Algorithm Design and Analysis
CSE222 Winter 2022

Tutorial 7

Problem 1 Given an undirected graph $G = (V, E)$, give an $O(|V|)$-time algorithm to check whether G has a cycle. The algorithm must run in $O(|V|)$-time, that is independent of $|E|$.

Solution (sketch): We shall be using the crucial property that a cycle exists in a an undirected graph if and only if there is a back edge between some pair of vertices in any dfs tree. The proof of this has two parts. Suppose there is a back edge between u and v. Then, the path from u to v in the dfs tree along with this back edge clearly forms a cycle. Now suppose there is a cycle which is reachable from the root vertex in the graph. Then one can demonstrate a back-edge. We are not going in to the formal proof since it has a lot of detail. Informally, consider a cycle C and let v_k be the node with the latest arrival time. Then it should be intuitively clear that one of its two adjacent edges inside the cycle C has to be a back edge. Again, this is not formal but just intuitive.

Now the algorithm is very simple. Suppose for some vertex v, inside the recursive call $DFS(v)$, there is some vertex in the adjacency list (the for loop) w such that w is already visited. Then this is clearly a back edge.

So why does it take $O(V)$ time ? If there is no cycle, then clearly there are only $V - 1$ edges and there is nothing left to prove.

In the other case, consider the total number of pointer movements happening inside the for loop that traverses the adjacency. Only $V - 1$ movements of the pointers can take you to a new vertex. After at most these, many pointer movements, you would definitely find a vertex inside some recursive call which is already visited. Hence, you will encounter a back edge. All other operations can be charged essentially to these pointer movements.

Note to TAs: The solutions for both Q2 and Q3 an be found in this link

Problem 2 A vertex in an undirected graph is a cut vertex if its removal (along with any incident edges) causes the graph to become disconnected. In this problem we will (at a high level) adapt the algorithm for computing bridges to compute cut vertices, by describing the conditions under which a vertex is a cut vertex. Let $G = (V; E)$ be a connected undirected graph, and suppose that you run DFS on G and (as in Lecture) you compute the value of $arr[u]$ for every vertex $u \in V$. Prove each of the following assertions:

(**Note to TAs:** Do not tell them the following upfront. Rather ask them to come up with what properties are required to design the algorithm. They should be able to come up with all these three things)

(a) The root of the DFS tree is a cut vertex if and only if it has two or more children.

(b) No leaf of the DFS tree can be a cut vertex.

(c) A non-root, internal vertex u of the DFS tree is a cut vertex if and only if it has a child v such that there is *no back edge from the subtree rooted at* v to a proper ancestor of u.

Problem 3 Present an efficient algorithm that, given a connected, undirected graph $G = (V, E)$, deter- mines whether it is possible to assign directions to the edges so that the resulting digraph has no sources. A vertex of a digraph is a source if it has no incoming edges. In other words, every vertex of your digraph should have at least one incoming edge. If this is not possible, your algorithm should indicate that this is so. Prove your algorithm's correctness and derive its running time. (can be done in linear time).