CSCI 3104 Fall 2021
Instructors: Profs. Grochow and Waggoner

Problem Set 8

Due Date ..... TODO
Name ..... **John Blackburn**
Student ID **JOBL2177**
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.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.

- My submission is in my own words and reflects my understanding of the material.
- Any collaborations and external sources have been clearly cited in this document.
- I have not posted to external services including, but not limited to Chegg, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

Agreed John Blackburn. □

3 Standard 21- 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.

3.1 Problem 2

Problem 2. Consider the Stair Climbing problem, defined as follows.

- Instance: Suppose we have n stairs, labeled $s_1, \ldots, s_n$. Each stair s_k has a number $a_k \geq 1$ associated to it. 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. The subproblems that need to be solved to solve subproblem at stair s_k are as follows: we need to know the amount of paths possible to reach each stair prior to stair s_k. That is we need to solve the amount of paths possible to stair s_1 all the way up to stair s_{k-1} individually, in order to solve the subproblem at stair s_k. If we know the amount of possible paths to each stair prior to s_k we can use that information to find the amount of possible paths to stair s_k.

□

3.2 Problem 3

Problem 3. 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 i, the sub-problems to be solved are: whose turn is it to give? how much money is given? Did the giver give \$0? How many rounds are left? Is this the last round? How much money is still in the pile? How much money does each player have locked away?

Once we know these things we can determine the next move the algorithm should take.

□

4 Standard 22- Dynamic Programming: Write Down Recurrences

4.1 Problem 4

Problem 4. 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. □

4.2 Problem 5

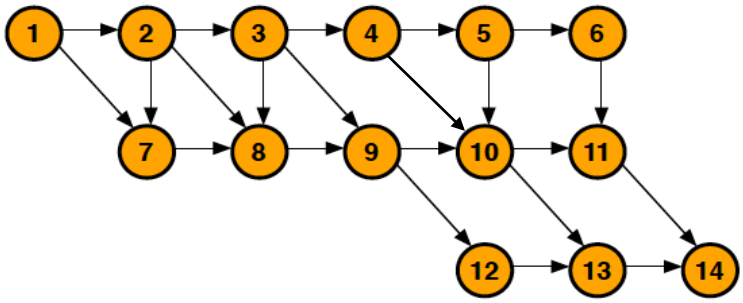
Problem 5. Suppose we have the alphabet $\Sigma = \{x, y\}$. For $n \geq 0$, let W_n be the set of strings $\omega \in \{x, y\}^n$ where ω contains yyy as a substring. Let $f_n := |W_n|$. Write down an explicit recurrence for f_n, including the base cases. Clearly justify each recursive term.

Answer. □

5 Standard 23- Dynamic Programming: Using Recurrences to Solve

5.1 Problem 6

Problem 6. 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. □

5.2 Problem 7

Problem 7. Suppose we wish to multiply an $n \times m$ and $m \times k$ matrix. This operation takes nmk multiplications. Consider the Matrix Product problem, which defined as follows.

- Instance: Matrices $A_1, \ldots, A_n$ with integer entries, where A_i has dimensions $n_i \times m_i$. Note that for $1 \leq i < n$, we necessarily have that $m_i = n_{i+1}$.
- Solution: Compute $A_1 A_2 \cdots A_n$, using the minimum number of integer multiplications.

We note that matrix multiplication is associative; that is, $A(BC) = (AB)C$. So we may parenthesize $A_1 A_2 \cdots A_n$ however we wish, without changing the expression. Certain parenthesizations require fewer multiplications than others. For instance, suppose A is a 10×30 matrix, B is a 30×5 matrix, and C is a 5×60 matrix. Observe that $A(BC)$ requires $(10 \cdot 30 \cdot 60) + (30 \cdot 60 \cdot 5) = 27000$ multiplications, while $(AB)C$ requires $(30 \cdot 5 \cdot 10) + (10 \cdot 5 \cdot 60) = 4500$ multiplications.

For $1 \leq i \leq j \leq n$, denote $m[i, j]$ as the minimum number of multiplications required to evaluate $A_i \cdots A_j$. Note that $m[i, j]$ is given by the following recurrence:

$$m[i,j] = \begin{cases} 0 & : i = j, \\ \min\limits_{i \leq k < j} (m[i,k] + m[k+1,j] + n_i m_k m_j) & : i < j. \end{cases}$$

Here, we have that:

- $m[i, k]$ is the minimum number of multiplications to compute $M_1 := A_i \cdots A_k$. Note that M_1 is an $n_i \times m_k$ matrix.
- $m[k + 1, j]$ is the minimum number of multiplications to compute $M_2 := A_{k+1} \cdots A_j$. Note that M_2 is an $m_k \times m_j$ matrix (here, $m_k = n_{k+1}$).
- $n_i m_k m_j$ is the number of multiplications required to compute $M_1 \cdot M_2$.

Your job is as follows. Suppose we are given matrices of the following dimension:

- A_1: 2×8.
- A_2: 8×9.
- A_3: 9×10.
- A_4: 10×20.
- A_5: 20×6.

Design and fill in a 2D lookup table to compute the minimum number of integer multiplications required to compute $A_1 \cdots A_5$. Include the optimal back pointers. [**Note:** You may hand-draw and embed your lookup table, provided it is legible and we do not have to rotate our screens to read it. All other accompanying work **must** be typed.]

Answer. □