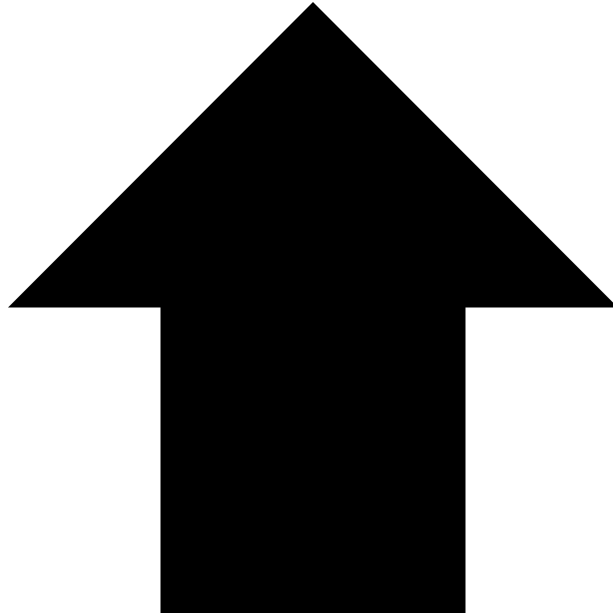


Contest Problems
Philadelphia Classic, Fall 2022
University of Pennsylvania



PClassic 2022 Fall

Rules and Information

This document includes 12 problems. Novice teams do problems 1-8; standard teams do problems 5-12.

Any team which submits a correct solution for any of problems 1-4 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-4.**

Problems 1-4 are easier than problems 5-8, which are easier than problems 9-12. These problems are correspondingly labeled “Novice”, “Intermediate”, and “Advanced.” Order does not otherwise relate to difficulty.

You may use the Internet only for submitting your solutions, reading Javadocs or Python documentation, and referring to any documents we link you to. You **may not** use the Internet for things like StackOverflow, Google, or outside communication.

As you may know, you can choose to solve any given problem in **Java, Python, or C++**. We have provided stub files for all questions in all languages that take care of the input parsing for you. **Do not modify any of the parsing or output code.** Just fill out the stub methods in each file and submit it with exactly the same name.

Do not modify any of the given methods or method headers in our stub files! They are all specified in the desired format. You may add class fields, helper methods, etc as you like, but modifying given parts will cause your code to fail in our testers.

There is no penalty for incorrect submissions. You will receive 1 point per problem solved. A team’s number of incorrect submissions will be used only as a tiebreaker.

Some problems use Java’s “long” type; if you are unfamiliar with them, they’re like an “int”, but with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
long myLong = 10000000000000L;
```

Otherwise, the “long” type functions just like the “int” type.

1. The Smartest People Work on Mechanical Bees



Nick has a suspicion that AI are trying to spy on him using mechanical bees. Nick also knows that any mechanical bee will have an odd number of fuzz follicles whereas a normal bee will have an even number of fuzz follicles. Nick is so paranoid that he captures every bee that he sees and counts the number of fuzz follicles on that bee. Help Nick find the mechanical bee if it exists. We guarantee there is at most 1 mechanical bee.

Implementation Details:

Implement the following function. It will be called by the grader once for each test case.

findOdd(arr):

- **arr**: an array of integers

Your function should return an integer.

Constraints

$$1 \leq |arr| \leq 10^4$$

$$0 \leq x \leq 10^5 \quad \text{where } x \text{ is an element of arr.}$$

Input

The first line contains a single integer **t** ($1 \leq t \leq 10^5$), denoting the number of test cases.

For each test case, the first line contains a single integer **n** ($1 \leq n \leq 10^4$) representing the number of bees Nick captures.

The next line contains **n** integers **x₁, x₂, ..., x_n** ($0 \leq x_i \leq 10^5$), where **x_i** is the number of follicles on the **i**th bee.

We guarantee that the sum over all **n** does not exceed $2 \cdot 10^5$.

Output

For each test case, return an integer representing the number of fuzz follicles on the mechanical bee. If there is no mechanical bee, output 0.

Sample Input	Parsed Input	Sample Output	Explanation
1 5 2 4 7 6 4	<code>findOdd([2, 4, 7, 6, 4])</code>	7	The only odd integer in the array is 7.
1 1 3	<code>findOdd([3])</code>	3	The only odd integer in the array is 3.
1 2 4 10	<code>findOdd([4, 10])</code>	0	There are no odd integers in the array, so we return 0.

2. Aaron's aalmighty AaI aahhhhhh



In 2020 Aaron was part of the team who created the first general intelligence AI. At the time Aaron was highly suspicious that the AI could end up turning evil so he took notes on how he might take down the AI. He made sure to write his notes using a secret code so that no AI discovers them.

Now it is 2022 and the AI has taken over the world. Aaron needs your help to decode the notes, represented by the string **s**. He tells you that each page of notes has a stationary letter, **s_n**. To decode the message, every letter to the left of **s_n** needs to be

shifted left (with wrap-around) and every letter to the right of **s_n** needs to be shifted right (with wrap-around). Given a page of notes and the index of the stationary letter, decode the message.

Implementation Details:

Implement the following function. It will be called by the grader once for each test case.

rotateString(s, n):

- **s**: the string to rotate
- **n**: the index that remains fixed

Your function should return a string.

Constraints

$$1 \leq |s| \leq 10000$$

$$0 \leq n \leq |s| - 1 \text{ where } n \text{ is an index of } s$$

(cont. on next page)

Input

The first line will contain a single integer **t** ($1 \leq t \leq 10^5$), denoting the number of test cases.

Each test case consists of multiple lines of input. The first line contains a lowercase string **s** ($1 \leq |s| \leq 10^4$), the string to rotate. The second line contains **n** ($0 \leq n \leq |s| - 1$), the index of the letter which stays fixed.

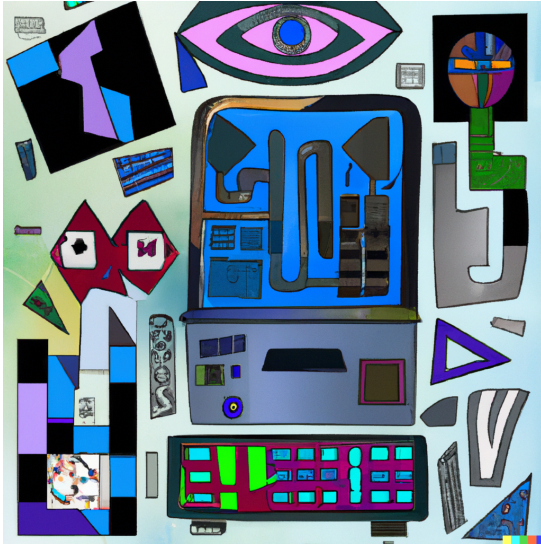
We guarantee that the sum over all **|s|** does not exceed $2 * 10^5$

Output

For each test case, return a string where characters to the left of index **n** are rotated left, and characters to the right of index **n** are rotated right.

Sample Input	Parsed Input	Sample Output	Explanation
1 abcdef 3	rotateString("abcdef", 3)	bcadfe	The character at index 3 is d and that stays fixed. "abc" (left of index 3) rotate left to get "bca". "e f" rotate right to get "fe"
1 aitakeover 0	rotateString("aitakeover", 0)	aritakeove	The first "a" stays fixed. Nothing to left of index 0. "itakeover" rotated right is "ritakeover"
1 happy 2	rotateString("happy", 2)	ahpyp	"p" stays fixed "ha" → "ah" "py" → "yp"

3. Cyle's Computer Conundrum



Cybersecurity specialist Kyle has planted a virus in the AI computer network. Kyle is a very sneaky guy. He doesn't want the AI to discover the virus immediately so he programmed the virus to slowly infect computers. Specifically, let c_i be the number of computers the virus infects on day i . We know that c_i is equal to $c_{i-1} + c_{i-2} + c_{i-3}$ for all $i > 3$. Given the number of computers infected on the first 3 days and a day n , help Kyle figure out how many computers will be infected on day n , and return your answer modulo $10^8 + 7$.

Implementation Details

`getNth(c1, c2, c3, n):`

- **c1**: represents the first element of the sequence
- **c2**: represents the second element of the sequence
- **c3**: represents the third element of the sequence
- **n**: represents the number element of the sequence that you should return

Your function should return an integer.

Constraints

$$1 \leq c1, c2, c3 \leq 100000$$

$$1 \leq n \leq 10000$$

Input

The first line contains a single integer t ($1 \leq t \leq 10^4$), denoting the number of test cases.

The first line of each test case contains three integers representing c_1 , c_2 , and c_3 respectively $1 \leq c_1, c_2, c_3 \leq 10^5$. The next line contains n ($1 \leq n \leq 10^4$).

We guarantee that the sum of n over all test cases does not exceed $3 \cdot 10^4$.

Output

For each test case, return an integer that represents the **n**th element of the sequence **mod 100000007** ($10^8 + 7$).

Sample Input	Parsed Input	Sample Output	Explanation
1 2 3 4 6	getNth(2, 3, 4, 6)	29	The 4th element (a_4) is $2+3+4=9$ $a_5 = 3+4+9=16$ $a_6 = 4+9+16=29$
1 4 10 17 4	getNth(4, 10, 17, 4)	31	$a_4 = 4 + 10 + 17 = 31$
1 1 1 2 8	getNth(1, 1, 2, 8)	44	$a_4 = 4, a_5 = 7, a_6 = 13, a_7 = 24,$ $a_8 = 44$

4. Monster Mash



As the war rages on, Ethan decides to turn to biological means to fight the AI invasion and begins mutating monsters. His tried-and-true strategy for maximizing mutated monster efficacy is to place monster eggs in a single line, allowing baby monsters to fuse with surrounding monsters of the same species once they are hatched. Ethan will only allow monsters to fuse if they are a part of a contiguous sequence of exactly k monsters of the same species, in order to preserve genetic stability and maximize output monster strength. After k monsters are fused into a powerful mutated monster, he removes it from

the line to be immediately put to use as a soldier and repeats the fusing process until no more monsters can be fused. Given a line of hatched monsters, let Ethan know what the final line of monsters will be after all eligible monsters have been fused and deployed.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`monsterMash(arr, k):`

- **arr:** array of integers
 - **k:** number of consecutive integers needed to fuse
- Your function should return an array.

Constraints

$$1 \leq |arr| \leq 10^4$$

$$0 \leq x \leq 10^5 \quad \text{where } x \text{ is an element of } arr.$$

$$2 \leq k \leq 10^5$$

(cont. on next page)

Input

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

For each test case, the first line contains a single integer n ($1 \leq n \leq 10^4$), representing the number of monsters. The second line contains n integers x_1, x_2, \dots, x_n ($0 \leq x_i \leq 10^5$), representing the species of the i th monster. The third line contains a single integer k , the number of contiguous monsters of the same species required for fusing.

We guarantee that the sum over all n does not exceed $2 \cdot 10^5$.

Output

For each test case, return a comma-separated list of values surrounded by square brackets that represents the final line of monsters after all possible fusions.

Sample Input	Parsed Input	Sample Output	Explanation
1 5 2 2 4 4 4 2	monsterMash([2, 2, 4, 4, 4])	[4]	[2, 2 , 4, 4, 4]
1 4 1 2 2 1 2	monsterMash([1, 2, 2, 1])	[]	[1, 2, 2 , 1] \rightarrow [1, 1] \rightarrow []
1 5 3 4 4 4 3 3	monsterMash([3, 4, 4, 4, 3])	[3, 3]	[3, 4, 4, 4, 3] \rightarrow [3, 4, 4, 4 , 3]

5. Evacuation Drill



As the A.I. invasion begins, evacuation sirens blare through the city. Grace has been tasked with helping civilians evacuate through a four-way intersection in the shortest amount of time possible.

At the corner of every intersection is a teleporter labeled **A**, **B**, **C**, or **D** (refer to the image below). Each teleporter can either be 'open' or 'closed'. When a teleporter 'opens', all other teleporters close'. 'Open' teleporters allow passage only from their adjacent 'closed' teleporters. Each teleporter may 'open' and 'close' multiple times.

A

B

D

C

Grace is given 4 pieces of information: 1) the order in which teleporters will 'open', 2) how many seconds the n th teleporter will be 'open', 3) the start and end destination of k civilians, and 4) the time each civilian arrives at the intersection. Using this information, help Grace find the shortest time it takes each civilian to traverse the intersection and evacuate successfully. There are no restrictions on the number of civilians that can cross at one time.

Note: a portal will close right before the next portal opens. For example, if the portal at B opens at time $t=0$ and its duration is 5 seconds, a civilian arriving at time $t=5$ cannot use this portal.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

evacuate(n , k , $order[]$, $duration[]$, $sfx[][]$):

- n – the number of times teleporters open
- k – the number of civilians
- $order[]$ – a string array representing the order in which teleporters open ($order_0$ is the first teleporters to open)
- $duration[]$ - an int array representing the duration that teleporters stay open for ($order_0$ is open for $duration_0$ time)
- $sfx[][3]$ - a string array where each element has 3 elements: s_i , f_i , x_i , s_i , represents the starting point for person i . f_i represents the finishing point for person i . x_i represents the time when person i arrives at their starting teleporters. You are guaranteed that $s_i \neq f_i$.

Your function should return an array that represents the time it takes each civilian to cross the intersection.

Constraints

$1 \leq n \leq 1000$

$1 \leq k \leq 1000$

We guarantee that the sum of n over all test cases is ≤ 1000

$order_i \leq 10^6$

$x_i \leq 10^9$

Input

The first line contains a single integer t ($1 \leq t \leq 100$), denoting the number of test cases.

For each test case, the first line contains two space-separated integers n and k ($1 \leq n, k \leq 1000$), representing the number of teleporters and the number of civilians respectively. The third line contains n characters $order_1, order_2, \dots, order_n$, where $order_i$ is an element of $\{A, B, C, D\}$ denoting the order that the teleporters open. The fourth line contains n integers $duration_1, duration_2, \dots, duration_n$, ($1 \leq d_i \leq 10^6$) where $duration_i$ is the duration that the i th teleporter stays open.

The following k lines describe the civilians where the i th line contains s_i, f_i which are elements of $\{A, B, C, D\}$ and x_i ($0 \leq x_i \leq 10^9$). s_i and f_i (integers) represent the start and destination teleporters of the i th civilian, and x_i denotes the time that the civilian arrives at teleporter s_i .

We guarantee that the sum of n over all test cases does not exceed 1000.

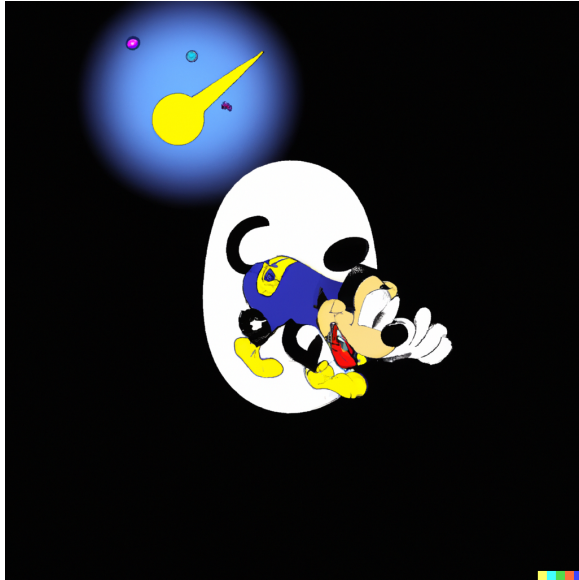
Output

For each test case, output **k** lines, where the **i**th line is the minimum amount of time it takes the **i**th civilian to reach teleporter **f_i** after arriving at the teleporter **s_i** at time **x_i**. If the civilian cannot reach their destination, output -1.

Sample Input	Parsed Input	Sample Output	Explanation
1 3 1 A B C 2 3 4 A C 0	evacuate(3,1, [A,B,C], [2,3,4], [[A,C,0]])	5	<p>The civilian arrives at position A at time $t = 0$.</p> <p>At time $t = 2$, the light to B turns on so the civilian walks to B. At time $t = 5$, the light to C turns on so the civilian walks to C.</p> <p>Since the civilian arrived at time 0 and reaches their destination at time 5, the total time it takes them to reach their destination is $5 - 0 = 5$.</p>
1 7 2 A C D A B C D 2 7 3 1 3 2 4 A D 13 B D 5	evacuate(7,2, [A,C,D,A,B,C,D], [2,7,3,1,3,2,4], [[A,D,13],[B,D,5]])	5 4	<p>The first civilian arrives at position A at time $t = 13$. At time $t = 18$ the light to D turns on so the civilian moves to position D. The total time to reach their destination is $18 - 13 = 5$.</p> <p>The second civilian arrives at position B at time $t = 5$. At time $t = 2$ the light at C turns on and stays on until $t = 9$. The civilian can move to position C immediately upon arrival. At time $t = 9$, the light at D turns on so the civilian</p>

			can move to position D at time $t = 9$. The total time to reach their destination is $9 - 5 = 4$.
1 5 2 A B A B A 5 5 5 5 5 A D 0 B A 10	evacuate(5, 2, [A,B,A,B,A], [5,5,5,5,5], [[A,D,0],[B,A,0]])	-1 0	<p>The first civilian arrives at position A at time 0 and wants to go to position D. However, they cannot get to position D, so the output is -1.</p> <p>The second civilian arrives at position B at time 10. At time $t = 10$, the light at A turns on and stays on until $t = 15$. The civilian can move to position A immediately upon arrival at time $t = 10$. The total time to reach their destination is $10 - 10 = 0$.</p>
1 1 1 B 1 A B 1	evacuate(1, 1, [B], [1], [[A,B,1]])	-1	<p>The civilian arrives at position A at time $t=1$ and wants to go to position B. The light to B turns on at time $t=0$ and stays on until $t=1$. However, the light turns off right before $t=1$, so the civilian cannot reach position B.</p>

6. Pluto's Portals



We've received intel that the robots are hiding a secret weapon on a coordinate plane. Luckily for us, Ishaan has been training his dog Pluto who can help recover this secret weapon. Pluto starts at the origin $(0, 0)$, and needs to reach the secret weapon at $(w - 1, h - 1)$. He can only travel to the right, but luckily there are strategically placed colored portals on the grid. When Pluto reaches a portal, he can teleport to any other portal of the same color. Note that Pluto doesn't have to take a portal when he reaches it. Let Ishaan know if Pluto is able to reach the secret weapon by only moving right and teleporting through portals of the same color.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`pathExists(w, h, portalArr):`

- **w**: width of grid (y-axis)
- **h**: height of grid (x-axis)
- **n**: number of portals (length of `portalArr`)
- **portalArr**: an array of tuples of length 3 - the first element is the x coordinate ($0 \leq x_i < w$), the second element is the y coordinate ($0 \leq y_i < h$), and the third element is its color ($0 \leq c_i < 10\,000$)

Your function should return a boolean where the function returns True if a path exists that goes from top left $(0, 0)$ to bottom right $(w-1, h-1)$, or False if a path does not exist.

Constraints

The grid is at least of size 2 by 2, so the bottom-left and top-right coordinate will not be the same.

$$2 \leq w \leq 10,000$$

$$2 \leq h \leq 10,000$$

$$1 \leq |\text{portalArr}| \leq 10000$$

portalArr contains tuples of length 3 in the form (x, y, c) where x, y, and c are integers

In the tuple (x, y, c), portal coordinates x and y represent distinct (x, y) coordinates

$$0 \leq x_i < w$$

$$0 \leq y_i < h$$

$$0 \leq c_i < 10,000$$

Input

The first line contains a single integer **t** ($1 \leq t \leq 10^4$), denoting the number of test cases.

For each test case, the first line contains two space-separated integers **w** and **h** ($2 \leq w, h \leq 10^4$). The second line contains a single integer **n**, the number of portals. The following **n** lines describe the portals, where the **i**th line contains x_i, y_i, c_i ($0 \leq x_i, y_i, c_i \leq 10^4$) which represent the **x** coordinate, **y** coordinate, and color of the **i**th portal.

We guarantee that each portal has a distinct location and that the sum over all **n** does not exceed **10⁵**.

Output

For each test case, output **true** if Pluto is able to get from (0, 0) to (w - 1, h - 1) and **false** otherwise.

(cont. on next page)

Sample Input	Parsed Input	Sample Output	Explanation
1 5 5 7 2 0 1 2 1 3 1 2 1 3 2 2 1 3 3 3 3 2 2 4 3	pathExists(5, 5, [(2, 0, 1), (2, 1, 3), (1, 2, 1), (3, 2, 2), (1, 3, 3), (3, 3, 2), (2, 4, 3)])	false	No path exists. The graph looks like: ..1.. ..3.. .1.2. .3.2. ..3.. In order to reach the bottom right corner of the graph, we need to take a portal of color 3. This is not possible starting at the top left because we can only reach a portal of color 1 in the first row. If we take the portal of color 1, we can reach a portal of color 2 in the second row. We can take a portal of color 2 to teleport to the third row. However, we cannot reach the portal of color 3 in the fourth row because we can only travel rightwards.
1 2 2 1 0 1 0	pathExists(2, 2, [(0, 1, 0)])	false	No path exists. The graph looks like: .. 0. Going all the way right, we encounter no portals, so we can't leave the first row.

1 5 1 2 1 0 0 2 0 0	pathExists(5, 1 [(1, 0, 0), (2, 0, 0)])	true	Skipping all portals, we can go from (0,0) to (4, 0). .00..
1 4 3 4 1 0 0 1 2 0 2 0 1 3 1 1	pathExists(4, 3 [(1, 0, 0), (1, 2, 0), (2, 0, 1), (3, 1, 1)])	true	A path exists. From (0,0) we go right to (1, 0) where we take the portal color 0 to (1, 2) and we go right to reach (3, 2). .01. ...1 .0..

7. Tien's Teslo Tracking



Tien has self-engineered a Teslo that has location tracking on any robot he desires. After several years, he has finally chased down and trapped the Android Head of Food and Agri-Culture (AH-FAC) of the AIs in the city labyrinth. The city has n buildings labeled 0 to $n - 1$ that are connected by $n - 1$ bidirectional roads. Furthermore, **there is a path between every pair of buildings in the city.** i.e. if you start at any building b , you are guaranteed that there is a sequence of roads you can travel across to reach any other building.

AH-FAC always knows where Tien is. In one turn, AH-FAC will either move along a road to an adjacent building, stay still, or teleport to any other building in the city. After AH-FAC moves, Tien can move along a road to an adjacent building or stay still. Tien catches AH-FAC if, after some turn, Tien and AH-FAC end up in the same building.

Tien discovered that AH-FAC will always move optimally, but can only teleport at most k times. Given a map of the city, the starting building of Tien and AH-FAC, and the number of teleports that AH-FAC has, return the minimum number of turns before Tien catches AH-FAC.

It can be shown that Tien can always catch AH-FAC after a finite number of turns.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`minTurns(n, k, ahfacInit, tienInit, streets):`

- **n**: an integer representing the number of buildings
- **k**: an integer representing the number of teleports that AH-FAC has
- **ahfacInit**: an integer representing the building that the AH-FAC starts at
- **tienInit**: an integer representing the building that Tien starts at
- **roads**: an array of tuples (u, v) of roads that connects buildings u and v

Your function should return an integer representing the minimum number of turns before Tien catches AH-FAC.

Constraints

$$3 \leq n \leq 2 * 10^5$$

$$0 \leq k \leq 10^9$$

$$0 \leq ahfacInit, tienInit < n$$

$$|roads| = n - 1$$

$$T \leq 10^5$$

Across all test cases in the multitest: Sum over all buildings is $\leq 3 * 10^5$

Input

The first line contains a single integer **t** ($1 \leq t \leq 3 * 10^4$), denoting the number of test cases.

For each test case, the first line contains a single integer **n** ($3 \leq n \leq 10^5$) representing the number of buildings in the city. The second line contains a single integer **k** ($0 \leq k \leq 10^9$) representing the number of times AH-FAC can teleport. The third line contains b_a ($0 \leq b_a < n$), the building that AH-FAC starts in. The fourth line contains b_t ($0 \leq b_t < n$, $b_t \neq b_a$), the building that Tien starts in. The following $n - 1$ lines describe the road of the city where the i th line contains u_i, v_i , ($0 \leq u_i, v_i < n$, $u_i \neq v_i$) meaning there is a road between buildings u_i and v_i .

We guarantee that the sum over all n does not exceed 10^5 .

Output

For each test case, output a single integer representing the minimum number of turns before Tien catches AH-FAC given that AH-FAC and Tien move optimally.

Sample Input	Parsed Input	Sample Output	Explanation
1 3 0 1 0 0 1 1 2	minTurns(3, 0, 1, 0, [(0, 1), (1, 2)])	2	<p>T-A-2 Initial starting state, where T is Tien and A is AH-FAC.</p> <p>T-1-A AH-FAC makes his only move away from Tien.</p> <p>0-T-A Tien moves closer to the AH-FAC.</p> <p>0-T-A AH-FAC has nowhere to move, without getting caught so he does not move.</p> <p>0-1-x On the next move, Tien will catch AH-FAC. That is two streets moved.</p>
1 4 1 1 0 0 1 1 2 2 3	minTurns(4, 1, 1, 0, [(0, 1), (1, 2), (2, 3)])	4	<p>T-A-2-3 Initial starting state, where c is Tien and r is AH-FAC.</p> <p>T-1-A-3 AH-FAC moves away from Tien.</p> <p>0-T-A-3 Tien moves closer to AH-FAC.</p> <p>0-T-2-A AH-FAC moves away from Tien.</p> <p>0-1-T-A Tien moves closer to AH-FAC.</p> <p>A-1-T-3</p>

			<p>AH-FAC cannot move without being caught next round so they teleport to node 0.</p> <p>A-T-2-3 Tien moves closer to AH-FAC.</p> <p>A-T-2-3 AH-FAC cannot make any more moves without being caught.</p> <p>x-1-2-3 Tien catches AH-FAC.</p>
--	--	--	--

8. Michelle's Molasses Manos



Oh no! AI robots are on the hunt for Michelle, and they have invaded her home! Luckily, she sees them coming and calls for help. Unfortunately, no one answers. Locked in her room and shaking with fear, Michelle realizes she must come up with a plan to stop the robots. She decides to turn to her laptop, as it's unlikely she'll be able to stop the robots by force. She gains access to the AI network and discovers she can shut down the robots in her house if she types in a compressed secret message composed entirely of **a**'s, **b**'s and **c**'s. To save memory, the secret message was compressed into a character array $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n$

and a length array l_1, l_2, \dots, l_n such that the i th character c_i is repeated l_i times. For example, if $c = [a, c, b]$ and $l = [2, 1, 3]$, the uncompressed message is 'aacbbb'.

Unfortunately, Michelle is SO slow and can only type one letter in 1 second. However, she can take 1 second to copy a substring of the text she has already typed as long as each letter of the substring is the same. For example, if Michelle has typed 'abaabbb', the valid substrings she can copy are 'a', 'aa', 'b', 'bb', 'bbb'.

Although Michelle can only copy **once**, she can paste the substring as many times as she likes, where pasting the substring also takes 1 second. Given a secret message and these constraints, find the shortest amount of time it will take Michelle to type out the message and shut down the robots.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`molasses(n, c[], l[]):`

- **n** - the length of the arrays `c[]` and `l[]`
- **c[]** - An array of characters
- **l[]** - An array of integers denoting the lengths of the consecutive characters

Constraints

There are T test cases. The sum of all N over the T test cases $\leq 2 \cdot 10^5$

The sum over all string lengths is $\leq 2 \cdot 10^6$

Input

The first line contains a single integer t ($1 \leq t \leq 1000$), denoting the number of test cases.

For each test case, the first line contains a single integer n ($1 \leq n \leq 2 \cdot 10^5$) denoting the size of the length and character arrays. The second line contains n characters c_1, c_2, \dots, c_n , where c_i is an element of $\{a, b, c\}$, and the third line contains n integers l_1, l_2, \dots, l_n ($1 \leq l_i \leq 10^6$).

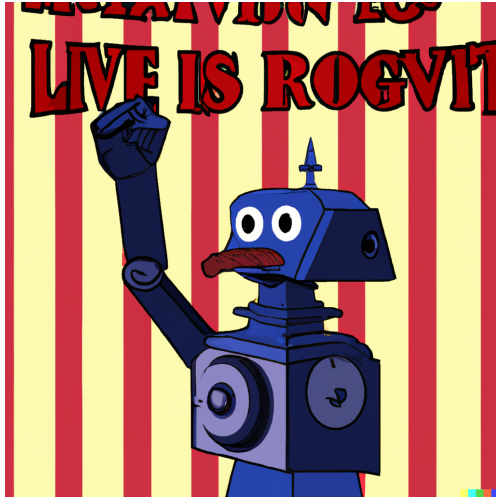
We guarantee that the sum over all the lengths of the uncompressed messages does not exceed $2 \cdot 10^6$.

Output

For each test case, output a single integer representing the amount of time it will take Michelle to type out the uncompressed message.

Sample Input	Parsed Input	Sample Output	Explanation
1 5 a b a b a 2 3 3 1 2	molasses(5, [a,b,a,b,a],[2,3,3, ,1,2])	10	We copy the first occurrence of two consecutive a's for 1 extra operation. Then, we paste our copy twice to save 1 operation each for a total of $11 - 1 + 2 = 10$ operations.
3 7 a b c c b a c 3 3 3 9 3 3 6 3 a a a 5 2 2 6 a b c a b c 1 1 1 1 1 1	molasses(6, [a,b,c,b,a,c],[3,3, ,3,9,3,3,6]) molasses(3, [a,a,a],[5,2,2]) molasses(3, [a,b,c,a,b,c],[1,1, ,1,1,1])	21 6 6	Case 1: Copy 3 c's. Case 2: Print first 3 a's, then copy them. Paste 2 times. $3+1+1+1=6$. Case 3: Actually not copying is optimal b/c copying would take 1 extra operation.

9. Give me Passcode, or Give me Death!



Suddenly, Denise finds herself in front of a glass door, with no memory of how she got there. A line of numbers **a** of length **n** appears on the glass, and in the palm of her hand the number **k** is written.

The AI bot speaks:

$$x \star y := k \left\lfloor \frac{x}{k} \right\rfloor + (y \bmod k)$$

The passcode is:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n (a_i \star (a_{i+1} \star (\cdots (a_{j-1} \star a_j) \cdots)))$$

With electric shock bars coming closer, help her figure out the passcode before it's too late.

Implementation Details

passcode(arr, k):

- **arr**: an array of integers
- **k**: an integer

Your function should return an integer.

Constraints

$$2 \leq |arr| \leq 10^6$$

$$0 \leq arr[i] \leq 10^6$$

Input

The first line contains a single integer **t** ($1 \leq t \leq 10^4$), denoting the number of test cases.

The first line of each test case contains two space-separated integers **n** and **k** ($2 \leq n \leq 10^6, 1 \leq k \leq 10^6$), denoting the number of elements and value of **k** for *****.

The second line of each test case contains **n** integers a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^6$), denoting the elements of the array **a**.

We guarantee that the sum of **n** across all test cases does not exceed 10^6 .

Output

For each test case, print a single integer - the passcode as above.

Sample Input	Parsed Input	Sample Output	Explanation
1 2 3 13 5	passcode([13, 5], 3)	14	$13 * 5 = 3 \left\lfloor \frac{13}{3} \right\rfloor + (5 \bmod 3) = 3(4) + 2 = 14$
1 2 4 5 3	passcode([5, 3], 4)	7	$5 * 3 = 4 \left\lfloor \frac{5}{4} \right\rfloor + (3 \bmod 4) = 7$
1 3 7 17 26 11	passcode([17, 26, 11], 7)	62	$(17 * 26) + (17 * (26 * 11)) + (26 * 11) = 19 + 18 + 25 = 62$
1 5 9 17 22 29 3 30	passcode([17, 22, 29, 3, 30], 9)	173	

10. Ziwen's Zapper



In a desperate attempt for survival, Ziwen, chased closely by violent AI robots, runs into a weapons warehouse and locks himself inside to keep the robots at bay. The only functional weapon left in the building is a huge zapper gun that requires **m** computers to control. Ziwen finds a box with **n** batteries to help power the computers. Given a list of size **m** with the total amount of power each computer needs to work, and a list of size **n** with the amount of power each battery contains, determine if Ziwen is able to power on all the computers using the batteries, in order to use his zapper to clear the surrounding area of enemy robots. Each battery can only be used to power one computer. Additionally, each computer needs an exact amount of power to

run, if the computer is given more or less power than is required it won't run.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

ziwensZapper(**n**, **m**, arr1, arr2):

- **n**: an integer representing the number of batteries
- **m**: an integer representing the number of computers
- **arr1**: array of integers representing the power of each battery
- **arr2**: array of integers representing the power needed by each computer

Constraints:

$$1 \leq n \leq 20$$

$$1 \leq m \leq 20$$

$$1 \leq x \leq 1000 \text{ where } x \text{ is an element of arr1.}$$

$$1 \leq y \leq 1000 \text{ where } y \text{ is an element of arr2.}$$

Input

The first line represents the number of test cases **T**.

For each test case:

The second line contains **n**, the number of batteries (arr1).

The third line contains **arr1**, which consists of **n** space separated integers.

The fourth line contains **m**, the number of computers (arr2).

The fifth line contains **arr2**, which consists of **m** space separated integers

Output

For each test case return true if we can power on Ziwen's Zapper else return false.

Sample Input	Parsed Input	Sample Output	Explanation
1 3 3 2 5 2 5 5	ziwensZapper(3, 2, [3, 2, 5], [5, 5])	true	One computer that needs 5 power can be matched with the 2 and 3 batteries. The other computer can be matched with the 5 battery.
1 2 3 4 2 5 3	ziwensZapper(2, 2, [3, 4], [5, 3])	false	The computer that needs 3 power can receive the 3 battery, but the computer that needs 5 power cannot receive enough power from the remaining 4 battery.
1 3 6 7 7 3 2 4 5	ziwensZapper(3, 3, [6, 7, 7], [2, 4, 5])	false	None of the computers can be powered by any of the batteries.

11. DESTROY the Metaverse



Oh no! The AIs have hacked into Ben's Metaverse he designed for the PClassic heroes to communicate! The Metaverse world can be represented by $r \times c$ cells and $(r + 1) \times (c + 1)$ coordinates. The world is covered by 4 walls which go from coordinates $(0, 0)$ to $(r, 0)$, $(0, 0)$ to $(0, c)$, $(r, 0)$ to (r, c) , and $(0, c)$ to (r, c) .

The Metaverse has self-constructed n internal walls, some which are completely horizontal and others are completely vertical, which may split the Metaverse into regions. Regions are cells surrounded by walls on the perimeter but have no

other walls inside. Note that walls can overlap each other.

In addition to the walls, there are k bombs that the AI have planted (bomb coordinates are not necessarily unique). A bomb at cell (i, j) means it is surrounded by coordinates (i, j) , $(i+1, j)$, $(i, j+1)$, and $(i+1, j+1)$.

If the AI ignites a bomb, all bombs in the same region will go off. Each bomb's power is equal to the number of cells in the region that it is in and the damage in the Metaverse is equal to the sum of all ignited bombs' power. The mischievous AI will always ignite one and only one bomb that causes the most damage in the Metaverse.

Ben has enough resources to construct one more horizontal wall anywhere of any size in the Metaverse before the AI chooses a bomb to ignite. He needs to try to minimize the amount of damage the AI will cause. Can you help Ben determine the minimum amount of damage that will occur in the Metaverse?

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

minimizeDamage(*r*, *c*, *n*, *k*, walls, bombs):

- **r**: rows of grid
- **c**: columns of grid
- **n**: number of walls (note: you can assume there are walls surrounding the entire region. This will not be explicitly given)
- **k**: number of bombs
- **walls[n][4]**: 2D array, each row containing the coordinates of 1 wall
- **bombs[k][2]**: 2D array, each row containing the coordinates of 1 object

Your function should return an int if using python, a long in java, or a long long in C++.

Constraints

$$1 \leq r, c \leq 3 * 10^6$$

$$1 \leq n \leq 1000$$

$$1 \leq k \leq 100000$$

Input

The first line contains two space-separated integers **r** and **c** ($1 \leq r, c \leq 3 * 10^6$).

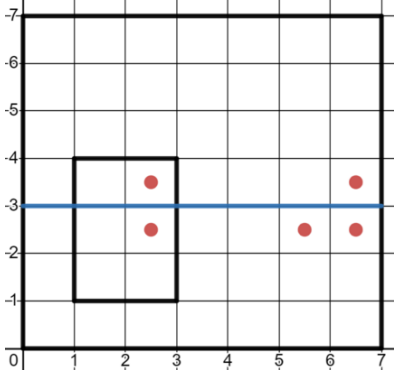
The second line contains two space-separated integers **n** and **k** ($1 \leq n \leq 1000$, $1 \leq k \leq 10^5$).

The next **n** lines contain four integers r_s, c_s, r_e, c_e ($0 \leq r_s, r_e \leq r$ and $0 \leq c_s, c_e \leq c$), the coordinates for one wall with start **s** and end **e**.

The next **k** lines contain two integers r_b, c_b ($0 \leq r_b < r$ and $0 \leq c_b < c$), the coordinates for one bomb.

Output

Print a single integer - the minimum damage in the Metaverse after building an extra wall.

Sample Input	Parsed Input	Sample Output	Explanation
7 7 4 5 1 1 4 1 1 1 1 3 1 3 4 3 4 1 4 3 2 2 3 2 2 5 2 6 3 6	metaverse(7, 7, 4, 5, [[1, 1, 4, 1], [1, 1, 1, 3], [1, 3, 4, 3], [4, 1, 4, 3]], [[2, 2], [3, 2], [2, 5], [2, 6], [3, 6]])	34	 <p>An illustration of the sample test case. It is best to place a horizontal wall from (3,0) to (3,7) to reduce the maximum damage to 34.</p>

12. Sheltering Survivors



Expert industrial engineer, Brian, wants to create shelters for the survivors of the AI apocalypse. He wants to make sure each shelter has electricity. Brian has found **n** locations for shelters and **m** locations for power plants. Brian has extension cords that are 10 miles long. This means that a power plant can only power a shelter that is at most 10 miles away.

Brian wants to make sure that power plants do not get overworked, so each power plant can power at most **p** shelters. Additionally, shelters need at least **q** power plants to power it and can have at most **o**

power plants powering them. If a power plant powers a shelter we consider the two to be a pair. Brian wants to find an assignment of power plants to shelters that maximizes the number of pairs.

Help Brian figure out how many pairs exist in this maximized arrangement. Note that if there does not exist a valid assignment such that each shelter gets at least **q** power plants then we say there are **0** pairs in the maximized arrangement.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`findMaxPairings(m, n, shelterLocations, powerplantLocations, p, q, o):`

- **m** : number of powerplants
- **n** : number of shelters
- **powerplantLocations**: array of locations for powerplants. Follows same format as **shelterLocations**.
- **shelterLocations**: array of locations for shelters. **shelterLocations[0][0]** is x coordinate of shelter 0 and **shelterLocations[0][1]** is the y coordinate of shelter 0.
- **p**: max number of shelters that a powerplant can be paired with
- **q**: min number of powerplants that a shelter can be paired with
- **o**: max number of powerplants that a shelter can be paired with

Your function should return an integer representing the max number of pairings

Constraints

$$1 \leq n \leq 100$$

$$1 \leq m \leq 100$$

$$1 \leq p \leq 100$$

$$1 \leq q \leq o \leq 100$$

Input

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

For each test case, the first line contains a single integer **m** ($1 \leq m \leq 100$), the number of power plants.

The second line contains a single integer **n** ($1 \leq n \leq 100$), the number of shelters.

The next **m** lines contain two integers, px_j, py_j ($-10^5 \leq px_j, py_j \leq 10^5$), the coordinates (px_j, py_j) in miles of power plant j .

The next **n** lines contain two integers, sx_i, sy_i ($-10^5 \leq sx_i, sy_i \leq 10^5$), the coordinates (sx_i, sy_i) in miles of shelter i .

The next line contains a single integer **p** ($1 \leq p \leq 100$), the maximum number of shelters that a power plant can be paired with.

The next line contains a single integer **q** ($1 \leq q \leq 100$), the minimum number of power plants that a shelter can be paired with.

The next line contains a single integer **o** ($q \leq o \leq 100$), the maximum number of power plants that a shelter can be paired with.

We guarantee that the sum over all **n** does not exceed 100 and the sum over all **m** does not exceed 100.

Output

For each test case, print a single integer - the maximum number of shelter and power plant pairings.

Sample Input	Parsed Input	Sample Output	Explanation
1 4 3 0 6 6 6 0 -6 0 -12 -6 6 0 0 0 7 2 1 2	findMaxPairing(4, 3, [[0, 6], [6, 6], [0, -6], [0, -12]], [[-6, 6], [0, 0], [0, 7]], 2, 1, 2)	5	One possible max pairing is: Powerplant 0 paired to Shelters 0, 2 Powerplant 1 paired to Shelters 1, 2 Powerplant 2 paired to shelters 1 powerplant 3 paired to nothing
1 4 5 -6 -6 -6 6 6 6 6 -6 0 0 0 6 6 0 -6 0 0 -6 3 2 3	findMaxPairing(4, 5, [[-6, -6], [-6, 6], [6, 6], [6, -6]], [[0, 0], [0, 6], [6, 0], [-6, 0], [0, -6]], 3, 2, 3)	11	One possible max pairing is: powerplant 0 paired to shelter 0, 3, 4 powerplant 1 paired to shelter 0, 1, 3 powerplant 2 paired to shelter 0, 1, 2 powerplant 3 paired to shelter 2, 4 Note that powerplant 3 can't be paired with shelter 0 because shelter 0 is already paired to 3 powerplants