SPRING 2003: **COT 6405** ANALYSIS OF ALGORITHMS
[HOMEWORK 3; DUE MAR 27 IN CLASS]

Signed Statement: For every homework, add a statement to your homework that states that you have read the submission guidelines and cheating policies outlined on the course web page and that you have followed them for the homework you are submitting.

Problems

21. (**Regular**) 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 $label[u] = label[v]$.

22. (**Regular**) 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 simple undirected graph $G(V, E)$. **Hint:** Use radix sort with n buckets.

23. (**Regular**) In an undirected graph, the adjacency list representation consists of n linked lists of "edge" objects. For each undirected edge of the form $e = \{v_i, v_j\}$, the adjacency list representation contains two edge objects. One edge object is in the list $Adj[i]$ containing the source index i and the destination index j, while the other edge object is in the list $Adj[j]$ containing the source index j and the destination index i. Since an undirected edge can be thought of as being composed of two "directed" edges, we could think of one edge object as representing the edge $e = (v_i, v_j)$, with the other referring to the edge $e' = (v_j, v_i)$. We also say that the two "directed" edges $e = (v_i, v_j)$ and $e' = (v_j, v_i)$ are *partners* of each other. In general, for a given edge e, it takes $O(n)$ time to locate its partner edge object, unless we can store in each edge object a pointer to its partner edge object. Note that storing the index of a vertex in the edge object is not sufficient to locate the partner edge object. Design a linear-time ($O(m + n)$) algorithm so that each edge object contains a pointer to its partner edge object. 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.