2013 National Collegiate Programming Contest

- Problems: There are 10 problems (22 pages in all) in this packet.
- Problem Input: Input to the problems are through the input files. Input filenames are given in the table below. Each input file may contain one or more test cases. Test cases may be separated by any delimiter as specified in the problem statements.
- Problem Output: All output should be directed to the standard output (screen output).
- Time Limit: The judges will run each submitted program with certain time limit (given in the table below).

Table 1: Problem Information Sheet

	Problem Name	Input File	Time Limit
Problem A	Matrix Decomposition	pa.in	5 sec.
Problem B	Symmetric Numbers	pb.in	3 sec.
Problem C	Binary String Check	pc.in	3 sec.
Problem D	Company Merge	pd.in	2 sec.
Problem E	Hitori	pe.in	5 sec.
Problem F	Tower Defense	pf.in	1 sec.
Problem G	3D Future Keys	pg.in	1 sec.
Problem H	Secrete Island	ph.in	3 sec.
Problem I	Disjoint Uni-color Paths	pi.in	2 sec.
Problem J	Square Robots in a Finite Group	pj.in	1 sec.

Problem A

Matrix Decomposition: $A = LL^t$

Input File: pa.in

Time Limit: 5 Seconds

A (real symmetric) positive definite matrix is a square matrix whose eigenvalues are all positive. A positive definite matrix plays an important role in the study of Multivariate Statistical Analysis which are used in a variety of research areas such as Archaeology, Biology, Computer Science, Electrical Engineering, Psychology, Statistics, Zoology, and etc.

One of the most important properties for a real positive definite matrix A is that A can be decomposed as the product of L and L^t with $A = L*L^t$, where L is a lower-triangular matrix (a matrix whose entries above its diagonal are all 0s), and L^t means the transpose of L. For example,

$$A = \begin{bmatrix} 9 & 3 \\ \\ 3 & 5 \end{bmatrix}, \quad then \quad L = \begin{bmatrix} 3 & 0 \\ \\ 1 & 2 \end{bmatrix} \quad and \quad A = L * L^t$$

Given an $n \times n$ positive definite matrix $A \in R^{n \times n}$, this problem asks you to write a program to compute the lower triangular matrix L such that $A = L*L^t$ with $2 \leq n \leq 5$, where each element of A is an integer whose absolute value is not greater than 50. In the following, we provide an algorithm which takes the array $A(1:n, 1:n)$ as input and the output for L are stored in the main diagonal part and the lower triangular part of the array $A(1:n, 1:n)$, that is, the solution for L is the output of $A(i,j),\ 1 \leq j \leq i \leq n$. Of course, the upper triangular part of L is 0.

Algorithm for $A = LL^t$.

for $j = 1, 2, \cdots, n$
    $t = A(j,j)$;
    for $k = 1, 2, \cdots, j-1$
        $t = t - A(j,k) * A(j,k)$;
    endfor
    $A(j,j) = \sqrt{t}$;
    for $i = j+1, j+2, \cdots, n$
        $s = A(i,j)$;
        for $k = 1, 2, \cdots, j-1$
            $s = s - A(i,k) * A(j,k)$;
        endfor
        $A(i,j) = s/A(j,j)$;
    endfor
endfor

Input File Format

The first line of the input file always contains one integer, $K \leq 5$, indicating the number of test cases to come. The next $\sum_{j=1}^{K}(n_j + 1)$ lines consists of K sets of an integer of the dimension of the so-called matrix A followed by its *integer* elements appearing row by row.

Output Format

The first line of the output file is the number of test cases K. The format of remaining lines is similar to those in the Input File Format with the contents of A being replaced by those of L.

Sample Input

```
3
2
9 3
3 5
5
9 0 0 0 0
0 9 0 0 0
0 0 4 0 0
0 0 0 4 2
0 0 0 2 2
```

Sample Output

```
3
2
3 0
1 2
5
3 0 0 0 0
0 3 0 0 0
0 0 2 0 0
0 0 0 2 0
0 0 0 1 1
```

Problem B
Symmetric Numbers
Input File: *pb.in*
Time Limit: *3 Seconds*

Given n and d, a number x with $0 \leqslant x \leqslant n-1$ is a *symmetric number* if $x \equiv dx \pmod{n}$. Note that $x \equiv dx \pmod{n}$ is equivalent to $x = dx - ny$ for some nonnegative integer y. After solving the equation, y must be a multiple of $\frac{d-1}{\gcd(n,d-1)}$. You are asked to find the number of symmetric numbers after n and d are given.

Technical Specification

- $3 \leqslant n \leqslant 2,000,000,000$.
- $2 \leqslant d \leqslant 10,000$.

Input File Format
The first line of the input file contains an integer, denoting the number of test cases. In each test case, there is a line containing a pair of integers for the values of n and d. Note that the values of n and d are separated by a blank.

Output Format
For each test case, output the number of symmetric numbers in one line.

Sample Input

```
4
9 4
25 4
3280 81
1194880 8541
```

Output for the Sample Input

```
3
1
80
20
```

Problem C
Binary String Check
Input File: *pc.in*
Time Limit: *3 Seconds*

Grammar-based codes or Grammar-based compression are compression algorithms based on the idea of constructing a context-free grammar (CFG) for the binary string to be compressed. To compress a data sequence, a grammar-based code transforms into a context-free CFG. The problem to find the smallest grammar for an input sequence is known to be NP-hard. Generally, the produced grammar is further compressed by statistical encoders like arithmetic coding.

To do so, we need to determine whether a binary string can be generated by some rules or not. Here we consider the following 2 types of replacement rules: $A \to BC$, and $A \to b$, where b can be 0 or 1 and the last type is also called *atomic rule*. It means the lefthand side symbol can be replaced by the string on the right. The process usually starts with a specific symbol called *starting symbol*. To help finding the best possible generating sequence, each non-atomic rule has a weight. The weight of a string is the sum of weights of all rules applied while generating it. For a binary string, there may be more than one way to generate it from the rules. We're interested in the one with the minimum weight. Your task is to determine whether a binary string can be generated by a given set of specific rules. If yes, then output the minimum weight required to generate the input string; otherwise, simply output -1.

Technical Specification

1. All the strings are binary and of length at most 200.
2. All the numbers are non-negative integers.
3. K: the number of test cases. $K \leq 10$.
4. M: the number of symbols. $M \leq 100$.
5. N: the number of rules. $N \leq 100$.
6. W: the integer weight of non-atomic rule. $W \leq 10000$.

Input File Format

The first line of the input file contains an integer $K(\leq 10)$ indicating the number of test cases to follow. Each test case starts with 2 positive integers: M and N indicating the number of symbols and the number of rules, respectively. I.e., for convenience, we use integers 2,. . . , $M+1$ to indicate the symbols. Also, all the derivation starts with 2. Then each of the following N lines is with one of the 2 formats: (1) I J K W -1 and (2) I b -1, where I, J, K represent symbols, W indicates the weight of corresponding non-atomic rule,

b is 0 or 1, and -1 indicates the end of a rule. Note that numbers are separated with space(s).

Formally, (1) I J K W indicates that there is a replacement rule $V_I \rightarrow V_J V_K$ with weight W and (2) I b indicates that there is a replacement rule $V_I \rightarrow b$.

The last line of each test case is a binary string, to be tested, of length at most 200, where the string consists of '0' and '1' characters enclosed with a pair of '$'s.

Output Format

For each test case, output the possible minimum weight or -1 in a separate line.

Sample Input

```
2
3 5
2 2 3 4 -1
2 4 2 1 -1
2 0 -1
3 0 -1
4 1 -1
$0000$
3 5
2 2 3 4 -1
2 4 2 1 -1
2 0 -1
3 0 -1
4 1 -1
$0101$
```

Output for the Sample Input

```
12
-1
```

Problem D
Company Merge
Input File: *pd.in*
Time Limit: *2 Seconds*

We need to solve an "Abundant Company Merge" problem. There is a sequence S of n companies $(c_1, \ldots, c_n)$, and company c_i has a capital p_i. Now the government wants to merge these companies. However, the government can only merge two *adjacent* companies in the sequence S. For example at the beginning we can merge company c_3 and c_4, but not c_3 and c_5. The new company will have a capital of the sum of the two companies, and replace the two companies in the sequence S. For example, if originally we have five companies $S = (c_1, c_2, c_3, c_4, c_5)$ then after merging c_3 and c_4 into $c_{3,4}$ the sequence S becomes $(c_1, c_2, c_{3,4}, c_5)$. In this case c_2 and $c_{3,4}$ can merge since they are adjacent, but c_2 and c_5 cannot merge because they are not adjacent. The merge will stop when there is only one company left.

The government can collect a tax when merging two companies, and the tax is 0.001 of the sum of the capitals of the two companies. For example, if c_3 has a capital of $30,000$ and c_4 has a capital of $35,000$, then the tax collected for merging c_3 and c_4 is 65. Now given the companies and their capitals, please calculate the maximum amount of tax the government can collect.

Technical Specification

1. The number of companies n is no more than $2,000$.
2. The initial capital of each company is no more than 100.

Input File Format
The first line of the input contains an integer T indicating the number of test cases. For each test case, the first line contains the number of the companies n. The following n lines represent the capitals of the companies, in unit of thousand.

Output Format
For each test case, output maximum amount of tax the government can collect.

Sample Input

```
1
3
1 2 3
```

Output for the Sample Input

```
11
```

Problem E
Hitori
Input File: *pe.in*
Time Limit: *5 Seconds*

Hitori is a popular single-player puzzle played in some Asian countries. Given an N by N grid with a positive integer no greater than N^2 on each cell, the task is to remove the **least** number of numbers from the grid so that there is no repeated numbers left **in any column or any row**. It can however have repeated numbers on a diagonal. Note there is a constraint that two removed numbers **cannot be horizontally or vertically adjacent**, though they can be diagonally adjacent. Given a Hitori puzzle, there may have no solution or many solutions.

Example 1: A 4 by 4 puzzle on the left and a solution on the right by removing 4 numbers.

2	6	9	2
8	7	1	1
4	3	7	6
2	3	1	2

$\Longrightarrow$

2	6	9	□
8	7	□	1
4	□	7	6
□	3	1	2

Example 2: A 3 by 3 puzzle that has no solution.

2	3	2
3	3	3
2	3	2

Technical Specification

- $0 < N \leq 12$

Input File Format

The first line of the input file contains an integer, denoting the number of test cases to follow. The number of test cases for this problem is at most 15. For each test case, the first line contains N. In the following N lines, the ith line contains N numbers, separated by one or many blanks, which are the N numbers in the ith row.

Output Format

For each test case, output the minimum number of numbers to be removed in order to solve the puzzle. If the puzzle is not solvable, then output "-1".

Sample Input

```
2
4
2 6 9 2
8 7 1 1
4 3 7 6
2 3 1 2
3
2 3 2
3 3 3
2 3 2
```

Output for the Sample Input

```
4
-1
```

Problem F
Tower Defense
Input File: *pf.in*
Time Limit: *1 Second*

Tower defense (TD) is a computer game in which we build towers to defend enemies from reaching our castles. This task will ask you to write a program to find the most economical way to achieve this goal. The scenario and rules of the TD game are described as follows.

Your territory is composed of $n \times m$ grids. The n rows are numbered from 0 to $(n-1)$, from top to bottom; the m columns are numbered from 0 to $(m-1)$, from left to right. Your castle spans across all rows at column 0. The game proceeds round by round. Let (x_i, y_i) denote the position of an enemy i. At the first round (round 0), there are n enemies, with

1. $x_i = i$, for $i = 0, 1, ..., n-1$
2. y_i is an integer in $[0, m-1]$.

At the end of each round, every enemy i moves one step toward your castle, i.e. $y_i = y_i - 1$. If any one enemy reaches your castle, you **lose** the game.

Now let's talk about how to defend. The life of an enemy i is denoted by h_i. In each round, you are allowed to build at most one tower in your castle, i.e. at column 0 of a certain row. There are K types of towers; type j $(0 \le j \le K-1)$ requires cost c_j and construction time t_j, and it causes damage d_j to an enemy. There are two construction rules:

1. You can build only one tower on each row.
2. Once you start to build a tower of type j at round t, this tower is *finished* at round $(t + t_j - 1)$ and can start to attack enemies at round $(t + t_j)$. You cannot start to build the next tower before round $(t + t_j)$.

In each round, you act first. If no tower is under construction, you can start to build a new tower. For each row, if there is a *finished* tower of type j, it can hurt the enemy i on the same row by damage d_j $(h_i = h_i - d_j)$. If $h_i \le 0$, the enemy i is dead and causes no threat along this row. At the end of this round, all live enemies move one step toward your castle. If you can eliminate all n enemies before any one reaches your castle, you **win** the game.

Figure 1 illustrates how the game proceeds. We assume that the territory is composed of 2×8 grids. There are two types of towers. The starting positions of enemy 0 and enemy 1 are (0, 7) and (1,6), respectively. The life of enemy 0 and enemy 1 are 10 and 3 in round 0, respectively. We start to build a tower of type 0 in round 0. At the end of round 0, the enemies move one step, staying at position (0, 6) and (1, 5). Since t_0 is 2, the tower is finished at round 1 and can start to attack enemy 0 at round 2. Thus, at the end of

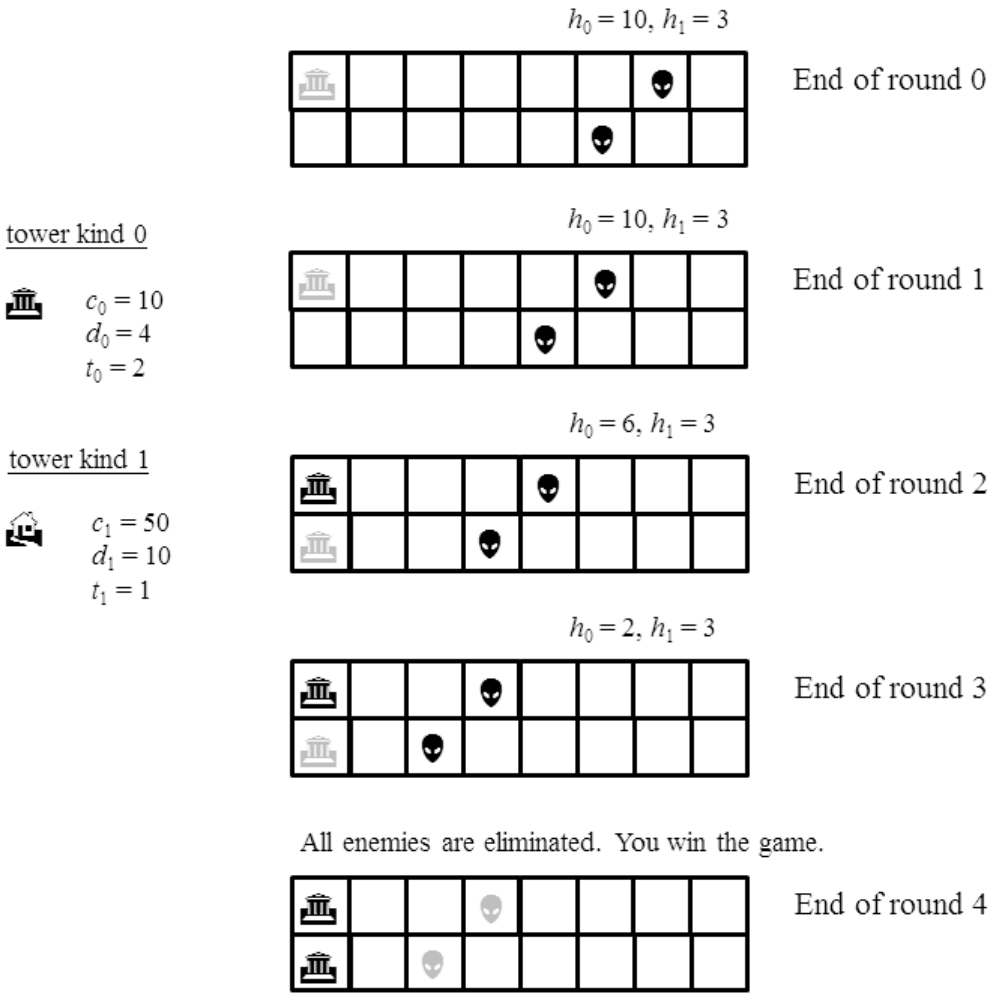

Figure 1: An example in which you win by building two towers of type 0. This is one of the most economical solutions.

round 2, enemy 0 moves to position (0, 4), and its life is reduced by 4 (d_0). At the end of round 4, the enemies are eliminated and you win the game. In this example, you can start to build a tower of type 0 on row 0 in round 1. You can also win the game by building a tower of type 1, but it costs more ($c_1 = 50 > c_0 = 10$) and is not the most economical way to win.

Figure 2 is another example. The enemys' starting positions are (0, 5) and (1, 4) in round 0. If you start to build a tower of type 0 on row 0 in round 2, the tower starts to attack enemy 0 in round 4. At the end of round 4, the enemy's life is reduced to 6 and reaches the castle. It means that you lose the game. In this example, you need to build a tower of type 1 in round 2 to win the game.

Your task is to write a program to determine the minimal cost required to win the game.

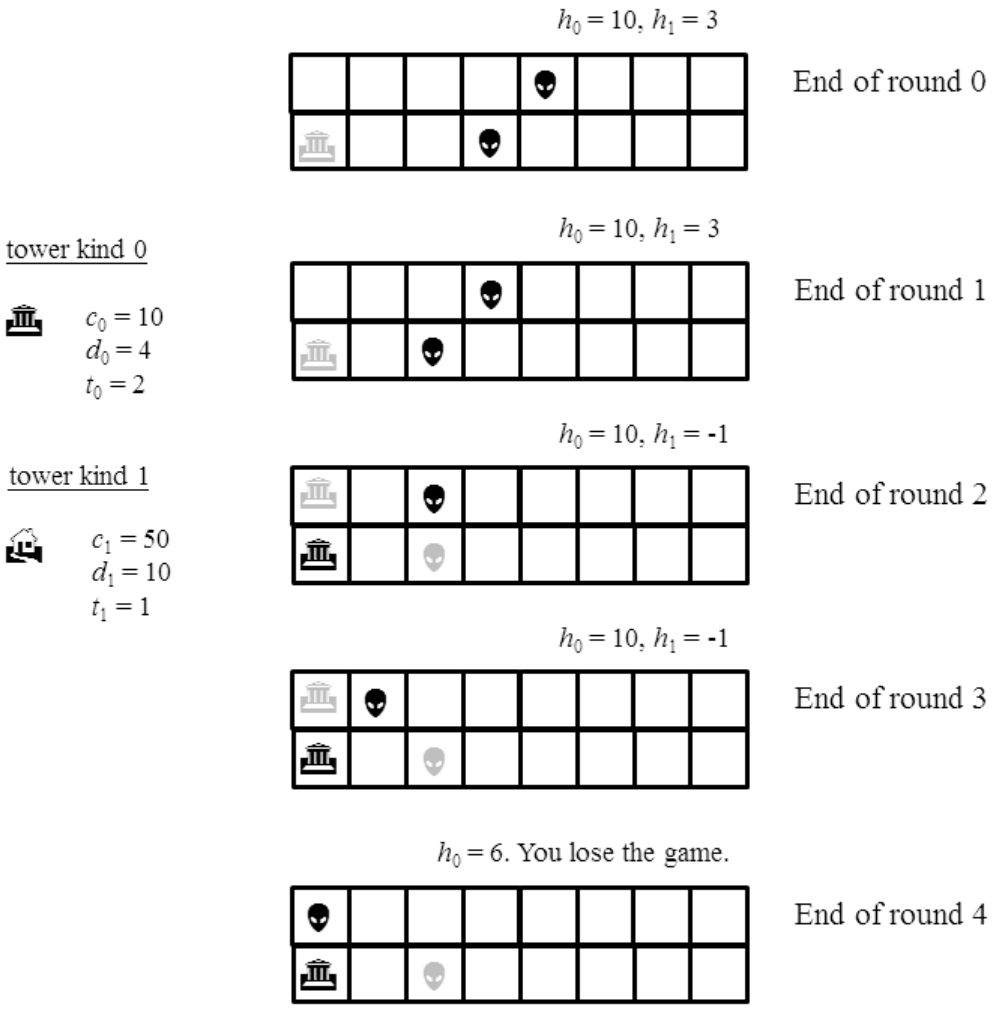

Figure 2: An example in which you lose by building only towers of type 0. You need to build at least one tower of type 1 to win this game.

Technical Specification

- The number of rows n and columns m, $1 \leq n \leq 10$ and $1 \leq m \leq 1,000$.
- The number of tower types K, $1 \leq K \leq 10$.
- The life of enemy h_i, $1 \leq h_i \leq 100$.
- The cost of tower c_j, $1 \leq c_j \leq 100$.
- The damage of tower d_j, $1 \leq d_j \leq 20$.
- The construction time of tower t_j, $1 \leq t_j \leq 10$.

Input File Format

1. The first line of the input file contains an integer, denoting the number of test cases to follow.
2. For each test case,
   (a) The first line contains three integers for K, n, and m, separated by a space.
   (b) In the next K lines, each line j ($j = 0, ..., K-1$) contains three integers, denoting the cost c_j, damage d_j, and construction time t_j. Every two integers are separated by a space.
   (c) In the last n lines, each line i ($i = 0, ..., n-1$) contains two integers, denoting the starting column y_i and the life h_i of enemy i. Two integers are separated by a space.
3. Every two cases are separated by an empty line.

Output Format

For each test case, output the minimal cost required to win the game if it is possible to win. Otherwise output -1. Output the result for each test case in a separate line.

Sample Input

```
2
2 2 8
10 4 2
50 10 1
7 10
6 3

2 2 8
10 4 2
50 10 1
5 10
4 3
```

Output for the Sample Input

```
20
60
```

Problem G
3D Future Keys
Input File: *pg.in*
Time Limit: *1 Second*

In year 2100, digitalized virtual 3D key technology is getting popular. 3D locks and keys are made of 3D cubes as in Fig. 1. A 3D key is consisted of blade part (black cubes) and bow part (white cubes). A 3D lock contains keyway part (greyed cubes) and other remaining part (white cubes). The cubes of a lock or key are connected; that is, they must be attached each other at cube faces.

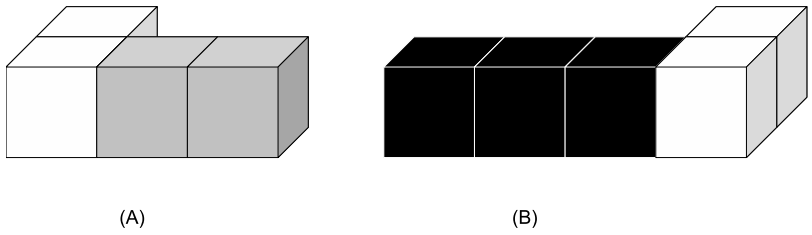

Figure 1: (A). A lock with greyed color keyway. (B). A key with black color blade.

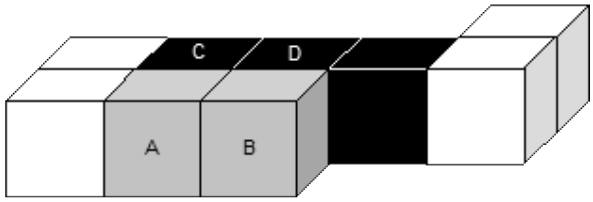

Figure 2: Contacted keyway and blade.

In real world, a physical key may not be able to insert into a lock because the entrance of a lock can block a key which has incorrect shape. However, 3D keys are digitized virtual keys. So, you do not need to check if a key is blocked while approaching a lock from some directions.

To open a 3D lock, you need to make the blade contact the keyway in a way that maximizes the total number of contacted cubes. White cubes can make some choices of contacting infeasible. The total number of contacted non-white cubes can be computed by counting the touched (i.e., cube face to cube face) opposite cubes for each keyway and blade cubes. For example, in Fig. 2, the keyway cube A has one blade cube C touching its one of 6 faces. So, cube A has one point. In this example, B has one point, C has one point, and D has one point. Touching with white cubes scores no points. 4 points are the maximum value for the example. Given a key and a lock, please compute the maximum value among all the combinations.

Input File Format

The test data begins with an integer N which is the number of test cases. In each test case, a lock is described first. The data of a lock or a key begins with an integer $C(1 \leq C \leq 20)$, which is the number of cubes. The cubes are described using a XYZ-axis in Fig. 3. The first cube always starts at position (0,0,0). The other cubes are described relatively based on (0,0,0). A cube is described by $(x\ y\ z\ t)$, where $x, y, z(-20 \leq x, y, z \leq 20)$ are 3D coordinates and t is the types of the cube. The types can be:

'K': A keyway cube
'B': A blade cube
'W': A white cube.

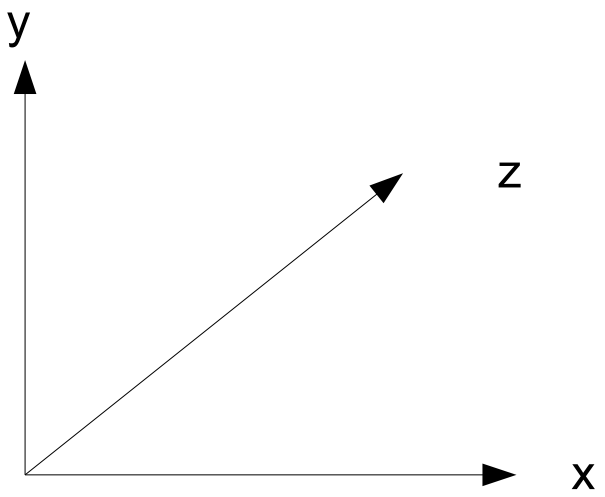

Figure 3: 3D XYZ-axis system.

Output Format

For each test case, please print the maximum points of contacted cubes.

Sample Input

```
2
4
0 0 0 W
1 0 0 K
2 0 0 K
0 0 1 W
5
0 0 0 B
1 0 0 B
2 0 0 B
```

```
3 0 0 W
3 0 1 W
4
0 0 0 W
1 0 0 K
2 0 0 K
0 0 1 W
6
0 0 0 B
1 0 0 B
2 0 0 W
1 1 0 B
1 1 -1 B
2 0 -1 W
```

Output for the Sample Input

```
4
6
```

Problem H
Secret Island
Input File: *ph.in*
Time Limit: *3 Seconds*

Professor Lee is an archaeologist who discovers a great treasure hidden on a secret island S. The island has N sculptures (numbered $1, 2, \ldots, N$) and $N-1$ trails. Each trail connects a pair of distinct sculptures, and can be followed in either direction. A *path* is a series of trails. Furthermore, there is **exactly one** path connecting any two sculptures in S.

The *distance* between two sculptures i and j, denoted by $d(i, j)$, is the minimum number of trails connecting them. For example, in Figure 1, the distance between Sculpture 7 and Sculpture 6 is 4. The *eccentricity* of a sculpture i, denoted by $\epsilon(i)$, is $\max_{1 \le j \le N} d(i, j)$. An *ideal sculpture* is a sculpture with the minimum eccentricity.

Professor Lee believes that the treasure is hidden under some ideal sculpture. Your task is to write a computer program to compute the number of ideal sculptures on the secret island, and compute the sum of the labels of ideal sculptures. For example, consider a secret island S with eight sculptures and seven trails shown in Figure 1. Sculptures 2 and 3 are ideal sculptures with $\epsilon(2) = \epsilon(3) = 3$. The sum of their labels is 2+3=5.

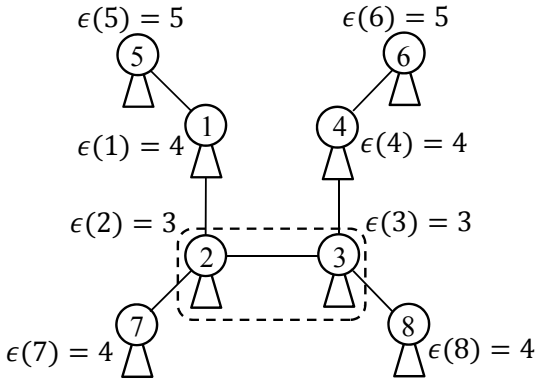

Figure 1: A secret island S with eight sculptures and seven trails.

Technical Specification

- $1 \le N \le 1,000,000$.

Input File Format

The first line of the input file contains an integer, denoting the number of test cases to follow. For each test case: the first line contains one positive integer N. The following $N-1$ lines represent $N-1$ trails in which each line contains two integers (separated by a space) representing two sculptures connected by the trail.

Output Format

For each test case, output the number of ideal sculptures and the sum of their labels. The two output numbers are separated by a space.

Sample Input

```
2
8
1 5
1 2
2 7
2 3
3 4
4 6
3 8
3
1 2
2 3
```

Output for the Sample Input

```
2 5
1 2
```

Problem I
Disjoint Uni-color Paths
Input File: *pi.in*
Time Limit: *2 Seconds*

A *social network* is a graph $G = (V, E)$ in which each node represents an actor and there is a link between two nodes if they have some kind of relation. For a *multi-relations social network*, there are different kinds of relations, and it can be described by an edge-colored graph. That is, each edge is labeled by a color number—a nonnegative integer at most $c - 1$. Note that there may be **several edges of the same or different colors between a pair of nodes**.

It is an important issue to compute the maximum number of disjoint paths between two nodes when analyzing a social network. Let $s, t \in V$. A path with endpoints s, t is called an st-path. Two st-paths are *disjoint* if they have no common internal node, and a set of st-paths are disjoint if they are mutually disjoint. A path is of uni-color if all the edges of the path have the same color. The length of a path is the number of edges in the path. For social networks, short paths are much more important, and therefore only paths of length at most three will be considered in this problem. Formally, your task is to write a program to solve the following problem.

> Given an edge-colored graph G and two nodes s, t, find the maximum number of disjoint uni-color st-paths of length at most three.

Technical Specification
In this problem, graphs are undirected. For each instance:

1. the node set $V = \{i | 0 \leq i \leq n - 1\}$ and $2 \leq n \leq 500$;
2. $s = 0$ and $t = n - 1$;
3. the number of color is at most 10; and
4. the number of edges m is at most 50,000.

Input File Format
The first line of the input file contains an integer, denoting the number of test cases to follow. For each test case, the first line contains two integers n and m. In the next m lines, each line contains three integers separated by a space, which are the two endpoints and the color of an edge (in this order).

Output Format
For each test case, output the solution in one line.

Sample Input

```
2
5 10
0 4 0
4 0 0
0 4 0
0 4 1
0 1 2
4 1 2
2 0 3
0 3 2
2 3 3
3 4 3
6 11
0 1 1
1 2 1
2 3 1
3 5 1
0 3 2
3 4 2
4 5 2
2 5 7
5 2 7
2 5 7
3 4 8
```

Output for the Sample Input

```
6
1
```

Problem J
Square Roots in a Finite Group
Input File: *pj.in*
Time Limit: *1 Second*

Finite groups are used in almost all modern cryptosystems. For example, RSA cryptosystem uses multiplicative group $\mathbf{Z}_n^*$, where n is a product of two large primes. They are also used in our daily life. For example, if we map Sunday to 0, Monday to 1, ..., Saturday to 6, then it is an additive group $\mathbf{Z}_7$.

Let n be a positive integer. We use $\mathbf{Z}_n$ to denote the *additive group* whose elements are $0, 1, \ldots, n-1$. We use $\mathbf{Z}_n^*$ to denote the *multiplicative group* whose elements are the integers a in $1, 2, \ldots, n-1$ with $\gcd(a, n) = 1$. For example, the elements in $\mathbf{Z}_6$ are $0, 1, 2, 3, 4, 5$. The elements of $\mathbf{Z}_6^*$ are $1, 5$.

In addition to the elements, a group has an operator associated with it. The operator of an additive group is *addition* $(+)$, and the operator of a multiplicative group is *multiplication* $(\times)$. These two operators are very similar to the ordinary addition and multiplication of integers, except that all results should be the reminder after dividing by n. For example, $3 + 4 \equiv 7 \equiv 1$ in $\mathbf{Z}_6$; $3 \times 4 = 12 \equiv 5$ in $\mathbf{Z}_7^*$.

In this problem, you are going to implement an efficient algorithm for computing square roots in a finite multiplicative group $\mathbf{Z}_p^*$, where p is a prime. Formally, the problem is: Given two positive integers a and p, solve the congruence equation

$$x^2 \equiv a \pmod{p}$$

for x. For example, let $p = 13$, $4^2 = 16 \equiv 3 \pmod{13}$. Therefore, the square root of 3 is 4 in $\mathbf{Z}_{13}^*$.

Note that we use $a \equiv b \pmod{n}$ to denote that a and b have the same reminder after dividing by n.

Of course not all elements in $\mathbf{Z}_p^*$ has square roots. Look at the following data for $p = 13$:

$1^2 \equiv 1$	$12^2 \equiv 1$
$2^2 \equiv 4$	$11^2 \equiv 4$
$3^2 \equiv 9$	$10^2 \equiv 9$
$4^2 \equiv 3$	$9^2 \equiv 3$
$5^2 \equiv 12$	$8^2 \equiv 12$
$6^2 \equiv 10$	$7^2 \equiv 10$

It can be seen that only half of the elements in $\mathbf{Z}_{13}^*$, namely $1, 3, 4, 9, 10, 12$, has square roots. This is true in $\mathbf{Z}_p^*$ for any prime $p > 2$.

In this problem, we assume that the prime $p > 2$. The following facts about the multiplicative group $\mathbf{Z}_p^*$ may help you in solving the square root problem.

1. For any $a \in \mathbf{Z}_p^*$, $a^{p-1} \equiv 1 \pmod{p}$.
2. An element $a \in \mathbf{Z}_p^*$ has square roots if and only if $a^{(p-1)/2} \equiv 1 \pmod{p}$.
3. For any element $a \in \mathbf{Z}_p^*$, if x is a square root of a in $\mathbf{Z}_p^*$, then $-x \equiv p - x \pmod{p}$ is also a square root of a, and these are the only square roots of a in $\mathbf{Z}_p^*$.

Note that, by the above facts, an element $a \in \mathbf{Z}_p^*$ does not have square roots if and only if $a^{(p-1)/2} \equiv -1 \equiv p-1 \pmod{p}$.

One way to solve the problem can be described briefly as follows.

1. Write $p - 1 = 2^s m$, where m is odd.
2. Find an element $b \in \mathbf{Z}_p^*$ which has no square root.
3. Let $b = b^m \bmod p$, $x = a^{(m+1)/2} \bmod p$ and $t = a^m \bmod p$. It can be verified that $x^2 \equiv at \pmod{p}$.
4. Design a loop to modify the values of x, t and b by using the old value of b.

Make sure after each iteration, the congruence relation $x^2 \equiv at \pmod{p}$ holds. The algorithm finds the solution x when $t = 1$. Of course, you need to handle the special case when $p - 1 = 2m$.

Input Format

The input to the problem is a file containing many test cases. Each test case is written in one line, which contains two integers a and p, $1 < a < p < 2^{31}$. The last line of the input file contains a single 0. You do not need to process this line.

Output Format

The outputs for each test case is an integer $x \in \mathbf{Z}_p^*$, satisfying $x^2 \equiv a \pmod{p}$. Since the solution is not unique, always print the solution with smaller value. Print the output of each test case in one line with no leading and tailing spaces. If there are no solutions, print "`no solutions`".

Sample Input

```
3 13
2 17
3 37
0
```

Sample Output for the Sample Input

```
4
6
15
```