TPOP Mini Challenges

AUTUMN TERM - PYTHON -

Introduction

The mini challenges are meant to challenge you as if you were a pioneer in the field of computing sciences. You could probably find the solution on the web but this is not the aim of these exercises. You should assume that you are the first person 'ever' to encounter the problem and work toward solving the problem.

You are strongly encouraged to work within a team, up to three persons. You don't need to be a skilled programmer to think about the problem, if you are just beginning programming why not team-up with someone with good programming skills (or with more experience). The best team are usually the team made up of people with mixed abilities and different skill set.

These exercises are not compulsory. They are relatively open as there is not a single solution. Some solutions are better than others; you will have to discuss how good yours is.

Finally, teaching assistants have been instructed not to help you with the challenges during practical sessions. However, my door is always open if you have any questions. I will be more than happy to discuss your solution with you, or give you guidance if you are stuck. The challenges are meant to be fun and not too easy, one could say quite difficult if not very difficult. So don't be discouraged if you don't succeed quickly. You may acquire some useful knowledge later during the year that may help you.

If you have an interesting challenge, do not hesitate to share it with me and if it is suitable I will add it to the list.

Useful Resources for graphics using Python:

- http://wiki.python.org/moin/TkInter
- http://docs.python.org/library/turtle.html

Useful Resource for writing games in Python

- http://www.pygame.org/news.html

Challenge I: prime numbers

THE PROBLEM

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself.

The property of being prime is called primality. A simple but slow method of verifying the primality of a given number n is known as trial division. It consists of testing whether n is a multiple of any integer between 2 and $\sqrt{n}$.

THE BASIC

Write a program that, given a number comprised between 2 and 49, returns if it is a prime number or not. We can assume that the computer knows (stores) that [2, 3, 5, 7] are prime numbers.

THE ADVANCED BIT

Write a program that, given a number greater than 2, returns if it is a prime number or not. We can assume that the computer at the start knows only that 2 is prime number. We should use a loop to test several numbers.

THE CLEVER ONE

Write a program that, given a number greater than 2, returns if it is a prime number or not. We can assume that the computer at the start knows only that 2 is prime number. Every time the program is ran, it should remember the prime numbers it has found before.

THE OLYMPIAN ONE

Taken from the 2012 British Informatics Olympiad.

Every integer greater than 1 can be uniquely expressed as the product of prime numbers (ignoring reordering those numbers). This is called the prime factorisation of the number.

For example:

- $100 = 2 \times 2 \times 5 \times 5$
- $101 = 101$ (since 101 is a prime number)

We are interested in the product of the distinct prime factors of a given number; in other words each number in the prime factorisation is to be used only once. Since $100 = 2 \times 2 \times 5 \times 5$ the product we require is 10 (i.e. 2×5). Write a program which reads in a single integer n ($1 < n < 1{,}000{,}000$) and outputs a single integer, the product of the distinct prime factors of n.

Challenge II: The Turtle and the Maze

The Problem

The aim of our program is to represent a maze and a moving turtle trying to exit the maze. A turtle starts form the position S and must exit via the door at position F. We assume that a turtle can move only up, down, left and right. A unit of time is represented by one iteration, e.g. one movement for each turtle in the maze.

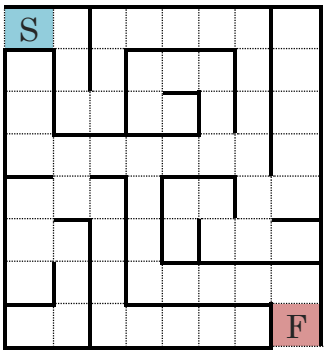


Figure 1: *a* **8** × **8** *maze*

The Basic

Define a data structure to represent a maze such as the one in *Figure 1*. What file format could be used to save and retrieve a maze?

Use the Turtle module to draw a maze. Using the same module for the rest of the challenge, draw a turtle moving randomly in a maze.

Examples of, and documentation on using `turtle` can be found at:
http://docs.python.org/library/turtle.html

The Advanced Bit

The lone turtle

The aim for the lone turtle is simply to find the exit.

The explorer and the one limb turtle

Yes you guess it right, in my world turtles walk on two legs, not four. This time, our unlucky lone turtle is not alone any more. However it is stuck in a maze with a one limb mate. To ease the suffering of his mate, the lone turtle must find a path to the exit and then send the shortest path he found to the one limb turtle. Of course they both have a smartphone, who doesn't have one nowadays?

THE CLEVER ONE

The team of explorers

This time we have three turtle explorers, each one with a tablet, and all tablets are connected so they can share the same map of the maze. Each explorer can update his knowledge of the maze and share is discovery to his teammates. The aim is to have all three explorers out of the maze as quickly as possible. The time stops when the last explorer is out.

The scouts and the battalion

The principle is quite similar to the previous problem. The battalion consists of X scouts and Y turtle troopers. The scouts explore the maze until they found one path to the exit. Once they have found a path they send it to the troopers, and from now on one trooper enters the maze at every iterations. The scouts are allowed to continue exploration to find a shorter path. The scouts can update the best path for each individual trooper. The aim is to get all the troopers out of the maze as quickly as possible. Imagine you are competing with another team.

Challenge III: Soduku

THE PROBLEM

Sudoku is an easy to learn logic-based number placement puzzle (see *Figure 2*). The word Sudoku is short for Su-ji wa dokushin ni kagiru which means "the numbers must be single".

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

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

Figure 2: *Left a* **4 × 4** *Sudoku for proof of concept and right a traditional* **9 × 9** *Sudoku.*

The roots of the Sudoku puzzle are in the Switzerland. Leonhard Euler created "carré latin" in the 18h century which is similar to a Sudoku puzzle but without the additional constraint on the contents of individual regions. The first real Sudoku was published in 1979 and was invented by Howard Garns, an American architect.

The Rules can be found at http://www.sudoku-space.com/sudoku.php. You should use a 4 × 4 puzzle to test your program as proof of concept (less computationally intensive). When you think you have a solution you should then test it with a 9 × 9 puzzle.

THE BASIC

Define a data structure to represent a Sudoku puzzle and define a file format to save it on disk. Write an algorithm to create a valid Sudoku table.

THE ADVANCED BIT

Write a program that take a seed table, such as the table in *Figure 3*, and return a valid, completed Sudoku table if it exists.

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

Figure 3: *Seed table*

THE CLEVER ONE

The speedy one

Assume you are competing in a programming challenge. You have to devise an algorithm that takes a puzzle similar to the puzzle in *Figure 3* and solves it as quickly as possible. Your program will have to go through a large number of puzzles and the winner is the one that solves all of them in the shortest time.

The extensive one

Given the puzzle in *Figure 3*, or one similar, write a program that enumerates all its possible solutions, or return impossible if no solution exists.

Challenge IV: Fractal Trees

THE PROBLEM

Fractal trees and plants are among the easiest of fractal objects to understand. They are based on the idea of self-similarity. As can be seen from the example of a fractal trees below

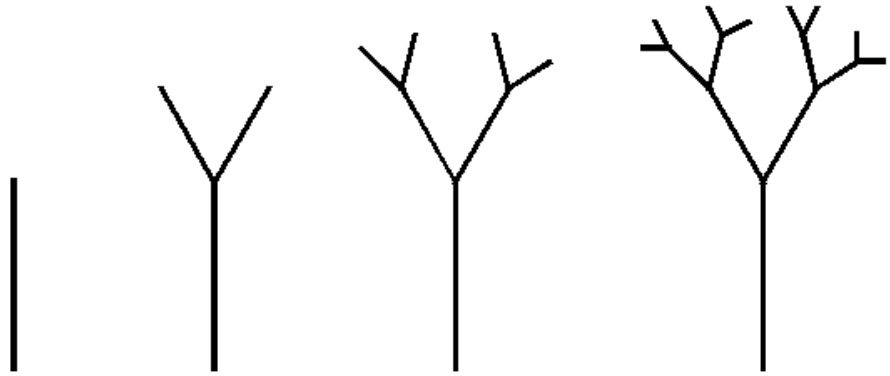
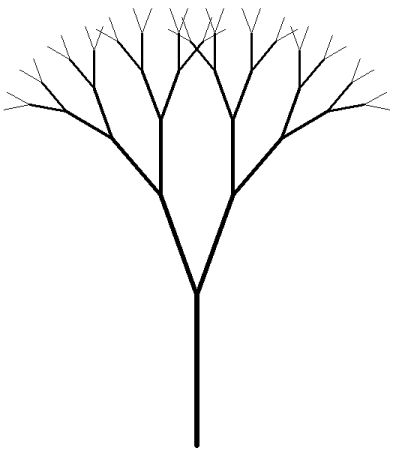


Figure 4: *From left to right, binary trees of depth 0, 1, 2, and 3.*

These trees clearly show the idea of self-similarity. Each of the branches is a smaller version of the main trunk of the tree. The main idea in creating fractal trees or plants is to have a base object and to then create smaller, similar objects protruding from that initial object. This method is a recursive method, meaning that it continues for each child down to a finite number of steps.

THE BASIC

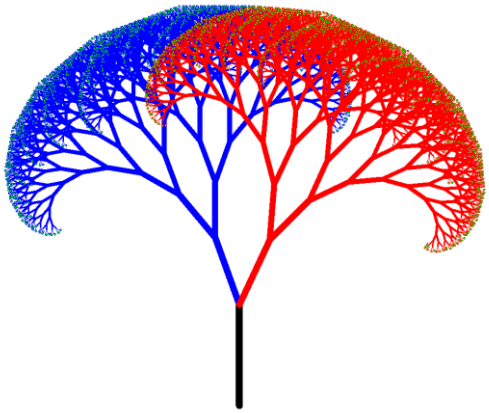


Write a program to draw such a binary tree. Practical 8 on recursion can be very useful. To draw the tree on the left I have used the `turtle` module from Python.

Examples of, and documentation on using `turtle` can be found at:

http://docs.python.org/library/turtle.html

THE ADVANCED BIT

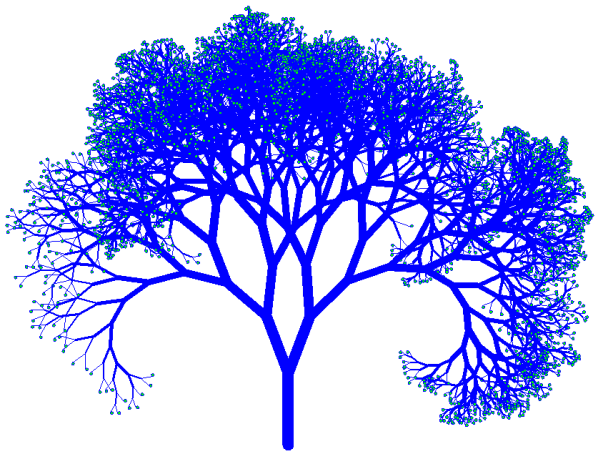


Binary trees are also used to represent data structure. To visualise such structures, we can draw it on a 2D plane (as shown on the left).

The problem encountered with the representation is that the left (blue) and right (red) children may overlap, making following a path from the root to a leaf very difficult. Define a representation that eliminate the problem and write a program to display your representation.

Later on, during the spring term you will learn more about Trees and data structure. You will learn to write program to create such data structures. Then you should try to write a program that takes a binary tree as input and outputs a drawing of that tree using your representation.

THE ARTISTIC ONE



The angle, length and other features of the "children" can be randomized for a more realistic look. At the last iteration of the tree or plant you can draw a leaf of some type depending on the nature of the plant or tree that you are trying to simulate. The image on the left was done using `turtle`.

Write a program that will create the most artistic tree, still using the principles laid down in the problem description. You should right a small abstract of no more than 500 words explaining how you achieved such master piece. We may even do a small competition to vote the best three solutions. You can even try to draw any type of trees (more than two children).

If you are really interested in fractals, more can be found at:
http://math.bu.edu/people/bob/papers.html

Challenge V: Staker

The Problem

From the South African Computer Olympiad: Third Round 2009

Stacker is a two player game. There is a shared stack of N numbers and each player takes it in turns to remove between 1 and M values (inclusive) from the top of the stack. A player's score is the sum of every value they remove from the stack. The game is over when every value has been removed from the stack. The objective of the game is to maximise your score.

The Task

The opposing player plays such that his final score is as large as possible. Assuming you start and both players play perfectly (they can plan ahead through the whole game, and never play sub-optimally), find the maximum score you can get in a game of Stacker.

Example

In the sample input you can remove 1, 2 or 3 values per turn and your stack initially looks like this:

5 2 0 1 4 3 5 2 0 0

At each turn you remove values from the right. The best you can score is 12, causing the opponent to score 10. This is done by:

Player	Removed	Stack	Score	
			You	Opp.
		5 2 0 1 4 3 5 2 0 0	0	0
You	0	5 2 0 1 4 3 5 2 0	0	0
Opponent	0, 2, 5	5 2 0 1 4 3	0	7
You	3, 4	5 2 0 1	7	7
Opponent	1, 0, 2	5	7	10
You	5		12	10

Input

The first line of the input contains two space-separated integers: N and M. The next N lines each contain a single integer, x_i. These are the values on the stack at the start of the game. The first value is at the bottom of the stack (so will be removed last). The last value is at the top of the stack (so will be removed first).

Output

The output contains a single integer T. T is the largest value you can score assuming the opposing player maximises his score.