LESE - Lista Estática Simplesmente Encadeada

A LESE (Lista Estática Simplesmente Encadeada) guarda semelhanças com a LSE.

Para o estudo da primeira será conveniente utilizar analogias com a segunda.

A LSE pode variar de comprimento:

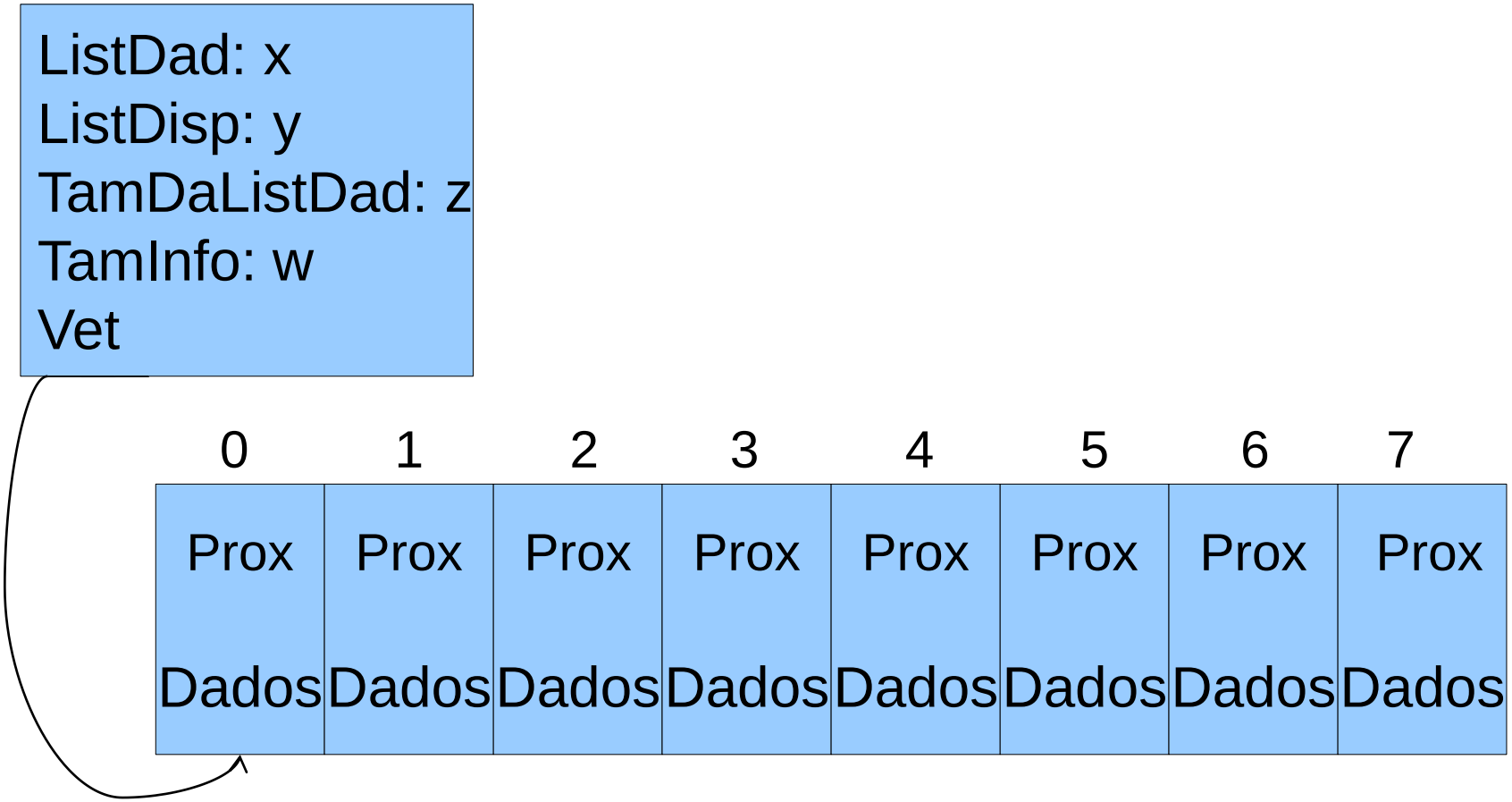
- **Gerenciamento por alocação (malloc) ou liberação (free) de memória sob a demanda das inserções ou remoções.**

A LESE é um encadeamento de células de um vetor:

- **LESE é implementada sobre um vetor e o encadeamento é feito por um campo inteiro que indexa a posição do sucessor;**
- **Possui tamanho máximo estático determinado na sua criação;**
- **Todas as alocações de memória (malloc) são realizadas na criação da LESE;**
- **Apenas na destruição ocorrem as liberações de memória (free);**
- **A LESE deve prover o próprio gerenciamento do espaço de memória (células) no vetor.**

Para a LESE, assim como para a LSE, cada nó possui um campo de ligação que o encadeia ao seu nó sucessor na lista.

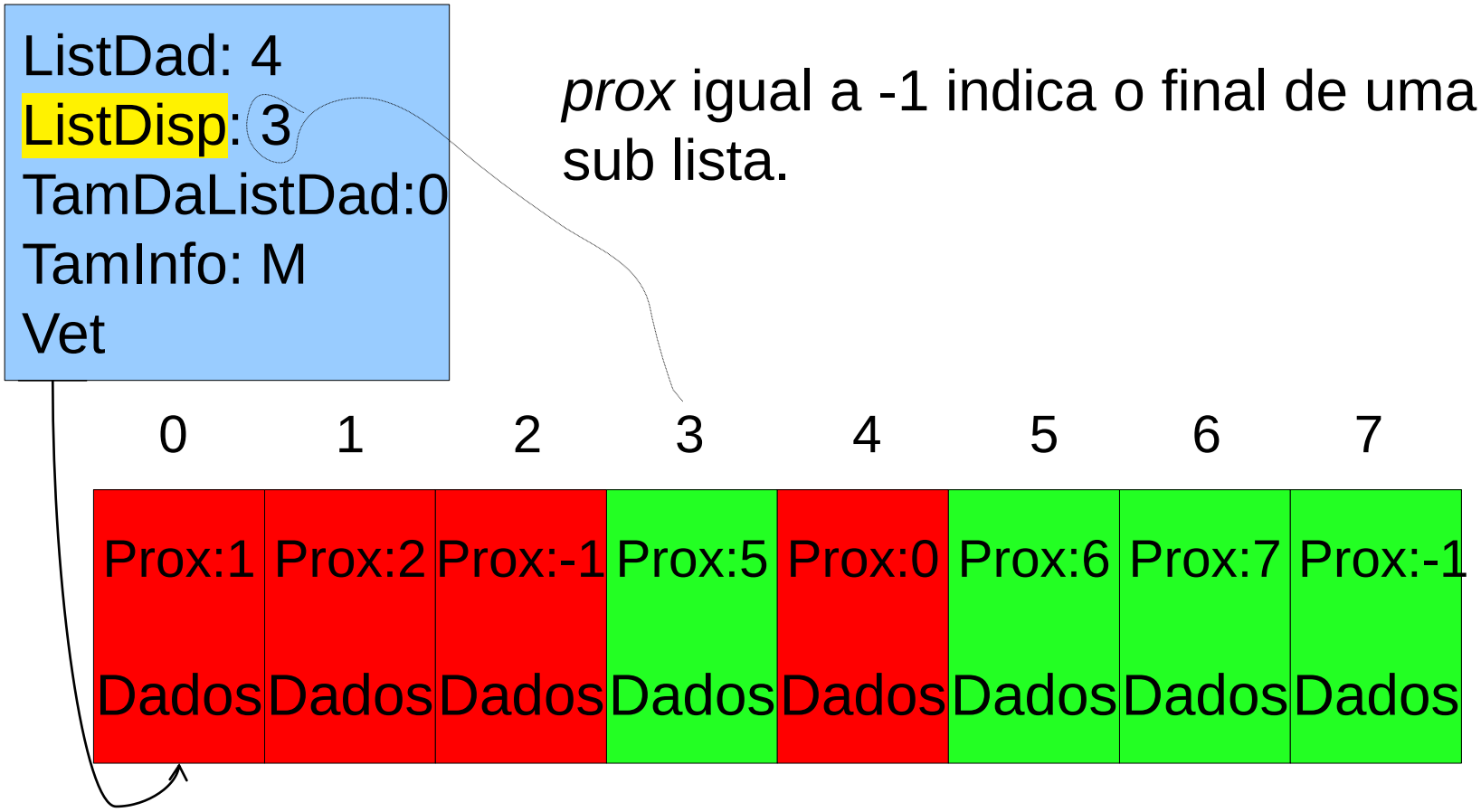
Porém... a LESE é implementada sobre um vetor e o encadeamento é feito por um campo inteiro que indexa a posição do sucessor. Na verdade há duas sub listas coexistindo.



A LESE é composta de duas sub listas no mesmo vetor:

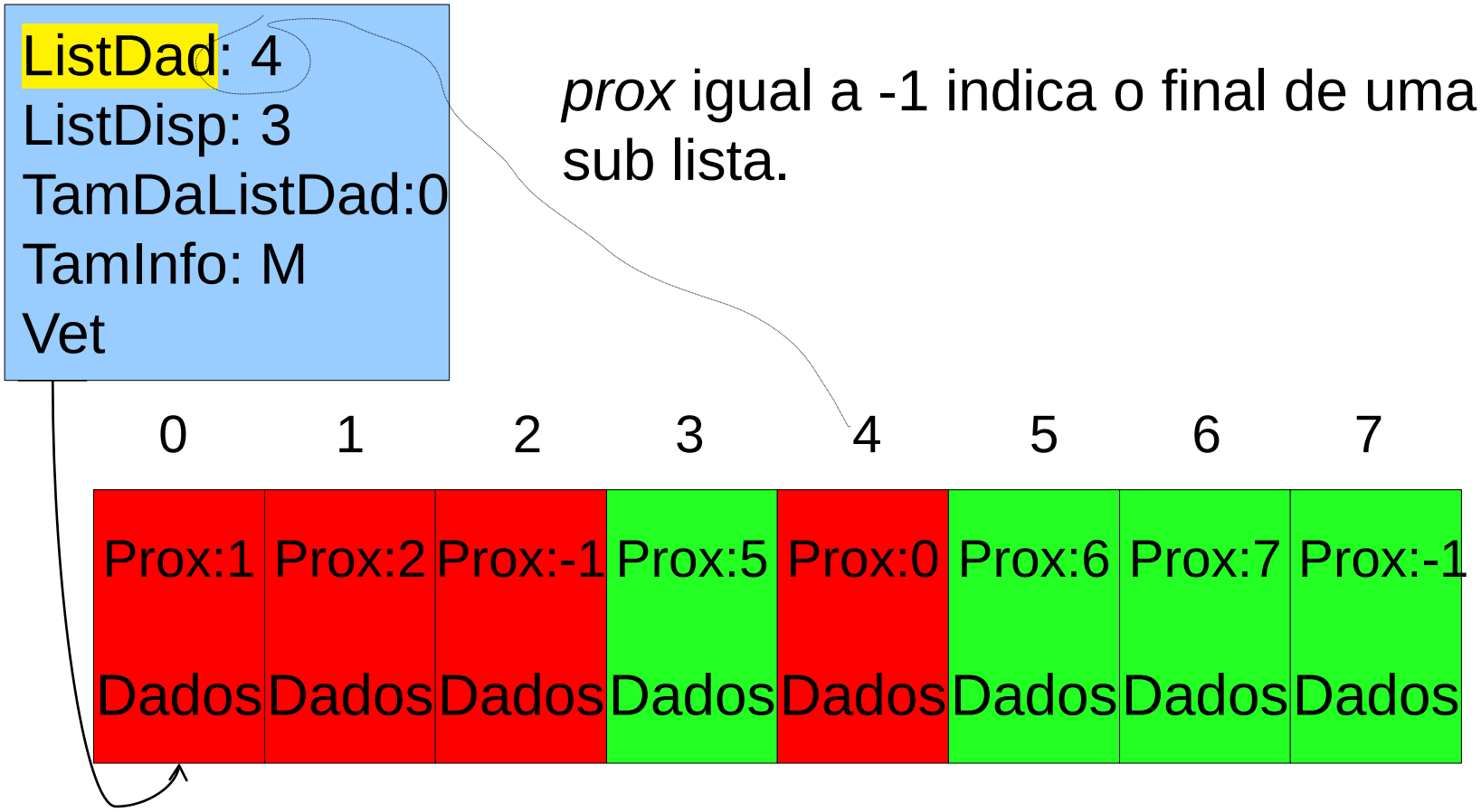
• Uma lista de disponibilidade iniciada em *listDisp*, privativa à LESE, a qual é utilizada no gerenciamento das posições disponíveis (células vagas) no vetor;

• A lista de disponibilidade aparece em verde na figura abaixo:



A LESE é composta de duas sub listas no mesmo vetor:

- **Ao mesmo tempo existe uma lista de dados propriamente dita, essa é iniciada em *listDad*, a lista de dados ativos na LESE.**
- **A lista de dados ativos aparece em vermelho na figura abaixo:**



A LESE provê as operações privadas *alocaPos(...)* e *liberaPos(...)* para interação com a lista de disponibilidade, alocando ou devolvendo posições para a mesma.

***alocaPos(...) e liberaPos(...)* são de uso privado ao TDA, não estão disponíveis na interface com o mundo.**

alocaPos(...) e liberaPos(...)* desempenham papéis similares às congêneres *malloc e free.

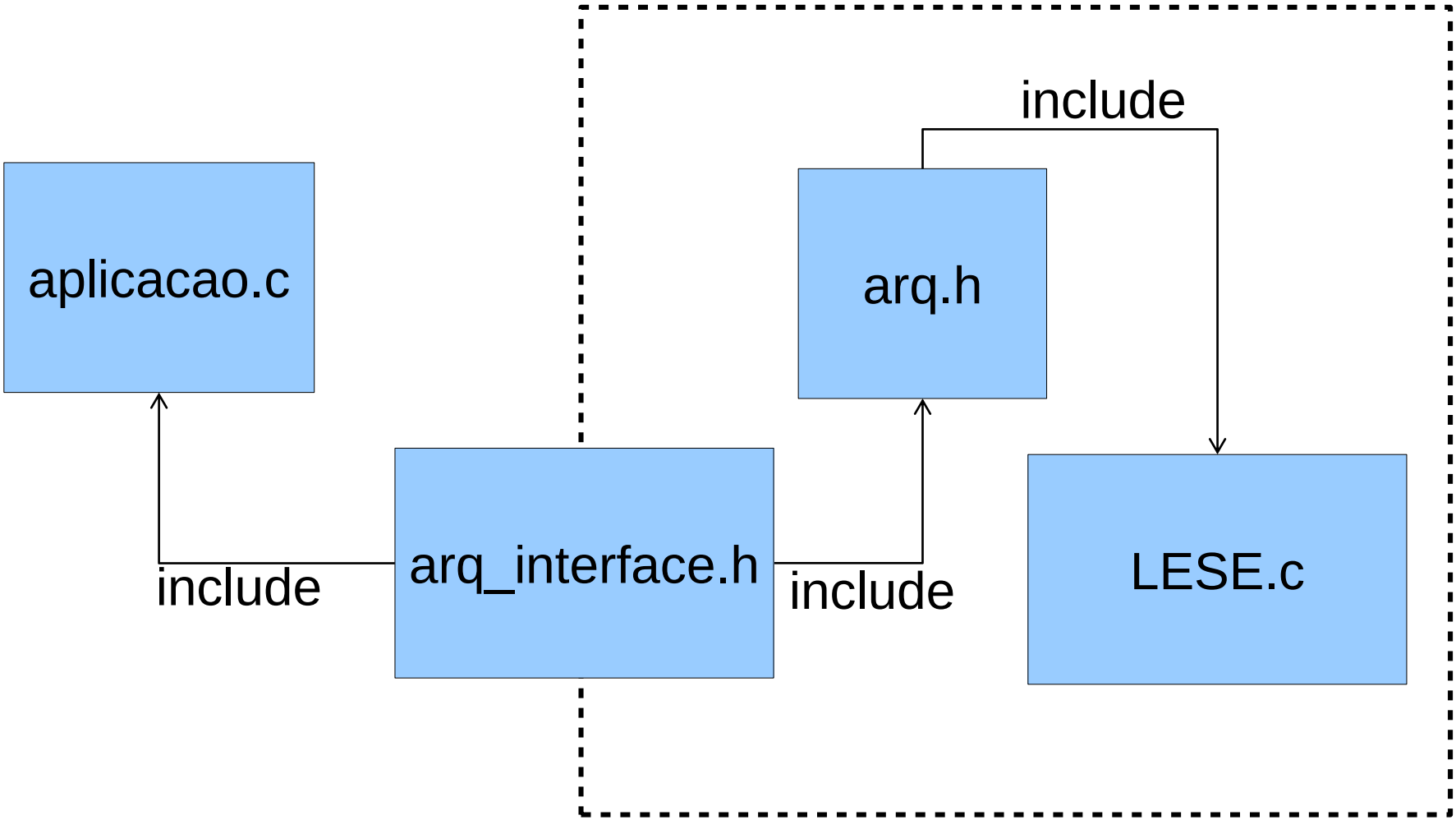
Para a LESE o valor –1 (menos um) será usado com a mesma função do NULL nos casos da LSE, LDE.

Operações privativas do TDA-LESE

alocaPos( ): obtém uma posição vaga, retornando o número inteiro que indexa tal posição no vetor. Esta função deve ser utilizada quando for necessário adicionar nova informação na lista.

liberaPos( ): devolve uma posição à lista de nós disponíveis, inserindo-a sempre no inicio desta. Esta função deve ser utilizada durante a remoção de elementos da lista de dados.

As operações privativas são escondidas da interface por meio de prototipagem em arquivo específico (arq.h) fora da interface geral (arq_interface.h)



arq_interface.h

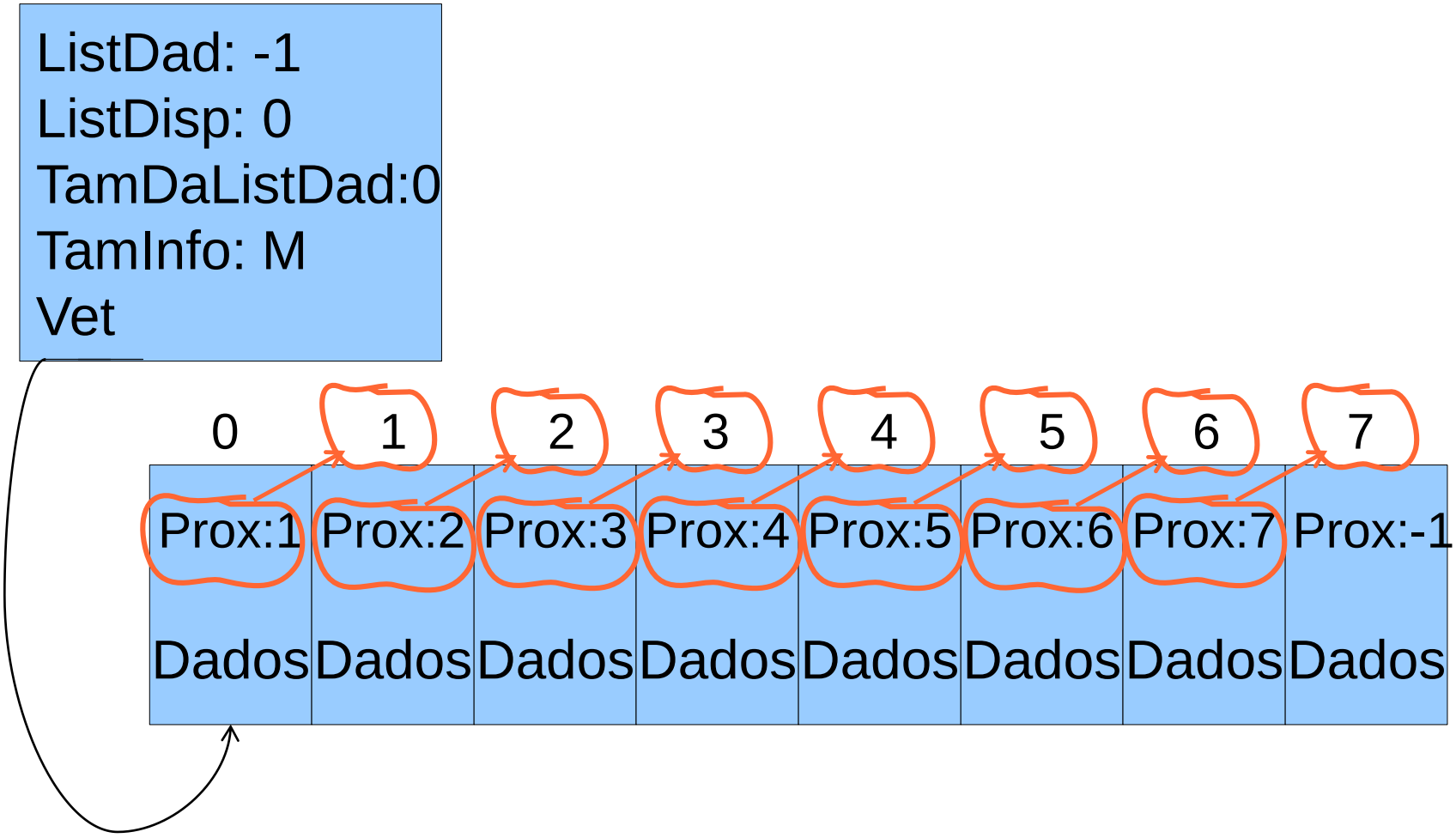
```
/* Nó de dados */
struct noLESE{
        info dados;
        int prox;    /* índice do vetor que corresponde ao próximo nó
                        de dados  */
};

/* Descritor */
struct descLESE{
        int listDad; /* início da lista de dados */
        int listDisp; /* início da lista de disponibilidade */
        int tamanhoDaListaDeDados;
        int tamInfo;
        noLESE **vetor;
} ;
```

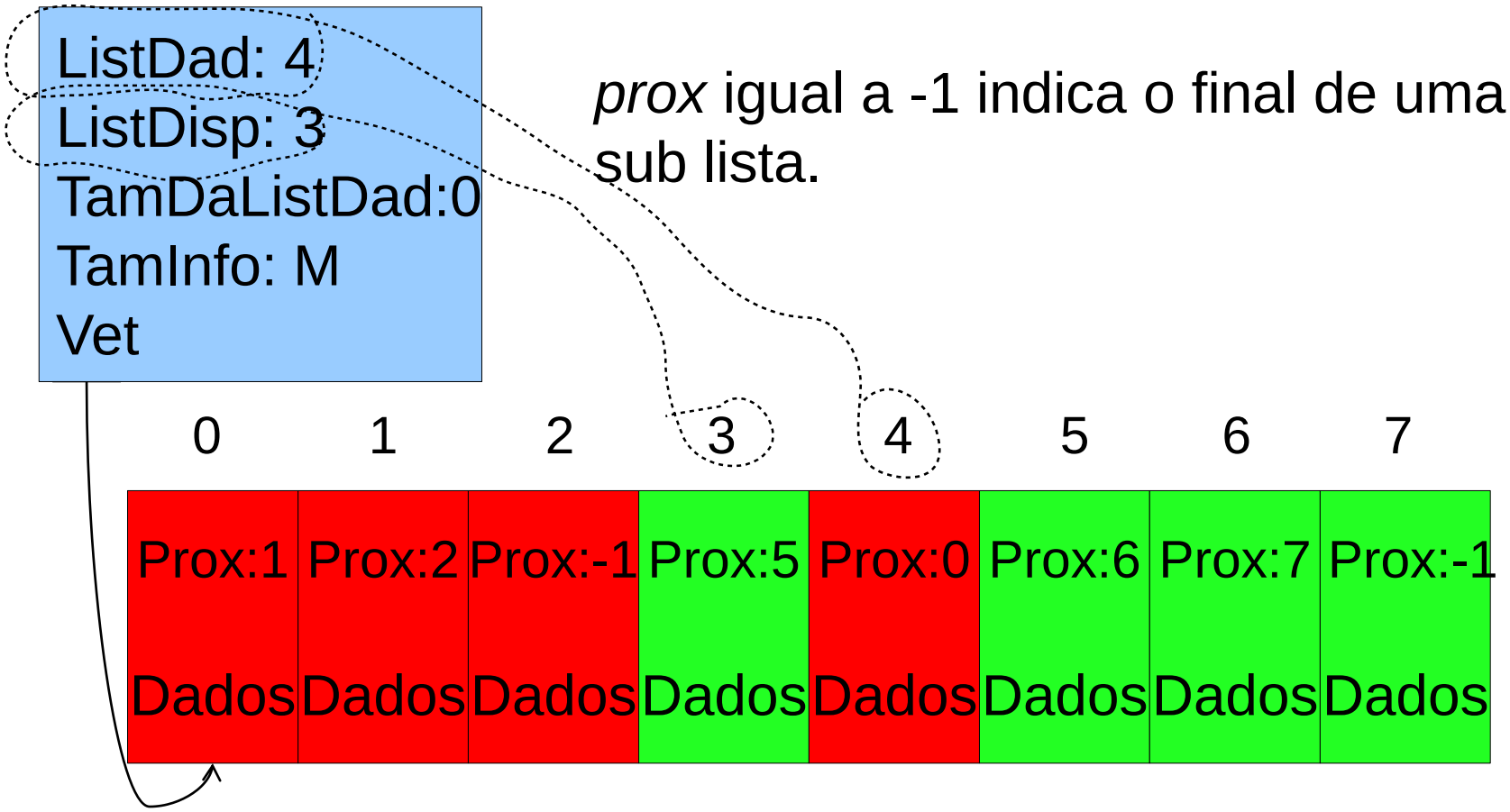
Criação da LESE

```
struct descLESE * cria(int tamanhoVetor, int tamInfo)
{   int i;
    struct descLESE *desc = (struct descLESE*) malloc(sizeof(struct descLESE));
    if( desc != NULL )
    {   if( (desc->vet = (struct noLESE*) malloc(tamanhoVetor*sizeof(struct noLESE))) != NULL )
        {   desc->listDados = -1;
            desc->listDispo = 0;
            desc->tamInfo = tamInfo;
            desc->tamanhoDaListaDeDados=0;
            // todas as posiçoes estão na lista de disponibilidade (listdispo)
            for(i=0; i < tamanhoVetor-1; i++)
                desc->vet[i].prox = i+1;
            desc->vet[i].prox = -1;
            return desc;
        }
        else
        {   free(desc);
            return NULL;
        }
    }
    return NULL;
}
}
```

LESE recém criada: vazia de dados, cheia de posições vagas



Estado da LESE após uma sequência de diferentes operações



Funções de gerenciamento de espaço no vetor

```
/* obtém  uma posição (índice do vetor) da lista de disponibilidade.*/
int alocaPos(pLista p )
{ int temp;
  temp = p->listDispo;
  if (temp > -1)
        p->listDispo = p->vet[p->listDispo].prox;
  return temp;
}

/* devolve uma posição à lista de disponibilidade, inserindo sempre no
inicio desta. Tal posição é um índice do vetor */
liberaPos(pLista p, int posicao)
{ p->vet[posicao].prox = p->listDispo;
  p->listDispo = posicao;
  return;
}
```

Verificação do estado da LESE: cheia ou vazia

```
int testaCheia(pLESE p)
{
   return ( p->listDisp == -1 ? SIM:NAO);
}


int testaVazia(pLESE p)
{
   return ( p->listDad == -1 ? SIM:NAO);
}
```

Semelhança lógica com a LDSE

```
insereNaPosicaoLogica(pLese, novo, posLog)
SE (posLog > 0 E vazia(pLese) == NAO)
        SE(posLog == 1)
                RETORNA insereNovoPrimeiro(pLese, novo)
        SENAO
                cont = 2
                aux1 = pLese->listDad
                aux2 = pLese->vet[aux1].proximo
        ENQUANTO(aux2 <> nulo E posLog > cont)
                aux1 = aux2
                aux2 = pLese->vet[aux2].proximo
                cont = cont + 1
        SE(posLog == cont) /*aux1 na posLog-1 e aux2 na posLog */
                pLese->vet[aux1].proximo = alocaPos(pLese);
                temp=pLese->vet[aux1].proximo
                        pLese->vet[temp].proximo = aux2
                pLese->vet[temp].dados =  novo
                RETORNA sucesso
       RETORNA fracasso
```