

## Dynamic Programming

1. (a) Solve the following instance of the  $\{0,1\}$  Knapsack Problem with four items where the maximum allowed weight is  $W_{\max} = 10$ .

$i$	1	2	3	4
$b_i$	25	15	20	36
$w_i$	7	2	3	6

- (b) For comparison, what is the solution to the corresponding Fractional Knapsack problem with the same value of  $W_{\max}$  (use the greedy method we discussed for this “Fractional” version)?
2. Do the same as above (find solutions to both the  $\{0,1\}$  and Fractional Knapsack problems) for this collection of items, where  $W_{\max} = 11$ .

$i$	1	2	3	4	5
$b_i$	14	12	15	20	16
$w_i$	4	3	8	7	3

3. (This is one of the problems from the sheet I gave out on the first day of class.) Consider a post office that sells stamps in three different denominations, 1p, 7p, and 10p. Design a dynamic programming algorithm that will find the *minimum number* of stamps necessary for a postage value of  $N$  pence. (Note that a greedy type of algorithm won’t necessarily give you the correct answer, and you should be able to find an example to show that such a greedy algorithm doesn’t work.)

What is the running time of your algorithm?

4. (This is also from the initial problem sheet that I gave to you.)

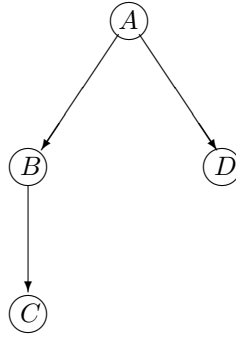
American coins come in five different values, namely 50-cent, 25-cent, 10-cent, 5-cent, and 1-cent coins. Therefore, if we wanted change for 11 cents (or 11¢), we can do this with one 10¢ coin and one 1¢ coin, or two 5¢ coins and one 1¢, or one 5¢ and 6 1¢ coins, or with eleven 1¢ coins. So there are four ways to give change for 11¢.

*How many* ways are there to make change for  $N$ ¢? (And how can you compute this number *efficiently*?)

5. (Adapted from *Algorithm Design* by Jon Kleinberg & Éva Tardos.)

The current class of students at Sandhurst has an annual picnic (ok, maybe they don’t really do this, but for the purposes of this question they do...). This year it happens that the picnic has fallen on a rainy day and the ranking officer decides to postpone the picnic and must notify everyone by phone. Here is the mechanism for doing this.

Each person except for the ranking officer reports to a unique *superior officer*. Thus, the reporting hierarchy can be described as a tree  $T$ , rooted at the ranking officer, in which each other node  $v$  has a parent node  $u$  equal to his or her superior officer. Conversely, we can call  $v$  a *direct subordinate* of  $u$ . See the picture below, where  $A$  is the ranking officer,  $B$  and  $D$  are direct subordinates of  $A$ , and  $C$  is a direct subordinate of  $B$ .



To notify everyone of the postponement, the ranking officer first calls each of her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he or she must notify each of his or her direct subordinates, one at a time. The process continues in this way until everyone is notified. Note that each person in this process will only call direct subordinates, for example  $A$  would not be allowed to call  $C$ .

We can picture this notification process as divided into rounds. In one round, each person who has already learned of the postponement can call one of his or her subordinates on the phone. The number of rounds it takes for everyone to be notified might depend upon the sequence in which each person calls their direct subordinates. In the given example, it will take only two rounds if  $A$  starts by calling  $B$ , but will take three rounds if  $A$  starts by calling  $D$ .

Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified for a rooted tree  $T$  as described above. How can you output a sequence of phone calls (i.e. a schedule for each person) that achieves this minimum number of rounds?

6. Consider the set of weighted intervals given below, where  $s_i$  is the start time,  $f_i$  is the finish time, and  $v_i$  is the value of the interval.

$j$	$s_j$	$f_j$	$v_j$
1	0	6	2
2	2	10	4
3	9	15	6
4	7	18	7

Solve this instance of the weighted interval scheduling problem, i.e. find a set of (non-conflicting) intervals with maximum total weight.

(**Note:** For such a small set of intervals, it's likely "easy" to see the solution without much work, but the intention is for you to understand how the algorithm we discussed in class works. So I would really expect you to find the  $p(j)$  values we defined, and the values  $Opt(j)$  that would be computed to find the solution.)

7. Do the same as the previous problem for this collection of weighted intervals.

$j$	$s_j$	$f_j$	$v_j$
1	1	4	4
2	2	6	2
3	3	9	1
4	5	10	6
5	7	14	3
6	12	16	1
7	9	18	5
8	15	20	3

8. Suppose you're managing construction of billboards on the Rocky & Bullwinkle Memorial Highway, a heavily traveled stretch of road that runs west-east for  $M$  miles. The possible sites for billboards are given by numbers  $x_1 < x_2 < \dots < x_n$ , each in the interval  $[0, M]$ , specifying their position in miles measured from the western end of the road. If you place a billboard at position  $x_i$ , you receive a revenue of  $r_i > 0$ .

Regulations imposed by the Highway Department require that no two billboards be within five miles or less of each other. You'd like to place billboards at a subset of the sites so as to maximize your total revenue, subject to this restriction.

For example, suppose  $M = 20$  and  $n = 5$  with

$$\{x_1, x_2, x_3, x_4, x_5\} = \{6, 7, 12, 13, 14\}$$

and

$$\{r_1, r_2, r_3, r_4, r_5\} = \{5, 6, 5, 3, 1\}.$$

Then the best solution is to place billboards at  $x_1$  and  $x_3$  to achieve a revenue of 10.

Describe a (dynamic programming) procedure to find a solution for this problem. What is the running time of your procedure (this should be polynomial in  $n$ )?

**Hint:** As a first step towards the solution, define (similar to the Weighted Interval Scheduling Problem)  $e(j)$  to be the easternmost site that is more than 5 miles away from  $x_j$ . In other words, if you place a billboard at  $x_j$ , then  $x_1, x_2, \dots, x_{e(j)}$  are also valid places to place billboards (subject to the same restriction about distances). Note that computing the  $e(j)$  values takes time  $O(n)$ .

9. Longest Increasing Subsequence. (This is a classical dynamic programming problem.) Given a sequence of real numbers  $A_1, A_2, \dots, A_n$ , determine a subsequence (not necessarily contiguous) of maximum length in which the values form a *strictly increasing sequence*.

For example, given the sequence of integers

$$-3, 4, 14, 0, -1, 3, 5, 2, 5, 10$$

the subsequence consisting of 4, 5, 10 is an increasing subsequence, as is  $-3, 0, 3, 5, 10$ . Your task is to find a longest subsequence so that the values are strictly increasing. (There might be repeated numbers in the original sequence, but there can be no repeats in a longest subsequence since it must be strictly increasing.)

**Hint:** Define  $L(j)$  to be the length of the longest increasing subsequence that ends at (and includes) position  $j$ . How can you find  $L(j)$  in terms of the values  $L(i)$  for  $i < j$ ? Once you know the  $L(j)$  values, what's the answer to the original problem? And how long does it take to compute all of the  $L(j)$  values?

10. Balanced Partition. Suppose that you have a set,  $A = \{a_1, \dots, a_n\}$ , of  $n$  integers, each of which is between 0 and  $K$  (inclusive). Your goal is to find a partition of  $A$  into two sets  $S_1$  and  $S_2$  (so  $S_1 \cup S_2 = A$  and  $S_1 \cap S_2 = \emptyset$ ) that *minimizes*

$$|\text{sum}(S_1) - \text{sum}(S_2)|,$$

where  $\text{sum}(S_i)$  equals the sum of the values in the set  $S_i$ .

**Hint:** Define  $V(i, j) = 1$  if there is some subset of  $\{a_1, \dots, a_i\}$  that sums up to the value  $j$  (i.e. some collection from the first  $i$  integers sums up to  $j$ ), otherwise set  $V(i, j) = 0$ .

How many different values of  $V(i, j)$  do you need to compute (in terms of  $n$  and  $K$ )? How can you compute the  $V(i, j)$  values if you know all of the  $V(i-1, j)$  values? How can you use the collection of  $V(i, j)$  values to answer the original question (and find the minimum value of  $|\text{sum}(S_1) - \text{sum}(S_2)|$ )?