1 Splay Trees

Sleator and Tarjan, "Self Adjusting Binary Search Trees" JACM 32(3) 1985
The claim "planning ahead." But matches previous idea of being lazy, letting potential build up, using it to pay for expensive operation.
Background.

- binary search trees
- bad worst case behavior, so balance
- Lots of variants. All mark. most rotate.
- idea of self adjusting
- Show splay tree is as good as any of them!
- Static finger theorem.
- Dynamic finger conjecture.

Philosophy

- describe a rotation (draw).
- past balancing schemes require maintenance of balance info at all times, are aggresive
- idea: be lazy, only rebalance when must
- use work of searches to pay for work of rebalancing

Intuition:

- When my search descends to smaller child, happy
- because can only happen $O(\log n)$ times before done
- So, "fat children are bad"
- Cause fat children to have high potential
- Make fat children go away
- potential pays for cost
- Simple idea: single-rotate to root (cheap next time!). Doesn't work.
- more sophisticated rotations work. Idea: halve depth of **all** nodes on search path.
- Note: don't actually destroy fat children—just promote them
- unfortunate: analysis is black magic. no idea how discovered.

Draw pictures of rotations.
Search algorithm:

- Walk down tree to find item
- Splay item to root

1.1 Access Theorem

Search only. Later show insert, delete.
Analysis: different choices of weights. Note *analysis only*, don't affect implementation.
Potential function:

- **weight** w_x on each node x
- **size** $s(x)$ is total weight of subtree nodes "number of nodes"
- **rank** $r(x) = \lg s(x)$ (base 2 log) "best depth" of subtree at x
- **potential** $\Phi = \sum r(x)$

Main lemma: Amortized time to splay node x given root t is at most $3(r(t) - r(x)) + 1 = O(\log(s(t)/s(x)))$.

- Intuition: $r(t) - t(x)$ is depth of x. So if x is high in tree (fat subtree) then amortized search time is small
- (just like normal BST)
- (but we will see power of changing notion of "fat subtree")
- Analyze change for one splay step
- new sizes/ranks s', r'.
- Show potential change is $3(r'(x) - r(x)$ except $+1$ for last single rot.
- telescope sum for overall result (since final $r'(x) = r(t)$.

Analyze one step:

- Do zig-zig (hardest). zig-zag in paper.
- old y parent x and z parent y.
- Only those nodes change ranks
- Real cost 2. Potential $r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$

- Simplify:

$$\begin{aligned} 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) &\leq 2 + r'(y) + r'(z) - r(x) - r(y) \text{since } r'(x) = r(z) \\ &\leq 2 + r'(x) + r'(z) - r(x) - r(x) \\ &= 2 + r'(x) + r'(z) - 2r(x) \end{aligned}$$

- So just need to show $2 + r'(z) - r(x) \leq 2(r'(x) - r(x))$

Some intuitive analysis:

- $r'(z) \leq r'(x)$, i.e. $r'(z) - r(x) \leq r'(x) - r(x)$
- So done if $2 \leq r'(x) - r(x)$
- Trouble if $r'(x) - r(x) < 2$
- i.e., $r(x)$ almost as big as $r'(x)$
- Means most of tree weight is under $r(x)$, ie x is fat child
- Consider rotation step: initially z above x so fat, then z below x so not fat
- i.e. $r'(z) \ll r(x)$
- In which case $r'(z) - r(x)$ term cancels additive 2

Math:

- Must show $2 + r'(z) - r(x) \leq 2(r'(x) - r(x))$
- i.e., that $r'(z) + r(x) - 2r'(x) \leq -2$
- Now note $r'(z) - r'(x) + r(x) - r'(x) = \lg s'(z)/s'(x) + \lg s(x)/s'(x)$
- And note $s'(z) + s(x) \leq s'(x)$ since are seperate subtree
- So, eqn is $\leq loga + log(1 - a)$ for some $0 < a < 1$
- $= \log a(1 - a)$
- which is maximized by maximizing $a(1 - a)$ at $a = 1/2$
- which yields -2

Usage:

- One more tricky problem with potential function. Have to account for initial potential
- (remember: real cost equals amortized cost *minus* change in potential.
- So must *add* overall decrease on potential to amortized cost in order to bound real cost.

- m accesses on n nodes
- item i weight w_i, $\sum w_i = W$.
- initial potential at most $n \log W$
- final potential at least $\sum \log w_i$
- max change at most $\sum \log W/w_i$
- i.e. add once to amortized cost of splaying item i amount $O(\log W/w_i)$.
- (note potential change equals cost of splaying each item once)

1.2 Applications

Balance theorem: total access for m ops is $O((m+n)\log n)$ (as good as any balanced tree)

- weight $1/n$ to each node.
- potential drop $n \log n$
- amortized cost of search: $1 + 3 \log n$

Static Optimality: (as good as any fixed tree)

- item i accessed $p_i m$ times
- lower bound for static access: $m \sum p_i \log 1/p_i$ (entropy)
- item weight p_i
- $W = 1$
- access time for item i at most $3(1 - \log p_i) + 1 = O(1 + \log 1/p_i)$
- potential drop $O(\sum \log 1/p_i)$.

Static finger theorem:

- $w_i = 1/(1 + |i - f|)^2$
- $\sum w_i \leq 2 \sum 1/k^2 = O(1)$
- access time $O(\log |i - f|)$
- potential drop $O(n \log n)$

Working set theorem:

- At access j to item i_j, let t_j be number of distinct items since that item was last accessed. Then time $O(n \log n + \sum \log t_j)$.

Unified theorem: cost is sum of logs of best possible choices from previous theorem.
Balance theorem: total access $O((m+n)\log n)$ (as good as any balanced tree)

- weight 1 to each node.
- potential drop $n \log n$
- amortized cost of search: $1 + 3\log n$

Static Optimality: (as good as any fixed tree)

- item i accessed $p_i m$ times
- lower bound for static access: $m \sum p_i \log 1/p_i$ (entropy)
- item weight p_i
- $W = 1$
- access time for item i at most $3(1 - \log p_i) + 1 = O(1 + \log 1/p_i)$
- total $O(\sum (p_i m) \log 1/p_i)$
- potential drop $O(\sum \log 1/p_i)$.

Static finger theorem:

- $w_i = 1/(1 + |i - f|)^2$
- $\sum w_i \leq 2 \sum 1/k^2 = O(1)$
- access time $O(\log |i - f|)$
- potential drop $O(n \log n)$

1.3 Updates

Update operations: insert, delete, search (might not be there)

- define split, join
- set $w_i = 1$ so splay is $O(\log n)$.
- to split, splay and separate—splay $O(\log n)$, potential drops
- to join, access largest item and merge—splay $O(\log n)$, root potential only up by $O(\log n)$
- splits and joints have amortized cost $O(\log n)$
- insert/delete via split/join
- important to splay on unsuccessful search

Remarks

- Top down splaying.
- can choose to splay only when path is "long" (real cost to large so need to amortize). Drawback: must know weights.
- can choose to stop splaying after a while. good for random access frequenies.
- Open: dynamic optimality.
- Open: dynamic finger
- tarjan: sequential splay is $O(n)$