Recitation 10: Dynamic Programming

1 Avoiding Substrings

Suppose we have the standard 26-letter English alphabet, $\Sigma = \{a, b, \ldots, y, z\}$. Let W_n be the set of strings of length n which do not contain the word "yay":

$$W_n = \{\omega \in \Sigma^n : \omega_i\omega_{i+1}\omega_{i+2} \neq \text{yay},\ \forall i = 1, \ldots, n-2\}.$$

Write a recurrence for $f_n = |W_n|$, including base cases, to count the number of character strings of length n that do not contain the word "yay".

(The notation Σ^n means "the set of any n characters from the alphabet Σ concatenated". So $\{x, y\}^3 = \{xxx, xxy, xyx, xyy, yxx, yxy, yyx, yyy\}$.)

NOTE: this problem is difficult to do as-is, as our recursion relies directly on all smaller values we calculate.

We begin by determining some base cases:

- if $n = 0$, then $W_n = \{\varepsilon\}$, where ε denotes the empty string, and we say that $f_0 = 1$. (Note that sometimes it can be notationally helpful to define base cases such as $n = -1$, which doesn't make sense interpreted as a string length but may be meaningful in a recursion.)
- If $n = 1$, then we have 26 possible values for the first character and $W_1 = \Sigma$, $f_1 = 26$.
- If $n = 2$, then we still have no restrictions on the characters and $W_2 = \Sigma^2$ and $f_2 = 26^2 = 676$.

We now consider $n \geq 3$. We denote a string in Σ^n by $\omega_1\omega_2 \ldots \omega_{n-1}\omega_n$, and break the problem into the first characters plus the rest of the string. We consider several cases:

- $\omega_1 \neq$ y. There are 25 cases in which this happens, and in each one we have no restrictions on the rest of the string, so there are $25|W_{n-1}|$ strings in which this happens.
- $\omega_1 =$ y. In this case, we cannot count all possible strings of length $n-1$ that do not contain "yay" as possible postfixes to ω_1: if we contatenate "y" with "aye" we get "**yay**e" which is not a valid string in W_n, despite "aye" being a valid word in W_{n-1}. Instead, we break this into several additional cases:
  - $\omega_2 \neq$ a, y. There are 24 cases in which the second letter is not an a or y, and in each case we have no restrictions on the remaining $n-2$ letters. Thus, there are $24|W_{n-2}|$ strings in which this case happens.

  - $\omega_2 =$ a. In this case, we have $\omega_1\omega_2 =$ ya, so we know that $\omega_3 \neq$ y. There are 25 remaining possibilities for ω_3, none of which run the risk of spelling out "yay" in $\omega_3\omega_4\omega_5$ for any choice of ω_4 and ω_5, so we have $25|W_{n-3}|$ possible strings starting from ω_3.
  - $\omega_2 =$ y. In this case, we must be careful not to pick a string beginning "ay". We recurse again, paying attention again to cases where $\omega_3 =$a,y:
    * $\omega_3 \neq$ a,y. As above, there are 24 characters that ω_3 can be where we don't have to care about what comes next. This case can happen in $24|W_{n-3}|$ ways.
    * $\omega_3 =$ a. Again, we must ensure that $\omega_4 \neq$ y, and have no further restrictions. This contributes $25|W_{n-4}|$ strings.
    * $\omega_3 =$ y. We have to recurse again in this case ensuring the next character is not an a, and if it is a y selecting it carefully. We continue this recurrence for each time we get another y.

We repeat this pattern until we end with y and are safe to pick anything else: when $\omega_{n-2} =$ y, we have $25f_1$ choices when $\omega_{n-1} \neq a$ and 25 choices when $\omega_{n-1} = a$. We can now write a recurrence for $n \geq 3$:

$$\begin{aligned} f_n &= 25f_{n-1} + 25f_{n-2} + 24f_{n-2} + 25f_{n-3} + 24f_{n-3} + \cdots + 25f_2 + 24f_2 + 25f_1 + 25 \\ &= 25f_{n-1} + 49\sum_{i=2}^{n-2} f_i + 25f_1 + 25 \end{aligned}$$

we leave this as-is, since computing this recurrence is beyond the scope of this class. Thus, our full recurrence is

$$f_n = \begin{cases} 1 & n = 0 \\ 26 & n = 1 \\ 26^2 & n = 2 \\ 25f_{n-1} + 49\sum_{i=2}^{n-2} f_i + 25f_1 + 25 & n \geq 3 \end{cases}$$

2 Trains

You've decided to leave CS to pursue a career in train robbery (it's the next big thing!). You've been observing the train schedules in the Boulder area, and have a pretty good idea of what trains will be running in the next month, and the approximate value of each train's cargo.

Over the next month, you know there will be n trains running in your target area, with train i carrying cargo worth some value v_i. Unfortunately, you expect the law to be close on your heels; you've decided after each heist it's best to lay low and leave the next 2 trains alone to avoid getting caught.

Give a dynamic programming algorithm to determine the maximum amount of loot you'll be able to make off with in the next month.

a. Identify the subproblem to solve.

> On the ith train, we choose whether to 1) rob the train or 2) let it go by.

b. Define a recurrence for V_i, the total value of loot you can boost over trains $i, i+1, \ldots, n$. Include your base cases.

> First, we define the value we get from each of the two choices we can make in our subproblem:
>
> $$\begin{array}{ll} V_{i+1} & \text{if we don't hold up train } i \\ v_i + V_{i+3} & \text{if we do hold up train } i \end{array}$$
>
> Thus, the best we can do on the last i trains is
>
> $$V_i = \begin{cases} \max(V_{i+1}, v_i + V_{i+3}) & 1 \leq i \leq n \\ 0 & i > n, \end{cases}$$
>
> since we get 0 value from robbing trains that don't run (trains after the nth train).

c. Say there are 12 trains running this month, with values

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
20	18	6	8	15	8	4	23	7	9	13	16

Use your recurrence to compute the maximum loot value you can get this month. What is the maximum value? How could you modify this to give a schedule for your train robbery, as well as your optimal value?

We create an array with an entry for each train we might rob, with the ith entry storing V_i. We also add entries for V_{13}, V_{14}, and V_{15} for our base cases, which we initialize to 0 according to our recurrence:

$i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$V_i =$													0	0	0

We fill out our lookup table working backward from $i = n$ according to our recurrence: $V_{12} = \max(0, 16 + 0) = 16$, $V_{11} = \max(16, 13 + 0) = 16$, $V_{10} = \max(16, 9 + 0) = 16$, $V_9 = \max(16, 7 + 16) = 23$, etc.

This gives us the table

$i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$V_i =$	74	72	54	54	54	39	39	39	23	16	16	16	0	0	0

and we read off the maximum value we can rob from $V_1 = 74$.

We can modify the table to add an indicator rob(i) showing whether or not we chose to rob train i:

$i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$V_i =$	74	72	54	54	54	39	39	39	23	16	16	16	0	0	0
rob(i) =	Y	Y	N	N	Y	N	N	Y	Y	N	N	Y	N	N	N

Using these indicators, we can find an ordering of heists to commit by starting at $i = 1$: if rob(i) = Y, we add train i to our to-rob list and skip to $i + 3$. If rob(i) = N, we move to $i + 1$.

This gives us the robbery schedule of trains 1, 5, 8, and 12.