Assignment 3

Design of Algorithms [CS212/CS514]

Q1.

Problem Statement:
You have your DSA exam tomorrow which is making you anxious. To relax, you went out for a drink with some friends in a nearby pub. To keep your mind sharp for the next day, your friends proposed you play a game.

You will be given an array of N integers and then your friends will give you K commands. Each of the K commands can be either:

- an "Update" command when your friends want to change the value at a given position in the sequence.
- an "Increment" command when your friends want to increase all the numbers of the sequence in a given range by some value.
- a "Get" command when your friends want you to get the value at a given position.

It is decided that for each wrong answer to a command, you will have to drink a pint of beer. Worried that this might affect your performance in the DSA exam the next day, you decided to write a program to help you win this game in one go. Further, since your friends are impatient, they require you to perform/answer each command in O(log N) time.

Input:
The first line contains N and K, indicating the length of sequence and the number of rounds respectively.
The second line contains N integers X[i] representing the sequence.
Each of the next K lines describe a command as either of the following:

- 'U <position> <new value>' for "Update"
- 'I <left> <right> <add value>' for "Increment"
- 'G <position' for "Get"

Output:
For each "Get" command, output the value in the sequence at that position.

Example:

Input	Output
5 8 1 5 -2 4 3 G 3 I 2 4 3 G 3 U 3 0 G 3 I 1 3 3 G 3 G 2	-2 1 0 3 11

Explanation:

Command	Array
Initially	[1 5 -2 4 3]
After 'I 2 4 3'	[1 8 1 7 3]
After 'U 3 0'	[1 8 0 7 3]
After 'I 1 3 3'	[4 11 3 7 3]

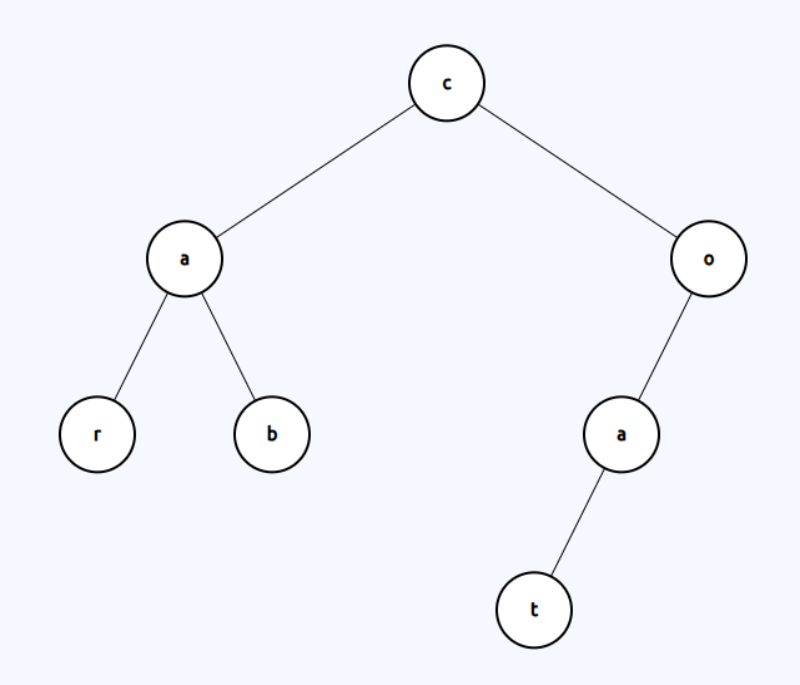
Q2.

Introduction:

A trie, short for "reTRIEval tree," is a tree-like data structure used for efficient retrieval of strings or keys.

In a trie:
1. Each node represents a single character.
2. The path from the root to a node spells out a string.
3. Common prefixes are shared among nodes, reducing space complexity.

Example: A trie of {"car", "cab", "coat"} is like:



Problem Statement:
Imagine you are building a text prediction system for a mobile keyboard application. Your goal is to provide users with real-time word suggestions efficiently as they type. The system should allow users to input partial words and provide auto-complete suggestions based on the words already stored in a dictionary.

Suppose the dictionary contains words like "apple," "appetizer," "application," and "banana." When the user types "app," your system should suggest "apple," "appetizer," and "application" in order of frequency of use.

You may assume that the dictionary only contains lowercase alphabets.

Input:
The first line contains an integer N.
The next N lines each contain a word representing the user's word history.
The last line contains the query pattern/word.

Output:
Output the auto-complete suggestions in decreasing order of frequency of use.

Example:

Input	**Output**
9 apple application apple banana appetizer banana application banana application app	application apple appetizer

Explanation:

Word	**Frequency**
apple application banana appetizer	2 3 3 1

Words matching the prefix “app” are {“apple”, “application”, “appetizer”}.