RELAY (EST, 2 sec per test, 30 points)

A young programmer has written software for orienteering competitions. The file RELAY2.EXE contains his program for ranking the participants of second relay by their individual results. As input data, the program uses the files START.IN and FINISH.IN presenting some parts of Start and Finish protocols containing all the participants of second relay and possibly some others. First line of both files contains the number of competitors in the file. Each of the remaining lines consists of the number of the competitor and his (her) starting/finishing time (hours, minutes, seconds), in order of starting/finishing. The participants of first relay may have the numbers 100, 101, 102, …, 199; the participants of second relay the numbers 200, 201, 202, … , 299 etc. Maximal possible number is 499. For example:

```
START.IN                FINISH.IN
6                       5
203 13 12 7             104 13 48 59
201 13 12 10            201 13 52 40
305 13 15 8             305 13 53 1
202 13 24 31            202 13 59 47
204 13 48 59            203 15 25 21
301 13 52 40
```

The output file RELAY2.OUT must contain the numbers of the participants of second relay having received positive result (i.e. running time not more than 2 hours), ranked by their individual results. If the results are equal then the participant having finished earlier must be higher in the table. In case of our example the output must be

```
202
201
```

The program RELAY2.EXE is not completely correct. Your task is to test the program, to diagnose the mistakes in it and to write in your programming language a program MYRELAY that makes the same mistakes as the prototype. Your program will be tested with some positive tests (where RELAY2 computes correct results) and some negative tests (where the output of RELAY2 is not correct). Full points for a positive test will be given if your program gives correct output. In case of negative test you get full points if your program gives the same output as RELAY2 and half of the points if your output has correct format and is wrong but different from the output of RELAY2. You get no points for a negative test where your program computes the correct result (this indicates an error in RELAY2 that you did not detect). The half-points will be given only in the case if your program fails not more than one time with positive tests.

Your program must not incorporate the original RELAY2.EXE nor any part of it. It is also forbidden to call RELAY2.EXE from your program. If such violation of the rules will be detected by the judges, your score for the entire problem will be 0.

All the test cases contain only correct (i.e. possible in real competition) data. All participants of second relay in FINISH.IN occur in START.IN, but some participants having started can be not in Finish protocol. In all test cases the output of RELAY2.EXE has right format, i.e. contains one integer on each line. In all test cases the correct output and the output of RELAY2.EXE contain at least one and not more than 100 participants.

Fruits (FIN, 10 sec per test, 30 points)

A reasonably well-known mathematics professor has the following habit. When he goes shopping and buys fruit, e.g. oranges, he always likes to buy exactly 1 kg, if it is possible. As you admire the professor, you would like to do the same. However, the professor is able to do the necessary calculations in his head, but you will need to resort to your portable programmable calculator.

You are to write a program, which helps you to do the following: when you go shopping, you can weigh all fruits of desired type. Given as input a sequence of weighs for fruit in grams, your program should find out whether it is possible to buy exactly 1 kg. If so, your program must compute the combination of fruits. In order to avoid getting thrown out of the shop (or the shop closing) before your shopping is done, your program must decide all inputs in a limited time.

INPUT DATA

On first line of input file FRUITS.IN, the number of fruit of desired type n, an integer ranging from 1 to 50. On n following lines one integer (weight of a fruit), each ranging from 1 to 1000, for example:

```
5
200
300
200
400
500
```

OUTPUT DATA

Write into the output file FRUITS.OUT

```
NO
```

if it is impossible to buy exactly 1 kg, otherwise a sequence of given weights in any order which sums up to exactly 1000, writing one number on each line, for example

```
500
200
300
```

Championship in Basketball (SWE, 2 min per test, 40 points)

Here you have the result from BSCB (Baltic Sea Championship in Basketball). As you can see the team from Lithuania won the victory, winning all four games. The team from Sweden made a bad tournament and was beaten in all games.

LIT	4	0	344 - 267
LAT	3	1	368 - 287
EST	2	2	396 - 397
FIN	1	3	414 - 473
SWE	0	4	267 - 365

You read the table in this way:

Column	Explanation
1	The name of the team (always the same and in the same order and never used in problem).
2	The number of won games.
3	The number of lost games.
4	The number of made points.
5	The number of lost points.

Your program will be given the table and 20 matchpoints (i.e. points collected by each team in each game) P_i ($1 \le P_i \le 200$, **all different**). Then it will have to find the **unique** results of every game for filling a table like this:

	LIT	LAT	EST	FIN	SWE
LIT	-	48	106	120	70
LAT	41	-	111	137	79
EST	102	89	-	123	82
FIN	66	97	117	-	134
SWE	58	53	63	93	-

For instance, the match LAT - SWE closed 79 - 53.

INPUT DATA

There are at first five rows in input file MATCH.IN, with four numbers in each (compare with the table given above).

```
4 0 344 267
3 1 368 287
2 2 396 397
1 3 414 473
0 4 267 365
```

And then in a single line, the 20 matchpoints in **some** order.

```
41 48 53 58 63 66 70 79 82 89 93 97 102 106 111 117 120 123 134 137
```

The teams in input table are not necessarily ranked by final result.

OUTPUT DATA

In the output file MATCH.OUT must be the gameschedule without text, written with at least one blank between numbers:

```
0  48 106 120  70
41  0 111 137  79
102  89 0 123  82
66  97 117 0  134
58  53  63  93  0
```

Use zero (0) in stead of -. Remember that in the testcases there is always a **unique** solution.