CSCI 136
Data Structures & Advanced Programming

Binary Search Trees Ia

Binary Search Trees I

Ordered Structures

Recall: Ordering the items in lists can improve search performance

OrderedVector

- Rank (find k^{th} smallest) takes O(1) time
- Search take O(log n) time

OrderedList

- Rank (find k^{th} smallest) takes O(k) time
- Search takes O(n) time
  - But faster than on unordered list

Ordered Structures

However: Adding and removing elements from ordered structures can require θ(n) time

OrderedVector

- Find location for new item in O(log n) time
- Insert new item can take θ(n) time

OrderedList

- Find location for new item can take θ(n) time
- Insert new item takes O(1) time

Ordered Structures

Conclusion: Updating an ordered vector or list can require θ(n) time

Can we do better?

Yes!

Store items in a binary tree

- As long as it's carefully constructed and maintained we can improve update times
- Let's explore this further….

Binary Trees and Orders

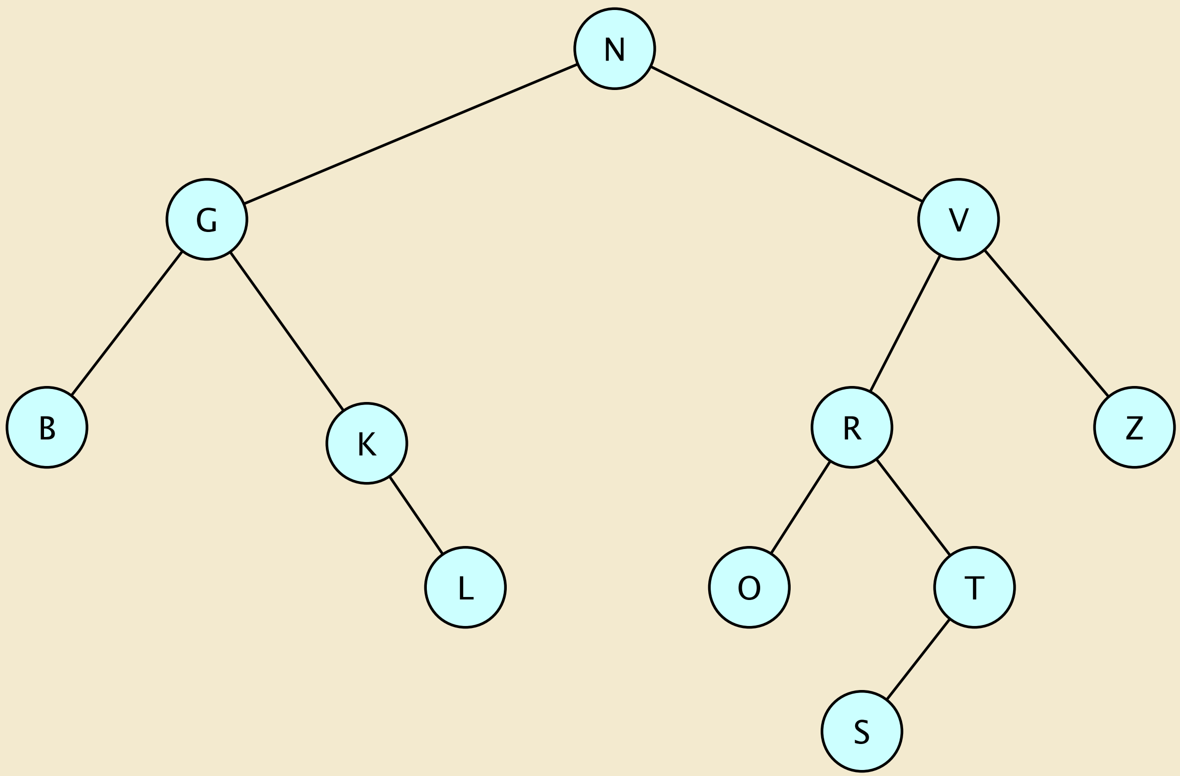
- Binary trees impose multiple orderings on their elements (pre-/in-/post-/level-orders)
- In particular, in-order traversal suggests a natural way to hold (comparable) items
  - For each node v in tree
    - All values in left subtree of v are $\leq$ v
    - All values in right subtree of v are $>$ v
- This leads us to...

Binary Search Trees

- Binary search trees maintain a *total* ordering among elements
- Definition: A BST T is either:
  - Empty
  - Has root r with subtrees T_L and T_R such that
    - All nodes in T_L have smaller value than r (or are empty)
    - All nodes in T_R have larger value than r (or are empty)
    - T_L and T_R are also BSTs
- Examples….

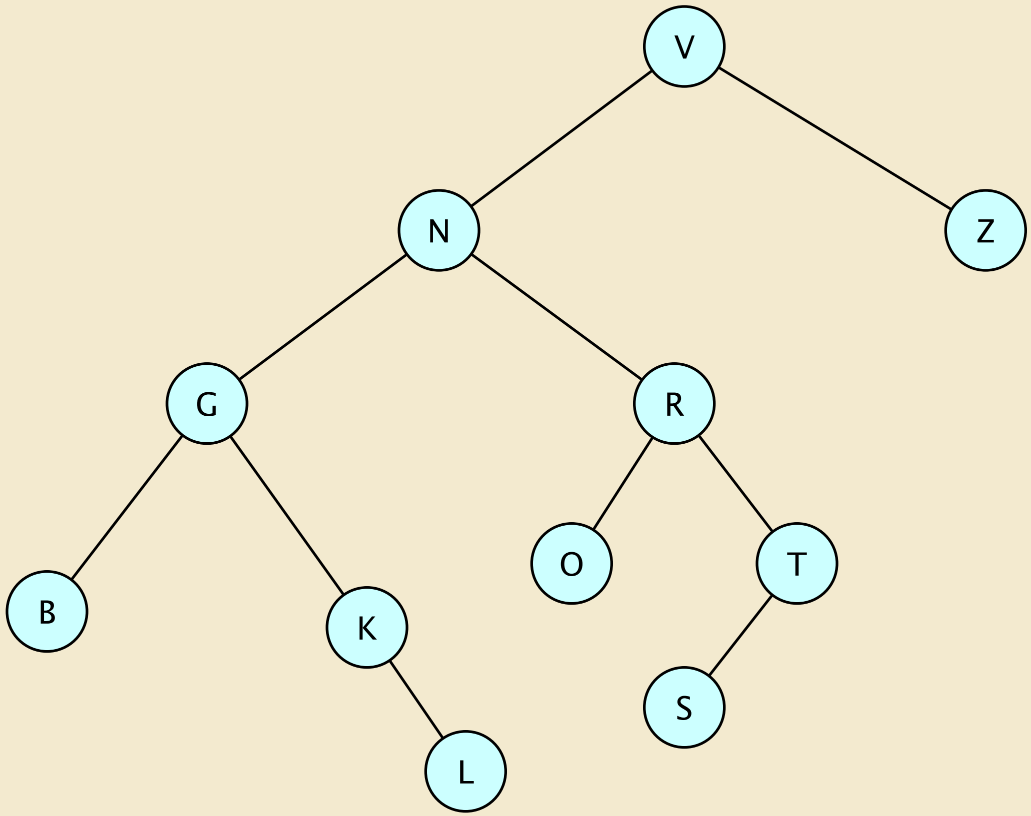
A Binary Search Tree

B G K L N O R S T V Z



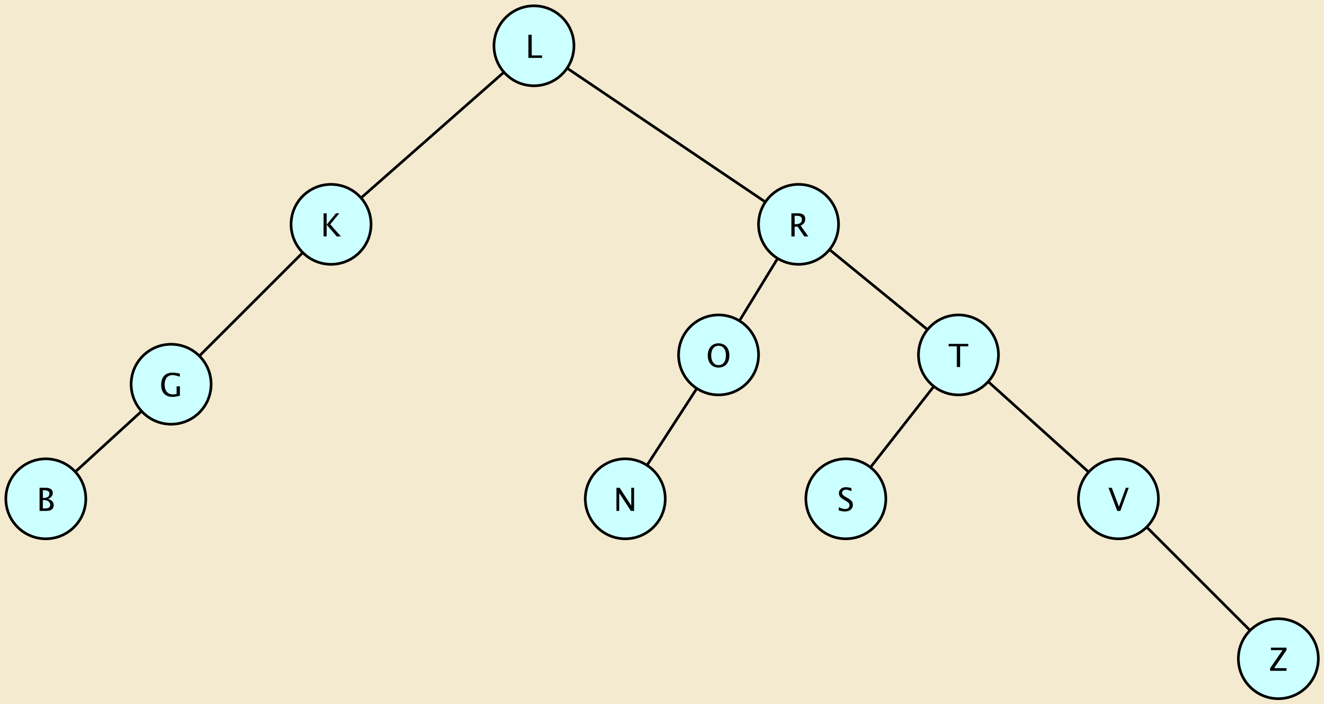
A Binary Search Tree

B G K L N O R S T V Z



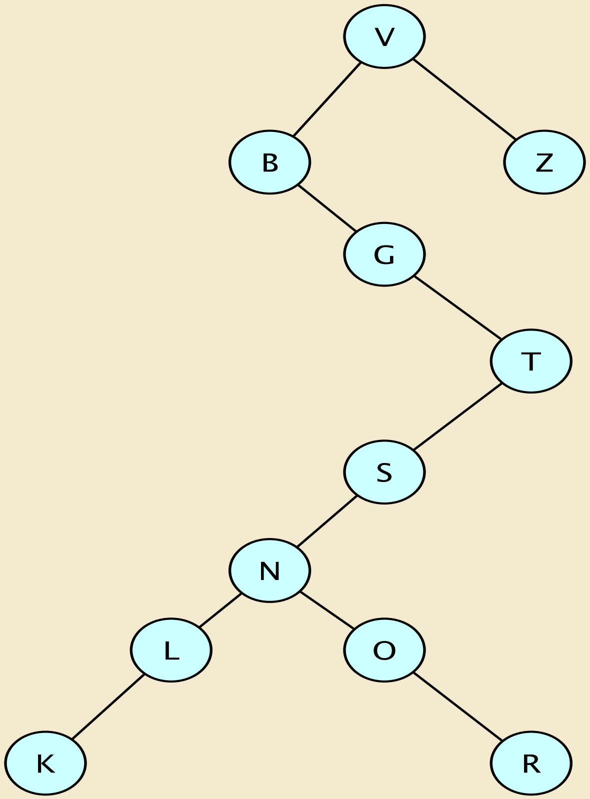
A Binary Search Tree

B G K L N O R S T V Z



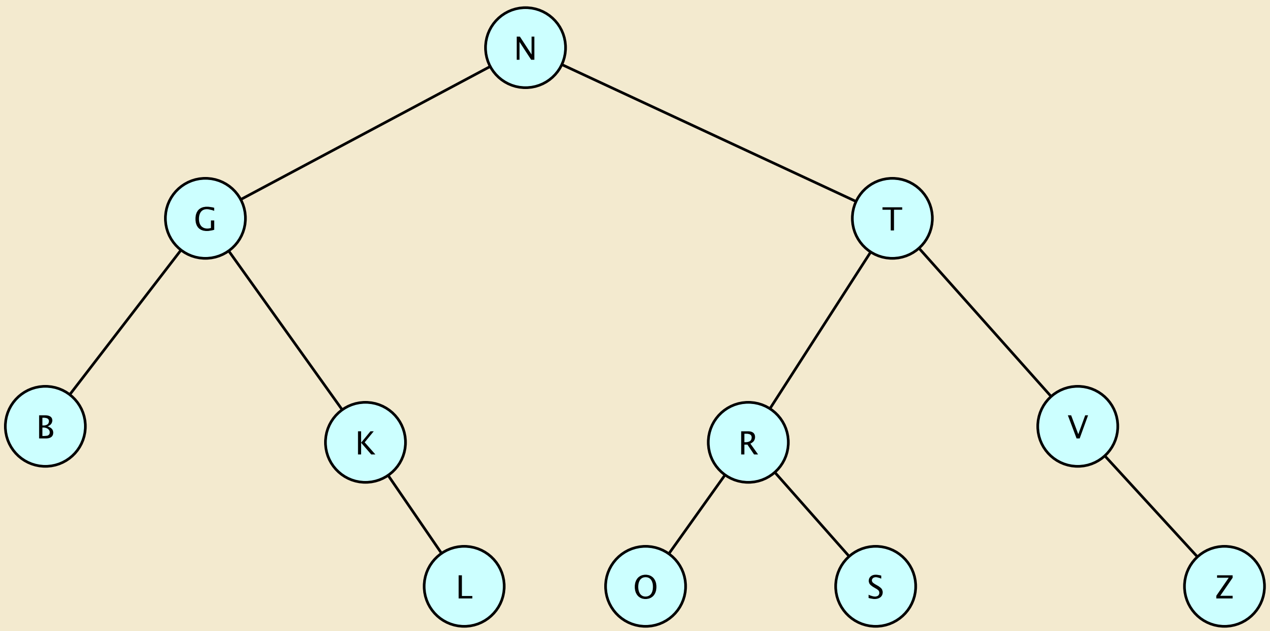
A Binary Search Tree

B G K L N O R S T V Z



A Binary Search Tree

B G K L N O R S T V Z



BST Observations

- The same data can be represented by many BST shapes
- Searching for a value in a BST takes time proportional to the height of the tree
  - Reminder: trees have height, nodes have depth
- Additions to a BST happen at nodes missing at least one child (*a constraint!*)
- Removing from a BST can involve *any* node

BST Operations

- BSTs will implement the OrderedStructure Interface
  - We'll focus on these methods
    - `add(E item)`
    - `contains(E item)`
    - `get(E item)`
    - `remove(E item)`
    - `iterator()`
      - This will provide an in-order traversal
  - Also supports: size(), isEmpty(), clear(),

BST Implementation

- A BST will store a few items, including
  - A root node of type BinaryTree<E>
  - The number of nodes in the tree
  - A comparator for imposing the order
    - If not provided, a NaturalComparator<E> will be used
- Helper methods (protected) include
  - locate(BinaryTree<E> node, E value)
    - Find node in subtree having node as root
      - Or return a location where node could be added
  - predecessor(BinaryTree<E> node)
    - Find node in tree immediately preceding node in ordering
    - Also a successor() method

BST Operations

- Runtime of add, contains, get, remove will depend on runtimes of locate and predecessor methods
- Runtime of locate and predecessor will be : O(height)
- Strategy: Keep the height small
    - BinarySearchTree class doesn’t attempt this…
    - But other implementations we explore will, including
        - AVL trees
        - RedBlackSearchTree
        - SplayTree
- In fact, we'll see that AVL trees and RedBlack trees maintain a height of O(log n)
    - So contains/add/remove/get all take O(log n) time!

Sample Applications

- We can use a BST to create a *dictionary*
  - Each node holds a ComparableAssociation
    - Nodes are compared using keys
    - Two objects are equal if keys are equal
- We can sort using a BST
  - Given any set of comparable items, insert them into a BST one by one.
    - Insert time is O(h), where h is current height of tree
    - If h_i it the height before inserting i^{th} item, then total insert time is $O(h_1 + \ldots + h_n)$
    - If h is the maximum of these heights, total time is $O(h \cdot n)$

Binary Search Tree Implementation