Problem Set 2 Due October. 11, 2012 11:55PM

Guidelines for writeups

When describing an algorithm first give a high level overview of your approach, then fill in details as needed to justify your time bound and correctness. Unless you are given a target time bound to achieve, assume you are trying to make your algorithm as fast as possible (in big O terms). Your pseudo-code should always be commented (unless it is self commenting). You are also encouraged to describe your solutions in terms of known algorithms and data structures (e.g. sort the numbers, insert into a balanced binary search tree, delete the minimum of a heap, could all be described with little or no further elaboration).

See the sample writeup (samp.hw.pdf) on the class webpage.

For proofs, strive for clear, clean arguments. It will always be somewhat a matter of taste which steps can be skipped, but try to avoid proofs of obvious points, and be clear on anything tricky. Define your notation carefully. This will often allow you to give a much crisper argument.

Do your own work

It is normal to discuss the homeworks with your classmates at a high level, and particularly to get clarifications of what is being asked (though of course you will get more reliable information from the instructors or TA). You can also post questions to the smartsite forum for the problem set and we will try and answer them.

However, the actual detailed solution you turn in should be your own work. Note that you can probably find solutions on the web, from books, or other students to some of these problems. To turn in such solutions as your own work is dishonest, and may subject you to a campus judicial conduct committee.

Submission details

Homework is due on the due date at the time listed and should be submitted electronically on Smart site. Everyone gets two free late days for the quarter. Once you have used up your late days there is a 20% penalty per day for turning in late homework. Also, no homework will be accepted more than 24 hours late (so solutions can be posted shortly after the due date).

Problem 1.(15) For the points to lines problem discussed in section 6.3 we can also model this as a shortest path problem in a directed graph. Describe how to model this as a graph with $n+1$ nodes (say numbered zero to $n$) such that any valid solution to the points to lines problem corresponds to a path from node zero to node $n$, and the length of this path is the cost of a partition that is naturally associated with this path.

Describe the arcs and their lengths to be used, argue that this finds an optimal solution to the original problem, and give the run time of this approach (Hint: we can find the shortest path in an acyclic graph $G = (V, E)$ in $O(|E| + |V|)$ time).
Problem 2.(20) Consider a more general version of the subset sum problem described in section 6.4 on page 267 where we are allowed to use multiple copies of an item. Thus, we now have a list of \( n \) ordered pairs of integers \((c_i, w_i)\) and we are allowed to use up to \( c_i \) copies of item \( i \) in our solution. For example, if we have input: \((3, 11), (2, 5), (2, 4)\) and a target \( W = 24\), an optimal solution sums to 24 and uses one copy of 11 and 5 and two copies of 4 (we can represent this as \((1, 11), (1, 5), (2, 4)\)).

We can of course convert this input to the form of the original problem by simply repeating the \( i \)th input number \( c_i \) times in the list of input numbers (since there is no problem with there being multiple copies of the same number in the input for the original algorithm). However, this would lead to a slow solution. Instead, show how to solve this problem in \( O(nW) \) time as we did for the original subset sum problem.

Hint for Ps1-problem 2: an alternate (often better) solution for the (basic) subset sum problem fills in a 2D array \( M \) where \( M[i, w] = 1 \) if a subset of the first \( i \) items sums to exactly \( w \), and \( M[i, w] = 0 \) if no such subset sums to \( w \). You may find it easier to modify this version to solve the multiple copy subset sum problem.

Problem 3.(10) In the RNA folding solution we discussed our dynamic programming solution for an interval \( i,j \) allowed us to choose not to match \( b_j \) with anything even though some \( b_t \) in the interval was a legal choice to match. Give an example where choosing not to match is better than matching (your example need not be a string of length more than 10).

Problem 4.(20) In the RNA folding problem discussed in class, a symbol is not permitted to be matched to something that is too close. The folding problem was solved by DP and ran in \( O(n^3) \) time. Now suppose we remove the distance restriction and allow a symbol \( b_j \) to match any earlier one, provided of course that the type of the earlier \( b_t \) is the appropriate type of symbol to match \( b_j \). Further, suppose our string has only G and U symbols. Give an algorithm for this setting that runs in \( O(n) \) time. Argue that your algorithm is correct and that it does run in \( O(n) \) time. The solution is not by DP in this case.

Problem 5.(25) We consider some variations of the sequence alignment problem. For each setting below, describe how to modify the basic dynamic programming formulation to find the minimum edit distance. Then analyze the time and space used by your solution. Note that pars a,b are independent (so consider each change separately).

a) Suppose that we have only a small additional charge for extending a gap. That is, if we match one character in a string to a gap we pay a cost of \( \delta \) as before. However, if we match \( k+1 \) consecutive characters of \( X \) (or consecutive characters of \( Y \)) to gaps we pay a total cost of \( \delta + c \times k \) where \( c << \delta \) is another given constant.

b) Now suppose we want to consider a low cost for transpositions (we switch two consecutive characters). Thus we will allow two consecutive characters in one string (say \( x_i, x_{i+1} \)) to match with two consecutive characters in the other string (say \( y_j, y_{j+1} \)) in reverse order. Thus we would match \((x_i, y_{j+1})\) and \((x_{i+1}, y_j)\). The cost for such a match is \( t \), the transposition penalty plus the normal costs of pairing \((x_i, y_{j+1})\) and \((x_{i+1}, y_j)\): \( \alpha_{x_i,y_{j+1}} + \alpha_{x_{i+1},y_j} \).

Problem 6.(10) In section 6.6, page 282 the book gives an \( O(mn) \) iterative algorithm to solve the sequence alignment problem. An alternative is a recursive function is described below for inputs \( i, j \geq 0 \).
OPT(i,j)
Begin
If (j = 0 ) RETURN iδ
if (i = 0) RETURN jδ
Else, use recurrence 6.16 to compute OPT(i,j). (note that this involves 3 recursive calls to OPT).
End OPT

Estimate the number of recursive calls used by this function to calculate OPT(m,n) (you need not give a precise answer, but argue that it is MUCH slower than the version in the book).