Educational Codeforces Round 98 (Rated for Div. 2)

A. Robot Program

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

There is an infinite 2-dimensional grid. The robot stands in cell $(0, 0)$ and wants to reach cell (x, y). Here is a list of possible commands the robot can execute:

- move north from cell (i, j) to $(i, j + 1)$;
- move east from cell (i, j) to $(i + 1, j)$;
- move south from cell (i, j) to $(i, j - 1)$;
- move west from cell (i, j) to $(i - 1, j)$;
- stay in cell (i, j).

The robot wants to reach cell (x, y) in as few commands as possible. However, he can't execute the same command two or more times in a row.

What is the minimum number of commands required to reach (x, y) from $(0, 0)$?

Input

The first line contains a single integer t ($1 \le t \le 100$) — the number of testcases.

Each of the next t lines contains two integers x and y ($0 \le x, y \le 10^4$) — the destination coordinates of the robot.

Output

For each testcase print a single integer — the minimum number of commands required for the robot to reach (x, y) from $(0, 0)$ if no command is allowed to be executed two or more times in a row.

Example

input
5 5 5 3 4 7 1 0 0 2 0
output
10 7 13 0 3

Note

The explanations for the example test:

We use characters `N`, `E`, `S`, `W` and `0` to denote going north, going east, going south, going west and staying in the current cell, respectively.

In the first test case, the robot can use the following sequence: `NENENENENE`.

In the second test case, the robot can use the following sequence: `NENENEN`.

In the third test case, the robot can use the following sequence: `ESENENE0ENESE`.

In the fourth test case, the robot doesn't need to go anywhere at all.

In the fifth test case, the robot can use the following sequence: `E0E`.

B. Toy Blocks

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

You are asked to watch your nephew who likes to play with toy blocks in a strange way.

He has n boxes and the i-th box has a_i blocks. His game consists of two steps:

1. he chooses an arbitrary box i;
2. he tries to move **all** blocks from the i-th box to other boxes.

If he can make the same number of blocks in each of $n - 1$ other boxes then he will be happy, otherwise, will be sad. Note that your nephew can only move the blocks from the chosen box to the other boxes; he cannot move blocks from the other boxes.
You don't want to make your nephew sad, so you decided to put several extra blocks into some boxes in such a way that no matter which box i he chooses he won't be sad. What is the minimum number of extra blocks you need to put?

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains the integer n ($2 \le n \le 10^5$) — the number of boxes.

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

It's guaranteed that the sum of n over test cases doesn't exceed 10^5.

Output
For each test case, print a single integer — the minimum number of blocks you need to put. It can be proved that the answer always exists, i. e. the number of blocks is finite.

Example

input
3 3 3 2 2 4 2 2 3 2 3 0 3 0
output
1 0 3

Note
In the first test case, you can, for example, put one extra block into the first box and make $a = [4, 2, 2]$. If your nephew chooses the box with 4 blocks, then we will move two blocks to the second box and two blocks to the third box. If he chooses the box with 2 blocks then he will move these two blocks to the other box with 2 blocks.

In the second test case, you don't need to put any extra blocks, since no matter which box your nephew chooses, he can always make other boxes equal.

In the third test case, you should put 3 extra blocks. For example, you can put 2 blocks in the first box and 1 block in the third box. You'll get array $a = [2, 3, 1]$.

C. Two Brackets

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

You are given a string s, consisting of brackets of two types: '(', ')', '[' and ']'.

A string is called a regular bracket sequence (RBS) if it's of one of the following types:

- empty string;
- '(' + RBS + ')';
- '[' + RBS + ']';
- RBS + RBS.

where plus is a concatenation of two strings.

In one move you can choose a non-empty subsequence of the string s (not necessarily consecutive) that is an RBS, remove it from the string and concatenate the remaining parts without changing the order.

What is the maximum number of moves you can perform?

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of testcases.

Each of the next t lines contains a non-empty string, consisting only of characters '(', ')', '[' and ']'. The total length of the strings over all testcases doesn't exceed $2 \cdot 10^5$.

Output
For each testcase print a single integer — the maximum number of moves you can perform on a given string s.

Example

input
5 () []() ([)])][( )[(]

output
1 2 2 0 1

Note
In the first example you can just erase the whole string.

In the second example you can first erase the brackets on positions 1 and 2: "**[]**()", then "()" is left. After that you can erase it whole. You could erase the whole string from the beginning but you would get one move instead of two.

In the third example you can first erase the brackets on positions 1 and 3: "**(**[**)**]". They form an RBS "()". Then "[]" is left, so you can erase it whole.

In the fourth example there is no subsequence that is an RBS, so you can't perform a move at all.

In the fifth example you can erase the brackets on positions 2 and 4: ")**[**(**]**" and get ")(" as a result. You can erase nothing from it.

D. Radio Towers

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

There are $n + 2$ towns located on a coordinate line, numbered from 0 to $n + 1$. The i-th town is located at the point i.

You build a radio tower in each of the towns $1, 2, \ldots, n$ with probability $\frac{1}{2}$ (these events are independent). After that, you want to set the signal power on each tower to some integer from 1 to n (signal powers are not necessarily the same, but also not necessarily different). The signal from a tower located in a town i with signal power p reaches every city c such that $|c - i| < p$.

After building the towers, you want to choose signal powers in such a way that:

- towns 0 and $n + 1$ don't get any signal from the radio towers;
- towns $1, 2, \ldots, n$ get signal from *exactly* one radio tower each.

For example, if $n = 5$, and you have built the towers in towns 2, 4 and 5, you may set the signal power of the tower in town 2 to 2, and the signal power of the towers in towns 4 and 5 to 1. That way, towns 0 and $n + 1$ don't get the signal from any tower, towns 1, 2 and 3 get the signal from the tower in town 2, town 4 gets the signal from the tower in town 4, and town 5 gets the signal from the tower in town 5.

Calculate the probability that, after building the towers, you will have a way to set signal powers to meet all constraints.

Input
The first (and only) line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$).

Output
Print one integer — the probability that there will be a way to set signal powers so that all constraints are met, taken modulo 998244353.

Formally, the probability can be expressed as an irreducible fraction $\frac{x}{y}$. You have to print the value of $x \cdot y^{-1} \bmod 998244353$, where y^{-1} is an integer such that $y \cdot y^{-1} \bmod 998244353 = 1$.

Examples

input
2

output
748683265

input
3

output
748683265

input
5
output
842268673

input
200000
output
202370013

Note

The real answer for the first example is $\frac{1}{4}$:

- with probability $\frac{1}{4}$, the towers are built in both towns 1 and 2, so we can set their signal powers to 1.

The real answer for the second example is $\frac{1}{4}$:

- with probability $\frac{1}{8}$, the towers are built in towns 1, 2 and 3, so we can set their signal powers to 1;
- with probability $\frac{1}{8}$, only one tower in town 2 is built, and we can set its signal power to 2.

The real answer for the third example is $\frac{5}{32}$. Note that even though the previous explanations used equal signal powers for all towers, it is not necessarily so. For example, if $n = 5$ and the towers are built in towns 2, 4 and 5, you may set the signal power of the tower in town 2 to 2, and the signal power of the towers in towns 4 and 5 to 1.

E. Two Editorials

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

Berland regional ICPC contest has just ended. There were m participants numbered from 1 to m, who competed on a problemset of n problems numbered from 1 to n.

Now the editorial is about to take place. There are two problem authors, each of them is going to tell the tutorial to **exactly** k **consecutive tasks** of the problemset. The authors choose the segment of k consecutive tasks for themselves independently of each other. The segments can coincide, intersect or not intersect at all.

The i-th participant is interested in listening to the tutorial of all consecutive tasks from l_i to r_i. Each participant always chooses to listen to only the problem author that tells the tutorials to the maximum number of tasks he is interested in. Let this maximum number be a_i. No participant can listen to both of the authors, even if their segments don't intersect.

The authors want to choose the segments of k consecutive tasks for themselves in such a way that the sum of a_i over all participants is maximized.

Input

The first line contains three integers n, m and k ($1 \le n, m \le 2000$, $1 \le k \le n$) — the number of problems, the number of participants and the length of the segment of tasks each of the problem authors plans to tell the tutorial to.

The i-th of the next m lines contains two integers l_i and r_i ($1 \le l_i \le r_i \le n$) — the segment of tasks the i-th participant is interested in listening to the tutorial to.

Output

Print a single integer — the maximum sum of a_i over all participants.

Examples

input
10 5 3 1 3 2 4 6 9 6 9 1 8
output
14

input
10 3 3

2 4 4 6 3 5

output
8

input
4 4 1 3 3 1 1 2 2 4 4

output
2

input
5 4 5 1 2 2 3 3 4 4 5

output
8

Note

In the first example the first author can tell the tutorial to problems from 1 to 3 and the second one — from 6 to 8. That way the sequence of a_i will be $[3, 2, 3, 3, 3]$. Notice that the last participant can't listen to both author, he only chooses the one that tells the maximum number of problems he's interested in.

In the second example the first one can tell problems 2 to 4, the second one — 4 to 6.

In the third example the first one can tell problems 1 to 1, the second one — 2 to 2. Or 4 to 4 and 3 to 3. Every pair of different problems will get the same sum of 2.

In the fourth example the first one can tell problems 1 to 5, the second one — 1 to 5 as well.

F. Divide Powers

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

You are given a multiset of powers of two. More precisely, for each i from 0 to n exclusive you have cnt_i elements equal to 2^i.

In one operation, you can choose any one element $2^l > 1$ and divide it into two elements 2^{l-1}.

You should perform q queries. Each query has one of two types:

- "$1\ pos\ val$" — assign $cnt_{pos} := val$;
- "$2\ x\ k$" — calculate the minimum number of operations you need to make at least k elements with value lower or equal to 2^x.

Note that all queries of the second type don't change the multiset; that is, you just calculate the minimum number of operations, you don't perform them.

Input

The first line contains two integers n and q ($1 \le n \le 30$; $1 \le q \le 2 \cdot 10^5$) — the size of array cnt and the number of queries.

The second line contains n integers $cnt_0, cnt_1, \ldots, cnt_{n-1}$ ($0 \le cnt_i \le 10^6$).

Next q lines contain queries: one per line. Each query has one of two types:

- "$1\ pos\ val$" ($0 \le pos < n$; $0 \le val \le 10^6$);
- "$2\ x\ k$" ($0 \le x < n$; $1 \le k \le 10^{15}$).

It's guaranteed that there is at least one query of the second type.

Output

For each query of the second type, print the minimum number of operations you need to make at least k elements with a value lower or equal to 2^x or -1 if there is no way to do it.

Example

input
6 11 0 1 0 0 1 0

```
2 1 5
2 4 18
1 1 0
2 2 5
2 0 17
1 0 3
2 1 2
1 1 4
1 4 0
1 5 1
2 2 8
```

output

```
4
16
4
-1
0
1
```

G. Game On Tree

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

Alice and Bob are playing a game. They have a tree consisting of n vertices. Initially, Bob has k chips, the i-th chip is located in the vertex a_i (all these vertices are unique). Before the game starts, Alice will place a chip into one of the vertices of the tree.

The game consists of turns. Each turn, the following events happen (sequentially, exactly in the following order):

1. Alice either moves her chip to an adjacent vertex or doesn't move it;
2. for each Bob's chip, he either moves it to an adjacent vertex or doesn't move it. Note that this choice is done independently for each chip.

The game ends when Alice's chip shares the same vertex with one (or multiple) of Bob's chips. Note that Bob's chips may share the same vertex, even though they are in different vertices at the beginning of the game.

Alice wants to maximize the number of turns, Bob wants to minimize it. If the game ends in the middle of some turn (Alice moves her chip to a vertex that contains one or multiple Bob's chips), this turn is counted.

For each vertex, calculate the number of turns the game will last if Alice places her chip in that vertex.

Input

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

Then $n - 1$ lines follow, each line contains two integers u_i, v_i ($1 \le u_i, v_i \le n$; $u_i \ne v_i$) that denote the endpoints of an edge. These edges form a tree.

The next line contains one integer k ($1 \le k \le n - 1$) — the number of Bob's chips.

The last line contains k integers a_1, a_2, ..., a_k ($1 \le a_i \le n$; $a_i \ne a_j$ if $i \ne j$) — the vertices where the Bob's chips are initially placed.

Output

Print n integers. The i-th of them should be equal to the number of turns the game will last if Alice initially places her chip in the vertex i. If one of Bob's chips is already placed in vertex i, then the answer for vertex i is 0.

Examples

input

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

output

```
2 1 2 0 0
```

input

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

2 8 3

output

3 0 0 3 1 2 3 0

input

10
2 5
4 3
7 3
7 2
5 8
3 6
8 10
7 9
7 1
4
10 6 9 1

output

0 2 2 2 2 0 2 2 0 0