Matrizes Esparsas

O Problema

- Representação de matrizes com muitos elementos 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
  - Por exemplo, 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
  - Por exemplo, matriz abaixo, de 5 linhas por 6 colunas: apenas 5 dos 30 elementos são não nulos
  - Precisamos de uma representação que evite o armazenamento de tantos zeros.
  - Solução: utilizar listas cruzadas como estruturas de dados

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

O Problema

- Representação de matrizes com muitos elementos nulos
  - Por exemplo, matriz abaixo, de 5 linhas por 6 colunas: apenas 5 dos 30 elementos são não nulos
  - Precisamos de uma representação que evite o armazenamento de tantos zeros.
  - Solução: estrutura de lista encadeada contendo somente os elementos não nulos

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

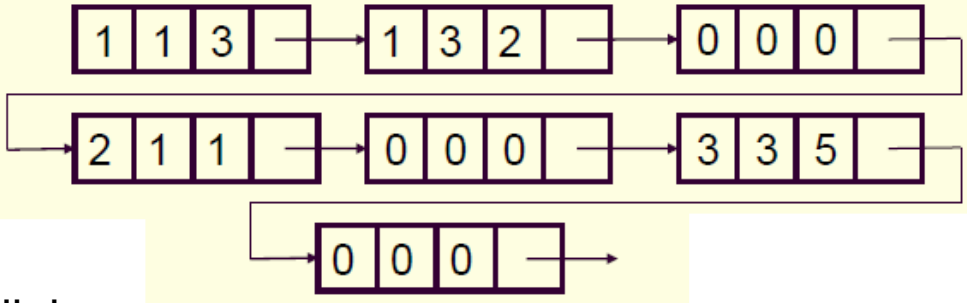
Solução 1

- Listas simples encadeadas

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

linha	coluna	valor	prox

Estrutura de um nó:
- linha, coluna: posição
- valor: ≠ zero
- prox: próximo nó



Nós zerados opcionais
para auxiliar na divisão de linhas

Solução 1

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

O Problema

- Representação de matrizes com muitos elementos nulos
  - Por exemplo, matriz abaixo, de 5 linhas por 6 colunas: apenas 5 dos 30 elementos são não nulos
  - Precisamos de uma representação que evite o armazenamento de tantos zeros.
  - Solução: utilizar listas cruzadas como estruturas de dados

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

Representação por listas cruzadas

- Cada elemento identificado pela sua coluna e valor

Representação por listas cruzadas

- Cada elemento identificado pela sua coluna e valor
- Cada elemento aij não-nulo pertence a uma lista de valores não nulos da linha *i* também a uma lista de valores não nulos da linha *j*

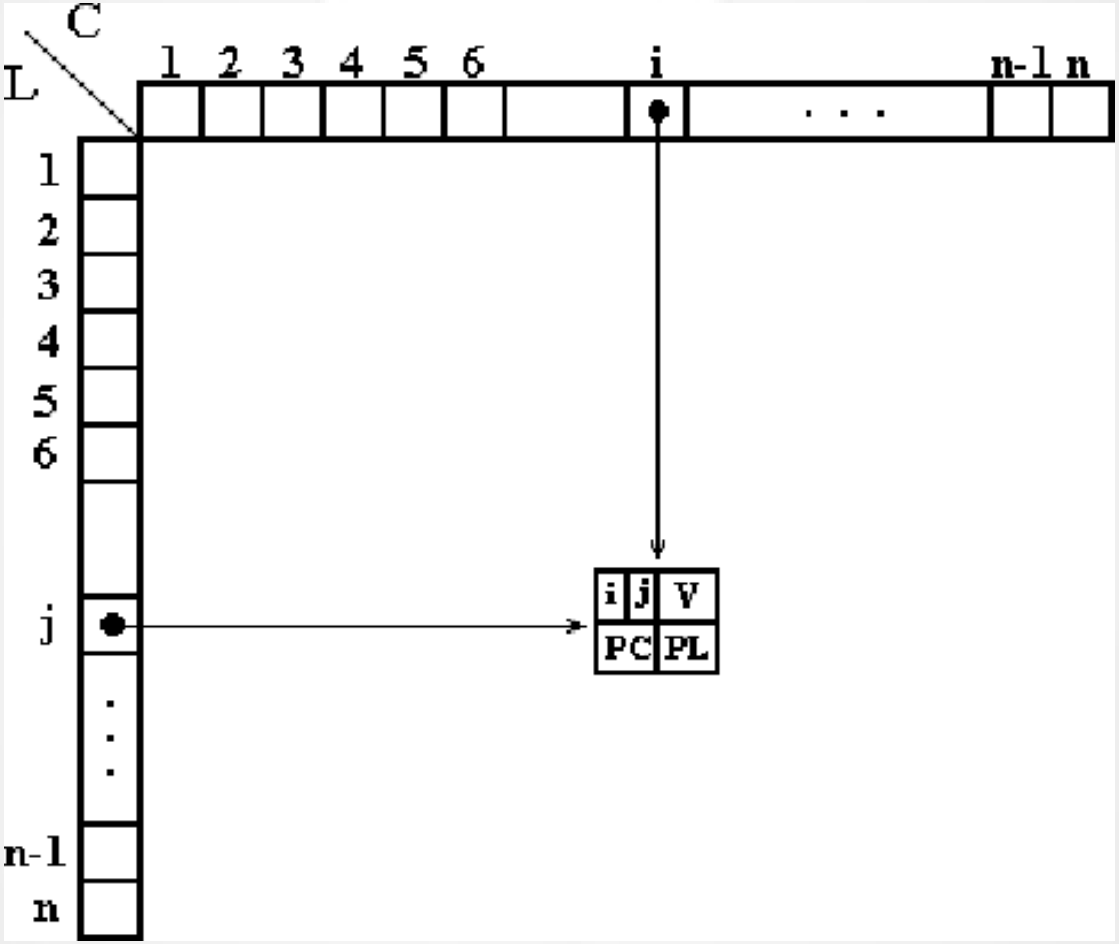
Representação por listas cruzadas

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

Representação por listas cruzadas

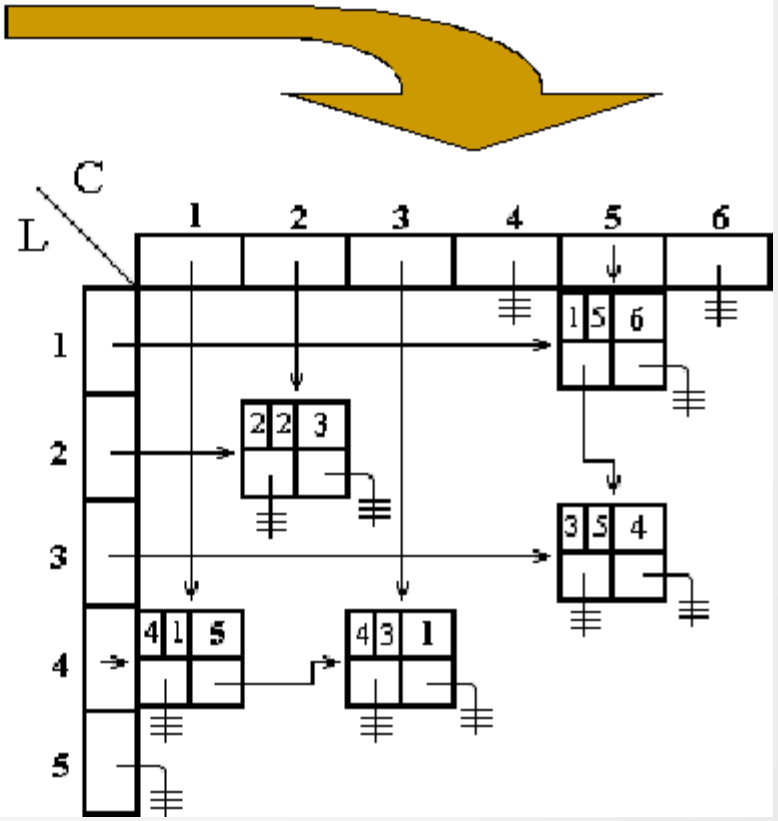
- Cada elemento identificado pela sua, coluna, e valor
- Cada elemento aij não-nulo pertence a uma lista de valores não nulos da linha *i* também a uma lista de valores não nulos da linha *j*
- Assim, para matriz *nl* linhas e *nc* colunas, teremos *nl* listas de linhas e *nc* listas de colunas
- Cada elemento não nulo é mantido simultaneamente em duas listas
  - Uma para sua linha
  - Uma para sua coluna

Listas Cruzadas



Listas Cruzadas

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



TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas

TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas
- Operações principais
  - criar_matriz(M) : cria uma nova matriz esparsa M vazia

TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas
- Operações principais
  - criar_matriz(M) : cria uma nova matriz esparsa M vazia
  - inserir(M, lin, col, valor): insere um valor na posição (lin,col) da matriz esparsa M

TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas
- Operações principais
  - criar_matriz(M) : cria uma nova matriz esparsa M vazia
  - inserir(M, lin, col, valor): insere um valor na posição (lin,col) da matriz esparsa M
  - Remover (M, lin, col): remove o valor na posição (lin,col) da matriz esparsa M

TAD Matriz Esparsa

- Pode-se criar um TAD bastante simples para matrizes esparsas
- Operações principais
  - criar_matriz(M) : cria uma nova matriz esparsa M vazia
  - inserir(M, lin, col, valor): insere um valor na posição (lin,col) da matriz esparsa M
  - remover (M, lin, col): remove o valor na posição (lin,col) da matriz esparsa M
  - consultar (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)

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

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 R

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 R
  - somar_coluna (M,V,Col): Soma uma constante V a todos os elementos da coluna col da Matriz 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 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

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 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..

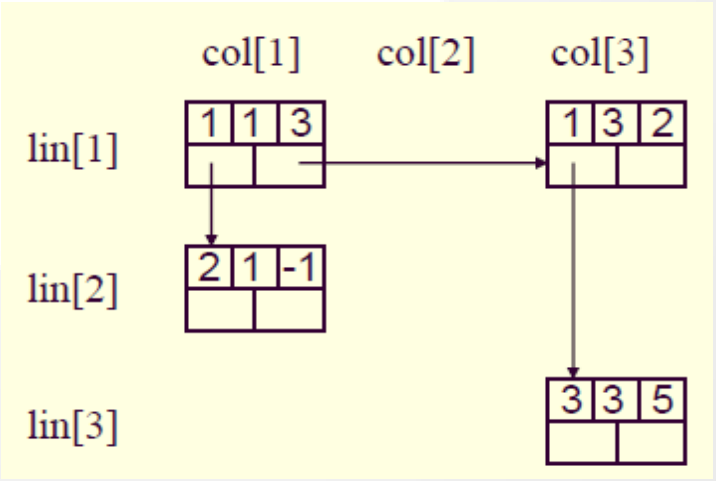
Estrutura de Dados – solução 1

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

Estrutura do nó

Linha	Coluna	Valor
Ponteiro p/Abaixo		Ponteiro p/Direita

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ -1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$



Estrutura de Dados – solução 1

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

```
# define NLINHAS  5
# define NCOLUNAS 6
```

```
typedef struct {
      int valor;
} DADO;
```

```
typedef struct CELULA {
    int lin;
    int col;
    DADO dado;
    struct CELULA *direira
    struct CELULA *abaixo;
} tCELULA;
```

```
typedef struct {
 tCELULA *linha[NLINHAS];
 tCELULA *coluna[NCOLUNAS];
} MatrizEsparsa;
```

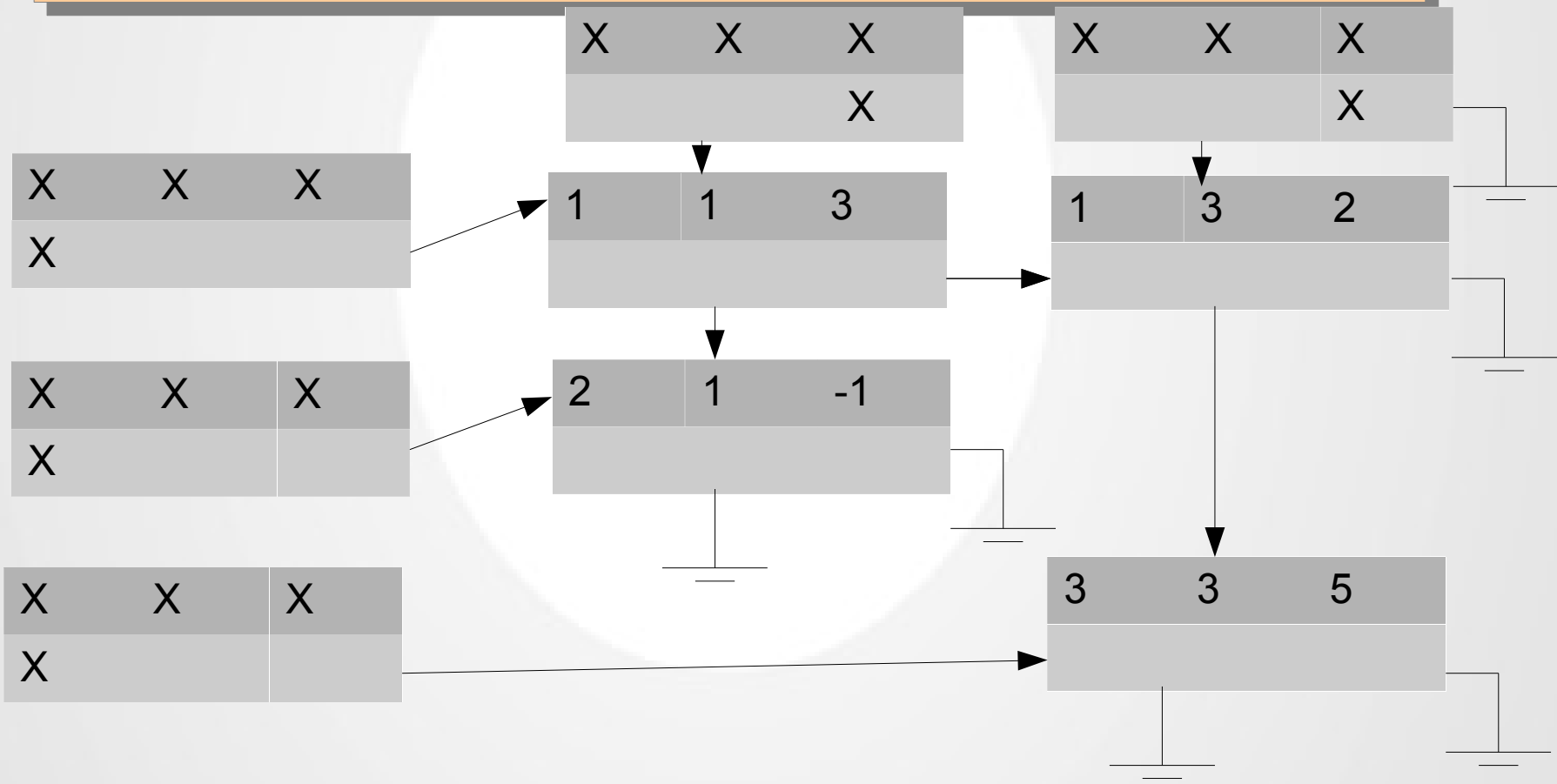
Estrutura de Dados – solução 1

- 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

- Em termos de tempo
  - Operações mais lentas em listas cruzadas: acesso não é direto

Estrutura de Dados - Solução 2

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



Estrutura de Dados - Solução 2

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

```
# define NLINHAS  5
# define NCOLUNAS 6
```

```
typedef struct {
      int valor;
} DADO;
```

```
typedef struct CELULA {
    int lin;
    int col;
    DADO dado;
    struct CELULA *direira
    struct CELULA *abaixo;
} tCELULA;
```

```
typedef struct {
 tCELULA linha[NLINHAS];
 tCELULA coluna[NCOLUNAS];
} MatrizEsparsa;
```

Operações

- Vamos implementar as operações criar_matriz(...), limpar_matriz(...), inserir(...), remover(...) e consultar(...) do conjunto de operações principais

Operações

- Vamos implementar as operações criar_matriz(...), limpar_matriz(...), inserir(...), remover(...) e consultar(...) do conjunto de operações principais

- As demais operações principais e auxiliares ficam como exercício.

Operações

- Vamos implementar as operações criar_matriz(...), limpar_matriz(...), inserir(...), remover(...) e consultar(...) do conjunto de operações principais

- As demais operações principais e auxiliares ficam como exercício.

- Entretanto, vamos discutir alguns aspectos importantes dessas operações

Criar Matriz – Solução 1

```
void criar (MatrizEsparsa *matriz){
 int i;
 for (i=0;i<Nlinhas,i++){
     matriz->linha[i] = NULL;
  }
  for (i=0;i<Ncolunas,i++){
   matriz->coluna[i] = NULL;
  }
}
```

Criar Matriz – solução 2

```
void criar (MATRIZ *matriz){
 int i;
 for (i=0;i<Nlinhas,i++){
     matriz->linha[i].direita = NULL;
     matriz->linha[i].abaixo=NULL;
     matriz->linha[i].lin=-1;
     matriz->linha[i].col=-1;
  }
 for (i=0;i<Ncolunas,i++){
    matriz->coluna[i].direita = NULL;
    matriz->coluna[i].abaixo=NULL;
    matriz->coluna[i].lin=-1;
    matriz->coluna[i].col=-1;
 }
}
```

Limpar Matriz – Solução 1

```
void limpar_matriz (MATRIZ *matriz){
 int i;
 for (i=0;i<Nlinhas,i++){
  tCELULA *paux = matriz->linha[i];
  while (paux!=NULL)
      tCELULA *prem = paux;
      paux = paux->direita;
      free(prem);
   }
  }
}
```

Limpar Matriz – solução 2

```
void limpar_matriz (MATRIZ *matriz){
 int i;
 for (i=0;i<Nlinhas,i++){
    CELULA *paux = matriz->linha[i].direita;
    while (paux!=NULL)
       CELULA *prem = paux;
       paux = paux->direita;
       free(prem);
    }
  }
}
```

Inserir Valor – Solução 2

```
int inserir(MATRIZ *matriz, int lin, int col, DADO *dado){
    if (lin<NLINHAS && col<NCOLUNAS){
    //aponta para a posição anterior de inserção
    CELULA *paux = &matriz->linha[lin];
    //procurar posição de inserção
    while (paux->direita!=NULL&&paux->direita->col<col){
       paux = paux->direita;
    }
    //celula já preenchida na matriz
   if (paux->direita != NULL&&paux->direita->col==col){
      paux->direita->dado = *dado;
   }else{
     CELULA *pnovo = (CELULA *)malloc(sizeof(CELULA));
     pnovo->dado = *dado;
     pnovo->lin = lin;
     pnovo->col = col;
     pnovo->direita = paux->direita;
     paux->direita = pnovo;
     //inserir na coluna
     paux = &matriz->coluna[col];
     //procurar posição de inserção
     while (paux->abaixo !=NULL && paux->abaixo->lin<lin){
        paux = paux->abaixo;}
     pnovo->abaixo = paux->abaixo;
     paux->abaixo = pnovo;
    }
    return 1;
   }
   return 0;
}
```

Remove Valor – Solução 2

```
int remover(MATRIZ *matriz, int lin, int col){
  if (lin<NLINHAS && col<NLCOLUNAS){
    //aponta para posição anterior de remoção
    CELULA *paux = &matriz->linha[lin];
    //procurar posição de remoção na linha
    while (paux->direita !=NULL){
      if (paux->direita->col<col) paux = paux->direita;
      else break;
    }
    if (paux->direita != NULL){//bloco existe
        if (paux->direita->col == col){//verifica o local
          CELULA *prem = paux->direita;
          paux->direita = paux->direita->direita
          //procura posição de remoção na coluna
          CELULA *paux = &matriz->coluna[col];
          while(paux->abaixo!=NULL) {
           if (paux->abaixo->lin<lin) paux = paux ->abaixo;
          else break;
          }
          if (paux->abaixo->lin==lin){
            paux->abaixo = paux->abaixo->abaixo;
            free(prem);
          }else return 0; //linha incorreta
        }else return 0; //local incorreto
     }else return 0; //bloco não existe
     return 1;
   }
   return 0;
}
```

Remove Valor – Solução 2 (Modificada)

```
int remover(MATRIZ *matriz, int lin, int col){
  if (lin<NLINHAS && col<NLCOLUNAS){
    //aponta para posição anterior de remoção
    CELULA *paux = &matriz->linha[lin];
    //procurar posição de remoção na linha
    while (paux->direita !=NULL){
      if (paux->direita->col<col) paux = paux->direita;
      else break;
    }
    if (paux->direita != NULL){//bloco existe
        if (paux->direita->col == col){//verifica o local
          CELULA *prem = paux->direita;
          paux->direita = paux->direita->direita
          //procura posição de remoção na coluna
          CELULA *paux = &matriz->coluna[col];
          while(paux->abaixo!=NULL) {
           if (paux->abaixo->lin<lin) paux = paux ->abaixo;
          else break;
          }
          paux->abaixo = paux->abaixo->abaixo;
          free(prem);
        }else return 0; //local incorreto
     }else return 0; //bloco não existe
    return 1;
   }
   return 0;
}
```

Consulta Valor

```
int consultar(MATRIZ *matriz, int lin, int col,DADO *dado)
{
   if (lin<NLINHAS && col<NLCOLUNAS){
     CELULA *paux = &matriz->linha[lin].direita;
     while (paux != NULL){
        if (paux->col == col) {
            *dado = paux->dado;
            return 1;
        }
        paux = paux->direita;
    }
    dado->valor = 0;
    return 1;
   }
   return 0;
}
```

Operações

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

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

Operações

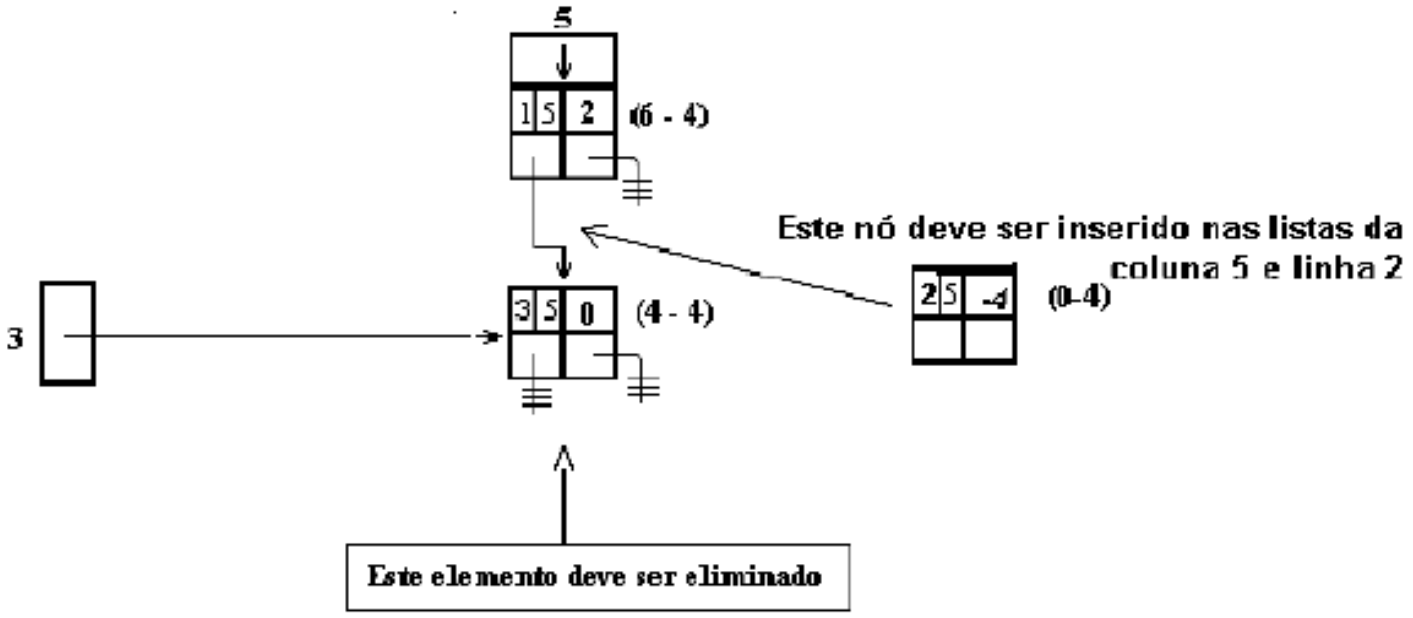
- E quando um elemento da matriz original se torna não nulo, em consequência de alguma operação? É necessário inserir na estrutura?
- E quando um elemento da matriz original se tornar nulo? É necessário eliminar da estrutura ?

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

Operações

- Por exemplo, somar -4 à coluna 5

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



Operações

- Após a realização de alguma operação sobre a matriz
  - Quando um elemento da matriz se torna nulo
    - Remoção do elemento
  - Quando algum elemento se torna não nulo
    - Inserção do elemento

Desempenho

- Quando a representação por listas cruzadas é vantajosa em relação à representação convencional?

Desempenho

- Quando a representação por listas cruzadas é vantajosa em relação à representação convencional?
- Fator Espaço. Suponhamos

Desempenho

- Quando a representação por listas cruzadas é vantajosa em relação à representação convencional?
- Fator Espaço. Suponhamos
  - matriz esparsa que armazena inteiros

Desempenho

- 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
- Matriz Esparsa (Listas Cruzadas)

Desempenho

- 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
- Matriz Esparsa (Listas Cruzadas)
  - Espaço ocupado por matriz de nl linhas, nc colunas e n valores não-nulos
    - 5n espaços para ponteiros (um para cada campo do registro: linha, coluna, valor, direita, abaixo)
    - 5nl espaços para ponteiros para o vetor linha
    - 5nc espaços para ponteiros para o vetor coluna
    - espaço total: 5n + 5nl + 5nc
- Na representação bidimensional: espaço total: nl x nc

Desempenho (Fator Espaço)

- Conclusão
  - Em termos de espaço ocupado, é vantajoso utilizar a representação de listas cruzadas quando
    - $5n + 5nl + 5nc < nl \times nc$
    - ou seja, quando $n < [(nl - 5) \times (nc - 5) - 25]/5$
    - Como $(nl-5)x(nc-5)$ é aproximadamente o tamanho da matriz, pode-se dizer, de uma maneira geral, que há ganho de espaço, quando um número inferior a 1/5 dos elementos da matriz forem não nulos

Desempenho (Fator Espaço)

- As operações sobre listas cruzadas podem ser mais lentas e complexas do que para o caso bidimensional

Desempenho (Fator Espaço)

- 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

Representação alternativa

- Existem ocasiões nas quais não se sabe a princípio qual será o número máximo de linhas ou colunas da matriz esparsa

Representação alternativa

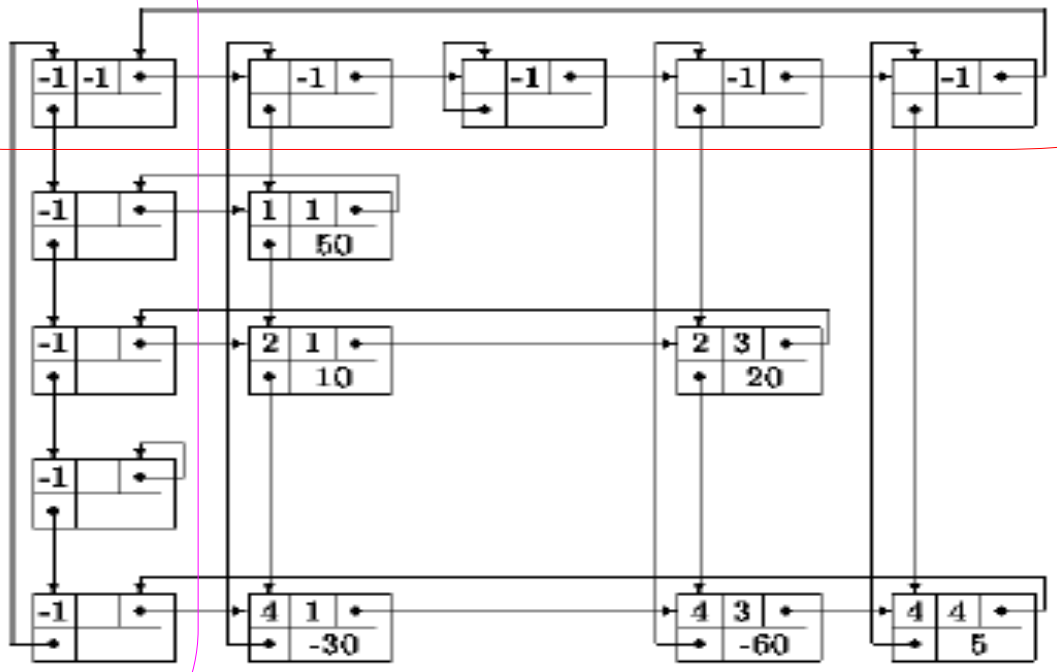
- Existem ocasiões nas quais NÃO se sabe a princípio qual será o número máximo de linhas ou colunas da matriz esparsa
- Nessas situações, os vetores Coluna e Linha podem ser substituídos por listas ligadas circulares

Representação alternativa

- Listas circulares com nós de cabeçalho
  - Ao invés de vetores de ponteiros, linhas e colunas são listas circulares com nós de cabeçalho
    - Nós de cabeçalho: reconhecidos por um -1 no campo linha ou coluna
      - 1 único ponteiro para a matriz: navegação em qualquer sentido

Representação alternativa

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



Por que?

Estrutura de Dados

```
typedef struct CELULA {
    int lin;
    int col;
    DADO dado;
    struct CELULA *direira
    struct CELULA *abaixo;
} tCELULA;
```

```
typedef struct {
      int valor;
} DADO;
```

```
typedef struct {
 tcelula *A;
}MatrizEsparsa;
```

Exercícios

- Representar a matriz abaixo com listas circulares com nós de cabeçalho

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ -1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

- Implementar em C uma sub-rotina que some todos os elementos de uma matriz qualquer representada dessa forma

Exercícios

- Desenvolva procedimentos para

Exercícios

- Desenvolva procedimentos para
  - Acessar o elemento aij

Exercícios

- Desenvolva procedimentos para
  - Acessar o elemento aij
  - Eliminar aij da matriz

Exercícios

- Desenvolva procedimentos para
  - Acessar o elemento aij
  - Eliminar aij da matriz
  - Somar a constante c todos os elementos da coluna j

Exercícios

- Desenvolva procedimentos para
  - Acessar o elemento aij
  - Eliminar aij da matriz
  - Somar a constante c todos os elementos da coluna j
    - pode resultar em inserção ou eliminação nas listas.

Matrizes esparsas -Lista Circular

- Quais as desvantagens dessa representação?
  - Mais complexa de se manipular
- Quais as vantagens dessa representação?
  - A matriz pode crescer dinamicamente