Problem Sets

Problem A :	Lattice Point
Problem B :	Coin Exchange
Problem C :	The Minimum Number of Rooks
Problem D :	Mars Colony Construction
Problem E :	Puzzle for Friends and Enemies
Problem F :	A Version of Nim
Problem G :	Tri-Colored Intersections
Problem H :	Deadlock Detection

Problem A
Lattice Point
Input File: pa.dat

A lattice point is a point (x, y) in the 2-dimensional xy-plane with $x, y \in Z$, where Z be the set of integers. Let

$$P(r) = \{(x, y) | x^2 + y^2 \leq r^2, (x, y) \text{ is a lattice point in the xy-plane}\}$$

and we denote $D(r)$ be the number of elements in $P(r)$. For each lattice point (x, y) in the xy-plane, let

$$S(x, y) = \{(u, v) | x \leq u \leq x + 1, y \leq v \leq y + 1\}$$

and

$$B(r) = \{(x, y) | x^2 + y^2 \leq r^2, x \text{ and } y \text{ are real numbers}\}.$$

Then it is easy to verify that when $r > \sqrt{2}$

$$B(r - \sqrt{2}) \subset \bigcup_{(x,y) \in P(r)} S(x, y) \subset B(r + \sqrt{2}).$$

We know that

$$Area\left(\bigcup_{(x,y) \in P(r)} S(x, y)\right) = \sum_{(x,y) \in P(r)} Area(S(x, y)) = \sum_{(x,y) \in P(r)} 1 = D(r).$$

Hence

$$\pi(r - \sqrt{2})^2 < D(r) < \pi(r + \sqrt{2})^2.$$

This implies

$$\pi(1 - \frac{\sqrt{2}}{r})^2 < \frac{D(r)}{r^2} < \pi(1 + \frac{\sqrt{2}}{r})^2.$$

It yields

$$\lim_{r \to \infty} \frac{D(r)}{r^2} = \pi.$$

So if we can calculate $D(r)$ for a large r, then we can estimate the value of π.

The following C function can be used to calculate the value of $D(r)$ within a reasonable amount of time when r is a small integer, say e.g., $1 \leq r \leq 10,000$.

```
long D(long r)
{   long x,y,count=0;
    for(x=-r ; x<=r ; x++)
        for(y=-r ; y<=r ; y++)
            if (x*x+y*y<=r*r)
                count++;
    return count;
}
```

It is easy to obtained $D(1) = 5$, $D(2) = 13$, $D(3) = 29$, and $D(10000) = 314159053$ using this program. Recall that $\pi = 3.14159\cdots$. Your task is to find $D(r)$ for a large r within a reasonable amount of time.

Input: There are five lines in the input file, the kth line contain an integer n_k $(1 \leq n_k \leq 100,000,000)$.

Output: List integer n_k in line $2k-1$ and the value of $D(n_k)$ in line $2k$ for $k = 1, 2, 3, 4, 5$.

Sample Input

```
1
2
3
10000
100000000
```

Output for the Sample Input

```
1
5
2
13
3
29
10000
314159053
100000000
31415926535867961
```

Problem B
Coin Exchange
input file: pb.dat

Let $a_1, a_2, \ldots, a_n$ be n relatively prime positive integers . A positive integer k has a representation by $a_1, a_2, \ldots, a_n$ if there exist non-negative integers $x_1, x_2, \ldots, x_n$ such that $k = x_1a_1 + x_2a_2 + \cdots + x_na_n$. The linear Diophantine problem of Frobenius is to determine the largest integer $g(a_1, a_2, \ldots, a_n)$ with no such representation.

The linear Diophantine problem of Frobenius has a very practical application. It is equivalent to the problem of coin exchange. Given sufficient supply of coins of denominations $a_1, a_2, \ldots, a_n$, determine the largest amount which cannot be formed by means of these coins.

Mathematical problems, such as this, are usually difficult to solve. We shall consider a simple case in which $n = 2$. In addition to $g(a_1, a_2)$, we shall also want to find $n(a_1, a_2)$ the number of positive integers that cannot be represented by a_1 and a_2.

Write a program to compute $g(a_1, a_2)$ and $n(a_1, a_2)$. The following theorem may help you in designing your program.

> **Theorem** A positive number $k = qa_1 + r$, $0 < r < a_1$, can be represented by $a_1, a_2, \ldots, a_n$ if and only if $k \geq t_r$, where t_r is the smallest positive integer which has the same residue r modulo a_1 and can be represented by $a_2, a_3, \ldots, a_n$.

Input

The input file will consist of one or more cases. Each case contains two positive integers a_1 and a_2 in a line. The product of a_1 and a_2 is less than 2^{32}. A line containing two zeros follows the last case, and terminates the input file.

Output

For each case a_1 and a_2 in the input file, the output file should contain a line with two numbers $g(a_1, a_2)$ and $n(a_1, a_2)$ separated by a blank.

Sample Input

```
3 5
23 20
0 0
```

Output for the Sample Input

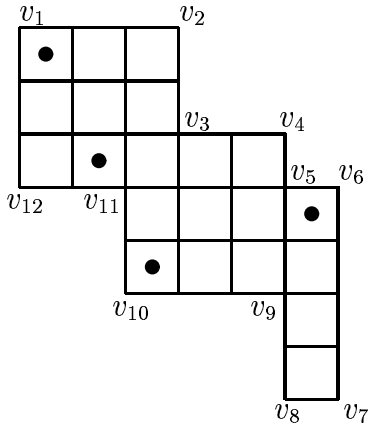
```
7 4
417 209
```

Problem C

The Minimum Number of Rooks

Input File: pc.dat

A sawtoothed chess-board is a chess-board whose boundary forms two staircases from left down to right. See the following figure for an example.



A square is *dominated* by a rook if it is in the same row or column with this rook. Your task is to determine the minimum number of rooks such that every square in a sawtoothed chess-board is dominated by at least one rook. For example, see the above figure again. It needs only four rooks located at the squares with a dot to dominate all of the squares.

Input: The input data will contain multiple test cases. Each test case will consist of less than 100 pairs of integers which represent the row and column indexes of the corner points, in clockwise order, in a sawtoothed chess-board starting from the upper-left most corner. For example, the corner points in the above figure are $v_1, v_2, \ldots, v_{12}$, respectively. Each corner point is represented by its x and y coordinates in the 2-dimensional xy-plane whose values are in the range from 1 to 100. For example, the indexes of v_1, v_2 and v_3 are (1,1), (1,4) and (3,4), respectively. The input for each test case will be terminated with a pair of zeros, which are not to be treated as the indexes of a corner point. An additional pair of zeros will follow the last test case.

Output: For each test case, determine the minimum number of rooks needed to dominate all of the squares in the sawtoothed chess-board described in the test case. Display, one test case a line, the test case number (they are

numbered sequentially starting with 1) and the number of rooks which are separated by one blank.

Sample Input

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

Output for the Sample Iutput

```
1 4
2 3
```

Problem D
Mars Colony Construction
Input File: pd.dat

In the year 2199, the earth is united into a Federation with the head of the Federation being the Prime Minister. The newly elected Prime Minister Q decided to colonize Mars by constructing cities and interconnection pathways between cities. According to the geographical survey, the Federation has chosen a square piece of flat land to construct N, $2 \leq N \leq 10,000$, cities. Each city h is given a coordinate (x_h, y_h) at the 2-dimensional xy-plane with the point of origin at the south-west corner of the piece of the land. Each city is in the shape of a circle with a radius of exactly 0.5 kilometer. The coordinate is the center of the city. It is known that the values of x and y coordinates of any city are in units of 1 kilometer and are positive integers smaller than or equal to $100,000$. No two cities have the same coordinates. Since Mars has unpredictable weathers, Prime Minister Q decided to build covered pathways between the cities. In order to speed-up the construction, simplify the construction plan and save budget, only north-south or east-west bounded pathways connecting exactly two cities are constructed. The height and width of each pathway is exactly 3 meters. In addition, no two pathways intersect except at the N cities. According to the above, Minister of Transportation T designs the following pathway constructing plan:

- If a pathway is built between two cities, then the the *distance* of this pathway is the actual distance outside the two city limits. The two cities are called the two *endpoints* of the pathway. For example, a pathway between a city at (1,1) and (4,1) is 2 kilometers long in distance. These 2 kilometers of the pathway is called the *body* of this pathway.

- A pathway is built between two cities u and v if u is directly to the north, south, east or west of v if there is no city in between u and v.

- Because it is possible for a vehicle to travel in Mars surface for a short distance without any covered protection, the distance between the centers of u and v must be strictly larger than 1 kilometer in order for the pathway to be built.

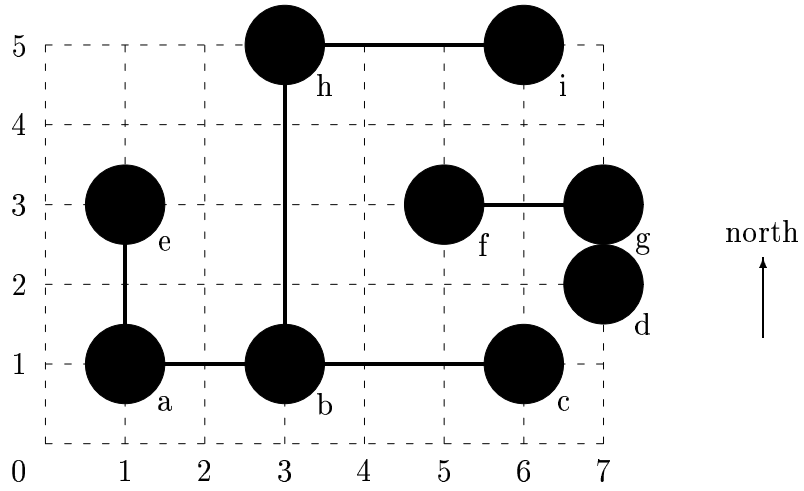
- In order not to cause noise in the cities, if any part of the body of a pathway is less than or equal to 0.5 kilometer of **any part** (not the

center) of a city that is not one of the two endpoints, then this pathway is not built.

- If a potential east-west bounded pathway is intersected with a north-south bounded pathway, then the east-west bounded one is not built.

The construction plan is approved by Primary minister Q. Your task is to help Minister of Transportation T finding the total number and their total distance (in kilometers) of the pathways constructed.

For example, there are 9 cities in the following piece of flat land whose coordinates are (1,1), (3,1), (6,1), (7,2), (1,3), (5,3), (7,3), (3,5) and (6,5), respectively. There are 6 pathways constructed with a total distance of 10 kilometers.



Input: The input file contains multiple test cases. In each test case, a list of coordinates for the cities are listed one city per line with 2 positive numbers indicating their x and y coordinates. A line with two 0's separates two test cases. The end of the file is a line with two -1's.

Output: For each test case, output the total number of pathways constructed and their total distance in kilometers in one line separated by one blank.

Sample Input

```
1 1
3 1
6 1
7 2
1 3
5 3
7 3
3 5
6 5
0 0
1 1
4 1
0 0
-1 -1
```

Output for the Sample Input

```
6 10
1 2
```

Problem E

Puzzle for Friends and Enemies

Input File: pe.dat

In a country, there are N persons, labeled by $1, 2, \ldots, N$. The country has exactly two political parties. All of the people in this country belong to at most one of the two parties. If two persons join the same party, then they are *friends* to each other and are *enemies* if they beong to different parties. Assume that N is known in advance. Let $p_1, p_2, \ldots, p_w, \ldots$ be a sequence of operations where each operation p_w is one of the following two types: (1) a pair of persons indicating that these two persons are enemies, or (2) a query asking the set of confirmed friends and enemies for a person using the information available in the prefix of sequence $p_1, p_2, \ldots, p_{w-1}$. Your task when N and the sequence is given, is to answer the queries mixed in the sequence one by one in sequence. We use a pair of positive integers (i, j), $1 \leq i, j \leq N$, to represent a type (1) operation and use the pair $(0, k)$, $1 \leq k \leq N$, to represent a type (2) operation. For example, when $N = 5$ in the following input sequence:

```
p1    1    3
p2    5    2
p3    2    4
p4    0    2
p5    1    4
p6    0    4
      .    .
      .    .
```

At p_4 it should show that there are no confirmed friend for person 2 and two confirmed enemies (4 and 5) for person 2 according to all inputs appeared before p_4. At p_6 it should show that there are two confirmed friends (3 and 5) and two enemies (1 and 2) for person 4.

Given the input sequence, let $p_w = (i, j)$ be a type (1) operation. Then p_w *conflicts* with the previous operations, if it is already confirmed that i and j are friends to each other according to the prefix $p_1, p_2, \ldots, p_{w-1}$. To handle this situation, we should completely ignore this conflict relation and assume that it does not exist. For example, in the following input sequence when $N = 5$, the operations $p_4 = (4, 5)$ and $p_7 = (1, 2)$ should be ignored and p_5

and p_8 will have the same outputs as the outputs for the queries p_4 and p_6 in the previous input sequence:

```
p1      1       3
p2      5       2
p3      2       4
p4      4       5
p5      0       2
p6      1       4
p7      1       2
p8      0       4
        .       .
        .       .
```

Input: The input file contains several test cases each of which separated by a pair of 0's. For each test case, the first line contains the number of persons N, $1 \leq N \leq 30,000$. Then in each of the following input lines, it includes two integers i and j separated by blanks where $0 \leq i \leq N$ and $1 \leq j \leq N$ to denote an operation. The number of operations for each test case is within $1,100,000$. If i is equal to 0 then a query for person j's current status should be printed out. Otherwise i and j are a pair of enemies. Note the enemy relation between i and j is ignored if a conflict condition occurred. A line consists of a single 0 indicates the end of the file.

Output: For each query of the person k, you should print out one line to list out number of friends and number of enemies for person k known at the current stage as follows:

person k's status: number of friends, number of enemies

Print a blank line after each output of a query. Print a line with ten ='s at the end of each test case.

Sample Input

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

Output for the Sample Input

```
person 2's status: 0, 2

person 4's status: 2, 2

==========
person 2's status: 0, 2

person 4's status: 2, 2

==========
```

Problem F
A Version of Nim
Input File: pf.dat

Here we introduce a special version of the combinatoric game called *Nim*. This progressively finite take-away game involves 4 piles of stones on a table. In this game, the two players take turns removing at least one but at most 3 stones from exactly one of the piles. The player who takes the last stone from the table losses.

The configuration of an instance of this Nim game can be described with four nonnegative integers representing the sizes of these four piles. i.e., (p_1, p_2, p_3, p_4), where the kth, $1 \leq k \leq 4$, number p_k representing the current size of the kth pile.

A configuration of this Nim game is *winnable* if a player facing this configuration can always find a way to win the game. To write a program that plays the Nim game perfectly, we need to decide whether a given instance of the Nim game is *winnable* to the current player or not facing the given configuration. For example, the configuration $(0, 0, 0, 2)$ is winnable by removing one stone from the last pile; however, you can verify that neither $(0, 0, 0, 5)$ nor $(2, 2, 0, 0)$ is winnable. Further, to make the problem easier, we assume that the number of stones on each pile is at most 9.

Input File

The first line contains n, the number of Nim game configurations, which can be as large as 20. After n, there will be n lists of Nim game configurations; each line contains four integers p_1, p_2, p_3 and p_4. Note that $0 \leq p_1, p_2, p_3, p_4 \leq 9$.

Output File

For each configuration appeared in the input, decide whether it is winnable or not. Output a single number '`1`' if the instance is winnable; otherwise, output '`0`'.

Sample Input

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

Output for the Sample Input

```
0
1
0
0
```

Problem G

Tri-Colored Intersections

Input File: pg.dat

There are N, $1 \leq N \leq 1,000$, rectangles in the 2-D xy-plane, each rectangle is associated with exactly one of the three colors: RED, BLUE or GREEN. The four sides of a rectangle are horizontal or vertical line segments. Rectangles are defined by their lower-left and upper-right corner points. Each corner point is a pair of two non-negative integers in the range of 0 through 50,000 indicating its x and y coordinates. We say that the intersection area of rectangles is *tri-colored* if there are at lease one RED, one BLUE and one GREEN rectangles covering this intersection area. Please compute the total tri-colored intersection area of these rectangles.

Example: Consider the following three rectangles and their colors: rectangle 1: $< (0,0)\ (4,4)\ RED >$, rectangle 2: $< (1,1)\ (5,2)\ BLUE >$, rectangle 3: $< (1,1)\ (2,5)\ GREEN >$. The tri-colored intersection rectangle is at (1,1) (2,2) with an area of 1.

Input: The input consists of multiple test cases. A line of 5 -1's separates each test case. An extra line of 5 -1's marks the end of the input. In each test case, the rectangles are given one by one in a line. In each line for a rectangles, 5 non-negative integers are given. The first two are the x and y coordinates of the lower-left corner. The next two are the x and y coordinates of the upper-right corner. The last integer indicates the color of the rectangle with 1 means RED, 2 means BLUE and 3 means GREEN.

Output: For each test case, output the total tri-colored intersection area in a line.

Sample Input

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

Output for the Sample Input

```
1
0
```

Problem H
Deadlock Detection

Input File: ph.dat

A *deadlock* is defined as a set of processes entering a state where each process is waiting for response (or computing resources) from other processes. For example, in OS text books, there are deadlock detection algorithms to detect if a set of processes is *deadlocked* during runtime.

The deadlock detection problem here is to prevent deadlocks from happening before a program is executed (or even before the program is implemented). The problem is to find if there are *potential deadlocks* among a set of processes without executing the program.

Assume that processes communicate via *send* and *receive* with buffer of zero length; that is, when a process invokes *send*, it must block (or wait) until the message is received by the designated process via *receive*. When a process invokes a *receive* command and there is no message available, the process must block (or wait) until a message arrives. Note that it is possible for two processes to send to the same process P and then P receive the two in sequence. For example, there are three processes *P1, P2,* and *P3* as follows:

```
P1:                    P2:                    P3:
do {                   do {                   do {
    msg1 = 1;              msg2 = 2;              receive(&msg);
    send(P3,msg1);         send(P3, msg2);        if (msg == 1)
    .....                  .....                     send(P1,''ACK'');
    receive(&msg);         receive(&msg);         else
} while (1);           } while (1);                 send(P2,''ACK'');
                                                } while (1);
```

A sender invoking a *send* command must specify the receiving process. When *P1* invokes *send*, it will wait until the message is retrieved by *P3*. The *receive* command, on the other hand, does not need to specify the process from which the message is sent, so there is no process ID in the *receive* command. The synchronization among the processes constitutes a synchronization structure. If there is no deadlock in a system, we say the system is well-synchronized.

Researchers discovered that we can abstract every process is abstracted into a finite-state machine which is expressed using a directed graph with labels on the vertices and edges. The following game on the abstracted graphs can be used to solve a simplified version of the deadlock detection problem. Your are given n, $1 \leq n \leq 6$, directed graphs $G_1, G_2, \ldots, G_n$ abstracted from

processes. Let V_i and E_i be the vertices and edges of G_i, respectively. You may assume each graph has less than 20 nodes. The nodes of V_i are numbered $0, 1, \ldots, |V_i| - 1$. Each edge $e_{i,j} \in E_i$ is denoted as $s_{i,j} \rightarrow t_{i,j}$ indicating an edge from the node $s_{i,j}$ to the node $t_{i,j}$. Each edge is labeled with an integer $label(e_{i,j})$ that can be either positive or negative, but not 0. Initially, the game starts by picking a node s_i in each graph G_i and then place a stone in it. In each of following steps, we move exactly two stones s_i and s_j using the following rules:

1. A stone can only be moved forward one edge in distance along an edge in the graph where the stone is in as a process can only advance to its next state.

2. Assume that s_i moves to s_i', and s_j moves to s_j'. Then $label(s_i \rightarrow s_i') + label(s_j \rightarrow s_j') = 0$. Note that a positive edge means the sending of a message and a negative edges denotes a receiving command. In this simplified version, a send command does not specify the destination.

Hence we can describe each step of the game using an n-tuple vector $S_i =< c_{i,1}, c_{i,2}, \ldots, c_{i,n} >$, where $c_{i,j}$ is a node in the graph G_j. The game ends when it is not possible to make any more moves. The steps that cannot be further advanced are called *deadlocked* steps.

An example is given below for a set of games without deadlocked steps and a set of games with a deadlocked step. In Figure 1, there are three graphs. In each graph, we initially put a stone in the nodes as shown in the figure. Following the rules described above, for example, we can move the stones of G1 and G3 to their next nodes by following the edges labeled with "111" and "-111," respectively. Continuously, you can always find two stones to move without any deadlock. We can verify that this example has no deadlock states. Figure 2(a) is an example that has deadlocks. Suppose we initially put the stones as shown in Figure 2(a). Suppose we move the stone of G1 and G3 to become a state in (b). In (b), we can not make any further movements, thus, is deadlocked. It can be verified that the number of all possible deadlock steps in this example is 1.

Given n, the set of graphs and their starting nodes, your task is to find the number of all the possible deadlocked steps from the starting step.

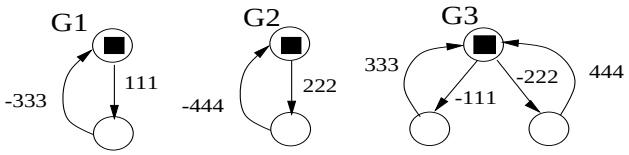


Figure 1: The game has no deadlock state.

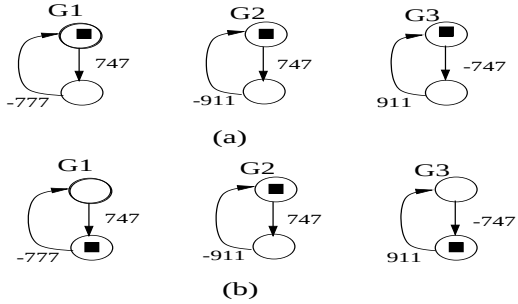


Figure 2: The game has 1 deadlock state.

Input: The input file contains multiple test cases. The first line contains the number of graphs n and then followed by the data of the n graphs. Each graph starts with three integers s t i, where s is the number of nodes, t is the number of transitions, and i is the starting node. Following the line of the three integers s ,t, and i is t lines of directed edges in the form of a b c, which represents an edge with label b from node a to node c. A line with three 0's means the end of a test case. When the number of graphs is 0 ($n = 0$), it is the end of all test cases.

Output: For each test case, output the total number of deadlock steps in a line.

Sample Input

```
3
2 2 0
0 111 1
1 -333 0
2 2 0
0 222 1
1 -444 0
3 4 0
0 -111 1
0 -222 2
1 333 0
2 444 0
0 0 0
3
2 2 0
0 747 1
1 -777 0
2 2 0
0 747 1
1 -911 0
2 2 0
0 -747 1
1 911 0
0 0 0
0
```

Output for the Sample Input

```
0
1
```