# Walls

## PROBLEM

In a country, great walls have been built in such a way that every great wall connects exactly two towns. The great walls do not cross each other. Thus, the country is divided into such regions that to move from one region to another, it is necessary to go through a town or cross a great wall. For any two towns A and B, there is at most one great wall with one end in A and the other in B, and further, it is possible to go from A to B by always walking in a town or along a great wall. The input format implies additional restrictions.

There is a club whose members live in the towns. In each town, there is only one member or there are no members at all. The members want to meet in one of the regions (outside of any town). The members travel riding their bicycles. They do not want to enter any towns, because of the traffic, and they want to cross as few great walls as possible, as it is a lot of trouble. To go to the meeting region, each member needs to cross a number (possibly 0) of great walls. They want to find such an optimal region that the sum of these numbers (crossing-sum, for short) is minimized.

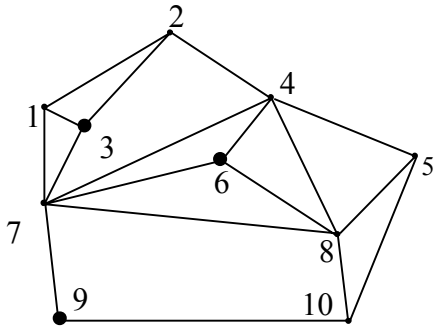


Figure 1

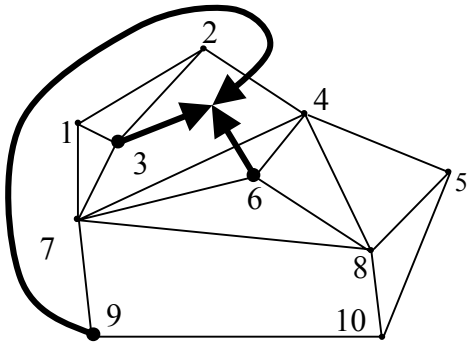


Figure 2

The towns are labeled with integers from 1 to *N*, where *N* is the number of towns. In Figure 1, the labeled nodes represent the towns and the lines connecting the nodes represent the great walls. Suppose that there are three members, who live in towns 3, 6, and 9. Then, an optimal meeting region and respective routes for members are shown in Figure 2. The crossing-sum is 2: the member from town 9 has to cross the great wall between towns 2 and 4, and the member from town 6 has to cross the great wall between towns 4 and 7.

You are to write a program which, given the towns, the regions, and the club member home towns, computes an optimal region and the minimal crossing-sum.

## INPUT

The input file name is `WALLS.IN`. The first line contains one integer: the number of regions *M*, $2 \le M \le 200$. The second line contains one integer: the number of towns *N*, $3 \le N \le 250$. The third line contains one integer: the number of club members *L*, $1 \le L \le 30$,

$L \le N$. The fourth line contains $L$ distinct integers in increasing order: the labels of the towns where the members live.

After that the file contains $2M$ lines so that there is a pair of lines for each region: the first two of the $2M$ lines describe the first region, the following two the second and so on. Of the pair, the first line shows the number of towns $I$ on the border of that region. The second line of the pair contains $I$ integers: the labels of these $I$ towns in some order in which they can be passed when making a trip clockwise along the border of the region, with the following exception. The last region is the "outside region" surrounding all towns and other regions, and for it the order of the labels corresponds to a trip in counterclockwise direction. The order of the regions gives an integer labeling to the regions: the first region has label 1, the second has label 2, and so on. Note that the input includes all regions formed by the towns and great walls, including the "outside region".

## OUTPUT

The output file name is `WALLS.OUT`. The first line contains one integer: the minimal crossing-sum. The second line contains one integer: the label of an optimal region. There may be several different solutions for the region and your program needs to output only one of them.

## EXAMPLE INPUT AND OUTPUT

The following input and output files correspond to the example given in the text.

`WALLS.IN`

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

`WALLS.OUT`

```
2
3
```

# Post Office

## PROBLEM

There is a straight highway with villages alongside the highway. The highway is represented as an integer axis, and the position of each village is identified with a single integer coordinate. There are no two villages in the same position. The distance between two positions is the absolute value of the difference of their integer coordinates.

Post offices will be built in some, but not necessarily all of the villages. A village and the post office in it have the same position. For building the post offices, their positions should be chosen so that the total sum of all distances between each village and its nearest post office is minimum.

You are to write a program which, given the positions of the villages and the number of post offices, computes the least possible sum of all distances between each village and its nearest post office, and the respective desired positions of the post offices.

## INPUT

The input file name is `POST.IN`. The first line contains two integers: the first is the number of villages $V$, $1 \le V \le 300$, and the second is the number of post offices $P$, $1 \le P \le 30$, $P \le V$. The second line contains $V$ integers in increasing order. These $V$ integers are the positions of the villages. For each position $X$ it holds that $1 \le X \le 10000$.

## OUTPUT

The output file name is `POST.OUT`. The first line contains one integer $S$, which is the sum of all distances between each village and its nearest post office as reported in the second line. The second line contains $P$ integers in increasing order. These integers are the locations of the distinct villages in which the post offices will be built. There may be several different solutions for the locations, and your program needs to output only one of them.

## EXAMPLE INPUT AND OUTPUT

`POST.IN`

```
10 5
1 2 3 6 7 9 11 22 44 50
```

`POST.OUT`

```
9
2 7 22 44 50
```

## PARTIAL CREDIT

If your program's output does not satisfy the output requirements, then your score is 0. Otherwise, your score will be computed based on Table 1 as follows. If your program outputs sum *S* and the actual least possible sum is *Smin*, then calculate $q=S/Smin$ and find your score *c* in the bottom row.

| $q=S/Smin$ | $q=1.0$ | $1.0<q\le1.1$ | $1.1<q\le1.15$ | $1.15<q\le1.2$ | $1.2<q\le1.25$ | $1.25<q\le1.3$ | $1.3<q$ |
|---|---|---|---|---|---|---|---|
| *c* | 10 | 5 | 4 | 3 | 2 | 1 | 0 |

Table 1.

# Building with Blocks

## PROBLEM

A unit cube is a 1x1x1 cube, whose corners have integer $x$, $y$, and $z$ coordinates. Two unit cubes are connected when they share a face. A 3-dimensional solid object (solid, for short) is a non-empty connected set of unit cubes (see Figure 1). The volume of a solid is the number of unit cubes it contains. A block is a solid with volume at most 4. Two blocks have the same type when one can be obtained from the other by translations and rotations (not reflections). There are exactly 12 block types (see Figure 2). The colors in the figures only help to clarify the structure of the solids; they have no other meaning.

A set D of blocks is a decomposition of a solid S when the union of all blocks in D equals S, and no two distinct blocks in D have a unit cube in common.

Your task is to write a program that, given a description of the block types and a solid S, determines a smallest set of blocks into which S can be decomposed. It only needs to report the types of these blocks as often as they occur in the decomposition.

## INPUT

In the input files, we identify a unit cube by a line with three integers $x$, $y$, and $z$, being the coordinate triple of its corner that minimizes $x + y + z$.

The input file describing the block types is named `TYPES.IN`. The contents of this file are listed below and are the same for all evaluation runs. It contains the descriptions of the 12 block types in Figure 2, sorted on type number. Each block type is described by a group of consecutive lines. The first line contains the integer $I$ identifying the block type ($1 \leqslant I \leqslant 12$). The second line contains the volume $V$ of the block type ($1 \leqslant V \leqslant 4$). The remaining V lines contain three integers $x$, $y$ and $z$, each being one unit cube of the block type ($1 \leqslant x, y, z \leqslant 4$).

The input file describing the solid is named `BLOCK.IN`. The first line contains the volume $V$ of the solid ($1 \leqslant V \leqslant 50$). The remaining $V$ lines contain three integers $x$, $y$, $z$, each being one unit cube of the solid ($1 \leqslant x, y, z \leqslant 7$).

## OUTPUT

The output file is named `BLOCK.OUT`. The first line must contain one integer $M$, being the smallest number of blocks into which the input solid can be decomposed. The second line lists $M$ type identifiers of the block types into which the input solid can be decomposed. There may be several solutions for each input file, and your program needs to output only one of them.

## EXAMPLE INPUT AND OUTPUT

TYPES.IN

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

BLOCK.IN

```
18
2 1 1
4 1 1
2 3 1
4 3 1
2 1 2
3 1 2
4 1 2
1 2 2
2 2 2
3 2 2
4 2 2
2 3 2
3 3 2
4 3 2
4 2 3
4 2 4
4 2 5
5 2 5
```

BLOCK.OUT

```
5
7  10  2 10  12
```

**Note:**

1. This input file BLOCK.IN describes the solid of the 'horse' in Figure 1.
2. Other solutions for the second line of the output file, which describes the types of the blocks used, are any of the following:

```
2  7  10  11  12
2  7  11  11  12
4  4   7   10  11
4  4   9   10   11
```

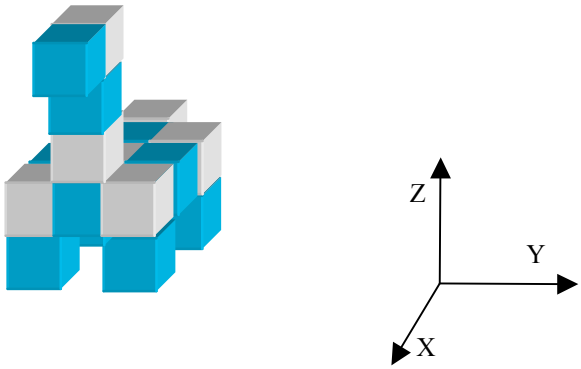


Figure 1. Horse

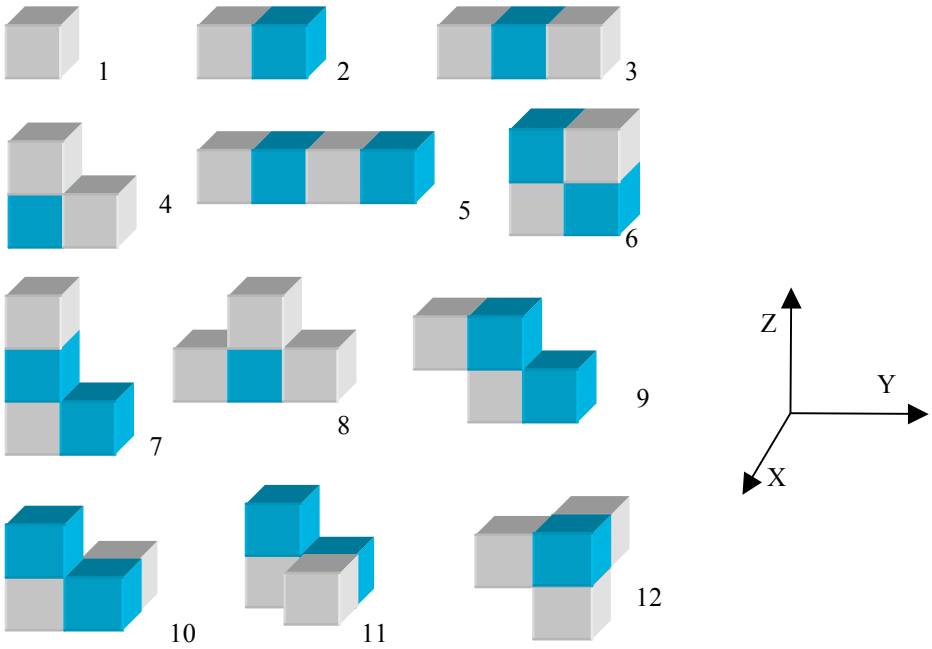


Figure 2. The 12 block types