

**CMSC 351**  
**Solutions to Homework 8**

- (1) (Problem 22.1-1, page 530) To compute the out-degree of each vertex, we traverse the adjacency list for each vertex and count the length of its list. This takes  $O(E + V)$  time.

For in-degree calculation, we maintain an array  $A[1..n]$  to keep a count of the in-degrees where  $n$  is the number of vertices.

1.  $\forall j, 1 \leq j \leq n, A[j] \leftarrow 0;$
2. For each vertex  $i \in V(G)$
3.     For each edge  $(i, j) \in Adj(i)$
4.          $A[j] \leftarrow A[j] + 1.$

This algorithm also takes  $O(E + V)$  time, but needs additional space to store the array.

- (2) (Problem 22.1-3 (page 530))

- (a) Let  $Adj$  denote the adjacency list of the directed graph  $G = (V, E)$  and  $Adj^T$  denote the adjacency list for  $G^T$  which is to be computed.

```

for each  $v \in V$  do
     $Adj^T[v] := \text{nil};$     (* Initialization step *)
for each  $v \in V$  do
    for each  $u \in Adj[v]$  do
        Insert( $Adj^T[u], v$ )    (* Insert  $v$  in the list for  $u$  *)

```

The running time of the algorithm is clearly  $O(|V| + |E|)$ .

- (b) Let  $A$  denote the adjacency matrix of directed graph  $G = (V, E)$  and  $Adj^T$  denote the adjacency matrix for  $G^T$  which is to be computed.

```

for  $i = 1$  to  $|V|$  do
    for  $j = 1$  to  $|V|$  do
         $A^T[j, i] := A[i, j]$ 

```

The running time of the algorithm is clearly  $O(|V|^2)$ .

- (3) (Problem 22.1-5 (page 530))

- (a) **adjacency list**

1.  $E(G^2) = \emptyset, V(G^2) = V(G)$
2. for each vertex  $u \in V(G)$
3.     for each vertex  $v \in Adj(u)$
4.         for each vertex  $w \in Adj(v)$
5.             if  $(u, w) \notin E(G^2)$  then  $E(G^2) = E(G^2) \cup (u, w)$

Time Analysis: To check the membership of an edge in  $E(G^2)$ , we maintain an adjacency matrix for  $G^2$ , so that line 5 takes only constant time. The total time is  $\sum_{u \in V(G)} \sum_{v \in \text{Adj}(u)} \text{outdegree}(v) = \sum_{u \in V(G)} O(E) = O(EV)$ . NOTE that according to the definition of the  $A^2$  in the problem, if  $(u, v), (v, u) \in E(G)$ , then  $(u, u), (v, v) \in E(G^2)$  (we assume that self-loops are allowed in the output).

(b) **adjacency matrix**

Let  $A$  be the matrix representing  $G$ . If we compute  $A^2$  (square of matrix  $A$ ), and  $A^2[u, w] > 0$ , then there exists at least one vertex  $v$  such that  $A[u, v] = 1, A[v, w] = 1$ , therefore  $(u, w)$  is in  $E(G^2)$ . On the other hand, if  $A^2[u, w] = 0$ , no vertex  $v$  exists such that  $A[u, v] = 1, A[v, w] = 1$ , hence  $(u, w) \notin E(G^2)$ . After we get  $A^2$ , we need to convert to all non-zero elements in  $A^2$  to 1, which will take  $O(V^2)$  time. The time for standard matrix multiplication is  $O(V^3)$  (one could also use fast matrix multiplication algorithms; more on this will be said later). So the total time is  $O(V^3)$ . (We could also convert the matrix representation to a list representation, and use the solution from part (a).)

(4) (Problem 22.2-3 (page 552)) The running time will be  $O(|V|^2)$  since when we scan the neighbors of each vertex, this will take  $O(|V|)$  steps.

(5) (Problem 22.4-3 (page 552)) We give one algorithm to detect and find a cycle (if one exists) in  $O(|V|)$  time in an undirected graph  $G$ . If the number of edges in an undirected graph exceeds  $|V| - 1$  then the the graph must have a cycle. If the number of edges is at most  $|V| - 1$  then any  $O(|E| + |V|)$  algorithm is an  $O(|V|)$  algorithm.

The algorithm essentially does DFS. If we ever encounter an edge that is not a tree edge, then it must be a back edge and we have found a cycle. The algorithm will find a back edge if one exists.

```

proc DFS-Visit(u);
    color[u] ← GRAY;
    count ← count+1;
    d[u] ← count;
    for each v ∈ Adj[u] do
        if color[v]=WHITE then
            parent[v] ← u;
            Add edge (u,v) to STACK;
            DFS-Visit(v);
            Pop(STACK);
        endif
    else if (not(parent[u]=v) and (d[v] < d[u])) then
        (* (u,v) is a back edge from u to its ancestor v *)
        Found cycle! Exit. If we pop edges from the stack
        until we find an edge incident to v, we have the cycle
    endif;
end for;
end proc;

```

