AVL Tree: insert & delete

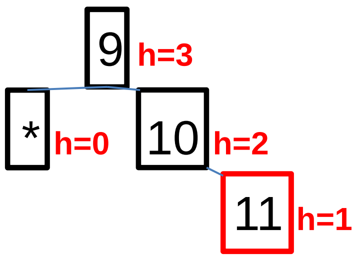
Simple Examples

Consider add & delete

- Add
  - When you have worked out the cases for add…
    - **Outside = single rotation**
    - **Inside = double rotation**
  - You can then treat delete as equivalent to add and derive the **general case** for add and delete to rebalance the tree

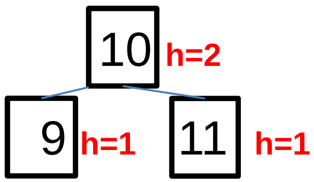
Add – outside → simple rotation

Look at simple cases



bf = -2

SLR(T)



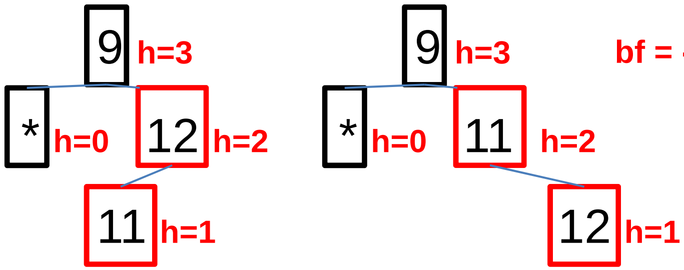
bf = 0

- **Add 11 to (*,9,10)**
- Adding is on the **OUTSIDE**
- Balance factor (bf) **h(L)-h(R)**
  - bf = 0 – 2 → -2 **→ ?LR**
- Now look at the R-child (10)
  - bf = -1 **→ SLR**
- <u>**Single Left Rotation**</u> required to re-balance the tree i.e. to maintain the AVL constraint.
- <u>**Single Right Rotation**</u>

SRR is the mirror image

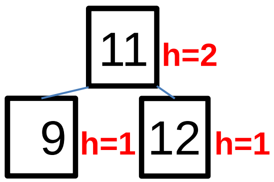
Add – inside → double rotation

Look at simple cases



bf = -2

DLR(T) = SRR(RC(T)) + SLR(T)



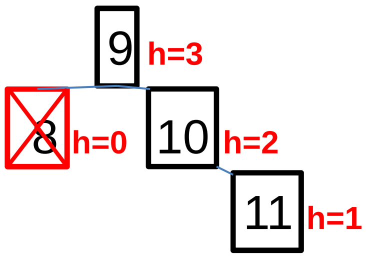
bf = 0

- **Add 11 to (*,9,12)**
- Adding is on the **INSIDE**
- Balance factor (bf) **h(L)-h(R)**
  - bf = 0 – 2 → -2 **→ ?LR**
- Now look at the R-child (12)
  - bf = 1 **→ DLR**
- **Double Left Rotation** required to re-balance the tree i.e. to maintain the AVL constraint.
- **Double Right Rotation**

  DRR is the mirror image

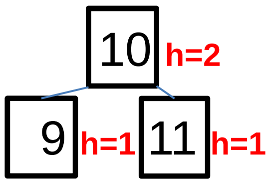
Delete → simple rotation (i)

Look at simple cases



bf = -2

SLR(T)



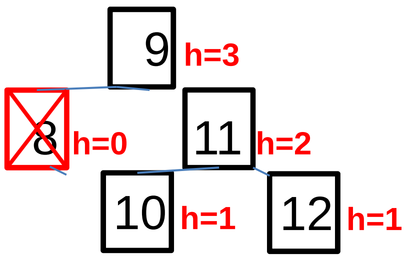
bf = 0

- **Delete 8 from (8,9,(*,10,11))**
- **Equivalent to add 11 to (*,9,10)**
- Adding is on the **OUTSIDE**
- Balance factor (bf) **h(L)-h(R)**
  - bf = 0 – 2 → -2 **→ ?LR**
- Now look at the R-child (10)
  - bf = -1 **→ SLR**
- **<u>Single Left Rotation</u>** required to re-balance the tree i.e. to maintain the AVL constraint.
- **<u>Single Right Rotation</u>**

SRR is the mirror image

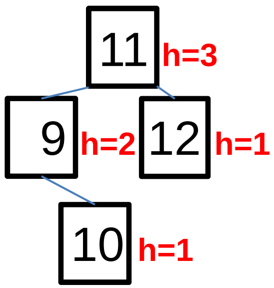
Delete → simple rotation (ii)

Look at simple cases



bf = -2

SLR(T)



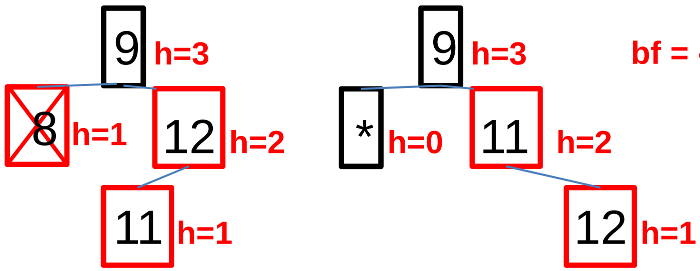
bf = 1

- **Delete 8 from (8,9,(10,11,12))**
- Balance factor (bf) **h(L)-h(R)**
  - bf = 0 – 2 → -2 **→ ?LR**
- Now look at the R-child (11)
  - bf = 0 **→ SLR**
- **Single Left Rotation** required to re-balance the tree i.e. to maintain the AVL constraint.
- **Single Right Rotation**

SRR is the mirror image

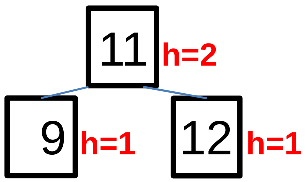
Delete → double rotation

Look at simple cases



bf = -2

DLR(T) = SRR(RC(T)) + SLR(T)



bf = 0

- **Delete 8 from (8,9,(11,12,*))**
- **Equivalent to add 11 to (*,9,12)**
- Adding is on the **INSIDE**
- Balance factor (bf) **h(L)-h(R)**
  - bf = 0 – 2 → -2 **→ ?LR**
- Now look at the R-child (12)
  - bf = 1 **→ DLR**
- **Double Left Rotation** required to re-balance the tree i.e. to maintain the AVL constraint.
- **Double Right Rotation**

  DRR is the mirror image

Comparing add and delete (i)

Look at simple cases

9

~~8~~ 10

11

SLR(T)

10

9 11

This represents 2 cases

1. **Add 11 to (*,9,10)**
2. **Remove 8 from (8,9,(*,10,11))**

- Both use a **Single Left Rotation** to re-balance the tree i.e. to maintain the AVL constraint.
- Adding Is on the **OUTSIDE**
- Single Right Rotation

SRR is the mirror image

Comparing add and delete (ii)

Look at simple cases

9

~~8~~ 11

10 12

SLR(T)

11

9 12

10

Think about this example

Delete 8 from (8,9,(10,11,12))

Is like

Add 12 to (*,9,11) ➔

SLR ➔ (9,11,12)

Add 10 gives

((*,9,10),11,12)

Comparing add and delete (iii)

Look at simple cases

9

~~8~~ 11

10

DLR(T)

10

9 11

DLR(T) = SRR(RC(T)) + SLR(T)

This represents 2 cases

1. **Add 10 to (*,9,11)**
2. **Remove 8 from (8,9,(10,11,*))**

- Both use a **Double Left Rotation** to re-balance the tree i.e. to maintain the AVL constraint.
- Adding Is on the **INSIDE**
- <u>Double Right Rotation</u> **DDR** is the mirror image

Next step – write the lab code

- The above is sufficient info to produce the code for the lab
- It is worth thinking where in the code the rebalancing function is called
- **Challenge**: it is possible to have just one call to the rebalancing function ☺

In summary

- Start with simple examples
- Derive general principles
- Balancing may be done **<u>just after</u>** the ADD / REMOVE
- Think carefully **<u>where</u>** you re-balance!
- Hint: in one place only in the BST code
- It’s a tree – balance takes 4 lines! ☺