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. When solving this problem we have to consider that there are $s_n - s_k$ stairs left to climb. Since we can go i stairs forward and $i \in \{1, 2, ..., a_k\}$. Then there are a_k sub problems that we need to consider. This is because to find the total number of ways to reach from s_n to s_k we have to sum all possible ways you can make it from s_n to s_k. □

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. We have that at round i, where $i \leq n$, the player at the current round takes a fraction of the money in the pile to send to their counter part and keeps the rest. This means that the current players has three precise sub-problems to consider:

- To take everything from the pile(they give nothing). In the event that the current player chooses this option, the game ends immediately. This could be beneficial IF the current player assume their counterpart will end the game the next round
- To give everything from the pile. In the even that the current player chooses this option, their counterpart would receive the largest amount of money. However, this would only be beneficial if the player assumes that their counterpart will give them a greater total amount back, since the money given is multiplied by 1.5.
- To keep a fraction of the pile. In the event that the current player chooses this option, it would only be beneficial if the player wish to receive a larger sum of money back from their counterpart, assuming that the amount of money they current round player gives is greater than $\frac{2}{3}$ of their current pile, then their counterpart will receive a larger sum, and has the possibility to send a larger sum back.

□

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. We first describe the initial conditions. Observe that $W_0 = \{\Sigma\}$, where Σ is a string of length 0, and $W_1 = \Sigma$. Thus $f_0 = 0$ and $f_1 = 0$.
Now considering a string $w \in W_n$ of length $2 \geq 2$. We note that the first character $w_0 \in \{0, 1, ..., m-1\}$. Consider the following cases:

- If $w_0 = 0$, then necessarily $w_1! = 0$, where we can assume $w_1 \in \{01, 02, ...0(m-1)\}$. Otherwise we would have a 00 sub string. Thus, $w = W_a\tau$, where $\tau \in W_{n-2}$, and w_a can be any word in a set of length $(m-1)$. This means that there are $f_{n-2} = |W_{n-2}|$ ways to select the value of τ. So there are (m-1)f_{n-2} such strings w in this case

- If $w_0! = 0$, then $w_0 \in \{1, 2, ..., m-1\}$, and the second character $w_1 \in \{0, 1, ..., m-1\}$. This means that $w = w_b\tau$, where $\tau \in W_{n-1}$, and w_b can be any word in a set of length $(m-1)$. Thus, we have f_{n-1} such ways to select w in this case

As case 1 and 2 are disjoint, we have that $f_n = (m-1)f_{n-1} + (m-1)f_{n-2}$, with the initial conditions that $f_0 = 1$ and $f_1 = m$ $\square$

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. To find the recurrence relation we first are going to look at all the strings without a substring yyy in them.
Case 1: If x occurs in the first position. So a string of length n looks like x followed by a string of n-1 avoiding yyy.
Case 2: If x occurs in the second position. So a string of length n looks like yx followed by a string of n-2 avoiding yyy.
Case 3: If x occurs in the third position. So a string of length n looks like yyx followed by a string of n-3 avoiding yyy.
Lets say that R_n denotes the number of strings without yyy in them. There fore.

$$R_n = \begin{cases} R_{n-1} + R_{n-2} + R_{n-3} & n > 3 \\ 2 & n = 1 \\ 4 & n = 2 \\ 7 & n = 3 \end{cases}$$

The number of required strings with yyy can be written as $f_n = 2^n - R_n$. By using unrolling we get that $f_n = 2^{n-3} + f_{n-1} + R_{n-2} + f_{n-3}$. The base cases are the following when $n = 0, 1 or 2$ then $f_n = 0$ because there can't be yyy in a string with length less than 3. Then when n=3 there is one possible string and it's yyy so $f_n = 1$. This means the recurrence for f_n is.

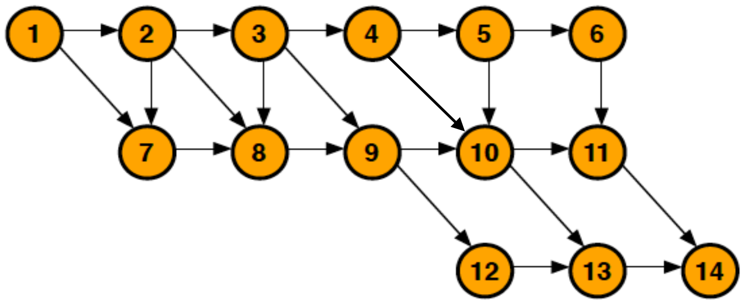
$$f_n = \begin{cases} 2^{n-3} + f_{n-1} + R_{n-2} + f_{n-3} & n > 3 \\ 0 & n = \{0, 1, 2\} \\ 1 & n = 3 \end{cases}$$

□

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.

Vertex	Number of paths	Work
14	1	Same vertex
13	1	$x(13) = x(14)$
12	1	$x(12) = x(13) = x(14)$
11	1	$x(11) = x(14)$
10	2	$x(10) = x(11) + x(13)$
9	3	$x(9) = x(10) + x(12)$
8	3	
7	3	
6	1	
5	3	
4	5	
3	11	
2	17	
1	20	

□

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	0	0	0
{1}	0	2	2	2	2	2	2	2	2	2	2	2	2
{1,2}	0	2	7	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	30	30	41	41
{1,2,3,4,5}	0	2	7	9	9	18	23	29	31	36	36	41	47
{1,2,3,4,5,6}	0	2	7	9	9	18	23	29	31	35	37	42	47

Work:

- w=7) this was 29 because the weight of item 5 is 7 and the value is 29
- w=8) This was item 5 and 1 added together because it equals 8 and the value is maximized
- w=9) This was the weight of item 6 so the value of 35 was added into the row
- w=10) This was the value of item 6 and 1 added together
- w=11) This was the value of item 6 and 2 added together
- w=12) This was the value of item 3 and 5 added together

□