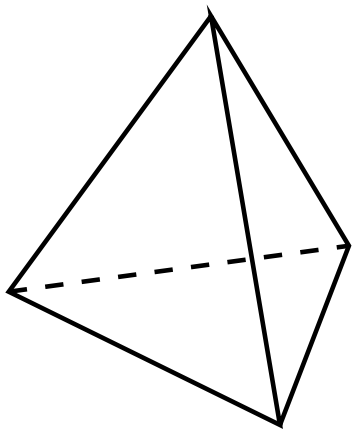
HKUST
Programming Contest
2005 Spring

Date: April 30th, 2005
Time: 13:00 – 17:00
Venus: CS Lab 3

Problem A. Coloring Pyramids



A regular tetrahedron is a regular and triangular pyramid. i.e. a pyramid composed of four equilateral triangles. Given *n* different colors, each face of the regular tetrahedron should be painted by one of the *n* colors. In this problem, you need to count the number of different regular tetrahedrons you can paint in this way. Two regular tetrahedrons are the same if you can rotate one of them such that the color of each face of one regular tetrahedron is the same as the color of the same face of the other.

Input

The input for each case is a single integer n in a single line where $1 \le n \le 10^7$. The input is terminated by an integer 0.

Output

For each case, output the number of different regular tetrahedrons could be painted with n different colors. Output the answer in a single line for each case.

Sample Input

```
1
2
3
4
5
9999999
10000000
0
```

Sample Output

```
1
5
15
36
75
833333000000141666645000001
833333333333425000000000000
```

Problem B. Touching Triangles

The problem is simple. Given two triangles in xy-plane, you just need to report whether the intersection of internal area (including boundary) of the triangles is non-empty.

Input

The input for each case is given as twelve integers x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5, x_6, y_6, in a single line. (x_1, y_1), (x_2, y_2) and (x_3, y_3) are the coordinates of the vertices of the first triangle. (x_4, y_4), (x_5, y_5) and (x_6, y_6) are the coordinates of the vertices of the second triangle. The absolute values of all the inputs are within 1000. And you can further assume that all input triangles are valid. i.e. No two points of a triangle are the same point and the three points of a triangle will not be collinear. The input is terminated by twelve integers of 0.

Output

For each case, output "`YES`" if the two input triangles have any point in touch, output "`NO`" otherwise.

Sample Input

```
0 1 0 2 2 1 1 2 1 0 2 2
0 0 1 9 2 8 0 0 9 1 8 2
0 0 9 0 0 9 1 1 6 1 1 6
0 4 4 0 4 4 2 4 6 4 2 6
0 4 4 0 4 4 5 4 7 4 6 5
0 4 4 0 4 4 4 5 4 8 9 -20
0 4 4 0 4 4 5 6 11 12 20 9
0 0 0 0 0 0 0 0 0 0 0 0
```

Sample Output

```
YES
YES
YES
YES
NO
NO
NO
```

Problem C. Robo Rally

Robo Rally is a robot race game. Each of the players controls one or more robots that race against one another. Player attempts to be the first to touch a series of flags by maneuvering a robot across a dynamic race course.

To control the movement of a robot, player programs a robot's operations by a series of program card. These program cards allow a robot to move forward or backward, and turn left or right.

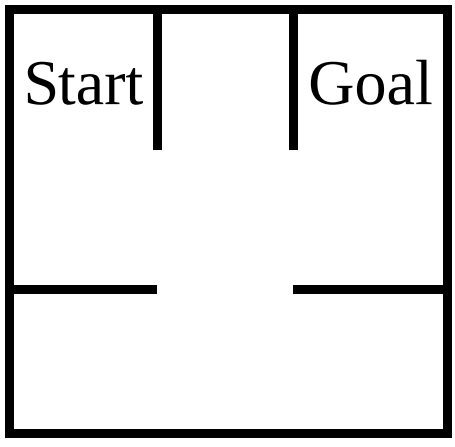
In this problem, it is not going to play a full version of the game. Instead it will play a simplified version.

Given a maze that consist of a rectangular array of square cells, each of which may have walls on the north, south, east and/or west sides of the cell. One cell will be identified as the starting point, and another will be identified as the goal. Your task is to find whether a possible way to program a robot from starting point to goal and minimize the number of program cards used. You may assume the initial direction of a robot is faced to south.

Type of program card:

Forward:	Move robot to next cell in which it is located in the front of robot and no wall existed between the current and next cell.
Backward:	Move robot to next cell in which it is located in the back of robot and no wall existed between the current and next cell.
Turn left:	Turn the front of robot to left.
Turn right:	Turn the front of robot to right.

Example:



Possible combination:
Forward, Turn Left, Forward, Forward, Turn Left, Forward

Input:
There will be one or more mazes to process in the input. The first line of input contains an integer T to indicate the number of test cases ($1 \leq T \leq 15$). For each test case, it is divides by several parts.

The first tine of test case contains 6 integers. The first two integer r and c ($1 \leq r, c \leq 10$) give the height (number of rows) and width (number of columns) of the maze (in cells). The next two give the position (row and column number) of the starting cell, and the last two give the position of the goal.

(View the maze as an array of cells, with the northernmost row being row 0, and the westernmost column being column 0. In the maze above, the starting cell is row 0, column 0, and the goal cell is row 0, column 2.)

Following the first six integers there will appear one integer for each cell, in row major order. There will have r number of lines, each with c number of integers. The value of each integer indicates whether a cell has a wall on its eastern side (1) and whether it has a wall on its southern side (2). For example, a cell with no eastern or southern wall has a value of 0. A cell with only a southern wall has a value of 2. A cell with both an eastern and a southern wall has a value of 3. The cells on the periphery of the maze always have appropriate walls to prevent the robot from leaving the maze; these are not specified in the input data.

Output:
For each test data, print an integer to represent the minimum number of program cards required to walk from starting point to goal. If no such path exists, instead you may print "`-1`" to represent this situation.

Input Sample

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

Output Sample

```
6
5
30
-1
```

Problem D. The Settlers of Catan

In the game of “The Settlers of Catan”, players represent the leader of a group which is attempting to settle the island of Catan. The island is constructed by nineteen tiles that placed on the table, each representing a different type of terrain. For each turn, player rolls two dice. Each tile containing a token with the same number as the total on the dice “produces” resources. There are six different types of terrain; forest (which produces lumber), pasture (wool), fields (grain), hills (bricks), mountains (ore) and desert (which produces nothing).

Resources are used for production. For example, a new settlement can be built with brick, wood, sheep, and grain resources and a new road can be built with wood and brick resources.

If you are already familiar with this game, don't hurry. Your mission is not to write a program to play a full version of game as it is difficulty to simulate the game within a short period of time. Instead, you are asked to write a program to play a modified version.

Rules:
Given a set of resources, each has victory points and it may used to produce something. The result product also has victory points. In this version of game, you are asked to achieve the maximum victory points based on a given set of resources.

Resources:

Resources	Victory points
Bricks	12
Grain	8
Sheep	10
Wood	8
Ore	16

Production:

Product	Victory Points	Building cost
Road	50	Brick + Wood
Settlement	100	Wood + Brick + Grain + Sheep
City	250	Settlement + Sheep * 2 + Ore * 3

Additional rule:
You may exchange any resource by using two units of resources with same type. For example, if you lack a unit of brick to build a road and you have two units of ore. You may use two units of ore to exchange a brick to build a road.

The objective of the game is to gain victory points. The one who gain the highest victory point wins the game. You are now asked to find out the highest victory points could be gain from a set of resources.

Example:

Suppose you have 2 Bricks, 1 Grain, 3 Sheep, 5 Woods and 2 Ores.

The maximum victory points could gain is 308 by:
1. Build a settlement
2. Build a road.
3. Two units of wood is used to exchange a Ore.
4. Build a city.
5. A unit of wood is leave at the end of game.

Input:
The first line of the input contains an integer T to indicate the number of test cases ($1 \leq T \leq 50$). The next following T lines, each line contains 5 integers B, G, S, W, O ($0 \leq B, G, S, W, O \leq 10$) to indicate the number of brick , grain , sheep, wood and ore respectively.

Output:
For each test case, display the maximum victory points should be achieved.

Sample Input
```
4
2 1 3 4 2
2 2 1 1 2
6 10 8 0 6
9 9 9 9 9
```

Sample Output
```
300
158
700
1150
```

Problem E. Packing Rectangles

(Acknowledgement: This question is extracted from IOI95)

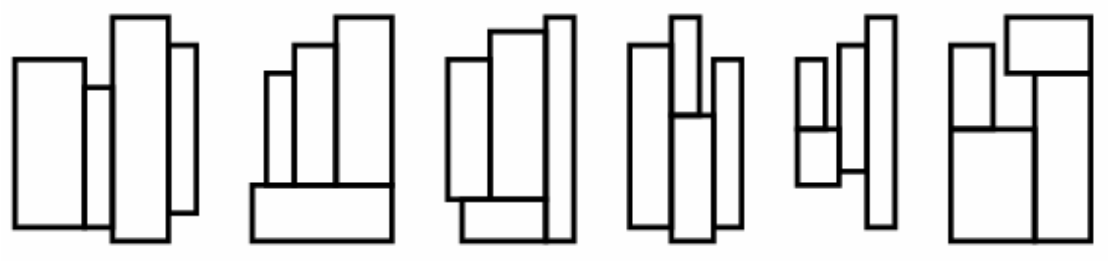


Figure 1: The six basic layouts of four rectangles

Four rectangles are given. Find the smallest enclosing (new) rectangle into which these four may be fitted without overlapping. By smallest rectangle we mean the one with the smallest area. All four rectangles should have their sides parallel to the corresponding sides of the enclosing rectangle. Figure 1 shows six ways to fit four rectangles together. These six are the only possible basic layouts, since any other layout can be obtained from a basic layout by rotation or reflection. There may exist several different enclosing rectangles fulfilling the requirements, all with the same area. You have to produce all such enclosing rectangles.

Input

The first line of the input is the number test cases. Each test case consists of four lines. Each line describes one given rectangle by two positive integers: the lengths of the sides of the rectangle. Each side of a rectangle is at least 1 and at most 50.

Output

The output of each test case consists of several lines. The first line contains a single integer: the minimum area of the enclosing rectangles. Each of the following lines contains one solution described by two numbers p and q with $p \leq q$. These lines must be sorted in ascending order of p, and must all be different.

Sample Input

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

Sample Output

```
40
4 10
5 8
```

Problem F. Minesweeper

Most of you know minesweeper. It is a single player game that sits in MS Windows, a pretty good time killer, especially during a programming contest.

The game starts with a board with $m \times n$ cells. Some of them contain mines (explosive devices), but you cannot see. Your goal is to discover all the mines. When you click on a cell, if it contains a mine, then you lose; otherwise it shows on the cell the number of mine(s) around it. If there is no mine around the cell you click on, then the program will help you click on all the cells around it, since it is always safe to do so. The numbers give some information of where the mines locate. A typical outcome after several clicks of the board is shown below, which is achieved by clicking on coordinates (0,5), (3,14) and (15,0):

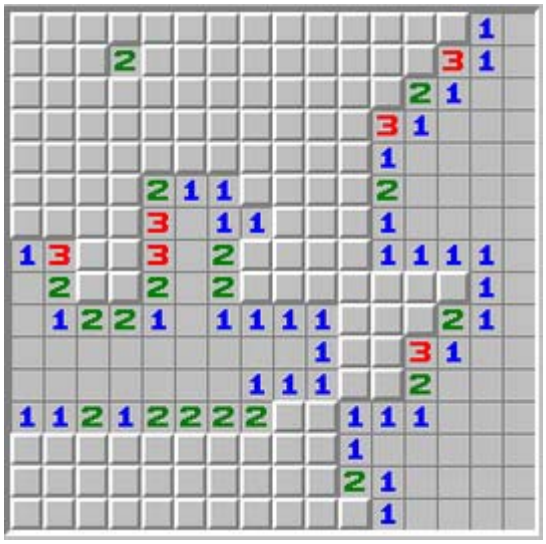


Figure 2

One can easily see that cells like (3,7) and (10,0) contain mines. One may take some time to see that (3,2) also contains a mine, since (2,3) says there are 2 mines around it, but they cannot be (1,2) and (2,2) since (1,3) says only one mine is next to it.

Be relaxed. We are not going to ask you write a program to solve the game. Here we only require your program to print the board like figure 2, given the locations of all mines and clicks.

Input

There are multiple test cases. Each test case starts with two numbers $1 \le m, n \le 100$ indicating the dimension of the board, or $m = n = 0$ indicating the end of input. m and n is the width and height of the board respectively. Then an integer $b \ge 0$ shows the number of mines on the board. It is then followed by $2b$ integers $u_1, v_1, u_2, v_2, \ldots, u_b, v_b$, where (u_i, v_i) is the location of the ith mine. Lastly, an integer $c \ge 0$ followed by $2c$ integers show the positions of the c clicks. Notice that the bottom left cell is always (0, 0) and the top left (0, n-1). You are assured that no clicks are on the mine sites.

Output
For each test case, print the outcome of the board after the clicks. For each cell, print either an integer showing the number of mines around it, or "*" if no information is known for it. See the sample outputs. Print an empty line between two cases.

Sample Input

```
3 3
2 0 0 2 2
1 1 1
9 9
10 0 4 1 8 2 8 3 3 3 5 3 8 5 6 7 5 8 2 8 6
4 0 0 0 8 8 0 8 8
5 3
1 2 0
1 0 1
0 0
```

Sample Output

```
***
*2*
***

1***10000
****21111
*********
****212**
****2011*
111*1001*
00111001*
000000011
000000000

00000
01110
01*10
```