The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

Problem #1: A Magic Square

For centuries, mathematicians have been fascinated with the Magic number squares. A Magic square is an $N \times N$ array of numbers consisting of the positive integers from 1 to N^2 arranged such that each integer is used exactly once, and the sum of the numbers in each horizontal, vertical and corner-to-corner line is the same. You will be given the size of the array (N) and an $N \times N$ array of integers. One cell in each row will be marked as a '0'. You must determine, if possible, whether or not the '0' values can be replaced by positive integers so that the array is a Magic square.

The input data begins with a line containing an integer N representing the number of rows in a square (1 < N <= 6). The next N lines represent the array of values, N integers per row.

If a solution is found, output the square array (N numbers per line). If no solution is found, output "no solution".

Sample Input:

```
5
24 8 17 0 15
0 5 14 23 7
13 22 0 20 4
10 0 3 12 21
2 11 25 9 0
```

Sample Output:

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

Sample Input:

```
4
8 13 0 2
0 3 11 7
5 0 14 6
10 12 1 0
```

Sample Output:

```
no solution
```

The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

Problem #2: Closest Visible Players

You are writing an AI for a 2 dimensional game. There are several players that move around a map trying to zap each other with lasers. The map is filled with walls that players may hide behind. Players cannot see through or zap through walls. Finally, lasers are more lethal the closer a player is to another player when zapping.

Your goal is to write an algorithm to find the most threatening players on a map with respect to a given player. The closest player is the most dangerous. However, players behind walls are not dangerous, since they cannot zap you. In short, you must find the closest visible players to your player in the game. Note: players do not obscure other players.

You will be given a list of each player's position and each wall's endpoints as Cartesian coordinates. The x-y components of these coordinates will be non-negative integers. Walls are line segments defined by their endpoints. Players are not allowed to occupy the same space as other players or walls. A player may come arbitrarily close to a wall so long as they never share the same coordinate. Walls may intersect other walls, forming 'x' like structures.

The first line of input contains the total number of players. Following this line, the players' positions are given, one per line. The first position listed is *your* player's position. A player's position is a Cartesian coordinate formatted

$x\ y$

where x and y are non-negative integers and are separated by a space.

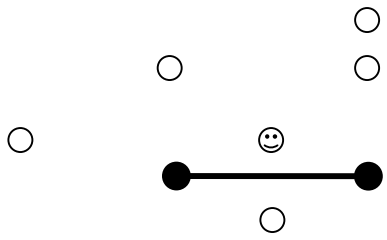
After all the players' positions have been given, the next line of input will contain the total number of walls. Following this line, a specification for each wall will be given, one specification per line. A wall's specification is composed of two endpoints formatted

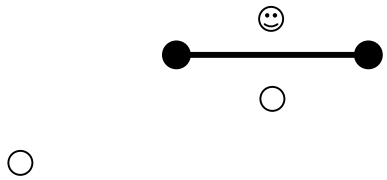
$x_1\ y_1\ x_2\ y_2$

where x_1, y_1, x_2, and y_2 are non-negative integers separated by spaces. x_1 and y_1 are the coordinate components for the first endpoint. x_2 and y_2 are the coordinate components for the second endpoint.

Output the distance to and the coordinates of each visible player. Output each player's information (distance and coordinates) pair on a separate line. Output the players in order from nearest to farthest, with the nearest reported first. When comparing and reporting distances, round to 2 decimal places. If two players are equally close after rounding, report first the one with the smallest x-coordinate. If these too are equal, then report first the one with the smallest y-coordinate. If no opponents are visible, report, "No opponents found."

Samples

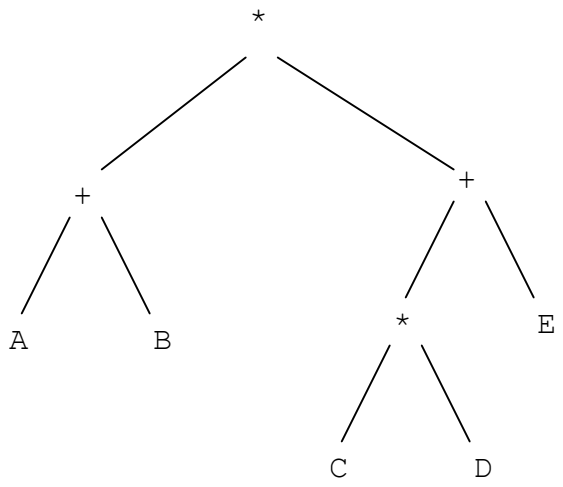
Input	Output	Picture (Smiley face is your player)
5 4 3 0 3 4 1 2 4 5 4 1 5 2 2 2	1.41 2 4 1.41 5 4 2.24 5 5 4.00 0 3	

Input	Output	Picture (Smiley face is your player)
3 4 3 0 0 4 1 1 2 2 5 2	No opponents found.	

The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

Problem #3: Word Puzzle

You are to write a program that processes a special type of expression tree for which each leaf node is an uppercase letter and each non-leaf node is either an operator '+' or an operator '*' and it must have a non-empty left subtree and a non-empty right subtree. Here is an example



In such an expression tree, a '*' operator represents a concatenation, and a '+' operator represents an alternation – take either the string represented by the left subtree or the string represented by the right subtree. Your program should produce all strings represented by the expression tree and display the total number of such strings. For the example above, a correct output would be

```
ACD
BCD
AE
BE

4 strings
```

The order that you display these strings is not important.

The first line of input data contains an integer n representing the number of nodes in the tree. You may assume that $1 \leq n \leq 40$. Each of the next n line represents one node with the following information:

the node number (between 1 and n)

the data in the node ('+', '*' or an uppercase letter)

the node number of the left child if one exists

the node number of the right child if one exists

There is a single space between two pieces of data. Note that nodes may appear in any order in the input.

Sample Input

```
9
7 C
6 B
1 + 3 4
8 * 9 1
5 D
2 A
3 * 7 5
9 + 2 6
4 E
```

Sample Output

```
ACD
BCD
AE
BE

4 strings
```

The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

Problem #4: The Drunken Yankees Fans

A group of New York Yankees fans, heartbroken after their team lost to the Red Sox, walk out of a bar and each begins wandering the streets of Manhattan. Each fan walks one block at a time in the directions north, south, east, or west (N, S, E, W). Your job is to determine whether or not each fan, at the end of his/her wanderings, ends up at the same bar he/she left.

The first line of the input will be a positive integer n representing the number of fans. Each of the n lines that follows contains the directions that each fan walked. The directions will be given as a string of characters (N, S, E, or W), separated by spaces. The maximum length of each line is 50 characters. Note that a fan may walk north and then turn around and walk south along the same street. Also, you may assume that there are no obstacles that would prevent the fan from walking in the indicated direction.

If the fan ends up at the same place, output the word "same". Otherwise, output "not same", followed by the ending position relative to the starting position. Output the number of blocks away in an east/west direction, followed by the number of blocks away in a north/south direction. For example, if the fan ended up 4 blocks east and 2 blocks south of the beginning position, output "not same 4E 2S". If the fan is 0 blocks away in one of the directions, suppress output for that direction.

Sample Input:
```
3
E S E N N W N W S W S S E N
S W N S W S E S
N E S S E N
```

Sample Output:
```
same
not same 1W 3S
not same 2E
```

The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

Problem #5: Road Rally

Write a program to help a road rally team plot the fastest route from one intersection to another in a city. The city is composed of intersections and roads that connect them. Each road connects exactly two different intersections; and for each pair of intersections, no more than one road connects the pair. An intersection may have any number of roads connected to it. Traffic may enter and exit an intersection from any road connected to it. Each road requires a certain amount of time to drive from one end to the other.

Traffic through an intersection is controlled by a traffic light. When an intersection's light is green for a road, traffic from that road, and that road only, may enter the intersection and turn onto any other road connected to the intersection. For a given intersection, its light is green for exactly one road at a time. So, traffic from exactly one road will be allowed to enter an intersection. To be fair, a light turns green for each road connected to the light's intersection in a round-robin rotation. Also, a light remains green equally long for each road at an intersection. The rotation and the duration of green lights for each traffic light will be given at the start of the rally, and will not change during the rally.

Our road rally will take place in the future, when cars will be able to start, stop, and reach cruising velocity instantaneously. If a car arrives at an intersection on a particular road at the same time the intersection's light turns green for that road, the car may immediately enter the intersection. Alternatively, if a car arrives at an intersection on a particular road at the same time the intersection's light turns red for that road, the car cannot enter the intersection, and will stop immediately and await the green.

Our car will start at one intersection, and will end in another. Since it begins <u>in</u> an intersection, it may immediately turn onto any road connected to that intersection. However, the car must enter the final intersection on a green. The coordinators of the road rally always ensure that there is a route between the starting and ending intersection.

The first line of input will contain the number of roads, N. The next N lines will contain a description for each road. Each road is described by a name and the time it takes to drive the road, separated by a space. Names will be no greater than 20 characters long, and will contain only alphabetic characters. Times will be

given as integers between 0 and 100 inclusively.

The next line of input contains the number of intersections, M. The next M lines contain a description for each intersection. The first intersection is the starting intersection. The last is the ending intersection. An intersection's description consists of an amount of time that its light remains green for each road, and a list of roads connected to that intersection. The time and road names are separated by a space. The time the light remains green is an integer between 1 and 100 inclusively. The list of roads specifies the rotation of that intersection's light from left to right. Consider the following intersection description:

3 A B C

The light at this intersection would stay green for 3 units of time for each road starting with A. Its rotation is then: A, B, C, A, etc.

Output the names of the roads on the fastest route in order from start to finish, and report the total time it will take. If there are multiple routes that are equally as fast, report the route that comes first alphabetically. For instance, if the following routes are equally fast

apples papayas 3
apples pairs 3

report "apples pairs 3" since it comes first in alphabetic order.

Sample input:

```
5
Vernon 2
Fairfax 3
Chester 2
White 8
Panama 2
4
1 Vernon Fairfax
1 White Chester Vernon
5 Chester Panama Fairfax
4 White Panama
```

Sample output:

```
Vernon Chester Panama 6
```

The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

Problem #6: Change the World

The Universal Change Machine Company has designed a vending machine that can be used in any country with any set of coin denominations. The vending machines have been programmed to give change under the principle "give highest coin". That is, when giving a certain amount of change, the vending machine dispenses as many coins of the largest denomination as possible, followed by the next highest denomination and so on. Note: Each coin set contains a coin of denomination 1 so that it is possible to dispense any change amount.

The problem is that customers from certain countries have been complaining that they have an excess number of coins jingling in their pockets and that the machine could in fact dispense a smaller number of coins.

For example, in Chordland, the set of coin denominations are 1, 4, and 5. The vending machine would give change for 12 cents by dispensing the following 4 coins: 5, 5, 1, 1. However, the same change can be given using only 3 coins: 4, 4, 4.

You would like to determine whether or not, for a given set of coin denominations, the vending machine will dispense the minimum number of coins using its current "give highest coin" principle. The largest change amount to be dispensed is 1000, thus it is sufficient to check all change amounts from 1 to 1000.

The input will contain data for a number of different coin sets, one coin set per line. Each line will contain the number of coin denominations, followed by a space, then the coin denominations in descending order, separated by single spaces. A line containing a single 0 signals the end of input and should not be processed.

For each coin set, test all change amounts from 1 to 1000. If the vending machine always gives the minimum number of coins possible, output the word "okay". Otherwise, output "not okay", followed by the first value for which the machine does not dispense the minimum number of coins.

Sample Input:

```
4 25 10 5 1
3 10 6 1
4 4 3 2 1
0
```

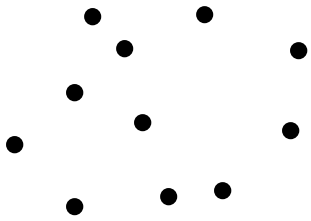
Sample Output:

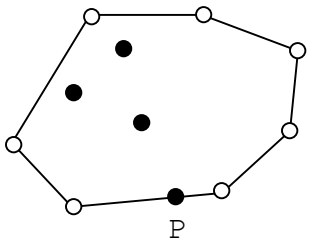
```
okay
not okay 12
okay
```

The Northeast Regional Competition
of the
2004-2005 ACM International Collegiate Programming Contest
sponsored by IBM
Eastern Preliminary: Western New England College, Springfield, MA
October 30, 2004

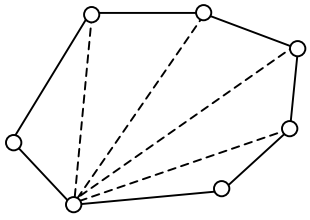
Problem #7: The Hull Area

Given a set of n points in the Cartesian plane, the convex hull is the set of vertices for the smallest convex polygon containing those points. For example, a set of points is shown on the left and the corresponding convex hull is shown by the open circles in the figure on the right. Notice that, in the figure, the point labeled P is collinear with two vertices of the convex hull, but is not part of the convex hull itself since it is not a vertex of the polygon.





You must determine the set of vertices that determine the convex hull, as well as the area of the convex hull. In order to calculate the area of the convex hull, you may want to consider triangulating the polygon, calculating the area of each triangle and then finding the sum of the areas of the triangles.



You may use Heron's formula to determine the area of any triangle as a function of the lengths of its sides a, b, and c.

$$area = \frac{\sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)}}{4}$$

(Note: In general, Heron's formula can sometimes produce round-off error. However, the input data provided will not cause unnecessary round-off error using Heron's formula.)

The input data begins with a line containing an integer $N \geq 3$ representing the number of points in the plane. The next N lines represent the points in the plane. Each line will contain 2 integers representing the x- and y-coordinates, respectively, separated by a space.

Your program must output the set of vertices for the convex hull in counterclockwise order. The point at which you start does not matter. Output one vertex per line, formatted as *x y* (where *x* represents the x-coordinate and *y* represents the y-coordinate). Then calculate and output the area of the convex hull, rounded to 2 decimal places.

Sample Input:

```
8
0 3
-5 6
-10 0
1 5
8 0
0 -4
3 -1
2 6
```

Sample Output:

```
0 -4
8 0
2 6
-5 6
-10 0
111.00
```