Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Planting Trees

This problem will guide you through the process of writing a dynamic programming algorithm.

You have a garden and want to plant some apple trees in your garden, so that they produce as many apples as possible. There are n adjacent spots numbered 1 to n in your garden where you can place a tree. Based on the quality of the soil in each spot, you know that if you plant a tree in the ith spot, it will produce exactly x_i apples. However, each tree needs space to grow, so if you place a tree in the ith spot, you can't place a tree in spots $i-1$ or $i+1$. What is the maximum number of apples you can produce in your garden?

(a) Give an example of an input for which:

- Starting from either the first or second spot and then picking every other spot (e.g. either planting the trees in spots $1, 3, 5\dots$ or in spots $2, 4, 6\dots$) does not produce an optimal solution.
- The following algorithm does not produce an optimal solution: While it is possible to plant another tree, plant a tree in the spot where we are allowed to plant a tree with the largest x_i value.

(b) To solve this problem, we'll think about solving the following, more general problem: "What is the maximum number of apples that can be produced using only spots 1 to i?". Let $f(i)$ denote the answer to this question for any i. Define $f(0) = 0$, as when we have no spots, we can't plant any trees. What is $f(1)$? What is $f(2)$?

(c) Suppose you know that the best way to plant trees using only spots 1 to i does not place a tree in spot i. In this case, express $f(i)$ in terms of x_i and $f(j)$ for $j < i$. (Hint: What spots are we left with? What is the best way to plant trees in these spots?)

(d) Suppose you know that the best way to plant trees using only spots 1 to i places a tree in spot i. In this case, express $f(i)$ in terms of x_i and $f(j)$ for $j < i$.

(e) Describe a linear-time algorithm to compute the maximum number of apples you can produce. (Hint: Compute $f(i)$ for every i. You should be able to combine your results from the previous two parts to perform each computation in $O(1)$ time).

Solution:

(a) For the first algorithm, a simple input where this fails is $[2, 1, 1, 2]$. Here, the best solution is to plant trees in spots 1 and 4. For the second algorithm, a simple input where this fails is $[2, 3, 2]$. Here, the greedy algorithm plants a tree in spot 2, but the best solution is to plant a tree in spots 1 and 3.

(b) $f(1) = x_1$, $f(2) = \max\{x_1, x_2\}$

(c) If we don't plant a tree in spot i, then the best way to plant trees in spots 1 to i is the same as the best way to plant trees in spots 1 to $i-1$. Then, $f(i) = f(i-1)$.

(d) If we plant a tree in spot i, then we get x_i apples from it. However, we cannot plant a tree in spot $i-1$, so we are only allowed to place trees in spots 1 to $i-2$. In turn, in this case we can pick the best way to plant trees in spots 1 to $i-2$ and then add a tree at i to this solution to get the best way to plant trees in spots 1 to i. So we get $f(i) = f(i-2) + x_i$.

(e) Initialize a length n array, where the ith entry of the array will store $f(i)$. Fill in $f(1)$, and then use the formula $f(i) = \max\{f(i-1), x_i + f(i-2)\}$ to fill out the rest of the table in order. Then, return $f(n)$ from the table.

2 Change making

You are given an unlimited supply of coins of denominations $v_1, \ldots, v_n \in N$ and a value $W \in N$. Your goal is to make change for W using the minimum number of coins, that is, find a smallest set of coins whose total value is W.

1. Design a dynamic programming algorithm for solving the change making problem. What is its running time?
2. You now have the additional constraint that there is only one coin per denomination. Does your previous algorithm still work? If not, design a new one.

Solution:

1. For $0 \le w \le W$, define

$$f(w) = \text{the minimum number of coins needed to make a change for } w.$$

It satisfies

$$f(w) = \min \begin{cases} 0 & \text{if } w = 0, \\ 1 + \min_{j:\, v_j \le w} f(w - v_j) \\ \infty \end{cases}$$

The answer is $f(W)$, where ∞ means impossible. It takes $O(nW)$ time.

2. For $0 \le i \le n$ and $0 \le w \le W$, define

$$f(i, w) = \text{the minimum number of coins (among the first } i \text{ coins) needed}$$
$$\text{to make a change for } w\text{, having one coin per denomination.}$$

It satisfies

$$f(i, w) = \min \begin{cases} 0 & \text{if } i = 0 \text{ and } w = 0, \\ f(i-1, w) & \text{if } i > 0, \\ 1 + f(i-1, w - v_i) & \text{if } i > 0 \text{ and } v_i \le w, \\ \infty \end{cases}$$

The answer to the problem is $f(n, W)$. It takes $O(nW)$ time.

3 Cards in a Line

You may have seen the idea of minimax decision making in two-player games in artificial intelligence classes. In this problem, we will see that for simple games, we can efficiently compute the minimax strategy using dynamic programming.

Alice and Bob are playing a game where there are n cards in a line, and the ith card is worth $p_i \ge 0$ points. Alice goes first, and Alice and Bob take turns either picking up the first or last card remaining in the line. e.g. Alice can take the 1st or nth card on her first turn. If she takes the 1st, on Bob's first turn, he can take the 2nd or nth card.

(a) Alice decides to use a greedy strategy: "on my turn, I will take whichever of the first and last card is worth more points". Show a small counterexample ($n \leq 5$) where Alice will lose if she plays this greedy strategy, assuming Alice goes first and Bob plays optimally, but she could have won if she had played optimally.

(b) After seeing your counterexample, Alice decides to use dynamic programming to play optimally, assuming Bob also plays optimally. Write a dynamic program that computes how many points Alice ends up with, and state its runtime.

(Hint: Let $s(i, j)$ denote Alice's score at the end of the game if the game consists only of cards i to j and Alice goes first. To write a recurrence for $s(i, j)$, first think about what Bob should do on his turn. Then use that to decide what Alice should do on her turn).

Solution:

(a) One possible arrangement is: $[1, 1, 100, 2]$. Alice first greedily takes the 3 from the right end, and then Bob snatches the 2, so Bob gets 101 and Alice gets a miserly 3. If Alice had started by craftily taking the 1 from the left end, then regardless of Bob's next move, she is free to take the 100.

There are many other counterexamples. They're all of length at least 4.

(b) We want to compute $s(1, n)$.

We have for all $s(i, i)$ that $s(i, i) = p_i$, since Alice will just take the remaining card worth p_i points.

We also have for all $s(i, i+1)$ that $s(i, i+1) = \max\{p_i, p_{i+1}\}$, since when there are two cards left it is optimal for Alice to take the one worth more points.

So, let's consider when $j > i + 1$. First consider what Bob will do on his turn if cards i to j remain. If Bob plays optimally, since the total number of points Alice and Bob can get from the remaining cards is fixed, on his turn he will take whichever card minimizes the remaining points Alice gets. That is, he will take card i if $s(i + 1, j) < s(i, j - 1)$, and otherwise take card j, so Alice will get $\min\{s(i + 1, j), s(i, j - 1)\}$ points in the rest of the game.

Now, let's consider Alice's decision. If Alice takes card i, she will get p_i points immediately, and in the rest of the game get $\min\{s(i + 2, j), s(i + 1, j - 1)\}$ points by our previous logic. Similarly, if Alice takes card j, she will get p_j points immediately, and in the rest of the game will get $\min\{s(i + 1, j - 1), s(i, j - 2)\}$ points. Alice will choose whichever card maximizes the points she gets, so putting it all together, we get:

$$s(i, j) = \max\{p_i + \min\{s(i + 2, j), s(i + 1, j - 1)\}, p_j + \min\{s(i + 1, j - 1), s(i, j - 2)\}\}.$$

There are $O(n^2)$ values to compute, and each can be computed in $O(1)$ time, so the runtime is $O(n^2)$.

4 Counting Targets

We call a sequence of n integers $x_1, \ldots, x_n$ *valid* if each x_i is in $\{1, \ldots, m\}$.

(a) Give a dynamic programming-based algorithm that takes in n, m and "target" T as input and outputs the number of distinct valid sequences such that $x_1 + \cdots + x_n = T$. Your algorithm should run in time $O(m^2 n^2)$. Note that you can assume $T \leq mn$ since no valid sequences sum to more than mn.

(b) **Challenge:** Give an algorithm for the problem in part (a) that runs in time $O(mn^2)$.

Solution:

(a) We use $f(s,i)$ to denote the number of sequences of length i with sum s. $f(s,i)$ is 0 when $i > 0$ and $s \leq 0$, and $f(s,1)$ is 1 if $1 \leq s \leq m$. Otherwise it satisfies the recurrence:

$$f(s,i) = \sum_{j=1}^{m} f(s-j, i-1)$$

There are a total of mn^2 subproblems since $T \leq mn$ and it takes $O(m)$ time to compute $f(s,i)$ from its subproblems, which leads to an $O(m^2n^2)$ DP algorithm. Our algorithm outputs $f(T,n)$.

(b) Manipulating the recurrence from the previous part, we get:

$$f(s,i) = \sum_{j=1}^{m} f(s-j, i-1) = \sum_{j=0}^{m-1} f(s-1-j, i-1) =$$

$$\left[\sum_{j=1}^{m} f(s-1-j, i-1)\right] + f(s-1, i-1) - f(s-m-1, i-1) =$$

$$f(s-1,i) + f(s-1,i-1) - f(s-m-1,i-1).$$

In other words, we're now exploiting the fact that the sums for $f(s,i)$ and $f(s-1,i)$ differ by two terms, and so rather than recompute $f(s,i)$ from scratch we can just add/subtract these terms from $f(s-1,i)$

Using this recurrence, there are still mn^2 subproblems, but it takes $O(1)$ time to compute $f(s,i)$ from its subproblems, and thus there is a $O(mn^2)$ time DP algorithm.