List update problem

linked list

$$a_1 \to a_2 \to a_3 \to \dots \to a_n$$

↑
start

operation: 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

(could make it dynamic: insert/delete - omitted for now)

After searching, we can re-arrange list:

- free rearrangement: move $\overset{x}{\overset{\shortparallel}{a_k}}$ anywhere towards left
- paid rearrangement: transpose two neighbors in list, (anywhere in list), cost = 1.

$$\dots \to x \to y \to \dots$$
$$\dots \to y \to x \to \dots$$

such as to prepare for future searches

Problem: future searches are not known

(other cost models possible)

Initial state

$\underline{a_1} \to \underline{a_2} \to \dots \to \underline{a_n}$

Note: finding OPT(R) is NP-complete problem, unlike for paging.

search sequence: $R = r_1, \dots, r_m$

$OPT(R)$ = optimal cost of serving R (incl. search cost and paid re-arrangements)

$Alg(R)$ = cost of serving R by alg. Alg

Example

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

search (d) cost: 4

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

search (f) cost: 6

search (f) cost: 6

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

search (c) cost: 5 + 1 transpose (a, b)

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

search (f) cost: 2

$R = d, f, f, c, f$

$Alg(R) = 24$

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

search (d) cost: 4

search (f) cost: 6

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

search (f) cost: 1

search (c) cost: 4

search (f) cost: 1

$Alg'(R) = 16$

Algorithms

1. Transpose: After accessing an item x, bring it one step closer to the start.
(free re-arrangement)

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

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

(1,2 are "memoryless")
(3 needs some bookkeeping)

Theorem 1. MTF is 2-competitive

Theorem 2. Transpose and FC are not competitive.

Proof of Thm2 (Transpose)

$a_1 \to a_2 \to \dots \to a_k \to a_{k+1} \to \dots \to a_n$

$R = (a_{k+1}, a_k)^m$

$a_1 \to \dots \to a_{k-1} \to a_{k+1} \to a_k \to \dots$

$\text{Transpose}(R) = \left.\begin{matrix} k+1+ \\ k+1 \\ k+1 \\ \dots \end{matrix}\right\} 2m$

$= 2m(k+1)$

$\frac{\text{Transpose}(R)}{\text{OPT}(R)} \geq \frac{2m(k+1)}{3m+2k-1}$

$a_k \to a_{k+1} \to \dots$

$\text{OPT}(R) = \begin{matrix} k+1+ \\ k+1+ \\ 3\cdot(m-1) \end{matrix}$

$= 3m-3+2k+2$

$$\frac{\text{Transpose}(R)}{\text{OPT}(R)} \geq \frac{2m(k+1)}{3m+2k-1} \geq \frac{2(k+1)}{3+2\frac{k}{m}-\frac{1}{m}} \geq \dots \geq \frac{2}{5}K$$

(arbitrarily large)

assume $m > k$

Theorem 3. No deterministic online list update algorithm is $(2-\varepsilon)$-competitive.

Theorem 4. Best <u>randomized</u> online ———— is c-competitive, for $c \in [\underline{1.5, 1.6}]$

Theorem 1. MTF is 2-competitive

Proof

$R = r_1, \dots, r_m$ (want to prove $\text{MTF}(R) \leq 2 \cdot \text{OPT}(R)$)

Assume both MTF and OPT start from same initial state

Run MTF and OPT side-by-side.

Potential fct Φ = # <u>inverted pairs</u> between MTF and OPT.

$\dots \to x \to \dots \to y \to \dots$ x before y in MTF

$\dots \to y \to \dots \to x \to \dots$ y before x in OPT

Initially $\Phi_0 = 0$

For request r_t denote $\underbrace{MTF_t}_{\text{by MTF}}$ and $\underbrace{OPT_t}_{\text{by OPT}}$ cost of serving search(r_t)

Φ_t : potential after serving r_t.

<u>Claim</u>

$$\underbrace{MTF_t + \phi_t - \phi_{t-1}}_{\text{increase in pot.}} \leq 2 \cdot OPT_t - 1 \qquad \forall\, t = 1, \dots, m$$

Suppose Claim true. Then:

$$\underbrace{\sum_{t=1}^{m} MTF_t}_{\shortparallel} + \phi_m - \underbrace{\phi_0}_{=0} \leq 2 \cdot \sum_{t=1}^{m} OPT_t - m$$

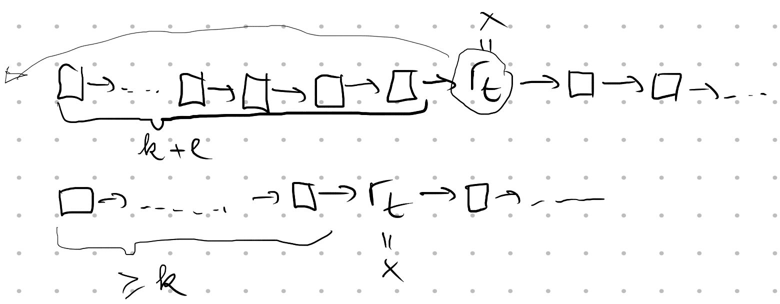
$$MTF(R) + \phi_m \leq 2 \cdot OPT(R) - m$$

$$\underline{MTF(R)} \leq 2 \cdot OPT(R) \underline{- m} - \underbrace{\phi_m}_{\geq 0} \leq \underline{2 \cdot OPT(R)}$$

(Theorem 1.)

<u>Proof of Claim</u>

look at accessing r_t



(MTF)

(OPT)

$k+l$ items before r_t in MTF

k items before r_t in MTF that are also before r_t in OPT

ℓ items before r_t in MTF that are after r_t in OPT

$MTF_t = k + \ell + 1$

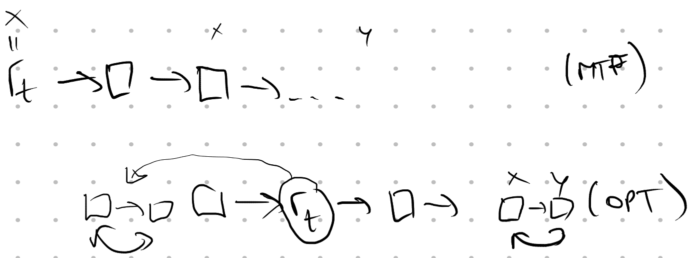
$OPT_t \geq k + 1$

k inversions created
ℓ inversions removed

$$\phi_t - \phi_{t-1} = k - \ell$$

$$\left(MTF_t + \phi_t - \phi_{t-1} \leq 2 \cdot OPT_t - 1 \right)$$

$$k + \cancel{\ell} + 1 + k - \cancel{\ell} \leq 2k + 1 = 2(k+1) - 1 \leq 2\, OPT_t - 1$$



(We looked so far at pot. change due to MTF, next we look at OPT operations for serving r_t)

OPT can move r_t to the left for free
→ this can only decrease ϕ
(bec. r_t already leftmost in MTF)
Claimed ineq. remains true

OPT can do paid transpose operations
→ each costs 1 to OPT_t
→ may increase ϕ by 1
Claimed ineq. remains true.

$\boxtimes$

E.g. $a_1 \to a_2 \to a_3 \to \dots \to a_n$

$R = a_n, a_{n-1}, \dots, a_1$

$$OPT(R) \leq n + n-1 + \dots + 1 = \frac{n(n+1)}{2} \sim \frac{n^2}{2}$$

$$MTF(R) = \underbrace{n + n + \dots + n}_{n^2}$$

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

Thm 1 can not be improved.

Application of list update problem (MTF algorithm)

→ Data compression

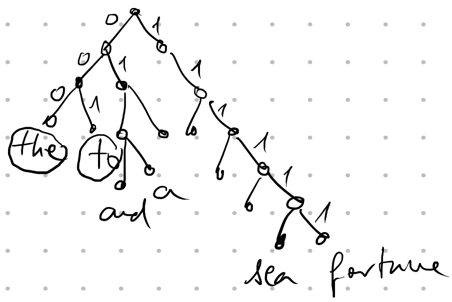
Text:

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;

...

001 ....

Huffman-coding (prefix-free)



the → 000
to → 001
...
fortune → 111111

- compute frequencies
- build optimal tree
use that for encoding/decoding

Compression using list update

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

integers

integer t takes $\sim \log_2 t$ bits

→ frequent words should have small integer codes.

→ maintain list using MTF 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;

(ignore punctuation, caps, etc.)

encoding:

list state as we process each word

to

be → to

or → be → to

not → or → be → to

to → not → or → be

be → to → not → or

that → be → to → not → or

output:

1 to 2 be 3 or 4 not 4 4 5 that

decoding:

to

be → to

or → be → to

not → or → be → to

to → not → or → be

be → to → not → or

Output:

to be or not to be

It can be shown that total code length never much worse than Huffman (see refs)
Sometimes it is better.

Advantages:

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

Disadvantage

1. Encoder/decoder need to synchronise carefully
→ sensitive to errors

"Self-adjusting lists"