CS 161 Summer 2012 Midterm Exam

July 25, 2012

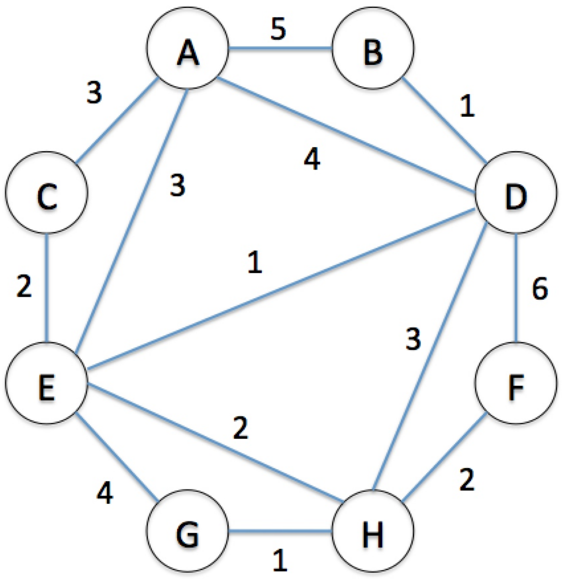
Full Name: ______________________________

Stanford ID: ______________________________

Instructions: This exam is open book, open notes, but closed computers/cell phones. Please answer all questions in the space provided. Any work outside of the space provided will not be graded. Write clearly and legibly. All statements made must be justified, and **any algorithm must be accompanied by an analysis of running time and a proof of correctness**. Any algorithm or theorem presented in class may be used without proof.

Problem	**Score**	**Out Of**
1		30
2		30
3		30
4		30
5		30
6		50
Total		200

Problem 1 (30 Points). Consider the following graph G:



(a) Suppose we run a DFS on G, where whenever we have a choice of nodes, we always pick the one alphabetically first. In what order are the nodes of G visited?

(b) Suppose we run Dijsktra's algorithm on G starting from A, breaking ties by picking the node that comes alphabetically first. In what order are nodes of G visited?

(c) Suppose we run Kruskal's algorithm on G, breaking ties by choosing the edge with the alphabetically first endpoint (if two nodes both share an alphabetically first endpoint, look at the other endpoint). In what order are edges added?

(d) Suppose we run Prim's algorithm on G, breaking ties the same way as in part (c). In what order are edges added?

Solution:

(a) A, B, D, E, C, G, H, F

(b) A, C, E, D, B, H, G, F

(c) $(B, D), (D, E), (G, H), (C, E), (E, H), (F, H), (A, C)$

(d) $(A, C), (C, E), (D, E), (B, D), (E, H), (G, H), (F, H)$

Problem 2 (30 Points).

(a) Give a linear-time algorithm for the following task: Given a directed graph G, find a node v such that all other nodes in G are reachable from v, or report that no such v exists.

(b) Give a linear-time algorithm for the following task: Given a directed graph G, find two nodes u and v such that adding the edge (u, v) makes the G strongly-connected, or report that this is impossible.

Solution:

(a) If such a node v exists, it must be in a source SCC (since a parent SCC would not be reachable from v). Moreover, it must be the **only** source SCC, since any other source SCC is only reachable from nodes in that SCC. It is also sufficient that v be in the only source SCC, since if we follow parents from any SCC, we will eventually arrive at a source SCC, which must be the one containing v.

Our algorithm is then as follows: run the strongly-connected components algorithm from class. If there is more than one source strongly connected component, report that there is no v. Otherwise, take the source strongly connected component, and output an arbitrary node from that component.

The strongly connected components algorithm takes $O(|V| + |E|)$ time, so this is the running time of our algorithm.

(b) If such an edge (u, v) exists, it must be the case that u is in the unique sink SCC, and v is in the unique source SCC. Otherwise, any sink SCC not containing u or any source SCC not containing v would remain a separate SCC after adding the edge (u, v). The existence of a unique source and sink SCC is also sufficient, since then adding the edge (u, v) for some node u in the sink and some node v in the source would create a path between any two nodes. Indeed, a path from s to t consists of a path from s to u (guaranteed since u is in the unique sink SCC), the edge (u, v), and a path from v to t (guaranteed since v in in the unique source SCC).

Our algorithm is then to run the strongly-connected components algorithm from class. If there is more than once source or sink SCC, report that there is no edge (u, v). Otherwise, let u and v be any nodes in the sink and source SCCs, respectively, and output the edge (u, v).

Problem 3 (30 Points).

(a) Suppose we have a MST T for a graph G, and now we are told that the weight of some edge (u, v) (not necessarily part of T) has decreased. Give a linear-time algorithm for constructing a MST T' of the new graph.

(b) Suppose instead we are told that the weight of (u, v) has increased. Give a linear-time algorithm for constructing a MST T' of the new graph.

Solution:

(a) There are two cases. First, if (u, v) is part of T, then T is still an MST (removing e from T gives induces a cut. Since T is a MST, e must have been the lightest edge across the cut before its weight was decreased, so it is still the lightest).

However, if (u, v) is not part of T, there might now be a lighter spanning tree. Add (u, v) to T, and run DFS to find the cycle this addition created. Then remove the heaviest edge e from this cycle, arriving at a tree T'. T' is a MST of the new graph

The running time of this algorithm is linear since we just do a DFS to find a cycle, and traverse the cycle, identifying the heaviest edge.

(b) There are two cases. First, if (u, v) is not part of T, then T is still an MST (adding (u, v) creates a cycle. Let e be some other edge in T in that cycle. Removing e from the original T induces a cut, which e and (u, v) both cross. Since T was an MST, it must be that e was the lightest edge across this cut. But it then it still is, so T is an MST).

However, if (u, v) is part of T, there might be a lighter spanning tree. Remove (u, v) from T, creating a cut. Run the connected components algorithm to find the two components this creates, and collect all the edges that cross between the components. Pick the lightest of these edges, and add it to T, obtaining a new tree T'. Since we added the lightest edge across a cut, this must be an MST.

The running time consists of a DFS, and a linear time operation to find the lightest edge across the cut. Hence, the total time is linear.

Problem 4 (30 Points). Let Γ be the alphabet consisting of characters A, B, C, D, and E. Let the string S be $EABADDCDAB$.

(a) Compute the frequencies f_A, f_B, f_C, f_D, and f_E of each character in the string S. Leaving frequencies as fractions is okay.

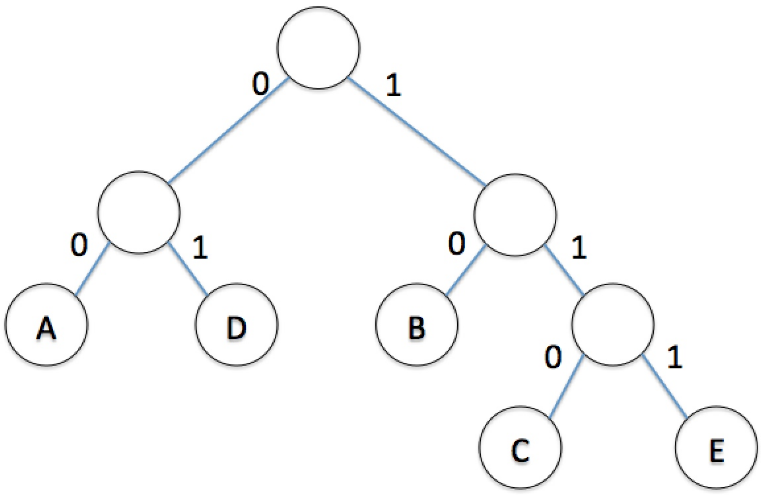
(b) Compute the optimal prefix-free encoding of the characters of Γ, and its associated binary tree. In the binary tree, from any node, the edge labeled 0 should point to the child with the alphabetically earliest descendant. For example, if one subtree contains the letters A and C, while the other contains B, the edge labeled 0 should point to the subtree containing A and C, and the edge labeled 1 should point to B.

(c) Compute the encoding of S under the encoding you found in part (b).

Solution:

(a) $f_A = 0.3$, $f_B = 0.2$, $f_C = 0.1$, $f_D = 0.3$, $f_E = 0.1$.

(b)



(c) 1110010000101110010010

Problem 5 (30 Points). Solve the following recurrence relations, giving the tightest asymptotic bound possible. You do not need to show that your bound is tight.

(a) $T(n) = 3T(\lceil n/2 \rceil) + O(n^2)$

(b) $T(n) = 9T(n/3 + 2) + O(n^2)$

(c) $T(n) = T(\lceil \sqrt{n} \rceil) + O(n)$

(d) $T(n) = 5T(\lceil n/4 \rceil + 1) + O(n)$

(e) $T(n) = T(n-1) + O(n^3)$

Solution:

(a) Master method with $a = 3$, $b = 2$, and $d = 2$. $a < b^d$, so $T(n) = O(n^d) = O(n^2)$

(b) Master method with $a = 9$, $b = 3$, $d = 2$. $a = b^d$, so $T(n) = O(n^d \log n) = O(n^2 \log n)$

(c) $\sqrt{n} < n/2$ for $n > 4$, so we can replace this recurrence with $T(n) < T(n/2) + O(n)$, which is solve by $T(n) = O(n)$. Since there is at least $O(n)$ work before the first recursive call, this bound is tight.

(d) Mast method with $a = 5$, $b = 4$, and $d = 1$. $a > b^d$, so $T(n) = O(n^{\log_b a}) = O(n^{\log_4 5})$

(e) $T(n) \leq T(n-1) + cn^3$, so

$$T(n) \leq \sum_{k=0}^{n} ck^3 < c \sum_{k=0}^{n} n^3 = cn^4 = O(n^4)$$

This bound is also tight, since

$$\sum_{k=0}^{n} k^3 = \frac{1}{4} n^2 (n+1)^2 = \Theta(n^4)$$

Problem 6 (50 Points). Determine whether each of the following statements is true for false, and explain why with a proof or counterexample.

(a) $(\log n)^{\log n} \in \Theta(n^{\log(\log n)})$. **True / False**

(b) If we perform a DFS on a dag, there will never be any cross edges. **True / False**

(c) Dijkstra's algorithm works on graphs with negative edges, provided any node with a negative incoming edge has no outgoing edges. **True / False**

(d) In the Huffman encoding algorithm where all frequencies are distinct, the character with the highest frequency is guaranteed to be one of the characters with the smallest codewords. **True / False**

(e) Every single operation in the disjoint-set data structure is guaranteed to take at most $O(\log^* n)$ time. **True / False**

(f) QuickSort runs in worst-case $O(n \log n)$ time. **True / False**

(g) If a graph G has a unique heaviest edge e, and e is in some MST, then e is in every MST. **True / False**

(h) If a weighted graph has edge lengths that are integers from 1 to 10, then we can compute shortest paths in linear (i.e. $O(|V| + |E|)$) time. **True / False**

(i) After running DFS on a directed graph, the node with the lowest `pre` number will be in a sink strongly connected component. **True / False**

(j) In a sparse graph ($|E| = O(|V|)$), we can compute shortest paths in linear time. **True / False**

Solution:

(a) **True**. $(\log n)^{\log n} = \left(2^{\log\log n}\right)^{\log n} = \left(2^{\log n}\right)^{\log\log n} = n^{\log\log n}$. Since the two functions are in fact equal, they are definitely Big Theta of each other.

(b) **False**. A DFS on a dag never produces any **back** edges. Cross edges are definitely allowed. Consider the dag where node 1 points to nodes 2 and 3, and nodes 2 and three each point to node 4. A DFS starting at node 1 will, say, explore 1, then 2, then 4. After backtracking back to 1, it will explore node 3, and see the edge $(3, 4)$. Since 4 is already explored but not a descendant of 3, this is a cross edge.

(c) **True**. Nodes with no outgoing edges never update the distance to any other nodes. Therefore, while nodes with negative incoming edges may have an incorrect distance when processed, this will not affect the correctness of any other distance if these nodes have no outgoing edges. Further, once each of the parents of one of these nodes are processed, nodes with incoming negative edges will actually have the correct distance.

(d) **True**. If not, swap the highest frequency character with a character that has a shorter codeword. This new encoding will have smaller cost than the original. meaning the original could not have been the Huffman encoding.

(e) **False** Every single operation is guaranteed to be $O(\log n)$, the $O(\log^* n)$ is only amortized time. Consider a sequence of union operations, where we always call union on the roots of the sets. There will be no path compression in this case. After all the nodes are unioned into one set, we can call `Find` on one of the leaves. This leaf can have depth $O(\log n)$, resulting in an $O(\log n)$ running time.

(f) **False** $O(n \log n)$ is the expected running time if we pick the pivot at random. However, we may be unlucky and always chose the pivot to be the smallest element in the list. In this case, the running time will be $O(n^2)$.

We could use the linear time select algorithm to choose a good pivot every time. In this case, the worst-case running time will in fact be $O(n \log n)$.

(g) **True**. We will actually prove something stronger: that removing e disconnects the graph. Indeed, suppose we take our MST T, and remove e. This induces a cut of the graph. If removing e does not disconnect the graph, then there is some other edge e' in the graph that crosses the cut. This edge has lighter weight, so if we add it to T in place of e, we get a new tree T' of lighter total weight. Since T was an MST, this is impossible, so the edge e' could not exist.

Now it is clear that e is part of every tree, since without e, no collection of edges could span the whole graph.

(h) **True** From class, we saw that we could replace an edge of weight w with w unweighted edges. The running time of this approach is $O(|V| + W)$, where W is the total weight of all edges. Since all edges have weight at most 10, $W \leq 10|E|$, so the running time is $O(|V| + |E|)$.

(i) **False**. The node with the lowest pre number is just the first node we visit. Therefore, it could be any node.

(j) **False**. In a sparse graph, the best we know how to do is $O(|V| \log |V|)$.