Alexandria University
Faculty of Engineering Specialized
Scientific programs
Spring 2018

CC376 Data Structure 2
6th Term
Lab 2
Due 11th May 2018

Lab 2

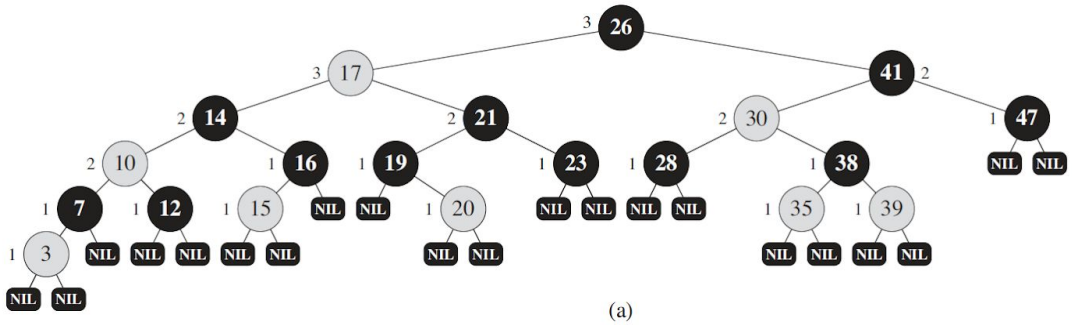
Implementing Red-Black Trees

1. Purpose

This lab assignment focuses on balanced binary search trees.

2. Background

A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either RED or BLACK. By constraining the node colors on any simple path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.



(a)

A red-black tree is a binary tree that satisfies the following red-black properties:

- Every node is either red or black.
- The root is black.
- Every leaf (NIL) is black.
- If a node is red, then both its children are black.
- For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

3 Requirements

3.1 Red-Black Tree Implementation

You are required to implement the Red-Black Tree data structure supporting the following operations:

1. Search:
Search for a specific element in a Red-Black Tree.

2. Insertion: Insert a new node in a Red-Black tree. Tree balance must be maintained via the rotation operations.

3. Deletions: Delete a node from a Red-Black tree. Tree balance must be maintained via the rotation operations.

4. Print Tree Height: Print the height of the Red-Black tree. This is the longest path from the root to a leaf-node. Please refer to the reference below for more implementation details.

3.2 Application: English Dictionary

As an application based on your Red-Black Tree implementation, you are required to implement a simple English dictionary, with a simple text-based user interface, supporting the following functionalities:

- **Load Dictionary**:
  - You will be provided with a text file, “dictionary.txt", containing a list of words. Each word will be in a separate line.
  - You should load the dictionary into a Red-Black Tree data structure to support efficient insertions, deletions and search operations.

- **Print Dictionary Size:**
  - Prints the current size of your dictionary.

- **Insert Word:**
  - Takes a word from the user and inserts it, only if it is not already in the dictionary. Otherwise, print the appropriate error message (e.g. “ERROR: Word already in the dictionary!").

- **Look-up a Word:**
  - Takes a word from the user and prints “YES" or “NO" according to whether it is found or not.

- **Remove Word:**
    - Takes a word from the user and removes it from the dictionary. If the word is not in the dictionary, then print the appropriate error message.

Note: For validation purposes, you are required to print both the size of the dictionary and the height of your Red-Black tree after each insertion or deletion.

4. References

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein “Introduction to Algorithms 3rd Edition - Thomas H. Cormen, Charles E. Leiserson, R”

Weiss, Mark Allen. “Data structures and algorithm analysis in Java." Addison-Wesley Long-man Publishing Co., Inc., 1998.

5. Notes

- Implement your algorithms using (Java or C/C++ or Python)
- **Objective is to work in group and be able to divide work so** You should work in groups **of 2 OR 3 members.**
- Send a copy from your implementation on 11th May 11:59pm and discussion to start from 12th.

Good Luck