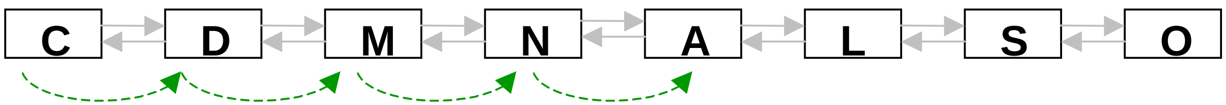
Self-Organizing Lists

COSC 2011, Fall 2003, Section A
Instructor: N. Vlajic

Self-Organizing List ADT

Binary Search on List ADT – **<u>inefficient</u>, since accessing the "middle" element involves traversal through the list**



Worst Case Search RT on List ADT

$$T(n) = O(n)$$

Average Search RT on List ADT

$$\overline{T}(n) = 1 \cdot p_1 + 2 \cdot p_2 + .. + n \cdot p_n = \sum_{i=1}^{n} i \cdot p_i$$

If searches for different items are equally likely, (p_i=1/n) it does not matter how we place the items, i.e. T(n) does not depend on individual p(i)-s.

$$\overline{T}(n) = \sum_{i=1}^{n} i \cdot p_i \Big|_{p_i = \frac{1}{n}} = \frac{n+1}{2}$$

Self-Organizing List ADT (cont.)

In a typical database, 80% of the access are to 20% of the items.

How to Minimize Average Search Time on List ADT with Non-uniform Access Probabilities ?? – match smaller i with larger p_i – i.e. place more frequently searched items closer to the beginning/front of the list

$$\overline{T}(n) = 1 \cdot p_1 + 2 \cdot p_2 + .. + n \cdot p_n = \sum_{i=1}^{n} i \cdot p_i$$

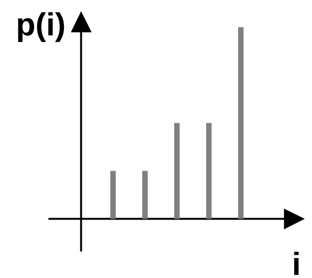
- ordering a list, so as to minimize the average access time, requires that the access pattern be known in advance
- for many applications, it may be difficult to obtain such information

Self-Organizing List ADT (cont.)

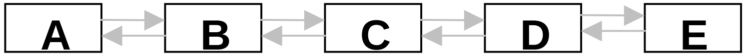
Example 2 **[self-organizing list]**

Assume 10 nodes, with the following access frequencies:

A	B	C	D	E
p_1=0.1	p_2=0.1	p_3=0.2	p_4=0.2	p_5=0.4

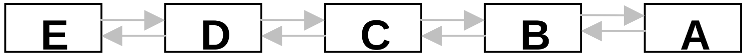


Node Arrangement (1):



$$\overline{T}(n) = 1 \cdot 0.1 + 2 \cdot 0.1 + 3 \cdot 0.2 + 4 \cdot 0.2 + 5 \cdot 0.4 = 3.7$$

Node Arrangement (2):

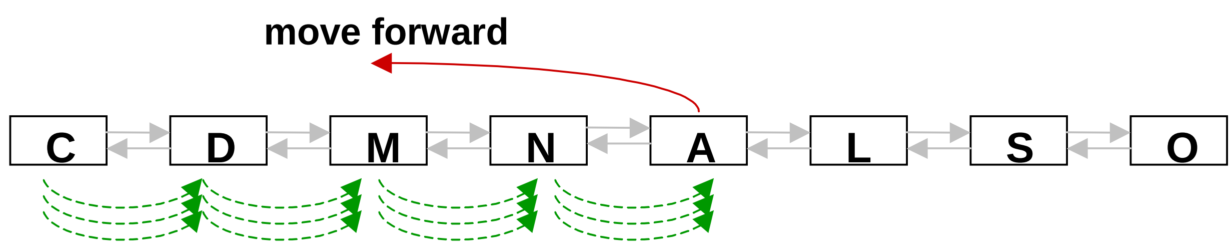


$$\overline{T}(n) = 1 \cdot 0.4 + 2 \cdot 0.2 + 3 \cdot 0.2 + 4 \cdot 0.1 + 5 \cdot 0.1 = 2.3$$

Self-Organizing List ADT (cont.)

Self-Organizing Lists – lists in which the order of elements changes based on searches which are done

- speed up the search by placing the frequently accessed elements at or close to the head



Examples – important tel. numbers placed near the front of tel. directory

Basic Strategies in Self-Organizing Lists

(1) Move-to-Front Method

(2) Count Method

(3) Exchange Method

Self-Organizing List ADT (cont.)

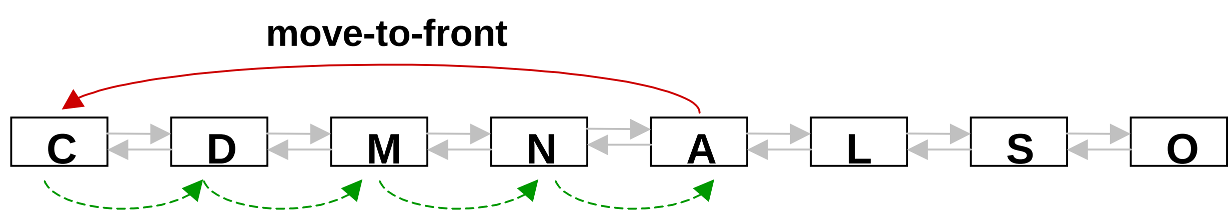
(1) <u>Move-to-Front Method</u>: **any node (position) searched / requested is moved to the front**

Pros:

- **easily implemented & <u>memoryless</u> – requires no extra storage**
- **adapts quickly to changing access patterns**

Cons:

- **may over-reward infrequently accessed nodes**
- **relatively short memory of access pattern**



Self-Organizing List ADT (cont.)

(2) <u>Count Method</u>: **each node (position) counts the number of times it was searched for – nodes are ordered by decreasing count**

Pros:

- **reflects the actual access pattern**

Cons:

- **must store and maintain a counter for each node**
- **does not adapt quickly to changing access pattern**

(3) <u>Transpose Method</u>: **any node searched is swapped with the preceding node**

Pros:

- **easily implemented & memoryless**
- **likely to keep frequently accessed nodes near the front**

Cons:

- **more cautious than "Move-to-Front" (it will take many consecutive accesses to move one node to the front)**

Self-Organizing List ADT: Implementation

Basic Set of Interface Methods

Generic Methods	public int size(); public boolean isEmpty();
Accessor Methods	**public boolean searchElement(Object e)**
Update Methods	public Object remove(Position p); **public Position insert(Object e);**

DLLNode Class in Self-Organizing List with "Count Method"

```
public class SelfOrganizingDLLNode implements Position {
    private Object element;
    private DLLNode prev, next;
    public int accessCounter;
    public Object element() {return element;}
    … /* getElement(), setElement(), getNext(), … */
}
```

Counts the number of times that an instance of this class was searched for and successfully found.

List ADT: Questions

Q.1 Assume we want to add the following method to the List ADT interface: insertAtRank(index k, Position p). What would be the running time of this method, assuming DLL implementation?

Q.2 Under what circumstances will a self-organizing list perform better than an ordinary (linked) list?

Q.3 What is the worst-case search-time on a list with the following access frequencies:

$$p_i = \begin{cases} \frac{1}{2^i}, & i = 1,..,n-1 \\ \frac{1}{2^{n-1}}, & i = n \end{cases}$$

(p_i represents the access probability of $item_i$.)

What would be the average search time on such a list, if the list was self-organized and employed “count method”?