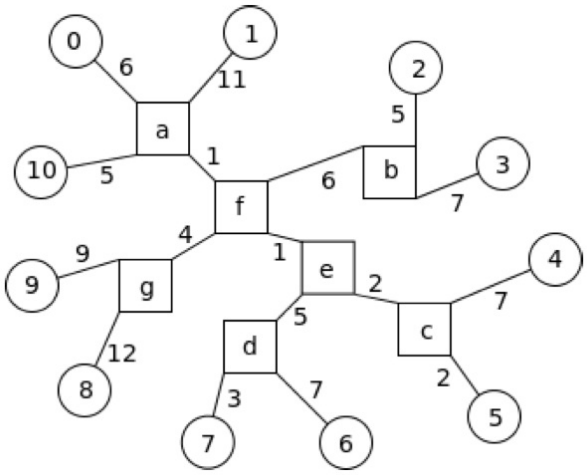
Towns

There are N small towns in Kazakhstan, numbered from 0 through $N-1$. There is also an unknown number of large cities. The small towns and large cities of Kazakhstan are jointly called *settlements*.

All the settlements of Kazakhstan are connected by a single network of bidirectional highways. Each highway connects two distinct settlements, and each pair of settlements is directly connected by at most one highway. For each pair of settlements a and b there is a unique way in which one can go from a to b using the highways, as long as no highway is used more than once.

It is known that each small town is directly connected to a single other settlement, and each large city is directly connected to three or more settlements.

The following figure shows a network of 11 small towns and 7 large cities. Small towns are depicted as circles and labeled by integers, large cities are depicted as squares and labeled by letters.



Every highway has a positive integer length. The distance between two settlements is the minimum sum of the lengths of the highways one needs to travel in order to get from one settlement to the other.

For each large city C we can measure the distance $r(C)$ to the small town that is the farthest away from that city. A large city C is a *hub* if the distance $r(C)$ is the smallest among all large cities. The distance between a hub and a small town that is farthest away from the hub will be denoted by R. Thus, R is the smallest of all values $r(C)$.

In the above example the farthest small town from city a is town 8, and the distance between them is $r(a) = 1 + 4 + 12 = 17$. For city g we also have $r(g) = 17$. (One of the small towns that are farthest away from g is town 6.) The only hub in the above example is city f, with $r(f) = 16$. Hence, in the above example R is 16.

Removing a hub divides the network into multiple connected pieces. A hub is *balanced* if each of

those pieces contains at most $\lfloor N/2 \rfloor$ small towns. (We stress that we do not count the large cities.) Note that $\lfloor x \rfloor$ denotes the largest integer which is not greater than x.

In our example, city f is a hub. If we remove city f, the network will break into four connected pieces. These four pieces consist of the following sets of small towns: $\{0, 1, 10\}$, $\{2, 3\}$, $\{4, 5, 6, 7\}$, and $\{8, 9\}$. None of these pieces has more than $\lfloor 11/2 \rfloor = 5$ small towns, hence city f is a balanced hub.

Task

Initially, the only information you have about the network of settlements and highways is the number N of small towns. You do not know the number of large cities. You also do not know anything about the layout of highways in the country. You can only obtain new information by asking queries about distances between pairs of small towns.

Your task is to determine:

- In all subtasks: the distance R.
- In subtasks 3 to 6: whether there is a balanced hub in the network.

You need to implement the function `hubDistance`. The grader will evaluate multiple test cases in a single run. The number of test cases per run is at most 40. For each test case the grader will call your function `hubDistance` exactly once. Make sure that your function initializes all necessary variables every time it is called.

- `hubDistance(N, sub)`
  - `N`: the number of small towns.
  - `sub`: the subtask number (explained in the Subtasks section).
  - If `sub` is 1 or 2, the function can return either R or $-R$
  - If `sub` is greater than 2, if there exists a balanced hub then the function must return R, otherwise it must return $-R$.

Your function `hubDistance` can obtain information about the network of highways by calling the grader function `getDistance(i, j)`. This function returns the distance between the small towns i and j. Note that if i and j are equal, the function returns 0. It also returns 0 when the arguments are invalid.

Subtasks

In each test case:

- N is between 6 and 110 inclusive.
- The distance between any two distinct small towns is between 1 and 1,000,000 inclusive.

The number of queries your program may make is limited. The limit varies by subtask, as given in the table below. If your program tries to exceed the limit on the number of queries, it will be terminated and it will be assumed to have given an incorrect answer.

subtask	points	number of queries	find balanced hub	additional constraints
1	13	$\frac{N(N-1)}{2}$	NO	none
2	12	$\lceil \frac{7N}{2} \rceil$	NO	none
3	13	$\frac{N(N-1)}{2}$	YES	none
4	10	$\lceil \frac{7N}{2} \rceil$	YES	each large city is connected to *exactly* three settlements
5	13	$5N$	YES	none
6	39	$\lceil \frac{7N}{2} \rceil$	YES	none

Note that $\lceil x \rceil$ denotes the smallest integer which is greater than or equal to x.

Sample grader

Note that the subtask number is a part of the input. The sample grader changes its behavior according to the subtask number.

The sample grader reads the input from file `towns.in` in the following format:

- line 1: Subtask number and the number of test cases.
- line 2: N_1, the number of small towns in the first test case.
- following N_1 lines: The j-th number $(1 \leq j \leq N_1)$ in the i-th of these lines $(1 \leq i \leq N_1)$ is the distance between small towns $i-1$ and $j-1$.
- The next test cases follow. They are given in the same format as the first test case.

For each test case, the sample grader prints the return value of `hubDistance` and the number of calls made on separate lines.

The input file corresponding to the example above is:

```
1 1
11
0 17 18 20 17 12 20 16 23 20 11
17 0 23 25 22 17 25 21 28 25 16
18 23 0 12 21 16 24 20 27 24 17
20 25 12 0 23 18 26 22 29 26 19
17 22 21 23 0 9 21 17 26 23 16
12 17 16 18 9 0 16 12 21 18 11
20 25 24 26 21 16 0 10 29 26 19
16 21 20 22 17 12 10 0 25 22 15
23 28 27 29 26 21 29 25 0 21 22
20 25 24 26 23 18 26 22 21 0 19
11 16 17 19 16 11 19 15 22 19 0
```

This format is quite different from specifying the list of highways. Note that you are allowed to modify sample graders, so that they use a different input format.

Sorting

Aizhan has a sequence of N integers $S[0], S[1], \ldots, S[N-1]$. The sequence consists of distinct numbers from 0 to $N-1$. She is trying to sort this sequence in ascending order by swapping some pairs of elements. Her friend Ermek is also going to swap some pairs of elements — not necessarily in a helpful way.

Ermek and Aizhan are going to modify the sequence in a series of rounds. In each round, first Ermek makes a swap and then Aizhan makes another swap. More precisely, the person making a swap chooses two valid indices and swaps the elements at those indices. Note that the two indices do not have to be distinct. If they are equal, the current person swaps an element with itself, which does not change the sequence.

Aizhan knows that Ermek does not actually care about sorting the sequence S. She also knows the exact indices Ermek is going to choose. Ermek plans to take part in M rounds of swapping. We number these rounds from 0 to $M-1$. For each i between 0 and $M-1$ inclusive, Ermek will choose the indices $X[i]$ and $Y[i]$ in round i.

Aizhan wants to sort the sequence S. Before each round, if Aizhan sees that the sequence is already sorted in ascending order, she will terminate the entire process. Given the original sequence S and the indices Ermek is going to choose, your task is to find a sequence of swaps, which Aizhan can use to sort the sequence S. In addition, in some subtasks you are required to find a sequence of swaps that is as short as possible. You may assume that it is possible to sort the sequence S in M or fewer rounds.

Note that if Aizhan sees that the sequence S is sorted after Ermek's swap, she can choose to swap two equal indices (e.g., 0 and 0). As a result the sequence S is also sorted after the entire round, so Aizhan reaches her goal. Also note that if the initial sequence S is already sorted, the minimal number of rounds needed to sort it is 0.

Example 1

Suppose that:

- The initial sequence is $S = 4, 3, 2, 1, 0$.
- Ermek is willing to make $M = 6$ swaps.
- The sequences X and Y that describe the indices Ermek is going to choose are $X = 0, 1, 2, 3, 0, 1$ and $Y = 1, 2, 3, 4, 1, 2$. In other words, the pairs of indices that Ermek plans to choose are $(0, 1)$, $(1, 2)$, $(2, 3)$, $(3, 4)$, $(0, 1)$, and $(1, 2)$.

In this setting Aizhan can sort the sequence S into the order $0, 1, 2, 3, 4$ in three rounds. She can do so by choosing the indices $(0, 4)$, $(1, 3)$, and then $(3, 4)$.

The following table shows how Ermek and Aizhan modify the sequence.

Round	Player	Pair of swapped indices	Sequence
beginning			$4, 3, 2, 1, 0$
0	Ermek	$(0, 1)$	$3, 4, 2, 1, 0$
0	Aizhan	$(0, 4)$	$0, 4, 2, 1, 3$
1	Ermek	$(1, 2)$	$0, 2, 4, 1, 3$
1	Aizhan	$(1, 3)$	$0, 1, 4, 2, 3$
2	Ermek	$(2, 3)$	$0, 1, 2, 4, 3$
2	Aizhan	$(3, 4)$	$0, 1, 2, 3, 4$

Example 2

Suppose that:

- The initial sequence is $S = 3, 0, 4, 2, 1$.
- Ermek is willing to make $M = 5$ swaps.
- The pairs of indices that Ermek plans to choose are $(1, 1)$, $(4, 0)$, $(2, 3)$, $(1, 4)$, and $(0, 4)$.

In this setting Aizhan can sort the sequence S in three rounds, for example by choosing the pairs of indices $(1, 4)$, $(4, 2)$, and then $(2, 2)$. The following table shows how Ermek and Aizhan modify the sequence.

Round	Player	Pair of swapped indices	Sequence
beginning			$3, 0, 4, 2, 1$
0	Ermek	$(1, 1)$	$3, 0, 4, 2, 1$
0	Aizhan	$(1, 4)$	$3, 1, 4, 2, 0$
1	Ermek	$(4, 0)$	$0, 1, 4, 2, 3$
1	Aizhan	$(4, 2)$	$0, 1, 3, 2, 4$
2	Ermek	$(2, 3)$	$0, 1, 2, 3, 4$
2	Aizhan	$(2, 2)$	$0, 1, 2, 3, 4$

Task

You are given the sequence S, the number M, and the sequences of indices X and Y. Compute a sequence of swaps, which Aizhan can use to sort the sequence S. In subtasks 5 and 6 the sequence of swaps you find has to be the shortest possible.

You need to implement the function `findSwapPairs`:

- `findSwapPairs(N, S, M, X, Y, P, Q)` — This function will be called by the grader exactly once.
  - `N`: the length of the sequence S.
  - `S`: an array of integers containing the initial sequence S.

- `M`: the number of swaps Ermek plans to make.
- `X, Y`: arrays of integers of length M. For $0 \le i \le M-1$, in round i Ermek plans to swap numbers at indices $X[i]$ and $Y[i]$.
- `P, Q`: arrays of integers. Use these arrays to report one possible sequence of swaps Aizhan can make to sort the sequence S. Denote by R the length of the sequence of swaps that your program has found. For each i between 0 and $R-1$ inclusive, the indices Aizhan should choose in round i should be stored into $P[i]$ and $Q[i]$. You may assume that the arrays `P` and `Q` have already been allocated to M elements each.
- This function should return the value of R (defined above).

Subtasks

subtask	points	N	M	extra constraints on X, Y	requirement on R
1	8	$1 \le N \le 5$	$M = N^2$	$X[i] = Y[i] = 0$ for all i	$R \le M$
2	12	$1 \le N \le 100$	$M = 30N$	$X[i] = Y[i] = 0$ for all i	$R \le M$
3	16	$1 \le N \le 100$	$M = 30N$	$X[i] = 0, Y[i] = 1$ for all i	$R \le M$
4	18	$1 \le N \le 500$	$M = 30N$	none	$R \le M$
5	20	$6 \le N \le 2,000$	$M = 3N$	none	minimum possible
6	26	$6 \le N \le 200,000$	$M = 3N$	none	minimum possible

You may assume that there exists a solution that requires M or fewer rounds.

Sample grader

The sample grader reads the input from the file `sorting.in` in the following format:

- line 1: `N`
- line 2: `S[0] ... S[N - 1]`
- line 3: `M`
- lines 4, …, `M` + 3: `X[i] Y[i]`

The sample grader prints the following output:

- line 1: the return value R of `findSwapPairs`
- line 2+i, for $0 \le i < R$: `P[i] Q[i]`

Horses

Mansur loves to breed horses, just like his ancient ancestors did. He now has the largest herd in Kazakhstan. But this was not always the case. N years ago, Mansur was just a dzhigit (Kazakh for *a young man*) and he only had a single horse. He dreamed to make a lot of money and to finally become a bai (Kazakh for *a very rich person*).

Let us number the years from 0 to $N-1$ in chronological order (i.e., year $N-1$ is the most recent one). The weather of each year influenced the growth of the herd. For each year i, Mansur remembers a positive integer growth coefficient $X[i]$. If you started year i with h horses, you ended the year with $h \cdot X[i]$ horses in your herd.

Horses could only be sold at the end of a year. For each year i, Mansur remembers a positive integer $Y[i]$: the price for which he could sell a horse at the end of year i. After each year, it was possible to sell arbitrarily many horses, each at the same price $Y[i]$.

Mansur wonders what is the largest amount of money he could have now if he had chosen the best moments to sell his horses during the N years. You have the honor of being a guest on Mansur's toi (Kazakh for *holiday*), and he asked you to answer this question.

Mansur's memory improves throughout the evening, and so he makes a sequence of M updates. Each update will change either one of the values $X[i]$ or one of the values $Y[i]$. After each update he again asks you the largest amount of money he could have earned by selling his horses. Mansur's updates are cumulative: each of your answers should take into account all of the previous updates. Note that a single $X[i]$ or $Y[i]$ may be updated multiple times.

The actual answers to Mansur's questions can be huge. In order to avoid working with large numbers, you are only required to report the answers modulo $10^9 + 7$.

Example

Suppose that there are $N = 3$ years, with the following information:

	0	**1**	**2**
X	2	1	3
Y	3	4	1

For these initial values, Mansur can earn the most if he sells both his horses at the end of year 1. The entire process will look as follows:

- Initially, Mansur has 1 horse.
- After year 0 he will have $1 \cdot X[0] = 2$ horses.

- After year 1 he will have $2 \cdot X[1] = 2$ horses.
- He can now sell those two horses. The total profit will be $2 \cdot Y[1] = 8$.

Then, suppose that there is $M = 1$ update: changing $Y[1]$ to 2.

After the update we will have:

	0	1	2
X	2	1	3
Y	3	2	1

In this case, one of the optimal solutions is to sell one horse after year 0 and then three horses after year 2. The entire process will look as follows:

- Initially, Mansur has 1 horse.
- After year 0 he will have $1 \cdot X[0] = 2$ horses.
- He can now sell one of those horses for $Y[0] = 3$, and have one horse left.
- After year 1 he will have $1 \cdot X[1] = 1$ horse.
- After year 2 he will have $1 \cdot X[2] = 3$ horses.
- He can now sell those three horses for $3 \cdot Y[2] = 3$. The total amount of money is $3 + 3 = 6$.

Task

You are given N, X, Y, and the list of updates. Before the first update, and after every update, compute the maximal amount of money that Mansur could get for his horses, modulo $10^9 + 7$. You need to implement the functions `init`, `updateX`, and `updateY`.

- `init(N, X, Y)` — The grader will call this function first and exactly once.
  - `N`: the number of years.
  - `X`: an array of length N. For $0 \leq i \leq N - 1$, $X[i]$ gives the growth coefficient for year i.
  - `Y`: an array of length N. For $0 \leq i \leq N - 1$, $Y[i]$ gives the price of a horse after year i.
  - Note that both `X` and `Y` specify the initial values given by Mansur (before any updates).
  - After `init` terminates, the arrays `X` and `Y` remain valid, and you may modify their contents if you wish.
  - The function should return the maximal amount of money Mansur could get for these initial values of X and Y, modulo $10^9 + 7$.
- `updateX(pos, val)`
  - `pos`: an integer from the range $0, \ldots, N - 1$.

- `val`: the new value for $X[\texttt{pos}]$.
- The function should return the maximal amount of money Mansur could get after this update, modulo $10^9 + 7$.

- `updateY(pos, val)`
  - `pos`: an integer from the range $0, \ldots, N - 1$.
  - `val`: the new value for $Y[\texttt{pos}]$.
  - The function should return the maximal amount of money Mansur could get after this update, modulo $10^9 + 7$.

You may assume that all the initial, as well as updated values of $X[i]$ and $Y[i]$ are between 1 and 10^9 inclusive.
After calling `init`, the grader will call `updateX` and `updateY` several times. The total number of calls to `updateX` and `updateY` will be M.

Subtasks

subtask	points	N	M	additional constraints
1	17	$1 \le N \le 10$	$M = 0$	$X[i], Y[i] \le 10$, $X[0] \cdot X[1] \cdot \ldots \cdot X[N-1] \le 1,000$
2	17	$1 \le N \le 1,000$	$0 \le M \le 1,000$	none
3	20	$1 \le N \le 500,000$	$0 \le M \le 100,000$	$X[i] \ge 2$ and $val \ge 2$ for `init` and `updateX` correspondingly
4	23	$1 \le N \le 500,000$	$0 \le M \le 10,000$	none
5	23	$1 \le N \le 500,000$	$0 \le M \le 100,000$	none

Sample grader

The sample grader reads the input from the file `horses.in` in the following format:

- line 1: `N`
- line 2: `X[0] ... X[N - 1]`
- line 3: `Y[0] ... Y[N - 1]`
- line 4: `M`
- lines 5, …, `M` + 4: three numbers `type pos val` (`type=1` for `updateX` and `type=2` for `updateY`).

The sample grader prints the return value of `init` followed by the return values of all calls to `updateX` and `updateY`.