CSCI 3104 Spring 2022
Instructors: Profs. Chen and Layer

Problem Set 7

Due Date .... March 15
Name .... **Chengming Li**
Student ID **109251991**
Collaborators .... **N/A**

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 $s_1, \dots, s_n$. Associated with each stair s_k is a number $a_k \geq 1$. At stair s_k, we may jump forward i stairs, where $i \in \{1, 2, \dots, a_k\}$. 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.

- The total steps we need to make is $n - 1$ steps.
- At the each sub problem $stairs_k$, we need to decide how many steps we want to jump. i.e $i \in \{1, 2, \dots, a_k\}$. The step choices at current stair s_k is $\{1, 2, ..., a_k\}$
- And at each sub problem $stairs_k$, we need to decide whether we jump to s_{k+1} or $s_{k+2} or ......s_n$
- For example, in order to reach stair s_6, the steps need to take from stair 5 to 6 is 1.
- At stair 4, the choices need to decide is 2, one choice is $4- > 5- > 6$ and another choice is $4- > 6$. The first path result comes from previous statement.
- Define T_k = number of steps from k to n.
- The recurrence can be expressed as $T_k = T_{k+1} + T_{k+2} + ... + T_{min(n,k+a_k)} + 1$

□

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 each round, the player have three choices as listed below.
- Giving all the money he has to maximize the the total amount of money he may receive if he 100 percent trusts the another player.
- Giving some fraction of the money he has to get some profit from the multiplication if he somehow trusts the another player will pay back.
- Giving nothing to the another player if he already satisfied the money he got.

$\square$

3 Standard 20- Dynamic Programming: Write Down Recurrences

3.1 Problem 3

Problem 3. Suppose we have an m-letter alphabet $\Sigma = \{0, 1, \dots, m-1\}$. 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.

- $f_n = |W_0| = |\Sigma^0| = 0$.
- $f_1 = |W_1| = |\Sigma^1| = m$. This is base case since every letter doesn't repeat. i.e. only occur once in all possible combinations.
- $f_2 = |W_2| = |\Sigma^2| = \{0, 1, \dots, m-1\} \cdot \{1, \dots, m-1\} = m \cdot (m-1) + (m-1)$. The "m-1" term is due to ω does not have 00 as a substring. For the first string,{0, 1, ..., m-1}, we have m combinations. And for each combination in the first string, the second string, $\{1, \dots, m-1\}$, can give(m-1) combinations with that. That is why we have $m \cdot (m-1)$ term as the first term. The first term only eliminates the possibility of the first 0 occurrence. So we need to add another (m-1) term to eliminate the possibility of the second 0 occurrence.
- $f_3 = |W_3| = |\Sigma^3| = \{0, 1, \dots, m-1\} \cdot \{1, \dots, m-1\} \cdot \{1, \dots, m-1\} = (m-1)f_2 + (m-1)f_1$. The "m-1" term is due to ω does not have 00 as a substring.
- Same thing apply to the general case. (m-1) is used to eliminate the possibility of 0 occurrence. And we multiply the result to f_{n-1} combinations, which we already known the solution of W that doesn't have substring 00. And same rule apply to the result to f_{n-2} combinations.

$$f_n = \begin{cases} 0 & : \text{n} = 0, \\ \text{m} & : \text{n} = 1, \\ f_1 \cdot (m-1) + (m-1) & : \text{n} = 2., \\ f_{n-1} \cdot (m-1) + f_{n-2} \cdot (m-1) & : n \geq 3., \end{cases}$$

□

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.

- $F_n = |W_0| = |\Sigma^0| = 0$.
- $F_1 = |W_1| = |\Sigma^1| = 2$. This is base case since every letter doesn't repeat. i.e. only occur once in all possible combinations.
- $F_2 = |W_2| = |\Sigma^2| = 2 * 2$
- $F_3 = |W_3| = |\Sigma^3| = (2-1) \cdot F_2 + (2-1) \cdot F_1 + (2-1)$
- the term (2-1) is used to eliminate the possibility of y occurrence. F_2 with (2-1) term is used to eliminates the possibility of first y occurrence. F_1 with (2-1) is used to eliminates the possibility of second y occurrence. (2-1) term is used to eliminates the possibility of third y occurrence.
- $F_n = |W_n| = |\Sigma^n| = (2-1) \cdot F_{n-1} + (2-1) \cdot F_{n-2} + (2-1) \cdot F_{n-3}$
- the term (2-1) is used to eliminate the possibility of y occurrence. F_{n-1} with (2-1) term is used to eliminates the possibility of first y occurrence. F_{n-2} with (2-1) is used to eliminates the possibility of second y occurrence. F_{n-3} with (2-1) term is used to eliminates the possibility of third y occurrence.

$$F_{n(noyyy)} = \begin{cases} 0 & : \text{n} = 0, \\ 2 & : \text{n} = 1, \\ 4 & : \text{n} = 2, \\ (2-1) \cdot F_2 + (2-1) \cdot F_1 + (2-1) & : \text{n} = 3, \\ (2-1) \cdot F_{n-1} + (2-1) \cdot F_{n-2} + (2-1) \cdot F_{n-3} & : n \geq 4., \end{cases}$$

- f_n have 2^n combinations. Above, I find the number of string that doesn't contain yyy as a substring. Then $2^n - F_n$ is the number of string that does contain yyy as a substring.
- $2^n = F_n + f_n$. So $F_n = 2^n - f_n$

$$f_{n(yyy)} = \begin{cases} 0 & : \text{n} = 0, \\ 0 & : \text{n} = 1, \\ 0 & : \text{n} = 2, \\ 1 & : \text{n} = 3, \\ 2^n - F_{n(noyyy)} & : n \geq 4., \end{cases}$$

$$\begin{aligned} f_n &= 2^n - F_n \\ &= 2^n - F_{n-1} - F_{n-2} - F_{n-3} \\ &= 2^n - (2^{n-1} - f_{n-1}) - (2^{n-2} - f_{n-2}) - (2^{n-3} - f_{n-3}) \\ &= 2^n(1 - 1/2 - 1/4 - 1/8) + f_{n-1} + f_{n-2} + f_{n-3} \\ &= 2^n \cdot 1/8 + f_{n-1} + f_{n-2} + f_{n-3} \\ &= 2^{n-3} + f_{n-1} + f_{n-2} + f_{n-3} \end{aligned}$$

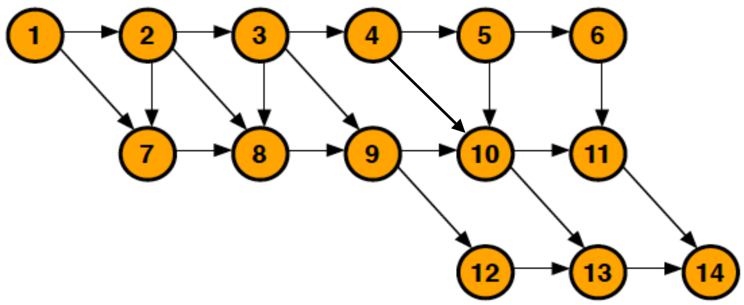
□

$$f_{n(yyy)} = \begin{cases} 0 & : \mathrm{n} = 0, \\ 0 & : \mathrm{n} = 1, \\ 0 & : \mathrm{n} = 2, \\ 1 & : \mathrm{n} = 3, \\ 2^{n-3} + f_{n-1} + f_{n-2} + f_{n-3} & : n \geq 4. \end{cases}$$

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. **Define expression:**
(1,14) = number of paths from node 1 to 14
Example: (11,14) = 1; (10,14) = (11,14) + (13,14) = 1 + 1

From	To 14	result
1	(2,14) + (7,14) = 17 + 3	20
2	(3,14) + (7,14) + (8,14) = 11+3+3	17
3	(4,14) + (8,14) + (9,14) = 5 + 3 + 3	11
4	(5,14) + (10,14) = 3 + 2	5
5	(10,14) + (6,14) = 2 + 1	3
6	(11,14)	1
7	(8,14)	3
8	(10,14) + (12,14) = 2 + 1	3
9	(10,14) + (12,14) = 2 + 1	3
10	(11,14) + (13,14)	2
11	1	1
12	1	1
13	1	1
14	1	1

- At vertex 14, there is 0 path from 14 to 14
- At vertex 13, there is 1 path from 13 to 14. i.e. $13->14$
- At vertex 12, there is 1 path from 12 to 14. i.e $12->13->14$
- At vertex 11, there is 1 path from 11 to 14. i.e. $11->14$
- At vertex 10, there are two branches. path1: $11->14$. path2: $13->14$. So 2 paths from 10 to 14.
- At vertex 9, there are two branches. path1: $10->14$. path2: $12->14$. Path1 result can be found from vertex 10 and path2 result can be found from vertex 12. So, 3 paths from 9 to 14.

□

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.

	0	1	2	3	4	5	6	7	8	9	10	11	12
{}	0	0	0	0	0	0	0	0	0	0	10	11	12
{1}	0	2	2	2	2	2	2	2	2	2	2	2	2
{1,2}	0	7	9	9	9	9	9	9	9	9	9	9	9
{1,2,3}	0	2	7	9	9	18	20	25	27	27	27	27	27
{1,2,3,4}	0	2	7	9	9	18	23	25	30	32	32	41	43
{1,2,3,4,5}	0	2	7	9	9	18	23	29	31	36	38	41	47
{1,2,3,4,6}	0	2	7	9	9	18	23	29	31	36	38	42	47

- In OPT(i,w), i refers to row i in the table, w refers to column w in the table
- OPT(6,7): $w_6 = 9 > 7$.So, OPT(6,7) = OPT(5,7) = 29.
  To obtain OPT(6,7), we know $w_6 = 9 > 7$. So we need to **pick OPT(i-1=5,7)**,based on Knapspack algorithm, which comes from row 5 and column 7 with value 29.
- OPT(6,8): $w_6 = 9 > 8$.So, OPT(6,7) = OPT(5,8) = 31
  To obtain OPT(6,8), we know $w_6 = 9 > 8$. So we need to **pick OPT(i-1=5,8)**,based on Knapspack algorithm, which comes from row 5 and column 8 with value 31.
- OPT(6,9): $w_6 = 9 = 9$.So max(OPT(5,9), v_6 + OPT(5,9-9)) = max(36,35+0) = 36
  To obtain OPT(6,9), we know $w_6 = 9 = 9$. So, based on the algorithm, we need to decide $max(OPT(6-1=5,9), v_6 + OPT(6-1=5, w-w_6 = 9-9=0))$.**I will pick the OPT(5,9) from the table as solution.** The result is $max(OPT(5,9), v_6 + OPT(5,0)) = 36$
- OPT(6,10):$w_6 = 9 < 10$.So, max(OPT(5,10), v_6 + OPT(5,10-9)) = max(38,35+2) = 38
  To obtain OPT(6,10), we know $w_6 = 9 < 10$. So, based on the algorithm, we need to decide $max(OPT(6-1=5,10), v_6 + OPT(6-1=5, w-w_6 = 10-9=1))$.**I will pick the OPT(5,10) from the table as solution.** The result is $max(OPT(5,10), v_6 + OPT(5,1)) = 38$
- OPT(6,11):$w_6 = 9 < 11$.So, max(OPT(5,11), v_6 + OPT(5,11-9)) = max(41,35+7 = 42) = 42
  To obtain OPT(6,11), we know $w_6 = 9 < 11$. So, based on the algorithm, we need to decide $max(OPT(6-1=5,11), v_6 + OPT(6-1=5, w-w_6 = 11-9=2))$.**I will pick the v_6 + OPT(5,2) = 42 from the table as solution.** The result is $max(OPT(5,11), v_6 + OPT(5,2)) = 42$
- OPT(6,12):$w_6 = 9 < 12$.So, max(OPT(5,12), v_6 + OPT(5,12-9)) = max(47,35+9 = 44) = 47
  To obtain OPT(6,12), we know $w_6 = 9 < 12$. So, based on the algorithm, we need to decide $max(OPT(6-1=5,12), v_6 + OPT(6-1=5, w-w_6 = 12-9=3))$.**I will pick the OPT(5,12) from the table as solution.** The result is $max(OPT(5,12), v_6 + OPT(5,3)) = 47$

□