UNIVERSITY OF TORONTO

Faculty of Arts and Science

term test #2, Version 3
CSC148H1S

Date: Wednesday March 14, 6:10–7:00 p.m.
Duration: 50 minutes
Instructor(s):
AbdulAziz Alhelali
Arnamoy Bhattacharyya
Danny Heap

No Aids Allowed

Name:

utorid:

U of T email:

Please read the following guidelines carefully!

- **Please write your name, utorid, and student number on the front of this exam.**
- This examination has **4** questions. There are a total of **7 pages, DOUBLE-SIDED**.
- Answer questions clearly and completely.
- You will receive 20% of the marks for any question you leave blank or indicate "I cannot answer this question."

Take a deep breath.
This is your chance to show us
How much you've learned.

We **WANT** to give you the credit

Good luck!

1. **[4 marks] tracing recursion.** Read the definition of function **m3** below, then trace the calls on that function, using the tracing technique from class. Remember, if there are recursive sub-calls on lists you have evaluated above **do not** expand them further, just replace them by their values.

```
def m3(list_: list, num: int) -> int:
    """ docstring omitted!
    """

    if num == 1:
        return max([x for x in list_ if not isinstance(x, list)] + [0])
    elif num > 1:
        return max([m3(x, num - 1) for x in list_ if isinstance(x, list)] + [0])
    else:
        return 0
```

(a) `>>> m3([6], 1)`

Solution

```
--> max([6, 0]) --> 6
```

(b) `>>> m3([1, [2, 3, 4], 5], 1)`

Solution

```
--> max([1, 5, 0]) --> 5
```

(c) `>>> m3([7, [1, [2, 3, 4], 5], [6]], 2)`

Solution

```
--> max([m3([1, [2, 3, 4], 5], 1), m3([6], 1), 0]) --> max(5, 6, 0) --> 6
```

(d) `>>> m3([[7, [1, [2, 3, 4], 5], [6]]], 3)`

Solution

```
--> max([m3([7, [1, [2, 3, 4], 5], [6]], 2), 0]) --> max([6, 0]) --> 6
```

2. **[6 marks] binary tree structure**. Read the (very abbreviated!) declaration of **BTNode** below. Assume that the Python statements below the class declaration have been executed, then answer the questions. **Hint**: you may find it helpful to make a sketch of the tree.

```
class BTNode:
    """Binary Tree node."""

    def __init__(self, data: object,
                 left: Union["BTNode", None]=None,
                 right: Union["BTNode", None]=None) -> None:
        """
        Create BTNode (self) with data and children left and right.

        An empty BTNode is represented by None.
        """
        self.data, self.left, self.right = data, left, right

>>> t1 = BTNode(7, BTNode(5, BTNode(11), BTNode(9)))
>>> t2 = BTNode(13, BTNode(12, BTNode(17)))
>>> t3 = BTNode(10, t1, t2)
```

(a) What is the **data** of t3's root node?

Solution

10

(b) What is the arity (branching factor) of the tree rooted at t3?

Solution

2

(c) Which of the tree rooted at t3's nodes is/are the leaves?

Solution

11, 9, 17

(d) Write down the values of the nodes if the tree rooted at t3 is visited in a **levelorder** traversal.

Solution

10, 7, 13, 5, 12, 11, 9, 17 (left)

10, 13, 7, 12, 5, 17, 9, 11 (right)

(e) What is the length of the longest path in the tree rooted at `t3`?

Solution

3

(f) What is the height of the tree rooted at `t3`?

Solution

4

3. **[5 marks] binary tree distance**. Read the (very abbreviated) declaration of class `BTNode` below. Then implement the body of `btnode_string_distance`. You may **not** assume any other functions or methods for binary tree nodes, unless you define them.

```
from typing import Union


class BTNode:
    """Binary Tree node."""

    def __init__(self, data: object,
                 left: Union["BTNode", None]=None,
                 right: Union["BTNode", None]=None) -> None:
        """
        Create BTNode (self) with data and children left and right.

        An empty BTNode is represented by None.
        """
        self.data, self.left, self.right = data, left, right


def btnode_string_distance(node: Union[BTNode, None], d: int) -> str:
    """ Return concatenation of data distance d from node.

    Assume all data in tree rooted at node are strings and d is non-negative.

    >>> btnode_string_distance(None, 1)
    ''
    >>> bt = BTNode("a", BTNode("b"), BTNode("c"))
    >>> btnode_string_distance(bt, 0)
    'a'
    >>> btnode_string_distance(bt, 1)
    'bc'
    >>> btnode_string_distance(bt, 2)
    ''
    """
```

Solution

```
    node.data: str
    if node is None:
        return ""
    elif d < 0:
        return ""
    elif d == 0:
        return node.data
    else:
        return btnode_string_distance(node.left, d - 1) + btnode_string_distance(node.right, d - 1)
```

4. **[5 marks] general tree.** Read the (very abbreviated) declaration of class Tree below. Then implement the body of list_postorder. You may **not** use any functions or methods for trees unless you define them here.

```
from typing import List


class Tree:
    """
    Abbreviated Tree class
    """

    def __init__(self, value: object, children: List["Tree"]=None) -> None:
        """
        Create Tree self with content value and 0 or more children
        """
        self.value = value
        self.children = children[:] if children is not None else []


def list_postorder(t: Tree) -> list:
    """
    Return a list of t's values in postorder.

    >>> list_postorder(Tree(0))
    [0]
    >>> t = Tree(1, [Tree(2, [Tree(3)]), Tree(4)])
    >>> list_postorder(t)
    [3, 2, 4, 1]
    """
```

Solution

```
        if t.children == []:
        return [t.value]
    else:
        return sum([list_postorder(c) for c in t.children], []) + [t.value]
```