CS 170
Spring 2000 **Midterm #2** **Demmel / Shewchuk**

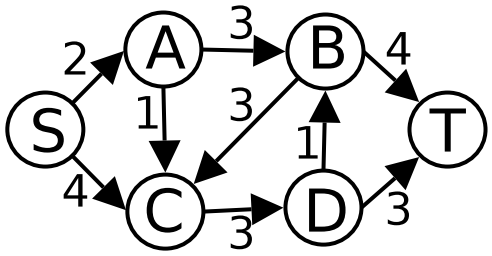
This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permitted a one-page, double-sided set of notes.

Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

Do not open your exam until you are told to do so!

1. (12 points) **Network Flows**

   In the flow network illustrated below, each directed edge is labeled with its capacity. We are using the Ford-Fulkerson algorithm to find the maximum flow. The first augmenting path is S-A-C-D-T, and the second augmenting path is S-A-B-C-D-T.



   (a) (6 points) Draw the residual network after we have updated the flow using these two augmenting paths (in the order given).

   (b) (4 points) List all of the augmenting paths that could have been chosen for the third augmentation step.

   (c) (2 points) What is the numerical value of the maximum flow? Draw a dotted line through the original graph to represent the minimum cut.

2. (16 points) **True or False?**

   Each true/false question is worth two points. You will **lose** two points for a wrong answer, so only answer if you think you are likely to be right.

   (a) $\log(c\sqrt{n}) \in O(\log\sqrt{n})$ for all $c > 0$.

   (b) $\log^* \log n \in \Theta(\log^* n)$.

   (c) A *grandchild* in a forest-based disjoint-set data structure is any node that has a grandparent. (In other words, neither the node nor its parent is a root.) If we use union-by-rank, a single union operation cannot increase the total number of grandchildren by more than $n/2$, where n is the total number of nodes.

   (d) Let f be a non-empty file that uses 3-bit fixed-length codewords for the characters a-h. The file cannot use any character other than these eight. True or false: Huffman coding can always compress any such file (i.e., produce an encoding with fewer bits). (Don't count the bits needed to express the mapping from bits to alphabet.)

   (e) If A is an NP-complete problem, and problem B reduces (polynomially) to A, then B is an NP-complete problem.

   (f) The problem of finding the length of the shortest path between two nodes in a directed graph (which we would normally solve using Dijkstra's algorithm) can be expressed as a linear program. (An ordinary, continuous linear program; we're not talking about integer linear programs.) Assume that there is only one shortest path.

   (g) Consider a linear program with three variables (dimensions). If x is an optimal point of the linear program, then x must be a vertex of the polyhedron formed by the constraints (inequalities).

   (h) Imagine we have a computer with an infinite amount of RAM, in which any memory address can be accessed in constant time. There exists a computer program of infinite length that solves SAT in linear time.

3. (12 points) **NP-Completeness**

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 all the usual constraints of flow problems. The modified problem asks us to find the maximum saturated flow.

(a) (1 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.

(b) (3 points) Show that MAX-SATURATED-FLOW is in NP.

(c) (8 points) Show that MAX-SATURATED-FLOW is NP-hard. Hint: the easiest reduction is *not* from one of the NP-complete *graph* problems you know. What other NP-complete problems do you know?

4. (15 points) **Dynamic Programming**

Consider the *Maximum Alternating Sum Subsequence (MASS)* problem. Given a sequence $S = [x_1, x_2, \ldots, x_n]$ of positive integers, find the subsequence $A = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$, where $i_1 < i_2 < \cdots < i_k$, that maximizes the alternating sum

$$x_{i_1} - x_{i_2} + x_{i_3} - x_{i_4} + \cdots \pm x_{i_k}.$$

For example, if $S = [4, 9, 2, 4, 1, 3, 7]$, the MASS is $A = [9, 2, 4, 1, 7]$ which evaluates to $9 - 2 + 4 - 1 + 7 = 17$. If $S = [7, 6, 5, 4, 3, 2, 1]$, the MASS is $A = [7]$. Clearly, the length of the MASS depends on the actual values in S. Assume that the length of S is at least one, and all its elements are integers greater than zero.

The following questions ask you to develop a dynamic programming algorithm to find the *value* of the MASS (but not the actual subsequence) for a given sequence. We would like a solution with optimal running time, but you will only lose a few points if you have a correct, suboptimal algorithm.

(a) (3 points) Describe the subproblems you will need to solve. How many tables (of subproblem solutions) do you need? What is the dimension of the table(s)? What do the table entries represent? (Hint: it helps to treat subsequences of even and odd lengths separately.)

(b) (7 points) Write a recursion for the values your algorithm will fill into the table(s). Also write the base case(s); i.e. the table value(s) that bootstrap(s) the recursion.

(c) (4 points) Write iterative, non-recursive pseudocode for your algorithm.

(d) (1 point) What is the running time of your algorithm?