Task 1: Labels (`labels`)

Today is the first day of work for Charles the Courier. He has been tasked with delivering N packages where each package has a (not necessarily unique) label number between 1 and N inclusive. At the end of each day, he is required to report a sequence A of N integers $A_1, \ldots, A_N$ where A_i is the label number of the i^{th} delivered package.

A mathematician at heart, Charles decides to use delta encoding to save on memory space and records a sequence D of $N - 1$ integers $D_1, \ldots, D_{N-1}$ instead, where $D_i = A_{i+1} - A_i$.

After delivering all the packages, Charles realises that he does not know how to recover A from D. Your task today is to help him, or state that it is not possible to uniquely recover A.

Input

Your program must read from standard input.

The first line contains a single integer N, the total number of packages.

The second line contains $N - 1$ space-separated integers, $D_1, \ldots, D_{N-1}$. D_i represents the difference between the label numbers of the $(i + 1)^{th}$ and i^{th} delivered package.

Output

Your program must print to standard output.

If it is possible to uniquely recover A from D, your output should contain N space-separated integers, the sequence A.

Otherwise, your output should contain a single integer on a single line, the integer -1.

Implementation Note

As the input lengths for subtasks 4 and 5 may be very large, you are recommended to use C++ with fast input routines to solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Labels.java` and place your main function inside `class Labels`.

Subtasks

The maximum execution time on each instance is 1.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $2 \leq N \leq 3 \times 10^5$
- $1 \leq A_i \leq N$
- $-N < D_i < N$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	**Marks**	**Additional Constraints**
1	7	$N = 2$
2	15	$2 \leq N \leq 6$
3	25	$2 \leq N \leq 10^3$
4	18	$-1 \leq D_i \leq 1$
5	35	-

Sample Testcase 1

This testcase is valid for subtasks 2, 3, and 5 only.

Input	Output

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

Sample Testcase 1 Explanation

We are able to uniquely recover $A = [1, 2, 5, 3, 4]$.

This is consistent with D since:

$$\begin{aligned} A_2 - A_1 &= 2 - 1 = 1 = D_1 \\ A_3 - A_2 &= 5 - 2 = 3 = D_2 \\ A_4 - A_3 &= 3 - 5 = -2 = D_3 \\ A_5 - A_4 &= 4 - 3 = 1 = D_4 \end{aligned}$$

Sample Testcase 2

This testcase is valid for subtasks 2, 3 and 5 only.

Input	Output

`5`
`2 2 -3 1`
	`1 3 5 2 3`

Sample Testcase 2 Explanation

We are able to uniquely recover $A = [1, 3, 5, 2, 3]$. Note that label numbers can appear more than once.

Sample Testcase 3

This testcase is valid for all subtasks.

Input	Output

`2`
`0`
	`-1`

Sample Testcase 3 Explanation

We are unable to uniquely recover A since we could have either $A = [1, 1]$ or $A = [2, 2]$.

Task 2: Discharging (`discharging`)

In a quest to be the very best, Pichuu the electric mouse has since started a new business that better caters to his strengths: using his favorite move Discharge to charge his customers' phones. A super effective business, it enjoys the patronage of many customers daily.

On a particular day, Pichuu has N customers queuing in wait to charge their phones. He is able to charge multiple phones simultaneously, where the power provided to each phone is equal and constant. Of course, different phone models possess different battery capacities, thus requiring varying lengths of time to charge fully. More specifically, the i^{th} phone takes T_i minutes to become fully charged.

When discharging, Pichuu does not stop until all the phones are fully charged. As such, to avoid getting shocked, customers are forced to wait until the last phone has finished charging before they retrieve their phones. However not all hope is lost: to minimise the total amount of time his customers spend waiting, Pichuu can divide them into an arbitrary amount of contiguous groups, before charging the groups in order. Thus, each group has to wait for the groups in front of them to finish before Pichuu can begin charging their phones.

Every customer will belong to exactly one group, where the i^{th} customer is assigned to the G_i^{th} group. Let the maximum T_i in the k^{th} group be M_k. Hence, the total amount of time spent waiting by the i^{th} customer, W_i, would be the sum of time taken for each group before him as well as his own group. **(For an illustration, refer to explanation for sample testcases)**

$$W_i = \sum_{n=1}^{G_i} M_n$$

To maximise customer satisfaction, Pichuu would like to minimise the sum of the waiting times.

Pichuu has but a mouse-sized brain that is incapable of calculating the optimal way to group his customers. Therefore, he requires your help to find the optimal grouping and hence the minimum sum of waiting time.

Input

Your program must read from standard input.

The first line contains an integer N, the number of customers.

The second line contains N integers, where the i^{th} integer represents the time taken to charge the i^{th} customer's phone T_i.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the minimum possible total waiting time.

Implementation Note

As the input lengths for subtasks 3, 4 and 6 may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Discharging.java` and place your main function inside `class Discharging`.

Subtasks

The maximum execution time on each instance is 1.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $1 \leq N \leq 10^6$
- $1 \leq T_i \leq 10^9$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Marks	Additional Constraints
1	9	$1 \leq N \leq 3$
2	13	$1 \leq N \leq 1500$ T_i is in non-decreasing order.
3	25	T_i is in non-decreasing order.
4	11	T_i is in non-increasing order.
5	14	$1 \leq N \leq 1500$
6	28	-

Sample Testcase 1

This testcase is valid for subtasks 5 and 7.

Input	Output
`5`	`27`
`1 3 2 6 3`	

Sample Testcase 1 Explanation

It is optimal to split the customers into two contiguous groups of (1, 3, 2) and (6, 3). The time taken for the two groups are 3 and 6 respectively as they are the maximum T_i in the groups. The waiting time for the first group is 3 per customer while the waiting time for the second group is 3 + 6 = 9 per customers as the second group has to wait for the first group to finish. Thus the total waiting time is 3 + 3 + 3 + 9 + 9 = 27.

Sample Testcase 2

This testcase is valid for subtasks 2, 3, 5 and 6.

Input	Output
`7`	`14`
`1 1 2 2 2 2 2`	

Sample Testcase 2 Explanation

The optimal grouping is to simply have all the customers in one group and charge their phones all at once. Thus the time taken for this group would be 2. The total waiting time would then be 2 + 2 + 2 + 2 + 2 + 2 + 2 = 14.

Task 3: Progression (`progression`)

Damian is making a video game! It consists of N missions with the i^{th} mission initially having a difficulty of D_i. Featuring state-of-the-art game design, the game can be played both forwards and backwards. Of course, it is also important that players feel a sense of progression — there should be a constant increase in difficulty. As such, Damian carries out a series of operations.

There are two types of operations that he does. The first type of operation is a *patch* operation defined by four integers L, R, S and C, meaning that mission i where $L \leq i \leq R$ will have its difficulty D_i increased by $S + (i - L) \times C$.

The second type of operation is a *rewrite* operation defined by four integers L, R, S and C, meaning that mission i where $L \leq i \leq R$ will have its difficulty D_i set to $S + (i - L) \times C$.

Damian decides that a contiguous subsequence of missions forms a *playable segment* if and only if their difficulties **change** at a constant rate (after all, players can always play the game backwards instead)! In other words, missions a to b where $1 \leq a \leq b \leq N$ form a playable segment if and only if $D_{i+1} - D_i = k$ for all $a \leq i < b$ where k is some integer (possibly $k \leq 0$). A single mission on its own forms a playable segment of length 1.

Occasionally, Damian will run an *evaluate* query defined by two integers L and R, meaning that he wants to find the length of the longest playable segment between missions L and R at that point in time.

Alas, Damian's operations do not necessarily improve the gaming experience. As such, he needs your help to answer the queries so that he can develop the best game possible.

Input

Your program must read from standard input.

The first line contains two integers, N and Q, representing the number of missions, and the total number of operations and queries.

The second line contains N space-separated integers, $D_1, \ldots, D_N$, defined above.

Q lines will follow, each representing either an operation or a query.

If the line begins with the integer 1, the next 4 integers are L, R, S and C, representing a *patch* operation.

If the line begins with the integer 2, the next 4 integers are L, R, S and C, representing a *rewrite* operation.

If the line begins with the integer 3, the next 2 integers are L and R, representing an *evaluate* query.

Output

Your program must print to standard output.

The output should consist of a single integer on a single line for each *evaluate* query, the length of the longest playable segment between missions L and R at that point in time.

Implementation Note

As the input lengths for subtasks 1, 3, 4, 5 and 6 may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Progression.java` and place your main function inside `class Progression`.

Subtasks

The maximum execution time on each instance is 3.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $1 \leq N, Q \leq 3 \times 10^5$
- $-10^6 \leq D_i, S, C \leq 10^6$
- $1 \leq L \leq R \leq N$

Under the given bounds, D_i is guaranteed to fit in a 64-bit signed integer at all times.

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Marks	Additional Constraints
1	9	$L = 1, R = N$ for all operations and queries.
2	15	$1 \leq N, Q \leq 10^3$
3	35	There are no *patch* and *rewrite* operations.
4	11	$L = R$ for all operations.
5	13	There are no *rewrite* operations.
6	17	-

Sample Testcase 1

This testcase is valid for subtasks 2 and 6 only.

Input	Output
`10 6`	`5`
`1 2 3 4 1 2 3 4 5 5`	`6`
`3 1 10`	`2`
`1 1 4 -1 -1`	`7`
`3 1 10`	
`3 9 10`	
`2 5 10 -2 -2`	
`3 1 10`	

Sample Testcase 1 Explanation

For the first *evaluate* query, missions 5 to 9 form the longest playable segment (with $k = 1$).

After the *patch* operation, the difficulties become $[0, 0, 0, 0, 1, 2, 3, 4, 5, 5]$.

For the second *evaluate* query, missions 4 to 9 form the longest playable segment (with $k = 1$).

For the third *evaluate* query, missions 9 to 10 form the longest playable segment (with $k = 0$), as we only consider missions between missions $L = 9$ and $R = 10$.

After the *rewrite* operation, the difficulties become $[0, 0, 0, 0, -2, -4, -6, -8, -10, -12]$.

For the final *evaluate* query, missions 4 to 10 form the longest playable segment (with $k = -2$).

Sample Testcase 2

This testcase is valid for subtasks 1, 2 and 6 only.

Input	Output
`10 5`	`5`
`1 2 3 4 1 2 3 4 5 5`	`5`
`3 1 10`	`10`
`1 1 10 1 2`	
`3 1 10`	
`2 1 10 3 4`	
`3 1 10`	

Sample Testcase 3

This testcase is valid for subtasks 2, 3, 4, 5 and 6 only.

Input	Output
`10 5`	`4`
`1 2 3 4 1 2 3 4 5 5`	`2`
`3 1 4`	`3`
`3 4 5`	`5`
`3 2 4`	`1`
`3 5 9`	
`3 10 10`	

Sample Testcase 4

This testcase is valid for subtasks 2, 4 and 6 only.

Input	Output
`10 5`	`6`
`1 2 3 4 1 2 3 4 5 5`	`5`
`2 10 10 6 1`	`5`
`3 5 10`	
`1 5 5 4 1`	
`3 1 5`	
`3 1 6`	

Sample Testcase 4 Explanation

Note that C is not necessarily 0 when $L = R$. However, its value is irrelevant to the operation.

Sample Testcase 5

This testcase is valid for subtasks 2, 5 and 6 only.

Input	Output
`10 5`	`4`
`1 2 3 4 1 2 3 4 5 5`	`2`
`1 1 4 -1 -1`	`9`
`3 1 5`	
`3 4 5`	
`1 5 10 -1 -1`	
`3 1 10`	

Task 4: Arcade (`arcade`)

Emily the alien octopus is playing an arcade game. The game machine consists of N buttons, numbered 1 to N from left to right. The game involves pressing a series of M buttons, one per second. At T_i seconds from the start of the game, button A_i must be pressed. Note that it is possible for $T_i = T_j, A_i = A_j$ even if $i \neq j$.

Each of Emily's hands can start at any position at the start of the game, and it takes exactly one second for Emily to move a hand from a button to an adjacent button. Emily's hands can move simultaneously, and it takes no time to press a button. As alien octopuses have an infinite number of hands, it is always possible to obtain the maximum score on the game by completing all M button presses. However, as Emily is lazy, she does not want to use all her hands. Let the minimum number of hands required to obtain the maximum score be S.

Given the exact series of button presses Emily has to accomplish, help her find out the minimum number of hands she needs to use to obtain the maximum score for the game. Help find and provide Emily the value of S.

Input

Your program must read from standard input.

The first line contains two integers N and M.

The second line contains M integers, where the i^{th} integer represents T_i.

The third line contains M integers, where the i^{th} integer represents A_i.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the minimum number of hands Emily needs to use to obtain the maximum score for the game.

Implementation Note

As the input lengths for subtask 7 may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Java or Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Arcade.java` and place your main function inside `class Arcade`.

Subtasks

The maximum execution time on each instance is 1.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $1 \leq N \leq 10^9$
- $1 \leq M \leq 5 \times 10^5$
- $1 \leq A_i \leq N$
- $1 \leq T_i \leq 10^9$

Your program will be tested on input instances that satisfy the following restrictions:

Let the minimum number of hands required to obtain the maximum score be S.

Subtask	Marks	Additional Constraints
1	7	$1 \leq N, M, T_i \leq 100$ $1 \leq S \leq 2$
2	11	$1 \leq N, M, T_i \leq 100$ $1 \leq S \leq 3$
3	12	$1 \leq N, M, T_i \leq 100$ $1 \leq S \leq 4$
4	21	$1 \leq M \leq 300$
5	14	$1 \leq M \leq 15\,000$
6	20	$1 \leq M \leq 100\,000$
7	15	-

Sample Testcase 1

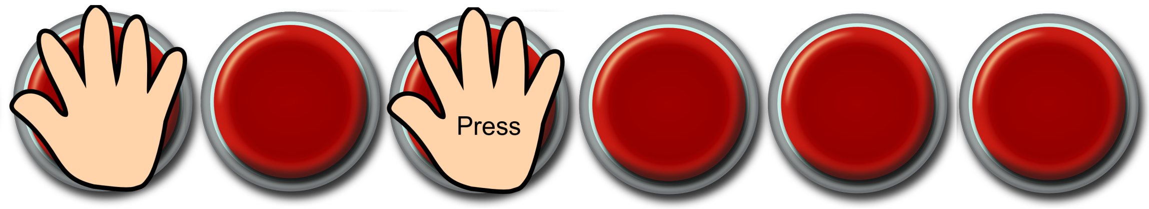
This testcase is valid for all subtasks.

Input	Output

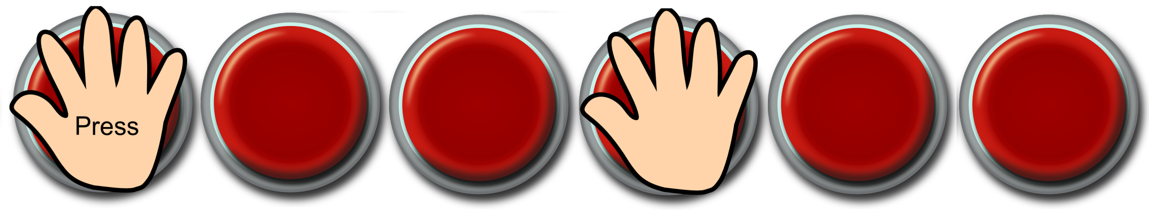
`6 4`
`1 2 3 4`
`3 1 2 6`
	`2`

Sample Testcase 1 Explanation

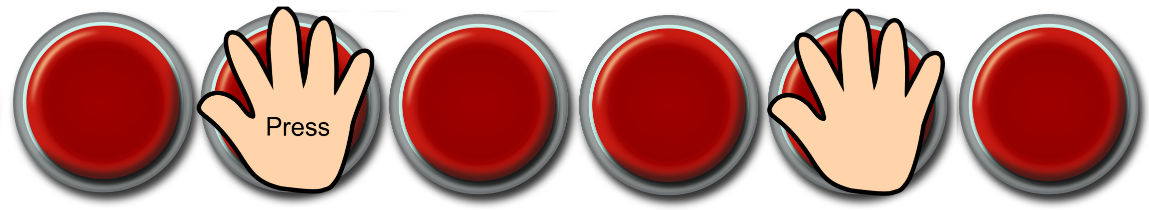
At the start of the game, Emily's hands will be in the following positions, with her right hand pressing button 3:



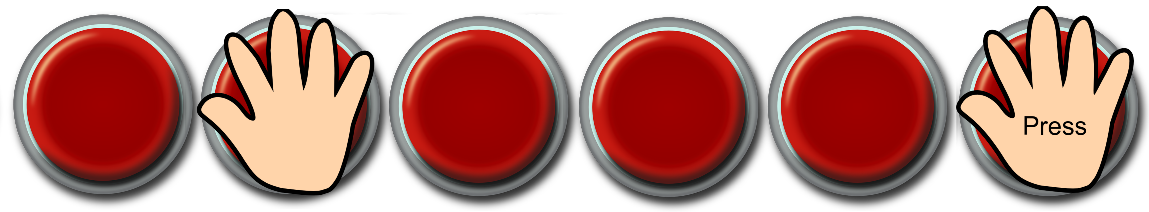
In the next second, Emily will move her right hand one button to the right, and press button 1 with her left hand.



After which, she will move both hands one button to the right simultaneously, and press button 2.



In the final second, she will move her right hand one step to the right and press button 6



Therefore, Emily would require **two** hands to complete the game. S would thus have the value of 2 and the program would output 2.

Task 5: Aesthetic (`aesthetic`)

Syrup the Turtle lives in a town with a layout consisting of N locations connected by M roads. Each location is indexed from $1, 2, ..., N$ and each road is indexed from $1, 2, ..., M$. The ith road directly connects locations A_i and B_i, has a length of W_i units, and can be traversed in either direction. These roads and locations are arranged such that it is possible to directly or indirectly travel between any pair of locations, and no two roads share the same endpoints.

Each road has its unique scenery, and the town's inhabitants have long come to a common consensus on how aesthetically pleasing each road is to travel through. The present-day index order of the roads reflects this; the roads are increasingly aesthetic from 1 to M.

Recently, the residents have sought to further refine the appeal of the town's topography. After weighing the many functional and aesthetic considerations of this task, an incredible compromise has been struck - an identical replica of a more aesthetic road is to be constructed within the trail of a strictly less aesthetic one. This action is possible for any pair of roads, and will extend the less aesthetic road by the length of the more aesthetic road. In other words, road j may be replicated into road i if and only if $i < j$, and doing so will change the length of road i to $W_i + W_j$. Now, all that remains is for a vote to be cast for one pair of roads to actually be chosen for this project.

Syrup makes frequent trips between his house in location N and the main square at location 1. He would like to know in advance how long the minimum distance between these two locations could get once the project is completed; your task is to determine the value of this distance.

Input

Your program must read from standard input.

The first line contains two integers, N and M.

M lines will follow. The i^{th} line contains three integers, A_i, B_i, and W_i, describing a single road.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the maximum possible length of the shortest path between locations 1 and N after the project is enacted.

Implementation Note

As the input lengths for subtasks 3, 4, 5, 6 and 7 may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Aesthetic.java` and place your main function inside `class Aesthetic`.

Subtasks

The maximum execution time on each instance is 2.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $3 \leq N \leq 300\,000$
- $2 \leq M \leq 300\,000$
- $1 \leq A_i \neq B_i \leq N$
- $0 \leq W_i \leq 10^9$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	**Marks**	**Additional Constraints**
1	5	$N, M \leq 100$
2	8	$N, M \leq 2000$
3	7	$M = N - 1$
4	15	$M = N$
5	16	$W_i = 1$
6	22	$0 \leq W_i \leq 10$
7	27	-

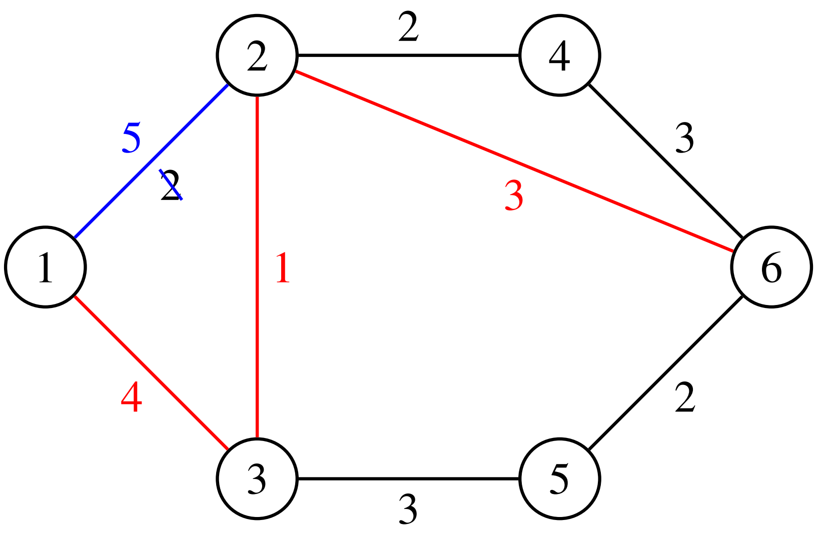
Sample Testcase 1

This testcase is valid for subtasks 1, 2, 6, and 7.

Input	Output

6 8
5 6 2
3 1 4
1 2 2
6 2 3
5 3 3
3 2 1
4 6 3
2 4 2
	8

Sample Testcase 1 Explanation



The shortest path between locations 1 and 6 is initially $1 \longrightarrow 2 \longrightarrow 6$, with length 5. If road 3, marked in blue, happens to be extended by any of the more aesthetic roads with length 3, the length of the shortest path could increase to 8 as marked in red.

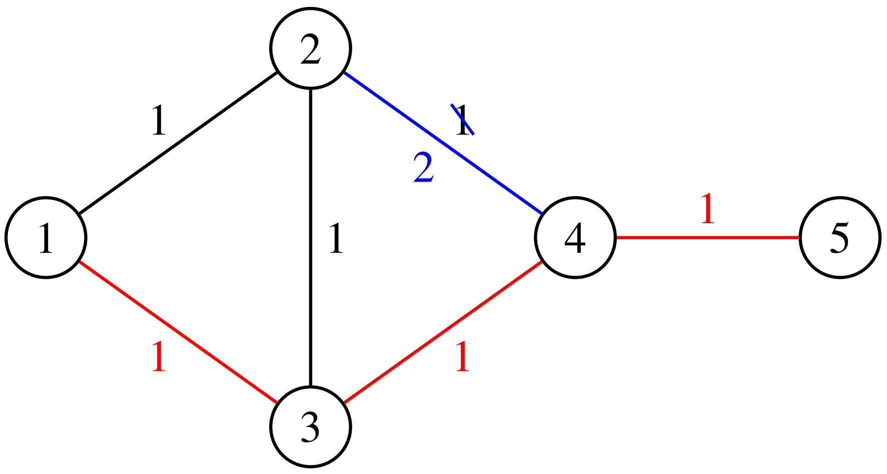
Sample Testcase 2

This testcase is valid for subtasks 1, 2, 5, 6, and 7.

Input	Output

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

Sample Testcase 2 Explanation



For this testcase, it can be proven that extending a less aesthetic road by a more aesthetic one will never increase the length of the shortest path beyond its original value of 3.

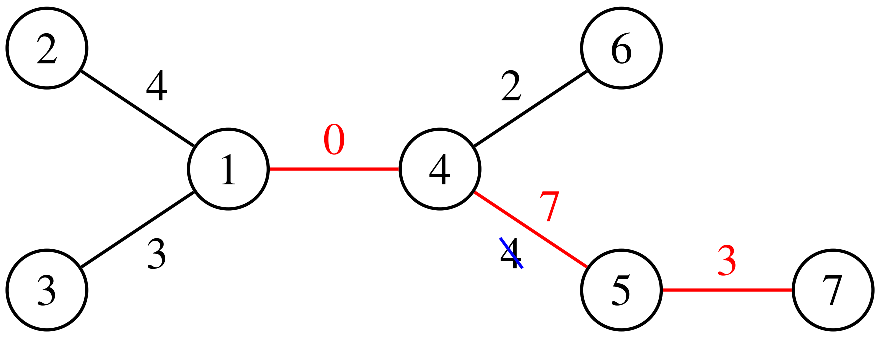
Sample Testcase 3

This testcase is valid for subtasks 1, 2, 3, 6, and 7.

Input	Output

`7 6`
`2 1 4`
`1 3 3`
`4 5 4`
`5 7 3`
`4 6 2`
`1 4 0`
	`10`

Sample Testcase 3 Explanation

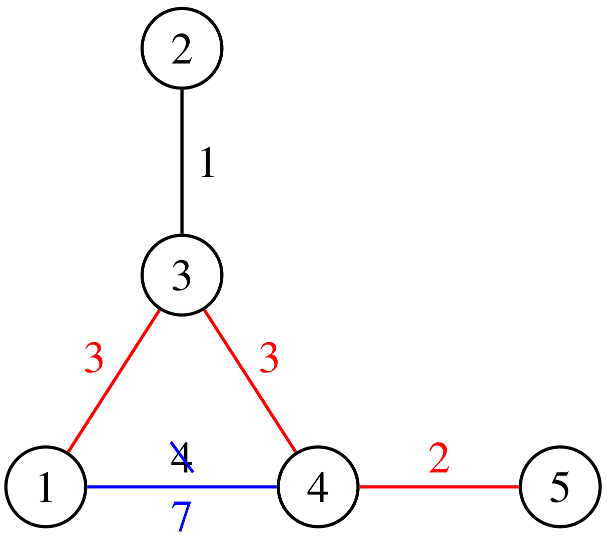


Sample Testcase 4

This testcase is valid for subtasks 1, 2, 4, 6, and 7.

Input	Output
`5 5`	`8`
`4 3 3`	
`1 4 4`	
`3 1 3`	
`4 5 2`	
`2 3 1`	

Sample Testcase 4 Explanation


2
1
3
3
3
1
~~4~~
7
4
2
5