Binary Search Tree
Sorting and Searching in a Dynamic Set

S.V.N. (vishy) Vishwanathan

University of California, Santa Cruz
vishy@ucsc.edu

February 29, 2016
Outline

1. Binary Search Trees
What is a Binary Search Tree?

- Binary tree (each node has 0, 1, or 2 children)
- Each node contains
  - key
  - satellite data
  - left
  - right
  - parent
- The keys satisfy the binary search tree property

Let \( x \) be a node in a binary search tree. If \( y \) is a node in the left subtree of \( x \), then \( y.key \leq x.key \). If \( y \) is a node in the right subtree of \( x \), then \( y.key \geq x.key \).
Inorder Tree Walk

INORDER-TREE-WALK(x)

1 if x ≠ NIL
2 INORDER-TREE-WALK(x.left)
3 print x.key
4 INORDER-TREE-WALK(x.right)
TREE-SEARCH(x, k)

1  if x == NIL or k == x.key
2     return x
3  if k < x.key
4     return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
Iterative Search

ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2       if  k < x.key
3           x = x.left
4       else x = x.right
5  return x
Search

```
       15
      /   \\
     6     18
    /     /  \
   3     17   20
 /    / \
2    7   13
   /    \
  4     9
```

Minimum

TREE-MINIMUM(x)

1 while x.left ≠ NIL
2 x = x.left
3 return x
Maximum

TREE-MAXIMUM(x)

1 while x.right ̸= NIL
2 x = x.right
3 return x
Minimum and Maximum

[Diagram of a binary search tree]

15
18
6
17
3
7
20
2
4
13
9
3
2
4
13
9
Successor

- Given a node $x$, its successor is the node with the smallest key greater than $x.key$
- It is the next node which occurs after $x$ in the sorted order determined by an inorder tree walk
Successor

Look at node 15 and 13
Successor

TREE-SUCCESSOR(x)

1. if x.right ≠ NIL
2. return TREE-MINIMUM(x.right)
3. y = x.parent
4. while y ≠ NIL and x == y.right
5. x = y
6. y = y.parent
7. return y
Time Complexity

**Theorem**

We can implement the dynamic-set operations \textsc{search}, \textsc{minimum}, \textsc{maximum}, \textsc{successor}, and \textsc{predecessor} so that each one runs in \(O(h)\) time on a binary search tree of height \(h\)
**Insertion**

TREE-INSERT\(( T, z )\)

1. \( y = \text{NIL} \)
2. \( x = T.\text{root} \)
3. while \( x \neq \text{NIL} \)
   4. \( y = x \)
   5. if \( z.\text{key} < x.\text{key} \)
      6. \( x = x.\text{left} \)
   7. else \( x = x.\text{right} \)
8. \( z.\text{parent} = y \)
9. if \( y == \text{NIL} \)
10. \( T.\text{root} = z \) // tree \( T \) was empty
11. elseif \( z.\text{key} < y.\text{key} \)
12. \( y.\text{left} = z \)
13. else \( y.\text{right} = z \)
Insertion

Look at node 13
Deletion: Three Cases

- **z** has no children. Delete z and modify its parent
- **z** has one child. Elevate child to take z’s position
- **z** has two children.
  - Find z’s successor **y**. We know that
    - **y** has no left child. Why?
    - **y** must lie in the right subtree of **z**
  - Replace z with **y**
  - However have to be careful if **y** is the right child of **z**
Deletion

(a)

(b)

(c)

(d)
Deletion: Split to Four Cases

- z has no left child. Replace z by its right child (which may be a NIL
- z has a left child. Replace z by its left child.
- z has both left and right children.
  - Find z’s successor y
  - y must lie in the right subtree of z and has no left child
    - If y is z’s right child, then replace z by y
    - Otherwise, replace y by its right child, and replace z by y
Pseudo-code

TREE-DELETE($T, z$)

1. if $z.left == NIL$
2. then TRANSPLANT($T, z, z.right$)
3. elseif $z.right == NIL$
4. then TRANSPLANT($T, z, z.left$)
5. else $y = TREE-MINIMUM(z.right)$
6. if $y.parent \neq z$
7. then TRANSPLANT($T, y, y.right$)
8. $y.right = z.right$
9. $y.right.parent = y$
10. TRANSPLANT($T, z, y$)
11. $y.left = z.left$
12. $y.left.parent = y$
Pseudo-code

TRANSPLANT($T, u, v$)

1. if $u$.parent == NIL
2.      $T$.root = $v$
3. elseif $u == u$.parent.left
4.      $u$.parent.left = $v$
5. else $u$.parent.right = $v$
6. if $v \neq$ NIL
7.      $v$.parent = $u$.parent
Questions?