Binary Trees

November 14, 2018

- Sorted Binary Trees
- Implementing Binary Trees
- AVL Trees

1: Sorted Binary Trees

1.1: Behaviour

- If the tree is currently empty, insert the data at the root;
- If the data to be inserted is *less* than the data at the root, insert the data into the left subtree;
- If the data to be inserted is *greater* than the data at the root, insert the data into the right subtree;
- If the data to be inserted and the data at the root are identical (consistently) do one of the following:
  - do not insert it;
  - insert it into the left subtree;
  - insert it into the right subtree.

Example

1.2: Example

"my" "favourite" "module" "is" "algorithms" "processes" "and" "data"

Binary Trees in Java

2: Implementing Binary Trees

2.1: Interface

```
public interface BTree <T extends Comparable<? super T>> {
    // Insert the given value into this tree
    public void insert(T value);

    // Is the tree empty?
    public boolean isEmpty();

    // Does the tree contain the given value?
    public boolean contains(T value);

    ...more methods follow
}
```

Binary Trees in Java

```
// getter and setter methods

// @return the value held at the root of this tree
public T getValue();
// @return the left (right) subtree of this tree
public BTree<T> getLeft();
public BTree<T> getRight();

// set the value at the root
public void setValue(T value);
// set the left (right) subtree
public void setLeft(BTree<T> tree);
public void setRight(BTree<T> tree);
```

Binary Trees in Java

```
// Traversal
public List<T> traverse();
```

Java

2.2: Tree Nodes

```
public class TreeNode<T extends Comparable<? super T>> {
    T value;
    BTree<T> left, right;

    public TreeNode(T value) {
        this.value = value;
        left = new BinaryTree<T>();
        right = new BinaryTree<T>();
    }

    public TreeNode(T value,BTree<T> left,BTree<T> right)
        this.value - value;
        this.left = left;
        this.right = right;
    }
```

...plus getter and setter methods

Java

2.3: Implementation

```
public void insert(T value) {
    if (isEmpty()) {
        root = new TreeNode(value);
    } else if (value.compareTo(getValue()) < 0) {
        getLeft().insert(value);
    } else {
        getRight().insert(value);
    }
}
```

3: AVL Trees

3.1: Worst Case Binary Trees

Binary trees provide fast access to sorted data. However...

> *apparently by chance data entered following general happenstance in juxtaposed key listing may not often provide quality resultant structures thus upsetting very worried ...*

AVL Trees

3.2: Balance Factor

AVL trees[1] are trees that "stay in balance".

The "balance factor" of a node is the difference in height of its two subtrees. An AVL tree will only allow balance factors of $-\mathbf{1}$, $\mathbf{0}$ or $\mathbf{1}$ (left, balanced, or right).

[1](Георгий Адельсон-Вельский & Евгений Ландис, 1962)

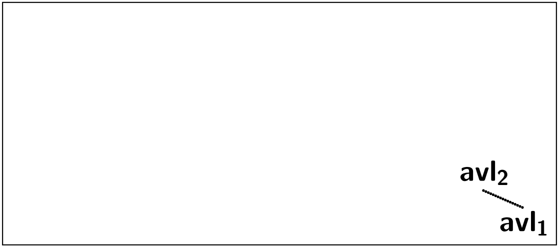
Worst Case AVL

3.3: Worst Case AVL Tree

$\mathbf{avl_1}$

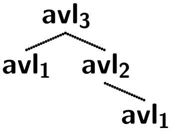
Worst Case AVL

3.3: Worst Case AVL Tree



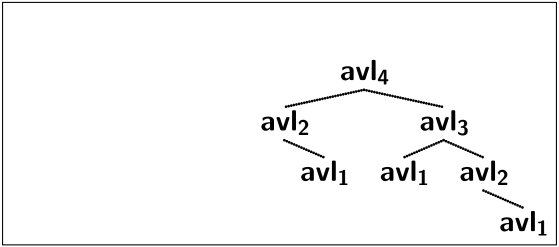
Worst Case AVL

3.3: Worst Case AVL Tree



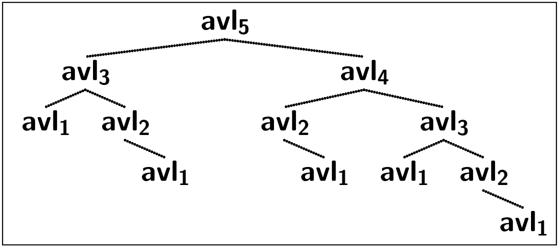
Worst Case AVL

3.3: Worst Case AVL Tree



Worst Case AVL

3.3: Worst Case AVL Tree



Worst Case AVL

Let $\mathbf{n_k}$ be the number of nodes in $\mathbf{avl_k}$. Then

$$\begin{aligned} \mathbf{n_1} &= \mathbf{1} \\ \mathbf{n_2} &= \mathbf{2} \\ \mathbf{n_i} &= \mathbf{n_{i-1}} + \mathbf{n_{i-2}} + \mathbf{1}, \text{ for } \mathbf{i} > \mathbf{2} \end{aligned}$$

$$\text{for large } \mathbf{i}\text{: } \mathbf{n_i} \geq \frac{\mathbf{1.62^{i-2}}}{\sqrt{\mathbf{5}}}$$

or, equivalently

$$\mathbf{d} \leq \mathbf{1.44 \log n}$$

where $\mathbf{d}$ is the height of an AVL tree, and $\mathbf{n}$ is the number of nodes.

3.4: Building AVL Trees

- Each node is aware of its own balance factor
- An insert notifies if the tree inserted into has become deeper
- Node uses this information to adjust balance factor
- If node is now too out of balance it must be rebalanced

Building AVL Trees

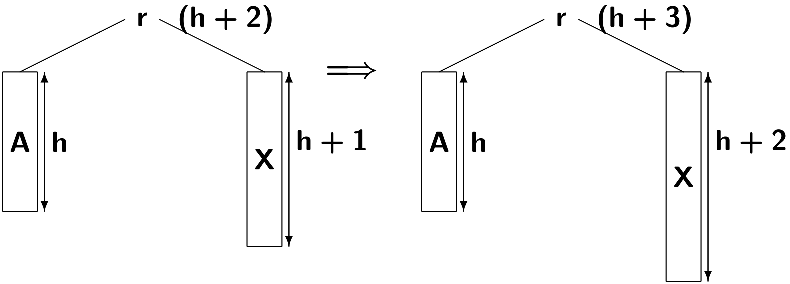
3.4.1: Adjusting the balance factor

If the inserted tree has increased in depth:

- if it *was* balanced it now “leans toward” the side of the inserted tree;
- if it leant in the opposite direction it is now balanced;
- otherwise it is now out of balance, and needs rebalancing.

Rebalancing AVL Trees

3.4.2: Rebalancing AVL trees



- $\text{depth}(\mathsf{A}) = \mathsf{h}$
- $\text{depth}(\mathsf{X}) = \mathsf{h} + 2$ ($\mathsf{h} + 1$ before item was added)
- $\text{depth}(\text{oldtree}) = \mathsf{h} + 2$

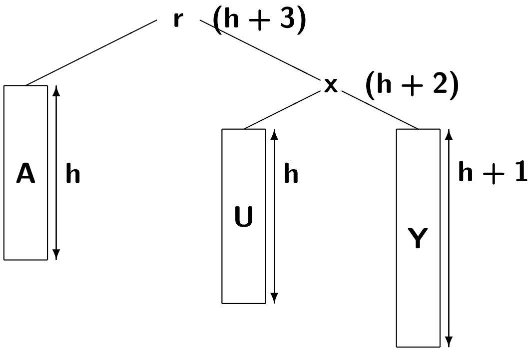
Rebalancing AVL Trees

There are three possibilities:

- $\mathbf{balance(X) = 1}$
- $\mathbf{balance(X) = 0}$
- $\mathbf{balance(X) = -1}$

Rebalancing AVL Trees

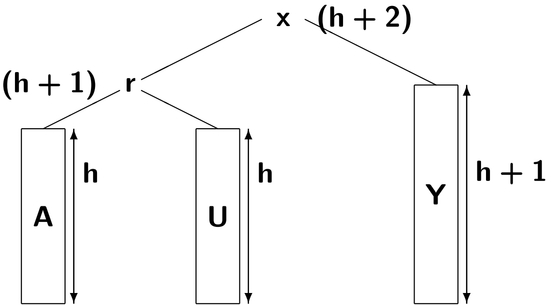
3.4.2 A: balance(X) = 1



$$\mathsf{A} < \mathsf{r} < \mathsf{U} < \mathsf{x} < \mathsf{Y}$$

$$\mathrm{depth}(\mathsf{A}) = \mathsf{h},\ \mathrm{depth}(\mathsf{U}) = \mathsf{h},\ \mathrm{depth}(\mathsf{Y}) = \mathsf{h} + 1$$

Rebalancing AVL Trees



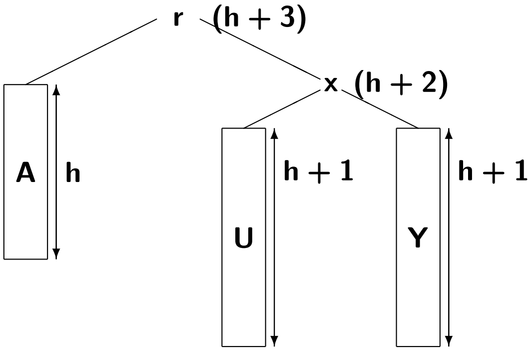
$$\mathsf{A} < \mathsf{r} < \mathsf{U} < \mathsf{x} < \mathsf{Y}$$

$$\text{balance}(\mathsf{r}) = \text{balance}(\mathsf{x}) = 0$$

$$\text{depth}(\text{newtree}) = \text{depth}(\text{oldtree}) = \mathsf{h} + 2$$

Rebalancing AVL Trees

3.4.2 B: $\mathbf{balance}(\mathsf{X}) = 0$

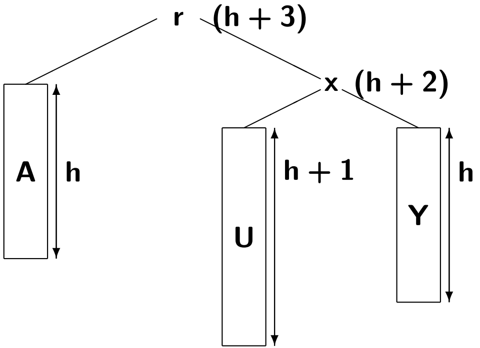


$$\mathbf{depth}(\mathsf{A}) = \mathsf{h},\ \mathbf{depth}(\mathsf{U}) = \mathbf{depth}(\mathsf{Y}) = \mathsf{h} + 1$$

Impossible!

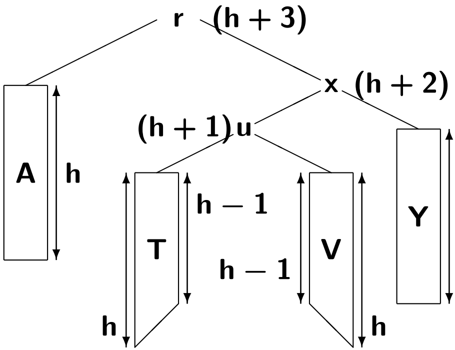
Rebalancing AVL Trees

3.4.2 C: $\mathrm{balance}(\mathbf{X}) = -1$



$$\mathrm{depth}(\mathbf{A}) = \mathbf{h},\ \mathrm{depth}(\mathbf{U}) = \mathbf{h} + 1,\ \mathrm{depth}(\mathbf{Y}) = \mathbf{h}$$

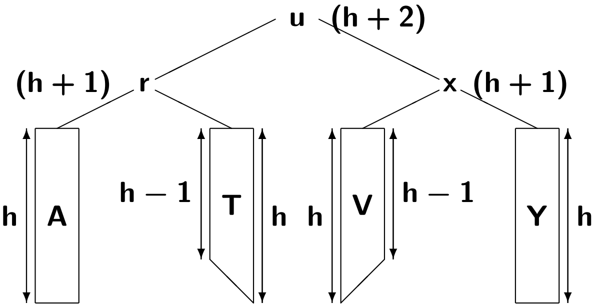
Rebalancing AVL Trees



$$A < r < T < u < V < x < Y$$

- $\text{depth}(A) = \text{depth}(Y) = h$
- $\text{balance}(u) = 1 \Rightarrow \text{depth}(T, V) = (h - 1, h)$
- $\text{balance}(u) = -1 \Rightarrow \text{depth}(T, V) = (h, h - 1)$

Rebalancing AVL Trees



- $\mathbf{balance(u) = 0}$
- $\mathbf{balance(u)}$ was $\mathbf{1 \Rightarrow balance(r, x) = (-1, 0)}$
- $\mathbf{balance(u)}$ was $\mathbf{-1 \Rightarrow balance(r, x) = (0, 1)}$
- $\mathbf{depth(newtree) = depth(oldtree)}$