

2021 Team Formation Test

Task Overview

ID	Name	Time Limit	Memory Limit	Subtasks
T211	Snow Way	1.000 s	1024 MB	5 + 23 + 26 + 34 + 12
T212	Food Kangaroo	2.000 s	1024 MB	6 + 11 + 15 + 24 + 22 + 22
T213	Game Developer	2.000 s	1024 MB	16 + 4 + 21 + 6 + 14 + 34 + 5
T214	Re:Zero	1.000 s	1024 MB	100

Notice:

Unless otherwise specified, inputs and outputs shall follow the format below:

- One space between a number and another number or character in the same line.
- No space between characters in the same line.
- Each string shall be placed in its own separate line.
- Outputs will be automatically fixed as follows: Trailing spaces in each line will be removed and an end-of-line character will be added to the end of the output if not present. All other format errors will not be fixed.

C++ programmers should be aware that using C++ streams (`cin` / `cout`) may lead to I/O bottlenecks and substantially lower performance.

For some problems 64-bit integers may be required. In C++ the data type is `long long` and its token for `scanf`/`printf` is `%lld`.

All tasks are divided into subtasks. Unless otherwise specified, you need to pass all test cases in a subtask to get points.

T211 - SNOW WAY

Time Limit: 1.000 s / Memory Limit: 1024 MB

Alice and Bob are training for their sledding competition, in which they slide down a snowy slope following the guidance of various signposts.

Consider the snowy slope as a triangular grid of size N , which has i cells on the i -th row: $(i, 1), (i, 2), \dots, (i, i)$. Initially, all the cells are empty with no signposts placed on them. Starting at $(1, 1)$, Alice wants to reach cell (N, X) , while Bob wants to reach cell (N, Y) ($X \leq Y$), but they cannot do so unless guided by the signposts appropriately.

There are two types of signpost: type-A signposts and type-B signposts. Each signpost can be placed in exactly one cell (except cells in row N) to mark the sledding direction. The type of the signpost would affect how Alice and Bob's moves:

- Suppose Alice is at cell (i, j) ,
 - If the cell is unmarked, Alice stops there.
 - If the cell is marked with a type-A signpost, Alice moves to $(i + 1, j)$.
 - If the cell is marked with a type-B signpost, Alice moves to $(i + 1, j + 1)$.
- Suppose Bob is at cell (i, j) ,
 - If the cell is unmarked, Bob stops there.
 - If the cell is marked with a type-A signpost, Bob moves to $(i + 1, j + 1)$.
 - If the cell is marked with a type-B signpost, Bob moves to $(i + 1, j)$.

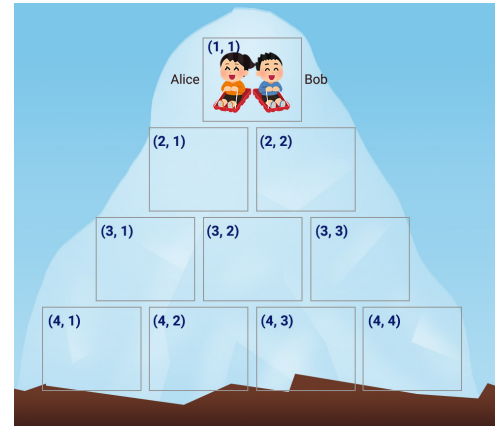


Figure 1: A triangular grid of size 4.

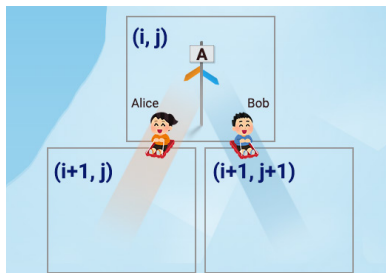


Figure 2a: Movements when a type-A signpost is placed.

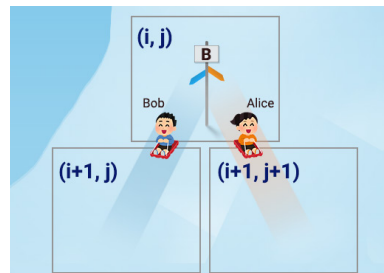


Figure 2b: Movements when a type-B signpost is placed.

You have A type-A signposts and B type-B signposts. You want to place these signposts on the grid, so that Alice and Bob can both reach their destinations. You should not place more than one signpost in a single cell to avoid confusing them. Again, signposts cannot be placed at row N for obvious safety reasons (they will fall out of the grid).

Your task is to either find a way to place the signposts, or determine that it is impossible for both Alice and Bob to reach their destinations. If there are more than one way, output any one of them. Please note that there is no need to use all the signposts and also no need to ensure that Alice or Bob can utilize a signpost.

INPUT

The first line contains three integers, N , A , and B .

The second line contains two integers, X and Y .

OUTPUT

If there is no solution, output **NO** in the first and only line.

Otherwise, output **YES** in the first line.

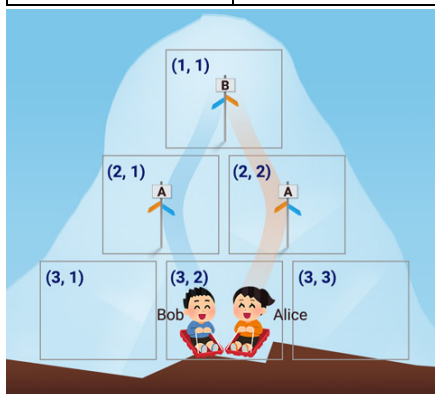
Then, on the next line, output an integer T_A ($0 \leq T_A \leq A$), the number of type-A signposts used. T_A lines should follow, each line containing two integers, the row and column of a type-A signpost.

On the next line, output an integer T_B ($0 \leq T_B \leq B$), the number of type-B signposts used. T_B lines should follow, each line containing two integers, the row and column of a type-B signpost.

The signposts can be output in any order. You score zero if you place more than one signpost on the same cell.

SAMPLE TESTS

Input	Output
1 3 2 2 2 2	YES 2 2 1 2 2 1 1 1



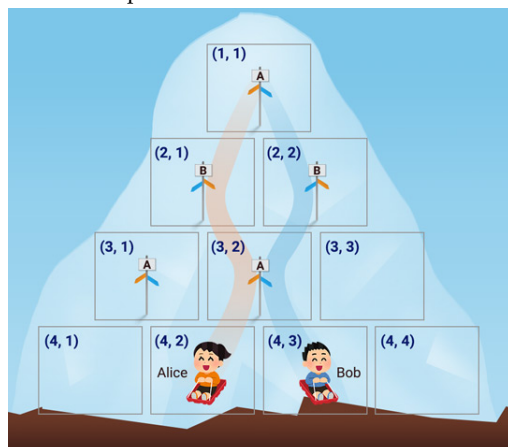
Alice's path: $(1, 1) \rightarrow (2, 2) \rightarrow (3, 2)$

Bob's path: $(1, 1) \rightarrow (2, 1) \rightarrow (3, 2)$

Note that not all signposts were used.

2	4 8 8	YES
	2 3	3
		1 1
		3 2
		3 1
		2
		2 2
		2 1

One of the possible solutions is as follows:



3	4 100 100	NO
	1 1	

SUBTASKS

For all cases:

$$2 \leq N \leq 10^5$$

$$0 \leq A, B \leq 2 \times 10^5$$

$$1 \leq X \leq Y \leq N$$

	Points	Constraints
1	5	$N = 3$
2	23	$A = B = 2 \times N$ $2 \leq N \leq 30$
3	26	$2 \leq N \leq 30$
4	34	$X < Y$
5	12	No additional constraints

T212 - FOOD KANGAROO

Time Limit: 2.000 s / Memory Limit: 1024 MB

Recently, everyone in Byteland are so lazy that they cannot be bothered cooking or going out for a meal. A food delivery platform "Food Kangaroo" was founded to collect food orders. Delivery riders can choose to take delivery orders and carry lunch boxes from restaurants to the destinations. After a hard-fought efforts to pass the driving test, Bob decides to be a delivery rider to earn some money.

In Byteland, there are N cities. Cities are numbered from 1 through N . There are $(H_F + H_T)$ highways, H_F of them are free while H_T of them are toll. Each highway connects a pair of distinct cities. No two highways connect the same pair of cities. Bob needs to pay toll every time he drives through a toll highway. Bob can drive on any highway in both directions.

There are D delivery orders posted, and each of them can be taken infinitely many times. The k^{th} order requires a lunch box delivery from city X_k to city Y_k . It is guaranteed that cities X_k and Y_k are connected through some highways and $X_k \neq Y_k$. Delivery rider can only take the order when he is at city X_k . When he takes the order, he carries a lunch box from city X_k to city Y_k . When he arrives city Y_k , the lunch box is considered as delivered and he receives a reward of R_k Bytecoins.

Bob starts his job as a delivery rider from city 1 and ends his duty in any city he wants. He uses a smart card called "Byteca" to pay tolls and receive rewards. The balance of "Byteca" ranges from negative infinity to positive infinity so tolls can still be paid even when the balance is less than the toll. Specifically, Bob can still pay tolls when the balance is negative.

Bob can drive along the roads to take orders and carry lunch boxes. Bob does not have any time limit to deliver orders and he can take infinitely many orders. However, at any give moment, he could carry at most one lunch box.

As a friend of Bob, please find out if he is able to earn infinitely many Bytecoins or the maximum Bytecoins he can earn if he takes and delivers orders optimally.

INPUT

The first line contains four integers: N , H_F , H_T and D .

The i^{th} of the next H_F lines, each contains two integers u_i and v_i : the i^{th} free highway connects city u_i and v_i .

The j^{th} of the next H_T lines, each contains three integers u_j , v_j and t_j : the j^{th} toll highway connects city u_j and v_j , and its toll is t_j Bytecoins.

Each highway connects a pair of distinct cities. No two highways connect the same pair of cities.

The k^{th} of the next D lines, each contains three integers X_k , Y_k and R_k : the k^{th} delivery order requires delivery from city X_k to Y_k , and its reward is R_k Bytecoins.

It is possible that multiple deliver orders share the same X_k and/or Y_k .

OUTPUT

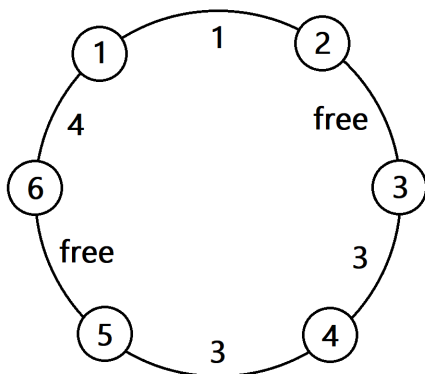
If there is a way to earn infinitely many Bytecoins, output **YES** on the first and only line.

Otherwise, on the first line, output **NO**. On the second line, output an integer, the maximum Bytecoins Bob can earn.

SAMPLE TESTS

	Input	Output
1	<pre> 6 2 4 1 2 3 6 5 1 2 1 3 4 3 4 5 3 6 1 4 2 5 7 </pre>	<pre> NO 1 </pre>

The following figure shows the roads connecting the cities:



Bob can travel in the following way:

$1 \rightarrow 2$ (pick up order 1) $\rightarrow 1 \rightarrow 6 \rightarrow 5$ (deliver order 1)

2	<pre> 6 2 4 1 2 3 6 5 1 2 1 3 4 3 4 5 3 6 1 4 2 5 7 </pre>	<pre> NO 1234567 </pre>
----------	--	-------------------------

This output scores 30% in subtasks excluding Subtask 5.

3	<pre> 6 2 4 3 2 3 6 5 1 2 1 3 4 3 4 5 3 6 1 4 2 5 7 6 3 6 5 6 3 </pre>	<pre> YES </pre>
----------	--	------------------

Bob can travel in the following way:

$1 \rightarrow 2$ (pick up order 1) $\rightarrow 3 \rightarrow 4 \rightarrow 5$ (deliver order 1, pick up order 3) $\rightarrow 6$ (deliver order 3, pick up order 2) $\rightarrow 1 \rightarrow 2 \rightarrow 3$ (deliver order 2) $\rightarrow 2$ (pick up order 1) $\rightarrow 3 \rightarrow 4 \rightarrow 5$ (deliver order 1, pick up order 3) $\rightarrow \dots$

4	7 3 5 3	NO
	2 1	2
	3 4	
	5 6	
	6 7 2	
	2 4 3	
	5 7 3	
	1 3 4	
	2 3 5	
	3 2 1	
	7 5 5	
	1 3 5	

SUBTASKS

For all cases:

$$2 \leq N \leq 2 \times 10^5$$

$$0 \leq H_F, D \leq 2 \times 10^5$$

$$0 \leq H_T \leq 500$$

$$1 \leq t_j \leq 10^9 \text{ (for all } 1 \leq j \leq H_T)$$

$$1 \leq R_k \leq 10^9 \text{ (for all } 1 \leq k \leq D)$$

	Points	Constraints
1	6	$H_T = 0$
2	11	$D = 1$
3	15	$2 \leq N \leq 50$ $H_F = 0$ $0 \leq D \leq 10$ $0 \leq H_T \leq 50$
4	24	$H_F = 0$
5	22	There is no way to earn infinitely many Bytecoins.
6	22	No additional constraints.

SCORING

In subtasks except Subtask 5, you can obtain a partial score. For each subtask (except Subtask 5):

- You score 100% if your output is correct in all test cases within the subtask; otherwise
- You score 30% if the first line of your output is correct in all test cases within the subtask; otherwise
- You score 0.

Note that your program shall still fulfill the output format specified (i.e. output an integer on the second line in NO cases) even if it is attempted for partial score.

T213 - GAME DEVELOPER

Time Limit: 2.000 s / Memory Limit: 1024 MB

Bob is a game developer and he is making a game for Alice, his girlfriend, as Valentine Day's gift. He knows that Alice likes fighter games, so he wants to make a game with $N + 1$ stages, the stage are numbered from 0 to N .

At the beginning, only stage 0 is unlocked. For each stage $1 \leq i \leq N$, if you beat stage P_i ($0 \leq P_i \leq i - 1$), then stage i is unlocked.

Bob, wanting to make Alice happy, has made N presents, numbered from 1 to N , and the j -th present has value V_j . He now has to assign a present to every stage except stage 0. Formally, Bob shall assign present m_i to stage i , where $m_{1..N}$ is a permutation of $1..N$. Denote R_i as the value of the present assigned for stage i , we have $R_i = V_{m_i}$.

He wants to assign the presents in such a way that for all $1 \leq i \leq N$, $R_{P_i} \leq R_i$, that is the value of present assigned to stage i is larger than or equal to the value of present of the stage that unlocks stage i . For simplicity, we treat $R_0 = 0$ here as no present is assigned to stage 0.

Bob introduces this game to Alice. Alice observes that playing the game in the order of stage $1, 2, \dots, N$ is always possible as $P_i \leq i - 1$, so Alice has decided to play the game in this order.

After Bob knows the order of stages that Alice will play in, he wants to maximize the value of presents Alice will get early so she will be happier, so he wants to assign the presents in a way that obtains the lexicographically greatest sequence of $R_{1..N}$ while satisfying the requirements above.

A sequence x_1, x_2, \dots, x_N is lexicographically greater than the sequence y_1, y_2, \dots, y_N if there exists i ($1 \leq i \leq N$) such that $x_i > y_i$ and for all j ($1 \leq j < i$), $x_j = y_j$.

INPUT

The first line contains an integer N .

The second line contains N integers, the j -th integer is V_j .

The third line contains N integers, the i -th integer is P_i .

OUTPUT

Output N space-separated integers on one line, the lexicographically greatest sequence of $R_{1..N}$ that satisfies all the requirements. The i -th integer is R_i . It can be proven that there always exist such $R_{1..N}$ that satisfies all the requirements.

SAMPLE TESTS

	Input	Output
1	<div>4</div> <div>6 7 8 9</div> <div>0 1 1 2</div>	6 8 7 9

$\{6, 7, 8, 9\}, \{6, 7, 9, 8\}, \{6, 8, 7, 9\}$ satisfy the requirement $R_{P_i} \leq R_i$
The lexicographically greatest one is $\{6, 8, 7, 9\}$.

2	<div>4</div> <div>5 5 6 6</div> <div>0 0 1 1</div>	5 6 6 5
----------	--	---------

$\{5, 5, 6, 6\}, \{5, 6, 5, 6\}, \{5, 6, 6, 5\}$ satisfy the requirement $R_{P_i} \leq R_i$
The lexicographically greatest one is $\{5, 6, 6, 5\}$.

SUBTASKS

For all cases:

$$1 \leq N \leq 10^6$$

$$1 \leq V_i \leq 10^9$$

$$0 \leq P_i \leq i - 1, \text{ for all } 1 \leq i \leq N$$

	Points	Constraints
1	16	$1 \leq N \leq 2 \cdot 10^5$ V_i are distinct
2	4	$1 \leq N \leq 8$
3	21	$1 \leq N \leq 200$
4	6	$1 \leq N \leq 1000$
5	14	$1 \leq N \leq 5000$
6	34	$1 \leq N \leq 2 \cdot 10^5$
7	5	No additional constraints

T214 - RE:ZERO

Time Limit: 1.000 s / Memory Limit: 1024 MB

In the anime 《Re:Zero – Starting Life in Another World》 (Re:ゼロから始める異世界生活), the protagonist -- Subaru finds himself transported to another world on his way home from the convenience store. Upon his arrival in the new world, Subaru realizes that he acquired a new ability "Return by Death" that allows him to go back to the time when he died while still retaining his memories of what he has experienced. In other words, this ability allows him to "foresee" the future in exchange for his death.

A villain, Elsa wants to kill Subaru. Therefore, she has summoned N monsters at Day 0 which are labelled 0 to $N - 1$. The monsters will move around the town and destroy the city unless Subaru surrenders to Elsa. The city can be viewed as a number line and the origin of the city is point 0. Each monster's starting position is an integer value that the absolute value shows the unit distance between the monster and the origin. If the value is negative, then the monster is on the left side of the origin. If the value is positive, then the monster is on the right side of the origin. Note that the starting positions might not be distinct as they may attack the city as a group.

The monster i 's movement can be described by integers X_i and T_i , where the monster would move $|X_i|$ units every T_i -th day (i.e. if $T_i = 7$, then the monster would move at the beginning of Day 7, 14, 21...). If X_i is positive, the monster would move to the right while if X_i is negative, the monster would move to the left. Unfortunately, all starting positions, X_i and T_i of the monsters are unknown to Subaru.

Subaru has found out that when he surrenders to Elsa at Day D , Elsa, without knowing Subaru has the ability "Return by Death", would expose the positions of all the monsters at the end of Day D before killing Subaru just to tease him. And Subaru will return to Day 0 as it activates his ability. In short, if Subaru surrenders at Day D , he can get the positions of all monsters at the end of Day D and returns to Day 0 again.

Defeating Elsa is easy, but Subaru doesn't want anyone in the city to be injured. So he has to gather the values of all X_i and T_i in order to schedule the evacuation plan for the citizens. As you can imagine, the feeling of dying is unbearable for anyone. Being a friend of Subaru, please help him to find all X_i and T_i with Subaru dying as few times as possible.

IMPLEMENTATION DETAILS

You should implement the following procedure:

```
std::vector<std::pair<int, int>> analyze_monsters(int N)
```

- N : the number of monsters.
- This procedure will be called exactly once by the grader.
- It should return an array of pairs P of length N , indicating the information of each monster's movement, with $P[i]$. *first* representing monster i 's X_i and $P[i]$. *second* representing T_i .

The above procedure can make calls to the following procedure:

```
std::vector<long long> surrender(long long D)
```

- D : the Day that Subaru surrenders to Elsa.
- D is at least 0 and at most 10^{15} .
- This procedure will return an array S of length N , where $S[i]$ represents the position of the monster i at the end of day D .

Notice: your submission must not perform any of the following actions, which result in an unspecified grading verdict:

- read from the standard input, write to the standard output, or interact with any other file,
- call `exit()`

Note that the grader is not adaptive. The labels, starting positions, X_i and T_i of the N monsters are fixed before the start of the interaction.

SAMPLE RUN

Suppose $N = 2$, monster 0 has a starting position of 0 and moves 2 units towards right every 3rd day, while monster 1 has a starting position of 2 and moves 3 units towards left every 4th day. The procedure `analyze_monsters` is called in the following way:

```
analyze_monsters(2)
```

This procedure may call the followings in such order:

Function Call	Returns	Explanation
<code>surrender(5)</code>	<code>[2, -1]</code>	Subaru surrenders at Day 5. The position of the monsters at the end of Day 5 are 2 and -1.
<code>surrender(6)</code>	<code>[4, -1]</code>	Subaru surrenders at Day 6. The position of the monsters at the end of Day 6 are 4 and -1.
<code>surrender(2)</code>	<code>[0, 2]</code>	Subaru surrenders at Day 2. The position of the monsters at the end of Day 2 are 0 and 2.
<code>surrender(4)</code>	<code>[2, -1]</code>	Subaru surrenders at Day 4. The position of the monsters at the end of Day 4 are 2 and -1.
<code>surrender(3)</code>	<code>[2, 2]</code>	Subaru surrenders at Day 3. The position of the monsters at the end of Day 3 are 2 and 2.

At this point, there is sufficient information to conclude that $X_0 = 2$, $T_0 = 3$, $X_1 = -3$ and $T_1 = 4$. As such, the `analyze_monsters` procedure should return `[(2, 3), (-3, 4)]`.

CONSTRAINTS

- $1 \leq N \leq 1000$

For all $0 \leq i \leq N - 1$:

- $-10^{18} \leq \text{monster } i\text{'s starting position} \leq 10^{18}$
- $-1000 \leq X_i \leq 1000$
- $X_i \neq 0$
- $1 \leq T_i \leq 100$

SCORING

If in any of the test cases, the calls to the procedure `surrender` do not conform to the rules mentioned above, or the return value of `analyze_monsters` is incorrect, the score of your solution will be 0. Otherwise, let W be the maximum number of calls to the procedure `surrender` among all test cases. Then, the score will be calculated according to the following table:

Condition	Score
$130 < W$	0
$8 \leq W \leq 130$	$72 - 6.5 \cdot \sqrt{W - 7.5}$
$4 \leq W \leq 7$	$100 - 6.9 \cdot (W - 3)$
$W \leq 3$	100

Your score on this task is the lowest score you get among all test cases.

SAMPLE GRADER

In order to test your program, you may download the sample grader files.

Language	Source Code Filename	Compilation Command	Execution Command
C++17	<code>rezero.cpp</code>	<code>./compile_cpp.sh</code>	<code>./rezero</code>

Please be advised that although the sample grader is intended to simulate the judging system, it is NOT the real judging system and might behave differently.

When testing your programs with the sample grader, your input should match the format and constraints from the task statement. Otherwise, unspecified behaviors may occur.

The sample grader reads the input in the following format:

- line 1: N
- next N lines: $Position_i \ X_i \ T_i$, where
 - $Position_i$ is the starting position of monster i .
 - X_i and T_i are the values of monster i mentioned above.

The sample grader prints your answers in the following format:

- first N lines: $X_i \ T_i$ as reported by `analyze_monsters`.
- line $N + 1$: the number of calls to `surrender`.