08 A Apocalyptic Alignment (`alignment.{c,cc,java}`)

A.1 Description

Apples and bananas are tasty, but also dangerous. An ancient prophecy said that when you align them in a certain order, the world will be destroyed! On a cloudy day, being tired of this world, you decide to try it out. You started with a line of apples and bananas, and there is one type of allowed operation: At each step, any number of consecutive items in the line can be chosen, replaced by the same amount of fruits of one kind. You can't wait to destroy the world, so you want to know the minimum number of steps to achieve your goal.

A.2 Input

The first line of the input file contains a single number t ($t \leq 100$), the number of test cases. Each test case consists of two lines, where the first line indicates the initial pattern, and the second line indicates the evil pattern that can destroy the world. Both lines contain only characters 'A' and 'B', where 'A' stands for an apple and 'B' stands for a banana. The two lines will have the same length (no greater than 200), and there is no leading or trailing white spaces. For example,

```
2
BB
AA
BAAAB
ABBAA
```

A.3 Output

For each test case, output a single line containing the minimum number of steps to destroy the world. For example,

```
1
2
```

(In the second case of this example, you can first transform the entire row of fruits into apples, and then turn the second through third fruits into bananas, which takes 2 steps total.)

08 B Boundless Boxes (`boxes.{c,cc,java}`)

B.1 Description

Remember the painter Peer from the 2008 ACM ICPC World Finals?[1] Peer was one of the inventors of *monochromy*, which means that each of his paintings has a single color, but in different shades. He also believed in the use of simple geometric forms.

Several months ago, Peer was painting triangles on a canvas from the outside in. Now that triangles are out and squares are in, his newest paintings use concentric squares, and are created from the inside out! Peer starts painting on a rectangular canvas divided into a perfect square grid. He selects a number of single grid cells to act as central seeds, and paints them with the darkest shade. From each of the seed squares, Peer paints a larger square using a lighter shade to enclose it, and repeats with larger squares to enclose those, until the entire canvas is covered. Each square is exactly one grid cell larger and one shade lighter than the one it encloses. When squares overlap, the grid cell is always filled using the darker shade.

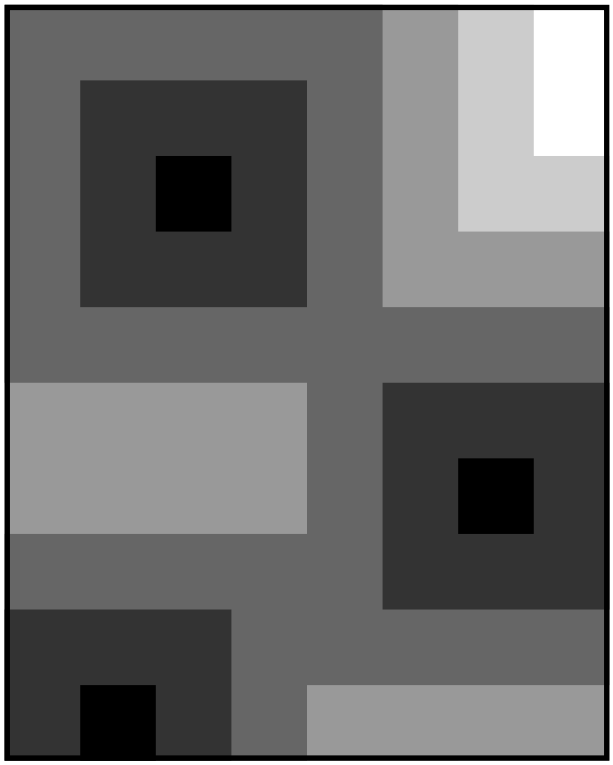

Figure 1: Example of one of Peer's most recent works using six shades of color.

After Peer decides where to place the initial squares, the only difficult part in creating these paintings is to decide how many different shades of the color he will need. To help Peer, you must write a program that computes the number of shades required for such a painting, given the size of the canvas and the locations of the seed squares.

[1] Aside from the artist Peer, this problem really does not have anything to do with the problem from the World Finals, so do not worry if you weren't there or have not read that problem before. In fact, we guarantee that any knowledge of the problem alluded to would not help you solve this one in the slightest bit!

B.2 Input

The input test file will contain multiple cases. Each test case begins with a single line containing three integers, m, n, and s, separated by spaces. The canvas contains exactly $m \times n$ grid cells ($1 \leq m, n \leq 1000$), numbered $1, \ldots, m$ vertically and $1, \ldots, n$ horizontally. Peer starts the painting with s ($1 \leq s \leq 1000$) seed cells, described on the following s lines of text, each with two integers, r_i and c_i ($1 \leq r_i \leq m$, $1 \leq c_i \leq n$), describing the respective grid row and column of each seed square. All seed squares are within the bounds of the canvas.

A blank line separates input test cases, as seen in the sample input below. A single line with the numbers "0 0 0" marks the end of input; do not process this case.

For example, the input for two paintings, including the one shown in the figure above, would look like:

```
10 8 3
3 3
7 7
10 2

2 2 1
1 2

0 0 0
```

B.3 Output

For each test case, your program should print one integer on a single line: the number of different shades required for the painting described. Output corresponding to the sample input would appear as such:

```
6
2
```

08 D Dreadful Deadlines (`deadlines.{c,cc,java}`)

D.1 Description

Contrary to popular belief, diligence does not always pay off! Over the course of his years as an earnest Stanford undergraduate, David found that despite his best efforts, work would always expand to fill the time available. In order to improve his day-to-day efficiency, David has decided to learn the art of procrastination.

David has n assignments due next week. The ith assignment takes x_i units of time and must be finished by time t_i. David can only work on one assignment at a time, and once David begins an assignment, he must work until it is finished. What is the latest time that David can start in order to ensure that all his deadlines are met?

D.2 Input

The input file will contain multiple test cases. Each test case consists of three lines. The first line of each test case contains a single integer n ($1 \leq n \leq 1000$). The second line of each test case contains n integers, $x_1\ x_2\ \ldots\ x_n$ ($1 \leq x_i \leq 10$) separated by single spaces. The third line of each test case contains n integers, $t_1\ t_2\ \ldots\ t_n$ ($1 \leq t_i \leq 1000$) separated by single spaces.

A blank line separates input test cases, as seen in the sample input below. A single line containing "0" marks the end of input; do not process this case.

```
3
1 2 1
9 9 7

2
2 2
3 3

0
```

D.3 Output

For each input test case, print a single line containing an integer indicating the latest time that Jim can start yet still manage to finish all his assignments on time. If the latest time would require Jim to start before time 0, print "impossible". For example,

```
5
impossible
```

08 F Folded Fixtures (`fixtures.{c,cc,java}`)

F.1 Description

You may know of a certain popular and coveted construction toy that Francis got for her birthday this week. This toy consists of a number of metal spheres that can be connected to each other using rigid links (all of the same length) with a magnet on each end. The ends of the links stick to the metal spheres, and the links can freely rotate and extend from the sphere in any direction (essentially forming a spherical joint), allowing you to create a variety of interesting structures.

Francis has assembled several such structures, and now wishes to store her creations by hanging them up in a corner of her room. She notices that, for some of her structures, when she picks it up and holds it by a single sphere, all the links collapse into a single, thin vertical line (see Figure 3) due to the pull of gravity and the spherical joints. Francis can hang up her creations by affixing one sphere on the structure to her ceiling, and wishes to save space by hanging each one up by the sphere that results in the shortest collapsed line. She deems those fixtures which do not hang as a single thin line to take up too much space, and discards them.

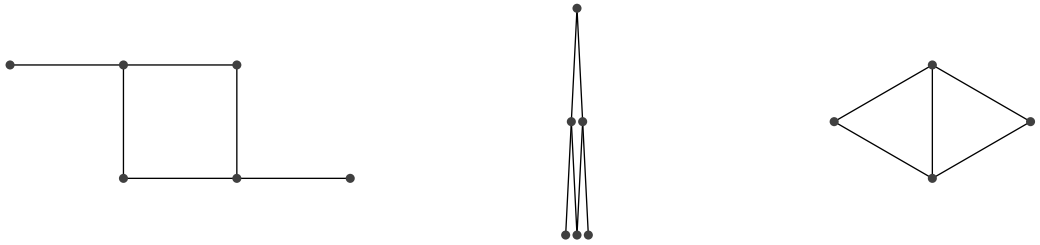

Figure 3: One of Francis's constructions (left), the same collapsed into a straight-line fixture of length 2 (middle), and a structure that will not collapse into a single line when hung up by any sphere (right). Note that in the middle diagram, the horizontal gaps shown between the spheres are for illustrative purposes only! Mathematically, the fixture would be a single, infinitely thin vertical line.

For simplicity, we can treat the metal spheres as infinitely small points and the links as line segments of unit length. Can you write a program to help Francis figure out how much space her fixtures will take, and which ones to discard? Your task is to find the shortest length possible for the collapsed fixture if you were to hang it up by a single sphere, or report that there is no way to hang up the fixture by so that it collapses into an infinitely thin straight line.

F.2 Input

The input will contain multiple test cases for you to analyze. Each test case describes a fully connected fixture (i.e. there are no loose, unattached components). The first line of a test case consists of two integers, n and m, separated by a space, indicating the number of spheres ($1 \leq n \leq 100$) and links ($0 \leq m \leq 1000$) used in the structure, respectively. The spheres are numbered uniquely from 1 to n. The following m lines of input each contain two integers, a_i and b_i ($1 \leq a_i, b_i \leq n$), indicating that Francis has attached sphere a to sphere b in her fixture. Note that both ends of a link cannot be attached to a single sphere, and no two links will attach the same two spheres.

A blank line separates input test cases, as seen in the sample input below. A single line containing "0 0" marks the end of input; do not process this case.

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

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

0 0
```

F.3 Output

For each input test case, print a single line containing the shortest length possible of the collapsed fixture. If it is not possible to hang the described fixture up by a single sphere so that it collapses into a line, print "impossible". For example,

```
2
impossible
```

09 A Bonsai (`bonsai.{c,cc,java}`)

A.1 Description

After being assaulted in the parking lot by Mr. Miyagi following the "All Valley Karate Tournament", John Kreese has come to you for assistance. Help John in his quest for justice by chopping off all the leaves from Mr. Miyagi's bonsai tree!

You are given an undirected tree (i.e., a connected graph with no cycles), where each edge (i.e., branch) has a nonnegative weight (i.e., thickness). One vertex of the tree has been designated the root of the tree. The remaining vertices of the tree each have unique paths to the root; non-root vertices which are not the successors of any other vertex on a path to the root are known as leaves.

Determine the minimum weight set of edges that must be removed so that none of the leaves in the original tree are connected by some path to the root.

A.2 Input

The input file will contain multiple test cases. Each test case will begin with a line containing a pair of integers n (where $1 \leq n \leq 1000$) and r (where $r \in \{1, \ldots, n\}$) indicating the number of vertices in the tree and the index of the root vertex, respectively. The next $n-1$ lines each contain three integers u_i v_i w_i (where $u_i, v_i \in \{1, \ldots, n\}$ and $0 \leq w_i \leq 1000$) indicating that vertex u_i is connected to vertex v_i by an undirected edge with weight w_i. The input file will not contain duplicate edges. The end-of-file is denoted by a single line containing "0 0".

```
15 15
1 2 1
2 3 2
2 5 3
5 6 7
4 6 5
6 7 4
5 15 6
15 10 11
10 13 5
13 14 4
12 13 3
9 10 8
8 9 2
9 11 3
0 0
```

A.3 Output

For each input test case, print a single integer indicating the minimum total weight of edges that must be deleted in order to ensure that there exists no path from one of the original leaves to the root.

```
16
```

09 E Universal Oracle (`oracle.{c,cc,java}`)

E.1 Description

In computer science, an oracle is something that gives you the answer to a particular question. For this problem, you need to write an oracle that gives the answer to everything. But it's not as bad as it sounds; you know that 42 is the answer to life, the universe, and everything.

E.2 Input

The input consists of a single line of text with at most 1000 characters. This text will contain only well-formed English sentences. The only characters that will be found in the text are uppercase and lowercase letters, spaces, hyphens, apostrophes, commas, semicolons, periods, and question marks. Furthermore, each sentence begins with a single uppercase letter and ends with either a period or a question mark. Besides these locations, no other uppercase letters, periods, or question marks will appear in the sentence. Finally, every question (that is, a sentence that ends with a question mark) will begin with the phrase "What is..." For example:

```
Let me ask you two questions. What is the answer to life? What is the answer to the universe?
```

E.3 Output

For each question, print the answer, which replaces the "What" at the beginning with "Forty-two" and the question mark at the end with a period. Each answer should reside on its own line. For example:

```
Forty-two is the answer to life.
Forty-two is the answer to the universe.
```

09 F Rectangles (`rectangles.{c,cc,java}`)

F.1 Description

A rectangle in the Cartesian plane is specified by a pair of coordinates (x_1, y_1) and (x_2, y_2) indicating its lower-left and upper-right corners, respectively (where $x_1 \leq x_2$ and $y_1 \leq y_2$). Given a pair of rectangles, $A = ((x_1^A, y_1^A), (x_2^A, y_2^A))$ and $B = ((x_1^B, y_1^B), (x_2^B, y_2^B))$, we write $A \preceq B$ (i.e., A "precedes" B), if

$$x_2^A < x_1^B \qquad \text{and} \qquad y_2^A < y_1^B.$$

In this problem, you are given a collection of rectangles located in the two-dimension Euclidean plane. Find the length L of the longest sequence of rectangles $(A_1, A_2, \ldots, A_L)$ from this collection such that

$$A_1 \preceq A_2 \preceq \ldots \preceq A_L.$$

F.2 Input

The input file will contain multiple test cases. Each test case will begin with a line containing a single integer n (where $1 \leq n \leq 1000$), indicating the number of input rectangles. The next n lines each contain four integers x_1^i y_1^i x_2^i y_2^i (where $-1000000 \leq x_1^i \leq x_2^i \leq 1000000$, $-1000000 \leq y_1^i \leq y_2^i \leq 1000000$, and $1 \leq i \leq n$), indicating the lower left and upper right corners of a rectangle. The end-of-file is denoted by a single line containing the integer 0.

```
3
1 5 2 8
3 -1 5 4
10 10 20 20
2
2 1 4 5
6 5 8 10
0
```

F.3 Output

For each input test case, print a single integer indicating the length of the longest chain.

```
2
1
```

11 A Another Rock-Paper-Scissors Problem

Sonny uses a very peculiar pattern when it comes to playing rock-paper-scissors. He likes to vary his moves so that his opponent can't beat him with his own strategy.

Sonny will play rock (R) on his first game, followed by paper (P) and scissors (S) for his second and third games, respectively. But what if someone else is using the same strategy? To thwart those opponents, he'll then play paper to beat rock, scissors to beat paper, and rock to beat scissors, in that order, for his 4th through 6th games. After that, he'll play scissors, rock, and paper for games 7–9 to beat anyone copying his last set of moves. Now we're back to the original order—rock, paper, scissors—but instead of being predictable and using the same moves, do you know what would be better? You guessed it! Sonny then plays the sequence of moves that would beat anyone trying to copy his whole strategy from his first move, and on it goes...

To recap, in symbolic form, Sonny's rock-paper-scissors moves look like this:

```
R P S PSR SRP PSRSRPRPS SRPRPSPSR PSRSRPRPSSRPRPSPSRRPSPSRSRP ...
```

The spaces are present only to help show Sonny's playing pattern and do not alter what move he'll play on a certain game.

Naturally, your job is to beat Sonny at his own game! If you know the number of the game that you'll be playing against Sonny, can you figure out what move you would need to play in order to beat him?

A.1 Input

Each line of the input contains a single integer N, $1 \leq N \leq 10^{12}$, the number of the game you'll be playing against Sonny. An integer $N = 1$ indicates it would be Sonny's first game, $N = 7$ indicates it would be the 7th game, and so forth. The input terminates with a line with $N = 0$. For example:

```
1
7
33
0
```

Warning: N may be large enough to overflow a 32-bit integer, so be sure to use a larger data type (i.e. `long` in Java or `long long` in C/C++) in your program.

A.2 Output

For each test case, output a single line which contains the letter corresponding to the move you would need to play to beat Sonny on that game. For example, the correct output for the sample input above would be:

```
P
R
S
```

11 B Ball Painting

There are $2N$ white balls on a table in two rows, making a nice 2-by-N rectangle. Jon has a big paint bucket full of black paint. (Don't ask why.) He wants to paint all the balls black, but he would like to have some math fun while doing it. (Again, don't ask why.) First, he computed the number of different ways to paint all the balls black. In no time, he figured out that the answer is $(2N)!$ and thought it was too easy. So, he introduced some rules to make the problem more interesting.

- The first ball that Jon paints can be any one of the $2N$ balls.
- After that, each subsequent ball he paints must be adjacent to some black ball (that was already painted). Two balls are assumed to be adjacent if they are next to each other horizontally, vertically, or diagonally.

Jon was quite satisfied with the rules, so he started counting the number of ways to paint all the balls according to them. Can you write a program to find the answer faster than Jon?

B.1 Input

The input consists of multiple test cases. Each test case consists of a single line containing an integer N, where $1 \leq N \leq 1{,}000$. The input terminates with a line with $N = 0$. For example:

```
1
2
3
0
```

B.2 Output

For each test case, print out a single line that contains the number of possible ways that Jon can paint all the $2N$ balls according to his rules. The number can become very big, so print out the number modulo 1,000,000,007. For example, the correct output for the sample input above would be:

```
2
24
480
```