Algorithms and Data Structures

CMPSC 465

Binary Search Trees

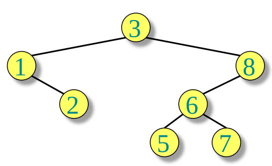
Paul Medvedev

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

1

Binary Search Trees

- Binary tree: every node has 0, 1 or 2 children
- BST property:
  - If *y* is in left subtree of *x*, then key[y] <= key[x]
  - If *y* is in right subtree of *x*, then key[y] >= key[x]



Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

2

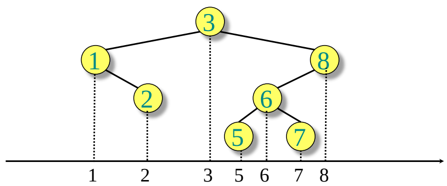
Implementation

- Keep pointers to parent and both children
- Each NODE has
  - left: pointer to NODE
  - right: pointer to NODE
  - p: pointer to NODE
  - key: real number (or other type that supports comparisons)

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

3

Drawing BST's



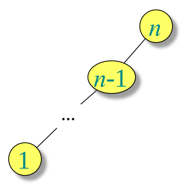
- To help you visualize relationships:
  - Drop vertical lines to keep nodes in right order
- For example: picture shows the only legal location to insert a node with key 4

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

4

Height of a BST can be...

- ... as little as log(n)
  - full balanced tree
- ... as much as n-1
  - unbalanced tree



Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

5

Searching a BST

```
TREE-SEARCH(x, k)
if x == NIL or k == key[x]
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)

Initial call is TREE-SEARCH(T.root, k).
```

- Running time: Θ(height)

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

6

Insertion

```
TREE-INSERT(T, z)
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z        // tree T was empty
elseif z.key < y.key
    y.left = z
else y.right = z
```

- Find location of insertion, keep track of parent during search
- Running time: Θ(height)

Tree Min and Max

```
TREE-MINIMUM(x)
while x.left ≠ NIL
    x = x.left
return x

TREE-MAXIMUM(x)
while x.right ≠ NIL
    x = x.right
return x
```

- Running time: Θ(height)

Deletion

```
TRANSPLANT(T, u, v)
if u.p == NIL
    T.root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
if v ≠ NIL
    v.p = u.p
```

- Running time: Θ(height)

```
TREE-DELETE(T, z)
if z.left == NIL
    TRANSPLANT(T, z, z.right)       // z has no left child
elseif z.right == NIL
    TRANSPLANT(T, z, z.left)        // z has just a left child
else // z has two children.
    y = TREE-MINIMUM(z.right)       // y is z's successor
    if y.p ≠ z
        // y lies within z's right subtree but is not the root of this subtree.
        TRANSPLANT(T, y, y.right)
        y.right = z.right
        y.right.p = y
    // Replace z by y.
    TRANSPLANT(T, z, y)
    y.left = z.left
    y.left.p = y
```

Tree Successor

```
TREE-SUCCESSOR(x)
if x.right ≠ NIL
    return TREE-MINIMUM(x.right)
y = x.p
while y ≠ NIL and x == y.right
    x = y
    y = y.p
return y
```

- Running time: Θ(height)

Traversals (not just for BSTs)

- Traversal: an algorithm that visits every node in the tree to perform some operation, e.g.,
  - Printing the keys
  - Updating some part of the structure
- 3 basic traversals for trees
  - Inorder
  - Preorder
  - Postorder

Inorder traversal

```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print key[x]
    INORDER-TREE-WALK(x.right)
```

- Running time: Θ(n)

Postorder traversal

- Write pseudocode that computes
  - height of a BST
  - number of nodes in a BST
  - average depth
  - ...
- Example: height
  - Find-height(T)
    - if T==NIL return -1
    - else
      - h1 := Find-height(T.left)
      - h2 := Find-height(T.left)
      - return max(h1, h2) + 1

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

13

Preorder Traversal

- Recall that insertion order affects the shape of a BST
  - insertion in sorted order: height n-1
  - random order: height O(log n) with high probability (we leave out proof)
- Write pseudocode that prints the elements of a binary search tree in a plausible insertion order
  - that is, an insertion order that would produce this particular shape
- (print nodes in a preorder traversal)

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

14

Exercise on insertion order

- Exercise: Write pseudocode that takes a sorted list and produces a "good" insertion order (that would produce a balanced tree)

Based on slides by S. Raskhodnikova, A. Smith, K. Wayne, C. Leiserson and E. Demaine.

15