# Practice HW

1. Let $G = (V, E)$ be a directed graph. Define a graph $G^2 = (V', E')$ as follows: $V' = V$; $\langle u, v\rangle \in E'$ if there is a a path of length 2 between $u$ and $v$ in $G$. Suppose that a directed graph $G$ is given as adjacency matrix. Given an algorithm to compute $G^2$. Derive the run time of your algorithm.

   *Ans.* Let $M$ be the adjacency matrix of $G$. Compute the adjacency matrix $N$ of $G^2$ as follows: To compute $N[u, v]$ do the following: Find a $w$ such that both $N[u, w]$ and $N[w, v]$ are 1. More formally;

   - Procedure **Compute (u, v)**
   - For $i$ in the range $\{1, 2, \cdots, n\}$
     - If $M(u, i) == 1$ and $M(i, v) == 1$, then $N(u, v) = 1$ and quit loop.

   Note that above algorithm takes $O(n)$ time. Now the algorithm to compute $G^2$ is

   - Input $G$
   - Initialize a 2-D array $N$ (of size $n \times n$) with all zeros.
   - For every $u, v \in V$, $\mathbf{Compute(u, v)}$.

   The loop is performed $n^2$ times and each iteration of the loop takes $O(n)$ time and thus the time taken is $O(n^3)$.

   *Correctness:* Suppose there is a path of length 2 from $u$ to $v$. Thus there is a vertex $w$ such that $\langle u, w\rangle \in E$ and $\langle w, v\rangle \in E$. Thus $M(u, w) = 1$ and $M(w, v) = 1$. When we call that procedure $Compute(u, v)$, when the value of $i$ equals $w$, both conditions $M(u, i) == 1$ and $M(i, v) == 1$ are satisfied and this $N(u, v)$ is set to 1.

2. Let $G = (V, E)$ be a directed graph where $V = \{1, 2, \cdots, n\}$ such that $n$ is odd; I.e n $= 2k+1$ for some $k > 0$. Given a vertex $v$, let $TO_v$ be the set of all vertices from which there is path to $v$. Let $FROM_v$ be the set all vertices for which there is a path from $v$. I.e,

   $$TO_v = \{u \mid \text{ There is a path from } u \text{ to } v\},$$

   $$FROM_v = \{w | \text{ There is a path from } v \text{ to } w\}.$$

   A vertex $v$ is *center vertex* of $G$ if all of the following conditions hold:

   - $|TO_v| = |FROM_v| = k$. I.e, both $TO_v$ and $FROM_v$ have exactly $k$ vertices.
   - $TO_v \cap FROM_v = \emptyset$. I.e, $TO_v$ and $FROM_v$ are disjoint.

Give an algorithm that gets a graph $G$ (with odd number of vertices) as input and determines if the graph has a center vertex or not. If the graph has a center vertex, then the algorithm must output it. Describe your algorithm, prove the correctness, and derive the time bound. Your grade partly depends on the efficiency of your algorithm.

*Ans.* We know the following about DFS and finish times (from lectures). If there is a path from $u$ to $v$ and there is no path from $v$ to $u$, then $FinishTime(u) > FinishTime(v)$. Suppose that center vertex $v$ exists. Then FinishTime of $v$ is greater than FinishTime of every vertex in $FROM_v$. Similarly, FinishTime of every vertex in $TO_v$ is greater than FinishTime of $v$. Thus if center vertex exists, then its finish time must be $k+1$. This suggests the following algorithm: Perform a DFS let $v$ be a vertex whose finish time is $k+1$. Now, perform a $DFS(G, v)$ and store all vertices visited, let this set be $A$. Now, reverse the direction of every edge in $G$, and perform $DFS(G^r, v)$, where $G^r$ is the reverse graph. Let $B$ be the set of all vertices visited. If both $A$ and $B$ are of size $k$ and are disjoint, then $v$ is the center vertex. Otherwise, there is no center vertex in the graph.

We can perform DFS in $O(m+n)$ time. Checking whether $A$ and $B$ are disjoint can be performed in $O(n)$ time, since $V = \{1, 2, \cdots, n\}$, by storing $A$ and $B$ as bit arrays. Thus the total time is $O(m+n)$.

3. Let $G = (V, E)$ be an undirected, connected graph. A vertex $v \in V$ is *bridge vertex* if removal of $v$ (and edges incident on $v$) makes the graph disconnected.

    - Suppose that $v \in V$ be a bridge vertex. Is there a vertex $u \in V$ such that if we perform DFS on $G$ starting at $u$, the vertex $v$ will be a leaf node in the resulting DFS tree?

      *Ans.* No. Consider DFS tree $T$ formed by doing DFS starting at a node $u$. Let $v$ be a leaf node. Since the graph is connected, ether is a path from every node to every other node in $T$. Pick any two vertices $a$ and $b$ such that neither is $v$. Consider a simple path from $a$ to $b$ in $T$. Let the path be $a, v_1, v_2, \cdots, v_\ell$. Note that there are tree edges from $a$ to $v_1$, $v_i$ to $v_{i+1}$ and from $v_\ell$ to $b$. Suppose that $v$ is $v_i$ for some $i$. Thus $v_{i-1}$ to $v_i$ is a tree edge, however $v_i$ to $v_{i+1}$ can not be a tree edge, as $v_i$ is a leaf node and the path is a simple path. Thus removal of $v$ will still preserve path from $a$ to $b$ in $G$. Thus $v$ can not be a bridge vertex.

    - Given an $O(m+n)$-time algorithm that gets a graph $G$ (which is undirected and connected) as input and outputs a vertex $v$ that is not a bridge vertex. Describe your algorithm, derive the time bound, prove the correctness of your algorithm.

      *Ans.* Perform DFS and output a leaf node of the DFS tree. Time is $O(m+n)$. Correctness follows from Part a.

4. Consider the following DFS algorithm on a directed graph.

    **Algo DFS**

    (a) Input $G = (V, E)$.

    (b) counter = 0;

    (c) for every $u \in V$, $start[u] = 0$, $finish[u] = 0$.

    (d) Unmark every vertex $u$ in $V$.

    (e) For $u \in V$

- If $u$ is unmarked, $DFS(G, u)$.

**Procedure** $DFS(G, u)$

(a) start[u] = counter;

(b) counter++;

(c) For every $v$ such that $\langle e, v\rangle \in E$

- If $v$ is unmarked, $DFS(G, v)$.

(d) finish[u] = counter;

(e) counter++;

The above algorithm can be easily modified to construct DFS forest $T$. Show that for every $u \in V$ the number of descendants of $u$ in $T$ equals $\lfloor \frac{finish[u] - start[u]}{2} \rfloor$.

*Ans.* We prove by the induction on the structure DFS Tree/Forest. Give $u$, net $C(u)$ denote the number of children of $u$. We will prove that for every $u$, $finish(u) - start(u) = 2C(u) + 1$. As base case consider the leaf nodes. Node that for every leaf node $u$, $finish(u) = start(u)+1$. Since leaf node does not have any children, the claim is true nodes.

Let $v$ be an internal node in the DFS Tree/Forest. Let $u_1, u_2, \cdots, u_\ell$ are its children. As induction hypothesis assume that $finish(u_i) - start(u_i) = 2C(u_i) + 1$, for $1 \le i \le \ell$. Note that $start(v) = start(u_1) + 1$ and $start(u_{i+1}) = finish(u_i) + 1$ $(1 \le i < \ell)$, and $finish(v) = finish(u_\ell) + 1$. Thus

$$\begin{aligned}
finish(v) - start(v) &= finish(u_\ell) - start(u_1) + 2 \\
&= finish(u_\ell) - start(u_\ell) + start(u_\ell) - start(u_1) + 2 \\
&= (finish(u_\ell) - start(u_\ell)) + finish(u_{ell-1}) - start(u_1) + 3 \\
&= \cdots \\
&= \cdots \\
&= [finish(u_\ell) - start(u_\ell)] + [finish(u_{\ell-1}) - start(u_{\ell-1})] + \cdots + [finish(u_1) - start(u_1)] + \ell + 1 \\
&= 2C(u_1) + 2C(u_2) + \cdots + 2C(u_\ell) + 2\ell + 1 (\text{ By induction Hypothesis}) \\
&= 2C(v) + 1
\end{aligned}$$

5. Draw a directed graph of your choice and identify strongly connected components in the graph.

6. Draw a directed graph of your choice and compute finish times of all vertices (when you perform DFS)