CMSC 341
Lecture 14

Announcements

Proj3 due today
Proj4 up by midnight
Splay Trees

Concept
– adjust tree in response to accesses to make common operations efficient
– after access node is moved to root by splaying

Performance
– amortized such that m operations take O(m lg n) where n is the number of insertions

Splay Operation

Traverse tree from node x to root, rotating along the way until x is the root

Each rotation
– If x is root, do nothing.
– If x has no grandparent, rotate x about its parent.
– If x has a grandparent,
  • if x and its parent are both left children or both right children, rotate the parent about the grandparent, then rotate x about its parent
  • if x and its parent are opposite type children (one left and the other right), rotate x about its parent, then rotate x about its new parent (former grandparent)
Operations in Splay Trees

insert
  – first insert as in normal binary search tree
  – than splay inserted node

find
  – search for node
  – if found, splay to root; otherwise splay last node on path

Operations on Splay Trees (cont)

remove
  – splay selected element to root
  – disconnect left and right subtrees from root
  – do one of:
    • splay max item in $T_L$ (then $T_L$ has no right child)
    • splay min item in $T_R$ (then $T_R$ has no left child)
  – connect other subtree to empty child
Performance of Splay Trees

insert
- regular bst insertion -- \(O(\text{depth})\)
- splay: \(O(1)\) for each rotation, \(O(\text{depth})\) rotations

Red-Black Trees

Concept
- BST with more relaxed notion of balance than AVL trees
- no path from \(N\) to leaf is more than twice as long as any other
- for RB tree with \(n\) nodes and height \(h\), \(h \leq 2 \log(n+1)\)

Definition:  A red-black tree is a binary search tree where:
- Every node is either red or black.
- Each NULL pointer is considered to be a black node
- If a node is red, then both of its children are black.
- Every path from a node to a leaf contains the same number of black nodes.

Definition:  The height of a node \(n\) in a red-black tree is the number of black nodes on any path to a leaf, not counting \(n\).
RedBlackNode

template <class Comparable>
class RedBlackNode{
    Comparable  element;
    RedBlackNode *left;
    RedBlackNode *right;
    int color;

    RedBlackNode(const Comparable & theElement =
                 Comparable(), RedBlackNode *lt = NULL,
                 RedBlackNode *rt = NULL, int c = RedBlackTree<Comparable>::BLACK) :
        element(theElement), left(lt), right(rt), color(c) {}

    friend class RedBlackTree<Comparable>;
};

RedBlackTree Class

template <class Comparable>
class RedBlackTree {
public:
    explicit RedBlackTree (const Comparable &negInf);
    RedBlackTree (const RedBlackTree &rhs);
    ~RedBlackTree();
    enum {RED, BLACK};
    // usual public member functions
private:
    RedBlackNode<Comparable> *header;
    const Comparable ITEM_NOT_FOUND;
    RedBlackNode<Comparable> *nullNode;
    RedBlackNode<Comparable> *current;
    RedBlackNode<Comparable> *parent;
    RedBlackNode<Comparable> *grand;
    RedBlackNode<Comparable> *great;
RedBlackTree (cont.)

void handleReorient(const Comparable &item);
RedBlackNode<Comparable> *rotate(const Comparable &item, RedBlackNode<Comparable> *parent) const;

// additional private member funcs
};

Constructor

template <class Comparable>
RedBlackTree<Comparable>::RedBlackTree(const Comparable &negInf) : ITEM_NOT_FOUND(negInf) {
    nullNode = new RedBlackNode<Comparable>;
    nullNode->left = nullNode->right = nullNode;
    header = newRedBlackNode<Comparable>(negInf);
    header->left = header->right = nullNode;
}