**Due on February 27th, 2017**
**60 points total**

**General Directions:** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time (for this assignment, this means arguing why your algorithm achieves the target running time specified by the question). There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice.

All the answers must be typed, preferably using LaTeX. If you are unfamiliar with LaTeX, you are strongly encouraged to learn it. However, answers typed in other text processing software and properly converted to a pdf file will also be accepted. Before submitting the pdf file, please make sure that it can be opened using any standard pdf reader (such as Acrobat Reader) and your entire answer is readable. **Handwritten answers or pdf files that cannot be opened will not be graded and will not receive any credit.**

Finally, please read the detailed collaboration policy given on the course website. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

**Problem 1 (10 points)**
The following are short questions regarding depth-first search (DFS) and breadth-first search (BFS).

(a) (5 points) Let $G = (V, E)$ be an undirected graph. Describe an algorithm that uses BFS or DFS to find the number of connected components in $G$ in $O(n + m)$ time (where $|V| = n$ and $|E| = m$).

(b) (5 points) Now let $G$ be a directed graph. In class, we saw an algorithm that uses the information obtained from a DFS to determine the strongly connected components of $G$. Make an argument for why using instead BFS will not work. Namely, focus on why the order in which we visit vertices in a BFS does not give us any information about the strongly connected component structure in $G$ (note a BFS does not label vertices with pre and post values, so in your argument, just work with the order in which vertices are dequeued from the BFS queue).

**Problem 2 (20 points)**
Let $G = (V, E)$ be an undirected graph where $|V| = n$ and $|E| = m$. Suppose for two vertices $u, v \in G$, we want to know if $u$ and $v$ are reachable from one another, i.e., if there exists a path in $G$ between $u$ and $v$.

(a) (5 points) It should be clear that we could use either DFS or BFS to answer this question; however, what is the worst-case *space complexity* of each of these approaches? Specifically, assume that each vertex is represented using $\Theta(\log n)$ bits. How many bits will you need to store in each algorithm in the worst case? Express your answer asymptotically (i.e., using $\Theta(\cdot)$ notation).

(b) (15 points) Let us try to develop a more space-efficient approach. Consider the procedure EXISTS-PATH$(a, b, k)$, which takes as parameters vertices $a, b \in V$, and returns **true** if there exists a path in $G$ connecting $a$ and $b$ that has at most $k$ edges; otherwise, it returns **false**. We can implement this function as follows:

$$\begin{aligned}
\text{For } k > 1, \text{EXISTS-PATH}(a,b,k) &= \bigvee_{w \in V} (\text{EXISTS-PATH}(a,w,\lceil k/2 \rceil) \wedge \text{EXISTS-PATH}(w,b,\lfloor k/2 \rfloor)) \\
\text{For } k = 1, \text{EXISTS-PATH}(a,b,k) &= \textbf{true} \text{ if and only if } (a,b) \in E \text{ or } a = b; \text{otherwise } \textbf{false}.
\end{aligned}$$

In other words, EXISTS-PATH$(a, b, k)$ returns **true** if and only if there exists some vertex $w \in V$ such that there is a path from $a$ to $w$ that uses at most $\lceil k/2 \rceil$ edges, and a path from $w$ to $b$ that uses at most $\lfloor k/2 \rfloor$ edges. Now, to answer the reachability question for a particular pair of vertices $u, v \in V$, we simply return the result of EXISTS-PATH$(u, v, n - 1)$.

First argue that this algorithm is correct. Next, bound the number of bits the algorithm needs to store in the worst case – give your answer in asymptotic notation. (As before, assume that vertices can be represented using $\Theta(\log n)$ bits.)

*Note: The running time of this algorithm can be much worse than that of DFS or BFS.*

**Problem 3 (20 points)**
You have been hired by the university to manage the wireless network across campus. The network is set up as follows: there is one modem and many routers, which need to be connected together by cables. A router can only function if there is some path of cables (possibly through other routers) that connects it to the modem. Your job is to establish this network and keep all the routers functioning properly. There are many cables already in place between the routers, but they are old and malfunctioning. It is not within your budget to replace them with new cables, and so you will need to manage this network by repairing faulty cables. (Assume that it is possible to connect the whole network with these faulty cables, and that all the cables can carry data in both directions.)

You think back to what you learned in 330, and remember that you can use DFS to find a tree that connects every router to the modem. So, you run DFS from the modem (faulty cables are edges and the routers/modem are vertices) and repair all the tree edges of the DFS tree. Good job! We will denote this tree of now functioning cables as $T$.

However, since these cables are old and prone to failure, one of them might stop working at any time! Therefore, the university wants to know how many malfunctioning cables can potentially replace a repaired cable in case it permanently fails. To formalize this, let $e$ be some repaired cable. If this cable were removed, $T$ would be split into two connected components. The modem would be in one of these components, and all the routers in the other component would no longer be connected to the modem! However, there may be other malfunctioning cables that connect these two components; therefore, repairing one of those cables will reconnect the network. The university wants to know how many such malfunctioning cables are there for each repaired cable.

Give an algorithm that computes all these counts (one for each edge in $T$) in $O(m)$ time, where $m$ is the total number of cables, both repaired and malfunctioning. You may assume that for a pair of routers/modem $u$ and $v$, you can determine if $u$ is an ancestor of $v$ in $T$ in $O(1)$ time.

Would your algorithm work if tree $T$ were *any* spanning tree of the network (instead of being a DFS tree)?

**Problem 4 (10 points)**
You are planning a second party (remember the first party?) for spring break. You have a total of $n$ friends, each of whom has given you two lists with a subset of these $n$ friends of yours. Each friend will attend your party if and only if everyone in their first list and no one in their second list is invited to your party. You quickly realize that you must throw multiple parties if you want to invite every friend. What is the most efficient algorithm you can come up with to decide if you can throw a set of parties such that each of your $n$ friends attends a party?