# A 11995 - I Can Guess the Data Structure!

There is a bag-like data structure, supporting two operations:

```
1 x
```

Throw an element x into the bag.

```
2
```

Take out an element from the bag.

Given a sequence of operations with return values, you're going to guess the data structure. It is a stack (Last-In, First-Out), a queue (First-In, First-Out), a priority-queue (Always take out larger elements first) or something else that you can hardly imagine!

## Input

There are several test cases. Each test case begins with a line containing a single integer n (1<=n<=1000). Each of the next n lines is either a type-1 command, or an integer 2 followed by an integer x. That means after executing a type-2 command, we get an element x *without error*. The value of x is always a positive integer not larger than 100. The input is terminated by end-of-file (EOF). The size of input file does not exceed 1MB.

## Output

For each test case, output one of the following:

```
stack
```

It's definitely a stack.

```
queue
```

It's definitely a queue.

```
priority queue
```

It's definitely a priority queue.

```
impossible
```

It can't be a stack, a queue or a priority queue.

```
not sure
```

It can be more than one of the three data structures mentioned above.

## Sample Input

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

## Output for the Sample Input

```
queue
not sure
impossible
stack
priority queue
```

## B 11979 - Hamming Base

You are given **N** integers in base-**N** each of them having exactly **M** digits (may be with some leading zeros). Two integers are called **K**-similar if they have the same digits in exactly **K** positions. For example 321 and 213 are **0**-similar. 3456 and 6453 are **2**-similar, 123 and 453 are **1**-similar. You want to change these given **N**-integers in such a way that each pair of these integers are **0**-similar. To achieve this goal you can change the integers in several steps. In a single step you can change a single digit of a single integer by 1 (incrementing or decrementing). But you can't decrement if the digit is **0** or you can't increment if the digit is **N-1**.

You need to achieve your goal in minimum number of steps.

## Input

Input starts with an integer **T (≤ 50)**, denoting the number of test cases.

Each case starts with a line containing two integers **N (2 ≤ N ≤ 2000)** and **M (1 ≤ M ≤ 10)**. Each of the next **N** lines contains **M** integers between **0** and **N-1** inclusive. These **M** integers form an **M** digit number in base **N**.

## Output

For each case, print the case number and the minimal steps required to achieve your goal.

| Sample Input | Output for Sample Input |
|---|---|
| 2<br>3 3<br>0 0 0<br>0 0 0<br>0 0 0<br>4 2<br>0 0<br>0 0<br>0 2<br>2 0 | Case 1: 9<br>Case 2: 8 |

## C 11970 - Lucky Numbers

Every person has its own numbers that he considers lucky. Usually the numbers are fixed like 3 or 7 and do not depend on anything. This lucky number model seems to be very primitive for John, so he decided to upgrade it for his own use. Maybe more complex model will bring more luck to him?

John has added a dependency for lucky numbers on specific integer `N` (for example `N` can be ordinal number of day in year or some other meaning). For each `N` John considers some number `X` lucky if and only if fraction $X/\sqrt{N-X}$ value is integer and greater than zero.

### INPUT

The number of tests `T` ($T \leq 100$) is given on the first line. `T` lines follow, each of them contains one integer `N` ($1 \leq N \leq 10^9$) described above.

### OUTPUT

For each test case output a single line `"Case T: S"`. Where `T` is the test case number (starting from 1) and `S` is increasing sequence of numbers considered lucky by John for specified `N`. Please refer to the sample output for clarity.

### SAMPLE INPUT

```
3
16
109
33
```

### SAMPLE OUTPUT

```
Case 1: 12 15
Case 2: 108
Case 3: 24 32
```

## D 11962 - DNA II

As it was mention in the previous task, any DNA sequence consists from four bases: adenine (A), cytosine (C), guanine (G) and thymine (T) and can be written as a string constructed from characters A, C, G or T. As any strings, DNA sequences can be ordered alphabetically. In order to reduce amount of memory DNA sequence can be described by its length and index in the ordered (index starts from 0) set of all possible DNA sequences of some specific length. So you are asked to write a program that will encode any given DNA sequence in a pair of numbers.

Lets take for example all DNA sequences of length 2. They form the ordered set:
{ AA; AC; AG; AT; CA; CC; CG; CT; GA; GC; GG; GT; TA; TC; TG; TT }
So sequence "CC" can be described by the pair (2;5) (2 is a length and 5 is an index in the set), "AG" by (2,2), "TG" by (2,14) and so on.

### INPUT

The number of tests `T` ($T \leq 100$) is given on the first line. Each of next `T` lines contains DNA sequence `S` of maximal length of 30 characters.

### OUTPUT

For each test case output a single line `"Case T: (A;B)"`. Where `T` is the test case number (starting from 1) and `(A:B)` is a pair describing the DNA sequence by a given approach.

### SAMPLE INPUT

```
3
AC
ATA
TAGCAGCAGCAGCGAA
```

### SAMPLE OUTPUT

```
Case 1: (2:1)
Case 2: (3:12)
Case 3: (16:3374617184)
```

## E 11955 - Binomial Theorem

John likes mathematics a lot. His main passion is the binomial theorem. However it is rather hard to calculate binomial coefficients, so he decided to write a computer program that can expand any power of a sum into a sum of powers. Mathematically it can be written like this:
$(a+b)^k = x_1a^k + x_2a^{k-1}b + x_3a^{k-2}b^2 + \ldots + x_{k+1}b^k$
where $x_{1 \ldots k+1}$ are binomial coefficients $x_i = C_k^i$ .

### INPUT

There is a number of tests T ($T \leq 100$) on the first line. After `T` test follows. Each test is written on a single line in form of `(a+b)^k`. Where `a` and `b` are same variables names. Variables names are strings constructed from 'a'–'z' characters. And `k` ($1 \leq k \leq 50$) is a power that you need to raise the sum. You can assume that there are no lines longer than 100 characters.

### OUTPUT

For each test output a single line "`Case N: T`". Where `N` is the test number (starting from 1) and `T` is an expanded expression (see examples for clarification). By the way, you shouldn't output coefficients and powers equal to one.

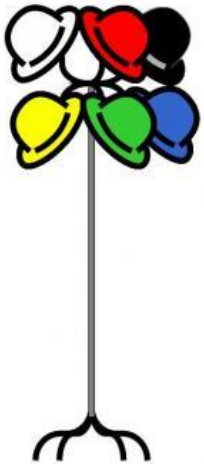
### SAMPLE INPUT

```
3
(a+b)^1
(alpha+omega)^2
(acm+icpc)^3
```

### SAMPLE OUTPUT

```
Case 1: a+b
Case 2: alpha^2+2*alpha*omega+omega^2
Case 3: acm^3+3*acm^2*icpc+3*acm*icpc^2+icpc^3
```

# F 12024 - Hats



## Background

John Hatman, the honest cloakroom attendant of the Royal Theatre of London, would like to know the solution to the following problem.

## The Problem

When the show finishes, all spectators in the theatre are in a hurry to see the Final of the UEFA Championship. So, they run to the cloakroom to take their hats back.

Some of them take a wrong hat. But, how likely is that everyone take a wrong hat?

## The Input

The first line of the input contains an integer, *t*, indicating the number of test cases. For each test case, one line appears, that contains a number *n*, 2<=*n*<=12, representing the number of people and hats.

## The Output

For each test case, the output should contain a single line with the number representing the number of favourable cases (i.e., the number of cases where all people take a wrong hat), followed by a bar, "/", and followed by a number representing the total number of possible cases.

## Sample Input

```
3
2
3
4
```

## Sample Output

```
1/2
2/6
9/24
```

# G 12027 - Very Big Perfect Squares

## Background

An integer, *n*, is called *a perfect square* if there is another integer, *m*, such that $n = m \times m$. Examples of perfect squares are 1, 4, 9, 16, 25, 36, etc. That is: 1, $2^2$, $3^2$, $4^2$, $5^2$, $6^2$, etc.

## The Problem

Given a natural number, *A*, we want to know how many perfect squares exist between 1 and *A*, inclusive. For example, between 1 and 1 there is only one perfect square (1), between 1 and 5 there are 2 (1 and 4), between 1 and 40 there are 6 (1, 4, 9, 16, 25 and 36).

We want to work with very big numbers, so we are not very interested in accuracy. We only need the first digit of the result, and the rest of digits can be 0.

## Input

The input can contain different test cases.

For each test case, there is a line with a natural number *A*.

You can assume that *A* is between 1 y $10^{1000}$, inclusive.

The output ends with a line with the value: 0.

## Output

For each test case, the output should be a line with a natural number *N*, indicating how many perfect squares exist between 1 and *A*, inclusive.

As we do not need too much precision, you only have to output the most significant digit of the result. You have to complete the rest of the result with zeros. For example, if the result is 59, then you have to output 50; if the result is 12345, then you have to output: 10000.

## Sample Input

```
1
5
40
1000
0
```

## Sample Output

```
1
2
6
30
```

## H 12010 - Boring Homework

Professor Z. always gives his students lots of boring homework. Last week, after explaining binary search trees (BSTs), he asked his students to draw a picture of BST according to the list of numbers inserted into the tree sequentially. Maryanna spent so much time playing the game "Starcraft II" that she can't finish her homework in time. She needs your help.

A binary search tree, which may sometimes also be called ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

-from Wikipedia

To draw a picture of BST, you may follow the rules listed below:

1. The picture of a 1-node BST, whose size is 1*1, is a single 'o' (15th small Latin letter).
2. If a BST has a non-empty subtree, draw a single '|' just above the subtree's root, and a single '+' just above previous drawn '|'. Finally, in the row of '+', use the least number (including 0) of '-'s to connect '+' (denoting the left subtree and right subtree) and 'o' (denoting the parent node of the subtree)
3. The left subtree (if exists) must be drawn on the left side of its parent. Similarly, the right subtree (if exists) must be drawn on the right side of its parent.
4. The column of the BST's root must not contain any character from left subtree or right subtree.
5. Any column containing any characters from BST's left subtree must not contain any characters from BST's right subtree, and vice versa. That is, for a node of the BST, the picture of its left subtree and the picture of its right subtree do not share common columns in the picture of the whole tree.

The sample output may give a clear clarification about the format of the picture.

### Input

The first line contains $T$ ( $T \leq 2500$), the number of test cases. $T$ lines follow. Each line contains a positive integer $N$ ($N < 80$), followed by $N$ integers - a permutation of 1 to $N$. The permutation indicates the insert order for the BST.

### Output

For each test case:

Output the case number counting from 1 in the first line. The next lines should be the image described above without any trailing spaces. See the sample for more format details.

**Note:** Notice that no trailing whitespaces after the last visible characters of each line are allowed.

## Sample Input

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

## Sample Output

```
Case #1:
+-o
|
o+
|
o
Case #2:
+--o+
| |
o-+ o+
| |
+o o
|
o
Case #3:
+o+
| |
+o o+
| |
o o
```

# I 12015 - Google is Feeling Lucky

Google is one of the most famous Internet search engines which hosts and develops a number of Internet-based services and products. On its search engine website, an interesting button `I'm feeling lucky' attracts our eyes. This feature could allow the user skip the search result page and goes directly to the first ranked page. Amazing! It saves a lot of time.

The question is, when one types some keywords and presses `I'm feeling lucky' button, which web page will appear? Google does a lot and comes up with excellent approaches to deal with it. In this simplified problem, let us just consider that Google assigns every web page an integer-valued relevance. The most related page will be chosen. If there is a tie, all the pages with the highest relevance are possible to be chosen.

Your task is simple, given 10 web pages and their relevance. Just pick out all the possible candidates which will be served to the user when `I'm feeling lucky'.

## Input

The input contains multiple test cases. The number of test cases *T* is in the first line of the input file.

For each test case, there are 10 lines, describing the web page and the relevance. Each line contains a character string without any blank characters denoting the URL of this web page and an integer $V_i$ denoting the relevance of this web page. The length of the URL is between 1 and 100 inclusively. ( $1 \leq V_i \leq 100$)

## Output

For each test case, output several lines which are the URLs of the web pages which are possible to be chosen. The order of the URLs is the same as the input. Please look at the sample output for further information of output format.

## Sample Input

```
2
www.youtube.com 1
www.google.com 2
www.google.com.hk 3
www.alibaba.com 10
www.taobao.com 5
www.bad.com 10
www.good.com 7
www.fudan.edu.cn 8
www.university.edu.cn 9
acm.university.edu.cn 10
www.youtube.com 1
www.google.com 2
www.google.com.hk 3
www.alibaba.com 11
www.taobao.com 5
www.bad.com 10
www.good.com 7
www.fudan.edu.cn 8
acm.university.edu.cn 9
acm.university.edu.cn 10
```

## Sample Output

```
Case #1:
www.alibaba.com
www.bad.com
acm.university.edu.cn
Case #2:
www.alibaba.com
```

## J 908 - Re-connecting Computer Sites

Consider the problem of selecting a set *T* of high-speed lines for connecting *N* computer sites, from a universe of *M* high-speed lines each connecting a pair of computer sites. Each high-speed line has a given monthly cost, and the objective is to minimize the total cost of connecting the *N* computer sites, where the total cost is the sum of the cost of each line included in set *T*. Consider further that this problem has been solved earlier for the set of *N* computer sites and *M* high-speed lines, but that a few *K* new high-speed lines have recently become available.

Your objective is to compute the new set *T'* that may yield a cost lower than the original set *T*, due to the additional *K* new high-speed lines and when *M+K* high-speed lines are available.

### Input

**The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.**

The input is organized as follows:

- A line containing the number *N* of computer sites, with *1 <= N <= 1000000*, and where each computer site is referred by a number *i*, *1 <= i <= N*.
- The set *T* of previously chosen high-speed lines, consisting of *N-1* lines, each describing a high-speed line, and containing the numbers of the two computer sites the line connects and the monthly cost of using this line. All costs are integers.
- A line containing the number *K* of new additional lines, *1 <= K <= 10*.
- *K* lines, each describing a new high-speed line, and containing the numbers of the two computer sites the line connects and the monthly cost of using this line. All costs are integers.
- A line containing the number *M* of originally available high-speed lines, with *N-1 <= M <= N (N-1) / 2*.
- *M* lines, each describing one of the originally available high-speed lines, and containing the numbers of the two computer sites the line connects and the monthly cost of using this line. All costs are integers.

### Output

**For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.**

The output file must have one line containing the original cost of connecting the *N* computer sites with *M* high-speed lines and another line containing the new cost of connecting the *N* computer sites with *M+K* high-speed lines. If the new cost equals the original cost, the same value is written twice.

## Sample Input

```
5
1 2 5
1 3 5
1 4 5
1 5 5
1
2 3 2
6
1 2 5
1 3 5
1 4 5
1 5 5
3 4 8
4 5 8
```

## Sample Output

```
20
17
```

## K 11817 - Tunnelling the Earth

There are different methods of transporting people from place to place: cars, bikes, boats, trains, planes, etc. For very long distances, people generally fly in a plane. But this has the disadvantage that the plane must fly around the curved surface of the earth. A distance travelled would be shorter if the traveller followed a straight line from one point to the other through a tunnel through the earth.

For example, travelling from Waterloo to Cairo requires a distance of 9293521 metres following the great circle route around the earth, but only 8491188 metres following the straight line through the earth.

For this problem, assume that the earth is a perfect sphere with radius of 6371009 metres.

### Input Specification

The first line of input contains a single integer, the number of test cases to follow. Each test case is one line containing four floating point numbers: the latitude and longitude of the origin of the trip, followed by the latitude and longitude of the destination of the trip. All of these measurements are in degrees. Positive numbers indicate North latitude and East longitude, while negative numbers indicate South latitude and West longitude.

### Sample Input

```
1
43.466667 -80.516667 30.058056 31.228889
```

### Output Specification

For each test case, output a line containing a single integer, the difference in the distance between the two points following the great circle route around the surface of the earth and following the straight line through the earth, in metres. Round the difference of the distances to the nearest integer number of metres.

### Output for Sample Input

```
802333
```

## L 11762 - Race to 1

Dilu have learned a new thing about integers, which is - any positive integer greater than 1 can be divided by at least one prime number less than or equal to that number. So, he is now playing with this property. He selects a number **N**. And he calls this **D**.

In each turn he randomly chooses a prime number less than or equal to **D**. If **D** is divisible by the prime number then he divides **D** by the prime number to obtain new **D**. Otherwise he keeps the old **D**. He repeats this procedure until **D** becomes 1. What is the expected nxumber of moves required for **N** to become 1.

[We say that an integer is said to be prime if its divisible by exactly two different integers. So, 1 is not a prime, by definition. List of first few primes are 2, 3, 5, 7, 11, …]

## Input

Input will start with an integer **T (T <= 1000)**, which indicates the number of test cases. Each of the next **T** lines will contain one integer **N (1 <= N <= 1000000)**.

## *Output*

For each test case output a single line giving the case number followed by the expected number of turn required. Errors up to 1e-6 will be accepted.

| Sample Input | Output for Sample Input |
|---|---|
| 3<br>1<br>3<br>13 | Case 1: 0.0000000000<br>Case 2: 2.0000000000<br>Case 3: 6.0000000000 |