COT5405 Analysis of Algorithms Spring 2012
Solution to Assignment 4

Problem 1:

a) Assume you have two different MST's. Take any cut, such that the MST's differ with regard to edges on the cut. Since the edges on the cut have distinct weights, it can be shown that one of them is not an MST (by adding an edged from the other MST and deleting and edge from itself, you can get smaller weight). Since this applies to all cuts, the MST's must have all edges in common, hence a contradiction.

b) It is not necessary that the second best be unique. Second best as the next theorem shows can be created by adding an edge to form a cycle on the MST and then deleting the next lightest edge. In order to create a counter-example create a graph with two cycles, and assume an MST that chooses the msecond heaviest edge in both cycles but not the heaviest edge. Now create 2 second-best MSTs by replacing the 2nd heaviest with the heaviest (choose the weights to be the same increase) for example, add an edge with weight 20 and delete an edge with weight 19 (increase of 1) and in the other, add edge with weight 30 and delete edge with weight 29 (also an increase of 1).

c) The proof is similar to the argument above. Let us say that the MST and the 2nd-MST differ in more than 2 edges, i.e. there are at least 2 edges in the MST that do not occur in the 2nd-MST and likewise 2 edges in the 2nd MST that do not occur in the MST.
One can then add an edge from the MST and delete one from the 2nd MST to get a new tree that has weight less than the 2nd MST, thus contradicting the fact that the 2^{nd} MST is 2nd lightest.

Problem 2:

In order to get a profit out of the trade, the product of the exchange rates to get back the same currency should be greater than 1.

Now since Neg-Cycle is to be used, this product of exchange rates should be converted into sum by some means. The best means to do this to take the log (exchange rate), since we have the property that

$$\log (ab) = \log (a) + \log (b)$$

Given the matrix A where A[i , j] which gives the exchange rate from i to j, we can draw a graph with the currencies as the vertices and the ***log (exchange rates)*** ie log (A[i,j]) as the weight for the edge between i and j.

Now in order to get profit, we have the property that

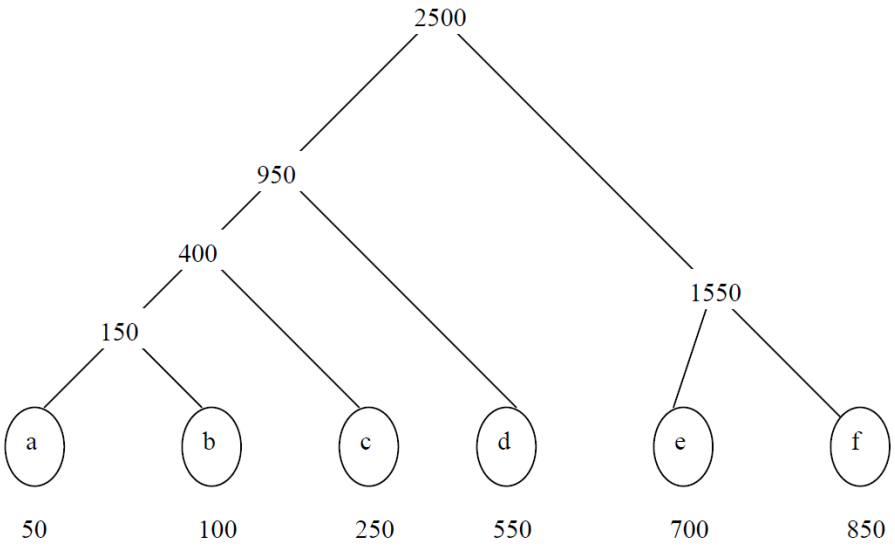
$$\text{Product of exchange rates} > 1$$

Since to check for a profit or a loss, we have to get back to the same currency and hence a cycle comes into picture.

Thus to check the product for weights along a cycle and a profit for that cycle, we have that the sum of the log (exchange rate) i.e. weights along the cycle be greater than 0. (as most of the students have given : sum of negated log(weights) should be less than 0).

Thus a Neg-Cycle algorithm can be run on this graph. A Neg-Cycle thus obtained will gurantee a profit along the cycle. Thus the trader can use the Neg-Cycle algorithm to make money.

Problem 3:



Number of bits necessary to code this file:
= 4*50 + 4*100 + 3*250 + 2*550 + 2*700 + 2*850
= 200 + 400 + 750 + 1100 + 1400 + 1700
= 5550

Problem 4:

Every internal node in a Huffman Tree has got exactly two children (otherwise, we can simply move that node up). This unique property helps us to efficiently encode the tree. This means, there are exactly (n-1) nodes, which makes it a total of (n-1) internal nodes + n leaves = (2n-1) nodes. Now, designate a node as 1 if it has got two kids, 0 if no kids (means leaf). This will mean the representation of (2n-1) nodes will take (2n-1) bits. However, just stating the size requirement is not enough – we need to come up with some efficient encoding/method that will be able to interpret the stream of (2n-1) bits into a Huffman tree structure. We employ a BFS-like level-wise top-down, left-to-right traversal method (which is preorder). Post-order would also be fine, though In-order would not be unique. And for designating n leaves, since we are just using natural numbers (0-n), every number (leaf) will require a maximum of _log n_ bits. Thus an additional n _log n_ bits would be required. We can use the first 2n-1 bit of the codecoding file to generate the tree structure, and then put the leaves one by one (each one will occupy exactly _log n_ bits) at the empty leaf nodes.

Problem 5:

The simple way to look at this problem would be to construct a new graph G', which is the same as graph G except that the in G', an edge's weight is the negation of the weight of the corresponding edge in G. Then running Kruskal's method on G', picking tree edges in increasing order of weight, which is analogous to get the maximum spanning tree T' by picking the edges in decreasing order of weight in G. So the modified Kruskal's method can get the maximum spanning tree.

[Approach 2: you can refer to theorem 23.1 (reference book) to find the safe edges for maximum spanning tree. Approach 3: Using the similar proof for Kruskal's method. Assuming Max-ST is maximum spanning tree, show that you can convert Max-ST to T' (which is produced by modified Kruskal's method) without decreasing weight.]

Problem 6:

We know that Kruskal's method can produce the Min-ST in polynomial time. We also know that modified Kruskal's method can produce Max-ST in polynomial time. Add up the weight of Min-ST and Max-ST can be done in polynomial time.

(a) Is there a spanning tree of weight atmost k?
if weight (Min-ST) <= k, answer "yes", else answer "no".
(a) can be solvable in polynomial time, so (a) is in P, thus also in NP and Co-NP.
(b) Is there a spanning tree of weight atleast k?
If weight (Max-ST) >=k, answer "yes", else answer "no".
So (b) is in P, NP and Co-NP.
(c) Do all spanning trees have weight less than k?
If weight (Max-ST) < k, answer "yes", else answer "no".
So (c) is in P, NP and Co-NP. Actually, (c) is the complement of (b).
(d) Do all spanning trees have weight greater than k?
If weight (Min-ST) > k, answer "yes", else answer "no".
So (d) is in P, NP and Co-NP. Actually (d) is the complement of (a).

Problem 7:

a) Hamiltonian Cycle to Hamiltonian Path:

Note: If it has been only proved that if G has a Hamiltonian Cycle then G' has a Hamiltonian path after the reduction, points have been cut since the reverse way has to be proved also. Incorrect reductions get even less credit.

For a given graph G = (V,E) construct a graph G' = (V',E') such that G' has a Hamiltonian path exactly iff G has a Hamiltonian cycle. If we can do that, we can construct an algorithm that solves Hamiltonian cycle by using Hamiltonian-path a subroutine.

This would prove that Hamiltonian-path is at least as hard Hamiltonian-cycle (up to a polynomial factor).

The construction is as follows. Let u Є V be an arbitrary vertex. We add a copy of u (together with all edges) and call it u'. Then we add two vertices v and v' and the edges (u, u') and (v, v'). We call the result G'. We now verify that G' has the properties we need. Assume G has a Hamiltonian cycle. Then G' has a path starting in u and ending in u0. We can extend this path to a Hamiltonian path by adding the edges (u, u' and (v, v').

Now assume that G' has a Hamiltonian path. Obviously this path must have v and v' as end-points, which implies that there must be a path from u to u'. We can then close the path by letting removing the edge to u' and let it go to u instead. (From the construction of u' this is always possible.)

Thus the hamiltonian cycle problem can be reduced to hamiltonian path problem.

b) Hamiltonian Path to Hamiltonian Cycle:

Note: If it has been only proved that if G has a Hamiltonian path then G' has a Hamiltonian cycle after the reduction, points have been cut since the reverse way has to be proved also. Incorrect reductions get even less credit.

Here create a new graph G' from G(V, E) such that the new graph G' has an extra node v' and it is connected to every other node in the graph G'.

Now if G has a Hamiltonian path from a node v0 to vn then G' has a Hamiltonian cycle starting from v' going to v0, following the path to vn and then returning to v'.

Similarly if G' has a Hamiltonian cycle from v', then G will have a Hamiltonian path from a node v1 to vn such that the cycle in G' is from v'->v1->…. -> vn->v'. Thus removing node v' will result in a Hamiltonian path v1 -> vn in G.

c) Here you need to prove that both the Hamiltonian Path and Hamiltonian Cycle problem belong to the class NP by giving nondeterministic polynomial time algorithms for them. Then using the above facts of reductions in ‘a’ and ‘b’, it can be proved that if either one of them is NP complete, the other is also NP-complete.

Note: Here many students have not proved that the problems ∈ NP class by giving NP time algorithms for them and hence have lost points.