Lecture 18:
Dealing with an uncertain future

- Amortized Analysis
  - table doubling
- Competitive Analysis
  - Move-to-front
Amortized Analysis

- Setting:
  Sequence of operations
  eg. Binary search tree - insert, delete, lookup

- Efficiency?
  So far: time per operation WORST CASE
  Today: average of sequence of operations \( S' = \langle B_1, B_2, \ldots, B_m \rangle \)
  - still worst case over choice of ops
  - but some in sequence can take a while
  as long as average remains low

Operation \( B_i \) has cost \( c_i \)
Total cost of \( S' = \Sigma c_i \)
Amortized cost of \( S' = \frac{\Sigma c_i}{m} \)

\( \text{Worst case over all inputs, all operation sequences, } \)
\( \text{average cost of op is same over the sequence} \)

**Sanity Check Question?**
Which should be easier to prove, worst case or amortized?

The difficulty:
- \( c_i \)'s can vary widely
- algorithm may try to optimize data structure
Table Doubling

Table of items - array of slots in contiguous memory \( \leftarrow \) no particular order

operations: insert (delete) \( \leftarrow \) not now!

How many slots to allocate? too many is a waste, not enough is an obvious problem!

Notation: \( D_0 \) = empty table
\( D_i \) = table with \( i \) elements inserted

Plan: what if table fills up? \( (i = 2^{k+1}) \)

- Allocate new table 2x size of old table \( (\text{size } 2^{kn}) \)
- copy elements over
- insert new element

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

Cost \( c_i = \frac{1}{d^k+1} \) \( i \geq 2^{k+1} \) for some \( k \)

just insert double, copy, insert

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

\( \Theta(m) \) easy to see (you probably saw in 6.006)

3 book methods:
- aggregate
- accounting
- potential \( \Rightarrow \) today!
Potential Method:

- Assign costs to ops which include extra charges to cover later big things
  (similar to paying constant monthly fee instead of unpredictable amounts that are sometimes small, sometimes large).
- Let $\Phi_i = \Phi_i(0_i) = \text{"potential" or "bank balance" associated with } i$
  $= \text{amt of prepaid work}$
  $\Phi_0 = 0$
  $\Phi_i \geq 0$

- Given $\Phi_i$, define "amortized cost" $\hat{C}_i$ of $i^{th}$ op as
  $\hat{C}_i = C_i + (\Phi_i - \Phi_{i-1})$
  \text{change in potential } = \Delta \Phi_i$
  Sum over \(1 \leq i \leq m\)
  if $\Delta \Phi_i > 0$: paying for later work $\uparrow$
  if $\Delta \Phi_i < 0$: use saved work, withdrawal $\downarrow$

  $\hat{C}(s) = \hat{C}(s) + \Phi_m - \Phi_0$ (since telescopes)
  $\geq C(s)$ (since $\Phi(m) = 0$ $\Phi(0) = 0$)

  so $\hat{C}$ is upper bound on $C$!!
Application to table doubling:

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

|=1 right after create table

=\( j \) when \( j \) inserts after creation

\( t = 2^{k+1} \) when \( i = 2^{k+1} \) for \( i \geq 3 \)

(right before table expand step)

\[
\begin{array}{c|cccccccc}
  i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
 \text{new table?} & \checkmark & \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\
 \Phi_i & 2 & 2 & 2 & 4 & 2 & 4 & 6 & 8 & 2 \\
 \Phi_i \Phi_i^{-1} & 2 & 0 & 2 & -2 & 2 & 2 & 2 & -6 \\
 C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 \\
 \hat{C}_i & 3 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\end{array}
\]

when table expands: \( (t,i \geq 3) \) note that if \( i \leq 2 \), \( \hat{C}_i \leq 3 \)

all items moved, so
1) \( \Delta \hat{\Phi} = 2 \left[ 1 - 2^{k-1} \right] = 2 \left[ 1 - \frac{t-1}{2} \right] = 3 - t \)

\[
\hat{C}_i = \frac{t-1}{2} \quad \text{when} \quad k \geq 1 \quad \text{for} \quad i \geq 3
\]

2) \( C_i = i \)

\( \Rightarrow \hat{C}_i = C_i + \Delta \hat{\Phi} = 3 \)

when table doesn't expand:

there is one new item that hasn't moved so
1) \( \Delta \hat{\Phi} = 2 \cdot 1 \) \( \Rightarrow \hat{C}_i = 1 + 2 = 3 \)

2) \( C_i = 1 \)

\( \Rightarrow \hat{C}_i = 1 + 2 = 3 \)
another way to view it:

<table>
<thead>
<tr>
<th>Insert</th>
<th>Pays for moving</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5 $1</td>
</tr>
<tr>
<td>6</td>
<td>6 $2</td>
</tr>
<tr>
<td>7</td>
<td>7 $3</td>
</tr>
<tr>
<td>8</td>
<td>8 $4</td>
</tr>
</tbody>
</table>

When 9 comes along, we have $8 in the bank!

by the above,

\[ C(s) \leq \hat{C}(s) \leq 3m \]

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

Note: other choices of \( \phi \) may give different (better?) bounds.
Move-to-Front via Competitive Analysis — will use amortized analysis as a tool!

Given a elements in unordered list
e.g., hash table with chaining

Operation: search (x)   X = key
Cost: # elements in list examined
given sequence \( S = \langle X_1, \ldots, X_m \rangle \) of keys
Cost is \( C(S) = \sum_{i=1}^{m} C_i \)
where \( C_i = \text{position of } X_i \text{ in list} \)

ORDER OF LIST MATTERS!!

What can we do?

- would like most frequent request up front

if we know statistics, can order
elements according to usage
(most accessed, 2nd most accessed, ...)

"Dynamic Updates": Keep counts & reorder list according to (past) counts

A = class of algorithms that move element
just searched for closer to front

- assume cost to move = 0, since just adjust
a few pointers
- e.g., Move-to-Front (MTF)
Move-up one
Let \( C_A(s) = \text{cost of running alg } A \text{ on sequence } S \)

\[ \uparrow \]

Can compare algs \( A \& B \)
\[ \text{via } C_A(s) + C_B(s) \]

- Algorithm is \underline{online} if doesn't know future requests
  - i.e. when processing \( x_i \) doesn't know \( x_{i+1}, x_{i+2}, \ldots, x_n \)

- Algorithm is \underline{offline} if knows all requests before processing
  Call best offline algorithm in \( A \) "\( \text{OPT} \)"

How much does it help to know the future?

is \( C_{\text{OPT}} \) much better than \( C_A \) for every online \( A \)?

\[ \text{or} \]

is there a really good online algorithm?

Surprising Theorem \[ \text{Sleator Tarjan} \]
\[ \text{Thm } \forall S, \quad C_{\text{HTF}}(s) \leq 2 \ C_{\text{OPT}}(s) \]

\[ \text{Competitive ratio} \]

is, \( \text{HTF never worse than twice as bad} \)

as any algorithm in \( A \) - even offline!

\[ \text{(useful in practice too!)} \]
Proof uses amortized analysis via potential fn,

depends on OPT!

ie. \( \Phi_i \) compares MTF list order to OPT list order

**Notation**

\( L_i^* \) = MTF's list after \( i \) ops of \( S \)

\( L_i^* \) = OPT's list after \( i \) ops of \( S \)

(assume \( L_0 = L_0^* \) i.e. that start in same order)

**i**th step \( (i = 1 \ldots m \text{ where } |S| = m) \)

Suppose \( X_i \) in posn \( j \) in OPT's list \( L_i^* \)

" " " " K " MTF's " L_{i-1}

\[ \text{OPT} \quad \begin{array}{c}
X_1 \\
\vdots \\
X_i \\
\vdots \\
x_{|S|}
\end{array} \]

\[ \text{MTF} \quad \begin{array}{c}
X_1 \\
\vdots \\
X_{i-1} \\
\vdots \\
x_{|S|}
\end{array} \]

\[ \text{OPT} \quad \begin{array}{cc}
X_r & X_3 \\
\vdots & \vdots \\
x_2 & x_3
\end{array} \]

\[ \text{MTF} \quad \begin{array}{cc}
x_2 & x_3 \\
\vdots & \vdots \\
x_3 & x_r
\end{array} \]

\[ \Phi_i = \text{# inversions in these lists} \]

i.e. \( \text{# pairs } X_i X_j \) in different relative orders

\[ 0 \leq \Phi_i \leq \binom{n}{2} \]

\[ \Phi_0 = 0 \]
\[ C_i^\text{MTF} = C_i^{\text{MTF}} + \Phi_i - \Phi_{i-1} \]

This is $K$ \[ \text{what is } \Delta\Phi? \]

**Main observation:**

Since $\text{OPT}$ & $\text{MTF}$ in $A$, only $X_i$ moves relative to other list elements \[ \Rightarrow \text{only pairs including } X_i \text{ affect } \Phi \]

**Two types of change:**

1) MTF moves $X_i$ to front
2) OPT moves $X_i$ forward by some (unknown) amount

**Type 1 Change**

MTF moves $X_i$ to front
OPT stays fixed

Say $v$ of these

Assume in $k$th location

Items before $X_i$ in MTF
Items before $X_i$ in MTF but after $X_i$ in OPT
Removed inversions

New inversions

$K - 1$ of these

$K - 1 - v$ of these

So focuses only on type I change
Before moving MTF:

OPT

| X_i |

MTF

| X_i |

blue: also before X_i in OPT

green: after X_i in OPT

When MTF moves, (i.e. X_i goes to front)
new inversions: K-1-V
removed inversions: V

\[
\text{Change} = K-1-V-V = K-1-2V
\]

Type 2 Change always \(< 0\)

Total change \(\leq K-1-2V\)

so \(C_i^{MTF} \leq K + K-1-2V = 2(k-V) - 1\)
What is \( k-v \)?

**Lemma** \( k-v \leq j \)

**Proof.** \( k-1-v \) "blue" els [ before \( x_i \) in \( \text{OPT} \) ]

all came before \( \text{OPT} \) in locn \( j \)

so \( k-1-v < j \)

\[ \Rightarrow k-v \leq j \]

so \( C^* \leq 2(k-v) - 1 \)

\[ \leq 2j - 1 \]

\[ \leq 2C^*_{\text{OPT}} - 1 \]

\[ C_{\text{HTE}}(s) = \sum_{i=1}^{m} \hat{C}_{i} \]

\[ \leq \sum_{i=1}^{m} \hat{C}_{i} \]

\[ \leq 2 \sum_{i=1}^{m} C^*_{\text{OPT}} - m \]

\[ \leq 2C^*_{\text{OPT}}(s) \]

**but** \( j \) is cost of \( \text{OPT}'s \) search for \( x_i \)!

So, "knowing future" helps by factor of at most 2.

Similar results even if \( \text{OPT} \) more flexible.