A. Closing The Gap

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n block towers in a row, where tower i has a height of a_i. You're part of a building crew, and you want to make the buildings look as nice as possible. In a single day, you can perform the following operation:

- Choose two indices i and j ($1 \leq i, j \leq n$; $i \neq j$), and move a block from tower i to tower j. This essentially decreases a_i by 1 and increases a_j by 1.

You think the ugliness of the buildings is the height difference between the tallest and shortest buildings. Formally, the ugliness is defined as $\max(a) - \min(a)$.

What's the minimum possible ugliness you can achieve, after any number of days?

Input

The first line contains one integer t ($1 \leq t \leq 1000$) — the number of test cases. Then t cases follow.

The first line of each test case contains one integer n ($2 \leq n \leq 100$) — the number of buildings.

The second line of each test case contains n space separated integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^7$) — the heights of the buildings.

Output

For each test case, output a single integer — the minimum possible ugliness of the buildings.

Example

input

```
3
3
10 10 10
4
3 2 1 2
5
1 2 3 1 5
```

output

```
0
0
1
```

Note

In the first test case, the ugliness is already 0.

In the second test case, you should do one operation, with $i = 1$ and $j = 3$. The new heights will now be $[2, 2, 2, 2]$, with an ugliness of 0.

In the third test case, you may do three operations:

1. with $i = 3$ and $j = 1$. The new array will now be $[2, 2, 2, 1, 5]$,
2. with $i = 5$ and $j = 4$. The new array will now be $[2, 2, 2, 2, 4]$,
3. with $i = 5$ and $j = 3$. The new array will now be $[2, 2, 3, 2, 3]$.

The resulting ugliness is 1. It can be proven that this is the minimum possible ugliness for this test.

B. And It's Non-Zero

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array consisting of all integers from $[l, r]$ inclusive. For example, if $l = 2$ and $r = 5$, the array would be $[2, 3, 4, 5]$. What's the minimum number of elements you can delete to make the bitwise AND of the array non-zero?

A *bitwise AND* is a binary operation that takes two equal-length binary representations and performs the *AND* operation on each pair of the corresponding bits.

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases. Then t cases follow.

The first line of each test case contains two integers l and r ($1 \le l \le r \le 2 \cdot 10^5$) — the description of the array.

Output

For each test case, output a single integer — the answer to the problem.

Example

input

```
5
1 2
2 8
4 5
1 5
100000 200000
```

output

```
1
3
0
2
31072
```

Note

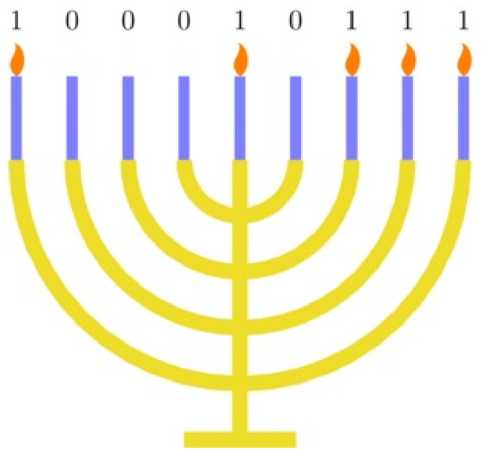
In the first test case, the array is $[1, 2]$. Currently, the *bitwise AND* is 0, as $1\ \&\ 2 = 0$. However, after deleting 1 (or 2), the array becomes $[2]$ (or $[1]$), and the *bitwise AND* becomes 2 (or 1). This can be proven to be the optimal, so the answer is 1.

In the second test case, the array is $[2, 3, 4, 5, 6, 7, 8]$. Currently, the *bitwise AND* is 0. However, after deleting 4, 5, and 8, the array becomes $[2, 3, 6, 7]$, and the *bitwise AND* becomes 2. This can be proven to be the optimal, so the answer is 3. Note that there may be other ways to delete 3 elements.

C. Menorah

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n candles on a Hanukkah menorah, and some of its candles are initially lit. We can describe which candles are lit with a binary string s, where the i-th candle is lit if and only if $s_i = 1$.



Initially, the candle lights are described by a string a. In an operation, you select a candle that is **currently lit**. By doing so, the candle you selected will remain lit, and every other candle will change (if it was lit, it will become unlit and if it was unlit, it will become lit).

You would like to make the candles look the same as string b. Your task is to determine if it is possible, and if it is, find the minimum number of operations required.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases. Then t cases follow.

The first line of each test case contains a single integer n ($1 \le n \le 10^5$) — the number of candles.

The second line contains a string a of length n consisting of symbols `0` and `1` — the initial pattern of lights.

The third line contains a string b of length n consisting of symbols `0` and `1` — the desired pattern of lights.

It is guaranteed that the sum of n does not exceed 10^5.

Output

For each test case, output the minimum number of operations required to transform a to b, or -1 if it's impossible.

Example

input

```
5
5
11010
11010
2
01
11
3
000
101
9
100010111
101101100
9
001011011
011010101
```

output

```
0
1
-1
3
4
```

Note

In the first test case, the two strings are already equal, so we don't have to perform any operations.

In the second test case, we can perform a single operation selecting the second candle to transform 01 into 11.

In the third test case, it's impossible to perform any operations because there are no lit candles to select.

In the fourth test case, we can perform the following operations to transform a into b:

1. Select the 7-th candle: $100010111 \to 011101100$.
2. Select the 2-nd candle: $011101100 \to 110010011$.
3. Select the 1-st candle: $110010011 \to 101101100$.

In the fifth test case, we can perform the following operations to transform a into b:

1. Select the 6-th candle: $001011011 \to 110101100$
2. Select the 2-nd candle: $110101100 \to 011010011$
3. Select the 8-th candle: $011010011 \to 100101110$
4. Select the 7-th candle: $100101110 \to 011010101$

D. X(or)-mas Tree

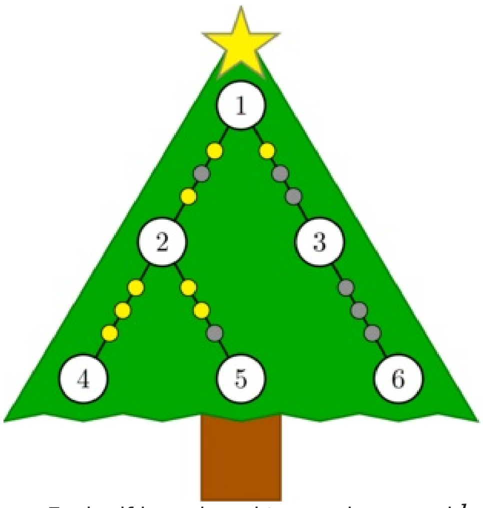
time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

'Twas the night before Christmas, and Santa's frantically setting up his new Christmas tree! There are n nodes in the tree, connected by $n-1$ edges. On each edge of the tree, there's a set of Christmas lights, which can be represented by an integer in binary representation.



He has m elves come over and admire his tree. Each elf is assigned two nodes, a and b, and that elf looks at all lights on the simple path between the two nodes. After this, the elf's favorite number becomes the bitwise XOR of the values of the lights on the edges in that path.

However, the North Pole has been recovering from a nasty bout of flu. Because of this, Santa forgot some of the configurations of lights he had put on the tree, and he has already left the North Pole! Fortunately, the elves came to the rescue, and each one told Santa what pair of nodes he was assigned (a_i, b_i), as well as **the parity** of the number of set bits in his favorite number. In other words, he remembers whether the number of 1's when his favorite number is written in binary is **odd or even**.

Help Santa determine if it's possible that the memories are consistent, and if it is, remember what his tree looked like, and maybe you'll go down in history!

Input

The first line contains one integer t ($1 \leq t \leq 2 \cdot 10^4$) — the number of test cases. Then t cases follow.

The first line of each test case contains two integers, n and m ($2 \leq n \leq 2 \cdot 10^5$; $1 \leq m \leq 2 \cdot 10^5$) — the size of tree and the number of elves respectively.

The next $n-1$ lines of each test case each contains three integers, x, y, and v ($1 \leq x, y \leq n$; $-1 \leq v < 2^{30}$) — meaning that there's an edge between nodes x and y. If

- $v = -1$: Santa doesn't remember what the set of lights were on for this edge.
- $v \geq 0$: The set of lights on the edge is v.

The next m lines of each test case each contains three integers, a, b, and p ($1 \leq a, b \leq n$; $a \neq b$; $0 \leq p \leq 1$) — the nodes that the elf was assigned to, and the **parity** of the number of set bits in the elf's favorite number.

It is guaranteed that the sum of all n and the sum of all m don't exceed $2 \cdot 10^5$ each.

It is guaranteed that the given edges form a tree.

Output

For each test case, first print either YES or NO (in any case), whether there's a tree consistent with Santa's memory or not.

If the answer is YES, print $n-1$ lines each containing three integers: x, y, and v ($1 \leq x, y \leq n$; $0 \leq v < 2^{30}$) — the edge and the integer on that edge. The set of edges must be the same as in the input, and if the value of some edge was specified earlier, it can not change. You can print the edges in any order.

If there are multiple answers, print any.

Example

input

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

output

```
YES
1 2 0
1 3 1
2 4 7
3 6 0
2 5 0
YES
1 2 1
1 3 0
1 4 1
4 5 1
NO
NO
```

Note

The first test case is the image in the statement.

One possible answer is assigning the value of the edge $(1, 2)$ to 5, and the value of the edge $(2, 5)$ to 3. This is correct because:

- The first elf goes from node 2 to node 3. This elf's favorite number is 4, so he remembers the value 1 (as 4 has an odd number of 1 bits in its binary representation).
- The second elf goes from node 2 to node 5. This elf's favorite number is 3, so he remembers the value 0 (as 3 has an even number of 1 bits in its binary representation).
- The third elf goes from node 5 to node 6. This elf's favorite number is 7, so he remembers the value 1 (as 7 has an odd number of 1 bits in its binary representation).
- The fourth elf goes from node 6 to node 1. This elf's favorite number is 1, so he remembers the value 1 (as 1 has an odd number of 1 bits in its binary representation).
- The fifth elf goes from node 4 to node 5. This elf's favorite number is 4, so he remembers the number 1 (as 4 has an odd number of 1 bits in its binary representation).

Note that there are other possible answers.

E. Purple Crayon

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Two players, Red and Blue, are at it again, and this time they're playing with crayons! The mischievous duo is now vandalizing a rooted tree, by coloring the nodes while playing their favorite game.

The game works as follows: there is a tree of size n, rooted at node 1, where each node is initially white. Red and Blue get **one turn each**. Red goes first.

In Red's turn, he can do the following operation **any number of times**:

- Pick any subtree of the rooted tree, and color every node in the subtree red.

However, to make the game fair, Red is only allowed to color k nodes of the tree. In other words, after Red's turn, at most k of the nodes can be colored red.
Then, it's Blue's turn. Blue can do the following operation **any number of times**:

- Pick any subtree of the rooted tree, and color every node in the subtree blue. However, he's not allowed to choose a subtree that contains a node already colored red, as that would make the node purple and no one likes purple crayon.

Note: there's no restriction on the number of nodes Blue can color, as long as he doesn't color a node that Red has already colored.
After the two turns, the score of the game is determined as follows: let w be the number of white nodes, r be the number of red nodes, and b be the number of blue nodes. The score of the game is $w \cdot (r - b)$.

Red wants to maximize this score, and Blue wants to minimize it. If both players play optimally, what will the final score of the game be?

Input

The first line contains two integers n and k ($2 \le n \le 2 \cdot 10^5$; $1 \le k \le n$) — the number of vertices in the tree and the maximum number of red nodes.

Next $n - 1$ lines contains description of edges. The i-th line contains two space separated integers u_i and v_i ($1 \le u_i, v_i \le n$; $u_i \ne v_i$) — the i-th edge of the tree.

It's guaranteed that given edges form a tree.

Output

Print one integer — the resulting score if both Red and Blue play optimally.

Examples

input

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

output

```
1
```

input

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

output

```
6
```

input

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

output

```
4
```

input

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

output

```
-1
```

Note

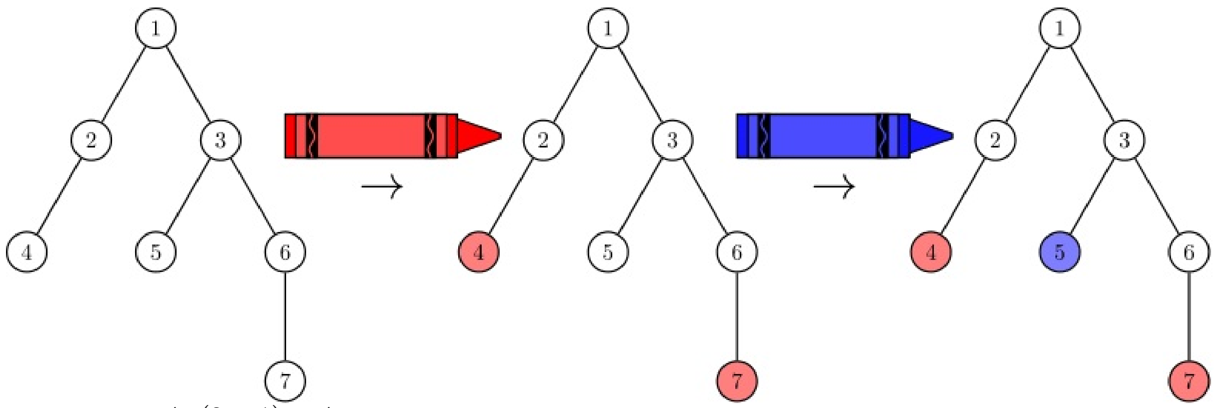
In the first test case, the optimal strategy is as follows:

- Red chooses to color the subtrees of nodes 2 and 3.
- Blue chooses to color the subtree of node 4.

At the end of this process, nodes 2 and 3 are red, node 4 is blue, and node 1 is white. The score of the game is $1 \cdot (2 - 1) = 1$.
In the second test case, the optimal strategy is as follows:

- Red chooses to color the subtree of node 4. This colors both nodes 4 and 5.
- Blue does not have any options, so nothing is colored blue.

At the end of this process, nodes 4 and 5 are red, and nodes 1, 2 and 3 are white. The score of the game is $3 \cdot (2 - 0) = 6$.
For the third test case:



The score of the game is $4 \cdot (2 - 1) = 4$.

F. LEGOndary Grandmaster

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

After getting bored by playing with crayons, you decided to switch to Legos! Today, you're working with a long strip, with height 1 and length n, some positions of which are occupied by 1 by 1 Lego pieces.

In one second, you can either remove two *adjacent* Lego pieces from the strip (if both are present), or add two Lego pieces to *adjacent* positions (if both are absent). You can only add or remove Lego's at two adjacent positions at the same time, as otherwise your chubby fingers run into precision issues.

You want to know exactly how much time you'll spend playing with Legos. You value efficiency, so given some starting state and some ending state, you'll always spend the least number of seconds to transform the starting state into the ending state. If it's impossible to transform the starting state into the ending state, you just skip it (so you spend 0 seconds).

The issue is that, for some positions, you don't remember whether there were Legos there or not (in either the starting state, the ending state, or both). Over all pairs of (starting state, ending state) that are consistent with your memory, find the total amount of time it will take to transform the starting state to the ending state. Print this value modulo $1\,000\,000\,007$ ($10^9 + 7$).

Input

The first line contains one integer t ($1 \le t \le 1000$) — the number of test cases. Then t cases follow.

The first line of each test case contains one integer n ($2 \le n \le 2000$) — the size of the Lego strip.

The second line of each test case contains a string s of length n, consisting of the characters `0`, `1`, and `?` — your memory of the starting state:

- `1` represents a position that definitely has a Lego piece,
- `0` represents a position that definitely does not have a Lego piece,
- and `?` represents a position that you don't remember.

The third line of each test case contains a string t of length n, consisting of the characters `0`, `1`, and `?` — your memory of the ending state. It follows a similar format to the starting state.

It's guaranteed that the sum of n over all test cases doesn't exceed 2000.

Output

For each test case, output a single integer — the answer to the problem modulo $1\,000\,000\,007$ ($10^9 + 7$).

Example

input

```
6
2
00
11
3
???
???
3
??1
0?0
4
??0?
??11
5
?????
0??1?
10
?01??01?1?
??100?1???
```

output

```
1
16
1
14
101
1674
```

Note

For the first test case, 00 is the only possible starting state, and 11 is the only possible ending state. It takes exactly one operation to change 00 to 11.

For the second test case, some of the possible starting and ending state pairs are:

- $(000, 011)$ — takes 1 operation.
- $(001, 100)$ — takes 2 operations.
- $(010, 000)$ — takes 0 operations, as it's impossible to achieve the ending state.

G. Maximum Adjacent Pairs

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given an array a consisting of n non-negative integers.

You have to replace each 0 in a with an integer from 1 to n (different elements equal to 0 can be replaced by different integers).

The *value* of the array you obtain is the number of integers k from 1 to n such that the following condition holds: there exist a pair of adjacent elements equal to k (i. e. there exists some $i \in [1, n-1]$ such that $a_i = a_{i+1} = k$). If there are multiple such pairs for some integer k, this integer is counted in the *value* only once.

Your task is to obtain the array with the maximum possible *value*.

Input

The first line contains one integer n ($2 \le n \le 3 \cdot 10^5$) — the number of elements in the array.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($0 \le a_i \le \min(n, 600)$) — the elements of the array.

Output

Print n integers not less than 1 and not greater than n — the array with the maximum possible *value* you can obtain.

If there are multiple answers, print any of them.

Examples

input
```
4
1 1 0 2
```
output
```
1 1 2 2
```

input
```
5
0 0 0 0 0
```
output
```
3 1 1 3 3
```

input
```
5
1 2 3 4 5
```
output
```
1 2 3 4 5
```

input
```
6
1 0 0 0 0 1
```
output
```
1 2 3 3 1 1
```

input
```
3
3 0 2
```
output
```
3 2 2
```

input
```
5
1 0 2 0 1
```
output
```
1 2 2 1 1
```

input
```
7
1 0 2 3 1 0 2
```
output
```
1 2 2 3 1 1 2
```

H. Reindeer Games

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n reindeer at the North Pole, all battling for the highest spot on the "Top Reindeer" leaderboard on the front page of CodeNorses (a popular competitive reindeer gaming website). Interestingly, the "Top Reindeer" title is just a measure of upvotes and has nothing to do with their skill level in the reindeer games, but they still give it the utmost importance.

Currently, the i-th reindeer has a score of a_i. You would like to influence the leaderboard with some operations. In an operation, you can choose a reindeer, and either increase or decrease his score by 1 unit. Negative scores are allowed.

You have m requirements for the resulting scores. Each requirement is given by an ordered pair (u, v), meaning that after all operations, the score of reindeer u must be less than or equal to the score of reindeer v.

Your task is to perform the minimum number of operations so that all requirements will be satisfied.

Input

The first line contains two integers n and m ($2 \leq n \leq 1000$; $1 \leq m \leq 1000$) — the number of reindeer and requirements, respectively.

The second line contains n integers $a_1, \ldots, a_n$ ($1 \leq a_i \leq 10^9$), where a_i is the current score of reindeer i.

The next m lines describe the requirements.

The i-th of these lines contains two integers u_i and v_i ($1 \leq u_i, v_i \leq n$; $u_i \neq v_i$) — the two reindeer of the i-th requirement.

Output

Print n integers $b_1, \ldots, b_n$ ($-10^{15} \leq b_i \leq 10^{15}$), where b_i is the score of the i-th reindeer after all operations.

If there are multiple solutions achieving the minimum number of operations, you may output any.

We can prove that there is always an optimal solution such that $|b_i| \leq 10^{15}$ for all i.

Examples

input

```
7 6
3 1 4 9 2 5 6
1 2
2 3
3 4
4 5
5 6
6 7
```

output

```
1 1 4 4 4 5 6
```

input

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

output

```
6 6 6 2
```

input

```
10 18
214 204 195 182 180 176 176 172 169 167
1 2
3 2
4 2
5 2
6 2
7 2
8 2
9 2
10 2
6 1
6 2
6 3
6 4
6 5
6 7
6 8
6 9
6 10
```

output

```
204 204 195 182 180 167 176 172 169 167
```