1 Splay Trees

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** $r(x) = \log s(x)$ "best depth" of subtree at x
- **potential** $\Phi = \sum r(x)$

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

- 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:

- old y parent x and z parent y.

Proof from paper.
Usage:

- tricky problem with potential function. Have to account for initial potential
- (remember: real cost equals amortized cost *minus* change in potential.
- just upper bound initial, charge as part of 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 $\sum \log W/w_i$
- amortized cost of splaying item i is $O(\log W/w_i)$.
- (note potential change equals cost of splaying each item once)

1.2 Applications

Balance theorem: total access $O((m+n)\log n)$ (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 $O(n)$

1.4 Persistent Data Structures

Sarnak and Tarjan, "Planar Point Location using persistent trees", Communications of the ACM 29 (1986) 669–679
"Making Data Structures Persistent" by Driscoll, Sarnak, Sleator and Tarjan Journal of Computer and System Sciences 38(1) 1989
Idea: be able to query and/or modify past versions of data structure.

- ephemeral: changes to struct destroy all past info
- partial persistence: changes to most recent version, query to all past versions
- full persistence: queries and changes to all past versions (creates "multiple worlds" situtation

Goal: general technique that can be applied to *any* data structure.
Application: planar point location.

- planar subdivision
  - n segments meeting only at ends
  - defines set of polygons
  - query: "what polygon contains this point"
- numerous special-purpose solutions
- One solution:
  - vertical line through each vertex
  - divides into slabs
  - in slab, segments maintain one vertical ordering
  - find query point slab by binary search
  - build binary search tree for slab with "above-below" queries
  - n binary search trees, size $O(n^2)$, time $O(n^2 \log n)$
- observation: trees all very similar
- think of x axis as time, slabs as "epochs"
- at end of epoch, "delete" segments that end, "insert" those that start.
- over all time, only n inserts, n deletes.
- must be able to query over all times

Persistent sorted sets:

- find(x, s, t) find (largest key below) x in set s at time t

- insert(i, s, t) insert i in s at time t
- delete(i, s, t).

We use partial persistence: updates only in "present"
Implement via persistent search trees.
Result: $O(n)$ space, $O(\log n)$ query time for planar point location.