Magic Magic Squares

A *magic square* of order n is an arrangement of the n^2 natural numbers $1, \ldots, n^2$ in a square array such that the sums of the entries in each row, column, and each of the two diagonals are the same. For example, here is a magic square of order 3.

4	9	2
3	5	7
8	1	6

The sum of each row, column, and diagonal in this case is 15. This is not the only magic squares of order 3. In fact, we can generate seven others by rotations or reflections of the above:

8	3	4
1	5	9
6	7	2

6	1	8
7	5	3
2	9	4

2	7	6
9	5	1
4	3	8

2	9	4
7	5	3
6	1	8

6	7	2
1	5	9
8	3	4

8	1	6
3	5	7
4	9	2

4	3	8
9	5	1
2	7	6

There is a magic trick for generating a magic square of any odd order n. We illustrate the construction with the following magic square of order 5.

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

After placing the first number, 1, in the bottom row and middle column, we always try to put the next number down-right diagonally. If this seems impossible because it requires moving out of bounds, we simply consider the first row as lying below the last row and the first column as lying to the right of the last column. On the other hand, if it seems impossible because the next square is occupied, we instead move up one square. Again, by rigid motions, we can magically generate seven additional magic squares.

Write a program that considers input lines, each of which is a triple of natural numbers n, r, s, with n odd and ≤ 25, and $1 \leq r, s \leq n^2$ with $r \neq s$, and determines which of the eight magic magic squares do not place r lower or to the right of s. The output, for each of those squares, should be the subrectangle with r and s at its corners. Columns should be separated by tabs and the rectangles separated by a line space. Note that the same subrectangle could appear in more than square and, if so, it should be repeated in the output.

Sample Input/Output on reverse

Sample Input

```
3 6 4
5 22 24
3 7 2
```

Sample Output

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

6       1       8
7       5       3
2       9       4

22      20      13      6
16      14      7       5
15      8       1       24

22      16      15
20      14      8
13      7       1
6       5       24

7       2

7
2


7       2

7
2
```

Interval Inside

In this problem, you are given a set $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_n\}$ of intervals on the line. The goal is to find the largest sequence $\mathcal{I}_{i_1}, \mathcal{I}_{i_2}, \ldots, \mathcal{I}_{i_m}$ such that $\mathcal{I}_{i_j}$ is contained in $\mathcal{I}_{i_{j+1}}$ for $1 \leq j \leq m-1$. (The subscripts $i_1, i_2, \ldots, i_m$ do not have to appear in increasing order.)

The input to your program will be sets of intervals. The first line for each set will indicate the number n of intervals in the set. The following n lines will each specify a interval, namely, each of the lines will consist of a pair x, y of integers with $x \leq y$, separated by a single space, representing the closed interval $[x, y]$, i.e., the set of real numbers $\{r \mid x \leq r \leq y\}$. (You may assume $n \leq 100$).

The output should give the *length* of the longest nested sequence(s), followed by the endpoints of the smallest intervals in each of the nested sequences achieving that length; these numbers should be separated by single spaces. If an interval is the smallest for more than one longest sequence, list it only once.

Sample Input

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

Sample Output

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

SuperKnights

Vulcan Chess is not only more complex than the classical Earth game, but begins with a random choice of rectangular board sizes and chess pieces. The critical piece is the *superknight*. An $\langle A, B\rangle$-superknight moves by jumping between the opposite corners of an $A \times B$ rectangle, with A and B fixed at the outset (knights in Earth chess, are $\langle 2, 3\rangle$-superknights). The power of superknights in Vulcan chess make them the dominant piece over the squares to which they can travel (in any number of moves), which may not be the entire board. For example, in the following 4×5 board a $\langle 3, 4\rangle$-superknight placed in the upper left is considered to "threaten" the marked squares.

*				*
	*		*	
		*		

Vulcan artificial intelligence researchers are fond of testing their search algorithms on the (L, W, A, B)-*SuperKnights Problem*. The problem asks how many mutually non-threatening $\langle A, B\rangle$-superknights can be placed on an $L \times W$ board. For example, the answer to the $(4, 5, 3, 4)$-SuperKnights Problem is 6, as is seen in the following 6 numbered regions, each of which is controllable by a single $\langle 3, 4\rangle$-superknight.

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

Write a program to solve the (L, W, A, B)-SuperKnights Problem. Input will be sequence of lines each giving an L, W, A, B instance, with the numbers separated by a single space. Output for each should be the maximum number n of mutually non-threatening superknights for that instance, followed by a mapping of the n regions with the integers $1, \ldots, n$. The numbers in each line of the board should be separated by tabs and the solutions to the individual problems should be separated by a space. The input numbers L, W, A, B will each be within the range $1 \ldots 12$.

Sample Input

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

Sample Output

```
1
1	1	1
1	1	1

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

Fibonacci Condensed

That most famous of sequences, the *Fibonacci numbers*, is defined recursively by

$$\begin{aligned} \text{Fib}(0) &= 0, \\ \text{Fib}(1) &= 1, \\ \text{Fib}(n) &= \text{Fib}(n-1) + \text{Fib}(n-2), \quad \text{for } n \geq 2. \end{aligned}$$

or in closed form by

$$\text{Fib}(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}, \quad \text{for } n \geq 0$$

where $\phi = \dfrac{1+\sqrt{5}}{2}$, the *Golden Section*.

It is tedious to write out $\text{Fib}(n)$ for large n since the series grows so rapidly, e.g., $\text{Fib}(1000)$ has 209 decimal digits. This problem asks only for the first and last digits and the *number* of digits in between.

Input will be a sequence of integers, one per line. The integers will be between 7 and 10^9. For each number n, there should be one line of output listing the *first digit* of $\text{Fib}(n)$, followed in braces by the *number of digits between* the first digit and the last digit, and then the *last digit* of $\text{Fib}(n)$.

Sample Input

```
7
12
100
1000
```

Sample Output

```
1{0}3
1{1}4
3{19}5
4{207}5
```

Pick-Up Sticks

A two-player game starts with a set of N sticks

and a list, $M_1, M_2, \ldots,$ of positive integers representing legal "moves". In each turn, the player selects a feasible move M_i, and removes M_i sticks from the set (where, of course, $M_i \leq$ the number of sticks remaining). The winner, if there is one, is the player who removes the last stick, leaving an empty set.

Write a program that determines, from a given game specification $N, M_1, M_2, \ldots$, whether you as an expert player moving *first* will win, lose, or draw against an expert second player.

Input will be a sequence of game specifications, one to a line, with the numbers separated by a single space.

Output for each is "`Win`", "`Lose`", or "`Draw`".

Sample input

```
3 1 2
5 1 2 3
7 2 3
9 2
```

Sample Output

```
Lose
Draw
Win
Draw
```