Matriz Esparsa (parte 1)

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Sumário

- Matriz Esparsa
- Implementação de Matrizes Esparsas

Introdução

- Matriz (bidimensional)
  - Arranjo bidimensional: composto por m linhas e n colunas
  - Utilizada para representação de diversos tipos de dados
    - Dados numéricos
    - Imagens
    - etc

0	3	0	1
0	0	5	0
0	0	0	0
2	7	1	0

Introdução

- Representação de matrizes por alocação estática:

```
int mat[4][4];

mat[0][0] = 0;
mat[0][1] = 3;
mat[0][2] = 0;
mat[0][3] = 1;
mat[1][0] = 0;
mat[1][1] = 0;
...
mat[3][3] = 0;
```

ou

```
int mat[4][4] = {{0, 3, 0, 1}, {0, 0, 5, 0}, {0, 0,
0, 0}, {2, 7, 1, 0}};
```

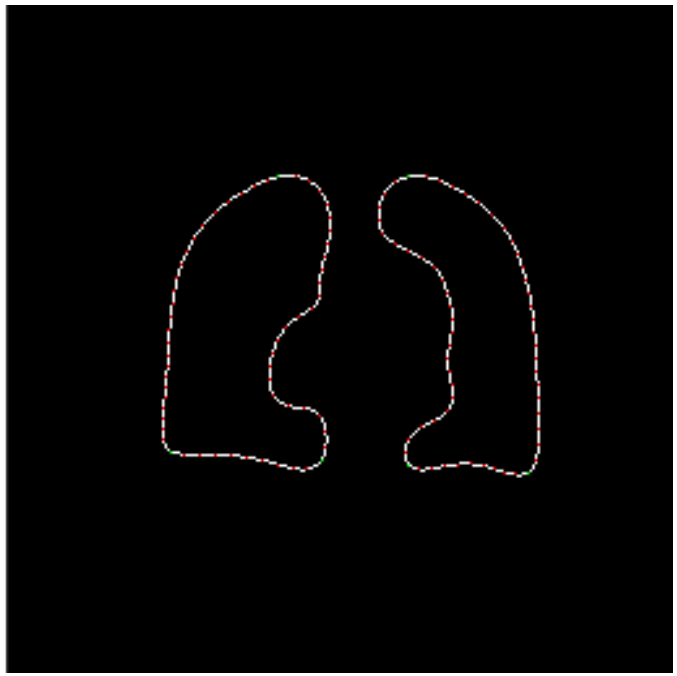
- Representação de matrizes por alocação dinâmica:

```
int i;
int **mat = (int**) malloc(4 * sizeof(int*));

for (i = 0; i < n; i++)
   mat[i] = (int*) malloc(4 * sizeof(int));

...
```

Matriz Esparsa

Matriz Esparsa

- Matriz em que a maioria dos elementos possui um valor padrão (0, por exemplo), ou a maioria dos valores são faltantes
  - Por exemplo: representar o contorno de uma imagem em preto e branco



Matriz Esparsa

- Nos exemplos do slide anterior seria um desperdício gastar $m \times n$ posições na memória para representá-los
  - Apenas uma pequena quantidade dos elementos tem um valor diferente de zero (preto)

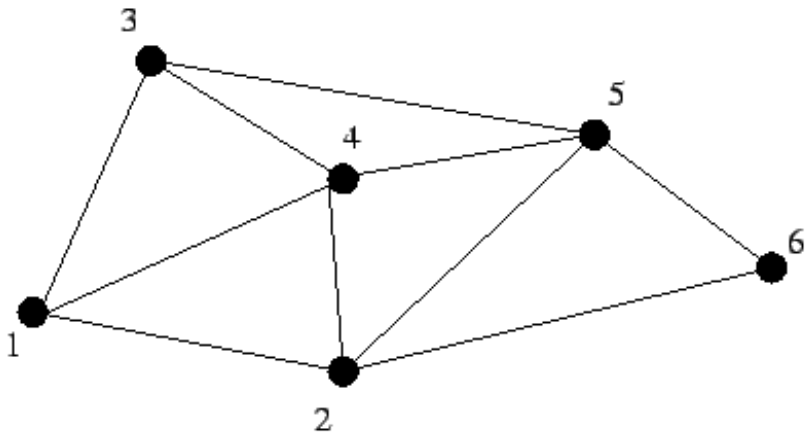
- Matrizes esparsas podem para evitar o desperdício de recursos computacionais (espaço e tempo para processamento)
  - Apenas elementos com valores diferentes de zero serão alocados

Matriz Esparsa

- Matrizes esparsas são aplicados em diversos problemas de:
  - Método das malhas para a resolução de circuitos elétricos
  - Sistemas de equações lineares
  - Armazenamento de dados (*e.g.* planilhas eletrônicas, mapas de bits)
  - Etc

Matriz Esparsa

- Grafos (serão vistos na disciplina AED2):

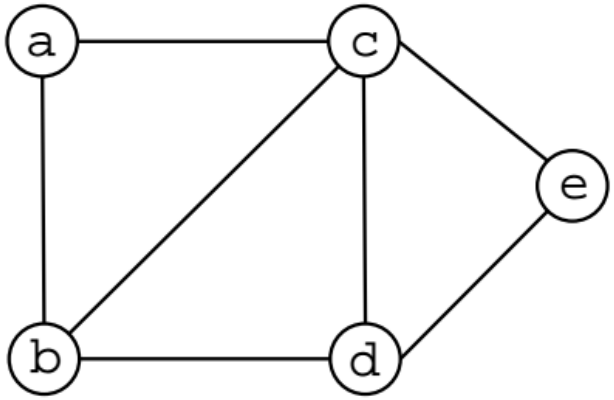


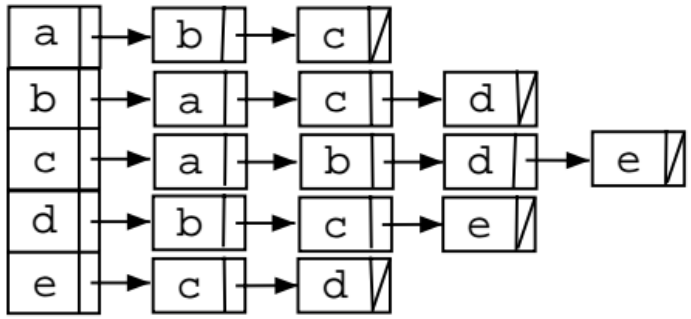
0	1	1	1	0	0
1	0	0	1	1	1
1	0	0	1	1	0
1	1	1	0	1	0
0	1	1	1	0	1
0	1	0	0	1	0

- Representação de grafos
  - Matriz de adjacência (figura acima)
  - **Lista de adjacência** (próximo slide)

Matriz Esparsa

- Exemplo de aplicação em grafos (serão vistos na disciplina AED2):





Matriz Esparsa

- Para representação de grafos, a matriz de adjacência pode acarretar em desperdício de memória
- Nesse caso, a lista de adjacência pode ser vantajosa para a representação de grafos "esparsos"
- A representação de matrizes esparsas podem ser representadas por meio de um vetor de listas encadeadas
  - Cada elemento desse vetor representaria uma linha da matriz
  - Em grafos, a lista de adjacência é similar à matriz esparsa

Implementação de Matrizes Esparsas

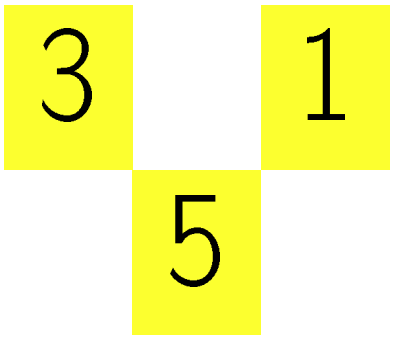
Implementação de Matrizes Esparsas

0	3	0	1
0	0	5	0
0	0	0	0
2	7	1	0

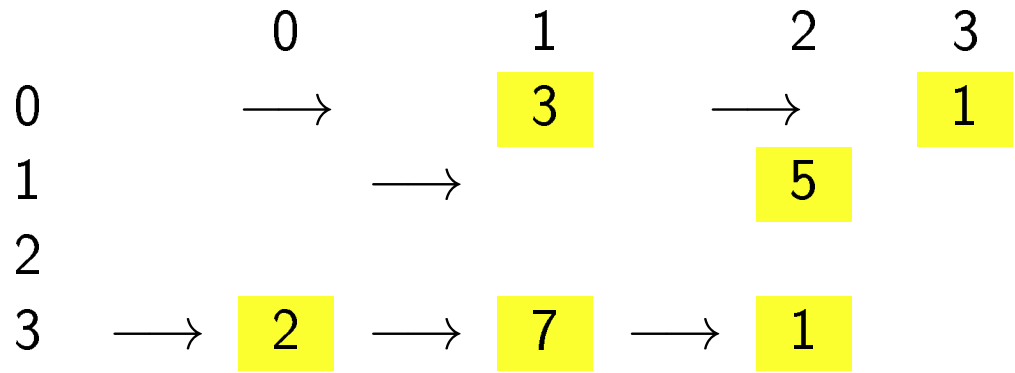
Implementação de Matrizes Esparsas

0	3	0	1
0	0	5	0
0	0	0	0
2	7	1	0

Implementação de Matrizes Esparsas



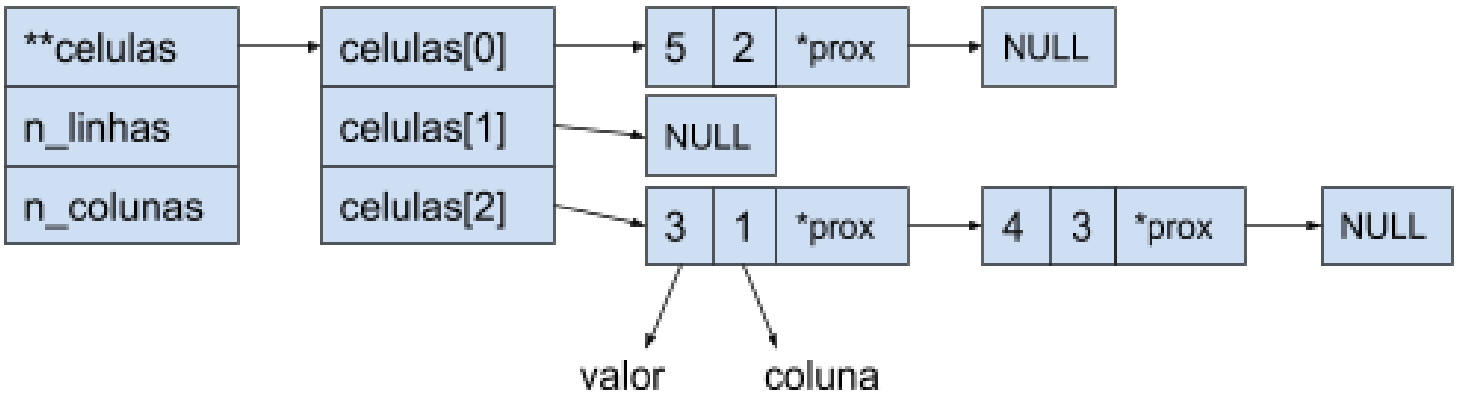
Implementação de Matrizes Esparsas



- Imagine cada linha da matriz como uma lista encadeada

Implementação de Matrizes Esparsas

	0	1	2	3
0	0	0	5	0
1	0	0	0	0
2	0	3	0	4



Implementação de Matrizes Esparsas

- Cada célula (*Cell*) contém:
  - A informação (*item*)
  - A coluna da matriz (*col*)
  - O ponteiro para a próxima célula (*next*)

```
typedef struct Cell Cell;

struct Cell{
   int item;
   int col;
   Cell *next;
};
```

Implementação de Matrizes Esparsas

- Representação de uma matriz esparsa:
  - Quantidade de linhas (*n_lin*)
  - Quantidade de colunas (*n_col*)
  - Arranjo de ponteiros do tipo *Cell* (*lin*)

```
typedef struct ListaE ListaE;

struct ListaE{
   Cell *head;
};

typedef struct Spa_Mat Spa_Mat;

struct Spa_Mat{
   int n_lin;
   int n_col;
   ListaE **lin;
};
```

Implementação de Matrizes Esparsas

- Operações básicas em uma matriz esparsa
  - Criar uma matriz esparsa
  - Buscar item
  - Inserir item
  - Remover item
  - Imprimir matriz

Implementação de Matrizes Esparsas

- A operação de busca é a mesma aplicada em listas encadeadas, mas ela pode ser na matriz inteira (ou seja, nas *n_lin* listas)

- A inserção é feita em ordem crescente e de acordo com o índice da coluna, considerando os seguintes casos:
  - Caso exista uma célula na posição e o valor a ser inserido for igual a zero, então a célula deve ser removida
  - Caso exista uma célula na posição e o valor a ser inserido for diferente de zero, então o valor existente deve ser substituído
  - Caso não exista uma célula na posição e o valor a ser inserido for diferente de zero, então uma nova célula é criada
  - Caso não exista uma célula na posição e o valor a ser inserido for igual a zero, então não é necessário fazer algo

Implementação de Matrizes Esparsas

- No caso de uma matriz esparsa, a operação de inserção e remoção é a mesma: depende do valor a ser inserido

- Para a impressão da matriz, para as "células inexistentes" é impresso 0 e para das demais células, é impresso os seus respectivos valores

Implementação de Matrizes Esparsas

- Implementação da operação criar:

```
Spa_Mat* criar(int l, int c){
   Spa_Mat* mat = malloc(sizeof(Spa_Mat));
   int i;

   mat->n_col = c;
   mat->n_lin = l;
   mat->lin = (ListaE**) malloc(sizeof(ListaE*) * l);

   for (i = 0; i < l; i++){
    mat->lin[i] = (ListaE*) malloc(sizeof(ListaE));

    mat->lin[i]->head = NULL;
   }

   return mat;
}
```

Implementação de Matrizes Esparsas

- Implementação da operação buscar:

```
static int procurar_lista(int item, ListaE *l){
    Cell *aux;

    if (l != NULL){
      aux = l->head;

      while ((aux != NULL) && (aux->item < item))
         aux = aux->next;
    }
    if ((aux != NULL) && (aux->item == item))
      return 1;
    else
      return 0;
}

int buscar(int item, Spa_Mat* mat){
    int i;
    int aux = 0;

    for (i = 0; (i < mat->n_lin) && (aux == 0); i++)
      aux = procurar_lista(item, mat->lin[i]);

    return aux;
}
```

Implementação de Matrizes Esparsas

- Implementação da operação inserir/remover: no TAD compartilhado no repositório da disciplina

- Implementação da operação imprimir:

```
void imprimir(Spa_Mat* mat){
    int i, j;
    Cell* aux;

    for (i = 0; i < mat->n_lin; i++){
      aux = mat->lin[i]->head;
      j = 0;

      while (aux != NULL){
        while (j < aux->col){
          printf("0 ");
          j++;
        }

        printf("%d ", aux->item);
        aux = aux->next;
      }
      for (j; j < mat->n_col; j++)
        printf("0 ");
      printf("\n");
    }
}
```

Implementação de Matrizes Esparsas

- Complexidade
  - Para criar uma matriz esparsa vazia, a complexidade é de $O(m)$, onde m é quantidade de linhas
  - Para criar uma matriz esparsa não vazia, a complexidade é de $O(m * n)$, onde n é quantidade de colunas
  - Para inserir/remover um item, a complexidade é de $O(n)$, ou seja, cada operação é proporcional a quantidade de itens guardados em cada linha
  - Dependendo da implementação da operação de busca, a complexidade pode ser na ordem de $O(m * n)$
  - Para imprimir, a complexidade é de $O(m * n)$

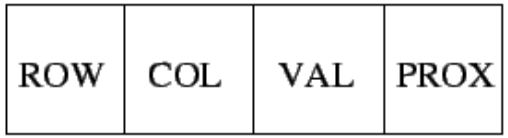
Implementação de Matrizes Esparsas

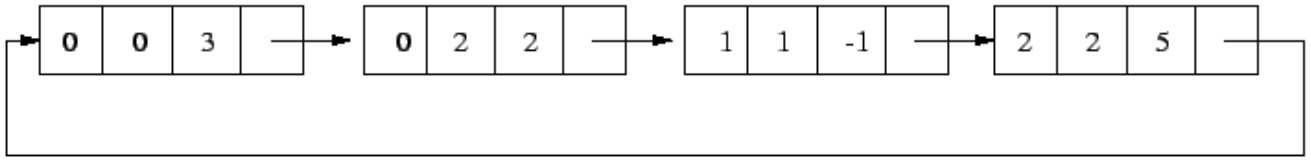
- Vantagens
  - Mantém a característica bidimensional da imagem (para a forma de representação desse tipo de matriz apresentada até o momento)
  - Economia de memória
- Desvantagem
  - Acesso sequencial

Implementação de Matrizes Esparsas

- Outras formas de representação
  - Listas duplamente encadeadas para cada linha em vez da simples
  - Uma só lista encadeada que guarda todos os itens não nulos

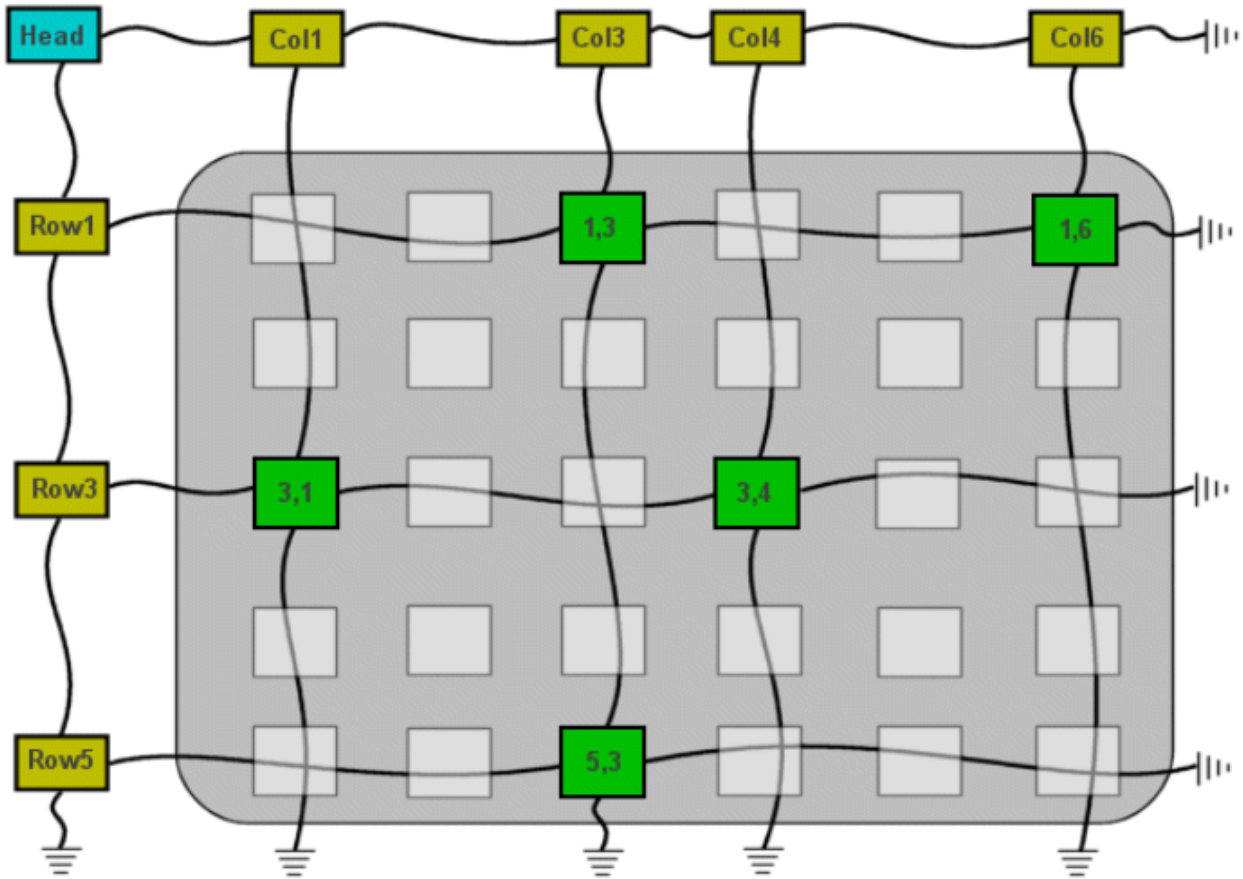
$$A_{3\times3} = \begin{bmatrix} 3 & 0 & 2 \\ 0 & -1 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$





Implementação de Matrizes Esparsas

- Outras formas de representação
  - Linhas e as colunas são representadas por listas encadeadas



Implementação de Matrizes Esparsas

- Exercício: implementar uma função que converta uma matriz de números inteiros em uma matriz esparsa.

Referências I

Oliva, J. T.
Matrizes Esparsas. AE22CP – Algoritmos e Estrutura de Dados I.
Notas de Aula. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2020.

Roman, N. T.; Digiampietri, L. A.
Matriz Esparsa. ACH2023 – Algoritmos e Estrutura de Dados I.
Notas de Aula. Sistemas de Informação. EACH/USP/São Paulo, 2018.