Amortized Analysis

Applications: Data Structures
   Eg: Binomial Heaps, Fibonacci Heaps
      Union-Find, Splay Trees

DS Applications: Dijkstra's Alg (shortest path)
                 Kruskal's Alg (MST)

See CLRS Ch 17 for intro

Main Trick: Lazy DS Versus Eager DS
Timing Analysis Methods

Worst Case: eg Strassen

Average Case (average over inputs):
eg Quick sort pivoting on first element

Amortized Analysis (worst case input, averaged over time):

Randomized (worst case input, averaged over coin flips):
eg (QS Randomized)

Amor Anal CHRS Chap 17

3 Methods

Aggregation Method (won't use)

Accounting Method (Kozen)

Potential Method (will use)
Example: Dijkstra's Alg

Recall: Dijkstra computes single source shortest path
\[ \text{dist}(s, v) \forall v \in V \quad \# \text{vertex} = n \quad \# \text{edges} = m \quad m > n \]

It uses priority queues:

**Input:** set of keys \( k_1, \ldots, k_n \)
with priorities \( \text{Prior}(k_i), \ldots, \text{Prior}(k_n) \)

<table>
<thead>
<tr>
<th>Dijkstra</th>
<th>( # \text{op} )</th>
<th>( # \text{op for Heap} )</th>
<th># total unit ( \text{cost} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>makeheap((S))</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>findmin((S))</td>
<td>( n )</td>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>insert((k,S))</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( n \log n )</td>
</tr>
<tr>
<td>deletemin((S))</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( n \log n )</td>
</tr>
<tr>
<td>decrease key</td>
<td>( m )</td>
<td>( \log n )</td>
<td>( m \log n )</td>
</tr>
</tbody>
</table>
Consider some operation $O_p$

Suppose it is called $n$ times

Costing $O_p = O_{pn}$

Goal has total cost $\sum_{i=1}^{n} O_{p_i} = TC$

Simplest idea: compute $WC = \max_{i} O_{p_i}$

then $TC \leq n \cdot WC$

2-Problems:

1) $n(WC)$ too big (over estimate)

2) We won't even consider better alg.

Idea 2: Compute $Avg = \{ O_{p_1}, \ldots, O_{p_n} \}$

then $TC \leq n \cdot Avg$

Prob: Hard to estimate $Avg$!
Trick: Add an artificial cost per operation!

Def \( E: \text{state} \rightarrow \mathbb{R} \) a potential

Def \( \text{unit-cost} = O_p_i \)

Def Amortized Cost = unit-cost + potential change

\[
\sum AC_i = \sum O_p_i + (E_i - E_{i-1}) = \sum O_p_i + E_n - E_0 = TC + \Delta \Phi
\]

if \( \Delta \Phi \geq 0 \) then \( TC \leq \sum AC_i \)

\[ AC = \max_i AC_i \] then \( TC \leq n \cdot AC \) if \( \Delta \Phi \geq 0 \)
Priority Queues

Heaps (Veg, Binomial, Fibonacci)

CLRS Chaps 6, 19, 20 Koen 8, 9

<table>
<thead>
<tr>
<th>Abstract Data Type</th>
<th>Binomial Eager/Hazy</th>
<th>Fibonacci Amort</th>
</tr>
</thead>
<tbody>
<tr>
<td>make-heap(S)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>find-min(S)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>insert(K, S)</td>
<td>O(log n)</td>
<td>O(log n) / O(1)</td>
</tr>
<tr>
<td>delete-min(S)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Union-Meld(A, B)</td>
<td>O(n)</td>
<td>O(log n) / O(1)</td>
</tr>
<tr>
<td>decrease-key(g, K, S)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
</tbody>
</table>
Regular Heap (recall)

Heap: 1) Balanced Binary Tree $T$ (Heap)
    2) Priority($p(x)$) $\leq$ Prior($x$)
        (Heap property) $p(x) = parent(x)$

makeheap = make empty tree
findmin = return root
insert = 1) add new leaf
        2) let it "float up"
deletemin = 1) replace root with a leaf
        2) "float down" root
meld = ?
Binomial Heaps (cheap meld)

Idea: 1) balanced $\rightarrow$ almost balanced
2) binary $\rightarrow (\log n)$-ary
3) link

Problem: $T_1 \cdots \hat{T}_{i-1} \hat{T}_i \cdots T_n$

\[ T'_i = \text{Meld} (T_{i-1}, T_i) \]
\[ \text{Meld} (T_{n-1}, T_n) \]

we get

(not balanced?)
Idea: Meld (link) trees of same size

**Def**: Binomial Tree

\[ B_0 = 0, \quad B_1 = 1 \]

\[ B_{k+1} = \begin{array}{c}
B_0 \\
B_1
\end{array} \]

**Note**: $B_{k+1}$

Claim: depth($B_k$) = $k$

\[ |B_k| = 2^k \]

Pointer stored
\[ C = c, c_x, A_3, c \]

\[ \begin{align*}
A &= A_0, A_1, A_2, A_3 \\
B &= B_0, B_1, B_2 \\
\text{Meld}(A, B) &= \text{link until one tree per rank} \\
\text{Link}(A, B) &= \text{combine 3 trees of same rank} \\
\end{align*} \]

**Eager Binomial Heap**

\[ \text{Claim: } B(k, i) = \binom{k}{i} \text{ prod } (\binom{2^{i+1}}{i}) \]

\[ B(k, i) \equiv \text{mod at depth } i \]

\[ \text{Prove: } \binom{2^{i+1}}{i} \]

\[ \begin{align*}
\binom{2^{i+1}}{i} &= \frac{(2^{i+1})!}{i!(2^{i+1}-i)!} \\
&= \frac{(2^{i+1})(2^{i+1}-1)(2^{i+1}-2)\cdots(2)}{i!} \\
&= (1 \cdot 2) \cdot (2 \cdot 3) \cdot \cdots \cdot (2^{i-1} \cdot 2^i) \\
&= 2^i \cdot (1 \cdot 2) \cdot (2 \cdot 3) \cdot \cdots \cdot (2^{i-1}) \\
&= 2^i \cdot \binom{2^i}{i} \\
\end{align*} \]

Why Binomial?
\[ \text{insert}(k, A) = 1) \quad \text{makeHeap}(k, B) \]
\[ \text{meld}(\overline{A}, \overline{B}) \]

\[ \text{deleteMin}(A) = 1) \quad \text{Binary trees of } A \quad T_i \quad T_k \]
\[ 2) \quad \text{find tree with min root } T_i \]
\[ 3) \quad \text{remove } T_i \text{ from } A \]
\[ 4) \quad \text{remove root of } T_i \text{ and write as } BA \quad A' \]
\[ 5) \quad \text{meld}(A, A') \quad (\text{Isager}) \]

\[ A = (T_0, T_1, T_2, T_3, T_4) \quad \text{and min key in root } T_4 \]

\[ \text{deleteMin}(A) \quad A = (T_0, T_1, T_2, T_3) \]
\[ A' = (T_0', \ldots, T_3') \]
\[ \text{meld}(A, A') = (T_1'', \ldots, T_4'') \]

\[ O(\log n) \quad \text{time} \]
Amortize analysis of insert
potential argument: $F(A) = \# \text{trees}$

Note: cost of insert = # of trees linked = # for trees removed

Amort cost: unit cost + $\Delta F$  
\[= 1 + \# \text{tree removed} + \# \text{trees removed} \]
\[= 1 \]

Long Melds

1) Trees are kept as a linked list
2) meld = list concatenation
3) determine first reduces # of trees to at most one per rank. b) find new min

$F(A) = 2 \cdot \# \text{trees} \quad m = \# \text{trees}$

unit cost of deletion = $m + k \quad K = \# \text{links} = \# \text{tree dele}$

Amort cost = $(m + k) + \Delta F$  
$m + k - 2x = m - k = \text{remaining trees} \leq \log n$