Matriz Esparsa

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Baseado nos slides do Prof. Rudinei Goularte

Conteúdo

- ➔ Matrizes Esparsas – Listas Cruzadas
- ➔ Representação Alternativa – Listas Cruzadas Circulares

O Problema

- Representação de matrizes com muitos elementos nulos
  - P.ex., matriz abaixo, de 5 linhas por 6 colunas: apenas 5 dos 30 elementos são não nulos

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

O Problema

- Representação de matrizes com muitos elementos nulos
  - Precisamos de uma representação que evite o armazenamento de tantos zeros
  - Solução: utilizar listas cruzadas como estruturas de dados

Uso da matriz tradicional

- Vantagem
  - Ao se representar dessa forma, preserva-se o acesso direto a cada elemento da matriz
    - Algoritmos simples
- Desvantagem
  - Muito espaço para armazenar zeros

Matrizes esparsas

- Necessidade
  - Método alternativo para representação de matrizes esparsas
- Solução
  - Estrutura de lista encadeada contendo somente os elementos não nulos

Matrizes esparsas - solução 1

➔ Listas simples encadeadas

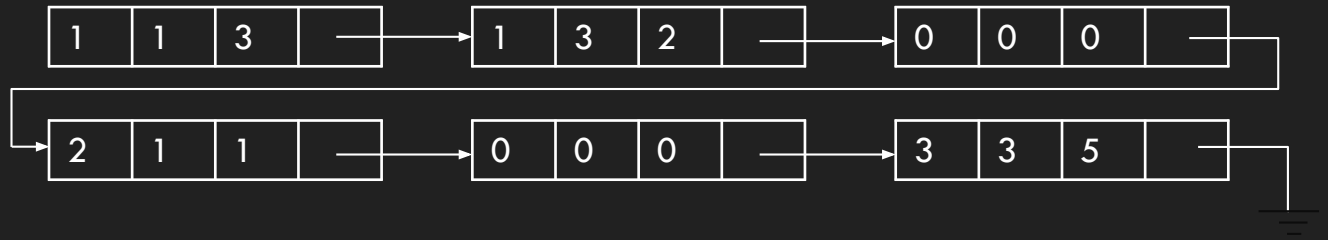
linha	coluna	valor	próximo

A
3 x 3

$$\begin{pmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

Estrutura de um nó:

- linha, coluna: posição
- valor: ≠ zero
- próximo: próximo nó



Nós zerados **opcionais** para auxiliar na divisão de linhas

Matrizes esparsas - solução 1

- ➔ Desvantagens
  - ◆ Perda da natureza bidimensional de matrizes
    - Acesso ineficiente à linha
      - Para acessar o elemento na i-ésima linha, deve-se atravessar as i-1 linhas anteriores
    - Acesso ineficiente à coluna
      - Para acessar os elementos na j-ésima coluna, tem que se passar por várias outras antes

Matrizes esparsas - solução 1

- Questão
  - Como organizar essa lista, preservando a natureza bidimensional de matriz?

Matrizes esparsas - solução 2

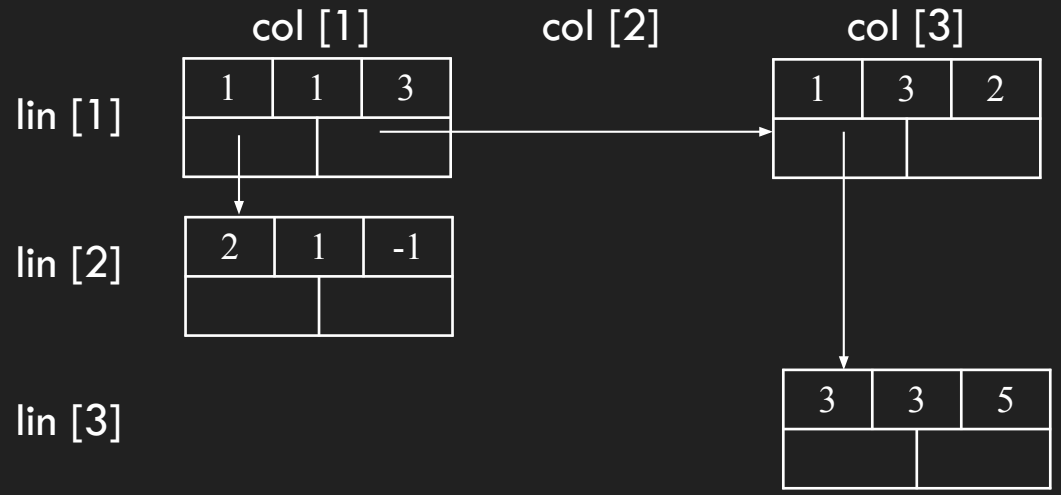
- Listas cruzadas
  - Para cada matriz, usam-se dois vetores com N ponteiros para as linhas e M ponteiros para as colunas

A
3 x 3

$$\begin{pmatrix} 3 & 0 & 2 \\ -1 & 0 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

Estrutura de um nó:

linha	coluna	valor
abaixo		direita



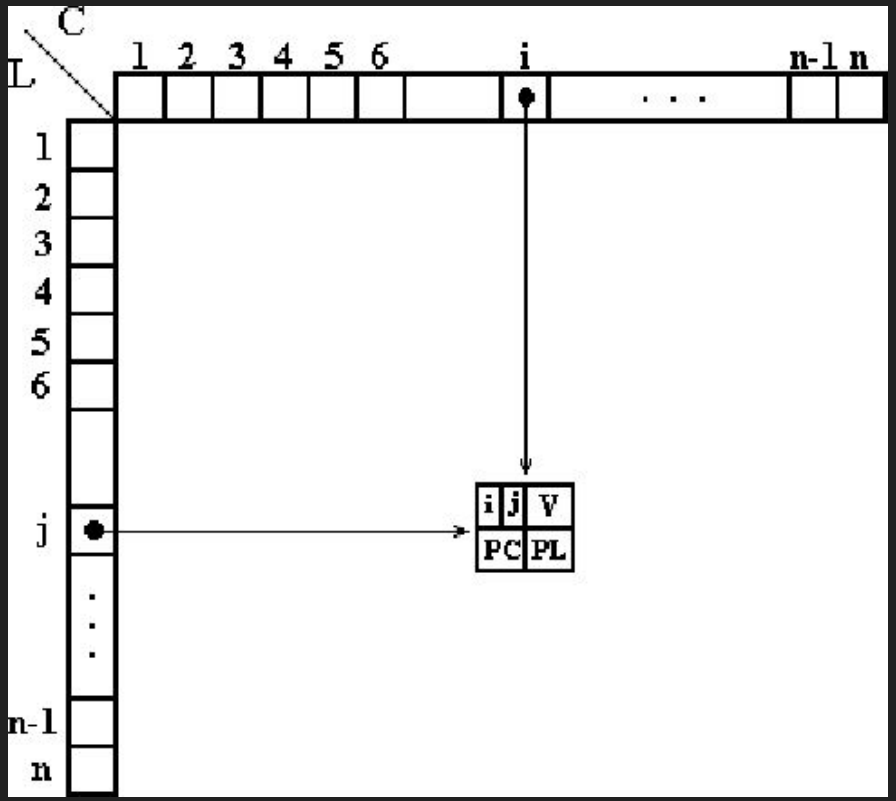
Matrizes esparsas - solução 2

- Listas cruzadas
  - Cada elemento não nulo é mantido simultaneamente em duas listas
    - Uma para sua linha
    - Uma para sua coluna

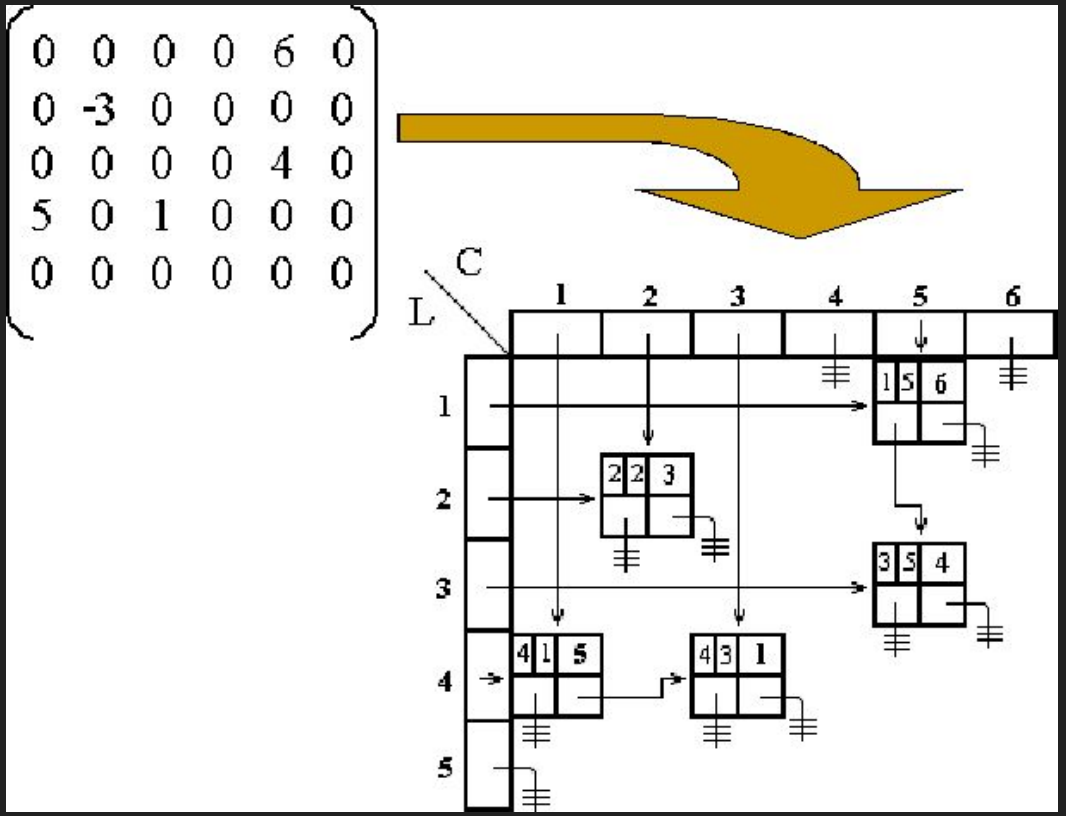
Representação por Listas Cruzadas

- Cada elemento identificado pela sua linha, coluna, e valor
- Cada elemento a_{ij} não-nulo pertence a uma lista de valores não nulos da linha *i* e também a uma lista de valores não nulos da coluna *j*
- Assim, para matriz de *nl* linhas e *nc* colunas, teremos *nl* listas de linhas e *nc* listas de colunas

Listas Cruzadas



Listas Cruzadas



Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Em termos de espaço
    - Supor que inteiro e ponteiro para inteiro ocupam um bloco de memória
    - Listas cruzadas: tamanho do vetor de linhas (nl) + tamanho do vetor de colunas (nc) + n elementos não nulos * tamanho do nó
      - nl+nc+5n
    - Matriz tradicional bidimensional
      - nl*nc

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Em termos de tempo
    - Operações mais lentas em listas cruzadas: acesso não é direto
  - Necessidade de avaliação tempo-espaço para cada aplicação
  - Em geral, usa-se listas cruzadas quando **no máximo** 1/5 dos elementos forem não nulos
    - De onde vem isso?
      - Dica: nl+nc+5n < nl*nc

TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas
- Operações principais
  - *criar_matriz(n, m)* : cria uma nova matriz esparsa vazia com *n* linhas e *m* colunas
  - *set(M, lin, col, valor)* : define um valor na posição *(lin, col)* da matriz esparsa *M*
  - *get(M, lin, col)* : retorna o valor na posição *(lin, col)* da matriz esparsa *M*

TAD Matriz Esparsa

- Operações auxiliares (podem ser criadas a partir das operações principais)
  - *somar_matriz(M1, M2, R)* : Soma as matrizes *M1* e *M2* e armazena o resultado em *R*
  - *multiplicar_matriz(M1, M2, R)* : Multiplica as matrizes *M1* e *M2* e armazena o resultado em *R*
  - *somar_coluna(M, V, col)* : Soma uma constante *V* a todos os elementos da coluna *col* da Matriz *M*
  - *somar_linha(M, V, lin)* : Soma uma constante *V* a todos os elementos da linha *lin* da Matriz *M*
  - E mais: inverter, transpor, calcular determinante, etc...

Estrutura de Dados

- A implementação é facilitada se as listas contêm o nó cabeça

```
typedef struct matriz_esparsa MATRIZ_ESPARSA;
typedef struct CELULA {
     int linha;
     int coluna;
     float valor;
     struct CELULA *direita;
     struct CELULA *abaixo;
} CELULA;

struct matriz_esparsa {
     CELULA **linhas;
     CELULA **colunas;
     int nr_linhas;
     int nr_colunas;
};
```

Operações

- Vamos implementar as operações criar_matriz(...), apagar_matriz(...), set(...) e get(...) do conjunto de operações principais
- As demais operações principais e auxiliares ficam como exercício
- Entretanto, vamos discutir alguns aspectos importantes dessas operações

TAD Matriz Esparsa

```c
#ifndef MATRIZ_ESPARSA_H
#define MATRIZ_ESPARSA_H

typedef struct matriz_esparsa MATRIZ_ESPARSA;

MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas);
void apagar_matriz(MATRIZ_ESPARSA **matriz);
int set(MATRIZ_ESPARSA *matriz, int lin, int col, float val);
float get(MATRIZ_ESPARSA *matriz, int lin, int col);
void imprimir_matriz(MATRIZ_ESPARSA *matriz);

#endif
```

Criar Matriz

```
MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas) {
     MATRIZ_ESPARSA *mat = (MATRIZ_ESPARSA *) malloc(sizeof (MATRIZ_ESPARSA));
     if (mat != NULL) {
          int i;
          mat->nr_colunas = nr_colunas;
          mat->nr_linhas = nr_linhas;
          mat->colunas = (CELULA **) malloc(sizeof (CELULA *) * nr_colunas);
          mat->linhas = (CELULA **) malloc(sizeof (CELULA *) * nr_linhas);
          if (mat->colunas != NULL && mat->linhas != NULL) {
               for (i = 0; i < nr_colunas; i++)
                    mat->colunas[i] = NULL;
               for (i = 0; i < nr_linhas; i++)
                    mat->linhas[i] = NULL;
          }
     }
     return (mat);
}
```

Apagar Matriz

```
void apagar_matriz(MATRIZ_ESPARSA **matriz) {
    int i;
    for (i = 0; i < (*matriz)->nr_linhas; i++) {
        if((*matriz)->linhas[i] != NULL){
            CELULA *paux = (*matriz)->linhas[i]->direita;
            while (paux != NULL) {
                CELULA *prem = paux;
                paux = paux->direita;
                free(prem); prem = NULL;
            }
        }
        free((*matriz)->linhas[i]); (*matriz)->linha[i] = NULL;
    }
    free((*matriz)->linhas); (*matriz)->linhas = NULL;
    free((*matriz)->colunas); (*matriz)->colunas = NULL;
    free((*matriz));
    *matriz = NULL;
}
```

Definir Valor

```
int set(MATRIZ_ESPARSA *matriz, int lin, int col, float val) {
      CELULA *p, *q, *qa;
      p = (CELULA *) malloc(sizeof(CELULA));
      if ((p == NULL) || (lin > matriz->nr_linhas) || (col > matriz->nr_colunas)) return(0);
      p->linha = lin; p->coluna = col; p->valor = val;
      //inserir na lista da coluna col
      q = matriz->colunas[col]; qa = NULL;
      while(q != NULL){
            if (q->linha < lin){
                  qa = q;
                  q = q->abaixo;
            }else{ //achou linhas maior
                  if (qa == NULL)
                        matriz->colunas[col] = p;
                  else
                        qa->abaixo = p;
                  p->abaixo = q;
                  break;}} //não façam isso, é só pra caber no slide!
      //inserir como último da lista colunas
      if (q == NULL)
            if (qa == NULL)
                  matriz->colunas[col] = p;
            else
                  qa->abaixo = p;
// algortimo simetrico para as linhas
```

Retornar Valor

```
float get(MATRIZ_ESPARSA *matriz, int lin, int col) {
     if (lin < matriz->nr_linhas && col < matriz->nr_colunas) {
          CELULA *paux = matriz->linhas[lin];
          if (paux != NULL){
               while (paux->direita != NULL && paux->direita->coluna <= col)
                    paux = paux->direita;
               if (paux->coluna == col)
                    return (paux->valor);
          }
     }
     return (0);
}
```

Operações

- E quando um elemento da matriz original se torna não nulo, em consequência de alguma operação? É necessário inserir na estrutura?
- E quando um elemento da matriz original se tornar nulo? É necessário eliminar da estrutura?

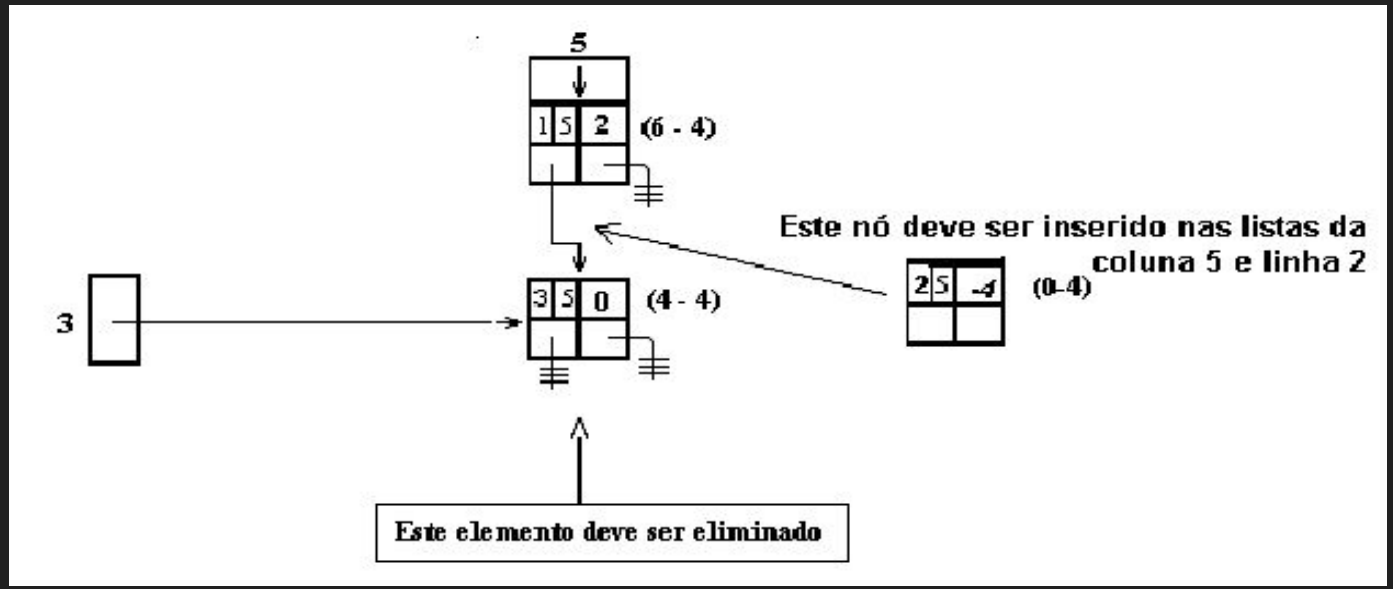
Operações

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Operações

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- P.ex., somar -4 à coluna 5



Desempenho (Fator Espaço)

- Quando a representação por listas cruzadas é vantajosa em relação à representação convencional?
- Fator Espaço. Suponhamos
  - matriz esparsa que armazena inteiros
  - ponteiro ocupa o mesmo espaço de memória que um inteiro

Desempenho (Fator Espaço)

- Matriz Esparsa (Listas Cruzadas)
  - Espaço ocupado por matriz de *nl* linhas, *nc* colunas e *n* valores não-nulos
  - há ganho de espaço, quando um número inferior a *1/5* dos elementos da matriz forem não nulos
- Na representação bidimensional: espaço total: *nl x nc*

Desempenho (Fator Tempo)

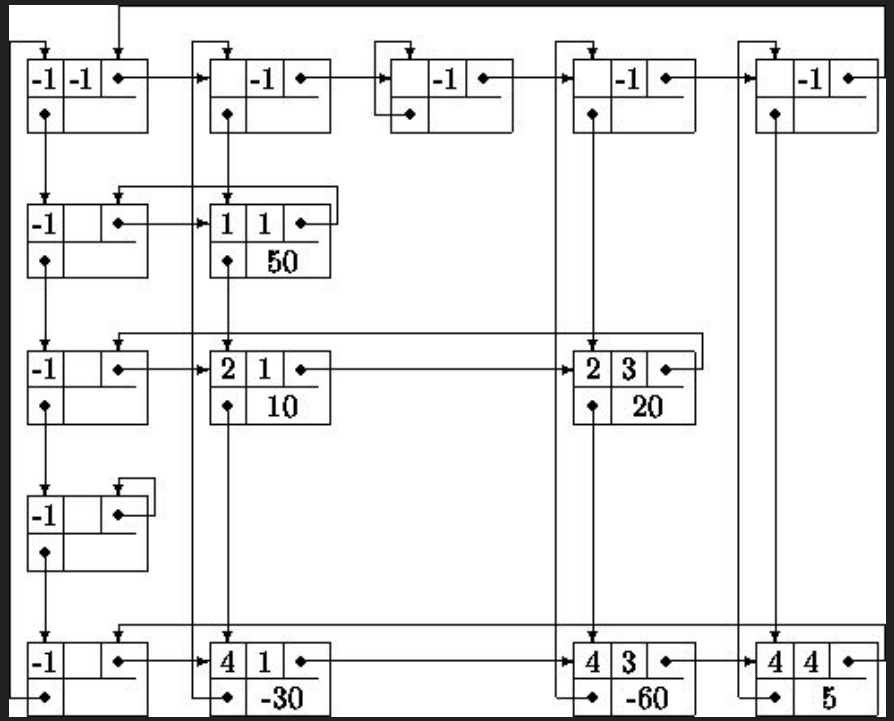
- As operações sobre listas cruzadas podem ser mais lentas e complexas do que para o caso bidimensional
- Portanto, para algumas aplicações, deve ser feita uma avaliação do compromisso entre tempo de execução e espaço alocado

Representação Alternativa – Listas Cruzadas Circulares

- Existem ocasiões nas quais não se sabe a princípio qual será o número máximo de linhas ou colunas da matriz esparsa
- Nessas situações, os vetores Coluna e Linha podem ser substituídos por listas ligadas circulares

Representação alternativa

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$



Exercícios

- Desenvolva procedimentos para (listas não circulares):
  - Atualizar o elemento a_{ij}
    - Modificar função “set” para não criar, desnecessariamente, um novo nó nesse caso.
  - Somar a constante c todos os elementos da coluna *j*
    - Pode resultar em inserção ou eliminação nas listas.

Referências

- Material baseado no originais produzidos pelos professores Rudinei Gularte, Gustavo E. de A. P. A. Batista e Fernando V. Paulovich
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e seus Algoritmos, Livros Técnicos e Científicos, 1994.
- TENEMBAUM, A.M., e outros Data Structures Using C, Prentice-Hall, 1990.
- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.