

Estruturas de Dados

Módulo 6 – Matrizes

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO



Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,
Introdução a Estruturas de Dados, Editora Campus
(2004)

Capítulo 6 – Matrizes

Tópicos

- Alocação estática versus dinâmica
- Vetores bidimensionais – matrizes
- Matrizes dinâmicas
- Operações com matrizes
- Representação de matrizes simétricas

Alocação Estática versus Dinâmica

- Alocação estática de vetor:
 - é necessário saber de antemão a dimensão máxima do vetor
 - variável que representa o vetor armazena o endereço ocupado pelo primeiro elemento do vetor
 - vetor declarado dentro do corpo de uma função não pode ser usado fora do corpo da função

```
#define N 10  
int v[N];
```

Alocação Estática versus Dinâmica

- Alocação dinâmica de vetor:
 - dimensão do vetor pode ser definida em tempo de execução
 - variável do tipo ponteiro recebe o valor do endereço do primeiro elemento do vetor
 - área de memória ocupada pelo vetor permanece válida até que seja explicitamente liberada (através da função free)
 - vetor alocado dentro do corpo de uma função pode ser usado fora do corpo da função, enquanto estiver alocado

```
int* v;  
...  
v = (int*) malloc(n * sizeof(int));
```

Alocação Estática versus Dinâmica

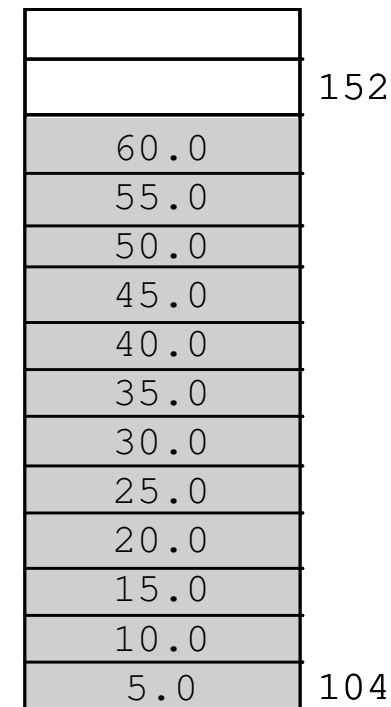
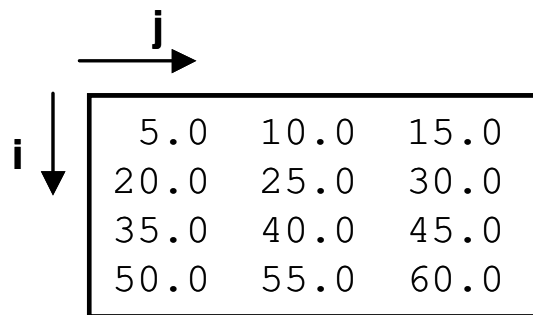
- Função “realloc”:
 - permite re-alocar um vetor preservando o conteúdo dos elementos, que permanecem válidos após a re-alocação
 - Exemplo:
 - m representa a nova dimensão do vetor

```
v = (int*) realloc(v, m*sizeof(int));
```

Vetores Bidimensionais - Matrizes

- Vetor bidimensional (ou matriz):

```
float m[4][3] = {{ 5.0,10.0,15.0},  
                 {20.0,25.0,30.0},  
                 {35.0,40.0,45.0},  
                 {50.0,55.0,60.0}};
```



Vetores Bidimensionais - Matrizes

- Vetor bidimensional (ou matriz):
 - declarado estaticamente
 - elementos acessados com indexação dupla `m[i][j]`
 - i acessa a linha e j acessa a coluna
 - indexação começa em zero:
 - `m[0][0]` é o elemento da primeira linha e primeira coluna
 - variável representa um ponteiro para o primeiro “vetor-linha”
 - matriz também pode ser inicializada na declaração

Vetores Bidimensionais - Matrizes

- Passagem de matrizes para funções:

declaração da matriz na função principal:

```
float mat[4][3]
```

protótipo da função (opção 1): parâmetro declarado como “vetor-linha”

```
void f (... , float (*mat)[3], ...);
```

protótipo da função (opção 2): parâmetro declarado como matriz, omitindo o número de linhas

```
void f (... , float mat[ ][3], ...);
```

Matrizes Dinâmicas

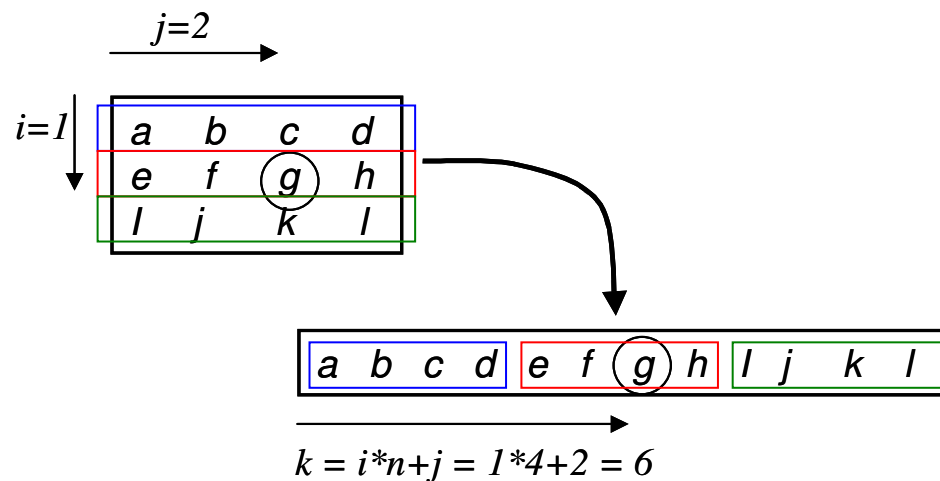
- Limitações:
 - alocação estática de matriz:
 - é necessário saber de antemão suas dimensões
 - alocação dinâmica de matriz:
 - **C só permite alocação dinâmica de conjuntos unidimensionais**
 - é necessário criar **abstrações conceituais** com vetores para representar matrizes (alocadas dinamicamente)

Matrizes Dinâmicas

- Matriz representada por um vetor simples:
 - conjunto bidimensional representado em vetor unidimensional
 - estratégia:
 - primeiras posições do vetor armazenam elementos da primeira linha
 - seguidos dos elementos da segunda linha, e assim por diante
 - exige disciplina para acessar os elementos da matriz

Matrizes Dinâmicas

- Matriz representada por um vetor simples (cont.):
 - matriz mat com n colunas representada no vetor v :
 - $mat[i][j]$ mapeado em $v[k]$ onde $k = i * n + j$



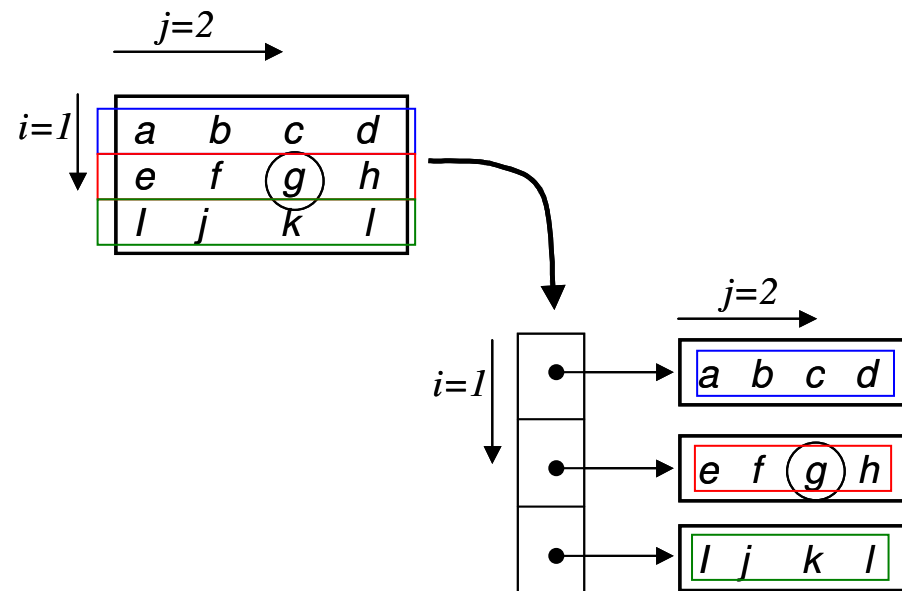
Matrizes Dinâmicas

- Matriz representada por um vetor simples (cont.):
 - `mat[i][j]` mapeado em `v[i * n + j]`

```
float *mat;    /* matriz m x n representada por um vetor */  
...  
mat = (float*) malloc(m*n*sizeof(float));
```

Matrizes Dinâmicas

- Matriz representada por um vetor de ponteiros:
 - cada elemento do vetor armazena o endereço do primeiro elemento de cada linha da matriz



Operações com Matrizes

- Exemplo – função **transposta**:
 - entrada: **mat** matriz de dimensão $m \times n$
 - saída: **trp** transposta de **mat**, alocada dinamicamente
 - Q é a *matriz transposta* de M se e somente se $Q_{ij} = M_{ji}$
- Solução 1: matriz alocada como vetor simples
- Solução 2: matriz alocada como vetor de ponteiros

```

/* Solução 1: matriz alocada como vetor simples */
float* transposta (int m, int n, float* mat)
{
    int i, j;
    float* trp;

    /* aloca matriz transposta com n linhas e m colunas */
    trp = (float*) malloc(n*m*sizeof(float));

    /* preenche matriz */
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            trp[ j*m+i ] = mat[ i*n+j ];

    return trp;
}

```

```

/* Solução 2: matriz alocada como vetor de ponteiros */
float** transposta (int m, int n, float** mat)
{
    int i, j;
    float** trp;

    /* aloca matriz transposta com n linhas e m colunas */
    trp = (float**) malloc(n*sizeof(float*));
    for (i=0; i<n; i++)
        trp[i] = (float*) malloc(m*sizeof(float));

    /* preenche matriz */
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            trp[ j ] [ i ] = mat[ i ][ j ];

    return trp;
}

```

Representação de Matrizes Simétricas

- Matriz simétrica $n \times n$:

- mat é uma matriz simétrica sse
 $mat[i][j] = mat[j][i]$

- estratégia de armazenamento:

- basta armazenar os elementos da diagonal e metade dos elementos restantes, por exemplo, os elementos abaixo da diagonal, ou seja, $mat[i][j]$, onde $i > j$

- em lugar de n^2 valores, armazena-se apenas s valores, onde:

$$s = 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

(1 elemento da primeira linha, 2 elementos da segunda, 3 da terceira, ...)

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 2 & 4 & 6 \\ 5 & 4 & 8 & 9 \\ 7 & 6 & 9 & 10 \end{bmatrix}$$

Representação de Matrizes Simétricas

- Exemplo:
 - função para criar matriz quadrada simétrica
 - função para, dada uma matriz já criada, acessar seus elementos
 - isola o fato da matriz não estar explicitamente toda armazenada
 - permite teste adicional para evitar acessos inválidos:
 - verifica se os índices representam elementos da matriz

Representação de Matrizes Simétricas

- Solução 1: matriz alocada como vetor simples
 - função para criar matriz quadrada simétrica
 - conforme discutido anteriormente
 - função para acessar os elementos da matriz
 - acesso a elemento `mat[i,j]` acima da diagonal ($i < j$):
 - retorna elemento simétrico abaixo da diagonal
 - acesso a elemento `mat[i,j]` abaixo da diagonal ($i > j$):
 - salta-se os elementos das linhas superiores, ou seja, $1+2+\dots+i = i*(i+1)/2$ elementos
 - utiliza-se o índice j para acessar a coluna
 - acesso a elemento `mat[i,i]` na diagonal:
 - retorna o elemento representado (como acima)

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 2 & 4 & 6 \\ 5 & 4 & 8 & 9 \\ 7 & 6 & 9 & 10 \end{bmatrix}$$

```
/* Solução 1: matriz alocada como vetor simples */  
float* cria (int n)  
{  
    int s = n*(n+1)/2;  
    float* mat = (float*) malloc(s*sizeof(float));  
    return mat;  
}
```

```

/* Solução 1: matriz alocada como vetor simples */
float acessa (int n, float* mat, int i, int j)
{
    int k; /* índice do elemento no vetor */

    if (i<0 || i>=n || j<0 || j>=n) {
        printf("Acesso inválido!\n");
        exit(1);
    }
    if (i>=j)
        k = i*(i+1)/2 + j; /* acessa elemento representado */
    else
        k = j*(j+1)/2 + i; /* acessa elemento simétrico */
    return mat[k];
}

```

Representação de Matrizes Simétricas

- Solução 2: matriz alocada como vetor de ponteiros
 - função para criar matriz quadrada simétrica
 - primeira linha representada por um vetor de 1 elemento, segunda linha representada por um vetor de 2 elementos e assim por diante
 - função para acessar os elementos da matriz
 - acesso aos elementos é natural
 - cuidado de não acessar diretamente elementos que não estejam explicitamente alocados (isto é, elementos com $i < j$)

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 2 & 4 & 6 \\ 5 & 4 & 8 & 9 \\ 7 & 6 & 9 & 10 \end{bmatrix}$$

```
/* Solução 2: matriz alocada como vetor de ponteiros */  
float** cria (int n)  
{  
    int i;  
    float** mat = (float**) malloc(n*sizeof(float*));  
    for (i=0; i<n; i++)  
        mat[i] = (float*) malloc((i+1)*sizeof(float));  
    return mat;  
}
```

```
/* Solução 2: matriz alocada como vetor de ponteiros */
float acessa (int n, float** mat, int i, int j)
{
    if (i<0 || i>=n || j<0 || j>=n) {
        printf("Acesso inválido!\n");
        exit(1);
    }
    if (i>=j)
        return mat[ i ] [ j ]; /* acessa elemento representado */
    else
        return mat[ j ][ i ]; /* acessa elemento simétrico */
}
```

Resumo

Matriz representada por vetor bidimensional estático:
elementos acessados com indexação dupla `m[i][j]`

Matriz representada por um vetor simples:
conjunto bidimensional representado em vetor unidimensional

Matriz representada por um vetor de ponteiros:
cada elemento do vetor armazena o endereço
do primeiro elemento de cada linha da matriz

Função adicional para gerência de memória:
`realloc` permite re-alocar um vetor preservando o conteúdo dos
elementos, que permanecem válidos após a re-alocação