Problem	**SOLDIERS**	**ROADS**	**BALL**
Program name	SOLDIERS.EXE	ROADS.EXE	BALL.EXE
Source name	SOLDIERS.PAS SOLDIERS.C SOLDIERS.CPP	ROADS.PAS ROADS.C ROADS.CPP	BALL.PAS BALL.C BALL.CPP
Input file	SOLDIERS.IN	ROADS.IN	BALL.IN
Output file	SOLDIERS.OUT	ROADS.OUT	BALL.OUT
Time limit per test	10 seconds	10 seconds	10 seconds
Number of tests	10	10	10
Points per test	3	3	4
Total points	**30**	**30**	**40**

The maximum total score for Round II is 100 points.

Problem description

N soldiers of the land Gridland are randomly scattered around the country.

A position in Gridland is given by a pair (x, y) of integer coordinates. Soldiers can move - in one move, one soldier can go one unit up, down, left or right (hence, he can change either his x or his y coordinate by 1 or -1).

The soldiers want to get into a horizontal line next to each other (so that their final positions are (x, y), (x+1, y), ..., (x+N-1, y), for some x and y). Integers x and y, as well as the final order of soldiers along the horizontal line is arbitrary.

The goal is to minimise the total number of moves of all the soldiers that takes them into such configuration.

Two or more soldiers must never occupy the same position at the same time.

Input data

The first line of the input file **SOLDIERS.IN** contains the integer N, 1 ≤ N ≤ 10000, the number of soldiers.

The following N lines of the input file contain initial positions of the soldiers: for each i, 1 ≤ i ≤ N, the (i+1)st line of the input file contains a pair of integers x[i] and y[i] separated by a single blank character, representing the coordinates of the ith soldier, -10000 ≤ x[i], y[i] ≤ 10000.

Output data

The first and the only line of the output file **SOLDIERS.OUT** should contain the minimum total number of moves that takes the soldiers into a horizontal line next to each other.

Examples

```
SOLDIERS.IN

3
1 0
2 4
3 2

SOLDIERS.OUT

4
```

```
SOLDIERS.IN

5
1 2
2 2
1 3
3 -2
3 3

SOLDIERS.OUT

8
```

Problem description

N cities named with numbers 1 ... N are connected with one-way roads. Each road has two parameters associated with it: the road length and the toll that needs to be paid for the road (expressed in the number of coins).

Bob and Alice used to live in the city 1. After noticing that Alice was cheating in the card game they liked to play, Bob broke up with her and decided to move away - to the city N. He wants to get there as quickly as possible, but he is short on cash.

We want to help Bob to find **the shortest path** from the city 1 to the city N **that he can afford** with the amount of money he has.

Input data

The first line of the input file **ROADS.IN** contains the integer K, 0 ≤ K ≤ 10000, maximum number of coins that Bob can spend on his way.

The second line contains the integer N, 2 ≤ N ≤ 100, the total number of cities.

The third line contains the integer R, 1 ≤ R ≤ 10000, the total number of roads.

Each of the following R lines describes one road by specifying integers S, D, L and T separated by single blank characters :

- S is the source city, 1 ≤ S ≤ N
- D is the destination city, 1 ≤ D ≤ N
- L is the road length, 1 ≤ L ≤ 100
- T is the toll (expressed in the number of coins), 0 ≤ T ≤ 100

Notice that different roads may have the same source and destination cities.

Output data

The first and the only line of the output file **ROADS.OUT** should contain the total length of the shortest path from the city 1 to the city N whose total toll is less than or equal K coins.

If such path does not exist, only number -1 should be written to the output file.

Examples

```
ROADS.IN

5
6
7
1 2 2 3
2 4 3 3
3 4 2 4
1 3 4 1
4 6 2 1
3 5 2 0
5 4 3 2

ROADS.OUT
```

```
11
```

```
ROADS.IN

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

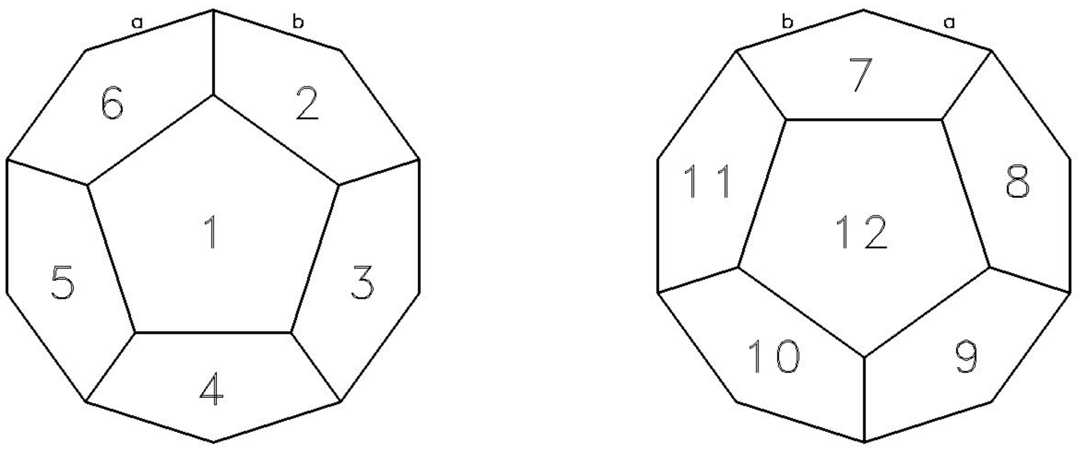
```
ROADS.OUT

-1
```

Problem description

Professor Balltazar is a big football fan. His birthday was just a couple of days before he was going to leave for the World Cup Football '98 in France, so his friends gave him as a present a dodecahedron-shaped puzzle as an entertainment while watching the boring games.

The puzzle has 12 equal pentagon sides, labelled with numbers 1 ... 12. The figure below shows the two hemispheres of the dodecahedron, together with the side labelling we will use in this problem. Hemispheres are "glued" together in such a way that the side 7 is adjacent to the sides 8, 12, 11, 2 and 6 (sides are adjacent if they share an edge). In particular, edges *a* and *b* on the left hemisphere will be glued to the edges *a* and *b* on the right hemisphere shown below.



In addition to that, there are 12 pentagon-shaped tiles, also labelled from 1 to 12. Every edge on each of the tiles is marked with a number from the set {0, 1, 2}. Each tile can be placed on each of the twelve sides in any of the 5 positions obtained by rotating the tile around its centre.

To solve the puzzle, we need to put each tile on some of the twelve dodecahedron sides in some position, so that every two adjacent tiles have their common edge marked with the same number.

Help Professor Balltazar to solve the puzzle !

Input data

The input file **BALL.IN** contains 12 lines. For each i, 1 ≤ i ≤ 12, the ith line describes the ith tile by specifying 5 numbers from the set {0, 1, 2} separated by single blank characters. This sequence shows the edge marking of the ith tile starting from an arbitrary edge (called the **ith reference edge**) going in the clockwise direction.

Output data

Output file **BALL.OUT** should contain the description of a solved puzzle in 12 lines with 2 integers in each line. For each i, 1 ≤ i ≤ 12, the ith line should contain integers t[i] and n[i] separated by a single blank character describing the tile and its position on the ith side:

The ith side will hold the tile labelled t[i].

The tile can be placed on the ith side in five different positions. The exact position is specified by n[i], which denotes the label of the adjacent side which is in the direction of the t[i]th reference edge (looking from the centre of the ith side). Precisely, the t[i]th reference edge is placed on the dodecahedron edge shared by the sides labelled i and n[i].

If the puzzle can not be solved, only number -1 should be written to the output file.

Examples

```
BALL.IN

0 0 1 1 2
0 2 1 0 1
2 0 1 0 1
0 0 1 2 1
0 2 1 1 2
2 0 1 2 1
0 2 1 2 1
2 2 1 0 1
1 2 2 0 0
0 2 1 0 2
0 2 1 2 0
2 0 1 2 0

BALL.OUT

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

```
BALL.IN

1 0 2 0 2
2 2 2 1 2
1 1 0 0 0
1 1 0 2 1
2 1 1 1 1
1 2 2 1 1
2 1 2 2 1
2 2 0 1 0
0 1 2 1 2
2 2 1 0 0
1 2 0 2 0
2 2 2 0 1

BALL.OUT

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