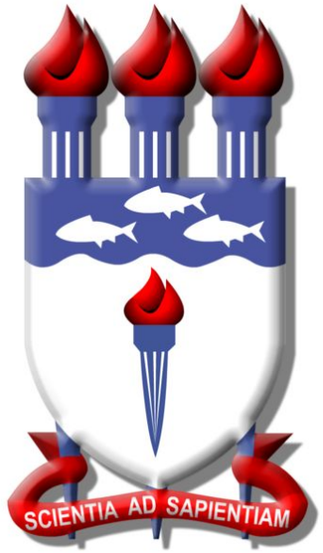


Self Organizing Lists

- Marcelo Lima
- Victor Hugo de Lima Araújo
- William Kleber

https://github.com/williamkleber1/data-structure-project

Motivação

- Listas normais possuem busca lenta, sempre linear.

- Não dá pra fazer busca binária pois para alcançar os nós sempre tem que percorrer a lista.

- Pensando nisso, buscou-se um modo de fazer a busca em listas um pouco melhor.

- “Geralmente em um banco de dados típico, 80% do acesso é feito em apenas 20% dos itens.”

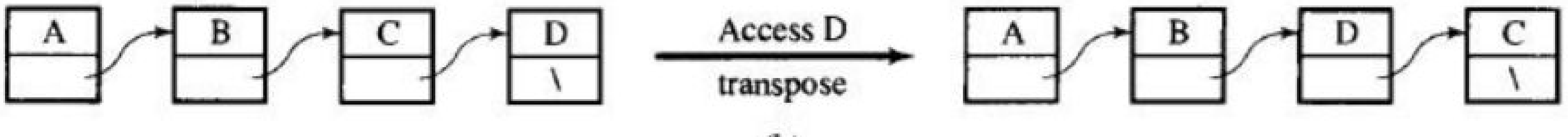
Self Organizing Lists

- Basicamente é uma lista que muda a posição dos elementos baseada em alguma heurística de forma a tornar a busca um pouco mais rápida.

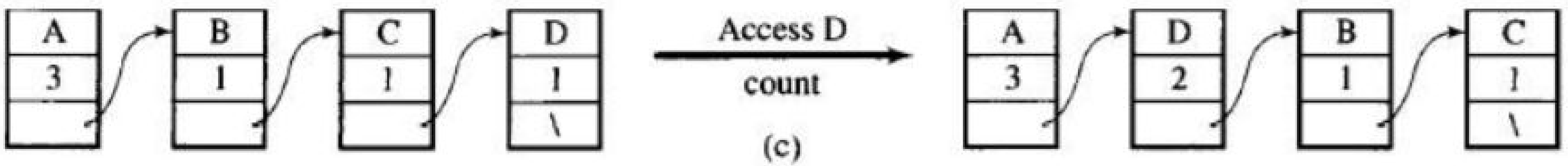
- Fazendo um balanceamento dos elementos, é possível por exemplo, deixar elementos mais comumente acessados na frente da lista.


A
B
C
D
\
Access D
move-to-front
D
A
B
C
\

(a)


A
B
C
D
\
Access D
transpose
A
B
D
C
\

(b)


A
3
B
1
C
1
D
1
\
Access D
count
A
3
D
2
B
1
C
1
\

(c)

Definições

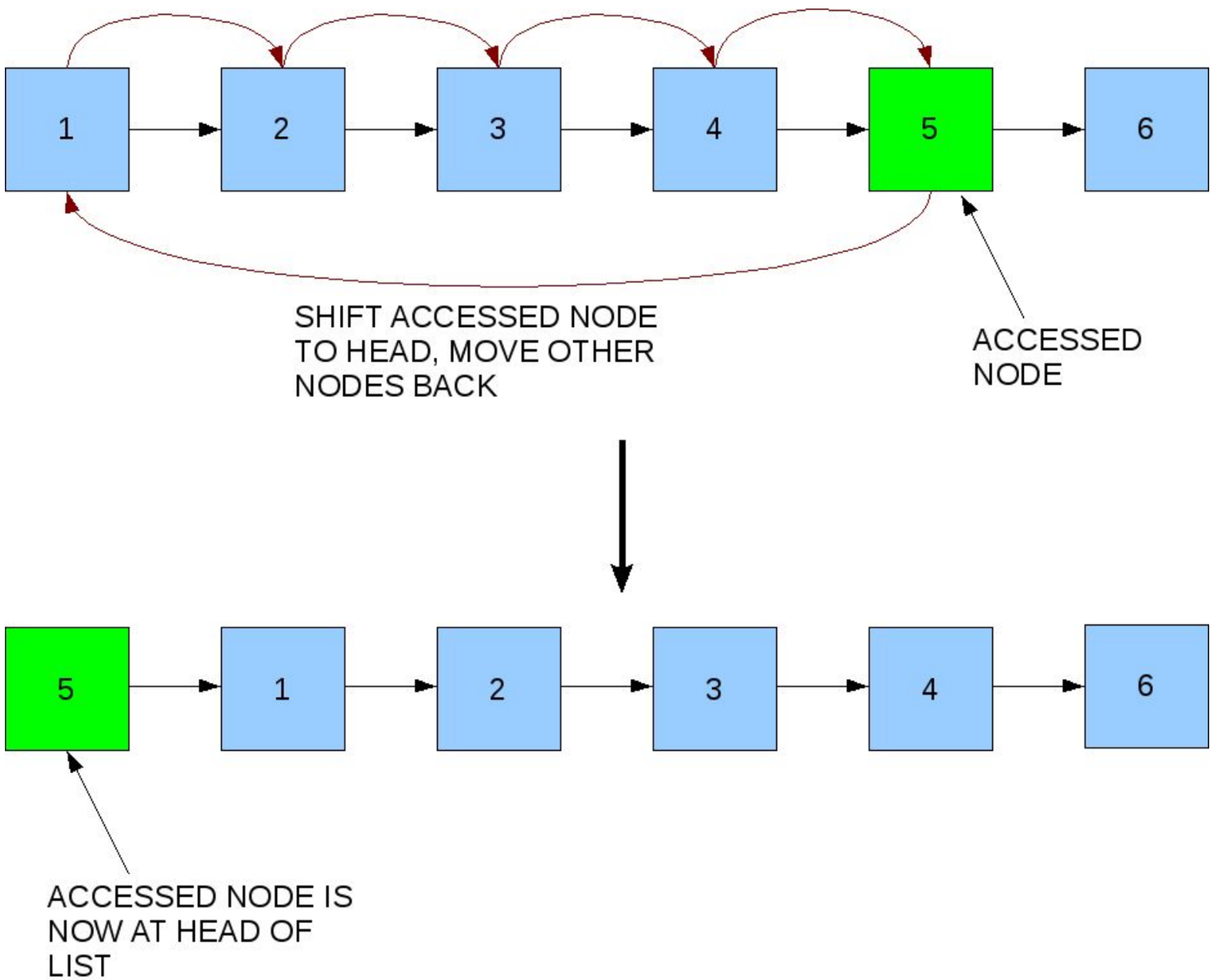
- Heurística: “é um conjunto de regras e métodos que conduzem à descoberta, à intervenção e resolução de problemas”

- Nó

- Métodos mais utilizados para se fazer uma lista alto-balanceável:
  - Move to Front
  - Count
  - Transpose

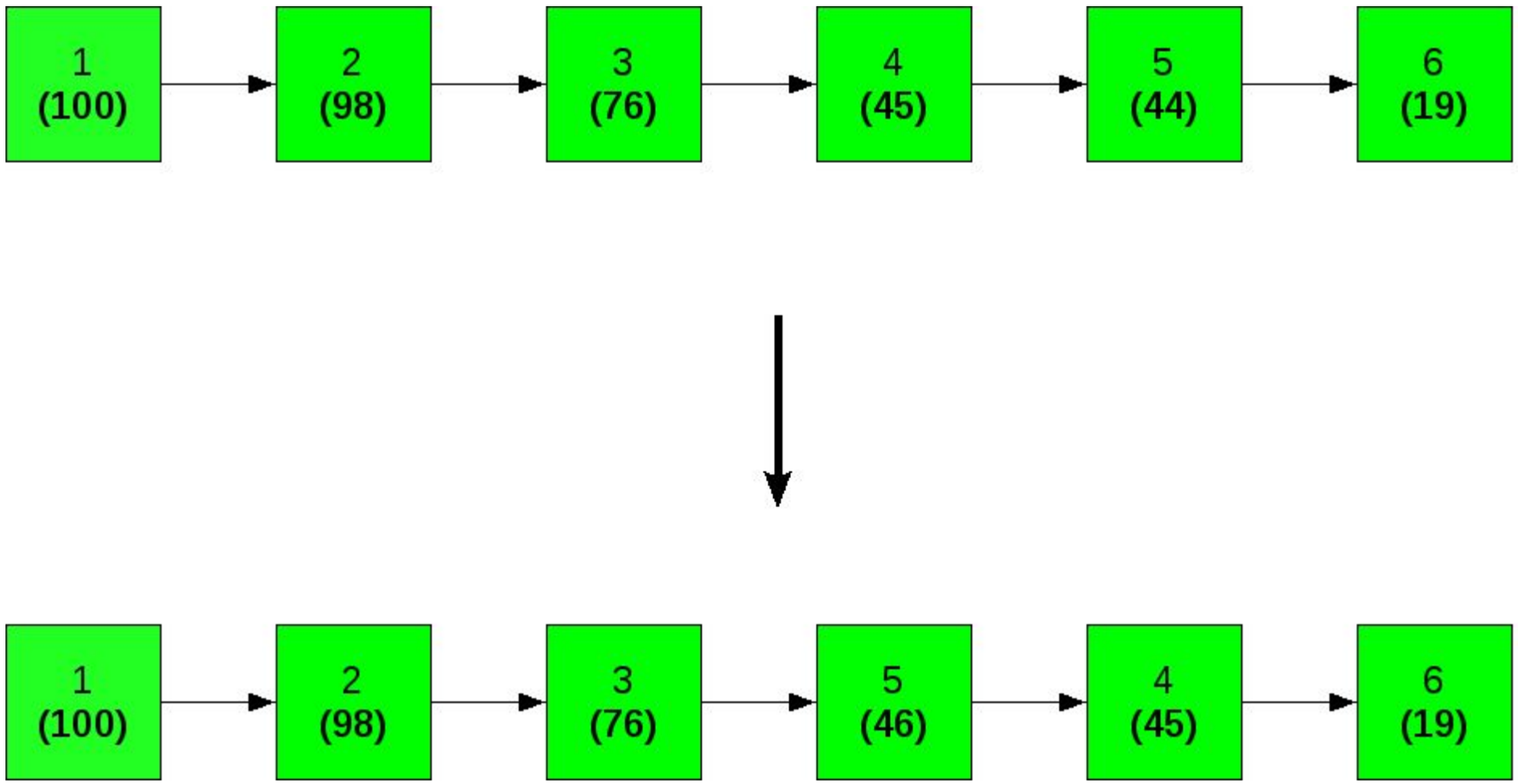
Método: Move to Front

- Cada vez que um elemento é buscado, ele é movido para frente da lista



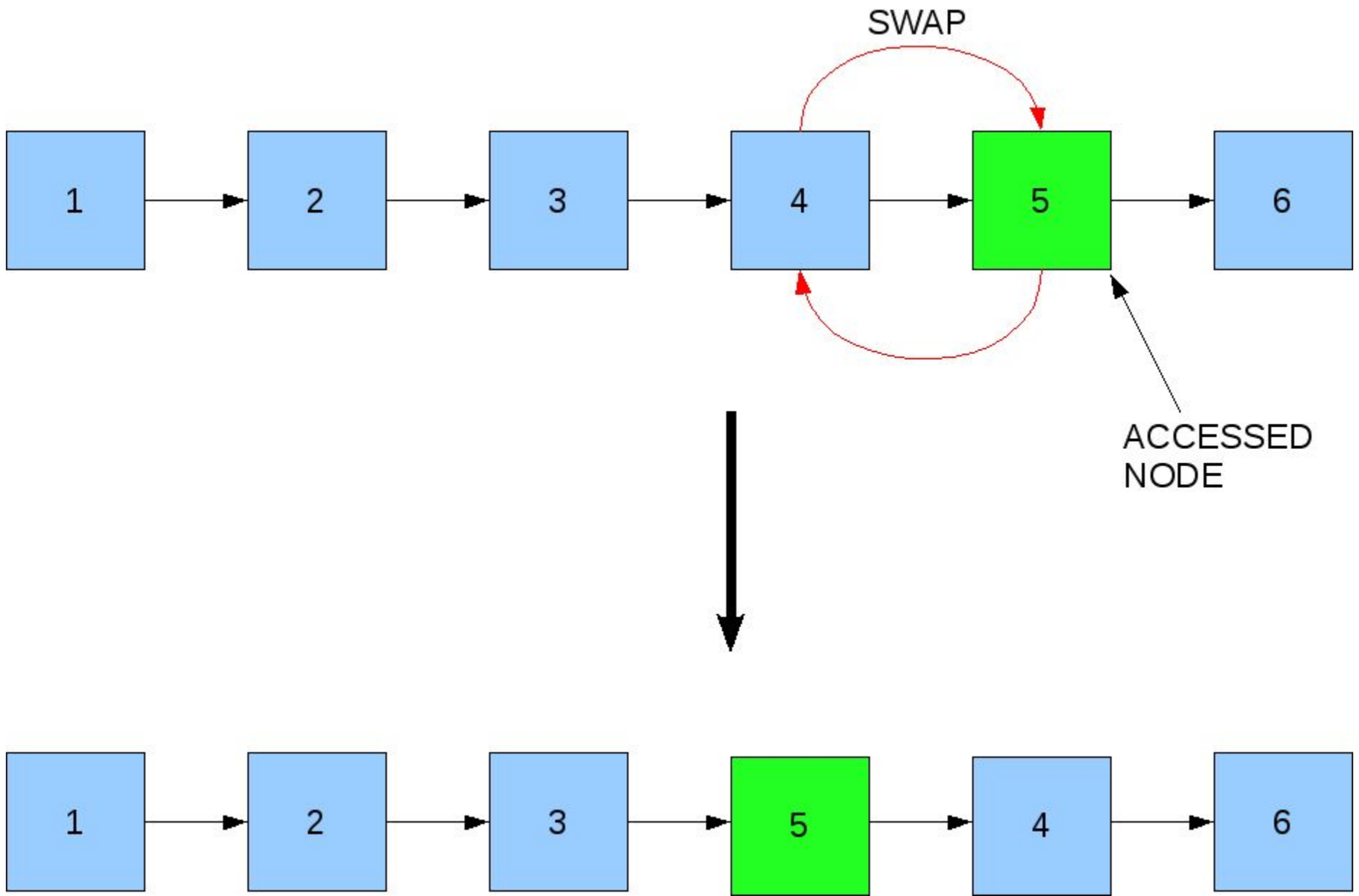
Método: Count

- Cada elemento da lista possui um contador que é acrescido sempre que o elemento é buscado, os elementos com maior frequência ficam no começo da lista.



Método: Transpose

- Cada vez que um elemento é buscado, ele é trocado com elemento que o precede.



TAD

```
typedef struct node Node;
struct node{
  int item;
  Node* next;
 Node* prev;
}
```

swap(int a, int b);

```
Node* search_node(Node *first, int item);
```

Busca usando Move to Front

```
Node* search(Node *first, int item)
{
    Node *node;
    Node *aux = first;
    for(node = first; node->next != NULL; node = node->next)
    {
        if(node->item == item)
        {
            if(item == first->item)
            {
            return first;
            }
            node->next->prev = node->prev;
            node->prev->next= node->next;
            node->next = aux;
            node->prev = NULL;
            aux->prev = node;
            return node;
        }
    }
    return NULL;
}
```

Busca usando o Transpose

```
Node* search(Node *first, int item)
{
    Node *node;
    Node *aux;
    for(node = first; node->next != NULL; node = node->next)
    {
        if(node->item == item)
        {
            if(item = first->item)
            {
                return first;
            }

                swap(node->item,node->prev->item);
                return node->prev;
        }
    }
    return NULL;
}
```

De volta à Motivação…

- Exemplo da busca em uma aplicação de dicionário.
- Dados mais acessados de um BD por exemplo, ficam sempre a frente o que torna a busca bem mais rápida.
- O pior caso continua O(n), porém o caso médio melhora consideravelmente.

Referências

- https://pt.wikipedia.org/wiki/Heur%C3%ADstica_(computa%C3%A7%C3%A3o)
- http://www.eecs.yorku.ca/course_archive/2003-04/F/2011/2011A/DatStr_071_SOLists.pdf
- https://en.wikipedia.org/wiki/Self-organizing_list
- http://www.sanfoundry.com/cpp-program-implement-self-organizing-list/
- http://courses.cs.vt.edu/~cs2604/spring04/Notes/C16.SelfOrganizingLists.pdf