



R o b o t i q u e
CRC
R o b o t i c s

CRC
**PROGRAMMING
COMPETITION**



**PROBLEM BOOKLET
BLOCK A**

A FEW NOTES

- The complete rules are in section 4 of the rulebook.
- You have until **19h59** to submit your solutions.
- Feel free to ask organizers questions and discuss the problem with your team members. It is not an exam!
- **We are giving you quick and easy-to-use template files. Please use them!**

USING THE TEMPLATE FILE

- All the files contain a function called `solve()` where you have to put your code.
DO NOT CHANGE THE NAME OF THAT FUNCTION!
- To run the tests you will have to open a terminal (we strongly recommend to use `vscode`) and run the command `pytest`. If the command is ran in the folder containing all the problems, all tests are going to be ran
- To run a specific test, run it in the folder containing the test you want. You can do the same for sections, just navigate in the folder before executing the command `pytest`

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.

First output explanation:

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.

Table of content

2D PROBLEMS (2D)	4
2D1: Squaresception (20 points)	4
2D2: Connect the Dots (30 points)	6
2D3: Anthill (45 points)	8
2D4: Mycology 101 (75 points)	11
DATA STRUCTURES (DS)	15
DS1: Vertical Horizon (15 points)	15
DS2: Shear Force (35 points)	16
DS3: Tree Parkour! (45 points)	19
SCIENTIFIC CALCULATIONS (SC)	21
SC1: Squared Circle (25 points)	21
SC2: Tip the Scales! (65 points)	22
SC3: Fractionally Additive (35 points)	24
TEXT PROCESSING (TP)	26
TP1: Fix the rules (15 points)	26
TP2: Zipper (25 points)	27
TP3: Palindrôle (30 points)	28

2D PROBLEMS (2D)

2D1: Squaresception (20 points)

You need to display squares inserted into each other. We'll provide you with the number of squares you want and you'll send us back this beautiful construction!!! You'll need the following symbols: (ascii 9488)'⌏', (ascii 9492)'⌐', (ascii 9496)'⌑', (ascii 9484)'⌎', (ascii 9472)'⌌' and (ascii 9474)'⌍'.

Input Specification:

You will receive:

- n : the number of squares to insert

Output Specification:

You will need to output:

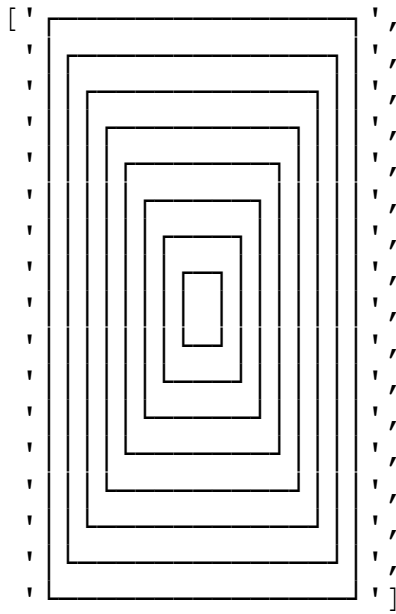
- an array of strings corresponding to nested squares

Sample input:

4
2
8

Sample output:

A diagram showing a nested structure of four rectangles. The outermost rectangle is labeled with a left square bracket '[' and a right square bracket ']' on its left and right sides, respectively. Inside this rectangle are three smaller, concentric rectangles. The second rectangle from the outside is labeled with a left square bracket '[' and a right square bracket ']' on its left and right sides. The third rectangle from the outside is labeled with a left square bracket '[' and a right square bracket ']' on its left and right sides. The innermost rectangle is labeled with a left square bracket '[' and a right square bracket ']' on its left and right sides. The rectangles are nested, with each inner rectangle centered within the previous one.



First output explanation:

We have four interlocking squares with no additional spacing. The center square has 4 corners, 2 horizontal bars, 2 vertical bars and a space in the center.

2D2: Connect the Dots (30 points)

You're playing chess with several queens on the board. You want to indicate by a line which ones of them are able to see any of the others. To do this, you're going to indicate by lines the possible connections between the different queens. Don't forget that the queens can move any number of squares from their current position, that is, in the same row, column or diagonal.

Empty squares on the board will be represented by . and squares containing queens will be replaced by x. You'll need to fill in the links between each of them with the symbols - | / \ to form a line that connects the queens it's possible to connect. No more than one line can pass through the same square.

N.B. Some queens can have no connection at all, while others can have multiple connections.

Input Specification:

You will receive:

- **board**: an array of strings corresponding to the 2D **board** with . to represent empty squares and x to represent queens

Output Specification:

You will need to output:

- an array of strings with . to represent empty cells, x to represent queens and the symbols - | / \ to represent connections

Sample input:

```
[ "x . . . . . x",
  ". . . . x . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . x" ]
[ ". x . . . x .",
  ". . . . . .",
  ". x . x . . .",
  ". . . . . .",
  ". . . x . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . ." ]
```

```
[ "...x.....",
  "...x...",
  ".....x",
  "x.....",
  "...x.....",
  ".....x.",
  "....x....",
  ".x....."]
```

Sample output:

```
[ "x-----x",
  ". \ . . x . |",
  ". . \ . . . |",
  ". . . \ . . . |",
  ". . . . \ . . |",
  ". . . . . \ . |",
  ". . . . . \ |",
  ". . . . . x"]
[ ".x---x..",
  ". | \ . / . .",
  ".x-x....",
  ". . \ | . . .",
  ". . .x....",
  ". . . . .",
  ". . . . .",
  ". . . . ."]
[ "...x.....",
  ".....x..",
  ".....x",
  "x.....",
  "...x.....",
  ".....x.",
  "....x....",
  ".x....."]
```

First output explanation:

The first queen on the top left sees the one on the top right as they are on the same line, as well as the one on the bottom right, as they are on the same diagonal. The top-right queen also sees the bottom-right queen because they're on the same column, but the last queen doesn't see any of the others, so has no connection.

2D3: Anthill (45 points)

A colony of ants has just moved into a new anthill. The worker ants set out to find food in their new environment to keep the colony alive. The new environment will be represented by a square of side N in the first quadrant of a 2D Cartesian plane that shares its center with the anthill.

To create a network of paths, they'll start by finding the food cache closest to the colony and building the shortest possible path between these two points. The network now comprises 2 points (the anthill and the first food cache) and 1 path. Next, they'll go and find the food cache with the shortest distance to any point on the existing network and create a path between the two. They will continue in this way until all the food caches in the new environment are part of the path network.

You will receive as input the length N of a side of the square that forms the area of the new environment, and a list with the positions of the food caches in the form of int lists. Your output will be a list of the indices (listed in ascending order, with "f" last) of the points in L that are connected by a path to the point with the same index i in L , ending with the connections to the anthill, signified by the first element of its connection list being a "f".

Input Specification:

You will receive:

- n : an integer corresponding to the length of the square environment
- L : a two-dimensional integer array corresponding to the list of coordinates (a 2D array of integers)

Output Specification:

You will need to output:

- a list of caches connected to each point (a 2D array of strings)

Sample input:

n: 10

L: [[1,2], [3,4], [5,6], [7,8], [9,1], [2,3]]

n: 8

L: [[4,5], [5,6], [6,7], [2,3]]

n: 6

L: [[1,4], [5,3], [5,2], [2,1], [4,4], [3,1]]

Sample output:


```
[[\'5\'],  
[\'5\', \'f\', ],  
[\'3\', \'f\' ],  
[\'2\'],  
[\'f\' ],  
[\'0\', \'1\' ],  
[\'f:\', \'1\', \'2\', \'4\' ]]
```

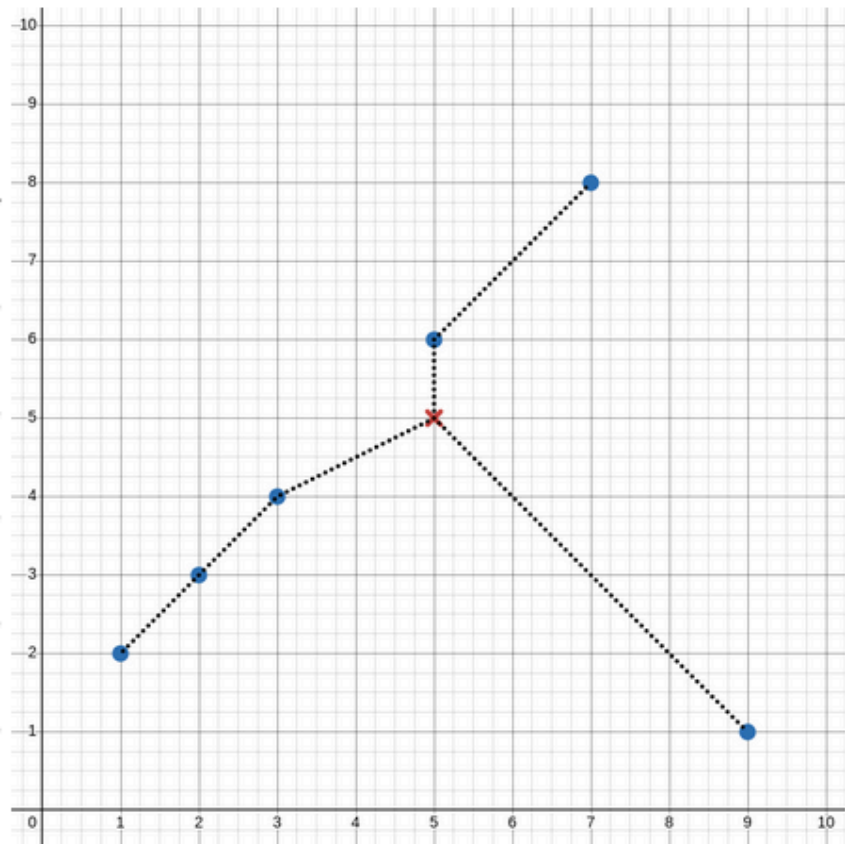
```
[[\'1\', \'f\' ],  
[\'0\', \'2\' ],  
[\'1\' ],  
[\'f\' ],  
[\'f:\', \'0\', \'3\' ]]
```

```
[[\'f\' ],  
[\'2\', \'4\' ],  
[\'1\' ],  
[\'5\' ],  
[\'1\', \'f\' ],  
[\'3\', \'f\' ],  
[\'f:\', \'0\', \'4\', \'5\' ]]
```

First output explanation:

Point (5,6) is closest to the anthill, so we connect the two. Point (3,4) is closest to the anthill network, so we make the shortest possible connection to it, which also links it to the anthill. We then continue with (in order) points (2,3), (1,2), (7,8) and, finally, (9,1). Now we need to give the connections to the points in the order they were given to us. The first point in the list is (1,2), which is connected only to point (2,3), i.e. index 5. Repeating the same for point (3,4), we obtain connections to 5 and to the anthill ("f"). We continue in this way until we reach the end of the list, where we give the connections to the anthill, preceded by an "f:" in the list. (see image below for network visualization)

x_1	 y_1
1	2
3	4
5	6
7	8
9	1
2	3

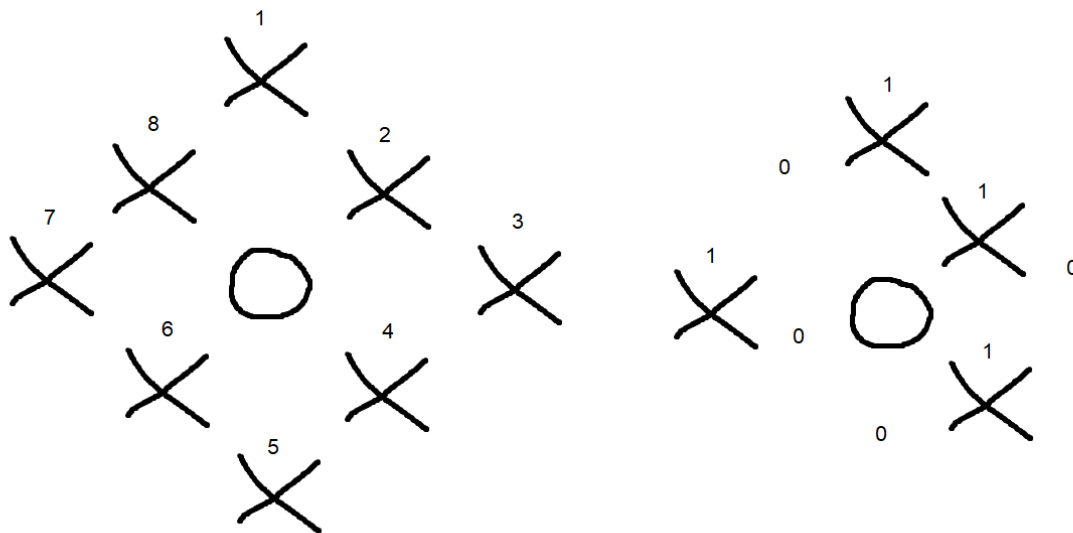


<https://www.desmos.com/calculator/saatdbz9pu?lang=en>

2D4: Mycology 101 (75 points)

By analyzing the reproduction of fungi through the release of spores, we can see that these spores follow a regular pattern. In your research laboratory, you will attempt to simulate the reproduction of these mushrooms using a fractal modeler. However, as the laboratory computer you're using is very old, the parameters will be submitted in the form of bytes to take up as little memory as possible. Two bytes will be passed to your simulation.

Firstly, we'll assume here that mushrooms are identical and differ only in their generation, so the directions of spore release will be the same for all mushrooms. However, their orientation can obviously change; they won't all be aligned. **The first byte determines whether or not a mushroom releases spores in each of the 8 radial directions of a "wheel":**



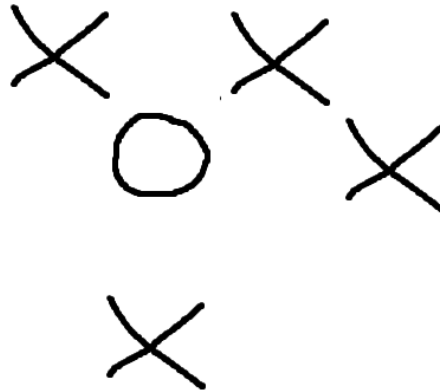
Here, the first shows the possible spore release directions and their respective positions in the release byte. On the right is an example of a byte with its release pattern. In this example, the input byte would be "11010010". This release example is for a release distance of 2.

The second byte contains all the other parameters for the spores in each generation:

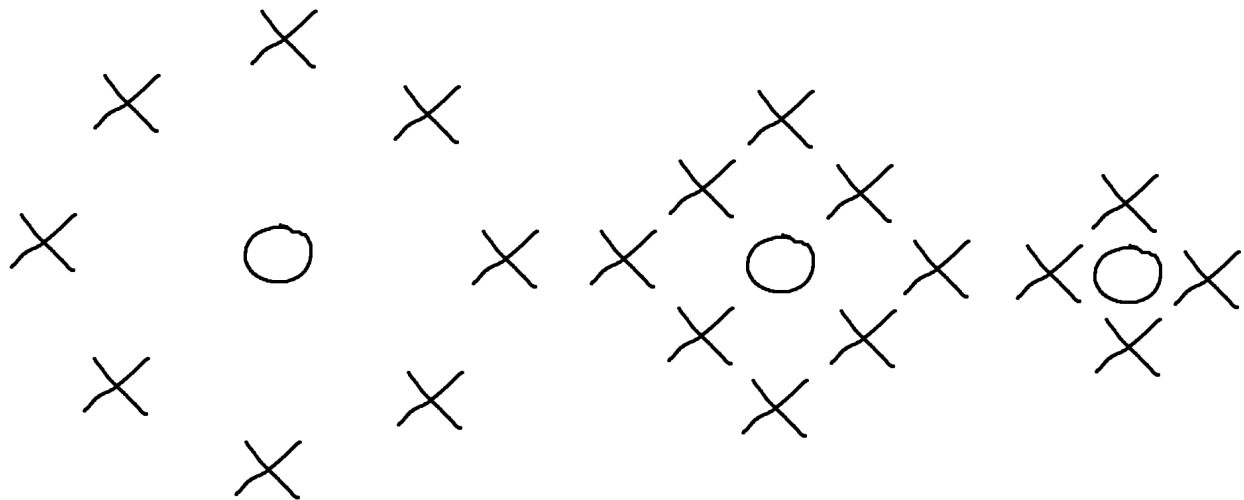


The rotation bit indicates whether or not the release pattern is rotated between generations. If the value of this bit is 0, the rotation direction and value parameters should be ignored, as no rotation should be performed. **The direction bit** sets the pattern rotation direction. If the value of this bit is 0, rotation will be clockwise; if it is 1, it will be counter-clockwise. **The**

next three bits form a three-digit number in binary, ranging from 0 to 7 in base 10, corresponding to the number of 45-degree rotations between generations in the given direction of rotation. It's by multiples of 45 degrees, as it's every 45 degrees that we find a possible spore release according to the rules of our simulation. Here's the above pattern rotated after one generation according to the above bit values (clockwise rotation of 45 degrees):



The last three bits are another three-digit number in binary corresponding to the initial spore release distance. It's important to consider that diagonal spores are released at a distance of $(\text{currentDistance}-1)$ in both coordinates instead of simply being released at a distance of currentDistance in a single coordinate. Here are release distances 3, 2 and 1 as examples, respectively:



Only three final specifications remain. For the pattern to end, **the spore release distance decreases by 1 with each generation until it reaches 0**, where there are no more spores (yes, this also means that there are no more spores diagonally at a release distance of 1). **All fungi in the simulation release spores in every generation, regardless of when they were born.** Finally, the result of your simulation will have to be a rectangle perfectly centered on the

initial mushroom (which will be symbolized by an "o" instead of an "x", meaning that the two dimensions can be different, but logically necessarily odd with an "o" right in the center) to effectively show how the spores have moved away from the initial point, i.e. the center. All cells left without a mushroom at the end of the simulation will be marked with an empty space.

Input Specification:

You will receive:

- ***spreadingByte***: a string corresponding to the spore release byte
- ***parametersByte***: a string containing all the other parameters needed to run the simulation

Output Specification:

You will need to output:

- a list of strings, each corresponding to a row of the simulation's final display, in the right order of course.

Sample input:

spreadingByte: "01010100", **parametersByte:** "10101011"

spreadingByte: "01010101", **parametersByte:** "11001010"

spreadingByte: "10001000", **parametersByte:** "10111011"

spreadingByte: "00100101", **parametersByte:** "01101101"

Sample output:

```
[ "  x  x  "
  "          "
  "x x x x  "
  "          "
  "  x o x  "
  "          "
  "    x x  "
  "          "
  "      x  " ]
```

```
[ " x x  "
  "xxxxx"
  " xox  "
  "xxxxx"
  " x x  " ]
```

```
[ "xxx  "
  " xxx  "
  "  xxx  "
  "xxx  "
  " xox  " ]
```

```

"   xxx"
"xxx  "
"   xxx "
"   xxx"]
[ "      xx          "
"      xxxxx        "
"      xxxxx  xx     "
"      xxxxxxxx      "
"      xxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxx "
"      xx  xxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxxxxxxxxxx  xx "
"      xxxxxxxx  xOxxxxxxxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxxxxxxxxxx  xx "
"      xxxxxxxxxxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxxxxxxxxxx "
"      xx  xxxxxxxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxxxxxxx "
"      xxxxxxxxxxxxxx "
"      xxxxxxxx "
"      xxxxx  xx "
"      xxxxx "
"      xx          " ]

```

First output explanation:

From the release byte we deduce that spores will be spread to the North-West, North-East and South-East of our initial mushroom. The parameter byte tells us that there is a clockwise rotation of 5 times 45 degrees and that the initial release distance is 3 squares. Assuming that the initial mushroom is at the origin of our Cartesian plane, the first generation emits spores at (-2,2), (2,2) and (2,-2). Then, due to rotation, North-West becomes South, North-East becomes West and South-East becomes North. As the distance decreases to 2, but this time the spores are orthogonal instead of diagonal, new mushrooms appear two squares north, west and south of all existing mushrooms. In the next and final generation, North becomes South-West, West becomes South-East and South becomes North-East. Since all points are diagonal and we're at a distance of 1, no new spores are created in this generation. All we need to do is ensure that the result is centered on the initial mushroom by adding two empty columns to the right, and we're done!

Footnote:

Have fun with some of the values in the examples above! You'll see that the result changes drastically depending on the initial data.

DATA STRUCTURES (DS)

DS1: Vertical Horizon (15 points)

You've made a mistake between horizontal and vertical in your two-dimensional table. You must, WITHOUT THE HELP OF A LIBRARY, invert the rows and columns of the table and send it back.

Input Specification:

You will receive:

- **array**: the array in which to exchange rows and columns

Output Specification:

You will need to output:

- the table with inverted rows and columns

Sample input:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[1, 2], [3, 4], [5, 6]]
[[12, 46, 213, 4123]]
[["ah", "bé", "cé", "dé"], ["euh", "èf", "jé", "ache"]]
```

Sample output:

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
[[1, 3, 5], [2, 4, 6]]
[[12], [46], [213], [4123]]
[["ah", "euh"], ["bé", "èf"], ["cé", "jé"], ["dé", "ache"]]
```

First output explanation:

The first row [1, 2, 3] is transposed into a column and thus becomes the first element of each row to form the same column. The entry can be written as:

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

Then output as:

[1, 4, 7]

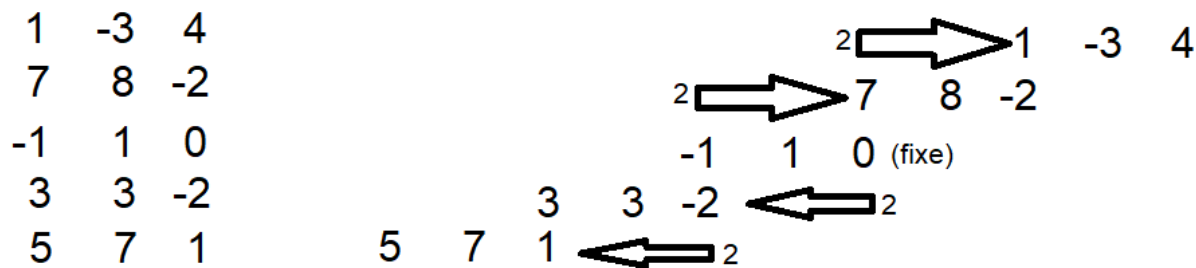
[2, 5, 8]

[3, 6, 9]

DS2: Shear Force (35 points)


A shear transformation is a geometric transformation that allows you to stretch and shift a geometric object such as an image. It's most useful when dealing with discrete things like pixels, and you want to make sure you don't lose any of them during a rotation: you can perform 3 shear mappings to get a rotation without losing any pixels!

Here, you won't be asked to shear pixels, but to shear a two-dimensional array in order to calculate a value that we'll call the "transvector". First step: the transformation! The transformation consists in forming a parallelogram from a rectangle of some kind. We'll take the central row as fixed. If the transformation is to the right, the top rows will shift to the right according to their height and shear modulus, and the bottom rows will shift to the left according to their height and shear modulus too. If the transformation is to the left, it will be the opposite, of course. **The offset between two rows is always equal to the shear modulus.** Here's an example with a shear modulus "m" of 2 to the right on any array:



Each row is offset by two from its neighbors. Since the center row is fixed, we'll always give you arrays with an odd height to guarantee the existence of a center row. Now we just need to calculate the "transvector". We define it as the number formed by the sums of the columns once the 2D array has been sheared. Here's what the transvector would look like, again using the same example:

$$\begin{array}{cccccccccccc}
 & & & & & & & & 1 & -3 & 4 \\
 & & & & & & & 7 & 8 & -2 & \\
 & & & & -1 & 1 & 0 & & & & \\
 & & 3 & 3 & -2 & & & & & & \\
 + & 5 & 7 & 1 & & & & & & & \\
 \hline
 5 & 7 & 4 & 3 & -3 & 1 & 7 & 8 & -1 & -3 & 4
 \end{array}$$


=57437178974

It's important to note that if the sum of a column is negative, we'll replace the digits in the sum with their inverse and then remove the negative, as we've done here. The inverse "x" of a digit "c" will be defined here as the digit for which $c + x = 10$. Thus, the inverse of 3 is 7 and the inverse of 1 is 9, as in the example above. **It's also important to mention that, although not the case here, sums can result in multi-digit numbers that will be concatenated in the transvector just like single digits.** If the multi-digit number is also negative, each digit must be replaced by its inverse. For example, "-583" becomes "527". The direction of shearing (indicated by a letter "D" or "G") and its modulus will be joined in the same parameter in the form of a string. **There will never be any empty columns after the transformation.**

Input Specification:

You will receive:

- **tableau**: a two-dimensional array containing random integers
- **mode**: a string containing the shear modulus (an integer from 0 to 9) and shear direction ("D" or "G")

Output Specification:

You will need to output:

- an integer corresponding to the transvector described above

Sample input:

```

tableau: [[1,-3,4],[7,8,-2],[-1,1,0],[3,3,-2],[5,7,1]], mode:"2D"
tableau: [[1,-3,4],[7,8,-2],[-1,1,0],[3,3,-2],[5,7,1]], mode:"2G"
tableau: [[37,-43,12,8,126],[42,-24,37,53,75],[122,7,0,-35,-88]],
mode:"3G"

```

```
tableau: [[1,-1],[0,0],[-1,1],[0,0],[1,-1],[2,-2],[3,-3],[4,-4],[  
5,-5]], mode: "1D"  
tableau: [[1,2,3],[4,5,6],[7,8,9]], mode: "0D"
```

Sample output:

```
57437178974  
171187133371  
37671250102371758207522  
5999999119  
121518
```

First output explanation:

See problem description for a detailed explanation of the first output.

Footnote:

Check out Matt Parker's video on the subject:

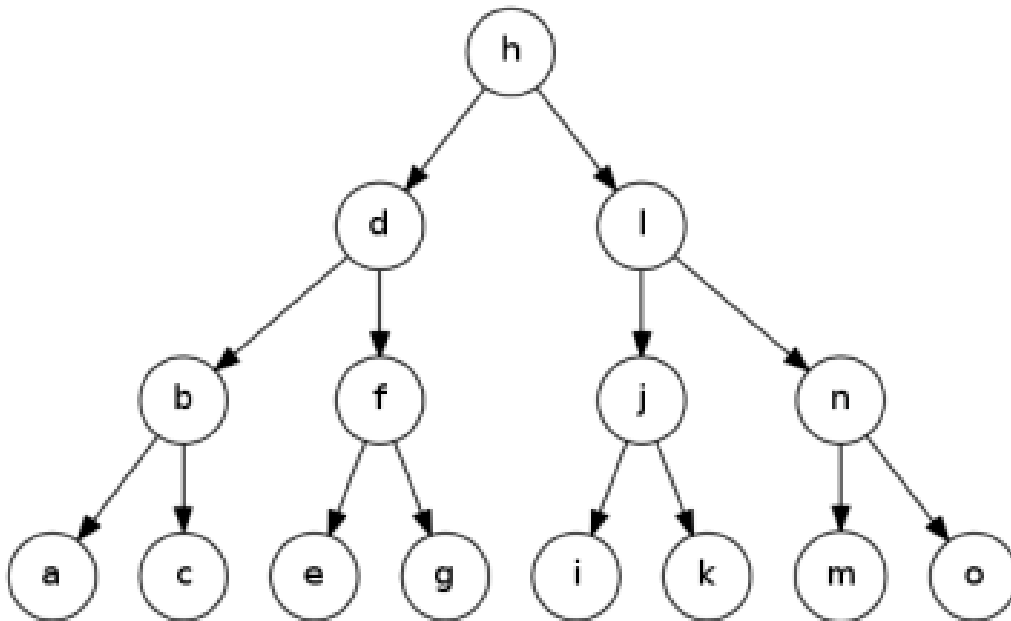
<https://www.youtube.com/watch?v=1LCEiVDHJmc!>

(It's unlikely to help much with this problem, in case you're looking for help).

DS3: Tree Parkour! (45 points)

Trees are an important data structure that is all too often overlooked in programming. The concept of a tree is to install data in a hierarchy so that it can be explored more quickly, without requiring too much memory space. In a tree, we have nodes, which have child nodes. In the trees we're going to use, the maximum number of child nodes will always be 2, and they will be perfect trees. This means that all levels containing children will have exactly 2 children for each node in the level. You'll need to traverse the tree as seen from left to right. See the explanation of the first output for more details. You'll be presented with the tree layer by layer. So the first array will contain just one element, the second array will contain 2, the next level will have 4 elements, then 8 and so on.

You'll need to traverse the tree in infix order and return the concatenation of these elements in a string. The infix order is to take the leftmost element, followed by its parent and then by the element on the right.



Input Specification:

You will receive:

- **tree:** an array with a variable number of strings at each level (2x more than the previous level)

Output Specification:

You will need to output:

- a string of all elements as seen from left to right

Sample input:

```
[['h'],  
['d', 'l'],  
['b', 'f', 'j', 'n'],
```

```
['a', 'c', 'e', 'g', 'i', 'k', 'm', 'o']]
```

```
[['t'],  
['R', 's'],  
['C', 'C', 'e', 't']]
```

```
[['t'],  
[' ', 'b'],  
['R', 's', 'e', 's'],  
['C', 'C', 'i', ' ', 'h', ' ', 'e', 't']]
```

Sample output:

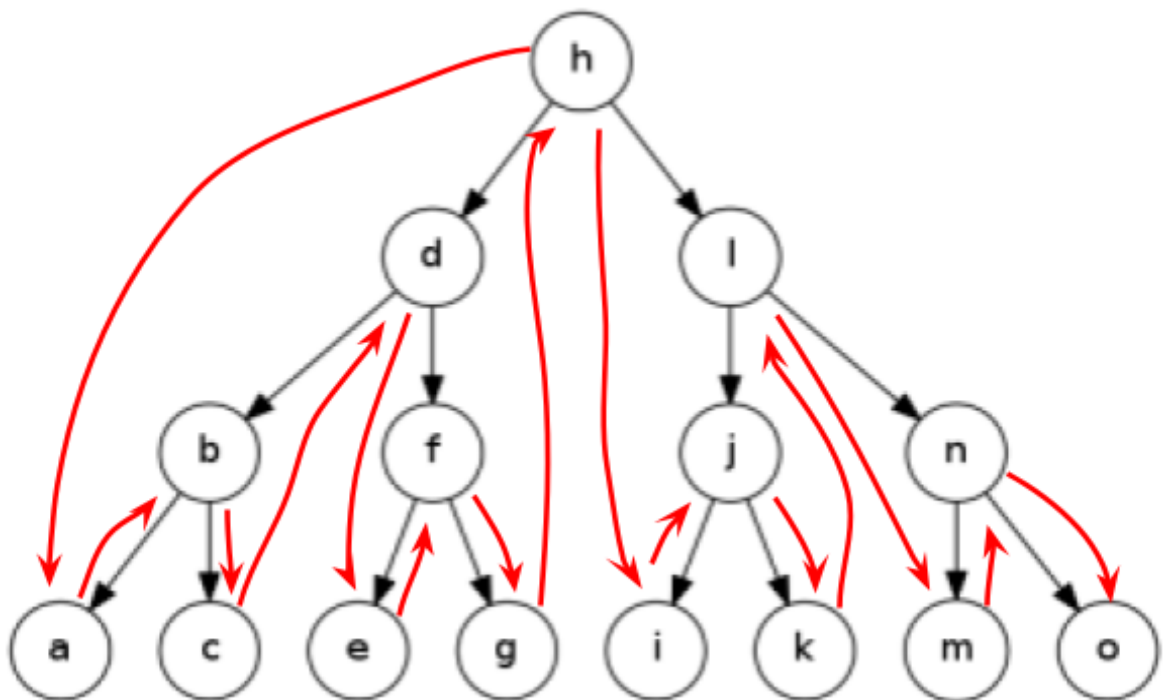
```
abcdefghijklmno
```

```
CRCtest
```

```
CRC is the best
```

First output explanation:

To display a tree in infix order, we look for the leftmost node, then its parent and then the node to its right. When we reach a node, we first look to see if it has any children and explore the left-hand portion first. We begin this journey at the root of the tree, at element h. You can visualize the path to follow in an infix path using the red arrows.



SCIENTIFIC CALCULATIONS (SC)

SC1: Squared Circle (25 points)

You need to calculate the ratio between the area of a circle and the area of a square. However, the circle can be larger or smaller than the square. Both will be centered at the same point. As you don't want to do pure maths, you use a statistical approach. You'll need to generate randomly distributed points in the square and see how many of them are also in the circle.

There is no restriction on the number of digits; try to be as close as possible to the answer although it is quite unlikely you will be right on target.

Input Specification:

You will receive:

- **rayon**: the radius of the circle
- **L**: the length of one side of the square
- **nbrPoints**: the number of realizations you'll need to generate to find the ratio of the area of the square covered by the circle

Output Specification:

You will need to output:

- a float of the fraction of the square covered by the circle

Sample input:

```
rayon: 2      L: 5      nbrPoints: 10 000
rayon: 3      L: 4      nbrPoints: 100 000
rayon: 2      L: 4      nbrPoints: 10 000
```

Sample output:

```
0.503061
1.0
0.785223
```

First output explanation:

For the first example, we have a circle of radius 2 centered at (0, 0) and a 5x5 square also centered at (0, 0), so the corners are at (-2.5, 2.5) (-2.5, -2.5) (2.5, -2.5) (2.5, 2.5). We generate a point 10,000 times that lies within the square and count the number of times the generated point also lies within the circle. In our case, we have 0.503561 as the percentage of the square covered by the circle. In fact, this is close to the theoretical value calculated by the following formula:

$$\frac{\text{rayon}^2 * \pi}{L^2} = \frac{2^2 * \pi}{5^2} = \frac{4\pi}{25} \approx 0.502654825$$

SC2: Tip the Scales! (65 points)

We haven't always had hyper-precise digital scales: in the past, we determined the weight of objects we wanted to weigh by comparing them with weights we already knew. Here, a merchant has only a very old scale with a series of small weights, and he wants to determine the weight of certain objects he's selling in kilograms. We want to know how many ways it is possible for him to get to the desired counterweight.

For example, let's consider that the object the merchant is trying to weigh totals 12 kilograms, and that the merchant only has weights of 4 and 6 kilograms. The merchant will have weighed the object when he has put 3 weights of 4 kilos on the scale or 2 weights of 6 kilos on the scale.

You must give all possible combinations to achieve this. (Example: the combination [6, 4, 2] would be ahead of the combination [4, 4, 4], but behind the combination [6, 6]. [2, 4, 6] is the same combination as [6, 4, 2], even if there are several of a certain weight type, and will therefore be omitted).

Input Specification:

You will receive:

- *poids*: an array of integers
- *contrepoids*: an integer

Output Specification:

You will need to output:

- an array composed of arrays of int

Sample input:

```
poids: [1, 2, 5, 6, 3, 12, 24, 2, 2, 4, 9, 10] contrepoids: 22
poids: [1, 3, 5, 7, 9] contrepoids: 25
poids: [14, 31, 17, 50, 7, 9, 11] contrepoids: 40
poids: [7, 8, 9, 12, 13, 13, 5, 6] contrepoids: 26
```

Sample output:

```
[[12, 10], [12, 9, 1], [12, 6, 4], [12, 6, 3, 1], [12, 6, 2, 2], [12,
5, 4, 1], [12, 5, 3, 2], [12, 5, 2, 2, 1], [12, 4, 3, 2, 1], [12, 4,
2, 2, 2], [12, 3, 2, 2, 2, 1], [10, 9, 3], [10, 9, 2, 1], [10, 6, 5,
1], [10, 6, 4, 2], [10, 6, 3, 2, 1], [10, 6, 2, 2, 2], [10, 5, 4,
3], [10, 5, 4, 2, 1], [10, 5, 3, 2, 2], [10, 5, 2, 2, 2, 1], [10, 4,
3, 2, 2, 1], [9, 6, 5, 2], [9, 6, 4, 3], [9, 6, 4, 2, 1], [9, 6, 3,
2, 2], [9, 6, 2, 2, 2, 1], [9, 5, 4, 3, 1], [9, 5, 4, 2, 2], [9, 5,
3, 2, 2, 1], [9, 4, 3, 2, 2, 2], [6, 5, 4, 3, 2, 2], [6, 5, 4, 2,
2, 2, 1]]
[[9, 7, 5, 3, 1]]
```

```
[[31, 9], [17, 14, 9]]  
[[13, 13], [13, 8, 5], [13, 7, 6], [12, 9, 5], [12, 8, 6], [8, 7, 6,  
5]]
```

First output explanation:

Starting with the heaviest weight at our disposal, 24, we can see that it's too heavy to balance the counterweight. Next, we continue with 12, which can be combined with 10 to give 22, as well as many other combinations. The list of possibilities continues in descending order. The final point of interest to note is that the presence of three 2-kilo weights in this problem allows us to perform combinations in which the 2-kilo weight is repeated up to precisely 3 times.

SC3: Fractionally Additive (35 points)

Adding fractions is not always a simple operation. It's harder for our brains to imagine than simple additions of integers. For example, if you want to know all the possible sums of integers leading up to 14, it's easy: just start in the middle and work your way out by increments of 1 to get more possibilities. For example, the possible sums would be 7+7, 6+8, 5+9 and so on.

The same cannot be said for the sum of two fractions, however. Take 2/85, an obvious two-term sum is 1/85 + 1/85, but 1/84 + 1/86 doesn't necessarily work, indeed 1/84 + 1/86 = 85/3612. Since it's difficult to calculate such pairs of fractions mentally, using a computer is very useful here. We want to calculate all possible solutions of the form:

$$\frac{1}{x} + \frac{1}{y} = \frac{2}{z}$$

where x is different from y to exclude the solution x=y=z. For a given z, you'll need to calculate all possible sums and return the x+y sum associated to every pair found.

Input Specification:

You will receive:

- z: an integer corresponding to the denominator of the fraction to be solved

Output Specification:

You will need to output:

- an array of integers x+y corresponding to all possible solutions to the problem, ranked in descending order

Sample input:

9
37
179
4279
888

Sample output:

[50,24]
[722]
[16200]
[9159200,836550,80000,28008]
[198025,99458,66603,50176,33750,25538,22801,17328,13225,11858,
9126,6400,6253,5043,3698,3626,2775,2401,2368,1998,1850,1813]

First output explanation:

For the first denominator 9, the two results found are $1/5 + 1/45 = 10/45 = 2/9$ and $1/6 + 1/18 = 4/18 = 2/9$. Adding the denominator pairs gives 50 and 24, respectively. Ensuring that they are in descending order, we thus obtain the first output of the problem.

Footnote:

The number of possible solutions for an integer z seems to be relatively related to the number of factors this integer has. More specifically, prime numbers seem to have only one solution with a very large fraction and a much smaller fraction. Can you prove or disprove this?

TEXT PROCESSING (TP)

TP1: Fix the rules (15 points)

You've read the rules for the CRC programming competition, but you don't understand them because there are too many errors to read any decently. You need to remove the capital letters that are not at the beginning of a word and add some at the beginning of each sentence. You must also make sure that the name CRC is always written with all three letters capitalized! As these are very strict rules, all sentences must be completed by an exclamation mark.

Input Specification:

You will receive:

- **text**: the text to be corrected

Output Specification:

You will need to output:

- a string of corrected text according to the correction rules

Sample input:

```
pour la compétiTion de la cRc de ceTTe année voUs devez écrire
en pYthon. Vous aLLez voir c'eST FAcile comme lanGage?
FaUTe de GRAND-MÈRE.
cRc.
TroP. De? pOINts! OUI Peut-ÊTRE.
```

Sample output:

```
Pour la compétition de la CRC de cette année vous devez écrire
en python! Vous allez voir c'est Facile comme langage!
Faute de Grand-mère!
CRC!
Trop! De! Points! Oui Peut-être!
```

First output explanation:

For the first output, the first correction made was that the capital letter at the beginning of the sentence was missing. In the word competition, the letters are lower-cased. For CRC, all letters are capitalized. The next correction is to replace the period at the end of the sentence with an exclamation mark. Finally, the last nuance to specify is that the word "Facile" keeps its capital F, because it's at the beginning of the word. The question mark must also be replaced by an exclamation mark.

TP2: Zipper (25 points)

You'll receive a message encrypted using the zipper encryption method. Zipper encryption involves separating the message character by character into two strings. You'll receive two strings without knowing which is the first and which is the second. You have to recompose the message in the right order.

You need to find a logic to always be able to know in which order to arrange the strings automatically. The messages you receive will always be in lower case.

Input Specification:

You will receive:

- **s1**: one of the two zipper sections in no specific order
- **s2**: one of the two zipper sections in no specific order

Output Specification:

You will need to output:

- la string recomposée dans le bon ordre

Sample input:

```
s1: h sti optto okytk  
s2: wyi hscmeiins rpi?
```

```
s1: l optto epormaind accetvamn u!  
s2: acmeiind rgamto el r s rietfn
```

```
s1: ial iprmrei o hthr!  
s2: fnlyzpe eg snnta ad
```

Sample output:

```
why is this competition so kryptik?  
la competition de programmation de la crc est vraiment fun!  
finally zipper merge is not that hard!
```

First output explanation:

For the first output, one of the two strings is longer than the other (s2). We can therefore guess that we start and end the zipper with string s2. So we take the first character of the string s2 w, then that of s1 h and s2 y, which give us one after the other the word why. The alternation continues until the end of the two strings. This gives us the message: why is this competition so kryptik?

TP3: Palindrôle (30 points)

While trying to create a new file compression format, Mr. C took a long look at a type of word with interesting properties: [palindromes](#). He notices that some words, like 'kayak', include symmetry and that others contain symmetry recursion, like 'kayak-a-kayak' would. He describes the amount of recursion within a word as the order of the palindrome. A word with no symmetry is a palindrome of order 0 (e.g. 'shoe') and a word with only one symmetry is a palindrome of order 1 (e.g. 'radar'). To find out whether a word is a palindrome of order greater than 1, we are left with 2 ways of deconstructing the word according to the number of letters it contains. If the word contains an even number of letters, we simply split it down the middle and evaluate the two new words. If the word contains an odd number of letters, we still split it into two resulting words, but we don't include the letter in the middle of the word. For example, 'miaim' becomes 'mi' and 'im'. For the initial word to be of order 2, the two resulting words must be the same and be palindromes themselves.

As input, you'll receive a list of strings. As output, you'll give a list of int with the palindrome order of the words in the list obtained as input.

Input Specification:

You will receive:

- *L*: a string containing a single word

Output Specification:

You will need to output:

- an integer that is the order of the palindrome

Sample input:

```
laval  
cook  
wwwwwwwww  
oborobo
```

Sample output:

```
1  
0  
3  
2
```

First output explanation:

"laval" is a 1st order palindrome, as its two halves are symmetrical when split down the middle. However, the remaining halves ("al" and "la") are not palindromes in themselves afterwards. So it's not a 2nd order palindrome.