Squarepoint Challenge (Codeforces Round 1055, Div. 1 + Div. 2)

A. Increase or Smash

1 second, 1024 megabytes

Geumjae has an array a consisting of n zeros. His goal is to transform it into a given target array using a minimum number of operations.

He can perform the following two types of operations any number of times, in any order:

1. **Increase**: Choose any positive integer x. Increase *all* elements of the array a by x. In other words, he chooses a positive integer x, and for each i ($1 \le i \le n$), he replaces a_i with $a_i + x$.
2. **Smash**: Set *some* elements (possibly none or all) of the array a to 0. In other words, for each i ($1 \le i \le n$), he either replaces a_i with 0 or leaves it as before.

Given the final target state of the array a, find the minimum total number of operations (both **Increase** and **Smash**) Geumjae needs to perform.

It can be shown that for any given final array, a sequence of operations always exists.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). The description of the test cases follows.

The first line contains a single integer n ($1 \le n \le 100$) — the number of elements in the array a.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 100$) — the elements of the target array a.

Output

For each test case, output a single integer — the minimum number of operations required.

input
3 3 1 1 3 1 100 9 9 9 3 2 4 4 8 5 3

output
3 1 11

Explanation of the first test case:

The target array is $[1, 1, 3]$. A possible sequence of 3 operations (which is the minimum) is:

1. Initially, the array is $[0, 0, 0]$. After an **Increase** operation with $x = 2$, the array becomes $[2, 2, 2]$.
2. Next, after a **Smash** operation on the first two elements, the array becomes $[0, 0, 2]$.
3. Finally, after an **Increase** operation with $x = 1$, the array becomes $[1, 1, 3]$.

We used 2 **Increase** operations and 1 **Smash** operation for a total of 3 operations.

Explanation of the second test case:

The target array is $[100]$. A single **Increase** operation with $x = 100$ gives the target array.

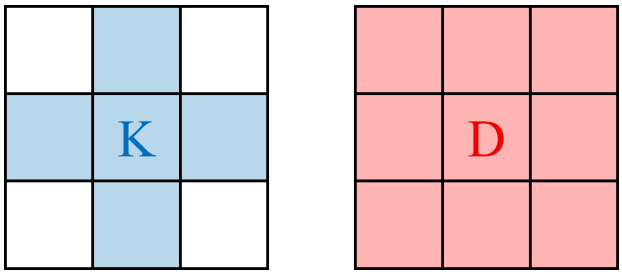
B. Catching the Krug

1 second, 1024 megabytes

Doran and the Krug are playing a game on a grid consisting of $(n + 1) \times (n + 1)$ cells whose coordinates are pairs of integers from 0 to n, inclusive. The Krug's goal is not to be *caught* by Doran for as long as possible, while Doran's goal is to *catch* the Krug as early as possible. We say Doran *caught* the Krug if they stand on the same grid cell.

To play the game, the Krug and Doran take turns alternately, starting from the Krug:

- The Krug can either stay in the same cell or move to a cell vertically or horizontally (but **not** diagonally) adjacent. Formally, if the Krug is currently at the cell (a, b), she can stay at (a, b) or move to either $(a - 1, b), (a, b - 1), (a, b + 1), (a + 1, b)$.
- Doran can either stay in the same cell or move to a cell vertically, horizontally, or diagonally adjacent. Formally, if Doran is currently at the cell (c, d), he can stay at (c, d) or move to either $(c - 1, d - 1), (c - 1, d), (c - 1, d + 1), (c, d - 1), (c, d + 1), (c + 1, d - 1), (c$.
- Both players cannot go outside of the grid.



Figures showing possible movements of the Krug and Doran. Letters 'K' and 'D' represent the current position of the Krug and Doran, and the colored cells represent possible positions each player can move to in their respective turn.

The Krug's *survival time* is defined as the number of Doran's turns until Doran *catches* the Krug for the given starting cells of the players. Assuming that both players play optimally, find the Krug's *survival time* or report that the Krug can survive for infinite turns.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

Each test case consists of a single line containing five integers n, r_K, c_K, r_D, and c_D ($1 \le n \le 10^9$, $0 \le r_K, c_K, r_D, c_D \le n$, $(r_K, c_K) \ne (r_D, c_D)$) — n is the size of the grid, (r_K, c_K) represents the Krug's starting cell, and (r_D, c_D) represents Doran's starting cell.

Output

For each test case, output the Krug's *survival time* when both players play optimally. If the Krug can survive for infinite turns, print -1.

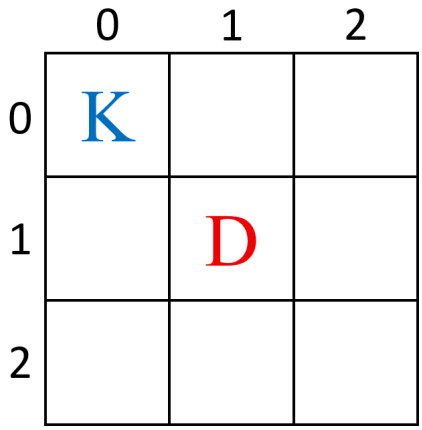
input
7 2 0 0 1 1 3 1 1 0 1 1 1 0 0 1 6 1 3 3 2 9 4 1 4 2 82 64 2 63 2 1000000000 500000000 500000000 1000000000 0

output
1 3 1 4 2 19 1000000000

Explanation of the first test case:

The Krug starts at $(0, 0)$ and Doran starts at $(1, 1)$. On the Krug's turn, she can only move to $(0, 0)$, $(0, 1)$, or $(1, 0)$.

Doran, starting at $(1, 1)$, can reach any cell on the 3×3 grid in a single move. Therefore, no matter where the Krug moves, Doran can always move to her cell on his first turn. The Krug's survival time is 1.



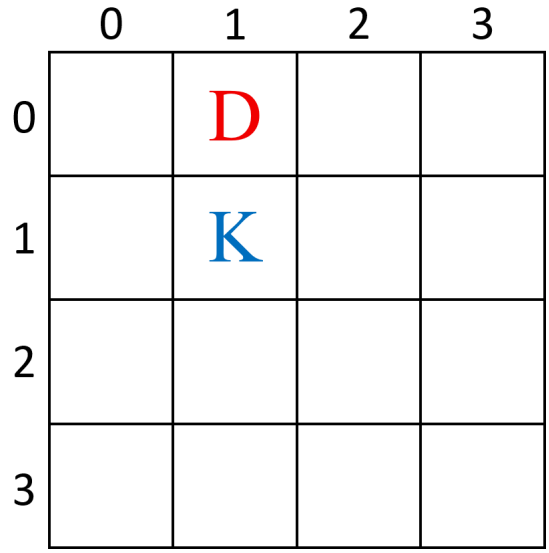
Explanation of the second test case:

The Krug starts at $(1, 1)$ and Doran at $(0, 1)$. To survive Doran's first turn, the Krug must move to a cell that Doran cannot reach. The only such cell is $(2, 1)$.

After the Krug moves to $(2, 1)$, Doran moves to $(1, 1)$ to get closer.

Now, for her second move, the Krug must again move to a cell Doran can't reach, which is $(3, 1)$. Doran then moves to $(2, 1)$.

At this point, no matter where the Krug moves for her third turn, Doran can always reach her cell. It can be shown that this is an optimal strategy for both, and therefore the Krug's survival time is 3.



C. Triple Removal

2 seconds, 1024 megabytes

Tired of supporting ranged carries, Keria is now creating a data structure problem about supporting range queries.

For an array $b = [b_1, b_2, \ldots, b_m]$ of length m where $b_i = 0$ or $b_i = 1$, consider the following **triple removal** operation:

1. Choose three indices $1 \le i < j < k \le m$ such that the elements at these positions are identical ($b_i = b_j = b_k$).
2. Remove these three elements from the array. The cost of this operation is defined as $\min(k - j, j - i)$. After the removal, the remaining parts of the array are concatenated, and their indices are relabeled.

We want to make the array b empty using the **triple removal** operation. Hence, we define the *total cost* of an array as the minimum possible sum of the costs of **triple removal** operations required to make the array empty. If it is impossible to make the array empty, the cost is defined to be -1.

Keria wants to test his data structure. For this, you must answer q independent queries. Initially, you are given an array $a = [a_1, a_2, \ldots, a_n]$ of length n where $a_i = 0$ or $a_i = 1$. For each query, you are given a range $1 \le l \le r \le n$ and must find the cost for the array $[a_l, a_{l+1}, \ldots, a_r]$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two integers n and q ($1 \le n, q \le 250\,000$) — the length of the array and the number of queries.

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($a_i = 0$ or $a_i = 1$) — the elements of the array.

Then q lines follow. The i-th of them contains two integers l_i and r_i ($1 \le l_i \le r_i \le n$) — the range of the subarray for the i-th query.

It is guaranteed that the sum of n over all test cases does not exceed $250\,000$.

It is guaranteed that the sum of q over all test cases does not exceed $250\,000$.

Output

For each test case, output q lines. The i-th line should contain a single integer representing the answer to the i-th query.

input
2 12 4 0 0 1 1 0 1 0 1 0 1 1 0 1 12 2 7 5 10 6 11 6 3 0 0 0 1 1 1 1 3 4 6 1 6

output
4 2 3 -1 1 1 2

Explanation of the first test case, first query (1 12):

The subarray is $[0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0]$. There are six 0s and six 1s. A possible optimal sequence of operations is:

1. Remove the three 1s at indices $3, 4, 6$. The cost is $\min(6 - 4, 4 - 3) = \min(2, 1) = 1$. The array becomes $[0, 0, 0, 0, 1, 0, 1, 1, 0]$.
2. Remove the three 0s at indices $1, 2, 3$. The cost is $\min(3 - 2, 2 - 1) = \min(1, 1) = 1$. The array becomes $[0, 1, 0, 1, 1, 0]$.
3. Remove the three 1s at indices $2, 4, 5$. The cost is $\min(5 - 4, 4 - 2) = \min(1, 2) = 1$. The array becomes $[0, 0, 0]$.
4. Remove the three 0s at indices $1, 2, 3$. The cost is $\min(3 - 2, 2 - 1) = \min(1, 1) = 1$. The array is now empty.

The total cost is $1 + 1 + 1 + 1 = 4$.

D. Division Versus Addition

2 seconds, 1024 megabytes

For an array $b = [b_1, b_2, \ldots, b_m]$ of length m ($b_i \ge 2$), consider the following two-player game played by Poby and Rekkles.

- The players take turns, with Poby moving first.
- On Poby's turn, he must choose an element $x \ge 2$ and replace it with $\left\lfloor \frac{x}{2} \right\rfloor$. In other words, he picks i ($1 \le i \le m$) such that $b_i \ge 2$, then does $b_i := \left\lfloor \frac{b_i}{2} \right\rfloor$.
- On Rekkles' turn, he must choose an element $x \ge 2$ from the array b and replace it with $x + 1$. In other words, he picks i ($1 \le i \le m$) such that $b_i \ge 2$, then does $b_i := b_i + 1$.

The game ends once all elements in the array b are equal to 1.

Define the **score** of the game as the number of moves that Poby makes. Poby's goal is to minimize the **score**, while Rekkles's goal is to maximize the **score**.

Then, the **value** of the array b is the score of the game when both players play optimally.

You are given an integer array a of length n ($a_i \geq 2$).

Answer q independent queries. In each query, you are given a range $1 \leq l \leq r \leq n$ and must find the **value** of the array $[a_l, a_{l+1}, \ldots, a_r]$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). The description of the test cases follows.

The first line of each test case contains two integers n and q ($1 \leq n, q \leq 250\,000$) — the length of the array and the number of queries.

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($2 \leq a_i \leq 10^9$) — the elements of the array a.

Then q lines follow. The j-th of them contains two integers l_j and r_j ($1 \leq l_j \leq r_j \leq n$) — the range of the subarray for the i-th query.

It is guaranteed that the sum of n over all test cases does not exceed $250\,000$.

It is guaranteed that the sum of q over all test cases does not exceed $250\,000$.

Output

For each test case, output q lines. The i-th line should contain a single integer representing the answer to the i-th query.

input
2 5 5 4 3 2 5 6 1 1 1 2 2 4 3 5 1 5 10 1 314 159 265 358 979 323 846 264 338 327 1 10

output
2 3 5 6 10 91

Explanation of the first test case, first query (1 1):

The subarray is $[4]$.

1. Poby: $4 \to \left\lfloor \frac{4}{2} \right\rfloor = 2$. The array is $[2]$.
2. Rekkles: $2 \to 3$. The array is $[3]$.
3. Poby: $3 \to \left\lfloor \frac{3}{2} \right\rfloor = 1$. The array is $[1]$, so the game ends.

It can be shown that this strategy is optimal for both players. Therefore, the value of the array $[4]$ is 2.

Explanation of the first test case, second query (1 2):

The subarray is $[4, 3]$.

1. Poby: $3 \to \left\lfloor \frac{3}{2} \right\rfloor = 1$. The array is $[4, 1]$.
2. Rekkles: $4 \to 5$. The array is $[5, 1]$.
3. Poby: $5 \to \left\lfloor \frac{5}{2} \right\rfloor = 2$. The array is $[2, 1]$.
4. Rekkles: $2 \to 3$. The array is $[3, 1]$.
5. Poby: $3 \to \left\lfloor \frac{3}{2} \right\rfloor = 1$. The array is $[1, 1]$, so the game ends.

It can be shown that this strategy is optimal for both players. Therefore, the value of the array $[4, 3]$ is 3.

E. Monotone Subsequence

2 seconds, 1024 megabytes

This is an interactive problem.

Faker is being naughty again. You asked him to create a nice query problem, but he created an interactive problem where he is answering a query instead! Faker hid a permutation from you, and you have to infer some interesting information by interacting with him.

You are given an integer n. Faker hid a hidden permutation* $p_1, p_2, \ldots, p_{n^2+1}$ of length $n^2 + 1$. Your goal is to find a monotone subsequence (either increasing or decreasing) of the hidden permutation, with length exactly $n + 1$. It can be proved that every permutation of length $n^2 + 1$ contains a monotone subsequence of length $n + 1$. For more information about the proof, you can check out this Wikipedia page.

To find it, you can make at most n **skyscraper queries** to the interactor, which is defined as follows:

- You provide a set of k indices as a strictly increasing sequence: $i_1, i_2, \ldots, i_k$.
- The interactor considers the values of the hidden permutation at these indices: $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$.
- The interactor then returns the indices corresponding to the **visible skyscrapers** from this set. An index i_j is visible if its value p_{i_j} is greater than the values of all preceding elements in your query, i.e., $p_{i_j} > p_{i_m}$ for all $1 \leq m < j$. This is equivalent to finding the indices of the left-to-right maxima of the sequence $(p_{i_1}, \ldots, p_{i_k})$.

After making at most n queries, you must report a valid monotone subsequence of length **exactly** $n + 1$.

Note that the permutation p is fixed **before** any queries are made and does not depend on the queries.

* A permutation of length m is an array consisting of m distinct integers from 1 to m in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($m = 3$ but there is 4 in the array).

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 5000$). The description of the test cases follows.

The first and only line of each test case contains a single integer n ($1 \leq n \leq 100$).

It is guaranteed that the sum of $n^2 + 1$ over all test cases does not exceed $10\,001$.

Interaction

The interaction for each test case begins by reading the integer n.

To make a query, print a line in the following format:

- `?` k i_1 i_2 $\ldots$ i_k

where k is the number of indices in your query ($1 \leq k \leq n^2 + 1$), and $i_1, \ldots, i_k$ are the indices themselves, satisfying $1 \leq i_1 < i_2 < \ldots < i_k \leq n^2 + 1$. **The indices should be presented in sorted order.**

In response, the interactor will print a line in the following format:

- c j_1 j_2 $\ldots$ j_c

where c is the number of visible skyscrapers from your query ($1 \leq c \leq k$), and $j_1, \ldots, j_c$ are their indices, satisfying $1 \leq j_1 < j_2 < \ldots < j_c \leq n^2 + 1$. The indices will be presented in sorted order.

To report your final answer, print a line in the following format:

- `!` s_1 s_2 $\ldots$ s_{n+1}

where $s_1, \ldots, s_{n+1}$ are the indices of the elements that form your found monotone subsequence of length $n + 1$, satisfying $1 \leq s_1 < s_2 < \ldots < s_{n+1} \leq n^2 + 1$. **The indices should be presented in sorted order.**

Note that answering **does not count** toward your limit of commands.

After printing the answer, your program should proceed to the next test case or terminate if there are no more.

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

If, at any interaction step, you read -1 instead of valid data, your solution must exit immediately. This means that your solution will receive `Wrong answer` because of an invalid query or any other mistake. Failing to exit can result in an arbitrary verdict because your solution will continue to read from a closed stream.

Hacks

To hack, use the following format.

The first line should contain a single integer t ($1 \le t \le 5000$).

The first line of each test case should contain a single integer n ($1 \le n \le 100$).

The second line of each test case should contain $n^2 + 1$ space separated integers $p_1, p_2, \ldots, p_{n^2+1}$ ($1 \le p_i \le n^2 + 1$). p should be a permutation of length $n^2 + 1$.

The sum of $n^2 + 1$ should not exceed $10\,001$.

For example, the following is the hack format of the example test:

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

*To flush, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `sys.stdout.flush()` in Python;
- see the documentation for other languages.

input

```
2
1

2 1 2

2

1 1

2 2 3
```

output

```

? 2 1 2

! 1 2

? 3 1 2 3

? 3 2 3 5

! 1 3 4
```

For the first test case, $n = 1$. The hidden permutation is $p = [1, 2]$.

- For the query `? 2 1 2`, the visible skyscrapers are at indices 1 and 2. The interactor returns `2 1 2`.
- An increasing subsequence of length 2 at indices $1, 2$ is reported.

For the second test case, $n = 2$. The hidden permutation is $p = [5, 3, 4, 1, 2]$.

- For the query `? 3 1 2 3`, the visible skyscraper is at index 1. The interactor returns `1 1`.
- For the query `? 3 2 3 5`, the visible skyscrapers are at indices 2 and 3. The interactor returns `2 2 3`.
- A decreasing subsequence of length 3 at indices $1, 3, 4$ is reported.

Although Faker will play the role of interactor, the interactor will never lie to you.

F. Triple Attack

3 seconds, 1024 megabytes

Zeus is analyzing a replay of the fight to understand his opponent's attack patterns. The opponent has a special ability: if they land three attacks on a target within a time frame of z, their third attack becomes a powerful, boosted attack.

To outplay his opponent, Zeus should not let his opponent trigger their boosted attack. Let $Y = \{y_1, y_2, \ldots, y_m\}$ be the multiset of m timestamps, where each y_i represents the time when the opponent's attack landed. We call Y to be **safe** if for every three timestamps $\{y_i, y_j, y_k\}$ such that $1 \le i < j < k \le m$, it holds that $\max(y_i, y_j, y_k) - \min(y_i, y_j, y_k) > z$, where z is the duration of the time frame that is given to you as an input.

Zeus has a log of n timestamps, $x_1, x_2, \ldots, x_n$, representing when the opponent's attacks landed. The timestamps are **sorted in nondecreasing order** of occurrence. In other words, $x_i \le x_{i+1}$ for all $1 \le i < n$.

Zeus has q intervals of interest, denoted as two integers $1 \le l \le r \le n$. For each interval, Zeus wants to find the maximum number of attacks among $[x_l, x_{l+1}, \ldots, x_r]$ that he could have let through: In other words, Zeus wants to find a maximum size subset of the multiset $\{x_l, x_{l+1}, \ldots, x_r\}$ such that the subset is **safe**.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 20\,000$). The description of the test cases follows.

The first line contains two integers n and z ($1 \le n \le 250\,000$, $1 \le z \le 10^9$).

The second line contains n integers $x_1, x_2, \ldots, x_n$ ($1 \le x_i \le 10^9$) — the timestamps of the opponent's attacks. It is guaranteed that the array x is sorted, i.e., $x_i \le x_{i+1}$ for all $1 \le i < n$.

The third line contains a single integer q ($1 \le q \le 250\,000$).

The next q lines each contain two integers l and r ($1 \le l \le r \le n$) — the endpoints of the interval.

It is guaranteed that the sum of n over all test cases does not exceed $250\,000$.

It is guaranteed that the sum of q over all test cases does not exceed $250\,000$.

Output

For each of the q queries, print a single integer — the maximum size of a safe subset of attack timestamps in the given interval.

input

```
3
6 10
1 5 7 8 11 12
6
1 6
1 5
2 6
1 4
2 5
3 6
6 1
1 1 1 3 3 3
2
3 3
1 6
12 15
4 5 15 24 27 32 36 39 40 46 48 48
20
1 12
1 11
6 10
1 8
8 12
11 12
2 9
3 8
7 8
7 10
4 8
9 12
9 10
2 12
1 5
3 12
4 8
3 7
7 12
10 11
```

output

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

In the first query of the first test case, we consider the timestamps $\{1,5,7,8,11,12\}$ with $z=10$. The subset $\{1,5,12\}$ is safe because its only triplet satisfies $12-1=11>10$. It's impossible to form a safe subset of size 4, hence the answer to this query is 3.

In the first query of the second test case, we consider the timestamps $\{1\}$ with $z=1$. The entire set $\{1\}$ is safe because there are no triplets. Hence the answer to this query is 1.

In the second query of the second test case, we consider the timestamps $\{1,1,1,3,3,3\}$ with $z=1$.

The subset $S=\{1,1,3,3\}$ is safe because:

- For the triple $(i,j,k)=(1,2,3)$, $\max(1,1,3)-\min(1,1,3)=2>1$.
- For the triple $(i,j,k)=(1,2,4)$, $\max(1,1,3)-\min(1,1,3)=2>1$.
- For the triple $(i,j,k)=(1,3,4)$, $\max(1,3,3)-\min(1,3,3)=2>1$.
- For the triple $(i,j,k)=(2,3,4)$, $\max(1,3,3)-\min(1,3,3)=2>1$.

It's impossible to form a safe subset of size 5, hence the answer to this query is 4.

G. Query Jungle

3 seconds, 1024 megabytes

Oner is a jungler — a role where you hunt monsters in a jungle. Given the number of trees he sees in a jungle, it's no surprise that he is addicted to tree query problems.

You are given a tree of n vertices, rooted at vertex 1. Each vertex either contains a monster or does not.

You want to find the minimum integer k such that there exist k paths that satisfy the following conditions:

- Each path must start at the root (vertex 1).
- Every vertex with a monster must be included in at least one of these paths. A vertex is considered included in a path if it is one of the path's vertices, including its endpoints.

To make this problem more challenging, you must also answer q queries. For each query, you are given a vertex v. For each vertex w in the **subtree** of v, its status is inverted — the one containing a monster starts to not contain one, and the one not containing a monster starts to contain one. After each query, you must solve the original problem again with the updated status.

Note that queries are **cumulative**, so the effects of each query carry on to future queries.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 20\,000$). The description of the test cases follows.

The first line contains a single integer n ($2 \le n \le 250\,000$) — the number of vertices in the tree.

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($a_i \in \{0,1\}$), representing the initial status. If $a_i = 1$, vertex i contains a monster; if $a_i = 0$, it does not.

The next $n-1$ lines each contain two integers u and v ($1 \le u, v \le n, u \ne v$), describing an edge between vertices u and v. It is guaranteed that these edges form a tree.

The next line contains a single integer q ($0 \le q \le 250\,000$) — the number of queries.

The next q lines each contain a single integer v_i ($1 \le v_i \le n$) — the vertex given for the i-th query.

It is guaranteed that the sum of n over all test cases does not exceed $250\,000$.

It is guaranteed that the sum of q over all test cases does not exceed $250\,000$.

Output

Print $q+1$ lines. The first line should contain the minimum number of paths k for the initial status. Each subsequent line should contain the answer after each query.

input

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

output

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

Test Case 1:

Initial State: The monsters are in vertex $\{2,4,5\}$. We need two paths: $1 \to 7 \to 3 \to 2$ and $1 \to 7 \to 5 \to 4$. The answer is 2.

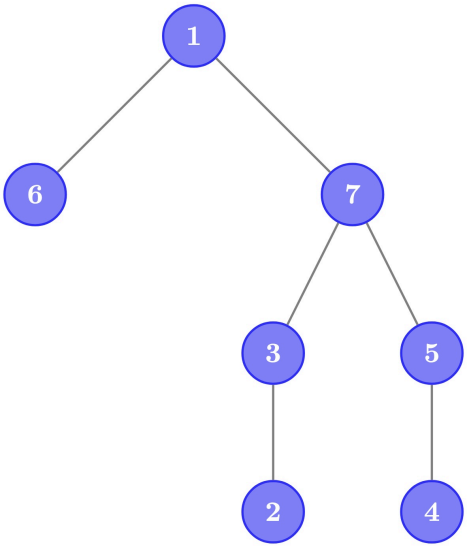
After Query 1 ($v=2$): The monsters are in vertex $\{4,5\}$. We only need one path, $1 \to 7 \to 5 \to 4$. The answer is 1.

After Query 2 ($v=4$): The monsters are in vertex $\{5\}$. We only need one path, $1 \to 7 \to 5$. The answer is 1.

After Query 3 ($v=6$): The monsters are in vertex $\{5,6\}$. We need two paths, $1 \to 7 \to 5$ and $1 \to 6$. The answer is 2.

After Query 4 ($v=7$): The monsters are in vertex $\{2,3,4,6,7\}$. We need three paths, $1 \to 6$, $1 \to 7 \to 5 \to 4$, and $1 \to 7 \to 3 \to 2$. The answer is 3.

The following figure denotes the tree in the example input.



Test Case 2:

Initial State: The monsters are in vertex $\{2\}$. We need one path: $1 \to 2$. The answer is 1.

After Query 1 ($v = 2$): There are no monsters. We need zero paths. The answer is 0.

After Query 2 ($v = 1$): The monsters are in vertex $\{1, 2\}$. We need one path: $1 \to 2$. The answer is 1.

H1. Victorious Coloring (Easy Version)

3 seconds, 1024 megabytes

This is the easy version of the problem. The difference between the versions is that in this version, $q \le 10$. You can hack only if you solved all versions of this problem.

You are given a tree with n vertices, where each vertex is numbered from 1 to n. Each edge is assigned a positive integer weight $w_1, w_2, \ldots, w_{n-1}$ as well.

A **victorious coloring** is a coloring of each vertex into two colors: *red* and *yellow*, where there should be at least one vertex colored in red (corresponding to the symbol of team T1).

Suppose that there is a **nonnegative** integer weight $x_1, x_2, \ldots, x_n$ assigned to each vertex. The cost of the victorious coloring is defined as the sum of the weights of all red vertices, plus the sum of the weights of all edges that connect vertices of different colors (between red and yellow). We define $f([x_1, x_2, \ldots, x_n])$ as the minimum possible cost for all victorious colorings.

Gumayusi considered the problem of computing $f([x_1, x_2, \ldots, x_n])$, given the sequence $x_1, x_2, \ldots, x_n$. However, this problem was too easy for him, so he devised a variation: Given an integer l, find a sequence of nonnegative integer vertex weights $[x_1, x_2, \ldots, x_n]$ such that $f([x_1, x_2, \ldots, x_n]) \ge l$ and the total sum $\sum_{i=1}^{n} x_i$ is minimized.

Gumayusi was satisfied, but there was a serious issue — this problem doesn't have any queries, which is a necessary component for any problem that isn't bad. So, he added queries to this problem. For each l given as a query, you must find the corresponding minimum possible sum of vertex weights.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line contains an integer n ($2 \le n \le 250\,000$) — the number of vertices.

The following $n - 1$ lines contain three integers u_i, v_i, w_i ($1 \le u_i, v_i \le n, 1 \le w_i \le 10^9, u_i \ne v_i$) — indicating an edge connecting the vertices u_i and v_i with weight w_i.

It is guaranteed that the edges form a tree.

The next line contains an integer q ($1 \le q \le 10$) — the number of queries.

The following q lines contain a single integer l_i ($1 \le l_i \le 10^9$) — the parameters of the i-th query.

It is guaranteed that the sum of n over all test cases does not exceed $250\,000$.

Note that there is no explicit upper bound on the sum of q.

Output

For each of the q queries, output the answer separated by lines.

input

```
2
5
3 5 10
2 3 4
3 1 10
3 4 2
5
28
32
11
17
23
2
1 2 3
1
1
```

output

```
88
108
21
42
66
1
```

The following list shows the possible optimal assignments for each query in the first test case:

- $[18, 24, 2, 26, 18]$
- $[22, 28, 6, 30, 22]$
- $[4, 7, 0, 9, 1]$
- $[7, 13, 0, 15, 7]$
- $[13, 19, 0, 21, 13]$

H2. Victorious Coloring (Hard Version)

3 seconds, 1024 megabytes

This is the hard version of the problem. The difference between the versions is that in this version, $q \le 250\,000$. You can hack only if you solved all versions of this problem.

You are given a tree with n vertices, where each vertex is numbered from 1 to n. Each edge is assigned a positive integer weight $w_1, w_2, \ldots, w_{n-1}$ as well.

A **victorious coloring** is a coloring of each vertex into two colors: *red* and *yellow*, where there should be at least one vertex colored in red (corresponding to the symbol of team T1).

Suppose that there is a **nonnegative** integer weight $x_1, x_2, \ldots, x_n$ assigned to each vertex. The cost of the victorious coloring is defined as the sum of the weights of all red vertices, plus the sum of the weights of all edges that connect vertices of different colors (between red and yellow). We define $f([x_1, x_2, \ldots, x_n])$ as the minimum possible cost for all victorious colorings.

Gumayusi considered the problem of computing $f([x_1, x_2, \ldots, x_n])$, given the sequence $x_1, x_2, \ldots, x_n$. However, this problem was too easy for him, so he devised a variation: Given an integer t, find a sequence of nonnegative integer vertex weights $[x_1, x_2, \ldots, x_n]$ such that $f([x_1, x_2, \ldots, x_n]) \ge t$ and the total sum $\sum_{i=1}^{n} x_i$ is minimized.

Gumayusi was satisfied, but there was a serious issue — this problem doesn't have any queries, which is a necessary component for any problem that isn't bad. So, he added queries to this problem. For each t given as a query, you must find the corresponding minimum possible sum of vertex weights.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line contains an integer n ($2 \le n \le 250\,000$) — the number of vertices.

The following $n - 1$ lines contain three integers u_i, v_i, w_i ($1 \le u_i, v_i \le n, 1 \le w_i \le 10^9, u_i \ne v_i$) — indicating an edge connecting the vertices u_i and v_i with weight w_i.

It is guaranteed that the edges form a tree.

The next line contains an integer q ($1 \le q \le 250\,000$) — the number of queries.

The following q lines contain a single integer t_i ($1 \le t_i \le 10^9$) — the parameters of the i-th query. It is guaranteed that the sum of n over all test cases does not exceed $250\,000$.

It is guaranteed that the sum of q over all test cases does not exceed $250\,000$.

Output

For each of the q queries, output the answer separated by lines.

input

```
2
5
3 5 10
2 3 4
3 1 10
3 4 2
5
28
32
11
17
23
2
1 2 3
1
1
```

output

```
88
108
21
42
66
1
```

The following list shows the possible optimal assignments for each query for the first test case:

- $[18, 24, 2, 26, 18]$
- $[22, 28, 6, 30, 22]$
- $[4, 7, 0, 9, 1]$
- $[7, 13, 0, 15, 7]$
- $[13, 19, 0, 21, 13]$