150 marks **CS 301 Endsemester Test** **9:30-12:30, 26/11/5**

Problem 1[30 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 $A[i]$. Your goal is to determine the largest rectangle that can be cut out of this cloth. The rectangle must have sides parallel or perpendicular to the bottom edge.

For example, for $A = [4, 3, 6, 4, 7, 2, 8]$, the largest rectangle would have height 3 and width 5.

Use divide and conquer. Divide A in the middle, find the largest rectangle in the left half, the largest in the right half, and the largest one that passes through the middle strip. Return the largest of those.

How to find the largest one that passes through the middle strip:

Find all maximal rectangles touching the middle line. This can be done as:

minheight = A[n/2] lrect = empty for i=n/2-1 to 1 minheight = min(minheight,A[i]) Append(minheight, i) to lrect. end for

lrect will consist of pairs (y,x) and will be in non-increasing order of y and strictly decreasing order of y. Suppose there are several entries for same y. Then keep the last one of these, i.e. the one with the smallest x.

Do the same and construct rrect. Prune rrect similarly, this time keeping the entry with the largest x.

Now use a mergeing procedure to combine rectangles from rrect and lrect, compute their area and determine the one having the largest area.

—-

The above can be done in time O(n). The recurrence is T(n)=O(n)+2T(n/2). The total time is thus O(nlogn).

Problem 2 Consider a factory in which cars are being built. One step in the manufacture is that of painting the cars, this happens in an area called "painting shop". Cars arrive, say one per hour, along with information about which colour each one needs. Each car must be painted with the specified colour and sent out subject to the following constraints:

1. Painting a car needs 1 hour.
2. At any time only one car can be painted. While a car is being painted, there can be at most one car waiting in the shop.
3. Each car will request for one of the following 4 colours: white, black, green, red.
4. If the colours of consecutively painted cars are different, then an extra cost C incurred.

It is not necessary to paint the cars in the order that they arrive. To save on the extra cost, it is possible to change the order, using the waiting space in the shop.

(a) [25 marks] Given a sequence of car colour requests $C[1..n]$, determine how they should be painted (e.g. which colour at which hour if any.) so that the total extra cost is minimized.

Example: Suppose the sequence is C=[W,B,B,G,W,R,G,R]. Then one idea might be to paint the cars in the order of arrival, with the painting sequence being same as C. In this case, there will be 6 colour changes. Another possibility would be the following painting sequence [-,B,B,W,W,G,G,R,R]. You should check that this sequence can be executed with only one car waiting in the shop besides the one being currently painted. This sequence has only 4 colour changes.

The solution space consists of a sequence of decisions, each being: 0,1,2,3,4 where 0 means dont paint in this step, and 1,2,3,4 denote use corresponding colours.

The first decision can be 0 or the colour of the arriving car. Given that decision, we have either committed to a colour or not. So in general after i cars have passed by we will be in some state s in [0..5] indicating which colour is currently being used; there will be at most 1 waiting car – so of this also there are 5 choices.

So we can write the following recurrence:

N(c,w,i) : number of changes needed given the current chosen colour is c (from [0..5]), the current waiting car is of colour c (from [0..5]), and the ith car is just arriving.

N(c,w,i) = min(N(C[i],w,i+1) + C*(c==C[i]) N(w,C[i],i+1) + C*(c==w)

15 marks to get the basic recurrence.

Recurrence should be evaluated in reverse order of n. w,c can be chosen in 5 ways each at most and i in n. Thus table has 25n entries. Each entry can be computed in O(1) time given relevant table entries.

Hence time = O(n)

Analysis 10 marks.

(b)[10 marks] Suppose you have space for s cars in the shop, including the one being painted, and there are c possible colours overall. How will the time taken change?

the s cars will have to be partitioned among c colours – that will be the number of car configurations, i.e. $\binom{s+c-1}{c-1}$. One of c possible colours will be active. So the table size will increase by this factor. Next, in the loop we will have to consider at most $\min(c, s) = m$ colours to use next, so we will need to solve m possible subproblems. So the time will be $O(m\binom{s+c}{c}cn)$.

Problem 3: Suppose we have a single processor which can execute one job at a time. We are given n jobs, and for each ith job a deadline d_i and time t_i, both integers. The interpretation is that job i must be given the processor for t_i steps in the first d_i steps. The allocation need not be contiguous. You are to give an algorithm to construct a schedule such that as many jobs as possible finish by their deadline.

(a)[5 marks] Show that given any schedule in which the time slots associated with a job are not contiguous, we can construct one in which they are, without violating any deadlines.

You may use the above claim in solving parts (b) and (c), even if you dont attempt this part.

Suppose there is a gap between the ith and the i+1th slots associated with a job. Delay the execution of the ith so that it happens just before the i+1th, while preponing the execution of the intervening jobs by 1 step. This will not hurt any jobs deadline. Repeat so long as there are gaps.

(b)[10 marks] Suppose the jobs are sorted in non-decreasing order of t_i. Show that there must exist an optimal schedule in which job 1 is present, assuming $d_1 \geq t_1$.

Suppose we have an optimal schedule in which job 1 is not present. Now consider time slots $1..d_1$. We replace the task in slots $1..t_1$ with task 1. Since task 1 is shortest, we will not have to throw out more than 1 job.

(c)[20 marks] Use above and some more ideas to devise a greedy algorithm to construct the schedule.

We first prove that there must exist an optimal schedule in which task 1 is placed contiguously at its deadline.

Suppose we are given an optimal schedule in which task 1 is present, but not at its deadline. We will first rearrange it so that its tasks are contiguous. The we will move task 1 to its deadline, shifting all intervening slots early.

If we are given an optimal schedule in which task 1 is absent, we again rearrange the schedule so that all tasks are contiguous. Then we place it at the beginning as in part (a) dislodging at most 1 task, and then move it to its deadline.

Our greedy algorithm will construct a schedule of this type. So our problem is: given that task1 is in its latest position, how do we fill the remaining slots.

Key idea: Construct new problem in which task1 is absent, and all tasks having deadline $\geq d_1$ have their deadline reduced by t_1, those whose deadline is in $[d_1 - t_1 + 1..d_1]$ have their deadline reduced to $d_1 - t_1$. and so on till those whose deadline is $d_1 - t_1$ or lower – whose deadline remains unchanged. If we get an optimal solution O' to this new problem, we will have construct a solution O for the old problem by shifting the tasks in slots $d_1 - t_1 + 1..d_1$ forward by t_1 and inserting task 1 in its place.

O must be a valid solution because: if a task appears at $d_1 - t_1 + i$ in O', then its deadline must be at least that. Hence in the original problem its deadline must have been higher by t_1, so shifting it forward will not cause a violation. Suppose O is not optimal, but some Q is. Construct Q' by removing task 1 and shifting everything ahead back by t_1. Q' must be an optimal solution to P'.

Why: All tasks in Q appearing after d_1 have deadline at least that, and hence will have deadline smaller by t_1 in P'. However, in Q' they will execute t_1 steps early, and hence they must satisfy their deadline. The deadline of a task that already appears at or before

d_1 in Q may reduce in P', but not below d_1. Hence Q' must be a valid solution for P'. But then Q' must have more tasks than O' which is not possible.

Time analysis: In each level of the recursion, we need to reduce the deadlines of some tasks. This can be done in $O(n)$ time. So total time is $O(n^2)$.

Problem 4(a)[5 marks] Define the notion of a polynomial time verifier for a decision problem. (b)[5 marks] Define *NP*. (c)[5 marks] Define *NP-complete*.

Problem 5: In *circuit-switched* communication, the entire path from the sender to the receiver must be reserved for the duration of the transmission. Thus, two transmissions can be conducted simultaneously only if the paths required for the two transmissions do not overlap in any edge. For example, in the network shown, the transmission along path P_1 can be done in parallel with the transmission along path P_2, but not in parallel with the transmission along path P_3. The natural question that arises is: how can we schedule the transmissions in the shortest amount of time? Assume for simplicity that each transmission takes unit time. More precisely, the input to the problem is a graph and a set of paths $P_1, \ldots, P_n$ in the graph. The output is a sequence of times $t_1, \ldots, t_n$ with the interpretation that the transmission along P_i is to be conducted at step t_i. The assignment must satisfy the condition that if $t_i = t_j$ then either $i = j$ or P_i and P_j are disjoint. Also part of the input is an integer T. In the decision version the goal is to decide whether the transmissions can be completed in time T. Call this problem the CSS problem.

(a)[10 marks] Show that CSS reduces to graph colouring. Consider a CSS instance consisting of the network and paths shown, with $T = 2$. If f is the instance map in your reduction, show the graph colouring instance $f(x)$.

(b)[15 marks] Show that graph colouring reduces to CSS. If g is the instance map in your reduction, show $g(x)$ for graph colouring instance $x = (G, k)$ where G is as shown and $k = 2$.

(c)[10 marks] Show that CSS is NP-complete using the above as necessary. You may use whatever definition of NP-completeness as convenient, (state it if it is different from the one you gave earlier) and give any additional argument as needed.

in NP and certificate	3 marks
NP-hard:	
Designate an edge for each conflicting pair of paths.	
Add edges to join conflicting edges into a path.	10 marks
If paths are not simple:	-3 marks
number of colours = length of schedule	3 marks
Polytime	3 marks
if colouring exists then schedule exists	3 marks
if schedule then colouring	3 marks