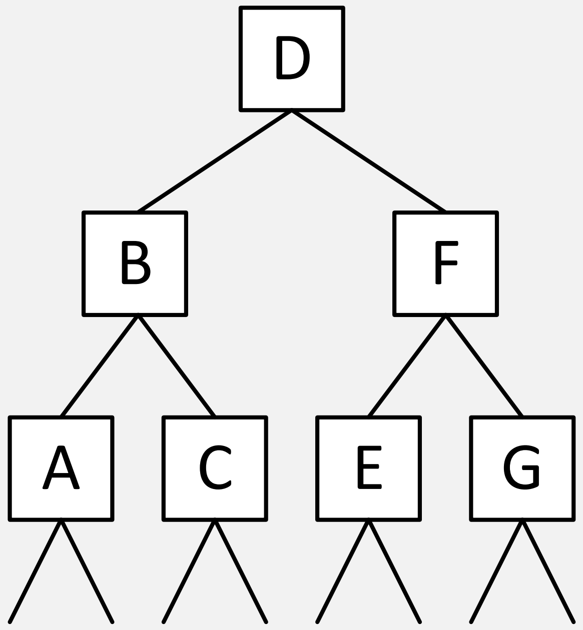
BINARY TREES: TEMPLATE EXAMPLES

Overview

Delroy A. Brinkerhoff

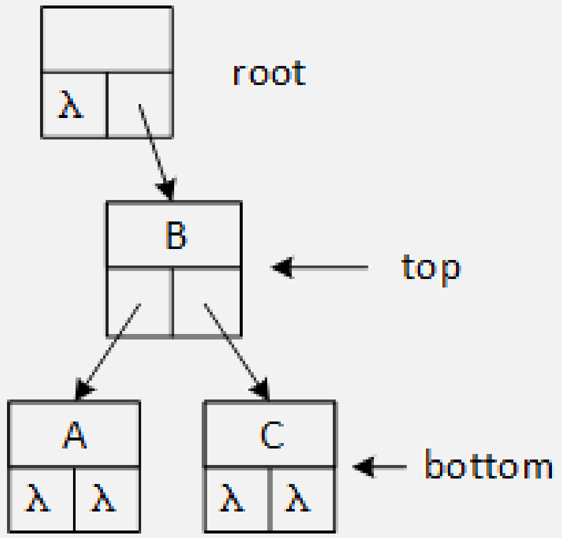
BINARY TREES



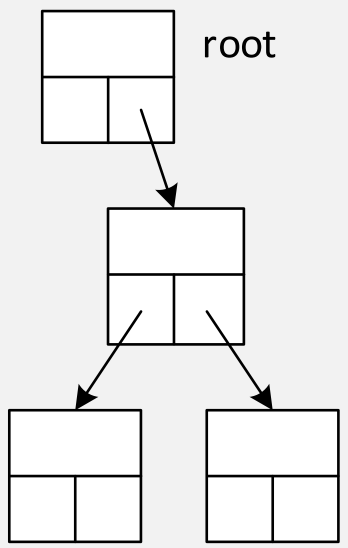
- Create
- Destroy
- Insert
- Search
- Remove

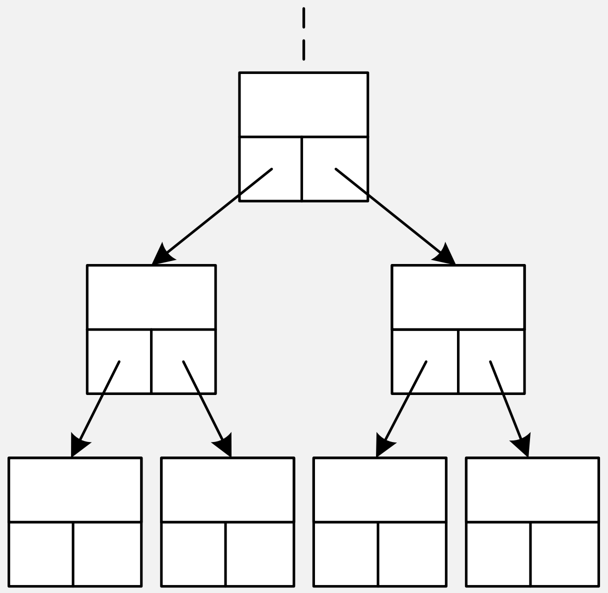
IMPLEMENTING BINARY TREES

```
template <class T>
class Tree
{
    private:
        T           data;
        Tree<T>*    left;
        Tree<T>*    right;
            . . .
};
```



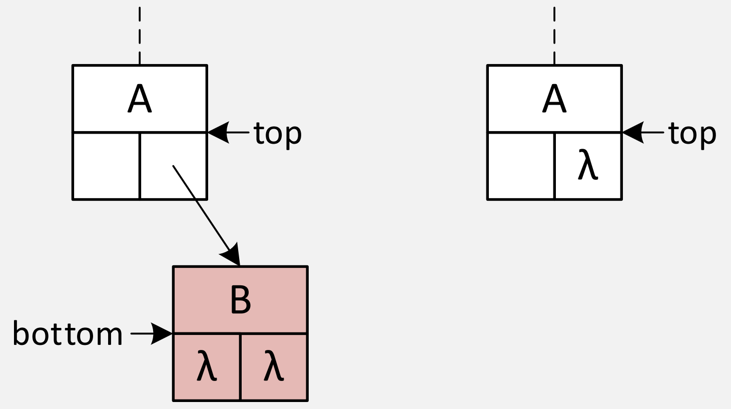
DESCENDING THE TREE





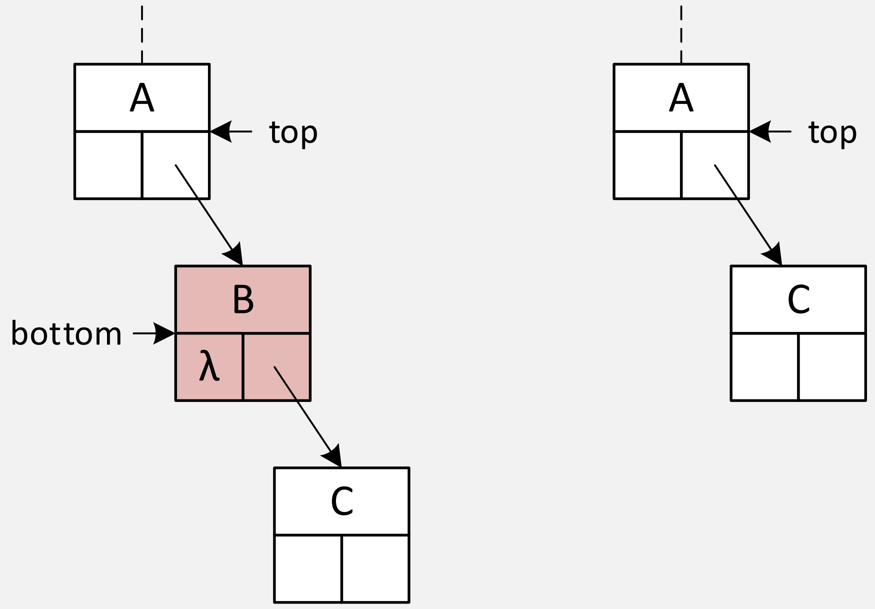
REMOVE (1)

- Slower than search or insert
- Three cases:
  - No subtrees (is a leaf)
  - One subtree
  - Two subtrees
- Case 1: No subtrees
  - Destroy the node
  - Set the appropriate top subtree pointer to null



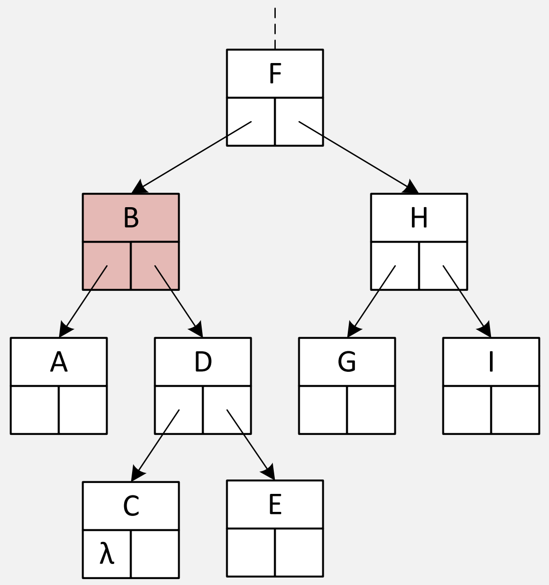
REMOVE (2)

- Case 2: One subtree
  - Set the appropriate top pointer to the bottom subtree
  - Destroy the node



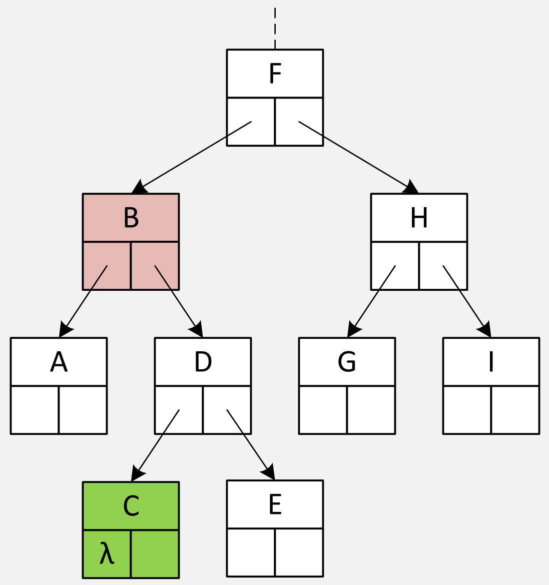
REMOVE (3)

- Case 3: Two subtrees – four phases
  - Find the removal node
  - Find the successor (the next node)
    - Go right
    - Go left until left is null
  - Copy the successor's data to the bottom
  - Destroy the successor (case 1 or 2)



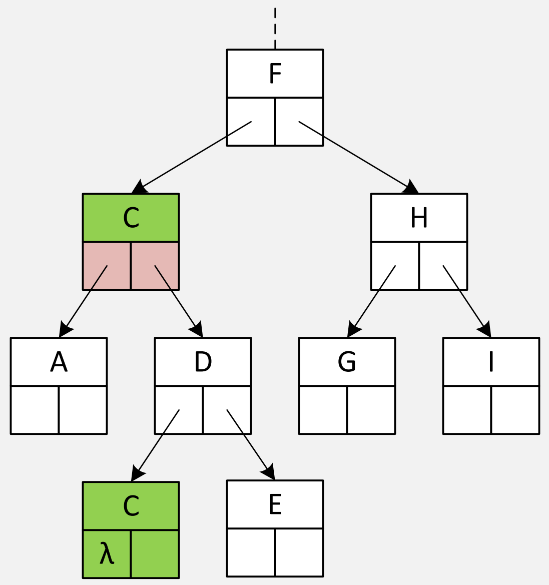
REMOVE (3)

- Case 3: Two subtrees
  - Find the removal node
  - Find the successor (the next node)
    - Go right
    - Go left until left is null
  - Copy the successor's data to the bottom
  - Destroy the successor (case 1 or 2)



REMOVE (3)

- Case 3: Two subtrees
  - Find the removal node
  - Find the successor (the next node)
    - Go right
    - Go left until left is null
  - Copy the successor's data to the bottom
  - Destroy the successor (case 1 or 2)



REMOVE (3)

- Case 3: Two subtrees
  - Find the removal node
  - Find the successor (the next node)
    - Go right
    - Go left until left is null
  - Copy the successor's data to the bottom
  - Destroy the successor (case 1 or 2)