Splay trees

April 15, 2015 1

Binary search trees

- A *binary search tree* is a binary tree storing entries (i.e., key-value pairs) at its internal nodes satisfying:

  If u, v, and w are nodes such that u is in the left subtree of v and w is in the right subtree of v, then

  $$key(u) < key(v) < key(w).$$

- External nodes do not store entries.

April 15, 2015 2

Performance

- Consider a map of n entries implemented by means of a binary search tree of height h.
- Search, insertion and removal all take time proportional to the height h.
- The height h is $O(n)$ in the worst case and $O(\log n)$ in the best case.

April 15, 2015 3

Performance

- Consider a map of n entries implemented by means of a binary search tree of height h.
- Search, insertion and removal all take time proportional to the height h.
- The height h is $O(n)$ in the worst case and $O(\log n)$ in the best case.
- *This can be improved by keeping the binary tree "balanced".*

Splay trees

- A *splay* tree is a self-balancing binary search tree in which *splaying* is performed after every search, insertion and removal.

Splay trees

- A *splay* tree is a self-balancing binary search tree in which *splaying* is performed after every search, insertion and removal.
- While splay trees are often "balanced", there is no explicit condition to maintain (unlike AVL trees).

Splay trees

- A *splay* tree is a self-balancing binary search tree in which *splaying* is performed after every search, insertion and removal.
- While splay trees are often “balanced”, there is no explicit bound on the height of the tree (unlike AVL trees).
- With a splay tree of n entries, search, insertion and removal all take $O(\log n)$-time *on average*.

Splaying

- *Splaying* is a *move-to-root* operation that causes frequently accessed elements to remain nearer the root.

Splaying

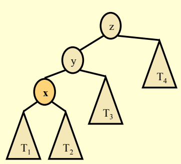
- *Splaying* is a *move-to-root* operation that causes frequently accessed elements to remain nearer the root.
- *Splaying an internal node x* means raising x up to the root through a sequence of rotations.

Splaying

- *Splaying* is a *move-to-root* operation that causes frequently accessed elements to remain nearer the root.
- *Splaying an internal node x* means raising x up to the root through a sequence of rotations.
- There are three types of rotation: *zig-zig*, *zig-zag* and *zig*.

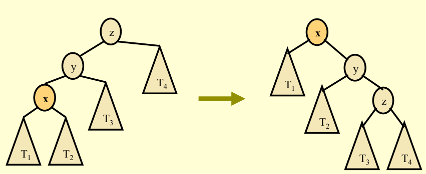
Zig-zig rotation

x is a left (resp. right) child of y, and y is a left (resp. right) child of z.



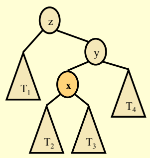
Zig-zig rotation

x is a left (resp. right) child of y, and y is a left (resp. right) child of z.



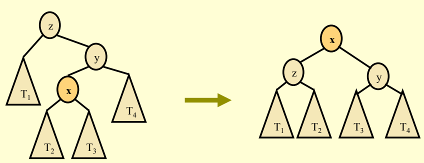
Zig-zag rotation

x is a left (resp. right) child of *y*, and *y* is a right (resp. left) child of *z*.



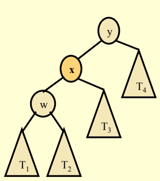
Zig-zag rotation

x is a left (resp. right) child of *y*, and *y* is a right (resp. left) child of *z*.



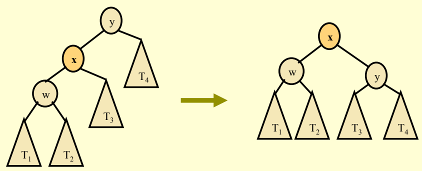
Zig rotation

x is a left (resp. right) child of *y*, and *y* has no parent.



Zig rotation

x is a left (resp. right) child of y, and y has no parent.



When to splay

Splaying is performed after every search, insertion and removal operation.

When to splay

Splaying is performed after every search, insertion and removal operation.

- get(k): If k is found at a node x, then splay x; otherwise, splay the parent of the leaf reached.

When to splay

Splaying is performed after every search, insertion and removal operation.

- get(k): If k is found at a node x, then splay x; otherwise, splay the parent of the leaf reached.
- put(k, v): Splay the new node with (k, v).

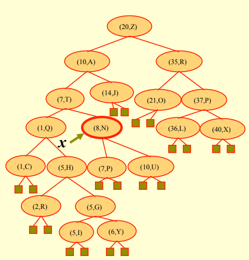
When to splay

Splaying is performed after every search, insertion and removal operation.

- get(k): If k is found at a node x, then splay x; otherwise, splay the parent of the leaf reached.
- put(k, v): Splay the new node with (k, v).
- remove(k): Splay the parent of the node removed (not necessarily the node that had the removed entry).

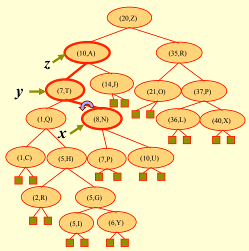
Splaying

Example. Splaying $x = (8, \mathrm{N})$.



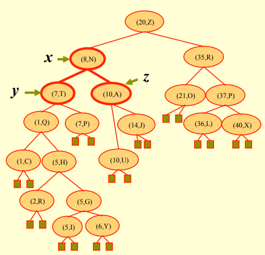
Zig-zag rotation

Example. Splaying $x = (8, \mathrm{N})$.



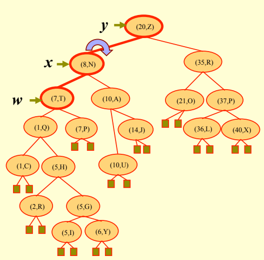
Zig-zag rotation

Example. Splaying $x = (8, \mathrm{N})$.



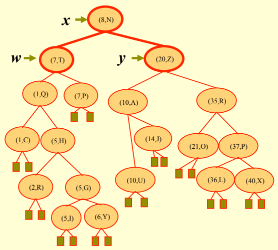
Zig rotation

Example. Splaying $x = (8, \mathrm{N})$.



Zig rotation

Example. Splaying $x = (8, \mathrm{N})$.



Splaying Visualization

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

Performance

- Performance analysis is very complex.

Performance

- Performance analysis is very complex.
- Starting from the empty map, consider performing a sequence of m search/ insertion/removal operations on n entries. The total running time of these m operations is $O(m \log n)$.

Performance

- Performance analysis is very complex.
- Starting from the empty map, consider performing a sequence of m search/ insertion/removal operations on n entries. The total running time of these m operations is $O(m \log n)$.
- Thus, *on average*, each operation takes $O(\log n)$ time.