Problem A. Average Problem

Source file name:	Average.c, Average.cpp, Average.java, Average.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

At the Institute of Computing & Programming College (ICPC), grades are given from 0 to 100, but in multiples of five, it means that if you get a 97% as an average, then your final grade would be 95. Just the same case happens when you get a 93%, you get a 95. This happens because 95 is in both cases, the closest multiple of five to the obtained grade. This means, for example that 67.5 rounds to 65 but 67.51 rounds to 70.

Now, there are many evaluation tables assigned to different courses. And the teachers want to know the final grade of each one of their students and if the final grade went `UP`, if it went `DOWN` or if it stays the `SAME` from the original obtained grade. An evaluation table is a list of points that each assignment has. A student get an individual grade on each assignment from 1 to 100.

For example, if the evaluation table is 50 and 30, and a student gets 100 and a 50, it means that he got 100% of the 50 points and a 50% of the 30 points, then his grade will be $\frac{65}{80}$.

Input

The first line has the positive integer number T of test cases $1 \leq T \leq 100$. Each test case begin with two integers A, S ($1 \leq A \leq 12$, $1 \leq S \leq 40$) where A is the number of evaluations on the curse, and S is the number of students. The next line has A positive integer numbers $1 \leq a_i \leq 100$ representing the value of each evaluation. The next S lines are the grades of each of the students. Every line has A positive integers, where s_i is a percentage obtained in a_i evaluation.

Output

Per each student you must print a single line with the final grade and if it went `UP`, `DOWN` or remain the `SAME` as it was. Each test case must be ended with a blank line.

Example

Input	Output
4 2 3 50 50 100 90 100 91 100 89 4 1 15 15 15 15 70 80 75 75 5 1 20 30 10 20 20 100 100 0 0 100 5 1 20 30 10 20 20 100 100 10 10 100	95 SAME 95 DOWN 95 UP 75 SAME 70 SAME 75 UP

Problem B. One Piece of Cake

Source file name:	One.c, One.cpp, One.java, One.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Rodrigo Chaves - TEC & UCR Costa Rica`

The Straw Hat Pirates are a very skilled crew that travels the seas. It consists of its captain and 9 other members. They were invited to a wedding by a famous pirate that has many children and loves to eat cake.

The crew has just arrived to the dinner party. At the start of the party there were P pieces of cake. Every crew member wants exactly one piece of cake. However, other people in the party have already eaten E pieces.

Please help them know if there is enough cake for them to eat!

Input

The first line consists of a single integer T ($1 \leq T \leq 20000$), the number of test cases. After that, T lines follow, which have two space-separated integers P and E, ($1 \leq E \leq P \leq 200$).

Output

For each case, print a single line consisting of '`YES`' (without quotes) if there is enough cake for the Straw Hat Pirates and '`NO`' (without quotes) otherwise.

Example

Input	Output
3 99 80 45 40 15 4	YES NO YES

Problem C. Cardinality of Sets

Source file name:	Cardinality.c, Cardinality.cpp, Cardinality.java, Cardinality.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

There are two sets of positive integers A and B with $|A| = m$ and $|B| = n$ (Remember that the cardinality of a set is the number of different elements it contains, the cardinality is indicated by the bars that enclose the set).

We want to determine the cardinalities of the sets that are obtained as a result $A - B$, $A \cap B$, $B - A$ and $A \cup B$.

Input

The input contains multiple test cases, each one has the following structure:

The first line contains two positive integers m n ($1 \leq m,\ n \leq 10^6$), which represent the cardinalities of the sets A and B respectively.

The second line contains m **different** positive integers (separated by a blank space) which make up the set A, these values are in the closed interval $[1,\ 10^9]$.

The third line contains n **different** positive integers (separated by a blank space) which make up the set B, these values are in the closed interval $[1,\ 10^9]$.

The input ends with a test case that has two zeros which should not be processed.

Output

For each test case, the output must contain a single line with four non-negative integers, separated by a single space, which respectively represent $|A - B|$ $|A \cap B|$ $|B - A|$ $|A \cup B|$.

Example

Input	Output
5 6 7 1 4 3 9 8 3 2 10 7 1 5 5 1 3 5 7 9 10 8 6 4 2 8 6 10 1 5 3 6 4 7 2 1 2 3 4 5 6 0 0	2 3 3 8 5 0 5 10 2 6 0 8

Problem D. Determining Fibonaccism

Source file name:	Determining.c, Determining.cpp, Determining.java, Determining.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Close to the year 1200, Leonardo Pisano Bigollo Fibonacci, invented his famous recursion:

$$fibonacci(n) = \begin{cases} 0 & \text{if} \quad n = 0 \\ 1 & \text{if} \quad n = 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{if} \quad n \geq 2,\ n \in \mathbb{N} \end{cases}$$

Fernando and Té are playing a game, Fernando says a non-negative integer number n $(0 \leq n < 2^{31} - 1)$ and Té has to say if exists a non-negative integer number m $(0 \leq m \leq 10^4))$ such as $fibonacci(m) \equiv n \pmod{(2^{31} - 1)}$ in other words, if the fibonacci of m is equivalent to $n \mod 2^{31} - 1$.

Your task is to help Té in order to beat Fernando at this game.

Input

The first line of the input is the number of test cases that Fernando is going to throw to Té, T $(1 \leq T \leq 50000)$. The next T lines contain a single non-negative integer number N $(0 \leq N < 2^{31})$ that is the number Fernando is asking Té.

Output

For each number Fernando asks, print in a single line `YES` if there is a fibonacci number that matches the description above, or `NO` if there is not.

Example

Input	Output
5 1 2 3 4 2144909037	YES YES YES NO YES

Explanation

$fibonacci(55) = 139583862445$ and $139583862445 \equiv 2144909037 \pmod{(2^{31} - 1)}$ therefore, 2144909037 is a valid number because exists 55 and $0 \leq 55 \leq 10000$.

Problem E. Queries on the Stack

Source file name:	Queries.c, Queries.cpp, Queries.java, Queries.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Yonny Mondelo Hernández - UCI Cuba`

You have an empty stack and you are given some queries. These queries are the basic stack operations such as Push, Pop and finding the Top element on the stack. Also, is needed to know in any moment the absolute difference between the maximum and the minimum value on the stack. You should process the given queries.

Input

The first line contain a integer T $(1 \leq T \leq 100)$ denoting the number of cases. Each case will begin with a line containing a integer Q $\left(1 \leq Q \leq 10^4\right)$ denoting the number of queries to process, followed by exactly Q lines based on these formats:

- 1 V: Push V $\left(0 \leq V \leq 10^4\right)$ on the Top of the stack.
- 2: Pop an element from the Top of the stack. If the stack is empty, do nothing.
- 3: Print the absolute difference between the maximum and the minimum value on the stack (see Output Format). If the stack is empty, show "`Empty!`" without quotes.

Output

For each query of type 3 find and show the absolute difference between the maximum and the minimum value on the stack.

Example

Input	Output
2 12 2 1 10 1 15 3 2 2 3 1 18 3 1 10 1 1 3 3 3 1 999 3	5 Empty! 0 17 Empty! 0

Problem F. Farthest Cell in a Maze

Source file name:	Farthest.c, Farthest.cpp, Farthest.java, Farthest.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

One of the passions of professor *Humbertov Moralov* is solving mazes. Now he is interested in identifying the farthest cell(s) from an initial starting point by making an optimal tour where the number of cells he has to visit is minimized. To make the challenge even more interesting, Professor Moralov wants the optimal route from the initial cell to each of the furthest cells to be independently written.

For ease, you can think of the maze as a grid (a matrix) of cells. A cell can be either a wall or a corridor (that means, a cell that you can walk through). At any time the professor *Moralov* can move to a new cell from their current position if they share one side and both are corridor cells.

As you are a student of the Data Structure course, the professor *Humbertov Moralov* needs your help to solve this challenge, for which you must write a program to calculate the minimum distances from the initial cell (professor's starting point) to each of the other corridor cells, then your program must identify the farthest corridor cell(s) in the minimum distances and independently write the route that must be made from the initial cell to each of the most distant cells.

Input

The input begins with a positive integer T ($1 \leq T \leq 10$), denoting the number of test cases.

Each test case begins with a line containing two positive integers W H (Wide, High, $3 \leq W, H \leq 1000$). The test case continues with H lines, each containing W characters. Each character represents the status of a cell in the maze as follows:

1. '.' - to indicate that it is a corridor cell (cell that you can walk through),
2. '#' - to indicate that it is a wall cell,
3. '@' - to indicate professor Humbertov Moralov's starting point (appears exactly once per test case).

Output

For each test case, print a first line in the following format `Case idCase:`, where `idCase` is replaced by the number of the test case in which it is found, then a second line is printed containing two positive integers F L, which respectively represent the total of cells furthest in the optimal path and the length in the optimal route. Finally, the lexicographically ordered F optimal runs of length L are presented, each one on one line. Each optimal path is a word of length L that can consist only of the uppercase letters '`D`' (Down), '`L`' (Left), '`R`' (Right) y '`U`' (Up). Since there can be multiple optimal paths to get from the professor *Moralov's* starting position to one of the farthest cells, then choose the optimal path that is first in the lexicographic order. For more clarity on the input and output format see the examples below.

Example

Input	Output
5 6 5 ...#.# .##... ..@.## #.#... ...#.# 6 5 .###.# .##... ..@.## #.#... ...#.# 5 6 ##### #..#. #.### #...# ###@# ..... 7 7 ..#.#.. ..#.#.. ###.### ...@... ###.### ..#...# ..#.#.. 13 13 ######.###### #####...##### ####.....#### ###.......### ##.........## #...........# ......@...... #...........# ##.........## ###.......### ####.....#### #####...##### ######.######	Case 1: 1 6 LLUURR Case 2: 7 4 LDDL LDDR LLUU RDRD RDRR RURR RURU Case 3: 1 6 ULLUUR Case 4: 1 6 DDRRDR Case 5: 24 6 DDDDDD DDDDDL DDDDDR DDDDLL DDDDRR DDDLLL DDDRRR DDLLLL DDRRRR DLLLLL DRRRRR LLLLLL LLLLLU LLLLUU LLLUUU LLUUUU LUUUUU RRRRRR RRRRRU RRRRUU RRRUUU RRUUUU RUUUUU UUUUUU

Problem G. K-Uniform Array

Source file name:	Kuniform.c, Kuniform.cpp, Kuniform.java, Kuniform.py
Input:	Standard
Output:	Standard
Author(s):	José Manuel Salazar Meza - UFPS Colombia

You are given an array a consisting of n positive integers.

The array a is called *k-Uniform* if it has some sub-array of length k whose elements are all equal (recall that a sub-array of a of length k is a contiguous part of a containing exactly k elements).

You must make the array a *k-Uniform*. To achieve it, you can apply a sequence of operations zero or more times. In one operation, you can choose an index i and remove from a all elements equal to a_i (including a_i).

Your task is to find the minimum number of such operations that you must perform to make the array a *k-Uniform*.

Input

The first line contains two positive integers n ($1 \leq n \leq 10^5$) and k ($1 \leq k \leq n$) — the size of the array and the *Uniform* value respectively.

The second line contains n positive integers a_i ($1 \leq a_i \leq 10^9$) — the numbers in the array.

Output

If it is not possible to make the array a *k-Uniform* print -1. Otherwise, print the minimum number of operations that you must perform to achieve it.

Example

Input	Output
10 3 1 2 3 1 2 4 1 2 4 4	2
8 3 5 5 6 6 5 5 6 6	1
3 1 7 8 9	0

Problem H. Hidden Professions

Source file name:	Hidden.c, Hidden.cpp, Hidden.java, Hidden.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

The Institute for Common Professions Characteristics (ICPC) is setting a way to determine how many professions a person has in a resume. But there are some professions that can be named different, for example, there are *doctors* that may write *dr* instead of the full word.

There is another problem, in some resumes, people like to exaggerate the number of professions they may have by placing a lot of abbreviations for the same career. And since there is not only one way to do it, anyone can put their careers in the order they prefer. There are even some people who put a profession that are not registered or they may even repeat their professions with a different abbreviation!

For all the careers that ICPC has registered, any prefix of a profession may be used as a reference for the profession if there is not ambiguity, for example, "*doc*" may be used as a valid abbreviation for "*doctor*". An ambiguity appears when two different professions share a prefix, then that prefix is not valid as an abbreviation for a profession. For example, if you have the professions "*doctor*" and "*dentist*", "*do*" is a valid abbreviation, but "*d*" is not, since it is a shared prefix with "*dentist*"

Your task is to write a program that determines the number of different registered professions a person has on his or her resume, given the list of professions that are considered to be equivalent and the list of professions in their resumes.

Input

The first line is a positive integer T, the number of test cases $1 \leq T \leq 200$.
Each test case begins with two integers separated by space, P ($1 \leq P \leq 30$) and R ($1 \leq R \leq 1000$), that stands for the number of professions and the number of resumes, respectively.

The next $2 \times P$ lines contain the description of the known professions. Each description of a career consist of two lines: The first line has an integer C ($1 \leq C \leq 40$) which stands for the number of known names of that career. The second line has C strings separated by space, the maximum length of this strings is 40 and the minimum is 1.

The next $2 \times R$ lines contain the description of the resumes. Each resume is also described as two lines. The first line has an integer N ($1 \leq N \leq 30$) that are the number of strings in the next line. The second line has N strings representing the professions that person has, separated by space, they might or not be registered. The maximum length of each string is 40 and the minimum is 1.

In every case the strings are composed only by lower case English letters. Is is guaranteed that there are no 2 different careers with the exact same name.

Output

For each resume, print the number of different valid professions' prefixes registered that appear on the resume

Example

Input	Output
1 3 4 2 dentist dntist 2 inga ingena 3 doctor phd dr 3 salmon sin dr 4 inga ingen ingena ing 2 ph ing 2 d perm	1 1 2 0

Problem I. Lighting System

Source file name:	Lighting.c, Lighting.cpp, Lighting.java, Lighting.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

Currently, the university campus has a system of lamps which can be turned on manually or with activation by a photocell sensor when a lamp adjacent to it is turned on (that is, the lamp turns on because in the reading range of the photocell sensor a lamp was turned on), this is a symmetric relationship. *Don Leo*, who is the administrator of the university campus lighting system, wants to determine the minimum number of lamps that he has to turn on manually, thereby guaranteeing that all the lamps on campus are turned on, additionally, the You also want to determine the number of lamps that are turned on from each of the lamps that are turned on manually.

Input

The input consists of multiple test cases. Your program must process all of these. The first line of each test case contains two non-negative integers n and p ($1 \leq n \leq 10^4$, $0 \leq p \leq n$), where n represents the number of lamps on the university campus and p represents the number of pairs of lamps that are adjacent (that is, that are at a distance less than or equal to the range in which one of the two photocell sensors in the lamps are activated by turning on the other). The next p lines contain two positive integer values a and b ($1 \leq a, b \leq n$, $a \neq b$), which indicate that the lamps a and b are adjacent. Entry ends with end of file (EOF).

Output

For each test case print k lines, where k represents the minimum number of lamps that must be turned on manually by Don Leo, in each of these lines two positive integer values must be printed c and t ($1 \leq c, t \leq n$), where c is the position of the lamp and t is the total number of lamps that are lit after manual lighting of the c lamp. Additionally, the k lines must be ordered in ascending order with respect to the positions of the lamps that are activated manually.

Don Leo is a very superstitious person, for this reason, if there is a group of lamps that can be turned on, then you must choose to manually turn on the lamp with the smallest position.

Example

Input	Output
7 6 1 3 2 4 3 4 5 6 5 7 6 7 11 8 1 2 2 3 3 4 5 6 6 8 7 8 8 9 10 11 6 0	1 4 5 3 1 4 5 5 10 2 1 1 2 1 3 1 4 1 5 1 6 1

Problem J. Johann's Function

Source file name:	Johann.c, Johann.cpp, Johann.java, Johann.py
Input:	Standard
Output:	Standard
Author(s):	Hugo Humberto Morales Peña - RPC & UTP Colombia

Typically in a first-year programming course in an engineering or computer science academic program, students are taught to build functions that make use of the doubly nested loop structure such as the following:

```
unsigned long long JohannsFunction(int n)
{
    int i, j;
    unsigned long long result = 0;

    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= i; j++)
        {
            result += j;
        }
    }

    return result;
}
```

The running-time function of the previous algorithm belongs to $O(n^2)$, with a value of $n = 10^6$ the total number of operations performed by the algorithm, in order to give the result would require 10^{12} steps. As a competitor of the different phases of the ICPC, you know that a solution that performs that amount of steps, will obtain a verdict of **Time Limit Exceeded** in competition, for this reason you are asked to propose a solution that has a running time as close to $O(1)$ as possible, regardless of the size of n, in which you must make use of all your experience as a competitor of ICPC!

Input

The input begins with a positive integer t ($1 \leq t \leq 10^6$), which represents the total number of test cases. Then t lines are presented, each one containing a positive integer n ($1 \leq n \leq 10^6$) for which the result of the **Johann's Function** must be calculated.

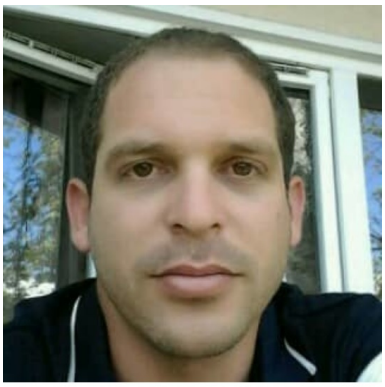
Output

The output must contain t lines, each containing a long positive integer as a result of the **Johann's Function**.

Example

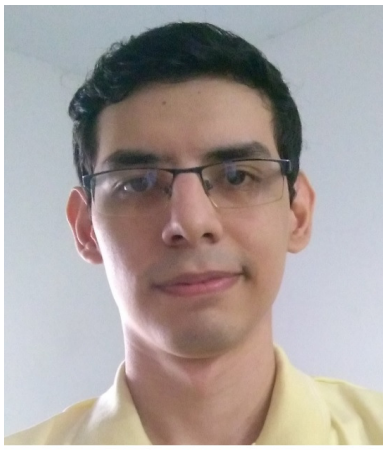
Input	Output
7 1 10 100 1000 10000 100000 1000000	1 220 171700 167167000 166716670000 166671666700000 166667166667000000

Problemsetters:



Prof. Yonny Mondelo Hernández
Universidad de las Ciencias Informáticas
Cuba

Prof. Hugo Humberto Morales Peña
Universidad Tecnológica de Pereira
Colombia



Est. José Manuel Salazar Meza
Universidad Francisco de Paula Santander
Colombia

Problemsetters and problemtesters:

Prof. Eddy Ramírez Jiménez
Universidad Nacional de Costa Rica
& Técnológico de Costa Rica

Ing. Rodrigo Chaves
Técnológico de Costa Rica
& Universidad de Costa Rica