Algorithm Design and Analysis
CSE222 Winter '20

Tutorial 7

Problem 1 Given a directed graph $G(V, E)$, the $\text{Rev}(G) = G_r(V, E_r)$ is defined as follows. Every $(u, v) \in E$ if and only if $(v, u) \in E_r$. Informally, the G_r is defined as the directed graph G_r such that all the edges of G are in the opposite direction in G_r Design a linear time, i.e., $O(V + E)$ times algorithm to compute G_r. The algorithm should be able to output the adjacency list of of G_r.

Note to TAs: This is a warm-up exercise to get familiar with adjacency list data structure.

Solution. Assume that G is represented in adjacency list format and $Adj[u]$ is the linked list containing all the edges going out of u in G. Initialize an adjacency list $Adj_r[u] =$ for all $u \in V(G)$. For all $u \in V$, if $v \in Adj[u]$, then insert u at $Adj_r[v]$.

```
for all u ∈ V do
 | Initialize Adj_r[u] ← ∅;
end
for all u ∈ V do
 | for all v ∈ Adj[u] do
 |  | Insert u into Adj_r(v);
 | end
end
Output the vertices V with Adj_r[v] for all v ∈ V;
```

Note to TAs: The solutions for both the following can 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 $start[u]$ (i.e. the starting timestamp of 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 (x, y) such that $x = v$ or x is a descendant of v but y is an ancestor of u.

(d) Let $LOW(v)$ is defined as follows.

$$= \min\{start(v), start(w) : (u, w) \text{ is a back edge for some descendant } u \text{ of } v\}$$

Show how you will use the ideas of DFS to show how to compute $LOW(v)$ for all $v \in V(G)$ in $O(|V| + |E|)$-time.

Solution. Here is a sketch of the solution. Observe that, if v is a cut vertex, then following the definition in (d) and condition in (c), it must satisfy the condition $LOW(v) < start(v)$. Once we know $LOW(v)$ for each of $v \in V(G)$, one can easily identify all the cut vertices by a linear scan through the vertex set. So the complete problem boils down to computing $LOW(v)$ with the complexity constraint mentioned in (d).

For the algorithm, we maintain an extra variable say $t(v)$ for each vertex v. We compute the BFS tree and simultaneously compute $LOW(v)$, so no pre-assumption is needed. We design the algorithm based on the following three strategies.

While doing DFS,

a) If a new vertex u is discovered, set
   $Low(u) = t(u) = start(u)$

b) If a back edge uv (u is descendant of v) is discovered, set
   $t(u) = \min[t(u), start(v)]$

c) During backtracking on the DFS tree, for an edge uv such that u is the parent of v, set
   $Low(u) = \min[t(v), Low(u)]$

Now we provide the modified DFS algorithm,

ModifiedDFS(vertex v)

if *v is visited* **then**
  return $start(v)$
end
visit(v), $LOW(v) \leftarrow start(v)$, $t(v) \leftarrow start(v)$;
for *each u in neighbours of V* **do**
  if *u is not visited* **then**
    $t_u \leftarrow$ *ModifiedDFS(vertex u)*
    $Low(v) \leftarrow \min[t_u, Low(u)]$
  end
  if *u is visited* **then**
    $start_u \leftarrow$ *ModifiedDFS(vertex u)*
    $t(v) \leftarrow \min[t(v), start_u]$
  end
end
return $t(v)$

The running time holds due to the running time of DFS.