# XXXI Maraton Nacional de Programacion
# Colombia - ACIS / REDIS 2017
# ACM ICPC

## Problems

(This set contains 11 problems; problem pages numbered from 1 to 20)

## General Information

Unless otherwise stated, the following conditions hold for all problems.

### Program name

1. Your source file (your solution!) must be called `<codename>.c`, `<codename>.cpp`, `<codename>.java` or `<codename>.py`, as indicated below the problem title.

### Input

1. The input must be read from standard input.
2. The input contains several test cases. Each test case is described using a number of lines that depends on the problem.
3. When a line of data contains several values, they are separated by single spaces. No other spaces appear in the input. There are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. The end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

### Output

1. The output must be written to standard output.
2. The result of each test case must appear in the output using a number of lines that depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest upper value.

# A: A Contest to Meet

*Source file name:* `acm.c`, `acm.cpp`, `acm.java`, *or* `acm.py`
*Author:* Rodrigo Cardoso

*A Contest to Meet* (ACM) is a reality TV contest that sets three contestants at three random city intersections. In order to win, the three contestants need all to meet at any intersection of the city as fast as possible. The contestants are given communication devices to help them in sharing information about the city (e.g., where the sun is, how far the mountains are, etc.).

It should be clear that the contestants may arrive at the intersections at different times, in which case, the first to arrive can wait until the others arrive. Moreover, it is guaranteed that there is a path between any pair of intersections.

From an estimated walking speed for each one of the three contestants, ACM wants to determine the minimum time that a live TV broadcast should last to cover their journey regardless of the contestants' initial positions and the intersection they finally meet. You are hired to help ACM answer this question.

You may assume the following:

- Each contestant walks at a given estimated speed.
- The city is a collection of intersections in which some pairs are connected by bidirectional streets that the contestants can use to traverse the city.

## Input

The input consists of several test cases.

The first line of each test case contains two blank-separated integer values $N$ and $S$, indicating the number of intersections and the number of streets in the city, respectively ($1 \leq N \leq 100$, $0 \leq S \leq 9000$). Then follow $S$ lines, each one with three blank-separated integer values $i$, $j$, and $d$ ($0 \leq i < N$, $0 \leq j < N$, $i \neq j$, $1 \leq d \leq 200$), meaning that there is a street of length $d$ meters connecting the $i$-th and the $j$-th intersections. Note that intersections are indexed from 0 to $N - 1$. You can assume that there is at most one street connecting any pair of intersections and that the input lists a street exactly once.

At the end of each test case there is one line with three blank-separated integer values $sA$, $sB$, $sC$ ($50 \leq sA, sB, sC \leq 100$), representing the walking speed of each of the three contestants, in meters per minute.

*The input must be read from standard input.*

## Output

For each test case, print a single line with an integer indicating the minimum number of minutes that will pass before the three contestants can meet, if they start to walk immediately after the show starts. Remember that the contestants can start at any random (unknown) intersection and can decide to meet at any intersection: you need to account for the worst case scenario.

The answer should be given rounding decimals to the next integer (e.g., 2.9 minutes rounds up to 3 minutes and 3.2 minutes rounds up to 4 minutes).

*The output must be written to standard output.*

**Sample Input**

```
2 1
0 1 150
60 50 75
3 2
1 0 100
1 2 80
60 80 50
4 5
0 1 200
0 2 200
0 3 200
2 1 200
2 3 200
50 100 100
```

**Sample Output**

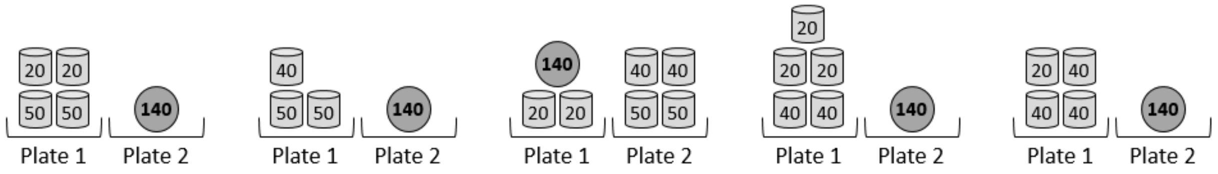
```
3
4
8
```

## B: **Balance Game**

*Source file name:* `balance.c, balance.cpp, balance.java`, *or* `balance.py`
*Author:* Rodrigo Cardoso

Consider a puzzle with three types of tokens. Tokens of the same type have the same weight and any two tokens of different type have different weight. There are $M$ copies of each token type. There is a two-plate scale where tokens can be placed. There is also a given object with known weight $W$ that can be put on the plates of the scale. The goal of the puzzle is to balance the scale with some of the tokens to check the weight of the object. If used, tokens of the same type must be allocated on the same plate.

For example, suppose that there are three token types with weights 20, 50, and 40 grams, respectively. Also, there are $M = 3$ copies of each token type. If the given object weights $W = 140$ grams, then its weight can be checked in five different ways (neglecting the order of the plates):



Five different ways to achieve the goal

However, it is impossible to measure the weight $W = 105$ with this set of tokens.

Your job is to write a computer program to determine the number of different ways to check the weight of the object.

### Input

The input consists of several test cases. The first line of each test case contains two blank-separated integers $M$ and $W$ ($1 \leq M \leq 5\,000$ and $1 \leq W \leq 5\,000\,000$) indicating, respectively, the number of copies of each token type and the weight of the object to be checked. Then there is a line containing three distinct blank-separated integers $n_1$, $n_2$, and $n_3$ representing the weigths of each token type ($1 \leq n_i \leq 1\,000$, for each $1 \leq i \leq 3$, $n_1 \neq n_2$, $n_1 \neq n_3$, $n_2 \neq n_3$).

*The input must be read from standard input.*

### Output

For each test case, print a single line indicating the number of different ways the given weight can be checked with the set of tokens.

*The output must be written to standard output.*

| **Sample Input** | **Sample Output** |
|---|---|
| `3 140`<br>`20 50 40`<br>`3 105`<br>`20 50 40` | `5`<br>`0` |

## C: **Compact Terms**

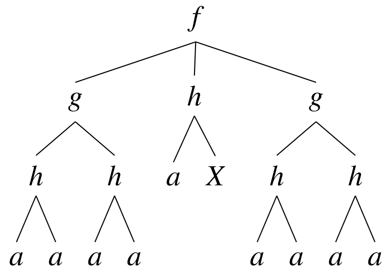
*Source file name:* `compact.c`, `compact.cpp`, `compact.java`, *or* `compact.py`
*Author:* Camilo Rocha

A term is a tree-like structure used in mathematics, computer science, and philosophy to identify entities in a domain of discourse. What is remarkable about terms is that they offer a direct encoding of such entities that a computer can process.

For a particular domain of discourse, the *set of terms* is parametric on a *set of variables* $\mathcal{V}$ and a *set of function symbols* $\mathcal{F}$; it is inductively defined by the following rules:

- Any variable is a term.
- Any function symbol $f$ with $\text{ar}(f) = 0$ is a term.
- Any expression $f(t_1, \ldots, t_n)$, with $\text{ar}(f) = n \geq 1$ and each argument $t_i$ a term $(1 \leq i \leq n)$, is a term.

Here ar is a function from the set of function symbols $\mathcal{F}$ to the natural numbers. For each $f \in \mathcal{F}$, the expression $\text{ar}(f)$ denotes the number of arguments of $f$. For example, for $\mathcal{F} = \{a, f, g, h\}$ with $\text{ar}(a) = 0$, $\text{ar}(g) = \text{ar}(h) = 2$, $\text{ar}(f) = 3$, and $X$ a variable, let $t$ be the term $f(g(h(a, a), h(a, a)), h(a, X), g(h(a, a), h(a, a)))$. The term $t$ can be graphically represented as a labeled tree with 18 vertices as follows:



A string representation of a term, in general, can be unnecessarily lengthy and thus inefficient to compute with. An important observation to cope with this limitation is that subterms can be shared and then a tree can be represented by a *directed acyclic multigraph* (or *dam*). In a dam, vertices are labeled with function symbols from $\mathcal{F}$ and variables from $\mathcal{V}$, and edges are labeled with the argument position of the corresponding subterm in the term structure. Note that there can be more than one edge between any pair of vertices of a dam.

The following dams are representations of the term $t$ abovementioned. In these representations, for example, the topmost function symbol $f$ is applied over three terms of which the first and third are shared: this is indicated by labels 1 and 3 in the edges with source $f$.

$$\begin{tikzcd}
& f \arrow[dl, bend right, "1"'] \arrow[dl, bend left, "3"] \arrow[dr, "2"] & \\
g \arrow[d, bend right, "1"'] \arrow[d, bend left, "2"] & & h \arrow[dl, "1"'] \arrow[d, "2"] \\
h \arrow[r, "2"] \arrow[d, "1"'] & a & X \\
a & &
\end{tikzcd}
\qquad
\begin{tikzcd}
& f \arrow[dl, bend right, "1"'] \arrow[dl, bend left, "3"] \arrow[dr, "2"] & \\
g \arrow[d, bend right, "1"'] \arrow[d, bend left, "2"] & & h \arrow[dl, "1"'] \arrow[d, "2"] \\
h \arrow[r, bend left, "2"] \arrow[r, bend right, "1"'] & a & X
\end{tikzcd}$$

Although these dams represent the same term, they differ in the number of vertices: the one on the left has 7 vertices and the one on the right has 6 vertices. There can be many other dams representing the term $t$, but any of them requires at least 6 vertices.

In this problem, the interest is in computing the size (with respect to the number of vertices) of a dam representing a term which shares the most number of subterms. More precisely, the goal is to compute the minimum number of vertices required to represent a term as a dam.

## Input

The input consists of several test cases, each being a line containing a string representation of a term $s$ ($1 \leq |s| \leq 10^5$). A variable in $\mathcal{V}$ is represented in $s$ by a string $v$ ($1 \leq |v| \leq 20$) made of lowercase and uppercase letters of the English alphabet starting with an uppercase letter. Similarly, a function symbol in $\mathcal{F}$ is represented in $s$ by a string $f$ ($1 \leq |f| \leq 20$) made of lowercase and uppercase letters of the English alphabet starting with a lowercase letter. You can assume that $s$ contains at most 2 000 variables and function symbols (not necessarily different).

*The input must be read from standard input.*

## Output

For each test case output the minimum number of vertices required to represent $s$ as a dam.

*The output must be written to standard output.*

| Sample Input | Sample Output |
|---|---|
| `f(g(h(a,a),h(a,a)),h(a,X),g(h(a,a),h(a,a)))` | `6` |
| `h(a,b)` | `3` |
| `h(a,a)` | `2` |
| `h(A,a)` | `3` |
| `CCPL` | `1` |
| `f(a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a)` | `2` |
| `f(g(a,X,a),g(a,a,X))` | `5` |
| `f(g(a,X,a),g(a,X,a))` | `4` |

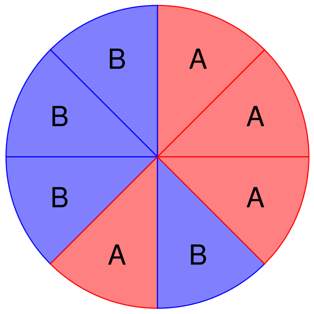
# D: **Rotating Drum**

*Source file name:* `drum.c, drum.cpp`, `drum.java`, *or* `drum.py`
*Author:* Rafael García

A rock 'n' roll band has $k$ musicians, any of them can play any of $n$ instruments, and they can be located in any order on the stage. This band has decided to make a drawing on the bass drum in order to characterize the way they perform on stage. The idea is to divide the surface of the bass drum into $m$ equal sections (like a large pizza) and then assign one of $k$ colors to each of the sections in a way that any possible sequence on $n$ colors is found exactly once clockwise on the drum.

Nick De Bruijn -- a musician in the band -- is a mathematician and he knows that every possible sequence of $n$ colors must be present on the bass drum. He knows that for $k \geq 2$ the value of $m$ must be equal to $k^n$ and for $k = 1$ the value of $m$ must be equal to $n$.

As an example, consider the following bass drum drawing satisfying the abovementioned constraints for $k = 2$ and $n = 3$:



In this case, each one of the 8 sequences appears exactly once clockwise in the drawing. Namely, the sequences *AAA*, *AAB*, *ABA*, *BAB*, *ABB*, *BBB*, *BBA*, *BAA*.

Your task is to help the band to find the sequence of colors that should be drawn on the bass drum for given $k$ and $n$.

## Input

The input consists of several test cases. Each test case is described by a line containing two blank-separated integers $k$ and $n$: the number of colors ($1 \leq k \leq 26$) and the length of the subsequences ($1 \leq n \leq 10$). You may assume that $1 \leq m \leq 10^5$.

*The input must be read from standard input.*

## Output

For each test case print a single line with the solution sequence. The $k$ colors shall be represented by the first $k$ uppercase letters of the English alphabet. If there is more than one solution, you must print the first sequence in lexicographical order.

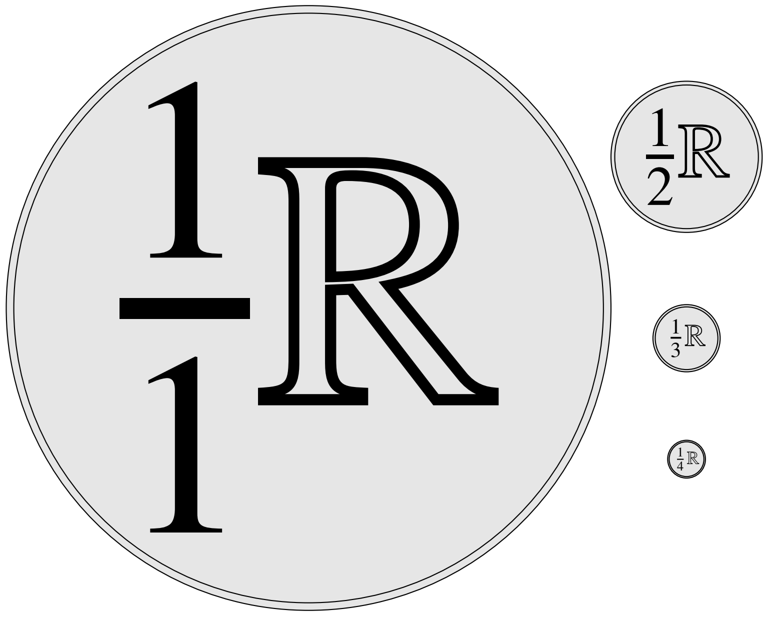
*The output must be written to standard output.*

| **Sample Input** | **Sample Output** |
|---|---|
| 4 2 | AABACADBBCBDCCDD |
| 2 3 | AAABABBB |
| 1 5 | AAAAA |

## E: **Rational Coins**

*Source file name:* `coins.c, coins.cpp`, `coins.java`, *or* `coins.py`
*Author:* Rafael García

Ratioland is a beautiful and highly rational country where the official currency is the *Ration* and the currency symbol is $\mathbb{R}$. Coins are available in denominations of $\frac{1}{q}\,\mathbb{R}$, for each natural number $q = 1, 2, 3, 4, \ldots$. Some years ago, the Royal Bank of Ratioland promoted a law to define the properties of all coins: each coin of denomination $\frac{1}{q}\,\mathbb{R}$ had to be a cylinder made of gold with a thickness of 0.1 inches, radius of $\frac{1}{2\cdot q^2}$ inches, and labeled with the fraction representing the magnificent Rational Number $\frac{1}{q}$.



Coins of denomination $\frac{1}{q}\,\mathbb{R}$, for values of $q = 1, 2, 3, 4$.

In order to celebrate the beauty of its currency, the Royal Bank of Ratioland wants to display some coins in its museum. For this purpose, it has designed a frame with a width of 2 inches, a height of 1 inch, and a thickness of 0.1 inches. In this frame, two coins of denomination $\frac{1}{1}\,\mathbb{R}$ fit exactly. Moreover, many other coins can be placed in the frame. Raphaello, a renowned and rational artist born in Ratioland, was invited by the bank manager to design the layout to place the coins in the frame. After a week of hard work, Raphaello invented a rational procedure to place the coins inside the frame without any overlapping:

- Embed the frame in a Cartesian coordinate system, locating its corners at $\left(-\frac{1}{2}, 0\right)$, $\left(\frac{3}{2}, 0\right)$, $\left(\frac{3}{2}, 1\right)$, $\left(-\frac{1}{2}, 1\right)$.
- Since it is possible to place only two coins of denomination $\frac{1}{1}\,\mathbb{R}$, these coins are to be placed with their centers at $\left(0, \frac{1}{2}\right)$ and $\left(1, \frac{1}{2}\right)$, respectively.
- For each natural number $q > 1$, and each natural value $p$ such that $0 < \frac{p}{q} < 1$ and $\frac{p}{q}$ is an irreducible fraction, a coin of denomination $\frac{1}{q}\,\mathbb{R}$ is to be placed with center at $\left(\frac{p}{q}, \frac{1}{2\cdot q^2}\right)$.

All coins are placed tangent to the horizontal axis of the Cartesian coordinate system. In order avoid confusion, Raphaello decided to name the coin of denomination $\frac{1}{q}\,\mathbb{R}$ with center at $\left(\frac{p}{q}, \frac{1}{2\cdot q^2}\right)$ as $M(p/q)$. For example:

- $M(0/1)$ is the coin of denomination $\frac{1}{1}\,\mathbb{R}$ with center at $\left(0, \frac{1}{2}\right)$;

- $M(1/1)$ is the coin of denomination $\frac{1}{1}\,\mathbb{R}$ with center at $\left(1, \frac{1}{2}\right)$; and
- $M(1/2)$ is the coin of denomination $\frac{1}{2}\,\mathbb{R}$ with center at $\left(\frac{1}{2}, \frac{1}{8}\right)$.

Raphaello wants to know the *first n* coins that touch the coin $M(p/q)$ inside the frame when sorting all these coins by decreasing order of radius (i.e., by increasing order of $q$). If two coins share the same radius, he wants them sorted by increasing order of $x$ coordinates (i.e., by increasing order of $p$).

## Input

The input consists of several test cases. Each test case is described by a line containing three blank-separated integers $p$, $q$, and $n$ ($0 \leq \frac{p}{q} \leq 1$, $1 \leq q \leq 10^6$, $1 \leq n \leq 10^3$), with $\frac{p}{q}$ an irreducible fraction.

*The input must be read from standard input.*

## Output

For each test case, print a single line with the $n$ blank-separated names of the first $n$ coins that touch $M(p/q)$ in the frame. The list of coins must be printed in the order previously described.

*The output must be written to standard output.*

| Sample Input | Sample Output |
|---|---|
| 2 5 4 | M(1/2) M(1/3) M(3/7) M(3/8) |
| 2 3 3 | M(1/1) M(1/2) M(3/4) |
| 0 1 2 | M(1/1) M(1/2) |
| 1 1 2 | M(0/1) M(1/2) |
| 1 2 5 | M(0/1) M(1/1) M(1/3) M(2/3) M(2/5) |

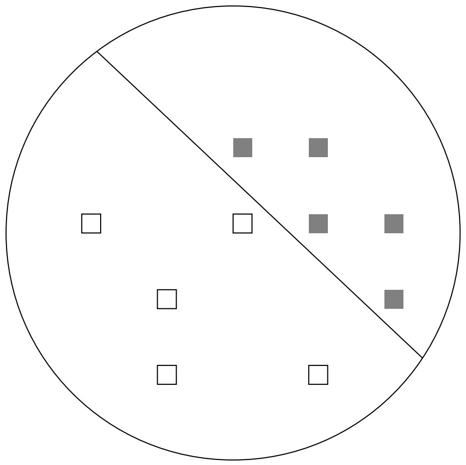
# F: **Fish**

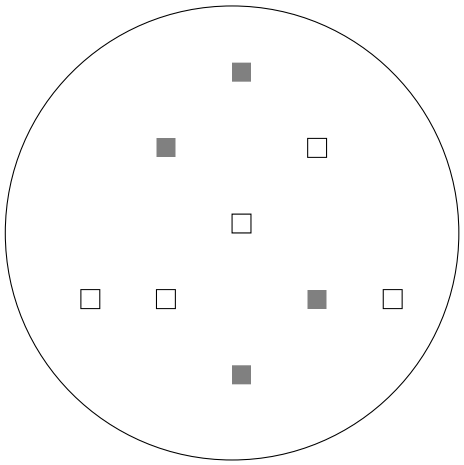
*Source file name:* `fish.c, fish.cpp`, `fish.java`, *or* `fish.py`
*Author:* Camilo Corena

Advanced Circular Aquariums (ACM) is a state-of-the-art manufacturer of aquariums that can detect when two types of fish can be separated into two different zones for feeding. This solves the problem of having fish with different dietary needs in the same aquarium.

The technology offered by ACM is based on a powerful real-time computer vision system placed on top of the aquariums. For each fish in the aquarium, the computer vision system determines the $(x, y)$ coordinate and type $t$, which is either `A` or `B`. The goal of the entire system is to detect when the two types of fish can be completely separated by a flat glass panel running from the edges of the aquarium and from surface to bottom (without touching any fish).

The figures below depict two scenarios output by the computer vision system. In the one on the left, the two types of fish can be separated by a single flat glass panel. However, in the scenario on the right, the two types of fish cannot be completely separated by a single flat glass panel.


Fish types can be separated


Fish types cannot be separated

You have been hired to process the output of the computer vision system. Your task is to write a computer program to determine, for a given state of the aquarium, whether the two types of fish can be separated with a single flat glass panel.

## Input

The input consists of several test cases. Each test case starts with a line containing two blank-separated integers $n$ and $r$, where $n$ is the total number of fish $(2 \le n \le 500)$ and $r$ is the radius of the aquarium $(0 < r \le 10^4)$. Then $n$ lines follow, each one containing three blank-separated values $x_i$, $y_i$, and $t_i$, where $(x_i, y_i)$ are the integer coordinates $({x_i}^2 + {y_i}^2 \le r)$ and $t_i$ is the type (`A` or `B`) of the $i$-th fish. A line with two zeros ‘‘`0 0`’’ indicates the end of the input.

You may suppose that: the aquarium's center is located at $(0, 0)$, no two fish share the same coordinates, the size and form of the fish are negligible, and there is at least one fish of each type.

*The input must be read from standard input.*

## Output

For each test case, print a single line with the text ‘`FEED`’ if the fish can be separated with a single glass panel without touching any fish or the text ‘`NOT YET`’ if this is not possible.

*The output must be written to standard output.*

**Sample Input**

```
10 3
0 0 A
-2 0 A
-1 -1 A
-1 -2 A
1 -2 A
1 1 B
0 1 B
1 0 B
2 0 B
2 -1 B
9 3
0 0 A
-1 -1 A
-2 -1 A
2 -1 A
1 1 A
-1 1 B
1 -1 B
0 2 B
0 -2 B
0 0
```

**Sample Output**

```
FEED
NOT YET
```

# G: **FujikoMine**

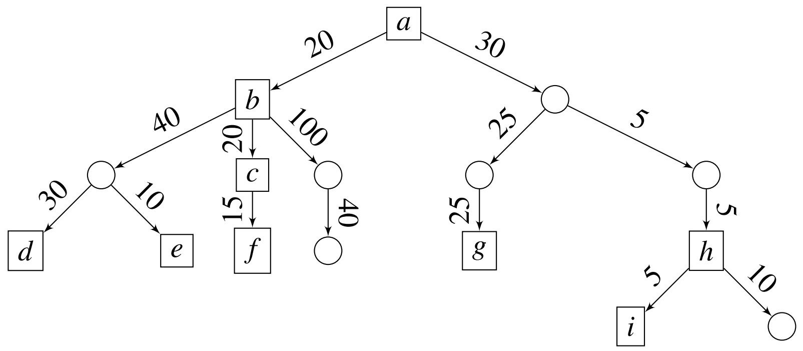
*Source file name:* `fujiko.c, fujiko.cpp`, `fujiko.java`, *or* `fujiko.py`
*Author:* Camilo Rocha

Lupin is back in business. Well, being the world's greatest thief, he is back in the business of making 'easy' money. He is grateful because last year, thanks to you and your folks in the Colombian programming contest, the bank robbery went smooth as silk. Money has ran out and this year he is planning a massive cryptocurrency heist using a tailored-designed ransomware known as *FujikoMine*. His right-hand man and closest ally Daisuke Jigen has identified the amount and distribution of cryptocurrency in the web.

The web setup is a tree with two different types of nodes: *(re-)transmission* and *bridge* nodes. A link between two nodes $u$ and $v$ is annotated with the (potential) number of cryptocurrency that can be stolen if both nodes $u$ and $v$ are infected with *FujikoMine*. There are some important observations about the ransomware:

- It can infect both transmission and bridge nodes, and attacks as many nodes as possible. However, since transmission nodes -- unlike bridge nodes -- are usually under heavy scrutiny by sysadmins, the plan is to limit the amount of transmission nodes to infect while maximizing the potential of the attack.
- Attacks are only effective on paths starting and ending with infected transmission nodes, where all intermediate nodes in the path are also infected.
- If $u$ and $v$ are infected transmission nodes and there is a path from $u$ to $v$, then the number of cryptocurrency that can be stolen by an attack from $u$ to $v$ is the sum of cryptocurrency that can be stolen in the path from $u$ to $v$, if in the path all nodes are infected with *FujikoMine* (otherwise, the sum is 0).
- The paths induced by the infected transmission nodes must form a subtree of the web setup.
- Jigen-san has assured Lupin that each node in the web setup has at most three children.

As an example, consider the depicted web setup with 9 transmission nodes (labeled from $a$ to $i$) and 7 bridge nodes (whose labels have been omitted).



- If the plan is to infect two transmission nodes and any number of bridge nodes, the best option is to infect $a$ and $g$ because $30+25+25 = 80$ cryptocurrency can be stolen. Infecting $a, d$ or $g, h$ yields 0 cryptocurrency. In the first case, the path from $a$ to $d$ includes the uninfected transmission node $b$ (otherwise, there would be three transmission nodes infected). In the second case, there is no path from $g$ to $h$ or from $h$ to $g$.
- If the plan is to infect three nodes, the greatest potential is 100 and it can be achieved by infecting, e.g., $a, b, g$. Infecting nodes $a, b, d$, nodes $b, c, d$, or nodes $a, g, h$, only has potential 90.

Given a distribution of cryptocurrency in the web and the number of transmission nodes to infect, your task is to aid Lupin in using *FujikoMine* by computing the maximum number of cryptocurrency that can be stolen.

## Input

The input consists of several test cases. In the first line of a test case there are three blank-separated integers $n, m, q$ ($2 \leq n \leq 500$, $1 \leq m \leq n$, and $1 \leq q \leq 100$), where $n$ is the number of nodes of the web setup, $m$ is the number of transmission nodes, and $q$ is the number of queries. The nodes of the web setup are identified with numbers from 0 to $n-1$. Each of the next $n-1$ lines contains three blank-separated integers $u, v, w$ ($0 \leq u, v < n$ and $0 \leq w \leq 500$) indicating that $u$ is a parent of $v$ and that there are $w$ cryptocurrency that can be stolen by attacking $u$ and $v$. Next comes a line with a sequence of $m$ blank-separated pairwise-distinct node labels $t_i$ ($0 \leq t_i < n$ for each $1 \leq i \leq m$) identifying the transmission nodes in the web setup. Finally, comes a line with a sequence of $q$ blank-separated integers $x_j$ ($0 \leq x_j \leq n$ for each $1 \leq j \leq q$) indicating the number of transmission nodes to infect with *FujikoMine*.

You can assume that the given setup corresponds to a valid web setup, i.e., that the given graph is a tree with each node having at most three children.

*The input must be read from standard input.*

## Output

For each web setup and each query $x_j$, print a single line with the maximum number of cryptocurrency that can be stolen by infecting exactly $x_j$ transmission nodes and any amount of bridge nodes.

*The output must be written to standard output.*

**Sample Input**

```
16 9 5
0 1 20
0 9 30
1 10 40
1 2 20
1 11 100
9 12 25
9 13 5
10 3 30
10 4 10
2 5 15
11 14 40
12 6 25
13 7 5
7 8 5
7 15 10
0 1 2 3 4 5 6 7 8
1 2 3 4 16
2 1 1
1 0 2
1
1
```

**Sample Output**

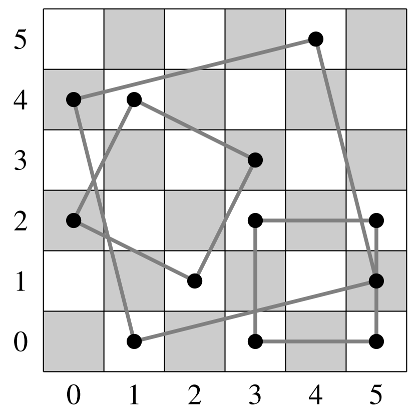
```
0
80
100
170
0
0
```

# H: **Hip-*n***

*Source file name:* `hipn.c`, `hipn.cpp`, `hipn.java`, *or* `hipn.py`
*Author:* Rodrigo Cardoso

Hip-*n* is a game in which two players take turns by placing tokens on the free cells of a non-empty $n \times n$ checkerboard. The game is lost by the first player placing four tokens identifying the vertices of a square: they can be of any size and tipped at any angle. The game ends in a tie when the board is full of tokens and no player has lost.

The following figure depicts a $6 \times 6$ checkerboard and three examples of squares: the first player putting four tokens on the vertices of any of these squares loses the game. Of course, there are many more options for losing a game in the $6 \times 6$ checkerboard.



Your task is to create a program that decides the outcome of a Hip-*n* game described as a sequence of plays, by identifying the player that loses or recognizing a tie.

## Input

The input consists of several test cases. It ends when there are no more cases to test.

The first line of each test case contains an integer $n$ ($1 \leq n \leq 200$) indicating the number of rows and columns of the checkerboard. The next line contains $n^2$ distinct pairs of blank-separated integers $r$ and $c$ in the checkerboard ($0 \leq r < n$ and $0 \leq c < n$): each such a pair identifies the placement of a token at row $r$ and column $c$ by the corresponding player. You can assume that player 1 makes the first move, player 2 the second one, player 1 the third one, and so on.

*The input must be read from standard input.*

## Output

For each test case, print a single line with 0 if the game ends in a tie, 1 if player 1 loses, and 2 if player 2 loses.

*The output must be written to standard output.*

| Sample Input | Sample Output |
|---|---|
| 3<br>1 0 1 1 2 1 0 2 0 1 2 0 0 0 1 2 2 2<br>3<br>1 0 1 1 2 1 0 2 0 1 2 0 1 2 0 0 2 2<br>3<br>1 0 2 2 2 1 0 0 1 2 0 2 1 1 2 0 0 1 | 0<br>1<br>2 |

## I: License Plates

*Source file name:* `plates.c, plates.cpp`, `plates.java`, *or* `plates.py`
*Author:* Edwin Niño

On their daily drive to high school, Antonia and her mom Maria, used to challenge each other using the license plates of other cars. In their country, license plates always contain a string of exactly three letters. The challenge was to tell the longest word that contains as a subsequence the string of letters found in a license plate. For example, if a license plate contained the string `OMI`, then `PROGRAMMING` and `COLOMBIA` could be used as answers to the challenge. However, the former word was preferred because it was longer.

Those were the old days. Antonia is now a Computer Science student and Maria wants to make some changes to keep the challenge interesting to her daughter. In the new version of the challenge, Maria writes down a word $S$ and Antonia must calculate the number of different three-letter strings that are each a subsequence of $S$.

You must write a program to help Maria calculate in advance the output of the new version of the challenge.

### Input

The first line of input contains $T$ ($T \geq 0$) indicating the number of test cases. Each test case consists of one single line containing a string $S$ ($1 \leq |S| \leq 10^5$). You may assume that $S$ is made only of uppercase letters from the English alphabet (i.e., `A`, `B`, ..., `Z`) and that there is no blank line between test cases.

*The input must be read from standard input.*

### Output

For each test case, print a single line containing the number of different three-letter strings that are each a subsequence of $S$.

*The output must be written to standard output.*

| Sample Input | Sample Output |
|---|---|
| 5<br>PROGRAMMING<br>COLOMBIA<br>NUEVE<br>AAAAAAA<br>PQ | 110<br>46<br>9<br>1<br>0 |

# J: Romeo and Juliet Secrets

*Source file name:* `romeo.c`, `romeo.cpp`, `romeo.java`, *or* `romeo.py`
*Author:* Edwin Niño

The Capulets and the Montagues hate each other to death and will never accept the love of Romeo and Juliet. This is why the two young lovers must hide their affair by making phone chats unintelligible to their parents. They do this by altering the messages in their chat using the following three-step process:

1. First, they remove spaces and punctuation marks, and make every letter lowercase.
2. Then, they choose a word $W$ (e.g., their names) they want to hide.
3. Finally, they replace in every occurrence of $W$ a substring of length $K$ with that many random letters.

For example, assume Juliet wants to hide the word *Romeo* in the message

*O Romeo, Romeo! Wherefore art thou Romeo?*

After the first step, the encrypted message will be

*oromeoromeowhereforeartthouromeo*

Note that the word *Romeo* appears at positions 2, 7, and 28. If $K = 2$, then in each one of these occurrences a substring of length two must be replaced with exactly two random letters. One possible encrypted version of the original message under this scheme could be

*oxwmeorozkowhereforeartthouromeo*

Unfortunately, Lord Capulet is aware of Romeo and Juliet secret, and he has intercepted one of Juliet's messages. He needs your help in writing a program to find a given word in the intercepted message, even if that word was altered using the secret scheme. Specifically, Lord Capulet wants to know how many times the given word could appear in the original message.

## Input

The first line of input contains the number of test cases. Each test case consists of three lines:

- A line containing a string $T$: the message intercepted by Lord Capulet ($1 \le |T| \le 10^5$).
- A line containing a string $W$: the word Lord Capulet wants to find in $T$ ($1 \le |W| \le |T|$).
- A line containing integer $K$ ($1 \le K \le |W|$).

Both $T$ and $W$ will exclusively consist of lowercase letters of the English alphabet (`a`, …,`z`).

*The input must be read from standard input.*

## Output

For each test case, print a single line indicating the number of occurrences of word $W$ in message $T$, including all possible encrypted versions of $W$. Occurrences that overlap must be counted separately.

*The output must be written to standard output.*

| **Sample Input** | **Sample Output** |
|---|---|
| <pre>3<br>oxwmeorozkowhereforeartthouromeo<br>romeo<br>2<br>abcabcabcabcabc<br>aba<br>1<br>abcabcabcabcabc<br>acb<br>2</pre> | <pre>3<br>5<br>9</pre> |

## K: **Soccer Championship**

*Source file name:* `soccer.c, soccer.cpp`, `soccer.java`, *or* `soccer.py`
*Author:* Rafael García

A sports journalist has gained access to the results of a soccer league and wants to calculate the final standings. In this league, three points are given to the team winning a match, one point for each team in a draw, and none for a defeated team.

The standing of each team in the league shall be determined as follows:

1. greatest number of points obtained in all matches;
2. greatest goal difference in all matches (i.e., goals scored minus goals against);
3. greatest number of goals scored in all matches; and
4. greatest number of goals scored playing as visitor in all matches.

If two or more teams are equal on the basis of the above criteria, their rankings shall be determined by lexicographic order on the team's name (characters are sorted by ASCII value).

The journalist also wants to determine the number of occurrences of the classic *sports journalist's paradox*, namely, the number of matches in which the team losing the game has a better final standing than the one winning that game.

### Input

The input consists of several test cases. The first line of each test case contains a natural number $M$ indicating the number of matches ($1 \leq M \leq 64$). Each one of the next $M$ lines contains the results of the matches in the format

```
  L X vs. Y V
```

where $X$ is the number of goals scored by the local team with name $L$ ($0 \leq X \leq 32$, $1 \leq |L| \leq 100$) and $Y$ is the number of goals scored by the visitor team with name $V$ ($0 \leq Y \leq 32$, $1 \leq |V| \leq 100$, $V \neq L$).

You can assume that each team name consists of uppercase characters `A-Z`, digits `0-9`, periods (`.`), and can contain blanks. However, blanks do not appear at the beginning or end of a name.

*The input must be read from standard input.*

### Output

For each test case, print a line with the text

```
The paradox occurs X time(s).
```

where $X$ is the number of paradoxes found by the end of the league. This line should be followed by the final standings in the format

`1.` *Name*$_1$
`2.` *Name*$_2$
...

`N.` $Name_N$

where $N$ is the number of teams in the league and such that the $i$-th place in the final standings is occupied by the team with name $Name_i$.

*The output must be written to standard output.*

| **Sample Input** | **Sample Output** |
|---|---|
| <pre>13<br>B. DORTMUND 2 vs. 2 REAL MADRID<br>SP. PORTUGAL 2 vs. 0 LEGIA<br>SP. PORTUGAL 1 vs. 2 B. DORTMUND<br>REAL MADRID 5 vs. 1 LEGIA<br>B. DORTMUND 1 vs. 0 SP. PORTUGAL<br>LEGIA 3 vs. 3 REAL MADRID<br>MONACO 3 vs. 0 CSKA M.<br>SP. PORTUGAL 1 vs. 2 REAL MADRID<br>B. DORTMUND 8 vs. 4 LEGIA<br>REAL MADRID 2 vs. 2 B. DORTMUND<br>LEGIA 1 vs. 0 SP. PORTUGAL<br>MONACO 1 vs. 0 SP. PORTUGAL<br>CSKA M. 1 vs. 0 B. DORTMUND<br>2<br>TEAM 1 4 vs. 2 TEAM 2<br>TEAM 2 2 vs. 0 TEAM 1</pre> | <pre>The paradox occurs 2 time(s).<br>1. B. DORTMUND<br>2. REAL MADRID<br>3. MONACO<br>4. LEGIA<br>5. CSKA M.<br>6. SP. PORTUGAL<br>The paradox occurs 1 time(s).<br>1. TEAM 2<br>2. TEAM 1</pre> |