1. (Exercise 22.1-6 from the text) Let $A = [a_{ij}]$ be the adjacency matrix of a directed graph $G = (V, E)$ with no self-loops. Construct an algorithm with asymptotic time complexity $\Theta(V)$ that finds a vertex $v$ with properties

(a) out-degree of $v$ is zero
(b) in-degree of $v$ is $|V| - 1$.

If there is no vertex with these properties, the algorithm should output a statement to that effect; otherwise it outputs the vertex number. Your writeup should include

(a) Pseudocode for the algorithm in the spirit of the examples covered in lecture.
(b) An explanation as to why at most one vertex can have desired properties.
(c) An argument that shows your algorithm is $\Theta(V)$.

Assume that the adjacency matrix is already in memory when your algorithm starts. Otherwise, there is no way to achieve $\Theta(V)$ since it could require $\Theta(V^2)$ operations just to read in the matrix.

2. (Exercise 22.1-7 from the text) The incidence matrix of a directed graph, $G = (V, E)$, with no self-loops is a $V \times E$ matrix $B = [b_{ij}]$ such that

$$b_{ij} = \begin{cases} -1, & \text{if edge } j \text{ leaves vertex } i \\ 1, & \text{if edge } j \text{ enters vertex } i \\ 0, & \text{otherwise.} \end{cases}$$

Let $B^T$ denote the transpose of $B$. That is, $[B^T]_{ij} = b_{ji}$. Let $C = [c_{ij}] = BB^T$. What is the meaning of each $c_{ij}$, in terms of edges in the original $G$.

3. (Variation on Exercise 22.2-8 of the text) Consider a tree rooted at vertex $r$ as an undirected graph $G = (V, E)$. For any two vertices $u, v \in V$, let $\delta(u, v)$ denote the shortest-path distance from $u$ to $v$ (number of links). Define

$$D = \max_{u, v \in V} \delta(u, v).$$

Starting with an adjacency list representation of $G$ as a directed graph, construct a $\Theta(V)$ algorithm that computes $D$. Note that the input adjacency list will contain only edges from a vertex to its children in the tree; it will not contain the reverse edges, although these reverse edges are used in computing the $\delta(u, v)$ values. Give an argument to show that your algorithm is $\Theta(V)$. As with the first problem, assume that the adjacency list is already in memory when your algorithm starts. Otherwise, the result may not be $\Theta(V)$, regardless of the efficiency of your computation, since it requires $\Theta(V + E)$ operations just to read in the adjacency list, and that may be $\omega(V)$ if $E = \omega(V)$. 