CSCI 3104 Spring 2022
Instructors: Profs. Chen and Layer

Problem Set 7

Due Date ..... March 15
Name ..... **Your Name**
Student ID ..... **Your Student ID**
Collaborators ..... **List Your Collaborators Here**

Contents

1 Instructions

- The solutions **must be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only. Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.
- Posting to **any** service including, but not limited to Chegg, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.

2 Standard 19 - Dynamic Programming: Identify the Precise Subproblems

The goal of this standard is to practice identifying the recursive structure. To be clear, you are **not** being asked for a precise mathematical recurrence. Rather, you are being asked to clearly and precisely identify the cases to consider. Identifying the cases can sometimes provide enough information to design a dynamic programming solution.

2.1 Problem 1

Problem 1. Consider the Stair Climbing problem, defined as follows.

- Instance: Suppose we have n stairs, labeled $s_1, \ldots, s_n$. Associated with each stair s_k is a number $a_k \geq 1$. At stair s_k, we may jump forward i stairs, where $i \in \{1, 2, \ldots, a_k\}$. You start on s_1.
- Solution: The number of ways to to reach s_n from s_1.

Your job is to clearly identify the recursive structure. That is, suppose we are solving the subproblem at stair s_k. What precise sub-problems do we need to consider?

Answer. Let $T(j)$ denote the number of ways to climb from stair s_j to s_n. At s_j, we consider the number of ways to climb from s_{j+i} to s_n, for each $i \in \{1, \ldots, a_j\}$. That is, we consider $T(j+1), T(j+2), \ldots, T(j+a_j)$. □

T(j) : # of ways to climb from s_j to s_n

Hence, our goal is to find T(1).

Since, we can jump at most a_k stairs when we are at s_k.

⇒. Our goal is to measure T(1). we need to consider T(1+1), T(1+2), ... T(1+a_1).

For each case T(j), we need to consider T(j+1), ..., T(j+a_j).

2.2 Problem 2

Problem 2. Fix $n \in \mathbb{N}$. The *Trust Game* on n rounds is a two-player dynamic game. Here, Player I starts with \$100. The game proceeds as follows.

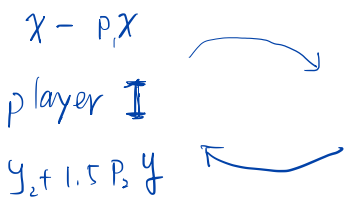
- **Round 1:** Player I takes a fraction of the \$100 (which could be nothing) to give to Player II. The money Player I gives to Player II is multiplied by 1.5 before Player II receives it. Player I keeps the remainder. (So for example, if Player I gives \$20 to Player II, then Player II receives \$30 and Player I is left with \$80).
- **Round 2:** Player II can choose a fraction of the money they received to offer to Player I. The money offered to Player I increases by a multiple of 1.5 before Player I receives it. Player II keeps the remainder.

More generally, at round i, the Player at the current round (Player I if i is odd, and Player II if i is even) takes a fraction of the money in the current pile to send to the other Player and keeps the rest. That money increases by a factor of 1.5 before the other player receives it. The game terminates if the current player does not send any money to the other player, or if round n is reached. At round n, the money in the pile is split evenly between the two players.

Each individual player wishes to maximize the total amount of money they receive.

Your job is to clearly identify the recursive structure. That is, at round i, what precise sub-problems does the current player need to consider? [**Hint:** Do we have a smaller instance of the Trust Game after each round?]

Answer. At round i, the current player may choose to take the entire pile. Otherwise, we have a smaller instance of the Trust Game, in which n is decreased by 1, with the current player as the initial player and the current amount of money in the pile as our initial endowment. □



At round i, the current player has money X.
They can give a fraction p of money, which is pX, where $p \le 1$.
Then, the other player will receive the money with a 50% bonus.
That means they will receive $(1.5p)X$.

When $i < n$

$$T(i) = (1-p_i)X_{i-1} + y_{i-1} + 1.5 p_i X_{i-1}$$
$$= X_{i-1} + y_{i-1} + 0.5 p_i X_{i-1}$$
$$= T(i-1) + 0.5 p_i X_{i-1}$$

$$X_i = (1-p_i)X_{i-1}$$
$$y_i = y_{i-1} + 1.5 p_i X_{i-1}$$

When $i = n$

$$T(i) = \frac{X_{i-1} + y_{i-1}}{2} + \frac{X_{i-1} + y_{i-1}}{2} = T(i-1) = X_i + y_i$$

3 Standard 20- Dynamic Programming: Write Down Recurrences

3.1 Problem 3

Problem 3. Suppose we have an m-letter alphabet $\Sigma = \{0, 1, \ldots, m-1\}$. Let W_n be the set of strings $\omega \in \Sigma^n$ such that ω does not have 00 as a substring. Let $f_n := |W_n|$. Write down an explicit recurrence for f_n, including the base cases. Clearly justify each recursive term.

Answer. If $n = 1$ then every string does not contain a "00"-substring and there are m strings. If $n = 2$ then there is only one possible string containing a "00"-substring and so there are m strings. We discuss cases when $n >= 3$ Here we firstly define w_i as the character at the position i.

if $w_0 \neq 0$ then there are $m-1$ ways to pick w_0 and f_{n-1} ways to pick $w_1 \ldots w_n$.

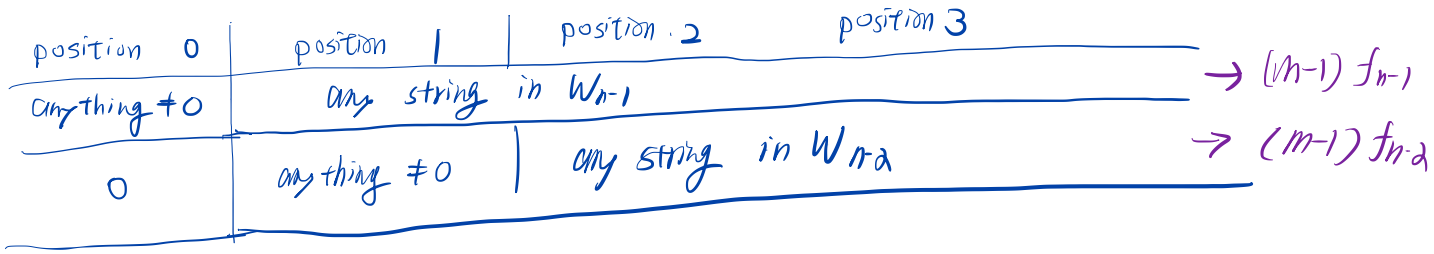
if $w_0 = 0$ and $w_1 \neq 0$ then there is only 1 way to pick w_0, $m-1$ ways to pick w_1, and f_{n-2} ways to pick $w_2 \ldots w_n$.

if $w_0 = 0$ and $w_1 = 0$ then there are 0 strings that do not contain a "00"-substring.

Because each of these cases are *disjoint* (no overlap), we can sum the contribution of each case. The final recurrence is then

$$f_n = \begin{cases} m & : n = 1 \\ m^2 - 1 & : n = 2 \\ (m-1)f_{n-1} + (m-1)f_{n-2} & : \text{otherwise} \end{cases}$$

□



In addition,

$|W_1| = m$

$|W_2| = m^2 - 1$

$f_n = |W_n| = \begin{cases} m, & n = 1 \\ m^2 - 1, & n = 2 \\ (m-1) f_{n-1} + (m-1) f_{n-2}, & n > 2 \end{cases}$

3.2 Problem 4

Problem 4. Suppose we have the alphabet $\Sigma = \{x, y\}$. For $n \geq 0$, let W_n be the set of strings $\omega \in \{x, y\}^n$ where ω contains yyy as a substring. Let $f_n := |W_n|$. Write down an explicit recurrence for f_n, including the base cases. Clearly justify each recursive term.

Answer. If $n < 3$ then every string does not contain a "yyy"-substring and there are 0 strings.

If $n = 3$ then there is only one string that contains a "yyy"-substring.

Otherwise

if $w_0 \neq y$ then there are $2 - 1 = 1$ ways to pick w_0 and f_{n-1} ways to pick $w_1 \dots w_n$.

if $w_0 = y$ and $w_1 \neq y$ then there is only 1 way to pick w_0 and w_1, and f_{n-2} ways to pick $w_2 \dots w_n$.

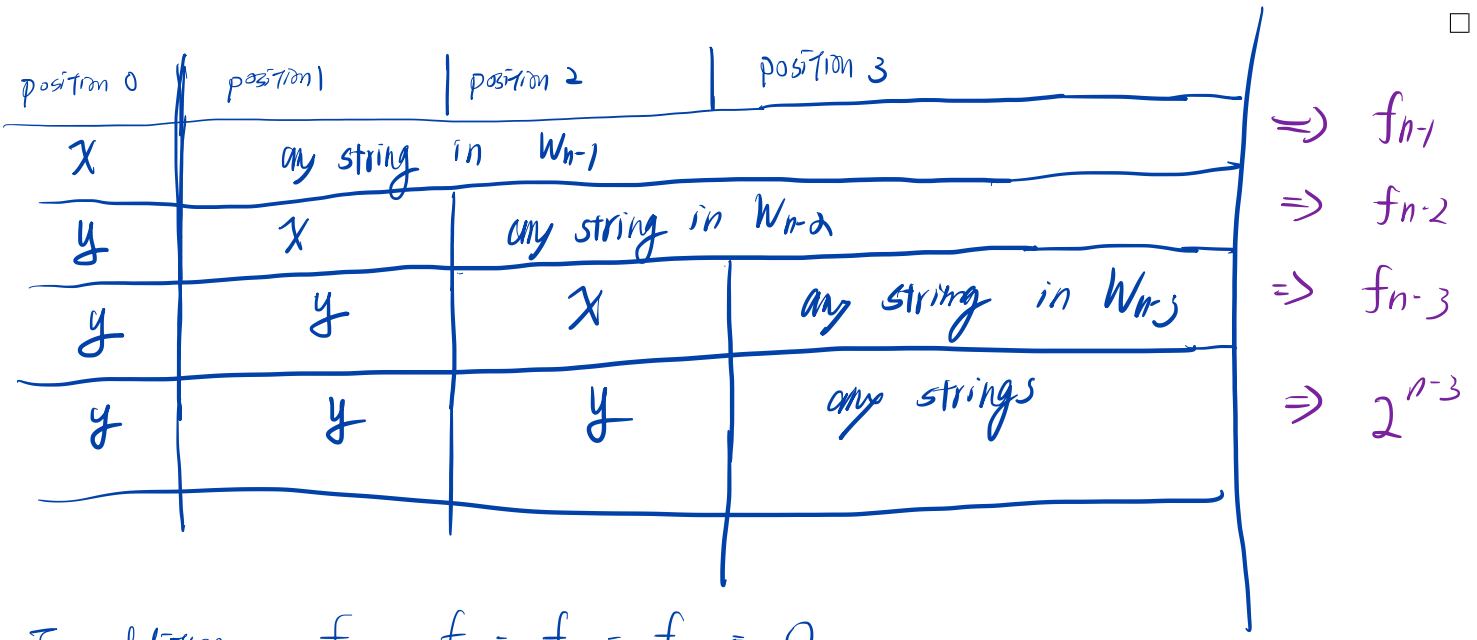
if $w_0 = y$ and $w_1 = y$ ans $w_2 \neq y$ then there is only one way to pick $w_0 \dots w_2$ and f_{n-3} ways to pick $w_3 \dots w_n$

if $w_0 = y$ and $w_1 = y$ ans $w_2 = y$ then there is only one way to pick $w_0 \dots w_2$ and 2^{n-3} ways to pick $w_3 \dots w_n$

Because each of these cases are *disjoint* (no overlap), we can sum the contribution of each case. The final recurrence is then

$$f_n = \begin{cases} 0 & : n < 3 \\ 1 & : n = 3 \\ f_{n-1} + f_{n-2} + f_{n-3} + 2^{n-3} & : \text{otherwise} \end{cases}$$

□



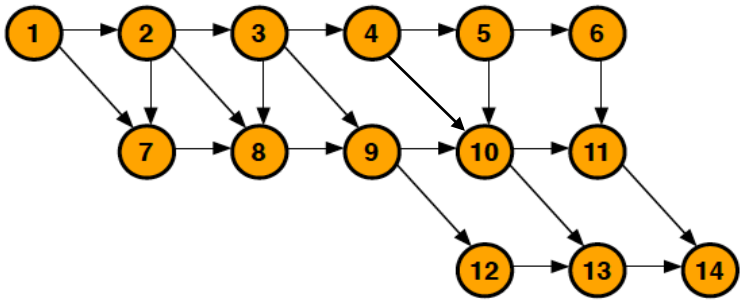
In addition, $f_0 = f_1 = f_2 = f_3 = 0$

$f_3 = 1$

4 Standard 21- Dynamic Programming: Using Recurrences to Solve

4.1 Problem 5

Problem 5. Given the following directed acyclic graph. Use dynamic programming to fill in a **one-dimensional** lookup table that counts number of paths from each node j to 14, for $j \geq 1$. Note that a single vertex is considered a path of length 0. **Fill in the lookup table for all vertices 1-14; and in addition, clearly show work for vertices 9-14**.



Answer. Let $T(i)$ denote the number of paths from vertex i to vertex 14. We explicitly compute $T(9), \ldots, T(14)$. In particular, we do so using a bottom-up approach.

- We first compute $T(14)$. Observe that there is only one path from $14 \to 14$: the path (of length 0) on one vertex. So $T(14) = 1$.
- We now compute $T(13)$. 13 has only one out-neighbor, which is vertex 14, so we have $T(13) = T(14) = 1$.
- Similarly, vertex 11 has only one out-neighbor, which is also 14, so we have $T(11) = T(14) = 1$.
- We now examine vertex 10. Observe that 10 has two outgoing edges: $(10, 11)$ and $(10, 13)$. So:

$$\begin{aligned} T(10) &= T(11) + T(13) \\ &= 1 + 1 \\ &= 2. \end{aligned}$$

- We now examine vertex 12. There is only one outgoing edge from vertex 12: $(12, 13)$. So $T(12) = T(13) = 1$.
- We now examine vertex 9. There are two outgoing edges from vertex 9: $(9, 10)$ and $(9, 12)$. So:

$$\begin{aligned} T(9) &= T(10) + T(12) \\ &= 2 + 1 \\ &= 3. \end{aligned}$$

table on following page

Node	#Path
1	20
2	17
3	11
4	5
5	3
6	1
7	3
8	3
9	3
10	2
11	1
12	1
13	1
14	1

□

4.2 Problem 6

Problem 6. Consider the following input for the Knapsack problem with a capacity $W = 12$:

item i	1	2	3	4	5	6
value v_i	2	7	18	23	29	35
weight w_i	1	2	5	6	7	9

Fill in a lookup table, similar to the example on page 12 of course notes for week 9 (see Week 9 under "Modules" of the course canvas). In addition, clearly explain how you obtain the maximum values/profits $OPT(6, w)$, $w = 7,\ 8,\ 9,\ 10,\ 11,\ 12$.

Answer. We can obtain the following table by applying the rule:

$$OPT(i,w) = \begin{cases} 0 & : i = 0 \\ OPT(i-1,w) & : w_i < w \\ max(OPT(i-1,w), v_i + OPT(i-1, w-w_i)) & : \text{otherwise} \end{cases}$$

weights limit	0	1	2	3	4	5	6	7	8	9	10	11	12
no items	**0**	0	0	0	0	0	0	0	0	0	0	0	0
items 1	**0**	2	2	2	2	2	2	2	2	2	2	2	2
items 1, 2	**0**	2	7	9	9	9	9	9	9	9	9	9	9
items 1, 2, 3	**0**	2	7	9	9	18	20	25	27	27	27	27	27
items 1, 2, 3, 4	**0**	2	7	9	9	18	23	25	30	32	32	41	43
items 1, 2, 3, 4, 5	0	2	7	9	9	18	23	**29**	31	36	38	41	47
items 1, 2, 3, 4, 5, 6	0	2	7	9	9	18	23	**29**	31	36	38	42	47

We can know how to obtain $OPT(6, 7)$ by backtracking from $OPT(6, 7)$ to $OPT(0, 0)$ using the rule. The **bold** numbers are the ones we tracked. This implies that we obtain the $OPT(6, 7)$ by taking the item 5.

weights limit	0	1	2	3	4	5	6	7	8	9	10	11	12
no items	**0**	0	0	0	0	0	0	0	0	0	0	0	0
items 1	0	**2**	2	2	2	2	2	2	2	2	2	2	2
items 1, 2	0	**2**	7	9	9	9	9	9	9	9	9	9	9
items 1, 2, 3	0	**2**	7	9	9	18	20	25	27	27	27	27	27
items 1, 2, 3, 4	0	**2**	7	9	9	18	23	25	30	32	32	41	43
items 1, 2, 3, 4, 5	0	2	7	9	9	18	23	29	**31**	36	38	41	47
items 1, 2, 3, 4, 5, 6	0	2	7	9	9	18	23	29	**31**	36	38	42	47

We can know how to obtain $OPT(6, 8)$ by backtracking from $OPT(6, 8)$ to $OPT(0, 0)$ using the rule. The **bold** numbers are the ones we tracked. This implies that we obtain the $OPT(6, 8)$ by taking the items with indexes 1,5.

weights limit	0	1	2	3	4	5	6	7	8	9	10	11	12
no items	**0**	0	0	0	0	0	0	0	0	0	0	0	0
items 1	**0**	2	2	2	2	2	2	2	2	2	2	2	2
items 1, 2	0	2	**7**	9	9	9	9	9	9	9	9	9	9
items 1, 2, 3	0	2	**7**	9	9	18	20	25	27	27	27	27	27
items 1, 2, 3, 4	0	2	**7**	9	9	18	23	25	30	32	32	41	43
items 1, 2, 3, 4, 5	0	2	7	9	9	18	23	29	31	**36**	38	41	47
items 1, 2, 3, 4, 5, 6	0	2	7	9	9	18	23	29	31	**36**	38	42	47

We can know how to obtain $OPT(6, 9)$ by backtracking from $OPT(6, 9)$ to $OPT(0, 0)$ using the rule. The **bold** numbers are the ones we tracked. This implies that we obtain the $OPT(6, 9)$ by taking items with indexes 2,5.

weights limit	0	1	2	3	4	5	6	7	8	9	10	11	12
no items	**0**	0	0	0	0	0	0	0	0	0	0	0	0
items 1	0	**2**	2	2	2	2	2	2	2	2	2	2	2
items 1, 2	0	2	7	**9**	9	9	9	9	9	9	9	9	9
items 1, 2, 3	0	2	7	**9**	9	18	20	25	27	27	27	27	27
items 1, 2, 3, 4	0	2	7	**9**	9	18	23	25	30	32	32	41	43
items 1, 2, 3, 4, 5	0	2	7	9	9	18	23	29	31	36	**38**	41	47
items 1, 2, 3, 4, 5, 6	0	2	7	9	9	18	23	29	31	36	**38**	42	47

We can know how to obtain $OPT(6, 10)$ by backtracking from $OPT(6, 10)$ to $OPT(0, 0)$ using the rule. The **bold** numbers are the ones we tracked. This implies that we obtain the $OPT(6, 10)$ by taking items with indexes 1,2,5.

weights limit	0	1	2	3	4	5	6	7	8	9	10	11	12
no items	**0**	0	0	0	0	0	0	0	0	0	0	0	0
items 1	**0**	2	2	2	2	2	2	2	2	2	2	2	2
items 1, 2	0	2	**7**	9	9	9	9	9	9	9	9	9	9
items 1, 2, 3	0	2	**7**	9	9	18	20	25	27	27	27	27	27
items 1, 2, 3, 4	0	2	**7**	9	9	18	23	25	30	32	32	41	43
items 1, 2, 3, 4, 5	0	2	**7**	9	9	18	23	29	31	36	38	41	47
items 1, 2, 3, 4, 5, 6	0	2	7	9	9	18	23	29	31	36	38	**42**	47

We can know how to obtain $OPT(6, 11)$ by backtracking from $OPT(6, 11)$ to $OPT(0, 0)$ using the rule. The **bold** numbers are the ones we tracked. This implies that we obtain the $OPT(6, 11)$ by taking items with indexes 3,4.

weights limit	0	1	2	3	4	5	6	7	8	9	10	11	12
no items	**0**	0	0	0	0	0	0	0	0	0	0	0	0
items 1	**0**	2	2	2	2	2	2	2	2	2	2	2	2
items 1, 2	**0**	2	7	9	9	9	9	9	9	9	9	9	9
items 1, 2, 3	0	2	7	9	9	**18**	20	25	27	27	27	27	27
items 1, 2, 3, 4	0	2	7	9	9	**18**	23	25	30	32	32	41	43
items 1, 2, 3, 4, 5	0	2	7	9	9	18	23	29	31	36	38	41	**47**
items 1, 2, 3, 4, 5, 6	0	2	7	9	9	18	23	29	31	36	38	42	**47**

We can know how to obtain $OPT(6, 12)$ by backtracking from $OPT(6, 12)$ to $OPT(0, 0)$ using the rule. The **bold** numbers are the ones we tracked. This implies that we obtain the $OPT(6, 12)$ by taking items with indexes 3,5.

□