Problem Set 4—Due February 21, 3:15 PM

(35) Problem 1. Activity Varients

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 \( w_1 > w_2 > \ldots > w_n \).

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 \( (v_1, w_1), \ldots (v_n, w_n) \) be the value weight pairs for the fractional items, and \( (v_{n+1}, w_{n+1}), \ldots (v_{n+k}, w_{n+k}) \) 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 1,000 MHZ 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.
Problem 4  a) Suppose you are given a directed graph G, (in adjacency list form), and you want to determine if G has a directed cycle. Describe an efficient algorithm to find a directed cycle in G or determine that no such cycle exists.

b) Give the running time of your solution to part b) in terms of \( n \) and \( m \).

c) Suppose you are given a C program and you can determine for each function, the set of functions which it calls (thus you are told Function1 calls 2 and 3, 3 calls 4 and 5, ...). Describe how you can efficiently determine if your C program is recursive (a recursive program has the possibility that one of its functions can call itself, possibly indirectly. Thus if A calls B which calls C which calls A, we have recursion).