Dayue Bai
dayueb@uci.edu
35439000
ICS 46
Extra HW 10

==========START OF REPORT==========
Part 1 (Code Implementation and Time complexity analysis):

```
12
13      void insert(keytype key, infotype info)
14      {
15          root = TreeNode<keytype, infotype>::insert(key, info, root);
16      }
17
18      infotype find(keytype key)
19      {
20          TreeNode<keytype, infotype>* tp = TreeNode<keytype, infotype>::find(key, root);
21          if (!tp)
22          {
23              insert(key, 0);
24              tp = TreeNode<keytype, infotype>::find(key, root);
25          }
26          return tp->info;
27      }
28
29      infotype& operator [](keytype key)
30      {
31          TreeNode<keytype, infotype>* tp = TreeNode<keytype, infotype>::find(key, root);
32          if (!tp)
33              cout << "Not Found" << endl;
34          else
35              return tp->info;
36      }
37
38      void remove(keytype key)
39      {
40          root = TreeNode<keytype, infotype>::remove(key, root);
41      }
42
```

Time Complexity Analysis:
Insert: The rotation operations (left and right rotate) take constant time as only few pointers are being changed there. Updating the height and getting the balance factor also take constant time. So the time complexity of AVL insert remains same as BST insert which is O(h) where h is height of the tree. Since AVL tree is balanced, the height is O(Logn). So time complexity of AVL insert is O(LogN)

Find: O(log N), in which h is the height of the tree and h = logN. Finding will find each element according to its key, either left or right down to the bottom of the tree, so the time complexity is O(logN)

Remove: Remove is similar to insert. It only takes constant time to rotate several pointers. It remove the elements according to the key, either left or right. The shape of the tree is maintained to be balanced. Therefore h = log N. The time complexity is O(log N)

Operator []: O(logN), h = log N. Explanation is the same as find.

Helper functions of the function above:

```
60
61      static TreeNode* insert(keytype key, infotype info, TreeNode* root)
62      {
63          if (!root)
64              return new TreeNode(key, info, 0, nullptr, nullptr);
65          if (root->key > key)
66              root->left = insert(key, info, root->left);
67          else if (root->key < key)
68              root->right = insert(key, info, root->right);
69          else{
70              ++root->info;
71              return root;
72          }
73          root->height = 1 + max(calc_height(root->right), calc_height(root->left));
74          int difference = calc_dif(root);
75
76          if (difference > 1 && root->left->key > key) // left, left
77              return ll_rotate(root);
78          else if (difference > 1 && root->left->key < key) // left, right
79              return lr_rotate(root);
80          else if (difference < -1 && root->right->key < key) // right, right
81              return rr_rotate(root);
82          else if (difference < -1 && root->right->key > key) // right, left
83              return rl_rotate(root);
84
85          return root;
86      }
```

bai — TreeNode.h (~/ics46/hw10) - VIM — ssh dayueb@openlab.ics.uci.edu — 101×51

```
15      static int calc_dif(TreeNode* root)
16      {
17          return root ? calc_height(root->left) - calc_height(root->right) : 0;
18      }
19
20      static int calc_height(TreeNode* root)
21      {
22          return root ? root->height : -1;

23      }
24
25      static TreeNode* ll_rotate(TreeNode* root)
26      {
27          TreeNode* p1 = root->left;
28          TreeNode* p2 = root->left->right;
29          p1->right = root;
30          p1->right->left = p2;
31          root->height = 1 + max(calc_height(root->left), calc_height(root->right));
32          p1->height = 1 + max(calc_height(p1->left), calc_height(p1->right));
33
34          return p1;
35      }
36
37      static TreeNode* rr_rotate(TreeNode* root)
38      {
39          TreeNode* p1 = root->right;
40          TreeNode* p2 = root->right->left;
41          p1->left = root;
42          root->right = p2;
43          root->height = 1 + max(calc_height(root->left), calc_height(root->right));
44          p1->height = 1 + max(calc_height(p1->left), calc_height(p1->right));
45
46          return p1;
47      }
48
49      static TreeNode* lr_rotate(TreeNode* root)
50      {
51          root->left = rr_rotate(root->left);
52          return ll_rotate(root);
53      }
54
55      static TreeNode* rl_rotate(TreeNode* root)
56      {
57          root->right = ll_rotate(root->right);
58          return rr_rotate(root);
59      }
```

```
88      static TreeNode* find(keytype key, TreeNode* root)
89      {
90          if (!root)
91              return nullptr;
92          else if (root->key > key)
93              return find(key, root->left);
94          else if (root->key < key)
95              return find(key, root->right);
96          return root;
97      }
```

bai — TreeNode.h (~/ics46/hw10) - VIM — ssh dayueb@openlab.ics.uci.edu — 101×51

```
99      static TreeNode* remove(keytype key, TreeNode* root)
100     {
101         if (!root)
102             return root;
103         if (root->key > key)
104             root->left = remove(key, root->left);
105         else if (root->key < key)
106             root->right = remove(key, root->right);
107         else{
108             if (!root->left){
109                 TreeNode* p = root->right;
110                 delete root;
111                 return p;
112             }
113             else if (!root->right){
114                 TreeNode* p = root->left;
115                 delete root;
116                 return p;
117             }
118             else{
119                 TreeNode* p = min(root->right);
120                 root->key = p->key;
121                 root->info = p->info;
122                 root->right = remove(p->key, root->right);
123             }
124         }
125         root->height = 1 + max(calc_height(root->left), calc_height(root->right));
126         int difference = calc_dif(root);
127
128         if (difference > 1 && calc_dif(root->left) >= 0)
129             return ll_rotate(root);
130         else if (difference > 1 && calc_dif(root->left) < 0)
131             return lr_rotate(root);
132         else if (difference < -1 && calc_dif(root->right) <= 0)
133             return rr_rotate(root);
134         else if (difference < -1 && calc_dif(root->right) > 0)
135             return rl_rotate(root);
136
137         return root;
138     }
139
140     static TreeNode* min(TreeNode* root)
141     {
142         if (root)
143         {
144             while(root->left)
145                 root = root->left;
146         }
147         return root;
148     }
```

TreeNode::insert()

```
60
61      static TreeNode* insert(keytype key, infotype info, TreeNode* root)
62      {
63          if (!root)
64              return new TreeNode(key, info, 0, nullptr, nullptr);
65          if (root->key > key)
66              root->left = insert(key, info, root->left);
67          else if (root->key < key)
68              root->right = insert(key, info, root->right);
69          else{
70              ++root->info;
71              return root;
72          }
73          root->height = 1 + max(calc_height(root->right), calc_height(root->left));
74          int difference = calc_dif(root);
75
76          if (difference > 1 && root->left->key > key) // left, left
77              return ll_rotate(root);
78          else if (difference > 1 && root->left->key < key) // left, right
79              return lr_rotate(root);
80          else if (difference < -1 && root->right->key < key) // right, right
81              return rr_rotate(root);
82          else if (difference < -1 && root->right->key > key) // right, left
83              return rl_rotate(root);
84
85          return root;
86      }
```

Time complexity analysis of TreeNode::insert(): O(log N)
The rotation operations (left and right rotate) take constant time as only few pointers are being changed there. Updating the height and getting the balance factor also take constant time. So the time complexity of AVL insert remains same as BST insert which is O(h) where h is height of the tree. Since AVL tree is balanced, the height is O(Logn). So time complexity of AVL insert is O(LogN)

bai — TreeNode.h (~/ics46/hw10) - VIM — ssh dayueb@openlab.ics.uci.edu — 101×51

```
15      static int calc_dif(TreeNode* root)
16      {
17          return root ? calc_height(root->left) - calc_height(root->right) : 0;
18      }
19
20      static int calc_height(TreeNode* root)
21      {
22          return root ? root->height : -1;

23      }
24
25      static TreeNode* ll_rotate(TreeNode* root)
26      {
27          TreeNode* p1 = root->left;
28          TreeNode* p2 = root->left->right;
29          p1->right = root;
30          p1->right->left = p2;
31          root->height = 1 + max(calc_height(root->left), calc_height(root->right));
32          p1->height = 1 + max(calc_height(p1->left), calc_height(p1->right));
33
34          return p1;
35      }
36
37      static TreeNode* rr_rotate(TreeNode* root)
38      {
39          TreeNode* p1 = root->right;
40          TreeNode* p2 = root->right->left;
41          p1->left = root;
42          root->right = p2;
43          root->height = 1 + max(calc_height(root->left), calc_height(root->right));
44          p1->height = 1 + max(calc_height(p1->left), calc_height(p1->right));
45
46          return p1;
47      }
48
49      static TreeNode* lr_rotate(TreeNode* root)
50      {
51          root->left = rr_rotate(root->left);
52          return ll_rotate(root);
53      }
54
55      static TreeNode* rl_rotate(TreeNode* root)
56      {
57          root->right = ll_rotate(root->right);
58          return rr_rotate(root);
59      }
```

Part 2 (Test BST under valgrind):

```
bai — dayueb@andromeda-40:~/ics46/hw10 — ssh dayueb@openlab.ics.uci.edu — 101×51
dayueb@andromeda-40 22:08:55 ~/ics46/hw10
$ ls
BinarySearchTree.h  Makefile  Random.txt  testTree*  testTree.cpp  Timer.h  TreeNode.h  Words.txt
dayueb@andromeda-40 22:08:56 ~/ics46/hw10
$ valgrind testTree
==20468== Memcheck, a memory error detector
==20468== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==20468== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==20468== Command: testTree
==20468==
File: Random.txt. Partition: 1/10, time of insertAllWords: 0.307723
File: Random.txt. Partition: 1/10, time of findAllWords: 0.160142
File: Random.txt. Partition: 1/10, time of removeAllWords: 0.243282

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

File: Random.txt. Partition: 2/10, time of insertAllWords: 0.55387
File: Random.txt. Partition: 2/10, time of findAllWords: 0.327066
File: Random.txt. Partition: 2/10, time of removeAllWords: 0.490956

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

File: Random.txt. Partition: 3/10, time of insertAllWords: 0.854566
File: Random.txt. Partition: 3/10, time of findAllWords: 0.505
File: Random.txt. Partition: 3/10, time of removeAllWords: 0.755609

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

File: Random.txt. Partition: 4/10, time of insertAllWords: 1.16026
File: Random.txt. Partition: 4/10, time of findAllWords: 0.689813
File: Random.txt. Partition: 4/10, time of removeAllWords: 1.02666

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

File: Random.txt. Partition: 5/10, time of insertAllWords: 1.47786
File: Random.txt. Partition: 5/10, time of findAllWords: 0.870979
File: Random.txt. Partition: 5/10, time of removeAllWords: 1.31075

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

File: Random.txt. Partition: 6/10, time of insertAllWords: 1.79412
File: Random.txt. Partition: 6/10, time of findAllWords: 1.06318
File: Random.txt. Partition: 6/10, time of removeAllWords: 1.59688

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

File: Random.txt. Partition: 7/10, time of insertAllWords: 2.11889
File: Random.txt. Partition: 7/10, time of findAllWords: 1.25322
File: Random.txt. Partition: 7/10, time of removeAllWords: 1.87498

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

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

File: Random.txt. Partition: 8/10, time of insertAllWords: 2.44263
File: Random.txt. Partition: 8/10, time of findAllWords: 1.44801
File: Random.txt. Partition: 8/10, time of removeAllWords: 2.17146

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

File: Random.txt. Partition: 9/10, time of insertAllWords: 2.77884
File: Random.txt. Partition: 9/10, time of findAllWords: 1.64249
File: Random.txt. Partition: 9/10, time of removeAllWords: 2.46115

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

File: Random.txt. Partition: 10/10, time of insertAllWords: 3.10034
File: Random.txt. Partition: 10/10, time of findAllWords: 1.84202
File: Random.txt. Partition: 10/10, time of removeAllWords: 2.75241

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

File: Words.txt. Partition: 1/10, time of insertAllWords: 0.28473
File: Words.txt. Partition: 1/10, time of findAllWords: 0.15482
File: Words.txt. Partition: 1/10, time of removeAllWords: 0.201476

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

File: Words.txt. Partition: 2/10, time of insertAllWords: 0.594728
File: Words.txt. Partition: 2/10, time of findAllWords: 0.326408
File: Words.txt. Partition: 2/10, time of removeAllWords: 0.424767

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

File: Words.txt. Partition: 3/10, time of insertAllWords: 0.924307
File: Words.txt. Partition: 3/10, time of findAllWords: 0.513155
File: Words.txt. Partition: 3/10, time of removeAllWords: 0.652898

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

File: Words.txt. Partition: 4/10, time of insertAllWords: 1.25206
File: Words.txt. Partition: 4/10, time of findAllWords: 0.704368
File: Words.txt. Partition: 4/10, time of removeAllWords: 0.900547

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

File: Words.txt. Partition: 5/10, time of insertAllWords: 1.60099
File: Words.txt. Partition: 5/10, time of findAllWords: 0.881803
File: Words.txt. Partition: 5/10, time of removeAllWords: 1.13187

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

```
File: Words.txt. Partition: 6/10, time of insertAllWords: 1.93865
File: Words.txt. Partition: 6/10, time of findAllWords: 1.06608
File: Words.txt. Partition: 6/10, time of removeAllWords: 1.38406

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

File: Words.txt. Partition: 7/10, time of insertAllWords: 2.27843
File: Words.txt. Partition: 7/10, time of findAllWords: 1.25705
File: Words.txt. Partition: 7/10, time of removeAllWords: 1.63647

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

File: Words.txt. Partition: 8/10, time of insertAllWords: 2.6258
File: Words.txt. Partition: 8/10, time of findAllWords: 1.45544
File: Words.txt. Partition: 8/10, time of removeAllWords: 1.88611

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

File: Words.txt. Partition: 9/10, time of insertAllWords: 2.9701
File: Words.txt. Partition: 9/10, time of findAllWords: 1.64398
File: Words.txt. Partition: 9/10, time of removeAllWords: 2.13313

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

File: Words.txt. Partition: 10/10, time of insertAllWords: 3.32178
File: Words.txt. Partition: 10/10, time of findAllWords: 1.84988
File: Words.txt. Partition: 10/10, time of removeAllWords: 2.39682

------------------------------------------------
==20468==
==20468== HEAP SUMMARY:
==20468==     in use at exit: 72,704 bytes in 1 blocks
==20468==   total heap usage: 601,515 allocs, 601,514 frees, 30,397,214 bytes allocated
==20468==
==20468== LEAK SUMMARY:
==20468==    definitely lost: 0 bytes in 0 blocks
==20468==    indirectly lost: 0 bytes in 0 blocks
==20468==      possibly lost: 0 bytes in 0 blocks
==20468==    still reachable: 72,704 bytes in 1 blocks
==20468==         suppressed: 0 bytes in 0 blocks
==20468== Rerun with --leak-check=full to see details of leaked memory
==20468==
==20468== For counts of detected and suppressed errors, rerun with: -v
==20468== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
dayueb@andromeda-40 22:10:25 ~/ics46/hw10
$
```

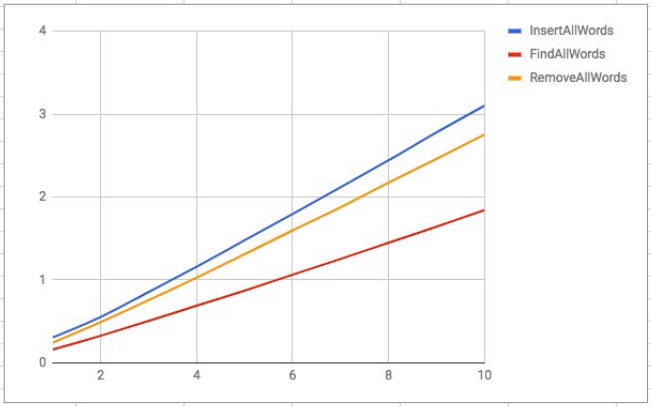
Part 3 (Graphing the data):
Random.txt:

Random.txt

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

Partition

	A	B	C	D
1	Partition	InsertAllWords	FindAllWords	RemoveAllWords
2	1	0.307723	0.160142	0.243282
3	2	0.55387	0.327066	0.490956
4	3	0.854566	0.505	0.755609
5	4	1.16026	0.689813	1.02666
6	5	1.47786	0.870979	1.31075
7	6	1.79412	1.06318	1.59688
8	7	2.11889	1.25322	1.87498
9	8	2.44263	1.44801	2.17146
10	9	2.77884	1.64249	2.46115
11	10	3.10034	1.84202	2.75241

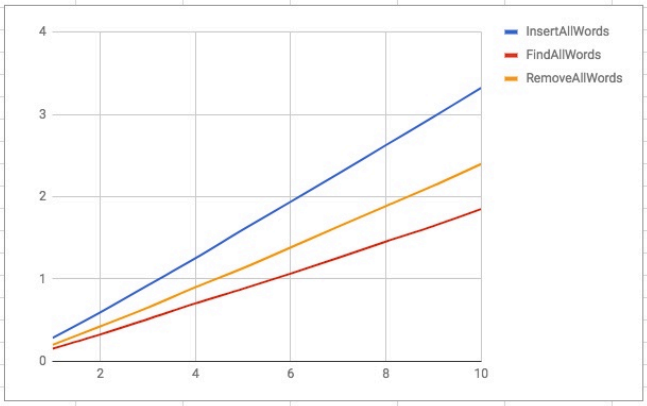


Words.txt:

Words.txt

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

Partition	InsertAllWords	FindAllWords	RemoveAllWords
1	0.28473	0.15482	0.201476
2	0.594728	0.326408	0.424767
3	0.924307	0.513155	0.652898
4	1.25206	0.704368	0.900547
5	1.60099	0.881803	1.13187
6	1.93865	1.06608	1.38406
7	2.27843	1.25705	1.63647
8	2.6258	1.45544	1.88611
9	2.9701	1.64398	2.13313
10	3.32178	1.84988	2.39682



Part 4 (Test BST and TreeNode as a template under valgrind):

```
2, 49
3, 535
4, 2237
5, 4174
6, 6175
7, 7366
8, 7076
9, 6084
10, 4594
11, 3069
12, 1880
13, 1137
14, 545
15, 278
16, 103
17, 57
18, 23
19, 3
20, 3
21, 2
22, 1
28, 1
==20468==
==20468== HEAP SUMMARY:
==20468==     in use at exit: 72,704 bytes in 1 blocks
==20468==   total heap usage: 601,515 allocs, 601,514 frees, 30,397,214 bytes allocated
==20468==
==20468== LEAK SUMMARY:
==20468==    definitely lost: 0 bytes in 0 blocks
==20468==    indirectly lost: 0 bytes in 0 blocks
==20468==      possibly lost: 0 bytes in 0 blocks
==20468==    still reachable: 72,704 bytes in 1 blocks
==20468==         suppressed: 0 bytes in 0 blocks
==20468== Rerun with --leak-check=full to see details of leaked memory
==20468==
==20468== For counts of detected and suppressed errors, rerun with: -v
==20468== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
dayueb@andromeda-40 22:10:25 ~/ics46/hw10
$
```