,9

CS 240 - Data Structures and Data Management **Spring 2017**

Lecture 8,9 : May 30 - June 1, 2017

Lecturer: Taylor Smith *Notes By: Harsh Mistry*

8.1 Dictionary ADT

A dictionary is a collection of items, each of which contains

- A key
- Some data

and is called a key-value pair (KVP). Keys can be compared and are typically unique

8.1.1 Operations

- Search(k)
- Insert(k,v)
- Delete(k)
- optional : join, isEmpty, size, etc,

8.1.2 Common Assumptions:

- Dictionary has n KVPs
- Each KVP uses constant space
- Comparing keys takes constant space

Unordered array of linked list

- Search $\theta(n)$
- Insert $\theta(1)$
- Delete $\theta(n)$ (need to search)

Ordered array

- Search $\theta(logn)$
- Insert $\theta(n)$
- Delete $\theta(n)$

8.2 AVL Trees

Introduced by Adelson-Velski and Landis in 1962, an AVL Tree is a BST with additional structural property : The heights of the left and right subtree differ by at most 1 and the height of an empty tree is defined to be -1.

At each non-empty node we store $height(R) - height(L) \in \{-1, 0, 1\}$:

- -1 means the tree is left heavy
- 0 means the tree is balanced
- 1 means the tree is right heavy

8.3 AVL Insertion

To perform insert(T,k,v)

- First, insert (k,v) into T using usual BST insertion
- Then, move up the tree from the new leaf, updating balance factors.
- If the balance factor is 1, 0, or 1, then keep going.
- If the balance factor is +2 or -2, then call the fix algorithm to rebalance at that node. We are done.

8.3.1 Rotations

```
rotate-right(T)
  T: AVL tree
  newroot <- T.left
  T.left <- newroot.right
  newroot.right <- T
  return newroot
```

```
rotate-left(T)
  T: AVL tree
  newroot <- T.right
  T.right <- newroot.left
  newroot.left <- T
  return newroot
```

8.3.2 Fixing a Slightly-Unbalanced AVL Tree

Idea : Identify one of the previous 4 situations apply rotations

```
Fix(T):
  T: AVL tree with T.balance = 2 || T.balance = -2
  if T.balance = -2 then
    if T.left.balance = 1 then
      T.left <- rotate-left(T.left)
    return rotate-right(T)
  else if T.balance = 2 then
    if T.right.balance = -1 then
      T.right <- rotate-right(T.right)
    return rotate-left(T)
```

8.3.3 AVL Tree Operations

- Search : costs $\theta(height)$
- Insert : Shown already, total cost $\theta(height)$
  - fix restores the height of the tree it fixes to what it was
  - so fix will be called at most once.
- Delete : First search, then swap with successor, then move the tree and apply fix (as with insert)
  - fix may be called $\theta(height)$ times

  Total cost is $\theta(height)$

8.3.4 Height of an AVL tree

Define N(h) to be the least number of nodes in a height-h AVL tree.

One subtree must have height at least $h-1$, the other at least $h-2$:

$$N(h) = \begin{cases} 1 + N(h-1) + N(h-2), & h \geq 1 \\ 1, & h = 0 \\ 0, & h = -1 \end{cases}$$

8.3.5 AVL Tree Analysis

Easier lower bound on N(h);

$$N(h) > 2N(h-2) > 4N(h-4) > 8N(h-6) > \ldots > 2^i N(h-2i) \geq 2^{[h/2]}$$

Since $n > 2^{h/2}, h \leq 2\log n$, and thus an AVL tree with n nodes has height $O(\log n)$. Also, $n \leq 2^{h+1} - 1$, so the hight height is $\theta(\log n)$

$\implies$ search, insert, delete all cost $\theta(\log n)$