Solutions of Assignment #1 part 2
(Course: CS 401)

Problem 1 (35 points): Let $G = (V, E)$ be a **directed** graph and let s and t be two nodes of G. Let n and m be the number of nodes and edges of G, respectively. In the class we say how to decide if there is a path **from** s **to** t, namely, we start a (directed) BFS starting from s and check if t appears among the list of nodes that are visited during BFS. The purpose of the assignment is to decide if such a path exists under some **additional constraints**. Let u and v be two other nodes of G that are **not** s or t.

(*i*) [15 points] Decide if G has a path from s and t that **avoids** using **both** the nodes u and v.

(*ii*) [20 points] Decide if G has a path from s and t that **uses both** the nodes u and v.

Note that your algorithm need only to "decide" about the path, *i. e.*, say "yes" if that kind of path existed and "no" otherwise; it is not necessary for your algorithm to actually provide the path. Both of your algorithms should run in $O(m + n)$ time. You may assume that the graph is given in its adjacency list representation. If you are using BFS, there is no need to give codes for it; simply saying "do a BFS starting at such-and-such node" will suffice.

Solution:

(*i*) We delete all the edges in-coming to nodes u and v, *i. e.*, we delete all the edges of the form (x, u) or (x, v) where $x \in V - \{u, v\}$. Then we simply run a BFS starting at s and check if t can be reached.

To delete all the edges as mentioned above, we go through the adjacency list of every node except u and v. For each such list, we traverse the list keeping two pointers, pointer p_1 one at the current entry and pointer p_2 at the entry before the current entry. If the current entry is u or v, we change the link of entry to p_2 to skip u or v. Pictorially, it looks like as shown below for the adjacency list of node a:

$$
\begin{array}{lccccc}
 & p_2 & p_1 & & \\
 & \downarrow & \downarrow & & \\
a \cdots\cdots \to & b \to & c \to & u \to & d \cdots\cdots
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{lccccc}
 & & p_2 & p_1 & \\
 & & \downarrow & \downarrow & \\
a \cdots\cdots \to & b \to & c \to & u \to & d \cdots\cdots
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{lccccc}
a \cdots\cdots \to & b \to & c & u & d \cdots\cdots
\end{array}
$$

(*ii*) We will use the notation $x \rightsquigarrow y$ to indicated a directed path from node x to node y. A path from s to t that uses both nodes u and v must be one of the following two types depending on whether node u is before or after node v:

(A) $s \rightsquigarrow u \rightsquigarrow v \rightsquigarrow t$

(B) $s \rightsquigarrow v \rightsquigarrow u \rightsquigarrow t$

For (A), we can use a BFS starting at s to find a path $s \rightsquigarrow u$, a BFS starting at u to find a path $u \rightsquigarrow v$, and a BFS starting at v to find a path $v \rightsquigarrow t$. (B) can be handled in a similar manner.

Problem 2 (15 points): Give an algorithm to detect whether a given undirected graph is a tree or not. The graph is given to you in its adjacency list representation. The running time of your algorithm should be $O(m + n)$ for a graph with n nodes and m edges.

Solution: Let G be the given graph. We run BFS starting from an arbitrary node s. If BFS cannot reach all nodes then the graph is not connected and hence not a tree. Otherwise, consider the obtained BFS tree T. If every edge of G appears in the BFS tree then $G = T$, hence G contains no cycle and therefore G is a tree. Otherwise, G is a non a tree by the following argument. There is some edge $e = \{v, w\}$ that belongs to G but not to T and thus G has strictly more than $n - 1$ edges.