125 marks **CS 218 Endsem** **5:30-8:30, 22/4/16**

1. Use 2 pages per problem. Solve the problems in order and please write the problem number on pages before you begin.
2. When describing an algorithm, mention (a) the algorithm design technique if any, (b) the overall idea of the algorithm, (c) the algorithm described as a function with arguments and return value explained clearly, (d) proof of correctness unless obvious, (e) running time analysis including justification. Please give these in order, with headings "Technique", "Overall idea", "Algorithm"...
3. When giving dynamic programming algorithms describe the terms in the recurrence very clearly.

Problem 1:[25 marks] Suppose you are given as input an array $A[1..n]$. Think of this array as defining a piece of cloth whose bottom boundary is a straight line of length n cm. The height of the cloth above this bottom boundary between $i-1$ cm and i cm (from left) is $A[i]$. The term "rectangle" in what follows is meant to denote rectangles whose sides are either vertical or horizontal.

Design an algorithm to determine the largest area rectangle that can be cut out of this cloth. As an example, let $A = (5, 1, 4, 3, 6, 4, 7, 2, 8)$. Then the largest rectangle would start 1 cm from the left end and would have height 3 cm and width 5 cm. Give an $O(n \log n)$ algorithm. Slower algorithms will get no credit.

Technique: divide and conquer

Overall idea: Divide the cloth in half. Recursion will take care of rectangles which are contained in each half. The conquer step will have to consider rectangles that occupy parts of both halves. 5 marks

Algorithm for the conquer step: Generate all maximal rectangles occupying both parts and pick the largest. For this start with L,R=n/2, H=A[n/2]. Decrement L and increment R while A[L],A[R] remain at least H. At this point we have found a maximal rectangle. Now the next candidate height is H=max(A[L-1],A[R+1]). Repeat the above process for this value of H. Do this until L,R reach the ends. Report the largest area of the rectangles seen so far. 15 marks

Time analysis: The conquer step takes O(n) time because there is unit work for each extension in either direction. So the recurrence is T(n) = 2T(n/2) + O(n), which solves to O(nlogn) 5 marks

An $O(n)$ time stack based solution is possible, and was given by several. students. Other solutions are also there. You will lose marks if the solutions are hard to understand.

Problem 2: [25 marks] Suppose you have two warehouses U and V, respectively holding quantities u and v of a certain item. There are n customers, the ith amongst whom needs to be supplied with

q_i items. The cost of shipping a single item to the ith customer from U and V are respectively u_i and v_i. So if r_i items are shipped to customer i from U and the remaining $q_i - r_i$ from V, then the shipping cost is $\sum_i r_i u_i + (q_i - r_i)v_i$. The problem is: given the numbers u, v and $q_1, \ldots, q_n$, $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$, determine $r_1, \ldots, r_n$ such that the shipping cost is minimized, and the total amount shipped out of each warehouse is no more than the quantity originally present there. Assume that $\sum_i q_i \leq u + v$. Give a polynomial time algorithm.

Technique: Greedy

Overall idea: The decision to be made for each i, is to decide r_i that is to be shipped from U. Gain from shipping a single item to i from U rather than V is $x_i = v_i - u_i$. Since we can decide separately for each item, each decision for the ith customer causes us to gain or lose x_i per item. We arrange items in decreasing order of $|x_i|$ and try to be greedy for each item

Algorithm: Greedy choice: Consider i such that $|x_i|$ is largest. Then if $u_i < v_i$, then we ship to i as much as possible from warehouse U and the rest from warehouse V, i.e. $r_i = \min(u, q_i)$. If $u_i \geq v_i$, then we ship to i as much as possible from warehouse V and the rest from warehouse U, i.e. $r_i = q_i - \min(v, q_i)$.

Claim: There is an optimal solution in which i is assigned as above.

Proof: Suppose the first case holds, i.e. $u_i < v_i$. Suppose an optimal solution Q ships a different value $r < r_i$ from U to i. ($r > r_i$ is invalid because either i will receive more than it needs, or i will receive more than what is present at U).

If in Q, U is finally not empty, we simply get an item from U to i and refuse one item at i coming from V. This will decrease the cost of Q by x_i, giving a contradiction. Else suppose U is empty in Q. But then some customer $j \neq i$ must have received at least 1 unit from U. Consider what happens if we divert 1 unit to i from j. The gain for i is x_i, while the gain for j is $-x_j$. The total gain is thus $x_i - x_j \geq 0$. In this way we can raise r upto r_i – and thus we will have proved that there exists an optimal solution in which r_i is shipped from U.

The case $u_i \geq v_i$ is similar.

Claim 2: There is optimal substructure, i.e. after assigning to one customer, the residual problem is of the same type etc.

Proof: Obvious.

Time analysis: The recursion just requires us to find the largest $|x_i|$ at each stage. So the greedy order is simply decreasing order of $|x_i|$. Thus the time $= O(n \log n)$ suffices.

About 12 marks for getting the algorithm idea: sort by decreasing $|x_i|$. 8 marks for correctness proof. 5 marks for stating the time. Note that sorting by x_i does not work: if all x_i are negative, you must decide first for customers with most negative x_i. – you lose about 10 marks if you are not careful about this.

If you gave a dynamic programming based pseudopolynomial time algorithm, you got upto 10 marks.

Problem 3:[25 marks] You are given as input two arrays $A[1..n][1..n], W[1..n][1..n]$. $A[i][j]$ represents the colour of pixel (i, j) of an image, and $W[i][j]$ gives the weight of pixel (i, j). Your goal is to "compress" A by removing one pixel from each column and get a new image $B[1..n-1][1..n]$ while losing minimum information and producing minimum visual discontinuity. Specifically the following conditions must be met (a) if pixels (i, j) and $(i', j+1)$ are removed for any j, then $|i - i'| \leq 1$, and (b) total weight of pixels removed is as small as possible.

Give a polynomial time algorithm to decide the minimum weight that needs to be removed, and what pixels to remove to acheive that.

Technique: Dynamic programming

Main idea: Let $R(i, j)$ denote the minimum weight that must be removed from columns $j + 1..n$ assuming pixel (i, j) is removed.

$R(i, j) = w(i, j) + min(R(i-1, j+1), R(i, j+1), R(i-1, j+1))$

where $R(i, j)$ to be considered ∞ if pixel (i, j) does not exist.

Also, $R(i, n+1) = 0$ for all i.

Our goal is to find $\min_i R(i, 1)$.

Algorithm: We build the table starting from the last column. Each update needs $O(1)$ time and so total time is $O(n^2)$.

Let $R(1, j)$ be the minimum. Then we will remove pixel $(1, j)$. In general, if we are removing pixel (i,j), then the above recurrence will give i' such that $R(i, j) = w(i, j) + R(i', j+1)$. So we remove pixel $(i', j+1)$ and proceed in a similar manner. This takes $O(n^2)$ time overall.

Recurrence 15 marks, time: 5 marks, extracting the pixels to remove: 5 marks.

Problem 4:[25 marks] Let G be a weighted undirected graph representing a road network. Each edge e in G has a length $L(e) \geq 0$ and a toll $T(e) \geq 0$. In the budgeted travel (BT) problem you are given G along with L, T and two vertices u, v and integers D, B. The goal is to decide if there exists a path from u to v of length at most D, and such that the total toll along the path is at most B. Show that $BT(G, L, T, u, v, D, B)$ is NP-complete.

Certificate for NP: the path itself. 5 marks

Reduce from partition.

$IM(a_1, \ldots, a_n)\{$

G is a path with 2 parallel edges, one with length a_i and 0 toll, and the other with 0 length and toll a_i.

$B = T = \sum_i a_i/2$ }

Problem 5: Only the answer and 2-3 line explanations are expected for the questions below.

(a)[5 marks] Solve the recurrence $T(n) = \log n + T(n/2)$, $T(1) = 0$. Assume n is a power of 2. An exact fully simplified answer is expected.

$T(k) = k + T(k-1) = k + k - 1 + ... + 1 = k(k+1)/2 = \log n(1 + \log n)/2$

(b)[5 marks] Replace α in the following

$$2^{n/\log n} = \alpha(n^{\log n})$$

with Ω, ω, O, o, or θ, or say none will work. If there are more than one possibilities, indicate all.

$\alpha = \Omega, \omega$

(c)[5 marks] I have a 4 letter alphabet, and the letters appear with frequencies 0.1, 0.1, 0.2, 0.6. What is the expected encoding length per letter when a Huffman code is used? Show your work, but you get marks only if you give a correct fully simplified numerical answer.

1.6

(d)[10 marks] Suppose there is an additional condition in problem 2: each customer must be supplied items from only one single warehouse, which one to be decided by the algorithm. How long do you think an algorithm for this variation will take?

We can reduce to this from partition by setting all costs being equal, the warehouses having equal content initially and customer demands being a_i, the input instance elements of partition.

You do not get marks unless you state something about reductions or about the problem becoming NP-complete.