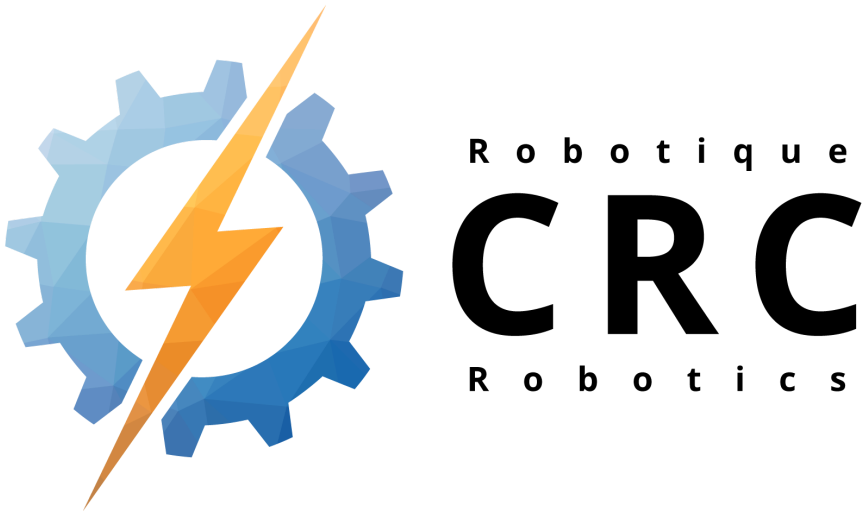

CRC

PROGRAMMING COMPETITION

PRELIMINARY PROBLEM 3

A FEW NOTES

- The complete rules are in section 4 of the rulebook.
- You have until **Sunday, February 18, 11:59 pm** to submit your code
- Feel free to use the programming forum on the CRC discord to ask questions and discuss the problem with other teams. That's what it's there for!
- **We are giving you quick and easy-to-use template files for your code and the tests. Please use them.**

USING THE TEMPLATE FILE

- Basically, the template tests calls the function to test and takes the output and allows you to quickly check if it did what was expected or not. **All your code (except additional functions and classes, which should go right above it in the file) should be written in the function of the part of the problem you are solving.**
- Points given in the document are indications of how difficult the section is and how many points you will get if you complete it. This preliminary problem is going to be 2% of the main challenge towards the global score of the programming competition and for more points related information consult the rulebook.

STRUCTURE

Every problem contains a small introduction like this about the basics of the problem and what is required to solve it. Points distribution is also given here for the preliminary problems.

Input and output specification:

In these two sections, we specify what the inputs will be and what form they will take, and we also say what outputs are required for the code to produce and in what format they shall be.

Sample input and output:

In these two sections, you will find a sample input (that often has multiple entries itself) in the sample input and what your program should produce for such an input in the sample output.

Explanation of the first output:

Sometimes, the problem might still be hard to understand after those sections, which is why there will also be a usually brief explanation of the logic that was used to reach the first output from the first input.

There was a little sailboat

A brave little ship is lost far out in the ocean when a storm begins to rage. Will you be able to guide it through the winds and tides and bring it home in one piece?

Part 1: **Vaguely** (15 points)

The complicated motion of ocean waves must first be broken down before the boat can be guided with any certainty. Let's start with the one-dimensional case. We're mainly interested in a kind of standing wave, but one that grows with time while attenuating the amplitude of oscillation so as not to simply create energy out of nowhere. In a way, it's both a progressive and a stationary wave. A simple standing wave of sinusoidal form has the peculiarity that after half a period of oscillation, wave "troughs" become "peaks" and vice versa. Discreetly, we'll use a finite odd integer number of digits to model a wave. So here's the **oscillation** of a wave of magnitude 7 and length 3:

-7 7-7

after a half-period => 7-7 7

after another half-period => -7 7-7, etc.

7 are the peaks and -7 are the troughs, the numbers roughly describing the height of the water at that point in relation to the height of the sea. However, on a body of water, waves propagate in addition to this. We'll consider that the waves propagate by exactly 1 on each side every half-period (which we'll now call simulation steps). So, with oscillation for the same wave, but this time adding **propagation**

-7 7-7

after a step=> -7 7-7 7-7

after another step => -7 7-7 7-7 7-7

Naturally, the wave grows by two in length per step. There's just one problem, however. With our current model, energy is created out of nowhere, which is not really allowed in the world of physics at the moment... So we'll have to introduce a decrease in the wave. At each step, the wave's "troughs" and "peaks" will decrease in magnitude by 1. By introducing this **attenuation**, we now obtain for the same wave:

-7 7-7

after a step => -6 6-6 6-6

after 2 steps => -5 5-5 5-5 5-5

For simplicity's sake, from now on, all waves will start with an "epicenter" where there will be only one number, and subsequent steps will be calculated according to **oscillation**, **propagation** and **attenuation**.

Input specification:

You will receive as an input a positive integer between 0 and 9 inclusively corresponding to the initial wave amplitude.

Output specification:

As an output, you will return a list of strings corresponding to all the modeled steps of the simulation sorted chronologically.

Sample input:

```
4
9
1
```

Sample output:

```
[' 4', ' 3-3 3', ' 2-2 2-2 2', ' 1-1 1-1 1-1 1', ' 0 0 0 0 0 0 0 0 0']
[' 9', ' 8-8 8', ' 7-7 7-7 7', ' 6-6 6-6 6-6 6', ' 5-5 5-5 5-5 5-5 5', ' 4-4 4-4 4-4 4-4 4-4 4', ' 3-3 3-3 3-3 3-3 3-3 3-3 3', ' 2-2 2-2 2-2 2-2 2-2 2-2 2-2 2', ' 1-1 1-1 1-1 1-1 1-1 1-1 1-1 1-1 1', ' 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0']
[' 1', ' 0 0 0']
```

Explanation of the first output:

A detailed example is used to explain the problem. In the case of the 4 for the first output here, we can simply observe that there are 5 steps with the same propagation pattern counting the 0-step, which is perfect.

Part 2: Vaguely **Limited** (15 points)

This is more or less the same problem as in Part 1. This time, you have to take into account that the region of interest may be potentially restricted. The entire region of interest will be given to you in its initial state, i.e. it will contain one and only one non-zero initial wave amplitude. You'll then need to calculate the evolution of the simulation for a number of precise steps as in number 1, but this time taking into account that wave propagation stops at the edges.

Input specification:

As input, you will receive an integer corresponding to the number of simulation steps to be modeled, followed by a string corresponding to the initial state of the system with the drop.

Output specification:

As output, you'll need to return a list of strings corresponding to the different stages modeled in the simulation

Sample input:

```
3, ‘ 0 0 0 0 0 0 0 9 0 0’
4, ' 0 6 0 0 0 0 0 0 0 0'
2, ' 2 0 0 0 0 0 0 0 0 0'
```

Sample output:

```
[' 0 0 0 0 0 0 8-8 8 0', ' 0 0 0 0 0 7-7 7-7 7', ' 0 0 0 0 6-6 6-6 6-6']
[' 5-5 5 0 0 0 0 0 0 0', '-4 4-4 4 0 0 0 0 0 0', ' 3-3 3-3 3 0 0 0 0 0', '-2 2-2 2-2 2 0 0 0 0']
['-1 1 0 0 0 0 0 0 0 0', ' 0 0 0 0 0 0 0 0 0 0']
```

Explanation of the first output:

In the first two stages of the simulation, it's just like the first part and all's well. However, at the stage with the 6s, since the 7s had reached the right edge of the simulation, we don't push the edge back and there's simply no propagation on that side. 7 becomes -6 according to our rules and that's the end of the story. On the left-hand side, the edge is still a long way off, so everything is proceeding normally.

Part 3: **Bidimensionally** Vaguely Limited (45 points)

The pattern is repeated: we take part 2, and add a dimension to simulate the surface of a body of water. The peculiarity of this is that propagation must take place radially and thus in all directions simultaneously, rather than on a single line. However, this would require a circle pattern. Rather inconsistent circle formatting rules would have to be established. So, the propagation will be in the form of a rhombus (really a square, but hey, that just indicates orientation). So, here's the isolated propagation pattern:

```
    6

   5
  5-5 5
   5

  4
 4-4 4
4-4 4-4 4
 4-4 4
  4
```

etc. Of course, the region of interest is still limited, so this must be taken into account. The body of water will be represented by 10 rows like those used in Part 2, and will contain a single initial drop.

Input specification:

As input, you'll receive a first integer, which is the number of simulation steps you need to provide, and then a list of 10 strings of the same length containing a single non-zero integer. The position of this number is the point from which a drop of water leaves with the wave amplitude specified by the value of the number.

Output specification:

As output, you'll need to return the simulation steps you've modeled as a list of lists (each step is a list) of strings.

Sample input:

```
2,[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 9 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0']
```

Sample output:

```
[[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 8 0 0',
' 0 0 0 0 0 0 8-8 8 0',
' 0 0 0 0 0 0 0 8 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0'],

[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 7 0 0',
' 0 0 0 0 0 0 7-7 7 0',
' 0 0 0 0 0 7-7 7-7 7',
' 0 0 0 0 0 0 7-7 7 0',
' 0 0 0 0 0 0 0 7 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0']]
```

Explanation of the first output:

Once more, an example was explained in detail to illustrate the problem. To get the first input/output pair from the example explained, all is needed is to put that propagation pattern into the grid.

Part 4: **Multiplely** Bidimensionally Vaguely Limited (15 points)

In the same way, we build on the previous step, this time adding several drops to the initial stage of the simulation. The waves will then purely interfere with each other. It's quite simple: the amplitudes of two different waves at the same point are added together. For example, 6 and -4 make a total amplitude of 2. You'll then need to display the total amplitude at each point of the simulation from these initial waves.

Input specification:

As input, you'll receive a first integer, which is the number of simulation steps you need to provide, and then a list of 10 strings of the same length containing 2 to 5 non-zero integers. Their position indicates where this drop of water leaves with the wave amplitude specified by the value of this number.

Output specification:

As output, you'll need to return the simulation steps (with total wave amplitudes) that you've modeled as a list of lists (each step is a list) of strings.

Sample input:

```
2,[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 5 0 0 6 0 0 0 0 0']
```

Sample output:

```
[[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 4 0 0 5 0 0 0 0 0',
' 4-4 4 5-5 5 0 0 0 0'],

[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 3 0 0 4 0 0 0 0 0',
' 3-3 3 4-4 4 0 0 0 0',
'-3 3 1-1 4-4 4 0 0 0']]
```

Explanation of the first output:

Propagation is explained in Part 3. In the final stage, when the waves combine, interference occurs. The 1 and -1 arise from the combination of the two waves or, more precisely, from 4-3 and 3-4, respectively.

Part 5: Multiplely Bidimensionally Vaguely Limited **(but in a boat)** (60 points)

The progressively more complex simulation of an ever-changing body of water was no mean feat! As a new expert navigator, you'll have to guide a brave little sailboat against all odds to make it through. At each stage of the simulation, the boat can move forward one square in the four orthogonal directions, or decide to stay on its current square. The 5 possible actions will therefore be designated by the following 5 letters: "R" for a square to the right, "L" for a square to the left, "D" for a square downwards, "U" for a square upwards and, finally, "S" to remain stationary. **It's important to note that the simulation step change takes place just as the boat is moving, so it's moving to the updated water surface.**

Your aim is to return the boat to its destination in the best possible condition. You'll need to do as little damage to your boat as possible. Your boat takes damage when the simulation step changes, according to the following formula:

$$Dmg = 1 + |A(initial\ position) - A(final\ position)|$$

where Dmg is what the boat picked up during this change of stage, and the A's are the wave amplitudes preceding and succeeding the movement. The difference in amplitude is framed by an absolute value. For example, consider two adjacent boxes in step 2 of a simulation: "4 -1", then in step 3: "1 3". Assuming that the boat moves from the left to the right square during this step change, the damage will be: dmg = 1 + |4-3| = 2. However, if the boat goes from right to left instead: dmg = 1 + |-1-1| = 3. **An instantaneous damage of 8 or more in one turn will immediately destroy your boat and invalidate your solution** (the little guy's not invincible, so watch out for waves). **Damage will be added up over the entire simulation to determine how well you've done** (the aim being to minimize it, of course).

Half the points will be awarded for a valid solution, the other half for the strength of your solution compared with that of other teams (by normalization).

Input specification:

As input, you'll receive a grid with numbers representing water drops at the start of the simulation. In the grid, your boat is marked with an x, and the goal is $

Output specification:

On exit, you'll need to return a sequence of actions (ex: "RDDRURRSLSUU") to be taken to move from the boat's current position to the final position, trying to achieve it with the least damage to the hull integrity of the boat.

Sample input:

```
[' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 x 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 7 0 0',
' 0 0 3 0 0 0 0 0 0 0',
' 0 0 0 0 0 $ 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0',
' 0 0 0 0 0 0 0 0 0 0']
```

Sample output:

```
It’s up to you to find the best possible answer you can!
```