

COMP2521 24T3

AVL Trees

Hao Xue

`cs2521@cse.unsw.edu.au`

AVL Trees

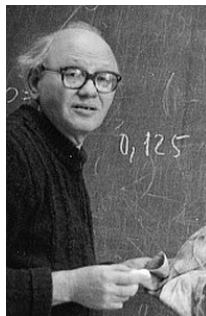
Insertion

Search

Deletion

Summary

Invented by Georgiy Adelson-Velsky and Evgenii Landis in 1962



AVL Trees

Insertion

Search

Deletion

Summary

Approach:

- Keep tree height-balanced
- Repair balance as soon as imbalance occurs
 - During insertion or deletion
- Repairs are done locally, not by restructuring entire tree

Height of an AVL tree

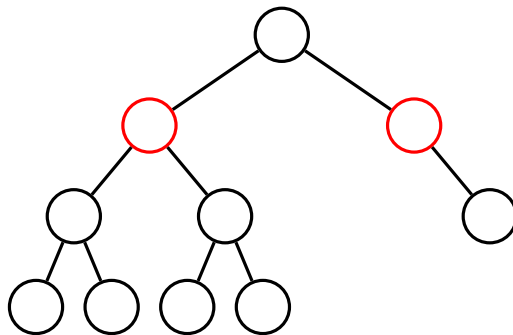
Since AVL trees are always height-balanced,
the height of an AVL tree is guaranteed to be at most
 $\log_{\phi}(n+2) - 0.3277$ (where ϕ is the golden ratio)
 $\approx 1.4404 \log_2(n+2) - 0.3277 = O(\log n)$

If you are interested in this:

- <https://people.csail.mit.edu/alinush/6.006-spring-2014/avl-height-proof.pdf>
- <https://www.youtube.com/watch?v=UJqkhRzBR9E>

One More Note:

AVL trees are not necessarily size-balanced.
For example, the following is a perfectly valid AVL tree:



AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Analysis

Search

Deletion

Summary

Method:

- Insert item recursively
- Check balance at each node along the insertion path *in reverse*
 - i.e., from bottom to top
- Fix imbalances as they are found

AVL Trees

Insertion

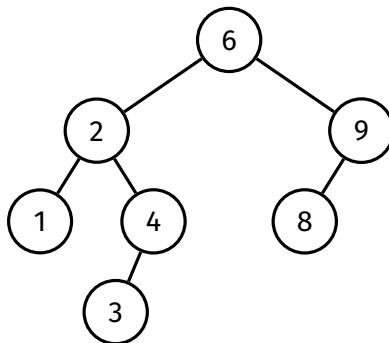
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Example: Insert 5 into this tree



AVL Trees

Insertion

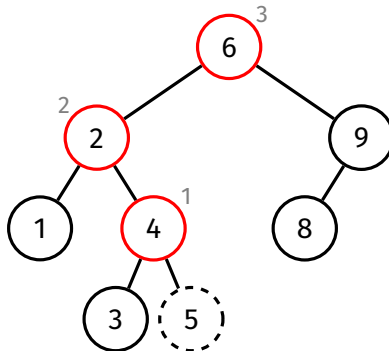
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Example: Insert 5 into this tree



Balance must be checked at 4, then at 2, then at 6

AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Analysis

Search

Deletion

Summary

How to check balance along insertion path *in reverse*?

- Perform balance checking as a *postorder* operation in the insertion function
 - In other words - add balance checking code *below* recursive calls
 - Insert operation after recursion: perform operation during unwinding
 - In this case, the balance checking will be done from bottom to top (postorder)

AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Analysis

Search

Deletion

Summary

Outline of insertion process:

- ① if the tree is empty:
 - return new node
- ② insert recursively
- ③ check (and fix) balance
- ④ return root of updated tree

AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Analysis

Search

Deletion

Summary

There are 4 rebalancing cases:

Left Left

Left Right

Right Left

Right Right

AVL Trees

Insertion

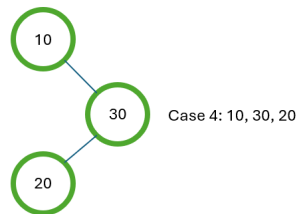
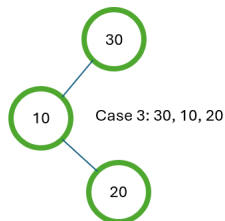
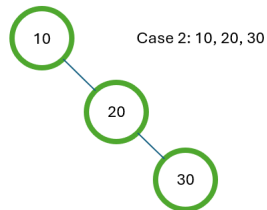
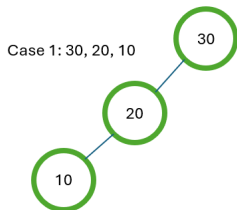
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Considering three items: 10, 20, 30
There are 6 insertion orders.



AVL Trees

Insertion

Pseudocode
Rebalancing
Height data
Analysis

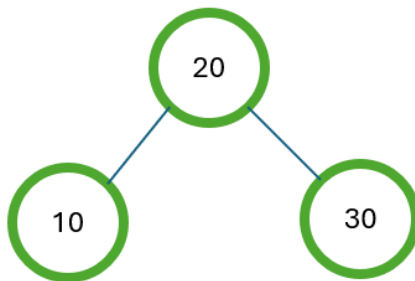
Search

Deletion

Summary

The rest 2 cases (balanced)

Best Case: 20, 30, 10 (20, 10, 30)



AVL Trees

Insertion

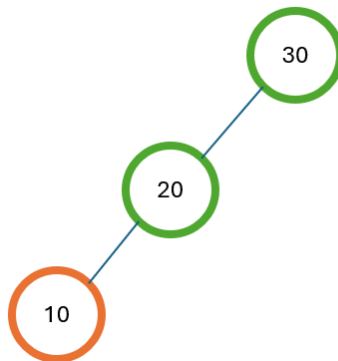
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 1: 30, 20, 10



LL-imbalance

AVL Trees

Insertion

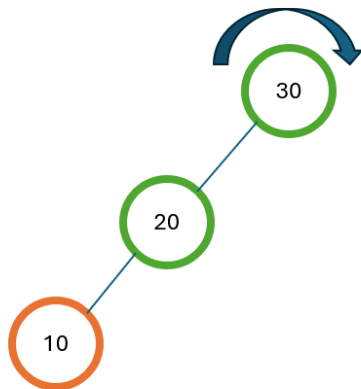
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 1: 30, 20, 10



LL-rotation

AVL Trees

Insertion

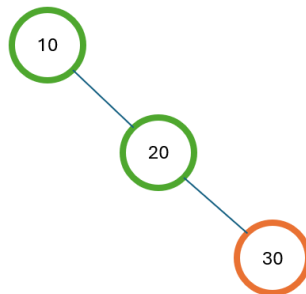
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 2: 10, 20, 30



RR-imbalance

AVL Trees

Insertion

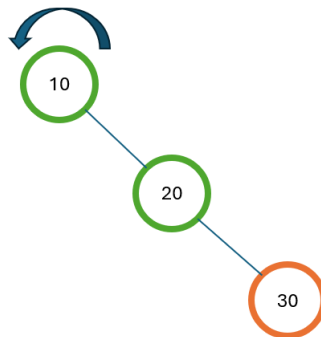
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 2: 10, 20, 30



RR-rotation

AVL Trees

Insertion

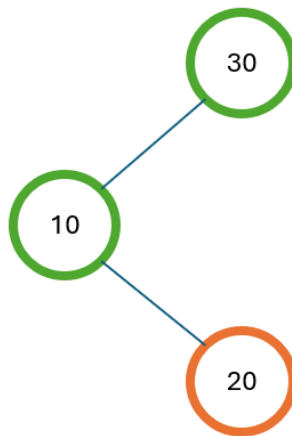
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 3: 30, 10, 20



LR-imbalance

AVL Trees

Insertion

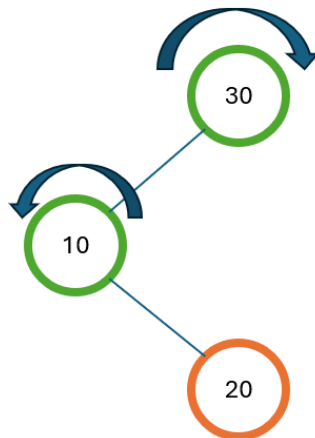
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 3: 30, 10, 20



LR-rotation

AVL Trees

Insertion

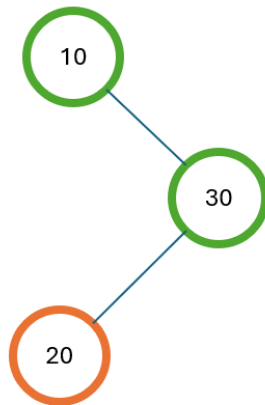
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 4: 10, 30, 20



RL-imbalance

AVL Trees

Insertion

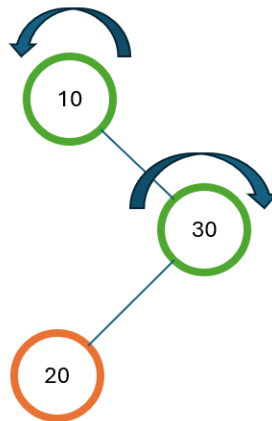
Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

Case 4: 10, 30, 20



RL-rotation

AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Analysis

Search

Deletion

Summary

```
avlInsert( $t$ ,  $v$ ):  
    Input: AVL tree  $t$ , item  $v$   
    Output:  $t$  with  $v$  inserted  
  
    if  $t$  is empty:  
        return new node containing  $v$   
    else if  $v < t \rightarrow \text{item}$ :  
         $t \rightarrow \text{left} = \text{avlInsert}(t \rightarrow \text{left}, v)$   
    else if  $v > t \rightarrow \text{item}$ :  
         $t \rightarrow \text{right} = \text{avlInsert}(t \rightarrow \text{right}, v)$   
    else:  
        return  $t$   
  
    return avlRebalance( $t$ )
```

AVL Trees

Insertion

Pseudocode
Rebalancing
Height data
Analysis

Search

Deletion

Summary

```
avlRebalance(t):
```

```
    Input: possibly unbalanced tree t
```

```
    Output: balanced t
```

```
    bal = balance(t)
```

```
    if bal > 1:
```

```
        if balance(t->left) < 0:
```

```
            t->left = rotateLeft(t->left)//case 3
```

```
        t = rotateRight(t)
```

```
    else if bal < -1:
```

```
        if balance(t->right) > 0:
```

```
            t->right = rotateRight(t->right)//case 4
```

```
        t = rotateLeft(t)
```

```
    return t
```

```
balance(t):
```

```
    Input: tree t
```

```
    Output: balance factor of t
```

```
    return height(t->left) - height(t->right)
```

AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

Analysis

Search

Deletion

Summary

There are 4 rebalancing cases:

Left Left

Left Right

Right Left

Right Right

AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

Analysis

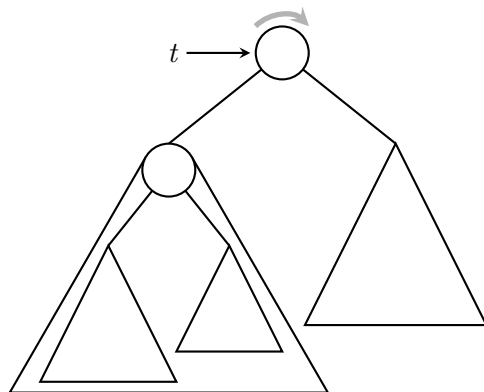
Search

Deletion

Summary

Left Left

```
bal = balance(t)
if bal > 1: (true)
    if balance(t->left) < 0: (false)
        t->left = rotateLeft(t->left)
    t = rotateRight(t)
else if bal < -1:
    if balance(t->right) > 0:
        t->right = rotateRight(t->right)
    t = rotateLeft(t)
```



AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

Analysis

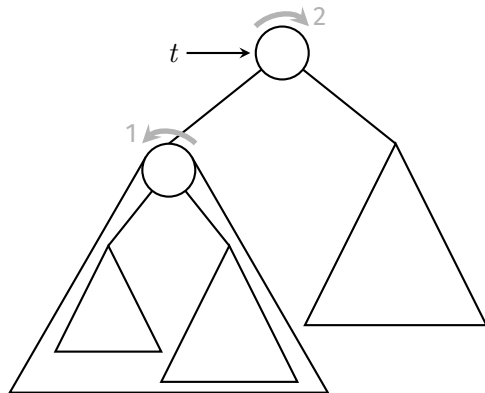
Search

Deletion

Summary

Left Right

```
bal = balance(t)
if bal > 1: (true)
    if balance(t->left) < 0: (true)
        t->left = rotateLeft(t->left)
    t = rotateRight(t)
else if bal < -1:
    if balance(t->right) > 0:
        t->right = rotateRight(t->right)
    t = rotateLeft(t)
```



AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

Analysis

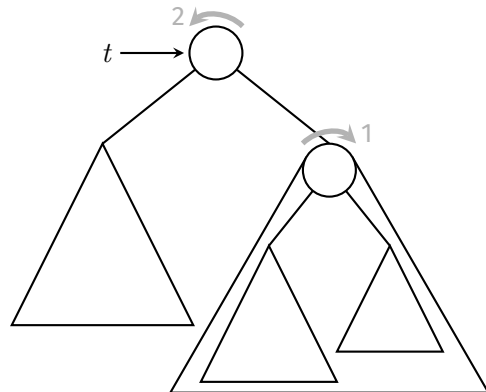
Search

Deletion

Summary

Right Left

```
bal = balance(t)
if bal > 1: (false)
    if balance(t->left) < 0:
        t->left = rotateLeft(t->left)
    t = rotateRight(t)
else if bal < -1: (true)
    if balance(t->right) > 0: (true)
        t->right = rotateRight(t->right)
    t = rotateLeft(t)
```



AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

Analysis

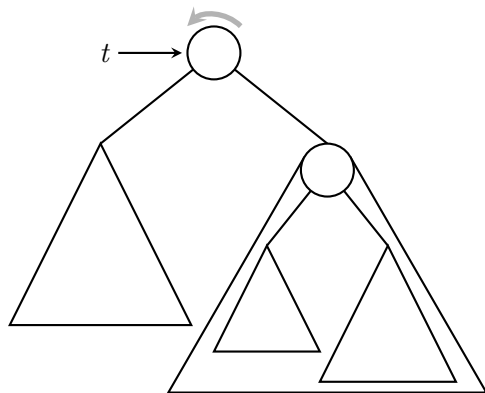
Search

Deletion

Summary

Right Right

```
bal = balance(t)
if bal > 1: (false)
    if balance(t->left) < 0:
        t->left = rotateLeft(t->left)
    t = rotateRight(t)
else if bal < -1: (true)
    if balance(t->right) > 0: (false)
        t->right = rotateRight(t->right)
    t = rotateLeft(t)
```



AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

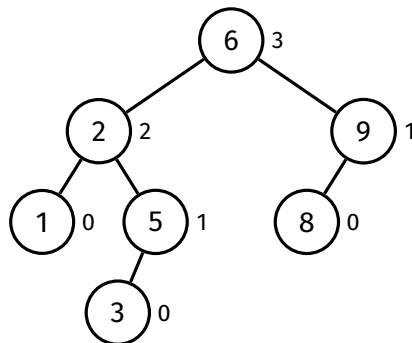
Analysis

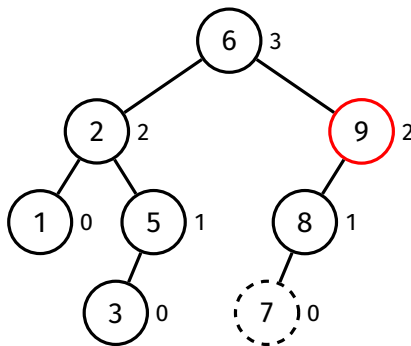
Search

Deletion

Summary

Insert 7 into this tree:





Check for balance at 8, then at 9, then at 6.

9 is unbalanced.

AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

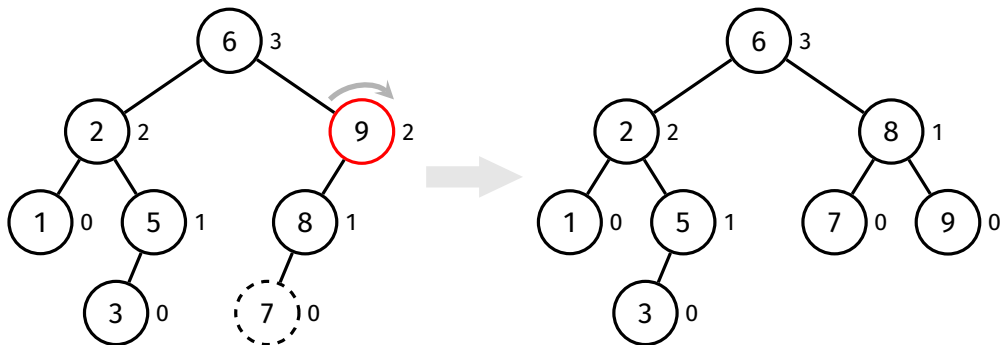
Height data

Analysis

Search

Deletion

Summary



AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

Height data

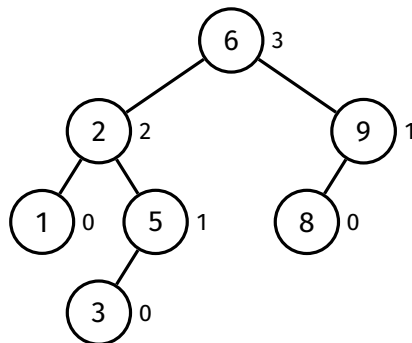
Analysis

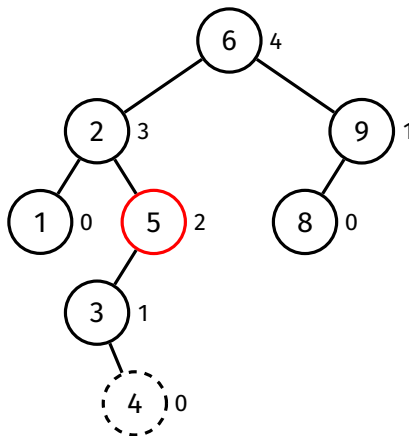
Search

Deletion

Summary

Insert 4 into this tree:





Check for balance at 3, then at 5, then at 2, then at 6.

5 is unbalanced.

AVL Trees

Insertion

Pseudocode

Rebalancing

Examples

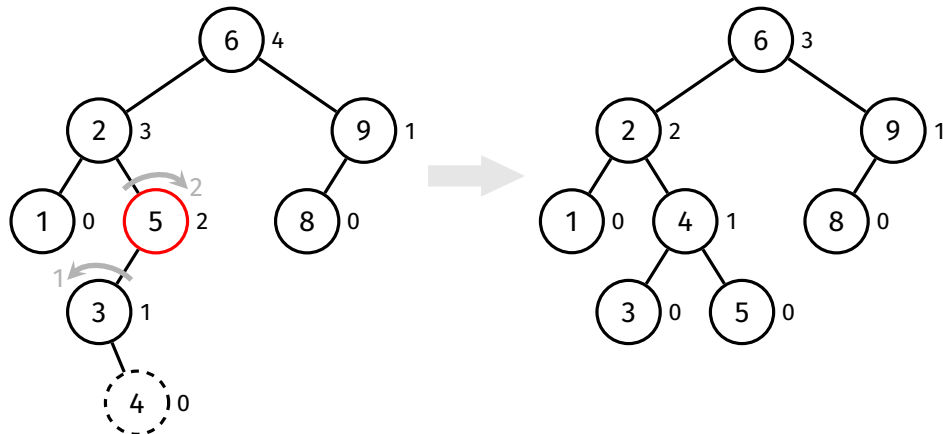
Height data

Analysis

Search

Deletion

Summary



AVL tree insertion requires balance checking
at each node on the insertion path...

...which requires the height of many subtrees to be computed

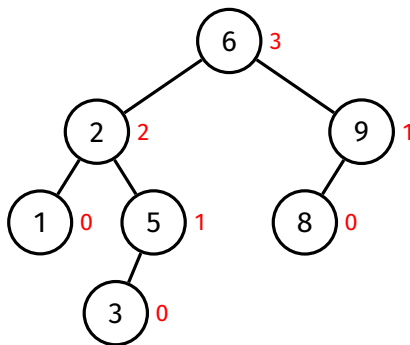
In an ordinary binary search tree, computing the height is $O(n)$!
(need to traverse whole (sub)tree)

Solution:

For each node, store the height of its subtree in the node itself:

```
struct node {  
    int item;  
    struct node *left;  
    struct node *right;  
    int height;  
};
```

Height of each node's subtree is stored in the node itself



AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Maintenance

Analysis

Search

Deletion

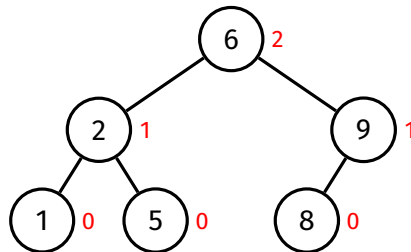
Summary

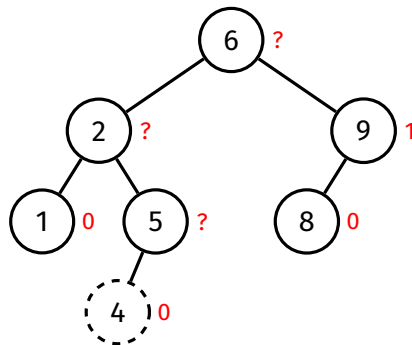
When does height data need to be maintained?

- Whenever a node is inserted
 - Heights of all ancestors may be affected
- Whenever a rotation is performed
 - Heights of original root and new root may be affected

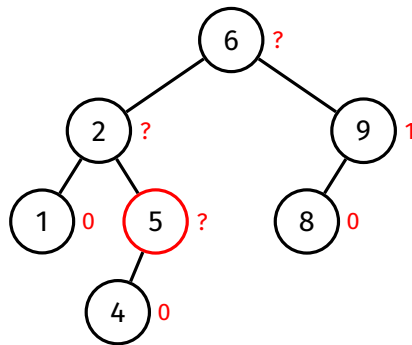
Whenever a node is inserted...
...heights of all ancestors may be affected

Example: Insert 4 into this tree





Recompute height of each ancestor (from bottom to top)
using the heights stored in its children.



The heights of 5's children are 0 and -1 (empty tree).

Thus, the height of 5 is $\max(0, -1) + 1 = 1$.

AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

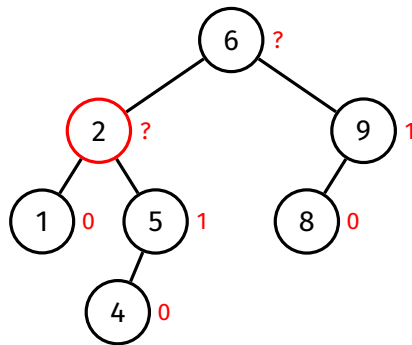
Maintenance

Analysis

Search

Deletion

Summary



The heights of 2's children are 0 and 1.

Thus, the height of 2 is $\max(0, 1) + 1 = 2$.

AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

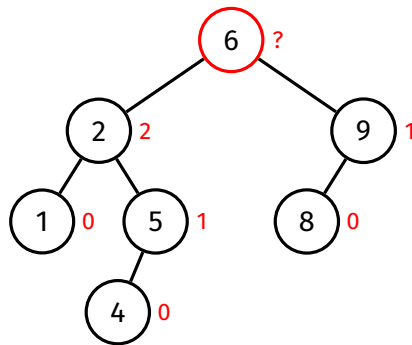
Maintenance

Analysis

Search

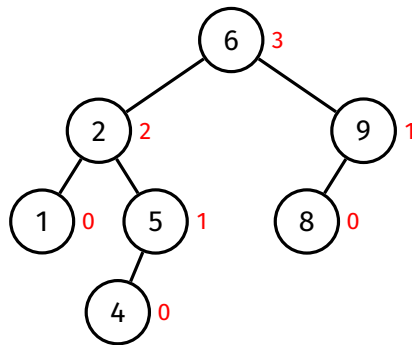
Deletion

Summary



The heights of 6's children are 2 and 1.

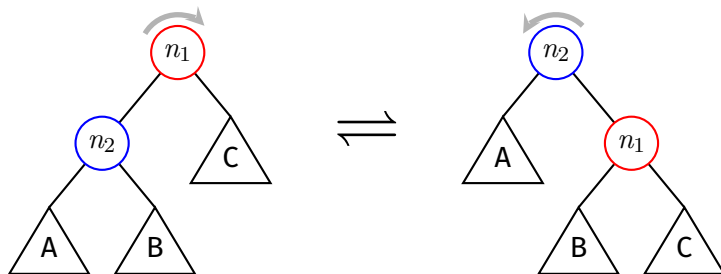
Thus, the height of 6 is $\max(2, 1) + 1 = 3$.



Done.

Note that recomputing the height of each node was done in $O(1)$ time.

Whenever a rotation is performed...
...heights of original root and new root may be affected



AVL Trees

Insertion

Pseudocode

Rebalancing

Height data

Maintenance

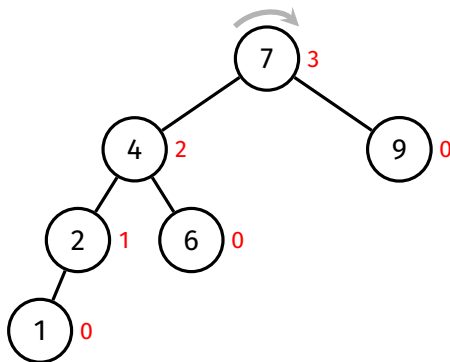
Analysis

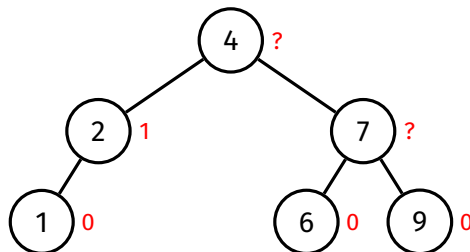
Search

Deletion

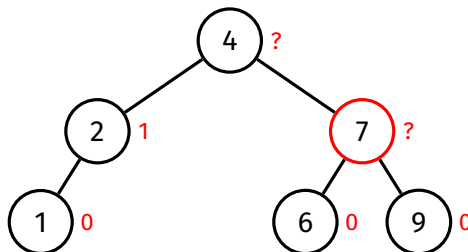
Summary

Example: Perform a right rotation at 7



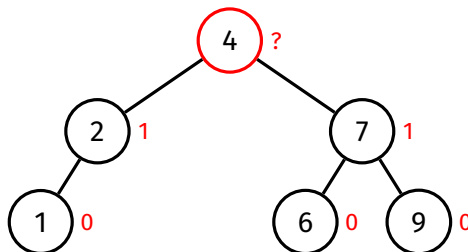


Recompute height of original root
then recompute height of new root
using the heights stored in their children.



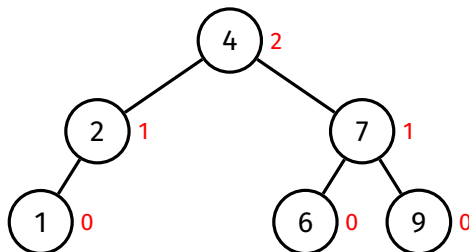
The height of 7's children are 0 and 0.

Thus, the height of 7 is $\max(0, 0) + 1 = 1$.



The height of 4's children are 1 and 1.

Thus, the height of 4 is $\max(1, 1) + 1 = 2$.



Done.

Every rotation, two height updates are performed, each in $O(1)$ time.

Analysis:

- Height of an AVL tree is $O(\log n)$
- In the worst case, length of insertion path is $O(\log n)$
- Have to maintain height data and check/fix balance at each node on insertion path
 - This is $O(1)$ per node
- Therefore, worst-case time complexity of AVL tree insertion is $O(\log n)$

AVL Trees

Insertion

Search

Deletion

Summary

Exactly the same as for regular BSTs.

Worst-case time complexity is $O(\log n)$,
since AVL trees are height-balanced.

AVL Trees

Insertion

Search

Deletion

Pseudocode

Rebalancing

Height data

Analysis

Summary

Method:

- Delete item recursively
- Check balance at each node along the deletion path* in reverse
- Fix imbalances as they are found

AVL Trees

Insertion

Search

Deletion

Pseudocode

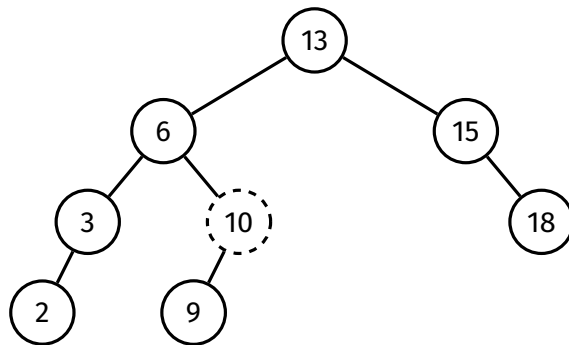
Rebalancing

Height data

Analysis

Summary

Example: Delete 10 from this tree



AVL Trees

Insertion

Search

Deletion

Pseudocode

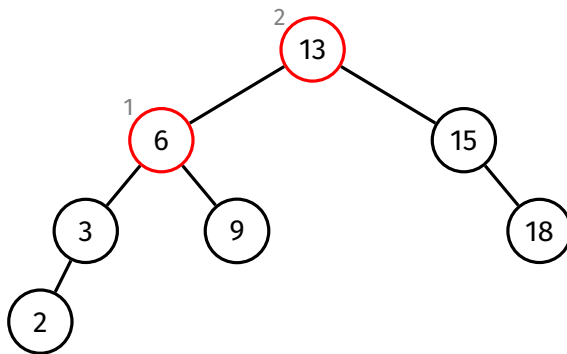
Rebalancing

Height data

Analysis

Summary

Example: Delete 10 from this tree



Balance must be checked at 6, then at 13

AVL Trees

Insertion

Search

Deletion

Pseudocode

Rebalancing

Height data

Analysis

Summary

Important:

If the item being deleted has two child nodes,
the deletion path includes the path to its successor
(the smallest value in its right subtree)

AVL Trees

Insertion

Search

Deletion

Pseudocode

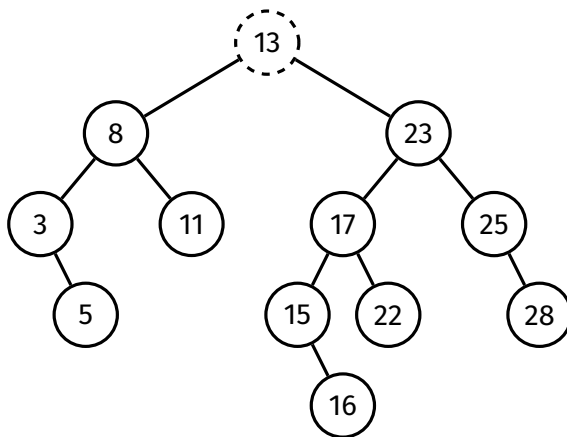
Rebalancing

Height data

Analysis

Summary

Example: Delete 13 from this tree



13 will be replaced by 15 (its in-order successor)

AVL Trees

Insertion

Search

Deletion

Pseudocode

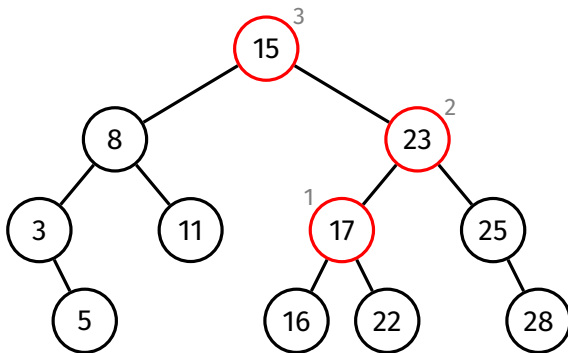
Rebalancing

Height data

Analysis

Summary

Example: Delete 13 from this tree



Balance must be checked at 17, then at 23, then at 15

```
avlDelete(t, v):  
    Input: AVL tree t, item v  
    Output: t with v deleted  
  
    if t is empty:  
        return empty tree  
    else if v < t->item:  
        t->left = avlDelete(t->left, v)  
    else if v > t->item:  
        t->right = avlDelete(t->right, v)  
    else:  
        if t->left is empty:  
            temp = t->right  
            free(t)  
            return temp  
        else if t->right is empty:  
            temp = t->left  
            free(t)  
            return temp  
        else:  
            successor = minimum value in t->right  
            t->item = successor  
            t->right = avlDelete(t->right, successor)  
  
    return avlRebalance(t)
```

Note: This is the same as in AVL tree insertion

`avlRebalance(t):`

Input: possibly unbalanced tree *t*

Output: balanced *t*

`bal = balance(t)`

if `bal > 1:`

if `balance(t->left) < 0:`

`t->left = rotateLeft(t->left)`

`t = rotateRight(t)`

else if `bal < -1:`

if `balance(t->right) > 0:`

`t->right = rotateRight(t->right)`

`t = rotateLeft(t)`

return *t*

`balance(t):`

Input: tree *t*

Output: balance factor of *t*

return `height(t->left) - height(t->right)`

AVL Trees

Insertion

Search

Deletion

Pseudocode

Rebalancing

Examples

Height data

Analysis

Summary

AVL tree deletion
has the same rebalancing cases
as AVL tree insertion.

AVL Trees

Insertion

Search

Deletion

Pseudocode

Rebalancing

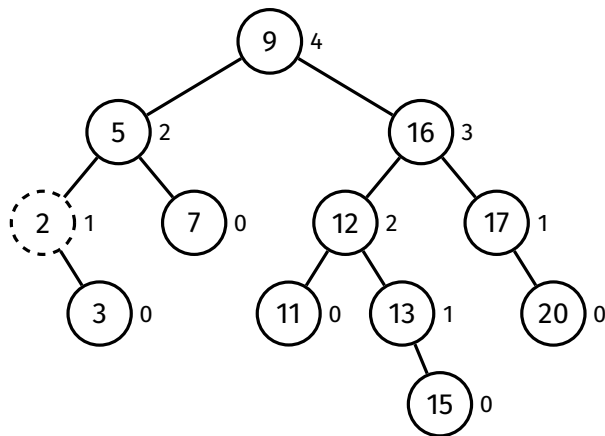
Examples

Height data

Analysis

Summary

Delete 2 from this tree:



AVL Trees

Insertion

Search

Deletion

Pseudocode

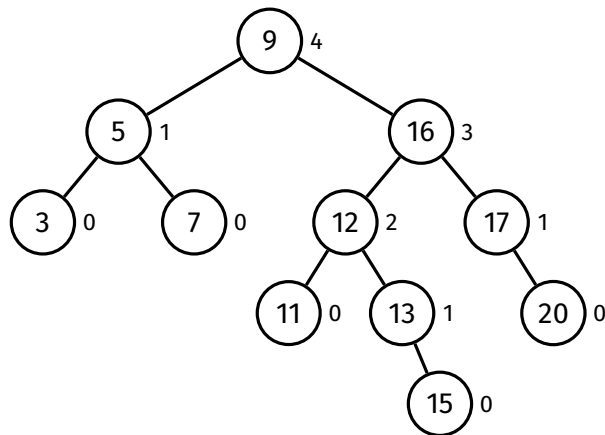
Rebalancing

Examples

Height data

Analysis

Summary



Check for balance at 5 and 9

AVL Trees

Insertion

Search

Deletion

Pseudocode

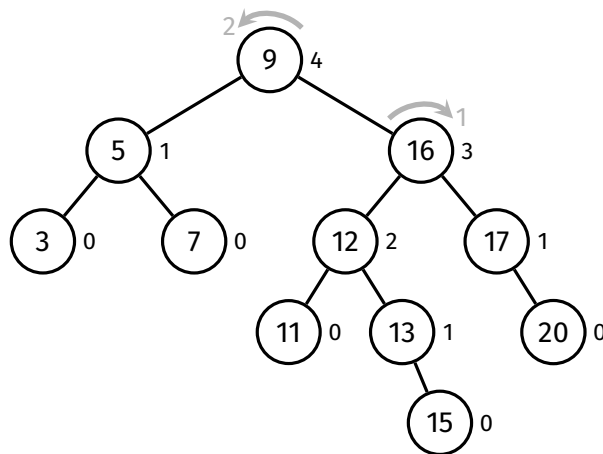
Rebalancing

Examples

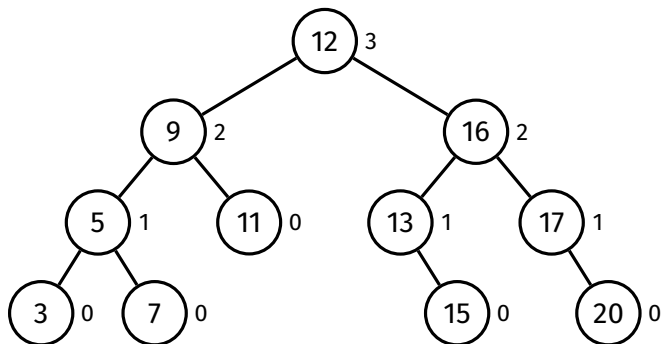
Height data

Analysis

Summary

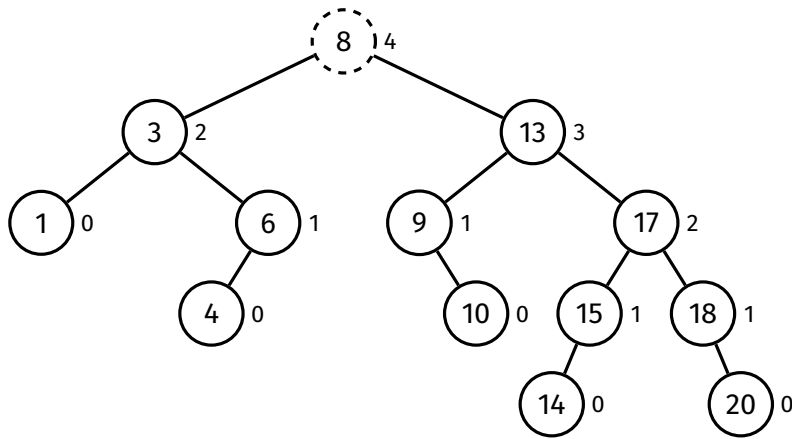


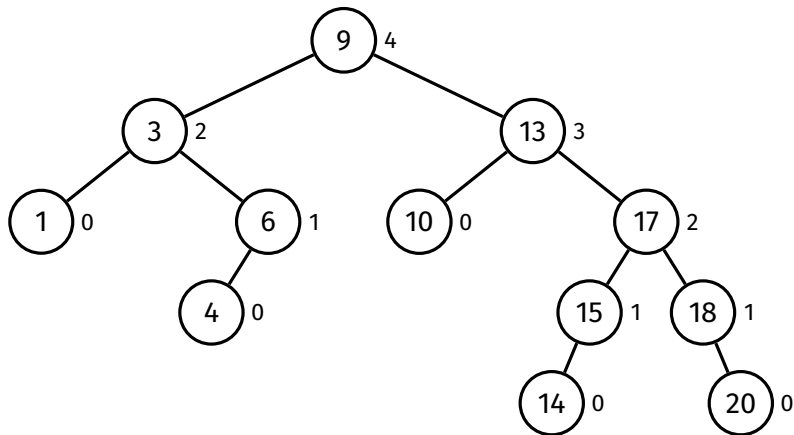
9 is unbalanced



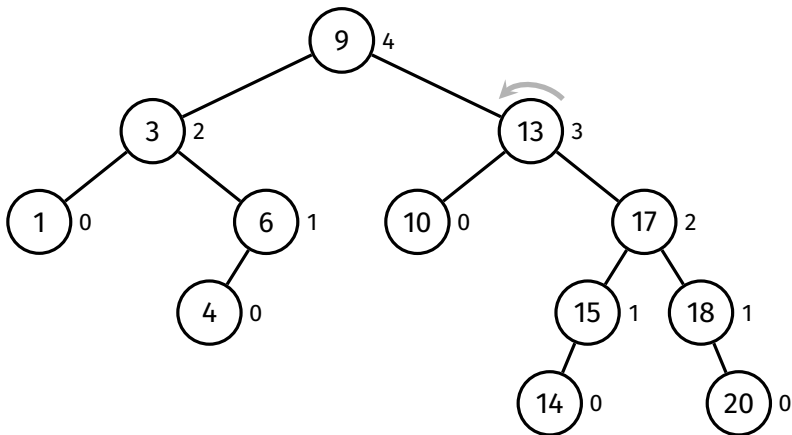
Balanced

Delete 8 from this tree:

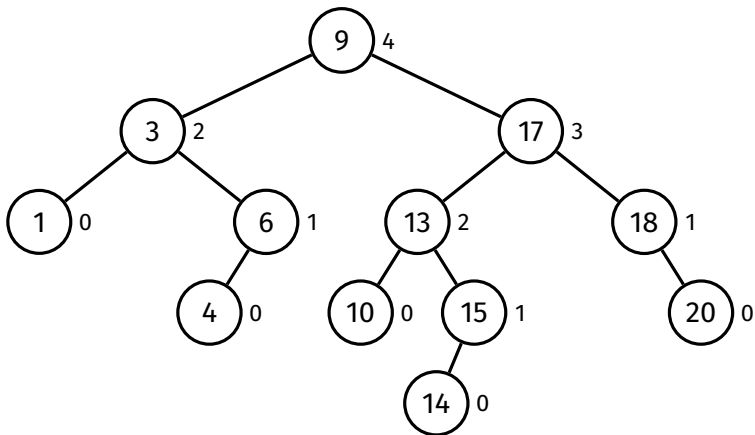




Check for balance at 13 and 9



13 is unbalanced



Balanced

AVL Trees

Insertion

Search

Deletion

Pseudocode

Rebalancing

Height data

Maintenance

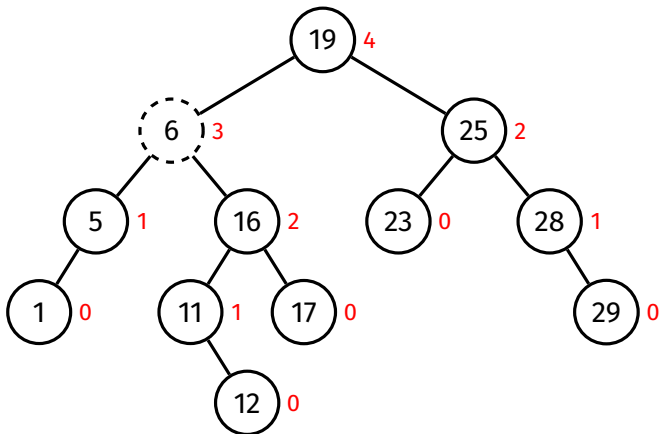
Analysis

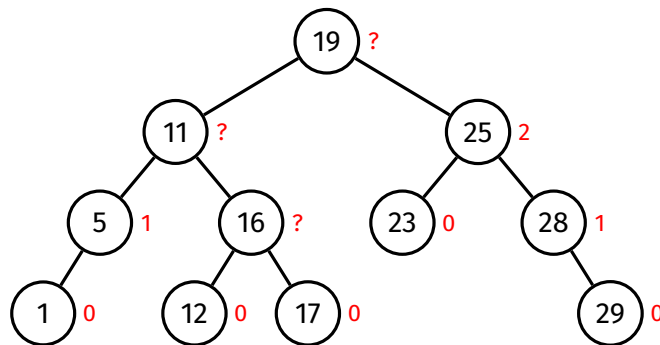
Summary

Height data also needs to be maintained...

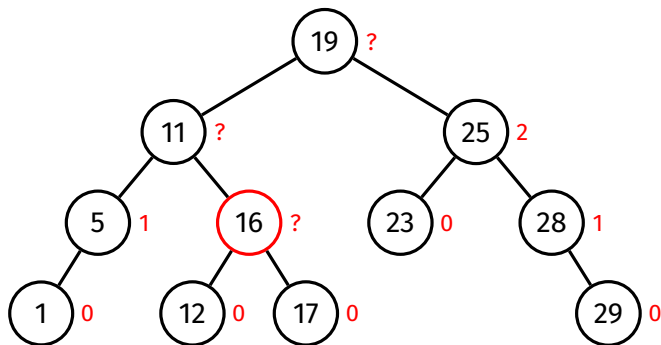
- Whenever a node is deleted
 - Heights of all nodes on deletion path may be affected

Example: Delete 6 from this tree



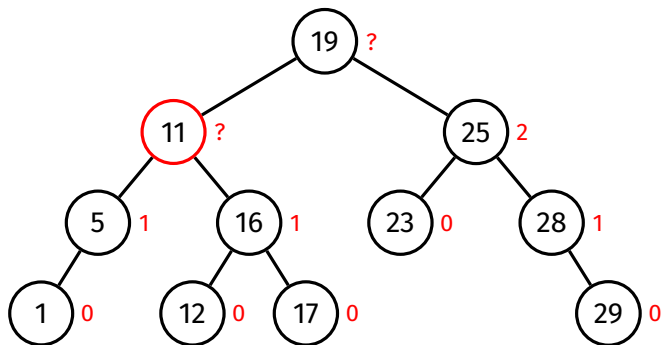


Recompute height of each node on the deletion path
using the heights stored in its children.



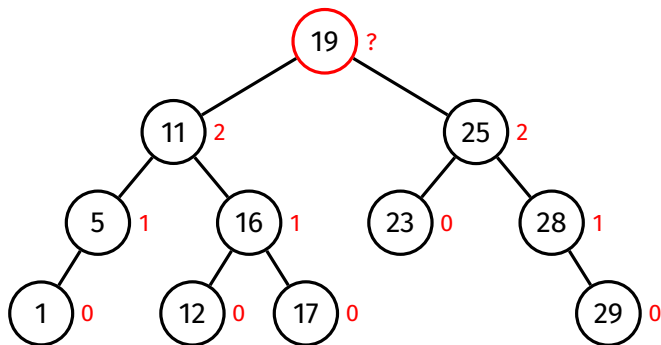
The heights of 16's children are 0 and 0.

Thus, the height of 16 is $\max(0, 0) + 1 = 1$.



The heights of 11's children are 1 and 1.

Thus, the height of 11 is $\max(1, 1) + 1 = 2$.



The heights of 19's children are 2 and 2.

Thus, the height of 19 is $\max(2, 2) + 1 = 3$.

AVL Trees

Insertion

Search

Deletion

Pseudocode

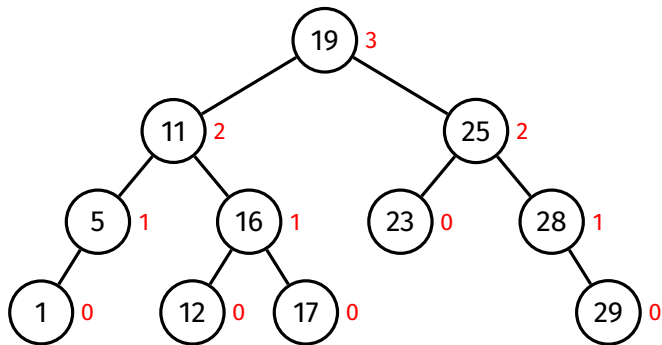
Rebalancing

Height data

Maintenance

Analysis

Summary



Done.

AVL Trees

Insertion

Search

Deletion

Pseudocode

Rebalancing

Height data

Analysis

Summary

Analysis:

- Height of an AVL tree is $O(\log n)$
- In the worst case, length of deletion path is $O(\log n)$
- Have to maintain height data and check/fix balance at each node on deletion path
 - This is $O(1)$ per node
- Therefore, worst-case time complexity of AVL tree deletion is $O(\log n)$

- AVL trees are always height-balanced
 - This means the height of an AVL tree is $O(\log n)$
- Rotations are used to fix imbalances during insertion and deletion
- Balance is checked efficiently by storing height data in each node, which needs to be maintained
- Worst-case time complexity of $O(\log n)$ for insertion, search and deletion

AVL Trees

Insertion

Search

Deletion

Summary

<https://forms.office.com/r/zEqxUXvmLR>

