AVL++
Problem ID: avlplusplus

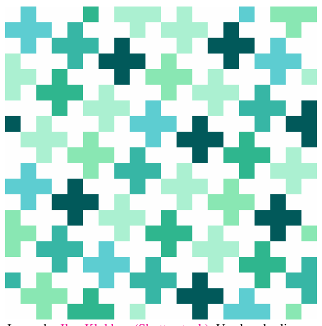
Image by Ilya Klukhov (Shutterstock), Used under license

An AVL tree, named after its inventors, Adelson-Velskii and Landis, is a binary search tree with an additional constraint that keeps the tree balanced. The *balance factor* of a node x, denoted $bf(x)$, is defined to be $H(x_R) - H(x_L)$, where $H()$ denotes height, x_R is the right subtree of x, and x_L is the left subtree of x. (The height of a tree is the number of edges in the longest simple path from the root to any leaf. It follows that the height of a tree containing a single node is 0. The height of the empty tree, i.e., the tree with no nodes, is typically defined to be -1.) In an AVL tree, the only balance factors permitted are $\{-1, 0, 1\}$, i.e., $|bf(x)| \leq 1$ for every node x.

Two AVL trees T_1 and T_2 are *isomorphic* if there is a bijection $f : X_1 \rightarrow X_2$, where X_1 is the set of nodes in T_1, and X_2 is the set of nodes in T_2, such that for all $x, y \in X_1$, y is the right (respectively, left) child of x in T_1 if and only if $f(y)$ is the right (respectively, left) child of $f(x)$ in T_2. In other words, two AVL trees are isomorphic if they have the same "shape" (we are not concerned with the values stored in the nodes).

Now consider one way to generalize the AVL constraint. Let $m \geq 0$ be an integer, and define an AVL-m tree to be a binary search tree satisfying $|bf(x)| \leq m$ for every node x. This gives rise to an interesting question: How many differently shaped (non-isomorphic) AVL-m trees are there with a given height h? The figure below illustrates the answer when $m = 2$ and $h = 2$.

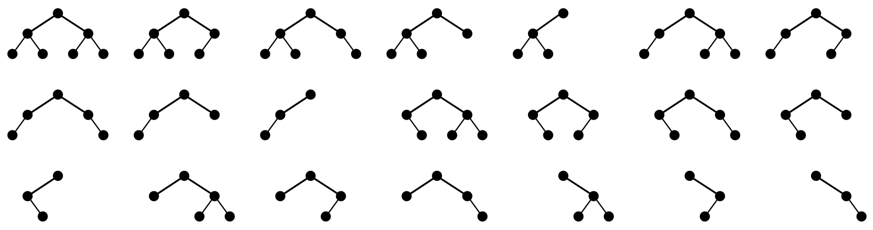

Figure 1: There are 21 AVL-2 trees with height 2

Input

The first line of input contains an integer Q $(1 \leq Q \leq 100)$ indicating the number of queries to follow. Each of the next Q lines contains a query consisting of two space-separated integers m and h $(0 \leq m \leq 100, 0 \leq h \leq 1000)$.

Output

For each query, output a line containing the number of non-isomorphic AVL-m trees with height h. Since these numbers may be large, report each answer mod $(10^9 + 7)$.

Sample Input 1	Sample Output 1

4
1 1
2 1
0 2
2 2
	3
	3
	1
	21

Baseball Packs
Problem ID: baseball

In Major League Baseball's American League there are 10 positions a player can occupy as follows: First Base (1B), Second Base (2B), Shortstop (SS), Third Base (3B), Catcher (C), Pitcher (P), Left Field (LF), Center Field (CF), Right Field (RF) and Designated Hitter (DH). Suppose Baseball cards come in packages of 10 cards per pack and that each card lists a single player (their name) with the positions that player is qualified to play. For example a baseball card may list:

- Ruth: 1B, P, LF, CF, RF, X

Indicating that Ruth is a baseball player who plays any of first base, pitcher, left field, center field or right field. The card doesn't list DH however it is always assumed that a player can be played as a designated hitter as well even though it is never listed as a qualified position. The list of qualified positions is terminated with an X (X is not a position).

Your task is to write a program that determines given a list of 10 baseball cards in a pack, whether a full baseball team can be fielded or not from those cards. I.e., can all of the 10 positions be filled with unique players who are qualified to play their assigned position. Aside from DH for which any player is able to play, a player may be assigned any single position listed on their baseball card.

Input

Input is a pack containing 10 baseball cards, one card per line. A Baseball card is formatted with "Name: p1, p2, ..., px, X" where each of the pi's is a position for which the player is qualified to play and is always terminated with 'X'

Note: DH is never listed as a qualified position but is always assumed to be a position that a player is qualified to play, and Names are a single sequence of upper and lower case letters terminated by a colon ':'.

Output

The output is a single word either True or False. True if those 10 cards may be distributed such that a unique player occupies each of the 10 positions in baseball and is qualified to occupy that position and False otherwise.

Note: Player names will be unique (containing only letters Aa-Zz) and a single player will not show up multiple times in the same pack of cards.

Sample Input 1

```
Ruth: 1B, P, LF, CF, RF, X
Musial: 1B, LF, RF, X
Jenkins: P, X
Cobb: CF, X
Berra: C, LF, X
Mantle: CF, 1B, X
Hornsby: 2B, SS, 3B, X
Wagner: SS, RF, 1B, X
Schmidt: 1B, 3B, X
Brett: 1B, 3B, X
```

Sample Output 1

```
True
```

Sample Input 2

```
Ruth: 1B, P, LF, CF, RF, X
Musial: 1B, LF, RF, X
Papi: 1B, X
Cobb: CF, X
Berra: C, LF, X
Mantle: CF, 1B, X
Hornsby: 2B, SS, 3B, X
Wagner: SS, RF, 1B, X
Schmidt: 1B, 3B, X
Brett: 1B, 3B, X
```

Sample Output 2

```
False
```

Euphemisms-Shmeuphemisms

Problem ID: euphemisms

Image by Lewis Clarke (geograph), Used under license

As stated in Wikipedia, "A euphemism is a generally innocuous word or expression used in place of one that may be found offensive or suggest something unpleasant". Thus, you are never *lost*, but "geographically mislocated". Some other creative examples include "penetrating deliverer of kinetic energy" for bullet, and the now classic "enhanced interrogation methods".

The administration of the local Bitville newspaper, being risk-averse and mindful of political correctness, introduces a list of forbidden words, along with the corresponding euphemisms to replace them with. Too late to teach an old dog new tricks! The staff members, who are supposed to replace the forbidden words with the officially endorsed equivalents, concurred on the following strategy instead: they will continue writing as they did before, but will simply remove some characters from their articles, so that no forbidden word appears in the text. After all, a Bitville text is merely a contiguous stretch of 0s and 1s, and as long as you remove the absolute minimum number of characters, you can hope no one will spot the difference! Your job is, given a list of forbidden words $w_1, w_2, \ldots, w_n$ and the text T to edit, to find the minimum number of characters to remove from T, so that none of the strings $w_1, w_2, \ldots, w_n$ occurs as a substring in the text T.

Input

The first line contains an integer $ts, 1 \leq ts \leq 100$, denoting the number of test cases. Then follow ts blocks, in the following format:

- a single line with an integer $n, 1 \leq n \leq 30$ – the number of forbidden Bitville-words
- n lines, each line containing a forbidden word w_i, with length $1 \leq |w_i| \leq 13$
- finally, a line with the text T to edit, with length $1 \leq |T| \leq 3000$.

Thus, a test case with n forbidden words consists of $n + 2$ lines. You are guaranteed that words and the text are non-empty, will not contain any leading or trailing blanks, and are composed of solely 0s and 1s.

Output

Per test case, a single line in the format "`Case #t: ans`", where "`t`" stands for the number of the test case (starting from 1), and "`ans`" is the answer to the problem, which is an integer indicating the minimum number of characters to remove for this test case.

Sample Input 1

```
3
2
0
1
101
1
0
111
2
01
010
1010
```

Sample Output 1

```
Case #1: 3
Case #2: 0
Case #3: 1
```

Groupthink

Problem ID: groupthink

Image by kabliczech (iStock), Used under license

One of the most fundamental structures in abstract algebra is called a *magma*. A magma is a pair $(M, \bullet)$, where M is a nonempty set and $\bullet$ is a binary operation on M. This means that $\bullet$ maps each ordered pair of elements (x, y) in M to an element in M (in other words, $\bullet$ is a function from $M \times M$ to M). The element to which $\bullet$ maps (x, y) is denoted $x \bullet y$.

Magmas that satisfy certain properties are given more specialized names. Here are three important properties that $(M, \bullet)$ could satisfy:

- **P1** *(associativity)*: For all $x, y, z \in M$, $(x \bullet y) \bullet z = x \bullet (y \bullet z)$.
- **P2** *(identity)*: There is an element $I \in M$ such that $x \bullet I = I \bullet x = x$ for all $x \in M$ (I is called an *identity*).
- **P3** [depends on **P2**] *(inverses)*: There is an identity $I \in M$, and for every $x \in M$, there exists $x^* \in M$ such that $x \bullet x^* = x^* \bullet x = I$ (x^* is called an *inverse* of x).

A magma that satisfies **P1** is called a *semigroup*. A semigroup that satisfies **P2** is called a *monoid*. A monoid that satisfies **P3** is called a *group*. Given a magma $(M, \bullet)$, determine its most specialized name.

Input

The input consists of a single test case specifying a magma $(M, \bullet)$. The first line contains an integer n ($1 \leq n \leq 120$), the size of M. Assume that the elements of M are indexed $0, 1, 2, \ldots, n-1$. Each of the next n^2 lines contains three integers i j k ($0 \leq i, j, k \leq n-1$), meaning that $x \bullet y = z$, where x is the element indexed by i, y is the element indexed by j, and z is the element indexed by k. Each ordered pair (i, j) will appear exactly once as the first two values on a line.

Output

The output consists of a single line. Output **`group`** if $(M, \bullet)$ satisfies **P1**, **P2**, and **P3**. Output **`monoid`** if $(M, \bullet)$ satisfies **P1** and **P2**, but not **P3**. Output **`semigroup`** if $(M, \bullet)$ satisfies **P1**, but not **P2** (and therefore not **P3**). Otherwise, output **`magma`**.

Sample Input 1

```
2
0 0 0
0 1 1
1 0 1
1 1 0
```

Sample Output 1

```
group
```

Sample Input 2

```
3
1 2 1
2 0 0
0 1 0
2 2 2
1 0 0
0 0 0
2 1 1
0 2 0
1 1 2
```

Sample Output 2

```
monoid
```

Integrity for spies
Problem ID: integrity

Two friends decide to communicate with each other using a secret code over numbers from 0 to $b - 1$. Each message can be arbitrarily long, but it must be validated using the following formula:

$$\text{current value of the message} = (\text{value of the message read so far}) * b + \text{new number read.}$$

The `current value of the message` at the end of the message must be a multiple of k.

Input

The input starts with n, the number of test cases, on a line by itself. The remainder of the file contains n test cases. Each test case consists of b, m (the number of messages in this test case), and k on lines by themselves, followed by m messages one per line, each line terminated by a -1. You may assume $2 \leq b \leq 36$. Schematically, the input looks like

```
n
b_1
m_1
k_1
message 1 of test case 1
...
message m1 of test case 1
...
b_2
m_2
k_2
m_2 messages of test case 2
the other n-2 test cases
```

Output

Output is a sequence of 0's and 1's, 1 for valid messages and 0 for those that are non-valid, i.e., n binary strings (each of length m_i for test case i). Messages with numbers larger than $b - 1$ should be considered invalid.

Sample Input 1

```
3
4
10
5
3 0 0 0 0 0 0 -1
3 0 0 0 0 0 1 -1
3 0 0 0 0 0 2 -1
3 0 0 0 0 0 3 -1
3 0 0 0 0 1 0 -1
1 0 0 0 1 -1
2 0 0 0 4 -1
2 0 0 0 1 -1
2 0 0 0 2 -1
2 0 0 0 3 -1
6
7
5
1 0 0 0 0 0 2 -1
1 0 0 3 0 0 2 -1
1 0 0 0 1 -1
2 0 0 0 4 -1
2 0 0 0 1 -1
2 0 0 0 2 -1
2 0 0 0 3 -1
4
6
5
1 0 0 0 1 0 2 -1
2 0 0 1 0 0 2 -1
3 0 1 0 0 0 2 -1
1 1 0 0 0 0 2 -1
1 1 0 0 0 0 0 -1
1 1 0 0 0 0 1 -1
```

Sample Output 1

```
0010000001
0000001
000010
```

Sample Input 2

```
2
17
4
101
5 16 -1
5 15 -1
11 15 -1
2 1 5 1 -1
33
3
1234
1 4 13 -1
1 9 12 10 4 -1
1 9 12 10 5 -1
```

Sample Output 2

```
1011
110
```

RPN Calculator
Problem ID: rpn

Your goal is to write a postfix ("Reverse Polish Notation") calculator. Unlike the probably more familiar "infix" notation, in postfix notation the numbers are given first, then the desired operation. The following example is from Wikipedia. The infix notation

$$((15/(7-(1+1)))*3)-(2+(1+1))$$

can be written as

$$15 \quad 7 \quad 1 \quad 1 \quad + \quad - \quad / \quad 3 \quad * \quad 2 \quad 1 \quad 1 \quad + \quad + \quad -$$

It can be evaluated by what Wikipedia calls the "left to right" algorithm as follows:

```
15 7 1 1 + - / 3 * 2 1 1 + + - =
15 7     2 - / 3 * 2 1 1 + + - =
15         5 / 3 * 2 1 1 + + - =
             3 3 * 2 1 1 + + - =
                 9 2 1 1 + + - =
                 9 2     2 + - =
                 9         4 - =
                             5
```

Your program will read and process lines of input one at a time, until it reads `quit` on a line by itself. Each line of input will either be an integer, or a string representing an operation. If the input is a number, your program should save it on a stack. If the input is an arithmetic operation, your program should remove the two most recently saved numbers, and replace them with the result of the operation. If before the operation, the most recently added number (i.e. the top of the stack) is b, and the second most recently added number is a, you should replace them with

operation	replacement
$+$	$a+b$
$-$	$a-b$
$/$	$\lfloor a/b \rfloor$
$*$	$a*b$
^	a^b

The non-arithmetic operations should be implemented as follows:

dup	copy the top element of the stack
print	print the top element of the stack
pop	remove the top element of the stack
swap	interchange the top two elements on the stack
quit	Stop reading and processing input

Input

Each line of input will either be an integer, or a string $+, -, *, /,$ ^, `dup`, `pop`, `print`, `quit`, or `swap`. You may assume that there will always be sufficient arguments for each operation (i.e. there is no "stack underflow"), and that no division by zero is attempted.

Output

The output should be the output from the print operations in the input (if any).

Sample Input 1

```
15
7
1
1
+
-
/
3
*
2
1
1
+
+
-
print
quit
```

Sample Output 1

```
5
```

Sample Input 2

```
17
3
2
*
print
dup
dup
pop
+
3
/
print
swap
-
13
+
2
^
print
quit
```

Sample Output 2

```
6
4
0
```

Sample Input 3

```
2
128
^
3
/
-2
*
print
quit
```

Sample Output 3

```
-226854911280625642308916404954512140970
```

Optimizing Stock on Shelves
Problem ID: shelvesproblem

Your friend, who is a store manager, is looking at optimising his sales by reorganizing his products strategically on the shelves. Each product being sold comes in one box, to be put on the shelves. The box contains multiple such items, and we assume that we will not run out of items. The boxes are of various sizes though. Note: the size here refers only to the visible (front) length of the box, not how tall it is or how deep it is. You cannot turn the box 90 degrees to reduce or enlarge its visible side.

The display area contains 4 shelves, each of a fixed length (provided in the input). When placing boxes on one shelf, you have to make sure that they all fit (i.e., the sum of the boxes' lengths does not exceed the size of the shelf). The input may contain more boxes or less than what can fit in the entire display area. Each box contain a different product.

The placement of a product on the shelves will have an impact on how many will be sold. For example, being on the top (4th) shelf brings higher visibility usually, so more customers will see it and buy it. Only the vertical placement is important here to the expected sales of the product, not the horizontal placement (i.e., we just want to know on which shelf, between 1 and 4, each box should go). For each product, you can actually evaluate how much can be sold (in dollars, referred to as "expected sales") when placed on each of the shelves. For simplicity, this is a function of the form $f(i) = a + b * i$ where i is the shelf number (between 1 and 4). Those functions are different for each product, and are provided in the input.

What you have to do is to find the optimal placement of the boxes that maximizes the sum of the expected sales of the products on display.

Input:

The first line contains one positive integer, smaller than 20, representing the size (length) of the shelves in the display area. The second line contains one positive integer N, smaller than 100, representing the number of boxes available, to be put on the shelves. Then there are N lines, one per box, each containing 3 integers: the first one (always positive, and smaller than 20) represent the size (length) of the box. The other two numbers are the a and b coefficients in the formula for the expected sales. Note that a or b might be negative, but the function $f(i)$ will always evaluate to a positive integer for the range $i = 1 \ldots 4$.

Output:

A single integer, representing the maximum total expected sales.

In the first example provided below, the optimal solution of 49 is achieved in the following way:

On row 4, put boxes $(3, 1, 4)$ and $(2, 1, 2)$ (expected sales $= 17 + 9 = 26$)

On row 3, put boxes $(3, 2, 3)$ and $(1, -1, 2)$ (expected sales $= 11 + 5 = 16$)

On row 2, do not put any boxes

On row 1, put box $(2, 8, -1)$ (expected sales $= 7$)

In the second example, the optimal result of 91 is achieved in the following way:

On row 4, put boxes $(2, 4, 4)$, $(1, 3, 3)$, and $(1, 1, 1)$ (expected sales $= 20 + 15 + 5 = 40$)

On row 3, put boxes $(3, 10, 4)$ and $(1, 1, 1)$ (expected sales $= 22 + 4 = 26$)

On row 2, put 4 boxes $(1, 1, 1)$ (expected sales $= 3 + 3 + 3 + 3 = 12$)

On row 1, put 3 boxes $(1, 1, 1)$ and the box $(19, -2)$ (expected sales $= 2 + 2 + 2 + 7 = 13$)

Sample Input 1	Sample Output 1

5
5
3 1 4
3 2 3
2 1 2
2 8 -1
1 -1 2
	49

Sample Input 2

```
4
18
1 1 1
1 1 1
1 1 1
1 9 -2
1 1 1
1 1 1
1 1 1
2 4 4
1 1 1
3 10 4
1 1 1
1 1 1
1 3 3
1 1 1
1 1 1
1 1 1
1 1 1
1 1 1
```

Sample Output 2

```
91
```

Slider Puzzle
Problem ID: sliderpuzzle

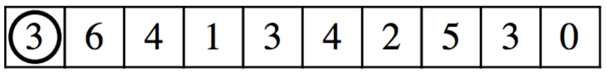

Image by Scott Bateman, Used under license

A slider puzzle consists of a row of squares each of which contains a single digit integer, like the illustration.

A circle marker starts on the initial square on the left and can be moved to other squares along the row. At each step in the puzzle, you may only move the marker the number of squares indicated by the integer in the square it currently occupies, but it the marker can be moved either left or right along the row. The marker may not move past either end of the puzzle. For example, in the puzzle above the only legal first move is to move the marker three squares to the right because there is no room to move three spaces to the left.

The goal of the puzzle is to move the marker to the 0 at the far end of the row, on the right. However, not every puzzle has a valid solution.

Input

Input for this problem will be a number between 1 and 100 inclusive representing the number of puzzles in the input, each puzzle is between 10 and 101 digits long. Each individual puzzle will be on one line and consist of a series of integers separated by spaces.

Output

Output should simply state whether "Puzzle N is solvable." or "Puzzle N is not solvable." where N refers to the puzzle instance.

Sample Input 1	Sample Output 1

2
3 6 4 1 3 4 2 5 3 0
3 2 1 1 1 2 2 5 3 0
	Puzzle 1 is solvable.
	Puzzle 2 is not solvable.