

Self-organizing lists modify the order in which records are stored based on the actual or expected access pattern.

The goal is to achieve an ordering that keeps the most frequently sought records closest to the front of the list.

Common heuristics:

- frequency count: order by the actual historical frequency of access (similar to LFU)
- move-to-front: when a record is accessed, move it to the front of the list
- transpose: when a record is accessed, swap it with the preceding record in the list

## Pros:

- reflects the actual access pattern
- likely to keep often-accessed elements near the front
- list adjustments frequently require swaps of adjacent records, or at least near neighbors

## Cons:

- must store and maintain a counter for each record
- without modification, does not adapt quickly to changing access patterns

## Pros:

- easily implemented, requiring no extra storage
- likely to keep very frequently accessed elements near the front
- adapts quickly to changing access patterns (with respect to the front element)

## Cons:

- may provide too great a reward for infrequently accessed records
- relatively short memory of access pattern

## Pros:

- easily implemented, requiring no extra storage
- likely to keep very frequently accessed elements near the front

## Cons:

- does not adapt quickly to changing access patterns
- relatively short memory of access pattern

Suppose that a list contains  $n$  records and that the probability that the  $k$ -th record will be accessed is  $p_k$ , where  $0 < p_k < 1$ .

The cost of a search is measured by the number of comparisons that must be done to find the target record. We assume that the sought record exists.

Then the expected cost of a search is given by:

$$\bar{C}_n = 1p_1 + 2p_2 + 3p_3 + \cdots + np_n$$

If each record is equally likely to be the target, then each probability is  $1/n$  and:

$$\bar{C}_n = \sum_{k=1}^n k / n = \frac{n+1}{2}$$

If the probability for the  $k$ -th record is  $1/2^k$  then:

$$\bar{C}_n = \sum_{k=1}^{n-1} \frac{1}{2^k} + \frac{1}{2^{n-1}} \approx 2$$

Rule of Thumb: in a typical database, 80% of the access are to 20% of the records.

Not a "rule" at all, but rather a characterization of typical behavior.

The point is that it is very important that a relatively large minority of the records be easy to find.

When the rule applies, if the records are properly organized:

$$\bar{C}_n \approx 0.122n$$

Given a set of search data, the optimal static ordering organizes the records precisely according to the frequency of their occurrence in the search data.

The frequency count and move-to-front heuristics are, in the long run, at worst twice as costly as the optimal static ordering.

The transpose heuristic appears, in the long run, to approach the cost of the move-to-front heuristic.

However, any such results are somewhat imprecise since the exact relationship is sensitive to the particular data and access patterns in question.