Homework 6

Ques 1. From the lecture, you know how to use dynamic programming to solve the 0-1 knapsack problem where each item is unique and only one of each kind is available. Now let us consider knapsack problem where you have infinitely many items of each kind. Namely, there are n different types of items. All the items of the same type i have equal size w_i and value ~~w_i~~ v_i. You are offered with infinitely many items of each type. Design a dynamic programming algorithm to compute the optimal value you can get from a knapsack with capacity W.

Ans 1.

Set initial condition as $OPT(0,0) = 0$.

$$OPT(i,j) = \max\{OPT(i-1,j), OPT(i, j-j_i) + V_i\}$$

for a knapsack of capacity $0 \le w \le W$ and with i here where $i \le n$, above will be the recurrence

Ques 2. Given a non-empty string s and a dictionary containing a list of unique words, design a dynamic programming algorithm to determine if S can be segmented into a space-separated sequence of one or more dictionary words. If S = "algorithmic design" and your dictionary contains "algorithm" and "design". Your algorithm should answer Yes as s can be segmented as "algorithmdesign".

Ans. 2 This can be solved using the recurrence relation,

- $$OPT(i) = \max_{0 < j < i \text{ and substr}_{j+1,i} \text{ is a word in the dictionary}} opt(j)$$

where for a $substr_{ij}$,
$substr_{ij}, substr_{ij+1} \ldots substr_{ji}$ can be denoted as $substr_{j,i}$.
and
$opt(i) = 1$ if segmentation is possible for $substr_{1,i}$ else 0.

- Segmentation of $substr_{1,i}$ is possible if only the last word (say $substr_{j,i} \ldots substr_{ji}$) is in the dictionary, the remaining substr $substr_{1,j}$ can be segmented.

- Time complexity – $\Theta(n^2)$

Guess. Given n balloons, indexed from 0 to n-1. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get nums[left] * nums[i] * nums[right] coins. Here left and right are adjacent indices of i. After the bursting the balloon, the left and right then becomes adjacent. You may assume nums[-1] = nums[n] = 1 and they are not real then becomes adjacent. You may assume nums[-1] = nums[n] = 1 and they are not real therefore you can not burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

Here is an example. If you have the nums arrays equal [3,1,5,8]. The optimal solution would be 167, where you burst balloons in the order of 1, 5 3 and 8. The left balloons after each step is

[3, 1, 5, 8] → [3, 5, 8] → [3, 8] → [8] = []

And the coins you get equals:

(3 * 1 * 5) + (3 * 5 * 8) + (1 * 3 * 8) + (1 * 8 * 1) = 167

Ans 3. This can be solved using the recurrence relation,

- $OPT(i,j) = \max\{OPT(i,p-1) + OPT(p+1,j) + nums[p]*nums[i-1]*nums[j+1]\}$ $i \le p \le j$

  and if $j < i$, $OPT(i,j) = 0$

- where,

  $OPT(i,j)$ is maximum coins for $i \ldots j$ balloons. If p is last balloon to burst, then i to $p-1$ are bursted and all balloons from $p+1$ to r which are now two subproblems.

- Time taken $- O(n^3)$.

Ques. Suppose you have a rod of length N, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length i is worth p_i dollars. Devise a dynamic programming algorithm to determine the maximum amount of money you can get by cutting rod strategically and selling the cut pieces.

Ans. • Let the length of rods array be initialized as rods[0, . . . n].
• Initialize rods[0] = 0
• Now, we can solve using the recurrence,

$$p = \max(p, p[j] + rod[i-j]) \quad \text{—— (1)}$$

• Time taken – $\Theta(n^2)$
• At each remaining length of rod, cut rod at a point and obtain points for one of the cut pieces and recursively compute maximum points to get for other pieces.
• Algorithm:

```
for i = 1 to n do
    p = -∞
    for j = 1 to i do                    using (1)
        p = max (p, p[j] + rod[i-j])
    end for
end for
return rods[n]
```

Ques 5. Solve Kleinberg and Tardos, Chapter 6, Exercise 6

In a word processor, the goal of "pretty-printing" is to take text with a ragged right margin, like this,

Call me Ishmael,
Some years ago,
never mind how long precisely,
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.

and turn it into text whose right margin is as "even" as possible, like this.

Call me Ishmael. Some years ago, never mind how long precisely, having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world.

To make this precise enough for us to start thinking about how to write a pretty-printer for text, we need to figure out what it means for the right margins to be "even". So suppose our text consists of a sequence of words, $W = \{w_1, w_2, \ldots, w_n\}$, where w_i consists of c_i characters. We have a maximum line length of L. We will assume we have a fixed width font and ignore issues of punctuation or hyphenation.

A formatting of W consists of a partition of the words in W into lines. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have

$$\left[\sum_{i=j}^{k-1} (c_i + 1) \right] + c_k \leq L.$$

We will call an assignment of words to a line valid if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the slack of the line — that is, the number of spaces left at the right margin. Give an efficient algorithm to find a partition of a set of words W into valid lines, so that the sum of the squares of the slacks of all lines (including the last line) is minimized.

Ans 5. • Let us represent words array as
words = { word1, word2, . . . , wordn }

• we can represent extra space-characters by

$$s(j,i) = L - i + 1 - \sum_{t=j}^{j+i-1} c_t$$ } if first k words are put in first line

where in one line has i words, then remaining lines consist solution for subproblem with set { word i+1, . . . n }

• Let OPT(j) be sum of squares of slacks for solution with words { wordsj, . . wordsn },

then

if $p \geq n - j + 1$ then
$OPT(j) = 0$

else if $p < n - j + 1$

$$OPT(j) = \min_{1 \leq i \leq p} \{ (s(j,i))^2 + OPT(j+1) \}$$

if we can put at most the first p words from wordsj to wordsn in a line,

i.e. $\sum_{t=i}^{p+i-1} c_t + p - 1 \leq L$ ad $\sum_{t=1}^{p+i} w_t + p > L$.

• Time to run is $O(nL)$ since we have need to calculate OPT(j) for n values of j.

Ques 6. Solve Kleinberg and Tardos, Chapter 6, Exercise 10.

You're trying to run a large computing job in which you need to simulate a physical system for as many discrete steps as you can. The lab you're working in has two large supercomputers (which we'll call A and B) which are capable of processing this job. However, you're not one of the high-priority users of these supercomputers, so that at any given point in time, you're only able to use as many spare cycles as these machines have available.

Here's the problem you face. Your job can only run on one of the machines in any given minute. Over each of the next n minutes, you have a "profile" of how much processing power is available on each machine. In minute i, you would be able to run $a_i > 0$ steps of the simulation if your job is on machine A, and $b_i > 0$ steps of the simulation if your job is on machine B. You also have the ability to move your job from one machine to the other; but doing this costs you a minute of time in which no processing is done on your job.

→

So, given a sequence of n numbers minutes, a plan is specified by a choice of A, B, or "move" for each minute, with the property that choices A and B cannot appear in consecutive minutes. For example, if your job is on machine A in minute i, and you want to switch to machine B, then your choice for minute i+1 must be move, and then your choice for minute i+2 can be B. The value of a plan is the total number of steps that you manage to execute over the n minutes: so it's the sum of a_i over all minutes in which the job is on A, plus the sum of b_i over all minutes in which the job is on B.

The problem, Given values $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, find a plan of maximum value. (Such a strategy will be called optimal.) Note that your plan can start with either of a machines A or B in minute 1.

Example. Suppose $n=4$, and the values of a_i and b_i are given by the following table.

	Minute 1	Minute 2	Minute 3	Minute 4
A	10	1	1	10
B	5	1	20	20

Then the plan of maximum value would be to choose A for minute 1, then move for minute 2, and then B for minutes 3 and 4. The value of this plan would be $10+0+20+20=50$

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

In minute 1, choose the machine achieving the larger of a_1, b_1
Set $i = 2$
While $i \leq n$
    what was the choice in minute $i-1$?
    If A :
        If $b_{i+1} > a_i + a_{i+1}$ then
            Choose move in minute i and B in minute $i+1$
            Proceed to iterate $i+2$
        Else
            Choose A in minute i
            Proceed to iteration $i+1$
        Endif
    If B : behave as above with roles of A and B reversed
End While

In your example, say what the correct answer is and also what the algorithm above finds.

(b) Give an efficient algorithm that takes values for $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ and returns the value of an optimal plan

Ans 6(a) In the example below,

	Minute 1	Minute 2	Minute 3	Minute 4
A	10	1	1	1000
B	1	1	200	200

- the optimal solution will play on A for 4 steps
- but the algorithm will choose A and then play on B for the final two steps.

(b)

Let $OPT_A(i)$ – max. value of plan in minute i through i that ends on machine A

$OPT_B(i)$ – max. value of plan in minute 1 through i that ends on machine B.

~~In minute i, choose the machine achieving the larger~~

0

At minute 1, $OPT_A(1)$ is the first value i.e. a_1

At minute 1, $OPT_B(1)$ is the first value i.e. b_1

Set $i=2$

while $i \leq n$

$OPT_A(i) = a_i + \max\{OPT_A(i-1), OPT_B(i-2)\}$;

record the action (either stay or move) in minute $i-1$ that achieves maximum.

$OPT_B(i) = b_i + \max\{OPT_B(i-1), OPT_A(i-2)\}$;

record the action in minute $i-1$ that achieves maxim

end while

- return $\max\{OPT_A(n), OPT_B(n)\}$;
- Time complexity – $O(n)$.