## A. Yet Another Dividing into Teams

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are a coach of a group consisting of $n$ students. The $i$-th student has programming skill $a_i$. **All students have distinct programming skills**. You want to divide them into teams in such a way that:

- No two students $i$ and $j$ such that $|a_i - a_j| = 1$ belong to the same team (i.e. skills of each pair of students in the same team have the difference strictly greater than $1$);
- the number of teams is the minimum possible.

You have to answer $q$ independent queries.

### Input

The first line of the input contains one integer $q$ ($1 \le q \le 100$) — the number of queries. Then $q$ queries follow.

The first line of the query contains one integer $n$ ($1 \le n \le 100$) — the number of students in the query. The second line of the query contains $n$ integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 100$, all $a_i$ are distinct), where $a_i$ is the programming skill of the $i$-th student.

### Output

For each query, print the answer on it — the minimum number of teams you can form if no two students $i$ and $j$ such that $|a_i - a_j| = 1$ may belong to the same team (i.e. skills of each pair of students in the same team has the difference strictly greater than $1$)

### Example

**input**

```
4
4
2 10 1 20
2
3 6
5
2 3 4 99 100
1
42
```

**output**

```
2
1
2
1
```

### Note

In the first query of the example, there are $n = 4$ students with the skills $a = [2, 10, 1, 20]$. There is only one restriction here: the $1$-st and the $3$-th students can't be in the same team (because of $|a_1 - a_3| = |2 - 1| = 1$). It is possible to divide them into $2$ teams: for example, students $1$, $2$ and $4$ are in the first team and the student $3$ in the second team.

In the second query of the example, there are $n = 2$ students with the skills $a = [3, 6]$. It is possible to compose just a single team containing both students.

## B1. Books Exchange (easy version)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between easy and hard versions is constraints**.

There are $n$ kids, each of them is reading a unique book. At the end of any day, the $i$-th kid will give his book to the $p_i$-th kid (in case of $i = p_i$ the kid will give his book to himself). It is guaranteed that all values of $p_i$ are distinct integers from $1$ to $n$ (i.e. $p$ is a permutation). The sequence $p$ doesn't change from day to day, it is fixed.

For example, if $n = 6$ and $p = [4, 6, 1, 3, 5, 2]$ then at the end of the first day the book of the $1$-st kid will belong to the $4$-th kid, the $2$-nd kid will belong to the $6$-th kid and so on. At the end of the second day the book of the $1$-st kid will belong to the $3$-th kid, the $2$-nd kid will belong to the $2$-th kid and so on.

Your task is to determine the number of the day the book of the $i$-th child is returned back to him for the first time for every $i$ from $1$ to $n$.

Consider the following example: $p = [5, 1, 2, 4, 3]$. The book of the $1$-st kid will be passed to the following kids:

- after the $1$-st day it will belong to the $5$-th kid,
- after the $2$-nd day it will belong to the $3$-rd kid,
- after the $3$-rd day it will belong to the $2$-nd kid,
- after the $4$-th day it will belong to the $1$-st kid.

So after the fourth day, the book of the first kid will return to its owner. The book of the fourth kid will return to him for the first time after exactly one day.

You have to answer $q$ independent queries.

### Input

The first line of the input contains one integer $q$ ($1 \le q \le 200$) — the number of queries. Then $q$ queries follow.

The first line of the query contains one integer $n$ ($1 \le n \le 200$) — the number of kids in the query. The second line of the query contains $n$ integers $p_1, p_2, \dots, p_n$ ($1 \le p_i \le n$, all $p_i$ are distinct, i.e. $p$ is a permutation), where $p_i$ is the kid which will get the book of the $i$-th kid.

### Output

For each query, print the answer on it: $n$ integers $a_1, a_2, \dots, a_n$, where $a_i$ is the number of the day the book of the $i$-th child is returned back to him for the first time in this query.

**Example**

**input**

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

**output**

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

## B2. Books Exchange (hard version)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between easy and hard versions is constraints**.

There are $n$ kids, each of them is reading a unique book. At the end of any day, the $i$-th kid will give his book to the $p_i$-th kid (in case of $i = p_i$ the kid will give his book to himself). It is guaranteed that all values of $p_i$ are distinct integers from $1$ to $n$ (i.e. $p$ is a permutation). The sequence $p$ doesn't change from day to day, it is fixed.

For example, if $n = 6$ and $p = [4, 6, 1, 3, 5, 2]$ then at the end of the first day the book of the $1$-st kid will belong to the $4$-th kid, the $2$-nd kid will belong to the $6$-th kid and so on. At the end of the second day the book of the $1$-st kid will belong to the $3$-th kid, the $2$-nd kid will belong to the $2$-th kid and so on.

Your task is to determine the number of the day the book of the $i$-th child is returned back to him for the first time for every $i$ from $1$ to $n$.

Consider the following example: $p = [5, 1, 2, 4, 3]$. The book of the $1$-st kid will be passed to the following kids:

- after the $1$-st day it will belong to the $5$-th kid,
- after the $2$-nd day it will belong to the $3$-rd kid,
- after the $3$-rd day it will belong to the $2$-nd kid,
- after the $4$-th day it will belong to the $1$-st kid.

So after the fourth day, the book of the first kid will return to its owner. The book of the fourth kid will return to him for the first time after exactly one day.

You have to answer $q$ independent queries.

### Input

The first line of the input contains one integer $q$ ($1 \le q \le 1000$) — the number of queries. Then $q$ queries follow.

The first line of the query contains one integer $n$ ($1 \le n \le 2 \cdot 10^5$) — the number of kids in the query. The second line of the query contains $n$ integers $p_1, p_2, \dots, p_n$ ($1 \le p_i \le n$, all $p_i$ are distinct, i.e. $p$ is a permutation), where $p_i$ is the kid which will get the book of the $i$-th kid.

It is guaranteed that $\sum n \le 2 \cdot 10^5$ (sum of $n$ over all queries does not exceed $2 \cdot 10^5$).

### Output

For each query, print the answer on it: $n$ integers $a_1, a_2, \dots, a_n$, where $a_i$ is the number of the day the book of the $i$-th child is returned back to him for the first time in this query.

**Example**

**input**

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

**output**

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

## C1. Good Numbers (easy version)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between easy and hard versions is the maximum value of $n$.**

You are given a positive integer number $n$. You really love *good numbers* so you want to find the smallest *good number* greater than or equal to $n$.

The positive integer is called *good* if it can be represented as a sum of **distinct** powers of $3$ (i.e. no duplicates of powers of $3$ are allowed).

For example:

- $30$ is a *good number*: $30 = 3^3 + 3^1$,
- $1$ is a *good number*: $1 = 3^0$,
- $12$ is a *good number*: $12 = 3^2 + 3^1$,
- but $2$ is **not** a *good number*: you can't represent it as a sum of distinct powers of $3$ ($2 = 3^0 + 3^0$),
- $19$ is **not** a *good number*: you can't represent it as a sum of distinct powers of $3$ (for example, the representations $19 = 3^2 + 3^2 + 3^0 = 3^2 + 3^1 + 3^1 + 3^1 + 3^0$ are invalid),
- $20$ is also **not** a *good number*: you can't represent it as a sum of distinct powers of $3$ (for example, the representation $20 = 3^2 + 3^2 + 3^0 + 3^0$ is invalid).

Note, that there exist other representations of $19$ and $20$ as sums of powers of $3$ but none of them consists of **distinct** powers of $3$.

For the given positive integer $n$ find such smallest $m$ ($n \le m$) that $m$ is a *good number*.

You have to answer $q$ independent queries.

### Input

The first line of the input contains one integer $q$ ($1 \le q \le 500$) — the number of queries. Then $q$ queries follow.

The only line of the query contains one integer $n$ ($1 \le n \le 10^4$).

### Output

For each query, print such smallest integer $m$ (where $n \le m$) that $m$ is a *good number*.

**Example**

**input**

```
7
1
2
6
13
14
3620
10000
```

**output**

```
1
3
9
13
27
6561
19683
```

## C2. Good Numbers (hard version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between easy and hard versions is the maximum value of $n$.**

You are given a positive integer number $n$. You really love *good numbers* so you want to find the smallest *good number* greater than or equal to $n$.

The positive integer is called *good* if it can be represented as a sum of **distinct** powers of $3$ (i.e. no duplicates of powers of $3$ are allowed).

For example:

- $30$ is a *good number*: $30 = 3^3 + 3^1$,
- $1$ is a *good number*: $1 = 3^0$,
- $12$ is a *good number*: $12 = 3^2 + 3^1$,
- but $2$ is **not** a *good number*: you can't represent it as a sum of distinct powers of $3$ ($2 = 3^0 + 3^0$),
- $19$ is **not** a *good number*: you can't represent it as a sum of distinct powers of $3$ (for example, the representations $19 = 3^2 + 3^2 + 3^0 = 3^2 + 3^1 + 3^1 + 3^1 + 3^0$ are invalid),
- $20$ is also **not** a *good number*: you can't represent it as a sum of distinct powers of $3$ (for example, the representation $20 = 3^2 + 3^2 + 3^0 + 3^0$ is invalid).

Note, that there exist other representations of $19$ and $20$ as sums of powers of $3$ but none of them consists of **distinct** powers of $3$.

For the given positive integer $n$ find such smallest $m$ ($n \leq m$) that $m$ is a *good number*.

You have to answer $q$ independent queries.

### Input

The first line of the input contains one integer $q$ ($1 \leq q \leq 500$) — the number of queries. Then $q$ queries follow.

The only line of the query contains one integer $n$ ($1 \leq n \leq 10^{18}$).

### Output

For each query, print such smallest integer $m$ (where $n \leq m$) that $m$ is a *good number*.

**Example**

**input**

```
8
1
2
6
13
14
3620
10000
1000000000000000000
```

**output**

```
1
3
9
13
27
6561
19683
```

```
1350851717672992089
```

## D1. Too Many Segments (easy version)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between easy and hard versions is constraints**.

You are given $n$ segments on the coordinate axis $OX$. Segments can intersect, lie inside each other and even coincide. The $i$-th segment is $[l_i; r_i]$ ($l_i \le r_i$) and it covers all integer points $j$ such that $l_i \le j \le r_i$.

The integer point is called **bad** if it is covered by **strictly more** than $k$ segments.

Your task is to remove the minimum number of segments so that there are no **bad** points at all.

### Input

The first line of the input contains two integers $n$ and $k$ ($1 \le k \le n \le 200$) — the number of segments and the maximum number of segments by which each integer point can be covered.

The next $n$ lines contain segments. The $i$-th line contains two integers $l_i$ and $r_i$ ($1 \le l_i \le r_i \le 200$) — the endpoints of the $i$-th segment.

### Output

In the first line print one integer $m$ ($0 \le m \le n$) — the minimum number of segments you need to remove so that there are no **bad** points.

In the second line print $m$ **distinct** integers $p_1, p_2, \dots, p_m$ ($1 \le p_i \le n$) — indices of segments you remove in any order. If there are multiple answers, you can print any of them.

#### Examples

**input**

```
7 2
11 11
9 11
7 8
8 9
7 8
9 11
7 9
```

**output**

```
3
1 4 7
```

**input**

```
5 1
29 30
30 30
29 29
28 30
30 30
```

**output**

```
3
1 2 4
```

**input**

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

**output**

```
4
1 3 5 6
```

## D2. Too Many Segments (hard version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

**The only difference between easy and hard versions is constraints**.

You are given $n$ segments on the coordinate axis $OX$. Segments can intersect, lie inside each other and even coincide. The $i$-th segment is $[l_i; r_i]$ ($l_i \le r_i$) and it covers all integer points $j$ such that $l_i \le j \le r_i$.

The integer point is called **bad** if it is covered by **strictly more** than $k$ segments.

Your task is to remove the minimum number of segments so that there are no **bad** points at all.

## Input

The first line of the input contains two integers $n$ and $k$ ($1 \le k \le n \le 2 \cdot 10^5$) — the number of segments and the maximum number of segments by which each integer point can be covered.

The next $n$ lines contain segments. The $i$-th line contains two integers $l_i$ and $r_i$ ($1 \le l_i \le r_i \le 2 \cdot 10^5$) — the endpoints of the $i$-th segment.

## Output

In the first line print one integer $m$ ($0 \le m \le n$) — the minimum number of segments you need to remove so that there are no **bad** points.

In the second line print $m$ **distinct** integers $p_1, p_2, \dots, p_m$ ($1 \le p_i \le n$) — indices of segments you remove in any order. If there are multiple answers, you can print any of them.

**Examples**

**input**

```
7 2
11 11
9 11
7 8
8 9
7 8
9 11
7 9
```

**output**

```
3
4 6 7
```

**input**

```
5 1
29 30
30 30
29 29
28 30
30 30
```

**output**

```
3
1 4 5
```

**input**

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

**output**

```
4
1 3 5 6
```

# E. By Elevator or Stairs?

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are planning to buy an apartment in a $n$-floor building. The floors are numbered from $1$ to $n$ from the bottom to the top. At first for each floor you want to know the minimum total time to reach it from the first (the bottom) floor.

Let:

- $a_i$ for all $i$ from $1$ to $n-1$ be the time required to go from the $i$-th floor to the $(i+1)$-th one (and from the $(i+1)$-th to the $i$-th as well) using the **stairs**;
- $b_i$ for all $i$ from $1$ to $n-1$ be the time required to go from the $i$-th floor to the $(i+1)$-th one (and from the $(i+1)$-th to the $i$-th as well) using the **elevator**, also there is a value $c$ — time overhead for elevator usage (you need to wait for it, the elevator

doors are too slow!).

In one **move**, you can go from the floor you are staying at $x$ to any floor $y$ ($x \neq y$) in two different ways:

- If you are using the stairs, just sum up the corresponding values of $a_i$. Formally, it will take $\sum\limits_{i=min(x,y)}^{max(x,y)-1} a_i$ time units.
- If you are using the elevator, just sum up $c$ and the corresponding values of $b_i$. Formally, it will take $c + \sum\limits_{i=min(x,y)}^{max(x,y)-1} b_i$ time units.

You can perform as many **moves** as you want (possibly zero).

So your task is for each $i$ to determine the minimum total time it takes to reach the $i$-th floor from the $1$-st (bottom) floor.

### Input

The first line of the input contains two integers $n$ and $c$ ($2 \le n \le 2 \cdot 10^5, 1 \le c \le 1000$) — the number of floors in the building and the time overhead for the elevator rides.

The second line of the input contains $n-1$ integers $a_1, a_2, \dots, a_{n-1}$ ($1 \le a_i \le 1000$), where $a_i$ is the time required to go from the $i$-th floor to the $(i+1)$-th one (and from the $(i+1)$-th to the $i$-th as well) using the stairs.

The third line of the input contains $n-1$ integers $b_1, b_2, \dots, b_{n-1}$ ($1 \le b_i \le 1000$), where $b_i$ is the time required to go from the $i$-th floor to the $(i+1)$-th one (and from the $(i+1)$-th to the $i$-th as well) using the elevator.

### Output

Print $n$ integers $t_1, t_2, \dots, t_n$, where $t_i$ is the minimum total time to reach the $i$-th floor from the first floor if you can perform as many **moves** as you want.

#### Examples

**input**

```
10 2
7 6 18 6 16 18 1 17 17
6 9 3 10 9 1 10 1 5
```

**output**

```
0 7 13 18 24 35 36 37 40 45
```

**input**

```
10 1
3 2 3 1 3 3 1 4 1
1 2 3 4 4 1 2 1 3
```

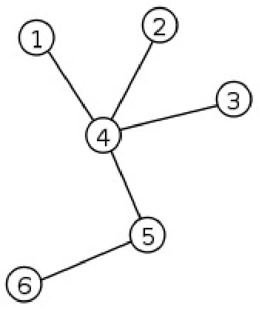
**output**

```
0 2 4 7 8 11 13 14 16 17
```

## F. Maximum Weight Subset

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a tree, which consists of $n$ vertices. Recall that a tree is a connected undirected graph without cycles.



Example of a tree.

Vertices are numbered from $1$ to $n$. All vertices have weights, the weight of the vertex $v$ is $a_v$.

Recall that the distance between two vertices in the tree is the number of edges on a simple path between them.

Your task is to find the subset of vertices with the maximum total weight (the weight of the subset is the sum of weights of all vertices in it) such that there is no pair of vertices with the distance $k$ or less between them in this subset.

### Input

The first line of the input contains two integers $n$ and $k$ ($1 \le n, k \le 200$) — the number of vertices in the tree and the distance restriction, respectively.

The second line of the input contains $n$ integers $a_1, a_2, \dots, a_n$ $(1 \le a_i \le 10^5)$, where $a_i$ is the weight of the vertex $i$.

The next $n-1$ lines contain edges of the tree. Edge $i$ is denoted by two integers $u_i$ and $v_i$ — the labels of vertices it connects ( $1 \le u_i, v_i \le n$, $u_i \ne v_i$).

It is guaranteed that the given edges form a tree.

## Output

Print one integer — the maximum total weight of the subset in which all pairs of vertices have distance more than $k$.

### Examples

**input**

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

**output**

```
11
```

**input**

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

**output**

```
4
```