University of Zurich[UZH]

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

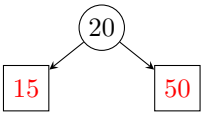
Data Structures and Algorithms
Assignment 5

Apr 18, 2016

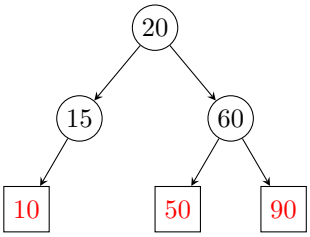
Red-Black Trees

Task 1.

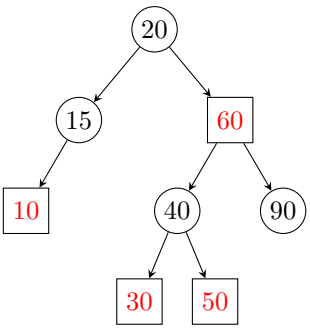
Insert (20, 50, 15)



Insert (10, 60, 90)



Insert (40, 30)



Task 2.

```
#include <stdlib.h>
#include <stdio.h>

#define black 0
#define red 1

struct rb_node {
    int key;
    int color;
    struct rb_node* left;
    struct rb_node* right;
    struct rb_node* parent;
};

struct rb_tree {
    struct rb_node *root;
    struct rb_node *nil;
};

struct rb_tree* rb_initialize() {
    struct rb_tree* tree;
    struct rb_node* node;

    tree = (struct rb_tree*) malloc(sizeof(struct rb_tree));

    tree->nil = (struct rb_node*) malloc(sizeof(struct rb_node));
    tree->nil->parent = tree->nil;
    tree->nil->left = tree->nil;
    tree->nil->right = tree->nil;
    tree->nil->color = black;
    tree->nil->key = -2;

    tree->root = tree->nil;

    return tree;
}

void bst_insert(struct rb_tree* tree, struct rb_node *nodeToInsert) {
    struct rb_node *oneDelayed = tree->nil;
    struct rb_node *insertPlace = tree->root;

    nodeToInsert->left=tree->nil;
    nodeToInsert->right=tree->nil;
    nodeToInsert->parent=tree->nil;
    while (insertPlace != tree->nil) {
        oneDelayed = insertPlace;
        if (nodeToInsert->key < insertPlace->key)
            insertPlace = insertPlace->left;
        else
            insertPlace = insertPlace->right;
    }
```

```
    if (oneDelayed == tree->nil)
        tree->root = nodeToInsert;
    else if (oneDelayed->key < nodeToInsert->key) {
        oneDelayed->right = nodeToInsert;
        nodeToInsert->parent = oneDelayed;
    } else {
        oneDelayed->left = nodeToInsert;
        nodeToInsert->parent = oneDelayed;
    }
}

void rb_printInorderDepth(struct rb_node *root, struct rb_tree *tree,
     int depth) {
  int i;

  if (root == tree->nil)
    return;

  rb_printInorderDepth(root->left, tree, depth+1);

  for (i = 0; i < depth*3; i++)
    printf(" ");
  printf("%d\n", root->key);

  rb_printInorderDepth(root->right, tree, depth+1);
}

void rb_print(struct rb_tree *tree) {
  rb_printInorderDepth(tree->root, tree, 0);
  printf("\n");
}

struct rb_node* rb_search(struct rb_tree* tree, int q) {
    struct rb_node* x = tree->root;

    if (x == tree->nil)
        return x;
    while (x->key != q) {
        if (q < x->key)
            x = x->left;
        else
            x = x->right;
        if (x == tree->nil)
            return x;
    }
    return x;
}

void rb_leftRotate(struct rb_tree* tree, struct rb_node* x) {
    struct rb_node* y;

    y = x->right;
    x->right = y->left;
    if (y->left != tree->nil)
```

```
        y->left->parent = x;
    y->parent = x->parent;
    if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;

    if (x == tree->root)
        tree->root = y;
}

void rb_rightRotate(struct rb_tree* tree, struct rb_node* y) {
    struct rb_node* x;

    x = y->left;
    y->left = x->right;
    if (tree->nil != x->right)
        x->right->parent = y;
    x->parent = y->parent;
    if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    x->right = y;
    y->parent = x;

    if (y == tree->root)
        tree->root = x;
}

void main() {
    struct rb_tree *tree;
    struct rb_node* n;

    tree = rb_initialize();

    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 5;
    bst_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 90;
    bst_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 20;
    bst_insert(tree, n);
    rb_print(tree);

    n = rb_search(tree, 90);
    rb_rightRotate(tree, n);
    n = rb_search(tree, 5);
    rb_leftRotate(tree, n);
    rb_print(tree);
```

```
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 60;
    bst_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 30;
    bst_insert(tree, n);
    rb_print(tree);
    n = rb_search(tree, 90);
    rb_rightRotate(tree, n);
    rb_print(tree);
}

// Linux, Mac: gcc task2.c -o task2; ./task2
// Windows: gcc task2.c -o task2; task2
```

Output:

```
5
        20
   90


    5
20
    90


    5
20
        30
    60
        90
```

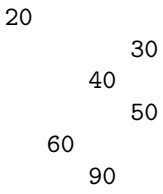
Task 3.

```
#include <stdlib.h>
#include <stdio.h>

#define black 0
#define red 1

// ... include code from Task 2 ...

void rb_insert_fixup(struct rb_tree* tree, struct rb_node* n) {
    struct rb_node* y;
    while (n->parent->color == red) {
        if (n->parent == n->parent->parent->left) { /* non-mirrored cases
             */
            y = n->parent->parent->right;
            if (y->color == red) { /* case 1 */
                n->parent->color = black;
                y->color = black;
                n->parent->parent->color = red;
                n = n->parent->parent;
            } else {
                if (n == n->parent->right) { /* case 2 */
                    n = n->parent;
                    rb_leftRotate(tree, n);
                }
                n->parent->color = black; /* case 3 */
                n->parent->parent->color = red;
                rb_rightRotate(tree, n->parent->parent);
            }
        } else { /* mirrored cases */
            y = n->parent->parent->left;
            if (y->color == red) { /* case 1m */
                n->parent->color = black;
                y->color = black;
                n->parent->parent->color = red;
                n = n->parent->parent;
            } else {
                if (n == n->parent->left) { /* case 2m */
                    n = n->parent;
                    rb_rightRotate(tree,n);
                }
                n->parent->color = black; /* case 3m */
                n->parent->parent->color = red;
                rb_leftRotate(tree, n->parent->parent);
            }
        }
    }
}

void rb_insert(struct rb_tree* tree, struct rb_node* n) {
    bst_insert(tree, n);
    n->color = red;
    rb_insert_fixup(tree, n);
```

```
    tree->root->color = black;
}

void main() {
    struct rb_tree *tree;
    struct rb_node *n;

    tree = rb_initialize();

    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 5;
    rb_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 90;
    rb_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 20;
    rb_insert(tree, n);
    rb_print(tree);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 60;
    rb_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 30;
    rb_insert(tree, n);
    rb_print(tree);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 50;
    rb_insert(tree, n);
    n = (struct rb_node*)malloc(sizeof(struct rb_node));
    n->key = 40;
    rb_insert(tree, n);
    rb_print(tree);
}

// Linux, Mac: gcc task3.c -o task3; ./task3
// Windows: gcc task3.c -o task3; task3
```

Output:

```
   5
20
   90


   5
20
      30
   60
      90


   5
```


20
30
40
50
60
90

Task 4.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define MAX_ARRAY_SIZE 1000000

// ... include code from Task 2 and Task 3 ...

// ------------------------------ main ---------------------------------------

void main() {
    int n=500000, i;
    clock_t start;
    clock_t end;
    float seconds;
    struct rb_tree *tree;
    struct rb_node* node;

    printf(" | time |\n");

    // ------------------------ red black tree --------------------------

    tree = rb_initialize();
    start = clock();
    for (i = 0; i < n; i++) {
        node = (struct rb_node*)malloc(sizeof(struct rb_node));
        node->key = i;
        rb_insert(tree, node);
    }
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf("red black tree | %20.15f |\n", seconds);

    // -------------------------- BST --------------------------------

    tree = rb_initialize();
    start = clock();
    for (i = 0; i < n; i++) {
        node = (struct rb_node*)malloc(sizeof(struct rb_node));
        node->key = i;
        bst_insert(tree, node);
    }
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf("\n");
    printf("BST | %20.15f |\n", seconds);
}

// Linux, Mac: gcc task4.c -o task4; ./task4
// Windows: gcc task4.c -o task4; task4
```

	time (sec.)
BST	772.413
red black tree	0.146