CS570
Analysis of Algorithms
Spring 2015
Exam III

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	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.

[TRUE]
If SAT $\leq_P$ A, then A is NP-hard.

[FALSE]
If a problem X can be reduced to a known NP-hard problem, then X must be NP-hard.

[TRUE]
If P equals NP, then NP equals NP-complete.

[FALSE]
Let X be a decision problem. If we prove that X is in the class NP and give a poly-time reduction from X to Hamiltonian Cycle, we can conclude that X is NP-complete.

[TRUE]
The recurrence $T(n) = 2T(n/2) + 3n$, has solution $T(n) = \theta(n \log(n^2))$.

[FALSE]
On a connected, directed graph with only positive edge weights, Bellman-Ford runs asymptotically as fast as Dijkstra.

[TRUE]
Linear programming is at least as hard as the Max Flow problem in a flow network.

[TRUE]
If you are given a maximum s-t flow in a graph then you can find a minimum s-t cut in time O(m) where m is the number of the edges in the graph.

[TRUE]
Fibonacci heaps can be used to make Dijkstra's algorithm run in O(|E| + |V| log|V|) time on a graph G=(V,E)

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

2) 16 pts
Given a graph G=(V, E) with an even number of vertices as the input, the HALF-IS problem is to decide if G has an independent set of size |V|/2. Prove that HALF-IS is in NP-Complete.

Solution:
Given a graph $G(V, E)$ and a certifier $S \subset V, |S| = |V|/2$, we can verify if no two nodes are adjacent in polynomial time ($O(|S|^2) = O(|V|^2)$). Therefore $HALF - IS \in NP$.
We prove the NP-Hardness using a reduction of the NP-complete problem Independent set problem (IS) to $HALF - IS$. Consider an instance of IS, which asks for an independent set $A \subset V, |A| = k$, for a graph $G(V, E)$, such that no two pair of vertices in A are adjacent to each other.

(i) If $k = \frac{|V|}{2}$, IS reduces to HALF-IS.

(ii) If $k < \frac{|V|}{2}$, then add m new nodes such that $k + m = (|V| + m)/2$, i.e., $m = |V| - 2k$. Note that the modified set of nodes V' has even number of nodes. Since the additional nodes are all disconnected from each other, they form a subset of independent set. Therefore, the new graph $G'(V', E')$ where $E' = E$ has an independent-set of size $\frac{|V'|}{2}$ if and only if $G(V, E)$ has an independent set of size k.

(iii) If $k > \frac{|V|}{2}$, then again add $m = |V| - 2k$ new nodes to form the modified set of nodes V'. Connect these new nodes to all the other $|V| + m - 1$ nodes. Since these m new nodes are connected to every other none of them should belong to an independent set. . Therefore, the new graph $G'(V', E)$ has an independent-set of size $\frac{|V'|}{2}$ if and only if $G(V, E)$ has an independent set of size k.

Hence, any instance of IS $(G(V, E), k)$, can be reduced to an instance of HALF-IS $(G'(V', E'))$.

$$IS \leq_P HALF - IS.$$

$HALF - IS \in NP$ and $HALF - IS \in NP - Hard \Rightarrow HALF - IS \in NP - complete.$

3) 16 pts

A variant on the decision version of the subset sum problem is as follows: Given a set of n integer numbers A = $\{a_1, a_2, \ldots, a_n\}$ and a target number *t*. Determine if there is a subset of numbers in A whose product is precisely *t*. That is, the output is *yes* or *no*. Describe an algorithm (and provide pseudo-code) to solve this problem, and analyze its complexity.

Solution:

Solution: *Use dynamic programming*:

Let S[i,j] shows if there exists a subset of $\{ a_1, a_2, \ldots, a_i \}$ that add up to j, where $0 \leq j \leq t$.

S[i,j] is true or false.

Initialize: S[0,0] = true; S[0,j] = false if $j \neq 0$

Recurrence formula:

S[i,j] = S[i-1,j] OR S[i-1, j/a_i]

Output is S[n,t]

Complexity is O(tn)

4) 16 pts

We've been put in charge of a phone hotline. We need to make sure that it's staffed by at least one volunteer at all times. Suppose we need to design a schedule that makes sure the hotline is staffed in the time interval $[0, h]$. Each volunteer i gives us an interval $[s_i, f_i]$ during which he or she is willing to work. We'd like to design an algorithm which determines the minimum number of volunteers needed to keep the hotline running. Design an efficient greedy algorithm for this problem that runs in time $O(n \log n)$ if there are n student volunteers. Prove that your algorithm is correct.
You may assume that any time instance has at least one student who is willing to work for that time.

Solution:
Algorithm: Initially, select the student who can start at or before time 0 and whose finish time is the latest. For each subsequent volunteer, select the one whose start time is no later than the finish time of the last selected volunteer and whose finish time is the latest. Keep selecting volunteers sequentially in this way until the interval [0, h] is covered.

The running time is O(n log n):
First, sort the start time of all the volunteers takes time O(n log n); then, searching and selecting the valid successive volunteers with the latest finish time takes O(n) time in total (each volunteer is checked at most once).

Proof:
Let $g_1, g_2, \ldots, g_m$ be the sequence of volunteers we selected according to the greedy algorithm; let $p_1, p_2, \ldots, p_k$ be the sequence of selected volunteers of an optimal solution.

Clearly, for any feasible solution, we must have someone who can starts before or at time 0. So in the above two solutions: g_1 and p_1 must start at or before time 0. According to our algorithm, we have $f_{g1} \geq f_{p1}$. By replacing p_1 by g_1 in the optimal solution, we get another solution: $g_1, p_2, \ldots, p_k$, which uses k volunteers and cover the interval [0, h] and therefore is another optimal solution.

Induction hypothesis: assume that our greedy solution is the same as an optimal solution up to the r-1 th selected volunteer, i.e., $g_1, \ldots, g_{r-1}, p_r, \ldots, p_k$ is an optimal solution. With the same argument: $f_{g\_r} \geq f_{p\_r}$ by replacing p_r by g_r in the optimal solution, we get another solution $g_1, \ldots, g_r, p_{r+1}, \ldots, p_k$, which is also optimal.

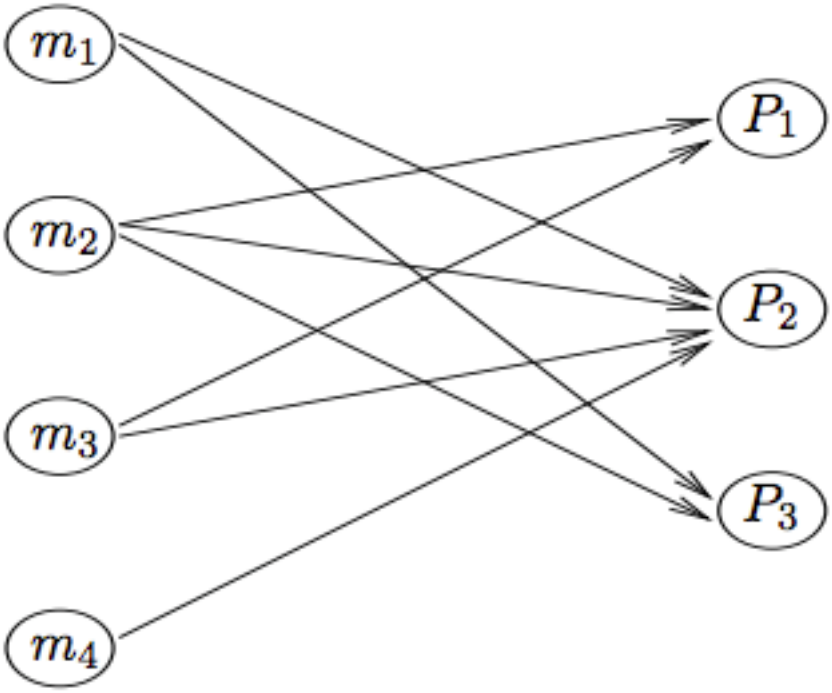
By induction: it follows that $g_1, g_2, \ldots, g_m$ is an optimal solution and m=k.

5) 16 pts

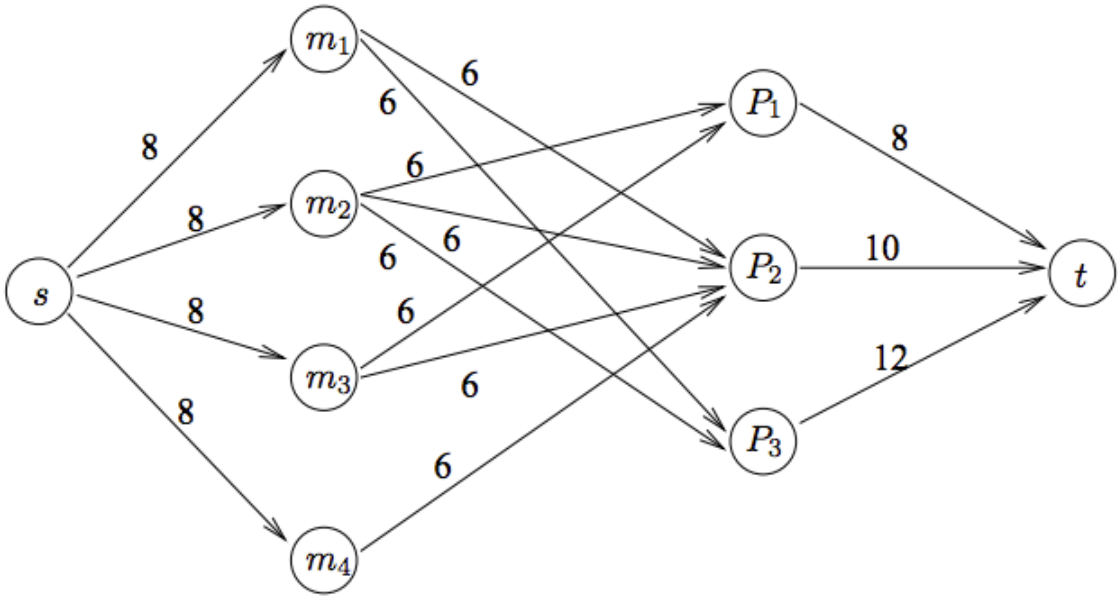
A software house has to handle 3 projects, P_1, P_2, P_3, over the next 4 months. P_1 can only begin after month 1, and must be completed within month 3. P_2 and P_3 can begin at month 1, and must be completed, respectively, within month 4 and 2. The projects require, respectively, 8, 10, and 12 man-months. For each month, 8 engineers are available. Due to the internal structure of the company, at most 6 engineers can be working, at the same time, on the same project. Determine whether it is possible to complete the projects within the time constraints. Describe how to reduce this problem to the problem of finding a maximum flow in a flow network and justify your reduction.

Solution

We build a product network where months and projects are represented by, respectively, month-nodes m_1, m_2, m_3, m_4 and project-nodes P_1, P_2, P_3. Each (i, j) edge denotes the possibility of allocating man-hours of month i to project j. For instance, since project P_1 can only begin after month 1 and must be completed before month 3, only arcs outgoing from m_2, m_3 are incident in P_1.



We add to super nodes s, t, denoting the source and sink of the flow that represents the allocation of men-hours.



All edges outgoing from s have capacity 8, equivalent to the number of available engineers per month. All edges connecting month-nodes to project-nodes have capacity 6, as no more than 6 engineers can work on the same project in the same month. All edges incident in t have capacity equivalent to the number of man-months needed to complete the project. Since all capacities are integer, the maximum flow will be integer as well. To check whether all projects can be completed within the time limits, it suffices to check whether the network admits a feasible flow of value 8 + 10 + 12 = 30.

6) 16 pts

There are n people and n jobs. You are given a cost matrix, C, where C[i][j] represents the cost of assigning person i to do job j. You want to assign all the jobs to people and also only one job to a person. You also need to minimize the total cost of your assignment. Can this problem be formulated as a linear program? If yes, give the linear programming formulation. If no, describe why it cannot be formulated as an LP and show how it can be reduced to an integer program.

Solution:

We need a 0,1 decision variable to solve the problem and therefore we need to formulate this as an integer program. Below is a formulation of integer program.

Let x_{ij} = 1, if job j is assigned to worker i.

= 0, if job j is not assigned to worker i.

Objective function: Minimize

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Constraints:

$$\sum_{i=1}^{n} x_{ij} = 1, \quad for\ j = 1,2,\dots,n$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad for\ i = 1,2,\dots,m$$

$$x_{ij} = 0\ or\ 1$$

Additional Space

Additional Space