SPRING 2002: **COT 6405** ANALYSIS OF ALGORITHMS
[HOMEWORK 4; DUE APR 4 **in my office** BEFORE CLASS]

Problems

27. Given below is an algorithm for checking whether a given connected, $G(V, E)$ undirected graph has an odd length cycle. The algorithm is a simple modification of DFS and it is called as ODDCYCLE-VISIT$(G, 1, +1)$. Note that instead of assigning DFS numbers to vertices, it assigns a label of $+1$ or -1 to vertices.

```
ODDCycle-Visit($G$, $u$, $b$)
Comment: DFS in graph $G$ from vertex $u$
1   $color[u] \leftarrow gray$
2   $label[u] \leftarrow b$
3   for each vertex $v \in Adj[u]$ do
4       if $color[v] = white$ then
5           $\pi[v] \leftarrow u$
6           ODDCycle-Visit($G$, $v$, $-b$)
7       else if $label[u] = label[v]$ then
8           Print "Odd Cycle Exists"; Stop
9   $color[u] \leftarrow$ BLACK
```

Now analyze the time complexity of the above algorithm. Then prove that it is correct by proving the following four claims (Convince yourself that the following four claims are enough to prove correctness.):

Claim 1: If $e = (u, v)$ is a **tree edge** of the DFS tree, then $label[u] \neq label[v]$.

Claim 2: If the above algorithm encounters an edge $e = (u, v)$ with $label[u] = label[v]$, then e is a **back edge** of the DFS tree, and this edge along with the unique path in the tree from u to v forms an odd cycle.

Claim 3: If there exists an edge $e = (u, v)$ with $label[u] = label[v]$, then the algorithm will find it.

Claim 4: If there exists an odd cycle in G, then there must be two adjacent vertices with the same label, i.e., there must be an edge $e = (u, v)$ with $label[u] = label[v]$.

28. The adjacency list representation consists of n lists, one for each vertex. This is usually implemented as a linked list of "edge" records. Sorting one of these n lists can be done in $O(n)$ time using bucket sort (with n buckets). So it is trivial to sort all the n lists in $O(n^2)$ time. Assuming the adjacency list representation, design a linear-time $(O(m + n))$ algorithm to sort each of the n adjacency lists of a given undirected graph $G(V, E)$. **Hint:** Use radix sort with n buckets.

29. Just as in the case of an undirected graph, the adjacency list representation for a directed graph also consists of n linked lists of "edge" records. For each directed edge of the form $e = (v_i, v_j)$, which is directed from v_i to v_j, the list $Adj[i]$ contains one

edge record containing the source index i and the destination index j. For each edge $e = (v_i, v_j)$, we refer to the edge $e' = (v_j, v_i)$ as its *partner* edge, if it exists. In general, for a given edge e, it takes $O(n)$ time to locate its partner edge (or even to know if it exists), unless we can store in each edge record a pointer to the record of its partner edge. Note that storing the index in the record is not sufficient to locate the partner edge. Design a linear-time ($O(m + n)$) algorithm so that each edge record contains a pointer to its partner edge (make it `NULL` if it does not exist). Assume that such a field already exists in each edge record (called `PartnerEdge`), initialized to `NULL`. **Hint:** It helps to use the algorithm from the previous problem and sort each of the adjacency lists first.

30. Given a weighted undirected graph G with non-negative edge weights, if the edge weights are all increased by a positive additive constant, can the minimum spanning tree change? Can the output of Dijkstra's algorithm change for some (fixed) start vertex s? What if it is decreased by a positive constant? What if the edge weights are all multiplied by a positive constant? Give (very) simple examples, if you claim that they can change.

31. (**Extra Credit**) Design a $O(\log n)$-time algorithm for the problems from the midterm exam. Remember that the problem was to design an efficient algorithm that takes as input the two order-statistic trees T_A and T_B (for two sets A and B) and computes the median of the elements in the set $A \cup B$. You may assume that there are no common elements in the two sets.