List update problem

Linked list

\[ a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots \rightarrow a_n \]

Operation: \text{search}(x) - search linked list from left, until some element \( a_k = x \) is found

Cost of search: \( k \)

In case \( x \) not found, cost is \( n \)

(Add mode it dynamic: insert/delete - required for now)

After searching, we can re-arrange list:

- Free rearrangement: move \( a_k = x \) anywhere towards left
- Paid rearrangement: transpose two neighbors in list
  (anywhere in list), \( \text{cost} = 1 \)

\[ \ldots \rightarrow x \rightarrow y \rightarrow \ldots \]

\[ \ldots \rightarrow y \rightarrow x \rightarrow \ldots \]

(other cost models possible)

Problem: future needs are not known
Initial state $a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n$

Search sequence: $R = r_1, \ldots, r_m$

$\text{OPT}(R) =$ optimal cost of securing $R$ (and search cost and paid $pi$-arrays end)

$\text{Alg}(R) =$ cost of securing $R$ by alg. Alg

---

Example

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f$

- Search (d) Cost: 4
- Search (f) Cost: 6
- Search (e) Cost: 6
- Search (c) Cost: 5+1
- Search (b) Cost: 2

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f$

$\text{Alg}^f(R) = 16$

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f$

$\text{Alg}(R) = 24$
1. **Transpose**: After accessing an item \( k \), bring it one step closer to the start. (free re-arrangement)

2. **Move-to-front (MTF)**: After accessing an item \( k \), bring it to the front. (free re-arrangement)

3. **Frequency count (FC)**: Keep track of the accesses for each item. After accessing item \( k \), re-arrange list so that it is sorted decreasingly by # accesses. (free re-arrangement)

\( (1,2 \text{ are } \text{"memoryless"}) \)

\( (3 \text{ needs some bookkeeping}) \)

**Theorem 1.** MTF is 2-competitive.

**Theorem 2.** Transpose and FC are not competitive.

**Proof of Theorem 2 (Transpose)**

\[ R = (a_k, a_k, \ldots, a_k) \]

**Transpose** \( R = \begin{cases} k+1 & \text{if } k+1 \leq 2m \\ k+1 & \text{if } k+1 \leq 2m \\ \vdots & \text{if } k+1 \leq 2m \\ k+1 & \text{if } k+1 \leq 2m \end{cases} \)

\[ \sum_{i=1}^{k} a_i \rightarrow a_{k-1} \rightarrow a_k \rightarrow a_{k+1} \rightarrow \cdots \]

\[ \text{Transpose (R)} = \frac{2m(k+1)}{3m+2k+2} \]

\[ \text{OPT (R)} = \frac{k+1}{3(m-1)} \]

\[ 3m-3 + 2k+2 \]
Theorem 3. No deterministic online list update algorithm is $(2-\epsilon)$-competitive.

Theorem 4. Best randomized online is $\bar{c}$-competitive, for $c \in [1.5, 1.6]$

Theorem 1. MTF is 2-competitive

Proof:

$$R = r_1, \ldots, r_m$$

(want to prove $MTF(R) \leq 2 \cdot OPT(R)$)

Assume both MTF and OPT start from same initial state.

Run MTF and OPT side-by-side.

Potential set $\Phi = \text{# inverted pairs between MTF and OPT}$.

Initially $\Phi_0 = 0$
for each $t$, denote $MTF_t$ and $OPT_t$ cost of solving $Search(q_t)$ by $MTF$ by $OPT$

$\phi_t$: potential after solving $R_t$.

Claim: $MTF_t + \phi_t - \phi_{t-1} \leq 2 \cdot OPT_t - 1$ for $t = 1, ..., m$

Suppose Claim true. Then:

$\sum_{t=1}^{m} MTF_t + \phi_m - \phi_0 \leq 2 \cdot \sum_{t=1}^{m} OPT_t - m$

$MTF(R) + \phi_m \leq 2 \cdot OPT(R) - m$

$MTF(R) \leq 2 \cdot OPT(R) - m - \phi_m \leq 2 \cdot OPT(R)$

(Theorem 1)

Proof of Claim

look at access $R_t$

$\Gamma_t$ (MTF)

$\Gamma_t$ (OPT)

$k + l$ items before $\Gamma_t$ in MTF
\( k \) items before \( x_t \) in MTF that are also before \( x_t \) in OPT

\( \geq \) items before \( x_t \) in MTF that are after \( x_t \) in OPT

\( \text{MTF}_t = k + R + 1 \)

\( \text{OPT}_t = k + 1 \)

\( k \) inversions created \( \phi_t - \phi_{t-1} = k - R \)

\( \ell \) inversions removed

\[
\left( \text{MTF}_t + \phi_t - \phi_{t-1} \leq 2 \cdot \text{OPT}_t - 1 \right)
\]

\[ k + R + 1 + k - 1 \leq 2k + 1 = 2(k+1) - 1 \leq 2 \cdot \text{OPT}_t - 1 \]

We looked so far at patches due to MTF, next we look at OPT operations for verifying \( x_t \)

| \( x_t \rightarrow \bullet \rightarrow \bullet \rightarrow \cdots \) | (MTF) |
| \( \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \phi \rightarrow \phi \rightarrow \text{OPT} \) |

OPT can move \( x_t \) to the left for free

\( \rightarrow \text{this can only decrease } \phi \)

(bec. \( x_t \) already tells that \( x_t \) in MTF)

Claimed \( \phi \) increases remains true

OPT can do paid transpose operation

\( \rightarrow \) each case \( 1 \rightarrow \text{OPT} \)

\( \rightarrow \) may increase \( \phi \) by 1

Claimed \( \phi \) remains true.
To be, or not to be, that is the question:
Whether it is nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take Arms against a Sea of troubles,
And by opposing end them: to die, to sleep;

Example: $a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots \rightarrow a_n$

$R = a_m, a_{m-1}, \ldots, a_1$

$\text{OPT}(R) \leq m + m-1 + \ldots + 1 = \frac{m(m+1)}{2} \sim \frac{m^2}{2}$

$\text{NTF}(R) = \frac{m + m + \ldots + m}{m^2}$

$\frac{\text{NTF}(R)}{\text{OPT}(R)} = 2 - o(1)$

Application of list update problem (NTF algorithm)

$\rightarrow$ Data compression

Huffman-coding

(prefix-free)

pronounce frequencies

build optimal tree

use that for encoding/decoding

Compress using list update

Idea: build a ordered list of words, encode word by index in list.

Interpret tests as log2 t bits

→ frequent words should have small integer codes.

→ maintain list using NTF list update.
To be, or not to be, that is the question:
Whether it is nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take Arms against a Sea of troubles,
And by opposing end them: to die, to sleep;

It can be shown that code length never much worse than Huffman
Sometimes it is better.

**Advantages:**
1. Single pass
2. Simple, no need to store frequencies
3. No need to transmit treemap/dictionary
4. Adaptive
5. Fast, practical

**Disadvantages:**
1. Encoder/decoder used to synchronize carefully
   → sensitive to errors

"Self-adapting lists"