Due on October 17th, 2014
100 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) (2.5 points) Let G be an undirected graph. Let T_D be a DFS tree on G; first argue that there can be no cross edges in G with respect to T_D. Next, let T_B be a BFS tree on G; now show that G can have cross edges with respect to T_B in this case.

(b) (2.5 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 $|E| = m$).

(c) (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 $v, u \in G$, we want to know if u and v are reachable from one another, i.e., if there exists a path in G between v and u.

(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. So for your answer, give the number of bits we will have to store in each algorithm in the worst case. Express your answer asymptotically (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:

$$\text{For } k > 1, \text{EXISTS-PATH}(a, b, k) = \bigvee_{w \in V} (\text{EXISTS-PATH}(a, w, \lceil k/2 \rceil) \wedge \text{EXISTS-PATH}(w, b, \lfloor k/2 \rfloor))$$

$$\text{For } k = 1, \text{EXISTS-PATH}(a, b, k) = \textbf{true} \text{ if and only if } (a, b) \in E \text{ or } a = b; \text{ return } \textbf{false} \text{ otherwise.}$$

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 also 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 indeed correct. Next, bound the number of bits the algorithm will ever need to store; give your answer asymptotically. Again, assume that vertices can be represented using $\Theta(\log n)$ bits.

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

Problem 3 (15 points)
After spending several exhausting weeks attempting to find locally highest points in Durham, our good friend Bob Bitfiddler is now planning a well-deserved vacation. While browsing online, he was fortunate enough to stumble upon a travel site offering an attractive international vacation package. For discounted flight fares, the deal works as follows: the travel site gives Bob a fixed sequence of n countries $S = C_1, C_2, \ldots, C_n$. First, Bob picks any country in the sequence where he wants to begin his trip. He then must proceed by flying country to country according to the sequence, but he can decide to travel back home whenever he likes. More formally, Bob determines his trip by picking some contiguous subsequence $S' = \langle C_s, C_{s+1}, \ldots, C_{e-1}, C_e \rangle$ of S, where $1 \leq s \leq e \leq n$.

Bob wants to be clever in selecting the countries that he will visit. He has an extra $\$100$ that he has decided he will not be spending during the vacation, but by using exchange rates, he hopes to turn a profit on this $\$100$. We will model exchange rates with the following assumptions:

1. Before Bob leaves for the trip, he will convert this $\$100$ to the native currency of the first country C_s that he will visit. This conversion is without change in value (i.e., he will have $\$100$ worth of currency in the first country he visits).

2. We define $\langle c_{1,2}, c_{2,3}, \ldots, c_{n-1,n} \rangle$ to be the $n-1$ *exchange factors* between the n countries in S, where each $c_{i,j} \in \mathbb{Q}^+$ gives the factor by which the value of Bob's assets change when he exchanges currency from country i to country j. (Note that $\mathbb{Q}^+$ denotes the set of positive rational numbers).

3. Once he returns home, he can convert the currency he has in the last country C_e that he visited back to USD without change in value.

Note that this is not how exchange rates are typically described. Here, each $c_{i,j}$ does *not* give the factor with which Bob multiplies his current currency amount in country i to obtain his new amount of currency in country j. Rather, when Bob is in some country i during his trip, his currency has some value in USD, and then when he travels to country j, $c_{i,j}$ gives the factor by which this value changes (in USD) when he travels to country j.

Bob would like to pick his trip sequence S' such that he maximizes his profit from the $\$100$ that he started with. Give an algorithm that determines such a sequence in $O(n)$ time.

Hint: If all the exchange rates are smaller than 1, then Bob will unfortunately have to cancel his vacation.

Problem 4 (15 points)
You are already planning a party for the fall break! You have a large a group of friends, but not all of your friends are friends with each other (and you know of all the friendships between your

friends). You wish to invite as many friends as possible to the party, but you do not want anyone to be lonely. So, you decide that you will select a subset of friends such that each invitee has at least 5 friends among the other invitees. Define an $O(m)$-time algorithm that finds such a subset of maximum size, where m is the total number of friendships among your friends.

Problem 5 (20 points)
Let $G = (V, E)$ be an undirected edge-weighted graph (i.e., edges have positive lengths). A *shortest path tree* with respect to vertex $s \in V$ is tree $T = (V, E')$ (where $E' \subseteq E$) such that the path in the tree from s to any other vertex v is also a shortest path between s and v in G. For example, we obtain a shortest path tree with respect to s by running Dijkstra's algorithm starting at s.

(a) (10 points) Give an example of a graph with 4 vertices for which every shortest path tree is *not* a minimum spanning tree.

(b) (10 points) Prove that for every edge-weighted graph G, every shortest path tree and every minimum spanning tree on G have at least one edge in common.

Problem 6 (20 points)
You have been hired by Duke 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, i.e., there exists some spanning tree that connects all the routers to the modem. Also, assume that all the cables can carry data in both directions.)

You think back to your undergraduate algorithms class, and remember that you can use DFS to find a tree that connects every router to the modem. So, you run a 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)?

Hint: Try to answer the following questions. They will help you find your algorithm.

(i) What types of edges can contribute to the count on a tree edge (e.g., other tree edges, back edges, cross edges, forward edges, etc)? Recall that this graph is undirected and T is a DFS tree.

(ii) Consider a leaf router (i.e., a vertex with only one incident edge in T) and look at the adjacent tree edge. What is the count on this edge?

(iii) If you know the counts on the edges from a router v to its children in T, can you determine the count on the edge from v to its parent in T?