# Splay Trees

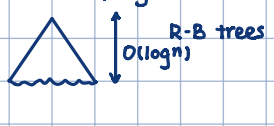
## Weight Dictionary Problem

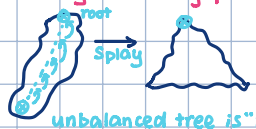
we want a data structure to insert, delete, search, sort.

If keys and the frequency they appear are known in advanced, then we can build an optimal BST using dynamic programming

If not, Splay Trees achieve the theoretical optional bound (in an amortized sense)


**Definition**: Splay trees are BSTs. key(left) ≤ key(parent) < key(right) but they have no balancing condition like R-B trees



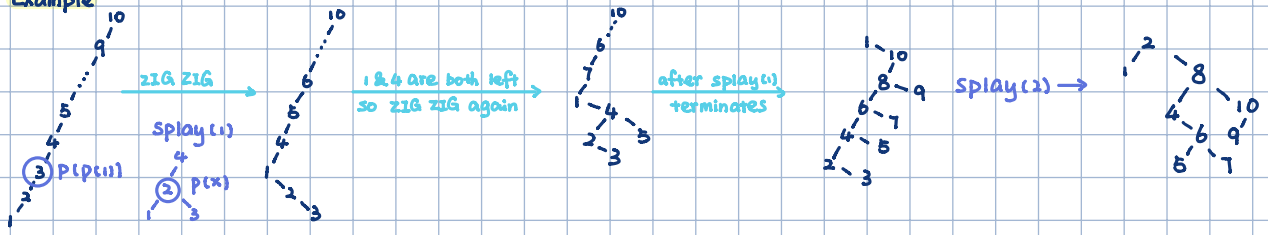

unbalanced tree is "rich" on credit

a more balanced tree has less credit

**SPLAY(x)**

```
while x ≠ root
  ZIG   if p(x) = root
          rotate p(x)
  ZIG   if p(x) & root both left or right children
  ZIG     rotate p(p(x))
          rotate p(x)
        else
  ZIG     rotate p(x)
  ZAG     rotate new p(x)
```

**Example**




## Operations on splay trees

| | cost |
|---|---|
| **Search(x)**: like BST and Splay(x) | $O(\log^n)$ |
| **Insert(x)**: like BST insert as a leaf and splay(x) | $O(\log^n)$ |
| **Delete(x)**: like a BST & splay(p(x)) | $O(\log^n)$ → not changing rank |
| **Split(x)**: | $O(\log^n)$ |


search(x) and then split into 3 splay components

| | |
|---|---|
| **Join**: | $O(\log^n)$ |


join together

## Cost of Splay

Let weight WT(x) be the # of nodes in subtree of x include x

Define rank(x) = log WT(x)

Credit Invariant: Every node x has rank(x) $s on it

Stack: ∀element has $1

Counter: ∀1-bit has $1

Claim (to be proved later) Every zig, zig-zig or zig-zag costs: $3[\text{newrank}(x) - \text{oldrank}(x)]$ except zig that may require an extra $1 sometimes

How much does this zig-zig cost?

$3(nr(x) - or(x))$

## How much splay costs? (in amortized)


total amount of dollars:

$3(\text{rank}_1 - \text{rank}_0) +$

$3(\text{rank}_2 - \text{rank}_1) +$

$\vdots$

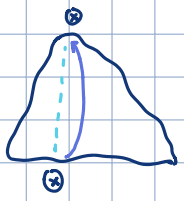
$3(\text{rank}_k - \text{rank}_{k-1}) + 1$ ⇒ zIG (maybe)

$= 3(\text{rank}_k - \text{rank}_0) + 1$

$\leq 3(\log n - \log 1) + 1$

$= 3\log n + 1 = O(\log n)$

## Proof of cost of insert (x)



A node changes in rank iff it had weight $2^k \to 2^{k+1}$

i.e. 0 node has no children, $\log 1 = 0$

1 node has 1 child, $\log 2 = 1$

2 node has 2 children, $\log 3 < 2$ ; 3 children, $\log 4 = 2$ ⇒ change rank from 2 ⇒ 3

3

4

5 ⋮ 8 maintain rank (3)

9 ⋮ 15 maintain rank (4)

16

17 ⋮ maintain rank (5)

⇒ At most $O(\log n)$ nodes may change in rank. By how much? By 1 unit

So we need at most $O(\log n)$ $ to fix those ranks