6331 - Algorithms, CSE, OSU
Binary search trees

Instructor: Anastasios Sidiropoulos

Binary search trees

For every node x:

- $x.k$: key
- $x.p$: pointer to the parent of x
- $x.left$: pointer to the left child of x
- $x.right$: pointer to the right child of x

Ordering in binary search trees

Let x be a node in a binary search tree.

For any node y in the left subtree of x, we have $y.key \leq x.key$.

For any node y in the right subtree of x, we have $y.key \geq x.key$.

Inorder traversal

```
Inorder-Tree-Walk(x)
  if x ≠ NIL
    Inorder-Tree-Walk(x.left)
    print x.key
    Inorder-Tree-Walk(x.right)
```

Inorder traversal

```
Inorder-Tree-Walk(x)
  if x ≠ NIL
    Inorder-Tree-Walk(x.left)
    print x.key
    Inorder-Tree-Walk(x.right)
```

What does this procedure do?

Running time of Inorder-Tree-Walk

$T(n) = \Omega(n)$, since it outputs n elements.

Let $d = O(1)$ be the time required to examine a node. We argue that $T(n) \leq (c+d)n + c$, for some constant c.

$$
\begin{aligned}
T(n) &\leq T(k) + T(n-k-1) + d \\
&= ((c+d)k + c) + ((c+d)(n-k-1) + c) + d \\
&= (c+d)n + c - (c+d) + c + d \\
&= (c+d)n + c \\
&= O(n)
\end{aligned}
$$

Therefore, $T(n) = \Theta(n)$.

Searching

```
Tree-Search(x, k)
  if x = NIL or k = x.key
    return x
  if k < x.key
    return Tree-Search(x.left, k)
  else return Tree-Search(x.right, k)
```

Searching

Tree-Search(x, k)
  if $x = \text{NIL}$ or $k = x.key$
    return x
  if $k < x.key$
    return Tree-Search($x.left$, k)
  else return Tree-Search($x.right$, k)

What does this procedure do?

Searching

Tree-Search(x, k)
  if $x =$ NIL or $k = x.key$
    return x
  if $k < x.key$
    return Tree-Search($x.left, k$)
  else return Tree-Search($x.right, k$)

What does this procedure do?

What happens if k does not appear in the tree?

Searching

Tree-Search(x, k)
  if $x =$ NIL or $k = x.key$
    return x
  if $k < x.key$
    return Tree-Search($x.left, k$)
  else return Tree-Search($x.right, k$)

What does this procedure do?

What happens if k does not appear in the tree?

What is the running time of Tree-Search?

Minimum and maximum

```
Tree-Minimum(x)
  while x.left ≠ NIL
    x = x.left
  return x

Tree-Maximum(x)
  while x.right ≠ NIL
    x = x.right
  return x
```

Minimum and maximum

Tree-Minimum(x)
  while $x.left \neq$ NIL
    $x = x.left$
  return x

Tree-Maximum(x)
  while $x.right \neq$ NIL
    $x = x.right$
  return x

What do these procedures do?

Minimum and maximum

Tree-Minimum(x)
  while $x.left \neq$ NIL
    $x = x.left$
  return x

Tree-Maximum(x)
  while $x.right \neq$ NIL
    $x = x.right$
  return x

What do these procedures do?

Running time?

Successor

Find the next element in the sorted order.

Tree-Successor(x)
  if $x.right \neq$ NIL
    return Tree-Minimum($x.right$)
  $y = x.p$
  while $y \neq$ NIL and $x = y.right$
    $x = y$
    $y = y.p$
  return y

Successor

Find the next element in the sorted order.

Tree-Successor(x)
  if $x.right \neq$ NIL
    return Tree-Minimum($x.right$)
  $y = x.p$
  while $y \neq$ NIL and $x = y.right$
    $x = y$
    $y = y.p$
  return y

How does this procedure work?

Successor

Find the next element in the sorted order.

Tree-Successor(x)
  if $x.right \neq$ NIL
    return Tree-Minimum($x.right$)
  $y = x.p$
  while $y \neq$ NIL and $x = y.right$
    $x = y$
    $y = y.p$
  return y

How does this procedure work?

Running time?

Insertion

Tree-Insert(T, z)

$y = $ NIL

$x = T.root$

while $x \neq$ NIL

$y = x$

if $z.key < x.key$

$x = x.left$

else $x = x.right$

$z.p = y$

if $y = $ NIL

$T.root = z$ // T was empty

elseif $z.key < y.key$

$y.left = z$

else $y.right = z$

Deletion

Deleting a node z.

- If z has no children, we remove z.
- If z has one child y, then we elevate y to the position of z.
- If z has two children, then we find the z's successor y. We replace z by y.

An auxiliary procedure

Replace the subtree rooted at u with the subtree rooted at v.

Transplant(T, u, v)
    if $u.p = \text{NIL}$
        $T.root = v$
    elseif $u = u.p.left$
        $u.p.left = v$
    else $u.p.right = v$
    if $v \neq \text{NIL}$
        $v.p = u.p$

Deletion

```
Tree-Delete(T, z)
    if z.left = NIL
        Transplant(T, z, z.right)
    elseif z.right = NIL
        Transplant(T, z, z.left)
    else y = Tree-Minimum(z.right)
        if y.p ≠ z
            Transplant(T, y, y.right)
            y.right = z.right
            y.right.p = y
        Transplant(T, z, y)
        y.left = z.left
        y.left.p = y
```

Performance of binary search trees

What is the worst-case running time for inserting n elements in an empty binary search tree?

Performance of binary search trees

What is the worst-case running time for inserting n elements in an empty binary search tree?

What is the best-case running time?

Performance of binary search trees

What is the worst-case running time for inserting n elements in an empty binary search tree?

What is the best-case running time?

What happens when we insert the same element n times, starting from an empty binary search tree?

Performance of binary search trees

What is the worst-case running time for inserting n elements in an empty binary search tree?

What is the best-case running time?

What happens when we insert the same element n times, starting from an empty binary search tree?

What is the worst-case running time for removing all elements from a binary search tree of height h?