**Question 1:-**

Ramu has N dishes of different types arranged in a row: A1,A2,…,AN where Ai denotes the type of the ith dish. He wants to choose as many dishes as possible from the given list but while satisfying two conditions:

1. He can choose **only one** type of dish.
2. No two chosen dishes should be adjacent to each other.

Ramu wants to know which type of dish he should choose from, so that he can pick the maximum number of dishes.

**Example :**

Given N= 9 and A= [1,2,2,1,2,1,1,1,1]

For **type 1**, Ramu can choose at most four dishes. One of the ways to choose four dishes of type 1 is A1,A4, A7 and A9.

For **type 2**, Ramu can choose at most two dishes. One way is to choose A3 and A5.

So in this case, Ramu should go for **type 1**, in which he can pick more dishes.

**INPUT FORMAT:**

* The first line contains T, the number of test cases. Then the test cases follow.
* For each test case, the first line contains a single integer N.
* The second line contains N integers A1,A2,…,AN.

**OUTPUT FORMAT**

For each test case, print a single line containing one integer ― the type of the dish that Ramu should choose from. If there are multiple answers, print the smallest one.

**CONSTRAINTS :**

* **1 <= T <= 10^3**
* **1 <= N <= 10^3**
* 1 <= Ai <= 10^3

**Sample Input :**

**3**

**5**

**1 2 2 1 2**

**6**

**1 1 1 1 1 1**

**8**

**1 2 2 2 3 4 2 1**

**Sample Output :**

**1**

**1**

**2**

**Question 2:-**

There are three piles of stones. The first pile contains a stones, the second pile contains b stones and the third pile contains c stones. You must choose one of the piles and split the stones from it to the other two piles; specifically, if the chosen pile initially contained s stones, you should choose an integer k (0≤k≤s), move k stones from the chosen pile onto one of the remaining two piles and s−k stones onto the other remaining pile. Determine if it is possible for the two remaining piles (in any order) to contain x stones and y stones respectively after performing this action.

**INPUT FORMAT :**

* The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.
* The first and only line of each test case contains five space-separated integers

          a,b,c, x and y.

**OUTPUT FORMAT :**

For each test case, print a single line containing the string “YES” if it is possible to obtain piles of the given sizes or “NO” if it is impossible.

**CONSTRAINTS :**

* 1 <= T <= 100
* 1 <= a,b,c,x,y <= 10^9

**SAMPLE INPUT :**

**4**

**1 2 3 2 4**

**3 2 5 6 5**

**2 4 2 6 2**

**6 5 2 12 1**

**SAMPLE OUTPUT :**

**YES**

**NO**

**YES**

**NO**

**Test case 1:**You can take the two stones on the second pile, put one of them on the first pile and the other one on the third pile.

**Test case  2:**You do not have enough stones.

**Test case 3:**You can choose the first pile and put all stones from it on the second pile.

#### **Problem Statement:**Let’s define a Beautiful Function F(x) in such a way: Add 1 to the value of x, if the result of addition contains any trailing zeros then remove them all. **Example:** F(11) = 12 F(19) = 2 (20 –> 2) F(99) = 1(100 –> 10 –> 1) Let’s define a number to be reachable from x , if we can apply Beautiful Function some number of times (possibly zero) to x and get that number as result Ex. 102 can be reachable from 1099 as F(F(1099)) = F(101) = 102 You are given a number N . Calculate how many numbers are reachable from N.

#### **Input Format:** The first line contains an integer N. denoting the given number.

#### **Constrains:** 1 <= N <= 10^9

|  |  |  |
| --- | --- | --- |
| **Sample Input** | **Sample output** | **Explanation** |
| 1 | 9 | 1,2,3,4,5,6,7,8,9 are reachable from 1. |

#### Question 1 - Slowest Key Press

##### Problem Statement

Alex wants to be a faster typist and is taking a typing test to find out which key takes the longest time to press.

Given the results of the test, determine which key takes the longest to press.

For example, given keyTimes =[[0, 2], [1, 5], [0, 9], [2, 15]]. Interpret each keyTimes[i][0] as an encoded character in the range ascii[a-z] where a = 0, b = 1,...z = 25.

The second element, represents the time the key is pressed since the start of the test.

In the example, keys pressed, in order are abac at times 2, 5, 9, 15. From the start time, it took 2 - 0 = 2 to press the first key, 5 - 2 = 3 to press the second, and so on.

The longest time it took to press a key was key 2, or 'c', at 15 - 9 = 6.

##### Function Description

Complete the function slowestKey in the editor below. The function must return a character, the slowest key that Alex presses.

slowestKey has the following parameter(s):

keyTimes[keyTimes[O],... keyTimes[n-1]]: an array of two integers: The character Alex pressed (encoded) and the time at which it was pressed

##### Constraints

• 1 <= n <= 10^5

• 0 <= keyTimes[i][0] <= 25 (0 <= i <= n)

• 1 <= keyTimes[i][1] <= 10^8 (0 <= i <= n)

• It is guaranteed that there will only be one key with the worst time. Input Format For Custom Testing

##### Sample Input

3

2

0 2

1 3

0 7

##### Sample Output

a

##### Explanation:

The time taken to press 'a' in the worst case is 7 - 3 = 4. To press 'b' the worst case is 3 - 2 = 1. Alex performed worst on 'a'.

#### Question 2 - Rock Paper Scissor

##### Problem Statement

In a Rock Paper Scissor tournament, the contestants stand in a straight line. Each pair of consecutive players compete in a round. If there is an odd number of players, the last one in the line qualifies for the next round without playing.

For each game, each player indicates either a rock, paper or scissors denoted 'R', 'P' or 'S' respectively. Outcomes are as follows:

• paper beats rock, P > R

• scissors beat paper, S > P

• rock beats scissors, R > S

After each round, the winners remain and the losers are out of the competition. In case of a tie, both players lose. A player would like to win the competition and has one advantage: the knowledge that each other player uses only one type of hand formation in all the rounds of the tournament. Determine how many times the player will have to change the hand formation in order to win the competition.

For example, the number of players, n = 3, and the player of interest, POI, is standing at position given by a = 2 (0-based index). The hand formations used by other players are given by formations = "PS" with length n - 1 = 2. In the first round, scissors (S) beats paper (P). POI must then beat the winner of round one who always chooses scissors (S). POI will choose rock (R) and win the tournament after having chosen only one hand formation, hence the answer is 0.

##### Function Description

Complete the function handFormationChange in the editor below. The function must return an integer, the number of times the POI needs to change hand formation.

handFormationChange has the following parameter(s): n: an integer, the number of players in the tournament a: an inteqer, the POI's position in the line, 0-indexed

##### Constraints

• 2 <= n <= 10^5

• 0 <= a <= n

• formations[i] belongs to {'R', 'S', 'P'} (0 <= i < n-1)

##### Sample Input

4

1

PRS

##### Sample Output

1

##### Explanation:

In the first round, POI is at position 1 and must beat paper at position O by choosing S, scissors. For the second pair, rock beats scissors. Now the POI must choose paper to beat rock in round two. It takes one change of formation to win the tournament.

#### Question 3 - Dangerous Script

##### Problem Statement

If you've ever looked at an assembly language dump of an executable file, you know that commands come in the form of hexadecimal digits that are grouped by the processor into instructions. It is important that the parsing be done correctly or code will not be executed as expected. Wrong parsing is the basis for Return Oriented Programming, a method of causing mahem.

You have developed a program in a new scripting language. Of course, it requires accurate parsing in order to perform as expeced, and it is very cryptic. You want to determine how many valid commands can be made out of your lines of script. To do this, you count all of the substrings that make up a valid command. Each of the valid commands will be in the following format:

• The first letter is a lowercase English letter.

• Next, it contains a sequence of zero or more of the following characters: lowercase English letters, digits, and colons.

• Next, it contains a forward slash '/'.

• Next, it contains a sequence of one or more of the following characters: lowercase English letters and digits.

• Next, it contains a backward slash '\'.

• Next, it contains a sequence of one or more lowercase English letters.

Given some string, s, we define the following: • s[i..j] is a substring consisting of all the characters in the inclusive range between index i and index j.

• Two substrings, s[i[1]..j[1]] and s[i[2]..j[2]] are said to be distinct if either i[1] != i[2] or j[1] != j[2].

For example, your command line is abc:/b1c\xy.

Valid command substrings are:

abc:/b1c\xy

bc:/b1c\xy

c:/b1c\xy

abc:/b1c\x

bc:/b1c\x

c:/b1c\x

There are six valid commands that may be parsed from that one command string.

##### Function Description

Complete the function commandCount in the editor below. The function must return an array of integers, each representing the number of valid command strings in commands[i].

commandCount has the following parameter(s): commands[commands[0],...commands[n-l]]: an array of strings

##### Constraints

• 1 <= n <= 50

• Each character of commands[i] belongs to [a-z, 0-9, /, \, :]

• Each |commands[i]| <= 5 \* 10^5.

##### Sample Input

6

w\\//a/b

w\\//a\b

w\\/a\b

w:://a\b

w::/a\b

w:/a\bc::/12\xyz

##### Sample Output

0

0

0

0

1

8

##### Explanation:

Let's call our return array ret.

We fill ret as follows:

• commands[0] = "w\//a/b" has no valid command substring, so ret[0] = 0.

• commands[1] = "w\//a\b" has no valid command substring, so ret[1] = 0.

• commands[2] = "w\/a\b" has no valid command substring, so ret[2] = 0.

• commands[3] = "w:://a\b" has no valid command substring, so ret[3] = 0.

• commands[3] = "w::/a\b" has one valid command substring, commands[O..6] = "w::/a\b" so ret[4] = 1

• commands[5] = "w:/a\bc::/12\xyz" has the following eight valid command substrings:

1. commands[0..5] = w:/a\b
2. commands[O..6] = w:/a\bc
3. commands[5..13] = bc::/12\x
4. commands[5..14] = bc::/12\xy
5. commands[5..15] = bc::/12\xyz
6. commands[6..13] = c::/12\x
7. commands[6..14] = c::/12\xy
8. commands[6..15] = c::/12\xyz

This means ret[5] = 8

We then return ret = [0, 0, 0, 0, 1, 8].