Binary Search Trees: Splay Trees: Operations

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Data Structures Fundamentals
Algorithms and Data Structures

Learning Objectives

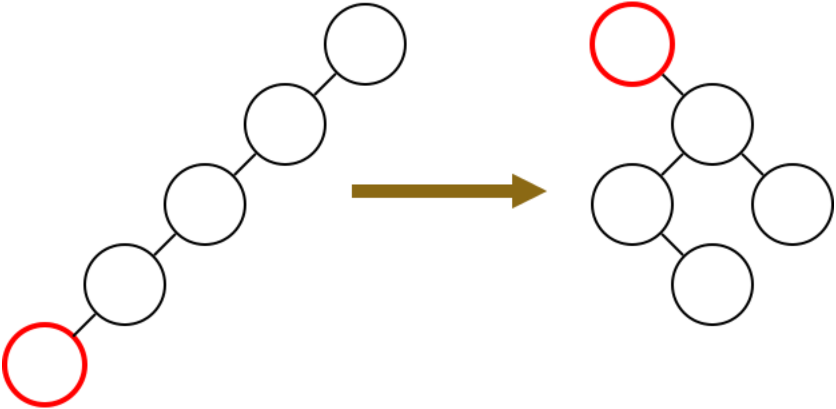
- Implement a splay tree.

Last Time

- Idea: bring each query node to the root.
- Introduced splay operation to bring node to root.

Sometimes Slow

Splay operation is sometimes slow:



Amortized Analysis

Need to amortize.

Theorem

The amortized cost of doing $O(D)$ work and then splaying a node of depth D is $O(\log(n))$ where n is the total number of nodes.

Amortized Analysis

Need to amortize.

Theorem

The amortized cost of doing $O(D)$ work and then splaying a node of depth D is $O(\log(n))$ where n is the total number of nodes.

We will prove this later, but using it we can implement all our operations.

Find

$\texttt{STFind}(k, R)$

$N \leftarrow \texttt{Find}(k, R)$
$\texttt{Splay}(N)$
$\texttt{return } N$

Runtime

Suppose node at depth D.

- $O(D)$ time to find N.
- Splay N.
- Amortized cost $O(\log(n))$.

Runtime

Suppose node at depth D.

- $O(D)$ time to find N.
- Splay N.
- Amortized cost $O(\log(n))$.

You pay for the work of finding N by splaying to rebalance the tree.

Important Point

Even if you fail to find k, you must still splay the closest node you found. Otherwise your operation did $O(D)$ work with nothing to amortize against.

Insert

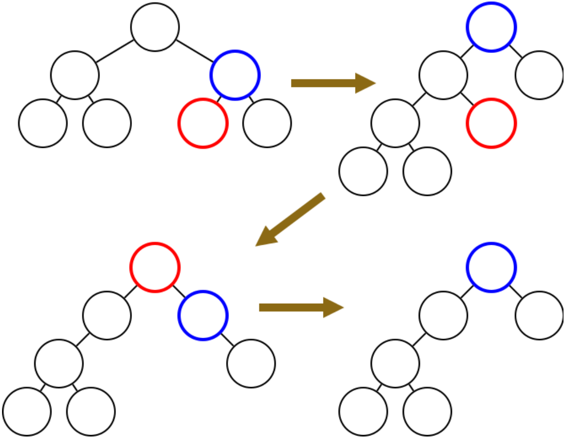
Insert, then splay

$\texttt{STInsert}(k, R)$

$\texttt{Insert}(k, R)$
$\texttt{STFind}(k, R)$

Delete

Bring N and successor to top. Deletes easily.



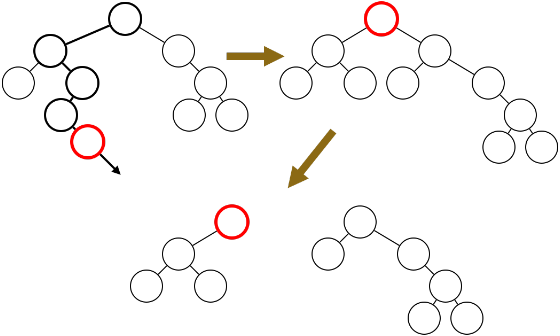
Delete

$\texttt{STDelete}(N)$

$\texttt{Splay}(\texttt{Next}(N))$
$\texttt{Splay}(N)$
$L \leftarrow N.\texttt{Left}$
$R \leftarrow N.\texttt{Right}$
$R.\texttt{Left} \leftarrow L$
$L.\texttt{Parent} \leftarrow R$
$\texttt{Root} \leftarrow R$
$R.\texttt{Parent} \leftarrow \texttt{null}$

Split

Splay and Cut.



Split

$\texttt{STSplit}(R, x)$

```
N ← Find(x, R)
Splay(N)
if N.Key > x:
  return CutLeft(R)
else if N.Key < x:
  return CutRight(R)
else
  return N.Left, N.Right
```

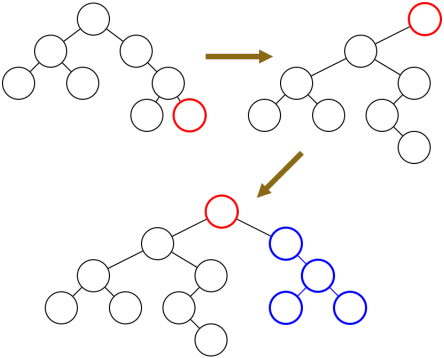
CutLeft

$\texttt{CutLeft}(N)$

$L \leftarrow N.\texttt{Left}$
$N.\texttt{Left} \leftarrow \texttt{null}$
$L.\texttt{Parent} \leftarrow \texttt{null}$
$\texttt{return}\ L, N.$

Merge

Splay largest element of first tree, and stick second tree as child of the root.



Note after splay, root has no right child.

Merge

$\texttt{STMerge}(R_1, R_2)$

$N \leftarrow \texttt{Find}(\infty, R_1)$
$\texttt{Splay}(N)$
$N.\texttt{Right} \leftarrow R_2$
$R_2.\texttt{Parent} \leftarrow N$

Summary

Splay trees perform all operations simply in $O(\log(n))$ amortized time.