CSE 326
Homework 5
Solutions

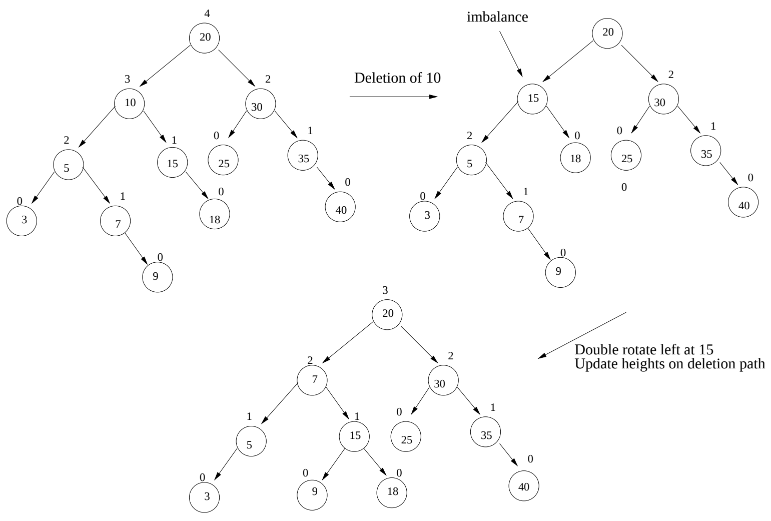
1. Assume we have the helper function `ht` that computes the height of a node of the AVL tree from its `height` field. That is,

```
ht(n : node) : integer
{
if n = null then return(-1) else return(n.height)
}
```

In order to do an AVL tree deletion we do exactly a normal binary tree deletion, but before returning from the recursive delete function we check for imbalance: if there is one we do the appropriate rebalancing and if there is no imbalance then we recompute the height of the node because if may have changed. We assume we have the four functions `single-rotate-right`, `single-rotate-left`, `double-rotate-right`, and `double-rotate-left`. Each of these functions recomputes the heights of the rotated nodes correctly. We assume that the four fields of the AVL-tree nodes are `data`, `left`, `right`, and `height`.

```
AVL-delete(x: key, p: reference node pointer)
{
local pointer q; \\ node to be physically deleted
local integer hl; \\ height of left child of p
local integer hr; \\ height of right child of p
if not(p=null) then
   {
   case
     p.data < x: AVL-delete(x, p.right);
     p.data > x: AVL-delete(x. p.left);
     p.data = x:
       case
         p.left = null: p := p.right;
         p.right = null: p := p.left;
```

```
        otherwise:
          {
          q := Successor(p);
          p.data := q.data;
          AVL-delete(q.data,p.right)
          }
        endcase
  endcase
  hl := ht(p.left);
  hr := ht(p.right);
  case
     abs(hl - hr) <= 1:   \\ no rebalance needed, but recompute height
        p.height := max(ht(p.left),ht(p.right)) + 1
     hr < hl: \\ left subtree is higher than the right and not null
        if ht(p.left.left) >= ht(p.left.right) then
            single-rotate-left(p)
        else
            double-rotate-left(p)
     otherwise: \\ right subtree is higher than the left and not null
        if ht(p.right.right) >= ht(p.right.left) then
            single-rotate-right(p)
        else
            double-rotate-right(p)
  endcase
  }
}
```



2. Insert 8

```
             20
         /        \
      10           30
    /    \       /   \
   5     15    25     35
 /  \      \            \
3    7     18            40
      \
       9
      /
     8
```

zig-zag from right

```
             20
         /        \
      10           30
    /    \       /   \
   5     15    25     35
 /  \      \            \
3    8     18            40
    / \
   7   9
```

zig-zag from left

```
             20
         /        \
        8          30
     /    \       /  \
    5     10    25    35
   / \   /  \           \
  3   7 9    15          40
               \
                18
```

zig from left

```
             8
         /       \
        5         20
     /    \      /   \
    3      7   10      30
              /  \    /  \
             9    15 25   35
                    \       \
                     18       40
```

```
To delete 10, first splay 10 to root with zig from left

                            10
                        /        \
                       5          20
                     /   \       /   \
                    3     7    15     30
                           \    \   /  \
                            9   18 25   35
                                          \
                                           40

remove 10


                       5          20
                    /    \       /   \
                   3      7    15     30
                           \    \   /  \
                            9   18 25   35
                                          \
                                           40

Splay 9 to root in left tree with a zig-zig from right


                      9          20
                     /          /   \
                    7         15     30
                   /            \   /  \
                  5             18 25   35
                 /                        \
                3                          40

Attach right tree to left tree

                            9
                         /     \
                        7        20
                       /       /   \
                      5      15     30
                     /         \   /  \
                    3          18 25   35
                                         \
                                          40
```

3. `insert 1`

```
    1
```

`insert 2`

```
   1 2
```

`insert 3`

```
   1 2 3
```

`insert 4`

```
1 2    3 4


     3
  /     \
1 2    3 4
```

`insert 5`

```
     3
  /     \
1 2    3 4 5
```

`insert 6`

```
     3 : 5
  /    |    \
1 2   3 4  5 6
```

`insert 7`

```
     3 : 5
  /    |    \
1 2   3 4  5 6 7
```

`insert 8`

```
     3 : 5
  /    |    \
1 2   3 4  5 6   7 8

         5
     /       \
    3         7
  /   \     /   \
1 2   3 4  5 6  7 8
```

```
insert 9

         5
     /       \
    3         7
  /   \     /   \
1 2   3 4  5 6  7 8 9

insert 10

         5
     /       \
    3         7 : 9
  /   \     /   |   \
1 2   3 4  5 6  7 8  9 10

insert 11

         5
     /       \
    3         7 : 9
  /   \     /   |   \
1 2   3 4  5 6  7 8  9 10 11

insert 12

         5
     /       \
    3         7 : 9
  /   \     /   |   \
1 2   3 4  5 6  7 8  9 10   11 12


         5
     /       \
    3         7          11
  /   \     /   \      /    \
1 2   3 4  5 6  7 8  9 10   11 12


          5   :   9
     /        |         \
    3         7          11
  /   \     /   \      /    \
1 2   3 4  5 6  7 8  9 10   11 12
```

4. Let M be the maximum number of children of an internal node and let L be the maximum number of keys per leaf. Because each internal node can have as many as M children, $M-1$ keys, and a 2 byte length field we have:

$$4M + 4(M-1) + 2 \leq 2048.$$

Hence $M \leq 2050/8 = 256.25$. So the maximum value of M is 256.

Because each leaf has L pointers to data, L keys, and a 2 byte length field we have:

$$4L + 4L + 2 \leq 2048.$$

Hence $L \leq 2046/8 = 255.75$. So the maximum value of L is 255.

To calculate the height we'll assume that leaves and internal nodes (other than the root) are about 3/4-th full. Hence leaves have about $255 \times .75 \approx 191$ keys each and internal nodes (other than the root have about $256 \times .75 \approx 192$ children each. The total number of leaves is about $100,000,000/191 \approx 523,560$. The total number of internal nodes at depth 1 is approximately $523,560/192 \approx 2,727$. The total number of nodes at depth 2 is approximately $2,727/192 \approx 15$. The root at depth 3 has about 15 children. Thus, the height of the tree is 3.

There are $2^{24}/2^{11} = 2^{13} = 8,192$ pages that fit in memory. Each page holds exactly one node of the B-tree. The root, its 15 children, and their 2,880 children all fit in memory. In addition, $8,192 - (1 + 15 + 2,880) = 5,246$ leaves fit in memory. So the first three levels and a small fraction, $5246/523560 \approx .01$, of the leaves fit in memory.

The worst case number of disk accesses is 2 because in the worst case a leaf and the data associated with the key must be accessed on disk. The average number of disk accesses is $1 \times .01 + 2 \times .99 = 1.99$ which is essentially 2.

5. Label the points

a	b	c	d	e
(10,20)	(20,10)	(15,15)	(5,10)	(10,4)

Sort in both coordinates

$$\begin{array}{cccccc} \text{X-} & d & a & e & c & b \\ \text{Y-} & e & b & d & c & a \end{array}$$

Find root: Widest spread is in the y-coordinate with e and a. The points in the y-dimension using say 12. Split the arrays, Y stays the same, but X has to be "un-shuffled".

$$\begin{array}{cccc|cc} \text{X-} & d & e & b & a & c \\ \text{Y-} & e & b & d & c & a \end{array}$$

Find the left child of root: The widest spread is the x-dimension among d,e,b. Split in the x-dimension say at 15. Split the arrays.

$$\begin{array}{ccc|c|cc} \text{X-} & d & e & b & a & c \\ \text{Y-} & e & d & b & c & a \end{array}$$

Find the left and right children of the left child of the root. The widest spread among d,e is the y-dimension. Split in the y-dimension say at 8. Split the arrays.

$$\begin{array}{lc|c|c|cc}
\text{X-} & \text{e} & \text{d} & \text{b} & \text{a} & \text{c} \\
\text{Y-} & \text{e} & \text{d} & \text{b} & \text{c} & \text{a}
\end{array}$$

Return to the right child of the root to find its two children. The widest spread among a,c is the same in both dimensions. Split in the x-dimension at 7. Split the arrays.

$$\begin{array}{lc|c|c|c|c}
\text{X-} & \text{e} & \text{d} & \text{b} & \text{a} & \text{c} \\
\text{Y-} & \text{e} & \text{d} & \text{b} & \text{a} & \text{c}
\end{array}$$

The resulting tree has the following structure.

```
             y
             12
        /         \
       x           x
       15          7
    /     \      /  \
   y       b    a    c
   8
 /  \
e    d
```