

Programação de Computadores II

Cap. 6 – Matrizes

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

Slides adaptados dos originais dos profs.: Marco Antonio Casanova e Marcelo Gattass (PUC-Rio)

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

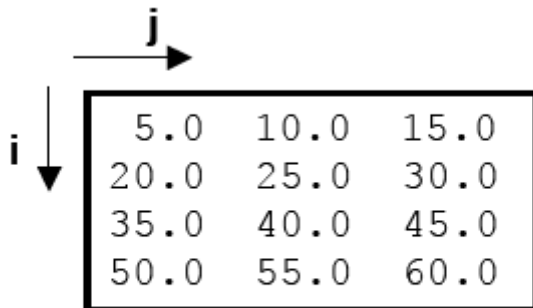
- Função “realloc”:
 - permite re-alocar um vetor preservando o conteúdo dos elementos, que permanecem válidos após a re-alocação
 - Exemplo:
 - m representa a nova dimensão do vetor

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

Vetores Bidimensionais - Matrizes

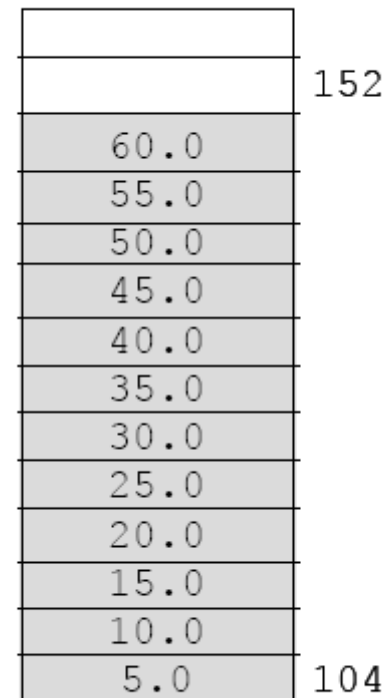
- Vetor bidimensional (ou matriz):

```
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}};
```



A diagram showing a 2D array with row index *i* (vertical arrow pointing down) and column index *j* (horizontal arrow pointing right). The array contains the following values:

5.0	10.0	15.0
20.0	25.0	30.0
35.0	40.0	45.0
50.0	55.0	60.0



A diagram showing a 1D array representing the 2D array in row-major order. The array contains the following values:

60.0
55.0
50.0
45.0
40.0
35.0
30.0
25.0
20.0
15.0
10.0
5.0

152

104

```
#include <stdio.h>

int main (void)
{
    int ms[2][3]={1,2,3}, {4,5,6} };
    int i,j;

    printf("Matrix:\n");
    for (i=0; i<2; i++) {
        for (j=0; j<3; j++)
            printf("%d  ",ms[i][j]);
        printf("\n");
    }
    return 0;
}
```

Matrix:

1 2 3

4 5 6

Press any key to continue

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

```
#include <stdio.h>

void showMat2x3(int a[2][3])
{
    int i,j;
    printf("Matrix:\n");
    for (i=0; i<2; i++) {
        for (j=0; j<3; j++)
            printf("%d  ",a[i][j]);
        printf("\n");
    }
}

int main (void)
{
    int ms[2][3]={{1,2,3}, {4,5,6} };

    showMat2x3(ms);
    return 0;
}
```

```
#include <stdio.h>

void showMat2x3(int a[][3])
{
    int i,j;
    printf("Matrix:\n");
    for (i=0; i<2; i++) {
        for (j=0; j<3; j++)
            printf("%d  ",a[i][j]);
        printf("\n");
    }
}

int main (void)
{
    int ms[2][3]={{1,2,3}, {4,5,6} };

    showMat2x3(ms);
    return 0;
}
```

```

#include <stdio.h>

void showMat2x3(int (*a)[3])
{
    int i,j;
    printf("Matrix:\n");
    for (i=0; i<2; i++) {
        for (j=0; j<3; j++)
            printf("%d  ",a[i][j]);
        printf("\n");
    }
}

int main (void)
{
    int ms[2][3]={{1,2,3}, {4,5,6} };

    showMat2x3(ms);
    return 0;
}

```

```
#include <stdio.h>

typedef int Matrix3[3][3];

void showMatrix3(Matrix3 a)
{
    int i,j;
    printf("Matrix:\n");
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++)
            printf("%d  ",a[i][j]);
        printf("\n");
    }
}

int main (void)
{
    Matrix3 ms={{1,2,3}, {4,5,6}, {7,8,9} };

    showMatrix3(ms);
    return 0;
}
```

Matrix:

1 2 3
4 5 6
7 8 9

Press any key to continue

Vetores Bidimensionais - Matrizes

- Passagem de matrizes para funções:

declaração da matriz na função principal:

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

protótipo da função (opção 0): parâmetro declarado como matriz

```
void f (... , 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

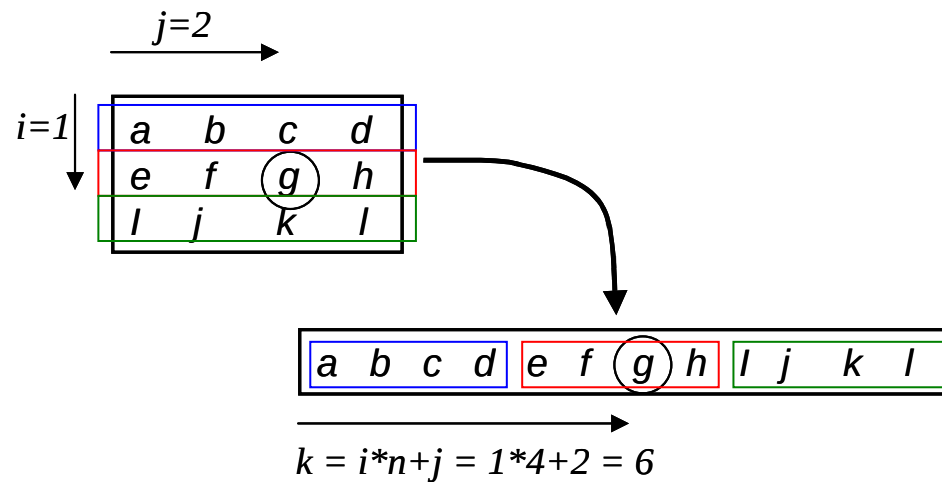
- Limitações:
 - alocação estática de matriz:
 - é necessário saber de antemão suas dimensões
 - alocação dinâmica de matriz:
 - C só permite alocação dinâmica de conjuntos unidimensionais
 - é necessário criar abstrações conceituais com vetores para representar matrizes (alocadas dinamicamente)

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 $k = i * n + j$



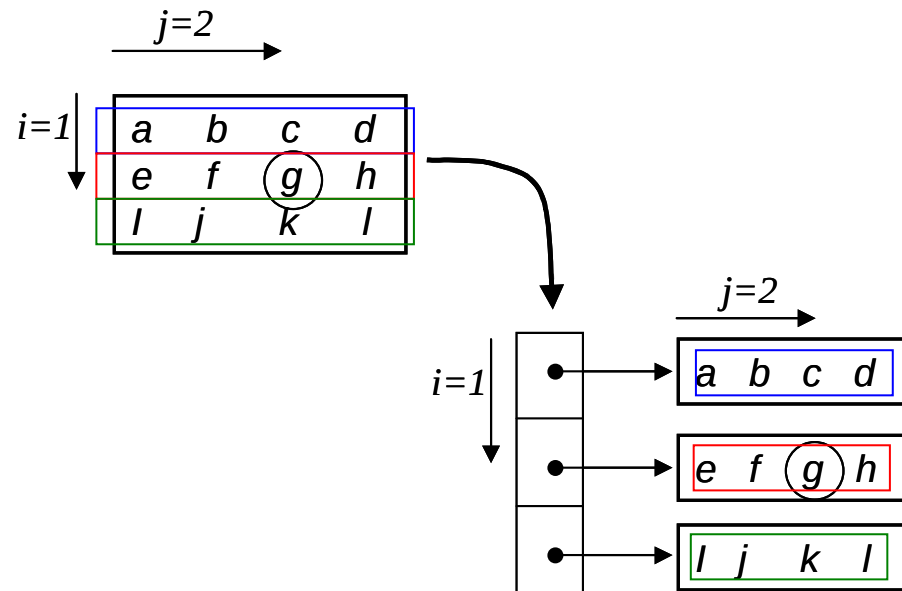
Matrizes Dinâmicas

- 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



```
float **allocMatrix(int m, int n)
{
    float **a;
    int i;

    a = (float **) malloc(m*sizeof(float *));
    if (a==NULL) exit(1);

    for (i=0;i<m;i++) {
        a[i]= (float*) malloc(n*sizeof(float));
        if (a[i]==NULL) exit(1);
    }
    return a;
}
```

```
float **freeMatrix(float **a,int m){  
    if (a!=NULL) {  
        int i;  
        for (i=0;i<m;i++)  
            if(a[i]!=NULL) free(a[i]);  
    }  
    free(a);  
    return NULL;  
}
```

```
void showMatrix(char * title, float **a, int m, int n)
{
    int i,j;
    printf("%s",title);
    for (i=0;i<m;i++){
        for (j=0;j<n;j++)
            printf("%12.6f ", a[i][j]);
        printf("\n");
    }
    printf("\n");
}
```

```

#include <stdio.h>
#include <stdlib.h>
...
int main( )
{
    float ** md = allocMatrix(2,3);
    int i,j;

    for (i=0; i<2; i++)
        for (j=0; j<3; j++)
            md[i][j]=(float) (i*3+j);

    showMatrix("Matriz md:",md,2,3);
    md=freeMatrix(md,2);
    return 0;
}

```

Matriz md:

0.000000	1.000000	2.000000
3.000000	4.000000	5.000000

Press any key to continue

Operações com Matrizes

- Exemplo – função **transposta**:
 - entrada: **mat** matriz de dimensão $n \times m$ ($n = \text{lin}$; $m = \text{col}$)
 - 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 n, int m, 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<n; i++)
        for (j=0; j<m; j++)
            trp[ j*m+i ] = mat[ i*m+j ];

    return trp;
}

```

```

/* Solução 2: matriz alocada como vetor de ponteiros */
float** transposta (int n, int m, 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<n; i++)
        for (j=0; j<m; j++)
            trp[ j ] [ i ] = mat[ i ][ j ];

    return trp;
}

```

Exercício

- Suponha uma matriz com n linhas e n números reais em cada linha, cada um dos valores representando o custo de comprar uma passagem da cidade i para a cidade j .
- Escreva uma função que recebe um vetor de índices (representando as cidades) e a matriz de custos e retorna o custo para passar por todas as cidades e retornar à cidade inicial.
- Escreva uma função main que utilize e teste essa função.

```
#include <stdio.h>
#include <stdlib.h>
```

```
float calcula_custo(int k, int* cidades, float** custos) {
    int i;
    float custo;
    custo = 0.0;
    for (i = 0; i < k-1; i++) {
        custo += custos[i][i+1];
    }
    if (k > 0) {
        custo += custos[k-1][0];
    }
    return custo;
}
```

```
int main(void) {  
    float** custos; int cidades[3]; int i, n; float custo;  
    n = 3;  
    custos = (float **)malloc(n*sizeof(float *));  
    if (custos == NULL) exit(1);  
    for (i = 0; i < n; i++) {  
        custos[i]= (float*)malloc(n*sizeof(float));  
        if (custos[i] == NULL) exit(1);  
    }  
    custos[0][0] = 0.0; custos[0][1] = 250.0; custos[0][2] = 100.0;  
    custos[1][0] = 300.0; custos[1][1] = 0.0; custos[1][2] = 500.0;  
    custos[2][0] = 120.0; custos[2][1] = 750.0; custos[2][2] = 0.0;  
    cidades[0] = 0; cidades[1] = 1; cidades[2] = 2;  
    custo = calcula_custo(n, cidades, custos);  
    printf("custo total: %f\n", custo);  
    return 0;  
}
```

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