Self-Adjusting Data Structures
Self-Adjusting Data Structures

Lists

Dictionaries

Priority Queues


Okasaki: *maxiphobic heaps are an alternative to leftist heaps ... but without the “magic”*
Heaps (via Binary Heap-Ordered Trees)

MakeHeap, FindMin, Insert, **Meld**, DeleteMin

\[\text{Meld} \quad \text{Cut root + Meld}\]

Leftist Heaps


Each node **distance to empty leaf**

**Inv.** Distance right child ≤ left child

⇒ rightmost path ≤ \(\lceil \log n + 1 \rceil\) nodes

Time \(O(\log n)\)

Maxiphobic Heaps

Meld (\(x\), \(y\))

\[\begin{align*}
T_1 & \quad x < y \\
T_2 & \\
T_3 & =
\end{align*}\]

\[\begin{align*}
\text{Meld} \quad (T_i, \quad ) \\
T_j & \quad \text{two smallest} \\
T_k & \quad \text{largest size}
\end{align*}\]

\[\begin{align*}
\text{Max size } n & \rightarrow \frac{2}{3}n \\
\text{Time } O(\log_{3/2} n)
\end{align*}\]
Skew Heaps


- Heap ordered binary tree with **no** balance information
- MakeHeap, FindMin, Insert, **Meld**, DeleteMin
- **Meld** = merge rightmost paths + swap *all* siblings on merge path

\[ \text{Meld (} 4, 2 \text{)} = 4 \]

\( v \text{ heavy if } |T_v| > |T_{p(v)}|/2, \text{ otherwise light} \)

\( \Rightarrow \) any path \( \leq \log n \) light nodes

Potential \( \Phi = \# \text{ heavy right children in tree} \)

\( O(\log n) \) amortized Meld

**Heavy** right child on merge path before meld \( \Rightarrow \) replaced by **light** child

\( \Rightarrow \) 1 potential released for heavy node

\( \Rightarrow \) amortized cost \( 2 \cdot \# \text{ light children on rightmost paths before meld} \)
Skew Heaps – O(1) time Meld


- **Meld** = Bottom-up merge of rightmost paths + swap all siblings on merge path

\[\Phi = \# \text{ heavy right children in tree} + 2 \cdot \# \text{ light children on minor \& major path}\]

**O(1) amortized Meld**

Heavy right child on merge path before meld \(\rightarrow\) replaced by light child \(\Rightarrow 1\) potential released

Light nodes disappear from major paths (but might \(\rightarrow\) heavy) \(\Rightarrow \geq 1\) potential released

\(4\) and \(5\) become a heavy or light right children on major path \(\Rightarrow\) potential increase by \(\leq 4\)

**O(log n) amortized DeleteMin**

Cutting root \(\Rightarrow 2\) new minor paths, i.e. \(\leq 2 \cdot \log n\) new light children on minor \& major paths
Splay Trees


- Binary search tree with **no** balance information
- **splay(x)** = rotate x to root (zig/zag, zig-zig/zag-zag, zig-zag/zag-zig)

---

**Binary search tree with no balance information**

**splay(x)** = rotate x to root (zig/zag, zig-zig/zag-zag, zig-zag/zag-zig)

**Search** (splay), **Insert** (splay predecessor+new root), **Delete** (splay+cut root+join), **Join** (splay max, link), **Split** (splay+unlink)

---

**Search** (splay), **Insert** (splay predecessor+new root), **Delete** (splay+cut root+join), **Join** (splay max, link), **Split** (splay+unlink)
Splay Trees


- The access bounds of splay trees are amortized
  
  (1) $O(\log n)$
  (2) Static optimal
  (3) Static finger optimal
  (4) Working set optimal (proof requires dynamic change of weight)

- **Static optimality:** $\Phi = \sum_v \log |T_v|$