Problem A: A+B

Our first problem today is as easy as $a + b$.

Problem specification

You are given a string of digits. Rearrange those digits to build two nonnegative integers a and b such that the sum $a + b$ is as large as possible.

Each number must consist of at least one digit. Leading zeros are not allowed, but the number zero consisting of a single digit 0 is allowed. You have to use each digit exactly as many times as it occurs in the given input string.

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line. Each test case consists of a single line containing one string of digits. If there are more than 2 digits in the string, not all of them are zeros.

In the **easy subproblem A1** we have $t = 900$ and each test case consists of exactly 3 digits.

In the **hard subproblem A2** we have $t = 1000$ and the number of digits in each test case is between 2 and 16, inclusive.

Output specification

For each test case, output a single line with a single integer: the largest sum that can be achieved.

Example

input

```
4

001

175

21

444444
```

output

```
10
76
3
44448
```

The first two test cases could appear in the easy input file `a1.in`*, the other two could only appear in the hard input file* `a2.in`*.*

One optimal arrangement of digits for each example test case: 10+0, 71+5, 1+2, and 44444+4.

Problem B: Bawdy Boil-brained Bugbear

A certain gentleman shamelessly ate another gentleman's cake. This inexcusable act led to a long merciless battle between them. Of course, proper gentlemen never fight with their fists – they use carefully chosen insults built of words from Shakespeare's time. This is how the battle went on:

"Thou atest my cake, thou bawdy doghearted nut-hook!"
"Aye? But thou art a goatish half-faced bugbear."
"Fain I am not a goatish doghearted puttock as thou."
"Thou art a bawdy half-faced nut-hook."
"Thou bawdy doghearted bladder!"
"Thou goatish half-faced puttock!"
"Thou spongy boil-brained bladder!"
"Thou bawdy boil-brained bugbear!"
"Fie, thou winnest!"

Problem specification

A valid insult consists of three words: The first one must be a *simple insulting adjective*, the second one must be a *compound insulting adjective*, and the third one must be an *insulting noun*. For each of these categories we give you a list of available words. Every word has a certain strength; the strength of an insult is the sum of the strengths of its three words.

You are then given a list of insults. For each one of them you have to come up with an appropriate response – an insult which is by 1 unit stronger than the challenging insult. You are not allowed to use the same response more than once.

Input specification

The input consists of four parts. The first three parts describe the three wordlists that the insults are built from. They have the same structure: The first line contains an integer m denoting the number of words in the wordlist. Then m lines follow; on each line there is a word and its strength (a positive integer). Words consist of lowercase English letters and at most one hyphen (-). Each word is between 1 and 15 characters long (inclusive). The words in all three wordlists are distinct.

The last part of the input describes a list of insults. On the first line there is an integer n denoting the number of insults. On each of the following n lines there are three space-separated words. You are guaranteed that the insults are valid with respect to the former wordlists.

In the **easy subproblem B1** we have $m = 50$ and $n = 500$. Maximum word strength is 10^3.

In the **hard subproblem B2** we have $m = 10\,000$ and $n = 10\,000$. Maximum word strength is 10^6.

Output specification

Output n lines. For each i, the i-th line should contain an insult whose strength is exactly one greater than the strength of the i-th insult in the input. You cannot use the same insult more than once. (I.e., your output must contain n distinct lines.)

You are guaranteed that there is a sufficient amount of appropriate insults.

Example

input

```
3
bawdy 6
goatish 4
spongy 1
3
boil-brained 10
half-faced 6
doghearted 3
3
bladder 7
bugbear 3
puttock 7
7
spongy boil-brained bladder
goatish half-faced bugbear
bawdy half-faced bugbear
spongy doghearted puttock
goatish half-faced bugbear
goatish half-faced bugbear
bawdy doghearted bladder
```

output

```
bawdy boil-brained bugbear
goatish doghearted bladder
bawdy doghearted puttock
bawdy doghearted bugbear
spongy half-faced puttock
spongy boil-brained bugbear
goatish half-faced bladder
```

The strengths of the input insults are 18, 13, 15, 11, 13, 13, and 16.

The strengths of the output insults are 19, 14, 16, 12, 14, 14, and 17. Note that we had to use three different insults with strength 14 each.

The conversation in the beginning of this task contains a sequence of valid insults if we consider the wordlists from this sample input and an additional insulting noun 'nut-hook' of strength 3. Each insult is an appropriate response to the previous one. There is no valid response to the last insult of strength 19.

Problem C: Chess Pieces

Yes, we do have an actual chess problem this year. In both subproblems we consider a standard chess game. The game is played:

- on a standard 8×8 chessboard from the standard initial configuration
- according to all standard chess rules that matter (e.g., including castling and en passant)

Problem specification

At the beginning of a chess game there are at most 8 pieces of each type on the board. For example, there are 8 white pawns, 8 black pawns, 2 white rooks, and only 1 black queen. In the **easy subproblem C1** we want you to produce any valid sequence of chess moves that will lead to a board that contains *more than 8 pieces* of any specific type.

At the beginning of a chess game there are 4 rooks on the chessboard: two white and two black ones. In the **hard subproblem C2** we want you to produce any valid sequence of chess moves that will lead to a board that contains *the largest possible total number of rooks*.

Input specification

There is no input.

Output specification

Output a valid plain text (7-bit ASCII) file containing a sequence of alternating white's and black's halfmoves in PGN notation. Any sequence of moves will be accepted as long as the final configuration has the desired property. More technical details are given at the end of this problem statement.

Example output

```
1. a4 b5    2. axb5 Nc6    3. b6 Nf6    4. b7 Nd5    5. bxa8=N Ndb4    6. f3 g6
7. g4 Bh6    8. Bh3 Nd3+    9. Kf1 O-O    10. Kg2
```

This is not a valid solution to any subproblem. It is just an arbitrary sequence of moves. The example has valid syntax that can be correctly parsed. Note that in turn 5 the white pawn got promoted to a knight instead of a queen.

Technical details

Your submissions will be parsed with the `chess.pgn.read_game` method of the python-chess library, version 0.8.1. Hence, we expect you to submit the record of a game in a valid PGN notation. The library should be pretty tolerant – it should be able to parse anything valid, including comments, headers, NAGs and games with multiple variations (i.e., branches of play). Still, **we recommend that you just submit a plain ASCII log that is as simple as possible**. The example above probably contains all the notation you'll need to use.

In particular, if the notation of your game contains variations, we give you no guarantee about which branch we will choose to evaluate.

Problem D: Dijkstra's Nightmare

Every computer scientist should be familiar with Dijkstra's algorithm: the basic algorithm to compute single-source shortest paths in a given graph. When executing the algorithm, we maintain a set of *active* vertices. In each iteration, we select and process the active vertex with the smallest current distance. When processing a vertex, we examine the outgoing edges. Whenever one of them can be used to improve the distance to an adjacent vertex, we do so and mark that adjacent vertex as active.

It is well-known that the algorithm is efficient whenever all edge lengths are non-negative. The reference implementation we provided in this problem runs in $O(m \log n)$ time for such graphs (where m is the number of edges and n is the number of vertices).

Things start getting hairy once we allow edges with negative lengths. Finding the shortest *simple* path (i.e., a path with no repeated vertices) in such a graph is actually an NP-hard problem. Some versions of Dijkstra's algorithm will terminate quickly for such graphs but sometimes they will give incorrect results. This is not the case for our reference implementation. Our reference implementation is actually solving a slightly different problem: for each vertex v, we are looking for the length of the shortest *walk* from the starting vertex to v. (A walk is a path that may contain repeated vertices and edges. Note that if all edge lengths are positive, the shortest walk has to be a simple path.)

Sometimes there is no shortest walk from the starting vertex to some vertex v, because for every walk we can find an even shorter one. For such inputs our reference implementation never terminates.

Problem specification

In this problem, the word "graph" always denotes a directed weighted graph with no duplicate edges and no self-loops. The letters n and m denote its numbers of vertices and edges. Vertices are numbered 0 through $n-1$. The starting vertex is 0.

We want you to show that there are graphs for which Dijkstra's algorithm terminates, but its time complexity is exponential in the number of vertices.

The reference implementation contains a variable named `PROCESSED_VERTICES`. In the **easy subproblem D1**, construct any valid graph (defined below) such that our implementation will terminate on it after finitely many steps, and the value of `PROCESSED_VERTICES` at the end will be at least 10 000.

In the **hard subproblem D2**, you are given several values p. For each p, construct any valid graph for which our implementation will terminate after finitely many steps, and the value of `PROCESSED_VERTICES` at the end will be exactly p.

Input specification

You are given the files `d.cc` and `d.py`. These contain equivalent reference implementations in C++11 and in Python 2.

You are also given the file `d2.in`. The first line of this file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line. Each test case consists of a single line containing the number p. You may assume that $1 \leq p \leq 10\,000\,000$.

Output specification

In a valid graph, $1 \leq n \leq 60$ and the length of each edge fits into a signed 32-bit integer variable.

Each graph should be output as a sequence of $m+2$ lines. The first of these lines should contain n, the second line should contain m, and each of the following m lines should contain three integers describing an edge – the vertex numbers of its two endpoints (between 0 and $n-1$) followed by its length.

The output for the easy subproblem D1 should contain exactly one such graph. The output for the hard subproblem D2 should contain a sequence of t such graphs. Do not output any empty lines between them.

Example (hard subproblem)

input

```
1

6
```

output

```
12
7
1 3 10
0 7 12
7 11 -4
0 3 9
3 7 1
11 1 12
0 11 9
```

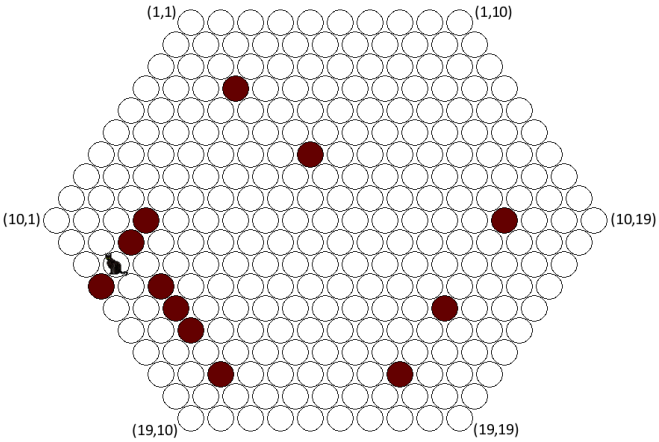
For this graph, our implementation will process exactly 6 vertices: 0, 3, 11, 7, 11 (again), and 1.

Problem E: Enclosure

"Enclosure" is a simple turn-based game. The game is played by two players. One of them is you, the other is an evil cat. The cat is located on a finite hexagonal grid. Whenever it's the cat's turn, the cat moves from its current hex to one of the neighboring hexes. (The cat cannot remain still, it always has to move.) Whenever it's your turn, you get to click on one of the empty cells to block it off forever. You start the game by blocking the first cell.

The cat wins the game if it reaches the border of the hexagon. You win the game if you block the cat completely – i.e., you win if the cat doesn't have any valid moves left.

A sample situation during the game is shown in the figure below. The blocked cells are filled. The image also shows the coordinate system used in the game. The cat starts in the middle: at $(10, 10)$.



Problem specification

Catch the cat!
(Also see below for the special Cat Challenge!)

Input specification

The cat's moves are completely deterministic. You are given an implementation of the cat's algorithm in the file `catlib.py`. You are also given several tools to help you interact with the cat:

- The script `cat-commandline.py` is a simple command line interface to the library. It alternately reads your move from the standard input and prints the cat's move to the standard output.
- The script `cat-pygame.py` is a simple implementation of the game using the Pygame library.
- Whenever you win or lose a game, the game library automatically appends a log of the game to the file `log.txt`. (So, for example, if you just won a game using the Pygame interface, you can find a log of that game at the end of the log file.)

Output specification

The output for the **easy subproblem E1** should contain any sequence of valid moves that catches the cat. The output for the **hard subproblem E2** should contain any sequence of **at most 17** valid moves that catches the cat. You may use any whitespace.

For a game in which you made m moves, the output should contain a sequence of $2m$ integers: the coordinates of blocked cells, in order. It should **not** contain any parentheses, commas, or the strings "`NEW GAME`" and "`WINNER:`" that are added to the game log by the library.

Cat Challenge!

This problem comes with an additional special challenge! If you are bored with the rest of the problem set, you can keep looking for the shortest solution you can find.

Note that the contest system will only allow you to submit a correct solution to each subproblem once. Once you solve both subproblems, you will not be able to make additional submissions. If you later improve your solution, you may e-mail the new, shorter game log to `ipscreg@ksp.sk` with the subject "Cat Challenge".

At the end of IPSC 2015 we will announce the three teams with the shortest solutions (breaking ties by submission time / e-mail reception time). The top three teams will also be enshrined for all eternity in the solution booklet.

Problem F: Familiar Couples

The rural village Spišský Štvrtok is populated by n married couples – n men and n women, both labeled from 1 to n in such a way that for each i, man i is married to woman i.

Whenever two men meet in the village pub, they become friends. Friendship lasts forever. We say two men are *acquaintances* if they are friends, or if they have a mutual acquaintance. This means that if man a and man b become friends, they also become acquaintances, and all acquaintances of a become acquainted with all acquaintances of b. Similarly, two women can become friends. We say two women are acquaintances if they are friends, or if they have a mutual acquaintance.

Couples living in the village can be familiar with each other. We say couple a is familiar with couple b (they're a "familiar pair of couples") if man a and man b are acquaintances, and woman a and woman b are also acquaintances.

Problem statement

At the beginning, no two people are friends with each other. Then q events happen in sequence. Each event is either a meeting between two men or a meeting between two women. If the two people who met aren't friends yet, they become friends.

After each event, compute the number of familiar pairs of couples. That is, count the number of pairs a, b ($a \neq b$) such that man a is acquainted with man b and woman a is acquainted with woman b. Note that the pairs are *unordered* – for example, $1, 2$ is the same pair as $2, 1$.

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a line containing the integers n and q. The i-th of the following q lines contains integers t_i, a_i, b_i ($1 \leq t_i \leq 2; 1 \leq a_i, b_i \leq n; a_i \neq b_i$) describing event i. If t_i is 1, man a_i and man b_i meet. If t_i is 2, woman a_i and woman b_i meet.

In the **easy subproblem F1**, $2 \leq n \leq 1\,500$ and $1 \leq q \leq 2\,000$.

In the **hard subproblem F2**, $2 \leq n \leq 1\,000\,000$ and $1 \leq q \leq 1\,000\,000$. Because the input file size for subproblem F2 is about 50 MB, you cannot download it directly. Instead, we have provided a small Python 2 program that will generate the file `f2.in` when executed.

Output specification

For each test case: Let y_i be the number of familiar pairs of couples after event i. Then, let z_i be $i \cdot y_i$. Output one line with a single number: the sum of $z_1 + \ldots + z_q$ modulo $10^9 + 7$.

Example

input	output
1 3 5 1 1 2 2 1 3 1 2 3 1 3 1 2 2 1	22

After event 3, couple 1 is familiar with couple 3. After event 5, all couples are familiar with each other. Together, we get $1 \cdot 0 + 2 \cdot 0 + 3 \cdot 1 + 4 \cdot 1 + 5 \cdot 3 = 22$.

Problem G: Generating Synergy

You are working for IPSCorp, a multinational corporation which is revolutionizing the world by providing end-to-end solutions for high-impact paradigm shifts. This important task requires that key players touch base with industry leaders to incentivize core competencies and faciliate organic growth.

As you can probably guess from this description, most of IPSCorp consists of layers and layers of incompetent managers. Since they need to look productive, they spend their workdays attending endless meetings and sending pointless memos to their subordinates. Making business decisions is hard, so the memos are always about trivial issues such as the office dress code.

When a manager writes a memo, he only sends it to his direct reports (i.e. subordinates of which he is the direct supervisor). They read the memo and forward it to their own direct reports. This way, the memo goes deeper and deeper in the company hierarchy. But after the memo has been forwarded a certain number of times, the recipients will just ignore it, hoping that the original author won't notice. This depends on the memo's general tone, number of exclamation marks, firing threats, and so on. We call the maximum number of forwards the "importance level".

A memo of importance level 0 ("I think we should wear red ties.") only affects the memo's author – he starts wearing a red tie, but even his direct reports ignore it. A memo of importance level 1 ("I want everyone in this room to wear blue ties from now on.") affects the author and his direct reports, but nobody else cares. A memo of importance level 2 ("New department policy: only green ties allowed!") also affects the direct reports of the author's direct reports. And so on.

Problem description

IPSCorp consists of n employees, labeled from 1 to n and organized in a hierarchical tree. All employees own ties of c colors labeled from 1 to c.

At the beginning, everyone is wearing a tie of color 1. Then, q events happen. Events are of two types: Either a given person writes a memo of a given importance level, and its recipients (including the person who wrote it) start wearing ties of a given color, or we want to know what tie a given employee is wearing at the moment.

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line.

Every test case starts with a line containing the integers n, c and q. The next line contains $n-1$ integers named s_2 to s_n, where s_i is the supervisor of employee i ($1 \leq s_i < i$). Employee 1 is the company president and has no supervisor.

The i-th of the following q lines contains three integers a_i, l_i, c_i describing event i ($1 \leq a_i \leq n; 0 \leq l_i \leq n; 0 \leq c_i \leq c$). If c_i is nonzero, it means person a_i sent a memo of importance level l_i saying the new tie color is c_i. If c_i is zero, then l_i is also zero, and it means you have to find the tie color worn by person a_i.

In the **easy subproblem G1**, $1 \leq n, c, q \leq 10\,000$.

In the **hard subproblem G2**, $1 \leq n, c, q \leq 1\,000\,000$. Because the input file size for subproblem G2 is about 100 MB, you cannot download it directly. Instead, we have provided a small Python 2 program that will generate the file `g2.in` when executed.

Output specification

For each test case: Let y_i be equal to 0 if c_i is nonzero, or the tie color you found in event i if c_i is zero. Then, let z_i be $i \cdot y_i$. Output one line with a single number: the sum of $z_1 + \ldots + z_q$ modulo $10^9 + 7$.

Example

input

```
1

4 3 7
1 2 2
3 0 0
2 1 3
3 0 0
1 0 2
2 0 0
4 1 1
4 0 0
```

output

```
32
```

At the beginning, everyone has tie color 1 (including employee 3). Then, employees 2, 3 and 4 change to color 3. Employee 1 then changes to color 2, but because his memo had importance level 0, everyone else stays unchanged. Finally, employee 4 changes his color back to 1. His direct reports would change too, but he doesn't have any. Together, we get $1 \cdot 1 + 2 \cdot 0 + 3 \cdot 3 + 4 \cdot 0 + 5 \cdot 3 + 6 \cdot 0 + 7 \cdot 1 = 32$.

Problem H: Humble Captains

Every day just after school n children rush out of their classrooms onto the field to play football. They choose two captains among themselves who then divide the remaining children into two teams. The teams do not need to be of the same size – in the extreme case, an overconfident captain may challenge all the other children to join their forces against him! Adam and Betka are the captains for today's game.

Problem specification

There are m pairs of children who are friends. Two friends playing for the same team are more likely to pass the ball to each other than two non-friends, so they increase the strength of their team. The total strength of a team can therefore be defined as the number of pairs of friends within that team.

Adam thinks a football match is fun when the players score many goals. He would like to **maximize the sum** of the two teams' strengths. On the other hand, Betka believes the most enjoyable matches are balanced ones, so she wants to **minimize the difference** between the teams' strengths.

Find the largest possible sum of teams' strengths and the smallest possible absolute difference between teams' strengths. (These are two independent problems – there may not necessarily be a team division that satisfies both criteria.)

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line.

The first line of each test case consists of integers n ($n \geq 2$) and m ($0 \leq m \leq n \cdot (n-1)/2$). Children are labelled $1, \ldots, n$. Adam has number 1 and Betka number 2. Each of the following m lines contains two integers u_i, v_i ($1 \leq u_i < v_i \leq n$) meaning that u_i and v_i are friends. Each pair of friends is only listed once.

In the **easy subproblem H1** we have $n \leq 24$.

In the **hard subproblem H2** we have $n \leq 200$.

Output specification

For each test case, output a single line with two integers: the largest possible sum of teams' strengths, and the smallest possible difference between the teams' strengths.

Example

input

```
2

3 3
1 2
2 3
1 3

3 1
1 3
```

output

```
1 1
1 0
```

In the first example, it does not matter whether Cyril (child 3) will play with Adam or with Betka. Either way, one of the teams will have strength 1 and the other 0, so the sum of strengths is 1 and their difference is 0. In the second example, letting Cyril play with Adam increases the sum of strengths strength, but letting him play with Betka keeps the teams more balanced.

Problem I: Inexpensive Travel

Cycling is a cheap and healthy way of getting from A to B, plus it is a lot of fun. That is why Peter invited Kamila to join him on an ambitious bike trip around the world. Unfortunately, cycling can sometimes be hard – just imagine having to ascend 1500 meters on some winding road to a mountain pass in the Alps. Different cyclists have a different degree of preference for the ascents. On one side of the spectrum there is Kamila who would like to avoid going uphill completely. On the other side you will find Peter who thinks that flat roads are painfully boring. Now Peter and Kamila have a problem – they need to figure out which route to choose so that they would both enjoy the trip. Help them by finding all the options they have.

Problem specification

There are n towns in the world. The towns are numbered 1 through n. Peter and Kamila want to start the trip in town 1 and end it in town n. Each road is a directed edge from one town to another. Each road has two parameters: the distance $d \geq 0$ between its endpoints, and the ascent $a \geq 0$. At least one of them is nonzero.

Each cyclist can be described by a single real number $p \in [0, 1]$: their preference for ascents. The value p determines the actual length of each edge for this particular cyclist: an edge with distance d and ascent a will have the length $pd + (1 - p)a$. For example, Kamila has $p = 0$ and only cares about ascents. Peter has $p = 1$: he ignores all ascents and only cares about the total distance.

Different values of p will obviously produce different shortest paths from 1 to n. Your task is to find all values p where the set of shortest paths changes.

Formally, let $S(p)$ be the set of all paths from 1 to n that have the shortest total length for a cyclist with preference p. Find all values p such that $0 < p < 1$ and the sets $S(p - \varepsilon)$ and $S(p + \varepsilon)$ differ for any $\varepsilon > 0$. (The two sets differ if there is some specific path from 1 to n that is among the shortest paths in one setting but not in the other one. Note that $p = 0$ and $p = 1$ are by definition never valid answers.)

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a line containing two integers n and m: the number of towns and the number of roads. Next, m lines follow. The i-th of these lines will contain four integers: x_i, y_i, d_i, and a_i ($1 \leq x_i \leq n$; $1 \leq y_i \leq n$; $0 \leq d_i$; $0 \leq a_i$; $0 < d_i + a_i$). These represent a directed road from the town x_i to the town y_i, with distance d_i and ascent a_i. Note that some of the roads may be self-loops (with $x_i = y_i$) and that each pair of towns can be connected by multiple roads in each direction.

In the **easy subproblem I1** you may assume that $n \leq 100$, $m \leq 5000$, and $0 \leq d_i, a_i \leq 10\,000$.

In the **hard subproblem I2** you may assume that $n \leq 5000$, $m \leq 500\,000$, and $0 \leq d_i, a_i \leq 500\,000$.

Because the input file size for subproblem I2 is about 100 MB, you cannot download it directly. Instead, we have provided a small Python 2 program that will generate the file `i2.in` when executed.

Note that the implementation of this generator might matter.

Output specification

For each test case, find the number v of valid answers and their values $p_1 < p_2 < \cdots < p_v$. Output a single line containing the integer v followed by the real numbers p_1 through p_v, in order. Output each p_i to at least 10 decimal places. Any answer that differs from the correct solution by at most 10^{-9} will be accepted.

Note that in some test cases the set of shortest paths never changes. In such case we have $v = 0$ and thus the output line will contain just a single zero. Notably, this includes all test cases in which it is impossible to get from 1 to n. (In those test cases, each set $S(p)$ is empty.)

Example

input

```
3

2 2
1 2 1 1
1 2 3 0

2 2
1 2 1 0
1 2 1 0

2 0
```

output

```
1 0.333333333333
0
0
```

In the first test case there are two roads from 1 to 2. The first road (distance 1, ascent 1) goes across a small hill, the other (distance 3, ascent 0) takes a detour around the hill. The preference between these roads changes at $p = 1/3$: any cyclist with $p > 1/3$ will prefer the first road while any cyclist with $p < 1/3$ will prefer the second one.

In the second test case there are two different roads, but they look the same to any cyclist. Each of them is a shortest path for any value of p.

Finally, in the third case it is impossible to reach the destination.

Problem J: Juicy Dot Coms

As you shall soon discover, the dot coms from the title of this task have nothing to do with the dot com boom. These are some dot coms from the previous generation.

You are given the files `j1.com` and `j2.com`. They know the passwords. The easy one might just tell you the password, but it may take some convincing. The hard one will be, of course, harder. It seems... who knows, broken? Some repairs may be in order.

Oh, and there's an obvious trap in the easy one. Don't fall into the trap. You'll know the right password once you get it.

Problem specification

The password is a sequence of upper- and lowercase English letters. Recover the password.

Input specification

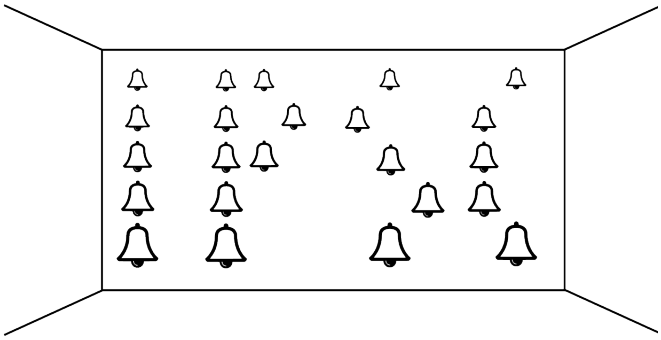
The input is the corresponding dot com file.

Output specification

For each test case, your output should contain a single line with the password.

Problem K: Klingelt das Glockenspiel

You have just bought an amazing new Carillon: a musical instrument that consists of bells of various sizes. You arranged all the bells into an interesting pattern and you mounted them onto your living room wall. While doing so, you followed two simple rules. First, the bells could only be mounted at regularly spaced grid points. Second, the smaller the bell, the higher you mounted it. For example, it could have looked like this:



You then sat into your armchair facing the wall with all the bells. You relaxed, closed your eyes, and enjoyed the music.

Problem specification

In each subproblem, you are given a different scenario that is consistent with the above description. You are given a recording of the bells. Listen to the performance and identify a short English word you should submit as your answer.

In the easy subproblem you will quickly discover that the music follows a nice regular pattern. The hard subproblem, on the other hand, is just pure chaos.

Input specification

The input file is a stereo MP3 file of the recording.

Output specification

Output a single line with the English word determined by the recording. The word should be written in UPPERCASE.

Problem L: Lunchtime!

A foreign restaurant recently opened in your area. You like the food they make, so you convinced your friends to start having lunch there each day. But all the dishes have foreign names and you don't know which is which. You can only identify a dish after you order it.

There are p people in your group (including yourself) and the restaurant serves d different dishes. Ordering at the restaurant works as follows: First, each person orders exactly one dish by specifying a number between 1 and d. Then, the p dishes ordered by the group are brought to the table all at once, in no particular order and with no information on which is which.

Your goal is to create a menu in your language: you want to map all numbers between 1 and d to names of dishes in your language. Assuming that your friends are willing to cooperate, what is the smallest number of lunches in which this can be done?

Problem specification

Find the smallest number of lunches ℓ such that there is a strategy for placing the orders in such a way that after ℓ lunches you will know the names of all dishes. In the **hard subproblem L2**, you also have to find the number of valid first day orders.

An order is a multiset of dish numbers your group ordered. For example, if $p = 3$ and $d = 2$, the order $\{1, 1, 2\}$ is the same as the order $\{2, 1, 1\}$ but different from the order $\{1, 2, 2\}$. A valid first day order is an order such that if your group makes the order on the first day, there will be a strategy that solves our task by making $\ell - 1$ additional orders.

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line. Each test case consists of a single line containing the numbers p and d.

In the **easy subproblem L1** we have $t = 81$, $1 \leq p \leq 9$, and $1 \leq d \leq 9$.

In the **hard subproblem L2** we have $t = 320$, $1 \leq p \leq 16$, and $1 \leq d \leq 20$.

Output specification

Output a single line for each test case.

In the **easy subproblem L1** the line should contain a single integer: the minimum number of days.

In the **hard subproblem L2** the line should contain two integers: the minimum number of days, and the number of valid first day orders.

Example (hard subproblem)

input

```
3

1 1

2 3

6 3
```

output

```
1 1
2 3
1 6
```

In the first example, note that the answer is not 0: you have to order the dish once to see what it is.

Here's one optimal strategy for $p = 2$ and $d = 3$: On the first day, one of you will order dish 1 and the other will order dish 2. On the second day, order dishes 1 and 3.

In the third example order one dish 1, two dishes 2, and three dishes 3.

Problem M: Make*me-an+[integer!]

Esoteric languages are a popular subject of IPSC problems. The home page lists many cases where we gave you an unusual language with strange syntax and even stranger semantics and tasked you with doing something useful in it.

But then we realized: why bother with esoteric languages? Why don't we just use a standard language everyone already knows? After all, that won't make it any easier for you!

Problem specification

In this problem, you will use JavaScript (as standardized by ECMA-262). All you have to do is produce the integers from 0 to 1000. There's just one catch: your programs can only use the characters `![]+-*`.

Input specification

There is no input.

Output specification

The output should contain exactly 1001 lines. For each i between 0 and 1000, line $i+1$ of your output should contain a valid JavaScript expression consisting only of the characters `![]+-*` that evaluates to a number (i.e., `typeof(result) == "number"`) with value i. Note that the expression must not contain any whitespace.

Additionally, your expressions must be short enough. For the **easy subproblem M1**, each JavaScript expression should be no longer than 200 characters. For the **hard subproblem M2**, no expression should exceed 75 characters.

Example output

```
+![]
+!![]
... (999 more lines)
```

Hints

Testing your solution: open your browser's developer console, or install node.js.

Pro tip: JavaScript seems like an easy language to learn. And it is, except for all its quirks and weird defaults. Unless you can score at least 21 out of 20 on quizzes such as this one, you shouldn't just assume some expression will throw a parse error or runtime error. Even strange expressions can be valid.

