Problem A
Compare Suffixes
Time limit: 2 seconds

The *judge program* possesses a hidden string S of length n consisting only of lowercase Latin letters (`a–z`). You cannot access the string. At the start, only the length n is given to you.

Your task is to determine the order of all n suffixes using a limited number of *queries*.

For an integer k ($1 \le k \le n$), let $S(k)$ denote the suffix of S starting at the k-th character. In particular, $S(1) = S$.

In a single query, you specify two distinct integers i and j. The judge program compares $S(i)$ and $S(j)$ lexicographically and returns whether $S(i) < S(j)$ or $S(i) > S(j)$ to you. Note that ties never occur for $i \neq j$ because all suffixes are distinct.

Find a permutation $(p_1, p_2, \ldots, p_n)$ of $(1, 2, \ldots, n)$ such that $S(p_1) < S(p_2) < \cdots < S(p_n)$.

Interaction

The first line of input contains the integer n ($2 \le n \le 1000$).

To issue a query, your program should write a line of the form "`query` i j" ($1 \le i \le n; 1 \le j \le n; i \neq j$). After that, an input line containing `first` or `second` becomes available. A line containing `first` means $S(i) < S(j)$, while a line containing `second` means $S(i) > S(j)$.

Once you determine the order of the suffixes, your program should write a line of the form "`answer` p_1 p_2 $\ldots$ p_n". After that, the interaction stops and your program should terminate without extra output.

Your program may issue at most 6260 queries. If your program issues more than 6260 queries, it will be judged as "Wrong Answer."

It is guaranteed that the hidden string S consists only of lowercase letters (`a–z`).

Notes on interactive judging:

- The evaluation is non-adversarial, meaning that the string S is chosen in advance rather than in response to your queries.
- Do not forget to flush output buffers after writing. See the "Judging Details" document for details.
- You are provided with a command-line tool for local testing, together with input files corresponding to the sample interactions. You can download these files from DOMjudge. The tool has comments at the top to explain its use.

Read	Sample Interaction #1	Write
4		
	query 2 1	
first		
	query 2 4	
second		
	query 1 3	
first		
	answer 4 2 1 3	

Explanation for the sample interaction #1

In this sample, $S =$ `icpc` is assumed. The order of four suffixes is $S(4) < S(2) < S(1) < S(3)$ because `c` $<$ `cpc` $<$ `icpc` $<$ `pc`.

In the first and third query, `first` is returned because $S(2) < S(1)$ and $S(1) < S(3)$. In the second query, `second` is returned because $S(2) > S(4)$. With these responses, you can determine the ordering.

Problem B
Subtree Removal Game
Time limit: 2 seconds

You are given a *rooted tree* with n nodes, numbered from 1 to n, rooted at node 1. For each node i ($2 \leq i \leq n$), its parent is node p_i. For each node i, if it is a *leaf* (a node having no children), the integer i is written on it. Otherwise, nothing is written.

Node u is a *descendant* of node v if $u = v$, or u is not the root and the parent of u is a descendant of v.

Now, you and your friend play a game on this tree, taking turns alternately: You move first, then your friend, and so on. On each turn, the current player must choose a node v and then remove the entire subtree of v (i.e., all descendants of v, including v itself). The move is allowed only if, *after* the removal, at least one node with an integer written on it remains in the tree.

The game ends when no further moves are possible. At that point, exactly one node with an integer written on it remains; the *score* of the game is the integer on that node.

You aim to minimize the score, while your friend aims to maximize it. Assuming you and your friend play optimally, determine the score of the game.

Input

The first line of input contains an integer n ($2 \leq n \leq 500\,000$).

The second line contains $n - 1$ integers $p_2, p_3, \ldots, p_n$ ($1 \leq p_i < i$ for all $2 \leq i \leq n$).

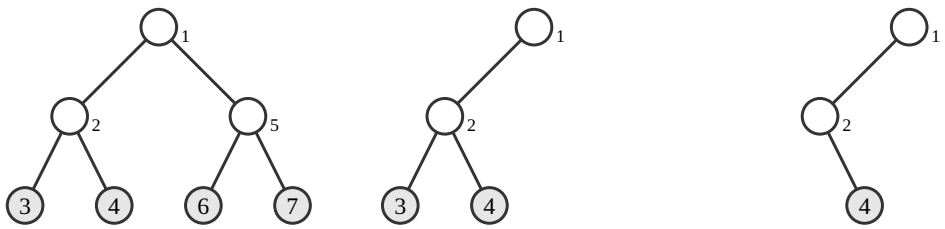
Output

Output the score of the game assuming you and your friend play optimally.

Sample Input #1	Sample Output #1
7 1 2 2 1 5 5	4

Explanation for the sample input/output #1

The given tree is illustrated in Figure B.1 (a).



(a) (b) (c)

Figure B.1: Illustration of Sample Input #1.

The only optimal moves for both players are as follows:

- You choose node 5. Then nodes 5, 6, and 7 are removed (Figure B.1 (b)).
- Your friend chooses node 3. Then only node 3 is removed (Figure B.1 (c)).
- You cannot make any moves, and the game ends.

After these moves, only one node with an integer written on it, node 4, remains. The score of this game is 4.

Sample Input #2	Sample Output #2
9 1 1 3 2 5 4 5 5	7

Explanation for the sample input/output #2

The given tree is illustrated in Figure B.2. One of the optimal moves for you is to choose node 2. The game ends after that, and the score is 7.

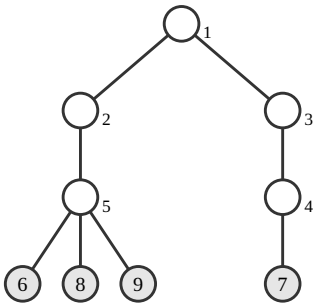


Figure B.2: Illustration of Sample Input #2.

Problem C
Upside Down Dijkstra
Time limit: 2 seconds

Your little sibling has a connected undirected graph of n vertices and m edges. The vertices are numbered from 1 to n, and the edges are numbered from 1 to m. Edge j connects vertices u_j and v_j with a **positive integer** weight of w_j.

Your little sibling coded Dijkstra's algorithm to find the shortest distances from vertex 1 to all other vertices, as shown in the pseudocode on the right. The array S records the order in which each vertex is popped from the heap for the first time. Note that even though tuples for each vertex may be pushed multiple times into the heap, each vertex is added to S exactly once.

```
visited ← bool array of size n, all FALSE
heap ← heap of tuples containing (0, 1)
S ← an empty array
while heap is not empty :
    (dist, u) ← top of heap
    pop top of heap
    if visited[u] = TRUE :
        continue
    visited[u] ← TRUE
    append u to the end of S
    for each v adjacent to u :
        w ← edge weight from u to v
        push tuple (dist + w, v) into heap
```

However, your little sibling made a big mistake. In your sibling's code, the heap always pops the *maximum* tuple instead of the minimum tuple. Here, the heap orders tuples lexicographically by $(dist, u)$, with larger $dist$ considered larger; ties are broken by larger u.

Your little sibling shares the graph structure with you, that is, all pairs of u_j and v_j ($1 \le j \le m$). However, your sibling does not share the edge weights w_j. Instead, your sibling shares an array $S = (s_1, s_2, \ldots, s_n)$ with you and asks you to reconstruct the edge weights. Your task is to find integer edge weights $w_1, w_2, \ldots, w_m$ ($1 \le w_j \le 10^9$ for all j) such that running your sibling's incorrect code yields the given array S.

Find any such assignment of edge weights. If no such assignment exists, report that it is impossible.

Input

The first line of input contains two integers n and m ($2 \le n \le 100\,000$; $n - 1 \le m \le 200\,000$).

The j-th of the next m lines contains two integers u_j and v_j ($1 \le u_j < v_j \le n$; $(u_j, v_j) \ne (u_k, v_k)$ for all $j \ne k$). The input guarantees that the graph is connected.

The next line contains n integers $s_1, s_2, \ldots, s_n$ ($1 \le s_i \le n$; $s_i \ne s_\ell$ for all $i \ne \ell$).

Output

If no assignment of edge weights yields the given array S, output `impossible`.

Otherwise, output a single line containing m integers $w_1, w_2, \ldots, w_m$ ($1 \le w_j \le 10^9$) for the weights of the edges such that your sibling's incorrect code yields the given array S.

If there are multiple outputs, any one of them will be accepted.

It can be shown that if there exists any assignment of positive integer weights that yields S, then there also exists one with $1 \le w_j \le 10^9$ for all j.

Sample Input #1	Sample Output #1
5 7 3 4 2 3 1 2 3 5 1 4 1 5 4 5 1 4 3 5 2	6 1 3 1 3 2 2

Explanation for the sample input/output #1

Figure C.1 illustrates the structure of the graph and the assignment of edge weights in the sample output.

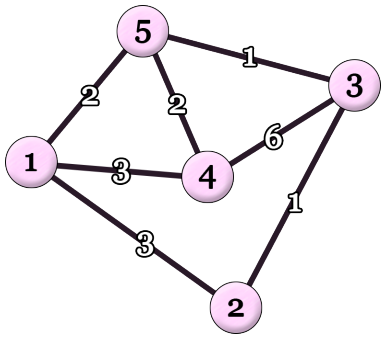


Figure C.1: Graph with assigned edge weights.

With this assignment, your sibling's incorrect code runs as follows:

1. Initially, the heap contains $\{(0, 1)\}$.
2. From $(0, 1)$, vertex 1 is popped from the heap for the first time. Now the heap contains $\{(3, 4), (3, 2), (2, 5)\}$.
3. From $(3, 4)$, vertex 4 is popped. Now the heap contains $\{(9, 3), (6, 1), (5, 5), (3, 2), (2, 5)\}$.
4. From $(9, 3)$, vertex 3 is popped. Now the heap contains $\{(15, 4), (10, 5), (10, 2), (6, 1), (5, 5), (3, 2), (2, 5)\}$.
5. From $(10, 5)$, vertex 5 is popped. Now the heap contains $\{(12, 4), (12, 1), (11, 3), (10, 2), (6, 1), (5, 5), (3, 2), (2, 5)\}$.
6. From $(10, 2)$, vertex 2 is popped.

This results in $S = (1, 4, 3, 5, 2)$.

Sample Input #2	Sample Output #2
4 4 1 2 2 3 1 3 2 4 1 3 4 2	impossible

Problem D
Christmas Tree Un-decoration
Time limit: 3 seconds

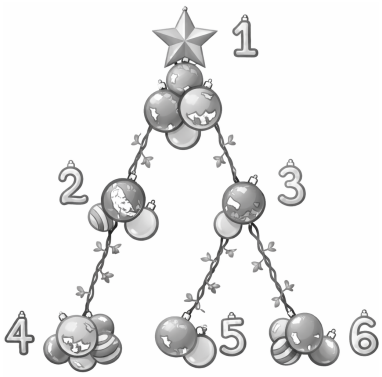


Figure D.1: Christmas tree.

Last Christmas, you had a lovely Christmas *tree* with n vertices, numbered from 1 to n and rooted at vertex 1. For each i ($2 \le i \le n$), vertex p_i is the parent of vertex i. The tree is decorated beautifully with a_i ornaments on vertex i ($1 \le i \le n$).

However, it has already been a few months since Christmas, and it is time to take down all the ornaments and put your tree away until next year. Since this process is tedious, you have decided to make it more fun by using only the following operation, which you can perform zero or more times:

> Choose a vertex u. For each vertex v on the unique simple path from vertex 1 to vertex u (inclusive), remove exactly one ornament from v if there is any left.

While you are still determining the minimum number of operations needed, your little child modifies the number of ornaments on the tree. More precisely, your child makes q changes. In the j-th change, the number of ornaments on vertex u_j is modified to x_j ($1 \le j \le q$). Note that these changes are persistent; the effect of each change carries over to subsequent changes.

Note that you do not actually perform any operations on the tree.

For the initial configuration and after each change, your task is to determine the minimum number of operations needed to remove all the ornaments from the tree at each moment.

Input

The first line of input contains one integer t ($1 \le t \le 10\,000$) representing the number of test cases. After that, t test cases follow. Each of them is presented as follows.

The first line of each test case contains two integers n and q ($2 \le n \le 200\,000$; $1 \le q \le 200\,000$). The second line contains $n-1$ integers $p_2, p_3, \ldots, p_n$ ($1 \le p_i < i$ for all $2 \le i \le n$). The third line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$ for all $1 \le i \le n$).

The j-th of the next q lines contains two integers u_j and x_j ($1 \le u_j \le n$; $1 \le x_j \le 10^9$).

The sum of n across all test cases in one input file does not exceed $200\,000$.

The sum of q across all test cases in one input file does not exceed $200\,000$.

Output

For each test case, output $q+1$ lines. The first line should contain the minimum number of operations required for the initial tree. The j-th of the following q lines should contain the minimum number of operations required after the j-th change happens.

Sample Input #1	Sample Output #1
2 6 3 1 1 2 3 3 5 3 2 5 2 4 4 1 3 10 1 6 8 6 1 2 3 4 5 5 3 1 1 1 1 1 1 1 1 6 3 8 3 5 5 6 1 3 7 5 1	11 9 13 13 3 5 7 8 8 8 7

Explanation for the sample input/output #1

For the first test case, its initial tree is illustrated in Figure D.1. Note that the star at vertex 1 in the figure is also an ornament.

- In the initial tree, you can remove all ornaments with 11 operations by choosing vertex 4 five times, vertex 5 twice, and vertex 6 four times.
- After the first change, there is only one ornament on vertex 4. You need 9 operations; choose vertex 4 once, vertex 2 twice, vertex 5 twice, and vertex 6 four times.
- After the second change, there are ten ornaments on vertex 3. You need 13 operations.
- After the third change, there are six ornaments on vertex 1. However, the required number of operations is unchanged.

Problem E
Parallel Sums
Time limit: 3 seconds

You are given two integers n and m. For a sequence of n integers $A = (a_1, a_2, \ldots, a_n)$, the *parallel sums* of A are the $n - m + 1$ integers $s_1, s_2, \ldots, s_{n-m+1}$ defined by $s_i = a_i + a_{i+1} + \ldots + a_{i+m-1}$ for each i ($1 \le i \le n - m + 1$).

You are given the values of $s_1, s_2, \ldots, s_{n-m+1}$. Your task is to answer q queries, described as follows. In the j-th query, you are given two integers l_j and r_j, and you are asked to find the smallest possible value of $\max(a_{l_j}, a_{l_j+1}, \ldots, a_{r_j})$ among all sequences $A = (a_1, a_2, \ldots, a_n)$ of n integers (possibly negative) such that $s_1, s_2, \ldots, s_{n-m+1}$ are the parallel sums of A. Or determine if this value can be arbitrarily small.

Input

The first line of input contains two integers n and m ($1 \le m \le n \le 200\,000$).

The second line contains $n - m + 1$ integers $s_1, s_2, \ldots, s_{n-m+1}$ ($-10^9 \le s_i \le 10^9$).

The third line contains a single integer q ($1 \le q \le 100\,000$).

The j-th of the next q lines contains two integers l_j and r_j ($1 \le l_j \le r_j \le n$).

Output

Output q lines. The j-th line should contain the smallest possible value of $\max(a_{l_j}, a_{l_j+1}, \ldots, a_{r_j})$. If that value can be arbitrarily small, output `unbounded` instead.

Sample Input #1	Sample Output #1
8 4 4 -4 2 6 5 4 3 7 4 6 1 8 2 5	2 unbounded 4 -1

Explanation for the sample input/output #1

For the first query, take $A = (9, -4, -3, 2, 1, 2, 1, 1)$. The parallel sums of A are $(4, -4, 2, 6, 5)$ as required. Then $\max(a_3, \ldots, a_7) = \max(-3, 2, 1, 2, 1) = 2$. It can be shown that 2 is the smallest possible value.

For the second query, you can make the value arbitrarily small.

For the third query, take $A = (4, -3, 0, 3, -4, 3, 4, 2)$. Then $\max(4, -3, 0, 3, -4, 3, 4, 2) = 4$, which is the smallest possible value.

This page is intentionally left blank.

Problem F
Minesweeper String
Time limit: 2 seconds

You are given a string S of length n consisting of digits `0`–`9`. You want to use this string to generate a variant of Minesweeper. In this variant, a cell may contain more than one mine, and the mine count in a cell is based on its 4-neighborhood (edge-adjacent cells), instead of the usual 8-neighborhood.

To do that, you choose an integer w ($1 \le w \le n$) representing the width of the grid. You arrange the n cells, numbered from 0 to $n-1$, into a grid. The grid has $\lceil n/w \rceil$ rows numbered from 0 to $\lceil n/w \rceil - 1$ from top to bottom, and w columns numbered from 0 to $w-1$ from left to right. For each $0 \le i < n$, cell i is located at row $\lfloor i/w \rfloor$ and column $i \bmod w$, and corresponds to the $(i+1)$-th digit of S. Thus, row 0 contains cells 0 to $w-1$, row 1 contains cells w to $2w-1$, and so on. Note that the bottommost row may contain less than w cells.

After arranging the grid, you perform the following steps:

1. For each cell corresponding to a non-zero digit x (`1`–`9`), you place x mines in that cell.
2. For each remaining cell corresponding to `0`, you write a number in that cell that is equal to the total number of mines in its adjacent cells. Two cells are adjacent if they share an edge. Note that each cell has at most **four** adjacent cells.

For a width w, define $f(w)$ as the sum of the numbers in all cells without mines on the generated grid. Given an integer k, your task is to compute the k-th largest value among the values $f(1), f(2), \ldots, f(n)$.

Input

The first line of input contains two integers n and k ($1 \le k \le n \le 500\,000$).

The second line contains a string S of length n consisting of digits `0`–`9`.

Output

Output the k-th largest value among $f(1), f(2), \ldots, f(n)$.

Sample Input #1	Sample Output #1
5 3 20103	7

Explanation for the sample input/output #1

Figure F.1 illustrates the resulting grids for widths 1 to 5. Each dot in a cell represents one mine.

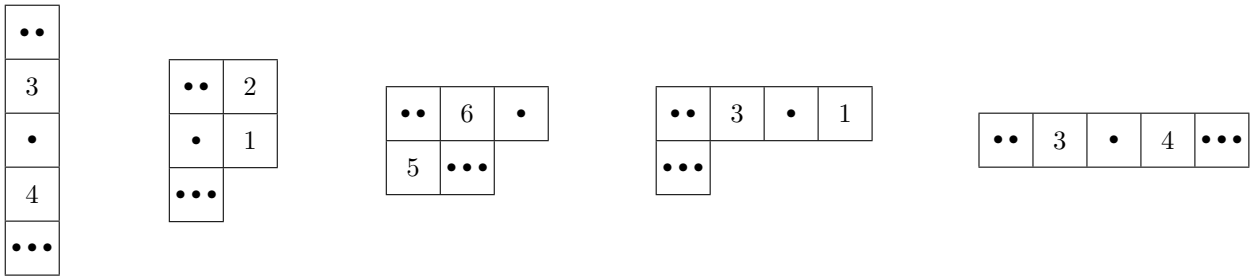


Figure F.1: The resulting grids for all 5 possible widths.

By summing up the numbers on the cells containing no mines, you obtain the following.

- $f(1) = 7$
- $f(2) = 3$
- $f(3) = 11$
- $f(4) = 4$
- $f(5) = 7$

The third largest value among them is 7.

Sample Input #2	Sample Output #2
5 1 20103	11

Sample Input #3	Sample Output #3
8 4 60409003	35

Problem G
Extra Transition
Time limit: 3 seconds

You are developing a game consisting of n levels, numbered from 1 to n. Levels are connected by a *transition network* consisting of m transitions, numbered from 1 to m. Transition k connects levels a_k and b_k bidirectionally ($1 \le k \le m$).

A single run of the game starts at level 1. Each time a player enters a new level, the player must complete it and then move to another level that is directly connected by a transition and has not been completed in the run. The run ends successfully when level n is completed.

A sequence of pairwise distinct levels starting from level 1 and ending at level n is called a *successful path* if a player can complete the levels in the order given by the sequence in a single run.

A transition network is *well-designed* if it satisfies both of the following:

- For any level i ($2 \le i \le n-1$), there exists a successful path containing level i.
- For any pair of levels i and j ($2 \le i < j \le n-1$), at most one of the following holds:
  - There exists a successful path containing levels i and j with level i appearing before level j.
  - There exists a successful path containing levels i and j with level j appearing before level i.

Your *first* task is to determine whether the given transition network is well-designed or not.

If the network is well-designed, you have a *second* task. Let S be the set of pairs of integers (i, j) ($1 \le i < j \le n$) such that levels i and j are not directly connected and adding a bidirectional extra transition between them keeps the network well-designed. You are interested in the set S. You are given a sequence of n integers $w_1, \ldots, w_n$. As a summary of S, compute the following sum:

$$\sum_{(i,j)\in S} w_i w_j.$$

Input

The first line of input contains one integer t ($1 \le t \le 50\,000$) representing the number of test cases. After that, t test cases follow. Each of them is presented as follows.

The first line of each test case contains two integers n and m ($3 \le n \le 200\,000$; $n-1 \le m \le 200\,000$).

The second line contains n integers $w_1, w_2, \ldots, w_n$ ($1 \le w_i \le 5000$ for all i).

The k-th of the next m lines contains two integers a_k and b_k ($1 \le a_k < b_k \le n$; $(a_k, b_k) \ne (a_\ell, b_\ell)$ for all $k \ne \ell$).

The input guarantees that the transition network is *connected*; for any levels i and j ($i \ne j$), there exists a sequence of transitions leading from level i to level j.

The sum of n across all test cases in one input file does not exceed 200 000.

The sum of m across all test cases in one input file does not exceed 200 000.

Output

For each test case, if the transition network is not well-designed, output `bad`. Otherwise, output the sum defined above.

Sample Input #1	Sample Output #1
3 4 4 1 2 3 4 1 2 1 3 2 4 3 4 3 2 2026 3 9 1 3 2 3 10 11 15 51 82 49 1 55 45 5 25 91 7 10 1 6 2 5 4 7 3 8 1 9 4 6 2 10 3 9 5 9 2 8	4 bad 23336

Explanation for the sample input/output #1

For the first test case, the given transition network is well-designed. Possible candidates for adding extra transitions (i, j) are $(1, 4)$ and $(2, 3)$.

- For $(1, 4)$, adding a transition between them keeps the network well-designed.
- For $(2, 3)$, on the other hand, adding a transition between them does not keep the network well-designed. There exist two successful paths $(1, 2, 3, 4)$ and $(1, 3, 2, 4)$. The second condition for the pair of levels 2 and 3 is not satisfied.

Thus, the answer is $w_1 w_4 = 1 \times 4 = 4$.

For the second test case, the given transition network is not well-designed because there is no successful path containing level 2.

Problem H
Reflect Sort
Time limit: 2 seconds

You have a sequence of n integers $(a_1, a_2, \ldots, a_n)$, the initial values of which are given to you. You may apply the following operation to the sequence any number of times (possibly zero):

1. Choose an index i ($1 \leq i \leq n$).
2. Choose a set S to be either the prefix $\{1, 2, \ldots, i-1\}$ or the suffix $\{i+1, i+2, \ldots, n\}$.
3. For every $j \in S$, replace a_j with $2a_i - a_j$.

Your goal is to make the sequence *non-decreasing* and all elements *positive*, while minimizing a_n, using the operation above any number of times. That is, $1 \leq a_1 \leq a_2 \leq \ldots \leq a_n$ must hold in the resulting sequence. What is the minimum possible value of a_n in the resulting sequence?

Input

The first line of input contains a single integer n ($2 \leq n \leq 100\,000$).

The second line contains n integers representing the initial values of $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$).

Output

Output the minimum possible value of a_n in the resulting sequence.

Sample Input #1	Sample Output #1
5 6 3 5 5 2	10

Explanation for the sample input/output #1

You can do the following sequence of operations.

1. Choose $i = 1$ and S as the suffix $\{2, 3, 4, 5\}$: $(6, 3, 5, 5, 2) \to (6, 9, 7, 7, 10)$.
2. Choose $i = 3$ and S as the prefix $\{1, 2\}$: $(6, 9, 7, 7, 10) \to (8, 5, 7, 7, 10)$.
3. Choose $i = 2$ and S as the prefix $\{1\}$: $(8, 5, 7, 7, 10) \to (2, 5, 7, 7, 10)$.

After these operations, the sequence is non-decreasing and all elements are positive. It can be shown that $a_5 = 10$ is the minimum possible value.

Sample Input #2	Sample Output #2
3 2 1 100000	100002

Explanation for the sample input/output #2

The minimum possible value of a_3 is 100002, which can be attained by the following operations.

1. Choose $i = 2$ and S as the suffix $\{3\}$: $(2, 1, 100000) \to (2, 1, -99998)$.
2. Choose $i = 1$ and S as the suffix $\{2, 3\}$: $(2, 1, -99998) \to (2, 3, 100002)$.

Note that the sequence may contain non-positive values during operations.

Problem I
Growth Factor
Time limit: 2 seconds

You are given an integer n and a sequence of integers $a_1, a_2, \ldots, a_n$. Your task is to determine the number of integer sequences $(b_1, b_2, \ldots, b_n)$ such that the following conditions are satisfied:

- $1 \leq b_i \leq a_i$ for each i $(1 \leq i \leq n)$.
- b_i is a factor of b_{i+1} for each i $(1 \leq i \leq n-1)$.

Two sequences are considered different if they differ in at least one position.

Since the number of such sequences may be large, compute the answer modulo $998\,244\,353$.

Input

The first line of input contains a single integer n $(1 \leq n \leq 200\,000)$.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 200\,000)$.

Output

Output the number of distinct sequences satisfying the conditions, modulo $998\,244\,353$.

Sample Input #1	Sample Output #1
2 2 4	6

Explanation for the sample input/output #1

The following are all sequences satisfying the conditions: $(2, 4)$, $(2, 2)$, $(1, 4)$, $(1, 3)$, $(1, 2)$, and $(1, 1)$.

Sample Input #2	Sample Output #2
6 265 9801 192168 200000 192018 199809	16555779

This page is intentionally left blank.

Problem J
Worldwide Playlist
Time limit: 2 seconds

You are planning a trip around the world. For this trip, you have installed a music app containing n songs, numbered from 1 to n.

Initially, the app generates a playlist for these n songs in the form of a permutation of $1, 2, \ldots, n$, denoted by $a_1, a_2, \ldots, a_n$. It plays the n songs in the order $a_1, a_2, \ldots, a_n$: song a_1 plays first, song a_2 plays second, and so on. The playlist is *infinite*: each time after song a_n plays, it starts all over from song a_1.

The next song starts playing automatically when the current song finishes. Before a song finishes, you may instead press a *skip* button to immediately jump to the next song in the playlist.

You have a desired permutation of the n songs, denoted by $b_1, b_2, \ldots, b_n$. This means that you want to listen to n songs *in full* in the order of $b_1, b_2, \ldots, b_n$ by strategically pressing the skip button zero or more times. In other words, you want the first song you listen to in full (not skipped) to be song b_1, the second to be song b_2, and so on, until the n-th is song b_n. After listening to these n songs in full, you stop listening.

You use this playlist for d days. Between each pair of consecutive days, you update the permutations using three integers c, x, and y as follows:

- If $c = 1$, then you swap a_x and a_y.
- If $c = 2$, then you swap b_x and b_y.

Note that the effect of each update persists to subsequent days.

For each day, assuming you start listening from song a_1, determine the minimum number of times you need to press the skip button so that the songs you listen to in full are $b_1, b_2, \ldots, b_n$ in this order.

Input

The first line of input contains two integers n and d ($2 \le n \le 200\,000$; $2 \le d \le 200\,000$).

The second line contains n integers representing the initial values of $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$; $a_i \ne a_j$ for all $i \ne j$).

The third line contains n integers representing the initial values of $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le n$; $b_i \ne b_j$ for all $i \ne j$).

The k-th of the next $d - 1$ lines contains three integers c, x, and y ($c \in \{1, 2\}$; $1 \le x < y \le n$), representing the update between the k-th and $(k + 1)$-th days.

Output

Output d lines, where the k-th line should contain the minimum number of skips for the k-th day.

Sample Input #1	Sample Output #1
4 3 1 4 2 3 3 2 1 4 1 3 4 2 1 3	6 2 6

Explanation for the sample input/output #1

On the first day, $(a_1, \ldots, a_4) = (1, 4, 2, 3)$ and $(b_1, \ldots, b_4) = (3, 2, 1, 4)$. You can listen to these songs in full in the desired order with 6 skips:

- Song 1 plays. Skip this song.
- Song 4 plays. Skip this song.
- Song 2 plays. Skip this song.
- Song 3 plays. Listen to this song in full.
- Song 1 plays. Skip this song.
- Song 4 plays. Skip this song.
- Song 2 plays. Listen to this song in full.
- Song 3 plays. Skip this song.
- Song 1 plays. Listen to this song in full.
- Song 4 plays. Listen to this song in full.

On the second day, the permutations are $(a_1, \ldots, a_4) = (1, 4, \mathbf{3}, \mathbf{2})$ and $(b_1, \ldots, b_4) = (3, 2, 1, 4)$. The minimum number of skips is 2.

On the third day, the permutations are $(a_1, \ldots, a_4) = (1, 4, 3, 2)$ and $(b_1, \ldots, b_4) = (\mathbf{1}, 2, \mathbf{3}, 4)$. The minimum number of skips is 6.

Sample Input #2	Sample Output #2
7 5 4 7 1 2 6 5 3 2 6 5 1 4 3 7 1 2 5 2 6 7 1 6 7 2 1 5	16 26 21 20 6

Problem K
Time Display Stickers
Time limit: 2 seconds

You have a collection of n digit stickers, represented by a string S of length n. Each character of S is a digit `0` through `9`, representing one sticker of that digit.

You want to create *time displays* using these stickers. Each time display shows a time in the format `HH:MM`, where:

- `HH` is a two-digit hour between 00 and 11 (inclusive), and
- `MM` is a two-digit minute between 00 and 59 (inclusive).

In other words, each time display requires exactly four stickers: two for the hours and two for the minutes. Each sticker can only be used for at most one time display.

What is the maximum number of time displays you can create?

Input

The first line of input contains one integer t ($1 \le t \le 10\,000$) representing the number of test cases. After that, t test cases follow. Each of them is presented as follows.

The first line of each test case contains an integer n ($1 \le n \le 10^6$).

The second line contains a string S of length n, consisting only of digits `0`–`9`.

The sum of n across all test cases in one input file does not exceed 10^6.

Output

For each test case, output the maximum number of time displays you can create.

Sample Input #1	Sample Output #1
4 10 0123456789 11 00123456789 8 99111111 4 1234	1 2 2 0

Explanation for the sample input/output #1

For the first test case, you can create one time display `10:59`. It can be shown that you cannot create two displays for a given collection of stickers.

For the second test case, you can create two time displays: `10:59` and `04:27`.

For the third test case, you can create two time displays: `11:19` and `11:19`.

For the fourth test case, you cannot create any time displays.

Problem L
Onion
Time limit: 5 seconds

Onions are widely used in cuisines around the world. This problem is about an "onion peeling" process in geometry.

A set of points in a two-dimensional plane is *convex* if, for any pair of points p and q in the set, the line segment connecting p and q is entirely contained in the set. For a set of points S, its *convex hull* is the smallest convex set containing all points in S.

You are given four integers n, a, b, and k. You have a set S of n points, initialized as follows:

$$S = \{(x, (ax + b) \bmod n) \mid x = 0, 1, \ldots, n - 1\}.$$

You apply the following operation k times: let H be the convex hull of the set S, and then remove from S all points that lie on the boundary of the convex hull H.

Note that S can become empty. In that case, its convex hull is also empty, and its area is zero.

For each operation, determine the *doubled* area of the convex hull H. It can be shown that this value is always an integer.

Input

The input consists of a single line containing four integers n, a, b, and k ($1 \le n \le 10^9$; $0 \le a, b < n$; $1 \le k \le 300$).

Output

Output k lines. The i-th line should contain an integer representing the doubled area of the convex hull in the i-th operation.

Sample Input #1	Sample Output #1
4 1 2 1	8

Explanation for the sample input/output #1

Figure L.1 illustrates the points in S and the boundary of the convex hull in the first operation. The area of the convex hull is 4, and thus the doubled area is 8.

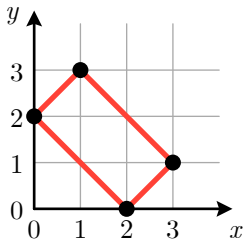


Figure L.1: Illustration of Sample Input #1.

Sample Input #2	Sample Output #2
37 14 7 5	2257 1406 592 74 0

Explanation for the sample input/output #2

Figure L.2 illustrates the boundaries in the first four operations. The set S becomes empty after the fourth operation. The area for the fifth operation is 0.

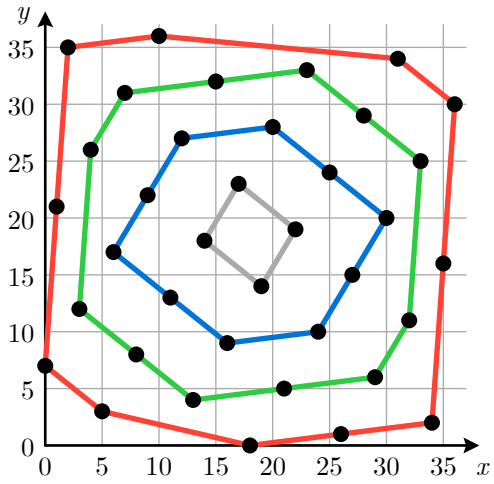


Figure L.2: Illustration of Sample Input #2.

Sample Input #3	Sample Output #3
280 40 12 4	131040 82880 39200 0

Problem M
Deformed Balance
Time limit: 2 seconds

In this problem, a concatenation of two strings T and U is denoted by $T + U$.

A string consisting only of parentheses (opening parentheses '`(`' or closing parentheses '`)`') is *balanced* if and only if it is one of the following.

- An empty string.
- The concatenation of two non-empty balanced strings.
- The concatenation "`(`" $+ T +$ "`)`", where T is a balanced string.

For example, "`()`" and "`(()())`" are balanced, while "`()(`" and "`(((`" are not.

A string is *deformed* if and only if it is one of the following.

- The string "`)`".
- The concatenation $T +$ "`)`" $+ U$, where T and U are deformed strings.
- The concatenation "`(`" $+ T +$ "`(`", where T is a deformed string.

For example, "`()(`" and "`))()(`" are deformed, while "`()`" and "`(()`" are not.

A string T *has a deformed balance* if T is deformed and the concatenation $T +$ "`)`" is balanced. For example, the string "`()(`" has a deformed balance.

You are given a string S of length n consisting only of parentheses. Under the given input constraints, it can be shown that there exist strings X and Y such that the concatenation $X + S + Y$ has a deformed balance. Determine the minimum possible value of $|X| + |Y|$ (the sum of the lengths of X and Y).

Input

The first line of input contains one integer t ($1 \le t \le 10\,000$) representing the number of test cases. After that, t test cases follow. Each of them is presented as follows.

The first line of each test case contains an integer n ($1 \le n \le 10^6$).

The second line contains a string S of length n, consisting only of '`(`' or '`)`'.

The sum of n across all test cases in one input file does not exceed 10^6.

Output

For each test case, output the minimum possible value of $|X| + |Y|$ such that the concatenation $X + S + Y$ has a deformed balance.

Sample Input #1	Sample Output #1
3 3 ()(1) 7 (())())	0 2 4

Explanation for the sample input/output #1

For the first test case, the given string already has a deformed balance.

For the second test case, setting both X and Y to "(" yields the concatenation "()(", which has a deformed balance. The value of $|X| + |Y|$ is 2.

For the third test case, it suffices to set X to "(((" and Y to an empty string to attain the minimum value of $|X|+|Y|$.