

1. [1993 Berkeley Contest] You’ve probably seen those “cryptarithmic” puzzles, in which one gets problems like these

```

    hello      too      i
+  there      + much    +  am
-----
    world      beer      not
    
```

and is asked to assign digits to each letter so that the resulting addition is correct. Each digit from 0 to 9 must be used at most once, and the leading digits may not be 0. In the above cases, for example, we can get the solutions

```

    56442      611      8
+ 15606      + 7942    + 97
-----
    72048      8553     105
    
```

and other solutions are possible, as well.

Write a program to find a solution to cryptarithmic problems for which the input consists of triples of strings each containing up to 128 lower-case letters and the output is in the form given in the sample below.

For the input	Produce the output
<pre> hello there world too much beer i am nuts </pre>	<pre> hello 56442 + there + 15606 ----- world 72048 too 611 + much + 7942 ----- beer 8553 i + am ----- nuts No solution </pre>

Precise horizontal spacing is not important; however, there must be a blank line after each problem, and the line of dashes must be two longer than the sum. You may assume the sum is at least as long as each addend. When multiple solutions are possible, prefer the one with the smallest units digit in the first addend. If there is more than one such solution, prefer the one with the smallest units digit in the second addend. If there are still multiple solutions, prefer the one with the smallest digit in the 10’s place of the first addend, and so forth.

2. [From the Southern California 2007 regional contest] You're given an unlimited number of pebbles to distribute across an $N \times N$ game board ($3 \leq N \leq 15$), where each square on the board contains some positive point value between 10 and 99, inclusive. A 6×6 board might look like this:

33	74	26	55	79	54
67	56	91	72	44	32
44	64	22	91	29	61
61	32	76	50	50	32
81	65	56	38	96	36
38	78	50	92	90	75

The player distributes pebbles across the board so that:

- At most one pebble resides in any given square.
- No two pebbles are placed on adjacent squares. Two squares are considered adjacent if they are horizontal, vertical, or even diagonal neighbors. There's no board wrap, so 44 and 61 of row three in the example above aren't neighbors. Neither are 33 and 75 nor 55 and 92.

The goal is to maximize the number of points claimed by your placement of pebbles.

Write a program that reads in a sequence of board descriptions from the standard input and prints to stdout the maximum number of points attainable by an optimal pebble placement for each. Each board description starts with an integer, N , giving the size of one side of the board. Next follow N^2 integers in the range 10–99. All input is in free format.

For each board, your program should print a line containing the maximum number of points one can get by optimally distributing pebbles while respecting the two rules.

Example:

Input	Output
5	572
71 24 95 56 54	683
85 50 74 94 28	2096
92 96 23 71 10	2755
23 61 31 30 46	
64 33 32 95 89	
6	
78 78 11 55 20 11	
98 54 81 43 39 97	
12 15 79 99 58 10	
13 79 83 65 34 17	
85 59 61 12 58 97	
40 63 97 85 66 90	
11	
33 49 78 79 30 16 34 88 54 39 26	
80 21 32 71 89 63 39 52 90 14 89	
49 66 33 19 45 61 31 29 84 98 58	
36 53 35 33 88 90 19 23 76 23 76	
77 27 25 42 70 36 35 91 17 79 43	
33 85 33 59 47 46 63 75 98 96 55	
75 88 10 57 85 71 34 10 59 84 45	
29 34 43 46 75 28 47 63 48 16 19	
62 57 91 85 89 70 80 30 19 38 14	
61 35 36 20 38 18 89 64 63 88 83	
45 46 89 53 83 59 48 45 87 98 21	
12	
15 95 24 35 79 35 55 66 91 95 86 87	
94 15 84 42 88 83 64 50 22 99 13 32	
85 12 43 39 41 23 35 97 54 98 18 85	
84 61 77 96 49 38 75 95 16 71 22 14	
18 72 97 94 43 18 59 78 33 80 68 59	
26 94 78 87 78 92 59 83 26 88 91 91	
34 84 53 98 83 49 60 11 55 17 51 75	
29 80 14 79 15 18 94 39 69 24 93 41	
66 64 88 82 21 56 16 41 57 74 51 79	
49 15 59 21 37 27 78 41 38 82 19 62	
54 91 47 29 38 67 52 92 81 99 11 27	
31 62 32 97 42 93 43 79 88 44 54 48	

3. [1993 Berkeley Contest] You are given a set of opaque rectangles in three-dimensional space and asked to find whether a point-sized observer located at one point in that space can see another observer located at a second point. Each rectangle is either vertical (upright) and runs north-south or east-west, or is horizontal (parallel to the ground). The data comprise

- A positive integer N giving the number of rectangles.
- A sequence of $6N$ floating-point numbers giving the x , y , and z coordinates of two opposite vertices of each of N rectangles. Each rectangle edge will be parallel to an axis, so such a vertex pair uniquely determines a rectangle. As further consequences, each vertex pair will have the same coordinate along either the x , y , or z axis (the axis may differ from one rectangle to another) and each rectangle will lie in a plane parallel to the xy , xz , or yz planes.
- A sequence of $6K$ floating-point numbers, for some $K \geq 0$ (which is not explicitly given). Each group of 6— $x_0, y_0, z_0, x_1, y_1, z_1$ —gives the coordinates of two observation points, (x_0, y_0, z_0) and (x_1, y_1, z_1) .

The data are free-form. The rectangles may intersect each other. Two observer points are visible to each other if and only if their line of sight is not blocked by any rectangle. A line of sight is blocked by a rectangle if and only if neither observer lies in the same plane as that rectangle and the line segment joining them intersects the interior of the rectangle. You may assume no lines of sight will intersect edges unless one of the observers is actually on that edge.

The output of the program will consist of a sequence of K lines, each having one of the two forms

(x_0, y_0, z_0) is visible from (x_1, y_1, z_1) .

or

(x_0, y_0, z_0) is invisible from (x_1, y_1, z_1) .

where all the coordinates are printed using the C `printf` format "`%.2f`", leaving at least one space after each comma.

Example. (See figure on the next page.)

Input	Output
5	(1.50, 0.00, 2.00) is invisible from (-3.00, 1.25, 1.50).
0 0 0 1.5 2.25 0	(3.00, 2.00, 0.50) is visible from (1.50, 0.00, 2.00).
3.5 2.25 0 2.5 0 0	(3.00, 0.00, 2.00) is invisible from (1.50, 2.50, -2.00).
0 2.5 0 3.5 2.25 0	(2.50, 0.00, 2.00) is visible from (1.50, 2.50, -2.00).
2.5 0.75 0 3.5 0.75 2	
0 0 -4 0 2.5 4	
1.5 0 2 -3 1.25 1.5	
3 2 0.5 1.5 0 2	
3 0 2 1.5 2.5 -2	
2.5 0 2 1.5 2.5 -2	

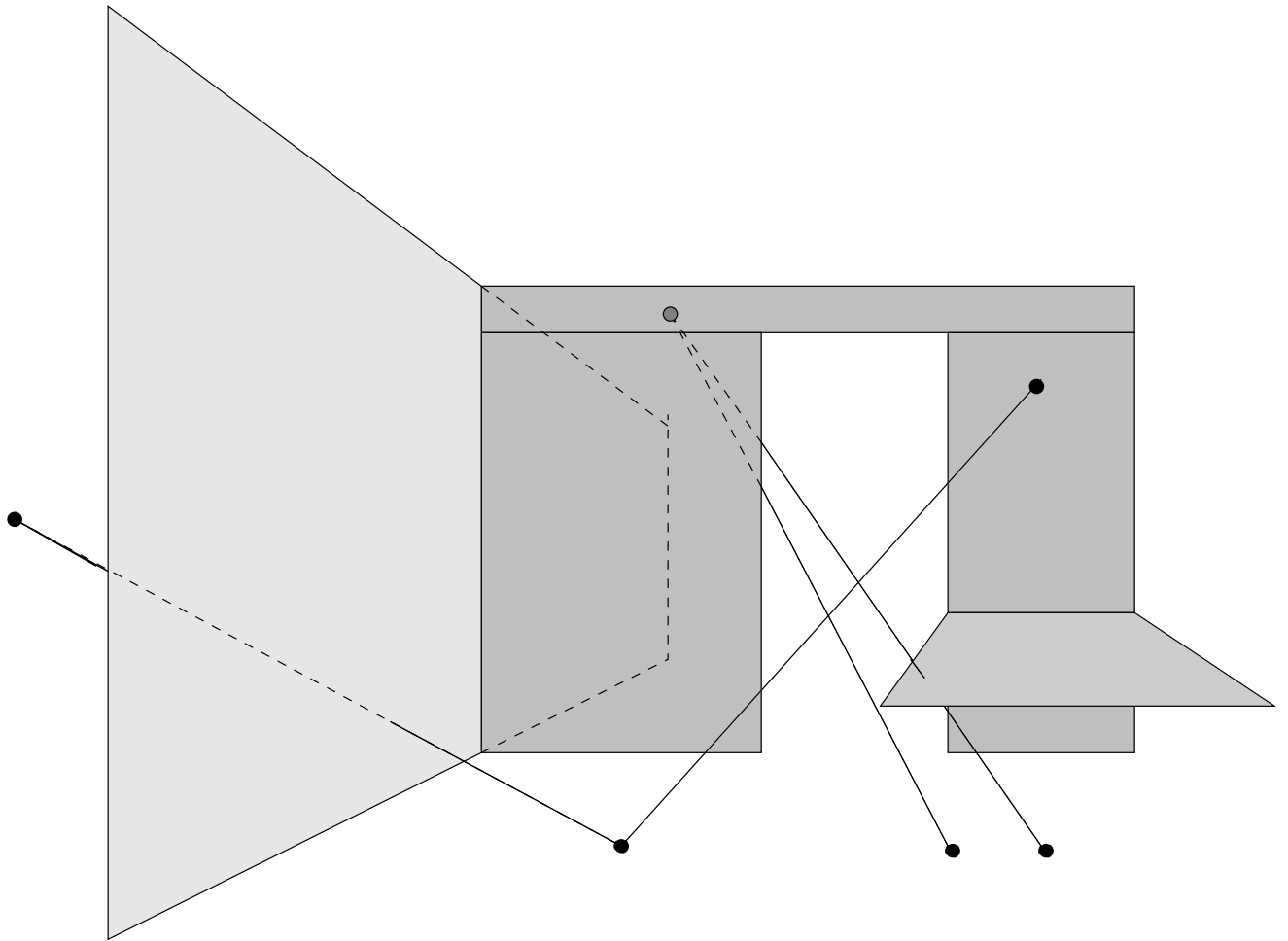


Figure 1: Rectangles and observation points described by the sample input. The positive z axis points up out of the page. The diagram also shows the lines of sight between the observation points. Hidden lines are dashed.