目录

Week 10 Exercises

BSTheap.txt

For each of the following trees, state whether it could be a BST or a heap

```
a)    f           b)     f           c)      96
     /  \                /  \                  /    \
    c    h             c    e             54     27
   /\   /             /\    /             / \    /
  a  e g             b  b d             13 53  26

d)    q           e)    b           f)     b
     /                 /                  / \
    m                 a                  a   b
   /\
  f  a
```

BSTinsertion.txt

a. Insert the following keys *10 20 5 30 15 25 24* into a BST using *at-the-leaf* insertion.
b. What is the height of the resulting tree?
c. Show the resulting trees after deleting *5*, then *30*, and finally *20*. Show the result for both DLMD and DRMD when appropriate.
d. What are the heights of the resulting trees?

BSTsum.txt

a. Write an algorithm to recursively sum the elements of a BST rooted at *Tree*, where a *Tree* contains *data*, and pointers to its sub-trees *left* and *right*.
b. Convert your algorithm into C code by writing a function with signature:

切换行号显示

```c
int sumTree(Tree t)
```

In the next question you can implement and test your function.

treet.c

In the BST tree lecture you saw a basic BST program **basic.c** that has hard-coded input and generates simple output (in effect, a flattened tree).

Cut&paste that code, call it **treet.c**, and do the following:

i. change the input to the command line
ii. change the output to show the BST 'on-its-side' (code is in the lecture notes)
iii. include the function *int sumTree(Tree t)* from the previous exercise
    - print the sum of the nodes in the tree

A sample execution is the following:

```
prompt$ ./treet 4 5 3 6 2 7 1 8 0
                                0
                        1
                2
        3
4
        5
                6
                        7
                                8
Sum = 36
```

Notice that with this printing function, the BST *left-to-right* ordering is *top-to-bottom* because the tree is 'lying on its side'.

treetre.c

You could also define a BST to have the reverse ordering: that is the left child node is larger than or equal to the parent and the right child node is smaller.

- Add a function

  切换行号显示

  ```
  Tree revTree(t)
  ```

  that takes a tree as argument and generates a copy of that tree but with all the children reversed, resulting in a reversed BST.
- Using *treet.c* as starting point, call this function and print the original tree and reversed tree with an appropriate message.
- Do not forget to 'clean up'.

Below is a sample execution.

```
prompt$ ./treetre 4 5 3 6 2 7 1 8 0
```

```
                                0
                        1
                2
        3
4
        5
                6
                        7
                                8
Sum = 36
Tree in reverse
                                8
                        7
                6
        5
4
        3
                2
                        1
                                0
Sum = 36
```

If there is no input, then the program should do nothing:

```
prompt$ treetre
prompt$
```

When writing this function, think carefully about the best way of *copying* a node and *reversing* children.

BSTrootleafinsert.txt

Consider an initially empty BST and the sequence of values *1 2 3 4 5 6*.

a. Show the tree resulting from inserting these values "at the leaf". What is its height?
b. Show the tree resulting from inserting these values "at the root". What is its height?
c. Show the tree resulting from an alternate at-root-insertion and at-leaf-insertion, starting with at-the-leaf. What is its height?

Splaying.txt

a. Given an initially empty splay tree:
    ○ show the changes in the splay tree after each splay-insertion of the node values *5 3 8 7 4*
b. Given an initially empty splay tree:
    i. show the changes in the splay tree after each splay-insertion of the node values *b c d e f g* ...
    ii. and then the splay tree after a search for node *a* ...
    iii. and then the splay tree after a search for node *d*

Week10Exercises (2019-08-12 15:15:35由AlbertNymeyer编辑)