Hemuppgift 5- Introduktion till Datalogi

Peter Boström

2009-02-16

Innehåll

1 Tidskomplexitet

Operation	worst-case BST (egen implementation)	Average för Treap
add(o)	$\mathcal{O}(n)$	$\mathcal{O}(logn)$
contains(o)	$\mathcal{O}(n)$	$\mathcal{O}(logn)$
size()	$\mathcal{O}(n)$ sparas inte i add pga iterativ	$\mathcal{O}(n)$
height()	$\mathcal{O}(n)$	$\mathcal{O}(n)$
leaves()	$\mathcal{O}(n)$	$\mathcal{O}(n)$
toString()	$\mathcal{O}(n)$	$\mathcal{O}(n)$

A Källkod

A.1 BinarySearchTree.java

```
1  /**
2   * ÄÖÄBINRSKTRDSMONSTERMONSTER
3   * @author lemming
4   */
5  public class BinarySearchTree<T extends Comparable<? super T>> {
6      private T value;
7      private BinarySearchTree<T> less, more;
8
9      /**
10      * Constructs a bst with specified value for root node.
11      * @param value Value for the (root) node.
12      */
13     public BinarySearchTree(T value){
14         this.value = value;
15     }
16
17     /**
18      * Add an object to the tree.
19      * @param object Object to be added.
20      * @return Returns true if the object was added, or false if it already
            existed.
21      */
22     public boolean add(T object){
23         BinarySearchTree<T> currentNode = this;
24
25         while(true){
26             int diff = object.compareTo(currentNode.value);
27             if(diff == 0)
28                 return false;
29             if(diff < 0){
30                 if(currentNode.less != null){
31                     currentNode = currentNode.less;
32                     continue;
33                 }
34                 currentNode.less = new BinarySearchTree<T>(object);
35                 return true;
36             }
37             if(currentNode.more != null){
38                 currentNode = currentNode.more;
39                 continue;
40             }
41             currentNode.more = new BinarySearchTree<T>(object);
42             return true;
43         }
44     }
45
46     /**
47      * Check for object is contained in tree.
48      * @param object Value to be looked for
49      * @return Returns true if the object was found inside the tree.
50      */
51     public boolean contains(T object){
```

```
52          BinarySearchTree<T> currentNode = this;
53
54          while(true){
55              int diff = object.compareTo(currentNode.value);
56              if(diff == 0)
57                  return true;
58              if(diff < 0){
59                  if(currentNode.less != null){
60                      currentNode = currentNode.less;
61                      continue;
62                  }
63              }
64              else if(currentNode.more != null){
65                  currentNode = currentNode.more;
66                  continue;
67              }
68
69              return false;
70          }
71      }
72
73      /**
74       * Get size of tree.
75       * @return Number of elements in tree.
76       */
77      public int size(){
78          if(less == null && more == null)
79              return 1;
80          if(less == null)
81              return more.size()+1;
82          if(more == null)
83              return less.size()+1;
84          return less.size() + more.size() + 1;
85      }
86
87      /**
88       * Calculates tree height.
89       * @return How tall the tree is.
90       */
91      public int height(){
92          return internal_height(this);
93      }
94
95      /**
96       * Internally used to calculate tree height.
97       * @param current Node to calculate from.
98       * @return height from current node.
99       */
100     private int internal_height(BinarySearchTree<T> current){
101         if(current == null)
102             return -1;
103         return 1 + Math.max(internal_height(current.less),
                internal_height(current.more));
104     }
105
106     /**
```

```
107         * Counts amount of leaves in tree.
108         * @return Amount of leaves.
109         */
110        public int leaves(){
111            if(less == null && more == null)
112                return 1;
113            if(less == null)
114                return more.leaves();
115            if(more == null)
116                return less.leaves();
117            return less.leaves() + more.leaves();
118        }
119
120        /**
121         * Returns string representation of tree.
122         * @return The tree as a string.
123         */
124        public String toString(){
125            String string = new String();
126            if(less != null)
127                string += less.toString() + ' ';
128            string += value.toString() + ' ';
129            if(more != null)
130                string += more.toString();
131            return string.trim();
132        }
133    }
```

A.2 BinarySearchTreeTest.java

```
1   import junit.framework.TestCase;
2
3   /**
4    * ÄJTTESERIER
5    * @author lemming
6    */
7   public class BinarySearchTreeTest extends TestCase {
8       private BinarySearchTree<Integer> tree;
9
10      public void testAdd(){
11          tree = new BinarySearchTree<Integer>(1);
12          assertTrue(tree.add(5));
13          assertTrue(tree.add(3));
14          assertTrue(tree.add(2));
15          assertTrue(tree.add(4));
16          assertTrue(!tree.add(2));
17      }
18
19      public void testSize() {
20          tree = new BinarySearchTree<Integer>(6);
21          tree.add(1);
22          tree.add(2);
23          tree.add(8);
24          tree.add(1); //dublett
25          tree.add(5);
26          tree.add(7);
```

```
27
28          assertEquals(tree.size(), 6);
29
30          tree.add(9);
31
32          assertEquals(tree.size(), 7);
33      }
34
35      public void testContains(){
36          tree = new BinarySearchTree<Integer>(5);
37          assertTrue(tree.contains(5));
38          assertTrue(!tree.contains(2));
39
40          tree.add(2);
41          assertTrue(tree.contains(2));
42
43          tree.add(4);
44          tree.add(1);
45          tree.add(42);
46          tree.add(13);
47          tree.add(5);
48          assertTrue(tree.contains(5));
49          assertTrue(tree.contains(13));
50          assertTrue(!tree.contains(0));
51      }
52
53      public void testHeight(){
54          tree = new BinarySearchTree<Integer>(5);
55          assertEquals(tree.height(), 0);
56
57          tree.add(4);
58          tree.add(3);
59          tree.add(7);
60          tree.add(6);
61          tree.add(5);
62          tree.add(8);
63          tree.add(10);
64          tree.add(9);
65          tree.add(11);
66
67          assertEquals(tree.height(), 4);
68      }
69
70      public void testLeaves(){
71          tree = new BinarySearchTree<Integer>(5);
72          assertEquals(tree.leaves(), 1);
73
74          tree.add(4);
75          tree.add(3);
76          tree.add(7);
77          tree.add(6);
78          tree.add(5);
79          tree.add(8);
80          tree.add(10);
81          tree.add(9);
82          tree.add(11);
```

```
83
84          assertEquals(tree.leaves(), 4);
85      }
86
87      public void testToString(){
88          tree = new BinarySearchTree<Integer>(5);
89          assertTrue(tree.toString().equals("5"));
90
91          tree.add(4);
92          tree.add(3);
93          tree.add(7);
94          tree.add(6);
95          tree.add(5);
96          tree.add(8);
97          tree.add(10);
98          tree.add(9);
99          tree.add(11);
100
101         assertTrue(tree.toString().equals("3 4 5 6 7 8 9 10 11"));
102     }
103 }
```