INDIAN STATISTICAL INSTITUTE

MTech(CS) I year 2025-2026

Subject: Computing Laboratory

Lab Test 3 (May 11, 2026)

Total: 60 marks Duration: 4 hours

GENERAL INSTRUCTIONS

1. All programs should take the required input via the standard input (terminal/keyboard), and print the desired output to the terminal.
2. Please make sure that your programs adhere strictly to the specified input and output format. Do not print extra strings asking the user for input, debugging messages, etc. These will cause the automatic checking system to fail.

Problem 1 / Q1. (20 marks)

Given n different jobs $0, 1, \ldots, n-1$, we need to assign them to computer clusters. Assume there are infinitely many clusters available and each cluster can be assigned any number of jobs. But there are two types of constraints that need to be satisfied while assigning jobs: the "same cluster" constraint and the "different cluster" constraint. The "same cluster" constraint is represented by two arrays S and T, each of size $m \leq n$ containing job numbers; it is required that for each i, $S[i]$ and $T[i]$ are to be assigned to the same cluster. The "different cluster" constraint is also represented by two arrays P and Q each of size $k \leq n$ and containing job numbers; it is required that for each i, $P[i]$ and $Q[i]$ are assigned to two different clusters. Given n, m, k and the arrays S, T, P, Q write a program which determines if a schedule satisfying both the constraints is possible. To get full credit, your program should run in time $O(n \lg^* n)$.

Input format: Integers n, m, k followed by the arrays S, T, P, Q.

Output format: If a schedule exists, your program should just print YES; otherwise, your program should print a list of all the jobs (in increasing order of job number) which fail to satisfy both constraints.

Problem 2 (7 + 18 = 25 marks)

You are given a non-empty binary search tree (BST) containing integer values in the "standard" tabular form used in class (so you can directly use the provided `read_tree()` function to read it into memory).

Q2. Add a `height` field to each node. For any node, this field should store the height of the subtree rooted at that node. Write a function that traverses the tree in postorder, and during the traversal:

(a) fills in the `height` field of each node, and

(b) checks whether the subtree rooted at that node is *balanced*.

Output format: Print the tree in the usual tabular form, along with two additional columns containing the height and the balance factor (bf) for each node.

Recall that the balance factor, bf, of a node v is defined as $bf(v) = height(L) - height(R)$, where L and R represent, respectively, the left and right subtrees of v. **For this question, the height of an empty / null tree should be taken as -1, not 0.** [7]

Q3. Write a function to convert the given BST to a balanced BST (AVL tree) using the following algorithm: traverse the original tree in postorder, and depending on the *bf* at each node, call any rotation operations needed at that node.

~~Several valid AVL trees may be obtained from a given BST. If your code does not precisely implement the algorithm specified above, you may still get an AVL tree, but it may not match the desired output.~~

CORRECTION: The algorithm given above is **NOT** guaranteed to result in a balanced BST (see Test Case 4, for example).[1] If you correctly implement the algorithm outlined above, however, your outputs should match the outputs of the provideed testcases.

Output format: As for Q2, but the first line should contain the index of the root in addition to the number of nodes in the tree. [18]

Problem 3 (10 + 5 = 15 marks)

You are given a set $S = \{s_1, s_2, \ldots, s_n\}$ of strings (note that the strings are indexed starting from 1) consisting only of lowercase letters (a–z). You have to store the strings in a slightly modified version of the trie data structure discussed in class. In the standard version of the trie, the last field of each node is either

- a flag that specifies whether that node corresponds to the end of a valid word, or
- an integer denoting the number of times this string has occurred as a complete word in the input.

See slides 5 and 6 of `tries.pdf` for details.

In the modified version, the last field of any trie node that corresponds to the end of a string s_i contains the integer $i > 0$. For example, if $S = \{s_1 = \texttt{bath}, s_2 = \texttt{bat}\}$, then the last field of the trie nodes corresponding to the strings `bath` and `bat` will contain 1 and 2, respectively, but the trie nodes corresponding to `b` and `ba` will have 0 in the last field.

Q4. Write a program to insert the strings s_1, s_2, …, s_n in that order into an initially empty trie. As described above, the last field of the trie node corresponding to the end of s_i should contain i.

Input format: The integer n, followed by n whitespace separated strings s_1, s_2, …, s_n. Each string will contain no more than 32 lowercase letters.

Output format: The number of nodes in your trie, followed by the nodes themselves, one per line. Each node should be printed as a list of 27 space separated integers. [10]

Sample input:

```
2
bath
bat
```

Sample output:

```
5
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

[1] Thanks to Rohan Raj Padhy for pointing this out.

Q5. Write a program that takes a second sequence of strings $p_1\ p_2\ \ldots\ p_k$ (in addition to the strings in S that are to be inserted into the trie), searches for each p_i in the trie, and prints

- the corresponding index if the search is successful, i.e., i_k if $p_i = s_{i_k}$, or
- -1 if $p_i \notin S$.

Input format: The integer n, followed by n whitespace separated strings s_1, s_2, ..., s_n, as described above, followed again by the integer k, and k whitespace separated strings $p_1\ p_2\ \ldots\ p_k$.

Output format: A single line of the form $i_1\ i_2 \ldots i_k$, where either $p_j = s_{i_j}$, or $i_j = -1$. [5]

Sample input:

```
3
the
bath
bat
5
i bat in the bath
```

Sample output:

```
-1 3 -1 1 2
```