CSC148-Section:L0301
Week#9-Firday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

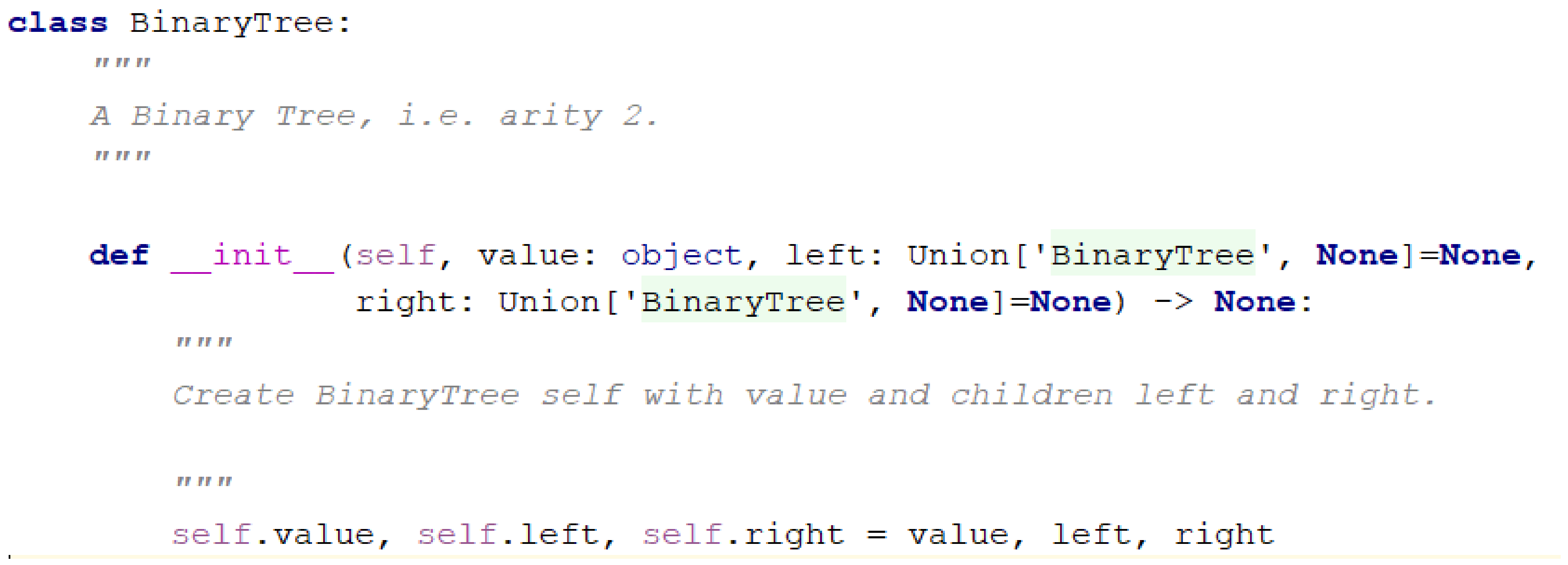
Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Binary Search Tree
    - insert
    - find_max
    - delete

Recall – BinaryTree node



```
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
```

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

    Assume node is the root of a Binary Search Tree

    >>> bst_contains(None, 5)
    False
    >>> bst_contains(BinaryTree(7, BinaryTree(5), BinaryTree(9)), 5)
    True
    """
```

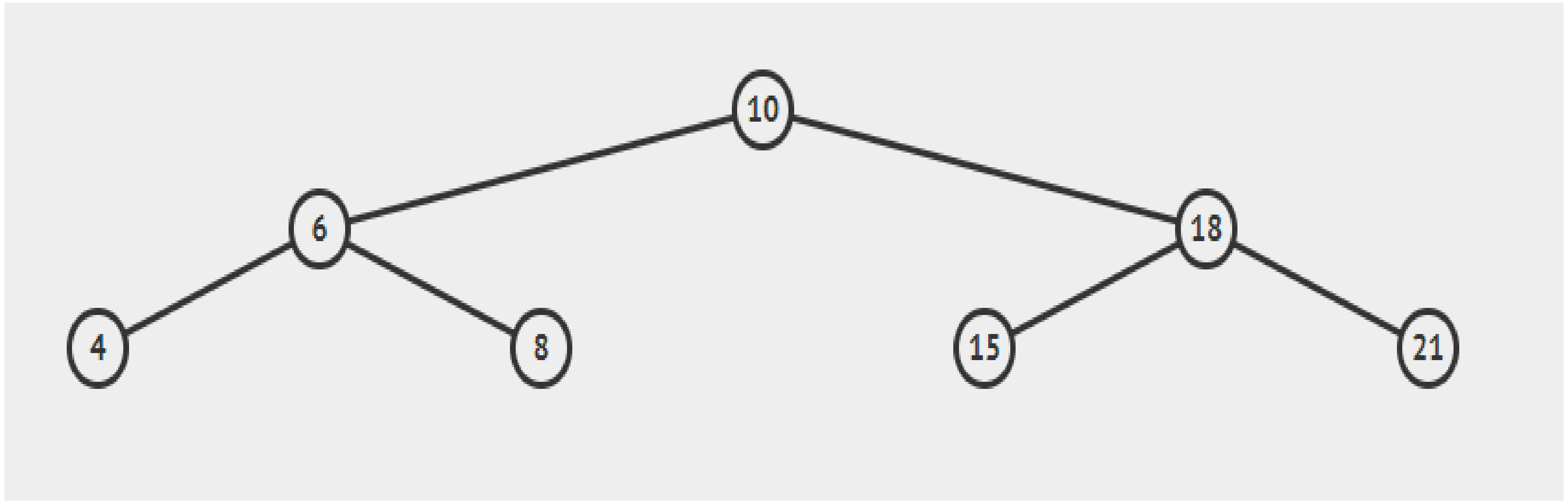
```python
def bst_contains(node: BinaryTree, value: object) ->bool:
    """
    Return whether tree rooted at node contains value.

    Assume node is the root of a Binary Search Tree

    >>> bst_contains(None, 5)
    False
    >>> bst_contains(BinaryTree(7, BinaryTree(5), BinaryTree(9)), 5)
    True
    """
    if node is None:
        return False
    elif node.value > value:
        return bst_contains(node.left, value)
    elif node.value < value:
        return bst_contains(node.right, value)
    else:
        return True
```

insert to BST

- Add new node to a BST must insure:
  - data in left subtree are less than root node
  - data in right subtree are more than root node

insert to BST

- Where do we add 9 and 13?

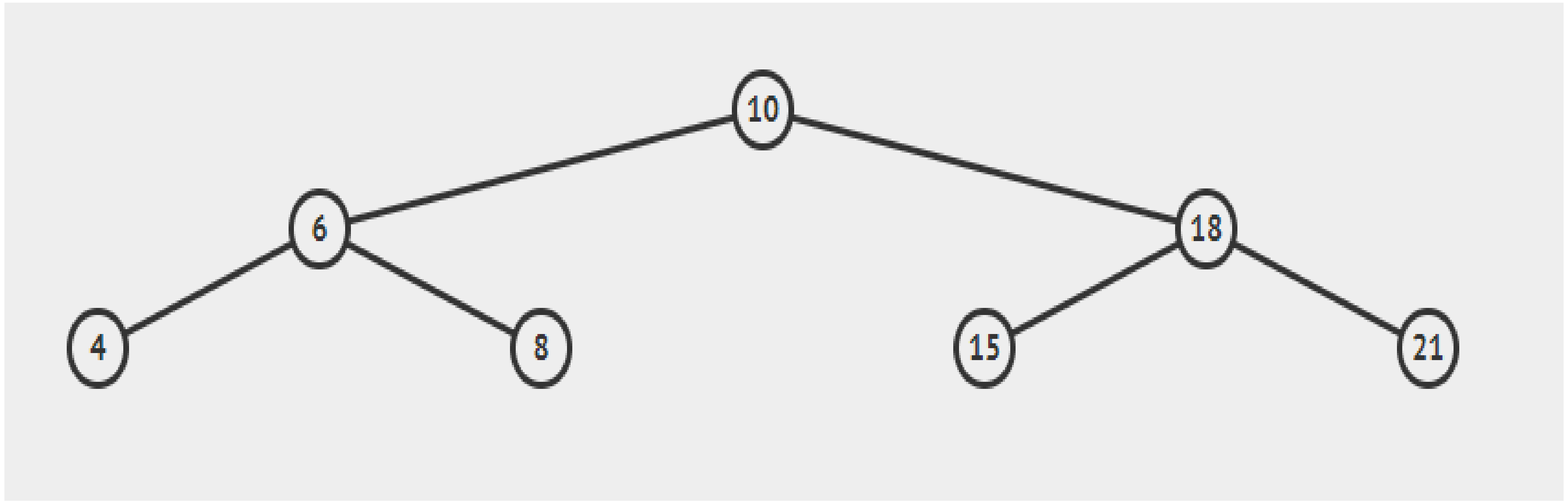


```
def insert(node: Union[BinaryTree, None], value: object)
        -> BinaryTree:
    """
    Insert value in BST rooted at node if necessary, and
    return new root.

    Assume node is the root of a Binary Search Tree.

    >>> b = BinaryTree(5)
    >>> b1 = insert(b, 3)
    >>> print(b1)
    5
        3
    <BLANKLINE>
    """
```

```python
def insert(node: Union[BinaryTree, None], value: object) -> BinaryTree:
    """
    Insert value in BST rooted at node if necessary, and return new root.

    Assume node is the root of a Binary Search Tree.

    >>> b = BinaryTree(5)
    >>> b1 = insert(b, 3)
    >>> print(b1)
    5
        3
    <BLANKLINE>
    """
    if node is None:
        node = BinaryTree(value)
    elif value > node.value:
        node.right = insert(node.right, value)
    elif value < node.value:
        node.left = insert(node.left, value)
    return node
```

find_max in BST

- What is the max?
- add 23. where is it located?



```python
def find_max(node: BinaryTree) ->BinaryTree:
    """
    Find and return subnode with maximum data.

    Assume node is the root of a binary search tree.

    >>> find_max(BinaryTree(5, BinaryTree(3), BinaryTree(7)))
    BinaryTree(7, None, None)
    """
```

```python
def find_max(node: BinaryTree) ->BinaryTree:
    """
    Find and return subnode with maximum data.

    Assume node is the root of a binary search tree.

    >>> find_max(BinaryTree(5, BinaryTree(3), BinaryTree(7)))
    BinaryTree(7, None, None)
    """
    if node.right is None:
        return node
    else:
        return find_max(node.right)
```

deletion of value from BST rooted at node?

- what return value?
- what to do if node is None?
- what if value to delete is less than that at node?
- what if it's more?
- what if the value equals this node's value and...
  - this node has no left child
  - ... no right child?
  - both children?

- **what return value?**
  - `return node`   (for every call to delete)

A. what to do if node is None?

```
A. if node is None:
           pass
```

B. what if value to delete is less than that at node?

```
A. #Branch to the left
B. elif value < node.value:
           node.left = delete(node.left, value)
```

C. what if it's more?

```
• #Branch to the right
• elif value > node.value:
           node.right = delete(node.right, value)
```

D. what if the value equals this node's value and... (neither greater nor smaller)

- **We have 3 cases:**

1. this node has no left child

   -
     ```
     elif node.left is None:
             node = node.right
     ```
2. ... no right child?

   -
     ```
     elif node.right is None:
            node = node.left
     ```
3. both children?
   - `# One way to` **`not break BST definition`**
   - `# find the` **`max`** `node in` **`left`** `tree and put it in place of`
   - `# deleted node`

     -
       ```
       node.value = find_max(node.left).value
       node.left = delete(node.left, node.value)
       ```
   - `# Alternatively`
   - `# find the` **`min`** `node in` **`right`** `tree and put it in place of`
   - `# deleted node`

     -
       ```
       node.value = find_min(node.right).value
       node.right = delete(node.right, node.value)
       ```

Con
UNIVERSITY OF TORONTO

algorithm

```
# Algorithm for delete:
# 1. If this node is None, return that
# 2. If value is less than node.value, delete it from left child and
#         return this node
# 3. If value is more than node.value, delete it from right child
#         and return this node
# 4. If node with value has fewer than two children,
#         and you know one is None, return the other one
# 5. If node with value has two non-None children,
#         replace value with that of its largest child in the left
#         subtree and delete that child, and return this node
```

```
def delete(node: Union[BinaryTree, None], value: object) \
        -> Union[BinaryTree, None]:
    """
    Delete data from binary search tree rooted at node, if it exists,
    and return root of resulting tree.

    >>> b = BinaryTree(8)
    >>> b = insert(b, 4)
    >>> b = insert(b, 2)
    >>> b = insert(b, 6)
    >>> b = insert(b, 12)
    >>> b = insert(b, 14)
    >>> b = insert(b, 10)
    >>> b = delete(b, 12)
    >>> print(b)
            14
        10
    8
            6
        4
            2
    <BLANKLINE>
    >>> b = delete(b, 14)
    >>> print(b)
        10
    8
            6
        4
            2
    <BLANKLINE>
    """
```

```
def delete(node: Union[BinaryTree, None], value: object) \
        -> Union[BinaryTree, None]:
    # 1. If this node is None, return that
    if node is None:
        pass
    # 2. If value is more than node.value, delete it from right child and
    #     return this node
    elif value > node.value:
        node.right = delete(node.right, value)
    # 3. If value is less than node.value, delete it from left child
    #     and return this node
    elif value < node.value:
        node.left = delete(node.left, value)
    # 4. If node with value has fewer than two children,
    #     and you know one is None, return the other one
    elif node.left is None:
        node = node.right
    elif node.right is None:
        node = node.left
    # 5. If node with value has two non-None children,
    #     replace value with that of its largest child in the left subtree,
    #     and delete that child, and return this node
    else:
        node.value = find_max(node.left).value
        node.left = delete(node.left, node.value)
    return node
```