Problem A: Watch the Skies!

Air PC, an up-and-coming air cargo firm specializing in the transport of perishable goods, is in the process of building its central depot in Peggy's Cove, NS. At present, this depot can handle at most one airplane on the ground at a time; however, given the increasing volume of business, up to 8 airplanes may arrive simultaneously at this terminal at a particular time T and need to land to offload cargo. Each of these airplanes has a servicing time st (in minutes) required to land, offload, and take off again. Safety-conscious company that it is, Air PC always makes sure that its airplanes carry enough fuel that any landing-order of the airplanes is possible, *i.e.*, does not result in crashes due to one or more airplanes running out of fuel while waiting to land. However, courtesy of a contract clause devised by an over-zealous (and now former) junior sales manager, all landing-orders are not equally profitable – this is so because each airplane also has a deadline time dt (in minutes) such that if the airplane is not landed or in the process of landing by time $T + dt$, Air PC must pay a perishable-cargo spoilage penalty sp (in dollars) to the customer.

Write a program which, given a set of $n \geq 1$ simultaneously-arriving airplanes with their respective servicing and deadline times and spoilage penalties, determines the landing-order that minimizes the total spoilage penalty that must be paid out by Air PC. The input will consist of a file in which the first line is the number of inputs and each input on n airplanes consists of $(n + 1)$ lines such that line 1 specifies n and lines 2 to $(n + 1)$ specify the name, st, dt, and sp for each airplane in that input. Each airplane name is a character string with no embedded spaces and st, dt, and sp are positive, non-zero integers. You may assume that all input files are formatted correctly and that each input file specifies an instance of the problem that has exactly one optimal landing-order.

Sample input (available as file "A.in"):

```
3
2
PC1175 30 20 100
PC1184 15 20 100
3
PC1175 30 20 350
PC0099 10 25 1000
PC1184 25 25 200
3
PC1175 30 30 350
PC0099 45 15 400
PC1184 25 25 100
```

Sample output: Note blank line after each input test case.

```
>>> Input #1:

Optimal Landing Schedule:
 1: PC1184 (penalty = $0) [Lands at time T]
 2: PC1175 (penalty = $0) [Lands at time T + 15 min]
>>> Total penalty: $0

>>> Input #2:

Optimal Landing Schedule:
 1: PC0099 (penalty = $0) [Lands at time T]
 2: PC1175 (penalty = $0) [Lands at time T + 10 min]
 3: PC1184 (penalty = $200) [Lands at time T + 40 min]
>>> Total penalty: $200

>>> Input #3:

Optimal Landing Schedule:
 1: PC1184 (penalty = $0) [Lands at time T]
 2: PC1175 (penalty = $0) [Lands at time T + 25 min]
 3: PC0099 (penalty = $400) [Lands at time T + 55 min]
>>> Total penalty: $400
```

Problem B: OK Sudoku?

Sudoku is a popular puzzle game where one puts numbers on a grid. The grid has 9 rows and 9 columns, and is divided into 9 blocks, each with 9 squares. In each square one of the numbers from '1' to '9' is placed. For example, the following is an example of a completed Sudoku grid:

```
123 456 789
456 789 123
789 123 456

234 567 891
567 891 234
891 234 567

345 678 912
678 912 345
912 345 678
```

The grid is considered *valid* if all of the following rules are met:

1.Each row contains the numbers from 1 to 9 (inclusive) exactly once each.
2. Each column contains the numbers from 1 to 9 (inclusive) exactly once each.
3. Each 3 by 3 block, of which there are 9, contains the numbers from 1 to 9 (inclusive) exactly once each.

Thus, the above example satisfies these rules and can be considered as valid. The grid is considered invalid if any of the rules are broken. For example, the following is an example of an invalid grid:

```
123 456 789
912 345 678
891 234 567

789 123 456
678 912 345
567 891 234

456 789 123
345 678 912
234 567 891
```

In this grid, the first block contains the number 1 three times, the number 2 twice, the number 9 twice, and is missing the numbers 4, 5, 6, and 7. For these reasons, among others, the grid is invalid. Your task is simple, to write a validity tester for Sudoku grids.

Input

Your input will consist of 11 lines of text. Lines 1 to 3, 5 to 7, and 9 to 11 each contain 3 groups of 3 integers separated by a single space. Each line is a row in a Sudoku grid, and the 9 integers are the values in the 9 columns. Lines 4 and 8 are blank.

Output

Your output will consist of a single word. Output the word `valid` if the input represents a valid Sudoku grid, otherwise output the word `invalid` if the input does not represent a valid Sudoku grid.

Example Input

```
111 222 333
222 333 444
333 444 555

444 555 666
555 666 777
666 777 888

777 888 999
888 999 111
999 111 222
```

Example Output

```
invalid
```

Problem C: Second Shortest Path

A number of agencies are after Max. Max is known to travel various graphs. His practice was to always choose a shortest path from a start vertex to his end vertex but he is suspects that the agencies are aware of this and are watching shortest paths. So Max has decided to use second shortest paths. Write a program to help Max find second shortest paths. In this problem only simple paths are considered, that is, paths which visit a vertex at most once and do not contains "cycles".

A shortest path P is a sequence of edges whose total length is less than or equal to the total length of any other path from the start to the end. A second shortest path Q has length less than or equal to all other paths except P. The length of Q might be equal to that of P in which case it is more likely that Max will be caught.

Input

The standard input will contain a positive integer n and then an n by n array of non-negative integer values. The i,j entry is the length of an edge from vertex i to vertex j or 0 if there is no such edge. Then there will be one or more pairs S,E of start,end verticies. All verticies will be numbers in the range 0,1,...,n-1.

Output

Your program is to print a second shortest path from S to E on one line for each such pair. There may not be two different paths (or even one) from S to E in which case the output should be as shown in C.out. If there is more than one shortest path any one of them will be accepted as second shortest.

Sample Input (6 lines available as file C.in)

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

Diagram for sample input:

```
   2       2
0 ---> 1 <--->3
 \     |     /
  \    |    /
   \5  |2  /1
    \  |  /
     \ v /
       2
```

Sample Output (4 lines available as file C.out)

```
SS 0 to 3: 0 1 2 3
SS 1 to 2: none
SS 3 to 0: none
SS 3 to 3: none
```

Note that the second shortest path from s to f in graph G is the shortest over all paths from s to f over all graphs which are the same as G but are missing one of the edges from the shortest path s to f in G.

Problem D: Hansel and Gretel

Hansel and Gretel are wandering in the woods when an evil witch gives them a bucket of chocolate. In the bucket there is a variable number of chocolate bars, from 0 to 1000. The bars can weigh any amount, from 1 to 1000 grams, but will always be an exact integer value. That is, bars can weigh 1 gram, 2 grams, 3 grams, and so on, and will never have a fractional weight such as 1.5 grams or 3.75 grams. The witch is evil because it is very hard for Hansel and Gretel to divide the chocolate fairly, causing them to get into very bad arguments. Your task is to help prevent Hansel and Gretel from fighting by dividing the chocolate for them.

Input

The first line of input will contain a single integer, *b*, indicating the number of buckets of chocolate you will have to divide. After this, there are b sets of data, one for each bucket. Each set of data begins with an a single integer, *n*, indicating the number of chocolate bars in the bucket. Then, the next n lines will each contain a single integer indicating the weight of a bar of chocolate.

Output

You are two output two integers, on the same line, with a single space between them. The first integer is the amount of chocolate that Gretel gets and the second integer is the amount of chocolate that Hansel gets. You must ensure that the difference between Hansel's and Gretel's amounts is minimal for each bucket. If the bucket can not be divided into two even sized piles, Gretel will get the larger pile.

Example Input

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

Example Output

```
7 7
2 1
```

Problem E: Party Ditches

Politics has become very nasty. In a nearby country with only 2 parties voters feel pressured to move if they happen to live in a district where the majority votes for the other party.

One community has the idea to dig a big ditch through the town separating the voters for one party from those for the other. You are to write a program to determine if this is possible using a straight line.

The input is a series of one or more test cases separated by blank lines. Each test case has two lines of one or more integer x,y coordinate pairs. The first line gives the coordinates of each voter for one party. The second line gives those for the voters for the other party. Your program should print `YES` if it is possible to draw a straight line separating the points on the first line on one side from those on the second line, and `NO` otherwise. See file E.in for sample input and E.out for the corresponding output expected. However all programs should use standard input/output and not any named files.

The distance of a point x,y from a line through x_1,y_1 and x_2,y_2 is:

$$(x*(y_1-y_2) + y*(x_2-x_1) + x_1*y_2 - y_1*x_2) / \mathrm{sqrt}((x_2-x_1)^2 + (y_2-y_1)^2)$$

and the numerator $x*(y_1-y_2) + y*(x_2-x_1) + x_1*y_2 - y_1*x_2$ is:
> 0 if the shortest path from x_1,y_1 to x_2,y_2 and then to x,y involves a left turn at x_2,y_2
= 0 it the points are colinear
< 0 if a right turn occurs.

Sample Input (available as file E.in with 5 lines)

```
1 1  3 3
2 2  4 4

2 1  6 1  5 1  4 1  3 1  6 2  4 2  6 3  5 3  6 4
4 3  3 3  3 2
```

Corresponding Output (file E.out with 2 lines)

```
NO
YES
```

Problem F: Enclosing Boxes in the Plane

For various reasons it is often useful to discover information about rectangular (or square) regions in the plane, and sometimes (as in our case) the rectangular regions (which we will simply call "boxes") can even have integer coordinates.

A box in the plane is determined by the (integer) coordinates of its lower left and upper right corners. If two boxes intersect (have at least one point in common), we are interested in finding the smallest box that encloses these two intersecting boxes.

Write a program that reads in the coordinates of the lower left corner and upper right corner of two such boxes from a single line of input and then produces a corresponding single line of output. Thus, a sample line of input would be as follows:

```
1 1 3 3 4 4 7 7
```

As you can see, there are 8 numbers per line of input. This line of input describes two boxes, the first with lower left corner (1,1) and upper right corner (3,3) and the second with lower left corner (4,4) and upper right corner (7,7). Note that these two boxes do not intersect.

If the two boxes do in fact intersect, the output line looks like this:

```
1. Enclosing box: (1,2) (7,8)
```

in which the number at the left is the number of the current line of input (since there may be several lines of input) and the first pair of integers is the lower left corner of the enclosing box, while the second pair is the upper right corner.

If the two boxes do not intersect, the single line of output looks like this:

```
1. Boxes do not intersect.
```