Easy4.0 CSC148

Mock Final Exam I I

This Final Examination paper consists of 6questions with a sum of 76 points in total, and you must earn at least 40% to pass the exam. Comments and docstrings are not required except where indicated, although they may help us mark your answers.

- You do not need to put import statements in your answers.
- No error checking is required: assume all user input and all argument values are valid.
- If you use any space for rough work, indicate clearly what you want marked.

Name:________________

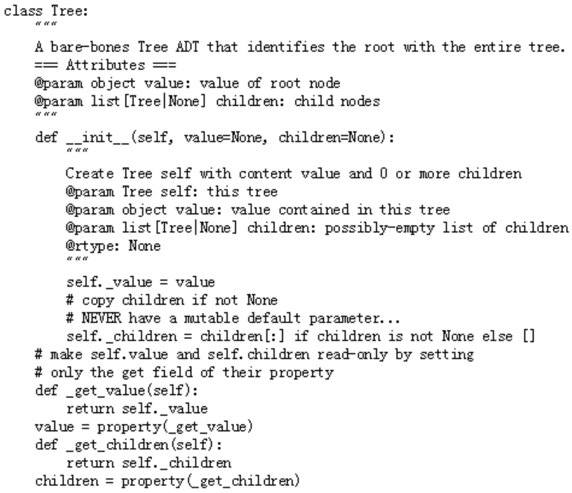
.

Marking Guide

Q1	/12
Q2	/12
Q3	/12
Q4	/12
Q5	/12
Q6	/6
Total	/66

Question 1 [12 Marks]

Recall the Tree data structure we've defined in class.



```
class Tree:
    """
    A bare-bones Tree ADT that identifies the root with the entire tree.
    === Attributes ===
    @param object value: value of root node
    @param list[Tree|None] children: child nodes
    """
    def __init__(self, value=None, children=None):
        """
        Create Tree self with content value and 0 or more children
        @param Tree self: this tree
        @param object value: value contained in this tree
        @param list[Tree|None] children: possibly-empty list of children
        @rtype: None
        """
        self._value = value
        # copy children if not None
        # NEVER have a mutable default parameter...
        self._children = children[:] if children is not None else []
    # make self.value and self.children read-only by setting
    # only the get field of their property
    def _get_value(self):
        return self._value
    value = property(_get_value)
    def _get_children(self):
        return self._children
    children = property(_get_children)
```

Part (a) [4 Marks]

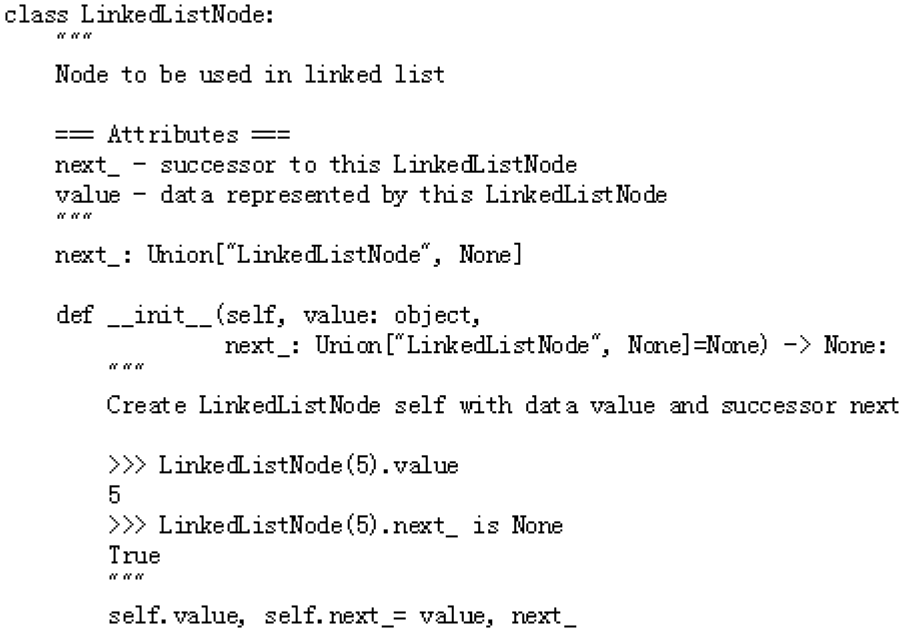
Implement the method "extend_leaves" defined below according to its docstring.

```
def extend_leaves(t: Tree) -> None:
  """ Add a child to each leaf in the tree with the same value, so that the
     original leaves become internal nodes.

  Precondition: The tree is not empty.

  >>> t = Tree(5)
  >>> extend_leaves(t)
  >>> str(t)
  5
    5
  """
```

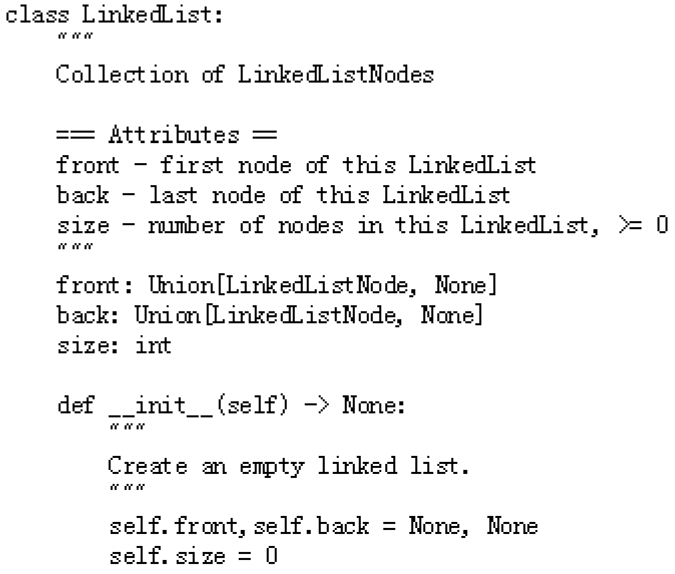
Part (b) [8 Marks]

```
def extend_even_level(t: Tree) -> None:
  """ Extend each even level node in the tree with the same value below it.

  Precondition: The tree is not empty.

  >>> t = Tree(5, [Tree(3), Tree(2)])
  >>> extend_leaves(t)
  >>> str(t)
  5
    3
      3
    2
      2
  """
```

Question 2 [12 Marks]
Recall the linkedList we have learned in class.



```
class LinkedListNode:
    """
    Node to be used in linked list

    === Attributes ===
    next_ - successor to this LinkedListNode
    value - data represented by this LinkedListNode
    """
    next_: Union["LinkedListNode", None]

    def __init__(self, value: object,
                 next_: Union["LinkedListNode", None]=None) -> None:
        """
        Create LinkedListNode self with data value and successor next

        >>> LinkedListNode(5).value
        5
        >>> LinkedListNode(5).next_ is None
        True
        """
        self.value, self.next_= value, next_
```



```
class LinkedList:
    """
    Collection of LinkedListNodes

    === Attributes ==
    front - first node of this LinkedList
    back - last node of this LinkedList
    size - number of nodes in this LinkedList, >= 0
    """
    front: Union[LinkedListNode, None]
    back: Union[LinkedListNode, None]
    size: int

    def __init__(self) -> None:
        """
        Create an empty linked list.
        """
        self.front,self.back = None, None
        self.size = 0
```

Part (a) Implement the following function for linkedlist [8 marks]

```
def merge(self, other: Linkedlist) -> Linkedlist:
  """ Assume both self and other are sorted linkedlists, return a new
  linkedlist that is the merge result from self and other.

  1. Do not change self and other.
  2. The returned linkedlist should not have any aliasing problem with self
     and other.

  Precondition: The tree is not empty.

  >>> a, b = Linkedlist(), Linkedlist()
  >>> a.load_list([1, 6])
  >>> b.load_list([3])
  >>> print(a.merge(b))
  1 -> 3 -> 6 ->|
  """
```

Part (b) Provide a Big-O runtime analysis on the implementation above [4 marks]

Question 3 [12Marks]

What’s the runtime of the following function, please indicate the Big-O runtime with a brief explaination, pick the worst case if applicable.

```
def mystery1(n):
    if n < 100:
        for temp in range(n):
            print("加油复习")
    while n > 1:
        print("加油复习")
        n = n / 2
```

```
def mystery2(n):
    if n == 1:
      print("加油复习")
    else:
      mystery2(n//2)
      mystery2(n//2)
```

```
def mystery2(n):
    if n % 2 == 1:
      for i in range(n * 2):
        print("加油复习")
    else:
      j = n
      while j < n ** 2:
         print("加油复习")
         j += 1
```

Question 4 [12 Marks]

In assignments, we have designed classes for different games, now please implement the game state for tic-tac-toe.

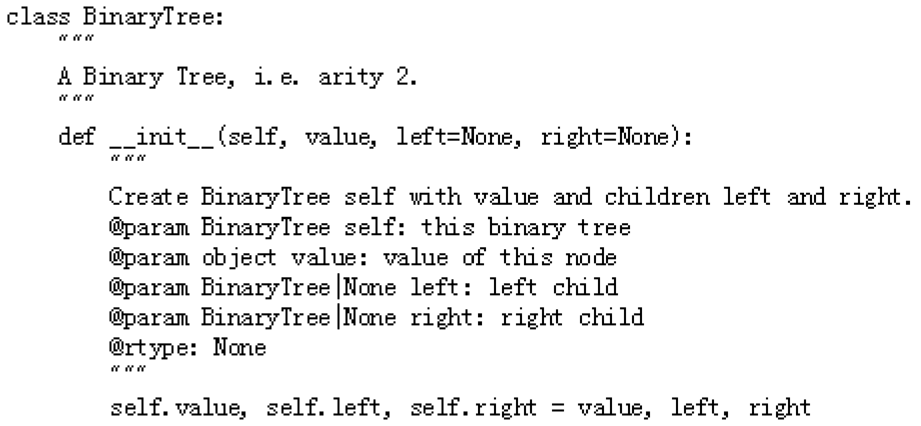
The tic-tac-toe game is two players game, *X* for player 1 and *O for player two*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. The game ends and results in tie if no one win but all the cells are taken.

Game states interface is provided as follow

```
class GameState:
    """
    The state of a game at a certain point in time.

    WIN - score if player is in a winning position
    LOSE - score if player is in a losing position
    DRAW - score if player is in a tied position
    p1_turn - whether it is p1's turn or not
    """
    WIN: int = 1
    LOSE: int = -1
    DRAW: int = 0
    p1_turn: bool

    def __init__(self, is_p1_turn: bool) -> None:
        """
        Initialize this game state and set the current player based on
        is_p1_turn.

        """
        self.p1_turn = is_p1_turn

    def get_possible_moves(self) -> list:
        """
        Return all possible moves that can be applied to this state.
        """
        raise NotImplementedError

    def get_current_player_name(self) -> str:
        """
        Return 'p1' if the current player is Player 1, and 'p2' if the current
        player is Player 2.
        """
        if self.p1_turn:
            return 'p1'
        return 'p2'

    def make_move(self, move: Any) -> 'GameState':
        """
        Return the GameState that results from applying move to this GameState.
        """
        raise NotImplementedError

    def is_valid_move(self, move: Any) -> bool:
        """
        Return whether move is a valid move for this GameState.
        """
        return move in self.get_possible_moves()
```

Question 5 [12 Marks]

Recall the Binary Search Tree data structure we’ve defined in class.



```
class BinaryTree:
    “““
    A Binary Tree, i.e. arity 2.
    “““
    def __init__(self, value, left=None, right=None):
        “““
        Create BinaryTree self with value and children left and right.
        @param BinaryTree self: this binary tree
        @param object value: value of this node
        @param BinaryTree|None left: left child
        @param BinaryTree|None right: right child
        @rtype: None
        “““
        self.value, self.left, self.right = value, left, right
```

Part (a) [2 Marks]

Draw **binary search tree**s with item 1 2 3 4 5 6 7 with both minimum and maximum height.

Part (b) [2 Marks]

Continue with previous question, is it possible to set 2 as root and still have the minimum height? Why?

Part (c) [8 Marks]

```
def recreate_tree(postorder: List[int]) -> BinaryTree:
    """Return the root of the binary search tree based on the given
    postorder of the binary search tree.
    """

    “
```

Question 6 [6 Marks]

Implement the following function without using any recursion, you may use any ADT from class.

```
def flatten_nested_list(nested_list]) -> list:
    """Return the list that is the flattened version of the given
    nested_list.
    >>> L = [1, [[7], 3]]
    >>> flatten_nested_list(L)
    [1, 7, 3]
    """
```