Due Date: Fri, Nov 22, 2024, 11:59 PM

CSC 2720: Data Structures
Homework 2

Task 1: Recover a Tree from Preorder Traversal

You are given a string representing a preorder traversal of a binary tree with depth-based encoding. In this encoding:

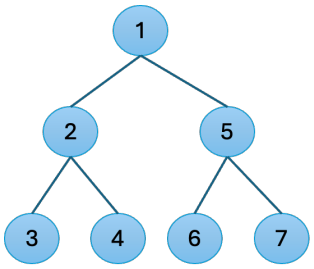
- At each node, D dashes (-) appear before the node's value, where D represents the depth of that node in the tree.
- The depth of the root node is 0, so it has no dashes before its value.
- If a node has only one child, that child is guaranteed to be the left child.

Write a function that reconstructs the binary tree from this traversal string and returns the level order traversal of the tree.

Example1.

Input: traversal = "1-2--3--4-5--6--7"

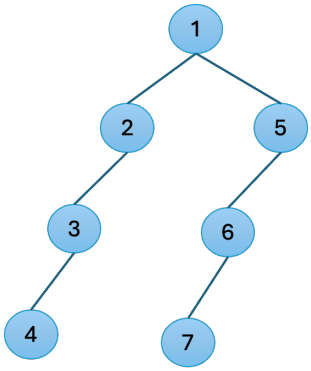
Output: The level order traversal of the reconstructed tree. Below is the reconstructed tree from the given input. The level order traversal should be [1, 2, 5, 3, 4, 6, 7].



Example 2:

Input: traversal = "1-2--3---4-5--6---7"

Output: Below is the reconstructed tree from the given input. The level order traversal should be [1,2,5,3, null,6, null,4, null,7]



Due Date: Fri, Nov 22, 2024, 11:59 PM

Task 2:

You are tasked with creating a secure system to store and compare user passwords using a hash table. To achieve this, follow these requirements:

1. **Storing Passwords:**
   - Write a function save_password(username, password) that takes a username and password as inputs.
   - Hash the password (using SHA-256) before saving it to a hash table, where the username is the key and the hashed password is the value.
2. **Comparing Passwords:**
   - Write a function compare_password(username, password) to check if a provided password matches the stored password for a given username.
   - This function should take a username and password, hash the input password, and compare it to the stored hashed password in the hash table.

3. **Change Password:**

   - Allow users to change their passwords if they provide their current password correctly.

4. **Implementation Constraints:**

   - Use a Python dictionary as the hash table.
   - Ensure that the hash table only stores hashed passwords, not plaintext passwords.

   - If save_password is called with an existing username, the function should return a message indicating the username is already taken.

```
# Sample function calls and expected outputs

save_password("user1", "mypassword123")
# Expected output: "Password saved for user: user1"

save_password("user1", "newpassword456")
# Expected output: "Error: Username 'user1' is already taken."

save_password("user2", "securepassword456")
# Expected output: "Password saved for user: user2"

compare_password("user1", "mypassword123")
# Expected output: True (correct password)

compare_password("user1", "wrongpassword")
# Expected output: False (incorrect password)

compare_password("user3", "any_password")
# Expected output: "Error: Username 'user3' not found."
```

Task 3:

Extend the concept of 'median maintenance' presented in class to maintain the 25th and 75th percentiles using heaps. Consider using a combination of max-heaps and min-heaps to extract the 25th and 75th percentiles from a stream of data in logarithmic time.

More specifically, write a class named 'PercentileMonitor' with the following methods:

add(self, num): Adds the new number to the appropriate heap. Also, this method is responsible for keeping 'balance' between the heaps. Balancing is needed to access the 25th and 75th percentile efficiently.

get_25th(self): returns the 25th percentile of the data. It should run in O(1) time.

get_75th(self): returns the 75th percentile of the data. It should run in O(1) time.

Example:

Consider the following data stream:

13, 24, 28, 32, 33, 39, 40, 45, 46, 55, 56, 57, 58, 59, 60, 67, 68, 71, 74, 75, 80, 83, 84, 89, 90

At this stage, get_25th() method should return: 40. get_75th() method should return: 74

Please refer to https://tinyurl.com/Task3Homework2 for more details on percentile calculation.

Task 4:

Prepare and submit a video demonstration of your understanding of Topological Sorting and its DFS and stack-based solution. The length of the video should be between 2 and 3 minutes. You are free to include any visuals, text, or code in your video demonstration. Ensure that the video size is within the allowed limit.

Submission Instructions

(Please follow the instructions carefully and submit accordingly.)

- Name your submission files as "FULL_NAME_HW2_Task1.py", "FULL_NAME_HW2_Task2.py", "FULL_NAME_HW2_Task3.py", and "FULL_NAME_HW2_Task4.mp4" (or any other valid video format)
- Submit this file in iCollege folder '**Homework2**'
- Due date: Fri, 11/22/2024 11:59 PM

Due Date: Fri, Nov 22, 2024, 11:59 PM

- Late submission will be accepted until: Mon, 11/25/2024 11:59 PM

The late submission penalty will be determined based on the following formula:

```
PENALTY = 0.4 * NUMBER_OF_HOURS_LATE
```

Examples:

If your submission is 2 hours late, PENALTY = 0.8%
If your submission is 24 hours late, PENALTY = 9.6%
If your submission is 72 hours late, PENALTY = 28.8%

Note:
-All submissions must be made through iCollege. No email submission will be accepted.

Grading Breakdown:

Task	Points
Task 1	30
Task 2	30
Task 3	30
Task 4	10

Note: Add a few comments in your python scripts to explain how your code works. Comments should be meaningful and concise.