BagsandDenseTreesProject

Generated by Doxygen 1.8.7

Wed Oct 28 2015 20:56:53

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Iterable

- **Bag< T >** **2**
- **DenseSearchTree< T extends Comparable< T > >** **4**

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Bag< T > **2**

DenseSearchTree< T extends Comparable< T > > **4**

3 Class Documentation

3.1 Bag< T > Class Reference

Inheritance diagram for Bag< T >:

Collaboration diagram for Bag< T >:

Classes

- class **_Iterator**

Public Member Functions

- Bag ()
- void add (T ele)
- void remove (T ele)
- boolean contains (T ele)
- int count (T ele)
- Set< T > asSet ()
- int size ()
- Iterator< T > iterator ()

3.1.1 Detailed Description

A "bag" is a data-structure that implements a *multiset*, which is a set that allows multiple (duplicates). The standard operators on such a data-structure include:

- Bag() –creates an empty bag (optionally, perhaps a default size?)
- add(T ele) –adds an element to the bag.
- remove(T ele) –removes an element from the bag; may throw IllegalStateException –in the event that ele is NOT in the Bag.
- contains(T ele) –returns True if ele is in bag.
- count(T ele) –returns the number of occurrences of ele in set. This is called the "multiplicity" of the element.
- asSet() –returns a collection of the elements in this bag as set.
- size() –returns the number of elements in this bag taking into account their multiplicities. For example: if the Bag contains the following elements with multiplicities:

  Key Count

  ----------—

  "A" 2

  "B" 3

  "C" 1

  size() => 6.
- iterator() –returns an Iterator that also takes into account the multiplicities of each key. So, an iterator() should iterate over exactly the same number of elements as size().

Author

UMD CS Department.

Parameters

<T>	

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Bag ( )

Default (sole) constructor. Sets up internal data structures.

3.1.3 Member Function Documentation

3.1.3.1 void add (T *ele*)

Adds `ele (T)` to the bag. If ele was already in the Bag, then its internal count is incremented by 1; otherwise a new entry is made, indicating that at least 1 ele exists in this Bag.

Parameters

ele	

3.1.3.2 Set<T> asSet ( )

Return the keys as a Set—i.e., no duplicates, order-unimportant. Use this function to determine the number of unique entries in the Bag!

Returns

3.1.3.3 boolean contains (T *ele*)

Returns true if this Bag contains at least one copy of the `key`.

Parameters

ele	

Returns

3.1.3.4 int count (T *ele*)

Returns the number of occurrences of `ele` in the Bag; returns 0 in the event that `ele` is not in Bag.

- **Parameters**

ele	

  Returns

3.1.3.5 Iterator$<$T$>$ iterator ( )

Returns an object that iterates (implements the Iterator interface) over objects in the Bag.

You will most like return an instance of an inner class that you design to manage an *immutable* Iteration of keys over this bag.

Returns

3.1.3.6 void remove (T *ele*)

Effectively *remove* the `ele` from the bag. Note, several possibilities here:

- If the `ele` isn't in the Bag, throw an Illegal State Exception.
- If only one `ele` exists in the Bag, then remove it.
- If more than one `ele` exists in the Bag, make whatever changes are necessary to indicate that one fewer `ele` now exists in the Bag. Note, how you do this will depend upon your internal data-structure.

Parameters

ele	

3.1.3.7 int size ( )

The "size" of a bag is the cardinality of the multiset that it embodies. In English, that means that the size of your bag must accumulate the multiplicities of each element. For example: suppose your Bag contains

Key Count

----------—

"A" 2

"B" 3

"C" 1

Then its size is 6.

Returns

The documentation for this class was generated from the following file:

- src/student_classes/Bag.java

3.2 DenseSearchTree$<$ T extends Comparable$<$ T $>$ $>$ Class Reference

Inheritance diagram for DenseSearchTree$<$ T extends Comparable$<$ T $>$ $>$:

Collaboration diagram for DenseSearchTree$<$ T extends Comparable$<$ T $>$ $>$:

Classes

- class **_Iterator**
- class **TNode**

Public Member Functions

- void add (T element)
- boolean contains (T target)
- int count (T target)
- int size ()
- Set< T > **asSet** ()
- void remove (T target)
- T getMin ()
- T getMax ()
- String toString ()
- Iterator< T > iterator ()

3.2.1 Detailed Description

Non-Standard interpretation of a Binary Search Tree data-structure. Rather than inserting duplicate values along the left or right branches, this version maintains a count (int) (defaults to 1) that indicates how many "copies" of the key are available on this node. Should the count go to 0, then the node is physically removed from the tree. This simplifies our remove logic in that we don't have to account for duplicate keys.

Author

UMD CS Dept.

Parameters

<T>	

3.2.2 Member Function Documentation

3.2.2.1 void add (T *element*)

Note: our trees follow a slightly different convention regarding both our ordering relation and the placement of duplicates, viz:

- Left branch contains all elements < tree.value;
- Right branch contains all elements >= tree.value.

Rather than place duplicate elements on the right branch, each Tree Node maintains a count of copies. For example, if we had a tree of Integers containing two instances of the number 1, our Tree Node would logically appear as

[left-branch 1:2 right-branch],

where 1:2 means 2 copies of the integer 1.

Thus, your add logic will either make a new node and insert it in the correct position in the tree, or it will find a node with the value (key) equal to the value you are adding, and increment its count.

This simplifies your remove logic: if the node's count is $>$ 0, then decrement by 1, otherwise invoke the remove logic to physically remove the node and replace it with a Binary Search Tree.

Finally, the "price you pay" for this simplification is that your Iterator (which presents an "in-order" view of the tree), needs to "inflate" or "expand" each node. That is, if you have a node

[left 3:4 right]

then 4 instances of the integer 3 need to be created and put into the iteration.

Parameters

element	

3.2.2.2 boolean contains (T *target*)

Returns true if at least instance of `target` is found in tree.

Parameters

target	

Returns

3.2.2.3 int count (T *target*)

Returns an int $>=$ 0, indicating how many occurrences of `target` reside in the tree. Note, this function returns 0 when the item is *not* found in tree.

Parameters

target	
tree	

Returns

3.2.2.4 T getMax ( )

Returns the max value (of type T), or throws an `IllegalStateException` if this function is called on an empty tree.

Returns

3.2.2.5 T getMin ( )

Returns a value of type T or throws an `IllegalStateException` if this function is called on an empty tree.

Returns

3.2.2.6 Iterator<T> iterator ( )

Returns an iterator over the DenseSearchTree.

Note, this iterator must present an "in order" view of the keys in the tree.

3.2.2.7 void remove (T *target*)

Somewhat streamlined or simplified version of the classic BST remove algorithm. Because we're maintaining counts of keys on each node, many times this method find the node whose value (key) equals the `target` and decrements the counter by 1. If that would result in the counter going to 0, however, then the remove logic finds the greatest in order successor and replaces the node to be removed with that, and then continues to remove the in order successor whose value was used to re-label the node to be removed.

Parameters

target	

3.2.2.8 int size ( )

Returns "inflated" size of tree, meaning a count of all keys in the tree. > Returns the set representation of this Tree. In this case, the set will contain unique elements (i.e., it should omit multiple instances) that comprise the tree.

Note: sets are *unordered* collections, but trees are *partial orderings*. This means, among other things, that your set *may* reflect an internal ordering, such as pre-, in- or post-ordering, but it need not.

Returns

3.2.2.9 String toString ( )

For sanity's sake ... please feel free to implement whatever makes sense to you here ... this method is NOT tested.

The documentation for this class was generated from the following file:

- src/student_classes/DenseSearchTree.java

Index