Class: S7 ICT

Date: Thursday, May the 12th, 2022

Teacher: Mr Barsamian

B Test — With computer

Family name: _____

First name: _____

Grade: ___ / 10

Duration: 1 hour (60 minutes).

This test has to be done both on paper and on computer. At the end of the test, make sure to upload the python file on the Teams assignment (or on your teacher's USB key).

Some questions are bonuses, and it is highly advised to do them only at the end, when everything else has been done.

If needed, the candidate can also handle some comments inside the code or on paper.

Please keep track of the clock, and avoid spending too much time on a question. Stay focused, and good luck!

Short description of this work:

You just received a labyrinth game, and you want to solve it using a Tree data structure, on computer.

We will thus simulate labyrinths in Python. The start of a labyrinth is the root of the Tree, and you have to find your way out.

1 Introduction — 0 points

Please start by downloading the following file, that you'll have to update for this test:

http://www.barsamian.am/2021-2022/S7ICTA/BTest_Labyrinth.py

A labyrinth consists of cells. To move, you must follow the direction of arrows.

We will model the labyrinths seen in this test by trees (the direction of arrows is done in such a way that a tree is a valid data structure to model our labyrinths), see Listing 1. The left tree is the cell that you'll reach if you turn left in the labyrinth (when possible), the straight tree is the cell you'll reach if you go straight in the labyrinth (when possible), and the right tree is the cell you'll reach if you go right in the labyrinth (when possible). Finally, each cell has some data which explains what's located on it (the start of the labyrinth, the end of labyrinth, empty otherwise).

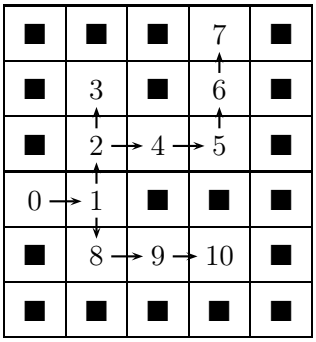
```
1  class Tree:
2      def __init__(self, data, left=None, straight=None, right=None):
3          self.data = data
4          self.left = left
5          self.straight = straight
6          self.right = right
7
8      def __str__(self):
9          return str(self.data)
```

Listing 1: Python code for the Tree data structure.

The file contains Listing 2, which implements the labyrinth on the right. Each cell has a number on the picture, to match the definition in the Python implementation. The starting cell is cell number 0, and you exit the labyrinth from cell number 7:

```
1  lab1_cell7 = Tree('End')
2  lab1_cell6 = Tree('', None, lab1_cell7, None)
3  lab1_cell5 = Tree('', lab1_cell6, None, None)
4  lab1_cell4 = Tree('', None, lab1_cell5, None)
5  lab1_cell3 = Tree('')
6  lab1_cell2 = Tree('', None, lab1_cell3, lab1_cell4)
7  lab1_cell10 = Tree('')
8  lab1_cell9 = Tree('', None, lab1_cell10, None)
9  lab1_cell8 = Tree('', lab1_cell9, None, None)
10 lab1_cell1 = Tree('', lab1_cell2, None, lab1_cell8)
11 lab1_start = Tree('Start', None, lab1_cell1, None)
```

Listing 2: Python code for the first labyrinth.



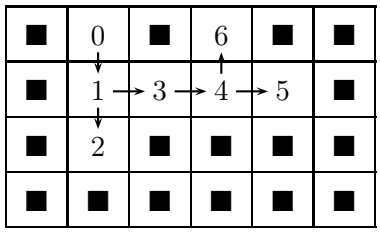
2 First steps — 6 points

1. The file also contains Listing 3. Please draw, on paper, a labyrinth which would correspond to this implementation.

```
lab2_cell5 = Tree('End')
lab2_cell4 = Tree('')
lab2_cell3 = Tree('', lab2_cell4, None, lab2_cell5)
lab2_cell2 = Tree('', None, lab2_cell3, None)
lab2_cell1 = Tree('', None, lab2_cell2, None)
lab2_start = Tree('Start', None, lab2_cell1, None)
```

Listing 3: Python code for the second labyrinth.

2. Let us consider the labyrinth drawn below. Please write, in the Python file, a Tree which would model this labyrinth (start in cell 0, end in cell 6).



3. In the Python file, the function `mystery` is provided (also given in Listing 4) which takes as argument a tree. Please:

   (a) explain step by step what happens if you execute `mystery(lab1_cell6)` and if you execute `mystery(lab2_start)`;

   (b) explain what `mystery` does in the general case.

```
def mystery(tree):
    if tree==None:
        return False
    if tree.data=='End':
        return True
    elif mystery(tree.left):
        print("Left")
        return True
    elif mystery(tree.straight):
        print("Straight")
        return True
    elif mystery(tree.right):
        print("Right")
        return True
    else:
        return False
```

Listing 4: Python code for the mystery function.

3 Find your way out — 4 points

1. Please write a function `count` that takes as argument a tree and that counts the number of nodes in that tree.

   Unit tests:

   - `count(lab1_start)` should return 11;
   - `count(lab2_start)` should return 6.

2. Please write a function `is_correct` that takes as argument a tree and that returns True if the tree has exactly one cell where data is "Start" and one cell where data is "End", and that returns False otherwise.

   Unit tests:

   - `is_correct(lab1_start)` should return True;
   - `is_correct(lab1_cell2)` should return False;
   - `is_correct(lab2_start)` should return True;
   - `is_correct(lab2_cell5)` should return False.

BONUS Please write a function `length` that takes as argument a tree and that counts the number of nodes you need to go through from the starting cell from the ending cell.

Unit tests:

- `length(lab1_start)` should return 6;
- `length(lab2_start)` should return 5.