CPSC 331: Assignment 2

Fall 2023

See the D2L site for due date/time.

1. [20%] Design a variant of the stack data structure, so that on top of the three primitives, PUSH, POP, and TOP, there is now a new primitive, MAX.

   - MAX(S): returns the maximum key currently in the stack S.

   It is assumed that the data type of the keys supports comparisons (i.e., "$\leq$"). Other than that, we do not know anything about the data type of the keys. Your solution is evaluated by how competitive its performance (running time) is.

2. [20%] Design a linear-time algorithm that converts a sorted, doubly linked list into a **balanced** binary search tree. The only thing we know about the data type of the keys is that they are comparable (i.e., "$\leq$"). You may further assume that the keys in the linked list are distinct. It is okay for the algorithm to be ***destructive*** (meaning it modifies the original linked list or even distroys it). It is also okay to use additional space (so long as the amount is reasonable).

3. [20%] Exercise 12.3-6 on page 326 of CLRS.

4. [40%] Implement a red-black tree class in Java.

   (a) The keys are of type **`int`**.

   (b) Your class shall be named **`RedBlackTree`**. It shall be declared in the package **`ca.ucalgary.cpsc331.a2`**. Submit the Java source file **`RedBlackTree.java`**. This is the only source file that you need to submit.

   (c) Your class shall implement the following Java interface:

```
package ca.ucalgary.cpsc331.a2;

public interface Dictionary {
    // empty(): returns true iff the dictionary
    // is empty.
    boolean empty();

    // insert(k): inserts k and returns true if k is
    // not already in the dictionary; otherwise
    // returns false without inserting k.
    boolean insert(int k);
```

```
    // delete(k): deletes k and returns true if k is
    // a member of the dictionary; otherwise returns
    // false without deleting k.
    boolean delete(int k);

    // member(k): returns true iff k is a member.
    boolean member(int k);
}
```

The file is available for download from D2L. You do not need to submit this file. Gradescope will compile your code against our version of this file.

(d) Tree nodes are represented as follows.

```
package ca.ucalgary.cpsc331.a2;

class Node {
    Node parent;
    Node left;
    Node right;
    boolean red; // true iff node is red
    int key;
    Node(int k) {
        key = k; red = false;
        parent = left = right = null;
    }
}
```

The file is available for download from D2L. You do not need to submit this file. Gradescope will compile your code against our version of this file.

(e) The class shall provide a constructor that does not take any arguments. This constructor initializes the tree to an empty one.

(f) The class shall override the **`toString()`** method. Your implementation of **`toString()`** shall return a **`String`** in the following format:

- The string consists of as many "lines" as there are internal nodes (i.e., a node that is not a leaf/NIL). If there are ten internal nodes, then there are ten lines. An empty tree is represented by an empty string.
- Each "line" is terminated by a newline character. No line shall contain leading or trailing whitespaces.
- Each line is a representation of an internal node. The order of the lines corresponds to a *preorder* tree walk.
- A line has the following format:

  `address:color:key`

  - **`key`** is the key stored in the node.

  - **`color`** is either **`red`** or **`black`**, representing the color of the node.
  - **`address`** is the ***tree address*** of the node, a concept explained below.
- The tree address of an internal node is defined inductively:
  - The tree address of the root is the string **`"*"`**.
  - If the tree address of an internal node n is the string s, then the tree addresses of the left and right children are respectively s **`+ "L"`** and s **`+ "R"`**. Here, **`+`** is the string concatenation operator.

  For example, the left child of the right child of the root has the following tree address:

  `*RL`