

CPT_S 580 HW3

Yang Zhang

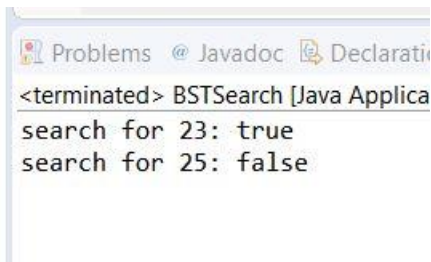
11529139 (graduate)

3.4.17

The problem is asking to implement the binary search tree search algorithm. By implementing the algorithm, we need firstly test the root of the tree, if is not the answer:

1. If the target $>$ root.data we recursively exam the right sub tree.
2. If the target \leq root.data we recursively exam the left sub tree.
3. If we reach the left and its data not equal to the target, then there is no such value in this tree.

The result:



```
<terminated> BSTSearch [Java Applica
search for 23: true
search for 25: false
```

From the result, the algorithm could determine if the target is in the tree correctly.

3.5.10

The problem is asking to implement the Huffman algorithm. By implementing the algorithm, we need firstly construct the Huffman tree:

1. Convert the data with its frequency into a Huffman tree node.
2. Put all those nodes into a priority queue (sorting based on its the frequency)
3. While the queue size is bigger than one, each time pop out two nodes from the priority queue, and use them to form a new node with frequency equals to the sum of the two nodes.
4. When queue size is 1, pop out the node as the tree root

To get the Huffman code:

Traverse the Huffman tree, record 0 when checking the left sub tree (record 1 for right sub tree), until reach the leaf node.

The result:

```
<terminated> TestMain [Java Application] C:\Prograr
exercise 3.5.4
a : 00
b : 1000
c : 011
d : 010
e : 111
f : 110
g : 101
h : 1001

exercise 3.5.5
a : 00
b : 1000
c : 011
d : 010
e : 111
f : 110
g : 101
h : 1001

exercise 3.5.6
a : 00
b : 1000
c : 011
d : 010
e : 111
f : 110
g : 101
h : 1001
```

According to the result, we can see that the Huffman algorithm always put the node with bigger frequency at the front.