Codeforces Round #664 (Div. 1)

A. Boboniu Chats with Du

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

Have you ever used the chat application QQ? Well, in a chat group of QQ, administrators can muzzle a user for days.

In Boboniu's chat group, there's a person called Du Yi who likes to make fun of Boboniu every day.

Du will chat in the group for n days. On the i-th day:

- If Du can speak, he'll make fun of Boboniu with fun factor a_i. But after that, he may be muzzled depending on Boboniu's mood.
- Otherwise, Du won't do anything.

Boboniu's mood is a constant m. On the i-th day:

- If Du can speak and $a_i > m$, then Boboniu will be angry and muzzle him for d days, which means that Du won't be able to speak on the $i+1, i+2, \cdots, \min(i+d,n)$-th days.
- Otherwise, Boboniu won't do anything.

The total fun factor is the sum of the fun factors on the days when Du can speak.

Du asked you to find the maximum total fun factor among all possible permutations of a.

Input

The first line contains three integers n, d and m ($1 \le d \le n \le 10^5, 0 \le m \le 10^9$).

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$).

Output

Print one integer: the maximum total fun factor among all permutations of a.

Examples

input

```
5 2 11
8 10 15 23 5
```

output

```
48
```

input

```
20 2 16
20 5 8 2 18 16 2 16 16 1 5 16 2 13 6 16 4 17 21 7
```

output

```
195
```

Note

In the first example, you can set $a' = [15, 5, 8, 10, 23]$. Then Du's chatting record will be:

1. Make fun of Boboniu with fun factor 15.
2. Be muzzled.
3. Be muzzled.
4. Make fun of Boboniu with fun factor 10.
5. Make fun of Boboniu with fun factor 23.

Thus the total fun factor is 48.

B. Boboniu Walks on Graph

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

Boboniu has a **directed** graph with n vertices and m edges.

The out-degree of each vertex is at most k.

Each edge has an integer weight between 1 and m. No two edges have equal weights.

Boboniu likes to walk on the graph with some specific rules, which is represented by a tuple $(c_1, c_2, \ldots, c_k)$. If he now stands on a vertex u with out-degree i, then he will go to the next vertex by the edge with the c_i-th $(1 \le c_i \le i)$ smallest weight among all edges outgoing from u.

Now Boboniu asks you to calculate the number of tuples $(c_1, c_2, \ldots, c_k)$ such that

- $1 \le c_i \le i$ for all i ($1 \le i \le k$).
- Starting from any vertex u, it is possible to go back to u in finite time by walking on the graph under the described rules.

Input

The first line contains three integers n, m and k ($2 \le n \le 2 \cdot 10^5$, $2 \le m \le \min(2 \cdot 10^5, n(n-1))$, $1 \le k \le 9$).

Each of the next m lines contains three integers u, v and w $(1 \le u, v \le n, u \ne v, 1 \le w \le m)$, denoting an edge from u to v with weight w. It is guaranteed that there are no self-loops or multiple edges and each vertex has at least one edge starting from itself.

It is guaranteed that the out-degree of each vertex is at most k and no two edges have equal weight.

Output

Print one integer: the number of tuples.

Examples

input

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

output

```
2
```

input

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

output

```
1
```

input

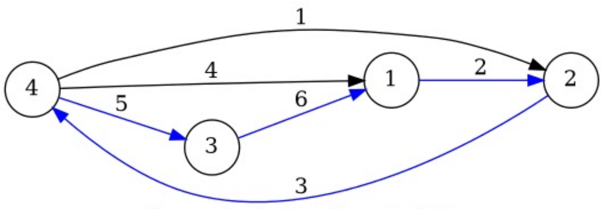
```
6 13 4
3 5 1
2 5 2
6 3 3
1 4 4
2 6 5
5 3 6
4 1 7
4 3 8
5 2 9
4 2 10
2 1 11
6 1 12
4 6 13
```

output

```
1
```

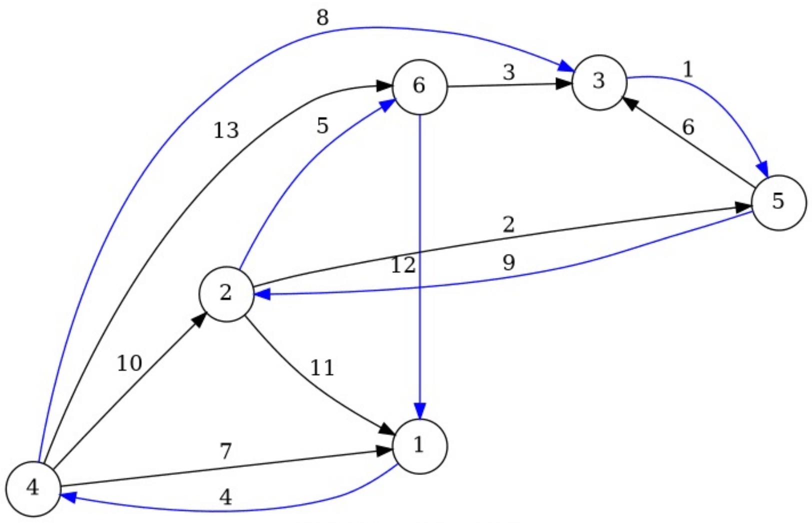
Note

For the first example, there are two tuples: $(1,1,3)$ and $(1,2,3)$. The blue edges in the picture denote the c_i-th smallest edges for each vertex, which Boboniu chooses to go through.



Example 1: (1,1,3) or (1,2,3)

For the third example, there's only one tuple: $(1,2,2,2)$.



Example 3: (1,2,2,2)

The *out-degree* of vertex u means the number of edges outgoing from u.

C. Boboniu and String

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

Boboniu defines *BN-string* as a string s of characters 'B' and 'N'.

You can perform the following operations on the BN-string s:

- Remove a character of s.
- Remove a substring "BN" or "NB" of s.
- Add a character 'B' or 'N' to the end of s.
- Add a string "BN" or "NB" to the end of s.

Note that a string a is a *substring* of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Boboniu thinks that BN-strings s and t are *similar* if and only if:

- $|s| = |t|$.
- There exists a permutation $p_1, p_2, \ldots, p_{|s|}$ such that for all i ($1 \le i \le |s|$), $s_{p_i} = t_i$.

Boboniu also defines $\text{dist}(s, t)$, the *distance* between s and t, as the minimum number of operations that makes s *similar* to t.

Now Boboniu gives you n non-empty BN-strings $s_1, s_2, \ldots, s_n$ and asks you to find a **non-empty** BN-string t such that the maximum distance to string s is minimized, i.e. you need to minimize $\max_{i=1}^{n} \text{dist}(s_i, t)$.

Input

The first line contains a single integer n ($1 \le n \le 3 \cdot 10^5$).

Each of the next n lines contains a string s_i ($1 \le |s_i| \le 5 \cdot 10^5$). It is guaranteed that s_i only contains 'B' and 'N'. The sum of $|s_i|$ does not exceed $5 \cdot 10^5$.

Output

In the first line, print the minimum $\max_{i=1}^{n} \text{dist}(s_i, t)$.

In the second line, print the suitable t.

If there are several possible t's, you can print any.

Examples

input
```
3
B
N
BN
```
output
```
1
BN
```

input
```
10
N
BBBBBB
BNNNBBNBB
NNNNBNBNNBNNNBBN
NBNBN
NNNNNN
BNBNBNBBBBNNNNBBBBNNBBNBNBBNBBBBBBBBB
NNNNBN
NBBBBBBBB
NNNNNN
```
output
```
12
BBBBBBBBBBBBNNNNNNNNNNNN
```

input
```
8
NNN
NNN
BBNNBBBN
NNNBNN
B
NNN
NNNNBNN
NNNNNNNNNNNNNNNNNBNNNNNNNNNBNB
```
output
```
12
BBBBNNNNNNNNNNNN
```

input
```
3
BNNNBNNNNNBNBBNNNBBNNNNNBBBBNNBBBBBBNBBBBBNBBBNNBBBNBNBBBN
BBBNBBBBNNNNNNBBNBBBBNNNBB
BBBBBBBBBBBBBBBNBBBBBNBBBBBNBBBBNB
```
output
```
12
BBBBBBBBBBBBBBBBBBBBBBBBBBNNNNNNNNNNNN
```

Note

In the first example $\text{dist(B,BN)} = \text{dist(N,BN)} = 1$, $\text{dist(BN,BN)} = 0$. So the maximum distance is 1.

D. Boboniu and Jianghu

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

Since Boboniu finished building his Jianghu, he has been doing Kungfu on these mountains every day.

Boboniu designs a map for his n mountains. He uses $n - 1$ roads to connect all n mountains. Every pair of mountains is connected via roads.

For the i-th mountain, Boboniu estimated the tiredness of doing Kungfu on the top of it as t_i. He also estimated the height of each mountain as h_i.

A *path* is a sequence of mountains M such that for each i ($1 \le i < |M|$), there exists a road between M_i and M_{i+1}. Boboniu would regard the *path* as a *challenge* if for each i ($1 \le i < |M|$), $h_{M_i} \le h_{M_{i+1}}$.

Boboniu wants to divide **all** $n - 1$ roads into several *challenges*. Note that each road must appear in **exactly one** challenge, but a mountain may appear in several challenges.

Boboniu wants to minimize the total tiredness to do all the *challenges*. The tiredness of a *challenge* M is the sum of tiredness of all mountains in it, i.e. $\sum_{i=1}^{|M|} t_{M_i}$.

He asked you to find the minimum total tiredness. As a reward for your work, you'll become a guardian in his Jianghu.

Input

The first line contains a single integer n ($2 \le n \le 2 \cdot 10^5$), denoting the number of the mountains.

The second line contains n integers $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 10^6$), denoting the tiredness for Boboniu to do Kungfu on each mountain.

The third line contains n integers $h_1, h_2, \dots, h_n$ $(1 \le h_i \le 10^6)$, denoting the height of each mountain.

Each of the following $n-1$ lines contains two integers u_i, v_i $(1 \le u_i, v_i \le n, u_i \neq v_i)$, denoting the ends of the road. It's guaranteed that all mountains are connected via roads.

Output

Print one integer: the smallest sum of tiredness of all challenges.

Examples

input

```
5
40 10 30 50 20
2 3 2 3 1
1 2
1 3
2 4
2 5
```

output

```
160
```

input

```
5
1000000 1 1 1 1
1000000 1 1 1 1
1 2
1 3
1 4
1 5
```

output

```
4000004
```

input

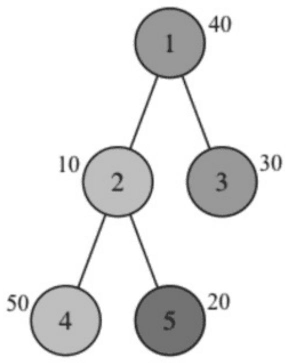
```
10
510916 760492 684704 32545 484888 933975 116895 77095 127679 989957
402815 705067 705067 705067 623759 103335 749243 138306 138306 844737
1 2
3 2
4 3
1 5
6 4
6 7
8 7
8 9
9 10
```

output

```
6390572
```

Note

For the first example:



In the picture, the lighter a point is, the higher the mountain it represents. One of the best divisions is:

- Challenge 1: $3 \to 1 \to 2$
- Challenge 2: $5 \to 2 \to 4$

The total tiredness of Boboniu is $(30 + 40 + 10) + (20 + 10 + 50) = 160$. It can be shown that this is the minimum total tiredness.

E. Boboniu and Banknote Collection

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

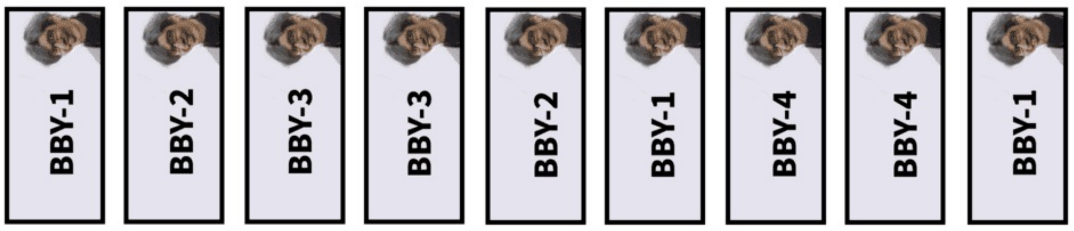
No matter what trouble you're in, don't be afraid, but face it with a smile.

I've made another billion dollars!

— Boboniu

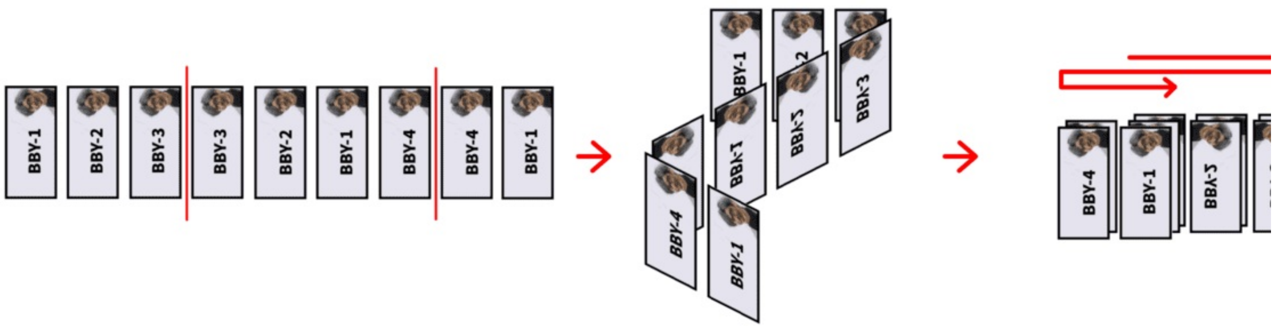
Boboniu has issued his currencies, named Bobo Yuan. Bobo Yuan (BBY) is a series of currencies. Boboniu gives each of them a positive integer identifier, such as BBY-1, BBY-2, etc.

Boboniu has a BBY collection. His collection looks like a sequence. For example:



We can use sequence $a = [1, 2, 3, 3, 2, 1, 4, 4, 1]$ of length $n = 9$ to denote it.

Now Boboniu wants to *fold* his collection. You can imagine that Boboniu stick his collection to a long piece of paper and fold it between currencies:



Boboniu will only fold the same identifier of currencies together. In other words, if a_i is folded over a_j ($1 \le i, j \le n$), then $a_i = a_j$ must hold. Boboniu doesn't care if you follow this rule in the process of folding. But once it is finished, the rule should be obeyed.

A formal definition of *fold* is described in notes.

According to the picture above, you can *fold* a two times. In fact, you can *fold* $a = [1, 2, 3, 3, 2, 1, 4, 4, 1]$ at most two times. So the maximum number of folds of it is 2.

As an international fan of Boboniu, you're asked to calculate the maximum number of folds.

You're given a sequence a of length n, for each i ($1 \le i \le n$), you need to calculate the maximum number of folds of $[a_1, a_2, \ldots, a_i]$.

Input

The first line contains an integer n ($1 \le n \le 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$).

Output

Print n integers. The i-th of them should be equal to the maximum number of folds of $[a_1, a_2, \ldots, a_i]$.

Examples

input
```
9
1 2 3 3 2 1 4 4 1
```
output
```
0 0 0 1 1 1 1 2 2
```

input
```
9
1 2 2 2 2 1 1 2 2
```
output
```
0 0 1 2 3 3 4 4 5
```

input
```
15
1 2 3 4 5 5 4 3 2 2 3 4 4 3 6
```
output
```
0 0 0 0 0 1 1 1 1 2 2 2 3 3 0
```

input
```
50
1 2 4 6 6 4 2 1 3 5 5 3 1 2 4 4 2 1 3 3 1 2 2 1 1 1 2 4 6 6 4 2 1 3 5 5 3 1 2 4 4 2 1 3 3 1 2 2 1 1
```
output
```
0 0 0 0 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 4 4 4 5 5 6 7 3 3 3 4 4 4 4 3 3 4 4 4 4 4 5 5 5 5 6 6 6 7 7 8
```

Note

Formally, for a sequence a of length n, let's define the *folding sequence* as a sequence b of length n such that:

- b_i ($1 \le i \le n$) is either 1 or -1.

- Let $p(i) = [b_i = 1] + \sum_{j=1}^{i-1} b_j$. For all $1 \le i < j \le n$, if $p(i) = p(j)$, then a_i should be equal to a_j.

($[A]$ is the value of boolean expression A. i. e. $[A] = 1$ if A is true, else $[A] = 0$).

Now we define the number of folds of b as $f(b) = \sum_{i=1}^{n-1} [b_i \ne b_{i+1}]$.

The maximum number of folds of a is $F(a) = \max\{f(b) \mid b \text{ is a folding sequence of } a\}$.