Lecture 4: Binary Search Tree (BST)

Yerlan Kuzbakov

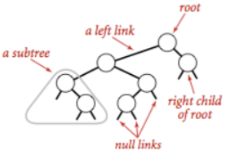
BST Interactive Visualization

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

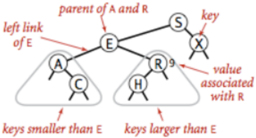
Binary Search Tree

The keys in a binary search tree are always stored in such a way as to satisfy the ***binary-search-tree property***:

Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $y.key \leq x.key$. If y is a node in the right subtree of x, then $y.key \geq x.key$.



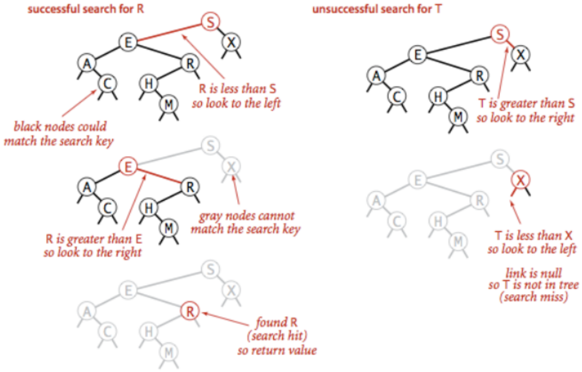
Anatomy of a binary tree



Anatomy of a binary search tree

Search in a Binary Search Tree

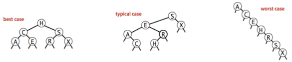
- Start at the root.
- If the key equals the root, search is complete.
- If the key is smaller, go to the left subtree.
- If the key is larger, go to the right subtree.
- Repeat until key is found or subtree is empty.



Worst Case in BST

- In the worst case, the BST becomes skewed (like a linked list).
- This happens if keys are inserted in sorted order without balancing.
- Time complexity of search/insert/delete becomes $O(n)$.

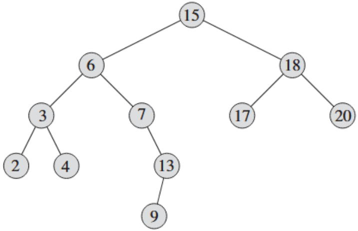
Analysis. The running times of algorithms on binary search trees depend on the shapes of the trees, which, in turn, depends on the order in which keys are inserted.



BST Inorder Traversal

```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print x.key
    INORDER-TREE-WALK(x.right)
```

Searching in BST




```
TREE-SEARCH(x, k)
if x == NIL or k == x.key
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)
```

BST: Minimum and Maximum

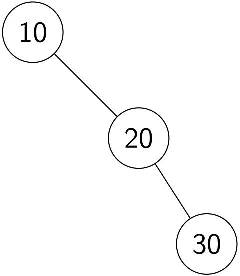
```
TREE-MINIMUM(x)
while x.left ≠ NIL
    x = x.left
return x
```

```
TREE-MAXIMUM(x)
while x.right ≠ NIL
    x = x.right
return x
```

- More on Binary Search Trees:

https://www.w3schools.com/dsa/dsa_data_binarysearchtrees.php

BST: Imbalanced (10 $\rightarrow$ 20 $\rightarrow$ 30)

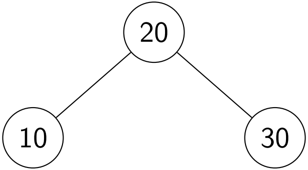


Definition. A (search) tree is *balanced* if its height h grows logarithmically with the number of nodes n so search/insert/delete are $O(\log n)$.

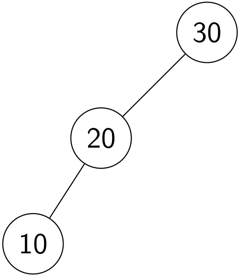
Examples

- **AVL**: $|\mathrm{bf}(v)| \le 1$; rotations restore balance (height $O(\log n)$).
- **Red–Black**: coloring rules, equal black-height; $h \le 2\log_2(n+1)$.

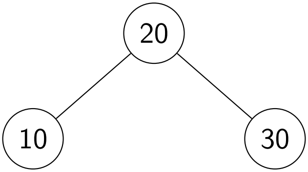
BST: Balanced (20 as root)



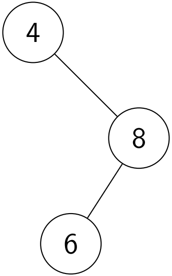
BST: Imbalanced (30 $\rightarrow$ 20 $\rightarrow$ 10)



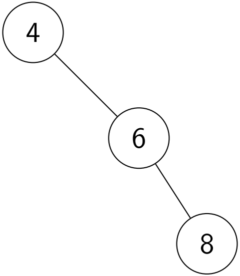
BST: Balanced (20 as root)



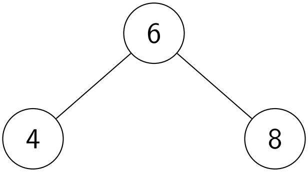
BST: Zig-Zag (4, 8, 6)



BST: Imbalanced (4, 6, 8)



BST: Balanced (6 as root)



BST Rotations — Video (Rob Edwards)

https://www.youtube.com/watch?v=NczBLeco6XA&ab_channel=RobEdwards

Further Reading

- Delete in Binary Search Trees: Introduction to Algorithms. pp 295-298

Further Reading

- More on Binary Search Trees:

  https://algs4.cs.princeton.edu/32bst/