CS570
Analysis of Algorithms
Summer 2010
Final Exam

Name: _____

Student ID: _____

____Check if DEN student

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	20	
Problem 4	15	
Problem 5	15	
Problem 6	10	
Total	100	

2 hr exam
Close book and notes

1) 20 pts

Mark the following statements as **TRUE**, **FALSE, or UNKOWN**. No need to provide any justification.

[TRUE/FALSE]

Given a network *G(V, E)* and flow *f*, and the residual graph G_f(V', E'), then |V|=|V'| and 2|E|>=|E'|.

[TRUE/FALSE]

The Ford-Fulkerson Algorithm terminates when the source *s* is not reachable from the sink *t* in the residual graph.

[TRUE/FALSE/UNKOWN]

NP is the class of problems that are not solvable in polynomial time.

[TRUE/FALSE/UNKOWN]

If problem A is NP complete, and problem B can be reduced to problem A in quadratic time. Then problem B is also NP complete

[TRUE/FALSE/UNKOWN]

If X can be reduced in polynomial time to Y and Z can be reduced in polynomial time to Y, then X can be reduced in polynomial time to Z.

[TRUE/FALSE]

Let G(V,E) be a weighted graph and let T be a minimum spanning tree of G obtained using Prim's algorithm. The path in T between *s* (the root of the MST) and any other node in the tree must be a shortest path in G.

[TRUE/FALSE]

DFS can be used to find the shortest path between any two nodes in a non-weighted graph.

[TRUE/FALSE]

The Bellman-Ford algorithm cannot be parallelized if there are negative cost edges in the network.

[TRUE/FALSE]

A perfect matching in a bipartite graph can be found using a maximum-flow algorithm.

[TRUE/FALSE]

Max flow in a flow network with integer capacities can be found exactly using linear programming.

2.We know that max-flow problems can be solved in polynomial time. However, consider a modified max-flow problem in which every edge must either have zero flow or be completely saturated. In other words, if there is any flow through an edge at all, the flow through the edge is the capacity of the edge. Let's call a flow a **saturated** flow if it satisfies this additional constraint, as well as the usual constraints of flow problems. The modified problem asks us to find the maximum saturated flow.

a.(2 point) Formulate this problem as a decision (yes/no) problem. (We'll call it MAX-SATURATED-FLOW.) Write your answer in the form of a question.

ANSWER.

Is there a saturated flow of value *k* or greater? (Alternative: is there a saturated flow of value k?) Here, *k* is an arbitrary number.

b. (5 points) Show that MAX-SATURATED-FLOW is in NP.

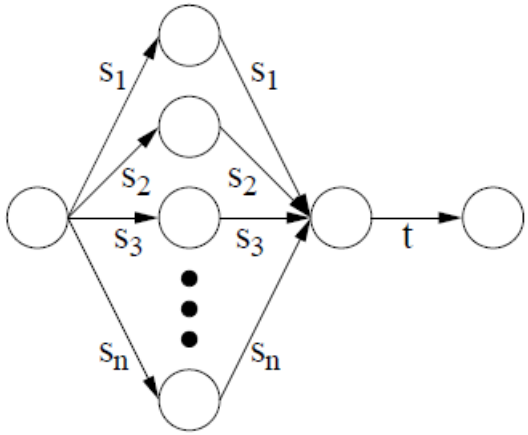
ANSWER.

For a certificate, we use a list of all the saturated edges. Given such a list, we can check in linear time the flow out of the source node, and whether the conservation condition for each node (exceptthe source and sink) is satisfied.

c. (13 points) Show that MAX-SATURATED-FLOW is NP-hard. Hint: use subset-sum problem.

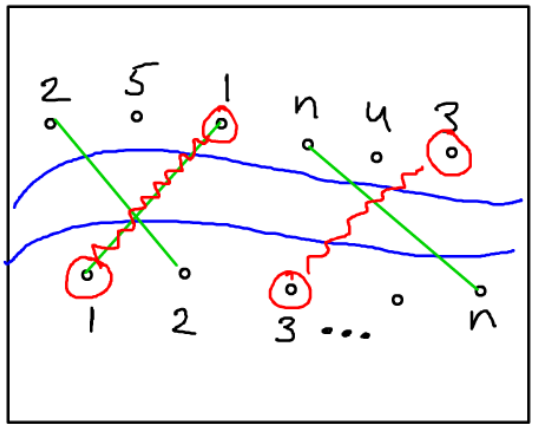
ANSWER.

It's relatively easy to reduce SUBSET-SUM to MAX-SATURATED-FLOW. Consider the NP-hard problem of finding a subset of $\{ s_1, s_2 \ldots s_n\}$that sums to *t*. This problem is solved if we find a saturated flow of value *t* in the following graph (or alternatively, the maximum saturated flow of this graph).



3.Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates a(1) ... a(n) and n cities on the northern bank with x-coordinates b(1) ... b(n). Note that the x-coordinates are NOT in sorted order. You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. But when connecting cities, you can only connect city i on the northern bank to the corresponding city i on the southern bank. Design an efficient algorithm for that.

Answer:
Let's us first interpret the question correctly. We want to build as many none crossing bridge as possible. As shown in the following image,



In the lower level of river, we use 1…n to denote all the cities in the south, in the upper level of the river, we use the same number to denote the corresponding city in the north. Our problem is to connect as many of them as possible.

Let X(i) to be the index of corresponding city on the northern bank (e.g., X(1) = 3 in the above figure). To compute each X(i), we need O(n) time
Therefore, the problem to reduce to "Find the longest increasing subsequence in X(1)…X(n)". This problem is solved in Homework 4, please refer to the solution of problem 3 in Homework 4.

The complexity of overall algorithm is $O(n^2)$

Grading Strategy:
Correct Solution: 12 points
Complexity : 4 points.
Correctness Proof: 4 points

Any ***CORRECT*** solution within time complexity $O(n^3)$ (including $O(n^3)$) will receive full credits for the solution part, otherwise, will minus one or two points.

4.15 pts

Consider the following vertex cover algorithm for an undirected graph G = (V, E).
0. Initialize C = Null set.
1. Pick any edge e = (u, v) ϵ E, add u and v to C. Delete all edges incident on either u or v.
2. If E is empty, Output C and exit. Otherwise, go to step 1.
Show that C is a vertex cover and that the size of C is at most twice as big as the minimum vertex cover. (Thus we have a 2-approximation)

Answer:

Every edge $e \in E$ was either picked in step 1 or deleted in step 1. If it was picked, then both its edges were added to C and is hence covered. If it was deleted, then it already had at least one incident vertex contained in C and is hence covered. Thus C is indeed a vertex cover.

Let M be the set of edges picked in step 1. Let C_{opt} be an minimal vertex cover. Observe that no two edges in M share a vertex (When we pick an e, the deletion step ensures that no other edge that shares an edge with e is ever picked). Thus for each edge $e \in M$, C_{opt} has to contain at least one of its incident vertices (Moreover these vertices are distinct as reasoned in the previous line). Hence

$$|C_{opt}| \geq |M|$$

From the algorithm, we have that $|C| = 2|M|$. Thus

$$|C| \leq 2|C_{opt}|$$

5.15 pts

a- Show that an undirected graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge (i.e., a unique edge of smallest cost) crossing the cut.

b- Show that the converse is not true by giving a counterexample.

Answer:

a) We will use the following facts proven in class

For every cut, if there are no ties, the lightest edge has to be a part of every MST.

For any cut, replacing the edge of a spanning tree crossing that cut with a lighter edge reduces the weight of the spanning tree

Consider any cut $(S, V \setminus S)$. The two facts imply that, to connect S and $V \setminus S$ we have to include the lightest edge in the MST and have no alternatives. Thus there is only one MST.

b) Let a, b, c, d be the set of vertices. There are 3 edges, $(a, b), (a, c)$ and (c, d) with weights 3,4 and 3 respectively. The graph is it self a tree and hence has a unique spanning tree. However, the cut $(\{a, c\}, \{b, d\})$ has two lightest edges crossing. Thus "unique MST" need not imply that every cut has a unique lightest edge.

Proof details (not expected to write these down)

Let $G = (V, E)$ be the graph. Define a subgraph $T = (V, F)$ where $F \subset E$ is an edge set that contains $e \in E$ if and only if e is the lightest edge of some cut. Here, T is in fact a tree because for every cut there is excatly one edge crossing the cut (unique lightest edge). And T spans the graph as every cut has at least one edge crossing it. Thus T is a spanning tree.

We claim that $w(T) < w(T')$ for every other tree T'.

Assume otherwise. Let T' be a minimal counterexample to our claim. Let e' be an edge in $T\prime$ that is not in T. Take any cut that has e' crossing it. As e' is not in E, we know that by replacing e' by the lightest edge of the cut, we contradict the minimality of T'.

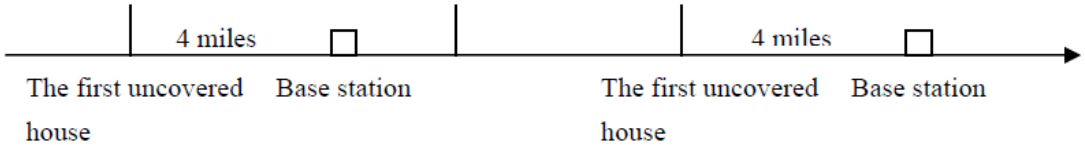
Thus there is a unique MST(which is T).

6. 10 pts

Consider a long country road with houses scattered very sparsely along it. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient algorithm that achieves this goal, using as few base stations as possible.

Answer:

The algorithm to locate the least number of base stations is given as follows:
Line 1 Start from the beginning of the road
Line 2 Find the first uncovered house on the road
Line 3 If there is no such a house, terminate this algorithm; otherwise, go to next line
Line 4 Locate a base station at 4 miles away after you find this house along the road
Line 5 Go to Line 2



This algorithm is O(n).
Now we prove its correctness. We want to show that this algorithm (denoted by S) does provide the smallest number of base stations. Let some other optimal solution be S'. Let DS(i) denote the number of homes covered by first i base stations in solution S. We now show DS(i)>DS'(i) for all i.
First, we can see DS(1)>=DS'(1) because S tries to put the first base station as faraway to the first home as possible.
Now given DS(i)>=DS'(i), we show that DS(i+1)>=DS'(i+1). This is true unless in S' the (i+1)th base station cover at least one more home than the (i+1)th base station in S further down the road. This is not possible because according to our greedy approach we have covered as far as possible in S. Thus this base station in S' much covered more than 8 miles. This is contradiction.
Therefore we proved the correctness of our algorithm.