Trees

abhi shelat
CUNY Oct 1 2015
Insert(A,x)

Search(A,x)

Delete(A,x)
struct node {
    int value;
    node *left;
    node *right;
}

0,1,2 children

Binary tree

left subtree

right subtree
Binary SEARCH Tree

left-child is **smaller** than parent

right-child is **larger** than parent
BST: Search(x)

Search(3, curr)
struct node {
    int value;
    node *left;
    node *right;
}

search(x, curr)
    if (curr == NULL)
        return NULL;
    else if (x < curr->value)
        return search(x, curr->left);  // search left
    else if (x > curr->value)
        return search(x, curr->right); //search right
    else
        return curr;
Insert

Insert(2, curr)
Insert

 Insert(2, curr)
insert(x, curr)
    if (curr == NULL)
        cure = new node(x);
    else if (x < curr->value)
        return insert(x, curr->left);  // search left
    else if (x > curr->value)
        return insert(x, curr->right); // search right
    else
        return curr;

If code reaches this point...
Find-Max
Delete

Delete(7)

deleting node with no children
Delete

Delete(7)
Delete

Delete(1)

deleting node with 1 child

Promote child
Delete(1)
Delete

deleting node with 2 children
Delete

deleting node with 2 children

1. find min on right subtree
2. promote
Delete

1. find min on right subtree
2. promote
Perfect tree

All nodes have the same height
binary heap

full tree, the value of parent $\leq$ value of children

insert
extract-min
decrease-key
binary heap

full tree, the value of parent $\leq$ value of children
binary heap

full tree, the value of parent $\leq$ value of children

insert(8)
binary heap

full tree, the value of parent <= value of children

insert(8)
binary heap

full tree, the value of parent $\leq$ value of children

insert(8)
binary heap

full tree, the value of parent \( \leq \) value of children

extract-min
binary heap

full tree, the value of parent $\leq$ value of children

how to extract min?
binary heap

full tree, key value <= to key of children
how to extract min?
binary heap
full tree, key value \leq to key of children

how to decreasekey?
binary heap

full tree, key value $\leq$ to key of children

how to extract min?
how to decrease key?
binary heap
full tree, key value $\leq$ to key of children

how to decreasekey?
Red-Black tree

Each node has a color.
Red-Black tree

Root is black.

Each node has a color.

Leaves are black.
Red-Black tree

Root is black.

Each node has a color.

Children of red are black.

Leaves are black.
Red-Black tree

Root is black.

Each node has a color.

Children of red are black.

Leaves are black.

For every node, all paths to its leaves have same # of black nodes.
Red-Black tree

bh(x): # of black nodes from x to any leaf of x

For every node, all paths to its leaves have same # of black nodes.
Nil child of a node is a special “Nil leaf.”
Red-Black Tree Thm

Thm: An RB tree with n nodes has height at most \(2\log(n + 1)\).

Proof: By induction.

1. A subtree at \(x\) has \(2^{bh(x)} - 1\) nodes.

2. \(bh(root) \geq h/2\)
Rotations

Figure 13.2
The rotation operations on a binary search tree. The operation \textsc{Left-Rotate}(T, x) transforms the configuration of the two nodes on the right into the configuration on the left by changing a constant number of pointers. The inverse operation \textsc{Right-Rotate}(T, y) transforms the configuration on the left into the configuration on the right. The letters \(\alpha\), \(\beta\), and \(\gamma\) represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in \(\alpha\) precede \(x\), which precedes the keys in \(\gamma\), which precede the key in \(\beta\). A rotation operation replaces one node and its right child with another node and its right child. Figure 13.3 shows an example of how \textsc{Left-Rotate} modifies a binary search tree. The code for \textsc{Right-Rotate} is symmetric. Both \textsc{Left-Rotate} and \textsc{Right-Rotate} run in \(O(1)\) time. Only pointers are changed by a rotation; all other attributes in a node remain the same.

Exercises

13.2-1 Write pseudocode for \textsc{Right-Rotate}.

13.2-2 Argue that in every \(n\)-node binary search tree, there are exactly \(\lfloor n/2 \rfloor + 1\) possible rotations.