York University
LE/EECS 3101, Fall 2023
Assignment 4

Due Date: November 24th, at 11:59pm

Beware that, when fighting monsters, you yourself do not become a monster; for when you gaze long into the abyss, the abyss gazes also into you ...

Friedrich Nietzsche

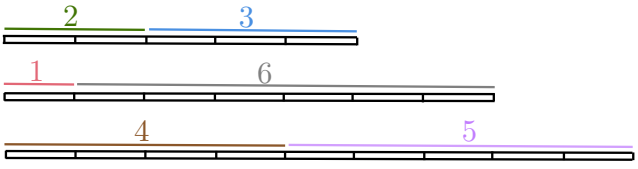
All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution. Please refer to the course webpage for guidelines on academic integrity.
The assignment includes a bonus question. This question is more complex and will be marked more strictly. Approach the bonus only if you have completed other questions.

Problem 1 The Rope Problem [6 marks]

We are given three ropes with lengths n_1, n_2, and n_3. Our goal is to find the smallest value k such that we can fully cover the three ropes with smaller ropes of lengths $1, 2, 3, \ldots, k$ (one rope from each length). For example, as the figure below shows, when $n_1 = 5, n_2 = 7$, and $n_3 = 9$, it is possible to cover all three ropes with smaller ropes of lengths $1, 2, 3, 4, 5, 6$, that is, the output should be $k = 6$.

Devise a dynamic-programming solution that receives the three values of n_1, n_2, and n_3 and outputs k. It suffices to show Steps 1 and 2 in the DP paradigm in your solution. In Step 1, you must specify the subproblems, and how the value of the optimal solutions for smaller subproblems can be used to describe those of large subproblems. In Step 2, you must write down a recursive formula for the minimum number of operations to reconfigure.

Hint: You may assume the value of k is guessed as k_g, and solve the decision problem that asks whether ropes of lengths n_1, n_2, n_3 can be covered by ropes of lengths $1, 2, \ldots, k_g$. Provided with an algorithm for the decision problem, you can find the best value of k using a binary search to find the smallest k_g for which the answer to the decision problem is 'yes'.



Answer: Let $M(m_1, m_2, m_3, \kappa)$ be a boolean variable that indicates whether ropes of lengths m_1, m_2, and m_3 can be covered by ropes of lengths $1, 2, \ldots, \kappa$. Note that we look for $M(n_1, n_2, n_3, k_g)$, where k_g is a guess value. If we know how to evaluate $M(n_1, n_2, n_3, k_g)$, we can use a binary search to find the desired $k$as the smallest k_g for which $M(n_1, n_2, n_3, k_g)$ is true.

To set $M(m_1, m_2, m_3, \kappa)$, we use a DP approach as follows. In the base case, we can cover ropes of negative length, regardless of the value of κ; that is, $M(m_1, m_2, m_3, \kappa)$ is true when $m_1, m_2, m_3 \leq 0$. When one of m_1, m_2, m_3 is larger than 0 and $\kappa = 0$, the value of $M(m_1, m_2, m_3, \kappa)$ is false; this is because we cannot cover a rope of non-negative length with no rope. In the inductive case, we note that the longest rope of length κ can be assigned to the first, second, or third rope. If assigned to the first one, we need to cover ropes of lengths $m_1 - \kappa, m_2$, and m_3 with ropes of length $1, 2, \ldots, \kappa - 1$. Similarly, if assigned to the second (resp. third) rope, we must cover ropes of lengths $m_1, m_2 - \kappa, m_3$ (resp. $m_1, m_2, m_3 - \kappa$) with ropes of lengths $1, 2, \ldots, \kappa - 1$. The value of $M(m_1, m_2, m_3, \kappa)$ is true if any of the recursive cases is true. That is

$$M(m_1, m_2, m_3, \kappa) = M(m_1 - \kappa, m_2, m_3, \kappa - 1) \vee M(m_1, m_2 - \kappa, m_3, \kappa - 1) \vee M(m_1, m_2, m_3 - \kappa, \kappa - 1).$$

In summary, we can write:

$$M(m_1, m_2, m_3, \kappa) = \begin{cases} true & \text{if } \max\{m_1, m_2, m_3\} \leq 0 \\ false & \text{if } \kappa = 0 \textbf{ and } \max\{m_1, m_2, m_3\} > 0 \\ M(m_1 - \kappa, m_2, m_3, \kappa - 1) \vee M(m_1, m_2 - \kappa, m_3, \kappa - 1) \vee M(m_1, m_2, m_3 - \kappa, \kappa - 1) & \text{if } \kappa > 0 \textbf{ and } \max\{m_1, m_2, m_3\} > 0 \end{cases}$$

Problem 2 String Reconstruction [6 marks]

Given two strings S of length n and T of length m, we want to *reconstruct* T from S with a minimum number of the following operations: i) delete a character from S, ii) insert a character into S, or iii) swap two consecutive characters in S. For example, when $S = yorkun$ and $T = ayokru$, one can insert an 'a' at index 1 of S, swap 'r' and 'k' in S, and remove 'n' from the end of S. Therefore, we can reconstruct T from S with 3 operations.

Devise a DP solution to report the minimum number of operations to reconfigure S to T. The actual operations in a minimum-cost reconfiguration are not needed. It suffices to show Steps 1 and 2 in the DP paradigm in your solution. In Step 1, you must specify the subproblems, and how the value of the optimal solutions for smaller subproblems can be used to describe those of large subproblems. In Step 2, you must write down a recursive formula for the minimum number of operations to reconfigure. Assume indices start at 1.

Answer: The minimum number of operations to reconfigure is indeed called **Edit Distance (ED)** between the two sequences. Edit distance has different variants depending on the allowed operations. A subproblem $ED(i, j)$ has two parameters (i, j) and indicates the distance between the prefix of S with length i, denoted with $S[1..i]$, and prefix of T with length j, denoted with $T[1..j]$. We want to find $ED(n, m)$.

To find $ED(i, j)$, we check whether $S[i]$ equals $S[j]$. If it does, then the ED distance between $S[1, i]$ and $T[1, j]$ equals to the distance between $S[1, i - 1]$ and $T[1, j - 1]$, that is, $ED(i - 1, j - 1)$. Otherwise, we either need to:

- Remove the last character of $S[1..i]$. The candidate distance in this case is $1 + LD(i-1, j)$
- Insert the last character of $T[1..j]$ to the end of $S[1..i]$. The candidate distance, in this case, is $1 + LD(i, j-1)$.
- If $S[i] = T[j-1]$ and $S[i-1] = T[j]$ (e.g., $S[1, i]$ ends with ab and $T[1, j]$ ends with ba), we can swap indices $i-1$ and i of S to have the two strings match in their last two characters. The candidate distance, in this case, is $1 + LD(i-2, j-2)$.

Since we are interested in the minimum distance, we must take the minimum value among the above three candidates. Note that if one of the two substrings is empty (base case), then LD constitutes deleting all characters of the other substring. Therefore, we can write:

$$LD(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ LD(i-1, j-1) & \text{if } S[i] = T[j] \\ 1 + \min\{LD(i, j-1), LD(i-1, j)\} & \text{if } S[i] \neq S[j] \text{ and } (S[i] \neq T[j-1] \text{ or } S[i-1] \neq T[j]) \\ 1 + \min\{LD(i, j-1), LD(i-1, j), LD(i-2, j-2))\} & \text{if } S[i] \neq S[j] \text{ and } S[i] = T[j-1] \text{ and } S[i-1] = T[j] \end{cases}$$

Problem 3 Distant Max-Product Subsequence [6 marks]

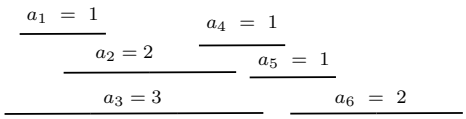
Given a sequence A formed by n positive numbers and a positive integer d, we are interested in a distant max-product subsequence (MPS) of A, which is a subsequence of A formed by elements whose indices are at least d units apart and have the maximum product. Describe a dynamic programming algorithm that reports the product of the MPS of A.
For example, if $A = [2, 10, 12, 9, 1, 3, 5]$ and $d = 2$, the output should be $10 \times 9 \times 5 = 450$. In your solution, it suffices to complete the first two steps of the DP algorithm. That is, define subproblems, describe the optimal for a larger subproblem as a function of the optimal solution for smaller subproblems, and write a recursive formula for the optimal value of a subproblem (remember to include the base case). Assume the indices start at 1.

Answer: The subproblem $MPS(i)$ asks for the value of the maximum MPS of the first i elements. We are interested in $MPS(n)$. The MPS may include or exclude element $A[i]$. In case it is included, $MPS(i)$ is the product of $A[i]$ and $MPS(i-d)$ (since $A[i]$ is included, indices within distance d of i cannot contribute). In case $A[i]$ is excluded, then $MPS[i]$ is simply $MPS[i-1]$. In the base case $MPS[i] = \max\{A[1], A[2], \ldots, A[i]\}$ for $i \leq d$. We can write

$$MPS[n] = \begin{cases} \max\{A[1], A[2], \ldots, A[i]\} & \text{if } i \leq d \\ \max\{MPS[i-1], A[i] \times MPS[i-d]\} & \text{if } i > d \end{cases}$$

Problem 4 Activity Selection by Length [6 marks]

Consider a variant of the activity-selection problem which asks for maximizing the **product of the length** of selected activities instead of the number of selected activities. For example, for the activities in the example below, an optimal solution selects activities a_3 and a_6 with value $3 \times 2 = 6$.



a) Devise a DP program to solve this variant of the activity selection problem. It suffices to complete the first two steps of the DP algorithm. That is, define subproblems, describe the optimal value for a larger subproblem as a function of the optimal value for smaller subproblems, and write down a recursive formula for the optimal value of a subproblem (remember to include the base case). As before, assume activities are sorted by their finish time. You may use $pred(i)$ to denote the index of the last interval that finishes before a_i starts (e.g., $pred(a_6) = a_4$ in the above example). Assume indices start at 1, and let $pred(i) = 0$ if no activity ends before a_i.

Answer: A subproblem $MaxLen(i)$ asks for the optimal solution value for the input formed by the first i intervals. We want to find $MaxLen(n)$. To find $MaxLen(i)$, we note that a_i is selected or not in an optimal solution.

- If a_i is selected, then no other activity j that intersects a_i (that is $s_i < s_j$) may be accepted. The value of the optimal solution will be $(f_i - s_i) \times MaxLen(pred(a_i))$.
- If a_i is not selected, then the value of the optimal solution is trivially $MaxLen(i-1)$.

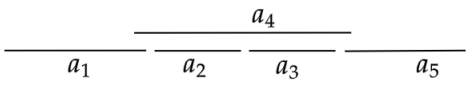
Since we want to maximize the length, we have to select the candidate that gives the maximum value. In the base case, when $i = 0$ (empty set of activities), we have $MaxLen[i] = 0$. We can write:

$$maxLen[i] = \begin{cases} 1 & \text{if } i = 0 \\ \max\{(f_i - s_i) \times maxLen[pred[i]], maxLen[i-1]\} \end{cases}$$

It is ok to write a_i instead of $f_i - s_i$ (given the way the figure is presented).

b) Emma suggests the following greedy algorithm for the problem: repeatedly select the longest interval that does not overlap previously-selected intervals. Prove or disprove whether Emma's algorithm is optimal. You must either prove that the problem admits greedy-choice property or present a counter-example in which Emma's algorithm is not optimal.

Answer: The following counter-example shows that the problem does not admit greedy-choice property. Suppose $a_4 = 4$ and $a_i = 2$ for $i \neq 4$. Emma's algorithm selects a_4 for a profit of 4, while the optimal solution must select all intervals except for a_4 for a profit of 16.

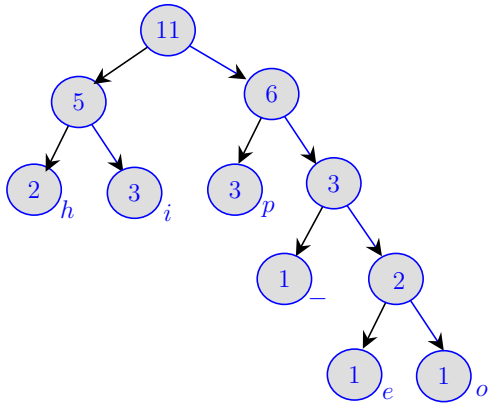


Problem 5 Huffman Encoding [6 marks]

a) Apply the Huffman encoding to encode the following string over alphabet $\Sigma = \{e.h, i, o, p, \_\}$. When selecting two trees with minimum frequencies, break ties by selecting the tree whose leftmost leaf appears earlier in the above ordering of characters in the alphabet. Also, assume the tree with a smaller frequency forms the left child of the merged tree (and in case of equal frequencies, the subtree whose left-most leaf appears earlier in the alphabet forms the left child).

hippie_hipo

Your solution must include the (final) Huffman tree and its code for the input sequence.
Answer: The frequencies are: $e : 1, h : 2, i : 3, o : 1, p : 3, \_ : 1$. The Huffman tree is formed as follows:



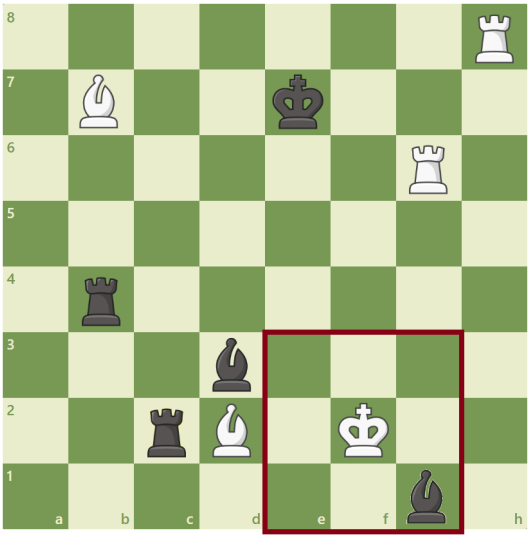
The code be 00 01 10 10 01 1110 110 00 01 10 1111

b) Use the same Huffman encoding to decode the following code: 0001110001111101110
Answer: *hi_hope*

Problem 6 Bonus Question [6 marks]

In the year 1352, during the years of the Black Death, Antonius Block played a game of chess with the angle Death[1]. Antonius had white pieces and the Death had black pieces. We want to know the exact configuration of the board at the end of the 30th move. We know that both players had exactly a king, two bishops, and two rooks at that moment of the game (all other pieces were captured). Our tool is to send "queries" back in time to inquire the pieces located in a query box. In the example below, the query returns "the king of Antonios and dark-square bishop of Death". Note that the query answer does not specify the location of the pieces, only what they are.

Use a decision tree approach to derive a lower bound for the number of required to retrieve the exact configuration. Note that there is no n in this problem; your final answer must be an actual integer. Show your steps and line of thought[2].



Answer: (sketch)

First, we need to find the number of possible configurations. That is, the number of possible positioning of pieces.

There are 32 potential squares for the white-square bishop of Antonius, and 31 potential squares for the white-square of the Death. Similarly, there are 32 potential squares for the black-square bishop of Antonius, and 31 potential squares for the black-square bishop of the Death. This means there are $32 \times 31 \times 32 \times 31$ ways to position the four bishops. The king of Antonius can be in any of the remaining 60 squares, and the king of Death in any of the remaining 59 (after positioning the first king). That is, after placing bishops, there are 60×59 possible ways to position kings. The two rooks of Antonius can be located in any of

[1] Check `https://en.wikipedia.org/wiki/The_Seventh_Seal`

[2] If you are not familiar with chess, you can find its basics here: `https://www.chess.com/learn-how-to-play-chess`.

the remaining 58 positions, there are $\binom{58}{2}$ ways to place them. After that, the two rooks of Death can be located in any of the remaining 56 positions, that is, there $\binom{56}{2}$ ways to place them. In total, the number of possible configurations will be:

$$A = (32 \times 31 \times 32 \times 31) \times (60 \times 59) \times \binom{58}{2} \times \binom{56}{2} = 8.8678876\text{e+}15$$

Next, we find the maximum number of possible outcomes for a query. There are two possible outcomes (presence or absence) for each of the kings and the bishops (note that the bishops are distinguishable unlike rooks). So, there are 2^6 possibilities for these pieces. There are three outcomes for the number of rooks of Antonius (0, 1, or 2 rooks in the query box) and three outcomes for the rooks of the Death. In total, there are

$$B = 2^6 \times 3 \times 3 = 576$$

possible query outcomes.

In summary, we have a decision tree with A leaves and at most B branches for each internal node. The height of such tree will be at least $\lceil \log_B A \rceil = \lceil \log A - \log B \rceil = 14$.

In the above solution, we assumed two kings can be side by side in adjacent cells. It is possible to write a (more complicated) solution in which kings are forced to be in non-adjacent cells. In such a solution, it is easier to first locate kings and then bishops (and consider different cases depending on the kings being in light or dark squares).