Week 7 : Trees

Arnamoy Bhattacharyya

Agenda

1. Will learn about the ADT - **Tree**
2. A few definitions of tree-related terms
3. How to define Tree Class
4. A few exercises

Announcements

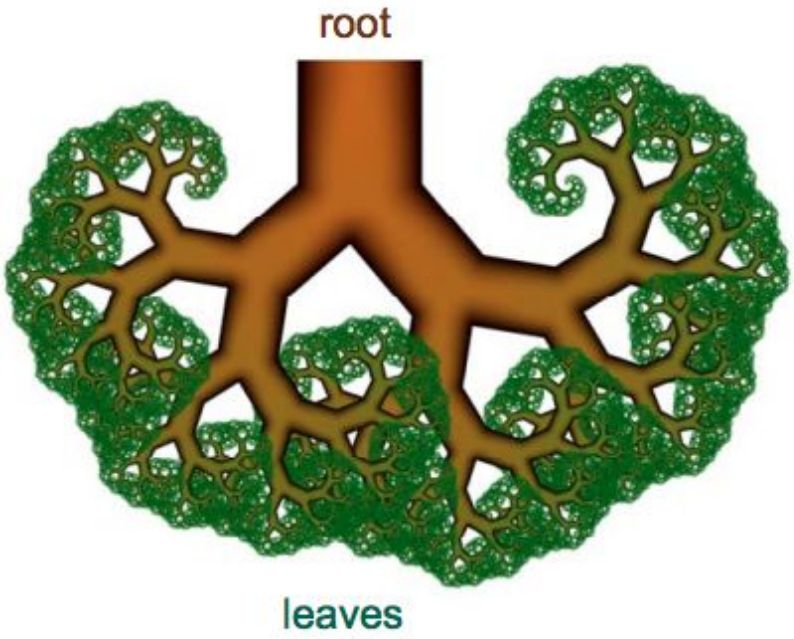
1. A2 is posted (Due March 6th)
2. Builds on top of Assignment 1
3. Have to implement:
    a. A new game (Stonehenge)
    b. A new game strategy (Minimax)
        i. Iterative (using stack)
        ii. Recursive
4. We have provided:
    a. game and game_state *super classes*
    b. A sample implementation of subtract square (to understand minimax)
    c. Description of both iterative and recursive minimax

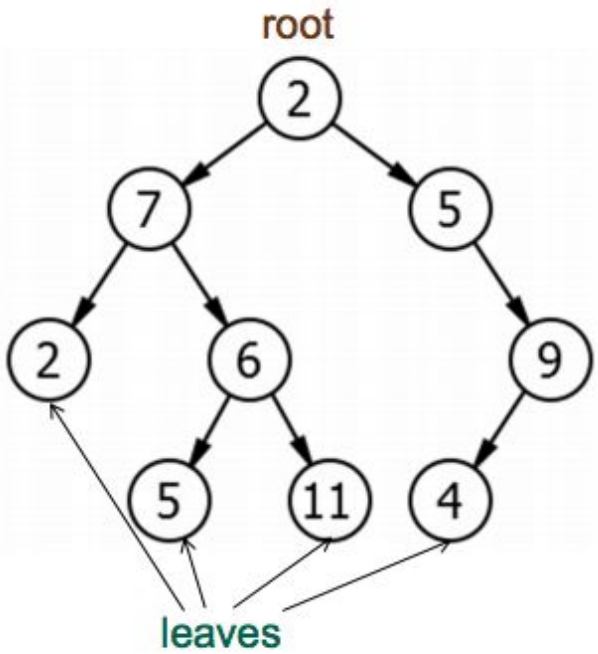
Announcements

- You will probably have an angry computer after you finish implementing the game and strategy
- Assignment 1 and Test 1 → Released next week
- Help Centre will be staffed during reading week

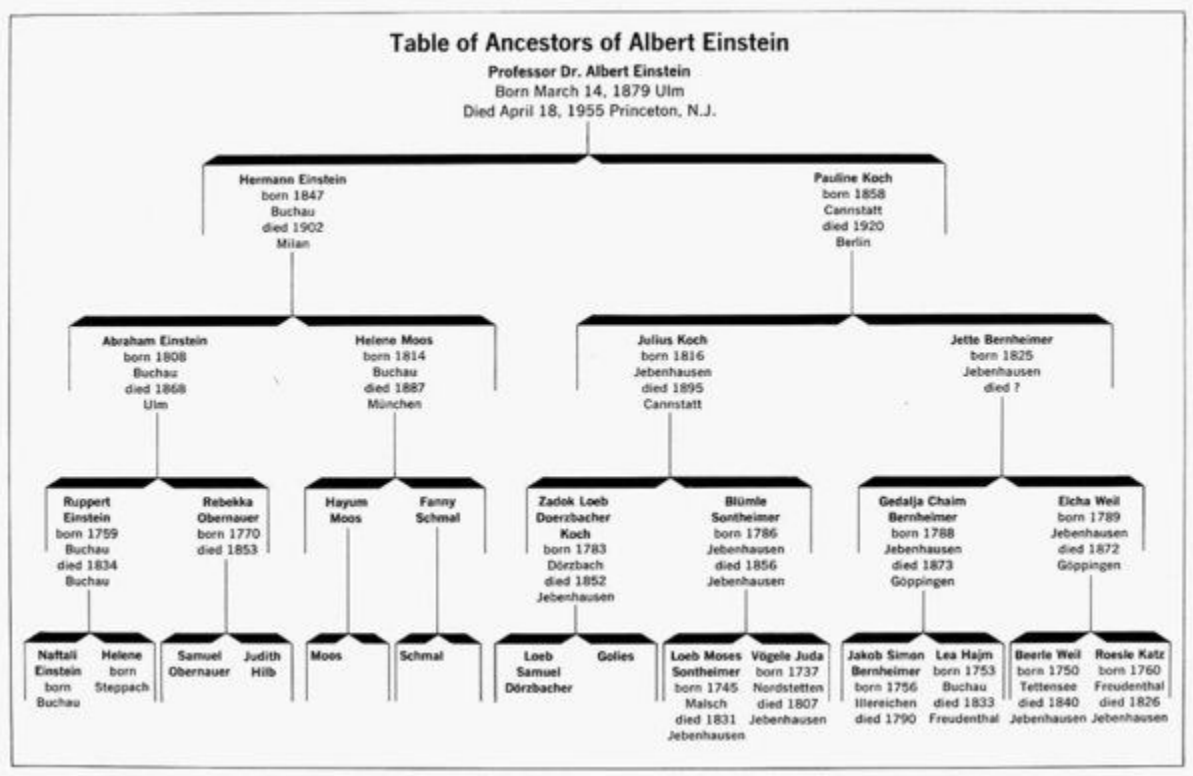
Trees in Nature

Trees in Nature vs in Computer Science

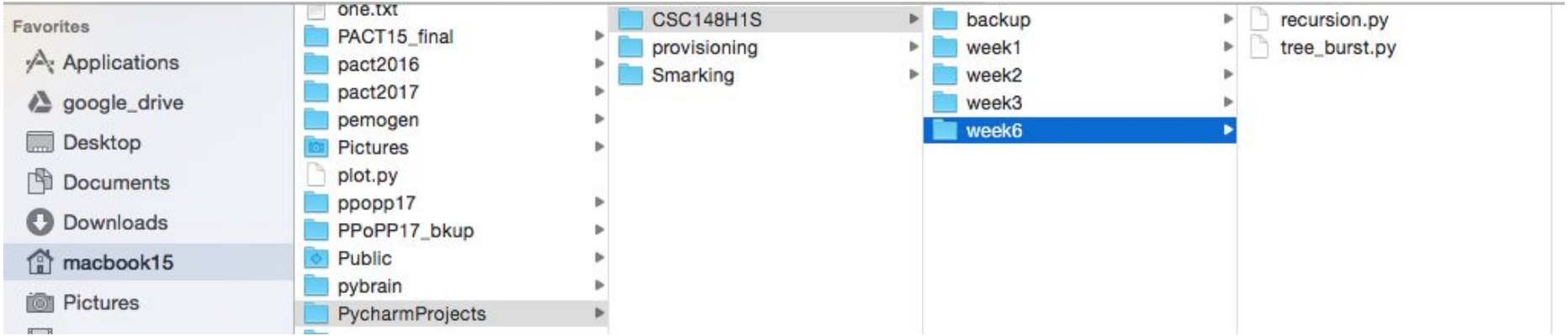




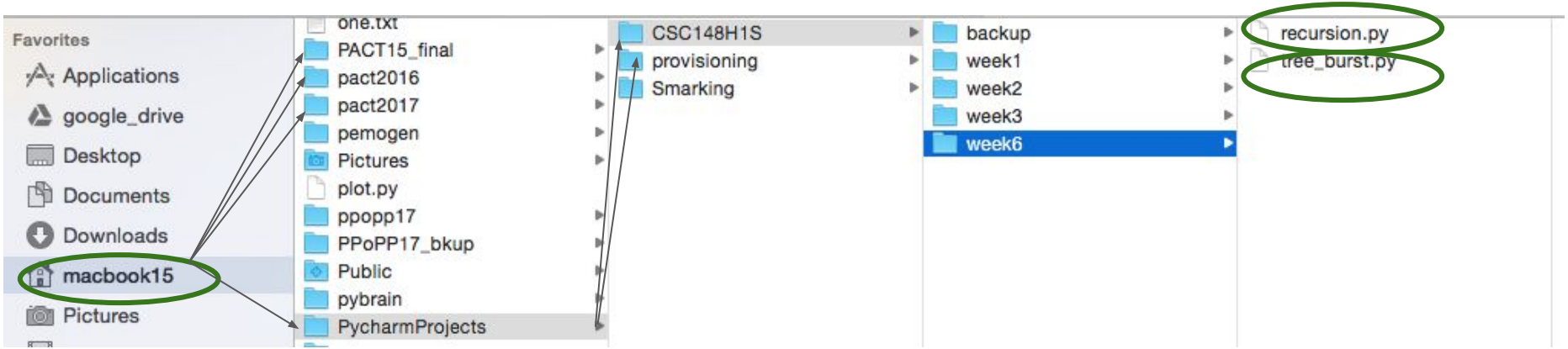
Patriarchal Trees



Use case of trees in Computer Science



Use case of trees in Computer Science



Root

leaves

Trees in CSC148

We will follow a programmatic approach

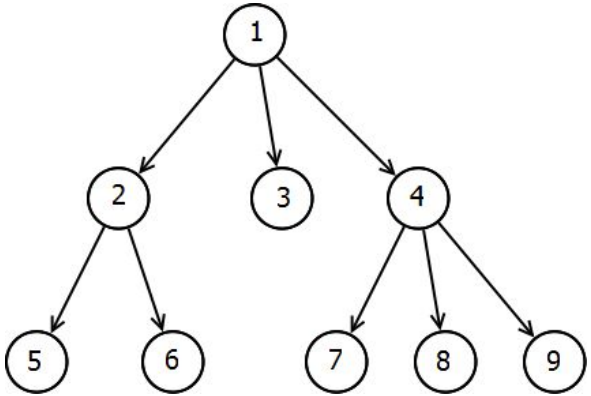
CSC165 follows an analytical approach

But we are talking about the same ADT

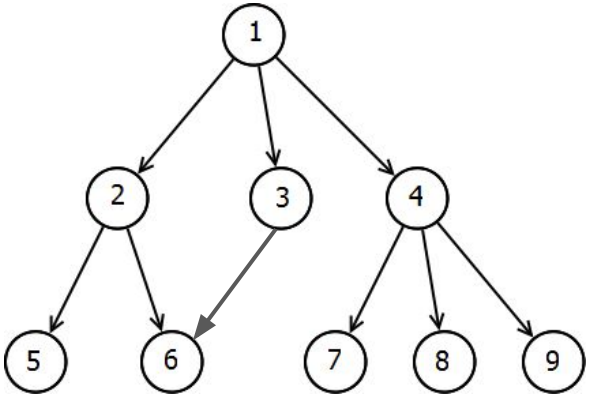
Tree terminology

- Set of nodes (possibly with values or labels), with directed edges between some pairs of nodes
- One node is distinguished as root
- Each non-root node has exactly one parent
- Each node has zero or more children
- A path is a sequence of nodes $n_1, n_2, \ldots, n_k$, where there is an edge from n_i to n_{i+1}, $i<k$
- The length of a path is the number of edges in it
- There is a unique path from the root to each node. In the case of the root itself this is just n_1, if the root is node n_1
- There are no cycles; no paths that form loops.

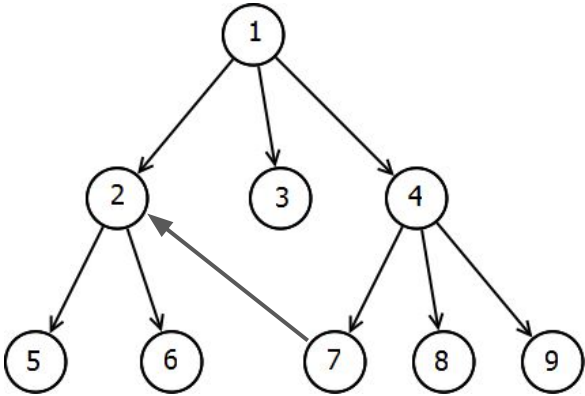
Task 1 : Identify the trees



1



2

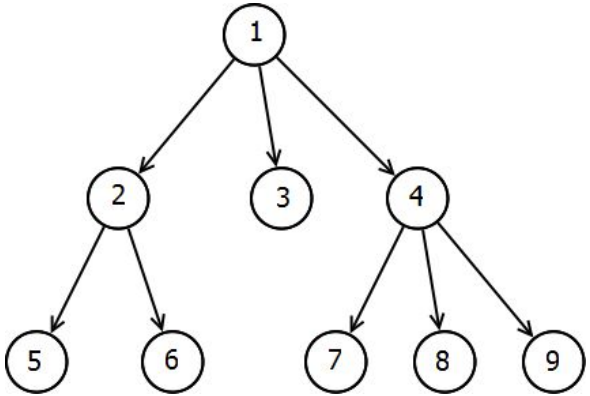


3

More tree terminology

- leaf: node with no children
- internal node: node with one or more children
- subtree: tree formed by any tree node together with its descendants and the edges leading to them.
- height: 1+ the maximum path length in a tree. A node also has a height, which is 1+ the maximum path length of the tree rooted at that node
- depth: length of the path from the root to a node, so the root itself has depth 0
- arity, branching factor: maximum number of children for any node

Test 2: Calculate Tree terms



Height of the tree (1+max path length)?

Height of subtree rooted at 2 ?

Depth of node 7?

Arity?

Creating a Tree

A tree has a number of nodes

- Each node has a **value**
- A node may/ may not have a **list** of children

General tree implementation

```
class Tree:
    """
    A bare-bones Tree ADT that identifies the root with the entire tree.
    """
    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
        self.children = children.copy() if children else []
```

NEVER have a mutable type as default value of function argument

Exercise 1: How many leaves?

```
def leaf_count(t):
    """
    Return the number of leaves in Tree t.

    @param Tree t: tree to count number of leaves of
    @rtype: int

    >>> t = Tree(7)
    >>> leaf_count(t)
    1
    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> leaf_count(tn1)
    5
    """
```

Tracing leaf_count

```
if t.children == []:
    return 1
else:
    return sum([leaf_count(c) for c in t.children])
```

Exercise 2: Height of a Tree

```
def height(t):
    """
    Return 1 + length of longest path of t.

    @param Tree t: tree to find height of
    @rtype: int

    >>> t = Tree(13)
    >>> height(t)
    1
    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> height(tn1)
    3
    """
```

Exercise 3: Arity of a tree

```
def arity(t):
    """
    Return the maximum branching factor (arity) of Tree t.

    @param Tree t: tree to find the arity of
    @rtype: int

    >>> t = Tree(23)
    >>> arity(t)
    0
    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> arity(tn1)
    3
    """
```