AVL Tree Heist

Even though Arup doesn't like data structures, he's decided that it's still in his students' best interest to get some practice with AVL Trees. As a compromise, he's decided that it's good enough if the students just implement the insert function instead of both the insert and delete. To make his assignment easy to grade, he'll simply give students a sequence of inserts to perform and then ask the students to output a pre-order traversal after each insertion to make sure that the structure of the tree is correct.

You have decided that while you like regular binary search trees, you feel that AVL Trees that have inserts with rebalancing are too much work. Thus, you've come up with a nefarious plan. You will infiltrate Arup's office by bribing the janitorial staff, steal his test cases and remove all of the cases where a rebalancing would have been required. This way, you can just write a regular binary search tree insert and STILL earn a 100!

For this program, read in a sequence of inserts into an AVL tree and simply determine if performing those inserts ever requires a rebalancing. If so, print out "REMOVE", otherwise print out "KEEP". (Note: In the spirit of the assignment, notice that you can solve the given problem without ever performing any rebalances!!!)

Input

The first line of input will contain a single positive integer, *T (T ≤ 100),* the number of input cases. The following *T* lines will contain each of the test cases, one test case per line. For each test case, the first integer on the line, *n (n < 1024),* will represent the total number of inserts for the test case. The following **distinct** *n* positive integers on the line represent the values to insert into the AVL tree, in the order that they are given. All of these values will be separated by a space. (There may or may not be a space at the end of each of these lines.)

Output

For each test case, start the output with a header of the following form:

```
Tree #k:
```

where k is the number of the test case, starting with 1.

Follow this with a space and either the string "REMOVE" or "KEEP", based on the cases described above.

Sample Input

```
2
3 1 2 3
7 4 2 6 7 3 1 5
```

Sample Output

```
Tree #1: REMOVE
Tree #2: KEEP
```

Problem C
Ceiling Function
Time limit: 5 seconds

Advanced Ceiling Manufacturers (ACM) is analyzing the properties of its new series of Incredibly Collapse-Proof Ceilings (ICPCs). An ICPC consists of n layers of material, each with a different value of collapse resistance (measured as a positive integer). The analysis ACM wants to run will take the collapse-resistance values of the layers, store them in a binary search tree, and check whether the shape of this tree in any way correlates with the quality of the whole construction. Because, well, why should it not?

To be precise, ACM takes the collapse-resistance values for the layers, ordered from the top layer to the bottom layer, and inserts them one-by-one into a tree. The rules for inserting a value v are:

- If the tree is empty, make v the root of the tree.
- If the tree is not empty, compare v with the root of the tree. If v is smaller, insert v into the left subtree of the root, otherwise insert v into the right subtree.

ACM has a set of ceiling prototypes it wants to analyze by trying to collapse them. It wants to take each group of ceiling prototypes that have trees of the same shape and analyze them together.

For example, assume ACM is considering five ceiling prototypes with three layers each, as described by Sample Input 1 and shown in Figure C.1. Notice that the first prototype's top layer has collapse-resistance value 2, the middle layer has value 7, and the bottom layer has value 1. The second prototype has layers with collapse-resistance values of 3, 1, and 4 – and yet these two prototypes induce the same tree shape, so ACM will analyze them together.

Given a set of prototypes, your task is to determine how many different tree shapes they induce.

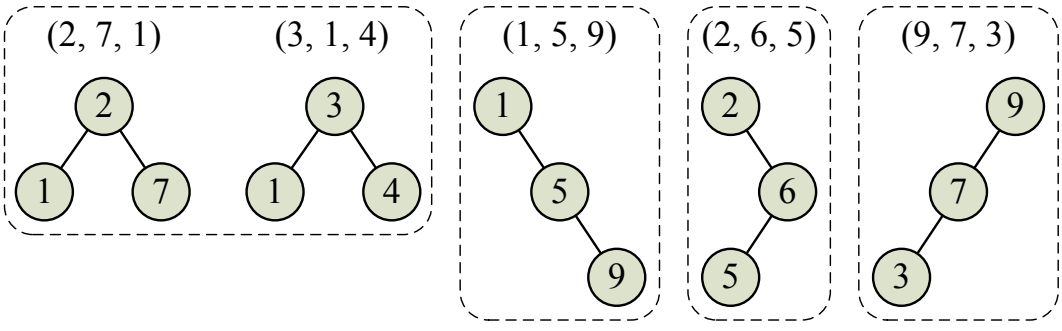


Figure C.1: The four tree shapes induced by the ceiling prototypes in Sample Input 1.

Input

The first line of the input contains two integers n ($1 \leq n \leq 50$), which is the number of ceiling prototypes to analyze, and k ($1 \leq k \leq 20$), which is the number of layers in each of the prototypes.

The next n lines describe the ceiling prototypes. Each of these lines contains k distinct integers (between 1 and 10^6, inclusive), which are the collapse-resistance values of the layers in a ceiling prototype, ordered from top to bottom.

Output

Display the number of different tree shapes.

Sample Input 1	Sample Output 1
5 3 2 7 1 3 1 4 1 5 9 2 6 5 9 7 3	4

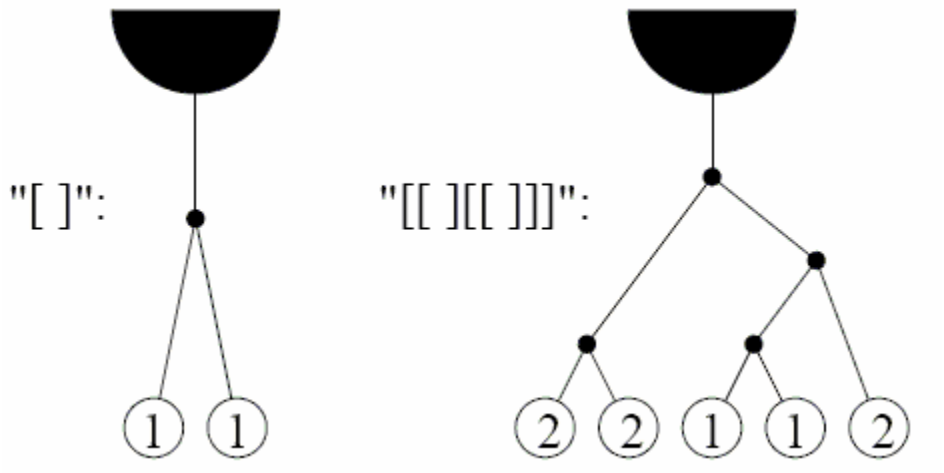
Sample Input 2	Sample Output 2
3 4 3 1 2 40000 3 4 2 1 33 42 17 23	2

F • Monkey Vines

Deep in the Amazon jungle, exceptionally tall trees grow that support a rich biosphere of figs and juniper bugs, which happen to be the culinary delight of brown monkeys.

Reaching the canopy of these trees requires the monkeys to perform careful navigation through the tall tree's fragile vine system. These vines operate like a see-saw: an unbalancing of weight at any vine junction would snap the vine from the tree, and the monkeys would plummet to the ground below. The monkeys have figured out that if they work together to keep the vines properly balanced, they can *all* feast on the figs and juniper bugs in the canopy of the trees.

A *vine junction* supports exactly two *sub-vines,* each of which must contain the same number of monkeys, or else the vine will break, leaving a pile of dead monkeys on the jungle ground. For purposes of this problem, a *vine junction* is denoted by a pair of matching square brackets **[ ]**, which may contain nested information about junctions further down its *sub-vines*. The nesting of vines will go no further than **25** levels deep.



You will write a program that calculates the *minimum* number of monkeys required to balance a particular vine configuration. There is **always** at least one monkey needed, and, multiple monkeys may hang from the same vine.

Input

The first line of input contains a single integer ***N***, ($1 \le N \le 1000$) which is the number of datasets that follow.

Each dataset consists of a single line of input containing a vine configuration consisting of a string of **[** and **]** characters as described above. The length of the string of **[** and **]** will be greater than or equal to zero, and less than or equal to 150.

Output

For each dataset, you should generate one line of output with the following values: The dataset number as a decimal integer (start counting at one), a space, and the minimum number of monkeys required to reach the canopy successfully. Assume that all the hanging vines are reachable from the jungle floor, and that all monkeys jump on the vines at the same time.

Sample Input	Sample Output
3 [] [[][[]]]	1 2 2 1 3 8

Note: The second line of sample input is a blank line.

Problem B

Spreading News

You are the manager of a company, and you want all of your employees to be notified of an important news item as quickly as possible. Your company is organized in a tree-like structure: each employee has exactly one direct supervisor, no employee is his own direct or indirect supervisor, and every employee is your direct or indirect subordinate. You will make a phone call to each of your direct subordinates, one at a time. After hearing the news, each subordinate must notify each of his direct subordinates, one at a time. The process continues this way until everyone has heard the news. Each person may only call direct subordinates, and each phone call takes exactly one minute. Note that there may be multiple phone calls taking place simultaneously. Compute the minimum amount of time, in minutes, required for this process to be completed. Employees will be numbered starting from 1, while you will be numbered 0. Furthermore, every supervisor is numbered lower than his or her direct subordinates.s

Input

First line of the input contains T the number of test cases. Each test case contains an integer N ($1 \leq N \leq 70$) denoting the number of employees in your company including you. Next line contains N-1 integer. The i'th integer denote the supervisor of i'th employee(i starts from 1) . Look you(employee 0) do not have any supervisor.

Output

For each test case output the minimum amount of time, in minutes, required.

Sample Input	Sample output
5 3 0 0 5 0 0 2 2 9 0 0 1 1 2 2 3 4 5 0 1 2 3 7 0 1 2 3 3 3	2 3 4 4 6

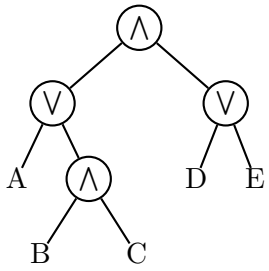
[C] Normalized Form

Program:	`tree.(c|cpp|java)`
Input:	`tree.in`
Balloon Color:	`Blue`

Description

As you most probably know, any boolean expression can be expressed in either a *disjunctive normal form* or a *conjunctive normal form.* In a disjunctive normal form, a boolean expression is written as a disjunct (logical or) of one-or more sub-expressions where each of these sub-expressions is written in a conjunctive normal form. Similarly, an expression written in a conjunctive normal form is a conjunct (logical and) of sub-expressions each written in a disjunctive normal form.

An AND/OR tree is a tree-like graphical-representation of boolean expressions written as either conjunctive- or disjunctive-normal form. Since the sub-expressions of a normalized form alternate in being either disjunctive or conjunctive forms, you'd expect the sub-trees on an AND/OR tree to alternate in being AND- or OR- trees depending on the sub-tree's depth-level. The example on the right illustrates this observation for the boolean expression $(A \bigvee (B \bigwedge C)) \bigwedge (D \bigvee E)$ where the trees in the 1st (top-most) and 3rd levels are AND-trees.



Write a program that evaluates a given and/or tree.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on exactly one line (which is no longer than 32,000 characters) of the form:

$$(E_1 \; E_2 \; \ldots \; E_n)$$

where $n > 0$ and E_i is either `T` for true, `F` for false, or a sub-expression using the same format. The trees at the deepest level are AND-trees. The last test case is followed by a dummy line made of ().

Output Format

For each test case, print the following line:

`k.␣E`

Where `k` is the test case number (starting at one,) and `E` is either `true` or `false` depending on the value of the expression in that test case.

Sample Input/Output

tree.in

```
((F(TF))(TF))
(TFT)
((TFT)T)
()
```

OUTPUT

```
1. false
2. false
3. true
```

Tree Sales

Filename: treesales

The Problem

Several well-known companies use a pyramid sales scheme. Being both an entrepreneur AND a computer scientist, however, you prefer to model your new business as a tree sales scheme, where the hierarchical structure of the company can be modeled as a tree.

In particular, the company initiator, or CEO, is the root of the tree structure of the company. From there, any current member of the company can hire a direct subordinate. So, at the beginning, it's up to the CEO to hire other employees who will be directly below the CEO in the tree structure. At any point in time, any employee can make a sale. Total compensation of any employee is calculated based on the sum of sales of all members of the subtree rooted at that employee, so it's important at any point in time to be able to calculate the total sales in any subtree of the company structure.

In order to start your company and allow others to start similarly structured companies, you have decided to write a computer program that will read in a set of operations from the following set:

1) Add an employee (first add is the CEO, rest are made by current employees)
2) An employee makes a sale
3) Query for the total sales in an employee's subtree at that point in time

and execute the appropriate command, in the order given, producing output for all commands of type three.

The Input

The first line of input will contain a single integer, T $(T \leq 10)$, representing the number of company structures to analyze. The first line of each company structure to analyze will contain a single positive integer, n $(n \leq 100000)$, representing the number of operations to execute for that company. The following n lines will each contain a single command with one of the following three formats:

```
ADD SPONSOR NEWEMPLOYEE
SALE EMPLOYEE X
QUERY EMPLOYEE
```

All employee names will be strings of 1 to 10 uppercase letters. In the first format, SPONSOR will be the current employee who is hiring a new employee, and the NEWEMPLOYEE will be the new employee to be added directly below the sponsor. The very first command for each company will be an add command with the sponsor "ROOT", indicating that NEWEMPLOYEE is the root of the tree structure for that company. No employee of any company will be named "ROOT". In the second format, EMPLOYEE will be the employee in question and X will be a positive integer less than 1000 representing the value of the sale made by the given employee. In the third format, EMPLOYEE will be the employee in question for which we must find the total

sales of her subtree in the company. All names given for current employees for all three types of commands are guaranteed to be valid current employees.

It is guaranteed that all employees added will be identified by distinct strings and that the tree structure produced will not have a height greater than 100. (Note: The height of a tree with two nodes is 1.)

The Output

For each company output a single line header of the form

```
COMPANY K
```

where K is the number of the company, starting with 1. For each query (command of type 3 in the input), output a single line with a positive integer representing the current total sales of the subtree of the employee queried. Note: Each company will have at least one query.

Sample Input

```
2
14
ADD ROOT BILL
ADD BILL EVELYN
ADD BILL SARAH
SALE BILL 25
SALE EVELYN 75
SALE SARAH 10
QUERY BILL
ADD EVELYN MATT
ADD MATT ANYA
SALE ANYA 1000
QUERY MATT
QUERY EVELYN
QUERY BILL
QUERY SARAH
11
ADD ROOT MARILYN
ADD MARILYN GARY
ADD MARILYN REMY
ADD MARILYN BRIANNE
ADD MARILYN TAJ
SALE TAJ 10
SALE REMY 20
SALE BRIANNE 40
SALE MARILYN 30
QUERY GARY
QUERY MARILYN
```

Sample Output

```
COMPANY 1
110
1000
1075
1110
10
COMPANY 2
0
100
```