SELF-ADJUSTING LINEAR LISTS
Analysis of the "Move to Front" Heuristic

The problem we shall study here (see [3]) is often called the *dictionary problem*: Maintain a set of items under an intermixed sequence of the following kinds of operations:

access(x): Locate item x in the set.

insert(x): Insert item x in the set.

delete(x): Delete item x from the set.

In discussing this problem, we shall use n to denote the maximum number of items ever in the set at one time and m to denote the total number of operations in the sequence of operations. We shall generally assume that the initial set is empty. A simple way to solve the dictionary problem is to represent the set by an unsorted list. To access an item, we scan the list from the front until locating the item. To insert an item, we scan the entire list to verify that the item is not already present and then insert it at the rear of the list. To delete an item, we scan the list from the front to find the item and then delete it. In addition to performing access, insert, and delete operations, we may occasionally want to rearrange the list by exchanging pairs of consecutive items. This can speed up later operations.

In this note we shall only consider algorithms that solve the dictionary problem in the manner described above. We define the cost of the various operations as follows: Accessing or deleting the i^{th} item on the list costs i units of time. Inserting a new item costs $l+1$ units of time, where l is the size of the list before the insertion. Immediately after an access or insertion of an item x, we allow x to be moved at no cost to any position closer to the front of the list; we call the exchanges used for this purpose *free exchanges*. Any other exchange , called *paid exchange*, costs one unit of time.

Our goal is to find a simple rule for updating the list (by performing exchanges) that will make the total cost of a sequence of operations as small as possible. Three rules have been extensively studied, under the rubric of *self-organizing linear lists*:

Move-to-Front (MF): After accessing or inserting an item, move it to the front of the list, without changing the relative order of the other items.

Transpose (T): After accessing or inserting an item, exchange it with the immediately preceding item.

Frequency Count (FC): Maintain a frequency count for each item, initially zero. Increase the count of an item by 1 whenever it is inserted or accessed; reduce its count to zero when it is deleted. Maintain the list so that the items are in nonincreasing order by frequency count.

Example 1: We give an example for the Move-to-Front rule.

$$a\ c\ d \xrightarrow[insert(b)]{} b\ a\ c\ d \xrightarrow[access(c)]{} c\ b\ a\ d \xrightarrow[delete(a)]{} c\ b\ d.$$

The cost of this sequence is 4 + 3 + 3 = 10 units of time. □

The Static Dictionary Problem

The *static dictionary problem* corresponds to the case where the list initially contains n items and only access operations are performed on the list (no delete or insert operations). In other words, the set of items on the list is fixed. To analyze the static dictionary problem the above rules are often compared against the optimum static rule, called *decreasing frequency (DF)*, which uses the fixed list with the items arranged in nonincreasing order by access frequency. Among algorithms that do no rearranging of the list, decreasing frequency minimizes the total access cost.

Note that the decreasing frequency rule assumes advance knowledge about the entire sequence of accesses. This kind of a problem is called *off-line*, that is the entire sequence of operations is known prior to any computation. (An *on-line* problem is one in which we have to complete the current operation before the next operation is known.)

Suppose the list contains the set of items $\{x_1, x_2, \cdots, x_n\}$, and s is a sequence of m access operations. Let k_i , $1 \le i \le n$, be the number of accesses to item x_i in sequence s. Note that $\sum_{i=1}^{n} k_i = m$. We may assume, without loss of generality, that $k_1 \ge k_2 \ge \cdots \ge k_n$.

In the decreasing frequency rule (DF) we organize the list in nonincreasing order of access frequency, that is in the order x_1 , x_2 , $\cdots$, x_n and never change this arrangement. The total cost of processing the sequence s on the list under the decreasing frequency rule is

$$C_{DF}(s) = \sum_{i=1}^{n} i \cdot k_i$$

In this section we will show that the total cost under the Move-to-Front rule is at most twice that much even though the MF-rule has no advance knowledge about access frequencies.

Theorem 1: *Let $C_{MF}(s)$ be the cost of processing sequence s of m access operations under the MF-rule starting with the initial list x_1 , x_2 , $\cdots$, x_n . Then $C_{MF}(s) \le 2\, C_{DF}(s) - m$.*

Proof: Let t_{ij} , $1 \le j \le k_i$, be the cost of the j^{th} access to item x_i under the move-to-front rule. We are going to derive an upper bound on

$$C_{MF}(s) - C_{DF}(s) = \sum_{i=1}^{n} \sum_{j=1}^{k_i} t_{ij} - \sum_{i=1}^{n} i \cdot k_i = \sum_{i=1}^{n} \left(\sum_{j=1}^{k_i} (t_{ij} - i) \right).$$

Consider an arbitrary pair i , j . There are $t_{ij} - 1$ items in front of x_i just prior to the j^{th} access to item x_i. There can be at most $i - 1$ items x_h, $h < i$, in front of x_i at that instant. We conclude that if $t_{ij} > i$, then there must have been at least $t_{ij} - i$ accesses to items x_h , $h > i$, between the $(j-1)^{th}$ and the j^{th} access to item x_i . (Why? Because each accessed

item is moved to the front of the list in the MF-rule. Also note that we consider the 0^{th} access to be the initial configuration.) Let us define

$A_{ij} = |\{ x_h \mid h > i$ and x_h is accessed between the $(j-1)^{th}$ and j^{th} access to $x_i \}|$.

Then, from the above discussion we conclude that

$$A_{ij} \geq t_{ij} - i$$

On the other hand, since the total number of accesses to items x_h, $h > i$, in sequence s is $k_{i+1} + k_{i+2} + \cdots + k_n$, we conclude

$$\sum_{j=1}^{k_i} A_{ij} \leq k_{i+1} + k_{i+2} + \cdots + k_n.$$

From the above two equations we conclude

$$\sum_{j=1}^{k_i} (t_{ij} - i) \leq k_{i+1} + \cdots + k_n$$

for all $i, 1 \leq i \leq n$, and therefore

$$C_{MF}(s) - C_{DF}(s) \leq \sum_{i=1}^{n} (k_{i+1} + \cdots + k_n)$$

$$= \sum_{i=1}^{n} (i-1) \cdot k_i$$

$$= \sum_{i=1}^{n} ik_i - \sum_{i=1}^{n} k_i$$

$$= C_{DF}(s) - m.$$

This completes the proof of the theorem. □

Amortized Analysis of the General Case

There is another way of interpreting the proof of Theorem 1 which leads to an account to be used shortly. We argued above that if $t_{ij} > i$ then there were at least $t_{ij} - i$ accesses $\cdots$. This suggests to charge the "excess cost" of $t_{ij} - i$ of the j^{th} access to x_i to the accesses to items x_h , $h > i$, between the $(j-1)^{th}$ and the j^{th} access to x_i, i.e. to charge the excess cost to those x_h , $h > i$, which are in front of x_i when item x_i is accessed for the j^{th} time. In other words, whenever we access x_h then we charge an extra $h-1$ time units to this access and use these time units to pay for the excess cost of accesses to items x_i , $i < h$. In this way, the amortized cost of accessing x_h is $2h-1$. We can generalize this idea in the amortized analysis as follows.

Let s be an arbitrary sequence of m access, insert, and delete operations starting with the empty list and let A be an arbitrary algorithm. Let $C_A^-(s)$ be the cost of algorithm A on sequence s not counting paid exchanges, let $X_A(s)$ be the number of paid exchanges, and let $F_A(s)$ be the number of free exchanges. Note that $X_{MF}(s) = X_T(s) = X_{FC}(s) = 0$

and that $F_A(s)$ for any algorithm A is at most $C_A^-(s) - m$ and the total cost of algorithm A, including paid exchanges, is $C_A(s) = C_A^-(s) + X_A(s)$.

Theorem 2: *For any algorithm A and any sequence s of m access, insert, and delete operations starting with the empty list*

$$C_{MF}(s) \le 2\,C_A^-(s) + X_A(s) - F_A(s) - m \le 2\,C_A(s) - m\,.$$

Proof: To obtain the theorem, we use as the potential function the number of inversions in *MF*'s list with respect to *A*'s list. For any two lists containing the same items, an *inversion* in one list with respect to the other is an unordered pair of items x, y, such that x occurs anywhere before y in one list and anywhere after y in the other. With this potential function we shall show that the amortized time for *MF* to access item i costs at most $2i-1$, the amortized time for *MF* to insert an item into a list of size i is at most $2(i+1)-1$, and the amortized time for *MF* to delete item i costs at most i, where we identify an item by its position in *A*'s list. Furthermore, the amortized time charged to *MF* when *A* does an exchange is –1 if the exchange is free and at most 1 if the exchange is paid.

The initial configuration has zero potential since the initial lists are empty, and the final configuration has a non-negative potential. Thus the actual cost to *MF* of a sequence of operations is bounded from above by the sum of the operations' amortized times. The sum of the amortized times is in turn bounded by the right-hand side of the inequality we wish to prove. (An access or an insert has amortized time $2c_A - 1$, where c_A is the actual cost of the operation to *A*; the amortized time of a deletion is $\le c_A \le 2c_A - 1$. The –1's, one per operation, sum to $-m$.)

It remains for us to bound the amortized times of the operations. Consider an access by *A* to item i (i.e. the i^{th} item in *A*'s list). Thus, *A*'s actual access cost to item i is $c_A = i$. Let k be the position of item i in *MF*'s list and let p be the number of items that precede item i in both lists. Thus, the number of items that precede i in *MF*'s list but succede i in *A*'s list is $k - 1 - p$. (See Figure 1.)

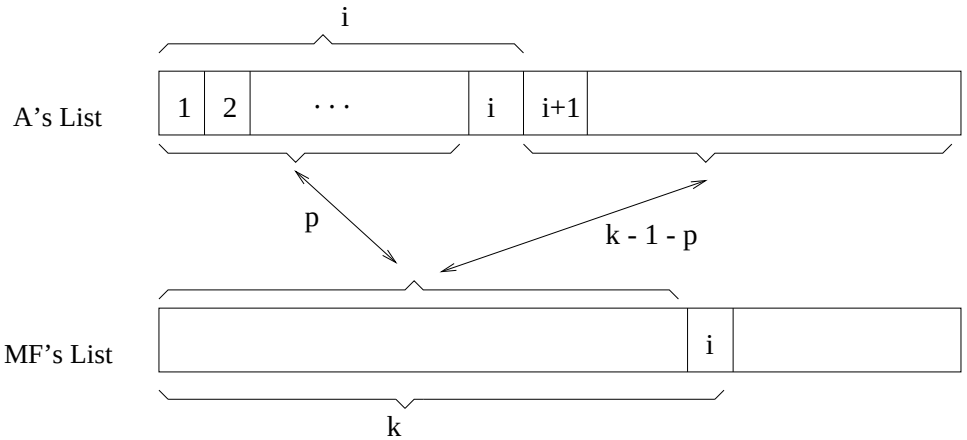

Figure 1. Arrangement of *A*'s and *MF*'s lists in the proof of Theorem 2.

The amortized cost of access to item i by *MF* is $\hat{c}_{MF} = c_{MF} + \Delta\Phi$. We see that $c_{MF} = k$. What is $\Delta\Phi$? Moving i to the front of *MF*'s list creates p inversions and destroys

$k-1-p$ other inversions. Thus, $\Delta\Phi = p-(k-1-p)$. Therefore, $\hat{c}_{MF} = k + p - (k-1-p) = 2p+1$. But $p \le i-1$ since only $i-1$ items precede i in A's list. Thus the amortized time to MF for the access is $\hat{c}_{MF} = 2p+1 \le 2i-1 = 2c_A - 1$.

The argument for an access applies virtually unchanged to an insertion or a deletion. In the case of a deletion no new inversions are created. There are $k-1-p$ inversions destroyed due to items preceding i in *MF*'s list but succeding it in A's list. Also, there are $i-1-p$ inversions destroyed due to items succeding i in *MF*'s list but preceding it in A's list. Thus, the amortized time for deletion is $\hat{c}_{MF} = k-(k-1-p)-(i-1-p) = 2p-i+2 \le 2(i-1)-i+2 = i = c_A \le 2c_A - 1$.

An exchange by A has zero cost to *MF*, so the amortized time for *MF* of an exchange made by A is simply the increase in the number of inversions caused by the exchange. This increase is at most 1 for a paid exchange and is –1 for a free exchange. †
□

Theorem 2 is a farely strong statement about the quality of the move-to-front rule: no conceivable algorithm, not even an algorithm which has advance knowledge about the entire sequence of requests, can beat the MF-rule by more than a factor of two. No such result is true for the transpose rule. Consider the following sequence: We first insert $x_1, x_2, \cdots, x_n$ in that order; this generates the list $x_2\, x_3 \cdots x_n\, x_1$ under the transpose rule. We now alternately access x_1 and x_n. The cost of this sequence of operations is $\Omega(n^2 + (m-n)n)$ under the transpose rule. However, the cost under the MF-rule is only $O(n^2 + m)$ which is smaller by an arbitrary factor when m is much larger than n.

Average-Case Analysis

Here we consider the expected access cost in a static dictionary (see [1,2]). Consider the static dictionary problem (i.e. only access operations). Suppose the list contains items $x_1, x_2, \cdots, x_n$. Furthermore, assume the access probability to item x_i is p_i and without loss of generality assume $p_1 \ge p_2 \ge \cdots \ge p_n > 0$. (Note that $\sum_{i=1}^{n} p_i = 1$.) Let E_A denote the expected (i.e. average) cost of a single access operation by algorithm A. Then we have:

Theorem 3: *The following hold:*

(a) $\quad E_{DF} = \sum_{i=1}^{n} i \cdot p_i$

(b) $\quad E_{MF} = 1 + 2 \cdot \sum_{1 \le i < j \le n} \frac{p_i\, p_j}{p_i + p_j}$

(c) $\quad E_{MF} \le 2 \cdot E_{DF} - 1$

† For an access/insert operation, the accessed/inserted item appears in front of the MF's list by now.

Proof:

Part (a): This follows directly from the definition of expectation.

Part (b): We can write

$$E_{MF} = \sum_{i=1}^{n} l_i \cdot p_i \, ,$$

where l_i is the expected position of item x_i on the list. But what is the value of l_i? Let us define the following 0/1 random variables:

$$Y_{ij} = \begin{cases} 1 & \text{if } x_j \text{ appears before } x_i \text{ in the list} \\ 0 & \text{otherwise} \end{cases}$$

Now we see that the number of items preceding x_i in *MF*'s list is $\sum_{\substack{j=1 \\ j \neq i}}^{n} Y_{ij}$. Thus, l_i is the expected value of this sum plus one (for item x_i itself), i.e.,

$$l_i = 1 + E\left[\sum_{\substack{j=1 \\ j \neq i}}^{n} Y_{ij} \right].$$

Using *linearity of expectation* we can move the expectation inside the sum and get

$$E\left[\sum_{\substack{j=1 \\ j \neq i}}^{n} Y_{ij} \right] = \sum_{\substack{j=1 \\ j \neq i}}^{n} E\left[Y_{ij} \right].$$

Now the question is what is $E[\, Y_{ij} \,]$? Let $p_{ij} = Prob[Y_{ij} = 1]$ be the probability that item x_j is in front of x_i. Then, we have

$$E[Y_{ij}] = 1 \cdot p_{ij} + 0 \cdot (1 - p_{ij}) = p_{ij}.$$

Thus,

$$l_i = 1 + \sum_{\substack{j=1 \\ j \neq i}}^{n} p_{ij} \, .$$

Now, what is p_{ij}? It is a conditional probability. Let us define the following two probabilistic events. Let A denote the event corresponding to the last access to either of items x_i or x_j, and let B be the event corresponding to that last event being an access to item x_j. Here we use the argument that currently x_j appears before x_i if and only if the latest access to either of x_i or x_j was an access to x_j. Thus, p_{ij} is the conditional probability $p_{ij} = Prob[B \mid A]$. Using the formula for conditional probabilities, we have

$$p_{ij} = Prob[B \mid A] = \frac{Prob[A \cap B]}{Prob[A]} = \frac{p_j}{p_i + p_j} \, .$$

Putting the above formulas together, we get the result:

$$E_{MF} = \sum_{i=1}^{n} p_i \cdot [1 + \sum_{\substack{j=1 \\ j \neq i}}^{n} p_{ij}]$$

$$= \sum_{i=1}^{n} p_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{p_i \, p_j}{p_i + p_j}$$

$$= 1 + 2 \cdot \sum_{1 \leq i < j \leq n} \frac{p_i \, p_j}{p_i + p_j} .$$

Part (c): Start with the formula in part (b) and note that $\frac{p_i}{p_i + p_j} \leq 1$, then use the formula in part (a). □

For example, part (c) of Theorem 3 says that even in the average case, the performance of the move-to-front heuristic is never worse than twice that of the best static algorithm even though that best static algorithm may have full knowledge of the access probabilities in advance while the MF-rule assumes no such advance knowledge.

Example 2: Assume $n \geq 3$, $p_1 = 2/n$, and $p_i = \frac{n-2}{n(n-1)}$, for $i = 2, 3 \cdots, n$. Note that $p_1 \geq p_2 \geq \cdots \geq p_n > 0$ and Theorem 3 applies. Using parts (a) and (b) of the theorem, after some elementary algebraic manipulations we get $E_{DF} = n/2$, and $E_{MF} = \frac{n}{2} + \frac{1}{3} - \frac{2}{3n(n-1)}$. □

References

[1] D.E. Knuth, *"The Art of Computer Programming"*, vol III, 1973, pages 398-399.

[2] K. Mehlhorn, *"Data Structures and algorithms"*, vol I, 1984, pages 256-263.
(This includes both the amortized and average case analysis of the move-to-front heuristic.)

[3] D.D. Sleator and R.E. Tarjan, *"Amortized efficiency of list update and paging rules"*, Communications of ACM, 28(2), 1985, pp. 202-208.
(This is the original paper that does the amortized analysis of the move-to-front heuristic.)