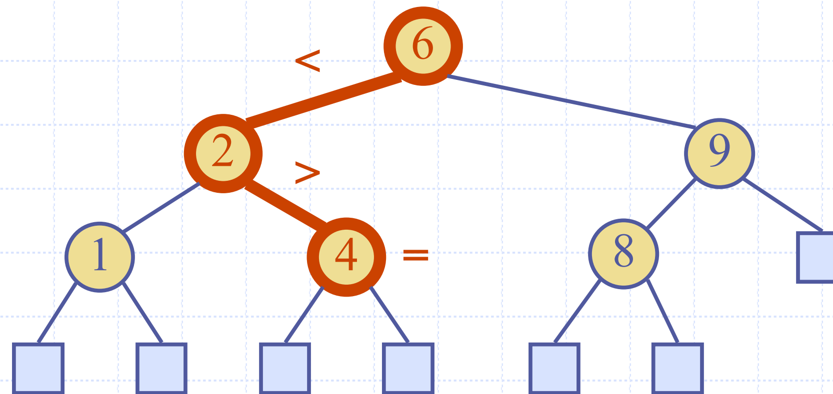
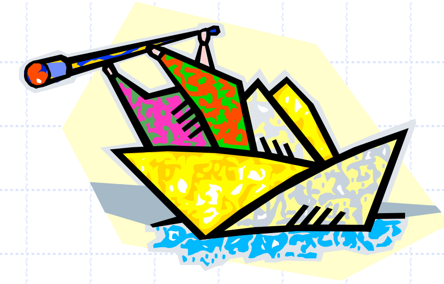
Binary Search Trees

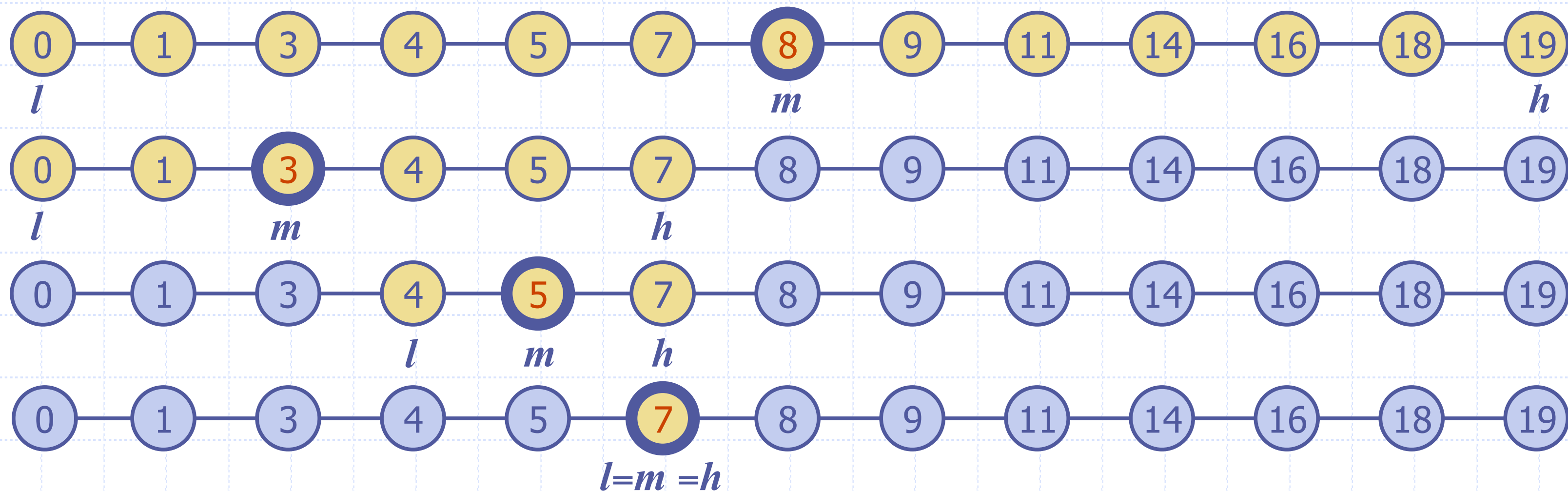


© 2004 Goodrich, Tamassia

Binary Search (§ 8.3.3)

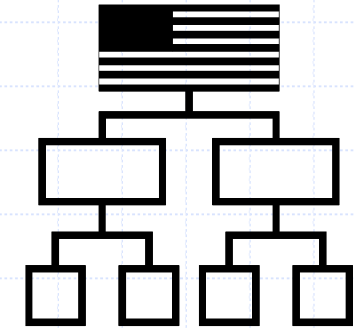


- Binary search can perform operation find(k) on a dictionary implemented by means of an array-based sequence, sorted by key
  - similar to the high-low game
  - at each step, the number of candidate items is halved
  - terminates after O(log n) steps
- Example: find(7)

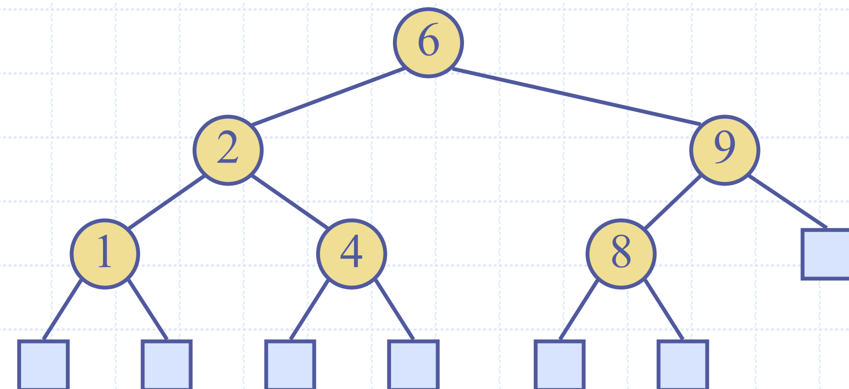


© 2004 Goodrich, Tamassia

Binary Search Trees (§ 9.1)



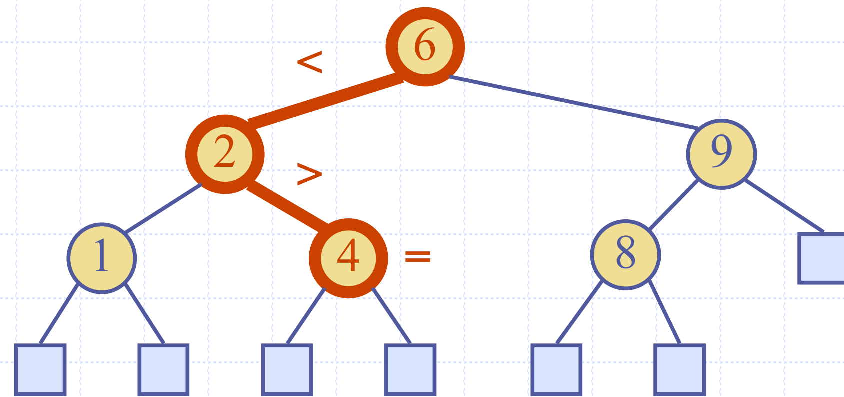
- A binary search tree is a binary tree storing keys (or key-value entries) at its internal nodes and satisfying the following property:
  - Let u, v, and w be three nodes such that u is in the left subtree of v and w is in the right subtree of v. We have $key(u) \leq key(v) \leq key(w)$
- External nodes do not store items
- An inorder traversal of a binary search trees visits the keys in increasing order



Search (§ 9.1.1)

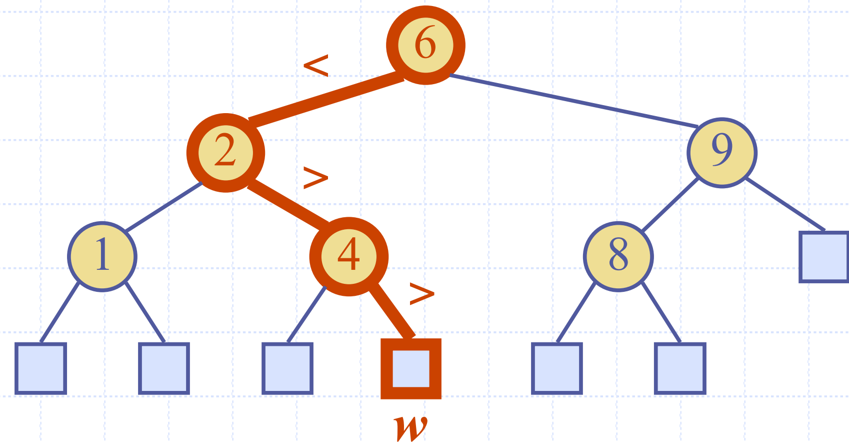
- To search for a key ***k***, we trace a downward path starting at the root
- The next node visited depends on the outcome of the comparison of ***k*** with the key of the current node
- If we reach a leaf, the key is not found and we return null
- Example: find(4):
  - Call TreeSearch(4,root)

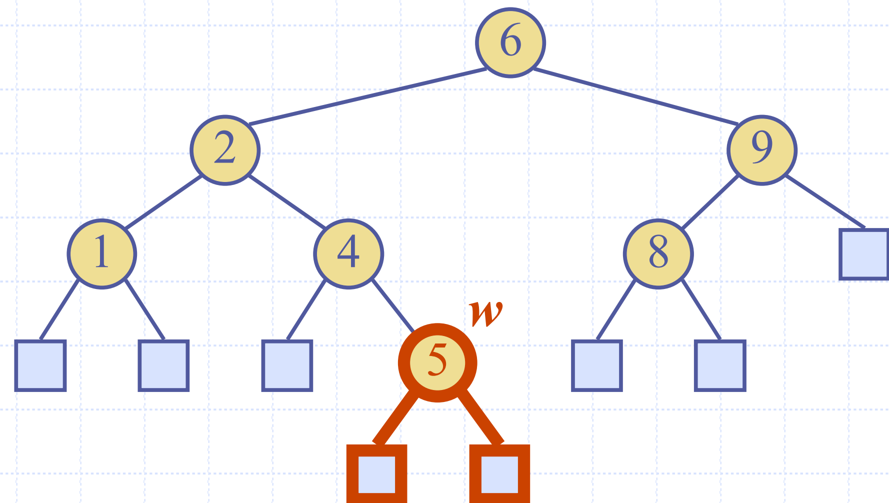
```
Algorithm TreeSearch(k)
    TreeNode v = root;
    while v.isInternal() ∧ v.key != k
        if k < v.key
            v = v.left
        else
            v = v.right
    return v
```



Insertion

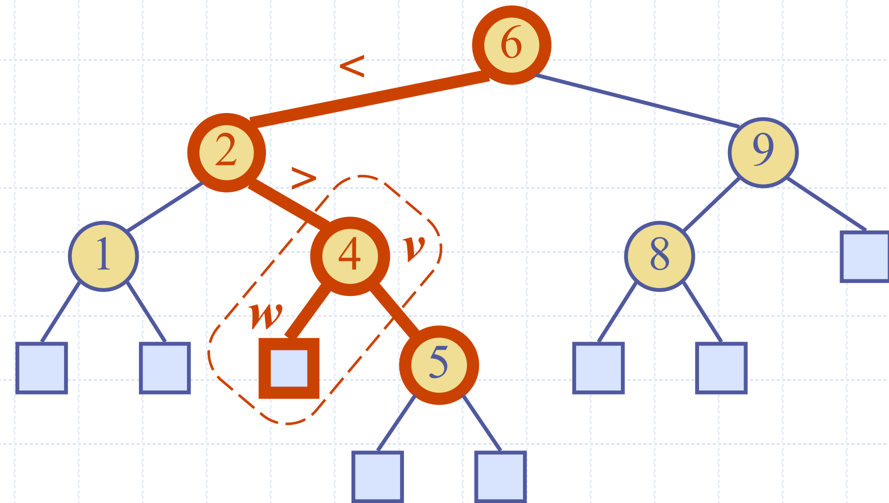
- To perform operation insert(k, o), we search for key k (using TreeSearch)
- Assume k is not already in the tree, and let w be the leaf reached by the search
- We insert k at node w and expand w into an internal node
- Example: insert 5

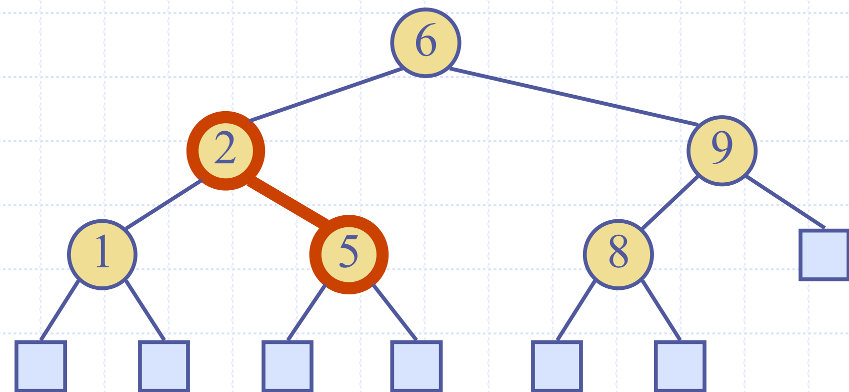




Deletion

- To perform operation remove(k), we search for key k
- Assume key k is in the tree, and let let v be the node storing k
- If node v has one child w, we remove v and w from the tree with operation removeExternal(w), which removes w and its parent
- Example: remove 4

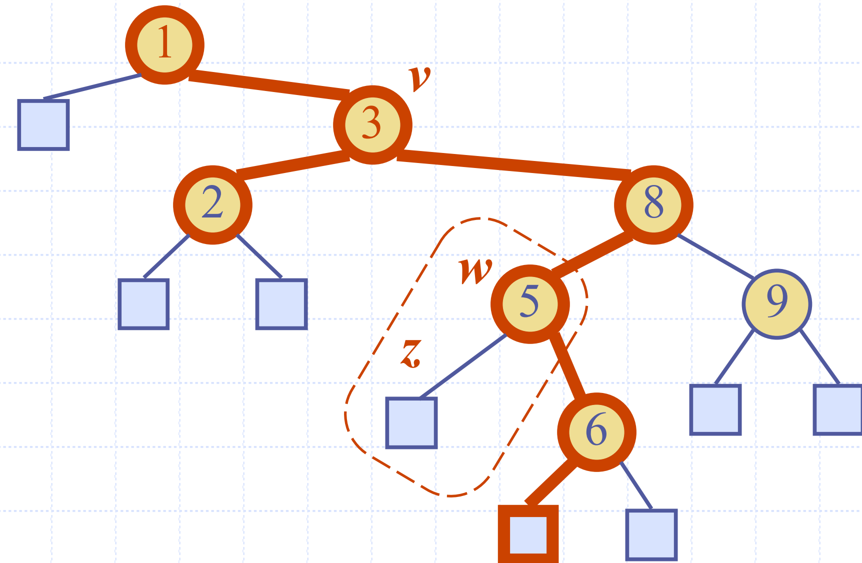


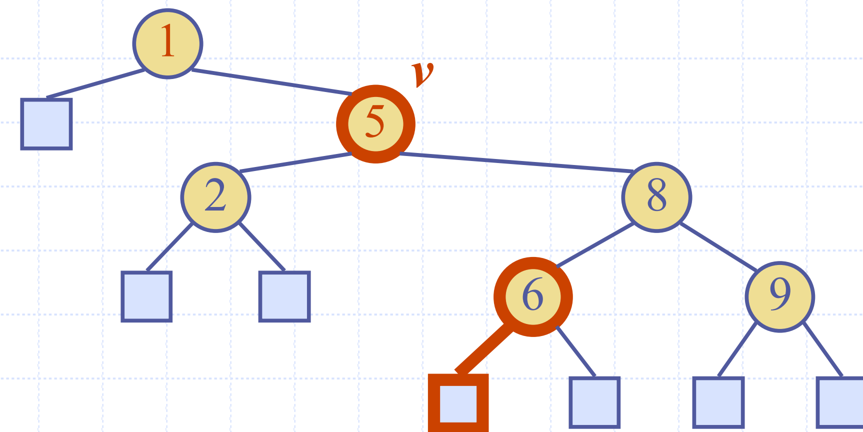


© 2004 Goodrich, Tamassia

Deletion (cont.)

- We consider the case where the key k to be removed is stored at a node v whose children are both internal
  - we find the internal node w that follows v in an inorder traversal
  - we copy $key(w)$ into node v
  - we remove node w and its left child z (which must be a leaf) by means of operation removeExternal(z)
- Example: remove 3





Performance

- Consider a dictionary with n items implemented by means of a binary search tree of height h
  - the space used is $O(n)$
  - methods find, insert and remove take $O(h)$ time
- The height h is $O(n)$ in the worst case and $O(\log n)$ in the best case