Red- Black Tree Reading

Updated ArrayListIterator

Before:
AListIterator() {
    this.current = 0;
    active = active + 1; //because another active
    //iterator is started
}

After:
AListIterator() {
    this.current = 0;
    if (this.hasNext()) {
        active = active + 1; //because another active
        //iterator is started
    }
}
Assignments 2 and 3
Assignment 4

• Due tonight!
Abstraction Function

[Liskov]

This relationship can be defined by a function called the abstraction function that maps from the instance variables that make up the rep of an object to the abstract object being represented:

\[ \text{AF} : \text{C} \rightarrow \text{A} \]

Specifically, the abstraction function AF males from a concrete state (i.e., the state of an object of the class C) to an abstract state (i.e., the state of an abstract object). For each object \( c \) belonging to \( \text{C} \), \( \text{AF}(c) \) is the state of the abstract object \( a \in \text{A} \) that \( c \) represents.
IntSet Abstraction Function

\[ \text{AF}(c) = \{c.\text{els}[i].\text{intValue} \mid 0 \leq i < c.\text{els}.\text{size}\} \]
Mutable Queue
Abstraction Function

// A typical Queue of integers is
// \{k_0, k_1, \ldots, k_n\}
// with k_0 as the first element added, and
// k_n as the last element added
//
// The abstraction function is
// AF(queue) =
// \{ queue.q[0] = k_0, queue.q[q.size() - 1] = k_n |
// for queue = \{k_0, k_1, \ldots, k_n\}\}
Mutable Queue Abstraction Function

// A typical Queue of integers is
// {k0, k1, ..., kn}
// with k0 as the first element added, and
// kn as the last element added
//
// The abstraction function is
// AF(queue) =
//   { queue.q[0] = k0, queue.q[q.size() - 1] = kn | 
//     for queue = {k0, k1, ..., kn}}

// For a queue created by adding elements k0, k1, k2, ...
// in this order
// the abstraction function is
// AF(queue) = {k0, k1, k2, ...}
// where queue.q[i] = ki for 0 <= i < queue.q.size()
Rep Invariant

A statement of a property that all legitimate objects satisfy is called a *representation invariant*, or *rep invariant*. A rep invariant $I$ is a predicate

$$I : C \rightarrow \text{Boolean}$$

that is true of legitimate objects.
IntSet Rep Invariant

// The rep invariant is:
// c.els != null &&
// all elements of c.els are Integers &&
// there are no duplicates in c.els
Mutable Queue Implemented as ArrayList Rep Invariant

// The rep invariant is:
// c.q != null &&
// all elements of c.q are Integers
equals and hashCode
Functional/Declarative vs. Procedural/Imperative
Functional Iterator
StackInt Signature

Public Static Methods:
empty:                           --> StackInt
push: StackInt x int            --> StackInt
isEmpty: StackInt               --> boolean
top: StackInt                   --> int
pop: StackInt                   --> StackInt
size: StackInt                  --> int

Public Dynamic Method:
traversal:                      --> Traversal<Integer>
StackInt

Algebraic Specifications

isEmpty (empty()) = true
isEmpty (push (s, n)) = false

top (push (s, n)) = n

pop (push (s, n)) = s

size (empty()) = 0
size (push (s, n)) = 1 + size (s)

empty().traversal().isEmpty() = true
push (s, n).traversal().isEmpty() = false

push (s, n).traversal().getFirst() = n

push (s, n).traversal().getRest() = s.traversal()
Total Order
Total Order

A total order on some set $D$ is a binary relation $R$ on $D$ such that

- $R$ is transitive
- $R$ is anti-symmetric
- $R$ satisfies the law of trichotomy
Total Order

A total order on some set $D$ is a binary relation $R$ on $D$ such that

- $R$ is **transitive**: if $xRy$ and $yRz$, then $xRz$
- $R$ is anti-symmetric
- $R$ satisfies the law of trichotomy
Total Order

A total order on some set $D$ is a binary relation $R$ on $D$ such that

- $R$ is transitive
- $R$ is anti-symmetric: if $xRy$ and $yRx$, then $x = y$
- $R$ satisfies the law of trichotomy
Total Order

A total order on some set $D$ is a binary relation $R$ on $D$ such that

- $R$ is transitive
- $R$ is anti-symmetric
- $R$ satisfies the law of trichotomy: $\forall x, \forall y$ either $xRy$ or $yRx$

The law of trichotomy (a division into three categories) can also be phrased as $\forall x, \forall y$ either

\[ x = y \]

\[ \text{or} (x \neq y \text{ and } xRy) \]

\[ \text{or} (x \neq y \text{ and } yRx) \]
Examples of Total Orders
Usual Ordering on Integers

\[(R :\leq)\]

- \(R\) is transitive
- \(R\) is anti-symmetric
- \(R\) satisfies the law of trichotomy
Usual Ordering on Integers
(R :\leq)

• R is transitive: if xRy and yRz, then xRz
  - x=1, y=2, z=3
  - 1\leq2 and 2\leq3, then 1\leq3 (TRUE)
• R is anti-symmetric
• R satisfies the law of trichotomy
Usual Ordering on Integers

\((R :\leq)\)

- \(R\) is transitive
- \(R\) is anti-symmetric: if \(xRy\) and \(yRx\), then \(x = y\)
  - \(x=1, y=1\)
  - \(1\leq1\) and \(1\leq1\), then \(1=1\) (TRUE)
- \(R\) satisfies the law of trichotomy
Usual Ordering on Integers
(R :<=)

- R is transitive
- R is anti-symmetric
- R satisfies the law of trichotomy
  - \( \forall x, \forall y \) either \( xRy \) or \( yRx \) (or \( x = y \))
  - The law of trichotomy (a division into three categories) can also be phrased as \( \forall x, \forall y \) either
    - \( x = y \) or \( x \neq y \) and \( xRy \) or \( x \neq y \) and \( yRx \)
  - \( x=1, y=2 \) --- \( xRy \)
  - \( x=3, y=2 \) --- \( yRx \)
  - \( x=1, y=1 \) --- \( x=y \)
Trees
Binary Trees
Labeled Binary Tree (LBT)

- an empty tree
- a node with three components:
  - a label
  - a left subtree, which is a labeled binary tree
  - a right subtree, which is a labeled binary tree
Binary Search Trees
Binary Search Tree (BST)

- \( t \) is empty
- \( t \) is a node
  - a label
  - the left subtree of \( t \) is a BST,
  - the right subtree of \( t \) is a BST,
  - every label within the left subtree of \( t \) is less than the label of \( t \),
  - every label within the right subtree of \( t \) is greater than the label of \( t \)
O-notation

A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large $n$, i.e., if there exist some positive constant $c$ and some nonnegative integer $n_0$ such that $t(n) \leq c \times g(n)$ for all $n \geq n_0$.

(from Levitin)
In-Class Exercise

• $t(n) = n$
• $t(n) = 100n + 5$
• $t(n) = 0.5 \times n(n-1)$
Sum of Integers from 1 to n
Sum of Integers from $1$ to $n$

\[ \sum_{i=1}^{n} i \]
Sum of Integers from 1 to n

\[ \sum_{i=1}^{n} i \in O(n^2) \]
$0$-notation

More general definition:

If $g : X \to R$, then

$O(g) = \{ f : X \to R \mid \exists c_0, c_1 > 0, \forall x \in X, f(x) \leq c_0 + c_1 \cdot g(x) \}$

(from Clinger)
Ω-notation

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large $n$, i.e., if there exist some positive constant $c$ and some nonnegative integer $n_0$ such that $t(n) \geq c \times g(n)$ for all $n \geq n_0$.

(from Levitin)
Θ-notation

A function \( t(n) \) is said to be in \( \Theta(g(n)) \), denoted \( t(n) \in \Theta(g(n)) \), if \( t(n) \) is bounded both above and below by some positive constant multiples of \( g(n) \) for all large \( n \), i.e., if there exist some positive constant \( c_1 \) and \( c_2 \) and some nonnegative integer \( n_0 \) such that \( c_2 \cdot g(n) \leq t(n) \leq c_1 \cdot g(n) \) for all \( n \geq n_0 \).

(from Levitin)