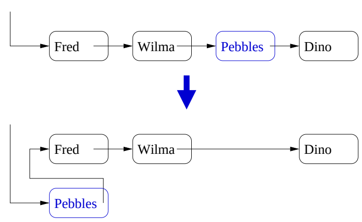
8: Splay Trees

CSE326 Spring 2002

April 16, 2002

Move-to-Front Heuristic

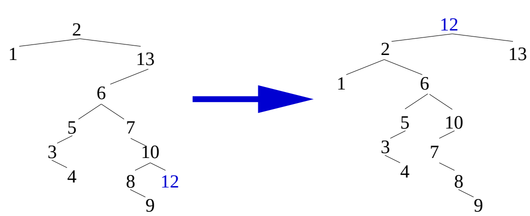
Search for Pebbles:



- Move found item to front of list
- Frequently searched items will move to start of list
  - Effective both theoretically and practically

Splaying

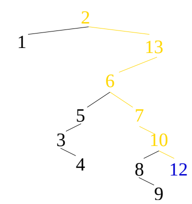
Splay(12)



Move-To-Front for Trees

Gettin' Down: Step 1

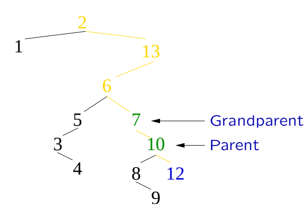
Splay(12)



Remember the path to the node

Know Who Begot You

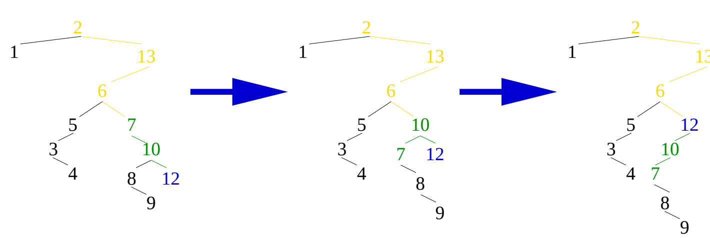
Splay(12)



Look at *Parent* and *Grandparent*

Splay Case 1

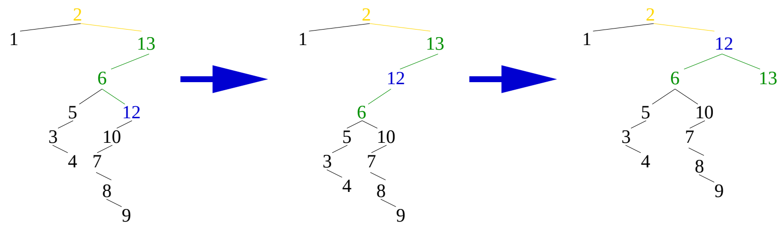
Splay(12)



Rotate Left 7, Rotate Left 10

Splay Case 2

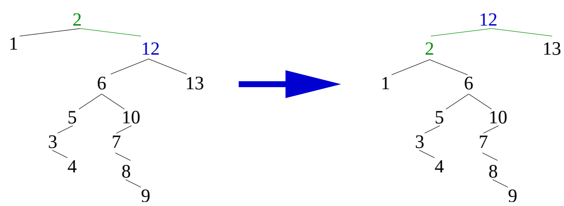
Splay(12)



Rotate Left 6, Rotate Left 13

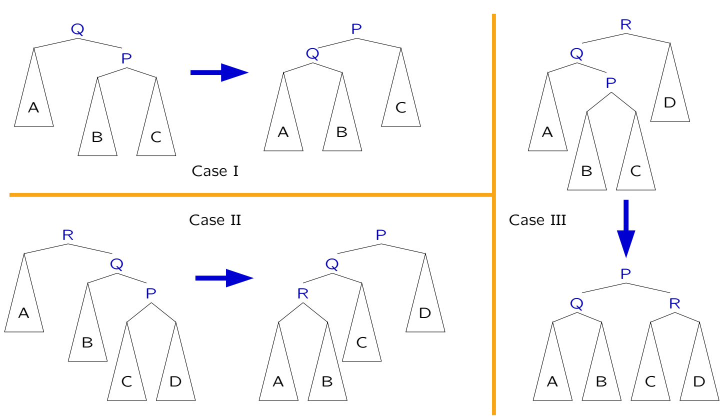
Splay Case 3

Splay(12)



Rotate Left 2

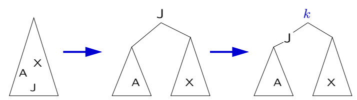
Splay Cases



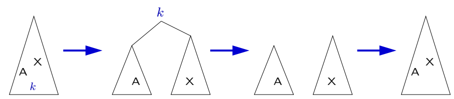
All Dressed Up

Splay(k) Splay k, or predecessor or successor to k, to root, depending if k is in the tree.

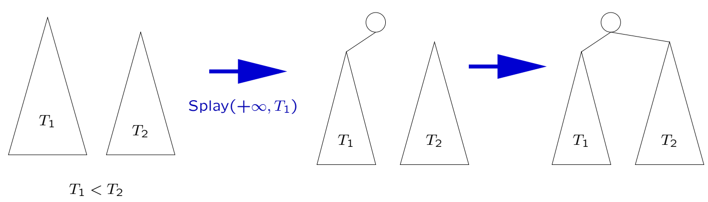
Insert(k) Splay(k), then update



Delete(k) Splay(k), if root is k, then remove it, and Concat(A, B)

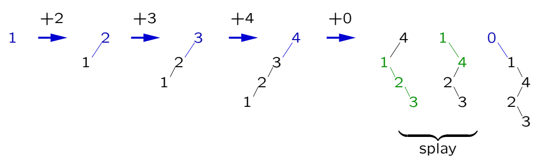


Concatenation

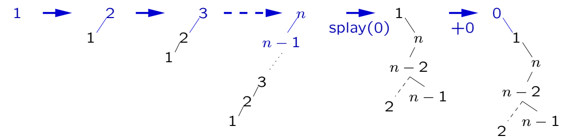


Concat(T_1, T_2): Splay($+\infty, T_1$), then join T_2 as right child of T_1.

Example



Generalize the Example

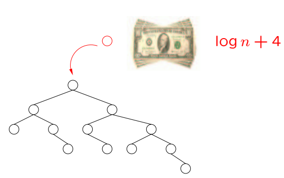


Amortized Analysis

- Splaying is the expensive operation
- Sometimes we do *more* than $O(\log n)$ work per node...
- Sometimes we do *less* than $O(\log n)$ work per node...
- But it balances out: m operations in a tree with at most n nodes takes $O(m \log n)$ time!
- Easy to say, harder to prove

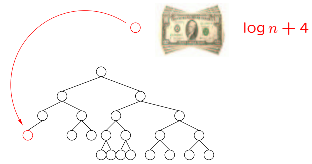
Worst-Case Analysis

Time = Money



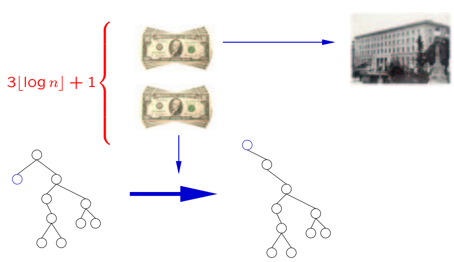
We proved we needed to spend at most $\log n + 4$ time per AVL insertion

Worst-Case Analysis



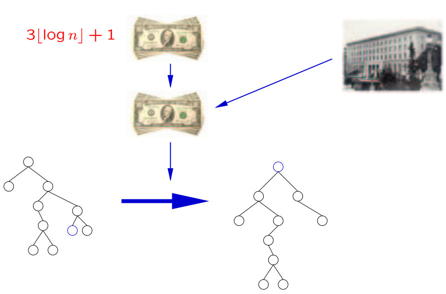
If the insertion was easy, our analysis loses

Amortized Analysis



If the splay was easy, *bank* the left-over money

Amortized Analysis

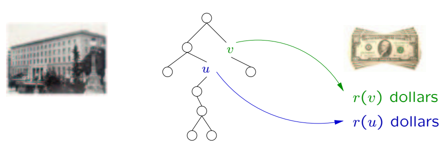


If the splay was hard, *use* money from the bank

Amortized Analysis

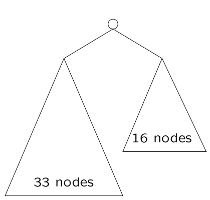
- Always *invest* $3\lfloor \log n \rfloor + 1$ per splay
- Prove there's *always* enough money in the bank for any operation
- Then $O(m \log n)$ time to do m operations

Store Money in the Tree



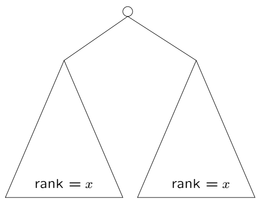
$r(v) = \lfloor \log \text{size of subtree at } v \rfloor$

Ranks are Logarithms



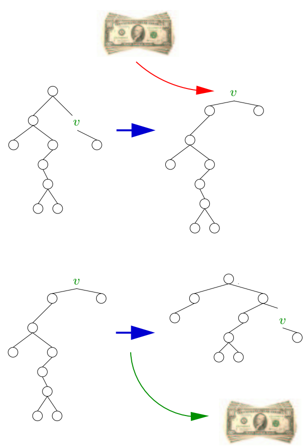
Rank of parent at least that of any child, but sometimes not greater.

Ranks are Logarithms



If both children have same rank, than rank of parent is larger

The Money Invariant

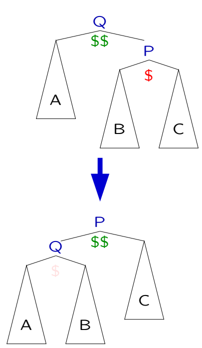


- Each node v has $r(v)$ dollars
- If v moves *up*, *add* more money to v

  $r'(v) > r(v)$
- If v moves *down*, *take* money from v

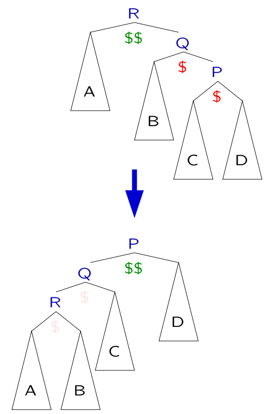
  $r'(v) < r(v)$

The Cost of Splaying: I



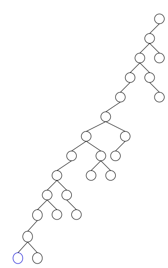
- Always the last step
- Only ranks of P and Q change
- $r'(P) = r(Q)$
- Get $r(P)$ dollars
- Need $r'(Q) \leq r'(P)$ dollars
- Need $1 to do the rotation
- Total: $\leq r'(P) - r(P) + 1$

The Cost of Splaying: II



- Need $r'(Q) + r'(R) - (r(P) + r(Q))$
  $\leq 2(r'(P) - r(P))$
- If $r'(P) > r(P)$, then $3(r'(P) - r(P))$ is enough to pay for the rotation, too
- Otherwise, $r'(P) = r(P)$, so do we need \$1 to pay for the rotation?

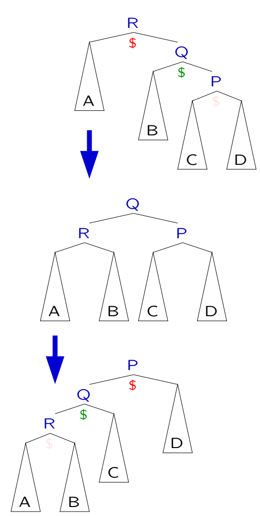
The Cost of Splaying



- If we pay \$1 for each case II, could pay $\Theta(n)$, and we need $O(\log n)$
- If cost only depends on *rank difference*, we'll be okay:

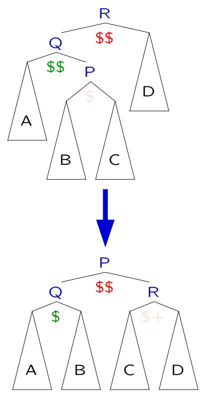
$$
\begin{aligned}
& 3(r^{(1)}(P) - r(P)) \\
+ \; & 3(r^{(2)}(P) - r^{(1)}(P)) \\
+ \; & 3(r^{(3)}(P) - r^{(2)}(P)) \\
& \vdots \\
+ \; & 3(r^{(k)}(P) - r^{(k-1)}(P)) + 1 \\
= \; & 3(r^{(k)}(P) - r(P)) + 1 \\
\leq \; & 3\lfloor \log n \rfloor + 1
\end{aligned}
$$

The Cost of Splaying: II



- If $r'(P) = r(P)$, then
  - ⋆ $r'(R) < r(P)$
    Otherwise $r'(P) > r(P)$
  - ⋆ $r'(Q) \leq r'(P) = r(P) \leq r(Q)$
  - ⋆ R's \$ ⇒ P
  - ⋆ P's \$ ⇒ R, with extra to pay for rotation

The Cost of Splaying: III



- R's \$ $\Rightarrow$ new P
- Q's \$ stays put

  (may waste some)
- P's \$ $\Rightarrow$ new R, and pay $r'(P) - r(P)$ extra \$s
- If $r'(P) > r(P)$, we're within $3(r'(P) - r(P))$ after paying for rotation
- If $r'(P) = r(P)$, then
  - ⋆ $r'(P) = r(P) = r(Q) = r(R)$
  - ⋆ Hence $r'(Q) < r'(P)$ or $r'(R) < r'(P)$, otherwise $r'(P) > r(P)$
  - ⋆ So $r'(Q) < r(Q)$ or $r'(R) < r(P)$, and can use extra \$ to pay for rotation

So What Does It All Mean?

If we perform m operations an have at most n nodes:

- *Any* Splay(K) needs at most $3\lfloor \log n \rfloor + 1$ \$ to maintain money invariant
- Any lookup or delete performs at most 2 splays: at most $\$(6\lfloor \log n \rfloor + 2)$
- Any insert performs 1 splay, plus money for the new root: at most $\$(4\lfloor \log n \rfloor)$
- O($m \log n$) dollars total needed—matches AVL trees!