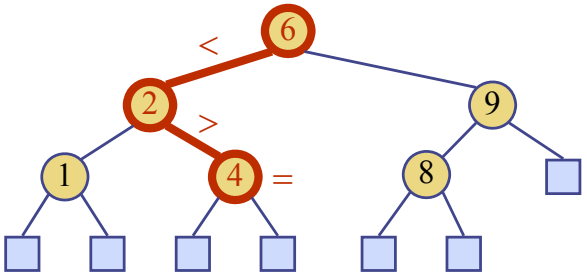
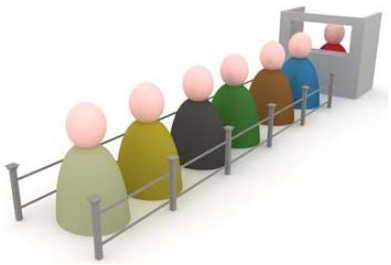
Lecture 8-1: Binary Search Trees



Sunghyun Cho
Professor
Division of Computer Science
chopro@hanyang.ac.kr

HANYANG UNIVERSITY

Keywords



Queue

Priority Queue

MAP and Dictionary

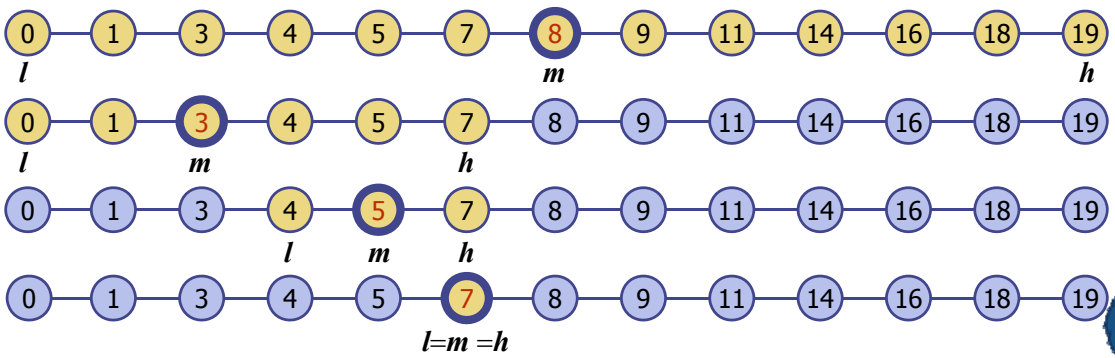
Think about the purpose of each data structure !

Ordered Maps

- Keys are assumed to come from a total order.
- New operations:
  - firstEntry(): entry with smallest key value null
  - lastEntry(): entry with largest key value
  - floorEntry(k):entry with largest key ≤ k
  - ceilingEntry(k): entry with smallest key ≥ k
  - These operations return null if the map is empty

Binary Search

- Binary search can perform operations get, floorEntry and ceilingEntry on an ordered map 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)



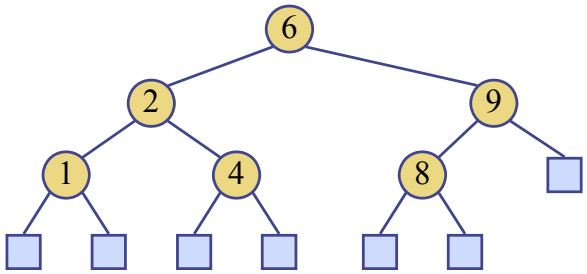
Search Tables (skip)

- A search table is an ordered map implemented by means of a sorted sequence
  - We store the items in an array-based sequence, sorted by key
  - We use an external comparator for the keys
- Performance:
  - get, floorEntry and ceilingEntry take $O(\log n)$ time, using binary search
  - put takes $O(n)$ time since in the worst case we have to shift $n/2$ items to make room for the new item
  - remove take $O(n)$ time since in the worst case we have to shift $n/2$ items to compact the items after the removal
- The lookup (search) table is effective only for dictionaries of small size or for dictionaries on which searches are the most common operations, while insertions and removals are rarely performed (e.g., credit card authorizations)

Binary Search Trees (ADT)

- 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



프로그램 구현상의 편의를 위해서 의미없는 외부노드를 추가(혹은 범위를 위해서)

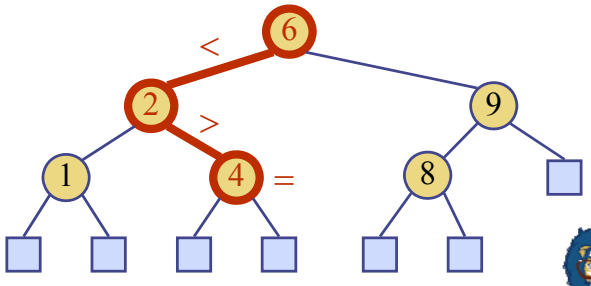
Binary Search Trees (ADT)

- Fundamental Methods
  - **get (*k*)**
    - return the value *v* for the entry, (*k*,*v*), with key equal to *k*, if it exists
  - **put (*k*,*v*)**
    - enter the entry (*k*,*v*) as the mapping of *k* to *v*
  - **remove (*k*)**
    - remove the entry with key equal to *k*, and return it's value

Search

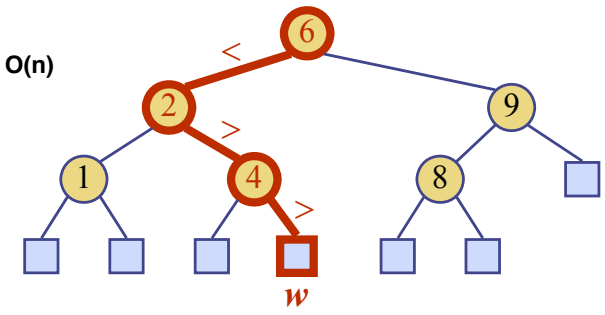
- To search for a key ***k***, we trace a downward path starting at the root
- The next node visited depends on the comparison of ***k*** with the key of the current node
- If we reach a leaf, the key is not found
- Example: get(4):
  - Call TreeSearch(4,root)
- The algorithms for floorEntry and ceilingEntry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

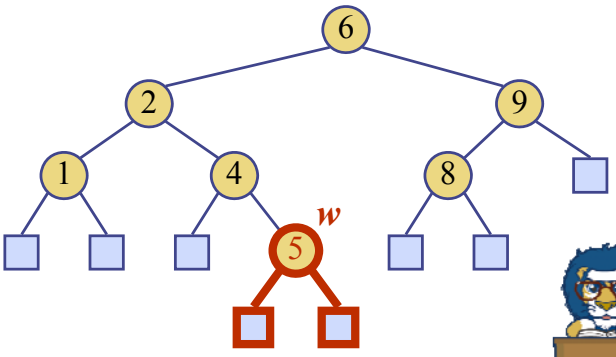
```
Algorithm TreeSearch(k, v)
    if T.isExternal (v)
        return v
    if k < key(v)
        return TreeSearch(k, T.left(v))
    else if k = key(v)
        return v
    else { k > key(v) }
        return TreeSearch(k, T.right(v))
```



Insertion

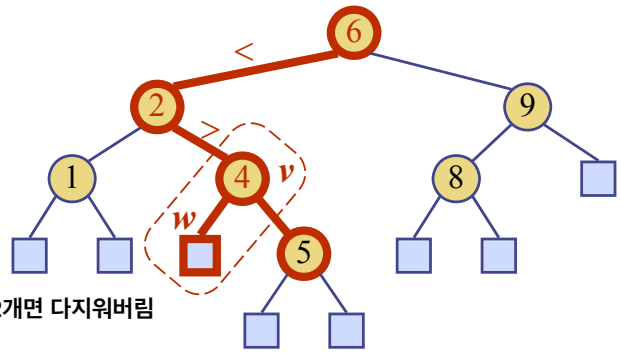
- To perform operation put(k, o), we search for key k (using TreeSearch)
- Assume k is not already in the tree, and let w be the leaf reached by the search
- We insert k at node w and expand w into an internal node
- Example: insert 5



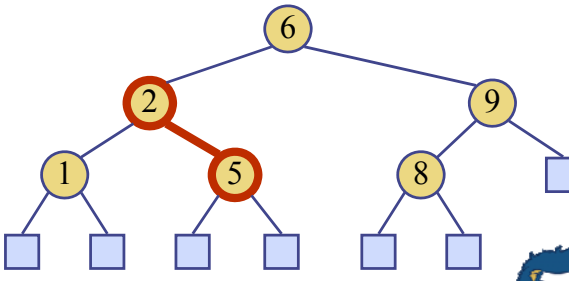


Deletion

- To perform operation remove(***k***), we search for key ***k***
- Assume key ***k*** is in the tree, and 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



leaf가 2개면 다지워버림

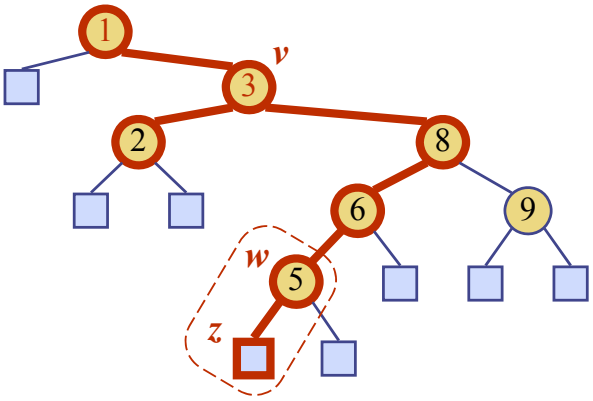


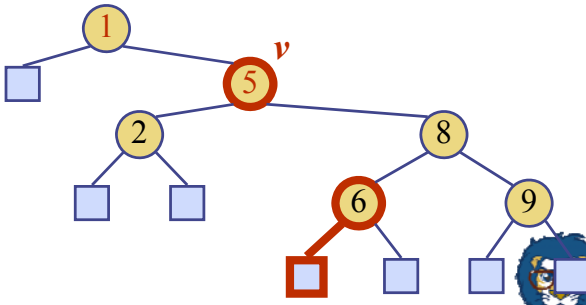
Deletion (cont.)

- swap with the rightmost node of its left subtree or
- swap with the leftmost node of its right subtree

inorder 했을때 앞에나오는 경우나 뒤에 나오는 경우가 위에 빨간색박스의 경우

- 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



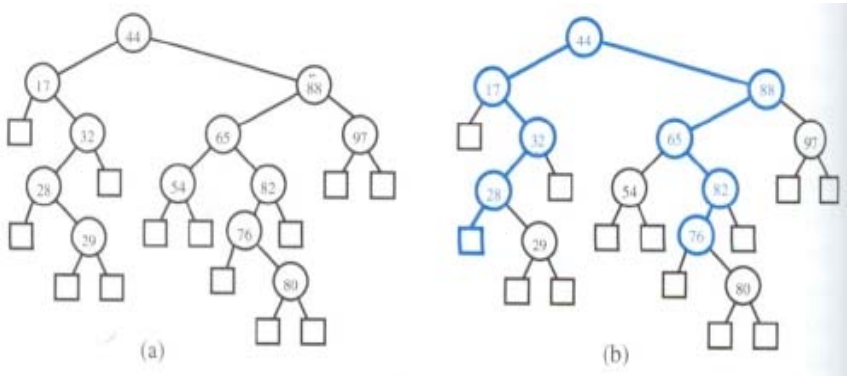


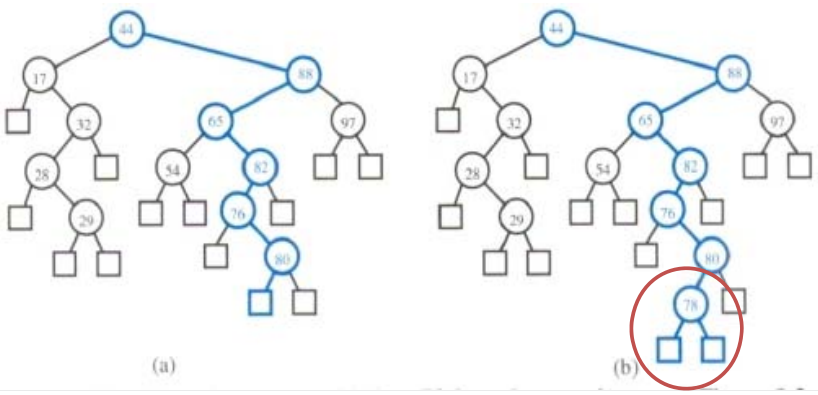
높이를 줄이기 위해서 2대신 5를 집어넣는게 적합

HANYANG UNIVERSITY

Example (insertion)

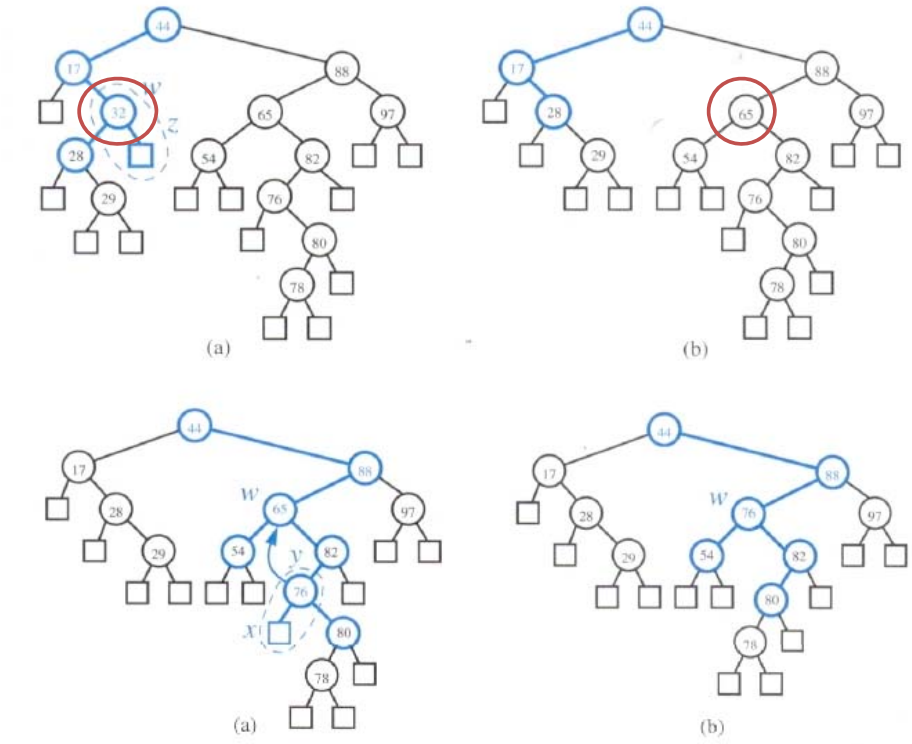
- BST





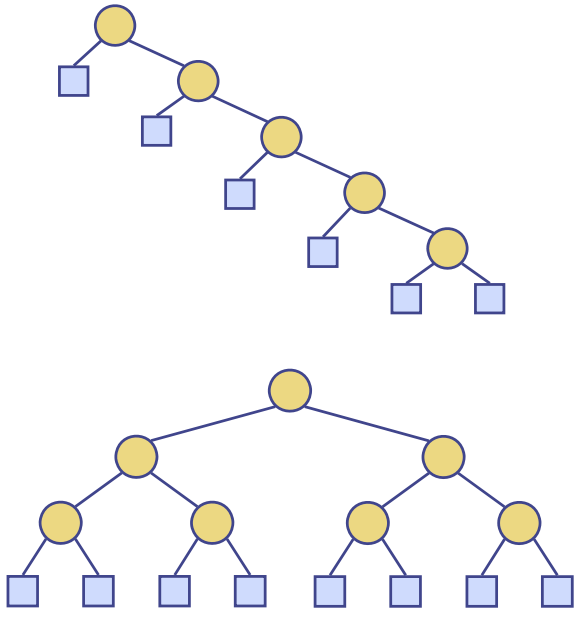
HANYANG UNIVERSITY

Example (deletion)



Performance

- Consider an ordered map with n items implemented by means of a binary search tree of height h
  - the space used is $O(n)$
  - methods get, floorEntry, ceilingEntry, 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



Q & A

한양대학교 ERICA 캠퍼스