

## Problem Set 4—NOT Due (don't turn it in)

### (35) Problem 1. *Activity Variants*

Suppose that we have  $n$  activities to schedule and as in the class example each activity has a start time  $s_i$  and a finish time  $f_i$ , and we cannot schedule two activities if they overlap. As a new variant, each activity has a weight  $w_i$  which is the profit you get for scheduling activity  $i$ . Our goal is to find a set  $S$  of non-overlapping activities which have maximum total weight. You may assume all the weights are distinct.

**Part A.** Consider the following greedy strategy:

(1) sort the activities so  $w_1 > w_2 > \dots > w_n$ .

(2) For  $i = 1$  to  $n$

    Put activity  $i$  into  $S$  unless it overlaps an activity  
    already in  $S$ .

Give an example which shows that this strategy will **not** always find an optimal solution.

**Part B.**

Suppose now that we have many ties among the weights so there are only  $k$  distinct weights and  $k$  is much smaller than  $n$  (the number of activities). Rather than sorting the items as is done in (1) above, we could instead first find the  $k$  distinct weights among the items and then sort them.

Describe a fast way to find the  $k$  distinct weights and give the running time of your implementation of the algorithm of part A).

**Part C.** Now suppose each activity has length 1 (so  $f_i = s_i + 1$ ) and the start (and thus end times) are integers. Prove that the algorithm of part A) finds an optimal solution.

**Part D.** For the setting of part C (length one, and weights), the algorithm of part A would take  $\Theta(n \log n)$  time for sorting. Instead, describe how to use hashing to find an optimal solution for this setting in  $O(n)$  expected time.

### (25) Problem 2. We know that the 0-1 knapsack problem can be solved by Dynamic programming, and the fractional problem can be solved more quickly by the greedy algorithm. In many settings there is a mixture of items: some can be split (as in the fractional problem) while others cannot. Suppose we have $n$ items which can be split and $k$ items which cannot. Let $(v_1, w_1), \dots, (v_n, w_n)$ be the value weight pairs for the fractional items, and $(v_{n+1}, w_{n+1}) \dots (v_{n+k}, w_{n+k})$ be the 0-1 items. As before we have a single weight limit $W$ .

**Part A.** Suppose that  $k = 1$ . Describe a fast algorithm to find an optimal choice of items. Give the running time of your solution and justify its correctness.

**Part B.** Describe how to solve this problem in general, but you may assume that  $k$  is much smaller than  $n$ .

**Part C.** Suppose that  $n = 10,000$   $k = 20$  and  $W = 10,000,000,000$ . Estimate how long your solution of part B) would take to solve this problem on a fast computer (say  $10^9$  operations per second and a gigabyte of memory).

### (15) Problem 3 We claimed that the stable marriage algorithm could be implemented to run in $O(n^2)$ time. Describe data structures which will allow this run time. In particular you should describe how to store the preference lists, the current marriage, and the men who are unpaired.

Justify that each proposal now takes  $O(1)$  time.

**(15) Problem 4**

Consider a variation of the assembly line scheduling problem. We now have a third line C, but this line is different, it has only odd numbered stations (1,3,5, ...) and each station  $C_i$  in line C does the operations of both the  $i$ th and  $(i + 1)$  stage of the line (for simplicity, assume  $n$  is even). As before, we can continue on line C with no delay, or switch to line A or B (but after completing  $C_i$  we would switch to  $A_{i+2}$ ). Similarly, after completing an even operation on line A or B we can switch to line C for the next operation. For simplicity we will assume a uniform delay  $d$  to switch to line C (from A or B) or to switch from line C to line A or B.

Modify the dynamic programming formulation given in class to find the fastest time using a combination of lines A,B and C (you don't have to output the actual sequence). Assume it takes time  $c_i$  for  $C_i$  to complete the  $i$ th and  $i + 1$  stages.

**(25) Problem 5** Suppose we have  $n$  boxes we want to stack. The  $i$ th box has an integer weight  $w_i$  and a strength  $s_i$ . The total weight of the boxes on top of box  $i$  plus  $w_i$  can be at most  $s_i$  (so a box's own weight limits what can be on top of it). Our goal is to find the best sequence of boxes to stack such that the total weight of the boxes used is as large as possible, and the total weight on top of each box is at most that box's strength.

For example, with boxes (3,3), (10,15), (8,30), (3,25) (16,25) We could stack (left most = bottom of the stack):

(16,25), (8,30) For a total weight of 24 (note we can't add any box since then we would have more than 25 on the bottom box)

A better solution is: (8,30) (16,25) (3,3) for a total of 27 (note we can't use (10,15) instead of (3,1) since then we would have too much weight for the second box in our stack).

- a) Prove that there is an optimal solution in which the strengths of the boxes decrease (or are tied) as we go up the stack.
- b) Suppose all boxes have the same strength  $S$ . Describe how to use dynamic programming to find an optimal solution.
- c) Now consider the general case where weights and strengths may vary. Describe how to use dynamic programming to find an optimal solution. Hints: i) first sort the boxes by strength, ii) consider optimal solutions using the first  $i$  strongest boxes, for each  $i$ , iii) compare to the knapsack solution.