XXIV Maratón Nacional de Programación
ACIS REDIS 2010
ACM ICPC

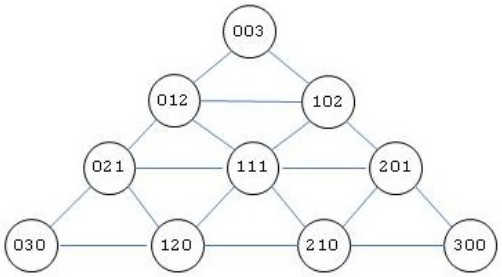
Problemas

(Este conjunto contiene 8 problemas; páginas numeradas de 1 a 17)


Problem A
Y-game

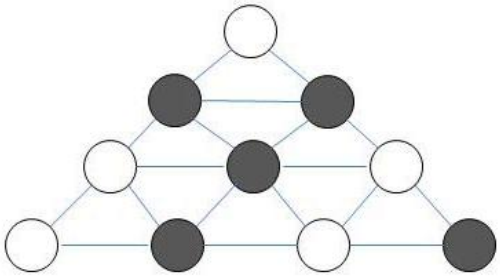
Source file name: `ygame.c`, `ygame.cpp` *or* `ygame.java`

Willy and Benny enjoy very much playing *Y-game*! This is a game in which white and black tokens are placed on a triangular *n-grid*, $n \geq 0$, where n is called the order of the grid. A 3-grid is depicted in the figure below:



In general, an n-grid has $(n+2)(n+1)/2$ points with nonnegative "baricentric coordinates" (x, y, z), where $x+y+z = n$. Coordinates in a n-grid are assigned in such way that along right to left paths x-coodinates are constant, y-coordinates increase by one unit, and z-coordinates decrease by one unit (observe that this construction maintains $x+y+z = n$ true). Symmetric situations may be observed for left to right (where y-coordinates are constant) and horizontal (where z-coordinates are constant) paths. A point (x, y, z) in a n-grid is said to lay on the x *side* (resp., y *side*, z *side*) if and only if $x = 0$ (resp., $y = 0$,$z = 0$).

Willy uses white tokens and Benny uses black ones. Y-game rules are rather complicated, but the end of the game is attained when there is a token placed on every node of the grid. The winner is that player that has formed a Y, that is, his/her tokens are so placed that they include a connected set of points with a point on each side. For example, the following figure represents an end situation where Benny wins:



The winner is rather easy to determine when the grid is small. But Willy and Benny are not interested in that discussion today. Actually, they just want a software solution that computes the winner of ended Y-games. Could you help them?

Input

The problem input consists of several cases. A case begins with a line with two integer numbers, n and m, where n is the order of the grid and m the number of positions that have a black-coloured token (Benny's tokens), with $0 \leq n \leq 20$ and $0 \leq m \leq (n+2)(n+1)/2$.

Then, m lines follow, each one with 3 values x, y and z representing coordinate (x, y, z) of a point in the n-grid with a black token. Values on each input line are separated by one or more spaces.

The end of the input is signaled by a line

```
0 0
```

The input must be read from the file ygame.in.

Output

Output texts for each input case are presented in the same order that the input is read. For an input case in the puzzle statement, the output should be a single line with the left-justified text

```
Willy
```

or

```
Benny
```

accordingly to the fact that Willy or, respectively, Benny wins in that case.

The output must be written to standard output.

Sample Input	**Sample output**
3 5 0 1 2 1 0 2 3 0 0 1 1 1 1 2 0 2 3 0 0 2 1 0 1 0 2 0 1 1 1 0 0 0 0	Benny Willy Willy

Problem B
Flatland

Source file name: `flatland.c, flatland.cpp` *or* `flatland.java`

Once upon a time there was *Flatland*, a world whose inhabitants believed was a 2D rectangular region. Flatlanders (the people inhabiting Flatland) assumed that if somebody traveled long enough in one direction, he/she would fall down over the edge of Flatland. Animated Caribbean Movies (ACM) plans to produce a film about Flatland. Before the script of the film is approved, ACM wants to become familiar with life as it was in Flatland by simulating how the world could evolve from given initial situations and some conditions determining life and death.

Flatlanders were nomads by nature as they were always traveling: all of them traveled at the same speed rate on straight lines but each individual had its own direction. As you may imagine, if one observed the life of a lonely Flatlander, he/she would eventually reach Flatland's edge and die. Nevertheless, if two Flatlanders collided, then their fate was improved as their directions changed: the resulting directions were as the former ones reflected in a mirror bisecting the angle between the former directions of the crashing inhabitants. So, a Flatlander survived because a collision with another one could change his/her direction.

However, there were bad news when more than two Flatlanders collided in a single crash: in that case all of them died, disappearing right there. Note that some Flatlanders could die at the same time. If this was the case, the last name of the list of deads was remembered as the *Last Dead One* in that moment (to simplify, we assume they used our modern English alphabet and lexicographic order). The survivors venerated the name of the Last Dead One until a new last dead appeared (and some Flatlander disappeared).

ACM's film begins with a given population in Flatland, where names, positions, and directions of every single individual are known. ACM wants you to help them to determine which would be the name of the last Last Dead One in the whole Flatland's life.

Input

There are NC test cases to solve, $0 < NC < 100$. The first line of the input file has NC. After that, for each testcase, a set of lines:

- the first line contains a number n, the number of Flatlanders in the initial world ($1 \leq n \leq 100$);
- the second line contains two positive integer numbers B and H separated by a space, representing the dimensions of Flatland ($2 \leq B, H \leq 100$). Coordinates in Flatland are points (i, j), with $0 \leq i \leq B$, $0 \leq j \leq H$. Flatland's edges are points with coordinates of the form $(0, j), (i, 0), (B, j)$ or (i, H);
- n lines (one per Flatlander) with four numbers and one string: x, y, d_1, d_2 and *name* separated by a blank. (x, y) represents the position of the Flatlander ($0 < x < B$, $0 < y < H$, and two Flatlanders cannot start in the same position), (d_1, d_2) represents the direction:

(d_1, d_2) is a point on some Flatland's edge, so that the Flatlander is moving towards it; *name* is a string of one to 10 alphabetical uppercase characters, which represents the name of the Flatlander. You may assume that every Flatlander has a unique name.

The input must be read from the file flatland.in.

Output

For each given case, output one line with the name of the Last Dead One.

The output must be written to standard output.

Sample input	**Output for the sample input**
2 2 20 23 1 1 0 0 BOB 3 3 3 0 ALICE 3 20 23 2 2 4 0 ALICE 4 2 2 0 BOB 1 3 0 3 CHARLES	ALICE BOB

Problem C
Guessing Game

Source file name: `guessing.c, guessing.cpp` *or* `guessing.java`

Alice and Bob love games, but they have already played every single game available at their local Games 'R U (Games foR U) store. Tired of playing the same games over and over again, and from not receiving news from the game store, they decided to create their own game. After a few weeks of hard intellectual work Alice came up with *Guessing Game*, a two player game consisting in Alice picking a number and letting Bob try to guess it.

Before playing the game, Alice and Bob agree in the *size* of the game, given by two integer positive numbers, the *range* N and the *limit of strikes* S. Alice chooses a secret integer number X, $0 \leq X < N$. Then Bob uses turns telling Alice integer numbers in order to guess her choice. Each Bob's guess is answered by Alice with a *strike*, if Bob's number is greater than X, with a *smile*, if Bob's number is less than X, or with a *stop*, if Bob's number is precisely X. In this last case the game ends and Bob wins. If Bob receives S strikes the game ends with Alice as winner.

Bob is very competitive. He wants to develop a winning strategy and he starts trying to do it in the case $N = 3$ and $S = 2$. He notes that guessing 1 in the first turn suffices to win: if he receives a strike, Alice's number must be 0 and he can guess it in the next turn; if Alice's number is 1, he already wins; otherwise, he receives a smile, and Alice's number must be 2. Either way he manages to guess the right number without receiving 2 strikes.

Before facing Alice in an official match, Bob asks for a little help with his training. Indeed, he wants you to make a program that computes the minimum number of guesses needed by him to guess any number X, $0 \leq X < N$, receiving at most $S - 1$ strikes.

Input

Input consists of several test cases. The first line of the input file contains a number C specifying the number of cases in the file. Then C lines follow, each one containing two integer numbers N and S, separated by a blank and representing the range and the limit of strikes of a Guessing Game, respectively. You may assume that $1 \leq N \leq 1000$ and $1 \leq S \leq 20$.

The input must be read from the file guessing.in.

Output

For each test case, a single line containing an integer number indicating the minimum number of guesses needed by Bob to guess every possible number picked by Alice.

The output must be written to standard output.

Sample Input	**Sample output**
3 5 1 3 2 7 2	5 2 4

Problem D
The Melding Plague

Source file name: `plague.c, plague.cpp` *or* `plague.java`

The *Nostalgia for Infinity* is an ancient ship that once carried hundreds of thousands, but now its crew is only a handful of Ultras –highly-modified humans adapted to the rigors of long interstellar spaceflight. And they are desperate to find a cure because most of their crew members are still infected with the *Melding Plague*, an alien virus that attacks human cells and machine nanotechnology in equal measure, perverting them into grotesque combinations. It is believed that some special mutations of the virus can help cure the dying Ultras.

The Ultras' patologists identified *protein configurations* as the essential constituents of the Melding Plague, some sort of genetic blueprint of the alien virus. Protein configurations are collections of proteins without any internal order and in which proteins can occur several times. For example, the following is the protein configuration last found in the infected blood of Nostalgia for Infinity's captain John Brannigan:

`POMC CAD CAD SCN5A XIRP2 SCN5A ELTD1.`

Protein configurations *mutate* according to the individual mutation of its proteins. In 1-step mutation *all* proteins in the configuration that can mutate indeed mutate, and those which cannot mutate stay the same. Mutations continue over and over, changing configurations step by step. Fortunately, Ultras' pathologists have identified protein configurations that are curable with appropriate therapies. Then, the hope for an Ultra infected with the Melding Plague is to have the protein configuration of the virus mutating to a curable protein configuration. Of course, therapies must be applied within a limit of mutation steps.

A *protein mutation* is described by an ordered pair of protein names (p, q) stating that protein p mutates to protein q. If $\mathcal{M} = \{(\texttt{CAD}, \texttt{CELR2}), (\texttt{ELTD1}, \texttt{XIRP2})\}$ is a collection of protein mutations, then the protein configuration of the virus in captain Brannigan's blood, depicted above, mutates by $\mathcal{M}$ in 1-step to the protein configuration:

`POMC CELR2 CELR2 SCN5A XIRP2 SCN5A XIRP2.`

Please remember that because the order in the protein configurations is immaterial, the first configuration can be written in 1260 different ways, and the 1-step mutation just shown in 630 different ways.

Your task today is to help the surviving Ultras by building a program that, given a collection of protein mutations $\mathcal{M}$, an initial protein configuration I, a cure protein configuration C, and a natural number n representing a search bound:

- if I mutates to C within at most n steps, computes the minimal number of such mutation steps;
- otherwise, it must inform the Ultras that I cannot mutate to C within n steps.

To easy your burden, Ultras' pathologist are providing you with extra knowledge: they have identified that if a cure by this method exists, one need to consider only *deterministic* mutations $\mathcal{M}$, i.e., if (p, q_1) and (p, q_2) are in $\mathcal{M}$, then $q_1 = q_2$.

Input

The input consists of several test cases. A test case begins with a line containing four natural numbers $N_{\mathcal{M}}$, N_I, N_C, and n separated by a blank, and with $0 \leq N_{\mathcal{M}}, N_I, N_C, n \leq 1000$.

If $N_{\mathcal{M}}$, N_I, and N_C are greater than 0, then $N_{\mathcal{M}} + N_I + N_C$ lines follow. The first $N_{\mathcal{M}}$ lines define the collection $\mathcal{M}$ of *protein mutations* in which each line consists of a pair of strings p and q separated by a blank, representing protein mutation (p, q). Each one of the following N_I lines consist of a string p and a natural number i separated by a blank, representing the number of occurrences i of protein p in the *initial protein configuration* I. Each one of the last N_C lines consist of a string q and a natural number c separated by a blank, representing the number of occurrences c of protein q in the *cure protein configuration* C. The natural number n defines the *search bound*.

The input ends with $N_{\mathcal{M}} = N_I = N_C = n = 0$.

The input must be read from the file plague.in.

Output

For each test case your program must output exactly one line as follows:

- if $\mathcal{M}$ is not deterministic, then output:

  `Protein mutations are not deterministic`

- if $\mathcal{M}$ is deterministic and I mutates to C by $\mathcal{M}$ in at most n mutation steps with a minimum of k mutation steps, then output:

  `Cure found in` k `mutation(s)`

- otherwise output:

  `Nostalgia for Infinity is doomed`

The output must be written to standard output.

Sample input	**Output for the sample input**
2 5 4 3 CAD CELR2 ELTD1 XIRP2 POMC 1 CAD 2 SCN5A 2 XIRP2 1 ELTD1 1 POMC 1 CELR2 2 SCN5A 2 XIRP2 2 2 3 3 3 GP183 NALCN CAC1S GP183 CAC1S 2 YCFI 1 MRP6 3 YCFI 1 MRP6 3 NALCN 2 2 3 3 1 GP183 NALCN CAC1S GP183 CAC1S 2 YCFI 1 MRP6 3 YCFI 1 MRP6 3 NALCN 2 3 2 1 2 CAD YCFI ELTD1 XIRP2 CAD SCN5A CAD 1 YCFI 1 YCFI 2 0 0 0 0	Cure found in 1 mutation(s) Cure found in 2 mutation(s) Nostalgia for Infinity is doomed Protein mutations are not deterministic

Problem E
Preferential Romance

Source file name: `romance.c, romance.cpp` *or* `romance.java`

Marriage Success (MS) is a marriage counseling service advising couples on how to improve the 'get along' experience. MS's idea is simple: each spouse writes down his/her preferences for various criteria of common interest. "Our criteria go beyond physical appearance and passion that guide early romance and tend to blind judgement. We want to understand your values as you live day by day. Happy couples are those whose preferences are compatible or can be made compatible."

Suppose X and Y are qualities to be considered. If a person declares that $X > Y$, it means that this person prefers quality X to quality Y (it does not mean that his/her mate should have a quality, it is only an opinion). Preferences are obviously irreflexive (i.e., $X \not> X$) and they are transitive (i.e., if $X > Y$ and $Y > Z$, then $X > Z$ –which can be abbreviated as $X > Y > Z$).

A couple is *fully compatible* if the preferences of the spouses are *consistent*, that is, if it is possible to arrange the qualities of interest of the spouses in a *compatibility list* reflecting both of their preferences. In this case, if a spouse says $X > Y$, qualities X and Y must occur in the compatibility list and moreover X must be preferred over Y. If a couple is not fully compatible, then perhaps at least it is *passably compatible*: their preferences can be made consistent if some spouse drops at most one preference.

For example, newly-wed Alice and Bob declare their preferences with respect to the following qualities (that they observe in a possible mate): biker, cultured, enthusiastic, foodie, juggler, kayaker, movies, organized, puzzles, rich, theatre, and windsurfer. Their preferences are (observe that they could say nothing about qualities meaning that such quality does not have any importance):

Alice: organized > puzzles > rich, windsurfer > theatre, and rich > movies.

Bob: kayaker > movies > puzzles and rich > theatre.

In this case Alice and Bob are not fully compatible. To see that, a compatibility list should have rich before movies (Alice), movies before puzzles (Bob), and puzzles before rich (Alice), meaning that rich must occur before rich which is impossible. However, the couple is passably compatible: if Alice drops her preference rich > movies, then there is a compatibility list modeling both of their preferences:

kayaker > organized > movies > puzzles > rich > windsurfer > theatre.

MS needs a software solution to determine if clients are full compatible, passably compatible or none of these. Can you help?

Input

The problem input consists of several test cases, each one defined by a set of lines establishing preferences of a couple. A test case is defined as follows:

- the first line contains two strings A and B, separated by blanks, representing the name of the spouses,
- the second line is a sequence of strings of the form (one or more blanks separating items, including commas and final semicolon):

$$q_{11}\ q_{12}\ \cdots\ q_{1r_1}\ ,\ q_{21}\ q_{22}\ \cdots\ q_{2r_2}\ ,\ \cdots\ ,\ q_{m1}\ q_{m2}\ \cdots\ q_{mr_m};$$

  meaning that person A has sets of preferences (q_{ij}'s are strings denoting qualities):

$$q_{11} > q_{12} > ... > q_{1r_1}, q_{21} > q_{22} ... > q_{2r_2}, ..., q_{m1} > q_{m2} > ... > q_{mr_m}$$

- the third line is of the form above representing the preferences of B.

Please consider that a quality name is a string with more than 0 and less than 11 characters, that couples may declare opinions about at most 100 different qualities, and that is guaranteed that the given data is well defined with respect to the above rules. Also, it is guaranteed that the information corresponding to each person does not include preference cycles (i.e., each person is self-compatible).

The end of the input is recognized by a line with $A = B = *$.

The input must be read from the file romance.in.

Output

For each given case, output one line with a single character `F`, `P`, or `N`, meaning that couple with spouses A and B is full compatible, passably compatible, or not compatible, respectively.

The output must be written to standard output.

Sample input	**Output for the sample input**

`Alice1 Bob1`
`organized puzzles rich , windsurfer theatre , rich movies ;`
`kayaker movies puzzles , rich theatre ;`
`Alice2 Bob2`
`organized puzzles rich , windsurfer theatre ;`
`kayaker movies puzzles , rich theatre ;`
`Alice3 Bob3`
`young busy rich , wallet tennis , rich movies , busy toys ;`
`toys movies busy , rich tennis busy , rich movies ;`
`* *`
	`P`
	`F`
	`N`

Problem F
Finding Seats Again

Source file name: `seats.c`, `seats.cpp` *or* `seats.java`

A set of n^2 computer scientists went to the movies. Fortunately, the theater they chose has a square layout: n rows, each one with n seats. However, these scientists are not all from the same research area and they want to seat together. Indeed, there are K independent research groups of scientists among them (no scientist belongs to two of them) with a distiguished leader for each group. Then the leader bought the tickets for his whole group, and he did it in such a way that all his group could seat occupying a rectangular set of seats (and everyone in this set of seats belongs to the same group). Every group was placed satisfying this bizarre condition, although the scientists did not care where the actual assigned areas were.

The usher was informed of the situation and he decided to annotate in a theater map a satisfactory seats deploying. He thought that if he wrote the position of each group's leader in the map indicating besides the corresponding group size, he could tell where to accomodate every scientist. But he discovered that it is not so easy!

The usher asks for your help. You must tell him a way to place the K rectangular areas with the given sizes, and with the corresponding leader for each group seated where it was originally assigned.

Input

Input consists of several test cases, each one defined by a set of lines:

- the first line in the case contains two numbers n and K separated by blanks, with n representing the size of the theater ($0 < n < 20$) and K the number of groups ($K \leq 26$);
- the next n lines describe the usher's map. A one-digit decimal number in the map indicates the seat of a leader and the size of his group. A point indicates that no leader will sit there.

The end of the input is indicated by the line

```
0 0
```

The input must be read from the file seats.in.

Output

For each test case, display an answer consisting in n lines each one of them with n characters representing a seat occupation for the theater. Each group is assigned to an uppercase letter and all of its members are identified with that letter. No two groups are assigned to the same letter.

The output must be written to standard output.

Sample input	**Output for the sample input**
3 3 3.4 ... .2. 7 18 ...4.2. ...45.. 222..3. ...2..3 .24...2 ...2.3. 22..3.. 0 0	ABB ABB ACC AAAABCC DDDDBEF GHIIBEF GHJKBEF LLJKBMM NOJPQQQ NOJPRRR

Problem G
Cut the Silver Bar

Source file name: `silver.c, silver.cpp` *or* `silver.java`

A creditor wants a daily payment during n days from a poor miner in debt. Since the miner can not pay his daily obligation, he has negotiated with the creditor an alternative way, convenient for both parties, to pay his debt: the miner will give an equivalent of a 1μ ($= 0.001mm$) long piece of a silver bar as a guarantee towards the debt. The silver bar owned by the poor miner is initially $n\mu$ units long.

By the end of n days the miner would not have any more silver to give and the creditor would have received an amount of silver equivalent to that of the silver bar initially owned by the miner. By then, the miner expected to have enough money to pay the debt at the next day so that he would have back all his silver.

With this negotiation in mind, the miner has realized that it was not necessary to cut exactly 1μ silver piece from the bar everyday. For instance, at the third day he could give the creditor a 3μ silver piece, taking back the equivalent of a 2μ silver piece which the creditor should already have.

Since cutting the bar was rather laborious and time consuming, the miner wanted to minimize the number of cuts he needed to perform on his silver bar in order to make the daily silver deposits during the n days. Could you help him?

Input

Input consists of several cases, each one defined by a line containing a positive integer number n (representing the length in micras of the silver bar and the number of days of the amortization period). You may assume that $0 < n < 20000$.

The end of the input is recognized by a line with 0.

The input must be read from the file silver.in.

Output

For each given case, output one line with a single number: the minimum number of cuts in which to cut a silver bar of length $n\mu$ to guarantee the debt during n days.

The output must be written to standard output.

Sample input	**Output for the sample input**
1 5 3 0	0 2 1

Problem H
Cargo Trains

Source file name: `trains.c`, `trains.cpp` *or* `trains.java`

The International Cargo and Packaging Company Inc. (ICPC Inc.) transports cargo between two cities: Source City and Sink City. ICPC Inc. does not have its own fleet, instead it contracts the service of two train companies, the company A and company B. Each company has its own network that connects some of the cities at prices that the company decides. For two given cities, it is possible that the route between them is served by both companies, only one company, or none. ICPC Inc. has reached an agreement with both companies that allows it to use their combined services at a discount price, but it has to follow these rules:

1. For a given shipment, ICPC Inc. must specify the percentage of participation of each company given by a for company A and $(1 - a)$ for company B, for a given real number a $(0 \leq a \leq 1)$.

2. When the segment between two cities is served by only one company, ICPC Inc. will pay the price corresponding to that company.

3. When the segment between two cities is served by both companies, the shipment can be shipped using a train from any of the two companies and will pay a fare equal to $a \times C_A + (1 - a) \times C_B$, where C_A and C_B correspond to the fares of company A and B respectively.

4. A shipment could pass through several intermediate cities. The total cost of the shipment corresponds to the sum of the costs of the individual segments between cities calculated according to rules 2 and 3.

ICPC Inc. needs your help to optimize its costs. Specificaly, ICPC Inc. needs to evaluate the cost of different alternatives that combine the participation of the two train companies in different proportions. Given the networks and prices of companies A and B, your task is to calculate the cost of k different combination alternatives. Each alternative is specified by the participation of company A, which corresponds to a real number $0 \leq a \leq 1$.

Input

The input contains multiple test cases. Each test case starts with a line with four integer values separated by spaces, n, m_a, m_b and k, that correspond to the number of cities, the number of edges in the network of company A, the number of edges in the network of company B, and the number of combination alternatives respectively. The ranges for the values are: $2 \leq n \leq 100$, $1 \leq m_a, m_b \leq 5000$, and $1 \leq k \leq 10000$.

The next m_a lines specify the network of company A. Each line has three integer values: N_i, N_j and $C_{i,j}$, separated by spaces, with $0 \leq N_i, N_j < n$ and $0 \leq C_{i,j} \leq 1000000$. The network is an undirected graph and each edge is only listed once. $C_{i,j}$ corresponds to the cost of sending one

kilogram of cargo from city N_i to city N_j or the other way around. Source City corresponds to 0 and Sink City to $n-1$. Then next m_b lines represent the graph corresponding to the network of company B represented in the same way as the network of company A. The last k lines of the test case contain the different combinations to be evaluated, one combination per line. A combination is represented by a real number, $0 \leq a \leq 1$, with maximum 4 decimal digits. The value a specifies company A's participation. Company B's participation is implicitely defined and corresponds to $1 - a$. You can suppose that there is at least one path between the Source City and the Sink City using routes served by any company.

The end of the input is indicated by the line
$-1\ -1\ -1\ -1$.

The input must be read from the file trains.in.

Output

For each combination alternative in each test case, the optimal cost of a trip from a city 0 to city $n-1$ must be printed. If the answer has a decimal part, it has to be truncated without approximation.

The output must be written to standard output.

Sample input	**Output for the sample input**
3 2 2 3 0 1 100 1 2 200 0 1 200 1 2 150 0 1 0.5 -1 -1 -1 -1	350 300 325