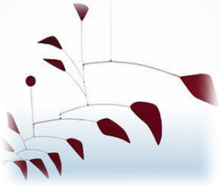


EECS 4101-5101
Advanced Data Structures

Shahin Kamali

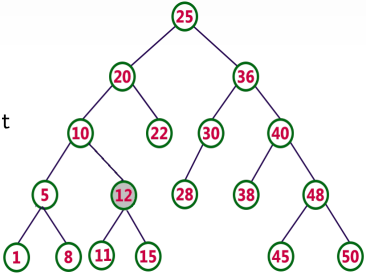
Topic 2d - Splay Trees & Dynamic Optimality Conjecture

York University

Picture is from the cover of the textbook CLRS.

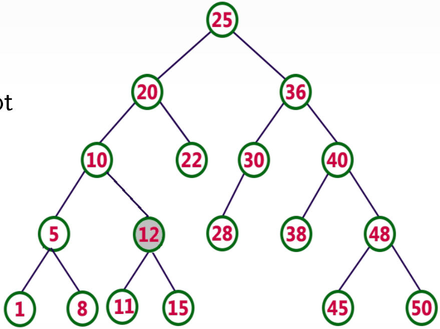
Self-Adjusting Binary Search Trees

- The input is a set of *requests* to items in a BST of size N.
    - The goal is to update the tree to adjust it into patterns in the input.
- There is a lot of **locality** in the input sequence.
- Can we apply Move-To-Front for trees?



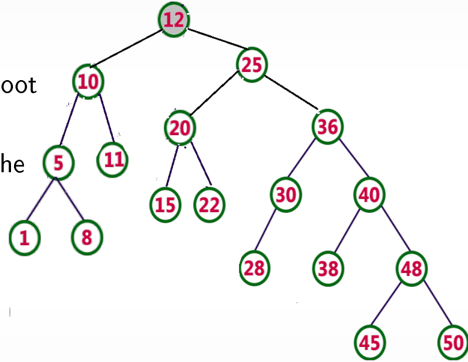
Splay Trees Idea

- When there is a request to item *a*, adjust the tree so that *a* becomes root in the new tree!
- Use tree **rotations** to 'bubble up' the accessed item.
- We say that we **splay** *a* to become root in the adjusted tree
    - It is a natural extension of Move-To-Front to the lists.



Splay Trees Idea

- When there is a request to item *a*, adjust the tree so that *a* becomes root in the new tree!
- Use tree **rotations** to 'bubble up' the accessed item.
- We say that we **splay** *a* to become root in the adjusted tree
    - It is a natural extension of Move-To-Front to the lists.



Splaying Rotations General Idea

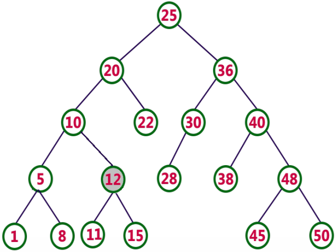
- Consider accessed item a, its parent p and grand-parent g (if they exist).
- Reorder a, p, and g so that a appears 'above' the other two
  - If a is smallest/largest, p and g will be in one side of a.
  - If a is in between, p and g will be on its left and right.

Splaying Rotations General Idea

- Consider accessed item a, its parent p and grand-parent g (if they exist).
- Reorder a, p, and g so that a appears 'above' the other two
  - If a is smallest/largest, p and g will be in one side of a.
  - If a is in between, p and g will be on its left and right.
- After re-ordering a, p, and g, 'place' the following four subtrees in their appropriate position to save BST property:
  - the two subtrees of a
  - the sibling of a in the subtree of p
  - the sibling of p in the subtree of g

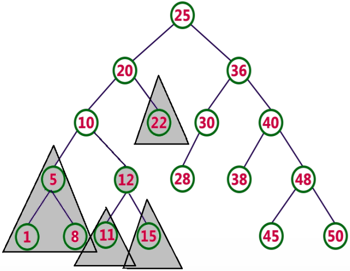
Splay Example

- E.g., Access $a = 12$



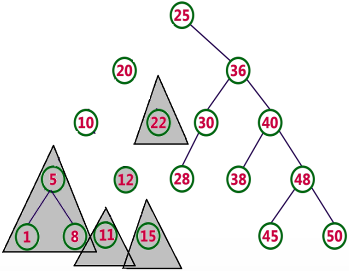
Splay Example

- E.g., Access $a = 12$



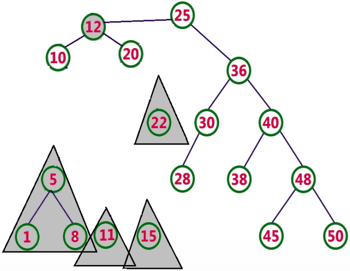
Splay Example

- E.g., Access $a = 12$



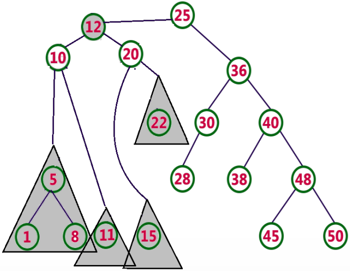
Splay Example

- E.g., Access $a = 12$



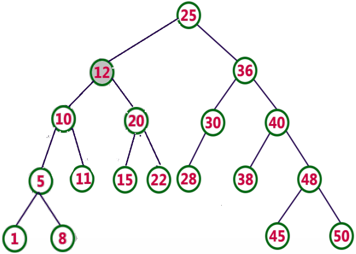
Splay Example

- E.g., Access $a = 12$



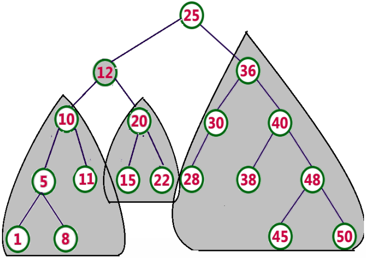
Splay Example

- E.g., Access $a = 12$



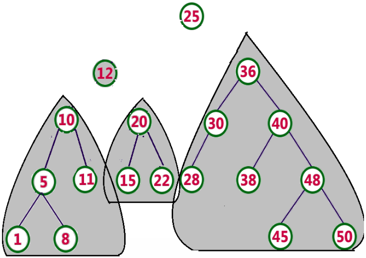
Splay Example

- E.g., Access $a = 12$



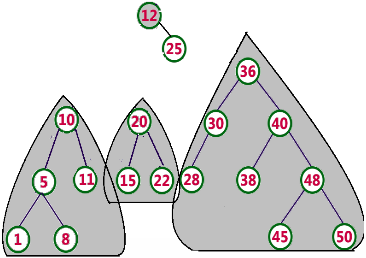
Splay Example

- E.g., Access $a = 12$



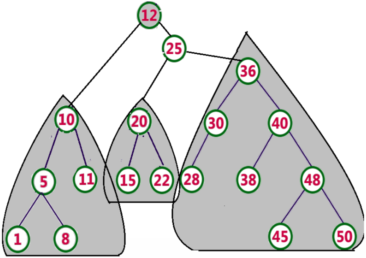
Splay Example

- E.g., Access $a = 12$



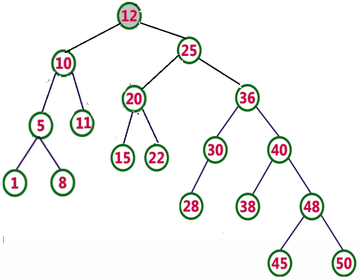
Splay Example

- E.g., Access $a = 12$



Splay Example

- E.g., Access $a = 12$

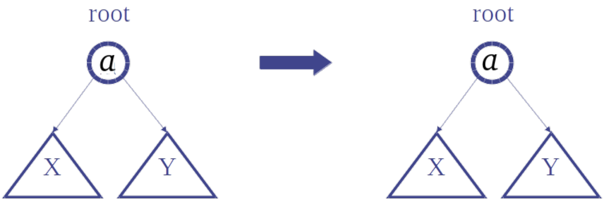


Splaying Cases (a bit more formal)

- The accessed node a is either
  - Root
  - Child of the root
  - Has both parent (p) and grandparent (g):
    - Zig-zig pattern: $g \rightarrow p \rightarrow a$ is left-left or right-right
    - Zig-zag pattern: $g \rightarrow p \rightarrow a$ is left-right or right-left

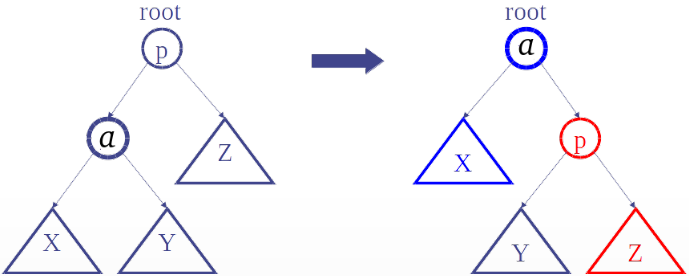
Access root

- if x is root, do nothing!



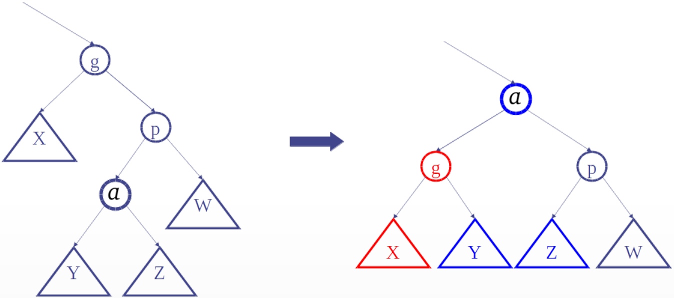
Access child of root

- When x is child of the root, do a single rotation to move it above its parent
  - It is called a **zig** operation



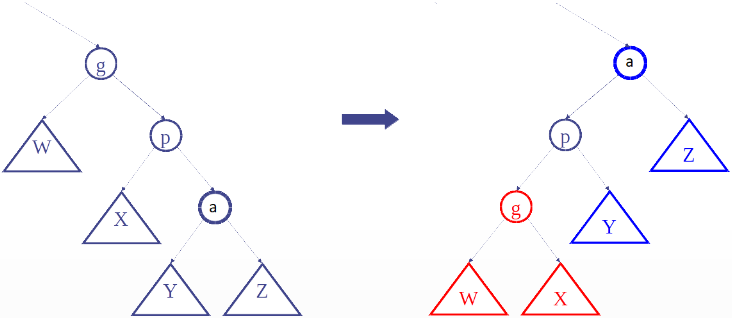
Access LR or RL grandchild

- When x is left-child (resp. right-child) of P and p is right-child (resp. left-child) of g, do a double rotation.
  - It is called a **zig-zag** operation



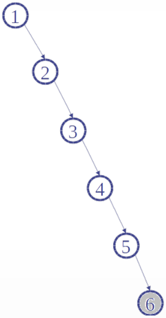
Access LL or RR grandchild

- Reverse the order of *a*,*p*,and *g*.
  - It is called a **zig-zig** operation



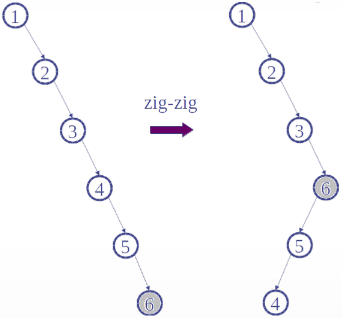
Splay Example

- E.g., Access $a = 6$



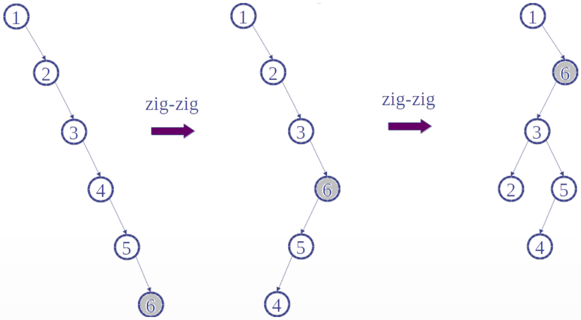
Splay Example

- E.g., Access $a = 6$



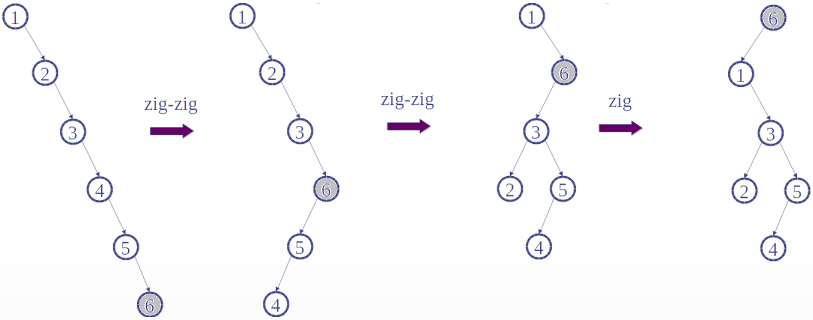
Splay Example

- E.g., Access $a = 6$



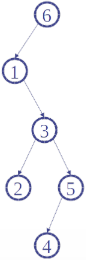
Splay Example

- E.g., Access $a = 6$



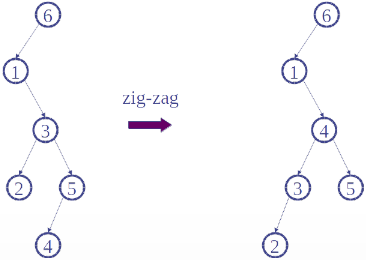
Splay Example

- E.g., Access $a = 4$



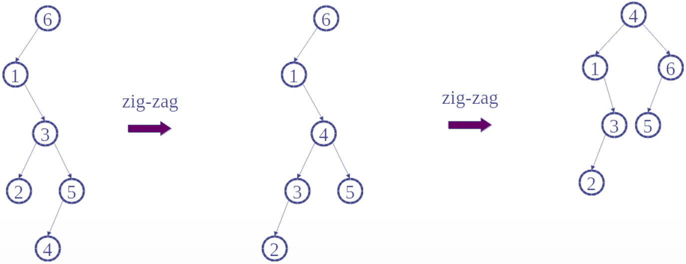
Splay Example

- E.g., Access $a = 4$



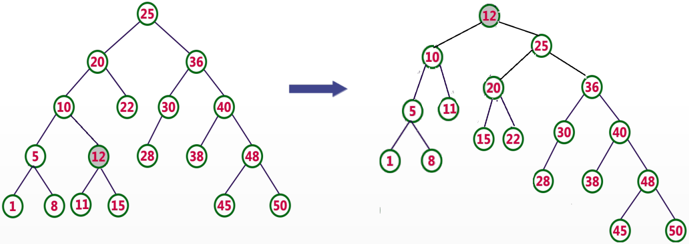
Splay Example

- E.g., Access $a = 4$



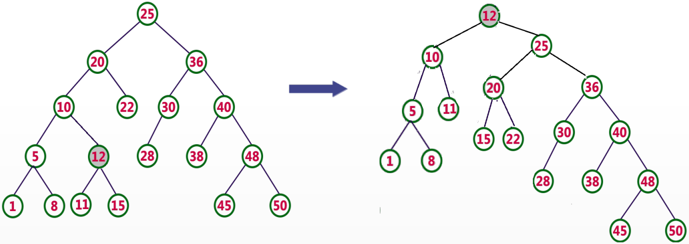
Splaying: Intuition

- The accessed node is moved to 'front' (i.e., is now root)
- Let b be a node on the access path from root to the accessed node a. If b is at depth d before the splay, it's at about depth $d/2$ after the splay.
    - 'Deeper nodes' on the access path tend to move closer to the root



Splaying: Intuition

- The accessed node is moved to 'front' (i.e., is now root)
- Let b be a node on the access path from root to the accessed node a. If b is at depth d before the splay, it's at about depth $d/2$ after the splay.
  - 'Deeper nodes' on the access path tend to move closer to the root
- Splaying gets amortized $O(\log N)$ amortized time.
  - N is the number of nods in the tree



BST-Update problem

- So far, we learned how Splay trees work; they are equivalent of self-adjusting lists updated with MTF.

BST-Update problem

- So far, we learned how Splay trees work; they are equivalent of self-adjusting lists updated with MTF.
- BST-Update problem:
  - The input is an online sequence of requests to items in a BST.
  - Each probe for finding an item x has cost 1.
  - On the path traversed from the root to x, the algorithm can make any number of rotations at a cost of 1 per rotation.

BST-Update problem

- So far, we learned how Splay trees work; they are equivalent of self-adjusting lists updated with MTF.
- BST-Update problem:
  - The input is an online sequence of requests to items in a BST.
  - Each probe for finding an item x has cost 1.
  - On the path traversed from the root to x, the algorithm can make any number of rotations at a cost of 1 per rotation.
- As before, the competitive ratio of an algorithm (self-adjusting tree) is defined as the maximum ratio between the cost of that tree and an optimal offline tree (which adjust itself via rotations.)

BST-Update problem

- So far, we learned how Splay trees work; they are equivalent of self-adjusting lists updated with MTF.
- BST-Update problem:
  - The input is an online sequence of requests to items in a BST.
  - Each probe for finding an item x has cost 1.
  - On the path traversed from the root to x, the algorithm can make any number of rotations at a cost of 1 per rotation.
- As before, the competitive ratio of an algorithm (self-adjusting tree) is defined as the maximum ratio between the cost of that tree and an optimal offline tree (which adjust itself via rotations.)
- E.g., AVL trees, red-black trees have a competitive ratio of $\Omega(\log n)$ (why?)

BST-Update problem

- **Dynamic Optimality Conjecture**: Splay trees have a competitive ratio independent of the size N of tree and length n of sequence.
- As before, the competitive ratio is defined as the maximum ratio between the cost of an algorithm and that of an optimal offline algorithm (which can update the tree using rotations)
    - We know the competitive ratio of splay trees is $O(\log N)$

BST-Update problem

- **Dynamic Optimality Conjecture**: Splay trees have a competitive ratio independent of the size N of tree and length n of sequence.
- As before, the competitive ratio is defined as the maximum ratio between the cost of an algorithm and that of an optimal offline algorithm (which can update the tree using rotations)
  - We know the competitive ratio of splay trees is $O(\log N)$
  - The best existing algorithm is provided by self-adjusting **Tango Trees**, and has a competitive ratio of $O(\log \log N)$.