Assignment 04

Instructions

1. Write your functions according to the signature provided below in a python(.py) file and submit them to the autograder for evaluation.
2. The autograder has a limited number of attempts. Hence, test it thoroughly before submitting it for evaluation.
3. Save your python file as text(.txt) file and submit it to the canvas.
4. If the assignment has theoretical questions, upload those answers as a pdf file on the canvas.
5. Submission on the canvas is mandatory and the canvas submitted text file should be the same as that of the final submission of the autograder. If not, marks will be deducted.
6. No submission on canvas or no submission on autograder then marks will be deducted.
7. Access the auto grader at https://c200.luddy.indiana.edu.

Question 1 - Wavelet Tree

Let A = [a1, a2, a3, ...an] be a given sequence of integers. Implement the wavelet tree data structure to support efficient rank and select queries. Rank returns the count of the desired element in the given range.

- The get_wavelet_level_order function returns all the levels using level order traversal of the Wavelet tree as list of lists.
- If the node is a leaf node, return 'X' as the value, based on the count of the element. Look at the examples for better understanding.
- The rank query returns the count of the given character in the given range of the array. If such a character is not present in the given range, return 0.

Test Cases

```
wv_tree = Wavelet_Tree([6, 2, 0, 7, 9, 3, 1, 8, 5, 4])

wv_tree.get_wavelet_level_order()

[['1001100110'], ['00101', '00110'], ['100', '01', '010', '10'],
['01', 'X', 'X', 'X', '10', 'X', 'X', 'X'],['X', 'X', 'X', 'X']]

wv_tree.rank(7, 3) # 0

Explanation: There is no 7 in the first 3 elements.

wv_tree = Wavelet_Tree([6, 2, 0, 7, 7, 9, 3, 1, 8, 5, 4])

wv_tree.get_wavelet_level_order()

[['10011100110'], ['00101', '000110'], ['100', '01', '0110', '10'],
['01', 'X', 'X', 'X', '10', 'XX', 'X', 'X'], ['X', 'X', 'X', 'X']]

wv_tree.rank(7, 5) # 2
```

```
Explanation - There are two 7's in the first 5 elements.
The two 7's are correctly represented by two X's in the leaf node over 4th level.
```

Constraints

- Each integer is between 0 and 9
- 1 <= position <= number of elements in the array
- The rank query should only be implemented using the built wavelet tree in log(n) time complexity
- The leaf nodes should represent the correct number of X's denoting the count of the element

Function Signature

```python
# Feel free to change the class Node as per your requirement
class Node:
    def __init__(self, data, left = None, right = None):
        self.data = data
        self.left = left
        self.right = right

class Wavelet_Tree:
    def __init__(self, A:list[int]=[]):
        pass

    def get_wavelet_level_order(self):
        # Return level order traversal of the tree.
        pass

    def rank(self, character, position):
        # Return the rank of the given character in the given position range.
        pass
```

Question 2

As a disciple of Doctor Stephen Strange, you are entrusted with the task of unraveling magical mantras. These mantras are encoded in a long string of characters, and legend says that the true power of these mantras can only be unlocked if you decipher them into palindromic sequences. Each mantra is a sequence of characters that reads the same forwards and backwards—a palindrome. Your task is to write a function to unlock the potential of the mantras by dividinng the string into every substring partition that is a palindrome.

Constraints

- 1 <= s.length <= 16
- s contains only lowercase English letters.

Hint: Use a recursive approach to explore all possible partitions of the string.

Example :

Input: s = “aab”

Output: [[“a”,“a”,“b”],[“aa”,“b”]]

Function Signature

```
def palin_break(s: str):
    #Your code goes here.
```

Question 3

In the celestial realms, the archangel Michael was given a divine mission by the Almighty. He needed to weave two sacred scrolls, scroll1 and scroll2, into a new scroll, scroll3, which held the key to unlocking the Gates of Heaven. However, this task wasn't as simple as merely combining the scrolls. Michael had to merge the words from each scroll carefully, ensuring that they blended together in a specific pattern. If he failed to merge them correctly, the sacred sequence would be broken, and the Gates would remain closed.

Gabriel, the archangel of wisdom, explained the challenge to Michael: When combining the 2 scrolls, each scroll can be split into multiple substrings and these substrings have to be alternatively merged together to form scroll3. Additionally, when merging the substrings, the substrings should follow the initial order of their respective scrolls, i.e. their positions and order cannot be altered. If Michael could correctly create the pattern for scroll3 by combining the 2 scrolls, Heaven's gates would open. But if any part of the new scroll didn't follow this interleaving, the sacred plan would fail, and Michael's mission would remain incomplete.

Given scroll1, scroll2 and scroll3 return True if the Gates of Heaven could be opened and False if they cannot be.

Constraints

1. `0 <= scroll1.length, scroll2.length <= 100`
2. `0 <= scroll3.length <= 200`
3. scroll1, scroll2 and scroll3 consist of lowercase letters.

Example:

Input: `scroll1 = 'aacbcc', scroll2 = 'bbbaccaa', scroll3 = 'aacbbbcbacccaa'`
Output: `true`
Explanation: Michael can obtain scroll3 by: Split scroll1 into “aac” + “bc” + “c”, and s2 into “bb” + “ba” + “ccaa”. Interleaving the two splits, we get “aac” + “bb” + “bc” + “ba” + “c” + “ccaa” = “aacbbbcbacccaa”.
Since scroll3 can be obtained by merging scroll1 and scroll2, we return true.

Input: `s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"`
Output: `false`
Explanation: Notice how it is impossible to interleave scroll2 with any other string to obtain scroll3.

Input: `s1 = "", s2 = "", s3 = ""`
Output: `true`

Function Signature

```
def heavenGates(scroll1, scroll2, scroll3):
    return ans
```

Question 4 - Chef's Magical Recipe

Chef is known for his culinary prowess, but now he's encountered a challenge in the kitchen involving binary strings. Chef has a binary string S and he can perform magical transformations on it. There are four possible operations:

1. Replace **01** with **a**.
2. Replace **10** with **b**.
3. Replace **010** with **ab**.
4. Replace **101** with **ba**.

Chef found that depending on the order of operations, different strings can be produced. After transforming the binary string, Chef is left with a string consisting of only **a** and **b**. Now Chef wants to know how many different binary strings could have led to the final string.

Your task is to help Chef by calculating the number of possible initial binary strings that can result in the final string modulo 998244353, as the number of possibilities can be large.

Input Format

- The first line contains an integer T, the number of test cases.
- Each of the next T lines contains a string S which consists of the characters 'a' and 'b'.

Output Format

- For each test case, return the number of possible initial binary strings that can result in the final string modulo 998244353.

Examples

Example 1:

```
Input:
3
ab
aa
abb

Output:
2
1
2
```

Explanation: - **Test Case 1**: The final string "ab" can result from either "0110" or "010".

- **Test Case 2**: The final string “aa” can result only from “0101”.
- **Test Case 3**: The final string “abb” can result from either “011010” or “01010”.

Example 2:

```
Input:
2
aaaa
bbb

Output:
1
1
```

Explanation: - **Test Case 1**: The final string is “aaaa”. The only possible binary string that can lead to “aaaa” is “01010101”.

- **Test Case 2**: The final string is “bbb”. The only possible binary string that can lead to “bbb” is “101010”.

In both cases, there’s only one possible binary string.

Constraints

- $1 \leq T \leq 5 \cdot 10^4$
- $1 \leq |S| \leq 10^5$
- The sum of $|S|$ over all test cases does not exceed $5 \cdot 10^5$
- S consists only of characters ‘a’ and ‘b’
- Time complexity O(n) per test case.

Function Signature

```python
def magical_recipe(T: int, cases: list[str]) -> list[int]:
    # Implement the logic here to return the result for each test case
    return result
```

Question 5 - Lelouch’s Tactical Task Scheduling

Lelouch vi Britannia is working with an advanced CPU system in the Black Knights’ headquarters, where various tasks (labeled with letters from A to Z) need to be completed. Each task represents an operation necessary for the Black Knights’ strategic plans. However, there’s a limitation: the CPU can’t perform the same task consecutively unless a minimum of `n` intervals have passed between them.

Lelouch needs your help to figure out the optimal schedule for these tasks to minimize the overall CPU usage time. The CPU can either execute a task or remain idle if it’s waiting for the gap between two similar tasks to pass.

Given an array of tasks, where each task is labeled with a letter (A to Z), and an integer `n` that represents the minimum gap between two identical tasks, return the **minimum number of CPU intervals** Lelouch will need to complete all the tasks.

Can you help Lelouch complete all the CPU tasks in the most efficient way possible, ensuring that all strategic operations for the Black Knights are completed on time?

Constraints

```
* 1 <= tasks.length <= 104
* tasks[i] is an uppercase English letter.
* 0 <= n <= 100
```

Example 1:

Input: tasks = [“A”,“A”,“A”,“B”,“B”,“B”], n = 2 Output: 8 Explanation: A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

After completing task A, you must wait two intervals before doing A again. The same applies to task B. In the 3rd interval, neither A nor B can be done, so you idle. By the 4th interval, you can do A again as 2 intervals have passed.

Example 2:

Input: tasks = [“A”,“C”,“A”,“B”,“D”,“B”], n = 1 Output: 6 Explanation: A possible sequence is: A -> B -> C -> D -> A -> B.

With a cooling interval of 1, you can repeat a task after just one other task.

Example 3:

Input: tasks = [“A”,“A”,“A”, “B”,“B”,“B”], n = 3 Output: 10 Explanation: A possible sequence is: A -> B -> idle -> idle -> A -> B -> idle -> idle -> A -> B.

There are only two types of tasks, A and B, w

Function Signature

```python
import heapq
from collections import deque
def lelouch_task_schedule(tasks: list[str], n: int) -> int:

    return time
```

Question 6 - Raj’s Trading Showdown

In the heart of Mumbai’s bustling stock market, *Raj Malhotra* finds himself embroiled in a high-stakes trading battle with his arch-rival, *Rahul Singhania*. The BSE is abuzz with buy and sell orders that arrive continuously. Raj needs to process these orders carefully to maximize his profits and stay ahead of Rahul.

Raj receives a series of orders where each order contains a price, an amount, and a type (either buy or sell). The orders come in two types: - **Buy order (0)**: Raj looks for a matching sell order in the backlog that has a price smaller than or equal to the buy order’s price. If a match is found, the orders are executed, and the remaining part (if any) is added to the backlog. - **Sell order (1)**:

Raj checks for a matching buy order with a price greater than or equal to the sell order's price. If a match exists, the orders are executed.

Orders that cannot be matched remain in the backlog. Your task is to help Raj by keeping track of all unmatched orders in the backlog. After processing all the orders, return the total number of orders in the backlog, modulo 10^9 + 7.

Example 1:

```
Input: orders = [[10,5,0],[15,2,1],[25,1,1],[30,4,0]]
Output: 6
```

Explanation:
- Raj gets a buy order for 5 stocks at 10. No matching sell orders, so the 5 orders are added to the backlog. - A sell order for 2 stocks at $15 comes in. No buy orders with prices higher than $15, so this sell order is added to the backlog. - Another sell order for 1 stock at $25 is placed. Again, no buy orders match, so this order is added to the backlog. - Raj gets a buy order for 4 stocks at $30. Two stocks match the sell order at $15 and one matches the sell order at $25. The remaining 1 buy order is added to the backlog.

In the end, there are 5 buy orders at $10 and 1 buy order at $30, making a total of 6 orders.

Example 2:

```
Input: orders = [[7,1000000000,1],[15,3,0],[5,999999995,0],[5,1,1]]
Output: 999999984
```

Explanation:
- A massive sell order for 1 billion stocks at $7 comes in. No matching buy orders, so the entire sell order is added to the backlog. - Raj gets a buy order for 3 stocks at $15. It matches 3 sell orders at $7, reducing the backlog. - Another buy order for 999,999,995 stocks at $5 is placed. No match for this buy order, so it's added to the backlog. - A sell order for 1 stock at $5 matches a buy order at $5, removing 1 from the backlog.

Finally, the backlog has (1,000,000,000 - 3) sell orders at $7 and (999,999,995 - 1) buy orders at $5. The total is 1,999,999,991, which is 999,999,984 when taken modulo 10^9 + 7.

Constraints:

- 1 <= orders.length <= 10^5.
- orders[i].length = 3.
- 1<= price,amount <= 10^9.
- orderTypei is either 0 buy or 1 sell.

Function Signature

```python
def rajsTradingShowdown(orders: List[List[int]]) -> int:
    # Your Code Here...
```

Question 7

In the middle of the semester, the Professor was overwhelmed with paperwork and unread research proposals, while the stressed Teaching Assistants (TAs) dealt with midterm grading and students flooding their office hours.

One day, the Professor burst into the TA office, brandishing a complex diagram. "I need your help with this!" he declared, slamming the paper onto the desk.

The TAs gathered around, puzzled. "What's this?" asked TA1.

"It's a challenge!" the Professor explained. Each cell in the grid represents either an obstacle, an empty space, or an item to collect, with the numbers indicating their value. **You must collect them in ascending order, starting from the top-left corner, while avoiding obstacles**. With that, he left, leaving the TAs in panic.

TA2 examined the diagram:

- 0 = obstacle (cannot cross)
- 1 = empty space (can move through)
- Any number > 1 = item to collect in ascending order "This is impossible!" exclaimed TA3. "We're already swamped!"

Suddenly, TA4 had an idea. "What if we ask the students for help?"

TA5 replied, "Great idea! We can promise them lenient grading if they assist us!"

Gathering a group of students, the TAs explained the task. "Think of this grid as a maze," TA6 said. **"Collect the items in the right order, while minimizing steps. You can move in any of the four directions: north, east, south, and west. If you can't collect all the items, you should return -1."**

One student asked, "What's in it for us?"

TA7 smiled. "You'll get lenient grading on your midterms. Help us with this, and we'll help you in return."

Now, brave students, can you assist the TAs by solving this challenge?

Example:

Example 1:

```
Input: matrix = [[1,2,3],[0,0,4],[7,6,5]]
Output: 6
Explanation: 1->2->3->4->5->6->7, following the path above allows you to collect
items from smallest to largest in 6 steps.
```

Example 1:

```
Input: matrix = [[1,2,3],[0,0,0],[7,6,5]]
Output: -1
Explanation: The items in the bottom row cannot be collected as the middle row
is blocked.
```

Function Signature

```python
def collect_items(matrix: list[list[int]]) -> int:
    pass
```

Question 8

Flash had a problem to solve: messages in Central City were scrambled. To fix it, he needed to sort the characters in each message based on how often they appeared.

First, Flash counted how many times each character showed up. Then, he sorted the characters so the most frequent ones came first. If two characters appeared the same number of times, Flash used their ASCII values to decide the order.

Constraints

- 1 <= s.length <= 50000
- s consists of uppercase and lowercase English letters and digits.

Example:

```
Input: s = "tree"
Output: "eert"
Explanation: 'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. 'r' is smaller than 't', so
it appears first.
```

```
Input: s = "cccaaa"
Output: "aaaccc"
Explanation: Both 'c' and 'a' appear three times. As 'a' is smaller than 'c',
it appears first.
Note that "acacac" is a wrong answer.
```

```
Input: s = "Aabb"
Output: "bbAa"
Explanation: "bbAa" is the valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

Function Signature

```python
 def frequencySort(s: str) -> str:
    return result
```

Question 9 - Smoky Mountains Trip

It's mid-October, a perfect time to enjoy the stunning fall colors in the Smoky Mountains. John, Jack, and their group of friends, all students from Indiana University Bloomington (IUB), are excited to plan their trip. Given that IUB students get a significant discount at a specific hotel, all groups from the university have decided to stay there. However, the hotel enforces a "one person per room" policy, which makes managing room assignments challenging.

The hotel manager has a list of all students, along with their respective arrival and departure dates. To efficiently allocate rooms, the manager needs to follow these rules:
A room can only be shared if the first student's departure date is strictly before the second student's arrival date.

Your task is to help the hotel manager figure out the minimum number of rooms required to accommodate all students while adhering to the hotel's policy.

Constraints

1 <= number of students (n) <= 200000
1 <= arr <= dep <= 10^9

n lines - each having an arrival day and a depature day

Example:

Input: 1 2 2 3 4 4 Output: 2

Explanation:

The first and third students can share one room. The second student needs a separate room because their departure date overlaps with the first student's departure.

Function Signature

```
    def min_rooms_required(arrivals_departures: List[Tuple[int, int]]) -> int:
        # pass
```

Question 10 - Huffman Coding

You recently got hired at WhatsApp, where they are constantly working to improve their end-to-end encryption. To make messages more efficient and secure, WhatsApp has decided to implement Huffman coding as part of their encryption process. This will help compress the messages before encrypting them, making the messages smaller and faster to send.

As part of your first task at WhatsApp, you were assigned to work on this Huffman coding system. While reviewing the project, you came across some partially written code that outlines the basic structure of the Huffman coding class. Your job is to complete this code to help WhatsApp implement message compression.

Below is the code you found, and it needs to be implemented:

Function Signature

```
class Huffman():
  def __init__(self):
    self.huffman_codes = {}
    self.source_string = ""

  def set_source_string(self, src_str):
    self.source_string = src_str
  def generate_codes(self):
    huffman_codes = {}
    # Code to generate Huffman codes needs to be written here
    self.huffman_codes = huffman_codes

  def encode_message(self, message_to_encode):
    encoded_msg = ""
    # Code to encode the message needs to be written here
    return encoded_msg

  def decode_message(self, encoded_msg):
    decoded_msg = ""
    # Code to decode the message needs to be written here
    return decoded_msg
```

Before starting, your project manager conducted a KT session to ensure you understood the requirements and implementation plan clearly. In the KT, the manager explained you how the encryption must work inorder to protect the users' privacy while maintaing efficient data transmission. It goes as follows:

First the user message is stored as source_string. Then it must compute the frequencies of characters in the source_string. Subsequently, they will invoke the generate_codes function to create Huffman codes, which will be stored in the huffman_codes attribute within the Huffman class. To encode or decode any message, they will utilize the encode_message and decode_message functions, respectively, leveraging the huffman_codes attribute.

Other important points discussed are: - While generating the Huffman codes, if two or more symbols have same frequency choose the one that is lexicographically smaller. (This is to make sure you have same codes as that of autograder) - When you move to a left child, append '0' to the current Huffman code. (This is to make sure you have same codes as that of autograder) - When you move to a right child, append '1' to the current Huffman code. (This is to make sure you have same codes as that of autograder) - Do not touch the **init** and set_source_string functions (The source_string attribute will be set by the Autograder) - generate_codes function will generate huffman codes for all the alphabets and store them in the huffman_codes attribute - encode_message will encode the input strring using the codes from the huffman_codes attribute which was set by generate_codes - decode_message will decode the input strring using the codes from the huffman_codes attribute which was set by generate_codes

Now it's your task to complete this Huffman coding implementation so that WhatsApp can efficiently compress and encrypt messages, contributing to their robust end-to-end encryption system!

Constraints

- All the strings will only contain lowercase English alphabets
- encoded_msg will only contain '0's and '1's

Test Cases:

Example:

If the source_string = "cbadeeee", generate_codes() will create the huffman codes. The huffman tree will look like below:

```
                              [abcde: 8]
                               /       \
                          [abcd: 4]    [e: 4]
                          /       \
                    [ab: 2]       [cd: 2]
                     /   \         /    \
                [a: 1]  [b: 1]  [c: 1]  [d: 1]
```

As discussed in the KT, left childs append 0s and right child append 1s. The huffman_codes attribute will be:

```
    {
      a: '000',
      b: '001',
      c: '010',
      d: '011',
      e '1'
    }
```

Now if encode_message('ae') is called it will return '0001' and if decode_message('0100011011') is called it will return 'cbed'