IEEE Extreme 9 – October 24, 2015.

Contents

Digit Fun!

Problem Statement

Recurrence relations are an important tool for the computer scientist. Many algorithms, particularly those that use divide and conquer, have time complexities best modeled by recurrence relations. A recurrence relation allows us to recursively define a sequence of values by defining the n^{th} value in terms of certain of its predecessors.
Many natural functions, such as factorials and the Fibonacci sequence, can easily be expressed as recurrences. The function of interest for this problem is described below.

Let $|A_n|$ denote the number of digits in the decimal representation of A_n. Given any numberA_0, we define a sequence using the following recurrence:
$A_i = |A_{i-1}|$ for $i > 0$
The goal of this problem is to determine the smallest positive i such that $A_i = A_{i-1}$.

Input Format

Input consists of multiple lines, each terminated by an end-of-line character. Each line (except the last) contains a value for A_0, where each value is non-negative and no more than a million digits. The last line of input contains the word END.

Output Format

For each value of A_0 given in the input, the program should output one line containing the smallest positive i such that $A_i = A_{i-1}$.

Sample Input

```
9999
0
1
9999999999
```

```
END
```

Sample Output

```
3
2
1
4
```

Explanation

The first input value is A_0 = 9999, resulting in A_1 = |9999| = 4. Because 4 does not equal 9999, we find A_2 = |A_1| = |4| = 1. Since 1 is not equal to 4, we find A_3 = |A_2| = |1| = 1. A_3 is equal to A_2, making 3 the smallest positive i such that A_i = A_{i-1}.
The second input value is A_0 = 0, resulting in A_1 = |0| = 1. Because 0 does not equal 1, we findA_2 = |A_1| = |1| = 1. A_2 is equal to A_1, making 2 the smallest positive i such that A_i = A_{i-1}.
The third input value is A_0 = 1, resulting in A_1 = |1| = 1. A_1 is equal to A_0, making 1 the smallest positive i such that A_i = A_{i-1}.
The last input value is A_0 = 9999999999, resulting in A_1 = |9999999999| = 10. Because 10 does not equal 9999999999, we find A_2 = |A_1| = |10| = 2. Since 2 is not equal to 10, we find A_3= |A_2| = |2| = 1. Since 1 is not equal to 2, we find A_4 = |A_3| = |1| = 1. A_4 is equal to A_3, making 4 the smallest positive i such that A_i = A_{i-1}.

Prediction Games

Problem Statement

In a *Prediction Game*, two or more players attempt to predict the score of a series of upcoming sporting competitions. Each player's predictions are then evaluated based on the sum of the categories listed below. These descriptions use the following variables:

- S_1: Actual score earned by team 1.
- S_2: Actual score earned by team 2.
- P_1: A player's predicted score for team 1.
- P_2: A player's predicted score for team 2.

Point System

- *Winner*: 10 points if they correctly picked the winner, i.e. the team with more points at the end of the competition.
- *Team 1 Score*: Maximum(0, 5 - $|S_1 - P_1|$). In other words, if they pick the correct score for Team 1, they earn 5 points. If they were off by one in their prediction, they earn 4 points. If they are off by two, they earn 3 points, and so on. If they are off by 5 or more points, they earn 0 points in this category.
- *Team 2 Score*: Maximum(0, 5 - $|S_2 - P_2|$).
- *Point Spread*: Maximum(0, 5 - $|(P_1 - P_2) - (S_1 - S_2)|$). In other words, if the player predicts the correct number of points by which one team beats another, they earn 5 points. If they are off by one, they are 4 points, and so on. If they are off by 5 or more points, they earn 0 points in this category.

Some, but perhaps not all, of the competition scores have been received. Your task is to provide a list of players who might end up with the highest point total when all scores are recorded.

Notes:

- Whenever we refer to a "game," we are referring to the "Prediction Game." When we refer to a competition, we mean the sporting event that teams are playing.

- No competition will end in a tie. In addition, the players are not allowed to predict a tie score as a final result for a competition.
- The smallest score possible in a competition is 0 points. The maximum possible score in a competition is 200 points.
- A player can earn points in the *Team 1 Score*, *Team 2 Score*, and *Point Spread* categories, even if they do not correctly pick the winner. For example, if a player predicts a score of 24-17 (i.e. 24 points for team 1 and 17 points for team 2) and the final result is 23-30, the player would earn 4 points for the *Team 1 Score* category (since the difference between the prediction (24) and the true score (23) is only one, he would get 5-1=4 points). He would receive no points for the *Team 2 Score* category or for the *Point Spread* since they are both off by more than 5 points. Similarly, if a player predicted a score of 10-9 and the actual score was 9-10, the player would earn a total of 11 points as follows: 4 points for*Team 1 Score* (again 5-1=4 points since the difference between the predicted and the actual score is 1), 4 points for *Team 2 Score* (for the same reason as with scoring analysis for Team 1), and 3 points for the *Point Spread* (since according to the formula the point spread is |(P1 - P2) - (S1 - S2)| = |(9-10) - (10-9)|=|-1-1|=|-2|=2, and thus the score obtained for this prediction should equal 5 - *Spread* = 5 - 2 = 3 points).

Input Format

The first line of input contains an integer t, 1 <= t <= 20, indicating how many test cases are in the input.
Each test case follows in the following format. First comes a line with two space separated integers, p and c, where p indicates how many players there are, and c indicates how many competitions are being played.
The following is repeated for each of the p players. The first line contains the player's name. Then follows c lines with two space separated integers. These are the predictions for this player. The

first of these prediction lines contains P_1 and P_2 for the first competition. The second contains the predicted scores for the second competition, and so on.
Finally there are c lines indicating the (possibly) partial results for the competition. The first competition's result is reported on the first of these lines, the second competition's on the second of these lines, and so on. Each of these lines will contain two space separated values. If the score for the team is known, the values will be integers giving the score. If the result of the competition has not yet reported, the line will read `? ?`.
Notes:

- Each player's name is unique. All of the names are comprised of an initial uppercase letter, followed by up to 30 lowercase letters.

- 2 <= p <= 20
- 1 <= c <= 1000
- Either both scores will be given or neither will be given. You will never be given a competition result where only one of the scores has not yet been reported.

Output Format

For each of the test cases, you should output a line containing, in alphabetical order, a space-separated list of the players who could earn the maximum number of points. If there is only one winner possible, then only the winner's name should be reported on a line of its own.

Sample Input

```
3
2 3
Alice
14 17
20 7
30 7
Bob
```

```
20 7
21 17
14 13
14 17
17 13
? ?
2 3
Dave
14 17
20 17
30 7
Chuck
20 10
27 17
30 7
? ?
27 17
30 7
3 1
Francis
14 7
Eve
10 21
George
7 30
0 1
```

Sample Output

```
Alice
Chuck Dave
Eve George
```

Explanation

There are three test cases.

Test Case 1

In the first competition, Alice predicted a score of 14-17, which was also the actual score. She earned 25 points for this prediction (10 points for the winner and 5 points in each of the other categories).

In the second competition, Alice predicted a score of 20-7, and the actual score was 17-13. She earned 12 points for this prediction, 10 points for the correct winner and 2 points for the score prediction for team 1 [5 - |20 - 17| = 2].

In the first competition, Bob predicted a score of 20-7, and the actual score was 14-17. He earned 0 points for this competition, because he was wrong about the winner, and he was off by more than 5 on both team scores and on the point spread.

In the second competition, Bob predicted a score of 21-17, and the actual score was 17-13. He earns 17 points for this prediction (10 points for picking the winner, 1 point for team 1 score, 1 point for team 2 score, and 5 points for correctly predicting the spread).

With these known results, Alice is leading Bob, 37 points to 17 points. Since both Alice and Bob picked the same winner for the third competition, there is no way for Bob to catch up. Therefore you would output just "Alice" for this test case.

Test Case 2

In this test case, Dave currently has 40 points: 15 points from the second competition (10 points for picking the winner and 5 points for picking team 2's score), and 25 points for picking the correct score for the third competition. Chuck has 50 points for picking the correct scores in the second and third competitions.

Suppose the result of the first competition was 10-17. Then, Dave would end with 57 points and Chuck would end with 50 points.

On the other hand, suppose the result was 30-0. Then Chuck would end with 60 points and Dave would end with 40.

Therefore, since either Chuck or Dave could end with the highest number of points in the game, you would output "Chuck Dave" (since "Chuck" comes before "Dave" in alphabetical order).

Test Case 3

Here all of the results are in, and Eve and George have the high score of 10 points each.

Inspirational Quote

Here's a quote to reward you for reading the entire problem, and inspire you to solve it!

"It's tough to make predictions, especially about the future." - Yogi Berra

Block Art

Problem Statement

The NeoCubist artistic movement has a very distinctive approach to art. It starts with a rectangle which is divided into a number of squares. Then multiple rounds of layering and scraping occur. In a layering round, a rectangular region of this canvas is selected and a layer of cubes, 1 cube deep, is added to the region. In a scraping round, a rectangular region of the canvas is selected, and a layer of cubes, again 1 cube deep, is removed.

The famous artist I.M. Blockhead seeks your help during the creation of this artwork. As he is creating his art, he is curious to know how many blocks are in different regions of the canvas.

Your task is to write a program that tracks the creation of a work of Pseudo-Cubist art, and answers I.M.'s periodic queries about the number of blocks that are on the canvas. Consider, for example, the following artwork created on a canvas with 5 rows and 15 columns or squares. The canvas starts without any blocks, like in the figure below. We label cells in the canvas based on the tuple (row,column), with the upper left corner being designated (1,1). The numbers in each cell represent the height of the blocks in that cell.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After adding a layer in blocks to the rectangle with upper left corner at (2,3) and a lower right corner of (4, 10), the canvas now looks like the following:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After adding a layer of blocks in the rectangle with upper left corner at (3,8) and a lower right corner of (5, 15), the canvas now looks like the following:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	2	2	2	1	1	1	1	1
0	0	1	1	1	1	1	2	2	2	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

If Blockhead were to ask how many blocks are currently in the artwork in the rectangle with upper left corner (1,1) and lower right corner (5,15), you would tell him 48.

Now, if we remove a layer of blocks from the rectangle with upper left corner at (3,6) and a lower right corner of (4, 12), the canvas now looks like the following:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1	0	0	1	1	1
0	0	1	1	1	0	0	1	1	1	0	0	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

If Blockhead were to ask how many blocks are now in the artwork in the rectangle with upper left corner (3,5) and lower right corner (4,13), you would tell him 10.

"Beautiful!" exclaims Blockhead.

Input Format

The first line in each test case are two integers r and c, 1 <= r <= 12, 1 <= c <= 10^6, where r is the number of rows and c is the number of columns in the canvas.
The next line of input contains an integer n, 1 <= n <= 10^4.
The following n lines of input contain operations and queries done on the initially empty canvas. The operations will be in the following format:
[operation] [top left row] [top left column] [bottom right row] [bottom right column]

[operation] is a character, either "a" when a layer of blocks is being added, "r" when a layer of blocks is being removed, and "q" when Blockhead is asking you for the number of blocks in a region.

The remaining values on the line correspond to the top left and bottom right corners of the rectangle.

Note: You will never be asked to remove a block from a cell that has no blocks in it.

Output Format

For each “q” operation in the input, you should output, on a line by itself, the number of blocks in the region of interest.

Sample Input

```
5 15
5
a 2 3 4 10
a 3 8 5 15
q 1 1 5 15
r 3 6 4 12
q 3 5 4 13
```

Sample Output

```
48
10
```

Explanation

This test case corresponds to the description given above.

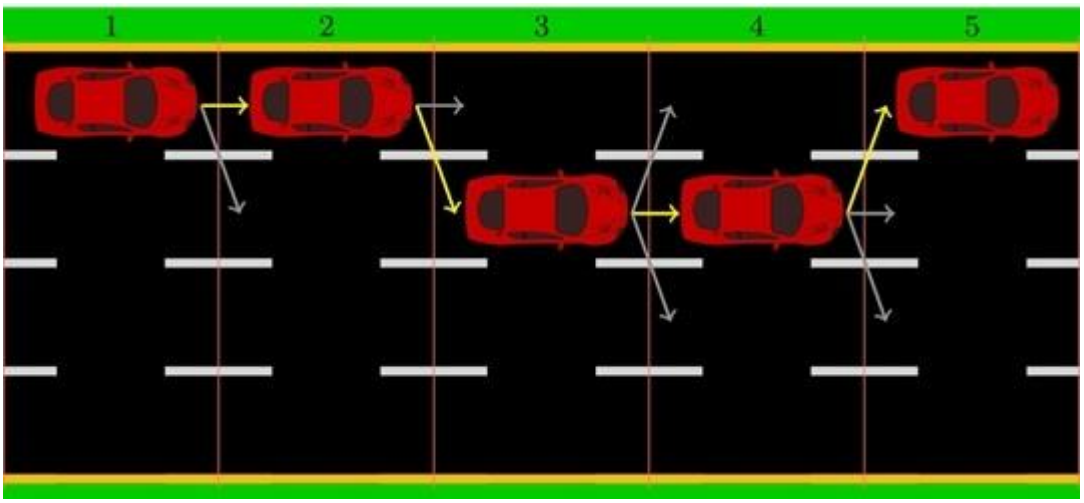
Xtreme Driving

Problem Statement

Alice, one of the IEEEXtreme participants, is on her way to her university to take part in this year's contest. To get to the university she has to drive on a four lane highway, but as the highway is very long she quickly becomes bored. She decides to practice for the contest by thinking about some problems related to the highway she's driving on. She comes up with the following problem:

Let's say that, in a single unit of time, her car, which is of unit-length, can perform one of the following three actions:

- Drive one unit forward, staying in the same lane
- If the car is not on the left-most lane, drive one unit forward and switch to the lane on the left
- If the car is not on the right-most lane, drive one unit forward and switch to the lane on the right



If the highway is K units in length, in how many ways is it possible to drive through the highway, provided that she starts on the first unit of the highway at the left-most lane, and ends at the last unit of the highway, also at the left-most lane?

As she's been training very hard for the contest, it doesn't take her long to come up with a solution to this problem. But all of a sudden, out of nowhere, a large cow appears in front of her car, and she just barely manages to avoid crashing into it. She got a bit too distracted thinking about the problem... But this incident gives her an idea: what if the highway had a set of stationary cows that the car must avoid crashing into?

Unfortunately she doesn't have time to think about this version of the problem as she just arrived at the university and the contest is about to start. When the contest starts, she is very surprised to see that the problem she was thinking about is just like one of the problems presented in the contest. What a coincidence! Again her hard practice pays off and she quickly solves the problem. But the real question is, can you?

Input Format

Input begins with two integers, K, the length of the highway, and N, the number of cows standing on the highway, subject to the following constraints:
2 <= K <= 10^{18}
0 <= N <= 100
N <= 4(K - 2)
Then follow N lines. The i^{th} line contains two integers describing the location of the i^{th} cow, x_i, the lane on which the cow is standing (1 being the left-most, and 4 being the right-most), and y_i, the unit on which the cow is standing, subject to the following constraints:
1 <= x_i <= 4
1 < y_i < K
Notice that no cows are on the first or last unit of the highway. It is also guaranteed that no two cows share the same position.

Output Format

You are to output the number of ways to drive from the left-most lane at the first unit of the highway to the left-most lane at the K^{th} unit of the highway subject to the rules described above, and

the additional constraint that the car must not drive to a position occupied by a cow.

As the number of ways can be quite large, please output the answer modulo $10^9 + 7$. Note that this is the same as the remainder when dividing the number of ways by $10^9 + 7$.

Sample Input

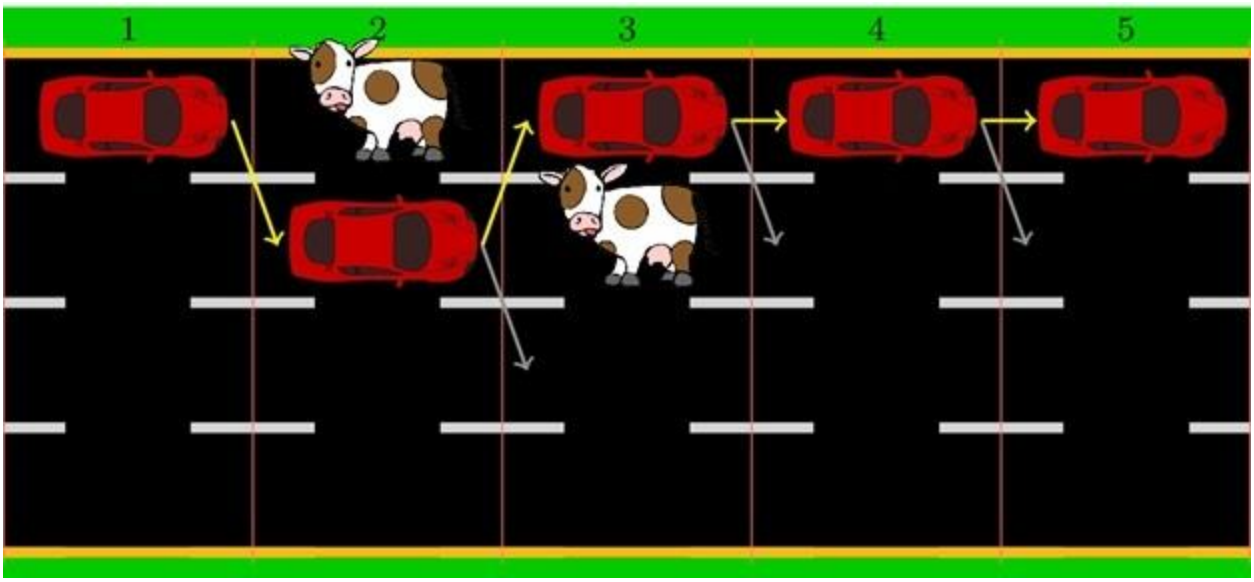
```
5 2
1 2
2 3
```

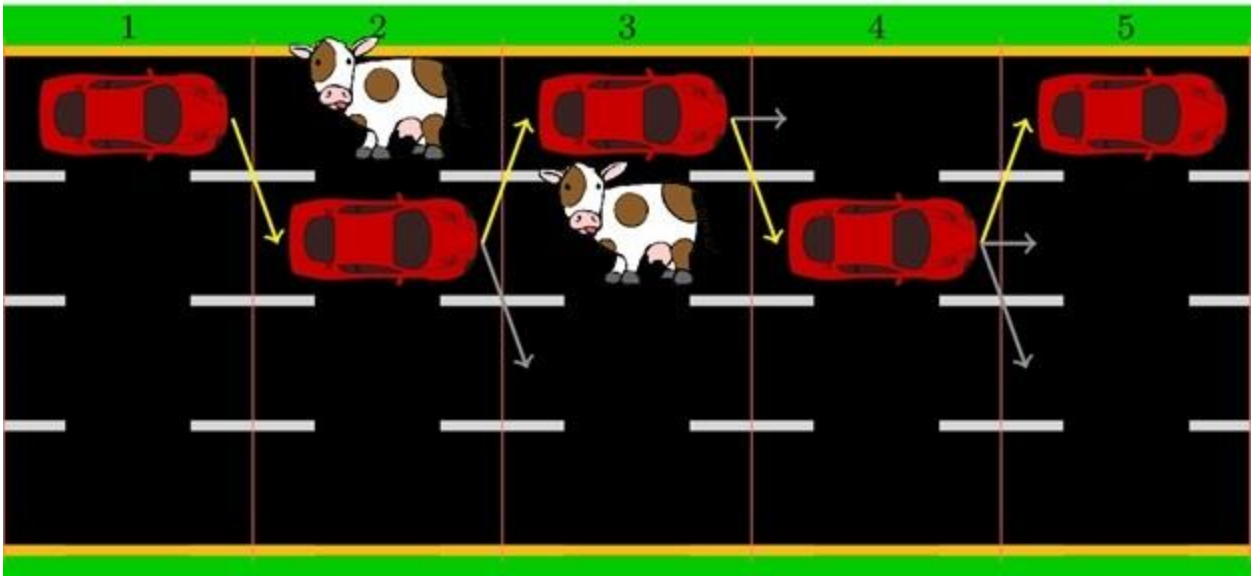
Sample Output

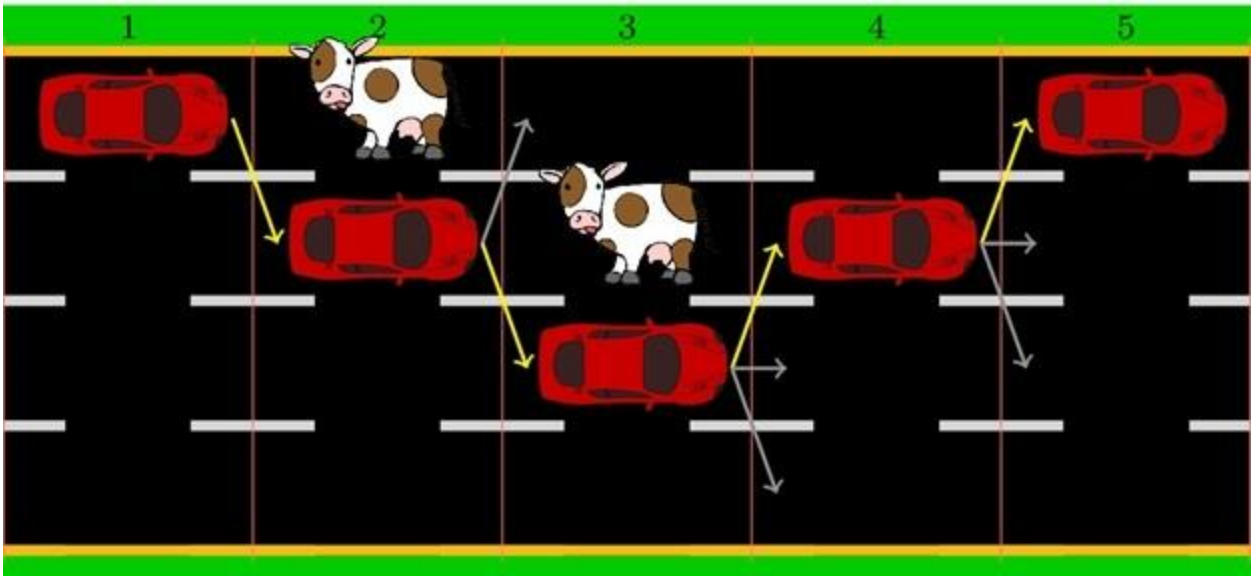
```
3
```

Explanation

In this example there are three ways to drive through the highway, and they are shown in the following figures. Notice that the car can drive in between cows (as long as other driving rules are fulfilled), but it must not drive to a unit occupied by a cow.







Note: Two more sample test cases are available if you click on the "Run" button.

Xtreme In Security

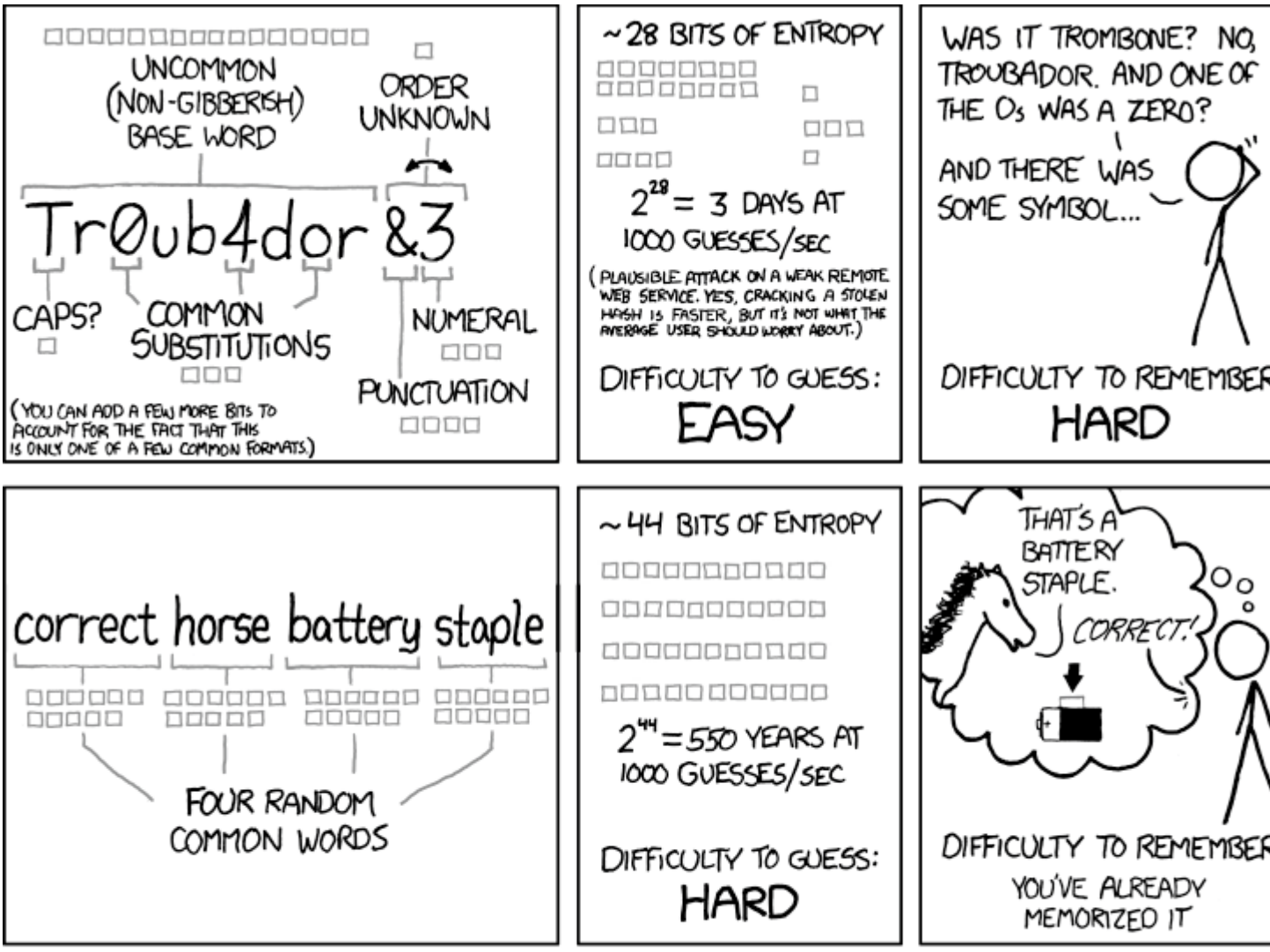
Problem Statement

This Challenge is sponsored by the IEEE Electron Devices Society.
The Xtreme In Security Corp. has devised a password-based authentication system for their new operating system. They have made some unfortunate design decisions that make their system vulnerable to attack.

The system stores in a file a salted and peppered password hash. All of the passwords use the same salt "IEEE" and the same pepper "Xtreme". When the user is enrolled, they supply a password p. The system checks to make sure that p is less than 20 characters long, and that it contains only lowercase letters and numbers. Then the salt is prepended to the password and the pepper value is appended. The resulting value, using UTF-8 encoding, is then put through the SHA-256 hash algorithm, and then base64 encoded. This base64 encoded value is stored in a password file, along with the user's name.
Note: Even though the password contains only lowercase letters and numbers, the salt and the pepper contain some uppercase letters. The SHA-256 hash algorithm is case sensitive.

When the user is authenticated, a similar process occurs. The user is prompted for a username and a password. Then the salt is prepended to the supplied password and the pepper is appended. The resulting value is hashed with SHA-256, and base64 encoded. If the resulting value matches what is stored in the password file for the specified user, then the authentication is successful.

Your task is to break as many of the hashed passwords as possible to try to convince the system designers to improve their approach. Here is some inspiration from XKCD:



Credits: XKCD

Input Format

The input will consist of a single salted, peppered password hash. This hash will be derived from a single password.

The hash will be chosen from the following set of values:

```
/PtjJboZGlsmTovvyOhBOoTVnQKUP/gJXxjLAW9Lppw=

05HwH93tksb69U1ifesCQuYFP+gKPVH2L6W8JeBdXy0=
```

```
0BkyqI3NHyjh0m20wNt6txW08dglSMP4/qzUEezq4Aw=
1mT5cdKRz4BbfMdc8LAdnxfjsGO4lV0k0/V1IHtidmY=
28AdfW0JHmCP4TbieGON8dafRaFpUgpzuX2bHZN6WsM=
3hoie8omUyvM/9Qfx9dKfoptlwemYe2os8aohTGzoyw=
3uDaglIdYUn11AadEhELBjE15A0L6hAaWnZmjCGtt+M=
4D4NstIYSjVN826Q+SXUDDmXglJplpYWiJYf9rt7H8U=
7FCsEXCDTAxyLh3EPnNx7YrJ44SzehQYv3GmPSA6pWk=
81X3NN5JgTuGgqq3ErF0eL/l/wZFYkCwur6fZ06L2Us=
8FzGA/nS7XizLrAVOr/FoeKSq4gaoQRq+kpBKNXHIzc=
8OtpJ+E1XHv2RDsKIEwJc9KUFwPRzaqeHJ735Er6AVE=
8cdgZu9dBOrcTBMqElM+y9Vh5FTBRQ7n9EGa/4qVHUk=
8esDbw8ZVmUuUMEy2Scf5qGiZYiykevrvKpq2bVYHj4=
9DS4orbhPFbjJcosEqQg+eg0Si5qSOnftfHiqK8sYug=
9XDFIu4RPH0EL5XR+5VYILJ3UFAyltpfjONJKp/vcLk=
B2E/K/DywbLEKutOKpS8HxHFrZwucwac4KjzYgsXg3g=
BJQeqlV+4ejv0je5GpekzGdHHWHL5nnrbtD/170LZCA=
BjQiH/A2FUNHlUwhBi8NWmj3HmhmAh6Ag0kRyVSaVo8=
BwbAsOqPsxteVCpAwIjrhYogsUS1bF/bLns2QdcLYUI=
CYDZabjeyTAwcEDEcvrX83514UmpjzvQUQ68DIZ/PXg=
CornANxoZ5FJnlhwHmK42CDXf3h6jFr3g73YIRuoymQ=
DDa2TJX20pPsNftfyJ3s6LBwSMSR3EADZGDxW2wThbs=
FFXy3vru2D8rTWZRlh9lSMvtEusfWgO17OmJCTQTECs=
FGkqFC/jLDqDZ10fql1nGw7AQNWioOrZ3ydEaJyXBwo=
Fz+Y0H/R2rMZlc1C88Yx0A0xluYnVTinlw5qaSx8vWQ=
GcJMWMDF6+f+onf6oi1FbpnN7dVrFEZnlXtHqmaygs4=
GsXTQM0w+Clb1c9B7n28mADU2quLeI1n91KTyBboeHI=
HLnuqQmCYetzrau3frCDEpZ52QCIby108gugsmwAwQ0=
HWv9gx+GL/6g+0b0eOc+1Z/8BHse91/5T/DdiDwEknU=
IDB/pOthrWobzapJ/N8HsraNhwfbExAa2uusdiKHFFI=
IXYqlHbVeONERbxFe8SaEPEEKex45EihiC/l8CR52kk=
Idvs4Al9YZsqPG8xkSxVqb6MOVhbw5k+qtF8UZKYVE8=
```

```
IxQxcFXR51q8h8FLblPhYfUR30lIAt6hX8TjZWVa/GE=
JCmqBN0MsW13tEmsQPYWg9Fj25MUrqFYvSK2arxTt0A=
JOXxLH/i8i+fxDIWP2cts5Si/5En1A8M3s/vy6Aadic=
KudA8vCEQdGaaCSxotpAcluXnVPS3MAZPkwI/lVupak=
LGZEfbUr/tMREpJtsao/uuewcJXApmgfHDbh1zzfdhU=
LZIzmWEqXDPJsnKttFGRaG/jUhHrbTEKt1XCO6XbdME=
Lq0kV5M0HDSgB4m5KZbbn6BYRNlkKfaaAr3/11ueopY=
Lqxt1UjT1ecV6ucgYn+yrGSUTxPWkZgdDtbygGwC/BA=
MHQTB1MSsvhBxMpdRUiaM9Uj1QxU7zYq3FqDlW2HT2s=
MKewBZryb2l36Y0tDyx+WuVeXUGfiSzcJplm9y1w9m0=
N5aunKGHeN2WETLXLzfhfCxAfkwtGU/imziiTF3t7oY=
NmrOUzHxKSfNT8UJ1YXbRL8I+HCAb+glJ0bBXcHfagE=
NnSS+AuW1Z6zytSfqaiIVp4xxHHe70Av+IdhDlkoItQ=
O9L1ZWYwuzgaImTjOwuogXfpC+C44zzcDhpt08LjR5c=
R/ye+L9W+l6hyZ/v1POsWYboEGemIisARL8ohUvfBLI=
R1v9fEb9VrZuU5xiYTKTqhHF03VtXg7+KtfFHPkQuCQ=
RVfhsLovxa+/6tWgeSBASIIkzXkVtDPT4yYvjboHhIk=
Rz38Ng2qI3mPkaRB6uDoCYmmfzbVTCzpt2sG1o+TZqo=
S98FBzlv2vMVP/q+23m1wrHMJIrcy1rhoQGy338c4Bs=
SzraQWWasG5ZO1tJq16DqU/7M/o/WRiAWRl1aFFvwr8=
TInfNYwXvofBA+9QIe3+XEfDpO5ER+R+Jn3BOshhZWU=
VDkcRd54BhYlK5Wg2PPDa/jzGrSkMepGIv6Tw3I2ksc=
VjFTqTEY27V8lK2yCvhLhYm2Brh9bN1vWckVaevsiiY=
Wau4ReopjFKk8SYYNq5lIBL+Rgg8aBR6h9UgTIv2u7I=
XtEWsXf16y6Bc7vQxDy7hwRdBVWo3dV9C6CDVSf4PLs=
Y5b6UztMueFYIFMl4a61jlD/ZhFG74/rVn8XaqqU+8c=
YollqBlewcxE/kF6PKvv0r1CLZkKx82657bB0eQbiK0=
YzSqlQTtq5j+Kd+hW1ISgBW0mn61vsQsxNeipq0sYCo=
aJIH+q0YjYZCpierKtbue5JDtZSF8tKxVKuHYUPQ65k=
bi6rh2HgTbJxR2GOTNWZLlxiiWZVnObptGj0KqOCSYo=
cX7VyMvSYhuRvEfAUb3uNh8kmjpNFg0tatUPN562iOg=
```

```
eSCTiCrzHPbngPu3F4ivPkLUv7MqLUlmWAhA4UO975Y=
eUqkcVCbgIx1bGhhmnN5MxJFJhVruINmG65TjT/EQ1k=
gE7PseB3mspPtYG3JROzT9FeqTfPFYQvBF2SJD9V19Y=
gMi4hC4o7Fmv6yIrU48BVy8I1khXwkaD36G7bWiZHeo=
h84yifAWGLj9sakEqxZ3QEjkXL42AoScP3L5Tdevm98=
hEGKCiTZSA5x560hodRoIBBTE8pv4sP1VXG4D0fXWcI=
ix+0IJIpLpHeSHEII+Q/IVY+FlRXn3xMA0ey6UITi34=
j2GTUqtqZpotY8wF16zkvnbdCLTnX3oOZ30SjQUnIUk=
jtUg00EsmzzFkk8JgKg3OpkmPRpN9xbwsdNXQSPczwo=
k1J9Dv3EI442CO6A2FGN1H8JgFO2kjNBvkjDR0WIvkA=
kuRpkIh9kaNz6XvG8U6GO/IARH/SqhRnTiJbZHXC0yQ=
lFgqRrqTz1WXmO22u+Is1ZmWWUtuYrTJigsSB7I9NHI=
lUfxHX9xH2aOHheMMqQF+f5BNh97avew2uOwEN3B7HE=
m0IeuugWDOl9cFUFRYJhouCBT39T0dpp1xBOKPqHP94=
mgA5kgALstQpGUBp4vZ8oiz0P4jjAGl0wjgls6kQyMA=
nMAwaDYvEIAwoqtqWMpBAWdhuRgRq/fmwWbRM7cOMIU=
opBtoC66YDRbLNqZAAu2FIeKfF0HMOGHCOCPYeNHdx4=
ot/igM+me4e3UTT731qcBkSAcToyADMCddr7i7LCWGo=
qrkd/8imuRtiDb9N1w2hQRxJAkdx3Wqh1HVXPS7dym8=
r6BN0tdyAYZD0Nmc7bfV0WRcFBb1A2WIPPKHVRG59k8=
rdALvOYVhA3hnUTBlaQXigWBSgMYzGTreSKyMoAoKfw=
rjwtKqkPc7cfQ+zZ+E9c+fzQYhRvhVtaKEFb+srIHcQ=
rwvmTDiJxIEETbsngvpxYGwZZK+FGo7527odGuQUjtQ=
socJeO02bT2w+XZUrLoopbZvQ1lRhDfE88GVrJQ8p+g=
tDdmKQpMiVDFA1YdblkHSFzL4Z9UIQ9FSouf3TybOu0=
tojYiNtlWmq+7r1dSvxDk3W5at/NMAi1uxCHY61WAKk=
tqpGCBzhR+0ONFk1sBiHPhz+kRiXmY3CGdUXqnMJwLg=
uAZthS7b4ySZtWpM9pJ7ulYnhFdpFABpR2iPRQEmff0=
uCG9dSBejCOrZWsX7+u8G340p74s8lDS/El8MIeOyMo=
ugcIIpDID0R1uFqBAcN3PNXhwlhen77GdAccFgpbs+A=
usg8BTtSfewL5M3OVg91TJCTc5vONLqgUCC/Si1Grsg=
```

```
vs2sCU8qG0pDYQfcjlPzDzvcbJnhP1OgFRcXP4i3ffw=
wEtqAs8JHjicWnXshAWF5Sg6NoswuG9qeJ7USw7YD7c=
y+zbMpySKY+WF97KkgRQ+tBpM7iTqTj9guWmGJcqfyA=
y1R6JQiUzUovgtdrvCkbeQAyMhFoupzhI5ZuQVPfCgw=
zqKPAOt5ziHSeRxc0TgUZF3rJxzBHAKdJeccvt3F7Jg=
```

Output Format

The output should consist of a single alphanumeric string, which hashes to the corresponding value given in the input.

Note: You may not be able to break all of the hashed passwords. If you are given a hash that you cannot break, you may output an arbitrary string. You will earn partial credit for this problem for each password that you do crack!

Sample Input

```
tDdmKQpMiVDFA1YdblkHSFzL4Z9UIQ9FSouf3TybOu0=
```

Sample Output

```
password1
```

Explanation

For all passwords, the salt value is "IEEE" and the pepper is "Xtreme". First, we can take the SHA-256 hash of the UTF-8 encoded string `IEEEpassword1Xtreme`. If we then base64 encode the result, the value equals the desired value:`tDdmKQpMiVDFA1YdblkHSFzL4Z9UIQ9FSouf3TybOu0=`.

Taco Stand

Problem Statement

Joe has been hired to make tacos at a series of baseball games. He wants to calculate the maximum number of tacos he can make based on the available ingredients. He always insists on fresh ingredients, so any leftover ingredients on a given day will be thrown away.

His ingredients are:

- Taco shells - every taco gets exactly one of these
- Meat
- Rice
- Beans

His recipe is to take one taco shell, then add exactly two of the ingredients: meat, rice, and beans. So, for example, one taco might have meat and rice, while another taco might be made with rice and beans. However, a taco *cannot* have two of the same ingredient. For example, Joe will never make a taco with two servings of meat. Your task is to write a program to calculate the maximum number of tacos Joe can make each day, given the amount of ingredients he will have.

Input Format

The first line of input is an integer n, 1 <= n <= 1000, specifying how many days Joe will be making tacos.
The following n lines contain 4 space-separated integers in the format:
$s\ m\ r\ b$
where s is the number of shells available, m is the amount of meat, r is the amount of rice, and$b$ is the amount of beans, each expressed in terms of the number of tacos they could make.
Note: s, m, r, and b are all non-negative integers less than 10^9.

Output Format

The output file is exactly n lines long, each line containing an integer specifying the maximum number of tacos Joe can make with that day's ingredients.

Note: There is a newline character at the end of the last line of the output.

Sample Input

```
2
5 3 4 1
1 9 9 9
```

Sample Output

```
4
1
```

Explanation

On the first day, Joe can make a total of 4 tacos - 3 meat and rice tacos and 1 rice and bean taco.

On the second day, Joe only has one shell, so he can make 1 taco with any two of the ingredients.

Bon Voyage (or Muddle over Cruise Cabins)

Problem Statement

The TransContinental Ocean-liner Company is celebrating their centennial anniversary. As a part of the celebrations, they launched an exciting contest for a free, around-the-world cruise on one of their ships. As part of the contest entry, passengers select two cabins in which they would like to stay. If they are a winner, they will be given one of the two cabins.

Interest was enormous, and great many people applied to be a part of the voyage. Winners will be decided on a lottery. The company does not have any control over whom all will win the final lottery or obtain the tickets, but only on how many tickets will be distributed.

Jerry, the tour manager, is finding it difficult to find the maximum number of prizes that can be awarded. He must make sure that each potential winner can be accommodated in one of the cabins that they specified when they entered the contest.

Your job is to find the maximum number of prizes that can be awarded subject to this constraint. Help Jerry to find a solution for his problem and earn some real good points for yourselves!

Input Format

The input begins with an integer T, 1 <= T <= 15, indicated the number of test cases.
Each test case consists of the following:

- A line containing an integer N, 2 <= N <= 200, where N is the total number of rooms on the ship.
- A line containing an integer M, 0 <= M <= 500, where M are the number of people who signed up.
- Next come M lines, the i^{th} line containing the pair of cabins selected by the i^{th} entrant in the contest. The cabin choices will

be two space-separated integers, each falling between 1 and N (inclusive).

Output Format

For each test case, you should output, on a line by itself, the maximum number of tickets that can be awarded such that every winner can be housed in one of the two cabins that they have selected.

Sample Input

```
5
6
4
1 2
1 2
3 4
5 6
6
5
1 2
1 2
1 2
3 4
5 6
4
5
1 2
1 3
1 2
2 3
3 4
5
11
```

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

Sample Output

```
4
2
3
3
2
```

Explanation

The sample input consists of 5 test cases.

In the 2nd test case, 3 people opted for the same 1 or 2 cabins, and in order to prevent the worst case of all the 3 of them getting the ticket, we can give out at most 2 tickets.

In the 3th test case, we can give out 3 tickets and still make sure that regardless of who gets a ticket, any winner can be housed in a cabin of their choice.

Palindromic Moments

Problem Statement

This challenge is sponsored by Morgan Claypool Publishers.

Bob, Hannah, and Otto like to celebrate palindromic dates. A palindromic date is one in which the numbers in the date read the same forwards and backwards. They were very happy with 2015, which has 15 palindromic dates:

- 5th of January, October, November and December in `day/month/year` format: 5/1/15, 5/10/15, 5/11/15, 5/12/15
- Eleven days in May listed in `month/day/year` format: 5/1/15 and 5/10/15 through 5/19/15

Note: These dates are palindromic in the sense that, if the forward slash delimiter (/) is disregarded, the numbers in the date read the same forwards and backwards.

While they wait to celebrate the next palindromic date on November 5th, they have decided to they would like to start celebrating palindromic moments. They define a palindromic moment as a palindromic string obtained by formatting a date-time combination using one of the following date prefixes and one of the following time suffixes:

- Date prefix:

  Combinations of month, day, year: `Mdyy`, `MMdyy`, `Mddyy`, `MMddyy`, `Mdyyyy`, `MMdyyyy`,`Mddyyyy`, `MMddyyyy`
  Combinations of day, month, year: `dMyy`, `dMMyy`, `ddMyy`, `ddMMyy`, `dMyyyy`, `dMMyyyy`,`ddMyyyy`, `ddMMyyyy`
- Time suffix:

  Combinations of hour, minutes, seconds: `hmmss`, `hhmmss`, `Hmmss`, `HHmmss`

Where

- `M` is the month of the year as a number, e.g. January = "1" and October = "10"
- `MM` is the month of the year as a number with a leading zero if the month of the year is one digit long, e.g. January = "01" and October = "10"
- `d` is the day of the month as a number, e.g. first = "1" and fifteenth = "15"
- `dd` is the day of the month as a number with a leading zero if the day of the month is one digit long, e.g. first = "01" and fifteenth = "15"
- `yy` is the last two digits of the year, e.g. 2000 = "00", 12015 = "15"
- `yyyy` is the complete year as a number, e.g. 512 = "512", 2000 = "2000", 12015 = "12015"
- `h` is the hour in (AM/PM), e.g. 1 pm = "1", 12 am = "12"
- `hh` is the hour in (AM/PM) with a leading zero if the hour is one digit long, e.g. 1 pm = "01", 12 am = "12"
- `H` is the hour in the day, i.e. 0 <= H <= 23, e.g. 1 pm = "13", 12 am = "0"
- `HH` is the hour in the day with a leading zero if the hour is one digit long, e.g. 1 pm = "13", 12 am = "00"
- `mm` is the minute in the hour, 5 past the hour = "05", 30 past the hour = "30"
- `ss` is the second in the minute, 5 past the minute = "05", 30 past the minute = "30"

For example, during this competition on October 24, 2015, there are four palindromic moments:

- 1:01:42 AM (using the `dMyy` prefix and the `hmmss` suffix): `24101510142`
- 1:42:01 AM (using the `Mdyy` prefix and the `hmmss` suffix): `10241514201`
- 1:01:42 PM (using the `dMyy` prefix and the `hmmss` suffix): `24101510142`
- 1:42:01 PM (using the `Mdyy` prefix and the `hmmss` suffix): `10241514201`

Note that for October 24, the `MMddyy`, `Mddyy`, and `MMdyy` prefixes are equivalent to the `Mdyy`prefix. Similarly for this date, the `ddMMyy`, `ddMyy`, and `dMMyy` prefixes are equivalent to the`dMyy` prefix.

Input Format

The input begins with an integer t, 1 <= t <= 20, on the first line, which indicates the number of test cases.
Following this, there are t test cases on separate lines, each containing a single integer $Year$, 10 <= $Year$ <= 10^6.

Output Format

For each test case, you should output, on a line by itself, the number of palindromic moments that occur during the year given in the test case.

Notes:

- We are interested in the total number of unique palindromic moments in a given year. If a moment is palindromic when using two or more different suffix/prefix combinations, this instance counts as a single palindromic moment. It should not be double counted.

  For example, `November 11, 2010 1:11:11 am` is palindromic when combining the`MMddyy` prefix and `Hmmss` suffix or the `MMddyy` prefix with the `HHmmss` suffix. Even though these combinations form two different palindromes, `11111011111` and `111110011111`, this should count as a single palindromic moment.
- You must account for the different number of days in each month and for leap years. However, you should ignore leap seconds.

Sample Input

```
2
2015
2016
```

Sample Output

```
5626
3761
```

Explanation

Test Case 1

There are 5626 palindromic moments during 2015. For example, the following list shows the 36 palindromic moments on January 1, 2015:

- 12:51:11 AM (using the `Mdyy` prefix and the `Hmmss` suffix): `111505111`
- 1:02:11 AM (using the `Mdyyyy` prefix and the `hmmss` suffix): `11201510211`
- 1:10:10 AM (using the `MMddyy` prefix and the `hmmss` suffix): `01011511010`
- 1:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111515111`
- 2:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111525111`
- 3:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111535111`
- 4:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111545111`
- 5:11:01 AM (using the `Mddyy` prefix and the `hmmss` suffix): `1011551101`
- 5:11:10 AM (using the `MMdyy` prefix and the `hmmss` suffix): `0111551110`
- 5:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111555111`
- 6:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111565111`
- 7:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111575111`
- 8:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111585111`

- 9:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `111595111`
- 10:21:01 AM (using the `Mddyyyy` prefix and the `hmmss` suffix): `1012015102101`
- 10:21:10 AM (using the `MMdyyyy` prefix and the `hmmss` suffix): `0112015102110`
- 11:51:11 AM (using the `Mdyy` prefix and the `hmmss` suffix): `1115115111`
- 1:02:11 PM (using the `Mdyyyy` prefix and the `hmmss` suffix): `11201510211`
- 1:10:10 PM (using the `MMddyy` prefix and the `hmmss` suffix): `01011511010`
- 1:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111515111`
- 2:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111525111`
- 3:11:01 PM (using the `Mddyy` prefix and the `Hmmss` suffix): `10115151101`
- 3:11:10 PM (using the `MMdyy` prefix and the `Hmmss` suffix): `01115151110`
- 3:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111535111`
- 4:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111545111`
- 5:11:01 PM (using the `Mddyy` prefix and the `hmmss` suffix): `1011551101`
- 5:11:10 PM (using the `MMdyy` prefix and the `hmmss` suffix): `0111551110`
- 5:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111555111`
- 6:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111565111`
- 7:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111575111`
- 8:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111585111`

- 9:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `111595111`
- 10:21:01 PM (using the `Mddyyyy` prefix and the `hmmss` suffix): `1012015102101`
- 10:21:10 PM (using the `MMdyyyy` prefix and the `hmmss` suffix): `0112015102110`
- 10:51:11 PM (using the `Mdyy` prefix and the `Hmmss` suffix): `1115225111`
- 11:51:11 PM (using the `Mdyy` prefix and the `hmmss` suffix): `1115115111`

In a similar way we may calculate the remaining palindromic moments for the rest of 2015, resulting in the reported overall sum of 5626 unique palindromic moments for the year.

Test Case 2

There are 3761 palindromic moments during 2016.

Mr. Pippo's Pizza

Problem Statement

Mr. Pippo wants to start a new pizza shop. Everything about his pizzas is unique — the recipe is unique, the taste is unique, and even the shape of pizzas is unique. Instead of having a round shape like every other pizza, Mr. Pippo makes his pizzas in polygon shapes. For example, he can make his pizzas in a triangular shape or in a pentagonal shape.

Before serving a pizza, Mr. Pippo cuts it into triangular pieces. However, there are different ways he can cut the pizza. For example, a pentagonal pizza can be cut in five different ways as shown in the following figure. Each day, Mr. Pippo chooses a particular shape which can only be cut in an odd number of ways. Note that all the five cuts in the figure happen to be rotationally symmetric, but each of them is considered distinct.

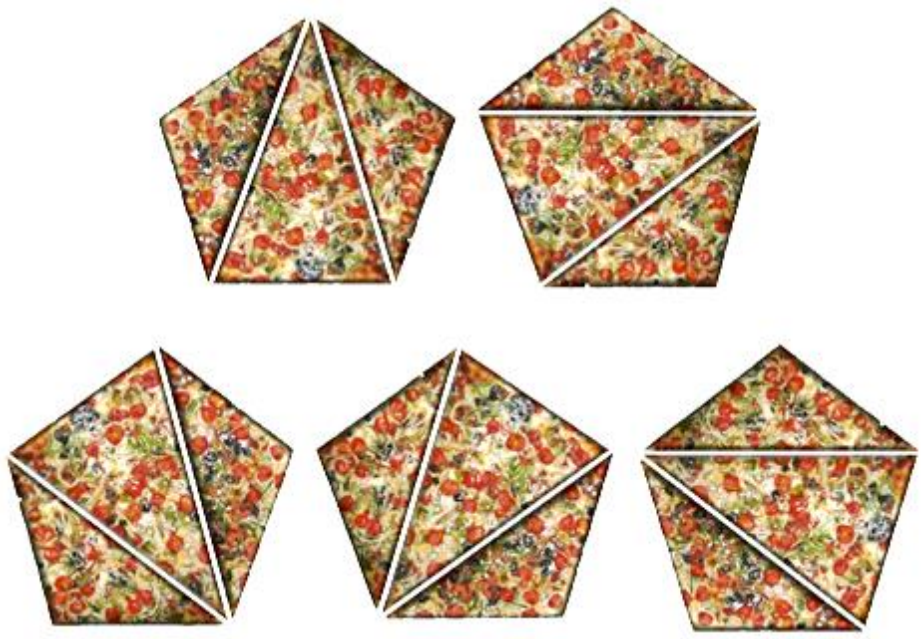


Figure: Different ways a pentagonal pizza can be cut

Your task in this problem is to determine the shape of the pizza. Given the number of ways the pizza can be cut, you have to determine how many sides the pizza has.

Further clarification regarding the ways a pizza can be cut is given below:

- A pizza can only be cut by connecting two vertices,
- Two cuts on the pizza cannot cross each other, and
- For an n-polygon there would be exactly (n-3) cuts which divide the pizza into (n-2) pieces.

Input Format

There will be up to 100 lines given where each line represents one test case. For each test case, the number of ways the pizza can be cut will be given. The number will be always odd and can be up to 308 digits long. The input is finished when end-of-file is reached.

Output Format

For each test case, print on a single line the number of sides the pizza has.

Sample Input

```
1
5
```

Sample Output

```
3
5
```

Explanation

For the first line of input, there is only one way to "cut" the pizza if the pizza is a triangle. This one way consists of no cuts at all.

For the second line of input, if there are five ways to cut the pizza, the pizza must be a pentagon, as shown in the figure in the problem description.

Zoom In

Problem Statement

Some years ago, we had terminals that were capable of supporting only ASCII characters. We would like your help to construct a program, which given an input string and specific printing rules, produces the same text in a bigger layout.

Input Format

On the first line of input is an integer n, 1 <= n <= 100, representing how many columns each character will use when printed "zoomed-in".
The next line contains an integer m, 1 <= m <= 100, representing how many rows each character will use when printed "zoomed-in". Note that n and m are not necessarily equal.
The third line contains an integer k, 3 <= k <= 95, which indicates how many characters may need to be translated.
Following these first lines, are k descriptions of the "zoomed-in" characters, formatted as follows:

- On a line by itself, a single character, which has an ASCII value of between 32 and 126, inclusive
- m lines, each containing n characters, that give the "zoomed-in" representation of the character on the previous line

Following the descriptions of the zoomed in characters, is an integer number x, 1 <= x <= 500.
Finally there are x lines, each containing a string of up to 2,000 characters, and ending with a new line. The characters in this string will be chosen from the set of k characters previously specified.
Notes:

- We don't know if k sets (i.e. the descriptions of the k "zoomed-in" characters) are given in a sorted or random order.
- The "zoomed-in" version of an empty string is m blank lines (i.e. lines with only a newline character).

Output Format

For each of the x strings, you should output the "zoomed-in" version. Each string should begin on a newline.
Note: You should perform only the transformation that is specified. You should not add any space between your printed letters that is not in the transformation.

Sample Input

```
4
4
3
H
H  H
H%%H
H%%H
H  H
i
 ()

 II
 II
!
 II
 II
 II
 ()
1
Hi!
```

Please note that the line after `()` and before `II` contains 4 spaces.

Sample Output

```
H  H ()  II
H%%H     II
H%%H II  II
H  H II  ()
```

Explanation

*For clarity, we will add dashes where the spaces would appear in the output in this explanation.*According to the input, each character will use 4 rows and 4 columns, and there are 3 characters that may be translated.
A capital H ('H') should be translated as

```
H--H
H%%H
H%%H
H--H
```

A lower-case i ('i') should be translated as

```
-()-
----
-II-
-II-
```

An exclamation mark ('!') should be translated as:

```
-II-
-II-
-II-
-()-
```

We are then asked to print the "zoomed in" version of the string "Hi!". The output would be the following (with spaces where the dashes are located):

```
H--H-()--II-
H%%H-----II-
H%%H-II--II-
H--H-II--()-
```

High Score

Problem Statement

This challenge is sponsored by the IEEE Women in Engineering.
Find an input which will make the provided Java program give the highest output. An equivalent Python v.3 program is also provided. The programs are accessible via the "Programs" tab in the ribbon above with options: "Problem", "Submissions", "Leaderboard", "Discussions", and "Programs"

The best solution will get a full score, while others will receive an exponentially decaying score: losing 1% of the score for each unit decrease in the output.

Input Format

None.

Output Format

Your program should produce a legal input for the Java (or Python equivalent) program listed in the "Programs" tab.

The first line of of your solution's output should be a non-negative integer N representing the number of lines to follow.
The following N lines should each contain exactly 10 non-whitespace characters.

Sample Output

```
2
Abcde_-!@#
0#T234<>?,
```

Explanation

No input will be provided to your program. Your program should produce a fixed output according to the instructions provided above which will be then automatically provided as input to the given Java

(or Python equivalent) program. For example, in this case, we may assume that you have written a program that outputs the 3 lines provided above as the *Sample Output*. This program outputs a value of 2 for N, and then two lines follow with 10 characters each.
When this output is provided as input to the Java program, it outputs a value equal to 998. As it turns out, this is not a very good input. The score for our solution would be:

$0.99^{MaxScore - 998}$
where $MaxScore$ is the maximum possible score for this challenge. $MaxScore$ is large enough that this rounds to zero.
Note: The scores will be rounded to two decimal places.

Threesome Poker

Problem Statement

Help IBM puzzlemaster to test answers to May's 2015 challenge:

```
Three people are playing the following betting game.

Every five minutes, a turn takes place in which a random player rests and the
other two bet against one another with all of their money. The player with the
smaller amount of money always wins, doubling his money by taking it from the
loser.

For example, if the initial amounts of money are 1, 4, and 6, then the result of
the first turn can be either 2,3,6 (1 wins against 4); 1,8,2 (4 wins against 6);
or 2,4,5 (1 wins against 6). If two players with the same amount of money play
against one another, one of the two players would win, and the game will
immediately end for all three players after that turn.

The task for the challenge is to find initial amounts of money for the three
players, where none of the three has more than 255, and in such a way that the
game cannot end in less than one hour.

In the example above (1,4,6), there is no way to end the game in less than 15
minutes.  One possible sequence of moves is:

(1,4,6)
(1,8,2), after player 2 plays player 3 in turn 1
(2,7,2), after player 1 plays player 2 in turn 2
(4,7,0), after player 1 plays player 2 in turn 3, and we arbitrarily choose
player 2 as the winner

Note:  The first turn is played after five minutes, so the game above, with its 3
turns, takes 15 minutes.  Furthermore, the maximum number of turns that can be
completed in one hour is 11.
```

Input Format

Three space-separated, positive integers representing a potential solution to the challenge.

Output Format

The program should output either the word "Ok" if the answer is correct (i.e. the game needs an hour or longer to end), or in case

there exists a series of turns in which a player wins the game in less than an hour, then the program should output at most 12 lines showing the amounts of money that each player has. In particular, in the latter scenario, the first line of the output should be identical to the given input, and then each subsequent line should represent a possible state of the game after a turn is played, with the previous line considered as a starting state. Since by definition in this scenario there will be a winner, one of the amounts in the last line of the output should be zero.

If there are several ways for the game to end in less than an hour, find the shortest one, i.e. the one with the fewest turns. If there are several shortest ones, choose the one in which the winning player and losing player in each round is first in lexicographical order. Please refer to the explanation of the sample input and output for more information about this tie breaking procedure.

Sample Input

```
3 27 8
```

Sample Output

```
3 27 8
6 24 8
6 16 16
6 32 0
```

Explanation

The first line of the output is identical to the given input. Then, in the first turn of the game (second line of the output), player 1 beats player 2. In the second turn (third line of the output), player 3 beats player 2. In the final round (final line of the output) player 2 beats player 3.

This game-play can be summarized in a single line comprised of space separated pairs that describe the winner and the loser of each turn in the format: ([winner],[loser]). For the example above, this string summary would be: "(1,2) (3,2) (2,3)" since at first player 1 beat player 2, then player 3 beat player 2 and finally the game ended with player 2 winning over player 3.

Of course, other game-plays could also result in the same result (i.e. have a player winning the game in less than an hour), some with more turns than the one described above, but some as equally as short (in term of number of turns). For example, such another equally short series of turns is the following:

```
3 27 8
3 19 16
6 16 16
6 0 32
```

However, this game-play would be summarized by the following string: "(3,2) (1,2) (3,2)" since at first player 3 beats player 2, then player 1 beats player 2 and finally the game ended with player 3 winning over player 2.

Note though that the last solution has a string representation that comes later in lexicographical order compared to our first solution (i.e. solution "(1,2) (3,2) (2,3)" comes before "(3,2) (1,2) (3,2)" in lexicographical order).

Similarly, another possible series of turns would be:

```
3 27 8
3 19 16
6 16 16
6 0 32
```

This series would be represented by the string: "(3,2) (1,2) (2,3)", which also comes after "(1,2) (3,2) (2,3)" in lexicographical order.

Indeed if you listed all possible games that end in three turns, "(1,2) (3,2) (2,3)" would be lexicographically smaller than all other string representations of games. Thus the expected output should be:

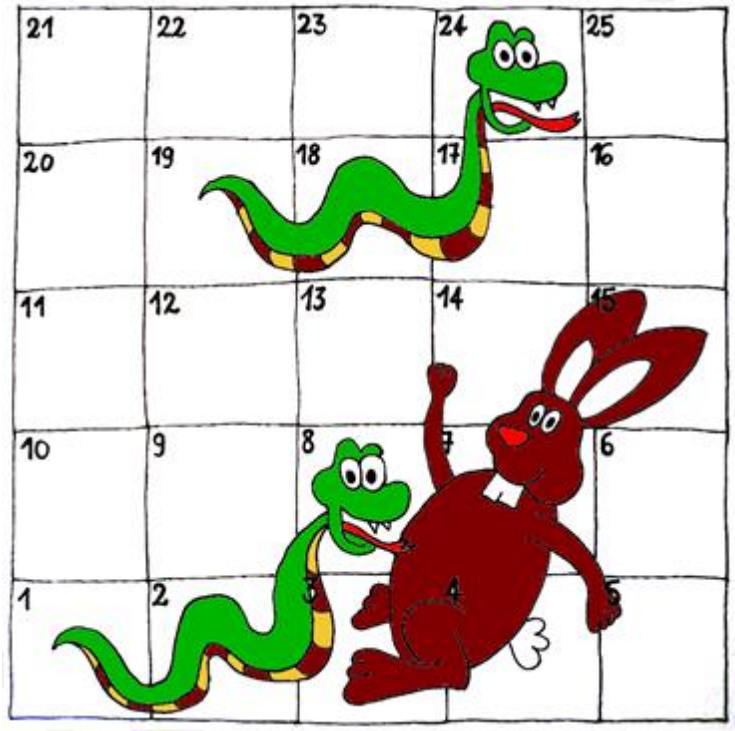
```
3 27 8
6 24 8
6 16 16
6 32 0
```

Note: There are two additional sample test cases. You can view these cases after clicking on the "Run" button.

Snakes & Bunnies

Problem Statement

A group of engineers are playing the Snakes and Bunnies board game. In this game, each player has one game piece, which moves according to dice rolls. The game-board is composed by a grid of N x N numbered squares, which are ordered row-wise from the bottom-left to the top-right (N is always odd). Some "bunnies" and "snakes" are depicted on the board, each connecting two squares. An example of a 5x5 board is shown in the following figure.



Each player starts off the board, next to the starting square. The gameplay is divided in rounds and players play in sequence, which is the same every round. Rules are simple:

1. Two standard 6-faced die are rolled by the player and his/her game piece moves forward on the board, following the square's numbers. The piece is advanced by a number of squares equal to the sum of the die.

2. If the dice roll is a double, then the player has an additional turn just after the current one. Note that the additional turn begins after applying the additional rules below. The additional turn follows the same rules of standard turns, except that only one die can be rolled.

3. If a square is already taken by another player’s token, then the current player’s token moves forward to the next square not occupied by a token.

4. If the final position of a player’s token is a square with the head of a snake, then it must be moved backwards to the square corresponding to the snake’s tail. Similarly, if the token ends on a square with bunny’s feet, it goes to the top of the bunny’s ears (both ears of the rabbit will always point to the same cell). No square has two or more snake's heads/tails or bunny's ears/feet on it; there is at most one special drawing for each cell. Moreover, the last square is always free of drawings.

5. Game ends when a player arrives at the last square or when a player can move over the last square (for example if the player is on the second-to-last square and rolls 3+4). In this latter case, the player stops on the last square and wins.

Note that infinite loops can happen while a player moves: this is the EVIL CYCLE case! When it happens, the game ends and the player in the evil cycle wins the game.

For example, consider a board in which squares 10 and 11 are already occupied, and there is a slide from square 12 back to square 10. If a player lands on square 10, they would advance to square 11, since square 10 is already occupied. Square 11 is also occupied, so they would advance to square 12. Here they take the slide back to square 10, and then repeat the moves. They would continue to move between squares 10, 11, and 12 forever!

Your task is to find the final position of every player's token, given as input the number of players, the game-board configuration, and the sequence of dice rolls.

Notes:

- The given sequence of dice rolls may not always lead a game to an end. There will be no extra dice rolls after a game has ended. There will always be sufficient dice roles for a player to complete their turn.
- If a player lands on the tail of snake or the ears of a bunny, the player does not make any special moves.

Input Format

The first line of the input contains the integer N ($1 < N < 100$ and N is odd), which is the dimension of the game-board.
The following N lines contain the game-board configuration: each line contains N characters, and each character represents a square of the board. The character '-' represents a normal square, i.e. one with no snake heads/tails nor bunny feet/ears depicted on it; digits (0-9) represent bunnies and letters (a-z) represent snakes. There can be at most 10 bunnies and 26 snakes, and each represented by an appropriate pair of digits or letters. Given a pair of identical letters representing a snake in two numbered squares, the head of the snake are located in the square with the higher number, and the tail is located in the square with the lower number. Given a pair of identical digits representing a bunny in two numbered squares, the feet of the bunny is located in the square with the lower number, and the ears are located in the square with the higher number.
So, for example, the game-board of the figure above could be represented as follows:

```
---a-
-a---
----1
--b--
```

```
b-1--
```

After the game board representation, the following line contains an integer M (2 <= M <= 10), which is the number of players.
Then, the sequence of dice throws follow, each dice result is represented by a single line containing one integer between 1 and 6, inclusive.

Output Format

The output is a single line containing M integers separated by a blank space. The first integer is the final position on the game-board of the first player (i.e. the one who rolled the dice first), the second integer is the final position of the second player, etc.
In case of evil cycle, the output is a single line containing the string "PLAYER x WINS BY EVIL CYCLE!", where x is the player number (1 to M).

Sample Input

```
5
---a-
-a---
----1
--b--
b-1--
2
6
2
2
1
3
2
1
2
3
4
1
1
5
```

Sample Output

```
13 25
```

Explanation

The board in this test case is the board in the figure below.

21	22	23	24 **SNAKE HEAD**	25
20	19 **SNAKE TAIL**	18	17	16
11	12	13	14	15 **BUNN**
10	9	8 **SNAKE HEAD**	7	6
1 **SNAKE TAIL**	2	3 **BUNNY FEET**	4	5

Note that in the board above there is one bunny, represented by the number '1'. It's feet are at square 3 and its ears are at square 15. There are two snakes. The snake indicated by the letter 'b' has a head at square 8 and a tail at square 1. The snake indicated by the letter 'a' has a head at square 24 and a tail at square 19.

Next we are told that there are 2 players.

First, player 1 rolls a 6 and a 2, advancing to square 8, at the head of the snake. Here, he moves back to the snake's tail, ending at square 1.

Next, player 2 rolls a 2 and a 1, advancing to square 3. Since 3 is at the feet of a bunny, she advances to square 15.

Next, player 1 rolls a 3 and a 2, advancing to square 6.

Next, player 2 rolls a 1 and a 2, advancing to square 18.

Next, player 1 rolls a 3 and a 4, advancing to square 13.

Next, player 2 rolls double 1's, advancing to square 20. Since the player rolled doubles, player 2 rolls a single die and gets a 5. She then advances to square 25.

Since player 2 has reached the final square, the game is over.

Note: If you click on the "Run Code" button, you will be able to see an additional sample test case with an example of an EVIL CYCLE.

IEEE State of Mind

Problem Statement

Finite State Machines (FSMs) can be used to model systems whose output depends on the current state and its inputs. The states can represent many different systems like the on/off state of traffic lights at an intersection, the valve state of thrusters on a spacecraft, the click/key combinations entered by a user for a keyboard shortcut, or the send/receive steps in a communication protocol.

An FSM is defined by a set of states, inputs, and outputs. The states represent a known configuration of the system at a specific time. Transitioning to a new state depends on the current state and the inputs provided. This will in turn also produce an associated output for the system.

One of the uses of FSMs is to design and analyze logic circuits with memories like flip-flops. Your task is to write a program to visualize the timing diagram relating the inputs and outputs while printing the current state number to help analyze the circuit.

For the purpose of the program, you can assume that there are N states with M inputs. States are labeled with numbered subscripts in the range S_0 to S_{N-1}. The inputs are each labeled with a single uppercase character in the range 'A' to 'J', inclusive. You may assume that no letter or state number is skipped. Each state has P uniquely defined transitions to other states (the transitions are unique in the sense that no set of inputs will trigger more than 1 transition). A transition is only taken when the input conditions are satisfied. Transitions are evaluated at the beginning of each of the T discrete time steps.

Input Format

Input begins with a line containing N and M, where $1 \leq N < 100$ and $1 < M < 10$.

Then there are N lines, each describing the output of the system and the transitions from this state. The first line contains a

description of state S_0, the second line describes S_1, etc. These lines all have the following format:

$StateOutput\ P\ VariableExpression_1/Dest_1\ VariableExpression_2/Dest_2\ \ldots VariableExpression_P/Dest_P$

where

- $StateOutput$ is either a 0 or 1, representing the value that is output when the system is in this state
- P is the number of transitions from this state
- $VariableExpression_i$ is a comma-delimited list of $Variable=Value$ tokens. For a transition $i$to be activated, all of the input variables given in the $VariableExpression_i$ must have the values listed in this expression
- $Dest_i$, $1 <= i <= P$, is a destination state number, in the range [0..N), for transition i

Following the description of the states, is a line containing two integers, T and I, where T gives the number of timesteps ($1 <= T < 1000$), and I gives the number of the initial state ($0 <= I <$ N). The input ends with $M * T$ lines that provide the inputs for all M variables in each of the Ttimesteps. The first M values provide inputs for the M variables during the first timestep, the second M values provide inputs for the M variables during the second timestep, and so on.

The transitions should be interpreted as follows. Assume that we are in a state S_j. Assume further that there are four inputs, and the current input provided for the current timestep is (1,0,1,0). We map these inputs to the variables, starting with letter 'A', so A = 1, B = 0, C = 1, and D = 0. We would then look for transitions defined on the line corresponding to state S_j. Since transitions must be unique, only one of the following $VariableExpressions$ could appear in the list of expressions: `A=1`, `B=0`, `C=1`, `D=0`, `A=1,B=0`, `A=1,C=1`, `A=1,D=0`, `A=1,B=0,C=1`, etc. If such a transition is the i^{th} transition specified, the new state would be $Dest_i$. If there are no matches, the system remains in the current state.

Output Format

The output consists of a waveform. The waveform is represented by underscores, '_', for the number 0, and asterisks, '*', for the number 1.

A maximum of 16 ticks are printed together, where each time tick is 3 characters wide. The line numbers are labeled with the signal names as shown in the example.

As shown in the example below, on the first line of the grouping is the text `Tick #X` where `X`is replaced by the number of the first tick in the grouping.
Next come waveforms for each of the variables, in alphabetical order, one per line. On these lines the variable name is output, followed by five spaces, and then the waveform.

Then the waveform of the system output is displayed. This line begins with the word `OUT`, followed by 3 spaces, and then the waveform.
Finally, on the last line the numeric value of the system output is displayed. This line begins with the word `STATE`. The numeric values of system output should always be aligned with the third column of the respective tick.
If not all ticks have been displayed, then a blank line should be output, followed by the next group of ticks in the same format.

Sample Input

```
2 3
1 2 A=1,B=1,C=1/1 A=1,B=0/0
0 1 A=0/0
20 0
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
0 0 0
```

```
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
0 0 0
0 0 1
0 1 0
0 1 1
```

Sample Output

```
Tick #1
A      _____________************_____________************
B      _______******_______******_______******_______******
C      ___***___***___***___***___***___***___***___***___***
OUT    ____________________  ____________________  ____
STATE    0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1

Tick #17
A      _____________
B      _______******
C      ___***___***
OUT    ************
STATE    0  0  0  0
```

Explanation

The sample input executes a parity check FSM. There are N=2 states with M=3 inputs. Since there are 3 inputs, these inputs are labeled `A`, `B`, and `C`.

State 0, S_0, outputs the value `1` (logic high). There are two transitions originating from this state. The first transition goes to S_1 when input `A`=1, `B`=1, and `C`=1. The second transition goes to S_0 when input `A`=1 and `B`=0 (that's a self-loop to the same state). `C` can take any value in the second transition.

S_1 outputs the value `0` (logic low). There is only one transition originating from this state defined in the input. This transition goes to S_0 when `A`=0.

Note that although not defined, there are implicit transitions to satisfy the other cases that are not explicitly described in the input. The implicit transitions are all self-loops to the same state.

Prior to tick t=1, we are in the initial state, S_0. The input is set to `A`=0, `B`=0, and `C`=0, therefore we take the implicit self-loop transition and stay in S_0 and output the value associated with the state, i.e. `1`.
When we start ticks t=2 through t=7, we are still in S_0. In each of these cases, we take the implicit loop to the same state and nothing changes in the output.
In tick t=8, the inputs, `A`=1, `B`=1, and `C`=1, trigger the transition to state S_1. This changes the output of our logic circuit to a `0`.
In tick t=9, the input, `A`=0, trigger the transition to state S_0. This changes the output of our logic circuit to a `1`.
The same pattern of inputs is repeated a few more times. Tick 17 begins on a new line because we limit the output to 16 ticks per line.

Car Spark

Problem Statement

John is a computer programmer and is highly known for his achievements in his field. In addition to being a passionate software professional, he is also passionate about motorcars and motorbikes.

So, after ending his successful and lengthy software career, he decides to take up his passion. He starts an organization by the name "Car Spark" (CS). CS is an organization from which you can rent luxury cars of your choice on an hourly rental basis.

CS would like to accept bookings for the weekend in advance, and then decide which bookings to process based on the profits that would be earned. When placing an order, customers quote the amount that they are willing to pay for that vehicle during that particular timespan. Since a car can only be given to one customer during a particular time period, CS must be careful about which bookings to process.

Initially CS has only one vehicle available for rent. To be the first hire for CS, you must develop a program to maximize revenue on bookings for this vehicle.

Input Format

Input begins with a single integer T, 1 <= T <= 100, which denotes number of test cases.
Each test case begins with a single integer N, 1 <= N <= 2000, which is the number of bookings John received.
The remainder of the test case consists of N lines containing three integers B_s, B_e, and A_i each separated by a space, where B_s is the booking start time, B_e is the booking end time, and A_i is the amount that the customer is willing to spend for the entire booking. Note that 0 <= B_s < B_e<= 48 and 1 <= A_i <= 100000.
Note: The car may only be rented during the weekend, meaning from 12:00 AM on Saturday to 12:00 AM on Monday. Since the two

days in the weekend have 48 hours, 12 noon on a Sunday would be the (24+12) 36th hour. Similarly, if the booking start time is 10:00 PM on Saturday and the booking end time is 12:00 AM on Sunday, then B_s would be 22 and B_e would be 24.

Output Format

You are to output a single line for each test case, giving the maximum revenue John can make from the orders he received.

Sample Input

```
2
4
1 2 100
2 3 200
3 4 1600
1 3 2100
3
1 10 2000
2 5 100
6 9 400
```

Sample Output

```
3700
2000
```

Explanation

For the first test case, for the time slot 1-3 maximum revenue John can make is 2100 (Max(100+200, 2100)) and for slot 3-4 he can make 1600. The maximum total revenue is 3700 (2100+1600).

Similarly for second test case, the maximum revenue he can generate is 2000.

Dictionary Strings

Problem Statement

Gopal is preparing for a competitive exam and he has to prepare many topics for it. To remember the concepts better he identified a set of words from each topic. He prepared dictionaries for each of these topics with the set of identified words so that he can refer to them easily.

While recollecting the topics Gopal sometimes could not remember to which dictionary a certain word belongs. After all the hard work, Gopal didn't want to lose marks due to this confusion. So he requested his friend, Govind, to help him identify a way to check if a word belongs to a dictionary.

Govind, being a very good friend of Gopal, wants to help him do better in the exam. So, after some thought, he finally came up with a solution.

For each dictionary, a string is chosen from which all the words can be made by selecting a subset of the characters from the string and rearranging them. (It is not necessary that the characters are consecutive and/or in the same order as in the string). They called this string a `Dictionary String`. When confused about to which dictionary a word belongs, Gopal can check if the word can be extracted from the `Dictionary String` for that dictionary.
To qualify as a `Dictionary String`, all the letters needed to explicitly form each word of the dictionary must be present in the string. You cannot reuse letters. Thus, the string `aab` is not a Dictionary String for a dictionary containing the word `aaa` since this word needs 3 `a`'s whereas the candidate Dictionary String has only two `a`'s.
To help Gopal memorize the Dictionary Strings better, Govind inserted extra characters in some of the Dictionary Strings that appeared harder to memorize. To distinguish those strings from others he calls a string without any extra characters, a `Perfect Dictionary String`.

Govind would like your help in verifying his program. For a set of words in a dictionary, you should indicate whether a string is a perfect dictionary string and/or a dictionary string. If a word is not a dictionary string, he would like you to tell him the minimum number of characters needed to convert the string to a dictionary string.

Notes:

Some of the test cases are very large, and may require you to speed up input handling in some languages.

In C++, for example, you can include the following line as the first line in your main function to speed up the reading from input:

```
std::ios_base::sync_with_stdio (false);
```

And in Java, you can use a BufferedReader to greatly speed up reading from input, e.g.:

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
// Read next line of input which contains an integer:
int T = Integer.valueOf(reader.readLine());
```

Input Format

Input begins with a single integer T, 1 <= T <= 100, which denotes number of test cases.
Each test case begins with a line, which contains 2 space-separated integers D and S. Drepresents the number of words in a dictionary, and S represents the number of potential dictionary strings to be checked. Note that 1 <= D, S <= 100.
Next follows D lines, each containing a word in the dictionary.
The remaining S lines in the test case each contain a potential dictionary string.
Notes: The words in the dictionary and the potential dictionary strings will consist of only lower-case letters. The lengths of these strings are greater than or equal to one character and less than or equal to 40,000 characters.

Output Format

For each of the S potential dictionary strings, you should output a line with two values separated by a space in the following format:
A_1 A_2
Where

- A_1 is either `Yes` or `No` denoting if a string is a Dictionary String or not.
- If A_1 is `No`, then A_2 is the minimum number of characters needed to make the string a Dictionary String. If A_1 is `Yes`, then A_2 is `Yes` if the string is a Perfect Dictionary String, and `No` otherwise.

Sample Input

```
1
5 3
ant
top
open
apple
lean
anteplop
antelope
penleantopan
```

Sample Output

```
Yes Yes
No 1
Yes No
```

Explanation

For the sample input, there is only one test case with 5 words in it and 3 strings to be checked.

`anteplop`: contains all the words from the dictionary and no extra characters. So it is both a Dictionary String and a Perfect Dictionary String. Hence, the output is `Yes Yes`.

`antelope`: the words “apple” cannot be made from this string. So it is not a Dictionary String and is missing 1 character (‘p’) to become a dictionary string. Hence the output `No 1`.
`penleantopan`: all the words of the dictionary can be made from this string but it also contains extra characters that are not required to build the words of the dictionary. So it is a Dictionary String but not a Perfect Dictionary String.

Light Gremlins

Problem Statement

This challenge is sponsored by Eta Kappa Nu.

There are a group of gremlins that live in a long hallway in which there are a series of light switches. At the beginning of each night, all of the light switches are off. Then, one at a time, each gremlin does the following:

- The gremlin chooses a prime number p, that has not been chosen by any previous gremlin that night.
- The gremlin runs down the hallway flipping every p^{th} switch.

The owner of the hallway, who is very concerned about his electricity bill, has asked you to determine how many switches are on at the end of the night.

Note: no two gremlins will choose the same prime number.

Consider the following example where the hallway has 21 switches and there are three gremlins. At the beginning of the night, all switches are off, as shown in the figure below.

Switch:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
State:	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off

The first gremlin chooses the prime number 7, and flips the 7^{th}, 14^{th}, and 21^{st} switch. Now the configuration is:

Switch:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
State:	Off	Off	Off	Off	Off	Off	**On**	Off	Off	Off	Off	Off	Off	**On**	Off	Off	Off	Off	Off	Off	**On**

The second gremlin chooses the prime number 13, and flips just the 13^{th} switch, because there is no 26^{th} switch. Now the configuration is:

Switch:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
State:	Off	Off	Off	Off	Off	Off	**On**	Off	Off	Off	Off	Off	**On**	**On**	Off	Off	Off	Off	Off	Off	**On**

The last gremlin chooses the prime number 3. It flips the 3rd, 6th, 9th, 12th, 15th, 18th, and 21stswitch. Note that when he flips the 21st switch, it is turned back off. The final configuration is:

Switch:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
State:	Off	Off	On	Off	Off	On	On	Off	On	Off	Off	On	On	On	On	Off	Off	On	Off	Off	Off

For this example, you would report that there are 9 lights on at the end of the night.

Input Format

The input begins with an integer t, 1 <= t <= 20, on a line by itself. Then follow t lines, each describing a test case that you must evaluate. The test cases have the following format:

```
[switch] [n] [prime_1] [prime_2] ... [prime_n]
```

Where

- [switch] is the number of switches in the hallway, 1 <= [switch] <= 10^{18}
- [n] is the number of gremlins who live in the hallway, 1 <= [n] <= 24
- The prime number chosen by the i^{th} gremlin is given by [prime_i]. All primes are greater than or equal to 2 and less than 10^4.

Output Format

For each test case, you should output a single integer that indicates how many switches are on at the end of the night.

Sample Input

```
3
21 3 7 13 3
20 1 31
30 3 2 3 5
```

Sample Output

```
9
0
15
```

Explanation

The first test case corresponds to the example given in the Problem Definition, which as described above results in 9 "on" switches at the end of the night.

In the second test case, there is a single gremlin, who chooses the prime 31. The hallway consists of only 20 switches, so there is no 31st switch. Thus, no switches are turned on.

The last test case consists of a hallway of length 30, and three gremlins. The action of the gremlins is as follows:

- The first gremlin flips switches {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30}. All of these switches were previously off, so they are now on.
- The second gremlin flips switches {3, 6, 9, 12, 15, 18, 21, 24, 27, 30}. Of these, {6, 12, 18, 24, 30} were previously on, so they are now off. This results in the following switches being on: {2, 3, 4, 8, 9, 10, 14, 15, 16, 20, 21, 22, 26, 27, 28}.
- The third gremlin flips switches {5, 10, 15, 20, 25, 30}. Of these, {10, 15, 20} were previously on, so they are now off. This results in the following switches being on: {2, 3, 4, 5, 8, 9, 14, 16, 21, 22, 25, 26, 27, 28, 30}.

Thus, there are 15 switches on at the end of the night.

Chocolate 2

Problem Statement

This challenge is sponsored by the IEEE University Partnership Program.

Let us consider a rectangular bar of chocolate containing n times d chunks of chocolate. Each chunk can be uniquely identified by a pair of integers (i_1, i_2), where each coordinate $i_1$indentifies the row of the chunk and i_2 identifies the column. i_1 can take values between 1 and$n$, inclusive, and i_2 can take values between 1 and d, inclusive. The chunk at the top left corner of the bar is identified by (1,1).

One wants to eat the whole bar using a very elaborate method. The principle is that when the chunk with coordinates (i_1,i_2) is selected to be eaten, all remaining chunks with coordinates (j_1, j_2) such that j_1 >= i_1 and j_2 >= i_2 are eaten at the same time. The question we ask is:

Given n and d, how many different ways of eating the whole bar are there?

Note: The timeouts have been increased by approximately 50% for this problem.

Input Format

The first line of input contains the integer n. The second line of input contains the integer d. Note that 1 <= n, d <= 100.

Furthermore, n and d are chosen such that the maximum number of ways of eating the whole bar will never exceed 10^{138}.

Output Format

The output contains the answer followed by a newline character.

Sample Input

```
2
2
```

Sample Output

```
10
```

Explanation

Suppose that we label the chocolate bar chunks as follows:

```
|A|B|
|C|D|
```

In this bar, chunk A identified as (1,1), chunk B is identified as (1,2), chunk C is identified as (2,1), and chunk D is identified as (2,2).

There are ten ways this bar could be eaten:

1) A (with B, C, and D at the same time)

2) B (with D), and then A (with C)

3) B (with D), then C, then A

4) C (with D), then A (with B)

5) C (with D), then B, then A

6) D, then A (with B and C)

7) D, then C, and then A (with B)

8) D, then B, and then A (with C)

9) D, then C, then B, then A

10) D, then B, then C, then A

Note: Two other test cases are available if you click on the "Run Code" button.

SAD: Long Way from Home

Problem Statement

Subterranean Antisocial Demons (SADs) live and roam in a network of connected caves. They move from cave to cave following very strict rules:

- Only one SAD may move at a time.
- Every time a SAD moves, it must move from one cave to an adjacent cave.
- No two SADs can occupy the same cave at the same time. Thus, if a cave is already occupied by a SAD, another SAD cannot move into it.
- A SAD may move out of its home cave to allow another SAD to pass through.
- Every SAD must be in its home cave at the end of the day.

The SADs have been busy roaming all day. Your task is to make the SADs happy. Help them find the fastest way for each one to return to its home.

Input Format

The first line of input will contain an integer t, 1 <= t <= 10 that indicates how many test cases are present.
Next follow t test cases with the following format:
The first line of the test case is an integer n, 1 <= n <= 15.
Next come n lines that give the current position of each SAD in the form:
D_i C_j
where D_i and C_j are integers between 1 and 16 inclusive. This indicates that the SAD with ID $D_i$is currently in cave with ID C_j. Note that the home cave for a SAD is the cave with an ID equal to the SAD's ID. Since a SAD can only move into an empty cave, there will

always be more caves than SADs, and therefore the number of cave ID's will always be larger than the number of SAD ID's.
On the next line is an integer l, $n <= l <= 120$.
Next come l lines that give connection between caves in the form:
C_i C_j
where both of these IDs are integers between 1 and 20, inclusive. This line indicates that it is possible to get from the cave with ID C_i to the cave with ID C_j, and from the cave with ID C_j to the cave with ID C_i.

Output Format

For each test case, you should output, on a line by itself, a single integer equal to the**minimum** number of moves needed to get **every SAD** from its current location to its home cave.
Note: it will **always be possible for every SAD to reach its home**, and the minimum number of moves needed is never greater than 30.

Sample Input

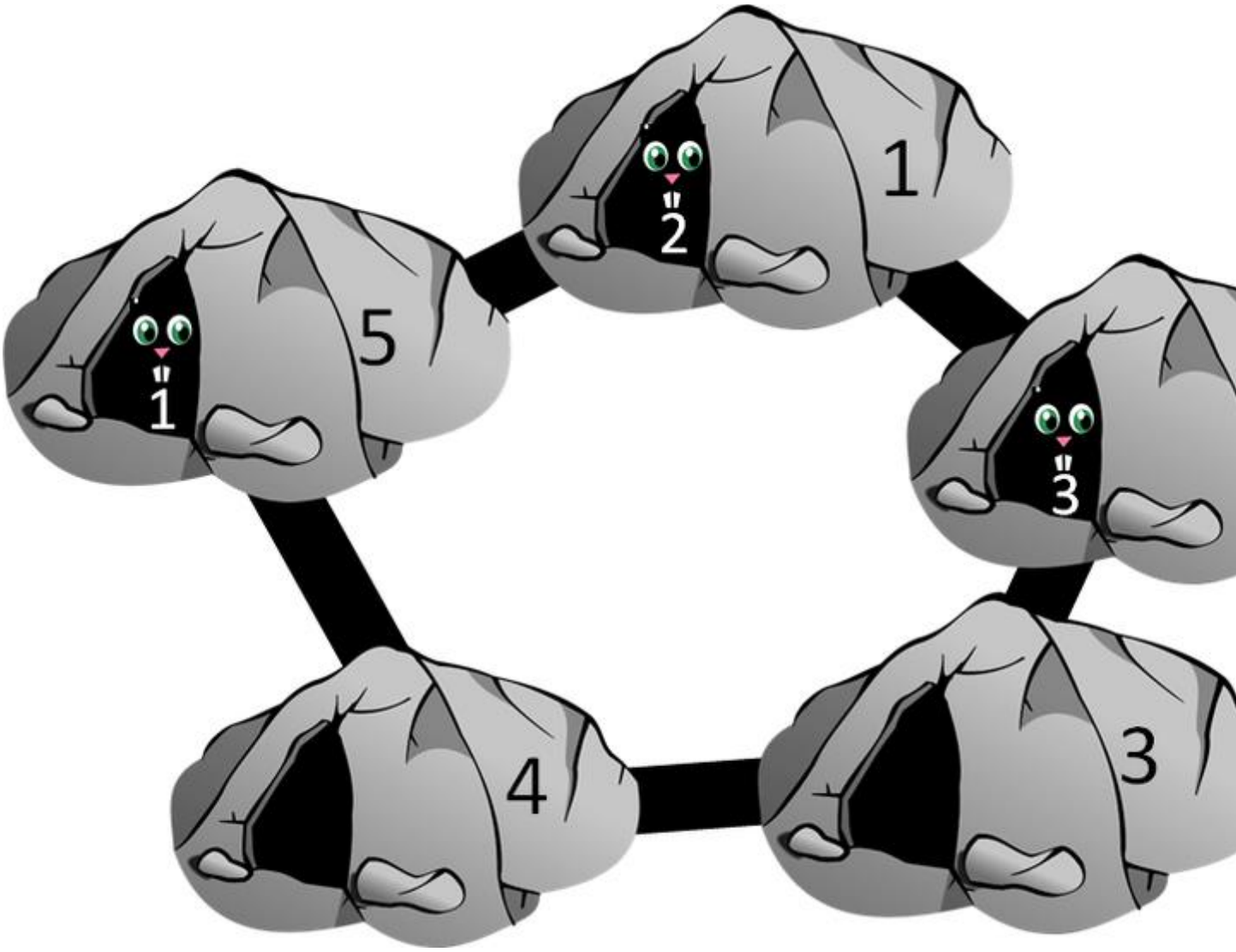
```
1
3
1 5
2 1
3 2
5
1 2
2 3
3 4
4 5
5 1
```

Sample Output

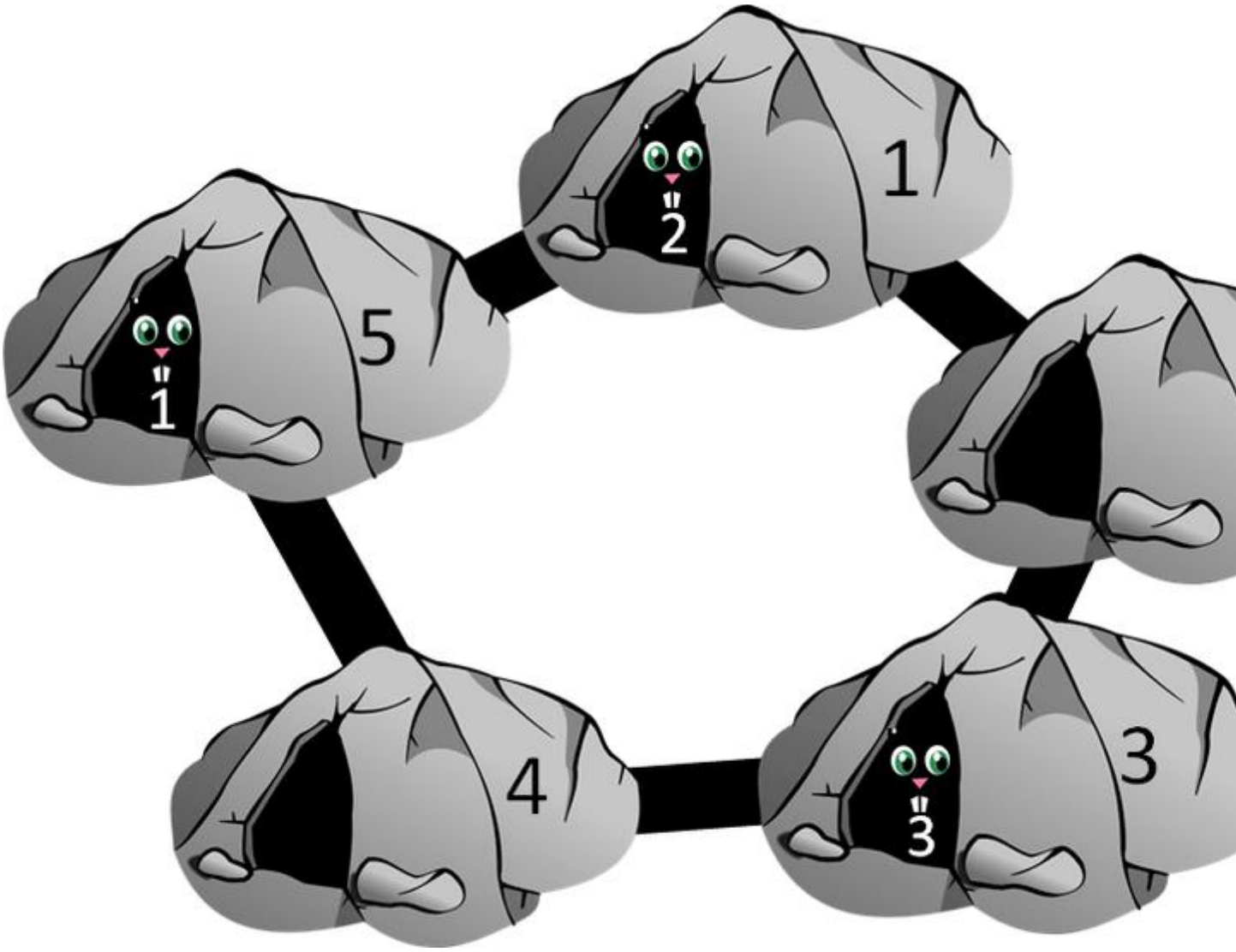
```
3
```

Explanation

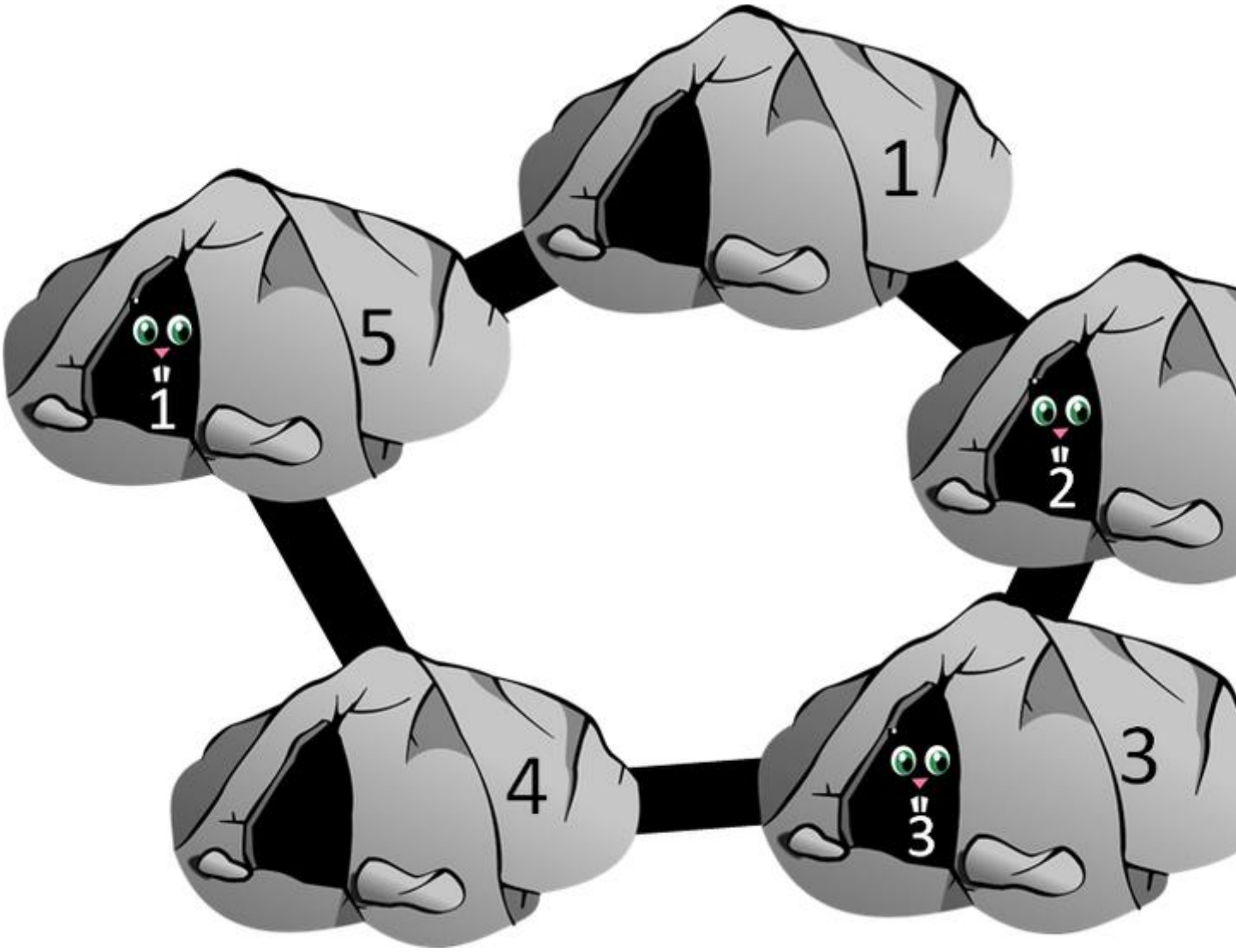
As shown in the next figure, at the beginning of the first test case, SAD 1 is in cave 5, SAD 2 is in cave 1, and SAD 3 is in cave 2. There are two legal moves in this scenario: SAD 1 could move to cave 4 or SAD 3 could move to cave 3.



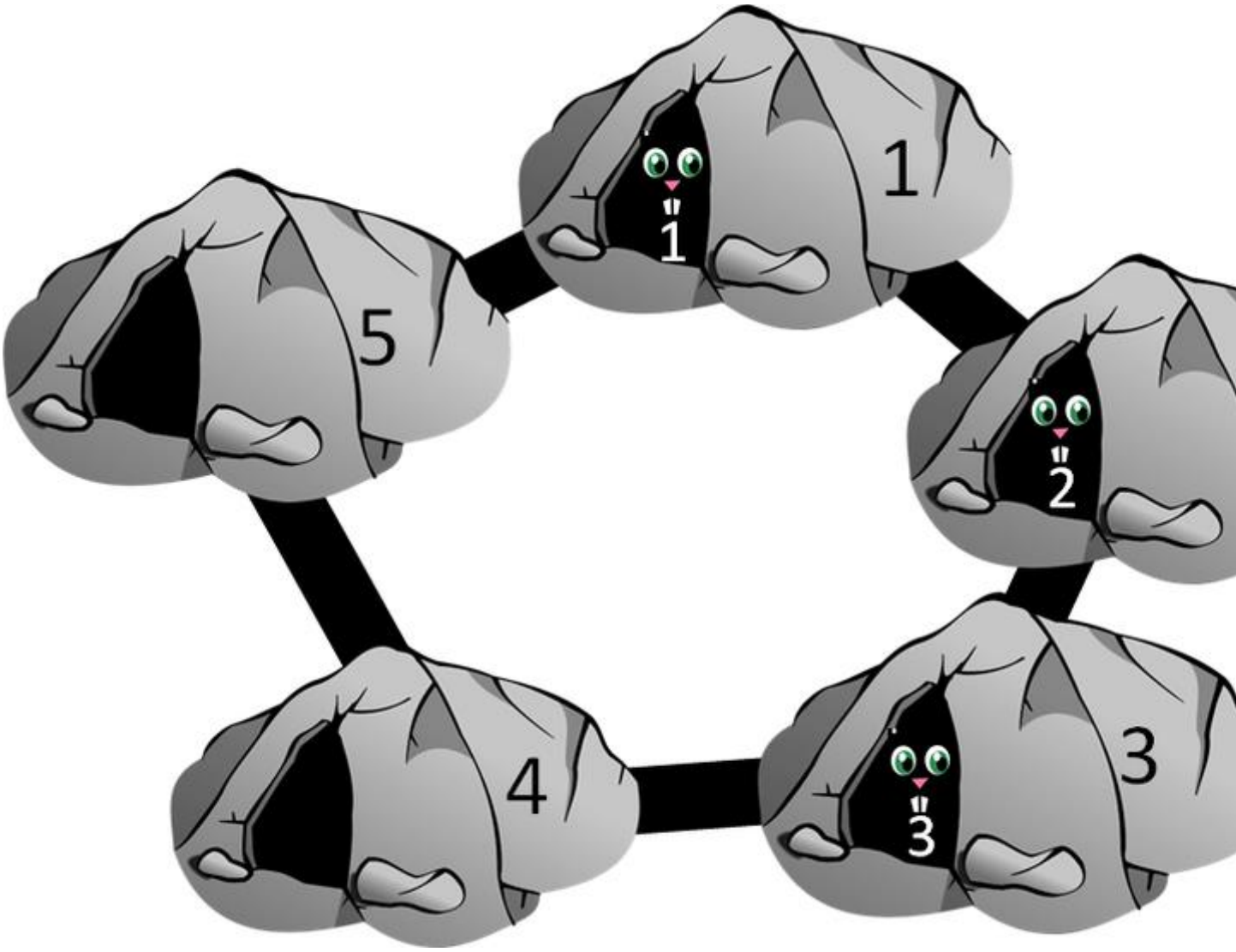
In the first move, then, SAD 3 moves to cave 3. As shown in the next figure, it is home. Now, for the second move, there are four legal moves: SAD 1 could move to cave 4, SAD 2 could move to cave 2, SAD 3 could move to cave 2, or SAD 3 could move to cave 4.



In the second move, then, SAD 2 moves to cave 2. As shown in the next figure, it is also home. For the third move, there are four legal moves: SAD 1 could move to cave 4, SAD 1 could move to cave 1, SAD 2 could move to cave 1, or SAD 3 could move to cave 4.



In the third and final move, SAD 1 moves to cave 1. As shown below, now all of the SADs are home, and it only took three moves!



Images generated from clipart from www.clker.com.

Shortening in the Real World

Problem Statement

You are part of a large gaming firm which is looking forward to offer online gaming competitions through live streaming. This means a lot of users will be looking to share their channels links through multiple social media networks (e.g: twitter, facebook, etc.)

Your company wants to implement URL redirection, where you will provide a URL on your website that when requested will redirect browsers to the links provided by users. Due to the burst of popularity of e-sport sites, your company did not plan for the large demand for URLs. The encoded URLs given by the system are very long and difficult to handle by regular users, reducing the usability of your system and service.

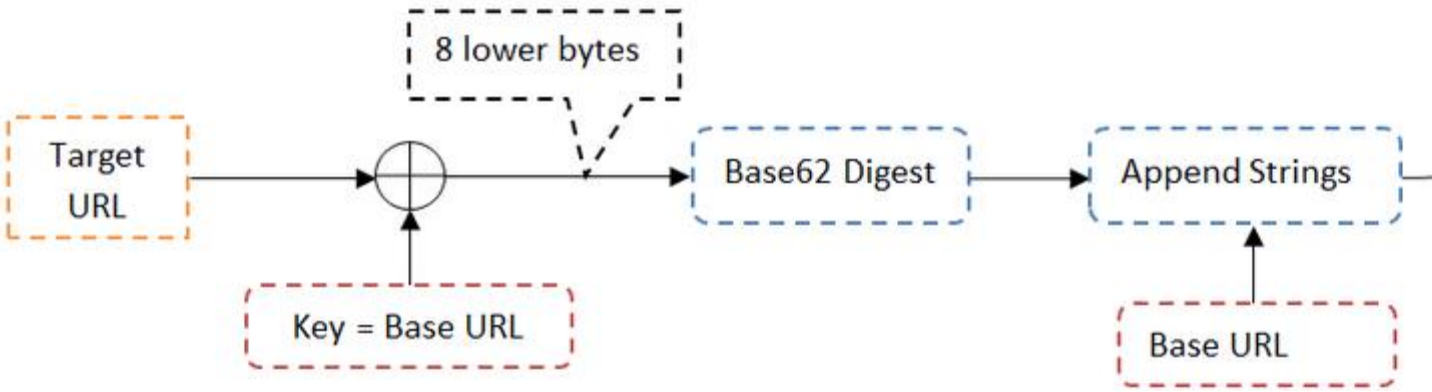
Your boss comes to you in a panic, hoping that you will be capable of solving this simple but critical task and help the company help users to handle URLs in an easy and comfortable manner. As a good software engineer with some basic understanding of Computer Science, you devise a way to encode the original encoded URL into a shorter form through a hash function. As you know, there is always the risk that your hash function will have a collision, but it is a good start.

The hash function takes as input the base URL of your company and the target URL you wish to encode. (For an example of this process, see the *Explanation* portion of the sample input, below). As shown in the figure below, the algorithm proceeds in the following steps.
1) An xor cipher is applied to the target URL, using the base URL of your company as a repeating key. Here we perform a bitwise exclusive-or between each byte of the target URL and the base URL. If the target URL is shorter than the base URL of your company, you would truncate the base URL so that the lengths are equal. If the base URL of your company is shorter than the target URL, you would repeat the base URL as many times as needed to make the lengths equal.

2) Take the last 8 bytes of the output from step 1, and convert this to the corresponding unsigned integer. (See the example below for more details.)

3) Encode this unsigned integer using Base62 encoding. In this encoding, you convert the integer to a base 62 number, where the digits 0-9 represent values 0-9, lowercase letters a-z represent values 10-35, and capital letters A-Z represent values 36-61.

4) The encoded url consists of the base URL, a backslash, and the base 62 encoded value produced in the previous step.



Notes:

- You should process the URLs using UTF-8 encoding.
- The URL to be encoded will always be at least 8 characters long.
- The base 62 number should not be padded with extra zeros. For example, the number "62" should be represented as "10", not as "010" or "0010" or "00010". The number "0" would be represented as "0".
- The base URL will never end with a backslash.

Input Format

The first line of input will contain the base URL of your company.

The second line will contain a number n (1 <= n <= 1000) which will indicate the number of URLs you will need to encode.
The following n lines will contain target URLs to encode.

Output Format

The output should be n lines, where each line will correspond to an encoded URL.

Sample Input

```
http://www.ieee.com
2
http://www.ieee.org/xtreme
http://www.ieee.org/membership_services/membership/young_professionals/index.html
```

Sample Output

```
http://www.ieee.com/SHPQ4gzW1Y
http://www.ieee.com/Btazwa9mke
```

Explanation

Consider the first target URL to be encoded: `http://www.ieee.org/xtreme`.
We start by finding the UTF-8 encoding of the base URL `http://www.ieee.com` and the target URL.

```
target URL = [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e,
0x69, 0x65, 0x65, 0x65, 0x2e, 0x6f, 0x72, 0x67, 0x2f, 0x78, 0x74, 0x72, 0x65,
0x6d, 0x65]
base URL   = [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e,
0x69, 0x65, 0x65, 0x65, 0x2e, 0x63, 0x6f, 0x6d]
```

Now we apply the exclusive-or cipher, repeating the bytes in the base URL so that the two strings are the same length:

```
    [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x69,
0x65, 0x65, 0x65, 0x2e, 0x6f, 0x72, 0x67, 0x2f, 0x78, 0x74, 0x72, 0x65, 0x6d,
0x65]
xor
    [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x69,
0x65, 0x65, 0x65, 0x2e, 0x63, 0x6f, 0x6d, 0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f,
0x2f]
```

```
=
    [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xc, 0x1d, 0x0a, 0x47, 0x0c, 0x00, 0x02, 0x5f, 0x42,
0x4a]
```

Next we take the last 8 bytes of this number and convert it to an unsigned integer:

0x0a470c00025f424a = 740,573,857,905,066,570

Next we convert this number from base 10 to base 62 to get: `SHPQ4gzW1Y`
We then append this number to the base URL to produce the output:`http://www.ieee.com/SHPQ4gzW1Y`

Communities

Problem Statement

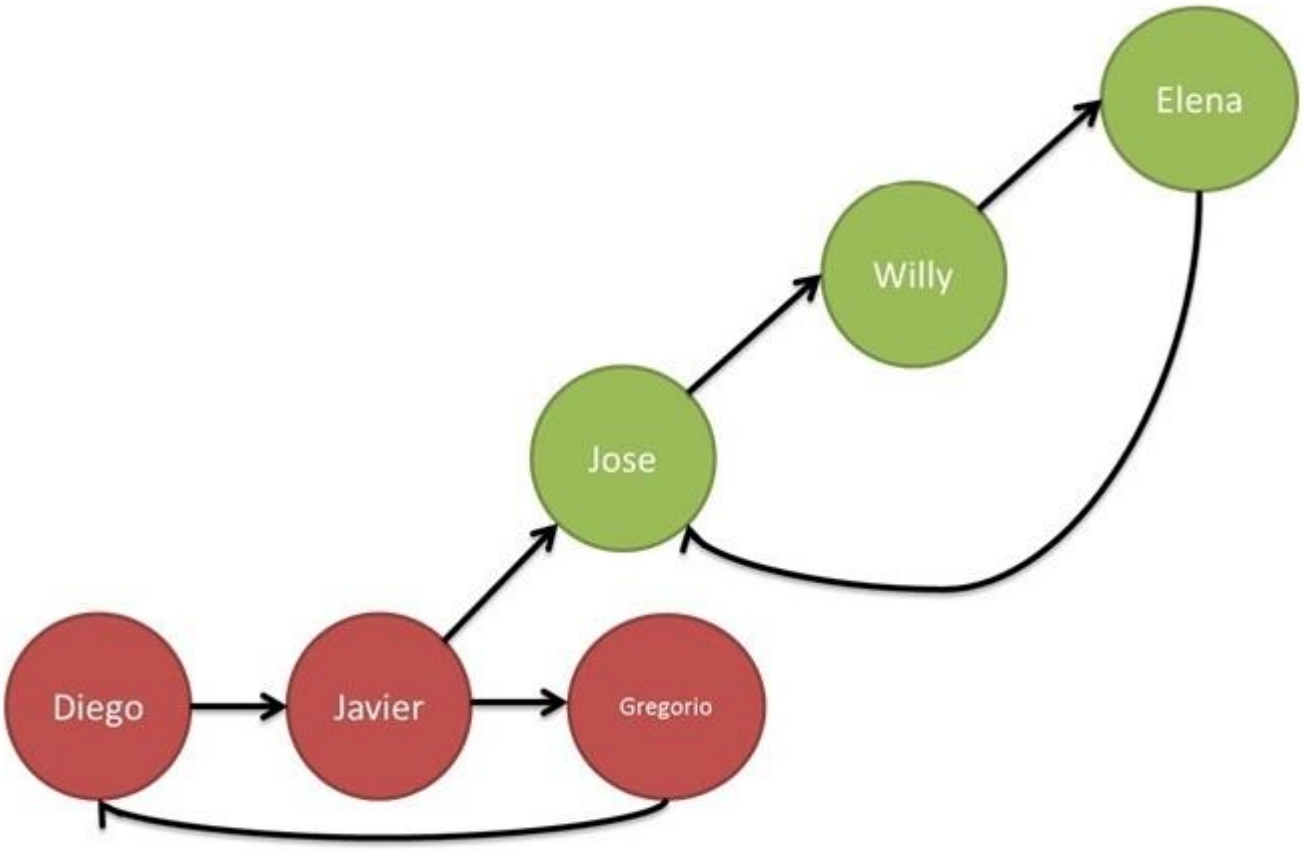
Social media networks and the amount of information they aggregate is astonishing. With so much information, new patterns and interactions of human society can be identified.

In our social network, the relationships model the flow of information, and they are directed. Alice may subscribe to Bob's newsfeed, while Bob does not subscribe to Alice's. Moreover, the flow of information in our network is such that a person can see the newsfeeds of all people who could reach the person following along a path in the network. Suppose, then, that Alice subscribes to Bob's newsfeed, Bob subscribes to Chuck's newsfeed, and Chuck subscribes to Dave's newsfeed. This would correspond to a simple linear graph:

Alice <- Bob <- Chuck <- Dave

Then Dave would be able to read his own news items only; Chuck would be able to read news items posted by either Dave or himself; Bob would be able to read news items posted by either Chuck, Dave or himself; and Alice would be able to read everyone's news items. Note that everyone can read their own newsfeed.

We are interested in the defining a community metric for our social network. We define a community as a group of people who are able to see all news items posted by any member of the group. As an example, in the figure below, there are two communities, each shown in a different color.



Note that in the community shown in green above, Jose, Willy, and Elena can all read each other's posts. While Jose, Willy, and Elena can also read Javier's news items. However, Javier cannot read news items from Jose, Willy, or Elena, and is therefore not included in their community.

Your task is to identify the sizes of these communities from biggest to smallest.

Input Format

The first line of input will contain two space separated integers: the total number of people that devise the social network, n (1 <= n <= 10000) and m, the number of communities for which you should print the size. The following lines will contain a directed relationship between 2 people. If the line reads "Jon Peter", then Peter subscribes to Jon's news feed, and the relation is Jon -> Peter.

The word "END" will appear on a line by itself after the list of relationships.

All of the names are strings containing fewer than 50 characters.

Output Format

The output consists of m lines, where each line will correspond to the size of a community from biggest to smallest. If there are fewer than m communities, after outputting the size of all existing communities, output lines containing “Does not apply!” for the missing values.

Sample Input

```
6 2
Jose Willy
Willy Elena
Elena Jose
Diego Javier
Javier Gregorio
Gregorio Diego
Javier Jose
END
```

Sample Output

```
3
3
```

Explanation

This input corresponds to the graph described in the problem statement above.

Note that two additional sample inputs are available if you click on the "Run Code" button.

Pattern 3

Problem Statement

Vangelis the bear received a digital signal pattern generator that his brother Mitsos built. The generator produces a signal that is encoded using the Latin alphabet. Vangelis starts the generator for some time and records the signal generated. He wants to study the sample he received and try to identify the smallest pattern that the generator could be using to generate the sample.

Your task is to help Vangelis by writing a program that will calculate the length of the smallest valid pattern.

Input Format

The input is made up of multiple test cases.

The first line contains an integer T (1 <= T <= 10), the number of test cases in this file.
Each line contains an encoded signal. The signal is encoded using the small letters of the Latin alphabet. The length of a signal is between 1 and 10^6 characters, inclusive.
Vangelis has started the recording at the beginning of a pattern, so each line begins with the first character in the pattern. His recording lasts for at least one pattern length, but the length of the recording **may not be an exact multiple of the pattern length**.

Output Format

There must be T lines of output and each line will contain a single non-negative integer number, the length of the minimum valid pattern.

Sample Input

```
6
abab
abababababababababab
abababababab
abc
```

```
aaaaaa
aabaabbaabaabbaabaabbaabaab
```

Sample Output

```
2
2
2
3
1
7
```

Explanation

The repeating patterns in each of the test cases are:

```
ab
ab
ab
abc
a
aabaabb
```

Land

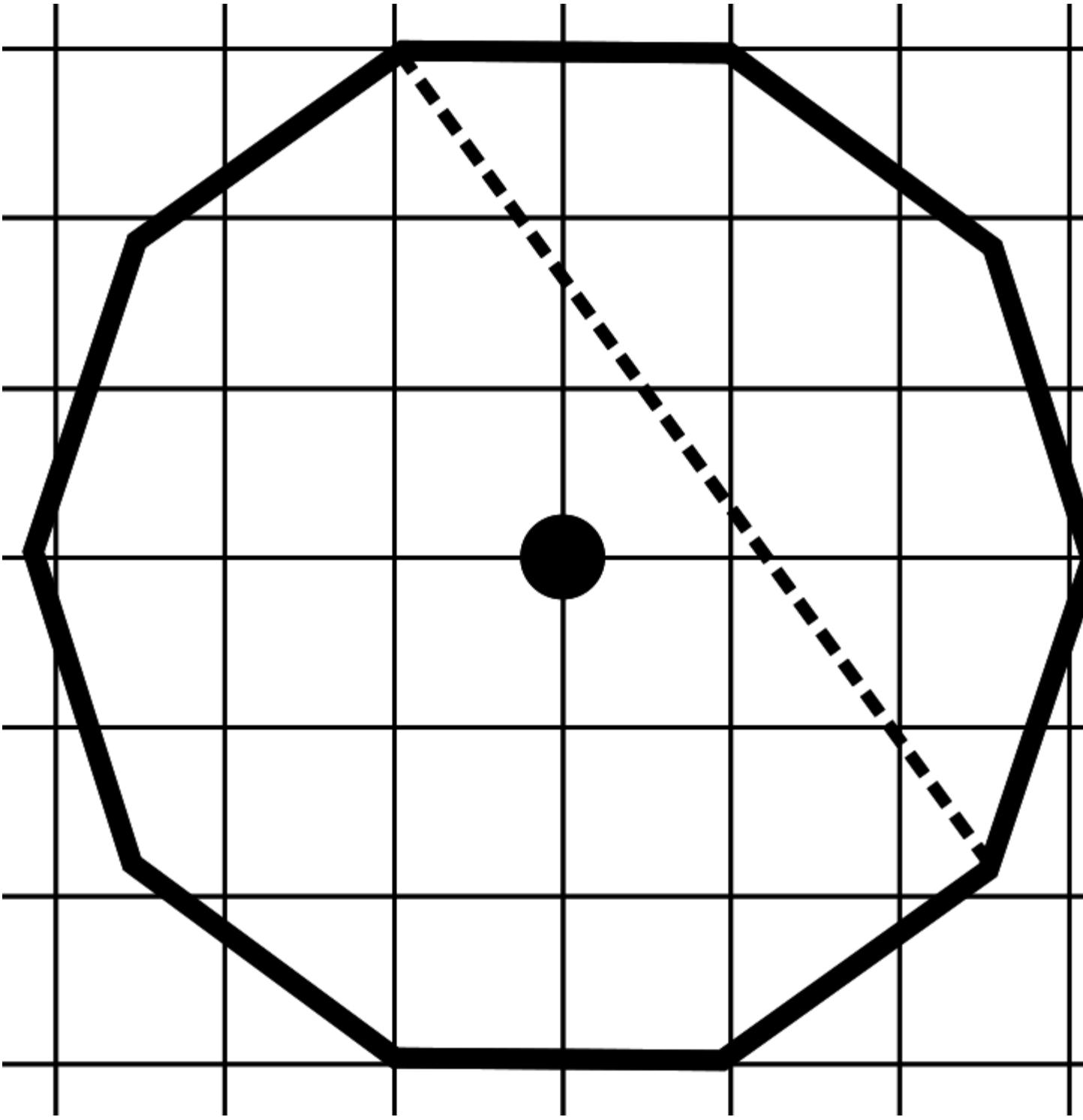
Problem Statement

Vangelis the bear bought a real property inside a forest where he wants to build his new corporate headquarters. The shape of the land is a strictly convex polygon. This means that the shape is a polygon with all its interior angles less than 180°. All vertices of the shape are on a plane, whose locations are given in a Cartesian coordinate system.

In the plot, there are some really old trees that Vangelis wants to protect. To do so, Vangelis decided that he will separate his land into two parts. The first part, where there must not be any trees, will be used for the headquarters building. The second part, where the ancient trees reside, will be preserved as a park.

Being a scientist, he decided to make the separation using a diagonal of the polygon. Your task is to help Vangelis compute the maximum possible land that he can use to build his headquarters.

Note: A tree cannot appear on the border between the regions. Thus, in the image below, the maximum area for the headquarters *is not equal* to one half the area of the land.



Input Format

The input is made up of multiple test cases.

The first line contains an integer T (1 <= T <= 10), the number of test cases in this file.
The following repeats T times:

- The first line in the test case contains an integer N (4 <= N <= 300000), the number of vertices that compose the land shape.
- Each of the following N lines contain two single-space-separated integers X and Y (-10^9<= X, Y <= 10^9), the coordinates of a vertex. The vertices are given in a counter-clockwise order going around the boundary of the land.
- The line after contains an integer M (0 <= M <= 300000), the number of trees inside the land.
- Each of the following M lines contain two single-space-separated integers I and J, the coordinates of a tree inside the land. None of the trees will be placed on an edge of the land.

Output Format

There must be T lines of output and each will contain a single non-negative real number, the maximum possible area that can be used for the headquarters in the specific test case. The numbers should always be rounded to two decimal places, and they should always display these two decimal places even if the final value is an integer. In the cases that there is no available land to be used, your program should write 0.00.

Sample Input

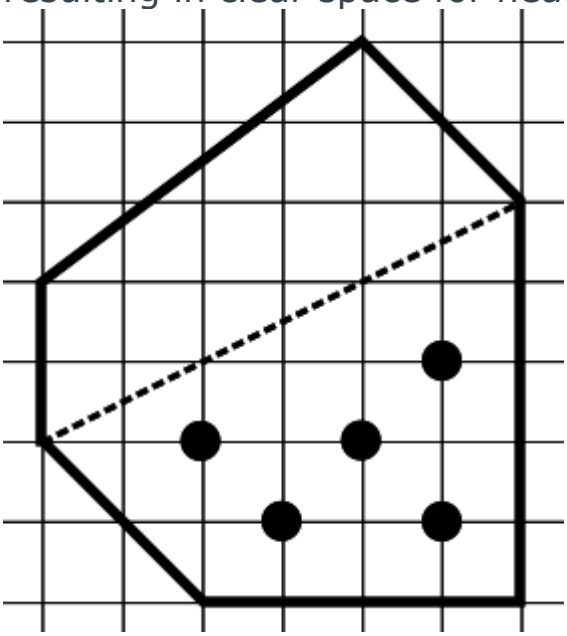
```
1
6
2 0
6 0
6 5
4 7
0 4
0 2
5
2 2
3 1
4 2
5 1
5 3
```

Sample Output

```
13.00
```

Explanation

The image below represents the optimal division of the land resulting in clear space for headquarters of 13 m^2.



Note: An additional test case is available if you click on the "Run Code" button.

Alice and Bob Play Sheldon's Favorite Game

Problem Statement

Rock, Paper, Scissors, Lizard, Spock is a game, invented by Sam Kass and Karen Bryla, that extends the Rock, Paper, Scissors game. It is a favorite of the character Sheldon from the TV Show, *The Big Bang Theory*.

The rules of the game are as follows. Each player chooses a shape from the set {*Rock*, *Paper*,*Scissors*, *Lizard*, *Spock*}. If the two players choose the same shape, they tie. Otherwise, the winner is the one with the shape listed first in the following set of rules:

```
Scissors cuts Paper
Paper covers Rock
Rock crushes Lizard
Lizard poisons Spock
Spock smashes Scissors
Scissors decapitates Lizard
Lizard eats Paper
Paper disproves Spock
Spock vaporizes Rock
(and as it always has) Rock crushes scissors
```

Below is a graphical representation from *The Big Bang Theory's* wiki:

Alice and Bob also enjoy this game, and they have decided to play a series of them. Each has a very specific strategy that they follow.

Alice's strategy is as follows:

1) If she wins a game, she keeps the same shape.

2) If she ties, she chooses a shape from one of the two that would beat her current shape. Of these two, she chooses the one that beats the other. For example, if she has tied when choosing *Rock*, her options are *Paper* and *Spock*. Since *Paper* beats *Spock*, she chooses *Paper*.
3) If she loses, she chooses a shape from one of the two that would beat her opponent's current shape. Of these two, she chooses the one that beats the other. For example, let's say she has lost by choosing *Rock*, when her opponent chose *Paper*. She will then choose from*Scissors* or *Lizard*. Since *Scissors* beats *Lizard*, she chooses *Scissors*.
Bob's strategy is as follows:

1) Every other turn, he chooses *Spock*.
2) If he won the previous turn when playing *Spock*, he chooses *Rock*.
3) If he tied the previous turn when playing *Spock*, he chooses *Lizard*.
4) If he lost the previous turn when playing *Spock*, he chooses *Paper*.
Your task is to write a program that evaluates a series of games between Alice and Bob.

Input Format
The first line of input contains a single integer t, 1 <= t <= 50, containing the number of test cases in the input.
Then come t lines each describing a test case, made up of a series of games for you to evaluate. These lines have the following format:

```
[AliceShape] [BobShape] [n]
```

Where

- [AliceShape] is the shape that Alice will choose in the first game of the series.
- [BobShape] is the shape that Bob will choose in the first game of the series.

- [n] is an integer, 1 <= n <= 10^{18}, indicating how many games Alice and Bob will play in the series.

Output Format

Output will consist of a single line in the appropriate one of the following forms:

```
[Player] wins, by winning [WinGames] game(s) and tying [TieGames] game(s)

Alice and Bob tie, each winning [WinGames] game(s) and tying [TieGames] game(s)
```

where

- [Player] is the name of the player with more wins (either "Alice" or "Bob")
- [WinGames] is the number of games won either by the winner or, in the case of a tie, by each player
- [TieGames] is the number of games in which the players tied

Notes:

- The output is case sensitive. The player names, for example, must be either "Alice" or "Bob". Neither "alice" nor "BOB" will be acceptable.
- The words are separated by a single space, and there are no spaces before the first word in the line, nor after the last word in the line.

Sample Input

```
2
Rock Spock 4
Paper Paper 1
```

Sample Output

```
Bob wins, by winning 2 game(s) and tying 1 game(s)
```

```
Alice and Bob tie, each winning 0 game(s) and tying 1 game(s)
```

Explanation

There are two test cases in this input:

Test Case 1

In the first game, Bob wins, since *Spock* vaporizes *Rock*.
Bob won when choosing *Spock* so he chooses *Rock*. Alice lost, so she chooses from *Paper* and*Lizard*, both of which beat Bob's last choice of *Spock*. Since *Lizard* beats *Paper*, she chooses*Lizard*.
In the second game, then, Bob wins again,
because *Rock* crushes *Lizard*.
Bob did not play *Spock* last turn, so he chooses *Spock* next. Alice lost, so she chooses from*Paper* and *Spock*, both of which beat Bob's last choice of *Rock*. Since *Paper* beats *Spock*, she chooses *Paper*.
In the third game, Alice wins, since *Paper* disproves *Spock*.
Bob lost when choosing *Spock* so he chooses *Paper*. Alice won, so she continues playing*Paper*.
In the fourth game, they tie by both choosing *Paper*.

Test Case 2

This test case consists of a single game in which both players play *Paper*.

Eat, Sleep, Drink, Code

Problem Statement

This challenge is sponsored by SyncFusion.

Alice has recognized that strategy is key for success in IEEEXtreme. Your task is to help her by writing a program that will calculate the maximum score that she can earn in marathon competitions like Xtreme.

Alice starts the competition with a certain energy level. She can only attempt a problem if her energy level prior to starting the problem is greater than or equal to the energy required by the problem. Furthermore, if she attempts to solve the problem, her energy level after the hour is reduced by the energy required by the problem.

Alice wants you to assume that every hour a problem is released, and she can make the following decisions:

- Attempt to solve the problem. She is able to accurately predict how many points she will earn by attempting the problem.
- Skip the problem and sleep. Note that she will not come back to this problem later. She will gain a fixed amount of energy by doing so.
- Drink a caffeinated cola and attempt the problem, if she has drinks remaining. She will gain a fixed amount of energy *immediately*. She can only choose this option if her resulting current energy level would be greater than or equal to the energy level required by the problem. As usual she will expend the energy required to solve the problem. In addition, as the caffeine wears off, she will lose a certain amount of energy units exactly two hours later.

Notes:

- It is ok for her energy level to become negative after losing the points due to the cola consumption. However, she will need to boost her energy by sleeping or drinking additional cola before she will be able to solve a problem.
- She can only drink one cola per hour.
- If she drinks a cola, she must attempt the problem.
- For each hour that she sleeps and skips a problem, she gains the fixed amount of energy. Thus, if she sleeps for two consecutive hours, she will gain twice as much energy as if she slept for one hour. If she sleeps for three consecutive hours, she will gain three times as much energy, etc.

Input Format

The first line of input contains an integer k, 1 <= k <= 20, which indicates how many test cases are present.
Each test case then has the following format. The first line of the test case consists of the following:

```
[Hours] [Energy] [Sleep] [DrinkCount] [Drink] [Crash]
```

Where

- [Hours] gives the length of the contest in hours, 1 <= [Hours] <= 168. (Alice envisions the day when Xtreme is a week-long contest!)
- [Energy] is Alice's energy level at the beginning of the contest, 0 <= [Energy] <= 10^7.
- [Sleep] is the amount of energy that Alice gains by skipping a problem and sleeping, 1 <= [Sleep] <= 10^6.
- [DrinkCount] is a count of colas that Alice has at the start of the contest, 0 <= [DrinkCount] <= 24.
- [Drink] is the initial boost that Alice receives from drinking a cola, 1 <= [Drink] <= 10^6.

- [Crash] is the amount of additional energy that Alice loses two hours after drinking a cola, 1 <= [Crash] <= 10^6.

Then there follow [Hours] lines, each describing a problem, and listed in the order in which the problem is released, i.e. the problem on the first line is released at the start of the contest, the second problem is released one hour later, the third problem is released an hour after that, etc. These lines have the following format:

```
[EnergyRequired] [Points]
```

Where

- [EnergyRequired] is an integer equal to the amount of energy that Alice will expend in attempting to solve the problem, 1 <= [EnergyRequired] <= 10^7.
- [Points] is equal to the points that Alice will earn if she attempts the problem. [Points] will be equal to an integer chosen from the following set {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

Output Format

For each test case, output on a separate line, a single integer equal to maximum amount of points Alice can earn in the contest.

Sample Input

```
1
4 5 16 2 7 8
10 100
6 50
15 20
3 10
```

Sample Output

```
160
```

Explanation

There is only one test case in the sample input. For this test case, the contest lasts 4 hours, Alice's initial energy level is 5, she gains 16 energy units by sleeping, she has 2 colas, she gains a boost of 7 energy units by drinking a cola, but then loses 8 energy units after two hours.

In order to solve the first problem, she drinks a cola in order to boost her energy level to 12. She can then solve the first problem, and earns 100 points, by doing so. The first problem takes 10 energy units to solve. Note that she will be penalized by 8 energy units when the third problem is released as the caffeine wears off.

At the start of the second hour, then, she has 100 points, and an energy level of 2. In order to solve the second problem, she drinks another cola. This raises her energy level to 9. She then solves the problem, earning 50 more points. She expends 6 energy units solving the problem. Note that she will lose an additional 8 energy units as the caffeine wears off prior to the release of the fourth problem.

At the start of the third hour, she has 150 points. Her energy level is depleted both by solving the previous problem and as the caffeine from the first cola wears off. Thus her energy level is -5.

She decides to sleep instead of solving the third problem. Her energy level is boosted to 11 by sleeping but reduced to 3 as the caffeine from the second cola wears off.

She then solves the fourth problem and earns 10 more points.

This sequence of decisions maximizes her point total, so your program should output 160, the total number of points that she earned.

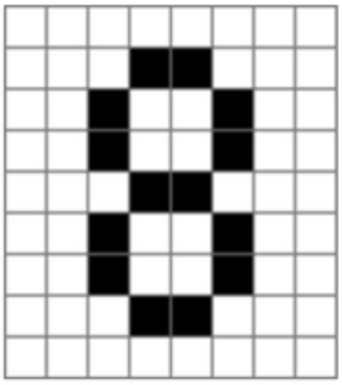
That's So Characteristically Euler!

Problem Statement

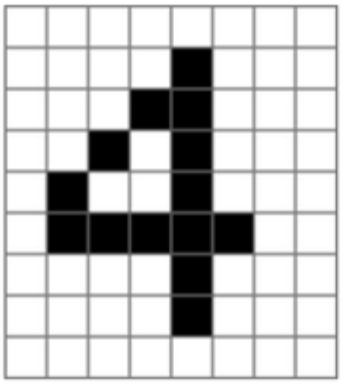
This challenge is sponsored by the IEEE Computer Society.

Topology is a branch of mathematics that examines the shapes and structures of objects. The*Euler characteristic* is a topological label that we can determine for an item or collection of items. For a group of two dimensional objects, one way to define the Euler characteristic is the number of connected objects minus the number of holes in the objects.

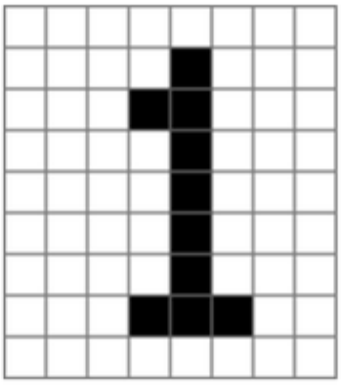
The Euler characteristic has useful applications in 2D image recognition and classification. Consider the following black and white representations of three digits and a letter and their corresponding Euler characteristics.



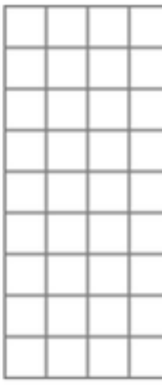
1 connected region
2 holes
Euler characteristic = -1



1 connected region
1 hole
Euler characteristic = 0



1 connected region
0 holes
Euler characteristic = 1



2 connect
0 h
Euler chara

As you can see, the Euler characteristic can give a lot of information about what a given image might or might not represent. Note that, in this problem, we consider filled-in regions that touch diagonally to be part of the same connected region; however, holes that only touch diagonally are considered to be separate. Your task is to compute the Euler characteristic (connected regions minus holes) for a black and white image similar to those above.

Input Format

Each input contains multiple test cases. The first line of input contains an integer t, 1 <= t <= 10.

Following this line are t test cases, each with the following format:

- The first line of the test case contains an integer n, giving the height of the image.
- The second line of the test case contains an integer m, giving the width of the image.
- The rest of the test case contains n lines of text. These n lines of text contain $m$characters, each of which is either the letter 'X' or the letter 'O'. An 'X' represents a black pixel, and an 'O' represents a white pixel.
- The possible ranges for n and m are 1 <= n <= 1,000 and 1 <= m <= 1,000. The input will always be correctly formatted.

Output Format

The correct output for each input is a single integer per test case (each integer on its own line) which gives the Euler characteristic for each input image.

Sample Input

```
1
4
4
0000
0XX0
0XX0
0000
```

Sample Output

```
1
```

Explanation

The sample input contains a single image. The image contains a single connected square region surrounded by empty space.

Thus, 1 region - 0 holes = 1.

Finite Domain Constraints

Problem Statement

A finite domain constraint is a linear equality or inequality over integer variables. Each variable has a known and finite domain (set of possible values). A finite domain constraint is satisfiable if each variable can be assigned a value from its domain in a way that makes the equality or inequality true. For example, given the finite domain constraint:

(X + Y) = 5
with the domain of X as {1, 2, 3, 4} and the domain of Y also as {1, 2, 3, 4}, the constraint is satisfied by assigning 2 to X and 3 to Y. Note that the constraint can also be satisfied by assigning 3 to X and 2 to Y, 1 to X and 4 to Y, and 4 to X and 1 to Y. Hence, there are 4 distinct variable assignments that satisfy the constraint.
If we change the constraint to the inequality:
(X + Y) < 5
with the same domains, there are now 6 distinct variable assignments that satisfy the constraint: (X = 1, Y = 1), (X = 1, Y = 2), (X = 1, Y = 3), (X = 2, Y = 1), (X = 2, Y = 2), and (X = 3, Y= 1).
Given a finite domain constraint and domains for each variable, your task is to determine how many distinct variable assignments satisfy the constraint.

Input Format

The input is made up of multiple test cases. Each test case ends with a 0 on a line by itself.

The first line in each test case is an integer n, 1 <= n <= 10. n is the number of variables in the finite domain constraint.
The next n lines are each of the form:
[Variable_Name] [Low] [High]

where:

- [Variable_Name] is the variable name

- [Low] is the lower bound of the variable's domain (inclusive)
- [High] is the upper bound of the variable's domain (inclusive)

For example:

```
X 1 4
```

means that the domain of X is {1, 2, 3, 4}. Variable names are not repeated within the same test case.
The next line of each test case is a finite domain constraint over those variable names.

Notes:

- The lower and upper bounds are integers from 1 to 10 (inclusive).
- The constraint will contain exactly one occurrence of either = or <. Note that the operator <= will never appear in a constraint.
- The constraint can additionally contain parentheses, +, -, *, variable names and integer constants.
- The constraint will be fully parenthesized so that precedence (order of operations) is not needed.
- Each distinct symbol in the constraint will be separated from other symbols by exactly one space.
- Each variable occurs exactly once in the constraint.
- The number of integer constants is less than or equal to the number of variables plus one, and each constant is between -100 and 100 (inclusive).
- When the * operator is used in a constraint, one operand will be a constant and the other will be a variable.

- You can assume that the constraint is syntactically correct.

Output Format

For each test case, your program should output, on a line by itself, the number of distinct variable assignments that satisfy the finite domain constraint with the given variable domains.

Sample Input

```
2
X 1 4
Y 1 4
( X + Y ) = 5
0
3
X 1 3
Y 2 4
Z 1 4
( ( X + Y ) - ( 2 * Z ) ) < 5
0
4
W 2 5
X 1 3
Y 2 9
Z 1 4
( ( X + ( Y - 3 ) ) + ( -2 * Z ) ) = ( ( 5 * W ) - 10 )
0
```

Sample Output

```
4
35
15
```

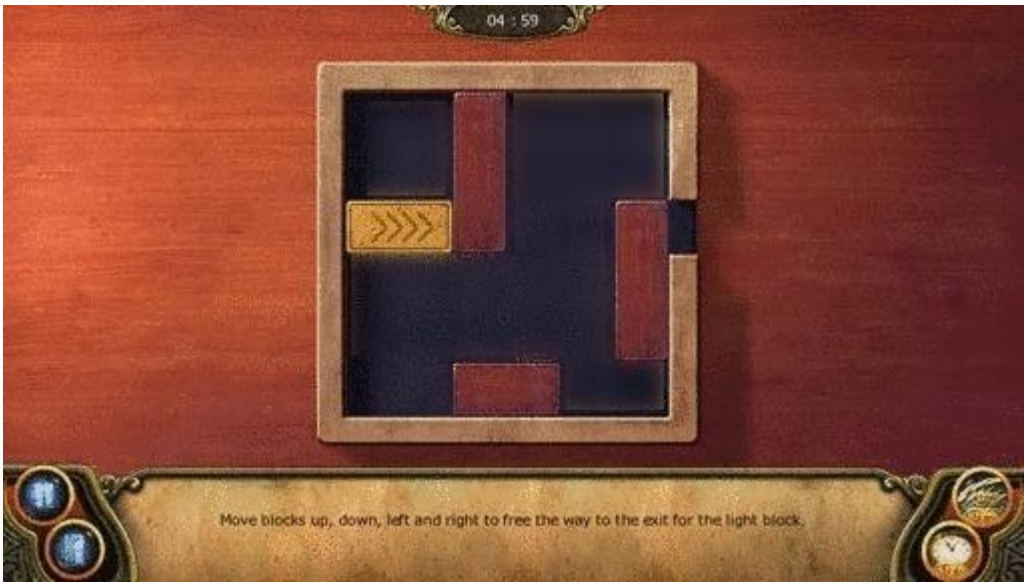
Explanation

The first test case in the sample input corresponds to the first example given in the problem statement.

Blocks Game

Problem Statement

In this problem, you will solve a puzzle from the Secret Society Hidden Mystery Game by G5 Games. A typical board for this game is shown here:



Each block can only move in one dimension: vertical blocks can only move vertically and horizontal blocks can only move horizontally. The goal of the game is to slide the yellow block out of the frame through the hole in the frame on the right side. To do that, you have to slide the rest of the blocks out of the way. In addition, for this problem, you must do it in the smallest number of moves possible.

Here are some details to constrain our game:

- Moving a block through two spaces counts as two moves.
- The task is just to find the minimum number of moves required to win the game – the sequence of specific moves is not important.
- The board is in form of a square.

- The yellow block is always horizontal, and it will always cover 2 spaces.
- Once the yellow block is located right next to the exit, you have won the game. You do not need to move the yellow block off the board.
- The blocks can only move inside the board, and movement of any block outside the board is not permitted.
- The game is always winnable, in a finite number of moves.

Input Format

Each run of the program will process a sequence of test cases each containing an instance of the game.

The first line of input contains a number T. T specifies the number of test cases in the input.
Each test case is specified as follows:

On the first two lines of the test instance is an integer Y, representing the column with the leftmost side of the yellow block and N, giving the dimensions of the board. Since the board is always square, N is the number of rows and the number of columns of the board. Hence, the board coordinates can be considered as an N x N matrix (i.e. the top left corner of the board is row 0 and column 0). The yellow block is always located in the third row (i.e. it is in the row labelled 2).
The third line of the test case contains an integer B representing the number of brown blocks in the problem.
The next B lines give the location of each of the brown blocks in the following format:
$S\ D\ R\ C$
Where

- S is an integer give the number of spaces the block covers. A block will always fit in the board.

- D is either 'V' or 'H' designating whether the block is horizontal or vertical
- R and C are the row and column, respectively, of the top left square of the block

Constraints:

- 1 <= T <= 5
- 0 <= Y < N - 2
- 4 <= N <= 6
- 1 <= B <= 12
- The maximum number of unique board configurations that can be reached in any test instance is 250,000.

Output Format

The output must be one integer for each instance of the problem on a separate line giving the minimum number of moves required to win the game.

Sample Input

```
1
0
6
3
3 V 0 2
3 V 2 5
2 H 5 2
```

Sample Output

```
9
```

Explanation

The input specification corresponds to the image below.

This game is winnable in 9 moves. For example, one possible sequence of moves that wins in the shortest number of moves is:

1) Move the horizontal brown block one square to the right.

2-4) Move the left vertical block down three squares. Note that this counts as three moves.

5-7) Move the yellow block to the right three squares.

8) Move the right vertical block down one square.

9) Move the yellow block right one square so that it is next to the exit.

Would Be... Could Be... BST!

Problem Statement

This challenge is sponsored by IEEE Xplore.

You are given an ambiguous description of one or more binary trees. Each key value is a non-empty string of lowercase letters, and each key occurs in only one node in the collection. You must decide if the description that is given must have been derived from a single binary search tree, could have been derived from a single binary search tree, or could not have been derived from a binary search tree. Please refer to http://en.wikipedia.org/wiki/Binary_search_tree for background information on binary search trees.

Note: The default time limits for all languages have been doubled for this program.

Input Format

The input is made up of multiple test cases. Each test case begins with an integer N, with 1 <= N <= 50,000. Next come N lines each describing an edge in the collection of binary trees:

[key1] [key2]

where [key1] and [key2] are strings corresponding the key values of two nodes, *node1* and *node2*, respectively, where *node1* is either the left child of *node2* OR *node2* is the right child of *node1*.

The last line of input following all of the test cases consists of the number 0 on a line by itself. The total number of edges in all test cases in a single input file will be less than or equal to 500,000.

Output Format

For each test case, you will output a single string followed by the newline character:

- If the description given in the test case must correspond to a binary search tree, you should output "BST!".

- If the description given in the test case corresponds to a binary search tree in some instantiations but not in others, you should output “BST?”.
- If the description given in the test case cannot describe a binary search tree, you should output “!BST”.

Sample Input

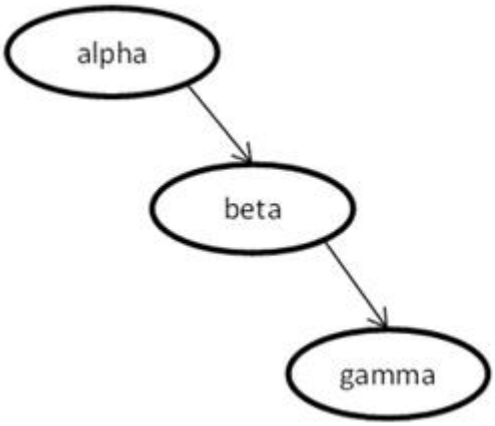
```
2
alpha beta
beta gamma
2
alpha delta
alpha gamma
2
alpha beta
delta gamma
1
beta alpha
0
```

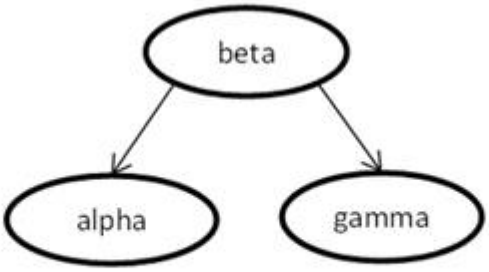
Sample Output

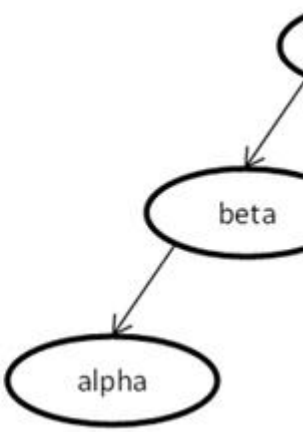
```
BST!
BST?
!BST
!BST
```

Explanation

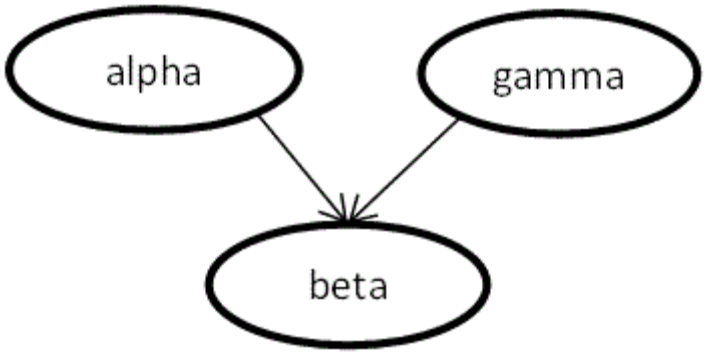
The first test case could have been derived from three different configurations, which are shown below. Note that all of these configurations are binary search trees.



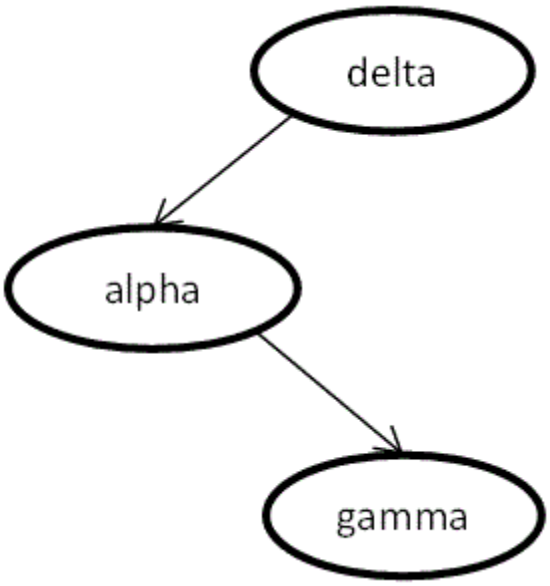


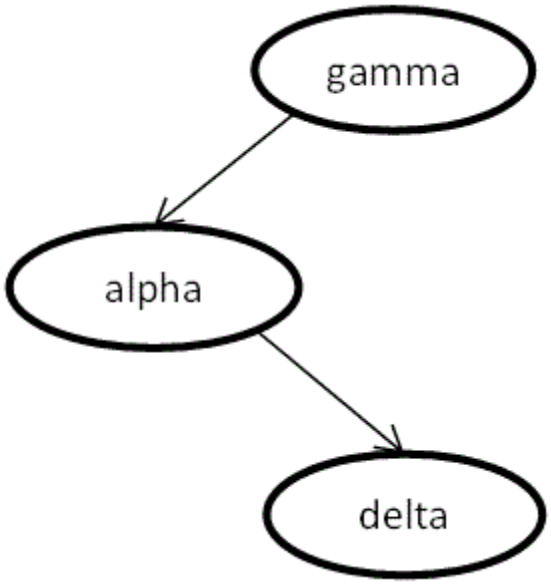


NOTE: Since each test case is a description of one or more binary trees, the description for the first test case could not have come from a configuration like the one below which is not a binary tree:



The second test case describes one of the configurations below. The tree on the right is a binary search tree because it fulfills the binary search tree property, (i.e. the key in each node of the tree is greater than all the keys stored in the left sub-tree, and smaller than all the keys in the right sub-tree). However, the configuration on the left is not a binary search tree because the key value stored at the root of the tree is not smaller than all the values stored in the left sub-tree (i.e. value "gamma" of the left subtree is greater than "delta").





The third test case describes a forest of two trees, which cannot be a BST.

The last test case cannot be a BST.