CS 373: Combinatorial Algorithms, Fall 2002

Homework 1, due September 17, 2002 at 23:59:59

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

1. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics, where *container*$[i]$ is the name of a container [1] that holds 2^i ounces of beer.

```
BARLEYMOW(n):
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

    "We'll drink it out of the jolly brown bowl,"
    "Here's a health to the barley-mow!"
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

    for i ← 1 to n
        "We'll drink it out of the container[i], boys,"
        "Here's a health to the barley-mow!"
        for j ← i downto 1
            "The container[j],"
        "And the jolly brown bowl!"
        "Here's a health to the barley-mow, my brave boys,"
        "Here's a health to the barley-mow!"
```

(a) Suppose each container name *container*$[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)

[1] One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

(b) Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)

(c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are over 21, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

2. Suppose you have a set S of n numbers. Given two elements you *cannot* determine which is larger. However, you are given an oracle that will tell you the median of a set of three elements.

(a) Give a linear time algorithm to find the pair of the largest and smallest numbers in S.

(b) Give an algorithm to sort S in $O(n \lg n)$ time.

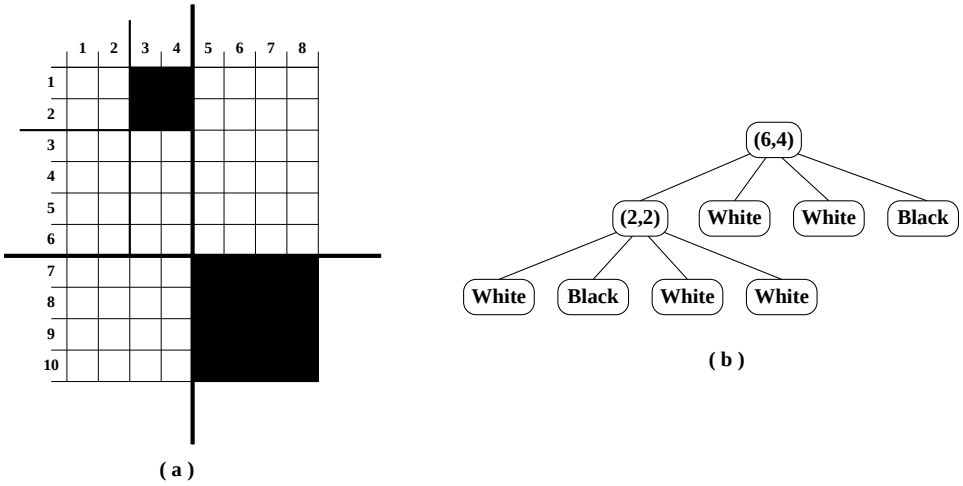
3. Given a black and white pixel image $A[1 \ldots m][1 \ldots n]$, our task is to represent A with a search tree T. Given a query (x, y), a simple search on T should return the color of pixel $A[x][y]$. The algorithm to construct T will be as follows.

```
CONSTRUCTSEARCHTREE(A[1...m][1...n]):
    //Base Case
    if A contains only one color
        return a leaf node labeled with that color

    //Recurse on Subtrees
    (i, j) ← CHOOSECUT(A[1...m][1...n])
    T1 ← CONSTRUCTSEARCHTREE(A[1...i][1...j])
    T2 ← CONSTRUCTSEARCHTREE(A[1...i][j+1...n])
    T3 ← CONSTRUCTSEARCHTREE(A[i+1...m][1...j])
    T4 ← CONSTRUCTSEARCHTREE(A[i+1...m][j+1...n])

    //Construct the Root
    T.cut ← (i, j)
    T.children ← T1, T2, T3, T4
    return T
```

That is, this algorithm divides a multicolor image into quadrants and recursively constructs the search tree for each quadrant. Upon a query (x, y) of T (assuming $1 \leq x \leq m$ and $1 \leq y \leq n$), the appropriate subtree is searched. When the correct leaf node is reached, the pixel color is returned. Here's a toy example.

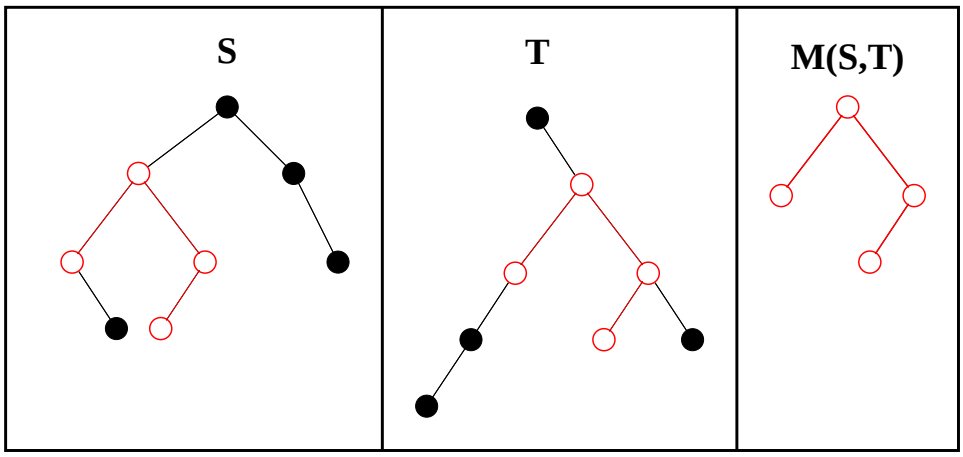


(a) An image A and the chosen cuts. (b) The corresponding search tree.

Your job in this problem is to give an algorithm for ChooseCut. The sequence of chosen cuts must result in an optimal search tree T. That is, the expected search depth of a uniformly chosen pixel must be minimized. You may use any external data structures (*i.e.*a global table) that you find necessary. You may also preprocess in order to initialize these structures before the initial call to ConstructSearchTree($A[1 \ldots m][1 \ldots n]$).

4. Let A be a set of n positive integers, all of which are no greater than some constant $M > 0$. Give an $O(n^2 M)$ time algorithm to determine whether or not it is possible to split A into two subsets such that the sum of the numbers in each subset are equal.

5. Let S and T be two binary trees. A *matching* of S and T is a tree M which is isomorphic to some subtree in each of S and T. Here's an illustration.



A matching $M(S,T)$ of binary trees S and T.

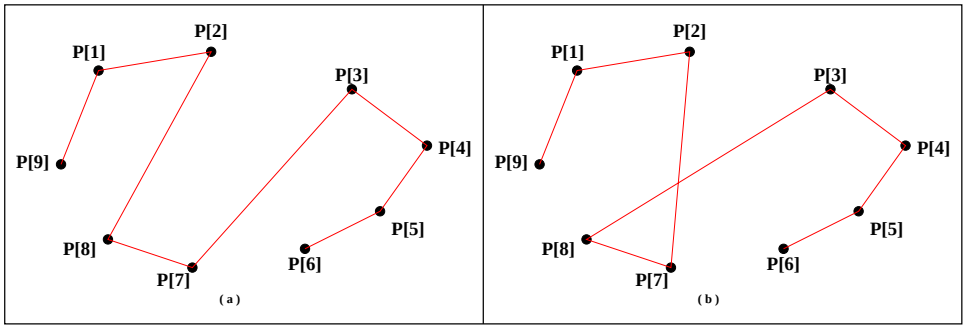
A *maximal* matching is a matching which contains at least as many vertices as any other matching. Give an algorithm to compute a maximal matching given the roots of two binary trees. Your algorithm should return the size of the match as well as the two roots of the matched subtrees of S and T.

6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

   Let $P[1, \ldots, n]$ be a set of n *convex* points in the plane. Intuitively, if a rubber band were stretched to surround P then each point would touch the rubber band. Furthermore, suppose that the points are labeled such that $P[1], \ldots, P[n]$ is a simple path along the convex hull (*i.e.* $P[i]$ is adjacent to $P[i+1]$ along the rubber band).

   (a) Give a simple algorithm to compute a shortest *cyclic* tour of P.

   (b) A *monotonic* tour of P is a tour that never crosses itself. Here's an illustration.



(a) A monotonic tour of P. (b) A non-monotonic tour of P.

   Prove that any shortest tour of P must be monotonic.

   (c) Given an algorithm to compute a shortest tour of P starting at point $P[1]$ and finishing on point $P[\lfloor \frac{n}{2} \rfloor]$.

Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

   (a) the square immediately above,

   (b) the square that is one up and one left (but only if the checker is not already in the leftmost column),

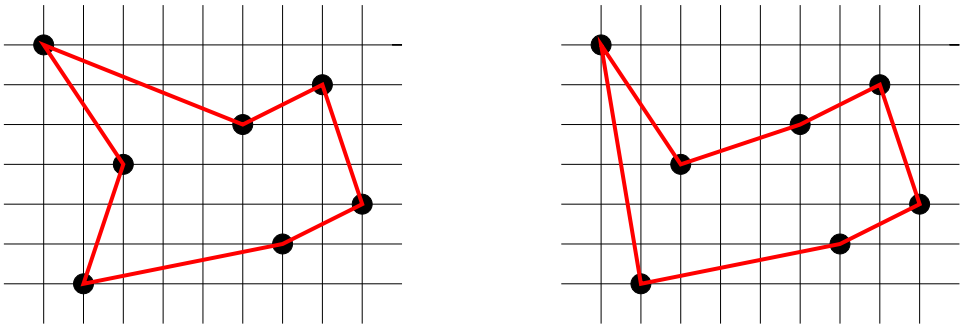
   (c) the square that is one up and one right (but only if the checker is not already in the rightmost column).

   Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given $p(x, y)$ for all pairs (x,y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

   Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

2. (CLRS 15-1) The *euclidean traveling-salesman problem* is the problem of determining the shortest closed tour that connects a given set of n points in the plane. Figure (a) below shows the solution to a 7-point problem. The general problem is NP-complete, and its solution is therefore believed to require more than polynomial time.

   J.L. Bentley has suggested that we simplify the problem by restricting our attention to *bitonic tours* (Figure (b) below). That is, tours that start at the leftmost point, go strictly left to right to the rightmost point, and then go strictly right to left back to the starting point. In this case, a polynomial-time algorithm is possible.



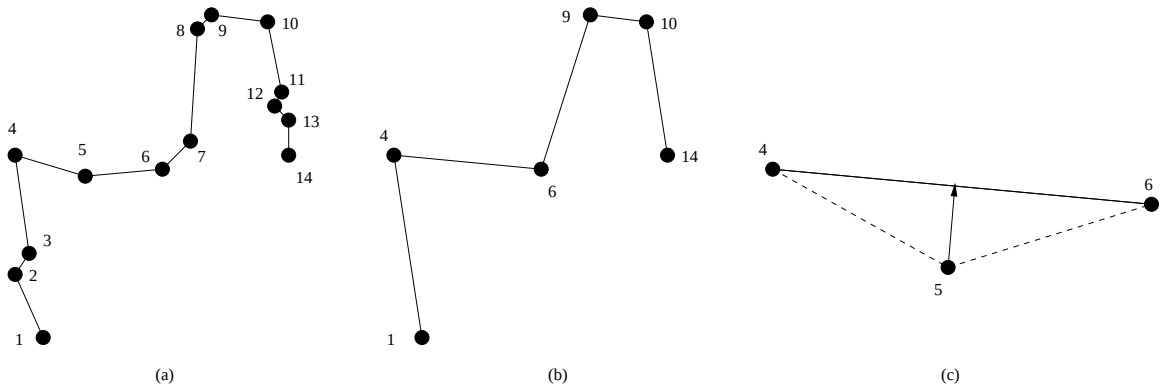
(a) (b)

Seven points in the plane, shown on a unit grid. (a) The shortest closed tour, with length approximately 24.89. This tour is not bitonic. (b) The shortest bitonic tour for the same set of points. It's length is approximately 25.58.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate. [Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.]

3. You are given a polygonal line γ made out of n vertices in the plane. Namely, you are given a list of n points in the plane $p_1, \ldots, p_n$, where $p_i = (x_i, y_i)$. You need to display this polygonal line on the screen, however, you realize that you might be able to draw a polygonal line with considerably less vertices that looks identical on the screen (because of the limited resolution of the screen). It is crucial for you to minimize the number of vertices of the polygonal line. (Because, for example, your display is a remote Java applet running on the user computer, and for each vertex of the polygon you decide to draw, you need to send the coordinates of the points through the network which takes a *long long long time*. So the fewer vertices you send, the snappier your applet would be.)

   So, given such a polygonal line γ, and a parameter k, you would like to select k vertices of γ that yield the "best" polygonal line that looks like γ.



(a) The original polygonal line with 14 vertices. (b) A new polygonal line with 6 vertices. (c) The distance between p_5 on the original polygonal line and the simplification segment p_4p_6. The error of p_5 is $\text{error}(p_5) = \text{dist}(p_5, p_4p_6)$.

Namely, you need to build a new polygonal line γ' and minimize the difference between the two polygonal-lines. The polygonal line γ' is built by selecting k vertices $\{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$ from γ. It is required that $i_1 = 1$, $i_k = n$, and $i_j < i_{j+1}$ for $j = 1, 2, ..., k-1$.

We define the error between γ and γ' by how far from γ' are the vertices of γ. More formally, The difference between the two polygonal lines is

$$error(\gamma, \gamma') = \sum_{j=1}^{k-1} \sum_{m=i_j+1}^{i_{j+1}-1} \text{dist}(p_m, p_{i_j} p_{i_{j+1}}).$$

Namely, for every vertex not in the simplification, its associated error, is the distance to the corresponding simplified segment (see (c) in above figure). The overall error is the sum over all vertices.

You can assume that you are provided with a subroutine that can calculate $\text{dist}(u, vw)$ in constant time, where $\text{dist}(u, vw)$ is the distance between the point u and the segment vw.

Give an $O(n^3)$ time algorithm to find the γ' that minimizes $\text{error}(\gamma, \gamma')$.