§14.7-14.8: Binary Search Trees

28 Nov 2007
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review last time (§12.8-12.12)

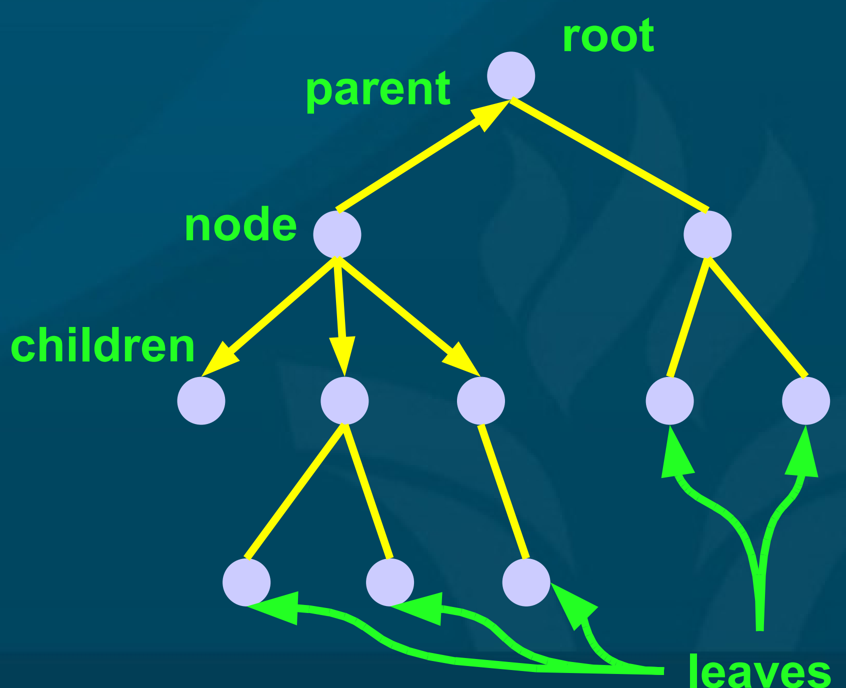
- Linked lists
  - Type definition, creating a new list
    - Inserting in nth position
    - Insert at head, append to tail
    - Deleting
  - Algorithmic efficiency
  - Circularly linked lists
  - Bidirectional lists

What's on for today

- Trees:
  - Definition of terms:
    - Parent, children, root, leaves, degree, depth, level, forest
  - Depth-first vs. breadth-first search
  - Binary trees: pre/in/post-order traversal
  - Binary search trees (BST):
    - Type definition
    - Search, Insert, Delete
    - Algorithmic efficiency of BST Search

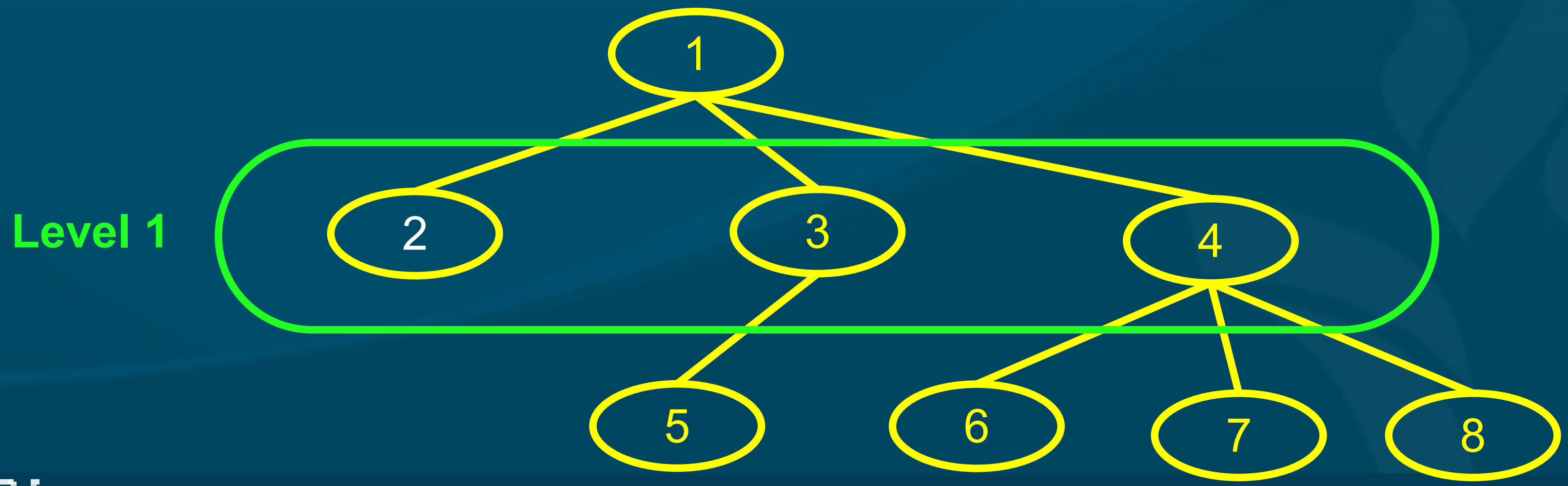
Trees

- Another kind of dynamic ADT is the tree:
  - Root: starting node (one per tree)
    - Could also have a forest of several trees
  - Each node has at most one parent, and zero or more children
  - Leaves: no children
  - Depth: length of longest path from root
  - Degree: max # of children per node



Searching trees

- A depth-first search of a tree pursues each path down to a leaf, then backtracks to the next path
  - 1-2 1-3-5 1-4-6 4-7 4-8
- A breadth-first search finishes each level before moving on to the next:
  - 1 2-3-4 5-6-7-8



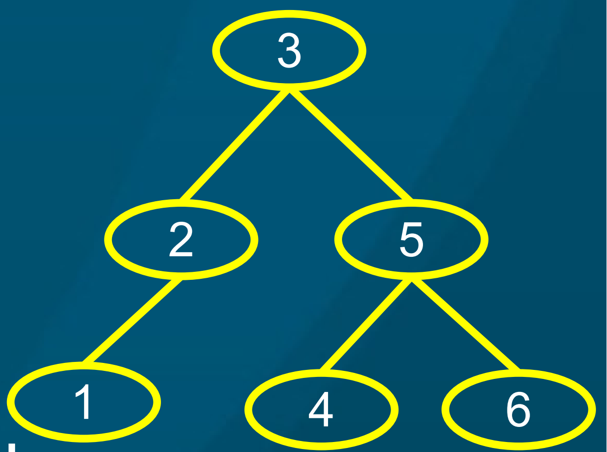
Binary search trees

- Binary trees (degree=2) are handy for keeping things in sorted order:


```
class BST:
    def __init__(self, data=None):
        self.data = data
        self.left = None
        self.right = None
            (* could also have a parent ptr *)
root = BST( 'Braeburn' )
root.left = BST( 'Ambrosia' )
root.right = BST( 'Gala' )
root.right.left = BST( 'Fuji' )
```

- Everything in left subtree is smaller
- Everything in right subtree is bigger

Binary tree traversals

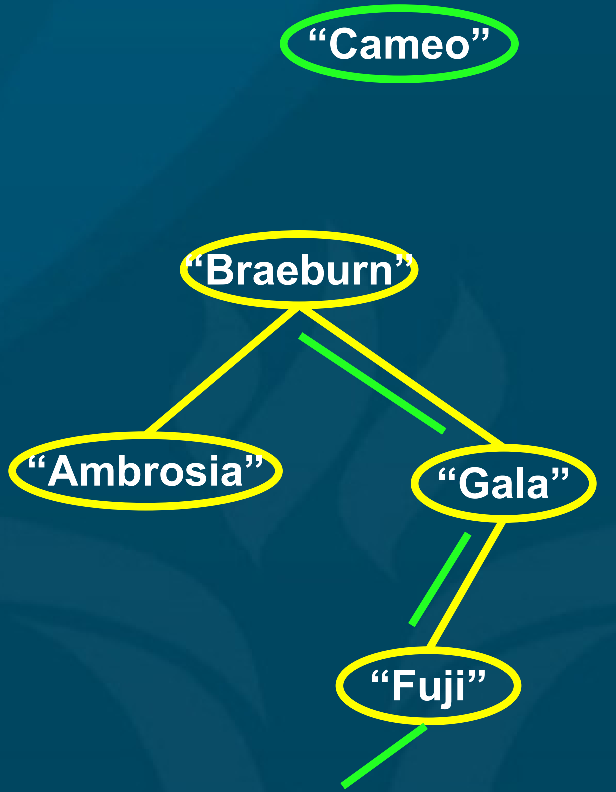


- Pre-order traversal of binary tree:
  - Do self first, then left child, then right
    - 3 – 2 – 1 – 5 – 4 - 6
- In-order traversal:
  - Do left child, then self, then right child
    - 1 – 2 – 3 – 4 – 5 – 6 (sorted order in BST)
    - e.g. expressions: "12 + (2 * 5)"
- Post-order traversal:
  - Do both children first before self
    - 1 – 2 – 4 – 6 – 5 - 3
    - e.g. Reverse Polish Notation: 12, 2, 5, *, +

Searching a BST

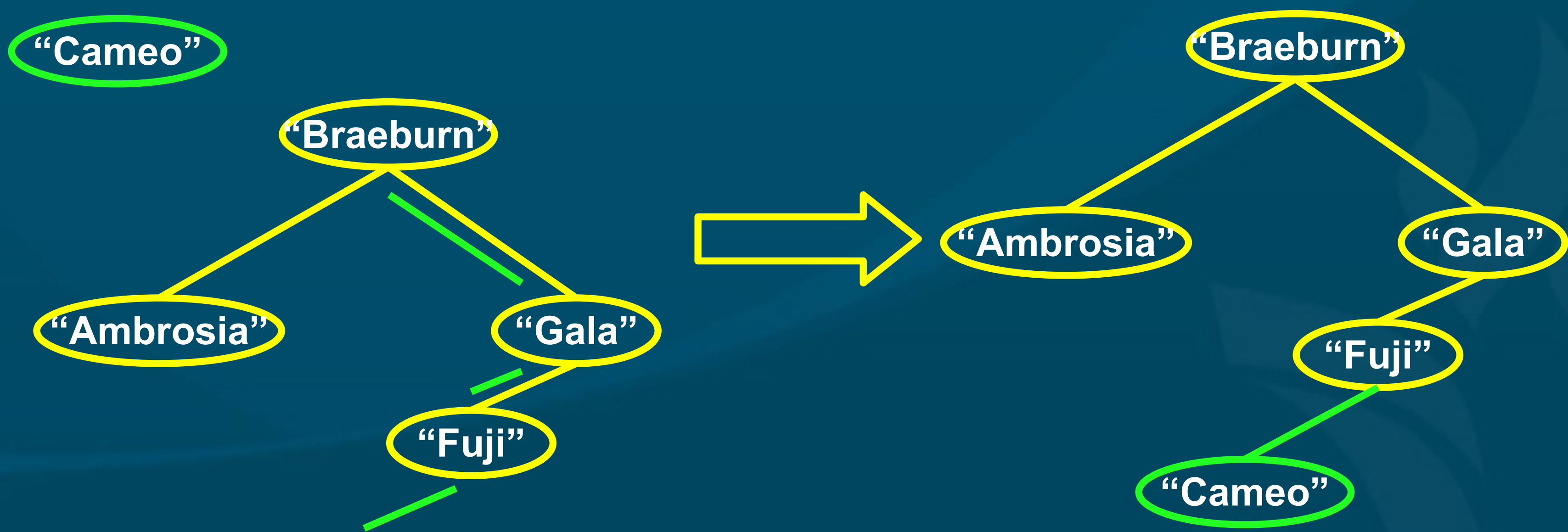
- Recursive algorithm:

```
def search (self, key):
    if key == self.data:
        return self
    elif key < self.data and self.left != None:
        return self.left.search(key)
    elif key > self.data and self.right != None:
        return self.right.search(key)
    else:
        return None
```



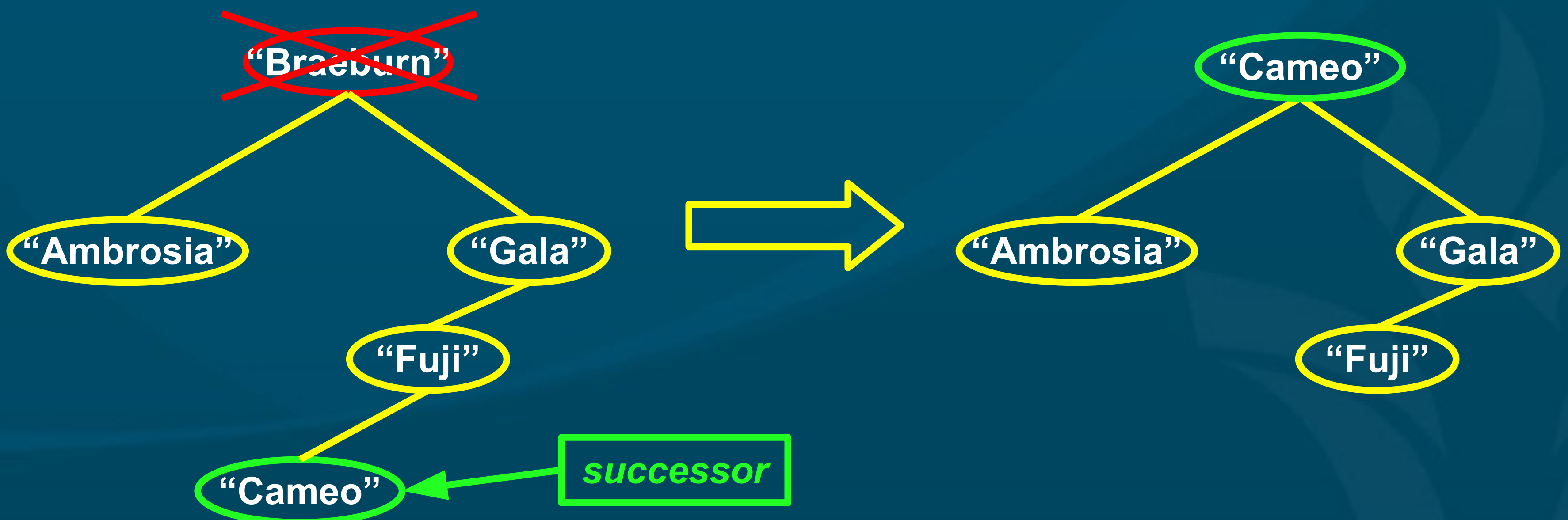
Inserting into a BST

- Keep it sorted: insert in a proper place
- One choice: always insert as a leaf
  - Use search() algorithm to hunt for where the node ought to be if it were already in the tree



Deleting from a BST

- Need to maintain sorted structure of BST
- Replace node with predecessor or successor leaf
  - Predecessor: largest node in left subtree
  - Successor: smallest node in right subtree



BSTs and algorithmic efficiency

- Searching in a balanced binary search tree takes worst-case O(log n) running time:
  - Depth of balanced tree is $\log_2 n$
  - Compare with arrays/linked lists: O(n)
- But depending on order of inserts, tree may be unbalanced:
  - Insert in order: Ambrosia, Braeburn, Fuji, Gala:
  - Tree degenerates to linked-list
  - Searching becomes O(n)

- Keeping a BST balanced is a larger topic
  - e.g., Splay-trees

Review of today

- Trees:
  - Definition of terms:
    - Parent, children, root, leaves, degree, depth, level, forest
  - Depth-first vs. breadth-first search
  - Binary trees: pre/in/post-order traversal
  - Binary search trees (BST):
    - Type definition
    - Search, Insert, Delete
    - Algorithmic efficiency of BST Search

TODO

- Lab09 due tonight:
  - Knight's tour
- HW10 due Fri:
  - delete() for doubly-linked list
- Paper due next Mon 3Dec
- Lab10 due next Wed 5Dec:
  - Implement one of your old Lab04-07 in M2
  - Full lab-writeup (may reuse parts of old writeup)