Assignment 3

1. Exercise [11.2-3 on p. 229] Modify hashing by chaining (i.e., bucketVector with BucketType = List) so that BucketType = OrderedList. How is the runtime of search, insert, and remove affected?
2. Exercise [11.2-5 on p. 229] Assume hashing by chaining as above (and as in fsu:THashTable<>). Denote by U the "universe" of keys. (This can be the set of all objects of the data type stored in the table, or some subset of that constrained by the particular use of the table.) Show that if $|U| \geq km$, where m is the number of buckets, then there is a subset of U of size k consisting of keys that all hash to the same bucket. (Note that k is just an integer, not the size of anything.)
3. For the following binary tree, complete the table showing each step in a traversal and the number of edges covered by each step (begin, next, ... , next):

```
                        iteration step  current location  no of edge moves
        Q               --------------  ----------------  ----------------
     /     \            1. initialize
    W       E           2. ++
     \     / \          3. ++
      R   T   Y         ...
                        7. ++           (null)
```

4. Describe an order in which vertices of a BST can be saved to ensure that reconstructing a BST from the saved data will result in the same tree. Argue correctness of your assertion.
5. Use the result of Assignment 1 Problem 3(e) [problem 4-5(e) in the textbook] to prove: Height-balanced binary trees satisfy $height(n) = \Theta(\log n)$, where $n =$ number of vertices in the tree.

Solutions

Problem 1: Modify hashing by chaining (i.e., bucketVector with BucketType = List) so that BucketType = OrderedList. How is the runtime of search, insert, and remove affected?

Effect on Search and Remove: Sequential search in an ordered list is, on average, 1/2 the size of the list. The expected search time is unchanged asymptotically [$\Theta(1)$ when the number of buckets is $\Theta(n)$], although the measured runtimes would be decreased somewhat.

Effect on Insert: Insert consists of search followed by insertion into the ordered list. This increases the insert time by 1/2 the size of the bucket, on average. Again, the expected insert time is not changed asymptotically, but the measured times would be increased somewhat.

Problem 2: Assume hashing by chaining as above (and as in fsu:THashTable<>). Denote by U the "universe" of keys. (This can be the set of all objects of the data type stored in the table, or some subset of that constrained by the particular use of the table.) Show that if $|U| \geq nm$, where m is the number of buckets, then there is a subset of U of size n consisting of keys that all hash to the same bucket.

Suppose we insert all elements of U into the table, and let $n(b)$ be the number of elements of U that hash to bucket b. Then

$$|U| = \sum_{0}^{m-1} n(b) \geq nm.$$

Since there are m items in the summation, at least one $n(b)$ must be at least as large as n, for otherwise

$$|U| = \sum_{0}^{m-1} n(b) < \sum_{0}^{m-1} n = nm.$$

Problem 3: For the following binary tree, complete the table showing each step in a traversal and the number of edges covered by each step (begin, next, ... , next):

```
                     iteration step  current location  no of edge moves
       Q             --------------  ----------------  ----------------
    /     \          1. initialize
   W       E         2. ++
    \     / \        ...
     R   T   Y       7. ++           (null)
```

Here is the completed table:

```
      Q
   /     \
  W       E
   \     / \
    R   T   Y
```

iteration step	current location	no of edge moves
1. initialize	W	2
2. ++	R	1
3. ++	Q	2
4. ++	T	2
5. ++	E	1
6. ++	Y	1
7. ++	(null)	3

Note the total of 12 edge moves, which is equal to $2n$, $n = 6$. This is the number of edge moves during an entire traversal, since each edge is traversed twice (once descending, once ascending) and there are two extra moves, ("jumping on" and "jumping off"), for a todal of $2e + 2 = 2(e + 1) = 2n$.

Problem 4: Describe an order in which vertices of a BST can be saved to ensure that reconstructing a BST from the saved data will result in the same tree. Argue correctness of your assertion.

Save the node data in *level order* (using a level order traversal, or a level order iterator) or *pre-order* (using a preorder traversal or preorder iterator). When reconstructing the BST, the data will be inserted in this order, because a former left child is encountered after its former parent and before its former right child. But this means the order will also have been preserved, so that the new insertion will end up as the left child of the previously inserted former parent. Similarly for the former right child.

Problem 5: Use the result of Assignment 1 Problem 3(e) [problem 4-5(e) in the textbook] to prove: Height-balanced binary trees satisfy $\textit{height}(n) = \Theta(\log n)$, where n = number of vertices in the tree.

An AVL tree is a binary tree in which the heights of the left and right subtrees of each node differ by at most 1. Note this definition is equivalent to the recursive version: an AVL tree is a binary tree in which the heights of the left and right subtrees of the root differ by at most 1 and in which the left and right subtrees are again AVL trees. The name "AVL" derives from the names of the two inventors of the technology, G.M. Adelson-Velskii and E.M. Landis [An algorithm for the organization of information, 1962.] We will adopt the more neutral terminology height-balanced tree (abbreviated HB tree or HBT) in discussing the abstract data type and its implementations, but retain the attributive names when discussing the underlying algorithms.

Because an HB tree (HBT) is a binary search tree (BST), there is a well defined binary search algorithm in an HBT that follows descending paths. An important feature of HBTs is that they have height $\Theta(\log n)$, where n is the number of nodes in the HBT, so an HBT must be fairly "bushy". (In stark contrast, a BST can be severely "leggy", with height $n-1$. We can define bushy and leggy asymptotically as having height $O(\log n)$ and $\Omega(n)$, respectively.) In the following, let H and n denote the height and number of vertices of a tree, respectively.

Theorem 1. For any height-balanced tree, $H = \Theta(\log n)$.

Proof. First note that $H \geq \Omega(\log n)$ for any binary tree, because the number of vertices cannot exceed the number of possible vertex locations:

$$n \leq 1 + 2 + ... + 2^H = 2^{H+1} - 1.$$

Taking the logarithm base 2 of each side, we get the result.

We concentrate now on second part of the claim, that $H \leq O(\log n)$ assuming the binary tree is height balanced. Let $n(H)$ be the minimum number of vertices an HB tree of height H can have. Clearly $n(0) = 1$, since a tree of height 0 consists exactly of the root vertex. Also $n(1) = 2$, by looking at all cases. As an inductive step, note that an HB tree of height H with minimal vertex count must have one subtree of height $H - 1$ and another subtree of height $H - 2$ (otherwise a vertex could be removed and $n(H)$ would not be minumal). Thus $n(H) = n(H-1) + n(H-2) + 1$. In summary, we have the following recurrance relation:

$$\begin{aligned} n(0) &= 1 \\ n(1) &= 2 \\ n(H) &= n(H-1) + n(H-2) + 1 \end{aligned}$$

Note the similarity with the Fibannaci recursion given by:

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(H) &= f(H-1) + f(H-2) \end{aligned}$$

Assertion 1: $n(H) > f(H+2) - 1$

Proof:

Base cases: $n(0) = 1$ and $f(2) - 1 = 1 - 1 = 0$, so $n(0) > f(2) - 1$.
$n(1) = 2$ and $f(3) - 1 = 2 - 1 = 1$, so $n(1) > f(2) - 1$.

Inductive step: Applying the definitions of the two recursions and the inductive hypothesis, we have:

$$\begin{aligned} n(H+1) &= n(H) + n(H-1) + 1 \\ &> (f(H+2) - 1) + (f(H+1) - 1) + 1 \\ &= f(H+2) + f(H+1) - 1 \\ &= f(H+3) - 1 \end{aligned}$$

This proves Assertion 1.

Because both sides of the inequality are integers, we can rephrase the previous assertion as:

Assertion 2: $n(H) >= f(H+2)$

Exercise 4-5 in your textbook concludes that $f(H+2) >= \phi^H/\sqrt{5}$, where $\phi = (1+\sqrt{5})/2$, the golden ratio. Whence we obtain:

Assertion 3: $n(H) >= \phi^H/\sqrt{5}$

Taking the logarithm (base ϕ) of both sides, we see that $H \leq O(\log n(H))$. Because $n(H)$ is the minimum number of vertices an HBT of height H can have, it follows that $H \leq O(\log n)$ which completes the proof of Theorem 1.

Theorem 2. BST search in an HBT has worst case run time $O(\log n)$.

Proof. The BST search algorithm consists of a loop that examines nodes in a descending path, starting at the root. Such a path has length no greater than the height of the tree, which is $\Theta(\log n)$ by Theorem 1.

Theorem 3. HBT insert and remove have worst case run time $O(\log n)$.

Proof. The challenge is to make sure that the HBT properties are maintained as we insert and remove elements. It turns out that the HBT properties do not necessarily hold after an ordinary BST insert or remove operation, but that there are "repair" algorithms that bring the resulting BST back into compliance with the HBT definition. These algorithms restructure the BST by pruning and re-hanging subtrees and are called rotations.

Rotations are constant time algorithms, and they are combined into repair algorithms that iterate along a descending path in the HBT. These repair algorithms thus have runtime bounded by $O(\log n)$. It follows that BST insert or remove, followed by HBT repair, has run time $O(\log n + \log n) = O(\log n)$.

Note that the constants associated with insert and remove will be significantly larger than those for search, so much so that it is sometimes more efficient to build ordinary BST structures, if the insertions are sufficiently random to ensure a bushy tree, and thus not pay the cost of maintaining the HBT property.