VK Cup 2012 Round 1

A. Dress'em in Vests!

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

The Two-dimensional kingdom is going through hard times... This morning the Three-Dimensional kingdom declared war on the Two-dimensional one. This (possibly armed) conflict will determine the ultimate owner of the straight line.

The Two-dimensional kingdom has a regular army of n people. Each soldier registered himself and indicated the desired size of the bulletproof vest: the i-th soldier indicated size a_i. The soldiers are known to be unpretentious, so the command staff assumes that the soldiers are comfortable in any vests with sizes from $a_i - x$ to $a_i + y$, inclusive (numbers $x, y \geq 0$ are specified).

The Two-dimensional kingdom has m vests at its disposal, the j-th vest's size equals b_j. Help mobilize the Two-dimensional kingdom's army: equip with vests as many soldiers as possible. Each vest can be used only once. The i-th soldier can put on the j-th vest, if $a_i - x \leq b_j \leq a_i + y$.

Input

The first input line contains four integers n, m, x and y ($1 \leq n, m \leq 10^5$, $0 \leq x, y \leq 10^9$) — the number of soldiers, the number of vests and two numbers that specify the soldiers' unpretentiousness, correspondingly.

The second line contains n integers $a_1, a_2, ..., a_n$ ($1 \leq a_i \leq 10^9$) in non-decreasing order, separated by single spaces — the desired sizes of vests.

The third line contains m integers $b_1, b_2, ..., b_m$ ($1 \leq b_j \leq 10^9$) in non-decreasing order, separated by single spaces — the sizes of the available vests.

Output

In the first line print a single integer k — the maximum number of soldiers equipped with bulletproof vests.

In the next k lines print k pairs, one pair per line, as "u_i v_i" (without the quotes). Pair (u_i, v_i) means that soldier number u_i must wear vest number v_i. Soldiers and vests are numbered starting from one in the order in which they are specified in the input. All numbers of soldiers in the pairs should be pairwise different, all numbers of vests in the pairs also should be pairwise different. You can print the pairs in any order.

If there are multiple optimal answers, you are allowed to print any of them.

Examples

input

```
5 3 0 0
1 2 3 3 4
1 3 5
```

output

```
2
1 1
3 2
```

input

```
3 3 2 2
1 5 9
3 5 7
```

output

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

Note

In the first sample you need the vests' sizes to match perfectly: the first soldier gets the first vest (size 1), the third soldier gets the second vest (size 3). This sample allows another answer, which gives the second vest to the fourth soldier instead of the third one.

In the second sample the vest size can differ from the desired size by at most 2 sizes, so all soldiers can be equipped.

B. Discounts

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

One day Polycarpus stopped by a supermarket on his way home. It turns out that the supermarket is having a special offer for stools. The offer is as follows: if a customer's shopping cart contains at least one stool, the customer gets a 50% discount on the cheapest item in the cart (that is, it becomes two times cheaper). If there are several items with the same minimum price, the discount is available for only one of them!

Polycarpus has k carts, and he wants to buy up all stools and pencils from the supermarket. Help him distribute the stools and the pencils among the shopping carts, so that the items' total price (including the discounts) is the least possible.

Polycarpus must use all k carts to purchase the items, no shopping cart can remain empty. Each shopping cart can contain an arbitrary number of stools and/or pencils.

Input

The first input line contains two integers n and k ($1 \le k \le n \le 10^3$) — the number of items in the supermarket and the number of carts, correspondingly. Next n lines describe the items as "c_i t_i" (without the quotes), where c_i ($1 \le c_i \le 10^9$) is an integer denoting the price of the i-th item, t_i ($1 \le t_i \le 2$) is an integer representing the type of item i (1 for a stool and 2 for a pencil). The numbers in the lines are separated by single spaces.

Output

In the first line print a single real number **with exactly one** decimal place — the minimum total price of the items, including the discounts.

In the following k lines print the descriptions of the items in the carts. In the i-th line print the description of the i-th cart as "t b_1 b_2 ... b_t" (without the quotes), where t is the number of items in the i-th cart, and the sequence $b_1, b_2, \ldots, b_t$ ($1 \le b_j \le n$) gives the indices of items to put in this cart in the optimal distribution. All indices of items in all carts should be pairwise different, each item must belong to exactly one cart. You can print the items in carts and the carts themselves in any order. The items are numbered from 1 to n in the order in which they are specified in the input.

If there are multiple optimal distributions, you are allowed to print any of them.

Examples

input

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

output

```
5.5
2 1 2
1 3
```

input

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

output

```
8.0
1 1
2 4 2
1 3
```

Note

In the first sample case the first cart should contain the 1st and 2nd items, and the second cart should contain the 3rd item. This way each cart has a stool and each cart has a 50% discount for the cheapest item. The total price of all items will be: $2 \cdot 0.5 + (3 + 3 \cdot 0.5) = 1 + 4.5 = 5.5$.

C. Abracadabra

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

Polycarpus analyzes a string called *abracadabra*. This string is constructed using the following algorithm:

- On the first step the string consists of a single character "a".
- On the k-th step Polycarpus concatenates two copies of the string obtained on the $(k - 1)$-th step, while inserting the k-th character of the alphabet between them. Polycarpus uses the alphabet that consists of lowercase Latin letters and digits (a total of 36 characters). The alphabet characters are numbered like this: the 1-st character is "a", the 2-nd — "b", ..., the 26-th — "z", the 27-th — "0", the 28-th — "1", ..., the 36-th — "9".

Let's have a closer look at the algorithm. On the second step Polycarpus will concatenate two strings "a" and insert the character "b" between them, resulting in "aba" string. The third step will transform it into "abacaba", and the fourth one - into "abacabadabacaba". Thus, the string constructed on the k-th step will consist of $2^k - 1$ characters.

Polycarpus wrote down the string he got after 30 steps of the given algorithm and chose two non-empty substrings of it. Your task is to find the length of the longest common substring of the two substrings selected by Polycarpus.

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". The string is its own substring.

The longest common substring of two strings s and t is the longest string that is a substring of both s and t. For example, the longest common substring of "contest" and "systemtesting" is string "test". There can be several common substrings of maximum length.

Input

The input consists of a single line containing four integers l_1, r_1, l_2, r_2 $(1 \le l_i \le r_i \le 10^9, i = 1, 2)$. The numbers are separated by single spaces. l_i and r_i give the indices of the first and the last characters of the i-th chosen substring, correspondingly $(i = 1, 2)$. The characters of string *abracadabra* are numbered starting from 1.

Output

Print a single number — the length of the longest common substring of the given strings. If there are no common substrings, print 0.

Examples

input

```
3 6 1 4
```

output

```
2
```

input

```
1 1 4 4
```

output

```
0
```

Note

In the first sample the first substring is "acab", the second one is "abac". These two substrings have two longest common substrings "ac" and "ab", but we are only interested in their length — 2.

In the second sample the first substring is "a", the second one is "c". These two substrings don't have any common characters, so the length of their longest common substring is 0.

D. Distance in Tree

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

A *tree* is a connected graph that doesn't contain any cycles.

The *distance* between two vertices of a tree is the length (in edges) of the shortest path between these vertices.

You are given a tree with n vertices and a positive number k. Find the number of distinct pairs of the vertices which have a distance of exactly k between them. Note that pairs (v, u) and (u, v) are considered to be the same pair.

Input

The first line contains two integers n and k ($1 \le n \le 50000$, $1 \le k \le 500$) — the number of vertices and the required distance between the vertices.

Next $n - 1$ lines describe the edges as "a_i b_i" (without the quotes) ($1 \le a_i, b_i \le n$, $a_i \ne b_i$), where a_i and b_i are the vertices connected by the i-th edge. All given edges are different.

Output

Print a single integer — the number of distinct pairs of the tree's vertices which have a distance of exactly k between them.

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

Examples

input

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

output

```
4
```

input

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

output

```
2
```

Note

In the first sample the pairs of vertexes at distance 2 from each other are (1, 3), (1, 5), (3, 5) and (2, 4).

E. Polycarpus the Safecracker

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

Polycarpus has t safes. The password for each safe is a square matrix consisting of decimal digits '0' ... '9' (the sizes of passwords to the safes may vary). Alas, Polycarpus has forgotten all passwords, so now he has to restore them.

Polycarpus enjoys prime numbers, so when he chose the matrix passwords, he wrote a prime number in each row of each matrix. To his surprise, he found that all the matrices turned out to be symmetrical (that is, they remain the same after transposition). Now, years later, Polycarp was irritated to find out that he remembers only the prime numbers p_i, written in the first lines of the password matrices.

For each safe find the number of matrices which can be passwords to it.

The number of digits in p_i determines the number of rows and columns of the i-th matrix. One prime number can occur in several rows of the password matrix or in several matrices. The prime numbers that are written not in the first row of the matrix may have leading zeros.

Input

The first line of the input contains an integer t ($1 \le t \le 30$) — the number of safes. Next t lines contain integers p_i ($10 \le p_i \le 99999$), p_i is a prime number written in the first row of the password matrix for the i-th safe. All p_i's are written without leading zeros.

Output

Print t numbers, the i-th of them should be the number of matrices that can be a password to the i-th safe. Print the numbers on separate lines.

Examples

input

```
4
11
239
401
9001
```

output

```
4
28
61
2834
```

Note

Here is a possible password matrix for the second safe:

```
239
307
977
```

Here is a possible password matrix for the fourth safe:

```
9001
0002
0002
1223
```