NAME: ______________________

CSE 421
Introduction to Algorithms
Final Exam Winter 2003

P. Beame

17 March 2003

DIRECTIONS:

- Answer the problems on the exam paper.
- Open book. Open notes.
- If you need extra space use the back of a page
- You have 1 hour and 50 minutes to complete the exam.
- Please do not turn the page until you are instructed to do so.
- Good Luck!

1	/20
2	/20
3	/20
4	/20
5	/20
6	/20
Total	/120

1. (20 points, 2 each) For each of the following problems circle **True** or **False**. You do not need to justify your answer.

   (a) If f and g are two different flows on the same flow graph (G, s, t) and if $\nu(f) \geq \nu(g)$ then every edge e in G has $f(e) \geq g(e)$. **True** **False**

   (b) If f is a maximum flow on a flow graph $(G = (V, E), s, t)$ and B is the set of vertices in V that can reach t in G_f then $(V - B, B)$ is a minimum capacity s-t cut in G. **True** **False**

   (c) If f is a maximum flow on a flow graph (G, s, t) and (S, T) is a minimum capacity s-t cut in G then *every* edge e having endpoints on different sides of (S, T) has $f(e)$ equal to the capacity of e. **True** **False**

   (d) If problem B is NP-hard and $A \leq_P B$ then A is NP-hard. **True** **False**

   (e) If problem A is NP-hard and $A \leq_P B$ then B is NP-hard. **True** **False**

   (f) If $P \neq NP$ then every problem in NP requires exponential time. **True** **False**

   (g) If problem A is in P then $A \leq_P B$ for every problem B in NP. **True** **False**

   (h) If G is a weighted graph with n vertices and m edges that does *not* contain negative-weight cycle, then the iteration of the Bellman-Ford algorithm will reach a fixed point in at most $n - 1$ rounds. **True** **False**

   (i) If G is a weighted graph with n vertices and m edges that *does* contain negative-weight cycle, then for *every* vertex v in G the shortest path from v to t in G containing n edges is strictly shorter than the shortest path from v to t in G containing $n - 1$ edges. **True** **False**

   (j) Given a connected weighted undirected graph $G = (V, E)$ having integer weights from $\{1, \ldots, k\}$, a minimum spanning tree in G may be found in $O(mk)$ time using Prim's algorithm. **True** **False**

(a) (10 points) Draw the residual graph G_f.

(b) (3 points) What augmenting path in this graph would result in the greatest increase in flow value? (List the names of the vertices on this path in order.)

(c) (7 points) On the diagram below, indicate the new flow values resulting from augmenting along the path you found in part (b).

3. (20 points) The HITTING SET problem asks, given a universe U, a collection of sets $E_1, E_2, \ldots, E_m \subseteq U$, and an integer k, whether there is a subset H of U of size at most k such that each set E_j for $j = 1, \ldots, m$ contains at least one element of H.

   (a) Write out the outline of what you would need to prove to show that the HITTING SET problem is NP-complete using the fact that VERTEX COVER is NP-complete.

   (b) Fill in the details of your NP-completeness proof for HITTING SET.

4. (20 points) You are given a sequence of n bits $x_1, \ldots, x_n \in \{0, 1\}$. Your output is to be either

- any i such that $x_i = 1$ or
- the value 0 if the input is all 0's.

The only operation you are allowed to use to access the inputs is a function **Group-Test** where

$$\textbf{Group-Test}(i, j) = \begin{cases} 1 & \text{if some bit in } x_i, \ldots, x_j \text{ has value 1} \\ 0 & \text{if all bits in } x_i, \ldots, x_j \text{ have value 0} \end{cases}$$

(Historical Note: In World War I, the army was testing recruits for syphillis which was rare but required a time-consuming though accurate blood test. They realized that they could pool the blood from several recruits at once and save time by eliminating large groups of recruits who didn't have syphillis.)

(a) (15 points) Design a divide and conquer algorithm to solve the problem that uses only $O(\log n)$ calls to **Group-Test** in the worst case. Your algorithm should *never* access the x_i directly.

(b) (5 points) Briefly justify your bound on the number of calls.

5. (20 points) You are given a *directed acyclic graph* $G = (V, E)$ and a node node $t \in V$. Design a linear time algorithm to compute for each vertex $v \in V$ the *number* of different paths from v to t in G. Analyze its running time in terms of $n = |V|$ and $m = |E|$.

6. (20 points) Rotciv Nap has designed a subroutine for multiplying arbitrary (possibly) non-square matrices together. Multiplying an $m \times n$ matrix by an $n \times p$ matrix takes this subroutine time $COST(m, n, p)$ and he also has code that will very efficiently compute $COST(m, n, p)$ for you. If you had 3 matrices, A, B, and C and you wanted to compute the matrix ABC there are two ways of computing it using Nap's subroutine: Compute $(AB)C$, i.e., compute AB first and then multiply by C, or compute $A(BC)$, i.e., compute BC first and then multiply by A. One of these orders might be cheaper than the other based on the dimensions of the matrices.

   The *Optimal Matrix Chaining* problem aims to figure out, given a sequence of $m_0, m_1, m_2, \ldots .m_n$ of integers, what would be the best order to multiply a sequence of matrices $A_1 A_2 \cdots A_n$ using the Nap subroutine, where each A_i is an $m_{i-1} \times m_i$ matrix.

   You observe that the size of the matrix $A_i A_{i+1} \ldots A_j$ is just $m_{i-1} \times m_j$ and since there must be some *last* multiplication that splits the matrix sequence at some position k if $n > 1$, the optimal time is

$$OPT(m_0, \ldots, m_n) = \min_{1 \leq k < n} \{OPT(m_0, \ldots, m_k) + OPT(m_k, \ldots, m_n) + COST(m_0, m_k, m_n)\}$$

   For $n = 1$ the multiplication time is 0. It doesn't take you long to figure out that coding this up directly as a recursive algorithm would take exponential time.

   (a) Design a dynamic programming algorithm based on these observations that runs in polynomial time and computes the optimal time for the matrix chaining problem using the Nap code. (*Note: You only need to figure out what the best time is. You don't need to actually find the complete order.*)

   (b) What is the running time of your algorithm if each call to the $COST$ function takes time 1?

(You can also use space on the next page for your answer)