CSC148-Section:L0301 Week#8-Monday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Announcement

- A1 marks posted

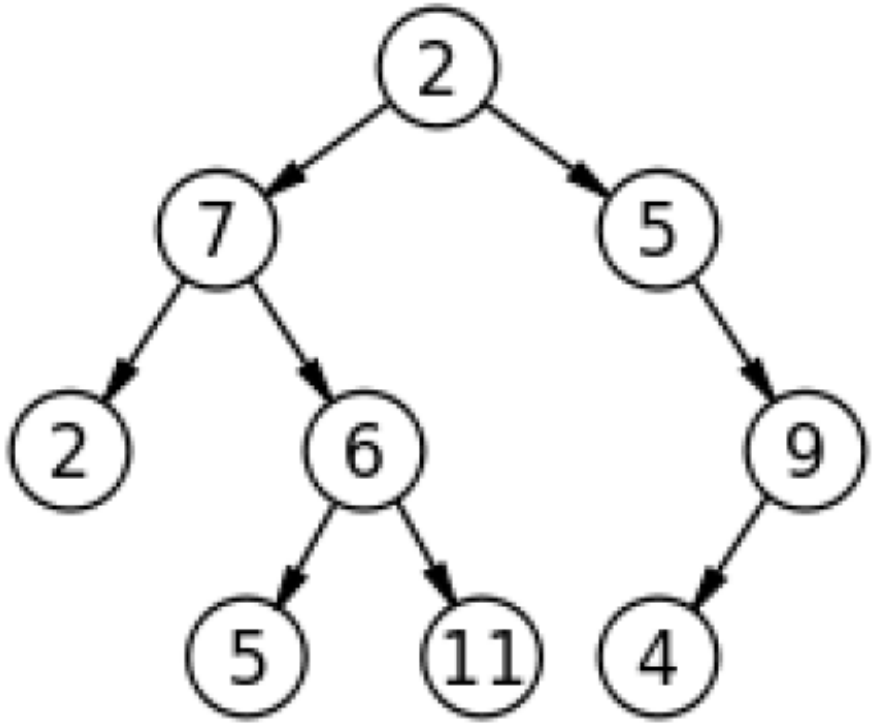
Outline

- Binary Trees

Binary Trees

- Each node: has at most two children
- Called: ***left child*** and the ***right child***

- ***Applications:***
  - Search algorithms
  - compression algorithms used in jpeg and .mp3
  - Compilers



BinaryTree

```
class BinaryTree:
    """
    A Binary Tree, i.e. arity 2.
    """

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

        """
        self.value, self.left, self.right = value, left, right
```

Computer Science
UNIVERSITY OF TORONTO

Contains – As Module Level functions

```
def contains(node: BinaryTree, value: object) -> bool:
    """
    Return whether tree rooted at self contains value.

    >>> t = BinaryTree(5, BinaryTree(7), BinaryTree(9))
    >>> contains(t,7)
    True
    """
    if node.left is None and node.right is None:
        return node.value == value
    else:
        return (node.value == value
                or contains(node.left,value)
                or contains(node.right,value))
```

The code **will NOT work** if the BinaryTree has one child as None ?
No, we need to handle those cases see next slide

```
def contains(node: Union[BinaryTree, None], value: object) -> bool:
    """
    Return whether tree rooted at self contains value.

    >>> t = BinaryTree(5, BinaryTree(7), BinaryTree(9))
    >>> contains(t,5)
    True
    >>> contains(t,2)
    False
    >>> t1 = BinaryTree(5, BinaryTree(7,BinaryTree(3)), None)
    >>> contains(t1,1)
    False
    """
    if node.left is None and node.right is None:
        return node.value == value
    elif node.left is None:
        return node.value == value or contains(node.right, value)
    elif node.right is None:
        return node.value == value or contains(node.left, value)
    else:
        return (node.value == value
                or contains(node.left, value)
                or contains(node.right, value))
```

If either left or right nodes is None
We should not call contains on that branch of the tree

```
def contains(node: Union[BinaryTree, None], value: object) -> bool:
    """
    Return whether tree rooted at self contains value.

    >>> t = BinaryTree(5, BinaryTree(7), BinaryTree(9))
    >>> contains(t,5)
    True
    >>> contains(t,2)
    False
    >>> t1 = BinaryTree(5, BinaryTree(7,BinaryTree(3)), None)
    >>> contains(t1,1)
    False
    """
    if node is None:
        return False
    elif node.left is None and node.right is None:
        return node.value == value
    else:
        return (node.value == value
                or contains(node.left, value)
                or contains(node.right, value))
```

Base Case

An alternative shorter solution that breaks the base case to two parts

Find – As Module Level functions

```
def find(node: Union[BinaryTree, None], data: object)-> Union[BinaryTree, None]:
    """
    Return BinaryTree containing data or else None.

    >>> find(None,15) is None
    True
    >>> bt = BinaryTree(5, BinaryTree(4),BinaryTree(3))
    >>> find(bt,7) is None
    True
    >>> find(bt,4)
    BinaryTree(4, None, None)
    >>> find(bt,3)
    BinaryTree(3, None, None)
    """
```

Computer Science
UNIVERSITY OF TORONTO

```
def find(node: Union[BinaryTree, None], data: object)-> Union[BinaryTree, None]:
    """
    Return BinaryTree containing data or else None.

    >>> find(None,15) is None
    True
    >>> bt = BinaryTree(5, BinaryTree(4),BinaryTree(3))
    >>> find(bt,7) is None
    True
    >>> find(bt,4)
    BinaryTree(4, None, None)
    >>> find(bt,3)
    BinaryTree(3, None, None)
    """
    if node is None:
        return None
    else:
        if node.value == data:
            return node
        elif find(node.left,data) is not None:
            return find(node.left,data)
        else:
            return find(node.right, data)
```

Height

```
def height(node: Union[BinaryTree, None]) -> int:
    """
    Return heigh of Binary tree.

    >>> t = BinaryTree(5, BinaryTree(7), BinaryTree(9))
    >>> height(t)
    2
    """
```

Height

```
def height(node: Union[BinaryTree, None]) -> int:
    """
    Return heigh of Binary tree.

    >>> t = BinaryTree(5, BinaryTree(7), BinaryTree(9))
    >>> height(t)
    2
    """
    if node.left is None and node.right is None:
        return 1
    else:
        return 1 + max(height(node.left),height(node.right))
```

The code **will NOT work** if the BinaryTree has one child as None ?
No, we need to handle those cases see next slide

```
  File "D:/csc148/lectures/week8/binary trees.py", line 217, in height
    if node.left is None and node.right is None:
AttributeError: 'NoneType' object has no attribute 'left'
```

Height

```
def height(node: Union[BinaryTree, None]) -> int:
    """
    Return heigh of Binary tree.

    >>> t = BinaryTree(5, BinaryTree(7), BinaryTree(9))
    >>> height(t)
    2
    """
    if node is None:
        return 0
    else:
        return 1 + max(height(node.left),height(node.right))
```