CSE 326
Winter 2008
Assignment 6
Due 3/5/08

For all algorithm and data structure design problems please provide elegant pseudocode and an adequate explanation of your methods. If is often helpful to include small examples demonstrating the method. Put your name at the top of each sheet of paper that you turn in.

1. Some project planning applications use a labeled acyclic directed graphs to represent the jobs and job times on a project. A vertex in the graph represents a job and its label represents the time the job will take. A directed edge from one vertex to another represents the fact the job represented by the first vertex must be completed before the job represented by the second vertex. Assume we have a directed acyclic graph $G = (\{1, 2, ..., n\}, E)$ with vertices labeled by non-negative integers $c_1, c_2, ..., c_n$. The label c_i represents the time job i will take. Assume futher that every vertex is reachable by some path from vertex 1, vertex 1 has in-degree 0, vertex n is reachable by some path from every vertex, and n has out degree 0. Vertex 1 represent the beginning of the project and vertex n represent the end of the project. The length of a path from 1 to n is the sum of the labels on the vertices along the path. Design an algorithm based on the topological sort algorithm to find the length of a longest path from 1 to n in the graph. The length of the longest path represents how long the entire project will take. Sometimes a longest path is called a critical path. Your algorithm should use the adjacency list representation of a graph. The labels can be stored in an additional array. Your algorithm should run in linear time. Hint: ultimately you will need to compute the length of the longest path from 1 to every other vertex. In the topological sort, when a vertex achieves in-degree 0, the length of the longest path from 1 to it should be known.

2. Consider the following sequence of disjoint union / find operations: union(1,2), union(2,3), union(3,4), union(4,5), union(5,6), union(6,7), union(7,8), union(9,10), union(11,12), union(13,14), union(15,16), union(1,10), union(1,12), union(14,15), union(1,16). In this problem we don't assume that the inputs to union are roots, so that two find operations are performed during the union to find the roots before pointing one root to another. Show the resulting up tree after these operations for each case below. In each case count the number of nodes visited in all the find operations. In the case of path compression some nodes are visited twice.

   (a) There is no path compression on the finds and the root of the first argument points to the root of the second argument.

   (b) There is path compression on the finds and the root of the first argument points to the root of the second argument.

   (c) There is no path compression on the finds and weighted union is used.

(d) There is path compression on the finds and weighted union is used.