Second Exam
CS 1102 Computer Science 2

Fall 2018

Tuesday November 20, 2018
Instructor Muller

KEY

Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name **on the back** of this exam.

This is a closed-notes and closed-book exam. Computers, calculators, and books are prohibited.

- Partial credit will be given so be sure to show your work.
- Feel free to write helper functions if you need them.
- **Please write neatly.**

Problem	Points	Out Of
1		5
2		2
3		3
4		3
5		3
6		4
Total		20

Part 1: (5 Points) Binary Trees

The problems in this section relate to binary trees of the form

```
class Node {
  int key;
  Node left, right;
  public Node(int key, Node left, Node right) {
    this.key = key;
    this.left = left;
    this.right = right;
  }
}
```

A binary tree is either empty (in Java, this is normally represented by `null`) or it is a `Node` with a `key` field and `left` and `right` fields.

1. (1 Point) Write a function `boolean isLeaf(Node t)` such that a call `isLeaf(t)` returns `true` if `t` is a leaf. Otherwise `isLeaf` should return `false`.

   Answer:

   ```
   public boolean isLeaf(Node t) {
     return (t != null) && (t.left == null) && (t.right == null);
   }
   ```

2. (2 Points) Write a function `int height(Node t)` such that the call `height(t)` returns the integer height of `t`.

   Answer:

   ```
   public int height(Node t) {
     if (t == null || isLeaf(t)) return 0;
     else return 1 + Math.max(height(t.left), height(t.right));
   }
   ```

3. (2 Points) Write a function `boolean isBST(Node t)` such that a call `isBST(t)` returns `true` if `t` is a binary search tree. Otherwise `isBST` should return `false`.

Answer:

```
public boolean isBST(Node t) {
  if (t == null) return true;
  boolean tIsOK = allLess(t.left, t.key) && allGreater(t.right, t.key);
  return tIsOK && isBST(t.left) && isBST(t.right);
}

private boolean allLess(Node t, int key) {
  if (t == null) return true;
  return (t.key < key) && allLess(t.left, key) && allLess(t.right, key);
}

// Efficient solution due to Bob Dondero. (From SW BST.java)
//
boolean isBST(Node t) { return isBST(t, null, null); }

boolean isBST(Node t, Key min, Key max) {
  if (t == null) return true;
  if (min != null && t.key.compareTo(min) <= 0) return false;
  if (max != null && t.key.compareTo(max) >= 0) return false;
  return isBST(t.left, min, t.key) && isBST(t.right, t.key, max);
}
```

Part 2: (2 Points) Storage Diagrams

Using the class `Node` from the previous problem, consider the following code and show the state of the stack and heap after (1) has executed but before (2) has executed (i.e., before `g` returns). Execution is initiated with the call `f(3, 4)`.

```
int g(Node a, Node b) {
  int k = a.key;                          (1)
  return k;                               (2)
}
int f(int i, int j) {
  Node n1 = new Node(i, null, null);
  return g(n1, new Node(j, n1, n1));
}
```

```
           stack                                       heap
```

Answer:

```
+---+
| g |
+---+----+                      +----------+
|  b : o-+--------------------> | key : 4  |
|  a : o-+--+                   +----------+
|  k : 3 |  |                   | o  | o   |
+--------+  |                   +-|--+-|---+
            |                     |    |
+---+       |                     |    |
| f |       |                     |    |
+---+----+  |                     |    |
|  i : 3 |  |                     v    v
|  j : 4 |  +-----------------> +----------+
| n1 : o-+--------------------> | key : 3  |
+--------+                      +----------+
                                | o  | o   |
                                +-|--+-|---+
                                  =    =
```

Part 3: (3 Points) Sets and Relations

For the questions below let $A = \{a, b, c\}$, $B = \{\spadesuit, \clubsuit, \heartsuit\}$ and let

- $R_0 = \{\}$
- $R_1 = \{(a, a), (b, b), (c, c)\}$
- $R_2 = R_1 \cup \{(a, b)\}$
- $R_3 = \{(a, \clubsuit), (b, \clubsuit)\}$
- $R_4 = A \times A = \{(a, a), (b, b), (c, c), (a, b), (a, c), (b, a), (b, c), (c, a), (c, b)\}$
- $R_5 = \{(a, \clubsuit), (b, \heartsuit), (c, \clubsuit)\}$
- $R_6 = A \times B = \{(a, \spadesuit), (a, \clubsuit), (a, \heartsuit), (b, \spadesuit), (b, \clubsuit), (b, \heartsuit), (c, \spadesuit), (c, \clubsuit), (c, \heartsuit)\}$

The following are half-point problems.

1. List the reflexive relations on A. (E.g., R_8, R_{12} etc).

   Answer: R_1, R_2 and R_4.

2. List the transitive relations on A.

   Answer: R_0, R_1, R_2 and R_4.

3. List the total maps from A to B.

   Answer: Only R_5.

4. List the partial maps from A to B.

   Answer: R_0, R_3 and R_5.

5. List the equivalence relations on A.

   Answer: R_1 and R_4.

6. List the partial orders on A.

   Answer: R_1 and R_2.

Part 4: (3 Points) Hash Tables

1. (2 Points) In Java, every value of a reference type (i.e., created with `new`) comes equipped with an `equals` function `boolean equals(Object other)` and a `hashCode` function `int hashCode()`.

   (a) True or false: if `x.hashCode() == y.hashCode()` then it must be the case that `x.equals(y)`.

   Answer: False. It is common for unequal keys to hash to the same table location.

   (b) True or false: if `x.equals(y)` then it must be the case that `x.hashCode() == y.hashCode()`.

   Answer: True. If two keys are equal as judged by the `equals` equivalence relation, then they should hash to the same table location.

2. (1 Point) In a sentence or two, what are the advantages and disadvantages in resolving hash collisions using chaining?

   Answer: Chaining is simple to implement and has very fast `insert`. If the load factor of the table is large, the chains can get long and the `find` operation can take time proportional to the chain length. Chains also have poor locality.

Part 5: (3 Points) Red/Black Trees

Show **all** of the successive trees that result from the left-to-right insertion of the letters `IRELAND` into an empty *left-leaning Red/Black tree*.

Answer:

```
-   =I=>  I   =R=>  I     =RL=>   R    =E=>   R    =RR=>   I   =FL=> I
                     \\         //          //           // \\      / \
                       R       I           I            E     R    E   R
                                          //
                                         E

=L=>  I    =A=>    I    =N=>   I    =RL=>    I    =RR=>   I
     / \          / \         / \           / \          / \
    E   R        E   R       E   R         E   R        E   N
      //       //  //      //  //        //  //       //  // \\
     L        A   L       A    L        A   N        A   L     R
                                \\        //
                                  N      L

=FL=>  I      =RL=>     N    =D=>    N     =RL=>   N    =RR=>  N
      / \\            // \         // \          // \        // \
     E    N          I    R       I    R        I    R      I    R
   //    / \        / \          / \           / \         / \
  A     L   R      E   L        E   L         E   L       D   L
                 //           //            //          // \\
                A            A             D           A     E
                              \\         //
                                D       A

=FL=>   N     =RR=>    I        =FL=>     I
      // \           // \\              /   \
     I    R         D     N            D     N
   // \            / \   / \          / \   / \
  D    L          A   E L   R        A   E L   R
 / \
A   E
```

Part 6: (4 Points) Huffman Coding

A zip file contains the following frequency table and bit sequence.

```
+---+---+---+---+---+
| A | H | N | O | S |
+---+---+---+---+---+
| 1 | 1 | 3 | 1 | 1 |
+---+---+---+---+---+
111101100001100
```

The file was constructed with the same assumptions as in problem sets 7 and 8:

1. Letters are initially entered into the PQ in alphabetical order;
2. Ties are broken by placing the newly inserted entry *behind* all entries with the same priority;
3. In a Huffman Tree traversal, left means 0 and right means 1.

What is the uncompressed text? Please show all of your work.

Answer:

```
PQ:

A1 H1 O1 S1 N3

  2     2   N3
 / \   / \
A   H O   S

      7
    /   \
   N     4
       /   \
      2     2
     / \   / \
    A   H O   S

111 101 100 0 0 110 0
 S   H   A  N N  O  N
```