CS180 Winter 2011

Homework 4

Due: February 2

When submitting your homework, please include your name at the top of each page. If you submit multiple pages, *please staple them together*. We also ask that you do something to indicate which name is your last name on the first page, such as underlining it.

Please submit your solutions with the problems solved in the order they are given.

1. Consider the following definitions:
   A Bottleneck Spanning Tree is a spanning tree $T = (V, E)$ built as follows: for any two vertices in G, define the *bottleneck path* between them to be the maximum cost of an edge traversed; a minimum bottleneck path between two vertices minimizes this value. The bottleneck spanning tree is a spanning tree that, for all vertices $u, v \in V$, the minimum bottleneck path is in T.
   A Lexicographic Spanning Tree is also a spanning tree $T = (V, E)$; if we have two Lexicographic Spanning Trees, and want to know which the cheaper is, we look at which has the cheaper most-expensive edge. If these are tied, we check the second most-expensive edge. We repeat this until we find a different cost edge; if we find no such edge, the two have the same cost.

   (a) Prove that a Minimum Bottleneck Spanning Tree is equivalent to a Minimum Lexicographic Spanning Tree.

   (b) Prove that these are equivalent to the Minimum Spanning Tree definition from lecture (minimize the sum of the cost of edges).

2. A given binary relation $\star$ on a set X is said to be an equivalence relation if and only if it is reflexive ($a \star a$), symmetric (if $a \star b$ then $b \star a$) and transitive (if $a \star b$ and $b \star c$ then $a \star c$). In mathematics, given a set X and an equivalence relation $\star$ on X, the equivalence class of an element a in X is the subset of all elements in X which are equivalent to a:

   $$[a] = \{x \in X | x \star a\}$$

   Consider a graph $G = (V, E)$. Consider the set E and the relation $\star$ such that, $e_1 \star e_2$ for $e_1, e_2 \in E$ if either $e_1 = e_2$ or there exists a simple cycle in G that has both e_1 and e_2. Prove that the relation $\star$ is an equivalence relation.

3. Consider n jobs $j_1 \dots j_n$. Lets say that we have a supercomputer and n smaller computers. Each of the n jobs have to be first executed on the supercomputer and then on one of the smaller computers. For every $i \in [n]$, the job j_i needs p_i time of the supercomputer and then f_i time on one of the smaller computers. We want to finish all the jobs as soon as possible. Give an efficient algorithm that gives an ordering of the way the jobs should be performed on the super computer. Prove and give time complexity.

4. Let $G = (V, E)$ be a connected, undirected graph. An *articulation point* of G is a vertex whose removal disconnects G. A *bridge* of G is an edge whose removal disconnects G. A

biconnected component of G is a maximal set of edges such that any two edges in the set lie on a common simple cycle. We can determine articulation points, bridges and biconnected components using depth-first search. Let $G_t = (V, E_t)$ be a depth first tree of G.

(a) Prove that the root of G_t is an articulation point of G if and only if it has at least two children in G_t.

(b) Let v be a nonroot vertex of G_t. Prove that v is an articulation point of G if and only if v has a child s such that there is no back edge from s or any descendent of s to a proper ancestor of v.

(c) Let $d[v]$ be the discovery time of node in the dfs tree G_t. Then let,

$$low[v] = \min\{d[v], d[w] : (u, w) \text{ is a back edge for some descendent } u \text{ of } v\}$$

Show how to compute $low[v]$ for all vertices $v \in V$ in $O(E)$ time.

(d) Show how to compute all articulation points in $O(E)$ time.

(e) Prove that an edge of G is a bridge if and only if it does not lie on any simple cycle of G.

(f) Show how to compute all the bridges of G in $O(E)$ time.

(g) Prove that biconnected components of G partition the nonbridge edges of G.

(h) Give an $O(E)$-time algorithm to label each edge e of G with a positive integer $bcc[e]$ such that $bcc[e] = bcc[e']$ if and only if e and e' are in the same biconnected component.