CS570
Analysis of Algorithms
Fall 2013
Exam I

Name: _____

Student ID: _____

____Tuesday/Thursday Session ____Wednesday Session ____DEN

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	20	
Problem 5	15	
Problem 6	15	
Total	100	

2 hr exam
Close book and notes

If a description to an algorithm is required please limit your description to within 150 words, anything beyond 150 words will not be considered.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
A directed graph G is strongly connected if and only if G with its edge directions reversed is strongly connected.
True, by definition of strongly connectedness.

[TRUE/FALSE]
If a weighted undirected graph has two MSTs, then its vertex set can be partitioned into two, such that the minimum weight edge crossing the partition is not unique.
True, Let T1 and T2 be two distinct MSTs. Let e1 be an edge that is in T1 but not in T2. Removing e1 from T1 partitions the vertex set into two connected components. There is a unique edge (call it e2) that is in T2 that crosses this partition. By optimality of T1 and T2, both e1 and e2 should be of the same weight and further should be of the minimum weight among all edges crossing this partition.

[TRUE/FALSE]
If the vertex set of a weighted undirected graph can be partitioned into two, such that the minimum weight edge crossing the partition is not unique, then the graph has at least two MSTs.
False. Consider the graph with edge set {(a,b),(b,c),(c,d)} where all edge weights are identical. ({a,d},{b,c}) is a partition that does not have a unique minimum weight edge crossing it, however the graph (being a tree) has a unique MST.

[TRUE/FALSE]
The number of binomial trees in a binomial heap with n elements is at most O(log n).
True. See the lecture notes.

[TRUE/FALSE]
A bipartite graph cannot contain a cycle of length 3 or greater.
False. Consider the graph which is also a cycle with 4 vertices. It is bipartite contradicting the assertion in question.

[TRUE/FALSE]
Since Fibonacci heaps guarantee amortized cost and not worst case cost, the run time complexity of Dijkstra's algorithm using Fibonacci heaps may be worse than that of a binary heap implementation.
False. For Fibonacci heaps with n entries, the amortized running time of delete min is O(log n) and the amortized running time for decrease key is O(1). Thus for every sequence of extract mins/decrease keys with (A extract mins and B decrease keys in total), the worst case running time is bounded by O(B+A log n).
When applied to the implement the priority queue in Dijkstra, A is O(B), that is the number of vertices visited is asymptotically bounded by the number of edges relaxed. A binary heap implementation would take at least Theta(B log n) time since decreasy key takes Theta(log n) time. Thus, the Fibonacci heap implementation is faster even in the worst case.

[TRUE/FALSE]
Implementations of Dijkstra's and Kruskal's algorithms are identical except for the relaxation steps.

False. It is Dijkstra's and Prim's that bear a resemblance not Dijkstra's and Kruskal's. For instance, at a generic stage in Kruskal's we could have multiple disconnected components (a forest) while Dijkstra's always grows a tree.

[TRUE/FALSE]
The divide step in the divide and conquer algorithm always takes less time than the combine step, therefore, the complexity of a divide and conquer algorithm is never dependent on the divide step.

False. Divide step could be slower than the combine step.

[TRUE/FALSE]
Suppose that in an instance of the original Stable Marriage problem with n couples, there is a man M who is first on every woman's list and a woman W who is first on every man's list. If the Gale-Shapley algorithm is run on this instance, then M and W will be paired with each other.

True. If (M,W) are not matched with each other, then since (M,W) prefer each other to everyone else, they can swap making the matching unstable.

[TRUE/FALSE]
For a search starting at node s in graph G, the DFS Tree is never as the same as the BFS tree.

False. For instance, G being a tree is a counterexample.

2) 15 pts
Suppose you are choosing between the following three algorithms:
I) Algorithm A solves problems by dividing them in constant time into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
II) Algorithm B solves problems of size n by dividing in constant time and recursively solving two subproblems of size n-1 and then combining the solutions in constant time.
III) Algorithm C solves problems of size n by dividing them in constant time into nine subproblems of size n/3, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.
What are the running times of each of these algorithms (in big-O notation), and which would you choose?

I. $O(n^{\log_2 5})$

II. $O(2^n)$

III. $O(n^2 \log(n))$ - the minimum time complexity

3) 15 pts

Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). The running time of your algorithm should be linear.

First do a BFS/DFS to make sure that the graph is a directed acyclic graph (DAG) (otherwise, the coursework cannot be completed in finitely many semesters since there would be a cycle of prerequisite dependencies).

The problem then reduces to finding the length of the longest path in the DAG which can be solved as follows.

Do a topological sort, let the resulting set of vertices be {v1,v2,v3,...,vn} in O(|V|+|E|) time. Recall from HW 4, prob 5 that the above topological sorting implies that for all j>i, there is no directed path from vj to vi.

Set all the edge lengths to equal 1.

Then iteratively (similar to Dijkstra's modification in HW 4, prob 5) compute the length of the longest path to v1, then the length of the longest path to v2, and so on until vn as explained below.

Let L(v) denote the length of the longest path ending at v. We will iteratively compute L(v).

Start with v1. Since v1 has no incoming edges, the length of the longest path to v1 is L(v1)= 0. The length of the longest path to v2 is L(v2) = 1 if the edge (v1,v2) is present and 0 otherwise. More generally, If vi does not have an incoming edge, then set L(vi)=0. If vi has an incoming edge, then L(vi) = max (L(vk) + 1) where the maximum is taken over all vk such that (vk,vi) is present in the graph. (Remark: The ordering implies that the only k values that we need to consider are k<i).

The number of semesters required then turns out to be 1 + max(L(v)) where the maximum is taken over all vertices.

Observe that once sorted, each edge (vk,vi) is considered at most once, hence the running time for computing the L() values given the ordering is O(|V|+|E|). The total running time thus turns out to be O(|V|+|E|)

4) 20 pts

In Internet routing, there are delays on lines but also, more significantly, delays at routers. Suppose that in addition to having edge lengths $\{l_e : e \in E\}$, a graph also has vertex costs $\{c_v : v \in V\}$. Now define the cost of a path to be the sum of its edge lengths, plus the costs of all vertices on the path (including the endpoints). Give an efficient algorithm for the following problem. Input: A directed graph G = (V; E); positive edge lengths l_e and positive vertex costs c_v; a starting vertex $s \in V$. Output: An array cost such that for every vertex u, cost[u] is the least cost of any path from s to u (i.e., the cost of the cheapest path), under the definition above. Notice that cost[s] = c_s.

We define a new length function l' on the edges as follows. For every (u,v) in E, let l'_(u,v) := l_(u,v) + c_v.

For a vertex v, Let d(v) denote the length of a shortest path to v from s obtained by applying Dijkstra's algorithm to G with this new edge length function l' with the modification that in the initial step d(s) is set to c_s instead of 0.

Clearly, cost[v] = d(v) and we are done. (For completeness you should prove this !)

5) 15 pts

There is a stream of integers that comes continuously to a small server. The job of the server is to keep track of k largest numbers that it has seen so far. The server has the following restrictions:

a) It can process only one number from the stream at a time, which means it takes a number from the stream, processes it, finishes with that number and takes the next number from the stream. It cannot take more than one number from the stream at a time due to memory restriction.
b) It has enough memory to store up to k integers in a simple data structure (such as an array), and some extra memory for computation, such as comparison, etc.
c) The time complexity for processing one number must be *better* than O(k). Anything from O(k) (including O(k)) or worse is not acceptable.

Design an algorithm on the server to perform its job with the requirements listed above.

Use a binary min heap on the server.

a) Do not wait until k numbers have arrived at the server to build the heap, otherwise you would create a time complexity O(k). Instead, build the heap on-the-fly: as long a a number arrives, if the heap is not full, insert the number to the heap and execute heapify. The first k numbers are obviously the k largest numbers that the server has seen
b) When a new number arrives and the heap is full, compare the number n to the minimum number of the heap at the root r, which can be found by O(1). If n <= r, ignore n. Otherwise, extract min (O(log k)), insert the new number n to the heap and call heapify to maintain the structure of the heap.
c) The overall complexity is O(log k)

Notes: Binary max heap is completely opposite to what one would need to solve this problem. Therefore, proposing max heap does not get any partial credit. The same policy applies to those, who propose to just save the numbers in an array because searching for the min number in the array has linear complexity.

6) 15 pts

N men and N women were participating in a stable matching process in a small town named Walnut Grove. A stable matching was found after the matching process finished and everyone got engaged. However, a man named Almazo Wilder, who is engaged with a woman named Nelly Oleson, suddenly changes his mind by preferring another woman named Laura Ingles, who was originally ranked right below Nelly in his preference list, therefore Laura and Nelly swapped their positions in Almanzo's preference list. Your job now is to find a new matching for all of these people and to take into account the new preference of Almanzo, but you don't want to run the whole process from the beginning again, and want to take advantage of the results you currently have from the previous matching.

Describe your algorithm for this problem.

Assume that no woman gets offended if she got refused and then gets proposed by the same person again.

First, Almanzo leaves Nelly and proposes to Laura:

a) **If** Laura rejects Almanzo and decides to stay with her current fiancé (her current fiancé is ranked higher than Almanzo in her preference list), then Almanzo just goes back and proposes to Nelly again and gets engaged with her. Algorithm stops. This is where you take advantage of the previous matching.

b) **Else**, Laura accepts Almanzo proposal, she will get engaged with him and leave her current fiancé single (say Adam). And here comes the main part of the matching puzzle.

b.1) Note that Adam may or may not simply propose to Nelly and be engaged with her. The reason is Nelly may be far below Laura in his preference list and there are more women (worse than Laura and better than Nelly), to whom he prefers to propose first. Furthermore, the fact that Nelly becomes single can create more instabilities for those men, who once proposed to Nelly and were rejected by her because she preferred Almanzo. Thus, Nelly is ranked higher than the current fiancées of those once "unlucky" men and they are ready to propose to Nelly again for a second chance.

b.2) Thus, one way is to put Adam and all those rejected-by-Nelly men in the pool of single men, and put Nelly and the fiancées of the rejected-by-Nelly men in the pool of free women. Similar to Nelly's case, moving any woman from the engaged state to the single state may create more instabilities; thus one needs execute step b.2) recursively to build the pools of free men and women. In the worst case, all men and all women will end up in the single pools; in the best case (when Laura accepted Almanzo's proposal), only Adam and Nelly are in the 2 single pools, one for men and another for women.

b.3) Execute G-S algorithm until there is no free man. For each man, if there is a woman who once rejected him and is now free, he will try to propose to her again. Otherwise, he will propose to those women that he has never proposed to, moving top down in his preference list. In the latter case, more men (subsequently more women) may go to the single pools.

Note: The above solution is described in details in order to clarify the problem. Students can simply write pseudo code without having to discuss the details of each case as above.

Additional Space