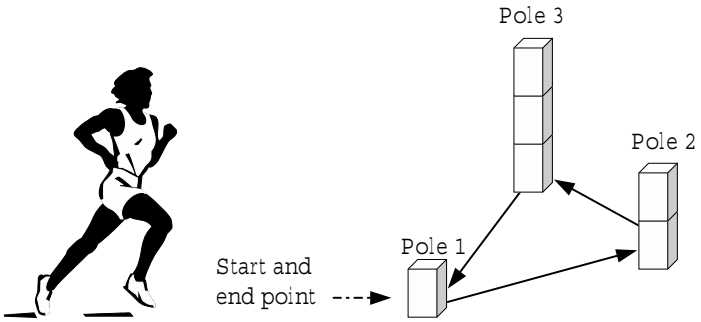
1 ROUTE

1.1 The Task

A jogging route in a countryside is a loop, so the starting point is also the ending point. The heights of the route at 1-metre interval are given as a list of measurements in centimetres above the sea-level.

Example 1. A simple 3-metre route is shown below. The heights are shown as "poles" in the diagram. The poles, from the starting point (Pole 1, which is also the ending point) in the jogging direction, are of heights 10, 20, and 30.

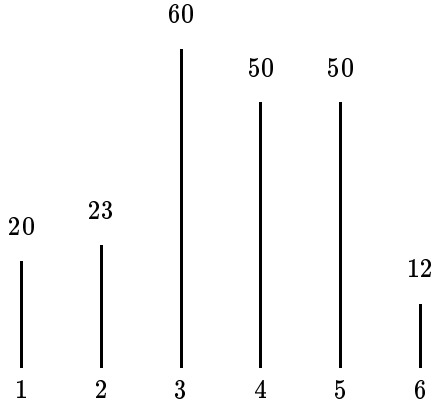


You may assume that the route between any two neighbouring poles is a straight line.

You are to compute (1) the number of plains, (2) the number of up-slopes, and (3) the number of down-slopes in the route. A plain, an up-slope, or a down-slope is the longest continuous stretch of the route that is level, up-hill or down-hill in the jogging direction, respectively. In the example above, there is one up-slope (from Pole 1 to Pole 2 and then to Pole 3), and one down-slope (from Pole 3 back to Pole 1, where the joggers must stop). There is no plain.

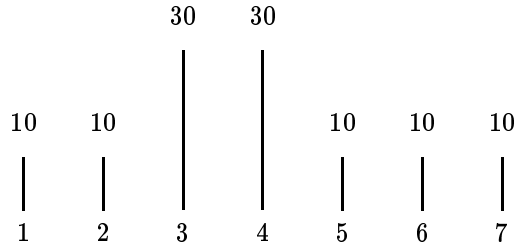
More examples are given below, with the "poles" drawn on a straight line instead of a loop for easy viewing (but remember that after the last pole, the joggers must go further to finish at the first pole).

Example 2. Consider the following jogging route with 6 poles:



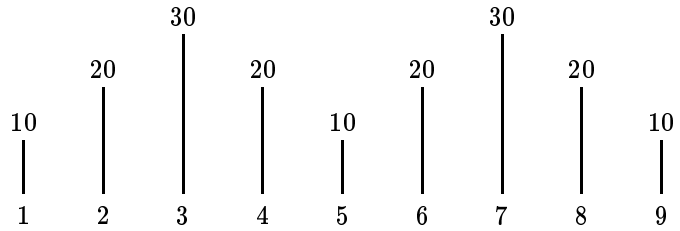
The only plain is Poles 4 to 5. The two up-slopes are Poles 1 to 3 and Poles 6 to 1. The two down-slopes are Poles 3 to 4 and Poles 5 to 6.

Example 3. Now consider a 7-pole jogging route.



The three plains are Poles 1 to 2, Poles 3 to 4, and Poles 5 to 1. The only up-slope is Poles 2 to 3. The only down-slope is Poles 4 to 5.

Example 4. The following is a 9-pole jogging route.



The only plain is Poles 9 to 1. The two up-slopes are Poles 1 to 3 and Poles 5 to 7. The two down-slopes are Poles 3 to 5 and Poles 7 to 9.

1.2 Requirements

1. Read the input file `ROUTE.IN` to obtain the length N of the route in metres ($3 \leq N \leq 30,000$) and the heights H_i, $1 \leq i \leq N$, of the poles in centimetres above sea-level ($1 \leq H_i \leq 30,000$). All values are positive integers.
2. Compute the number of plains, up-slopes, and down-slopes.
3. Write the 3 numbers to the output file `ROUTE.OUT` on one line in **this order**: the number of plains, the number of up-slopes, the number of down-slopes.

1.3 Input, Output Files: `ROUTE.IN`, `ROUTE.OUT`

The input file `ROUTE.IN` contains the integer N, the length of the route in metres, on the first line. The subsequent N lines contains the heights of the poles (in centimetres above sea-level) in the jogging direction: the first, second, $\cdots$, last of these N lines contains the heights of the first, second, $\cdots$, last poles respectively.

The output file, `ROUTE.OUT`, should contain 3 numbers on one line in this order: number of plains, number of up-slopes, number of down-slopes. There should be a space between two adjacent numbers.

The input and output files of Example 1 are:

ROUTE.IN	ROUTE.OUT
3 10 20 30	0 1 1

The input and output files of Example 2 are:

ROUTE.IN	ROUTE.OUT
6 20 23 60 50 50 12	1 2 2

The input and output files of Example 3 are:

ROUTE.IN	ROUTE.OUT
7 10 10 30 30 10 10 10	3 1 1

The input and output files of Example 4 are:

ROUTE.IN	ROUTE.OUT
9 10 20 30 20 10 20 30 20 10	1 2 2

2 IMAGE

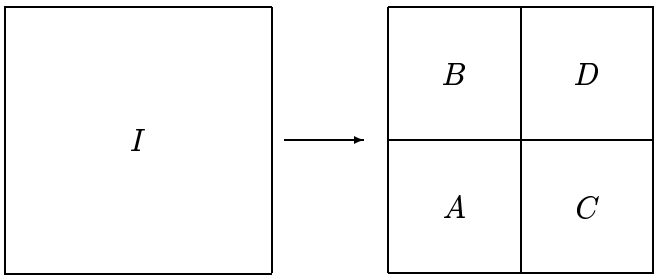
2.1 The Task

This task concerns the output length of the quadtree image compression scheme. Only $L \times L$ images consisting of L^2 pixels are considered. An image pixel is either a 0-pixel or a 1-pixel.

The quadtree image compression scheme is as follows:

(1) If the image consists of both 0-pixels and 1-pixels, encode a 1 to indicate that the image will be partitioned into 4 sub-images as described in Step (2). Otherwise, encode the entire image as 00 or 01 to indicate that the image consists of only 0-pixels or only 1-pixels respectively.

(2) An image I is partitioned into 4 equal size sub-images A, B, C, D as shown:



Step (1) is then performed on each of the four sub-images in the order of A, B, C, D.

Let (I) be the encoding of the image I. The following examples show the encoding process to compress an image.

Example 1. This example shows the encoding process of a 4×4 image.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} = 1 \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$= 1 \;\; 1(0)(0)(0)(1) \;\; 01 \;\; 1(1)(0)(1)(0) \;\; 1(0)(1)(1)(1)$$

$$= 1\ 1\ 00\ 00\ 00\ 01\ 01\ 1\ 01\ 00\ 01\ 00\ 1\ 00\ 01\ 01\ 01$$

Since there are 30 bits (0's and 1's) in the encoding, the length of the compressed image is 30.

Example 2. This example shows the encoding process of a 8×8 image.

$$\begin{pmatrix} 0&0&0&0&1&1&1&1 \\ 0&0&0&0&1&1&1&1 \\ 0&0&0&0&1&1&1&1 \\ 0&0&0&0&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1 \end{pmatrix} = 1 \begin{pmatrix} 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \end{pmatrix} \begin{pmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{pmatrix} \begin{pmatrix} 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \end{pmatrix} \begin{pmatrix} 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \end{pmatrix}$$

$$= 1\ 01\ 00\ 01\ 01$$

Thus the length of the compressed image is 9 (bits).

2.2 Requirements

1. Read an $L \times L$ image ($1 \leq L \leq 64$ and L is a power of 2) from the input file `IMAGE.IN`.
2. Compute the length (the number of bits) of the compressed image encoded by the quadtree compression scheme.
3. Write the length to the output file `IMAGE.OUT`.

2.3 Input, Output Files: `IMAGE.IN`, `IMAGE.OUT`

For an $L \times L$ square image, the input file contains $L + 1$ lines. The first line consists of the single integer L. Each line of the subsequent L lines consists of L bits (a bit is either a 0 or a 1) with a blank between two adjacent bits.

The output file consists of one integer which is the length (the number of bits) in the compressed image.

The input and output files for Example 1 are:

```
IMAGE.IN                    IMAGE.OUT

4                           30
1  1  1  1
1  1  0  1
0  1  0  0
0  0  1  1
```

The input and output files for Example 2 are:

```
IMAGE.IN                    IMAGE.OUT

8                           9
0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1
1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1
```

3 COIN

3.1 The Task

A certain currency is issued with N coin denominations. (That is, the currency has N types of coins.) Each coin denomination i has a monetary value of v_i cents and a weight of w_i grams, $1 \leq i \leq N$. Two coin denominations may have the same value or the same weight, but not both. For given values of V and W, you are to find M, the minimum number of coins needed, such that these M coins have a total monetary value of V cents and a total weight of W grams. Set M to zero when there is no such a collection of coins. You may assume an infinite supply of coins for each denomination.

Example 1. Let the coin denominations be

i	v_i	w_i
1	1	1
2	2	1
3	4	1
4	8	1
5	16	1
6	32	1
7	64	1
8	128	1

and $V = 141$, $W = 4$. We have $M = 4$ because the 4 coins, one from each of the denominations 1, 3, 4, 8, have the monetary value 141 and weight 4.

Example 2. Let the coin denominations be

i	v_i	w_i
1	12	3
2	4	7
3	8	10
4	21	9

For $V = 11$ and $W = 17$, we have $M = 0$ because obviously there is no such a set of coins.

3.2 The Requirements

1. Read the input file `COIN.IN` to obtain N ($1 \leq N \leq 20$, the number of coin denominations), V ($1 \leq V \leq 150$, the required total monetary value), W ($1 \leq W \leq 150$, the required total weight), the monetary value v_i ($1 \leq v_i \leq 150$) and the weight w_i ($1 \leq w_i \leq 150$), $1 \leq i \leq N$, of each denomination.

2. Determine M, the minimum number of coins needed to give a total monetary value of V cents and a total weight of W grams. The value of M should be zero if there are no such coins.

3. Write the value of M to the output file `COIN.OUT`.

3.3 Input, Output Files: COIN.IN, COIN.OUT

The input file contains $N + 1$ lines. The first line consists of the integers N, V and W in that order. Each of the subsequent N lines consists of 2 integers separated by a space describing one coin denomination: the first integer is the value of the coin in cents and the second integer is the weight of the coin in grams.

For Example 1, the input and output files are

```
COIN.IN                          COIN.OUT

8  141  4                        4
1  1
2  1
4  1
8  1
16  1
32  1
64  1
128  1
```

For Example 2, the input and output files are

```
COIN.IN                          COIN.OUT

4  11  17                       0
12  3
4  7
8  10
21  9
```

4 MAGIC

4.1 The Task

A magic square is an $N \times N$ matrix such that

1. Every entry of the matrix is an integer between 1 and N^2 inclusively.
2. The entries of the matrix are distinct.
3. The N row sums, the N column sums, and the two main diagonal sums are all equal.

For example,

8	3	4
1	5	9
6	7	2

is a 3×3 matrix that is a magic square. The three row sums are $8+3+4$, $1+5+9$, $6+7+2$; the three column sums are $8+1+6$, $3+5+7$, $4+9+2$; the two main diagonal sums are $8+5+2$, $4+5+6$. All these sums are 15.

You are to find out if the remaining entries of a partially filled $N \times N$ matrix can be completed so that the matrix becomes a magic square.

Example 1. The partially filled matrix

	1		24	
		8		
	9			
	10	21		
				11

can be completed to become

2	1	18	24	20
25	23	8	4	5
16	9	12	13	15
3	10	21	17	14
19	22	6	7	11

It can be checked the five row sums, the five column sums, and the two main diagonal sums are all 65. Furthermore, all entries are distinct with values from 1 to 25 inclusively. Thus the given partially filled matrix can become a magic square.

4.2 Requirements

1. Read the input file `MAGIC.IN` to obtain the size of the matrix and the values of the filled entries.
2. Check if the partially filled matrix can be completed to become a magic square.
3. Write the word "yes" or "no" to the output file `MAGIC.OUT` accordingly.

4.3 Input, Output Files: `MAGIC.IN`, `MAGIC.OUT`

The first line of the input file consists of two integers: the first integer N ($2 \leq N \leq 5$) is the number of rows (or columns) of the partially filled matrix, the second integer E is the number of the filled entries of the partially filled matrix. Each of the remaining E lines of the input file consists of three integers with a space between two adjacent integers: the row index R ($1 \leq R \leq N$), the column index C ($1 \leq C \leq N$), and the value V ($1 \leq V \leq N^2$) of a filled entry. All the V's are distinct.

The output file contains only one word: "yes" if the given matrix can be completed to become a magic square, "no" otherwise.

The input and output files of Example 1 are:

```
MAGIC.IN                MAGIC.OUT

5 7                     yes
1 4 24
4 2 10
5 5 11
2 3 8
3 2 9
1 2 1
4 3 21
```

5 TRIPLE

5.1 The Task

An ordered triple of ordered pairs is written as $< (a,b),(c,d),(e,f) >$.

Rule 1. A triple vanishes if two of its three ordered pairs are identical. That is

$$\begin{aligned} < (a,b),(a,b),(a,b) > &= 0 \\ < (a,b),(a,b),(e,f) > &= 0 \\ < (a,b),(c,d),(a,b) > &= 0 \\ < (a,b),(c,d),(c,d) > &= 0 \end{aligned}$$

A triple that vanishes is called a zero triple.

Rule 2. Two triples differ in sign if one can be transformed to another with an odd number of adjacent swaps. Two triples are equal if one can be transformed to another with an even number of adjacent swaps. Let "$\rightarrow$" mean the operation of adjacent swap. Since

$$\begin{array}{c} < (1,2),(3,4),(5,6) > \\ \downarrow \\ < (3,4),(1,2),(5,6) > \\ \downarrow \\ < (3,4),(5,6),(1,2) > \\ \downarrow \\ < (5,6),(3,4),(1,2) > \\ \downarrow \\ < (5,6),(1,2),(3,4) > \\ \downarrow \\ < (1,2),(5,6),(3,4) > \end{array}$$

we have

$$\begin{aligned} < (1,2),(3,4),(5,6) > &= - < (3,4),(1,2),(5,6) > \\ &= < (3,4),(5,6),(1,2) > \\ &= - < (5,6),(3,4),(1,2) > \\ &= < (5,6),(1,2),(3,4) > \\ &= - < (1,2),(5,6),(3,4) > \end{aligned}$$

Consider summing all the triples $< (a,b),(c,d),(e,f) >$ where

$$a_1 \le a \le a_2, \quad b_1 \le b \le b_2; \qquad c_1 \le c \le c_2, \quad d_1 \le d \le d_2; \qquad e_1 \le e \le e_2, \quad f_1 \le f \le f_2.$$

and all the values are integers. After discarding zero triples (Rule 1) and cancelling pairs of triples that differ in sign (Rule 2), you are to determine the number of distinct non-zero triples in the sum.

Example 1. Let $1 \le a \le 1$, $2 \le b \le 2$, $3 \le c \le 3$, $4 \le d \le 4$, $5 \le e \le 5$, $6 \le f \le 8$. There are only 3 triples in these ranges and none is zero and no two differ in sign, so there are 3 distinct non-zero triples if we sum them:

$$< (1,2),(3,4),(5,6) > + < (1,2),(3,4),(5,7) > + < (1,2),(3,4),(5,8) > .$$

Example 2. Now consider $1 \le a \le 1$, $2 \le b \le 2$, $4 \le c \le 5$, $7 \le d \le 8$, $5 \le e \le 5$, $6 \le f \le 8$. There are 12 triples in these ranges. The 2 triples $< (1,2),(5,7),(5,7) >$ and $< (1,2),(5,8),(5,8) >$ are zero by Rule 1. The 2 triples $< (1,2),(5,7),(5,8) >$ and $< (1,2),(5,8),(5,7) >$ cancel each other (they sum to zero) by Rule 2. Thus the sum of the original 12 triples has 8 distinct non-zero triples:

$$< (1,2),(4,7),(5,6) > + < (1,2),(4,7),(5,7) > + < (1,2),(4,7),(5,8) > +$$
$$< (1,2),(4,8),(5,6) > + < (1,2),(4,8),(5,7) > + < (1,2),(4,8),(5,8) > +$$
$$< (1,2),(5,7),(5,6) > + < (1,2),(5,8),(5,6) > .$$

Example 3. Let the ranges be $0 \le a \le 0$, $0 \le b \le 0$, $0 \le c \le 0$, $0 \le d \le 1$, $0 \le e \le 1$, $0 \le f \le 1$. The six triples

$$< (0,0),(0,0),(0,0) >, < (0,0),(0,0),(0,1) >, < (0,0),(0,0),(1,0) >,$$
$$< (0,0),(0,0),(1,1) >, < (0,0),(0,1),(0,0) >, < (0,0),(0,1),(0,1) >$$

are zero by Rule 1. Thus the sum of the original eight triples has two distinct non-zero triples:

$$< (0,0),(0,1),(1,0) > + < (0,0),(0,1),(1,1) >$$

5.2 Requirements

1. Read the input file `TRIPLE.IN` to obtain the ranges of a, b, c, d, e, f. For each of a, b, c, d, e, f, the lowest value is 0 and the highest value is 100.
2. Sum all the triples $< (a,b),(c,d),(e,f) >$ within the given ranges using Rule 1 and Rule 2. Determine the number of non-zero distinct triples in the sum. This number is between 0 and 1000 inclusively.
3. Write the number of non-zero distinct triples in the sum to the output file `TRIPLE.OUT`.

5.3 Input, Output Files: `TRIPLE.IN`, `TRIPLE.OUT`

The input file contains twelve integers (there is a space between two adjacent integers) for the ranges on a single line. The integers are given in the order of

$$a_1\ a_2\ b_1\ b_2\ c_1\ c_2\ d_1\ d_2\ e_1\ e_2\ f_1\ f_2$$

The output file contains one integer which is the number of non-zero distinct triples in the sum of the original triples within the given ranges.

For Example 1, the input and output files are

`TRIPLE.IN`	`TRIPLE.OUT`
`1 1 2 2 3 3 4 4 5 5 6 8`	`3`

For Example 2, the input and output files are

`TRIPLE.IN`	`TRIPLE.OUT`
`1 1 2 2 4 5 7 8 5 5 6 8`	`8`

For Example 3, the input and output files are

`TRIPLE.IN`	`TRIPLE.OUT`
`0 0 0 0 0 0 0 1 0 1 0 1`	`2`