Assignment 06

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 - Longest Uniform Segment

In a factory, a conveyor belt transports boxes labeled with lowercase English letters. Due to quality control requirements, it's necessary to create a uniform segment of boxes labeled with the same letter. However, some boxes can be relabeled if needed.

You are given a string boxes where each character represents the label of a box on the conveyor belt, and an integer changes, which represents the maximum number of boxes you can relabel.

Your task is to determine the length of the longest possible segment of boxes that can be made uniform (all boxes having the same label) after making at most changes modifications.

Test Cases

```
Input: boxes = "aabbcc", changes = 6
Output: 6
Explanation: Any of either "aa," "bb," or "cc" can all be changed to either "aa,"
"bb," or "cc" to get all six boxes with the same characters.
```

```
Input: points = boxes = "abcbbab", changes = 2
Output: 6
Explanation: The third box labelled "c" and the sixth box labelled "a" both can be
changed to "b" to get six consecutive boxes with same characters.
```

Constraints

- 1 <= boxes.length <= 1000
- 0 <= changes.length <= boxes.length

Function Signature

```python
def longestUniformSegment(boxes: str, changes: int) -> int:
    pass
```

Question 2

Ella and her best friend, Maya, are avid treasure hunters who enjoy exploring ancient maps and finding hidden treasures. They have recently discovered two pieces of parchment that seem to contain part of an old treasure map. However, the two pieces are incomplete and slightly different from each other, each one showing fragments of paths and clues. They need to find the longest common path fragment in both parchments, as they believe that the common part might reveal a vital section of the original map leading to the treasure.

Both parchments contain a sequence of numbers, each representing different landmarks or coordinates on the map. Ella and Maya know that finding the longest shared sequence between the two parchments will bring them closer to understanding the map's original path. They decide to compare both sequences and locate the longest sequence of landmarks that appears in both parchments. They hope that by following this common path, they will discover clues to locate the treasure's resting place.

Now, Ella and Maya have a challenge ahead: they must analyze the two parchment sequences to find the longest shared series of landmarks. They gather around, excited to calculate and trace the path. If they find the common path, they'll mark it as the next part of their adventure. Can they uncover the hidden treasure with this shared piece of the ancient map?

Constraints

1. 1 <= parchment1.length, parchment2.length <= 1000
2. 0 <= parchment1[i], parchment2[i] <= 100

Example:

Input: `parchment1 = [5, 7, 8, 7, 5, 6], parchment2 = [8, 7, 5, 3, 9]`
Output: `3`
Explanation: The repeated sequence with the maximum length is [8, 7, 5].

Input: `parchment1 = [1, 1, 1, 1, 1], parchment2 = [1, 1, 1, 1, 1]`
Output: `5`
Explanation: The repeated sequence with the maximum length is [1, 1, 1, 1, 1].

Function Signature

```
def treasureHunt(parchment1, parchment2):
    return ans
```

Question 3 - Armin's Scout Regiment Formation

Armin Arlert is analyzing a sequence of scout formations, represented by a binary array `nums`. Each formation in `nums` is either a defense squad (`0`) or an attack squad (`1`). Armin's goal is to find the maximum length of a contiguous segment within this formation where there is an equal number of defense and attack squads.

Help Armin by finding this maximum balanced segment length, where the number of defense and attack squads is equal. Since time is of the essence, time complexity of the algorithm should be o(n).

Return the maximum length of this balanced formation.

Constraints

```
* 1 <= nums.length <= 10^5
* nums[i] is either 0 or 1
* Time complexity should be o(n)
```

Example 1:

Input: nums = [0,1] Output: 2 Explanation:

[0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

Example 2:

Input: nums = [0,1,0]

Output: 2 Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

Function Signature

```
def findMaxLength(nums:list[int]) -> int:
    return ans
```

Question 4 - Arkham Asylum

Batman is patrolling Arkham Asylum, where his enemies—the Joker, Two-Face, Poison Ivy, and others—are locked up. Unknown to Batman, the villains have been hatching a plan to escape! Batman manages to catch some clues: an array representing the energy levels of the villains' cell security systems. However, due to interference from the Joker, some energy values are positive, some negative, and some neutral.

When the sum of energy levels over consecutive cells equals zero, it indicates a breach in security, potentially allowing villains to escape!

Help Batman identify all such "security breaches" (zero-sum subarrays).

Constraints

- Solve the question using O(n) time complexity
- 1 < Length of array < 1000

Example:

```
Input: levels = [3, 4, -7, 1, 2, -6, 4, -1]
Output: [(0, 2), (1, 4), (4, 6), (0, 7), (3, 7)]
```

```
Input: points = [6, 3, -1, -3, 4, -2, 2, 4, 6, -12, -7]
Output: [(2, 4), (2, 6), (5, 6), (6, 9), (0, 10)]
```

Function Signature

```
def breaches(levels : List[int]):
    return result
```

Question 5 - Remove Duplicate Itineraries

You are given a list of travel itineraries represented as a 2D array, where each itinerary is a path starting with the traveler's name followed by a sequence of cities they visit. These itineraries resemble a hierarchical tree structure, with the traveler's name as the root node and cities forming the subsequent branches.

Goal

The goal is to identify and remove duplicate itineraries. An itinerary (or any part of it) is considered a duplicate if: 1. It has the same structure (same sequence of cities) as another itinerary, even if the traveler's name differs. 2. All its sub-branches (sub-itineraries) are also identical.

Once duplicate itineraries are detected, both the original and their duplicates (including sub-itineraries) should be removed.

The function should return the remaining unique itineraries in any order.

Hierarchy Representation

Itineraries can be visualized as folders in a file system, where: - The **traveler's name** represents the root folder. - Each **city** in the itinerary is a subfolder under the root.

Example Input

```
[
    ["John"],
    ["John", "Chicago"],
    ["John", "Chicago", "New York"],
    ["Priya"],
    ["Priya", "Chicago"],
    ["Priya", "Chicago", "New York"]
]
```

Example Visualization

```
John
 |___ Chicago
      |___ New York
Priya
 |___Chicago
      |___ New York
```

If two itineraries (e.g., "John" and "Priya") have the same cities arranged in the same structure, they are duplicates. The entire substructure for "Chicago → New York" will also be marked as a duplicate and removed.

Rules for Detecting Duplicates

1. **Identical Structures**: Two or more itineraries are identical if their city sequences match at every level of the hierarchy. For example:

   ```
   /John/Chicago/New York
   /Priya/Chicago/New York
   ```

   These are duplicates and should be removed entirely.

2. **Sub-Itinerary Duplication**: Sub-itineraries are also considered when detecting duplicates. If part of an itinerary is identical to another, the sub-itinerary is marked as a duplicate and removed.

3. **Divergent Paths**: Itineraries with some shared structure but differing sub-paths are not duplicates. For example:

   ```
   /Alice/Paris/Rome
   /Bob/Paris/London
   ```

   These share the same root "Paris" but differ afterward and are considered unique.

Output

The function should return the list of remaining itineraries after removing all duplicates. Each itinerary is represented as an array starting with the traveler's name and followed by the sequence of cities.

Examples

Example 1: Simple Identical Itineraries

Input:

```
[
    ["John"],
    ["Priya"],
    ["John", "Chicago"],
    ["Priya", "Chicago"],
    ["John", "Chicago", "New York"],
    ["Priya", "Chicago", "New York"]
]
```

Visualization:

```
John
 |__ Chicago
      |__ New York
Priya
 |__ Chicago
      |__ New York
```

Explanation: - "John" and "Priya" have identical structures at every level. Both are duplicates.
- Remove all their itineraries.

Output:

```
[]
```

Example 2: Divergent but Unique Paths

Input:

```
[
    ["Alice"],
    ["Bob"],
    ["Alice", "Paris"],
    ["Bob", "Paris"],
    ["Alice", "Paris", "Rome"],
    ["Bob", "Paris", "London"]
]
```

Visualization:

```
Alice
 |__ Paris
      |__ Rome
Bob
 |__ Paris
      |__ London
```

Explanation: - "Alice" and "Bob" share the same root ("Paris") but diverge at the sub-paths ("Rome" vs. "London"). - Both itineraries are unique and remain.

Output:

```
[
    ["Alice"],
    ["Alice", "Paris"],
    ["Alice", "Paris", "Rome"],
    ["Bob"],
    ["Bob", "Paris"],
    ["Bob", "Paris", "London"]
]
```

Example 3: Sub-Itinerary Duplication

Input:

```
[
    ["Alice"],
    ["Charlie"],
    ["Alice", "Paris"],
    ["Charlie", "Paris"],
    ["Alice", "Paris", "Rome"],
    ["Alice", "Paris", "London"],
    ["Charlie", "Paris", "Rome"]
]
```

Visualization:

```
Alice
 |__Paris
      |-- Rome
      |__ London
Charlie
 |__Paris
      |__ Rome
```

Explanation: - The sub-path "Paris → Rome" is identical under both "Alice" and "Charlie." - The sub-path "Paris → London" exists only under "Alice" and is not a duplicate. - As a result: - - The duplicate sub-path "Paris → Rome" is removed from both "Alice" and "Charlie." - - The unique sub-path "Paris → London" under "Alice" is retained

Output:

```
[
    ["Alice"],
    ["Alice", "Paris"],
    ["Alice", "Paris", "London"]
]
```

Constraints

1. 1 <= number of itineraries <= 2 x 10^4
2. 1 <= length of each itinerary <= 500
3. All city names and traveler names are lowercase English letters.
4. No two identical itineraries are provided in the input unless they are considered duplicates.

Function Signature

```
def deleteDuplicateItineraries(itineraries: List[List[str]]) -> List[List[str]]:
    // Your Code Here ...
```

Question 6 - Simulation

Now it is time for simulations as we end the course. Write the Python function which takes n as input and generates a point whose coordinates lie between-1 and 1 for n number of times. You can use the random library of Python to generate the values. Out of these n points find the count of points whose value of x2 + y2 is less than 1 and assign it to the count variable.

1. Report the value of 4*count/n for different values of n.
2. Using libraries of Python plot the n on the x-axis and 4*count/n on the y-axis for different values of n
3. Write your observations for the above graph. Note: Please submit the answer to this question in a PDF file.

The answer in the PDF file should contain the following

1. Python code that is used to generate the table and graph.
2. A table containing n and 4*count/n values
3. Graph for the above table.
4. Observations regarding the same