UC Berkeley—CS 170 Midterm 2
Lecturer: David Wagner April 10, 2003

Midterm 2 for CS 170

PRINT your name: _____, _____
(last) (first)

SIGN your name: _____

PRINT your username on `cory.eecs`: _____

WRITE your section number (e.g., 101–108): _____

This exam is open-book, open-notes. Calculators are permitted. Do all your work on the pages of this examination. If you need more space, you may use the reverse side of the page, but try to use the reverse of the same page where the problem is stated.

You have 80 minutes. There are 3 questions, of varying credit (100 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

In this exam, I only care about asymptotic running times. You may freely ignore constant factors.

This exam consists of only short-answer and true/false questions. You do not need to justify your answer on any question on this exam. On the true/false questions, circle either TRUE or FALSE.

Do not turn this page until the instructor tells you to do so.

Problem 1. [Quick answer] (30 points)

(a) TRUE or FALSE: In a weighted graph G, the min-cut is always unique.

(b) TRUE or FALSE: In a weighted graph G, the minimum spanning tree is always unique.

(c) TRUE or FALSE: There is a disjoint sets algorithm so that every sequence of n MAKE-SET, n UNION, and m FIND operations can be made to run in $O((m+n)\lg\lg n)$ time, total.

(d) TRUE or FALSE: There is a disjoint sets algorithm so that every sequence of n MAKE-SET, $O(1)$ UNION, and m FIND operations can be made to run in $O(m+n)$ time, total.

(e) TRUE or FALSE: Any dynamic programming algorithm that solves N subproblems in the process of computing its final answer must run in $\Omega(N)$ time.

(f) TRUE or FALSE: Every Horn formula with at most n clauses and at most n variables per clause can be solved in $O(n)$ time.

(g) TRUE or FALSE: The following is an example of a Horn formula:
$((\overline{x_1} \wedge \overline{x_3} \wedge \overline{x_5}) \Rightarrow \overline{x_2}) \wedge (\overline{x_2} \Rightarrow \overline{x_1}) \wedge \overline{x_5}$.

(h) TRUE or FALSE: The following is an example of a Horn formula:
$((x_1 \wedge x_3 \wedge x_5) \Rightarrow x_2) \wedge (x_2 \Rightarrow x_1) \wedge x_5$.

(i) The letters 'A', 'B', 'C', 'D', and 'E' occur in some language with frequencies $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, and $\frac{1}{16}$ (respectively). (Assume each letter is chosen independently, so that, for instance, "CA" has probability $\frac{1}{8} \times \frac{1}{2} = \frac{1}{16}$.)

If you use a variable-length prefix-free compression scheme for this language, what is best achievable rate? (Remember that the rate is the average number of bits needed per letter.)

Best rate = _____ bits per letter.

Problem 2. [True or false] (30 points)

Let $G = (V, E)$ be an undirected graph, where the vertex set is given by $V = \{1, 2, \ldots, n\}$. Let $m = |E|$. We say that G forms an *almost-tree* if there is some edge e whose addition to G makes the resulting graph a tree. In other words, G is an almost-tree if there exists e so that $G' = (V, E \cup \{e\})$ is a tree.

First, I want an *efficient* algorithm to check whether the graph G forms an almost-tree, when G is provided as input in adjacency list format.

(a) What is the worst-case running time of your best algorithm for this problem? State your answer in terms of n and m. Use $\Theta(\cdot)$ notation.

(b) If you had the opportunity to give a friend in CS170 any hint you like on how to solve this problem, but you were *allowed only 10 words or less*, what would you say?

For your second task, the edges of G will be presented to you one at a time. Let $e_1, e_2, \ldots, e_m$ denote the sequence of edges, in the same order as they are presented to you. After receiving each edge e_j (and before receiving e_{j+1}), your algorithm must say either "Yes" or "No" according to whether $G_j = (V, \{e_1, \ldots, e_j\})$ is an almost-tree or not.

The running time of your algorithm is the total time it takes you to process all m of these edges. For this second task, you may assume that $m = \Theta(n)$.

(A) What is the worst-case running time of your best algorithm for this problem? State your answer in terms of m. Use $\Theta(\cdot)$ notation.

(B) If you had the opportunity to give a friend in CS170 any hint you like on how to solve this problem, but you were *allowed only 10 words or less*, what would you say?

Problem 3. [Optimal traffic enforcement] (40 points)

You're the chief of police at Smallville, US. The recession has forced you to recoup a budget shortfall by trying to maximize the number of speeding tickets you hand out on Route 80, which passes through town. You have an unlimited number of police officers available. There are n locations on Route 80 (let's call them position 1, 2, ..., n) where an officer can hide and try to catch speeders.

Of course, some locations are better than others—some are especially well-hidden, some are on a downhill, and so on—and you've calculated, for each i, the number $T[i]$ of tickets an officer stationed at position i can expect to issue. At the same time, you've decided to avoid stationing two officers next to each other (i.e., at positions i and $i+1$), so motorists who notice one officer and slow down won't affect the number of tickets issued by other officers. You want to optimize the placement of your officers to maximize the number of tickets issued.

More precisely, you're given an array $T[1..n]$ of positive integers. Your goal is to maximize the value $\mathcal{M} = A[1] \cdot T[1] + \ldots + A[n] \cdot T[n]$, subject to the following constraints on $A[1..n]$:

- $A[i]$ is 1 if an officer is stationed at position i, and 0 otherwise. No more than one officer per position allowed, i.e., we require $A[i] \in \{0, 1\}$ for all i.
- No two officers may be stationed at consecutive positions on Route 80, i.e., $A[i] + A[i+1] \leq 1$ for all i.

Your goal is to build an *efficient* algorithm that, given $T[1..n]$, finds the maximal $\mathcal{M}$ by using DYNAMIC PROGRAMMING.

The class of subproblems: Each $j \in \{1, 2, \ldots, n\}$ identifies a subproblem, as follows:

> Let $M[j]$ denote the maximum number of tickets you can issue by placing officers at some subset of locations $1, \ldots, j$.

In other words, $M[j]$ is the maximum value of $A[1] \cdot T[1] + \ldots + A[j] \cdot T[j]$ over all $A[1..j]$ that satisfy the above constraints.

(a) What order should you solve the subproblems in?

(b) Write a recurrence relation for $M[j]$ in terms of the solutions to easier subproblems (i.e., subproblems that have been solved before $M[j]$).

$M[j] =$ _____

(c) Suppose we use your recurrence from parts (a) and (b) to build a dynamic programming algorithm in the obvious way. What will its worst-case running time be? State your answer in terms of n. Use $\Theta(\cdot)$ notation.

Now I'll change the problem. I'm now going to limit you to a total of k officers. However, I'll remove the prohibition against stationing officers at consecutive locations; now you can place officers at any subset of locations you like (e.g., you're allowed to have $A[i] = A[i+1] = 1$). In other words, the goal is to maximize $\mathcal{M} = A[1] \cdot T[1] + \ldots + A[n] \cdot T[n]$ subject to

- $A[i] \in \{0, 1\}$ (same as before).
- No more than k officers in all are used, i.e., $A[1] + \ldots + A[n] \leq k$ (*this part is new*).

Your goal is to build an *efficient* algorithm that, given $T[1..n]$, finds the maximal $\mathcal{M}$.

(A) One could imagine many possible ways to solve this problem, but which of the following four approaches would be *best* suited to the above problem? Circle only one choice.

RANDOMIZATION	DYNAMIC PROGRAMMING
A GREEDY ALGORITHM	HORN CLAUSES

(B) *Briefly* describe your best algorithm for this problem. (Use at most one or two lines.)

(C) What is the worst-case running time of your algorithm from part (B)? State your answer in terms of k and n. Use $\Theta(\cdot)$ notation.