CS570
Analysis of Algorithms
Fall 2014
Exam III

Name: _____

Student ID: _____

Email: _____

Wednesday Evening Section

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
Total	100	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

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

[/FALSE]
All integer programming problems can be solved in polynomial time.

[TRUE]
Fractional knapsack problem has a polynomial time greedy solution.

[/FALSE]
For any cycle in a graph, the cheapest edge in the cycle is in a minimum spanning tree.

[/FALSE]
Every decision problem is in NP.

[TRUE/]
If P=NP then P=NP=NP-complete.

[/FALSE]
Ford-Fulkerson algorithm can always terminate if the capacities are real numbers.

[/FALSE]
A flow network with unique edge capacities has a unique min cut.

[/FALSE]
A graph with non-unique edge weights will have at least two minimum spanning trees.

[TRUE/]
A sequence of $O(n)$ priority queue operations can be used to sort a set of n numbers.

[/FALSE]
If a problem X is polynomial time reducible to an NP-complete problem Y, then X is NP-complete.

2) 16 pts

Consider the **complete** weighted graph $G = (V, E)$ with the following properties

a. The vertices are points on the X-Y plane on a regular $n \times n$ grid, i.e. the set of vertices is given by $V = \{(p, q) | 1 \leq p, q \leq n\}$, where p and q are integer numbers.

b. The edge weights are given by the usual Euclidean distance, i.e. the weight of the edge between the nodes (i, j) and (k, l) is $\sqrt{(k-i)^2 + (l-j)^2}$.

Prove or disprove: There exists a minimum spanning tree of G such that every node has degree at most two.

Solution:

There does exist an MST of G with every node having degree ≤ 2. One such MST is obtained by the edges joining **nodes** in the following order: $(1,1)$—$(1,2)$—$(1,3)$--…--$(1, n-1)$—$(1, n)$—$(2, n)$—$(2, n-1)$--…--$(2,2)$—$(2,1)$—$(3,1)$--...—(n, n).

Let us denote the above set of edges by E. To prove that E indeed forms an MST, we need to show that it forms a spanning tree and that it is of minimal weight.

***E* forms a spanning tree:** It is clear that all nodes in the above sequence are distinct. This implies that there are no cycles since any cycle, when written out as a sequence of connected nodes, would necessarily repeat the starting node at the end. It is also clear that all nodes of the graph are covered in the above sequence. In particular, nodes $(i, 1)$, $(i, 2)$, …, (i, n) are connected in that order for odd integers i and in the reverse order for even integers i. Hence, E forms a spanning tree and has $n^2 - 1$ edges.

***E* has minimal weight:** It is easy to see that every edge in G has at least unit weight, since weight of edge between distinct nodes is minimal for edges between node pairs of the form $\{(i, j), (i+1, j)\}$ or $\{(i, j), (i, j+1)\}$, evaluating to an edge weight of 1. Since all edges in E are of this form, all edges in E have unit weight and total weight of E is equal to $n^2 - 1$. Finally, any spanning tree of G has exactly $n^2 - 1$ edges, so the weight of the MST must be at least $n^2 - 1$, thus proving that weight of E is minimal.

3) 16 pts

You are trying to decide the best order in which to answer your CS-570 exam questions within the duration of the exam. Suppose that there are n questions with points $p_1, p_2, \dots, p_n$ and you need time t_i to completely answer the i^{th} question. You are confident that all your completely answered questions will be correct (and get you full credit for them), and the TAs will give you partial credit for an incompletely answered question, in proportion to the time you spent on that question. Assuming that the total exam duration is T, give a greedy algorithm to decide the order in which you should attempt the questions and prove that the algorithm gives you an optimal answer.

Solution: This is similar to the fractional knapsack problem. The greedy algorithm is as follows. Calculate $\frac{p_i}{t_i}$ for each $1 \leq i \leq n$ and sort the questions in descending order of $\frac{p_i}{t_i}$. Let the sorted order of questions be denoted by $s(1), s(2), \dots, s(n)$. Answer questions in the order $s(1), s(2), \dots$ until $s(j)$ such that $\sum_{k=1}^{j} t_{s(k)} \leq T$ and $\sum_{k=1}^{j+1} t_{s(k)} > T$. If $\sum_{k=1}^{j} t_{s(k)} = T$ then stop, otherwise partially solve the question $s(j+1)$ in the time remaining.

We'll use induction to prove optimality of the above algorithm. The induction hypothesis $P(l)$ is that there exists an optimal solution that agrees with the selection of the first l questions by the greedy algorithm.

Inductive Step: Let $P(1), P(2), \dots, P(l)$ be true for some arbitrary l, i.e. the set of questions $\{s(1), s(2), \dots, s(l)\}$ are part of an optimal solution O. It is clear that O should also be optimal with respect to the remaining questions $\{1,2, \dots, n\} \setminus \{s(1), s(2), \dots, s(l)\}$ in the remaining time $T - \sum_{k=1}^{l} t_{s(k)}$. Since $P(1)$ is true, there exists an optimal solution, not necessarily distinct from O, that selects the question with the largest value of $\frac{p_i}{t_i}$ in the remaining set of questions, i.e. the question $s(l+1)$ is selected. Since $s(l+1)$ is also the choice of the proposed greedy algorithm, $P(l+1)$ is proved to be true.

Induction Basis: To show that $P(1)$ is true, we proceed by contradiction. Assume $P(1)$ to be false. Then $s(1)$ is not part of any optimal solution. Let $a \neq s(1)$ be a partially solved question in some optimal solution O' (if O' does not contain any partially solved questions then we take a to be any arbitrary question in O'). Taking time $\min(t_a, T, t_{s(1)})$ away from question a and devoting to question $s(1)$ gives the improvement in points equal to $\left(\frac{p_{s(1)}}{t_{s(1)}} - \frac{p_a}{t_a}\right) \min(t_a, T, t_{s(1)}) \geq 0$ since $\frac{p_i}{t_i}$ is largest for $i = s(1)$. If the improvement is strictly positive then it contradicts optimality of O' and implies the truth of $P(1)$. If improvement is 0, then a can be switched out for $s(1)$ without affecting optimality and thus implying that $s(1)$ is part of an optimal solution which in turn means that $P(1)$ is true.

4) 16 pts

An Edge Cover on a graph $G = (V; E)$ is a set of edges $X \subseteq E$ such that every vertex in V is incident to an edge in X. In the **Bipartite** Edge Cover problem, we are given a bipartite graph and wish to find an Edge Cover that contains $\leq k$ edges. Design a polynomial-time algorithm based on network flow (max flow or circulation) to solve it and justify your algorithm.

Solution:

If the network contains isolated node, then the problem is trivial since there is no way to do edge cover. Now we only consider the case that each node has at least 1 incident edge.

i) The goal of edge cover is to choose as many edges as possible which cover 2 nodes. You can find this subset of edges by running Bipartite Matching on the original graph, and taking exactly the edges which are in the matching. (Equivalently, you can set capacity of each edge in the graph as 1. Set a super source node s connecting each "blue" node with edge capacity 1 and a super destination t connecting each "red" node with edge capacity 1. Then run max-flow to get the subset of edges connecting 2 nodes in G)

ii) What remains is to cover the remaining nodes. Since you can only cover a single node (of those remaining) with each selected edge, simply choose an arbitrary incident edge to each uncovered node.

iii) Set the set of edges you choose in the above two steps as set X. Count the total number of edges in X and compare the size with k.

Proof:

It is obvious that X is an edge cover. The remaining part is to show that X contains the minimum number of edges among all possible edge covers.

Denote the number of edges we find in step i) as x1; denote the number of edges we find in step ii) as x2.

Then we have x1 * 2 + x2 = |V|.

Consider an arbitrary edge cover set Y. Suppose Y contains y1 edges, each of which is counted as the one covering 2 nodes. Suppose Y contains y2 edges, each of which is counted as the one covering 1 nodes. (We can ignore the edges covering zero nodes, because we can delete those edges from Y without affecting the coverage)

Then we have y1 * 2 +y2 = |V|.

Here x1 must be the maximum number of edges that covers 2 nodes in the bipartite graph, because we do bipartite matching in G (max-flow in G' including s and t). Therefore, we have x1 >= y1.

Then we have:
x1+x2 = |V| - x1 = y1* 2 +y2 –x1 = y1 + y2 - (x1-y1) <= y1 + y2.

The above algorithm gives the minimum number of edges for covering the nodes.

5) 16 pts

Imagine starting with the given decimal number n, and repeatedly chopping off a digit from one end or the other (your choice), until only one digit is left. The square-depth SQD(n) of n is defined to be the maximum number of perfect squares you could observe among all such sequences. For example, SQD(32492) = 3 via the sequence

32492 → **3249** → **324** → 24 → **4**

since 3249, 324, and 4 are perfect squares, and no other sequence of chops gives more than 3 perfect squares. Note that such a sequence may not be unique, e.g.

32492 → **3249** → 249 → **49** → **9**

also gives you 3 perfect squares, viz. 3249, 49, and 9.

Describe an efficient algorithm to compute the square-depth SQD(n), of a given number n, written as a d-digit decimal number $a_1 a_2 \ldots a_d$. Analyze your algorithm's running time. Your algorithm should run in time polynomial in d. You may assume the availability of a function IS_SQUARE(x) that runs in constant time and returns 1 if x is a perfect square and 0 otherwise.

Solution:

We can solve this using dynamic programming.

First, we define some notation: let nij = ai . . . aj. That is, nij is the number formed by digits i through j of n. Now, define the subproblems by letting D[i, j] be the square-depth of nij .

The solution to D[i, j] can be found by solving the two subproblems that result from chopping off the left digit and from chopping off the right digit, and adding 1 if nij itself is square. We can express this as a recurrence:

D[i, j] = max(D[i + 1, j], D[i, j − 1]) + IS-SQUARE(nij)

The base cases are D[i, i] = IS-SQUARE(ai), for all $1 \leq i \leq d$. The solution to the general problem is SQD(n) = D[1, d].

There are $\Theta(d^2)$ subproblems, and each takes $\Theta(1)$ time to solve, so the total running time is $\Theta(d^2)$.

*** Some students did a greedy approach to solve this problem which is not true for this problem. In each step of the program they explore removing which digit results in a perfect square number, however it might be the other non-removed digit that will produce more squares in the future steps.

6) 16 pts

The Longest Path is the problem of deciding whether a graph has a simple path of length greater or equal to a given number k.

a) Show that the Longest Path problem is in NP (2 pts)

b) Show how the Hamiltonian Cycle problem can be reduced to the Longest Path problem. (14 pts)

Note: You can choose to do the reduction in b either directly, or use transitivity of polynomial time reduction to first reduce Hamiltonian Cycle to another problem (X) and then reduce X to Longest Path.

a) Polynomial length certificate: ordered list of nodes on a path of length >= k

polynomial time Certifier:

check that nodes do not repeat in O(n log n)

check that the length of the path is at least k in O(n)

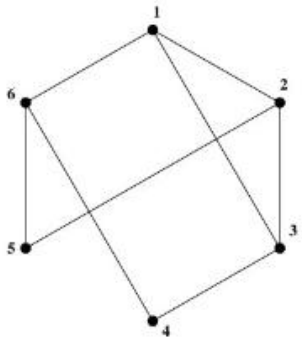
check that there are edges between every two adjacent nodes on the path in O(n)

check that the length of the path is at least k in O(n)

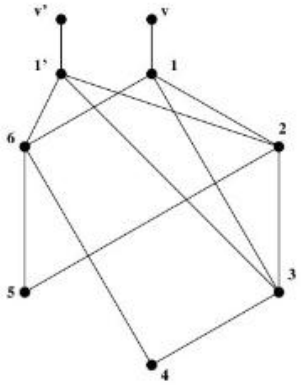
b) Two possible solutions:

I. To reduce directly from Hamiltonian Cycle

Given a graph G= (V,E) we construct a graph G' such that G contains a Ham cycle iff G' contains a simple path of length at least N+2. This is done by choosing an arbitrary vertex u in G and adding a copy, u', of it together with all its edges. Then add vertices v and v' to the graph and connect v with u and v' to u'. We give a cost of 1 to all edges in G'. See figure below:



G



G'

Proof:

A) If there is a HC in G we can find a simple path of length n+2 in G': This path starts at v' goes to U' and follows the HC to u and then to v. The length is n+2

B) If there is a simple path of length n+2 in G', there is a HC in G: This path must

include nodes v' and v because there are only n nodes in G and a simple path of length n+2 must include v and v'. Moreover, v and v' must be the two ends of this path, otherwise, the path will not be a simple path since there is only one way to get to v and v'. So, to find the HC in G, just follow the path from u' to u.

II. To reduce using Hamiltonian Path

First reduce Ham Cycle to Ham Path (very similar to the above reduction)

Then reduce Ham Path to Longest path. This is very straightforward. To find out of there is a Ham Path in G you can assign weights of 1 to all edges and ask the blackbox if there is a path of length at least n in G.

Additional Space