



The 2026 ICPC Vietnam Southern Provincial Programming Contest

Date: October 18th, 2026

Problem A: Bromine playing Billiard	1
Problem B: Crystal Reactor	3
Problem C: Sodium Sort	5
Problem D: Late Failure	7
Problem E: Dynamic Memory	8
Problem F: Locker Riddle I	10
Problem G: Black Start	12
Problem H: DNA Reconstruction	14
Problem I: Lost Layers	16
Problem J: Phosphorus studying Polygon	18
Problem K: Locker Riddle II	20
Problem L: Grief	22

Problem A

Bromine playing Billiard

Time limit: 1.5s
Memory limit: 1GB

Bromine is playing on a rectangular billiard table whose sides are parallel to the coordinate axes. Its bottom-left corner is $(0.5, 0.5)$, and its top-right corner is $(n + 0.5, m + 0.5)$, so the table has dimensions $n \times m$. He places a cue ball of diameter 1 on the table, with its center initially at (x, y) .

Bromine strikes the cue ball, giving it an initial velocity vector of (v_x, v_y) . The components v_x and v_y describe the initial velocity of its center along the X - and Y -axes, respectively, in coordinate units per second.

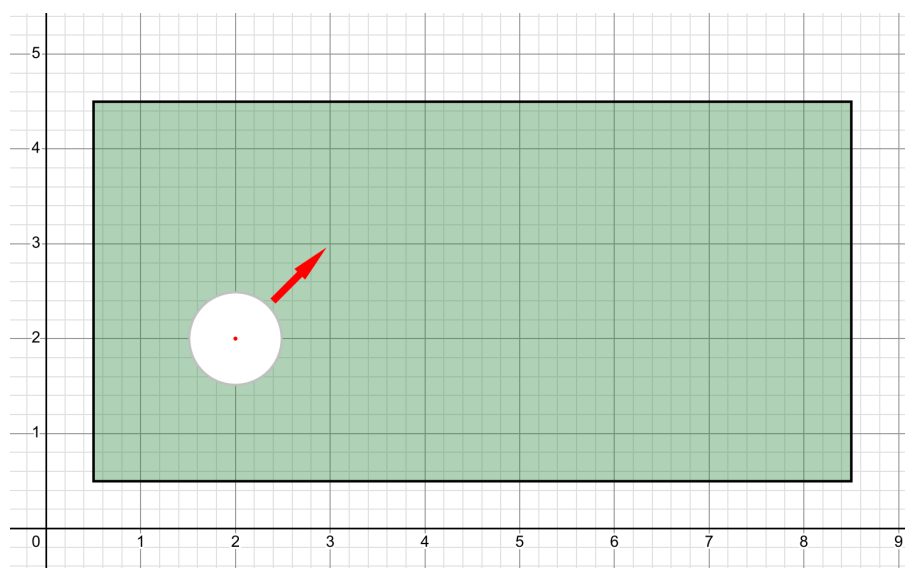


Figure 1: Bromine's billiard table. The cue ball's center is initially at $(2, 2)$, and its initial velocity is $(1, 1)$.

Assume that the ball moves indefinitely without losing any speed. Whenever it hits a table cushion, it undergoes a perfectly elastic collision. The reflection rules are:

- When the ball's center reaches $X = 1$ or $X = n$, it hits a vertical cushion, and the X -component of its velocity is negated.
- When the ball's center reaches $Y = 1$ or $Y = m$, it hits a horizontal cushion, and the Y -component of its velocity is negated.

If the ball hits two cushions simultaneously, both components of its velocity are negated. If the ball initially touches a cushion and its velocity points toward that cushion, it reflects immediately at time 0.

Bromine is an exceptionally skilled billiard player. He can make every shot and calculate where the cue ball will be for up to one billion seconds in the future.

Bromine is tired of not being able to find a rival, so to test if someone is worthy to play a game with him, Bromine will ask them this kind of question: "Given a time t , you need to determine the exact (X, Y) coordinates of the ball's center exactly t seconds after it was struck."

You want to have a game with Bromine. Thus, he asked you q queries, and you need to answer all of them quickly!

Input

- The first line contains six integers n, m, x, y, v_x, v_y ($2 \leq n, m \leq 10^9$, $1 \leq x \leq n$, $1 \leq y \leq m$, $-10^9 \leq v_x, v_y \leq 10^9$), the dimensions of the table, the initial coordinates of the ball's center, and its initial velocity vector.
- The second line contains a single integer q ($1 \leq q \leq 10^5$), the number of queries.
- Each of the next q lines contains a single integer t ($0 \leq t \leq 10^9$), the time in seconds for the query.

Output

Print q lines. The i -th line should contain two integers X_t and Y_t — the coordinates of the ball's center at t seconds for the i -th query. It can be shown that both coordinates of the ball's center are integers at every queried time.

Sample Input	Sample Output
8 4 2 2 1 1	2 2
5	3 3
0	4 4
1	5 3
2	6 2
3	
4	

Notes

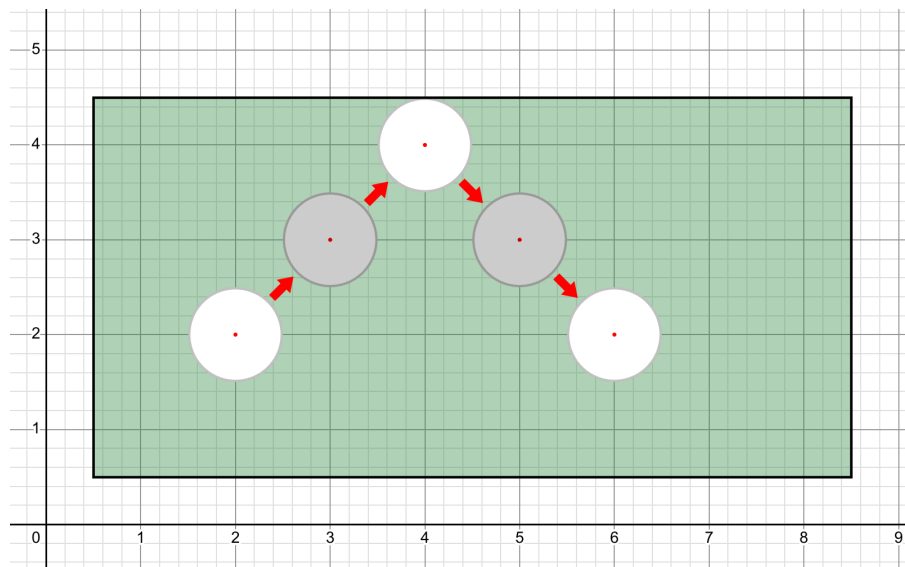


Figure 2: The positions of the cue ball's center at $t = 0, 1, 2, 3, 4$.

Problem B

Crystal Reactor

Time limit: 2s
Memory limit: 256MB

A research facility contains a three-dimensional crystal reactor consisting of $N \times M \times P$ cells. Every cell (x, y, z) , where $1 \leq x \leq N$, $1 \leq y \leq M$ and $1 \leq z \leq P$, initially contains zero energy. We interpret x as the row coordinate, y as the column coordinate, and z as the layer coordinate.

The reactor is activated Q times. During one activation, an energy source is placed at the corner (x_1, y_1, z_1) and affects every cell inside the axis-aligned cuboid $[x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$.

At the source cell (x_1, y_1, z_1) , this activation contributes S units of energy. Starting from that corner, every step of one cell in the positive X direction increases the contribution by A , every step in the positive Y direction increases it by B , and every step in the positive Z direction increases it by C .

The effect exists only inside the given cuboid. If several sources affect the same cell, all of their contributions are added.

After all Q activations have been applied, the control system inspects L cells. For each inspected cell, report the final energy stored in that cell.

Input

- The first line contains five integers N, M, P, Q and L : the reactor dimensions, the number of activations and the number of inspected cells ($1 \leq N, M, P \leq 200$, $N \times M \times P \leq 2 \cdot 10^6$, $1 \leq Q \leq 4 \cdot 10^5$, $1 \leq L \leq 10^6$).
- Each of the next Q lines contains ten integers $x_1, y_1, z_1, x_2, y_2, z_2, S, A, B$ and C ($1 \leq x_1 \leq x_2 \leq N$, $1 \leq y_1 \leq y_2 \leq M$, $1 \leq z_1 \leq z_2 \leq P$, $|S|, |A|, |B|, |C| \leq 10^6$). They describe one activation with source corner (x_1, y_1, z_1) , opposite corner (x_2, y_2, z_2) , initial energy S , and per-step increases A, B and C along the positive X, Y and Z axes, respectively. All cuboid endpoints are inclusive.
- Each of the next L lines contains three integers x, y and z ($1 \leq x \leq N$, $1 \leq y \leq M$, $1 \leq z \leq P$): the coordinates of an inspected cell. Inspection coordinates may repeat.

It is guaranteed that every final cell energy fits in a signed 64-bit integer.

Output

Print L lines. On the i -th line, print the final energy of the i -th inspected cell, in the same order as the inspection coordinates appear in the input.

Sample Input	Sample Output
2 2 3 2 4	10
1 1 1 2 2 3 10 100 10 1	116
2 1 2 2 2 3 5 0 2 3	132
1 1 1	21
2 1 2	
2 2 3	
1 2 2	

Notes

For the first activation, the source at $(1, 1, 1)$ contributes 10 there. Moving one cell along X adds 100, along Y adds 10, and along Z adds 1. Thus, for example, the first activation contributes 111 at $(2, 1, 2)$ and 122 at $(2, 2, 3)$.

The second source is at $(2, 1, 2)$. It contributes 5 at that cell; one step along Y adds 2 and one step along Z adds 3. Therefore the final energies at the four inspected cells are 10, 116, 132 and 21.

- Activations may overlap; all contributions are cumulative.
- S , A , B and C may be negative.
- There are no inspections between activations; all L inspections occur after all Q activations.
- The inspection points are arbitrary; no distributional assumption should be made about them.

Problem C

Sodium Sort

Time limit: 2s

Memory limit: 1GB

Sodium has implemented his own version of the Quicksort algorithm, called Sodium Sort, to sort an array a of length n . To choose the pivot for the partitioning step, he uses a pre-generated sequence of integers R .

Sodium Sort is implemented with the following pseudo-code:

```
int it = 1;
long long tot_length = 0;

void sodium_sort(int l, int r) {
    if (l > r) return;
    tot_length += r - l + 1;
    int pivot_index = l + (R[it] % (r - l + 1));
    it++;
    int p = partition(l, r, pivot_index);
    sodium_sort(l, p - 1);
    sodium_sort(p + 1, r);
}
```

It can be seen that the global variable `tot_length` acts as a cost measure representing the total length of all subarrays processed during the sort. Furthermore, the global counter `it` is incremented exactly n times when sorting an array of size n .

When `partition(l, r, pivot_index)` is called on a subarray, it uses the element at index `pivot_index` as the pivot value x and rearranges the subarray $a[l \dots r]$ such that:

- All elements less than x are moved to the left of the pivot.
- All elements greater than x are moved to the right of the pivot.
- Elements placed in the left and right subarrays retain their relative order from immediately before the current call. For example: When calling `partition(1, 5, 5)` on $[1, 5, 2, 4, 3]$ (where the pivot value is $x = 3$), the final array becomes $[1, 2, 3, 5, 4]$.
- It returns the new index p where the pivot x is finally placed.

Sodium will initially call `sodium_sort(1, n)` to sort his array. Sodium Sort always sorts the array correctly, but its running time depends on the initial order of the elements. Sodium's friend Sulfur wants to hack Sodium Sort by finding an input that makes it as slow as possible.

Your task is to help Sulfur. Given the length of the array n and the sequence R , construct a permutation of the integers from 1 to n for the array a such that the final value of the global counter `tot_length` is maximized.

Input

- The first line contains a single integer n ($1 \leq n \leq 2 \times 10^5$) — the size of the array to be sorted.
- The second line contains n integers $R_1, R_2, \dots, R_n$ ($0 \leq R_i \leq 10^9$) — the elements of the pre-generated sequence used for Sodium's pivot selection.

Output

Output a single line containing n integers, representing a permutation of $1, 2, \dots, n$ that maximizes the final value of the counter `tot_length` after the array is completely sorted. (A permutation of length n is an array containing each integer from 1 to n exactly once). If there are multiple valid permutations, you may output any of them.

Sample Input	Sample Output
3 5 2 9	2 3 1

Problem D

Late Failure

Time limit: 1s

Memory limit: 256MB

A string is called a *palindrome* if it reads the same from left to right and from right to left.

To check whether a string is a palindrome, a programmer uses the following C++ function:

```
bool isPalindrome(string s) {  
    int low = 0;  
    int high = s.size() - 1;  
  
    while (low < high) {  
        if (s[low] != s[high])  
            return false;  
        ++low;  
        --high;  
    }  
    return true;  
}
```

You are given a string S consisting of uppercase English letters. You may rearrange the letters of S in any order you like (keeping the original order is also allowed) to obtain a new string T .

You want `isPalindrome(T)` to return `false`, but you also want the function to detect this as late as possible.

Find the largest number of iterations of the `while` loop that can be executed before the function returns `false`. The iteration in which the function returns `false` is counted, so this is the number of times the condition `s[low] != s[high]` is checked. If no rearrangement makes the function return `false`, print `-1` instead.

Input

- The only line contains the string S ($2 \leq |S| \leq 2 \cdot 10^5$). It consists only of uppercase English letters from A to Z.

Output

Print a single integer: the largest number of iterations described above, or `-1` if no rearrangement makes the function return `false`.

Sample Input	Sample Output
AABBCDEF	3
AAAAA	-1

Notes

In the first example, the letters can be arranged as `ABCDEFBA`. The first two comparisons both find two equal letters (A and A, then B and B). At the third comparison the function meets two different letters, C and F, and returns `false`. So the loop is executed 3 times.

In the second example, every rearrangement gives the same string `AAAAA`, which is a palindrome, so the function can never return `false`.

Problem E

Dynamic Memory

Time limit: 2s
Memory limit: 1GB

A memory system consists of 2^k cells, numbered from 0 to $2^k - 1$. Each cell stores a single bit, either 0 or 1.

The system supports the operation $\text{STORE}(i, j, x)$, where $0 \leq i \leq k$, $0 \leq j < 2^{k-i}$ and $x \in \{0, 1\}$. This operation sets every cell in the interval $[j \cdot 2^i, (j+1) \cdot 2^i - 1]$, both ends included, to x . Its cost is w_i , regardless of j and x . Operations are performed one after another, so a later operation may overwrite values written by earlier operations.

You are also given a target configuration T consisting of 2^k bits. Initially, every bit of T is 0. There are q queries, each given by two integers L and R . A query flips every bit of T in the interval $[L, R]$, both ends included:

$$T[p] \leftarrow T[p] \oplus 1 \quad (L \leq p \leq R),$$

where $\oplus$ denotes bitwise XOR. The interval $[L, R]$ can be any contiguous interval: unlike the interval of a STORE operation, its length need not be a power of two and it need not satisfy any alignment restriction.

The queries change T cumulatively. After each query, find the minimum total cost of a sequence of STORE operations that turns a memory in which every cell contains 0 into the current configuration T . The operations considered for earlier queries play no role. The sequence may be empty, so the answer is 0 when every bit of T is 0.

Input

- The first line contains two integers k and q ($1 \leq k \leq 30$, $1 \leq q \leq 10^5$).
- The second line contains $k+1$ integers $w_0, w_1, \dots, w_k$ ($1 \leq w_i \leq 10^6$), where w_i is the cost of a STORE operation on an interval of length 2^i .
- Each of the next q lines contains two integers L and R ($0 \leq L \leq R < 2^k$), describing a query that flips the bits of T at all positions from L to R , inclusive.

Output

Print q lines. The t -th line must contain a single integer: the minimum total cost of obtaining the target configuration T after the t -th query.

The answers can exceed 10^{12} , which is beyond the range of a 32-bit integer type. Use a 64-bit integer type.

Sample Input	Sample Output
3 3 1 1 1 1 0 7 2 3 0 1	1 2 1
2 4 2 5 1 0 3 2 2 0 3 1 1	1 3 2 4

Notes

In the first example, there are $2^3 = 8$ memory cells and every **STORE** operation costs 1.

- After query 1, $T = [1, 1, 1, 1, 1, 1, 1, 1]$. The single operation **STORE**(3, 0, 1) sets all eight cells to 1. The minimum cost is 1.
- After query 2, $T = [1, 1, 0, 0, 1, 1, 1, 1]$. Apply **STORE**(3, 0, 1), followed by **STORE**(1, 1, 0). The minimum cost is 2.
- After query 3, $T = [0, 0, 0, 0, 1, 1, 1, 1]$. The single operation **STORE**(2, 1, 1) sets the last four cells to 1. The minimum cost is 1.

In the second example, $w_0 = 2$, $w_1 = 5$ and $w_2 = 1$. After query 2, $T = [1, 1, 0, 1]$. Applying **STORE**(2, 0, 1) and then **STORE**(0, 2, 0) costs $1 + 2 = 3$. If no operation were allowed to overwrite a cell written by an earlier operation, the cheapest way would be to set cells 0, 1 and 3 one by one, at a cost of 6.

Problem F

Locker Riddle I

Time limit: 1s
Memory limit: 1GB

A room contains n lockers, numbered from 1 to n . Each locker is either open or closed. A group of n heirs must solve a puzzle with these lockers before they can receive their inheritance.

The heirs are also numbered from 1 to n . In each step, you may choose one heir to perform an operation. When the i -th heir is chosen, they flip the state of every locker whose index is a multiple of i . More precisely, for every positive integer j such that $i \times j \leq n$, the state of the $(i \times j)$ -th locker is changed from open to closed or vice versa. An heir may be chosen more than once.

The puzzle is solved when all n lockers are closed.

Find a sequence of operations that closes all the lockers, or determine that no such sequence exists.

Input

- The first line contains a single integer t ($1 \leq t \leq 100$), the number of test cases.
- For each test case:
 - The first line contains a single integer n ($1 \leq n \leq 100$), the number of lockers.
 - The second line contains n integers $a_1, a_2, \dots, a_n$ ($0 \leq a_i \leq 1$), the initial states of the lockers, where $a_i = 1$ means the i -th locker is open and $a_i = 0$ means it is closed.

Output

For each test case, output your answer as follows:

- If no valid sequence of operations exists, print a single integer -1 .
- Otherwise, print a single integer k ($0 \leq k \leq n^2$) on the first line, the number of operations in your sequence.
- If $k > 0$, print k integers $b_1, b_2, \dots, b_k$ ($1 \leq b_i \leq n$) on the second line, where b_i denotes the heir chosen in the i -th operation.

If there are multiple valid sequences, you may output any of them.

Sample Input	Sample Output
<pre> 8 3 0 0 0 3 0 0 1 3 0 1 0 3 0 1 1 3 1 0 0 3 1 0 1 3 1 1 0 3 1 1 1 </pre>	<pre> 0 1 3 1 2 2 2 3 3 1 2 3 2 1 2 2 1 3 1 1 </pre>
<pre> 3 7 1 0 0 0 0 1 1 11 1 1 1 1 1 1 0 1 0 1 0 15 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 </pre>	<pre> 4 1 2 3 5 4 1 7 9 11 1 1 </pre>

Notes

In the first test case of the first example, all lockers are already closed, so no operations are needed.

In the fifth test case of the first example, choosing heirs 1, 2, and 3, in this order, changes the locker states as follows: $(1, 0, 0) \rightarrow (0, 1, 1) \rightarrow (0, 0, 1) \rightarrow (0, 0, 0)$. Thus, the sequence shown in the output closes all lockers.

Problem G

Black Start

Time limit: 4s

Memory limit: 1GB

A storm has knocked out the regional power grid, and every substation is dark. The grid consists of n substations, numbered from 1 to n , and $n - 1$ transmission lines. Each line connects two different substations and has a positive integer length. Every substation can be reached from every other one along the lines, so the lines form a tree. The distance between two substations is the total length of the lines on the unique route between them.

Substation i is equipped with a booster of range r_i . As soon as substation i is energized, it energizes every substation whose distance from substation i is at most r_i . The newly energized substations energize further substations by the same rule, and so on, until no new substation can be energized. In particular, a substation with range 0 energizes no other substation.

To restart the grid, the operators switch on the emergency generators of some substations, which energizes these substations at the very beginning. Find the minimum number of substations whose generators must be switched on so that, in the end, every substation is energized.

Input

- The first line contains a single integer n ($1 \leq n \leq 3 \cdot 10^5$) — the number of substations.
- The second line contains n integers $r_1, r_2, \dots, r_n$ ($0 \leq r_i \leq 10^9$) — the ranges of the boosters.
- Each of the next $n - 1$ lines contains three integers u, v and ℓ ($1 \leq u, v \leq n, u \neq v, 1 \leq \ell \leq 10^9$), describing a transmission line of length ℓ between substations u and v . It is guaranteed that the lines form a tree.

Output

Print a single integer — the minimum number of substations whose generators must be switched on.

Sample Input	Sample Output
<pre> 7 4 3 0 5 2 0 7 1 2 4 2 3 3 2 4 5 4 5 2 4 6 6 6 7 1 </pre>	<pre> 2 </pre>
<pre> 4 10 10 0 0 1 2 10 2 3 10 3 4 1 </pre>	<pre> 2 </pre>

Notes

In the first example, switch on the generators of substations 1 and 7. Substation 7 energizes substation 6 (at distance 1) and substation 4 (at distance exactly 7). Substation 4 energizes substations 2 and 5, and

substation 2 energizes substation 3. No substation can energize substation 1 or substation 7; for example, substation 2 is at distance 4 from substation 1, but its range is only 3. So both generators are necessary.

In the second example, substations 1 and 2 energize each other, so a generator at either of them energizes substations 1, 2 and 3. Substation 4 is at distance 1 from substation 3, but the range of substation 3 is 0, so substation 4 needs its own generator.

Problem H

DNA Reconstruction

Time limit: 2s
Memory limit: 256MB

In bioinformatics, scientists often study DNA through short pieces of equal length, called K -mers. A DNA string is a string consisting only of the four characters A, C, G and T.

A DNA string of length $L \geq K$ yields $L - K + 1$ contiguous substrings of length K , starting at positions $1, 2, \dots, L - K + 1$. For example, for $K = 3$ the string ACGTA yields three K -mers: ACG, CGT and GTA.

After a sequencing experiment, the scientists obtained N pairwise distinct DNA fragments $S_1, S_2, \dots, S_N$, each of length exactly K . They want to reconstruct a shortest DNA string from these fragments.

A DNA string T is called *valid* if both of the following conditions hold:

- every S_i ($1 \leq i \leq N$) occurs at least once as a contiguous substring of T ;
- every contiguous substring of T of length K belongs to the set $\{S_1, S_2, \dots, S_N\}$.

Thus a fragment S_i may occur in T several times, but no K -mer outside the given list may occur in T .

Moreover, because of storage limits, the length of T must not exceed

$$N + K - 1 + R,$$

where R is a given non-negative integer.

Find the minimum length of a valid DNA string whose length does not exceed this limit. If no such string exists, print -1 .

Input

- The first line contains three integers N , K and R ($1 \leq N \leq 2 \cdot 10^5$, $2 \leq K \leq 12$, $0 \leq R \leq 15$).
- The i -th of the next N lines contains the string S_i , which has length exactly K and consists only of the characters A, C, G and T.

The strings $S_1, S_2, \dots, S_N$ are pairwise distinct.

Output

Print a single integer: the minimum length of a valid DNA string whose length is at most $N + K - 1 + R$, or -1 if no such string exists.

Sample Input	Sample Output
5 2 1 AC AG AT CG GA	7
5 2 0 AC AG AT CG GA	-1
2 3 9 ACT GGA	-1

Notes

In the first example, the string **ACGAGAT** has length 7 and contains all five given fragments; the fragment **GA** occurs in it twice. There is no valid string of length 6.

In the second example, at least 7 characters are needed to form a valid string, while the length limit is $5 + 2 - 1 + 0 = 6$.

In the third example, no string can contain both given fragments without also containing a 3-mer that is not in the list.

Problem I

Lost Layers

Time limit: 2s

Memory limit: 256MB

Linh is restoring a video project whose file was damaged. The project has n clips, numbered from 1 to n . Every clip lies on its own layer, and the layers are stacked one above another: when several clips are on screen at the same frame, a clip on a higher layer covers the clips below it. Clip i is on screen at frames $l_i, l_i + 1, \dots, r_i$, including both l_i and r_i .

The frame ranges of the clips survived, but the order of the layers was lost. Linh still has the log of the preview renderer. Each line of the log is a snapshot taken at some frame x . It names the clips that are on screen at frame x , starting from the top layer and going down, but it stops after the first k of them. The value of k may differ from snapshot to snapshot. It can be 0, and it can be equal to the number of clips on screen, in which case all of them are named.

Help Linh find an order of all n layers, from top to bottom, that agrees with every snapshot. An order agrees with a snapshot $(x, k, s_1, \dots, s_k)$ if the following holds: take the clips i with $l_i \leq x \leq r_i$ and sort them from top to bottom according to the order; then the first k of them are $s_1, s_2, \dots, s_k$, in this order. In particular, if $k \geq 1$, every clip on screen at frame x that is not among $s_1, \dots, s_k$ must be below clip s_k . A snapshot with $k = 0$ places no constraint on the order.

Input

- The first line contains a single integer n ($1 \leq n \leq 3 \cdot 10^5$), the number of clips.
- The i -th of the next n lines contains two integers l_i and r_i ($0 \leq l_i < r_i \leq 10^9$), the first and the last frame of clip i .
- The next line contains a single integer m ($1 \leq m \leq 3 \cdot 10^5$), the number of snapshots.
- Each of the next m lines describes one snapshot. It starts with two integers x and k ($0 \leq x \leq 10^9$, $0 \leq k \leq n$), the frame of the snapshot and the number of clips it names. They are followed by k integers $s_1, s_2, \dots, s_k$ ($1 \leq s_j \leq n$), the named clips from top to bottom. The clips $s_1, \dots, s_k$ are pairwise distinct, and each of them is on screen at frame x . If $k = 0$, the line contains only x and k .

The snapshots are given in non-decreasing order of x ; several snapshots may be taken at the same frame. The sum of k over all snapshots does not exceed 10^6 .

It is guaranteed that at least one order agrees with all snapshots.

Output

Print n distinct integers $p_1, p_2, \dots, p_n$: the numbers of the clips from the top layer to the bottom layer. If several orders agree with all snapshots, print any of them.

Sample Input	Sample Output
5 2 6 0 4 4 9 6 8 1 10 4 1 1 2 4 1 3 6 1 4 9 2 3 5	4 3 2 5 1
4 0 5 5 10 3 7 20 30 4 0 0 5 1 3 5 2 3 2 12 0	3 2 4 1

Notes

In the first example, clips 2 and 5 are on screen at frame 1, so clip 2 is above clip 5. At frame 4, clips 1, 2, 3 and 5 are on screen (clip 2 ends and clip 3 starts at this frame), and clip 3 is the top one. At frame 6, clips 1, 3, 4 and 5 are on screen (clip 1 ends and clip 4 starts at this frame), and clip 4 is the top one. At frame 9, only clips 3 and 5 are on screen, and the snapshot names both of them. Besides the order in the example output, the orders 4 3 1 2 5 and 4 3 2 1 5 are also accepted.

In the second example, the snapshots at frames 0 and 12 name no clips, and the two snapshots at frame 5 see clips 1, 2 and 3. Together they say that clip 3 is above clip 2, and clip 2 is above clip 1. No snapshot sees clip 4, so it can be anywhere in the order.

Problem J

Phosphorus studying Polygon

Time limit: 6s
Memory limit: 1GB

Phosphorus has drawn n regular polygons on a 2D Cartesian plane. The polygons are numbered from 1 to n .

Each regular polygon i is described by its center coordinates (x_i, y_i) , the distance from the center to its vertices r_i (the circumradius), and the number of sides k_i . To avoid ambiguity with rotation, assume that one of the vertices of the i -th polygon is always located at exactly $(x_i, y_i + r_i)$, and the remaining $k_i - 1$ vertices are spaced evenly along the circumcircle.

We say that two polygons intersect if they share at least one common point (this includes points on their boundaries and within their interiors). Note that a polygon always intersects itself.

It is guaranteed that whenever two polygons are disjoint, their minimum Euclidean distance is at least 10^{-3} . Here, the distance between two polygons is the minimum distance between a point in the first polygon and a point in the second polygon. This guarantee does not exclude polygons that touch: touching polygons intersect and have distance zero.

Phosphorus will give you q queries. Each query consists of a single integer u ($1 \leq u \leq n$). For each query, consider all polygons that intersect the u -th polygon. Among these intersecting polygons, you must find the index j ($1 \leq j \leq n$) of the polygon that has the **maximum total area**.

Hint: The area of a regular polygon with k sides and circumradius r can be calculated using the formula $\text{Area} = \frac{1}{2}kr^2 \sin\left(\frac{2\pi}{k}\right)$. It is guaranteed that whenever two polygons have different pairs (k_i, r_i) , the larger of their two areas is at least $(1 + 10^{-12})$ times the smaller one, so evaluating this formula with standard 64-bit floating-point types (e.g., `double` in C++ and Java, or `float` in Python) is precise enough to compare areas.

Input

- The first line contains two integers n and q ($1 \leq n, q \leq 3 \times 10^5$) — the number of polygons and the number of queries.
- Each of the next n lines contains four integers x_i, y_i, r_i , and k_i ($-10^6 \leq x_i, y_i \leq 10^6$, $1 \leq r_i \leq 10^6$, $3 \leq k_i \leq 7$) — the coordinates of the center, the circumradius, and the number of sides of the i -th polygon.
- Each of the next q lines contains a single integer u ($1 \leq u \leq n$) — the index of the polygon for the current query.

Output

Print q lines. For the m -th query, output a single integer j — the index of the polygon with the maximum total area that intersects the polygon u . If there are multiple intersecting polygons that share the same maximum area, output the one with the **minimum index**.

Sample Input	Sample Output
4 4	1
0 0 2 3	3
2 0 2 3	3
6 0 3 3	4
0 5 2 3	
1	
2	
3	
4	

Notes

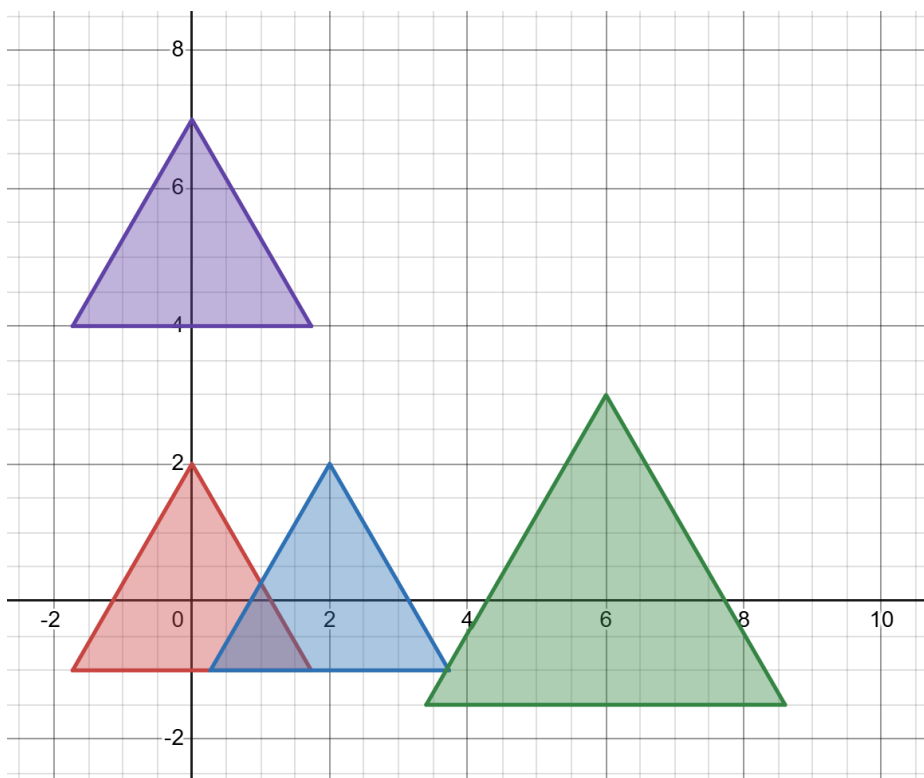


Figure 1: The polygon formation in the example.

Problem K

Locker Riddle II

Time limit: 1s
Memory limit: 1GB

Locker Riddle I

A room contains n lockers, numbered from 1 to n . Each locker is either open or closed. A group of n heirs must solve a puzzle with these lockers before they can receive their inheritance.

The heirs are also numbered from 1 to n . In each step, you may choose one heir to perform an operation. When the i -th heir is chosen, they flip the state of every locker whose index is a multiple of i . More precisely, for every positive integer j such that $i \times j \leq n$, the state of the $(i \times j)$ -th locker is changed from open to closed or vice versa. An heir may be chosen more than once.

The puzzle is solved when all n lockers are closed.

Find a sequence of operations that closes all the lockers, or determine that no such sequence exists.

This problem is a randomized variant of the puzzle above.

The setting and rules are the same as in **Locker Riddle I**, except that you can no longer choose which heir performs each operation. At each step, exactly one heir is chosen at random. The i -th heir is chosen with probability r_i . The choices made in different steps are independent. The process stops as soon as all lockers are closed; if they are initially all closed, no operations are performed.

Find the expected number of operations performed, including the one after which all lockers are closed.

Input

- The first line contains a single integer t ($1 \leq t \leq 100$), the number of test cases.
- For each test case:
 - The first line contains two integers n and S ($1 \leq n \leq 100$, $n \leq S \leq 100$), the number of lockers and the sum of all probability weights.
 - The second line contains n integers $a_1, a_2, \dots, a_n$ ($0 \leq a_i \leq 1$), the initial states of the lockers, where $a_i = 1$ means the i -th locker is open and $a_i = 0$ means it is closed.
 - The third line contains n integers $p_1, p_2, \dots, p_n$ ($1 \leq p_i \leq S$, $\sum_{i=1}^n p_i = S$), where the i -th heir is chosen with probability $r_i = \frac{p_i}{S}$ at each step.

Output

For each test case, print the expected number of operations modulo 998244353, as defined below.

It can be proved that the expected number of operations is always a rational number. Also, under the constraints of this problem, it can be proved that if this rational number is represented as an irreducible

fraction $\frac{P}{Q}$, then $Q \not\equiv 0 \pmod{998244353}$. Therefore, there uniquely exists an integer R such that $R \times Q \equiv P \pmod{998244353}$ and $0 \leq R < 998244353$. Print this R .

Sample Input	Sample Output
8 3 3 0 0 0 1 1 1 3 3 0 0 1 1 1 1 3 3 0 1 0 1 1 1 3 3 0 1 1 1 1 1 3 3 1 0 0 1 1 1 3 3 1 0 1 1 1 1 3 3 1 1 0 1 1 1 3 3 1 1 1 1 1 1	0 7 7 9 10 9 9 7
3 7 67 1 0 0 0 0 1 1 6 7 6 7 6 7 28 11 12 1 1 1 1 1 1 0 1 0 1 0 2 1 1 1 1 1 1 1 1 1 15 15 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	482862219 84701850 32767

Notes

In the first test case of the first example, all lockers are already closed, so the process stops immediately and the answer is 0.

In the second test case of the first example, only locker 3 is open. If heir 3 is chosen first, which happens with probability $\frac{1}{3}$, the puzzle is solved in one operation. Other choices open additional lockers and require further operations. Taking all possible random sequences into account, the expected number of operations is 7.

In the first test case of the second example, the expected number of operations is $\frac{48413814955469214029}{259441760155978400}$.

Problem L

Grief

Time limit: 1s
Memory limit: 256MB

Congrats Doggie!

— Demen

Doggie works with liquid mixtures of **Mercury (Hg)** and **Tacos (Ts)**. A mixture represented by (x, y) contains x grams of Hg and y grams of Ts.

Doggie has n available mixtures. He may also buy any finite number of new mixtures. Buying a mixture containing x grams of Hg and y grams of Ts costs $ax + by$, where x and y may be any nonnegative real numbers. In what follows, mixture i (existing or purchased) contains x_i grams of Hg and y_i grams of Ts.

To produce a mixture, Doggie takes an independent fraction $p_i \in [0, 1]$ of each mixture he owns, existing and purchased alike. The fractions need not sum to 1. The result contains exactly $\sum_i p_i x_i$ grams of Hg and $\sum_i p_i y_i$ grams of Ts.

Doggie is considering m possible target mixtures for his experiment, but he has not decided which one to produce. During the later experiment phase, the shop will have no mixtures left, so he cannot buy any more. He wants to buy enough now to guarantee that he can produce whichever target he chooses later. He will produce only one target, and he may choose the fractions after deciding which target to produce. The result must contain **exactly** the target's amounts of Hg and Ts.

Help Doggie choose his purchases so that every possible target can be produced, while minimizing the total cost of the mixtures he buys.

Input

- The first line contains four integers $n, m, a,$ and b ($1 \leq n, m \leq 436879$; $1 \leq a, b \leq 396$).
- Each of the next n lines contains two integers x_i and y_i ($0 \leq x_i, y_i \leq 436879$), the grams of Hg and Ts in an existing mixture.
- Each of the next m lines contains two integers X_j and Y_j ($0 \leq X_j, Y_j \leq 436879$), the grams of Hg and Ts in a target mixture.

Output

Print one real number: the minimum total cost of buying the new mixtures.

Your answer will be accepted if its absolute or relative error does not exceed 10^{-6} . Formally, let your output be u and the jury's answer be v ; your output is accepted if and only if $\frac{|u-v|}{\max(1, |v|)} \leq 10^{-6}$.

Sample Input	Sample Output
5 2 1 1 1 2 3 4 5 6 7 8 9 10 80 80 100 100	145.000000000000
3 3 3 1 6 4 0 0 0 2 2 1 1 3 4 5	1.833333333333

Notes

Together, the five available mixtures contain 25 grams of Hg and 30 grams of Ts. If Doggie later chooses the target containing 100 grams of each ingredient, he will need at least 75 more grams of Hg and 70 more grams of Ts. Thus, he must spend at least 145.

Doggie can buy one mixture containing exactly 75 grams of Hg and 70 grams of Ts. If he chooses the target containing 100 grams of each ingredient, he uses all six mixtures. If he chooses the target containing 80 grams of each ingredient, he uses 80% of each mixture. Therefore, the answer is 145.