2006 ACM International Collegiate Programming Contest - KaoHsiung

2006/11/18

There are 9 problems in this problem set. The team who solves the most number of problems wins the contest. If more than one team solve the same number of problems, then the team who takes less time to solve the problems wins.

Each submitted program will be judged to be either `correct` or `incorrect`. A correct program must solve all the test cases of the problem correctly. The time to solve a problem is measured from the beginning of the contest to the time the submission of the program which solves the problem correctly, plus a penalty of 20 minutes for each incorrect submission for the problem. There will be no penalties for those problems which cannot be solved correctly.

Sample input data and sample output are provided in each problem. You may create these sample input data to test your program. However, test cases used by the Judges may not be the same as the sample input.

It is not necessary to include comments in your programs. Any algorithm can be used to solve the problems. However, a program whose running time exceeds 30 seconds will be considered to be incorrect unless the time is particularly specified by the problem.

Each input file may contain more than one test case. The description of the input format is based on records. A record is usually contain a list of data. Unless stated otherwise, these data may span more than one line. The number of characters in a line is no more than 255. A special record, usually one or more 0's, indicates the end of input data. Anything after this special record should be ignored.

The name of the input file for problem i is `p`i`.in`. Error checking of the input data is not required. You may assume that the input data are always correct. The output file must be the standard output, unless it is specified in the problem.

The table below is the list of 9 problems and their time limits.

	Problem	Time Limit
A	Print Words in Lines	0.5 seconds
B	The Bug Sensor Problem	30 seconds
C	Pitcher Rotation	10 seconds
D	Lucky and Good Months by Gregorian Calendar	3 seconds
E	Route Planning	10 seconds
F	Shift Cipher	3 seconds
G	A Scheduling Problem	3 seconds
H	Click the Lines	10 seconds
I	Perfect Service	3 seconds

Problem A

Print Words in Lines

Input File: pa.in

Time Limit: 0.5 seconds

We have a paragraph of text to print. A text is a sequence of words and each word consists of characters. When we print a text, we print the words from the text one at a time, according to the order the words appear in the text. The words are printed in lines, and we can print at most M characters in a line. If there is space available, we could print more than one word in a line. However, when we print more than one word in a line, we need to place exactly one space character between two adjacent words in a line. For example, when we are given a text like the following:

```
This is a text of fourteen words and the longest word has ten characters
```

Now we can print this text into lines of no more than 20 characters as the following.

```
This is
a text of
fourteen words
and the longest
word
has ten characters
```

However, when you print less than 20 characters in a line, you need to pay a penalty, which is equal to the square of the difference between 20 and the actual number of characters you printed in that line. For example in the first line we actually printed seven characters so the penalty is $(20-7)^2 = 169$. The total penalty is the sum of all penalties from all lines. Given the text and the number of maximum number of characters allowed in a line, compute the minimum penalty to print the text.

Input

The first line of the input is the number of test cases (C). The first line of a test case is the maximum number of characters allowed in a line (M). The second line of a test case is the number of words in the text (N). The following N lines are the length (in character) of each word in the text. It is guaranteed that no word will have more than M characters, N is at most 10000, and M is at most 100.

Output

The output has C lines. Each line has the minimum penalty one needs to pay in order to print the text in that test case.

Sample input

```
2
20
14
4
2
1
4
2
8
5
3
3
7
4
3
3
10
30
14
4
2
1
4
2
8
5
3
3
7
4
3
3
10
```

Sample output

```
33
146
```

Problem B

The Bug Sensor Problem

Input File: pb.in

Time Limit: 30 seconds

Mr. Macdonald is a farmer. He has a huge land to manage. To monitor the number of bugs in his land, he asks help from the famous professor T. Professor T is an expert on computer science.

Professor T studies several efficient approach and suggests Mr. Macdonald to setup a wireless sensor system in his land. The system will be setup as follows:

In each predefined location, one wireless sensor will be established. Since all sensors are operated by batteries, the powers consumed by the sensors are determined by the effective communication distance (ECD) between sensors. Mr. Macdonald is a nice old man. He prefers not to trouble professor T much. Therefore, he decides that all sensors will be set in the same power level. That is, all sensors will have the same effective communication distance.

The land is so huge that it is not possible to cover all spots by the sensors. However, each sensor can broadcast the collected data to their neighbors as long as the neighbors are in its effective communication distance. But the total number of sensors is relative small comparing to the land. Mr. Macdonald needs to travel the whole land to collect the data from sensors everyday.

Mr. Macdonald is getting old. He hopes that the computer in his house can collect all data from all sensors automatically. Again, he called professor T for help. This time, professor T suggests Mr. Macdonald to setup a base station in his house. The house is right in the center of the land.

Due to the limited budget, the number of receiver/transmitter Mr. Macdonald can afford is limited and is relatively small comparing to the total number of sensors. It is impossible to assign receiver/transmitter to every sensor. Therefore, sensors with no receiver/transmitter need to send their data to the sensor with receiver/transmitter (directly or indirect) first, After that, the sensor with receiver/transmitter sends all data it receives from other sensors along with the data it collects back to the base station. You may assume that the receiver/transmitter has enough power such that it always can send data back to the base station.

Professor T promises to write the necessary software to make all sensors work together in this way, but one important issue need to be studied. If all sensors are set at the maximum power level, all sensors might be able to send their data back without troubles, but the battery will be out-of-power soon. To save power, professor T need to decide the minimum power level needed such that the battery can have longest operating time while all sensor data can be collected by the base station.

Although professor T is good in programming, he is weak in algorithm design. Your goal is to help professor T to write a program to determine the minimum power level needed to set all sensors accordingly. To simplify our problem, please report the ECD corresponding to the

minimum power level. Please apply the ceiling function to your answer.

Here is an example:

Assume that the land is 10x10. There are three sensors, located at (1,1), (2,1) and (8,7). We also assume that there are 2 receiver/transmitters. In this example, the ECD of all sensors need to be set at 1.

Input

The first line contains the number of test cases w, $1 \leq w \leq 10$. Then the w test cases are listed one by one. In each test case, the first line is a single integer, representing the number of receiver/transmitters for that test case. After that, the test case consists of some lines with two numbers each line:

X Y

Here two numbers are separated by a single blank, X is an integer, denoting the x-coordinate of the sensor, and Y is also an integer, denoting the y-coordinate of the same sensor. Each test case is ended by the following line: -1

Please note that the land is a rectangle with dimemsions 100000x100000.

Output

For each test case, output the corresponding ECD.

Sample input

```
1
2
1 1
2 1
8 7
-1
```

Sample output

```
1
```

Problem C

Pitcher Rotation

Input File: pc.in

Time Limit: 10 seconds

For professional baseball team managers, it is an important task to decide the starting pitcher for each game. In the information era, massive data has been collected in professional sports. The manager knows the winning percentage of each pitcher against each team. Unfortunately, when playing against a certain team you cannot always pick the pitcher with the highest winning percentage against that team because there is a rule saying that after pitching a game the pitcher has to rest for at least four days. There are n pitchers ($5 \leq n \leq 100$), m opponent teams ($3 \leq m \leq 30$), and there are g ($3 \leq g \leq 200$) games in a season, and the season lasts for $g + 10$ days. Furthermore, there is at most one game per day. You are given an m by n matrix P, where an element in P, p_{ij}, denote the winning percentage of pitcher j against team i, and a list of $g + 10$ numbers, $d_1, d_2, \ldots, d_{g+10}$, to represent the schedule of the team, where d_i denotes the opponent team and $d_i = 0$ denotes that there is no game at the i^{th} day of the season. Your task is to decide the starting pitcher for each game so that the expected number of winning game is maximized.

Input: The first line contains an integer t ($1 \leq t \leq 5$) indicating the number of teams that need your help. The data about these t teams follows. For each team, the first line contains the number of pitchers n, the number of opponent teams m, and the number of games in a season g. The next m lines contains the information about winning percentage of each pitcher against each team; the first line is $p_{11}, p_{12}, \ldots, p_{1n}$, and the i^{th} line is $p_{i1}, p_{i2}, \ldots, p_{in}$, where each p_{ij} is a two-digit number(for example, 92 represents 0.92). The next $g + 10$ lines describe the schedule of the season, $d_1, d_2, \ldots, d_{g+10}$.

Output: The maximum value of expected game won for these t teams in the order of their appearance in the input file, output the answer for each team in separated lines.

Sample Input

```
1
5 3 6
91 90 50 50 50
65 40 60 60 60
66 40 60 60 60
1
2
3
3
2
1
0
0
0
0
0
0
0
```

0
0
0

Sample Output for the Sample Input

4.26

Problem D

Lucky and Good Months by Gregorian Calendar

Input File: pd.in

Time Limit: 3 seconds

Have you ever wondered why normally an year has 365 days, not 400 days? Why August have 31 days, but February have only 28 days? Why there are 7 days, not 6 days, in a week? Do people in ancient time use the same calendar as we do? There are many interesting conjectures and theories about those problems. Now we will tell you one story that may help explaining plausible answers to these questions. Using information in the story, you are then ask to solve an interesting problem using computer. Note that there are many theories about the calendar system discussed. This problem set will tell only one of them in a simplified way.

Throughout history, people keep track of time by observing the relative positions of the earth, the moon and the sun. A *day* is the amount of time the earth completes a self rotation. An *year* is defined to be the amount of time the earth orbits the sun. The earth takes roughly 365.242190 days to orbit the sun with some small variations. For practical purpose, a calendar year needs to have an integral number of days. Hence people need to add *leap* days to keep the calendar synchronized with the sun. If you keep a calendar year to have 365 years, you need to add one more day in a leap year roughly about every 4 years. However, this kind of calendar will not be in perfect synchronization with the earth's position orbiting the sun because it advanced 365.25 days in average, which is slightly more than the actual period.

Depending on how accurate you can measure the period of the earth orbiting the sun, you need to invent different formulas for leap years. Several famous Western calendar systems have been invented, not to mention the more complex Oriental systems. In order to save programmers' efforts, we will not discuss the Oriental, such as Chinese, calendar systems. We will focus on major Western calendar systems. The earliest one may be the Julian calendar created by Julius Caesar in 46 BC. It is not accurate enough and will have one day off every 128 years. The next one is the Astronomical Julian calendar invented by Joseph Justus Scaliger around the 16th century. Both have simple formulas to determine which year is a leap year.

The next major one is called Gregorian calendar that was invented at the year 1582 because the synchronization of the earth's orbiting and the calendar is finally noticed by people. In this system, a leap year is dropped every 100 years unless it is every 400 years. By doing this modification, the average number of days in a calendar year is 365.2425. Note that this system is also not perfect. It adds one more day every 3289 years. There are other more modifications suggested, such as the one by Astronomer John Herschel, the Greek Orthodox, and the SPAWAR group in the US Navy. For simplicity, people use Gregorian calendar system though it may not be perfect.

The following is the formula for the Gregorian calendar to determine whether an year is a leap year or not. An year y, $y \geq 1582$ and $y \neq 1700$, is a leap year if and only if

- y is divisible by 4, and
- y is not divisible by 100 unless it is divisible by 400.

An year y, $0 < y < 1582$ is a leap year if and only if

- y is divisible by 4.

Hence year 4 is a leap year, year 100 is a leap year, year 1900 is not a leap year, but year 2000 is a leap year. A leap year has 366 days with the extra day February 29. A non-leap year has 365 days.

During your computation, you may also want to observe the following facts about Gregorian calendar. Many calendar systems were used by people in different areas in the Western world at the same time. The current Western calendar system, primarily follows Gregorian calendar, and is so called the Gregorian Reformation, was adopted by Britain and the possessions on September 3, 1752. For lots of reasons that we are sure you do not want to read in this problem description, 11 days are eliminated starting September 3, 1752 in order for people not to rewrite history. That is, in the Gregorian calendar, there is no days in between September 3, 1752 and September 13, 1752. Note that Rome adopted the Gregorian calendar at the year 1582, when it was invented. Also for historical reasons, the year 1700 is declared a leap year in the Gregorian calendar. There are other variations about the Gregorian calendar system, however, we will use the one that is defined above.

A lunar *month* is defined to be the average time between successive new or full moons which is 29.531 days. People observe in average 12.368 full moons in an year. Unfortunately, this is also not an integral number in terms of days. Hence if we set an year to have 12 months with each month having 30 days, we need to add several days each year. To save the trouble, an alternative way is to have the number of days in a month to alternative between 30 and 31. However, this introduces one extra day. After lots of struggle, the Gregorian calendar defined the numbers of days in each month during a non-leap year to be 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, respectively from the first month to the 12th month. One more day is added on February in the leap year. The names for the months in sequence from the first month — January, February, March, April, May, June, July, August, September, October, November, and December, are also very interesting and have lots of stories associated with them. For example, the Roman Senate named the month of July after Julius Caesar to honor him for reforming their calendar. However, we do not have time to cover them here.

In ancient time, a *week* may have different number of days, say from 4 to 10 days. In the Gregorian calendar system, a week corresponds roughly to the moon's quarter phase whose position can be clearly observed by people. Hence people can easily measure a week. Also for some other reasons, such as religion, it is defined as 7 days. The names of the 7 days in sequence — Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday, all have interesting stories. However, we also do not have time to cover them here.

Throughout history, people believe the relative positions of the stars can decide their fate. This is also true for people living in an island T. In island T, people are working from Monday through Friday every week and enjoy holidays on every Saturday and Sunday without exceptions. There is no other holidays. From ancient tales, a month is called *lucky* if the last working day in this month is Friday. For examples, the last working day of September, 2006 is September 29, 2006 — which is Friday. Hence it is lucky. The last working day of July, 2006 is July 31, 2006 — which is Monday. Hence it is not lucky. The last working day of August, 2006 is August 31, 2006 — which is Thursday. Hence it is also not lucky. It is believed that if one eats only vegetable everyday during a lucky month, he/she will have a good fortune in getting rich.

Also from ancient tales, a month is called *good* if the first working day in this month is Monday. For examples, the first working day of July, 2006 is July 3, 2006 — which is Monday. Hence it is good. The first working day of October, 2006 is October 2, 2006 — which is Monday. Hence it is also good. The first working day of August, 2006 is August 1, 2006 — which is Tuesday. Hence it is not good. The first working day of September, 2006 is September 1, 2006

— which is Friday. Hence it is also not good. It is believed that if one goes to bed before 10 PM every day during a good month, he/she will be very healthy. A month can be both good and lucky at the same time.

Given a period of time, your task is to report the number of lucky months and the number good months during this period of time using the described Gregorian calendar system.

Input: The first line contains the number of test cases w, $1 \leq w \leq 10$. Then the w test cases are listed one by one. Each test case consists of 1 line with four numbers:
Y_s M_s Y_e M_e
where two numbers are separated by a single blank, Y_s is an integer, $0 < Y_s < 10000$, denoting the starting year in western style, M_s is an integer, $1 \leq M_s \leq 12$, denoting the starting month, Y_e is an integer, $0 < Y_e < 10000$, denoting the ending year in western style, M_e is an integer, $1 \leq M_e \leq 12$, denoting the ending month.

Note that you can be sure the month indicated by M_s, Y_s is never before the month indicated by M_e, Y_e.

Output: For each test case, output the number of lucky months and the number of good month in between the month M_s of the year Y_s (including this month) and the month M_e of the year Y_e (including this month) in one line. The two numbers are separated by a single blank.

Sample Input

2
2006 9 2006 9
2006 7 2006 9

Sample Output for the Sample Input

1 0
1 1

Problem E

Route Planning

Input File: pe.in

Time Limit: 10 seconds

Superior Island is a very picturesque island and only bicycles are allowed on the island. Therefore, there are many one-way bicycle roads connecting the different best photo-shooting spots on the island. To help the visitors plan their trip to the island, the tourism commission wants to designate r different bicycle routes that go through some of the best photo-shooting spots on the island. Given a map of all the bicycle roads on the island and a list of the best photo-shooting spots to be included on each of the three planned routes (non-listed spots must not be included in the route), please write a program to plan each of the r routes so that the distance on each route is minimal. Note that each best photo-shooting spot may only appear at most once on the route.

Input

There are two parts to the input. The first part of input gives the information of the bicycle roads on the island. The first line contains two integer $n$and r, $n \leq 100$ and $r \leq 10$, indicating that there are n best photo-shooting spots on the island and there are r routes to be planned. The next n lines (line 2 through line $n+1$) contains $n \times n$ integers (n lines with n integers on each line), where the j^{th} integer on line i denotes the distance from best photo-shooting spot i-1 to best photo-shooting spot j; the distances are all between 0 and 10, where 0 indicates that there is no one-way road going from best photo-shooting spot i-1 to spot j.

The second part of input has r lines, denoting the r sightseeing routes to be planed. Each line lists the best photo-shooting stops to be included in that route. The integers on each line denote the recommended photo-shooting stops on that particular sightseeing route. The first integer on the line is the starting point of the route and the last integer is the last stop on the route. However, the stops in between can be visited in any order.

Output

Output r integers on r lines (one integer per line) indicating the distance of each of the r planned routes. If a route is not possible, output 0.

Sample Input

```
6 3
0 1 2 0 1 1
1 0 1 1 1 0
0 2 0 1 3 0
4 3 1 0 0 0
0 0 1 1 0 0
1 0 0 0 0 0
1 3 5
6 3 2 5
6 1 2 3 4 5
```

Sample Output

```
5
0
7
```

Shift Cipher

A *cryptosystem* is a method to convert a *message* into a *cipher*, which is difficult to understand by unauthorized people. Assume that both the message and the cipher are strings over the alphabet $\{a,\ b,\ \ldots,\ z\}$.

A *shift cipher* is a cryptosystem that shifts each character in the message by k positions. For example, if $k = 3$, then `a` is converted into `d`, `b` into `e`, ..., `x` into `a`, `y` into `b`, and `z` into `c`. The number k is called the *key* of the cryptosystem.

To make the cipher more difficult to understand, spaces and all punctuations are removed from the message before encryption. For example, assume that $k = 3$, the message:

```
we will meet at midnight.
```

is encrypted into the cipher:

```
zhzloophhwdwplgqljkw
```

Since there are only 26 different keys, given a cipher text, it is easy to convert each character back to the original message. However, by using a computer, it may not be trivial to insert spaces so that the original message can be recovered automatically.

For simplicity, we assume that a message is recovered if spaces are inserted into the text so that each word separated by spaces is a word in the dictionary. Given a cipher text, write a program to recover the message. You may assume that each cipher is less than 256 characters, and each word used in the message appears in the dictionary. The dictionary is located in

/usr/share/dict/american-english.

This dictionary is a text file; each line contains a word. There are words with capital or special letters in the dictionary. These words will not be used in our system. You may want to look at the dictionary before programming. It is not necessary to check if the sentence is grammatically correct or not. The answer will be considered correct if no adjacent words are single character and the average number of characters in the words is greater than 2.

Input File

The input data is a set of ciphers. Each cipher is written in a lines. A line containing only the character `0` signals the end of a test data.

Output Format

The output is the key k and the recovered message for each of the cipher. Print the solution of each test case in a line. If the solution is not unique, print the solution with minimum number of words. If there is no solution, print "`NO SOLUTIONS`".

Sample Input

```
zhzloophhwdwplgqljkw
lowder
0
```

Sample Output for the Sample Input

```
k=3: we will meet at midnight

NO SOLUTIONS
```

Problem G

A Scheduling Problem

Input File: pg.in

Time Limit: 3 seconds

There is a set of jobs, say $x_1, x_2, \cdots, x_n$, to be scheduled. Each job needs one day to complete. Your task is to schedule the jobs so that they can be finished in a minimum number of days. There are two types of constraints: *Conflict constraints* and *Precedence constraints*.

Conflict constraints: Some pairs of jobs cannot be done on the same day. (Maybe job x_i and job x_j need to use the same machine. So they must be done in different dates).

Precedence constraints: For some pairs of jobs, one needs to be completed before the other can start. For example, maybe job x_i cannot be started before job x_j is completed.

The scheduling needs to satisfy all the constraints.

To record the constraints, we build a graph G whose vertices are the jobs: $x_1, x_2, \cdots, x_n$. Connect x_i and x_j by an undirected edge if x_i and x_j cannot be done on the same day. Connect x_i and x_j by a directed edge from x_i to x_j if x_i needs to be completed before x_j starts.

If the graph is complicated, the scheduling problem is very hard. Now we assume that for our problems, the constraints are not very complicated: The graph G we need to consider are always trees (after omitting the directions of the edges). Your task is to find out the number of days needed in an optimal scheduling for such inputs. You can use the following result:

If G is a tree, then the number of days needed is either k or $k+1$, where k is the maximum number of vertices contained in a directed path of G, i.e., a path $P = (x_1, x_2, \cdots, x_k)$, where for each $i = 1, 2, \cdots, k-1$, there is a directed edge from x_i to x_{i+1}.

Figure 1 below is such an example. There are six jobs: $1, 2, 3, 4, 5, 6$. From this figure, we know that job 1 and job 2 must be done in different dates. Job 1 needs to be done before job 3, job 3 before job 5, job 2 before job 4 and job 4 before job 6. It is easy to verify that the minimum days to finish all the jobs is 4 days. In this example, the maximum number k of vertices contained in a directed path is 3.

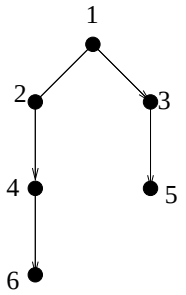


Figure 1: Example

Input

The input consists of a number of trees (whose edges may be directed or undirected), say $T_1, T_2, \cdots, T_m$, where $m \leq 20$. Each tree has at most 200 vertices. We represent each tree as a rooted tree (just for convenience of presentation, the root is an arbitrarily chosen vertex). Information of each of the trees are contained in a number of lines. Each line starts with a vertex (which is a positive integer) followed by all its sons (which are also positive integers), then followed by a 0. Note that 0 is not a vertex, and it indicates the end of that line. Now some of the edges are directed. The direction of an edge can be from father to son, and can also

be from son to father. If the edge is from father to son, then we put a letter "d" after that son (meaning that it is a downward edge). If the edge is from son to father, then we put a letter "u" after that son (meaning that it is an upward edge). If the edge is undirected then we do not put any letter after the son.

The first case of the sample input below is the example in Figure 1.

Consecutive vertices (numbers or numbers with a letter after it) in a line are separated by a single space. A line containing a single 0 means the end of that tree. The next tree starts in the next line. Two consecutive lines of single 0 means the end of the input.

Output

The output contains one line for each test case. Each line contains a number, which is the minimum number of days to finish all the jobs in that test case.

Sample Input

```
1 2 3d 0
2 4d 0
3 5d 0
4 6d 0
0
1 2d 3u 4 0
0
1 2d 3 0
2 4d 5d 10 0
3 6d 7d 11 0
6 8d 9 12 0
0
1 2 3 4 0
2 5d 0
3 6d 0
4 7d 0
5 8d 0
6 9d 0
7 10d 0
0
0
```

Sample Output

```
4
3
4
3
```

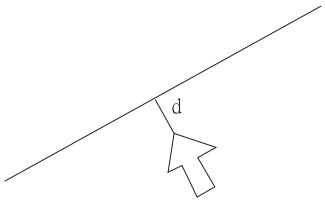
Problem H

Click the Lines

Input File: ph.in

Time Limit: 10 seconds

In many graphic drawing applications, you can click on a line object. A well-known approach to detect if a line is clicked by a mouse is to compute the distance between the click position and the line. For example, in Figure 1, the line is said to be *selected* if the distance d is less than a threshold D ($d \leq D$).



Give you a set of lines (with equation of the form $ax+by+c=0$), please compute the minimum clicks to select all the lines. For example, to select two lines, the minimum number of clicks is one, that is, you can make a click near the cross point of the two lines.

Note: In the test data, each line at least intersects with another line.

Input

The first line of input data begins with a number N– the number of test cases. Each test case begins with two integers L and D, where $L(L \leq 200)$ is the number of lines and $D(D > 0)$is the threshold for testing if a line is selected. The data of lines listed one by one. Each line is represented by ($ax+by+c=0$). The three *floating* numbers a,b,c, separated by space are given to represent the line.

Output

Please output the minimum number of clicks in a new line for each test case.

Sample Input

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

Sample Output

```
1
2
```

Problem I

Perfect Service

Input File: pi.in

Time Limit: 3 seconds

A network is composed of N computers connected by $N-1$ communication links such that any two computers can be communicated via a unique route. Two computers are said to be *adjacent* if there is a communication link between them. The *neighbors* of a computer is the set of computers which are adjacent to it. In order to quickly access and retrieve large amounts of information, we need to select some computers acting as *servers* to provide resources to their neighbors. Note that a server can serve all its neighbors. A set of servers in the network forms a *perfect service* if every client (non-server) is served by **exactly one** server. The problem is to find a minimum number of servers which forms a perfect service, and we call this number *perfect service number*.

We assume that $N(\leq 10000)$ is a positive integer and these N computers are numbered from 1 to N. For example, Figure 1 illustrates a network comprised of six computers, where black nodes represent servers and white nodes represent clients. In Figure 1(a), servers 3 and 5 do not form a perfect service because client 4 is adjacent to both servers 3 and 5 and thus it is served by two servers which contradicts the assumption. Conversely, servers 3 and 4 form a perfect service as shown in Figure 1(b). This set also has the minimum cardinality. Therefore, the perfect service number of this example equals two.

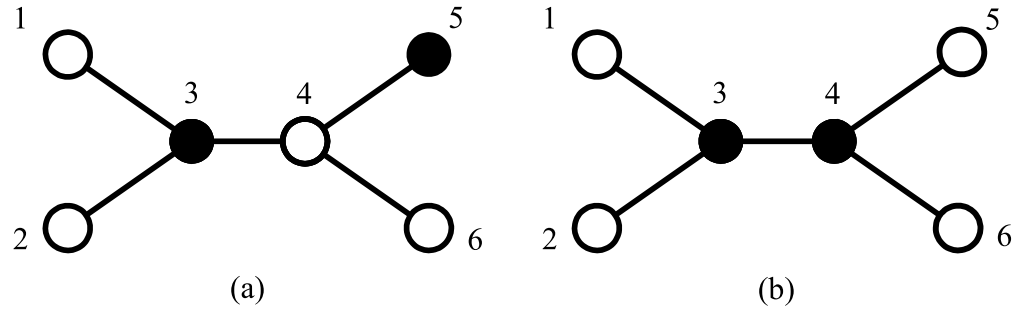


Figure 1

Your task is to write a program to compute the perfect service number.

Input File Format

The input consists of a number of test cases. The format of each test case is as follows: The first line contains one positive integer, N, which represents the number of computers in the network. The next $N-1$ lines contain all of the communication links and one line for each link. Each line is represented by two positive integers separated by a single space. Finally, a 0 at the $(N+1)$th line indicates the end of the first test case.

The next test case starts after the previous ending symbol 0. A -1 indicates the end of the

whole inputs.

Output Format

The output contains one line for each test case. Each line contains a positive integer, which is the perfect service number.

Sample Input

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

Output for the Sample Input

```
2
1
```