Informatics II
Exercise 9

April 26, 2020

Learning Goals:

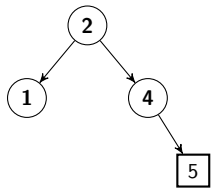
- Practise inserting nodes in a Red Black Tree
- Practise deleting nodes from a Red Black Tree.
- Implement Red Black Trees on C.

Red Black Trees

Consider the red-black tree in Figure 1a where black nodes are denoted with a circle and red nodes are denoted with a square. We use the key to represent a node. For example, 2 represents the node with key 2. We consider four operations:

1. Creating a new node (left and right are set to NULL):
   Example: create node 9: `create(9)`
2. Setting a property (color, key, left, right) of a node:
   Example: set the key of node 9 to 5: `9->key = 5`. Here, 9 represents the node with key 9.
3. Rotating a node:
   Example: right rotate node 5: `RightRotate(5)`
4. Deleting a node:
   Example: delete node 9: `delete(9)`

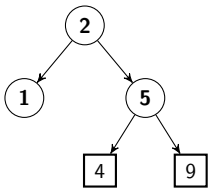
The result of applying the operations in Figure 1b to the red-black tree in Figure 1a yields the red-black tree in Figure 1c.



(a) Before operations

```
create(9)
9->color = red
5->right = 9
5->color = black
4->color = red
LeftRotate(4)
```

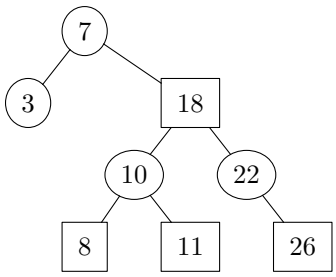
(b) Operations



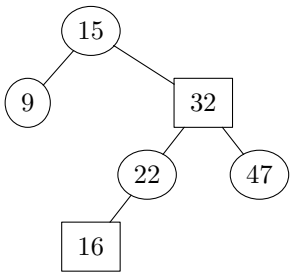
(c) After operations

Figure 1: Tree before and after operations

Task 1. Consider the red-black tree shown below. State the operations that are required to insert 15 into the red-black tree.



Task 2. Consider the red-black tree below.



State the operations that are required to delete **9** from the red-black tree.

Task 3. You are asked to complete an implementation of red black trees. A **red-black node** is of the following type:

```
1   struct rb_node {
2     int key, color;
3     struct rb_node *left, *right, *parent;
4   };
```

A **red-black tree** is of the following type:

```
1   struct rb_tree {
2     int bh;
3     struct rb_node *root;
4     struct rb_node *nil;
5   };
```

In datatype `rb_tree`, `root` points to the root of the tree. Sentinel `nil` is a convenient node that deals with boundary conditions in red-black tree code. For a red-black tree `T`, the sentinel `T.nil` is an object with the same attributes as an ordinary node in the tree. Its color attribute is `black`, its `parent, left, right` are `T.nil`, and its `key` can take on any arbitrary values. We use the sentinel so that we can treat a `NIL` child of a node `x` as an ordinary node whose parent is `x`. We use one

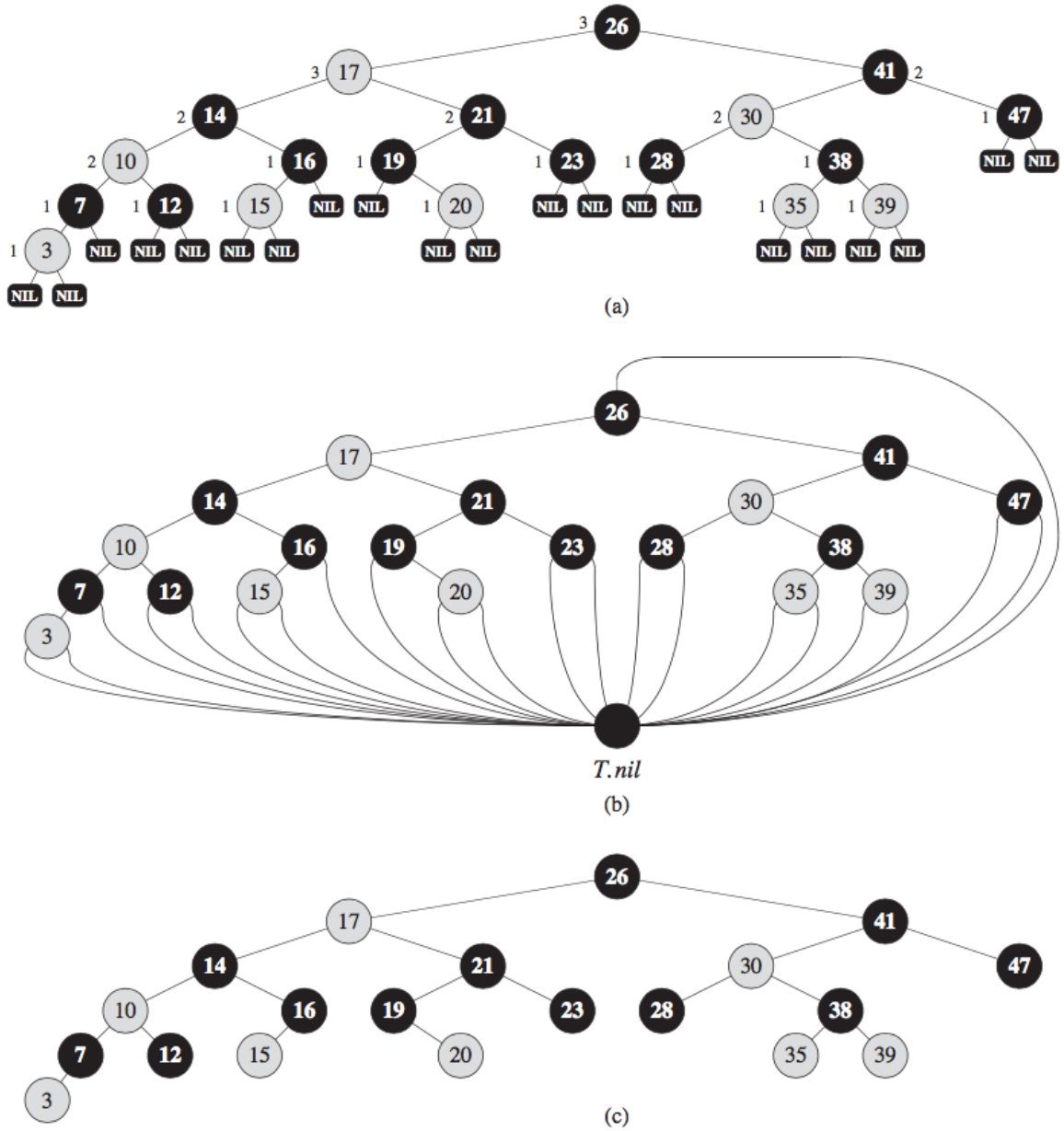


Figure 2: A red-black tree with black nodes darkened and red nodes shaded. *(a)* Every leaf, shown as a NIL, is black. *(b)* The same red-black tree but with each `NIL` replaced by the single sentinel `T.nil`, that is always black. The root's parent is also the sentinel. *(c)* The same red-black tree but with leaves and the root's parent omitted entirely.

sentinel `T.nil` to represent all NIL nodes of a red-black tree `T` (all leaves and the root's parent). Refer to Fig. 2 for illustration.

Along with the above datatypes create two constants, *red* and *black* equal to 0 and 1 respectively, and the following functions:

- *struct rb_tree* rb_initialize()* that creates a red black tree `T` with a *root* and a *NIL node* (`left = right = parent = T.nil` and `color = black`).
- *void rb_leftRotate(struct rb_tree* tree, struct rb_node* x)* that does left rotation on node `x` in `tree`.
- *void rb_rightRotate(struct rb_tree* tree, struct rb_node* x)* that does right rotation on node `x` in `tree`.
- *struct rb_node* rb_insert_fixup(struct rb_tree* tree, struct rb_node* n)* that *fixes* node `n` in `tree` after insertion to restore the red-black properties. Make sure your function covers all the cases mentioned in the lecture and their mirror cases.
- *void rb_insert(struct rb_tree* T, int key)* that inserts a new node with key value `k` into `tree` and then uses *rb_insert_fixup* to restore the red-black properties.

Test your implementation by performing the following operations:

- Initialize a red-back tree `T`;
- Insert 5, 90, 20 into `T`.
- Print the tree.
- Right rotate node 20.
- Left rotate node 5.
- Print the tree.
- Insert 60, 30 into `T`.
- Print the tree.
- Right rotate node 90.
- Print the tree.