The 2025 ICPC Caribbean Finals Qualifier

Real Contest Problem Set

Problem set developers

Carlos Joa Fong – SODOCOMP
Marcelo Fornet Fornés – NEAR AI
Rubén Alcolea Núñez – Leil Storage
Tomas Orlando Junco Vázquez – T-Systems Iberia
Anier Velasco Sotomayor – Harbour Space University
José Andrés González Barrameda – Universidad de La Habana
Jorge Alejandro Pichardo Cabrera – Universidad de La Habana
Humberto Díaz Suárez – Universidad de Puerto Rico, Mayagüez

Matcom Online Grader

Leandro Castillo Valdés – ICPC Caribbean
Frank Rodríguez Siret – Harbour Space University
Karel Antonio Gonzalez Zaldivar – Universidad de la Habana

October 4th, 2025

Problem A. Altered States

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Initially, this story was about Axel defeating Demogorgons, the Mind Flayer, and Vecna with his telekinetic superpowers. But things took a strange turn: Axel ended up teaming with them, forming the notorious *UpsideDown ICPC Team*.

Now, their mission is no longer just survival — it's winning the ICPC Indiana Qualifier against the formidable Hawkins Team. And to do that, they need your help to solve a peculiar challenge involving strings:

You are given several strings $w_1, w_2, \dots, w_n$. Your task is to find the number of distinct substrings present in all of these strings combined. Formally, if $S(w_i)$ is the set of all distinct substrings of string w_i, compute $|S(w_1) \cup S(w_2) \cup \cdots \cup S(w_n)|$.

Input

The first line contains an integer n ($1 \leq n \leq 5.5 \cdot 10^5$) — the number of strings.

Each of the next n lines contains a string consisting of lowercase English letters.

It is guaranteed that the sum of the lengths of all the strings does not exceed $5.5 \cdot 10^5$.

Output

Output a single integer — the total number of distinct substrings across all strings.

Examples

standard input	standard output
1 abacaba	21
2 abba bcab	14
5 abvsvdh ahusywbag inzuqwbd iajhhdt qweqweqwe	142

Note

In the second example, the first string has substrings

$S(w_1) =$ {“a”, “b”, “ab”, “ba”, “bb”, “abb”, “bba”, “abba”}

and the second one has substrings

$S(w_2) =$ {“b”, “c”, “a”, “bc”, “ca, “ab”, “bca”, “cab”, “bcab”}

The cardinality of the union of these two sets is 14.

Problem B. Breaking Bad

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Axel White and Lexa Pinkman are cooking some crystal in their lab... I mean, they are making high-quality and refined crystal panels for decorative purposes through a process of heating and hitting.

Axel, the artist of the duo, has a very specific vision of what constitutes a “beautiful crystal panel”. For him, beauty lies in the process. A panel is crafted using a delicate heating process that starts at the top-left corner, coordinate $(1, 1)$. During this process, the heating head can only move down (from (i, j) to $(i + 1, j)$) or right (from (i, j) to $(i, j + 1)$). For a panel to be considered “beautiful”, there must exist at least one valid “heating path” that, in addition to following the movement rules, passes through all the essential decorative cells that Axel has designed.

Lexa, on the other hand, is more pragmatic and sometimes a bit chaotic. Her goal is to sabotage the production with the minimum possible effort. She can hit the panel in certain areas to create fractures, making it impossible for the heating head to pass through them. Her task is to find the cheapest way to create a set of fractures such that no valid “heating path” can be completed, thus rendering the panel “not beautiful”.

Panel Specification:

The crystal panel is represented as an $N \times N$ grid. Each cell (i, j) on the panel has a type and a cost (effort, resources, etc.) associated with fracturing it.

There are 5 types of cells on the panel:

- Type 0 (Impurity): Zones with factory defects. The heating head cannot pass through here. They are already “fractured”, and no interaction is possible.
- Type 1 (Standard Crystal): Normal zones of the panel. Lexa can hit them to create a fracture. The cost of fracturing the cell (i, j) is the cost specified for that cell.
- Type 2 (Decorative Cell): Key points in Axel’s design. For the panel to be “beautiful”, the heating path must pass through all of these cells. Lexa can also fracture these cells by paying their corresponding cost. If a single decorative cell is fractured, it becomes impassable, and therefore no valid path can exist.
- Type 3 (Reinforced Cell): Zones of tempered crystal. The heating head can pass through them, but Lexa cannot fracture them. They are impervious to her strikes.
- Type 4 (Reinforced Decorative Cell): The centerpiece of the design. These are decorative cells that are also reinforced. The heating path must pass through them, and Lexa cannot fracture them.

Your task is to help Lexa. You must find a set of type 1 or type 2 cells to fracture such that no valid heating path exists starting from $(1, 1)$. The goal is for the total cost of the fractures (the sum of the costs of the selected cells) to be the minimum possible.

Input

The first line contains the integer N ($2 \le N \le 60$).

The next N lines contain N integers each. The j-th integer of the i-th line is t_{ij} ($0 \leq t_{ij} \leq 4$) – the type of cell (i,j).

Then we have another N lines with N integers each. The j-th integer of the i-th line is c_{ij} ($1 \leq c_{ij} < 10^9$) – the cost of fracturing cell (i, j). If this cell is type 0, 3, or 4, the cost will be -1 since they cannot be fractured.

We guarantee that cell $(1, 1)$ is always of type 4.

Output

On the first line print two integers: the cost W for making the panel "not beautiful" and an integer K (the number of cells selected for the fracturing).

Then print K lines with two integers i, j each, the selected cells.

All selected cells must be of type 1 or 2, and the sum of the fracturing costs of the cells must be W. If no fracture is necessary because there is no valid heating path, W must be 0 and K must also be 0 (see example 2).

If there is no valid way of making the panel "not beautiful", print -1 -1.

Examples

standard input	standard output

3
4 1 3
1 0 1
1 3 2
-1 3 -1
2 -1 4
1 -1 6
	4 2
	1 2
	3 1

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

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

Note

In example 1 note that a valid path needs to go from (1,1) to (3,3). Fracturing (3,1) and (2,3) achieves Lexa's goal with minimum cost.

In example 2 there is no path that can go through all the decorative cells, so Lexa doesn't need to waste her time.

In example 3 given that (1,1) is the only decorative cell the path can start and end in the same cell. But (1,1) is a reinforced cell so Lexa can't break this path.

Problem C. Circle vs Square

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	1024 megabytes

You are given integer points sampled uniformly from the area of a hidden shape. It is known that the shape can be either a circle or a square.

The orientation, size, or position of the shape is also hidden.

The points are sampled uniformly and independently from each other, among all integer points inside this shape.

It is guaranteed that the area of this shape is at least 10^6, and the whole shape lies inside the axis aligned square between -10^9 and 10^9 in both axes.

Given the points, you should determine if the shape used to sample them was a circle or a square.

Input

The first line contains one integer ($1 \le t \le 100$), the number of test cases. Each test case starts with a line ($10^3 \le n \le 10^5$), the number of points sampled from the random variable. The next n lines contain two integers ($-10^9 \le x, y \le 10^9$), the points.

The sum of n across all test cases does not exceed 10^5.

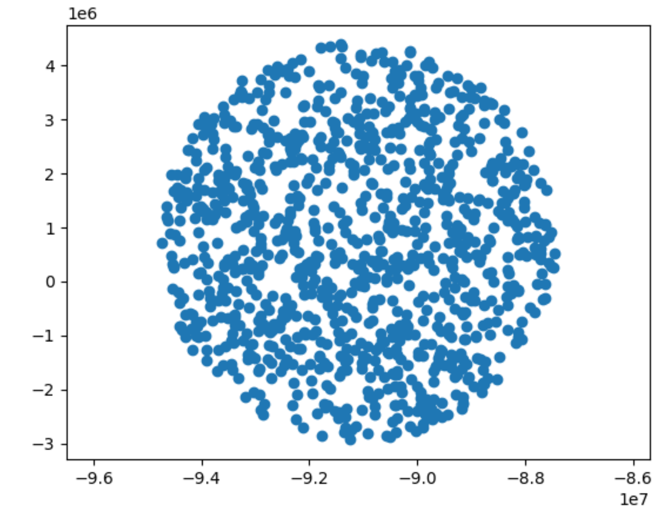
Output

For each test case, print one word, either "circle" or "square".

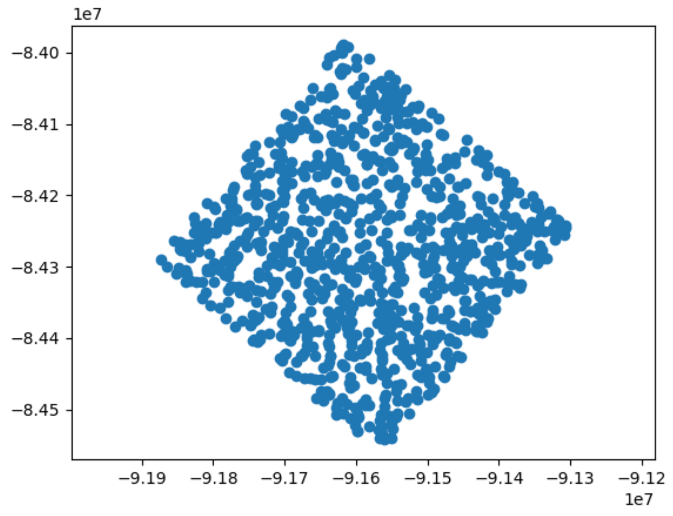
Example

standard input	standard output
2 1000 -89138442 -2570 -89872005 1798962 -89112850 701427 -91711055 3151678 -89831652 -527468 -94175065 -605963 -90001360 -1936799 -90510049 -2869287 -92455184 3495836 -92170488 4106603 -91680250 -8800 -92954091 -918463 -92892836 -641456 -94514914 276930 -90870492 -1775414 -91071548 -74689 -89425956 -862564 -87564256 344237 -88350939 -280404 -88321750 447291 --- Showing first 20 lines ---	circle square

Note



Plot of test 1:



Plot of test 2:

Problem D. Doctoral Thesis

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 megabytes

Professor Jun, at a prestigious Caribbean university, is currently pursuing his doctoral research in secure and reliable data transmission. Although simple at first glance, the **anagrams** are among the core elements in which he has placed great confidence to advance his work.

Two strings X and Y are anagrams if and only if X can be obtained by rearranging the characters of Y; for example, $X = aaabbbccca$ and $Y = abcabcabac$ are anagrams. In particular, a string X is an anagram of itself.

Accustomed to the traditional academic hierarchy followed in universities during research, Professor Jun has proposed several topics suitable for bachelor's and master's theses. He would like to share these topics with ICPC contestants to identify potential students with the right skills to collaborate on the very complex topics he is involved in. Those who are selected will, of course, benefit from working closely with him.

Taking advantage of the *The 2025 ICPC Caribbean Finals (Qualifier)*, Professor Jun has rewritten one of the subproblems from his doctoral project as a typical ICPC problem. Can you prove your skills by solving it here, in this contest? The description is as follows:

You are given a string **s**. With the simple goal of learning a little about that string, you must process several queries of two types:

- **INSIDE**: Given two indices l and r, and a non-empty string $\boldsymbol{w}$, determine whether $\boldsymbol{w}$ is an anagram of the substring $\boldsymbol{s[l..r]}$.
- **COUNT**: Given two indices l and r, compute the number of distinct anagrams of the substring $\boldsymbol{s[l..r]}$.

Here $\boldsymbol{s[l..r]}$ denotes the substring of s starting at index l and ending at index r, inclusive. Indices l and r are 1-based.

Input

The first line contains an integer t ($1 \le t \le 10^4$), the number of test cases. Each test case is described as follows:

- The first line contains a string s of length n ($1 \le n \le 10^6$), consisting of lowercase letters of the English alphabet.
- The second line contains an integer q ($1 \le q \le 10^5$) representing the number of queries.
- The next q lines describe the queries in one of the following formats:
  - INSIDE l r w — where $1 \le l \le r \le n$, and w is a non-empty string of lowercase English letters.
  - COUNT l r — where $1 \le l \le r \le n$.

It is guaranteed that:

- The total length of all strings s over all test cases does not exceed 10^6.
- The total length of all strings w in INSIDE queries does not exceed 10^6.
- The total number of queries across all test cases does not exceed 10^5.

Output

For each test case, process the queries in order:

- For each INSIDE query, print "YES" if w is an anagram of $s[l..r]$, and "NO" otherwise.
- For each COUNT query, print the number of distinct anagrams of $s[l..r]$ modulo 998244353.

Examples

standard input	standard output
4 iktljylbfiufyfcaancewszq 3 INSIDE 11 14 uyff COUNT 7 18 COUNT 17 21 corklxhpqiooexbjuimbqvla 2 INSIDE 1 3 ocr COUNT 9 10 dcaqusosnkwdqpklhflemxeq 2 INSIDE 5 10 wlxaet INSIDE 10 13 dqkw pavfjnmlgsdciimwdlxvtuejsdth 3 COUNT 16 28 INSIDE 11 13 idc COUNT 6 6	YES 39916800 120 YES 2 NO YES 558510847 YES 1
4 xsiqusxuqbytnwahjmazia 2 COUNT 15 22 INSIDE 7 13 btquynx nttcyymqmauhekmsyevkixyzqgen 3 COUNT 1 12 INSIDE 14 18 ykmse COUNT 18 23 yooaxejnjjiaflzefgck 3 INSIDE 1 1 u INSIDE 11 13 wox INSIDE 6 6 u arunbkdrjzmrimtsbjatyihsjccptl 2 COUNT 9 14 INSIDE 22 24 hsi	6720 YES 59875200 YES 720 NO NO NO 360 YES

Problem E. Exact Steps

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

Peter is teaching his little sister Roxs the amazing world of algorithms. Last week, Peter sat down with Roxs and said:

I will teach you something cool: binary search. Imagine you have a sorted list of n numbers, and you want find the position of a target number. Instead of checking each number one by one, here's what you do:

1. Look at the number right in the middle.
2. If it's the one you're searching for, you're done.
3. If your number is smaller, keep searching in the left half.
4. If it's larger, search in the right half.
5. Repeat until you find it or there are no numbers to check.

The number of times you check a middle element is the number of steps it takes to find the target number.

Roxs's eyes lit up. She understood the principle: binary search always cuts the problem in half.

Then Peter smiled:

Good job, Roxs! Now here's a challenge for you:

Given the sorted list of n numbers, which positions could hold a number that is found in exactly s steps?

To make sure everything is clear, Peter showed Roxs the exact algorithm she can use:

```
function BinarySearch(a[0..n−1], n, x):
    lo := 0
    hi := n − 1
    steps := 0
    while lo <= hi:
        mid := floor((lo + hi) / 2)
        steps := steps + 1
        if a[mid] = x:
            return steps
        else if a[mid] < x:          # search in the right half
            lo := mid + 1
        else:                        # search in the left half
            hi := mid − 1
    return −1  # target not found
```

Peter is only interested in the list of valid positions. A position i ($0 \le i < n$) is considered **valid** when the element at position i can be found in exactly s steps.

Can you help Roxs to solve the challenge and determine all valid positions?

Input

The first line contains an integer t ($1 \le t \le 100$), the number of test cases.

Each of the next t lines contains two integers n and s ($1 \le n \le 10^5$, $1 \le s \le \lceil \log_2(n+1) \rceil$).

It is guaranteed that $\sum n \le 2 \cdot 10^5$ over all test cases.

Output

For each test case, output a single line with the following:

- Print all valid positions, in increasing order and separated by a single space.
- It's guaranteed that a valid solution will always exist.

Example

standard input	standard output

2
7 3
5 1
	0 2 4 6
	2

Problem F. Final Shutdown

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

Your task is to shut down a series of n power generators, each located at a distinct position along a one-dimensional line. Starting from a specific generator, carefully determine the sequence of shutdowns and movements between generators to minimize the total energy consumed.

Energy Consumption

Each generator i has a base energy consumption rate of e_i, but its energy rate changes over time. Let $E(i, t)$ be the rate of generator i during the time interval $[t, t + 1)$.

- The rate starts at its base value: $E(i, 0) = e_i$.
- The rate increases by **1 unit per second**.
- If a generator's rate reaches 1000, it **resets** to its base rate e_i in the next second. Formally, if $E(i, t) = 1000$, then $E(i, t + 1) = e_i$.

Shutdown Process

1. You start at time $t = 0$ at a given generator s.
2. The starting generator s is shut down **instantly**.
3. You then repeatedly travel to an adjacent active generator (to the immediate left or right of the contiguous block of shut-down generators).
4. Once a generator is shut down, it becomes part of a **teleportation network**. As a result, you may move instantly to any previously shut-down generator without incurring **travel time** or extra **energy cost**.
5. Travel time equals the **distance** between generators. The distance between generators i and j is $|x_i - x_j|$ for all $1 \leq i, j \leq n$.
6. When you arrive, the generator is shut down **instantly**.
7. During travel, all generators that are still active continue to consume energy.

Your goal is to find a shutdown sequence that **minimizes the total energy consumed**.

Input

The first line contains an integer t ($1 \leq t \leq 10$), the number of test cases.

For each test case:

- The first line contains two integers, n and s ($1 \leq n \leq 1000$, $1 \leq s \leq n$), the number of generators and the index of the starting generator.
- The next n lines each contain two integers, x_i and e_i ($1 \leq x_i, e_i \leq 1000$), representing the position and base energy consumption of the i-th generator.
- It is guaranteed that the generators' positions are strictly increasing, i.e., $x_1 < x_2 < \cdots < x_n$.

Output

For each test case, print a single integer representing the minimum total energy consumed when shutting down all generators in an optimal order.

Example

standard input	standard output

2
3 2
1 4
6 5
9 7
3 2
1 2
5 10
9 2
	84
	58

Note

In the first case, there are three generators: generator 1 ($x_1 = 1$ and $e_1 = 4$), generator 2 ($x_2 = 6$ and $e_2 = 5$), and generator 3 ($x_3 = 9$ and $e_3 = 7$).

The optimal order to shutdown the generators is: $2 \to 3 \to 1$ (turn off generator 2 at $t = 0$, then generator 3 at $t = 3$, teleport to generator 2 instantly ($t = 3$), and finally turn off generator 1 at $t = 8$).

Move 1: from $x_2 = 6$ to $x_3 = 9$ ($\Delta = 3$ seconds), starts at $t_0 = 0$.

Active during this move: generator 1 ($E = 4$) and generator 3 ($E = 7$), while generator 2 is already off.

Per-second table for $t = 0, 1, 2$:

second t	gen1 rate $E_1 + t$	gen2 (off)	gen3 rate $E_3 + t$	sum
0	$4 + 0 = 4$	—	$7 + 0 = 7$	11
1	$4 + 1 = 5$	—	$7 + 1 = 8$	13
2	$4 + 2 = 6$	—	$7 + 2 = 9$	15
Move 1 total				**39**

Generator 3 is turned off at $t = 3$.

Move 2: teleport from $x_3 = 9$ to $x_2 = 6$ as they are part of the teleportation network, t remains 3.

Move 3: from $x_2 = 6$ to $x_1 = 1$ ($\Delta = 5$ seconds), starts at $t_0 = 3$.

Active during this move: only generator 1 ($E = 4$) because generator 3 was turned off at $t = 3$.

Per-second table for $t = 3, 4, 5, 6, 7$:

second t	gen1 rate $E_1 + t$	gen2 (off)	gen3 (off)	sum
3	$4 + 3 = 7$	—	—	7
4	$4 + 4 = 8$	—	—	8
5	$4 + 5 = 9$	—	—	9
6	$4 + 6 = 10$	—	—	10
7	$4 + 7 = 11$	—	—	11
Move 3 total				**45**

Generator 1 is turned off at $t = 8$. At this point, all generators are already turned off.

Therefore, the total energy consumed during the shutdown process is $39 + 45 = 84$.

Problem G. Gotcha Station

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Gabriel is taking the subway to visit Gloria, unaware that she is planning a surprise. She wants to intercept him along his route. The subway consists of a network of stations connected to each other by two-way train lines. Gloria knows his departure and destination stations, and that he will only travel via subway, moving from station to station along the lines.

However, Gloria does not know which exact route Gabriel will take. He might choose the shortest path, or he might be forced to take a detour due to service disruptions. Regardless, Gloria believes there are some stations he must pass through on any route. She could wait at one of these stations.

Your task is to write a program that, given a description of the subway network and the start and end stations, determines the stations that Gabriel must pass through on any route from start to end.

Input

The first line contains an integer t ($1 \le t \le 10^5$), the number of test cases.

Each test case begins with a line containing four integers n, m, s, e ($2 \le n \le 10^5$, $1 \le m \le 2 \cdot 10^5$, $1 \le s, e \le n$, $s \ne e$), denoting the number of stations, the number of connections between stations, the starting station, and the ending station, respectively. Stations are numbered from 1 to n.

Then follow m lines, each containing two integers u and v ($1 \le u < v \le n$), representing a two-way train line connecting stations u and v. Connections are distinct.

It is guaranteed that:

- The subway network is connected — every station can be reached from every other station.
- The sum of n over all test cases does not exceed 10^5.
- The sum of m over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output an integer k, the number of stations that Gabriel must pass through. Then output the k station numbers, one per line, in ascending order.

Example

standard input	standard output

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

Problem H. Hyper Disjoint Set

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

You are asked to maintain a dynamic undirected graph under a sequence of operations.

Initially, the graph consists of a single node (labeled 1) and no edges. Then, you must process a sequence of operations of the following types:

- **Add edge**
  `LINK u v`
  Add an edge between nodes u and v. It is guaranteed that both nodes exist at this moment.
- **Duplicate graph**
  `DUP`
  If the current graph contains n nodes, extend it to contain $2n$ nodes as follows:
  - For every edge (u, v) with $1 \le u, v \le n$ in the current graph, add edges (u, v) and $(u+n, v+n)$ in the new graph.
  - No additional edges are added between the two halves.
- **Duplicate and link graph**
  `DUPLINK`
  If the current graph contains n nodes, extend it to contain $2n$ nodes as follows:
  - For every edge (u, v) with $1 \le u, v \le n$ in the current graph, add edges (u, v) and $(u+n, v+n)$ in the new graph.
  - Additionally, for every $1 \le u \le n$, add an edge $(u, u + n)$ between the original node and its copy.
- **Connectivity query**
  `ASK u v`
  Determine whether nodes u and v are in the same connected component.

Input

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

For each test case:

- The first line contains an integer $(1 \le q \le 10^5)$ — the number of operations.
- Each of the next q lines describes an operation in one of the following formats:
  - `LINK u v` $(1 \le u, v \le 10^{18})$
  - `DUP`
  - `DUPLINK`
  - `ASK u v` $(1 \le u, v \le 10^{18})$

Notes:

- The sum of q across all test cases is at most 10^5.
- All node labels used in the operations are guaranteed to be valid at the moment they are referenced.

Output

For each test case, output the answers to all operations of type `ASK`, in the order they appear. Each answer should be printed as `YES` if the two nodes belong to the same connected component, and `NO` otherwise.

Example

standard input	standard output
2 9 DUP DUP LINK 1 2 LINK 1 3 ASK 1 3 ASK 2 3 ASK 1 4 LINK 3 4 ASK 1 4 11 DUPLINK DUP LINK 2 3 ASK 1 3 DUP ASK 8 7 ASK 2 8 LINK 4 6 ASK 2 8 LINK 6 6 ASK 5 5	YES YES NO YES YES YES NO YES YES

Problem I. Inverse Harmony

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

You are given a sequence of positive integers $a_1, a_2, \ldots, a_n$, and an integer k. The operation described above must be performed **exactly k times**:

- Choose an index i ($1 \le i \le n$) and increase a_i by 1 (i.e., set $a_i = a_i + 1$).

It is allowed to choose the same index i more than once. After performing the operation exactly k times, let H denote the harmonic mean of the sequence:

$$H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n}}$$

Your task is to maximize H by distributing the k operations optimally, and then compute its value.

Print the inverse of H with respect to the prime modulus 998244353.

$$\frac{1}{H} \pmod{998244353}$$

Input

The first line contains a single integer t ($1 \le t \le 10^5$) — the number of test cases.

Each test case begins with a line containing two integers n and k ($1 \le n \le 10^5,\ 0 \le k \le 10^8$) — the length of the array and the number of operations.

The next line contains n positive integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^8$).

The sum of n across all test cases does not exceed 10^5.

Output

For each test case, print a single integer — the value of $\frac{1}{H}$ modulo 998244353, where H is the maximum possible harmonic mean after applying exactly k operations.

Example

standard input	standard output

4
2 10
2 5
1 7
6
3 9
9 6 2
8 5
9 9 8 1 8 5 9 4
	658563983
	460728163
	254182590
	983686623

Problem J. Judicious XORcery

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Axel the XORcerer is brewing a potion to become wiser. The cauldron he uses is divided into n contiguous sections, each with a magic value $a_1, a_2, \ldots, a_n$. He needs to stir the potion and add ingredients to specific sections, and he also wants to be able to check the magic value of any section so he can adjust his brew on the fly.

He has a mixing rod, which starts in the first section. The rod can only move to a section immediately adjacent to its current position, either to the left or to the right, if such a section exists. When Axel moves it, the magic values change automatically depending of the direction:

- Single step to the **right**: The rod is in section i $(1 \le i \le n-1)$ and Axel moves it to section $i+1$, then $a_{i+1} := a_{i+1} \oplus a_i$
- Single step to the **left**: The rod is in section i $(2 \le i \le n)$ and Axel moves it to section $i-1$, then $a_i := a_i \oplus a_{i-1}$

Moving the rod k times in any direction means applying the corresponding single step (to the right or to the left) sequentially k times. For example, let the initial magic values $a = [\mathbf{2}, 1, 5, 4]$ (rod in section 1). If Axel has to move the rod 3 times to the right:

- Move 1: from section 1 to section 2
  - Apply $a_2 := a_2 \oplus a_1 \to a_2 = 1 \oplus 2 = 3$
  - Magic values become $[2, \mathbf{3}, 5, 4]$ (rod in section 2)
- Move 2: from section 2 to section 3
  - Apply $a_3 := a_3 \oplus a_2 \to a_3 = 5 \oplus 3 = 6$
  - Magic values become $[2, 3, \mathbf{6}, 4]$ (rod in section 3)
- Move 3: from section 3 to section 4
  - Apply $a_4 := a_4 \oplus a_3 \to a_4 = 4 \oplus 6 = 2$
  - Magic values become $[2, 3, 6, \mathbf{2}]$ (rod in section 4)

The Axel assistant (in this case, you) will do all the hard work. Axel may give you three types of instructions:

- `1 k` — Move the mixing rod k times to the right if $k > 0$, or $|k|$ times to the left if $k < 0$. Axel ensures $k \neq 0$ and the rod will stay inside the cauldron (it won't go outside the limits 1 to n).
- `2 p x` — Change the magic value of section p to x, i.e., $a_p := x$.
- `3 p` — Axel will ask you to report the current magic value of section p.

Input

The first line contains two integers n, q $(2 \le n \le 2 \cdot 10^5, 1 \le q \le 2 \cdot 10^5)$ — the number of sections and the number of operations.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(0 \le a_i < 2^{30})$ — the initial magic values.

The next q lines each describe one operation in one of the three formats above. For type 2 operations, it is guaranteed that $0 \le x < 2^{30}$.

It is guaranteed that there is at least one operation of type 3.

Output

For each operation of type 3, output the current magic value of the required section on a separate line.

Examples

standard input	standard output

4 4
2 1 5 4
1 3
3 4
1 -2
3 3
	2
	5

7 10
3 1 2 10 8 8 7
1 5
2 6 5
3 7
1 -2
3 6
2 3 4
3 7
3 4
1 -1
3 4
	7
	7
	7
	10
	14

Problem K. Kekkaishi

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

In the world of Kekkaishi, the duty of protecting the sacred Karasumori site is entrusted to sorcerers who wield barrier magic. A kekkaishi can create barriers (kekkai) to trap spirits that threaten the land. These barriers manifest as axis-aligned rectangular boxes.

However, the spirits are cunning. They emerge at scattered points and flee to evade capture. To contain them decisively, a kekkaishi may form a single barrier large enough to enclose all of them at once. The amount of magic spent on a barrier is proportional to its surface area. Therefore, a smaller barrier is better.

Your role is to assist the kekkaishi in training. You will receive an ordered sequence of operations that add or remove spirits, represented as points. After each operation, you must calculate the surface area of the smallest barrier that can contain all of the remaining spirits.

Input

The first line contains an integer t ($1 \le t \le 10^5$), the number of test cases.

Each test case begins with an integer n ($1 \le n \le 10^5$), the number of operations.

The next n lines describe the operations, each in one of the following formats:

- `ADD x y z` — Add a spirit at (x, y, z), where $-10^8 \le x, y, z \le 10^8$.
- `REMOVE k` — Remove the k-th spirit that was added. Spirits are numbered starting from 1 in order of appearance.

It is guaranteed that:

- Every index removed corresponds to a spirit that currently exists (not yet removed).
- Each spirit is removed at most once.
- The sum of n across all test cases does not exceed 10^5.

Output

For each test case, print n lines.

On the i-th line, output a single integer: the surface area of the smallest barrier that contains the spirits after the i-th operation.

Examples

standard input	standard output
2	0
6	70
ADD 1 -5 6	170
ADD 6 0 5	112
ADD 2 -4 0	0
REMOVE 1	0
REMOVE 3	0
REMOVE 2	544
4	0
ADD -4 -4 6	0
ADD 4 2 -10	
REMOVE 2	
REMOVE 1	
3	0
8	700
ADD -4 2 4	0
ADD 3 14 18	382
REMOVE 2	1158
ADD 5 -5 -4	1018
ADD 3 -9 -20	0
REMOVE 3	0
REMOVE 4	0
REMOVE 1	1468
6	0
ADD -15 5 -6	288
ADD 11 -16 -2	0
REMOVE 1	0
ADD 11 -8 -20	0
REMOVE 2	32
REMOVE 3	0
6	3040
ADD 9 17 -11	0
ADD 6 19 -9	0
REMOVE 2	
ADD -11 -11 9	
REMOVE 3	
REMOVE 1	

Problem L. Lost Score

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

There are three problems and three kids. The problems are worth 1, 2, and 4 points, respectively. Each problem is solved by exactly 2 kids, earning them points. Given the scores of two kids, find the score of the third kid.

Input

The first line is an integer ($1 \le t \le 10^5$), the number of test cases. Each test case contains a line with two integers ($1 \le a, b \le 7$), which are the scores obtained by two kids.

Output

For each test case, print a single integer in a line: the score of the third kid.

Example

standard input	standard output
1 6 3	5

Problem M. Matrix Operations

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

You are given a matrix A with n rows and m columns. The entry in the i-th row and j-th column is denoted by $A_{i,j}$, where $1 \leq i \leq n$ and $1 \leq j \leq m$.

You must process q queries of three different types:

- type 1: given a row index r, reverse the r-th row. This operation swaps $A_{r,j}$ with $A_{r,m-j+1}$ for all $1 \leq j \leq \lfloor m/2 \rfloor$.
- type 2: given a column index c, reverse the c-th column. This operation swaps $A_{i,c}$ with $A_{n-i+1,c}$ for all $1 \leq i \leq \lfloor n/2 \rfloor$.
- type 3: given a row and a column indices r, c, print the current value of the element $A_{r,c}$.

Input

The first line contains three integers n, m, and q ($1 \leq n, m \leq 100; 1 \leq q \leq 1000$) — the number of rows, the number of columns, and the number of queries, respectively.

Each of the next n lines contains m integers, where the j-th integer of the i-th line is the element $A_{i,j}$ ($0 \leq A_{i,j} \leq 1000$).

The next q lines describe the queries. Each line begins with an integer t, the type of the query ($1 \leq t \leq 3$).

- If $t = 1$, it is followed by one integer r ($1 \leq r \leq n$).
- If $t = 2$, it is followed by one integer c ($1 \leq c \leq m$).
- If $t = 3$, it is followed by two integers r and c ($1 \leq r \leq n, 1 \leq c \leq m$).

Output

For each query of type 3, print the value of the element at the given row and column on a new line.

Example

standard input	standard output

```
3 4 5
1 2 3 4
5 6 7 8
9 10 11 12
3 2 2
1 2
3 2 2
2 4
3 2 4
```
	```
	6
	7
	5
	```