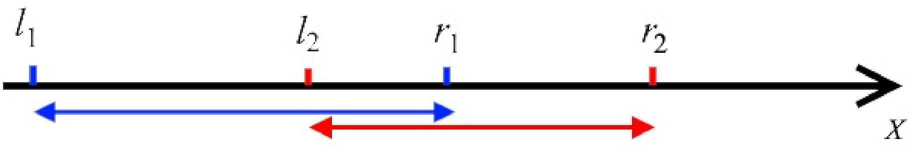
A. Two distinct points

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

You are given two segments $[l_1; r_1]$ and $[l_2; r_2]$ on the x-axis. It is guaranteed that $l_1 < r_1$ and $l_2 < r_2$. Segments **may intersect, overlap or even coincide with each other**.



The example of two segments on the x-axis.

Your problem is to find two **integers** a and b such that $l_1 \le a \le r_1$, $l_2 \le b \le r_2$ and $a \ne b$. In other words, you have to choose two **distinct** integer points in such a way that the first point belongs to the segment $[l_1; r_1]$ and the second one belongs to the segment $[l_2; r_2]$.

It is guaranteed that **the answer exists**. If there are multiple answers, you can print **any** of them.

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.

Each of the next q lines contains four integers $l_{1_i}, r_{1_i}, l_{2_i}$ and r_{2_i} $(1 \le l_{1_i}, r_{1_i}, l_{2_i}, r_{2_i} \le 10^9, l_{1_i} < r_{1_i}, l_{2_i} < r_{2_i})$ — the ends of the segments in the i-th query.

Output

Print $2q$ integers. For the i-th query print two integers a_i and b_i — such numbers that $l_{1_i} \le a_i \le r_{1_i}$, $l_{2_i} \le b_i \le r_{2_i}$ and $a_i \ne b_i$. Queries are numbered in order of the input.

It is guaranteed that **the answer exists**. If there are multiple answers, you can print **any**.

Example

input

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

output

```
2 1
3 4
3 2
1 2
3 7
```

B. Divisors of Two Integers

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

Recently you have received two **positive** integer numbers x and y. You forgot them, but you remembered a **shuffled** list containing all divisors of x (including 1 and x) and all divisors of y (including 1 and y). If d is a divisor of both numbers x and y at the same time, there are two occurrences of d in the list.

For example, if $x = 4$ and $y = 6$ then the given list can be any permutation of the list $[1, 2, 4, 1, 2, 3, 6]$. Some of the possible lists are: $[1, 1, 2, 4, 6, 3, 2]$, $[4, 6, 1, 1, 2, 3, 2]$ or $[1, 6, 3, 2, 4, 1, 2]$.

Your problem is to restore suitable **positive** integer numbers x and y that would yield the same list of divisors (possibly in different order).

It is guaranteed that the answer exists, i.e. the given list of divisors corresponds to some **positive** integers x and y.

Input

The first line contains one integer n ($2 \le n \le 128$) — the number of divisors of x and y.

The second line of the input contains n integers $d_1, d_2, \ldots, d_n$ ($1 \le d_i \le 10^4$), where d_i is either divisor of x or divisor of y. If a number is divisor of both numbers x and y then there are two copies of this number in the list.

Output

Print two **positive** integer numbers x and y — such numbers that merged list of their divisors is the permutation of the given list of integers. It is guaranteed that the answer exists.

Example

input

```
10
10 2 8 1 2 4 1 20 4 5
```

output

```
20 8
```

C. Nice Garland

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

You have a garland consisting of n lamps. Each lamp is colored red, green or blue. The color of the i-th lamp is s_i ('R', 'G' and 'B' — colors of lamps in the garland).

You have to recolor some lamps in this garland (recoloring a lamp means changing its initial color to another) in such a way that the obtained garland is **nice**.

A garland is called **nice** if any two lamps of the same color have distance divisible by three between them. I.e. if the obtained garland is t, then for each i, j such that $t_i = t_j$ should be satisfied $|i-j|~mod~3 = 0$. The value $|x|$ means absolute value of x, the operation $x~mod~y$ means remainder of x when divided by y.

For example, the following garlands are **nice**: "RGBRGBRG", "GB", "R", "GRBGRBG", "BRGBRGB". The following garlands are not **nice**: "RR", "RGBG".

Among all ways to recolor the initial garland to make it **nice** you have to choose one with the **minimum** number of recolored lamps. If there are multiple optimal solutions, print **any** of them.

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of lamps.

The second line of the input contains the string s consisting of n characters 'R', 'G' and 'B' — colors of lamps in the garland.

Output

In the first line of the output print one integer r — the **minimum** number of recolors needed to obtain a **nice** garland from the given one.

In the second line of the output print one string t of length n — a **nice** garland obtained from the initial one with **minimum** number of recolors. If there are multiple optimal solutions, print **any** of them.

Examples

input

```
3
BRB
```

output

```
1
GRB
```

input

```
7
RGBGRBB
```

output

```
3
RGBRGBR
```

D. Diverse Garland

time limit per test: 1 second

memory limit per test: 256 megabytes
input: standard input
output: standard output

You have a garland consisting of n lamps. Each lamp is colored red, green or blue. The color of the i-th lamp is s_i ('R', 'G' and 'B' — colors of lamps in the garland).

You have to recolor some lamps in this garland (recoloring a lamp means changing its initial color to another) in such a way that the obtained garland is **diverse**.

A garland is called **diverse** if any two adjacent (consecutive) lamps (i. e. such lamps that the distance between their positions is 1) have distinct colors.

In other words, if the obtained garland is t then for each i from 1 to $n-1$ the condition $t_i \neq t_{i+1}$ should be satisfied.

Among all ways to recolor the initial garland to make it **diverse** you have to choose one with the **minimum** number of recolored lamps. If there are multiple optimal solutions, print **any** of them.

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of lamps.

The second line of the input contains the string s consisting of n characters 'R', 'G' and 'B' — colors of lamps in the garland.

Output

In the first line of the output print one integer r — the **minimum** number of recolors needed to obtain a **diverse** garland from the given one.

In the second line of the output print one string t of length n — a **diverse** garland obtained from the initial one with **minimum** number of recolors. If there are multiple optimal solutions, print **any** of them.

Examples

input

```
9
RBGRRBRGG
```

output

```
2
RBGRGBRGR
```

input

```
8
BBBGBRRR
```

output

```
2
BRBGBRGR
```

input

```
13
BBRRRRGGGGGRR
```

output

```
6
BGRBRBGBGBGRG
```

E1. Array and Segments (Easy 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 a number of elements in the array.

You are given an array a consisting of n integers. The value of the i-th element of the array is a_i.

You are also given a set of m segments. The j-th segment is $[l_j; r_j]$, where $1 \le l_j \le r_j \le n$.

You can choose some subset of the given set of segments and decrease values on each of the chosen segments by one (**independently**). For example, if the initial array $a = [0, 0, 0, 0, 0]$ and the given segments are $[1; 3]$ and $[2; 4]$ then you can choose both of them and the array will become $b = [-1, -2, -2, -1, 0]$.

You have to choose some subset of the given segments (**each segment can be chosen at most once**) in such a way that if you apply this subset of segments to the array a and obtain the array b then the value $\max\limits_{i=1}^{n} b_i - \min\limits_{i=1}^{n} b_i$ will be **maximum** possible.

Note that **you can choose the empty set**.

If there are multiple answers, you can print **any**.

If you are Python programmer, consider using PyPy instead of Python when you submit your code.

Input

The first line of the input contains two integers n and m ($1 \le n \le 300, 0 \le m \le 300$) — the length of the array a and the number of segments, respectively.

The second line of the input contains n integers $a_1, a_2, \dots, a_n$ ($-10^6 \le a_i \le 10^6$), where a_i is the value of the i-th element of the array a.

The next m lines are contain two integers each. The j-th of them contains two integers l_j and r_j ($1 \le l_j \le r_j \le n$), where l_j and r_j are the ends of the j-th segment.

Output

In the first line of the output print one integer d — the **maximum** possible value $\max\limits_{i=1}^{n} b_i - \min\limits_{i=1}^{n} b_i$ if b is the array obtained by applying some subset of the given segments to the array a.

In the second line of the output print one integer q ($0 \le q \le m$) — the number of segments you apply.

In the third line print q distinct integers $c_1, c_2, \dots, c_q$ in **any order** ($1 \le c_k \le m$) — indices of segments you apply to the array a in such a way that the value $\max\limits_{i=1}^{n} b_i - \min\limits_{i=1}^{n} b_i$ of the obtained array b is **maximum** possible.

If there are multiple answers, you can print **any**.

Examples

input

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

output

```
6
2
1 4
```

input

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

output

```
7
2
3 2
```

input

```
1 0
1000000
```

output

```
0
0
```

Note

In the first example the obtained array b will be $[0, -4, 1, 1, 2]$ so the answer is 6.

In the second example the obtained array b will be $[2, -3, 1, -1, 4]$ so the answer is 7.

In the third example you cannot do anything so the answer is 0.

E2. Array and 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 a number of elements in the array.

You are given an array a consisting of n integers. The value of the i-th element of the array is a_i.

You are also given a set of m segments. The j-th segment is $[l_j; r_j]$, where $1 \le l_j \le r_j \le n$.

You can choose some subset of the given set of segments and decrease values on each of the chosen segments by one (**independently**). For example, if the initial array $a = [0, 0, 0, 0, 0]$ and the given segments are $[1; 3]$ and $[2; 4]$ then you can choose both of them and the array will become $b = [-1, -2, -2, -1, 0]$.

You have to choose some subset of the given segments (**each segment can be chosen at most once**) in such a way that if you apply this subset of segments to the array a and obtain the array b then the value $\max\limits_{i=1}^{n} b_i - \min\limits_{i=1}^{n} b_i$ will be **maximum** possible.

Note that **you can choose the empty set**.

If there are multiple answers, you can print **any**.

If you are Python programmer, consider using PyPy instead of Python when you submit your code.

Input

The first line of the input contains two integers n and m ($1 \le n \le 10^5, 0 \le m \le 300$) — the length of the array a and the number of segments, respectively.

The second line of the input contains n integers $a_1, a_2, \dots, a_n$ ($-10^6 \le a_i \le 10^6$), where a_i is the value of the i-th element of the array a.

The next m lines are contain two integers each. The j-th of them contains two integers l_j and r_j ($1 \le l_j \le r_j \le n$), where l_j and r_j are the ends of the j-th segment.

Output

In the first line of the output print one integer d — the **maximum** possible value $\max\limits_{i=1}^{n} b_i - \min\limits_{i=1}^{n} b_i$ if b is the array obtained by applying some subset of the given segments to the array a.

In the second line of the output print one integer q ($0 \le q \le m$) — the number of segments you apply.

In the third line print q distinct integers $c_1, c_2, \dots, c_q$ in **any order** ($1 \le c_k \le m$) — indices of segments you apply to the array a in such a way that the value $\max\limits_{i=1}^{n} b_i - \min\limits_{i=1}^{n} b_i$ of the obtained array b is **maximum** possible.

If there are multiple answers, you can print **any**.

Examples

input

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

output

```
6
2
4 1
```

input

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

output

```
7
2
3 2
```

input

```
1 0
1000000
```

output

```
0
0
```

Note

In the first example the obtained array b will be $[0, -4, 1, 1, 2]$ so the answer is 6.

In the second example the obtained array b will be $[2, -3, 1, -1, 4]$ so the answer is 7.

In the third example you cannot do anything so the answer is 0.

F. MST Unification

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an undirected weighted **connected** graph with n vertices and m edges **without loops and multiple edges**.

The i-th edge is $e_i = (u_i, v_i, w_i)$; the distance between vertices u_i and v_i along the edge e_i is w_i ($1 \le w_i$). The graph is **connected**, i. e. for any pair of vertices, there is at least one path between them consisting only of edges of the given graph.

A minimum spanning tree (MST) in case of **positive** weights is a subset of the edges of a connected weighted undirected graph that connects all the vertices together and has minimum total cost among all such subsets (total cost is the sum of costs of chosen edges).

You can modify the given graph. The only operation you can perform is the following: increase the weight of some edge by 1. You **can** increase the weight of each edge multiple (possibly, zero) times.

Suppose that the initial MST cost is k. Your problem is to increase weights of some edges **with minimum possible number of operations** in such a way that the cost of MST in the obtained graph remains k, but MST is **unique** (it means that there is only one way to choose MST in the obtained graph).

Your problem is to calculate the **minimum** number of operations required to do it.

Input

The first line of the input contains two integers n and m ($1 \le n \le 2 \cdot 10^5, n - 1 \le m \le 2 \cdot 10^5$) — the number of vertices and the number of edges in the initial graph.

The next m lines contain three integers each. The i-th line contains the description of the i-th edge e_i. It is denoted by three integers u_i, v_i and w_i ($1 \le u_i, v_i \le n, u_i \ne v_i, 1 \le w \le 10^9$), where u_i and v_i are vertices connected by the i-th edge and w_i is the weight of this edge.

It is guaranteed that the given graph **doesn't contain loops and multiple edges** (i.e. for each i from 1 to m $u_i \ne v_i$ and for each unordered pair of vertices (u, v) there is at most one edge connecting this pair of vertices). It is also guaranteed that the given graph is **connected**.

Output

Print one integer — the **minimum** number of operations to unify MST of the initial graph without changing the cost of MST.

Examples

input

```
8 10
1 2 1
2 3 2
2 4 5
1 4 2
6 3 3
6 1 3
3 5 2
3 7 1
4 8 1
6 2 4
```

output

```
1
```

input

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

output

```
0
```

input

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

output

```
0
```

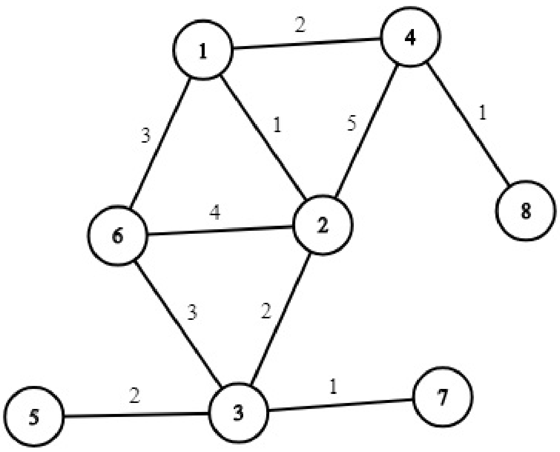
input
3 3 1 2 1 2 3 3 1 3 3
output
1

input
1 0
output
0

input
5 6 1 2 2 2 3 1 4 5 3 2 4 2 1 4 2 1 5 3
output
2

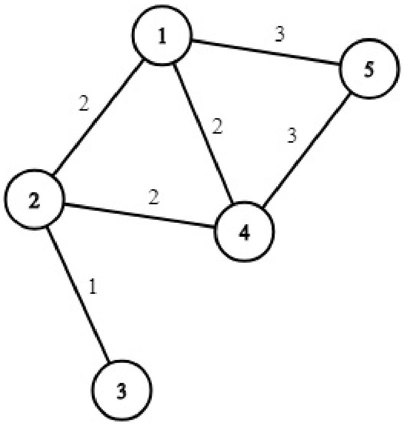
Note

The picture corresponding to the first example:



You can, for example, increase weight of the edge $(1, 6)$ or $(6, 3)$ by 1 to unify MST.

The picture corresponding to the last example:



You can, for example, increase weights of edges $(1,5)$ and $(2,4)$ by 1 to unify MST.