CPSC 420 Lecture 24: Today's announcements:

- HW4 (coming soon) due Mar 30, 23:59
- Reading: `https://student.cs.uwaterloo.ca/~cs466/Old_courses/F10/online_list.pdf` [by López-Ortiz]
  `https://courses.csail.mit.edu/6.897/spring03/scribe_notes/L5/lecture5.pdf` [by Demaine]
  `https://courses.csail.mit.edu/6.897/spring03/scribe_notes/L6/lecture6.pdf` [by Demaine]

Today's Plan

- Online Algorithms
  - Hiring problem ✓
  - Page replacement ✓
  - List Update
  - Experts

List Update

Suppose we use a linear structure (like an array, linked-list, pile of paper, etc.) to store a set of data (no duplicates).

Operations:

insert(x)	scan list from front (to verify that x is not there) and add x to the end (short form: $+x$)
find(x)	scan list from front to find x (short form: $?x$)
delete(x)	like **find** but remove x (short form: $-x$)

The **cost** of each operation is the position of x in the list.

List update:

free	to move x anywhere closer to front after find(x) (not on insert(x))
pay 1	per swap of any two adjacent list items

List Update Example

Oper	$+a$	$+b$	$+c$		$?c$	$?b$	$?b$	$?c$	
list	a	ab	abc		abc	abc	abc	abc	
cost	1	2	3		3	2	2	3	$= 16$
list	a	ab	abc		abc	cab	bca	bca	
cost	1	2	3		3	3	1	2	$= 15$
list	a	ab	abc	bac	bac	bca	bca	bca	
cost	1	2	3	1	3	1	1	2	$= 13$
list	a	ab	abc		abc	acb	abc	bac	
cost	1	2	3		3	3	2	3	$= 17$

Frequency Count, Move-to-front, Transpose

Oper	$+a$	$+b$	$+c$		$?c$	$?b$	$?b$	$?c$	
list	a	ab	abc		abc	abc	abc	abc	
cost	1	2	3		3	2	2	3	$= 16$
list	a	ab	abc		abc	cab	bca	bca	
cost	1	2	3		3	3	1	2	$= 15$
list	a	ab	abc	bac	bac	bca	bca	bca	
cost	1	2	3	1	3	1	1	2	$= 13$
list	a	ab	abc		abc	acb	abc	bac	
cost	1	2	3		3	3	2	3	$= 17$

Frequency Count Order items by #finds so far (decreasing).

Move-to-front On every find(x), move x to the front.

Transpose On every find(x), swap x one closer to the front.

Frequency Count, Move-to-front, Transpose

Oper	$+a$	$+b$	$+c$		$?c$	$?b$	$?b$	$?c$	
list	a	ab	abc		abc	abc	abc	abc	
cost	1	2	3		3	2	2	3	$= 16$
MTF	a	ab	abc		abc	cab	bca	bca	
FC	1	2	3		3	3	1	2	$= 15$
list	a	ab	abc	bac	bac	bca	bca	bca	
cost	1	2	3	1	3	1	1	2	$= 13$
TR	a	ab	abc		abc	acb	abc	bac	
cost	1	2	3		3	3	2	3	$= 17$

Frequency Count Order items by #finds so far (decreasing).

Move-to-front On every find(x), move x to the front.

Transpose On every find(x), swap x one closer to the front.

FC, MTF, TR versus Static OPT

Suppose all n inserts occur at the beginning and there are no deletes.

Static OPT (SOPT) orders items by total number of finds:

$$f_1 \geq f_2 \geq \cdots \geq f_n \text{ where } f_i = \text{ number of find}(x_i) \text{ ops.}$$

Transpose is bad. Suppose the m operations are

$$+x_3 \ +x_4 \ +x_5 \ldots + x_n \ +x_1 \ +x_2 \ \underbrace{?x_1 \ ?x_2 \ ?x_1 \ ?x_2 \ldots}_{m-n \text{ operations}}$$

initial cost ~~to~~ insert items

so the list starts as $x_3 x_4 \ldots x_n x_1 x_2$.

$\text{cost(TR)} = \left(\sum_{i=1}^{n} i\right) + (m-n)n \sim mn$ for large m.

$\text{cost(SOPT)} = \left(\sum_{i=1}^{n} i\right) + 1.5m \sim 1.5m$ for large m. □

1/2 (1+2) cost.

2n?

1.5 (m−n) + 2n

FC, MTF, TR versus Static OPT

cost of find(j)

Suppose all n inserts occur at the beginning and there are no deletes.

Static OPT (SOPT) orders items by total number of finds:

$$f_1 \geq f_2 \geq \cdots \geq f_n \text{ where } f_i = \text{ number of find}(x_i) \text{ ops.}$$

Move-to-front is good[1]

Let $\text{cost}_A(i,j)$ be the number of times algorithm A sees item x_i before finding item x_j.

$$\text{cost}(A) = \sum_{i<j} \text{cost}_A(i,j) + \text{cost}_A(j,i)$$

For $i < j$, $\text{cost}_{\text{SOPT}}(i,j) = f_j$ and $\text{cost}_{\text{SOPT}}(j,i) = 0$.

find(x_j)

SOPT

i precedes j

[1] Similarly, so is Frequency Count.

FC, MTF, TR versus Static OPT

When $f_i \geq f_j$, the worst case for MTF is:

$$\underbrace{?x_j\ ?x_i\ ?x_j\ ?x_i \ldots ?x_j\ ?x_i}_{2f_j \text{ find ops}}\ \underbrace{?x_i\ ?x_i \ldots ?x_i}_{f_i - f_j \text{ find ops}}$$

$\emptyset$

So $\text{cost}_{\text{MTF}}(i,j) \leq f_j$ and $\text{cost}_{\text{MTF}}(j,i) \leq f_j$.

Thus,

$$\begin{aligned} \text{cost}_{\text{MTF}}(i,j) + \text{cost}_{\text{MTF}}(j,i) &\leq 2f_j \\ &\leq 2(\text{cost}_{\text{SOPT}}(i,j) + \text{cost}_{\text{SOPT}}(j,i)) \end{aligned}$$

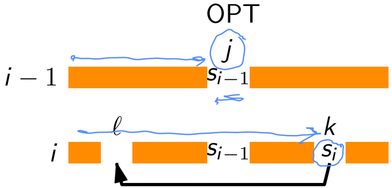
and so

$$\text{cost(MTF)} \leq 2\text{cost(SOPT)} \qquad \square$$

MTF versus Dynamic OPT

Lemma: OPT only needs paid swaps.

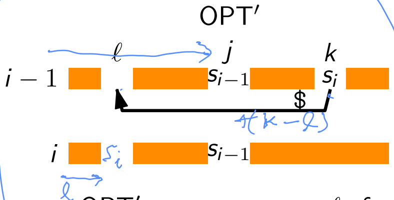
Proof: Let $s = s_1 s_2 \ldots s_m$ be a sequence of items to find.



OPT free-moves s_i to ℓ after ith access.

cost $= j + k$

uses free move



OPT′ pay-moves s_i to ℓ after $i-1$th access.

cost $= (j + (k - \ell)) + \ell$ □

MTF versus Dynamic OPT

Theorem: For any sequence $s = s_1 s_2 \dots s_m$ of items to find,

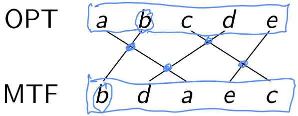
$$\text{cost(MTF)} \leq 2\text{cost(OPT)}$$

where cost(A) is the cost (including paid swaps) of algorithm A on sequence s.

Proof: Let $\phi(i)$ be the number of inversions between the list orders of MTF and OPT after find(s_i).

intersections in bipartite matching between two orders

$\phi(0) = 0$



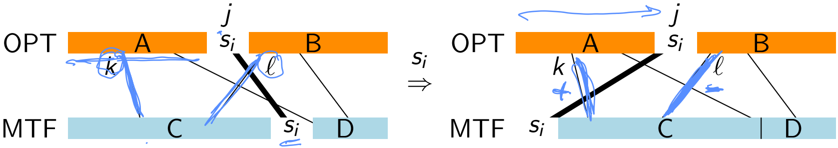
$\phi(i) = 4$

Let $c_i(A)$ be the cost of A on find(s_i).

We first show that $\underbrace{c_i(\text{MTF})} + \underbrace{\phi(i) - \phi(i-1)}_{\Delta} \leq \underbrace{2c_i(\text{OPT})} - 1$.

MTF versus Dynamic OPT

$|A| \geq k$



k matches from A to C
ℓ matches from B to C

$k+\ell+1$

$\phi(i) - \phi(i-1) = k - \ell$

1. $c_i(\text{MTF}) = k + \ell + 1$
2. $\phi(i) - \phi(i-1) = k - \ell$
3. $c_i(\text{OPT}) = j + P(i) \geq k + 1 + P(i)$ where $P(i)$ is #paid swaps by OPT on ith find.
4. Each of $P(i)$ paid swaps increases $\phi(i)$ by ≤ 1.

So

$$c_i(\text{MTF}) + [\phi(i) - \phi(i-1)] \leq k + \ell + 1 + [k - \ell + P(i)]$$
$$= 2k + 1 + P(i) \leq 2c_i(\text{OPT}) - 1$$

MTF versus Dynamic OPT

Sum over all i to get: $\text{cost}(\text{MTF}) + \sum_{i=1}^{m}[\phi(i) - \phi(i-1)] =$

$$\begin{aligned}
\sum_{i=1}^{m}(c_i(\text{MTF}) + [\phi(i) - \phi(i-1)]) &\leq \sum_{i=1}^{m}(2c_i(\text{OPT}) - 1) \\
&= 2\text{cost}(\text{OPT}) - m \\
&\leq 2\text{cost}(\text{OPT})
\end{aligned}$$

Since $\sum_{i=1}^{m}[\phi(i) - \phi(i-1)] = \phi(m) - \phi(0) \geq 0$, we're done. $\square$