Codeforces Round #730 (Div. 2)

A. Exciting Bets

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Welcome to Rockport City!

It is time for your first ever race in the game against Ronnie. To make the race interesting, you have bet a dollars and Ronnie has bet b dollars. But the fans seem to be disappointed. The excitement of the fans is given by $gcd(a,b)$, where $gcd(x,y)$ denotes the greatest common divisor (GCD) of integers x and y. To make the race more exciting, you can perform two types of operations:

1. Increase both a and b by 1.
2. Decrease both a and b by 1. This operation can only be performed if both a and b are greater than 0.

In one move, you can perform any one of these operations. You can perform arbitrary (possibly zero) number of moves. Determine the maximum excitement the fans can get and the minimum number of moves required to achieve it.

Note that $gcd(x,0) = x$ for any $x \geq 0$.

Input

The first line of input contains a single integer t ($1 \leq t \leq 5 \cdot 10^3$) — the number of test cases.

The first and the only line of each test case contains two integers a and b ($0 \leq a, b \leq 10^{18}$).

Output

For each test case, print a single line containing two integers.

If the fans can get infinite excitement, print `0 0`.

Otherwise, the first integer must be the maximum excitement the fans can get, and the second integer must be the minimum number of moves required to achieve that excitement.

Example

input
4 8 5 1 2 4 4 3 9
output
3 1 1 0 0 0 6 3

Note

For the first test case, you can apply the first operation 1 time to get $a = 9$ and $b = 6$. It can be shown that 3 is the maximum excitement possible.

For the second test case, no matter how many operations you apply, the fans will always have an excitement equal to 1. Since the initial excitement is also 1, you don't need to apply any operation.

For the third case, the fans can get infinite excitement by applying the first operation an infinite amount of times.

For the fourth test case, you can apply the second operation 3 times to get $a = 0$ and $b = 6$. Since, $gcd(0,6) = 6$, the fans will get an excitement of 6.

B. Customising the Track

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Highway 201 is the most busy street in Rockport. Traffic cars cause a lot of hindrances to races, especially when there are a lot of them. The track which passes through this highway can be divided into n sub-tracks. You are given an array a where a_i represents

the number of traffic cars in the i-th sub-track. You define the inconvenience of the track as $\sum_{i=1}^{n} \sum_{j=i+1}^{n} |a_i - a_j|$, where $|x|$ is the absolute value of x.

You can perform the following operation any (possibly zero) number of times: choose a traffic car and move it from its current sub-track to any other sub-track.

Find the minimum inconvenience you can achieve.

Input
The first line of input contains a single integer t ($1 \le t \le 10\,000$) — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$).

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$).

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output
For each test case, print a single line containing a single integer: the minimum inconvenience you can achieve by applying the given operation any (possibly zero) number of times.

Example

input
3 3 1 2 3 4 0 1 1 0 10 8 3 6 11 5 2 1 7 10 4
output
0 4 21

Note
For the first test case, you can move a car from the 3-rd sub-track to the 1-st sub-track to obtain 0 inconvenience.

For the second test case, moving any car won't decrease the inconvenience of the track.

C. Need for Pink Slips

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

After defeating a Blacklist Rival, you get a chance to draw 1 reward slip out of x hidden valid slips. Initially, $x = 3$ and these hidden valid slips are Cash Slip, Impound Strike Release Marker and Pink Slip of Rival's Car. Initially, the probability of drawing these in a random guess are c, m, and p, respectively. There is also a volatility factor v. You can play any number of Rival Races as long as you don't draw a Pink Slip. Assume that you win each race and get a chance to draw a reward slip. In each draw, you draw one of the x valid items with their respective probabilities. Suppose you draw a particular item and its probability of drawing before the draw was a. Then,

- If the item was a Pink Slip, the quest is over, and you will not play any more races.
- Otherwise,
    1. If $a \le v$, the probability of the item drawn becomes 0 and the item is no longer a valid item for all the further draws, reducing x by 1. Moreover, the reduced probability a is distributed equally among the other remaining valid items.
    2. If $a > v$, the probability of the item drawn reduces by v and the reduced probability is distributed equally among the other valid items.

For example,

- If $(c, m, p) = (0.2, 0.1, 0.7)$ and $v = 0.1$, after drawing Cash, the new probabilities will be $(0.1, 0.15, 0.75)$.
- If $(c, m, p) = (0.1, 0.2, 0.7)$ and $v = 0.2$, after drawing Cash, the new probabilities will be $(Invalid, 0.25, 0.75)$.
- If $(c, m, p) = (0.2, Invalid, 0.8)$ and $v = 0.1$, after drawing Cash, the new probabilities will be $(0.1, Invalid, 0.9)$.
- If $(c, m, p) = (0.1, Invalid, 0.9)$ and $v = 0.2$, after drawing Cash, the new probabilities will be $(Invalid, Invalid, 1.0)$.

You need the cars of Rivals. So, you need to find the expected number of races that you must play in order to draw a pink slip.

Input
The first line of input contains a single integer t ($1 \le t \le 10$) — the number of test cases.

The first and the only line of each test case contains four real numbers c, m, p and v ($0 < c, m, p < 1$, $c + m + p = 1$,

$0.1 \le v \le 0.9$).

Additionally, it is guaranteed that each of c, m, p and v have at most 4 decimal places.

Output
For each test case, output a single line containing a single real number — the expected number of races that you must play in order to draw a Pink Slip.

Your answer is considered correct if its absolute or relative error does not exceed 10^{-6}.

Formally, let your answer be a, and the jury's answer be b. Your answer is accepted if and only if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-6}$.

Example

input
4 0.2 0.2 0.6 0.2 0.4 0.2 0.4 0.8 0.4998 0.4998 0.0004 0.1666 0.3125 0.6561 0.0314 0.2048
output
1.532000000000 1.860000000000 5.005050776521 4.260163673896

Note
For the first test case, the possible drawing sequences are:

- P with a probability of 0.6;
- CP with a probability of $0.2 \cdot 0.7 = 0.14$;
- CMP with a probability of $0.2 \cdot 0.3 \cdot 0.9 = 0.054$;
- CMMP with a probability of $0.2 \cdot 0.3 \cdot 0.1 \cdot 1 = 0.006$;
- MP with a probability of $0.2 \cdot 0.7 = 0.14$;
- MCP with a probability of $0.2 \cdot 0.3 \cdot 0.9 = 0.054$;
- MCCP with a probability of $0.2 \cdot 0.3 \cdot 0.1 \cdot 1 = 0.006$.

So, the expected number of races is equal to $1 \cdot 0.6 + 2 \cdot 0.14 + 3 \cdot 0.054 + 4 \cdot 0.006 + 2 \cdot 0.14 + 3 \cdot 0.054 + 4 \cdot 0.006 = 1.532$.
For the second test case, the possible drawing sequences are:

- P with a probability of 0.4;
- CP with a probability of $0.4 \cdot 0.6 = 0.24$;
- CMP with a probability of $0.4 \cdot 0.4 \cdot 1 = 0.16$;
- MP with a probability of $0.2 \cdot 0.5 = 0.1$;
- MCP with a probability of $0.2 \cdot 0.5 \cdot 1 = 0.1$.

So, the expected number of races is equal to $1 \cdot 0.4 + 2 \cdot 0.24 + 3 \cdot 0.16 + 2 \cdot 0.1 + 3 \cdot 0.1 = 1.86$.

D1. RPD and Rap Sheet (Easy Version)

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is the easy version of the problem. The only difference is that here $k = 2$. You can make hacks only if both the versions of the problem are solved.

This is an interactive problem.

Every decimal number has a base k equivalent. The individual digits of a base k number are called k-its. Let's define the k-itwise XOR of two k-its a and b as $(a + b) \bmod k$.

The k-itwise XOR of two base k numbers is equal to the new number formed by taking the k-itwise XOR of their corresponding k-its. The k-itwise XOR of two decimal numbers a and b is denoted by $a \oplus_k b$ and is equal to the decimal representation of the k-itwise XOR of the base k representations of a and b. All further numbers used in the statement below are in decimal unless specified. When $k = 2$ (it is always true in this version), the k-itwise XOR is the same as the bitwise XOR.

You have hacked the criminal database of Rockport Police Department (RPD), also known as the Rap Sheet. But in order to access it, you require a password. You don't know it, but you are quite sure that it lies between 0 and $n - 1$ inclusive. So, you have decided to guess it. Luckily, you can try at most n times without being blocked by the system. But the system is adaptive. Each time you make an incorrect guess, it changes the password. Specifically, if the password before the guess was x, and you guess a different number y, then the system changes the password to a number z such that $x \oplus_k z = y$. Guess the password and break into the system.

Input
The first line of input contains a single integer t ($1 \le t \le 10\,000$) denoting the number of test cases. t test cases follow.

The first line of each test case contains two integers n ($1 \le n \le 2 \cdot 10^5$) and k ($k = 2$).

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Interaction

For each test case, first read two integers n and k. Then you may ask up to n queries.

For each query, print a single integer y ($0 \le y \le 2 \cdot 10^7$). Let the current password be x. After that, read an integer r.

If $x = y$, you will read $r = 1$ and the test case is solved. You must then continue solving the remaining test cases.

Else, you will read $r = 0$. At this moment the password is changed to a number z such that $x \oplus_k z = y$.

After printing a query, do not forget to output the end of line and flush the output. Otherwise, you will get the `Idleness limit` exceeded verdict.

To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

If you ask an invalid query or exceed n queries, you will read $r = -1$ and you will receive the `Wrong Answer` verdict. Make sure to exit immediately to avoid unexpected verdicts.

Note that the interactor is **adaptive**. That is, the original password is not fixed in the beginning and may depend on your queries. But it is guaranteed that at any moment there is at least one initial password such that all the answers to the queries are consistent.

Hacks:

To use hacks, use the following format of tests:

The first line should contain a single integer t ($1 \le t \le 10\,000$) — the number of test cases.

The first and only line of each test case should contain two integers n ($1 \le n \le 2 \cdot 10^5$) and k ($k = 2$) denoting the number of queries and the base respectively. The optimal original password is automatically decided by the adaptive interactor.

You must ensure that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Example

input
1 5 2 0 0 1
output
3 4 5

Note

In the example test case, the hidden password is 2.

The first query is 3. It is not equal to the current password. So, 0 is returned, and the password is changed to 1 since $2 \oplus_2 1 = 3$.

The second query is 4. It is not equal to the current password. So, 0 is returned, and the password is changed to 5 since $1 \oplus_2 5 = 4$.

The third query is 5. It is equal to the current password. So, 1 is returned, and the job is done.

Note that this initial password is taken just for the sake of explanation. When you submit, the interactor might behave differently because it is adaptive.

D2. RPD and Rap Sheet (Hard Version)

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is the hard version of the problem. The only difference is that here $2 \le k \le 100$. You can make hacks only if both

the versions of the problem are solved.

This is an interactive problem!

Every decimal number has a base k equivalent. The individual digits of a base k number are called k-its. Let's define the k-itwise XOR of two k-its a and b as $(a + b) \bmod k$.

The k-itwise XOR of two base k numbers is equal to the new number formed by taking the k-itwise XOR of their corresponding k-its. The k-itwise XOR of two decimal numbers a and b is denoted by $a \oplus_k b$ and is equal to the decimal representation of the k-itwise XOR of the base k representations of a and b. All further numbers used in the statement below are in decimal unless specified.

You have hacked the criminal database of Rockport Police Department (RPD), also known as the Rap Sheet. But in order to access it, you require a password. You don't know it, but you are quite sure that it lies between 0 and $n - 1$ inclusive. So, you have decided to guess it. Luckily, you can try at most n times without being blocked by the system. But the system is adaptive. Each time you make an incorrect guess, it changes the password. Specifically, if the password before the guess was x, and you guess a different number y, then the system changes the password to a number z such that $x \oplus_k z = y$. Guess the password and break into the system.

Input

The first line of input contains a single integer t ($1 \le t \le 10\,000$) denoting the number of test cases. t test cases follow.

The first line of each test case contains two integers n ($1 \le n \le 2 \cdot 10^5$) and k ($2 \le k \le 100$).

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Interaction

For each test case, first read two integers n and k. Then you may ask up to n queries.

For each query, print a single integer y ($0 \le y \le 2 \cdot 10^7$). Let the current password be x. After that, read an integer r.

If $x = y$, you will read $r = 1$ and the test case is solved. You must then continue solving the remaining test cases.

Else, you will read $r = 0$. At this moment the password is changed to a number z such that $x \oplus_k z = y$.

After printing a query, do not forget to output the end of line and flush the output. Otherwise, you will get the `Idleness limit exceeded` verdict.

To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

If you ask an invalid query or exceed n queries, you will read $r = -1$ and you will receive the `Wrong Answer` verdict. Make sure to exit immediately to avoid unexpected verdicts.

Note that the interactor is **adaptive**. That is, the original password is not fixed in the beginning and may depend on your queries. But it is guaranteed that at any moment there is at least one initial password such that all the answers to the queries are consistent.

Hacks:

To use hacks, use the following format of tests:

The first line should contain a single integer t ($1 \le t \le 10\,000$) — the number of test cases.

The first and only line of each test case should contain two integers n ($1 \le n \le 2 \cdot 10^5$) and k ($2 \le k \le 100$) denoting the number of queries and the base respectively. The optimal original password is automatically decided by the adaptive interactor.

You must ensure that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Example

input
2 5 2 0 0 1 5 3 0 0 1
output
3

```
4

5


1

4

6
```

Note

Test Case 1:

In this case, the hidden password is 2.

The first query is 3. It is not equal to the current password. So, 0 is returned, and the password is changed to 1 since $2 \oplus_2 1 = 3$.

The second query is 4. It is not equal to the current password. So, 0 is returned, and the password is changed to 5 since $1 \oplus_2 5 = 4$.

The third query is 5. It is equal to the current password. So, 1 is returned, and the job is done.

Test Case 2:

In this case, the hidden password is 3.

The first query is 1. It is not equal to the current password. So, 0 is returned, and the password is changed to 7 since $3 \oplus_3 7 = 1$. $[3 = (10)_3, 7 = (21)_3, 1 = (01)_3$ and $(10)_3 \oplus_3 (21)_3 = (01)_3]$.

The second query is 4. It is not equal to the current password. So, 0 is returned, and the password is changed to 6 since $7 \oplus_3 6 = 4$. $[7 = (21)_3, 6 = (20)_3, 4 = (11)_3$ and $(21)_3 \oplus_3 (20)_3 = (11)_3]$.

The third query is 6. It is equal to the current password. So, 1 is returned, and the job is done.

Note that these initial passwords are taken just for the sake of explanation. In reality, the grader might behave differently because it is adaptive.

E. The Final Pursuit

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

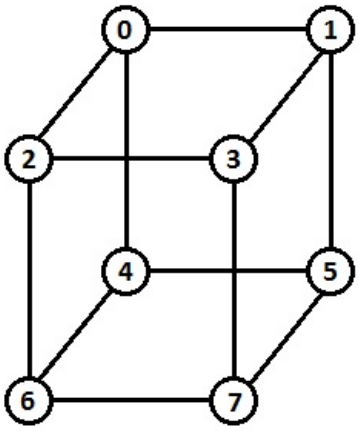
Finally, you have defeated Razor and now, you are the Most Wanted street racer. Sergeant Cross has sent the full police force after you in a deadly pursuit. Fortunately, you have found a hiding spot but you fear that Cross and his force will eventually find you. To increase your chances of survival, you want to tune and repaint your BMW M3 GTR.

The car can be imagined as a *permuted* n-dimensional hypercube. A simple n-dimensional hypercube is an undirected unweighted graph built recursively as follows:

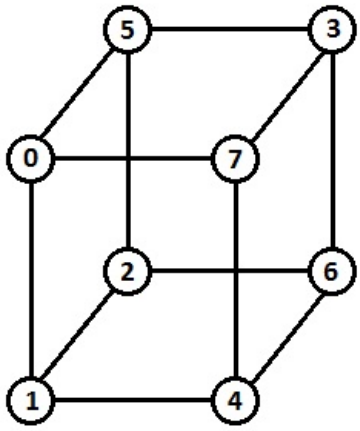
- Take two simple $(n-1)$-dimensional hypercubes one having vertices numbered from 0 to $2^{n-1}-1$ and the other having vertices numbered from 2^{n-1} to 2^n-1. A simple 0-dimensional Hypercube is just a single vertex.
- Add an edge between the vertices i and $i+2^{n-1}$ for each $0 \leq i < 2^{n-1}$.

A permuted n-dimensional hypercube is formed by permuting the vertex numbers of a simple n-dimensional hypercube in any arbitrary manner.

Examples of a simple and permuted 3-dimensional hypercubes are given below:



A simple 3-D Hypercube



A permuted 3-D Hypercube
The Permutation used is
[5, 3, 0, 7, 2, 6, 1, 4]

Note that a permuted n-dimensional hypercube has the following properties:

- There are exactly 2^n vertices.
- There are exactly $n \cdot 2^{n-1}$ edges.
- Each vertex is connected to exactly n other vertices.
- There are no self-loops or duplicate edges.

Let's denote the permutation used to generate the permuted n-dimensional hypercube, representing your car, from a simple n-dimensional hypercube by P. Before messing up the functionalities of the car, you want to find this permutation so that you can restore the car if anything goes wrong. But the job isn't done yet.

You have n different colours numbered from 0 to $n-1$. You want to colour the vertices of this permuted n-dimensional hypercube in such a way that for each and every vertex u satisfying $0 \leq u < 2^n$ and for each and every colour c satisfying $0 \leq c < n$, there is at least one vertex v adjacent to u having a colour c. In other words, from each and every vertex, it must be possible to reach a vertex of any colour by just moving to an adjacent vertex.

Given the permuted n-dimensional hypercube, find any valid permutation P and colouring.

Input

The first line of input contains a single integer t ($1 \leq t \leq 4096$) — the number of test cases.

For each test case, the first line contains a single integer n ($1 \leq n \leq 16$).

Each of the next $n \cdot 2^{n-1}$ lines contain two integers u and v ($0 \leq u, v < 2^n$) denoting that there is an edge between the vertices numbered u and v.

It is guaranteed that the graph described in the input is a permuted n-dimensional hypercube.

Additionally, it is guaranteed that the sum of 2^n over all test cases does not exceed $2^{16} = 65\,536$.

Output

For each test case, print two lines.

In the first line, output any permutation P of length 2^n that can be used to transform a simple n-dimensional hypercube to the permuted n-dimensional hypercube given in the input. Two permuted hypercubes are considered the same if they have the same set of edges. If there are multiple answers, output any of them.

In the second line, print the colouring. If there is no way to colour the vertices satisfying the conditions, output -1. Otherwise, output a single line containing 2^n space separated integers. The i-th integer must be the colour of the vertex numbered $(i-1)$ in the permuted n-dimensional hypercube. If there are multiple answers, output any of them.

Example

input
3 1 0 1 2 0 1 1 2 2 3 3 0 3 0 1 0 5 0 7 1 2 1 4 2 5

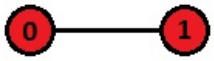
```
2 6
3 5
3 6
3 7
4 6
4 7
```

output

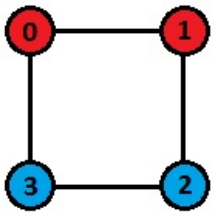
```
0 1
0 0
0 1 3 2
0 0 1 1
5 3 0 7 2 6 1 4
-1
```

Note

The colouring and the permuted hypercube for the first test case is shown below:



The colouring and the permuted hypercube for the second test case is shown below:



The permuted hypercube for the third test case is given in the problem statement. However, it can be shown that there exists no way to colour that cube satifying all the conditions. Note that some other permutations like $[0, 5, 7, 3, 1, 2, 4, 6]$ and $[0, 1, 5, 2, 7, 4, 3, 6]$ will also give the same permuted hypercube.