Task 1: Cryptography (`crypto`)

Charles the Cryptographer has been researching novel methods of generating random numbers. In particular, by combining multiple sources of random numbers, he hopes to create a *cryptographically secure pseudorandom number generator (CSPRNG)*.

One algorithm that he has recently invented is as follows:

1. Randomly generate a sequence S of N **distinct** positive integers $S_1, \ldots, S_N$
2. Randomly shuffle S to obtain a permutation[1] P of N elements $P_1, \ldots, P_N$
3. Find the lexicographical order of P
4. As the answer can be very large, output the value modulo[2] 1 000 000 007

The lexicographical order of P is defined as the number of permutations of S that are lexicographically smaller than[3] or equal to P.

Unfortunately, Charles is a Cryptographer and not a Coder. Given the resultant permutation P, help Charles to find its lexicographical order, modulo 1 000 000 007.

Input

Your program must read from standard input.

The first line contains a single integer N.

The second line contains N space-separated integers, $P_1, \ldots, P_N$.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the lexicographical order of P, modulo 1 000 000 007.

Implementation Note

As the input lengths for subtasks 3, 4, 7, and 8 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

[1] A permutation P of a sequence S is a rearrangement of the elements of S

[2] The remainder when the value is divided by 1 000 000 007

[3] A permutation $P_1, \ldots, P_N$ is considered lexicographically smaller than another permutation $P_1', \ldots, P_N'$ if there exists $1 \leq k \leq N$ such that $P_k < P_k'$ and $P_i = P_i'$ for $i = 1, \ldots, k-1$.

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 `Crypto.java` and place your main function inside `class Crypto`.

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:

- $1 \leq N \leq 3 \times 10^5$
- $1 \leq P_i \leq 10^9$
- $P_i \neq P_j$ for $i \neq j$

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

Subtask	Marks	Additional Constraints
1	5	$N = 2$
2	9	$1 \leq N \leq 8$
3	10	P is either strictly increasing or decreasing.
4	11	$P = [k, 1, \ldots, k-1, k+1, \ldots, N]$ where $1 \leq k \leq N$
5	21	$1 \leq N \leq 3 \times 10^3, 1 \leq P_i \leq N$
6	13	$1 \leq N \leq 3 \times 10^3$
7	19	$1 \leq P_i \leq N$
8	12	-

Sample Testcase 1

This testcase is valid for subtasks 2, 6 and 8 only.

Input	Output
3 42 100 1	4

Sample Testcase 1 Explanation

We have the following 6 permutations in lexicographical order:

1. $[1, 42, 100]$
2. $[1, 100, 42]$
3. $[42, 1, 100]$
4. $[42, 100, 1]$
5. $[100, 1, 42]$
6. $[100, 42, 1]$

Hence, the lexicographical order of [42, 100, 1] is 4.

Sample Testcase 2

This testcase is valid for subtasks 2, 5, 6, 7 and 8 only.

Input	Output
5 1 5 2 4 3	20

Sample Testcase 3

This testcase is valid for all subtasks.

Input	Output
2 2 1	2

Task 2: Fuel Station (`fuelstation`)

As oil prices plummet, Pengu the Penguin has decided to visit Squeaky the Mouse who lives D kilometres away.

Pengu's *spheniscidae-mobile* starts its journey with F litres of fuel, consumes 1 litre of fuel per kilometre, and is able to hold any amount of fuel at any point in time.

Furthermore, there are N fuel stations between Pengu and his destination, with the i^{th} fuel station being X_i kilometres away from Pengu's house. At each fuel station, Pengu is only able to top up A_i litres of fuel (a limit imposed to prevent drivers from hoarding cheap fuel), and only if $F \leq B_i$ (to ensure that the fuel goes to drivers who most need it), Here, F refers to the amount of fuel (in litres) that Pengu **started** with.

Being an efficient penguin, Pengu would like to minimise the value of F while still being able to reach his destination.

Input

Your program must read from standard input.

The first line contains two integers N and D.

N lines will follow. The ith line contains three integers X_i, A_i and B_i, which represent the i^{th} fuel station.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the minimum value of F needed to reach the destination.

Implementation Note

As the input lengths for subtasks 2, 4, 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 `FuelStation.java` and place your main function inside `class FuelStation`.

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:

- $1 \leq N \leq 3 \times 10^5$
- $1 \leq A_i, B_i, D \leq 10^9$
- $0 < X_i < D$

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

Subtask	Marks	Additional Constraints
1	7	$N = 1$
2	13	$B_i = 10^9$
3	17	$1 \leq D \leq 10^4, 1 \leq N \leq 10^4$
4	12	$1 \leq D \leq 10^4$
5	19	$1 \leq N \leq 16$
6	11	$1 \leq N \leq 10^4$
7	21	-

Sample Testcase 1

This testcase is valid for subtasks 1, 3, 4, 5, 6 and 7 only.

Input	Output
`1 10` `4 8 6`	`4`

Sample Testcase 1 Explanation

We start with $F = 4$ litres of fuel and

1. Reach the only fuel station ($X_1 = 4$) with $4 - 4 = 0$ litres of fuel left.
2. Top up $A_1 = 8$ litres of fuel since $F \leq B_1 = 6$ to obtain $0 + 8 = 8$ litres of fuel left.
3. Reach our destination at $X = 10$ with $8 - (10 - 4) = 2$ litres of fuel left.

This is the minimum F possible.

Sample Testcase 2

This testcase is valid for subtasks 3, 4, 5, 6 and 7 only.

Input	Output
5 100 50 30 25 50 40 25 25 25 25 75 20 25 5 5 25	20

Sample Testcase 2 Explanation

We start with $F = 20$ litres of fuel and

1. Reach the 5^{th} fuel station ($X_5 = 5$) with $20 - 5 = 15$ litres of fuel left.
2. Top up $A_5 = 5$ litres of fuel since $F \leq B_5 = 25$ to obtain $15 + 5 = 20$ litres of fuel left.
3. Reach the 3^{rd} fuel station ($X_3 = 25$) with $20 - (25 - 5) = 0$ litres of fuel left.
4. Top up $A_3 = 25$ litres of fuel since $F \leq B_3 = 25$ to obtain $0 + 25 = 25$ litres of fuel left.
5. Reach the 1^{st} and 2^{nd} fuel stations ($X_1 = X_2 = 50$) with $25 - (50 - 25) = 0$ litres of fuel left.
6. Top up $A_1 + A_2 = 70$ litres of fuel since $F \leq B_1 = B_2 = 25$ to obtain $0 + 70 = 70$ litres of fuel left.
7. Reach our destination at $X = 100$ with $70 - (100 - 50) = 20$ litres of fuel left.

This is the minimum F possible.

Note that we could have also used the 4^{th} fuel station ($X_4 = 75$) if we wanted to. Although we reach it with $70 - (75 - 50) = 45 \not\leq B_4 = 25$ litres of fuel left, we can still use it since $F = 20 \leq B_4 = 25$. However, we are still able to reach our destination even if we do not use the fuel station.

Task 3: Relay Marathon (`relaymarathon`)

You are Nilan, the marathon organiser of the country Berapur. There are N cities in the country, connected by M roads. The cities are indexed from $1, 2, \ldots, N$ each road is indexed from $1, 2, \ldots, M$. The i_{th} road directly connects city u_i with city v_i, and it takes w_i seconds to travel on this road. The roads are bidirectional in nature and it is also ensured that there are no self-loops or multi-edges in the road network. Out of the N cities, there is a list of K distinct cities $A_1, A_2, \ldots, A_k$ which are **special**.

As the marathon organiser, you are going to try to organise a **relay marathon**. A **relay marathon** is defined as follows: 2 groups of people are present at the starting cities start_1 and start_2 respectively. Firstly the people from start_1 travel to finish_1. Once the first group reaches finish_1, then immediately (i.e without any delay), the second group of people start travelling from start_2 to finish_2. Once the second group reaches finish_2, the **relay marathon** ends. Do note that a **relay marathon** is **valid** only if $\text{start}_1, \text{finish}_1, \text{start}_2$ and finish_2 all are **special** and distinct from one another.

Let $D(a, b)$ denote the shortest time to travel from city a to city b in the above road network. In case there is no path from city a to city b, then let us define $D(a, b) = \infty$. Then the total time taken in such a **valid relay race** is defined as $D(\text{start}_1, \text{finish}_1) + D(\text{start}_2, \text{finish}_2)$.

Given this, your job is to find the minimum possible value of $D(\text{start}_1, \text{finish}_1)+D(\text{start}_2, \text{finish}_2)$ amongst all **valid** tuples $(\text{start}_1, \text{finish}_1, \text{start}_2, \text{finish}_2)$.

Note: In the input network of roads, it will always be ensured that there are exists at least one **valid** tuple of four distinct cities, (a, b, c, d), such that a, b, c, d all are **special** and $D(a, b) + D(c, d) < \infty$ (i.e there exists a path from city a to city b and another path from city c to city d.)

Input

Your program must read from standard input.

The first line of the input contains 3 integers, $N\ M\ K$ denoting the number of cities, the number of roads and the number of special cities respectively.

M lines will follow. Each consisting of 3 integers $u_i\ v_i\ w_i$, meaning that there is a road between city u_i and city v_i which takes w_i seconds to travel in either direction.

The next line contains K distinct integers $A_1, A_2, \ldots, A_k$ denoting the list of **special** cities.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the optimal minimum value of $D(\text{start}_1, \text{finish}_1) + D(\text{start}_2, \text{finish}_2)$ (in seconds).

Implementation Note

As the input lengths for subtasks 2, 3, and 4 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 `RelayMarathon.java` and place your main function inside `class RelayMarathon`.

Subtasks

The maximum execution time on each instance is 6.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $4 \leq K \leq N \leq 10^5$
- $2 \leq M \leq \min(\frac{N(N-1)}{2}, 3 \times 10^6)$
- $1 \leq w_i \leq 1000$ for all $1 \leq i \leq M$
- $1 \leq u_i \neq v_i \leq N$ for all $1 \leq i \leq M$

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

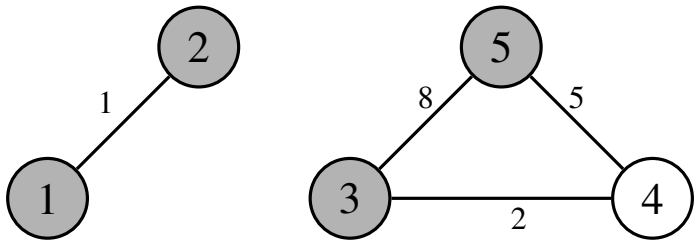
Subtask	Marks	Additional Constraints
1	5	$4 \leq K \leq N \leq 50$
2	12	$4 \leq K \leq N \leq 500$
3	25	- City 1 and City 2 both are **special** and directly connected with one another by an edge that takes 1 second to travel. - City 1 is NOT connected to any other city except City 2. - City 2 is NOT connected to any other city except City 1.
4	58	-

Sample Testcase 1

This testcase is valid for all subtasks.

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

Sample Testcase 1 Explanation



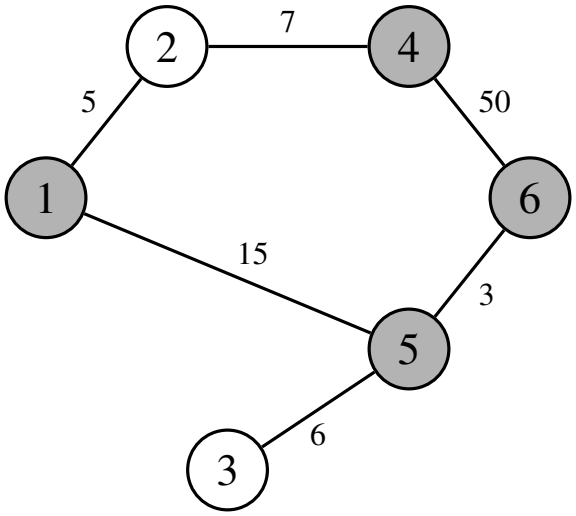
The cities in grey denote the **special** cities. We can observe that $D(1,2) = 1$ and $D(3,5) = \min(8, 2+5) = 7$. The optimal pairing here is $D(1,2) + D(3,5) = 1 + 7 = 8$ (i.e $\text{start}_1 = 1$, $\text{finish}_1 = 2$, $\text{start}_2 = 3$ and $\text{finish}_2 = 5$), any other kind of pairing configuration of $\{3, 1, 5, 2\}$ will not be better than this.

Sample Testcase 2

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

Input	Output
6 6 4 1 2 5 2 4 7 4 6 50 6 5 3 1 5 15 3 5 6 1 5 4 6	15

Sample Testcase 2 Explanation



The cities in grey denote the **special** cities. We can observe that $D(1,4) = 5 + 7 = 12$ and $D(5,6) = 3$. The optimal pairing here is $D(1,4) + D(5,6) = 12 + 3 = 15$ (i.e $\text{start}_1 = 1$, $\text{finish}_1 = 4$, $\text{start}_2 = 5$ and $\text{finish}_2 = 6$), any other kind of pairing configuration of $\{1, 4, 5, 6\}$ will not be better than this.

Task 4: Firefighting (`firefighting`)

A terrible fire swept through Mouseopolis, the capital of the Great Kingdom of Mouseland. Many residents of Mouseopolis suffered great losses as the fire razed through large chunks of the town. As the residents of Mouseopolis picked themselves up and slowly put their livelihoods back together after the fire, Squeaky — the benevolent king of Mouseland — vowed to prevent such a horrifying disaster from ever occurring again.

Mouseland consists of N towns, numbered from 1 to N. There are $N - 1$ bidirectional roads linking towns — each road is a direct link between some pair of towns. These roads may have varying lengths. The road network is designed such that it is possible to travel between any two towns using some sequence of roads. To quickly put out any fires, Squeaky has decided that fire stations should be built in some of the towns in Mouseland. These fire stations would then be able to dispatch fire trucks to mount an effective response to fires in any town.

Squeaky's advisors have determined that an effective response can be mounted only if the nearest fire station is no more than K kilometres away from the fire, as fire trucks would only then be able to reach the fire before the fire gets out of control. However, fire stations are expensive to maintain, so Squeaky would like to minimise the number of fire stations that need to be built.

Your task is to determine which towns fire stations should be built in, such that the required number of fire stations is minimised.

Note that fire stations may only be built in towns, and fires may only occur in towns.

Input

Your program must read from standard input.

The first line contains two integers, N and K, defined above.

$N - 1$ lines follow, describing the $N - 1$ bidirectional roads. Each of these lines contain three integers, A_i, B_i, and D_i, meaning that towns A_i and B_i are linked by a bidirectional road of length D_i kilometres.

Output

Your program must print to standard output.

The output should contain exactly two lines.

The first line should contain a single integer, F, the number of fire stations required.

The second line should contain exactly F distinct integers, specifying the towns in which fire stations should be built. This list may be printed in any order. If there are multiple possible ways, all of them will be accepted.

Note: If you receive the "Output is invalid" verdict, it means that you did not follow the required output format exactly. These are errors that can be detected with no information apart from the value of N (i.e. without knowing the road network configuration). For example: the F you printed is not an integer between 1 and N, or there are not exactly F integers that follow, or that they are not distinct integers between 1 and N.

Implementation Note

As the input and output lengths for subtasks 1, 4, and 6 may be very large, you are recommended to use C++ or Java with fast input and output 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 `Firefighting.java` and place your `main` function inside `class Firefighting`.

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:

- $1 \leq N \leq 3 \times 10^5$
- $1 \leq K \leq 10^{15}$
- $1 \leq A_i, B_i \leq N$ for all $1 \leq i \leq N-1$
- $1 \leq D_i \leq 10^9$ for all $1 \leq i \leq N-1$

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

Subtask	Marks	Additional Constraints
1	3	$K \leq 20$, $D_i \geq 30$ for all $1 \leq i \leq N-1$
2	5	$N \leq 17$, $K \leq 17$, $D_i = 1$ for all $1 \leq i \leq N-1$,
3	9	$N \leq 17$, $K \leq 10^6$, $D_i \leq 10^4$ for all $1 \leq i \leq N-1$,
4	19	$K \leq 30$, $D_i \geq 20$ for all $1 \leq i \leq N-1$
5	26	$N \leq 3 \times 10^3$
6	38	-

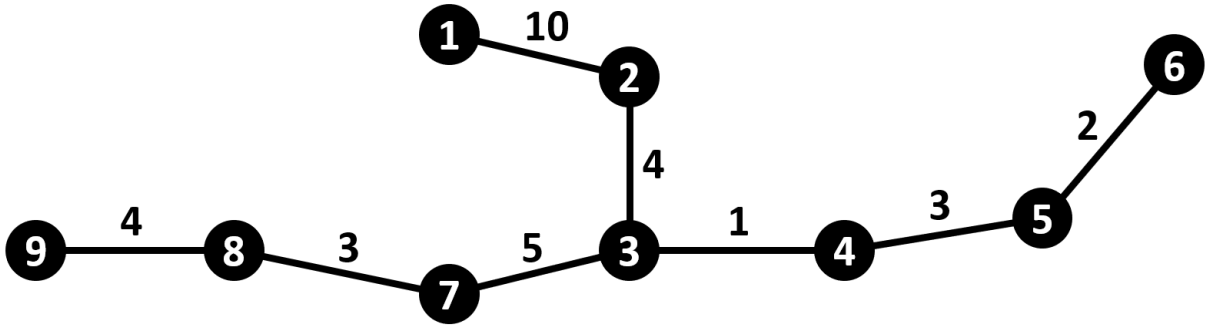
Sample Testcase 1

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

Input	Output
9 4 1 2 10 2 3 4 8 9 4 8 7 3 7 3 5 3 4 1 4 5 3 5 6 2	4 1 8 3 6

Sample Testcase 1 Explanation

Mouseland looks like this:



The minimum number of fire stations required is 4, and we can build fire stations at towns 1, 3, 6, and 8. After building fire stations at those towns, every town will be at most 4 kilometres away from the nearest fire station. Observe that the list of towns may be printed in any order.

Note that there are other valid solutions — for example, we can instead build fire stations at towns 1, 2, 5, and 8. All valid solutions will be accepted.

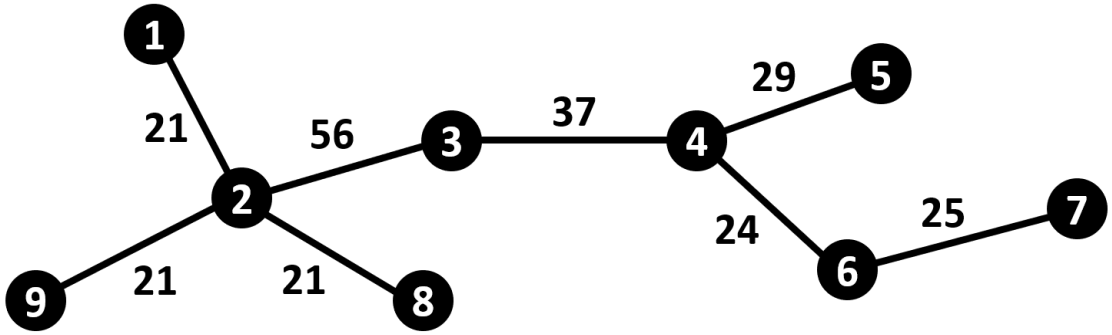
Sample Testcase 2

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

Input	Output
9 26 2 1 21 2 9 21 2 8 21 2 3 56 3 4 37 4 5 29 7 6 25 6 4 24	4 5 6 2 3

Sample Testcase 2 Explanation

Mouseland looks like this:



The minimum number of fire stations required is 4, and we can build fire stations at towns 2, 3, 5 and 6. After building fire stations at those towns, every town will be at most 26 kilometres away from the nearest fire station. The list of towns may be printed in any order.

It happens that for this test case there are no other set of 4 towns that will satisfy the requirements.

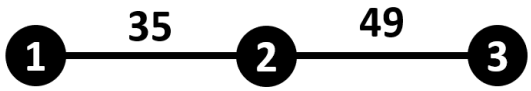
Sample Testcase 3

This testcase is valid for subtasks 1, 3, 4, 5, and 6.

Input	Output
3 18 1 2 35 2 3 49	3 1 2 3

Sample Testcase 3 Explanation

Mouseland looks like this:



The minimum number of fire stations required is 3 — we have to build a fire station at every town.

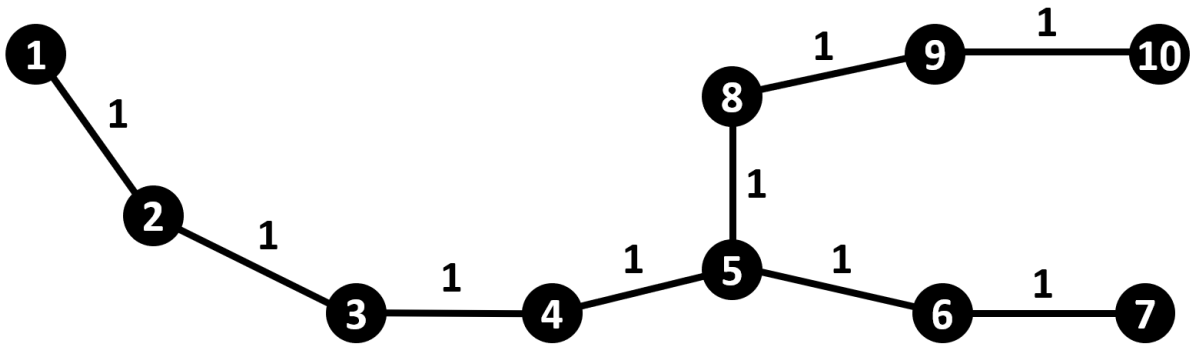
Sample Testcase 4

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

Input	Output
10 3 10 9 1 9 8 1 8 5 1 1 2 1 2 3 1 5 4 1 3 4 1 5 6 1 6 7 1	2 4 10

Sample Testcase 4 Explanation

Mouseland looks like this:



The minimum number of fire stations required is 2, and we can build fire stations at towns 4 and 10. After building fire stations at those towns, every town will be at most 3 kilometres away from the nearest fire station. The list of towns may be printed in any order.

Note that there are other valid solutions — for example, we can instead build fire stations at towns 3 and 5. All valid solutions will be accepted.