CS29003 Algorithms Laboratory

Lab Test

Date: 26–March–2019

EVEN PC

Let T be a binary search tree (BST) storing n integer-valued keys k_i satisfying $0 < k_1 < k_2 < k_3 < \cdots < k_n$. Two consecutive keys in T are called a *neighbor pair*, that is, T contains $n-1$ neighbor pairs (k_i, k_{i+1}) for $i = 1, 2, 3, \ldots, n-1$. The *distance* of the neighbors k_i and k_{i+1} is $k_{i+1} - k_i$. Consider the two problems.

(a) Find the closest neighbor pair in T (with respect to the distance defined above). If multiple neighbor pairs have the same distance as the closest one, finding any one of these pairs suffices.

(b) If the closest pair is (x, y), apply a sequence of rotations to T until x becomes the left child of y.

Problem (b) is solved if you make T completely left-skew, but this process may require about n rotations. You need to solve Problem (b) by a restricted number of rotations. Let $level(u)$ denote the level of the node u in T, that is, the distance of u from the root of T. For example, the root itself is at level 0, its children are at level 1, the grandchildren of the root are at level 2, and so on. If (x, y) is the closest pair found by solving Problem (a), you are allowed to make a maximum of $|level(x) - level(y)|$ rotations. Note that the *only* operation permitted for solving Problem (b) is rotation (both left and right rotations are allowed). Note also that you do not need to compute $level(x)$ and $level(y)$ explicitly.

Each node in T should consist of an integer key value and two child pointers. Since rotations are involved, a node is additionally allowed to have a *parent pointer*. A node must not contain any other field. Implement the following parts in order to solve the two problems mentioned above.

Part 1: No black box is provided for constructing T. Write a function *insert*( ) to implement the *standard BST insertion* algorithm. For inserting a key k to T, first search for k in T. If k is found, do nothing. Otherwise, the search fails when you follow a NULL pointer. Create a new node storing k, and replace the NULL pointer you encountered during the search by a pointer to the new node. Adjust the parent pointer of the new node if your node contains this pointer. Do not perform any type of height balancing. **(5)**

Part 2: Write a function *preorder*( ) to print the preorder listing of the keys stored in T. **(3)**

Part 3: Write a function *findnbr*( ) for finding the closest neighbor pair (x, y). In order to do so, modify the inorder traversal procedure in T. This traversal prints the keys of T in sorted order. Instead of printing, you should find the minimum distance between neighbors. Your function should run in $O(n)$ time. You are allowed (but not forced) to use an external array to store the sorted list of keys of T. **(8)**

Part 4: Write two functions *lrotate*( ) and *rrotate*( ) to perform a left or right rotation at a specified node u. Each function should return a pointer to the new node v that occupies the position of u after the rotation. Your functions should change the parent pointers of the affected nodes (if you use these pointers). **(4)**

Part 5: Write a function *makechild*( ) to make x the left child of y, where (x, y) is the pair obtained in Part 3. This function should run in $O(|level(x) - level(y)|)$ time, and achieve its desired goal only by applying rotations to the nodes of T. Recall that the maximum number of rotations allowed is $|level(x) - level(y)|$. (**Hint:** Here, x is the immediate predecessor of y, and y is the immediate successor of x.) **(8)**

The *main*() function and output

- Initialize T to an empty tree. Read n, and n keys from the user. Insert in T the keys using the function of Part 1. Print the preorder listing of the keys of T after all the n keys are inserted. **(2 + 2)**
- Call *findnbr* of Part 3 to locate the closest neighbor pair (x, y) in T. Print the keys at x and y. **(2 + 2)**
- Call *makechild* of Part 5 to make x the left child of y. Print how many rotations your function applies on nodes of T. Print also the two children of y, and finally the preorder listing of the keys in T. **(2 + 2)**

Submit a single C/C++ source file. Do not use global/static variables.

Sample outputs

```
--------------------------------------------------------------------------------

n = 40

Keys to insert:
2076  1228  2585  1905  1823  2372  1348  2255  3209  2940   675   493
1434  2427  3148   328   194   385  1639  2805  1089   805   258  3300
3624   899  3014  2502   636  1526  2031  2128  1688  2178  3471  3367
1788  3512  2683  1045

+++ Preorder traversal of initial tree
     2076  1228   675   493   328   194   258   385   636  1089   805   899
     1045  1905  1823  1348  1434  1639  1526  1688  1788  2031  2585  2372
     2255  2128  2178  2427  2502  3209  2940  2805  2683  3148  3014  3300
     3624  3471  3367  3512

+++ Finding closest neighbor
    x = 1788, y = 1823, gap = 35

+++ Bringing neighboring key to child position
    Number of rotations = 4
    y = 1823, y -> L = 1788, y -> R = NULL

+++ Preorder traversal of final tree
     2076  1228   675   493   328   194   258   385   636  1089   805   899
     1045  1905  1823  1788  1688  1639  1434  1348  1526  2031  2585  2372
     2255  2128  2178  2427  2502  3209  2940  2805  2683  3148  3014  3300
     3624  3471  3367  3512

--------------------------------------------------------------------------------

n = 40

Keys to insert:
1569  2477  2335  1836  1672   625  2148  1022  1720   414   725  1799
2374  1448  1173  1578  1285  2275  1049  2883   889  2070  1953   332
 512   447   276   257  1887  2617  1361   792   602  2589  1189  2660
 951  2768  2385   153

+++ Preorder traversal of initial tree
     1569   625   414   332   276   257   153   512   447   602  1022   725
      889   792   951  1448  1173  1049  1285  1189  1361  2477  2335  1836
     1672  1578  1720  1799  2148  2070  1953  1887  2275  2374  2385  2883
     2617  2589  2660  2768

+++ Finding closest neighbor
    x = 1569, y = 1578, gap = 9

+++ Bringing neighboring key to child position
    Number of rotations = 5
    y = 1578, y -> L = 1569, y -> R = 1672

+++ Preorder traversal of final tree
     1578  1569   625   414   332   276   257   153   512   447   602  1022
      725   889   792   951  1448  1173  1049  1285  1189  1361  1672  1836
     1720  1799  2335  2148  2070  1953  1887  2275  2477  2374  2385  2883
     2617  2589  2660  2768

--------------------------------------------------------------------------------
```

CS29003 Algorithms Laboratory

Lab Test

Date: 26–March–2019

ODD PC

Let T be a binary search tree (BST) storing n integer-valued keys k_i satisfying $0 < k_1 < k_2 < k_3 < \cdots < k_n$. Two consecutive keys in T are called a *neighbor pair*, that is, T contains $n-1$ neighbor pairs (k_i, k_{i+1}) for $i = 1, 2, 3, \ldots, n-1$. The *distance* of the neighbors k_i and k_{i+1} is $k_{i+1} - k_i$. Consider the two problems.

(a) Find the farthest neighbor pair in T (with respect to the distance defined above). If multiple neighbor pairs have the same distance as the farthest one, finding any one of these pairs suffices.

(b) If the farthest pair is (x, y), apply a sequence of rotations to T until y becomes the right child of x.

Problem (b) is solved if you make T completely right-skew, but this process may require about n rotations. You need to solve Problem (b) by a restricted number of rotations. Let $level(u)$ denote the level of the node u in T, that is, the distance of u from the root of T. For example, the root itself is at level 0, its children are at level 1, the grandchildren of the root are at level 2, and so on. If (x, y) is the farthest pair found by solving Problem (a), you are allowed to make a maximum of $|level(x) - level(y)|$ rotations. Note that the *only* operation permitted for solving Problem (b) is rotation (both left and right rotations are allowed). Note also that you do not need to compute $level(x)$ and $level(y)$ explicitly.

Each node in T should consist of an integer key value and two child pointers. Since rotations are involved, a node is additionally allowed to have a *parent pointer*. A node must not contain any other field. Implement the following parts in order to solve the two problems mentioned above.

Part 1: No black box is provided for constructing T. Write a function *insert*( ) to implement the *standard BST insertion* algorithm. For inserting a key k to T, first search for k in T. If k is found, do nothing. Otherwise, the search fails when you follow a NULL pointer. Create a new node storing k, and replace the NULL pointer you encountered during the search by a pointer to the new node. Adjust the parent pointer of the new node if your node contains this pointer. Do not perform any type of height balancing. **(5)**

Part 2: Write a function *preorder*( ) to print the preorder listing of the keys stored in T. **(3)**

Part 3: Write a function *findnbr*( ) for finding the farthest neighbor pair (x, y). In order to do so, modify the inorder traversal procedure in T. This traversal prints the keys of T in sorted order. Instead of printing, you should find the maximum distance between neighbors. Your function should run in $O(n)$ time. You are allowed (but not forced) to use an external array to store the sorted list of keys of T. **(8)**

Part 4: Write two functions *lrotate*( ) and *rrotate*( ) to perform a left or right rotation at a specified node u. Each function should return a pointer to the new node v that occupies the position of u after the rotation. Your functions should change the parent pointers of the affected nodes (if you use these pointers). **(4)**

Part 5: Write a function *makechild*( ) to make y the right child of x, where (x, y) is the pair obtained in Part 3. This function should run in $O(|level(x) - level(y)|)$ time, and achieve its desired goal only by applying rotations to the nodes of T. Recall that the maximum number of rotations allowed is $|level(x) - level(y)|$. (**Hint:** Here, x is the immediate predecessor of y, and y is the immediate successor of x.) **(8)**

The *main*() function and output

- Initialize T to an empty tree. Read n, and n keys from the user. Insert in T the keys using the function of Part 1. Print the preorder listing of the keys of T after all the n keys are inserted. **(2 + 2)**
- Call *findnbr* of Part 3 to locate the farthest neighbor pair (x, y) in T. Print the keys at x and y. **(2 + 2)**
- Call *makechild* of Part 5 to make y the right child of x. Print how many rotations your function applies on nodes of T. Print also the two children of x, and finally the preorder listing of the keys in T. **(2 + 2)**

Submit a single C/C++ source file. Do not use global/static variables.

Sample outputs

```
-------------------------------------------------------------------------------

n = 40

Keys to insert:
1687  1365  3905  1793  1547   861  3755  2307   448  1622   538  2400
 719  1211  2539   350   159  2884   601  2164  2790  2667  3416  1945
3489   801  1080  2994   204  2251  1279  3232  2079  3601  1492  3705
 667  3335  3115   937

+++ Preorder traversal of initial tree
     1687  1365   861   448   350   159   204   538   719   601   667   801
     1211  1080   937  1279  1547  1492  1622  3905  1793  3755  2307  2164
     1945  2079  2251  2400  2539  2884  2790  2667  3416  2994  3232  3115
     3335  3489  3601  3705

+++ Finding farthest neighbor
    x = 1793, y = 1945, gap = 152

+++ Bringing neighboring key to child position
    Number of rotations = 3
    x = 1793, x -> L = NULL, x -> R = 1945

+++ Preorder traversal of final tree
     1687  1365   861   448   350   159   204   538   719   601   667   801
     1211  1080   937  1279  1547  1492  1622  3905  1793  1945  2164  2079
     2307  2251  3755  2400  2539  2884  2790  2667  3416  2994  3232  3115
     3335  3489  3601  3705

-------------------------------------------------------------------------------

n = 40

Keys to insert:
2980  1612   464   913   866  3284  1574  1421  1126  1275  1302   528
1892  3206  1705  3138  3250   658  1158  1994  2104   766  3034  2279
2663  2588  1341  1772   119  2361  2840   616  2493  2733  1075   329
1446   243  2217  1018

+++ Preorder traversal of initial tree
     2980  1612   464   119   329   243   913   866   528   658   616   766
     1574  1421  1126  1075  1018  1275  1158  1302  1341  1446  1892  1705
     1772  1994  2104  2279  2217  2663  2588  2361  2493  2840  2733  3284
     3206  3138  3034  3250

+++ Finding farthest neighbor
    x = 2840, y = 2980, gap = 140

+++ Bringing neighboring key to child position
    Number of rotations = 7
    x = 2840, x -> L = 2663, x -> R = 2980

+++ Preorder traversal of final tree
     2840  2663  2279  2104  1994  1892  1612   464   119   329   243   913
      866   528   658   616   766  1574  1421  1126  1075  1018  1275  1158
     1302  1341  1446  1705  1772  2217  2588  2361  2493  2733  2980  3284
     3206  3138  3034  3250

-------------------------------------------------------------------------------
```