AVL Trees

Subhabrata Samajder

IIIT, Delhi

Winter Semester,
12^{th} April, 2023

AVL Trees

Binary Tree: Problems

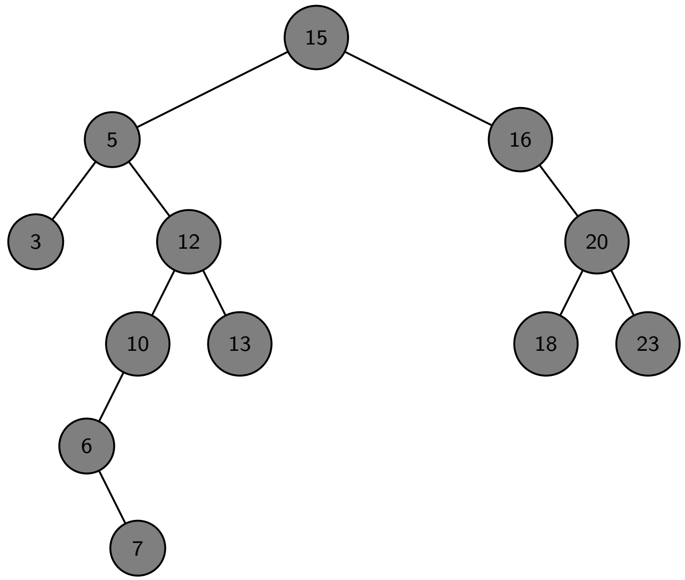
- The height of the tree depends on input sequence!
- If the input sequence is either *sorted* or *reverse-sorted*, then the BST is essentially a linked list.
- **Worst-case complexity:** $\mathcal{O}(n)$.
- **Motivation:** Would like to take advantage of the $\mathcal{O}(\log n)$ search time that a balanced tree can provide.

Full and Complete Binary Tree: Recall

- A full binary tree is one in which all nodes have either two children or none.

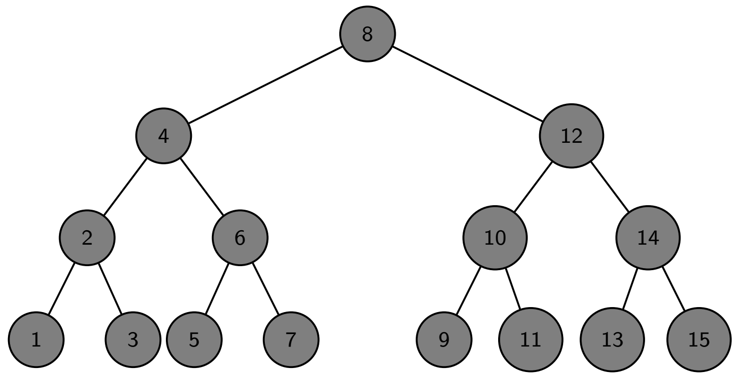
- In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible.

What is a balanced tree?



- This is not a balanced tree.
- Since the left-sub tree is deeper than the right sub-tree.

What is a balanced tree?



- But this is a balanced tree.
- Since left-sub tree is the same depth as the right sub-tree.

Problems With Balanced Trees

- A perfectly balanced tree is a far too restrictive a condition.

Loosening The Rules

- **Idea:** Allow the height of the left and right sub-trees to differ by at most one?
- This solution was proposed by G. M. **A**delson-**V**elskii and E. M. **L**andis.
- Henceforth known as the **AVL** Tree

Why should we use it?

- It does not yield an average search in $\log n$ comparisons.
- However, we can achieve a solution in $1.44 \log n$ comparisons (pretty close!!).
- Relatively easy to implement (there are a maximum of two modifications to the tree after an insert)

Overview of AVL Trees

- **It is Height-balanced:** For every node in the tree, the height of it's left and right sub-trees differ by at most one.
- If required, rebalance the tree after each insertion or deletion to keep the tree balanced.
- This is done by performing *rotations*.
- Insertion and deletion are handled in the same manner as with an ordinary BST.
- A value is kept in each node to denote the balance condition of that node or the current height of the node (depending on implementation).

The Balancing Act

- Use single and double rotations to keep the balance.
- The balance factor of a node N is defined as

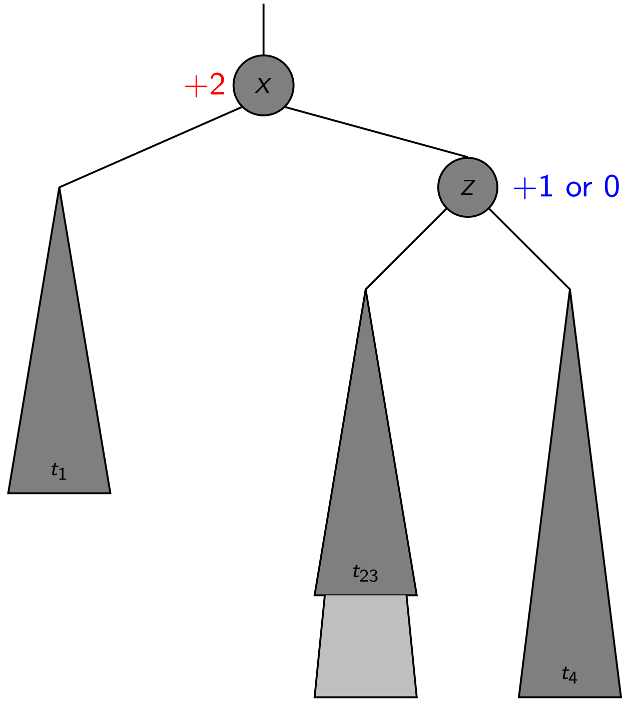
$$BalanceFactor(N) = Height(RightSubtree(N)) - Height(LeftSubtree(N)).$$

- A binary tree is an **AVL tree** if

$$BalanceFactor(N) \in \{-1, 0, 1\}$$

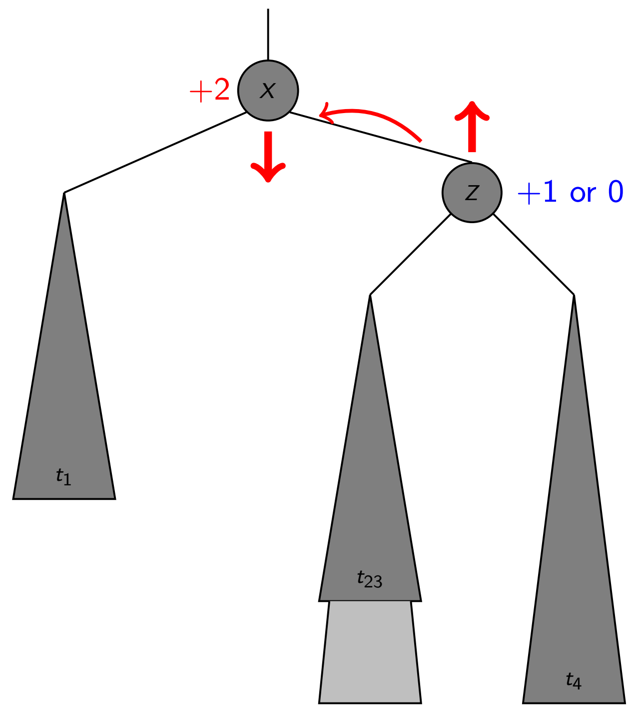
  holds for every node N in the tree.
- Left-heavy Node: $BalanceFactor(N) < 0$.
- Right-heavy Node: $BalanceFactor(N) > 0$.
- Balanced Node: $BalanceFactor(N) = 0$.

The Balancing Act: Single Rotation



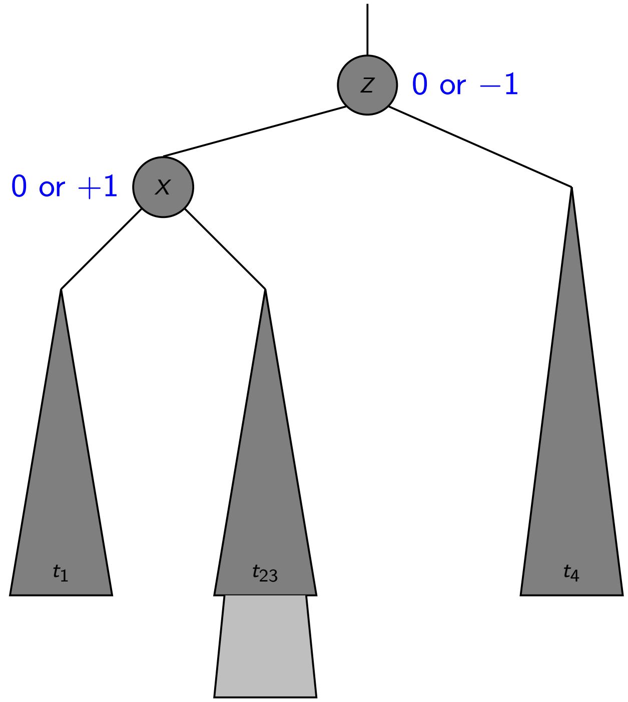
- Node X has two child trees with a balance factor of $+2$.
- The left child t_{23} of z is not higher than its sibling t_4.
    - Can happen by a height increase of t_4 or by a height decrease of t_1.
- **Note:** t_{23} can have the same height as t_4.
- The mirror case is easily derived.

The Balancing Act: Single Rotation



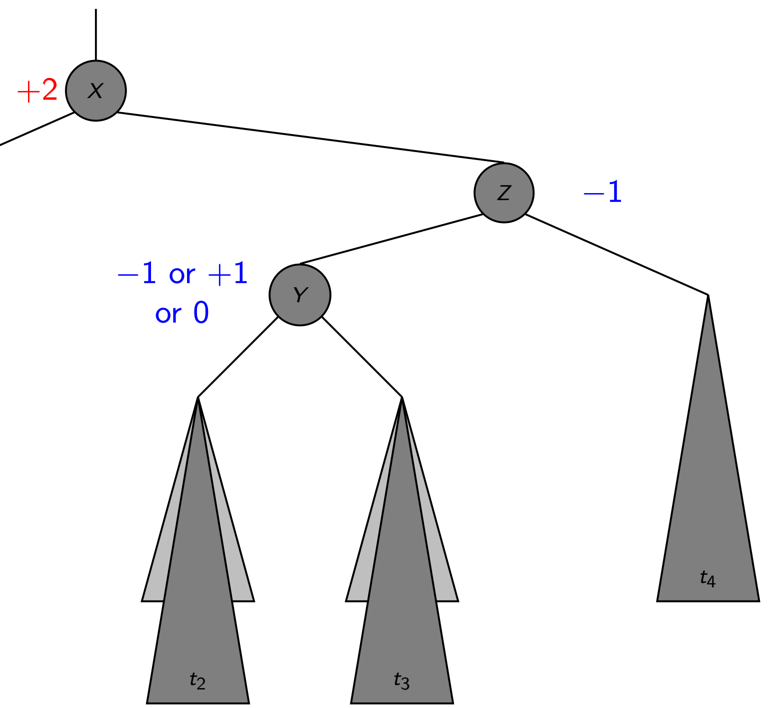
- Node X has two child trees with a balance factor of $+2$.
- The left child t_{23} of z is not higher than its sibling t_4.
  - Can happen by a height increase of t_4 or by a height decrease of t_1.
- **Note:** t_{23} can have the same height as t_4.
- The mirror case is easily derived.

The Balancing Act: Single Rotation



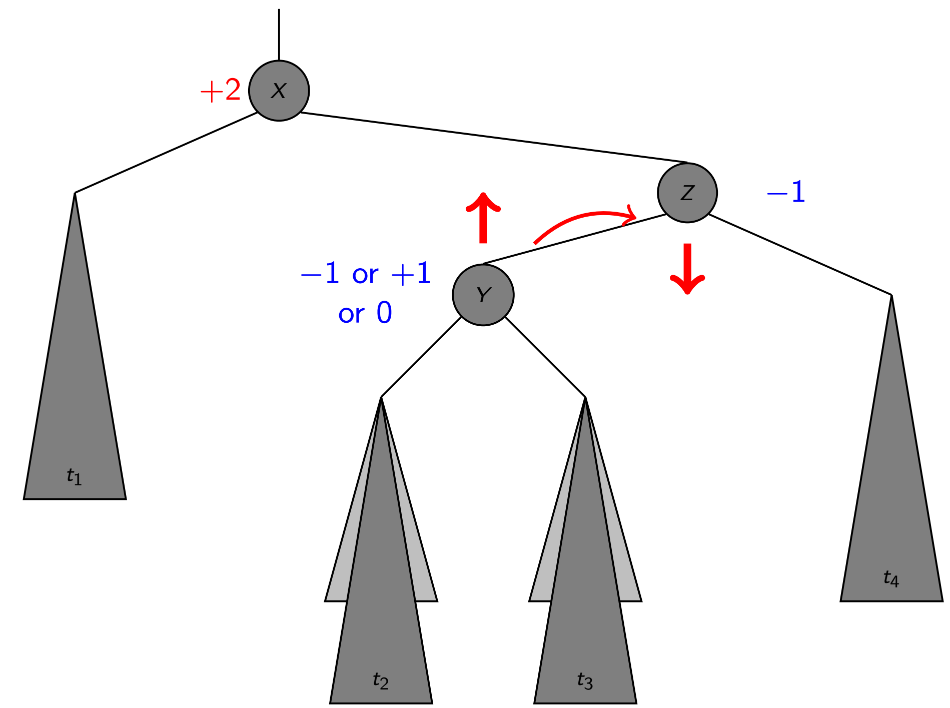
- Node X has two child trees with a balance factor of $+2$.
- The left child t_{23} of z is not higher than its sibling t_4.
  - Can happen by a height increase of t_4 or by a height decrease of t_1.
- **Note:** t_{23} can have the same height as t_4.
- The mirror case is easily derived.

The Balancing Act: Double Rotation



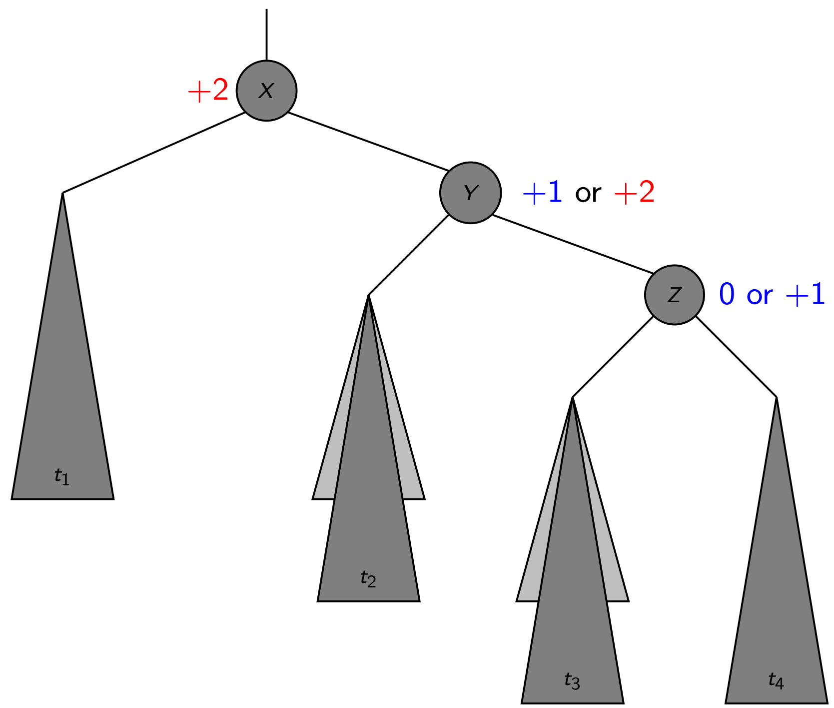
- Node X has two child trees with a balance factor of $+2$.
- The left child Y of Z is higher than its sibling t_4.
  - Can happen by the insertion of Y itself or a height increase of one of its subtrees t_2 or t_3 or by a height decrease of subtree t_1.
- **Note:** t_2 and t_3 may also be of same height.
- The mirror case is easily derived.

The Balancing Act: Double Rotation



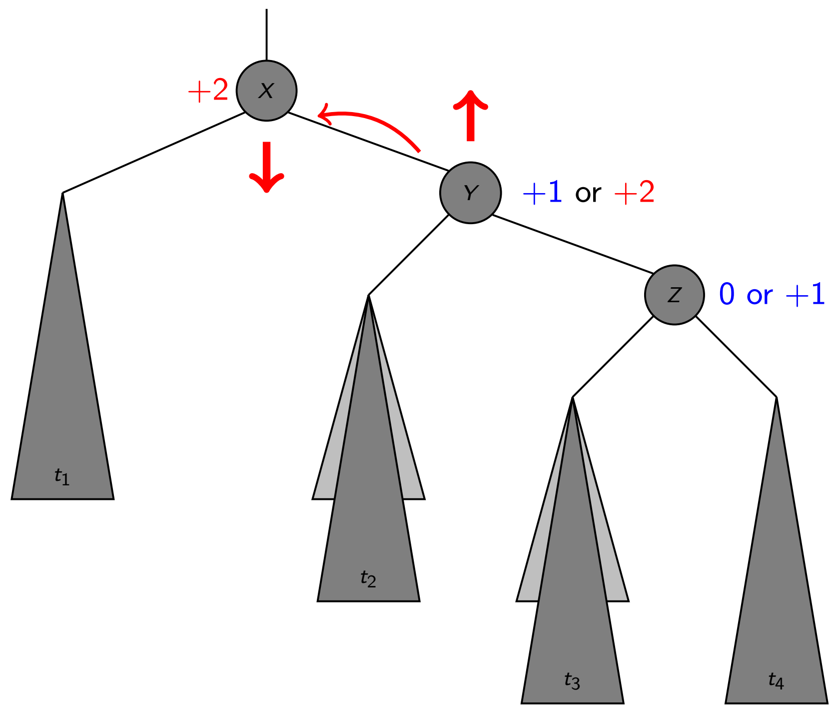
- Node X has two child trees with a balance factor of $+2$.
- The left child Y of Z is higher than its sibling t_4.
  - Can happen by the insertion of Y itself or a height increase of one of its subtrees t_2 or t_3 or by a height decrease of subtree t_1.
- **Note:** t_2 and t_3 may also be of same height.
- The mirror case is easily derived.

The Balancing Act: Double Rotation



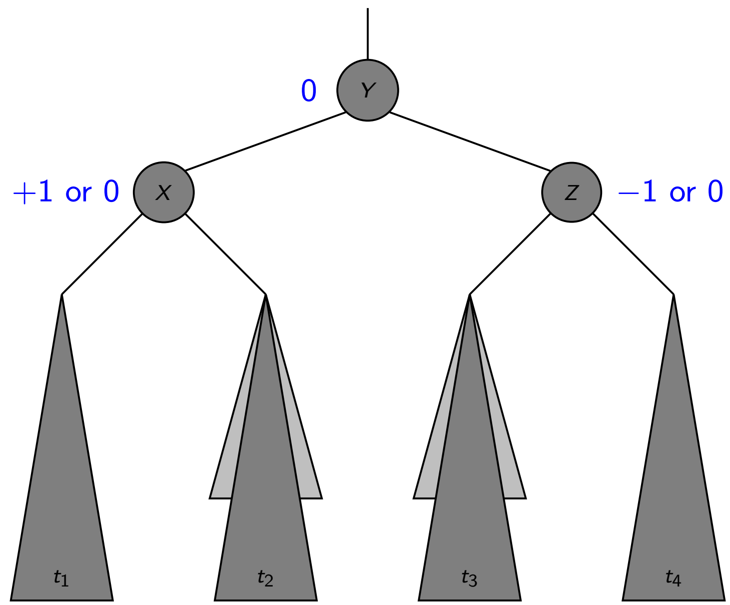
- Node X has two child trees with a balance factor of $+2$.
- The left child Y of Z is higher than its sibling t_4.
  - Can happen by the insertion of Y itself or a height increase of one of its subtrees t_2 or t_3 or by a height decrease of subtree t_1.
- **Note:** t_2 and t_3 may also be of same height.
- The mirror case is easily derived.

The Balancing Act: Double Rotation



- Node X has two child trees with a balance factor of $+2$.
- The left child Y of Z is higher than its sibling t_4.
  - Can happen by the insertion of Y itself or a height increase of one of its subtrees t_2 or t_3 or by a height decrease of subtree t_1.
- **Note:** t_2 and t_3 may also be of same height.
- The mirror case is easily derived.

The Balancing Act: Double Rotation



- Node X has two child trees with a balance factor of $+2$.
- The left child Y of Z is higher than its sibling t_4.
  - Can happen by the insertion of Y itself or a height increase of one of its subtrees t_2 or t_3 or by a height decrease of subtree t_1.
- **Note:** t_2 and t_3 may also be of same height.
- The mirror case is easily derived.

Exercise

1. **Try sequence:** 20, 10, 30, 8, 6^*, 9^+

   $*$: will cause a single left rotation (not effecting root)
   $+$: will cause a double rotation (effects root)

Number of Rotations

- At this point you may think that the algorithm is going to be really complex (due to all the rotations and manipulations).

- **# rotations per insertion:** at most one (single or double).
  - Note that before insert the tree was height balanced.
  - $\therefore$ the height balance of each node is at most 1.
  - After the insertion the height of the tree is increased by one.
  - But it is decreased back by one via a rotation.
  - So the tree remains at its original height and no more changes to the bigger tree is needed.

Height of an AVL Tree

- x: The root of an AVL tree of height h.
- N_h: Minimum number of nodes in an AVL tree of height h
- Clearly, by definition $N_i \geq N_{i-1}$. Therefore, we have

$$\begin{aligned} N_h &\geq N_{h-1} + N_{h-2} + 1 \\ &\geq 2N_{h-2} + 1 \\ &> 2N_{h-2}. \end{aligned}$$

- By repeated substitution, we obtain the general form

$$N_h > 2^i N_{h-2i}.$$

- **Boundary conditions:** $N_1 = 1$ and $N_2 = 2$.
- Which implies that $h = O(\log N_h)$.
- Thus, searching, insertion, deletion takes $\mathcal{O}(\log N)$ time.

Height of a Node in an AVL Tree

- Height of a node
  - The height of a leaf is 1. The height of a null pointer is zero.
  - The height of an internal node is the maximum height of its children plus 1

Note: This definition is different from our earlier definition where the height of a leaf was zero.

Thank You for your kind attention!

Books and Other Materials Consulted

1. AVL Trees portion taken from Prof. Roy P. Pargas's webpage.

Questions!!