1 Introduction

In this lecture we will deal with updating a linked list of size $n$ according to a sequence of requests of size $t$. Although, in most cases today, linked lists aren’t widely used as they have been replace by hash tables and other data structures.

```
$X_1 \rightarrow X_2 \rightarrow X_3$
```

Example: The supermarket holds a list of all its groceries and an item is accessed every time it passed through the registry.

2 Problem description

**definition**: The cost of accessing an item is equal to its current position ($i$) in the list. We will see two models for the cost

- Accessing the item cost $i$
- Accessing the item cost $i - 1$

When dealing with objects very far in the list, the cost is very high. So the question is whether or not should we move the item to a new location to improve future accesses and of course what should the new location be.

**Objective**: Reaching the lowest cost for a sequence of requests of size $t$.

$\sigma = X_1, X_2, ...$

$Cost = \sum_{j=1}^{t} i_{X_j}$

Where $i_{X_j}$ is the position of the $X_j$ item in the sequence of requests.
2.1 Formalization

- **access(X)** - accessing item X in the list. The algorithm is allowed to move the item to every location which precede it, free of cost (for example: remember the pointer of the desire location).

- **exchange(j,j-1)** - replace positions between the j and j - 1 items. This action cost is 1.

3 Possible solutions

- **Move to front (MF)** After each access move the item to the head of the list
- **Transpose(T)** After each access move the item one spot toward the head of the list
- **Frequency Count(FC)** Count each time the item is access and sort list in decreasing order

**Example for FC algorithm:** Assuming after few accesses the list frequency counters are sort as follow:

\[X_1 - 7, X_2 - 6, X_3 - 5, X_4 - 5, X_5 - 4, X_6 - 2\]

If the next access is to the second element so the new state is

\[X_1 - 7, X_2 - 7, X_3 - 5, X_4 - 5, X_5 - 4, X_6 - 2\]

Alternatively, if the next access is to the fourth element so its counter will be change to 6 and it will be moved in front of the third element so the new state is:

\[X_1 - 7, X_2 - 6, X_4 - 6, X_3 - 5, X_5 - 4, X_6 - 2\]

Of course, moving the element has no cost as describe above.

3.1 Decreasing Probability

Decreasing Probability is statistical model, assuming a distribution for which each element has probability (e.g. \(X_1 \to 0.2, X_2 \to 0.3, X_3 \to 0.1, X_4 \to 0.2, X_5 \to 0.2\)) and each access is independent to other accesses and the distribution is the same for all accesses, it’s best to sort the list in Decreasing probability and in the expectation this will be the optimal algorithm. In the above example the list will be \(X_2, X_1, X_4, X_5, X_3\).

In the case, the distribution is known - if the probability for item \(i\) is \(P_i ((\sum_{i=1}^{n} P_i = 1)\) then the expected cost is \(\sum_{i=1}^{n} t_i P_i\), where is the sequence length \((tP_i\) is the number of time item \(i\) was accessed and \(i\) is the cost for each such step).

When the distribution is unknown and independent, it can be proven:

\[
\frac{FC(\sigma)}{DP(\sigma)} \to 1
\]

\[2 - 2\]
when $t$, $\sigma$’s length approached $\infty$. In the expectation the following claim is correct (stated without proof):

$$E(FC(\sigma)) \leq E(T(\sigma)) \leq E(MF(\sigma)) \leq 2E(FC(\sigma))$$

4 Analysis

let’s review our solutions:

Starting with the Transpose (T) algorithm: T will be very inefficient for a sequence which hold only two items:

$$X_1, X_2, X_1, X_2, X_1, X_2, \ldots$$

The algorithm will only switch between the two items, in case the items are in the end of list: The cost of each request is $n$ and there are $t$ steps so the total cost will be $n \times t$. The optimal algorithm would move both element to the beginning of the list and the average cost for step is 1.5 so the total cost for OPT will be $2n + 1.5t$.

$$\frac{T(\sigma)}{OPT(\sigma)} = \frac{n \times t}{2n + 1.5t} = \Omega(n)$$

So Transpose is worst than the OPT for this sequence in factor of $n$.

An example for a sequence which is very inefficient for Frequency Count (FC) algorithm is:

$$\begin{align*}
X_1 & \text{ is accessed } 2n \text{ afterward} \\
X_2 & \text{ is accessed } 2n - 1 \text{ afterward} \\
X_3 & \text{ is accessed } 2n - 2 \text{ afterward} \\
& \ldots \\
X_n & \text{ is accessed } n + 1 \text{ afterward}
\end{align*}$$

The cost for $FC(\sigma)$:

$$FC(\sigma) = \Theta(n(1 + 2 + 3 + \ldots + n)) = \Theta(n^3)$$

An optimal algorithm will move the accessed item to the beginning each time so the cost will be (which is simply to MF):

$$OPT(\sigma) = MF(\sigma) = \Theta(n \times n + t) = \Theta(n^2)$$

Since we have $n$ items and reaching the item for the first time is at most $n$. So Frequency Count is worst than the OPT for this sequence in factor of $\Theta(n)$. 

2 - 3
4.1 MF is 2-Competitive

**Theorem 4.1. MF is 2-Competitive**

We will prove that in the model where the cost for the $i-th$ item is $i-1$. It is easy to see that proving in the $i-1$-model is harder since in $i$-model we should only add $t$ to both the nominator and the denominator, which will only decrease the ratio.

$$
\text{"i} - 1\text{"}
\quad \frac{A(\sigma)}{OPT(\sigma)} \leq 2
\\downarrow
\quad \frac{A(\sigma)}{OPT(\sigma)} \leq 2
$$

We will use the following Potential function $\Phi$ which is equal to the number of item pairs between the OPT list to MF list, which are in reverse order.

**Example:** In case we have the following list:

OPT list - 1, 2, 3, 4, 5
MF list - 2, 5, 3, 1, 4

We have $\binom{5}{2}$ pairs.

We can see which pairs are reversed between the two lists:

$(1, 2) \rightarrow 1, (1, 3) \rightarrow 1, (1, 4) \rightarrow 0, (1, 5) \rightarrow 1$
$(2, 3) \rightarrow 0, (2, 4) \rightarrow 0, (2, 5) \rightarrow 0$
$(3, 4) \rightarrow 0, (3, 5) \rightarrow 1$
$(4, 5) \rightarrow 1$

So in this case $\Phi = 5$. Let us formalize it:

$$
\Phi = \#\{(x, y)\mid x \text{ appear before } y \text{ in one list and after } y \text{ in the other list}\}
$$

We shall prove that for every sequence $\sigma$ applies $MF(\sigma) + \Phi(\sigma) \leq 2OPT(\sigma)$. Since $\Phi \geq 0$ this will implies that MF is 2-competitive. The equation is obviously correct for the empty sequence.

$OPT(\sigma) = 0$

$MF(\sigma) = 0$

$\Phi(\sigma) = 0$

We shall prove that in every step the following equation holds:

$$
\Delta MF(\sigma) + \Delta \Phi \leq 2\Delta OPT(\sigma)
$$
Initially: $0 + 0 \leq 2 \cdot 0$

\[
\begin{align*}
\Delta_1 MF(\sigma) + \Delta_1 \Phi & \leq 2\Delta_1 OPT(\sigma) \\
+ \Delta_2 MF(\sigma) + \Delta_2 \Phi & \leq 2\Delta_2 OPT(\sigma) \\
\vdots \Delta_n MF(\sigma) + \Delta_n \Phi & \leq 2\Delta_n OPT(\sigma)
\end{align*}
\]

Let’s start with the harder case where MF cost is much larger than OPT (the same process apply in the opposite case):

Let’s assume $X$ is the $a + 1$ item in OPT list and the $b + 1$ item in MF list.

For now, let’s assume that OPT doesn’t move the item in this step.

\[
\begin{align*}
\Delta OPT(\sigma) &= a \\
\Delta MF(\sigma) &= b
\end{align*}
\]

Every pair which holds $y, z \neq x$ won’t affect the Potential function since only pairs which hold $X$ are affected. The Potential function can only increase at most by $a$ since every item which is after $X$ in OPT list will be in same order in MF list. The function also must decrease by at least $b - a$ pairs, which were in reverse order.

\[
\begin{align*}
\Delta OPT(\sigma) &= a \\
\Delta \Phi & \leq a - (b - a) = 2a - b \\
\Downarrow \ 
\Delta MF(\sigma) + \Delta(\Phi) & \leq b + 2a - b = 2a \leq 2OPT(\sigma)
\end{align*}
\]

In case OPT does move items, we should review two case:

- If OPT moves $X$ toward the head of the list, we shall receive an improvement in the potential function since the max growth will decrease (before it was $a$). So if $\Delta MF(\sigma) + \Delta(\Phi) \leq b + 2a - b = 2a \leq 2OPT(\sigma)$ before the movement, this will definitely be valid after the movement.

- If OPT exchanges other item on the way to $X_j$ then the potential function might increase by one, but then OPT also pays one since exchange cost is 1.

\[
\Delta \Phi = 1 \leq 2 \cdot 1 = 2OPT \leq 2 - 5
\]
Let’s see what will happen in case we add two new actions:

- *Insert*(X) - add a new item to the head of the list
- *Delete*(X) - delete item X from the list

Claim: the new actions doesn’t change the competitive rate between MF and OPT.

It is easy to see insert has the same cost for MF and OPT. Also, although there are now n new pairs to check in the potential function, they aren’t affected since they look the same in both lists.

\[ \Delta MF(\sigma) = \Delta OPT(\sigma) = 1, 0 \]

and

\[ \Delta \Phi(\sigma) = 0 \]

*Delete*(X) = *access*(X) + *MoveToFront* + *delete*(X)

*Delete*(X) is the same as accessing X, moving it to the head of the list and deleting the first item. Since deleting the first item has no affect on the potential function (all the pairs which disappear were in the same order in both list). The cost is the same as *access*(X) which we have already prove before.

### 4.2 Lower bound for list-update problem

**Theorem 4.2.** Every Deterministic Algorithm for list update problem is at least \(2 - \epsilon\)-Competitive for every \(\epsilon > 0\) in \(i-1\)-model \((in i-model the bound is 2(1 - \frac{1}{n+1})\)

**Proof.** Given Algorithm A, we shall build a sequence \(\sigma\) which always requests the last item in its current list (This is called **H.T.W.T.A.N** - hit them where they are not).

In each step we should pay \(\Delta A(\sigma) = n - 1\), so the total cost for A is \(A(\sigma) = t(n - 1)\). We will compare it to a static OPT’ algorithm which set the list according to Decreasing Probability order, as follow: if \(P_0 \geq P_1 \geq P_2 \geq \ldots \geq P_{n-1}\) then our list be be ordered:

\[ X_0, X_1, \ldots, X_{n-1} \]

As we saw before, for each item the cost is \(t \cdot i \cdot P_i\), we will also have to pay for the sorting process which should cost as \(n^2\).

It is important to understand OPT’ is not the OPT algorithm but if A is at least 2-Competitive with OPT’, then it has to be at least 2-Competitive with OPT.

\[ OPT'(\sigma) = t \sum_{i=0}^{n-1} iP_i + n^2 \]

The sum of \(iP_i\) is maximum when \(P_0 = P_1 = P_2 = \ldots = P_{n-1} = \frac{1}{n}\).

\[ \frac{\sum_{i=0}^{n-1} iP_i}{n} = \frac{(0 + 1 + 2 + 3 + \ldots + n - 1)}{n} = \frac{n - 1}{2} \]

2 - 6
We can see why the claim is correct by observing this histogram.

When you level it, there are more accesses to items with higher cost. When adding all this together we get:

\[ OPT'(\sigma) \leq \frac{n-1}{2} * t + n^2 \]

So when \( t \) approached \( \infty \)

\[ \frac{A(\sigma)}{OPT(\sigma)} \geq \frac{n-1}{\frac{n-1}{2} + \frac{n^2}{t}} = \frac{2}{1 + \frac{2n^2}{(n-1)t}} \quad t \to \infty \to 2 \]

Next we show that allowing additive constant does not help

**Theorem 4.3:** Assume that we show a lower bound for any algorithm \( A \) such that

\[ A(\sigma) \geq c \cdot OPT(\sigma) \]

for a certain \( c \) and for a family of sequences of unbounded length such \( A(\sigma) \to \infty \) for \( \sigma \to \infty \). Then for any fixed \( \gamma \) and \( \epsilon > 0 \) it is impossible to have for every \( \sigma \)

\[ A(\sigma) \leq (c - \epsilon)OPT(\sigma) + \gamma \]

**Proof:** If it was possible then we would have

\[ c \cdot OPT(\sigma) \leq A(\sigma) \leq (c - \epsilon)OPT(\sigma) + \gamma \]

which implies that \( OPT(\sigma) \leq \gamma/\epsilon \) which implies that \( A(\sigma) \leq (c - \epsilon)\gamma/\epsilon + \gamma \) is bounded which is a contradiction.