Algorithms I

Tutorial 6 Solution Hints

October 21, 2016

Problem 1

Both can be calculated in $O(E)$ time total. The out degree of a node is just the size of its adjacency list. For in degree, we can maintain an array initialized with 0. We iterate over the adjacency list of all the nodes, whenever we encounter an edge (u, v), we increment $in - deg(v)$ by 1.

Problem 2

The adjacency matrix for G' is the transpose of the adjacency matrix for G which can be calculated in $O(V^2)$ time. The adjacency list representation of G' can be obtained from the adjacency list representation of G by iterating over the vertices and adding u to the adjacency list of v whenever we encounter an edge (u, v).

Problem 3

This problem can be reduced to the problem $4(a)$ from the mid semester examination.

Problem 4

There was a mistake in the tutorial. It should have been 23.4-3.
Let us two cases:

- $|E| < |V|$: We just do a dfs and check for a cycle. This takes $O(V + E) = O(V)$ time
- $|E| \geq |V|$: The graph is guaranteed to have a cycle. This is because a tree is a maximal acyclic graph i.e. if we add any extra edge to a tree, it will no longer be acyclic. A tree has $|V| - 1$ edges. Hence, any acyclic graph on V can not have more than $|V| - 1$ edges.

Problem 5

The first thing to observe is that the cities form a tree. The greatest distance between any pair of vertices in a graph is known as *diameter*. The simplest algorithm to find diameter is to do a BFS starting from each node and find the distance of the farthest node. It takes $O(V \cdot (V + E)$ time using adjacency list representation.

However, there is an algorithm with better complexity. Start BFS from any node say u. Let v be a farthest node from u. Now, do a BFS from v. Let w be a farthest node from v. The distance between v and w is the diameter. Note that this algorithm is only for tree and not for general graphs.

Problem 6

Let $u_1, u_2, \dots u_N$ be a topological order of the vertices where $N = |V|$. Let us define $dp[i]$ as the length of the longest path starting at i. Now, $dp[i] = max(1, dp[j] + 1 \forall (i, j) \in E)$. If we start at a vertex i, either we finish the path at i or take one of its neighbour j as the next vertex in the path. The maximum over $dp[i]$ is the length of the longest path.

Problem 7

The simplest algorithm is to check for each vertex individually. We remove a vertex and all the edges incident on it. If the graph is disconnected, the vertex is an articulation point otherwise not. There is also a linear time algorithm by John Hopcroft and Robert Tarjan which is based on DFS.

Problem 8

Let us define a graph $G_S(V, E)$ such that $V = x_1, x_2, \dots x_N$ and $E = \{(x_i, x_j)$ iff $(x_i < x_j) \in S\}$ where $S \subseteq M$. A subset S is inconsistent if ad only if there is a cycle in G_S. Also if S is inconsistent then so is M. So, we just need to check whether there is a cycle in G_M.