Assignment 3

Binary Trees and Hash Tables
Data Structures and Algorithms
Due Date: 14 May, 2022

1 Tread Litely

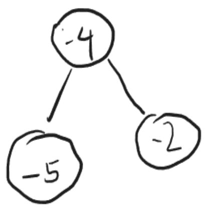
If that's true, if you don't know who I am, then maybe your best course would be to tread lightly.

W.W.

You are waiting for the assignment problems to appear on OJ. To make sure you spend your time constructively, W.W. comes up with the following task.
Define noise of a tree as the sum of values of all its nodes. Find the minimum noise **lite** subtree of a given binary tree.

A **lite** tree is a rooted binary tree whose inorder traversal is always sorted - either in ascending or in descending order.

For e.g.



A **lite** tree with total noise -11.

1.1 Input

The first line contains a single integer $t (1 \leq t \leq 100)$ - the number of testcases.

The first line of each testcase contains a single integer n - the number of elements in the level order traversal array of the tree. (`https://en.wikipedia.org/wiki/Tree_traversal#Breadth-first_search_2`)

The second line of the test case contains $n (1 \leq n \leq 10^4)$ space-separated integers $k_1, k_2, \ldots, k_n$, $(-4 \times 10^4 \leq k_i \leq 4 \times 10^4)$ the elements of the level order array. If $k_i = 0$, it means that the corresponding node is *NULL*. Otherwise, the corresponding node i contains a value k_i. So, a node can take all values $\{-10^4, .., -1, 1, ..., 10^4\}$. Please refer to the description of test cases in the next page for more clarity.

Note: You must delete the tree at the end of each test case and create a new tree for the next test case.

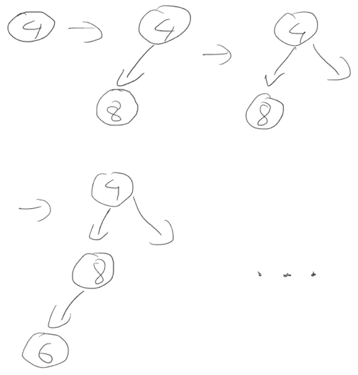
1.2 Output

Print the minimum noise possible in any **lite** subtree of the given tree.

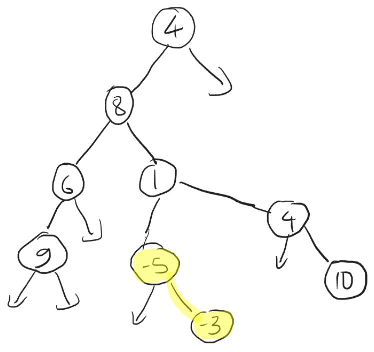
1.3 Example

Input	Output
3	
3	
-4 -5 -2	-11
1	
2	2
15	
4 8 0 6 1 9 0 -5 4 0 0 0 -3 0 10	-8

Let's consider test case 3. The tree will be constructed by sequentially going over the elements of the array like this



Finally, the minimum noise comes out to be -8.

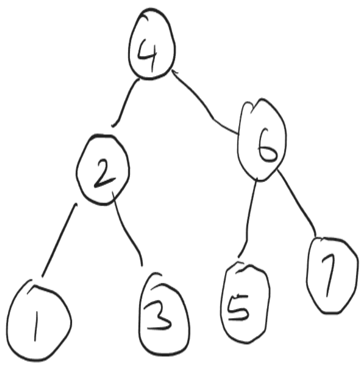


2 Fearless Jesse

I have spent my whole life scared. Frightened of things that could happen, might happen, might not happen. Fifty years I spent like that. Finding myself awake at three in the morning. But you know what? Ever since my diagnosis, I sleep just fine. And I came to realize it's that fear that's the worst of it. That's the real enemy...

W.W.

Everybody knows that Bakul Nivas forms a perfect binary tree with its n rooms as the nodes. A perfect binary tree is a binary tree in which all interior nodes have two children and all leaves have the same depth or same level. The rooms are numbered from 1 to n such that the inorder traversal of the hostel is sorted in ascending order.



Bakul hostel, not to scale

Jesse a $UG1$ also stays here. He follows the two rituals regularly.

1. Verifying himself on the DSA discord server
2. Going to rooms of his friends

From any room, he can go up to the parent room, or to any of its children. So there are three possible moves - U(up), L(left) and R(right). If Jesse performs a move which is invalid, like going left of a leaf node, nothing happens i.e., he stays at the same room.

As he visits so many rooms, he *fears* that he might enter a senior's room by mistake. Please help him so that he does not commit this blunder. You would be given the initial room number and sequence of moves he performs. You need to find the room where he ends up.

2.1 Input

The first line contains a two integers $n(1 \leq n \leq 10^{18})$ - the number of rooms in Bakul and t - the number of test cases. ($n + 1$ will be a power of 2)

The next t lines will consist of an integer r_i and string s_i. r_i is the initial room number, s_i is the string containing moves performed by Jesse starting from room r_i. (s_i only contains the characters *'L', 'R', 'U'* and $\sum |s_i| \leq 10^5$).

2.2 Output

Print one line containing the room number Jesse ends up, $r_i^{'}$.

2.3 Example

Input	Output
15 2	
4 UURL	10
8 LRLLLLLLLL	5

3 Hash Table

You need to implement a hash table which will use **chaining** method for collision resolution.

3.1 Implementation

The code must consists of two files:

1. *hash.h* - Defines the ADT and the function prototypes.
2. *hash.c* - Implements the ADT and the functions of hash.h.

3.2 Data Types

You can only use the given below definition as your code will be evaluated using our own driver code. You should make following structures and functions inside hash.h:-

```c

struct Item{
    int key;
    int frequency;
    struct item * next;
};

struct Bucket{
    struct Item * items;
};

struct HashTable {
     int a, b, countBucket;
     // countBucket = number of buckets
     // hash function will be: (a*key+b)%countBucket
     struct Bucket * buckets;
 };

 struct HashTable * init_hash_table(int a, int b, int countBucket); // function for initializing a hash table with given parameters.

 struct HashTable * insert(struct HashTable * T, int key); // function for inserting new key into hash table.

 bool search(struct HashTable * T, int key); // return true if element is present in hash table, otherwise false.

 struct HashTable * delete(struct HashTable * T, int key); // delete given key from the hash table. Do nothing if element is not present inside hash table.

 void print_table(struct HashTable * T); // function should print one line for each bucket. A one line bucket output looks like: key_1 freq_1 key_2 freq_2 ...... key_m freq_m
```

Now implement all the above ADT and functions inside hash.c . Note: If a key is repeated then you will need to increase it's frequency. Similarly, while deletion if key already exists then reduce it's frequency.

3.3 Report and Analysis:

For this part you will be provided with 10 input files. Each input file contains 3 types of command:

- INSERT x : insert key x inside hash table.
- SEARCH x : search whether key x is present inside hash table or not.
- DELETE x : delete key x from hash table.

Now for each input file you need to write the average number of operations that were required to carry out all the mentioned commands.
Write down your conclusions based on the output that you get.

3.4 Submission format for this question

Just submit hash.h and hash.c file along with the report on moodle. To check your own code and to prepare your report you will need to create a main.c file also, but no need to submit it for evaluation.

4 No Title

You are given a binary tree and each node of this tree has lower case alphabets $\{a - z\}$ as their value. You are also given a string **S** of lower case alphabets.
You need to process **Q** queries which are of two types as described below:

- Type 1: Change value of I^{th} node to alphabet **x**.
- Type 2: Given a substring **R** of **S** and a node **I** of given binary tree, output whether the string formed by the preorder traversal of binary subtree rooted at node **I** is identical to **R**.

4.1 Input:

- The first line will contain an integer **N**, number of nodes in tree.
- Next one line will contain a string H of length N, where H[i] is the value of I^{th} node.
- Next N-1 lines, will contain 3 integers- A, B, C denoting that B is left child of A if C=1 or B is right child of A if C=2.
- Next one line will contain input of string **S**.
- Next line contain input of one integer **Q**, number of total queries.
- Following Q lines, each line containing input for one query, where each query firstly takes one integer input **T** for type of query.
- **Query type 1:** If T=1 then the query input is followed by an integer **I** and a character **C**, meaning that value of I^{th} node is changed to **C**.
- **Query type 2:** If t=2 then the query input is followed by 3 integers **L**, **R** and **I** denoting that, check if the substring of **S** from index L to R(inclusive) is identical to preorder traversal of subtree rooted at node **I** or not.

4.2 Output:

For each type-2 query, print ”YES” if the required strings match and otherwise print ”NO”.
Note: print output of one type-2 query in a single line and for next type-2 query print it’s output in a newline. So total lines of output = Number of type-2 queries.

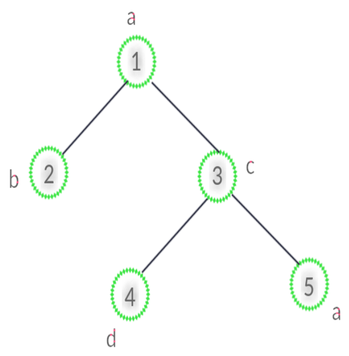
4.3 Tasks:

- **Subtask 1 (20):**
  $1 \leq N * Q \leq 10^6$
  $1 \leq length\ of\ string\ S \leq 10^6$
- **Subtask 2 (40):**
  $1 \leq N, Q \leq 10^5$
  $1 \leq length\ of\ string\ S \leq 10^6$
  $1 \leq total\ number\ of\ update\ queries \leq 100$
- **Subtask 3 (40):**
  $1 \leq N, Q \leq 10^5$
  $1 \leq length\ of\ string\ S \leq 10^6$

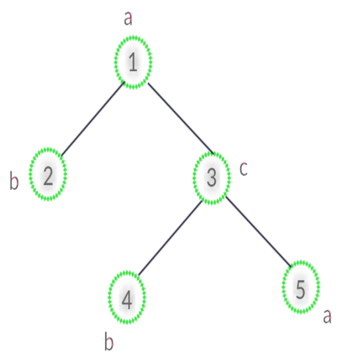
4.4 Example

Input	Output
5	
abcda	
1 2 1	
1 3 2	
3 4 1	
3 5 2	
bacbabcdab	
3	
2 3 5 3	NO
1 4 b	
2 3 5 3	YES

The tree initially looks like this:



Query 1: 2 3 5 3, so substring of S: ”cba” , preorder traversal of subtree rooted at 3 = ”cda”. Therefore the output is ”NO” as the strings does not match.
Query 2: 1 4 d, meaning that we need to change value of 4th node to alphabet- ’b’. So the new tree will now look like:



Query 3: 2 3 5 3, so substring of S: ”cba” , preorder traversal of subtree rooted at 3 = ”cba”. Therefore the output is ”YES” as the strings do match now.

5 Submission format

Upload a zip file *roll_number .zip* which has a folder named as your roll number. This folder should contain

q1/

- tree.c - containing routines like FindLiteTree(Tree *T), DeleteTree() etc
- tree.h
- remaining .c and .h files for any other ADT you may require
- main1.c

q2/

- .c and .h files for any ADT you may require
- main2.c

q3/

- hash.h
- hash.c
- main3.c
- report.pdf

q4/

- .c and .h files for any ADT you may require
- main4.c

Online Judge Submission
You need to create a single *.c file for each question.
Use command like

```
$ cat tree.h tree.c main.c > submission.c
```

to concatenate the files. Then perform the required changes to make sure it runs independently without errors.