**TreePriorityQueue**

In some applications of priority queues, each data value can be added to the priority queue multiple times. In this case, time and space can be saved by storing each data value only once, with a count field to keep track of the number of times that the data value is in the priority queue.

Class TreePriorityQueue is implemented using a binary search tree. Each TreeNode of the binary search tree consists of an Item object and the links to the left and right subtrees. An Item object contains a Comparable data value and a count of the number of occurrences of the data. When the count goes to zero as a result of a deletion, the Item object is removed from the tree.

The declarations for the TreePriorityQueue class and the Item class are shown below.

**public class** TreePriorityQueue **implements** PriorityQueue

{

**private** TreeNode root;

**public** TreePriorityQueue()

{ root = **null**; }

//postcondition: returns true if the priority queue is empty;

// otherwise, returns false

**public** **boolean** isEmpty()

{ /\* implementation not shown \*/ }

//postcondition: obj has been added to the priority queue

**public void** add(Object obj)

{ root = addHelper((Comparable)obj, root); }

//postcondition: obj has been added to the subtree rooted at t;

// the updated subtree is returned

**private TreeNode** addHelper(Comparable obj, TreeNode t)

{ /\* to be implemented in part (b) \*/ }

//precondition: the priority queue is not empty

//postcondition: the data value of highest priority (smallest

value) has been removed and returned

**public** Object removeMin()

{ /\* implementation not shown \*/ }

//precondition: the priority queue is not empty

//postcondition: returns the data value of highest priority

(smallest value)   
 // the priority queue is unchanged

**public** Object peekMin()

{ /\* to be implemented in part (a) \*/ }

}

**public** **class** Item

{

**private** Comparable data;

**private int** count;

**public** Item(Comparable d)

{ data = d; count = 1; }

**public void** incrementCount()

{ count++; }

//precondition: the count of this item is greater than 0;

**public void** decrementCount()

{ count--; }

**public int** getCount()

{ **return** count; }

**public** Comparable getData()

{ **return** data; }

}

For example, if the items 13, 11, 14, 11, 15, 14, and 14 are added to an initially empty TreePriorityQueue, then the resulting binary search tree is as shown below (with 11 occurring two times, 14 occurring three times, and 13 and 15 each occurring one time.)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 13 | 1 |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 11 | 2 |  |  | 14 | 3 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 15 | 1 |

(a) Write the TreePriorityQueue method peekMin. Method peekMin returns the minimum data value in the priority queue.

Complete method peekMin below.

//precondition: the priority queue is not empty

//postcondition: returns the data value of highest priority

(smallest value)   
// the priority queue is unchanged

**public** Object peekMin()

{

(b) Write the TreePriorityQueue private method addHelper, which adds obj to the subtree rooted at t. If a node containing an Item with data obj is already in the subtree, addHelper increments the count; otherwise, addHelper adds a new node containing an Item with data obj and a count of 1 to the subtree, maintaining the binary search tree property. Method addHelper returns a reference to the root of the resulting subtree.

Complete method addHelper below.

//postcondition: obj has been added to the subtree rooted at t;

// the updated subtree is returned

**private** TreeNode addHelper(Comparable obj, TreeNode t)

{

To turn this AP test question into a lab, finish the TreePriorityQueue\_Driver methods peekMin, addHelper, and removeMin. Add a toString method to Item. Copy the display methods from TreeLab. Here is a sample run for the example array

int[] array = {13,11,14,11,15,14,14};

[15, 1]  
 [14, 3]  
[13, 1]  
 [11, 2]  
  
Peek min: 11  
Removing  
 [11, 1]  
 [11, 0]  
 [13, 0]  
 [14, 2]  
 [14, 1]  
 [14, 0]  
 [15, 0]