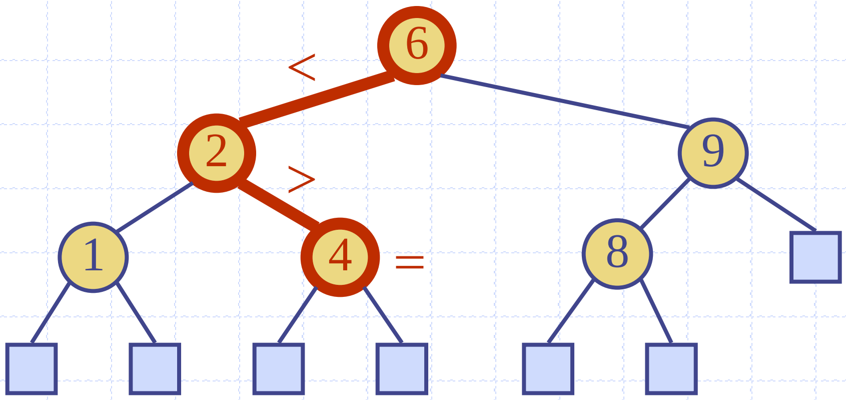
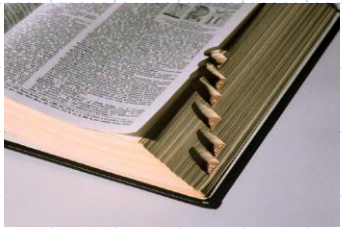
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Binary Search Trees



Ordered Maps

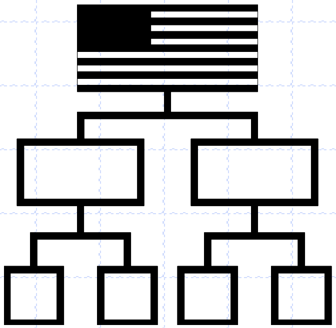


- Keys are assumed to come from a total order.
- Items are stored in order by their keys
- This allows us to support nearest neighbor queries:
  - Item with largest key less than or equal to k
  - Item with smallest key greater than or equal to k

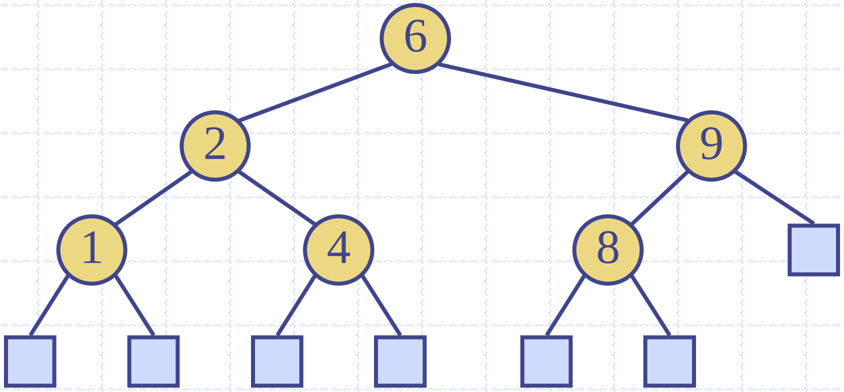
Ordered/Sorted Maps

- Sorted Array
  - Insertion and deletion could move a lot of entries
- Skip Lists
  - Keys could be duplicated in multiple levels
  - Expected O(log n) time for search/get
    - Not worst-case O(log n) time

Binary Search Trees



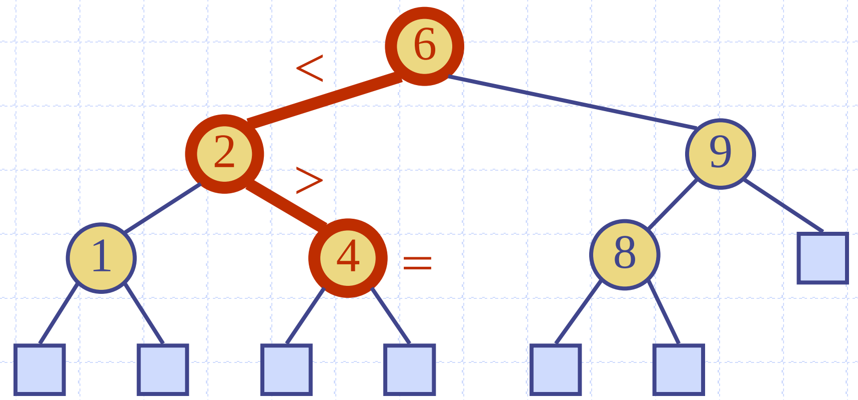
- A binary search tree is a binary tree storing keys (or key-value entries) at its internal nodes:
  - Let u, v, and w be three nodes such that
  - u is in the left subtree of v and
  - w is in the right subtree of v.
  - We have $key(u) \leq key(v) \leq key(w)$
- External nodes do not store items
-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



Search

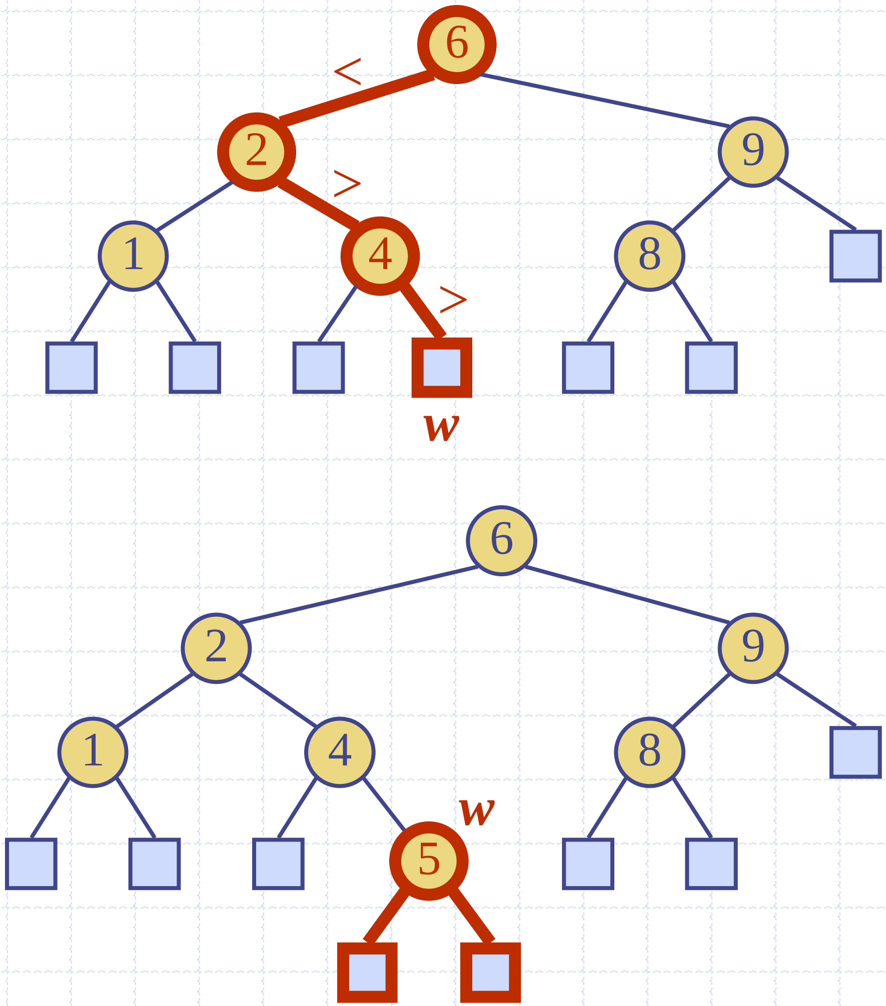
- To search for a key k, we trace a downward path starting at the root
- The next node visited depends on the comparison of k with the key of the current node
- If we reach a leaf, the key is not found
- Example: get(4):
  - Call TreeSearch(4,root)
- The algorithms for nearest neighbor queries are similar

```
Algorithm TreeSearch(k, v)
    if T.isExternal (v)
        return v
    if k < key(v)
        return TreeSearch(k, left(v))
    else if k = key(v)
        return v
    else { k > key(v) }
        return TreeSearch(k, right(v))
```



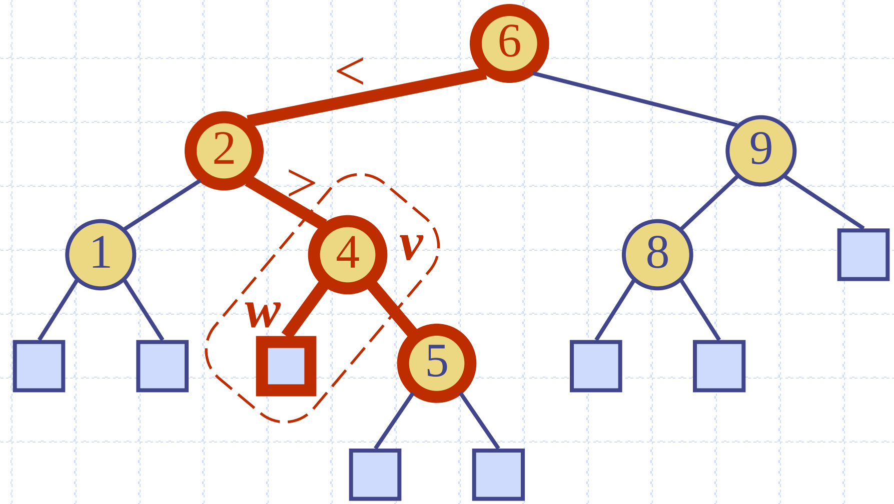
Insertion

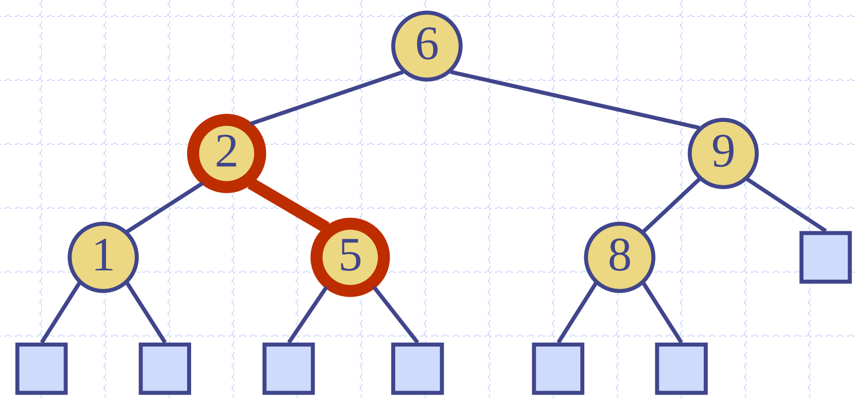
- To perform operation put(k, o), we search for key k (using TreeSearch)
- Assume k is not already in the tree, and let w be the leaf reached by the search
- We insert k at node w and expand w into an internal node
- Example: insert 5



Deletion (one child is a leaf)

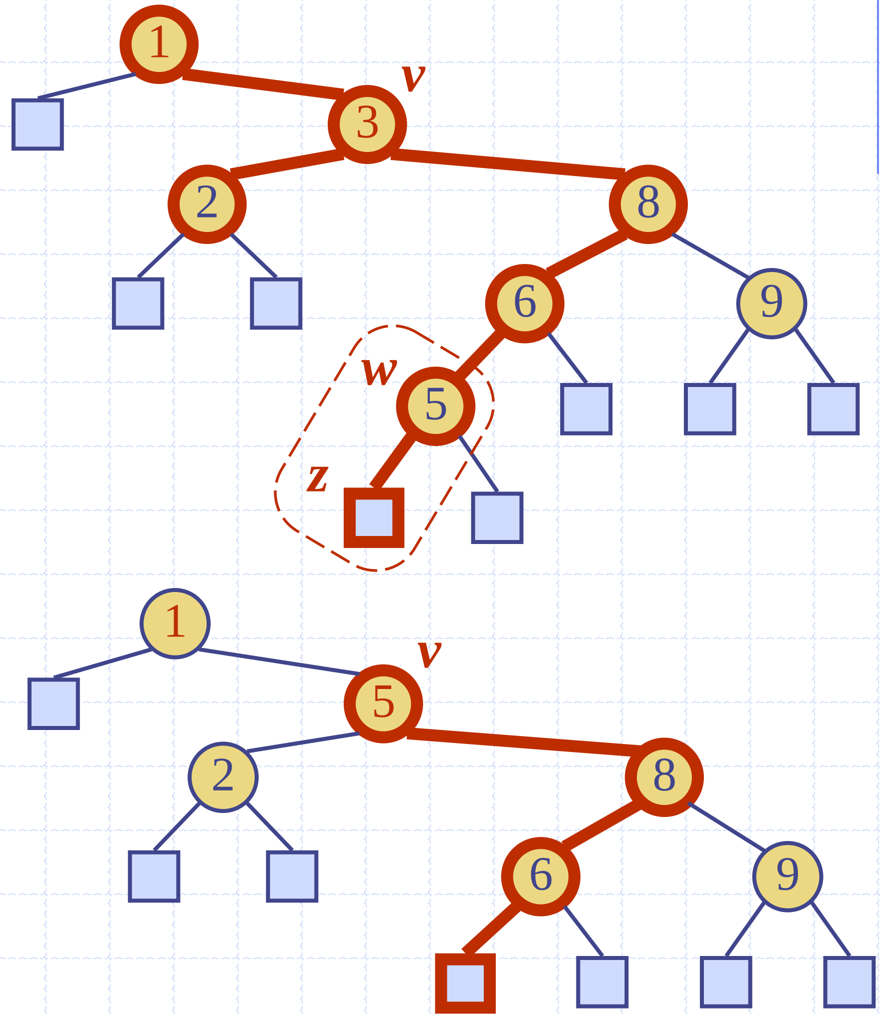
- To perform operation remove(k), we search for key k
- Assume key k is in the tree, and let let v be the node storing k
- If node v has a leaf child w, we remove v and w from the tree with operation removeExternal(w), which removes w and its parent
- Example: remove 4





Deletion (both children are not leaves)

- find the internal node w that follows v in an inorder traversal
- copy $key(w)$ into node v
- remove node w and its left child z (which must be a leaf) by means of operation removeExternal(z)
- Example: remove 3

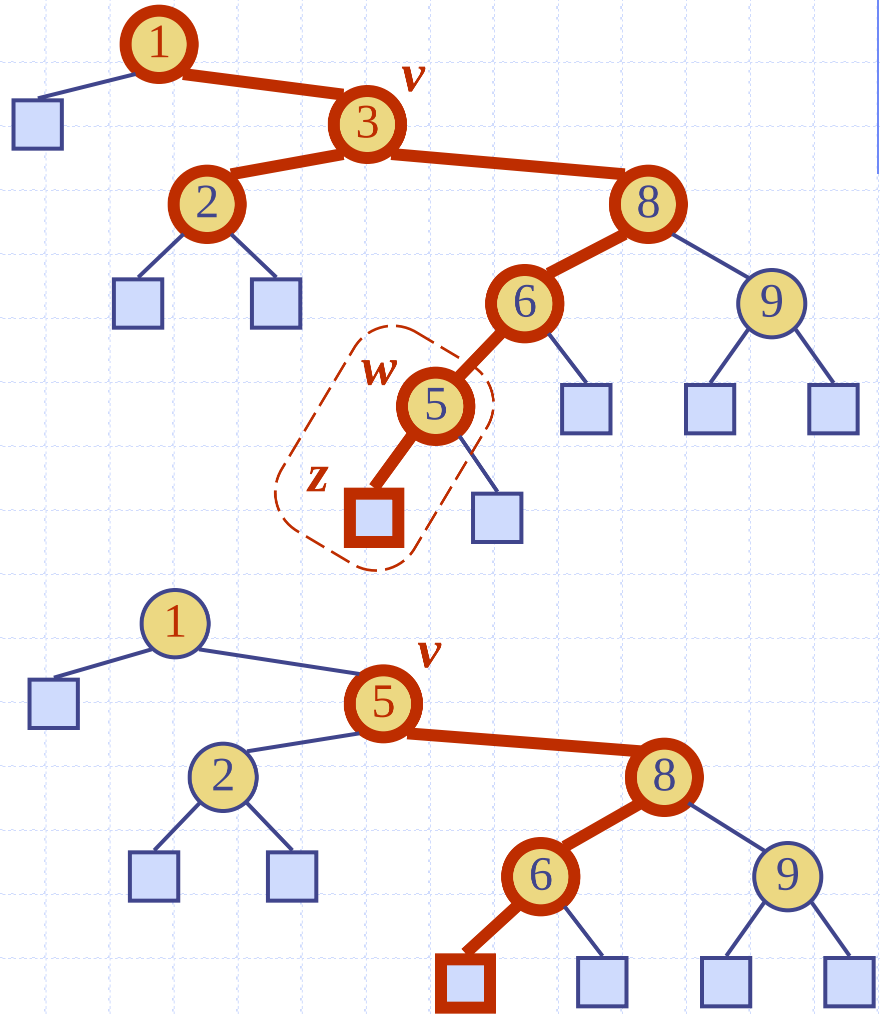


Deletion (both children are not leaves)

- find the internal node w that follows v in an inorder traversal

Why?

- copy $key(w)$ into node v
- remove node w and its left child z (which must be a leaf) by means of operation removeExternal(z)
- Example: remove 3



Performance

- Consider an ordered map with n items
- BST of height h
  - the space used is $O(n)$
  - methods get, put and remove take $O(h)$ time
- The height h is $O(n)$ in the worst case and $O(\log n)$ in the best case