Admin: Read CLRS Chapter 17

Today: Amortized & Competitive Analysis

- [ ] Table-Doubling (Amortized analysis)
- [ ] Move-To-Front (Competitive analysis)
Amortized Analysis

- Assume we have data structure supporting some basic ops (e.g., insert, search)
- So far, looked at w.o.c. time per opn (e.g., $\Theta(\log n)$ for lookup in BST)
- Now, we look at sequence of opns $S = \langle B_1, B_2, \ldots, B_m \rangle$
  
  opn $B_i$ has cost $c_i$
  
  and transforms data structure from $D_{i-1}$ to $D_i$
  (e.g., $D_0 = \text{empty data structure}$)

Then $\sum_{i=1}^{m} c_i$ is (actual) cost of $S \triangleq \mathcal{C}(S)$

How to bound $\mathcal{C}(S)$ when
  
  - $c_i$'s may vary widely (e.g., table doubling), or
  - alg may try to optimize data structure (MTF)
Consider table of items with just insert:

- $D_0 =$ empty table
- $D_i =$ contains $i$ elements that have been inserted

What to do when table becomes full? $(i = 2^k + 1)$

- Allocate new table of $2 \times$ size $(size = 2^{k+1})$
- Copy elements over $(cost = 2^k)$
- Insert new element

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\downarrow & \\
1 & 2 & 3 & 4 & 5 & \cdots
\end{array}
\]

$c_i = \begin{cases} 
1 & \text{if } i \neq 2^k + 1 \text{ for any } k \\
2^k + 1 & \text{if } i = 2^k + 1 \text{ for some } k
\end{cases}
$

(just insert) (double, then insert)

What is $C(S) = \sum_{i=1}^{m} c_i$ ?

Not hard to do sum in this case to get $\Theta(m)$, but let's develop general method. Book gives 3 methods:

- aggregate 
- accounting 
- potential
Potential method

Let $\Phi_i = \Phi(D_i) = "potential" \ or \ "bank \ balance" \ associated \ with \ D_i$

$\Phi_0 = 0 \ \Rightarrow \ \text{required \ for \ this \ method}$

$\Phi_i \geq 0 \ \Rightarrow$

Given $\Phi$, we can define amortized cost $\hat{C}_i$ of $i$th operation as

$$\hat{C}_i = C_i + (\Phi_i - \Phi_{i-1})$$

change in potential = change in bank balance $\Delta \Phi_i$

if $\Delta \Phi_i > 0 \Rightarrow$ prepaying for later work $\Phi \uparrow$

if $\Delta \Phi_i < 0 \Rightarrow$ use saved work, withdrawal $\Phi \downarrow$

$$\sum_{i \leq m} \hat{C}(S) = C(S) + \Phi_m - \Phi_0 \quad \text{(telescopes)}$$

$\Rightarrow C(S)$ since $\Phi_0 = 0$

$\Phi_m \geq 0$

Amortized cost of sequence is upper bound on actual cost.
Apply to table doubling:

Let \( \Phi_i = 2^i \) (#items that have never been moved)

\[
\begin{array}{cccccccccc}
  i & = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\Phi_i & = & 2 & 2 & 2 & 4 & 2 & 4 & 6 & 8 & 2 \\
\hat{c}_i & = & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 \\
\hat{c}_i & = & 3 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\end{array}
\]

E.g., inserting 5 prequets for moving 5 & 1

inserting 6 " " " " 6 & 2

" 7 " " " 7 & 3

" 8 " " " 8 & 4

When 9 comes along we have $8$ in bank

So \( \hat{c}(s) \leq 3 \ast m \)

and \( c(s') \leq \hat{c}(s') \leq 3m \)

Total actual cost for any \( m \) is \( \Theta(m) \)

Note: amortized cost is defined relative to \( \Phi \); different choices for \( \Phi \) can yield different bounds.
Competitive Analysis (Move to Front)

- Uses amortized analysis as a tool here

- Assume we have n elements in a list, unordered (think: list of a hash table with chaining)

- Only operation is search \( x \) \( (x = \text{key}) \)
  (no insert or delete)

- \( \text{cost} = \# \text{elements examined} \in \{1, 2, \ldots, n\} \)

- For a sequence \( S = \langle x_1, \ldots, x_m \rangle \) of keys
  
  \[
  \text{cost} = C(S) = \sum_{i=1}^{m} c_i
  \]
  
  where \( c_i = \text{position of } x_i \text{ in list} \)

- Order of list matters!

- Want most frequent requests up front
  (could keep count of # requests/item, &
  keep list sorted into decreasing order by count...)

- Maybe then, we should consider algorithms
  that dynamically update order after each request...

- It may help to move element just searched
  for closer to front of list.
• Consider class of algs that may move elt just
  searched for closer to front. (Assume cost of movement
  = 0, it just involves adjusting a few pointers...)
• Move-To-Front (MTF)
• Move-Up-One
• Let \( C_A(S) \) = cost of running alg A on seq \( S \)
  of requests \( B \) (e.g., \( A = \text{MTF} \))
• can compare \( C_A(S), C_B(S') \) for different algs \( A, B \)
• algorithm is on-line if processing \( x_i \), it
  doesn't know future requests \( (x_{i+1}, ..., x_n) \)
• algorithm is off-line if it does know future
  (sometimes called "God's algorithm")
• Let \( \text{OPT} \) be best off-line algorithm of this class.
• Would expect \( C_{\text{OPT}}(S') \) to be much smaller
  than \( C_{\text{MTF}}(S) \) for many (most?) seqs \( S' \);
  knowing future should help!
Beautiful result (Sleator & Tarjan 1985):

Theorem: \((\forall S)\ C_{MTF}(S) \leq 2C_{OPT}(S)\)

"competitive analysis": MTF is not worse than 2* any other alg, online or off-line!

MTF very useful in practice, too!

Proof defines potential \(\Phi_i\) at 1st order for MTF after \(i\) ops wrt 1st order after \(i\) ops of \(OPT_i\); then does amortized analysis to bound \(C_{MTF}(S)\). (Unusual amortized analysis since it depends on \(OPT_i\).)
Let \( L_i = \text{MTF's list after } i \text{ opns} \)
\[ L_i^* = \text{OPT's list after } i \text{ opns} \]
(Assume \( L_0 = L_0^* \) — they start with same list order)

Consider \( i^{th} \) step \( (i = 1, 2, \ldots, m) \) \(|S| = m\)

Suppose \( x_i \) is in position \( j \) in \( \text{OPT's list } L_i^* \)
\( x_i \) is in position \( k \) in \( \text{MTF's list } L_{i-1} \)

Let \( \Phi_{i-1} = \# \text{ inversions in these lists} \)
= \# pairs \( \{x_i, x_j\} \) in different relative order

\[ 0 \leq \Phi_{i-1} \leq \binom{n}{2} \] \( \Phi_0 = 0 \)
\[ \hat{C}_i = C_i + \overline{\Phi}_i - \overline{\Phi}_{i-1} \quad \text{for MTF} \]

\[ = k + \Delta \overline{\Phi}_i \]

\[ \Delta \overline{\Phi}_i = \overline{\Phi}_i - \overline{\Phi}_{i-1}, \quad \text{composed of two parts} \]

- Change caused by MTF moving \( x_r \) to front
- Change caused by OPT moving \( x_r \) up some unit

Let \( v \) = \# items after \( x_r \) in OPT's list, but before \( x_r \) in MTF's list

\[ \hat{C}_i \leq k + \# \text{increase in } \Phi \text{ by moving } x_r \text{ to front} \]

\[ \leq k + ( (k-1-v) - v ) \]

\[ = ( k - v ) - 1 \]

\[ \text{Lemma } k-v \leq j \]

\[ \text{Pf: There are } k-1-v \text{ in MTF list, which are also before } x_r \text{ in OPT's list.} \]
So \( \hat{c}_i \leq 2j - 1 \) for \( i = \text{search} \)

\[
C_{MTF}(S') = \sum_{i=1}^{13} c_i \\
\leq \sum_{i=1}^{13} \hat{c}_i \\
\leq 2C_{opt}(S') - m \\
\leq 2C_{opt}(S)
\]

Knowing the future helps by factor of at most 2.

Similar result even if OPT is allowed other rearrangements, in addition to moving searched-for elt closer to front.