Amortized Analysis 1/18/19

Applications: Data Structures (DS)
DS eg: Binomial Heaps, Fibonacci Heaps
    Union-Find, Splay Trees

Applications using DS
    Dijkstra's Alg (graph shortest paths)
    Kruskal's Alg (Min Spanning Tree)

See CLRS Ch17 for intro

Main Trick: Sometimes faster to be lazy than eager
Timing Analysis Methods

1) Worst Case over all inputs: eg Strassen

2) Average over inputs:
   eg Quick Sort pivoting on first element.

3) Amortized Analysis:
   (worst case input, average over time)

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

Amortized Analysis methods

1) Aggregation Method (won't use)
2) Accounting Method or Token method (Kogen)
3) Potential Method (we will use)
Regular Heap \ (recall) 

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

makeheap \equiv\ make\ empty\ tree
findmin \equiv\ return\ root
insert \equiv\ 1)\ add\ new\ leaf,
        2)\ let\ it\ "float\ up".
deletemin \equiv\ 1)\ replace\ root\ with\ a\ leaf,
        2)\ let\ root\ "float\ down".
meld \equiv\ Combine\ 2\ heaps\ into\ one.

By\ adding\ meld\ we\ speedup\ other\ ops.
**Application: Dijkstra's Alg**

Recall Dijkstra computes single source shortest path in weighted graph.

\[ \text{i.e. Computes } \text{dis}(s,v) \forall v \in V \]

\[ \text{#vertices } = n \]
\[ \text{#edges } = m \quad m > n \]

It uses a priority queue:

Over time it receives keys \( k_1, \ldots, k_n \) with priorities \( \text{Prior}(k_1), \ldots, \text{Prior}(k_n) \)

### Dijkstra Alg Doses:

<table>
<thead>
<tr>
<th>Ops</th>
<th>#calls</th>
<th>Worst case cost</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>make heap ((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>delete( \text{min} ( (s) )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( n \log n )</td>
</tr>
<tr>
<td>decrease( \text{key} ((k,s))</td>
<td>( m )</td>
<td>( \log n )</td>
<td>( m \log n )</td>
</tr>
</tbody>
</table>

Dominate cost is decrease key!

Goal in next 2 lectures

Make AC of decrease\( \text{key} \) to be \( O(1) \).
Consider some operation $O_p$

Suppose it is called $n$ times

Costing $O_p, \ldots, O_n$

Good bd 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 $\text{Avg} = \{O_{p_1}, \ldots, O_{p_n}\}$

then $TC \leq n \cdot \text{Avg}$

Prob: Hard to estimate $\text{Avg}$!
Faster Priority Queues

Types of Heaps: Reg, Binomial, Fibonacci

See CLRS Chaps 6, 19, 20 Kojun 8, 9

<table>
<thead>
<tr>
<th>Ops</th>
<th>Reg</th>
<th>Eager/Lazy</th>
<th>Amort</th>
</tr>
</thead>
<tbody>
<tr>
<td>make-heap(s)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>find-min(s)</td>
<td>O(1)</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>
<td>O(1)</td>
</tr>
<tr>
<td>delete-min(s)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>meld(A, B)</td>
<td>O(n)</td>
<td>O(log n) / O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>decrease-key(k, s)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
Binomial Heaps (cheap meld)

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

Problem: \[ T_1 \cdots T_n \]

\[ T'_2 = \text{Meld} (T_1, T_2) \]

\[ \vdots \]

\[ \text{Meld} (T_{n-1}, T_n) \]

We may get: 

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

Def: Binomial Tree

\[ \begin{align*}
B_0 & = \cdot \quad B_1 = \cdot 1 \quad B_2 = \cdot B_1 \quad B_{k+1} = \cdot B_k \cdot B_k \\
\end{align*} \]

Note: \( B_{2n} \) in

Claim: depth(\( B_k \)) = \( k \) &
\[ |B_k| = 2^k \]

Pointers stored

(Children)
Why Binomial? \( B_4 = \)

\[
\text{Def: } B(k, i) = \# \text{ nodes at depth } i
\]

\[
\text{Claim: } B(k, i) = \binom{k}{i} \quad \text{proof} \quad \binom{n+1}{i} = \binom{n}{i-1} + \binom{n}{i}
\]

Eager Binomial Heap

\[
\text{Link}(A, B) = \begin{align*}
1) & \text{ combine 2 trees of same rank} \\
2) & \text{ increment rank of root}
\end{align*}
\]

\[
\text{Meld}(\overline{A}, \overline{B}) = \text{Link until at most one tree per rank.}
\]

\[
\begin{align*}
\bar{A} &= A_0, A_1, A_3, A_4 \\
\bar{B} &= B_0, B_1, B_4 \\
\bar{C} &= C_1, C_2, A_3, C_5
\end{align*}
\]
\[ \text{msert}(k, \bar{A}) \equiv \]
\[ 1) \text{makeheap}(k, \bar{B}) \]
\[ 2) \text{meld}(\bar{A}, \bar{B}) \]

\[ \text{deletemin}(\bar{A}) \equiv 1) \text{Suppose } \bar{A} = (T_1, \ldots, T_K) \text{ (trees)} \]
\[ a) \text{Find tree with min root}, T_i. \]
\[ 3) \text{remove } T_i \text{ from } \bar{A} \]
\[ 4) \text{remove root}(T_i) \text{ giving } BH, \bar{B} \]
\[ 5) \text{return meld}(\bar{A}, \bar{B}) \text{ (Eager)} \]

**Example:**
\[ \bar{A} = (T_0, T_1, T_2, T_3, T_4) \] where

- 0
- 1
- 2
- 3
- 4

```
                4                     min key
               / \                     / \ \ / \ / \
     3        / \ \                     / \ \ / \ / \
   / \ \    /   \ \  \                     /   \ / \
  /     \   /       \ \ \                     /       \
 0  1  2  3            T_2 T_1 T_0
```

\[ T_3' T_2' T_1' T_0' \]
Eager-Delete min

\( \text{deletemin}(A) \quad \bar{A} = (T_0, T_1, T_2, T_3) \)

\( \bar{B} = (T_0', T_1', T_2', T_3') \)

\( \text{meld}(\bar{A}, \bar{B}) = (T_1'', T_2'', T_3'', T_4'') \)

Note: deletemin is \( O(\log n) \) time.
Potential Method

Trick: Add an artificial cost per operation!

\[ \text{Def } \Phi : \text{state } \rightarrow \mathbb{R} \text{ a potential} \]

\[ \text{Def } \text{unit-cost} = O_{\Phi i} \]

\[ \text{Def } \text{Amortized Cost} = \text{unit-cost} + \text{potential change} \]

\[ \sum AC_i = \sum [O_{\Phi i} + (\Phi_i - \Phi_{i-1})] = \sum O_{\Phi i} + \Phi_n - \Phi_0 \]
\[ = TC + \Delta \Phi \]

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

\[ AC = \max_i AC_i \text{ then} \]
\[ TC \leq n \cdot AC \text{ if } \Delta \Phi \geq 0 \]
Amortized analysis of insert

Use potential argument

\[ \Phi(A) = \# \text{trees} \]

Def unit-cost of insert = \#trees linked + 1

= \#trees removed + 1

Amortized Cost = unit-cost + \Delta \Phi

= 1 + \#trees removed - (\#tree removed - 1)

= 2
Note Amort-Cost of Eager-Deletemin still $O(\log n)$

Lazy Melds

1) Trees are kept as a linked list.
2) Lazy-Meld $\equiv$ list concatenation
3) Lazy-Deletemin $\equiv$
   a) Link trees till at most one per rank.
   b) Do Eager-Deletemin
4) Insert using Lazy-Meld.

Amort Analysis of Lazy-Meld DS.

Potential function $\Phi \equiv \# \text{trees}.$

$AC(\text{Insert}) = 2$

$AC(\text{Lazy-Deletemin})$

Step $a$: Unit-Cost $\equiv \log n + \# \text{trees deleted}$

$\Delta \Phi = \# \text{trees deleted}$

$AC(\text{Step } a) = \log n$

$AC(\text{Step } a) = \log n$ from above

$\equiv O(\log n)$
Next Time

Goal: Find a lazy DecreaseKey.

Note: DeleteMin worked because removing the root of a binomial shattered it into smaller binomials.