A. Friends or Not

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

Polycarpus has a hobby — he develops an unusual social network. His work is almost completed, and there is only one more module to implement — the module which determines friends. Oh yes, in this social network one won't have to add friends manually! Pairs of friends are deduced in the following way. Let's assume that user A sent user B a message at time t_1, and user B sent user A a message at time t_2. If $0 < t_2 - t_1 \le d$, then user B's message was *an answer* to user A's one. Users A and B are considered to be friends if A answered at least one B's message or B answered at least one A's message.

You are given the log of messages in chronological order and a number d. Find all pairs of users who will be considered to be friends.

Input

The first line of the input contains two integers n and d ($1 \le n, d \le 1000$). The next n lines contain the messages log. The i-th line contains one line of the log formatted as "A_i B_i t_i" (without the quotes), which means that user A_i sent a message to user B_i at time t_i ($1 \le i \le n$). A_i and B_i are non-empty strings at most 20 characters long, consisting of lowercase letters ('a' ... 'z'), and t_i is an integer ($0 \le t_i \le 10000$). It is guaranteed that the lines are given in non-decreasing order of t_i's and that no user sent a message to himself. The elements in the lines are separated by single spaces.

Output

In the first line print integer k — the number of pairs of friends. In the next k lines print pairs of friends as "A_i B_i" (without the quotes). You can print users in pairs and the pairs themselves in any order. Each pair must be printed exactly once.

Examples

input

```
4 1
vasya petya 1
petya vasya 2
anya ivan 2
ivan anya 4
```

output

```
1
petya vasya
```

input

```
1 1000
a b 0
```

output

```
0
```

Note

In the first sample test case Vasya and Petya are friends because their messages' sending times are one second apart. Anya and Ivan are not, because their messages' sending times differ by more than one second.

B. Matchmaker

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

Polycarpus has n markers and m marker caps. Each marker is described by two numbers: x_i is the color and y_i is the diameter. Correspondingly, each cap is described by two numbers: a_j is the color and b_j is the diameter. Cap (a_j, b_j) can close marker (x_i, y_i) only if their diameters match, that is, $b_j = y_i$. Besides, a marker is considered to be *beautifully closed*, if the cap color and the marker color match, that is, $a_j = x_i$.

Find the way to close the maximum number of markers. If there are several such ways, then choose the one that has the maximum number of *beautifully closed* markers.

Input

The first input line contains two space-separated integers n and m ($1 \le n, m \le 10^5$) — the number of markers and the number of caps, correspondingly.

Next n lines describe the markers. The i-th line contains two space-separated integers x_i, y_i ($1 \le x_i, y_i \le 1000$) — the i-th marker's color and diameter, correspondingly.

Next m lines describe the caps. The j-th line contains two space-separated integers a_j, b_j ($1 \le a_j, b_j \le 1000$) — the color and diameter of the j-th cap, correspondingly.

Output

Print two space-separated integers u, v, where u is the number of closed markers and v is the number of *beautifully closed* markers in the sought optimal way. Remember that you have to find the way to close the maximum number of markers, and if there are several such ways, you should choose the one where the number of *beautifully closed* markers is maximum.

Examples

input

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

output

```
3 2
```

input

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

output

```
1 0
```

Note

In the first test sample the first marker should be closed by the fourth cap, the second marker should be closed by the first cap and the third marker should be closed by the second cap. Thus, three markers will be closed, and two of them will be *beautifully closed* — the first and the third markers.

C. String Manipulation 1.0

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

One popular website developed an unusual username editing procedure. One can change the username only by deleting some characters from it: to change the current name s, a user can pick number p and character c and delete the p-th occurrence of character c from the name. After the user changed his name, he can't undo the change.

For example, one can change name "`arca`" by removing the second occurrence of character "a" to get "`arc`".

Polycarpus learned that some user initially registered under nickname t, where t is a concatenation of k copies of string s. Also, Polycarpus knows the sequence of this user's name changes. Help Polycarpus figure out the user's final name.

Input

The first line contains an integer k ($1 \le k \le 2000$). The second line contains a non-empty string s, consisting of lowercase Latin letters, at most 100 characters long. The third line contains an integer n ($0 \le n \le 20000$) — the number of username changes. Each of the next n lines contains the actual changes, one per line. The changes are written as "p_i c_i" (without the quotes), where p_i ($1 \le p_i \le 200000$) is the number of occurrences of letter c_i, c_i is a lowercase Latin letter. It is guaranteed that the operations are correct, that is, the letter to be deleted always exists, and after all operations not all letters are deleted from the name. The letters' occurrences are numbered starting from 1.

Output

Print a single string — the user's final name after all changes are applied to it.

Examples

input

```
2
bac
3
2 a
1 b
2 c
```

output

```
acb
```

input

```
1
abacaba
4
1 a
1 a
1 c
2 b
```

output

```
baa
```

Note

Let's consider the first sample. Initially we have name "`bacbac`"; the first operation transforms it into "`bacbc`", the second one — to "`acbc`", and finally, the third one transforms it into "`acb`".

D. Palindrome pairs

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

You are given a non-empty string s consisting of lowercase letters. Find the number of pairs of non-overlapping palindromic substrings of this string.

In a more formal way, you have to find the quantity of tuples (a, b, x, y) such that $1 \le a \le b < x \le y \le |s|$ and substrings $s[a... b]$, $s[x... y]$ are palindromes.

A *palindrome* is a string that can be read the same way from left to right and from right to left. For example, "abacaba", "z", "abba" are palindromes.

A *substring* $s[i... j]$ ($1 \le i \le j \le |s|$) of string $s = s_1s_2... s_{|s|}$ is a string $s_is_{i+1}... s_j$. For example, substring $s[2...4]$ of string s = "abacaba" equals "bac".

Input

The first line of input contains a non-empty string s which consists of lowercase letters ('a'...'z'), s contains at most 2000 characters.

Output

Output a single number — the quantity of pairs of non-overlapping palindromic substrings of s.

Please do not use the %lld format specifier to read or write 64-bit integers in C++. It is preferred to use cin, cout streams or the %I64d format specifier.

Examples

input

```
aa
```

output

```
1
```

input

```
aaa
```

output

```
5
```

input

```
abacaba
```

output

```
36
```

E. Zebra Tower

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Little Janet likes playing with cubes. Actually, she likes to play with anything whatsoever, cubes or tesseracts, as long as they are multicolored. Each cube is described by two parameters — color c_i and size s_i. A Zebra Tower is a tower that consists of cubes of exactly two colors. Besides, the colors of the cubes in the tower must alternate (colors of adjacent cubes must differ). The Zebra Tower should have at least two cubes. There are no other limitations. The figure below shows an example of a Zebra Tower.

A Zebra Tower's height is the sum of sizes of all cubes that form the tower. Help little Janet build the Zebra Tower of the maximum possible height, using the available cubes.

Input

The first line contains an integer n ($2 \le n \le 10^5$) — the number of cubes. Next n lines contain the descriptions of the cubes, one description per line. A cube description consists of two space-separated integers c_i and s_i ($1 \le c_i, s_i \le 10^9$) — the i-th cube's color and size, correspondingly. It is guaranteed that there are at least two cubes of different colors.

Output

Print the description of the Zebra Tower of the maximum height in the following form. In the first line print the tower's height, in the second line print the number of cubes that form the tower, and in the third line print the space-separated indices of cubes in the order in which they follow in the tower from the bottom to the top. Assume that the cubes are numbered from 1 to n in the order in which they were given in the input.

If there are several existing Zebra Towers with maximum heights, it is allowed to print any of them.

Please do not use the `%lld` specificator to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specificator.

Examples

input

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

output

```
9
3
2 3 1
```

input

```
2
1 1
2 1
```

output

```
2
2
2 1
```