CS570
Analysis of Algorithms
Fall 2014
Exam I

Name: _____

Student ID: _____

Thursday Evening Section **DEN Yes / No**

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
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]
Adding a number *w* on the weight of every edge of a graph might change the shortest path between two vertices u and v.

True:
Consider G = (V, E) with V = {u, x, v} and E = {ux, xv, uv}. Let weight of edge ux = w_{ux} = w_{xv} = 3 and w_{uv} = 7. Then the shortest path from u to v is given by u->x->v with a total weight of 6. However, if we now add 2 to every edge weight, the path u->x->v will have a total weight of 10 while the path u->v will have a weight of 9, making it the new shortest path.

[TRUE/FALSE]
Suppose that for some graph G we have that the average edge weight is *A*. Then a minimum spanning tree of G will have weight at most $(n-1) \cdot A$.

False:
We may be forced to select edges with weight much higher than average. For example, consider a graph G consisting of a complete graph G' on 4 nodes, with all edges having weight 1 and another vertex u, connected to one of the vertices of G' by an edge of weight 8. The average weight is (8 + 6)/7 = 2. Therefore, we would expect the spanning tree to have weight at most 4*2 = 8. But the spanning tree has weight more than 8 because the unique edge incident on u must be selected.

[TRUE/FALSE]
DFS finds the longest paths from start vertex *s* to each vertex *v* in the graph.

False:
Depends on the order in which the nodes are traversed

[TRUE/FALSE]
If one can reach every vertex from a start vertex s in a directed graph, then the graph is strongly connected.

False

[TRUE/FALSE]
F(n) = $4n+3\sqrt{n}$ is both $O(n)$ and $\Omega(n)$.

True:
The dominant term is 4n, which is obviously both O(n) and \Omega (n)

[TRUE/FALSE]
In Fibonacci heaps, the decrease-key operation takes $O(1)$ time.

True:
Just as definition of decrease-key operation in Fibonacci heaps

[TRUE/FALSE]
If the edge weights of a weighted graph are doubled, then the number of minimum spanning trees of the graph remains unchanged.

True:
With edge weights doubled, the weights of all possible spanning trees in the graph are doubled. So any MST in the original graph is also the MST in the new graph; any spanning tree that is not the MST in the original graph is still not the MST in the new graph.

[TRUE/FALSE]
Given a binary max-heap with n elements, the time complexity of finding the smallest element is O(lg n).

False:
The smallest element should be among the leaf nodes. Consider a full binary tree of n nodes. It has (n+1)/2 leafs (you can think of why). Then the worst case of finding the smallest element of a full binary tree (heap) is \Theta(n)

[TRUE/FALSE]
An undirected graph $G = (V, E)$ must be connected if $|E| > |V| - 1$

False:
Consider a graph having nodes: a, b, c, d, e. {a, b, c, d} forms a fully connected subgraph, while e is isolated from other nodes. Now the fully connected subgraph has 6 edges, and there are only 5 nodes in total. But this graph is not a connected graph.

[TRUE/FALSE]
If all edges in a connected undirected graph have unit cost, then you can find the MST using BFS.

True:
Any spanning tree of a graph having only unit cost edges is also a MST, because the weight is always n-1 units. Of course, BFS gives a spanning tree in the connected graph.

2) 16 pts

At the Perfect Programming Company, programmers program in pairs in order to ensure that the highest quality code is produced. The productivity of each pair of programmers is the speed of the slower programmer. For an even number of programmers, give an efficient algorithm for pairing them up so that the sum of the productivity of all pairs is maximized. Analyze the running time and prove the correctness of your algorithm.

Solution:

A simple greedy algorithm works for this problem. Sort the speeds of the programmers in decreasing order using an optimal sorting algorithm such as merge sort. Consecutive sorted programmers are then paired together starting with pairing the fastest programmer with the second fastest programmer.

Sorting takes $O(n \lg n)$ time while pairing the programmers takes $O(n)$ time giving a total running time of $O(n \lg n)$.

Correctness: Let P be the set of programmers. The problem exhibits an optimal substructure.

Assume the optimal pairing. Given any pair of programmers $(i; j)$ in the optimal pairing, the optimal sum of productivity is just the sum of the productivity of $(i; j)$ with the optimal sum of the productivity of the all pairs in $P-\{i,j\}$.

We now show that the greedy choice works by showing that there exists an optimal pairing such that the two fastest programmers are paired together. Assume an optimal pairing where fastest programmer i is not paired with the second faster programmer j. Instead let i be paired with k and j be paired with l. Let p_i, p_j , p_k and p_l be the programming speeds of i, j, k and l respectively. We now change the pairings by pairing i with j and k with l. The change in the sum of productivities is

$(p_j + \min(p_k; p_l)) - (p_k + p_l) = p_j - \max(p_k , p_l) >= 0$

since p_j is at least as large as the larger of p_k and p_l. We now have an optimal pairing where the fastest programmer is paired with the second fastest programmer. Hence to find the optimal solution, we can keep pairing the two fastest remaining programmers together.

3) 16 pts

The graph K_n is defined to be an undirected graph with n vertices and all possible edges (a fully connected graph). That is, the vertices are named $\{1, \ldots, n\}$ and for any numbers i and j with $i \neq j$, there is an edge between vertex i and vertex j. Describe the result of a breadth-first search and a depth-first search of K_n. For each search, describe the resulting search tree.

Solution:

For the BFS, the algorithm looks at all the neighbors of the root node before going to the second level. Since every node in the graph is a neighbor of every other node, every node other than the root is put at level 1. Thus the tree has one node at level 0 and n-1 nodes at level 1. The non-tree edges are exactly those edges between different nodes on level 1 -- there are (n-1 choose 2) of these.

For the DFS, the search of the root node (call it 1) will find another node 2, and then the recursive call on node 2 will find node 3, and so forth. None of the recursive calls can terminate while any undiscovered nodes remain. So the tree edges form a single path of n-1 edges, with one node on each level and thus a single leaf. There are back edges, (n-1 choose 2) of them, as each node has a back edge to each of its ancestors except its parent.

4) 16 pts

A *d-ary heap* is like a binary heap, but instead of 2 children, nodes have *d* children.

(a) How would you represent a d-ary heap in an array? [4 points]

If we know the maximum number, N, of nodes that we may store in the d-ary heap, we can allocate an array H of size N indexed by $i = 1, 2, \ldots, N$. The heap nodes will correspond to positions in H. $H[1]$ will be the root node and for any node at position i in H, its children will be at positions $d*(i-1) + 2$, $d*(i-1) + 3$, ... $d*(i-1) + d + 1$, and the parent position at $\lfloor (i-2)/d \rfloor + 1$. If there are $n < N$ elements in the heap at any time, we will use the first n indices of the array to store the heap elements.

(b) What is the height of a d-ary heap of n elements in terms of n and d? [4 points]

There is 1 node at level 0 (L_0), d at L_1, d^2 at L_2 and so on. Let h be the height of the d-ary heap. Then there are d^h nodes at the last level of the heap. Thus,

$$n = 1 + d + d^2 + d^3 + \ldots + d^h$$

Solving this, we get

$$h = \log_d (n(d-1)+1) - 1$$

(c) Give an efficient implementation of ExtractMin. Analyze its running time in terms of d and n. [8 points]

Similar to the ExtractMin operation for a binary heap, for a d-ary heap with n elements, we find the minimum (the root node) in O(1) time and to delete it, we replace $H[1]$ with $H[n] = w$. If the resulting array is not a heap, we use Heapify-up or Heapify-down to fix the heap in $O(h) = O(\log_d n)$ time.

5) 16 pts

You are given a directed graph representing several career paths available in the industry. Each node represents a position and there is an edge from node v to node u if and only if v is a pre-requisite for u. Starting positions are the ones which have no pre-requisite positions. Except starting positions, we can only perform a position only if any of its pre-requisite positions are performed. Top positions are the ones which are not pre-requisites for any positions. Ivan wants to start a career at any of the starting positions and achieve a top position by going through the minimum number of positions. Using the given graph provide a linear time algorithm to help Ivan choose his desired career path. Note that this graph has no cycles.

Solution: Add a node s and connect an edge from s to all nodes that have no incoming edges (starting positions). Add a node t and connect an edge from all nodes with no outgoing edges (top positions) to t. Do a BFS from s and find the shortest path to t. This will give you the shortest path from a starting position to a top position in $O(m+n)$.

6) 16 pts

Suppose we are given an instance of the Minimum Spanning Tree problem on a graph G. Assume that all edges costs are distinct. Let T be a minimum spanning tree for this instance. Now suppose that we replace each edge cost c_e by its square, c_e^2 thereby creating a new instance of the problem with the same graph but different costs.
Prove or disprove: T is still a MST for this new instance.

Solution:
The claim is false.
Note that the edge cost can be negative. (This is the key difference between this problem and a homework problem in HW4)

Consider the following example:

The original graph: G = (V, E) being an undirected graph, where V = {a, b, c}, E = {(a,b),(b,c), (a,c)}. Weights: w(a,b) = 1, w(b,c) = -3, w(a,c)= 2.

The MST is : T = (T_v, T_E), where T_v = {a, b, c}, T_E = {(a,b), (b,c)}

After squaring the edge weights, we get a new graph G' with w'(a,b) = 1, w'(b,c) = 9, w'(a,c)= 4.
The MST is: T' = (T'_v, T'_E), where T'_v = {a, b, c}, T'_E = {(a,b), (a,c)}.

So the MST changes..

Additional Space