges picked (and any intermediate structures) when Prim's algorithm and Kruskal's algorithm are run on this graph.

Question 5

(10 Points)

Do Questions 1 (page 188), and Question 2(a) (page 189) in text. True or False questions about MST.