Note: The homework is due electronically on the course website on Friday, April 1 by 10:30 am, the beginning of the class. For late submissions, please refer to the Late Submission Policy on the course webpage: http://www.seas.upenn.edu/~cis121

You must use the hw121.cls template provided on the course website.

Please write concise and clear solutions; you will get only a partial credit for correct solutions that are either unnecessarily long or not clear.

You are allowed to discuss ideas for solving homework problems in groups of up to 3 people but you must write your solutions independently. Also, you must write on your homework the names of the people with whom you discussed.

Finally, you are not allowed to use any material outside of the class notes and the textbook. Any violation of this policy may seriously affect your grade in the class.

[15] 1. We have a connected graph $G = (V, E)$, and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at $u$, and obtain a tree $T$ that includes all nodes of $G$. Suppose we then compute a breadth-first search tree rooted at $u$, and obtain the same tree $T$. Prove that $G = T$. (In other words, if $T$ is both a depth-first search tree and a breadth-first search tree rooted at $u$, then $G$ cannot contain any edges that do not belong to $T$.)

[20] 2. In sociology, one often studies a graph $G$ in which nodes represent people ($n$ of them) and edges represent those who are friends with each other. Let’s assume for the purposes of this question that friendship is symmetric, so we can consider an undirected graph. Let’s also assume that each person has at least two friends, i.e., each node in $G$ has at least two neighbors.

Let $S$ be the set of people, each having three friends or more. Prove that there is a set of people $P = \{P_1, P_2, \ldots, P_k\}$, $k \leq n$, with the following properties:

a. For $1 \leq i < k$, $P_i$ is friends with $P_{i+1}$, and $P_k$ is friends with $P_1$. In other words the corresponding nodes form a cycle in $G$.

b. $|P \cap S| \leq 2\lceil \log n \rceil$.

You should give a linear time algorithm to find the set $P$. 
3. Consider the following problem: there are \( n \) cities in a road network and there are direct roads connecting some pairs of cities. All roads are one-way, but a pair of cities, say \( A \) and \( B \) may have two one-way roads between them, one going from \( A \) to \( B \) and the other going from \( B \) to \( A \). Let \( m \) be the total number of roads in this network. Call a city \( C \) well-connected, if every other city in the country is reachable from \( C \). Design an \( O(n+m) \)-time algorithm that takes as input a road network as described above and outputs the set of all well-connected cities in the network.

4. Consider the following problem: there are \( n \) cities and there are direct roads connecting some pairs of cities. All roads are one-way, but a pair of cities, say \( A \) and \( B \) may have two one-way roads between them, one going from \( A \) to \( B \) and the other going from \( B \) to \( A \). Associated with each road is a positive number that indicates the time it takes to go from one city to another (if there are two roads connecting a pair of cities, the time associated with each road may be different). This network of one-way roads is such that every city is reachable from every other city. There is a special city \( S \). Give an efficient algorithm to find the smallest amount of time to reach from every city to every other city with the constraint that all routes must pass through \( S \).

5. Let \( G = (V, E) \) be a connected undirected graph with positive edge weights. Assume that all edge weights are distinct. For any edge \( e \in E \), let \( T(e) \) denote the spanning tree of \( G \) that has minimum cost among all spanning trees of \( G \) that contain \( e \). Design an efficient algorithm that takes as input the graph \( G \) and an edge \( e \in E \) and outputs \( T(e) \). The running time of your algorithm should be \( O(m \log n) \).

6. In this problem, we consider implementing the Union-Find data structure using arrays – we have an array \( \text{Component} \) that maintains the name of the set containing each element. Let \( S \) be a set containing \( n \) elements denoted \( \{1, 2, \ldots, n\} \). The array \( \text{Component} \) has size \( n \), where \( \text{Component}[s] \) is the name of the set containing \( s \). To implement \( \text{MAKE-SET}(S) \), we set up the array and initialize it to \( \text{Component}[s] = s \), for each \( s \in S \). In this implementation, \( \text{FIND}(v) \) is a simple lookup and takes \( O(1) \) time. However, \( \text{UNION}(x, y) \) can take as long as \( O(n) \) time, as we have to update the values of all elements in the sets containing \( x \) and \( y \).

Use the \( \text{Component} \) array along with some optimizations (which you must describe clearly) to implement Kruskal’s algorithm on a graph \( G \) with \( n \) vertices and \( m \) edges in \( O(m \log n) \) time.