UDESC – Universidade do Estado de Santa Catarina

DCC – Departamento de Ciência da Computação

Curso: BCC

Disciplina: Teoria dos Grafos – TEG0001

Notas de Aula

Parte II

2.) **Representação computacional dos grafos**

Um grafo pode ser representado por alguns tipos de estruturas: **matriz de adjacência, matriz de incidência, listas e vetores**.

Matriz de adjacência: Suponha que um grafo tem **n** nós numerados $n_0, n_1, n_{k-1}, \ldots, n_k$. Esta numeração impõe uma ordem arbitrária nos nós, porém, não é dado significado algum ao fato de um nó aparecer antes do outro. Após a ordenação dos nós, podemos formar uma matriz N x N, onde o elemento i, j corresponde ao número de arcos entre os nós n_i e n_j. A matriz é denominada de **matriz adjacente A** do grafo em relação à ordem que foi estabelecida.

a_{ij} = p , se existem p arcos entre os nós n_i e n_j.

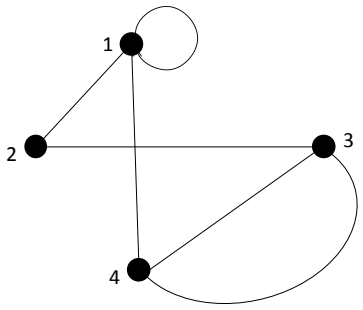
Seja G = (V,E)

A matriz de adjacência M=[mi,j] simétrica n x n armazena o relacionamento entre os vértices do grafo.

(mij), pode ser:

- **. 0 se i não é adjacente a j;**
- **. m, onde m representa a quantidade de arestas incidentes tanto em i quanto em j (com i ≠ j);**
- **. p, onde p é a quantidade de laços incidentes em i = j;**

Dado o grafo da figura 2.1, a matriz de adjacência é dada por uma matriz 4x4.



1			
1	**0**		
0	**1**	**0**	
1	**0**	**2**	**0**

Fig. 2.1 – matriz adjacente

Para todos os grafos não direcionados as respectivas matrizes adjacentes são simétricas, o que significa que precisamos preencher ou armazenar apenas os elementos pertencentes à diagonal principal e abaixo dela (arranjo triangular inferior).

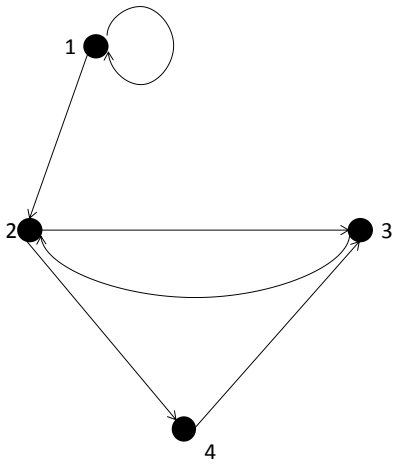
Uma matriz adjacente de um grafo simples G qualquer possui 0 na diagonal principal.

Em um **grafo direcionado**, a respectiva matriz de adjacência reflete o sentido de orientação dos arcos. Portanto:

a_{ij} = p , se existem p arcos de n_i , para n_j.

A matriz de adjacência de um grafo direcionado não será simétrica, pois a existência de um arco de n_i , para n_j, não implica a existência de um arco de n_j , para n_i.

Considere o grafo direcionado da figura 2.2. A matriz de adjacência A é dado por:



1	**1**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**0**	**1**	**0**

Fig. 2.2 – matriz adjacente p/ grafo direcionado

Matriz de incidência: uma matriz A de dimensão MxN é denominada de matriz de incidência de um grafo G = (N,M), quando:

A_{ki} = 1; se a aresta u_k tem origem no vértice i;

A_{ki} = -1; se i é o vértice de destino da aresta u_k;

A_{ki} = 0 ; se a aresta u_k não incide no vértice i;

As figuras abaixo apresentam respectivamente exemplos de matrizes de incidência para grafos não direcionados e direcionados.

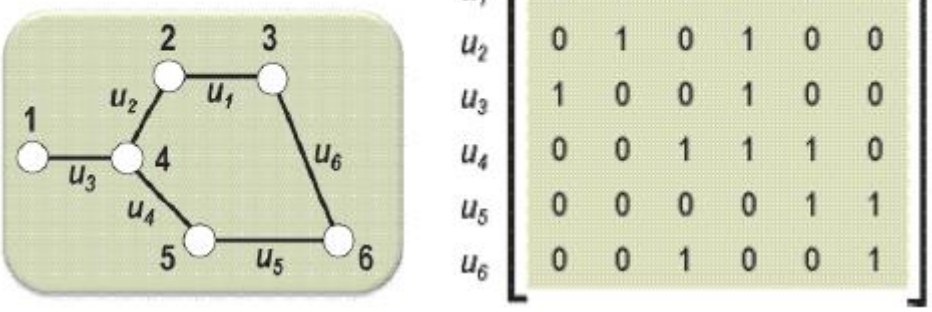


Fig. 2.2 – matriz incidência p/ grafo não direcionado

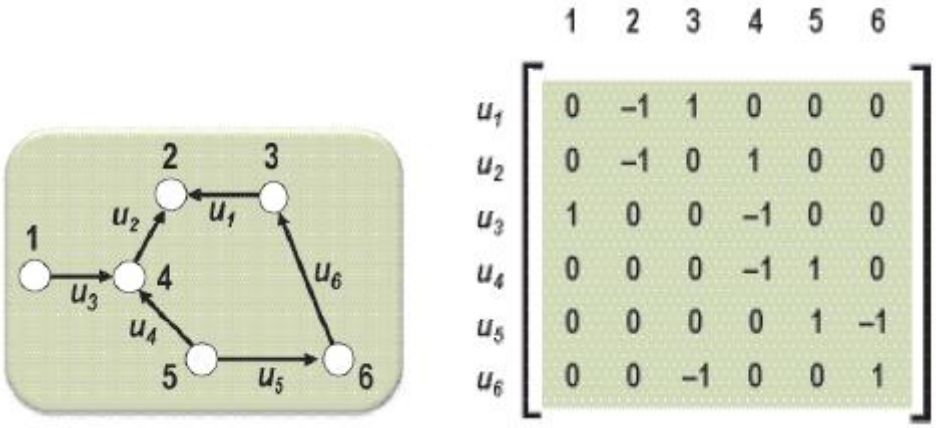


Fig. 2.3 – matriz incidência p/ grafo direcionado

Análise matriz adjacência:

. Para Grafos grandes (com muitos vértices) e esparsos (com poucas arestas), a representação na forma de matriz não torna-se interessante, pois, neste caso a matriz será composta principalmente por zeros, acarretando em um consumo maior e desnecessário de memória. Deve ser utilizada para grafos densos, onde |E| é próximo de $|V|^2$.

. O tempo necessário para acessar um elemento é independente de |V | ou |E|.

. É muito útil para algoritmos em que necessitamos saber com rapidez se existe uma aresta ligando dois vértices.

. A maior desvantagem é que a matriz necessita O(V^2) de espaço.

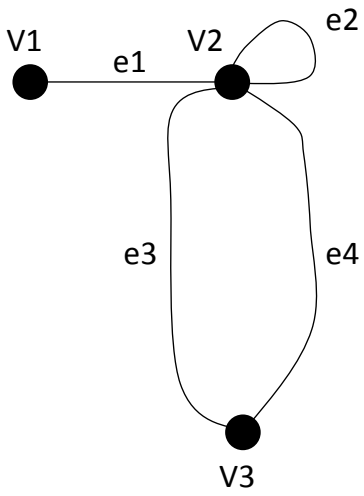
. Ler ou examinar a matriz tem complexidade de tempo O(V^2).

Exercício: Implemente um programa que dado um grafo da figura 2.1, apresente a matriz de adjacência e o grau de cada nó. Repita o procedimento par o grafo da figura 2.2 (direcionado).

Exercício: Implemente um programa que dado um grafo qualquer (que pode ser direcionado), apresente a matriz de adjacência e o grau de cada nó.

O tamanho ou comprimento de um caminho é o número de arestas do mesmo, ou seja, número de vértices menos um.

Por exemplo, no grafo abaixo, o caminho {v2e3v3e4v2e2v2e3v3} possui tamanho 4.



Existem 6 (seis) caminhos distintos de tamanho 2 conectando v2 a v2, que são:

v2e1v1e1v2
v2e2v2e2v2
v2e3v3e4v2
v2e4v3e3v2
v2e3v3e3v2
v2e4v3e4v2

Se A é uma matriz de adjacência n x n de um grafo e se a_{ij}(k) representa o elemento (i,j) de A^k, então a_{ij}(k) é igual ao número de caminhos de comprimento k de Vi a Vf.

Portanto, através da multiplicação das matrizes de adjacência de um grafo G, identifica-se quantos caminhos distintos possíveis de

tamanho n existem conectando dois vértices de um dado grafo G.

A matriz de adjacência A do grafo da figura anterior é dada por:

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} \end{array}$$

O resultado do cálculo do produto de A x A , que é igual a $\boldsymbol{A^2}$ é dado pela matriz abaixo que representa a entrada aij da matriz AxA indica a quantidade de caminhos de tamanho 2 conectando vi a vj no grafo G. Este resultado é válido para caminhos de tamanho n calculando $\boldsymbol{A^n}$.

O cálculo de $\boldsymbol{A^2}$ **é dado por:**

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 6 & 2 \\ 2 & 2 & 4 \end{bmatrix}$$

Por exemplo, a entrada a22 da matriz acima é dado pelo valor 6 que, conforme visto anteriormente, representa número de caminhos de tamanho 2 de v2 para v2.

Observe que a matriz $\boldsymbol{A^2}$ é simétrica, ou seja, por exemplo, existe o mesmo número de caminhos de **tamanho** $\boldsymbol{a_{ij}}$**(k) tanto de Vi para Vj, como de Vj para Vi.**

. Lista de adjacência: Um grafo com poucos arcos pode ser representado de modo mais eficiente armazenando-se apenas os elementos não nulos da matriz de adjacência. Esta representação consiste em uma lista, para cada nó, de todos os nós adjacentes a ele. São usados ponteiros para irmos de um item na lista para o próximo (lista encadeada).

Para se encontrar todos os nós adjacentes a n_i é preciso percorrer a lista encadeada para n_i , que pode ter muito menos do que os n elementos que teríamos que examinar na matriz de adjacência.

No entanto, existem algumas desvantagens se quisermos determinar se um determinado nó n_j é adjacente a n_i , podemos ter que percorrer toda a lista encadeada de n_i , ao passo que, na matriz de adjacência, podemos acessar o elemento i,j diretamente.

A lista adjacente para o grafo da fig. 2.1 é apresentada na fig. 2.3. Esta lista contêm um arranjo de ponteiros com quatro elementos, um para cada nó. O ponteiro de cada nó aponta para um nó adjacente, que aponta para outro nó adjacente, e assim por diante.

Na figura 2.3, o ponto indica um ponteiro nulo, ou seja, que chegou-se ao final da fila.

Em um grafo não direcionado, cada arco é representado duas vezes. Se n_j está na lista de adjacência de n_i , então n_i também está na lista de adjacência de n_j .

A lista de adjacência para um grafo direcionado coloca n_j na lista de adjacência de n_i se existir um arco de n_i para n_j ; n_i não precisa, necessariamente, estar na lista de adjacência de n_j .

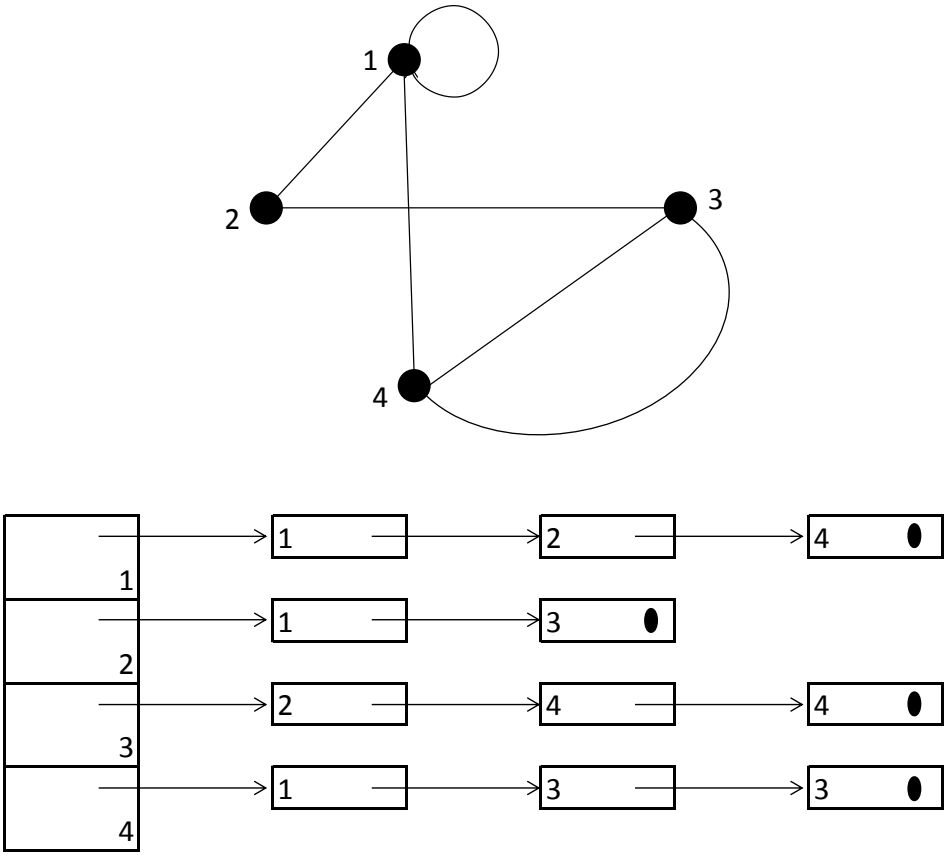


Fig. 2.3 – lista de adjacência

A figura 2.4 apresenta um grafo direcionado com pesos e a sua respectiva lista de adjacências. Para cada registro na lista, o primeiro componente é o nó, o segundo componente é o peso do arco que termina no nó, e a terceira é o ponteiro. Para o nó 4 não existe nenhum arco saindo, portanto, o primeiro componente é nulo.

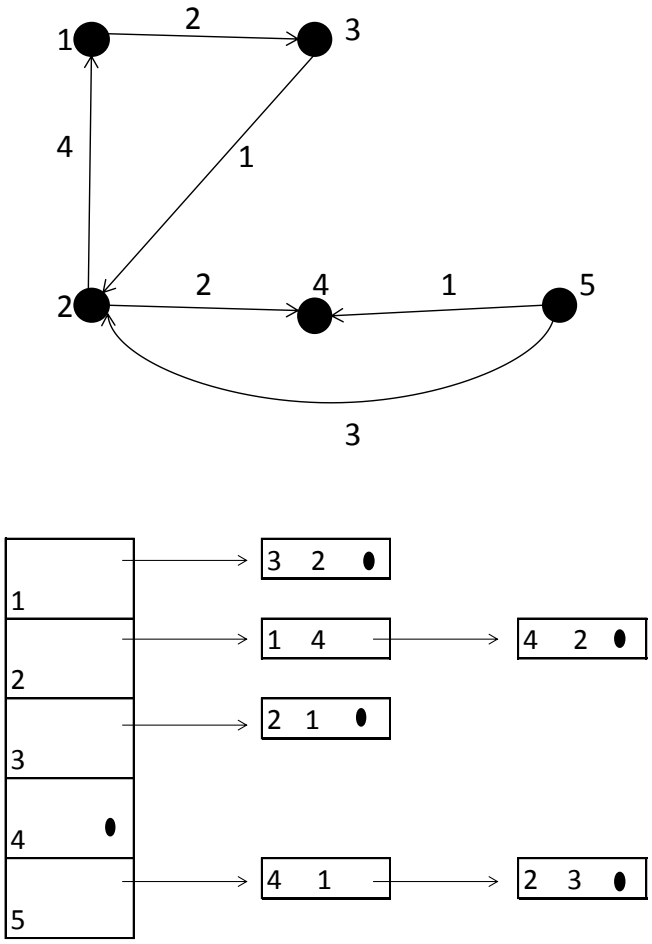
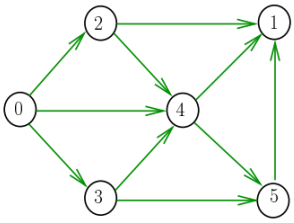


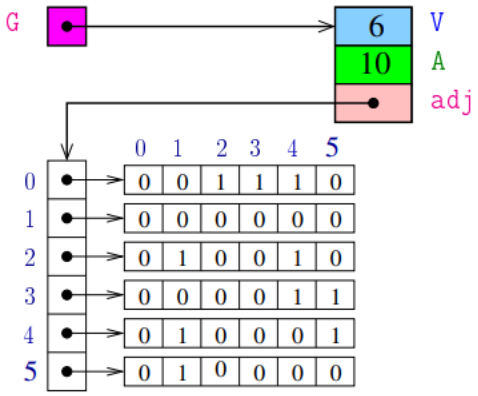
Fig. 2.4 – lista de adjacência direcionada com pesos

A desvantagem da utilização das listas de adjacências está em situações onde o grafo possui uma quantidade grande de arestas, e portanto, as listas serão longas.

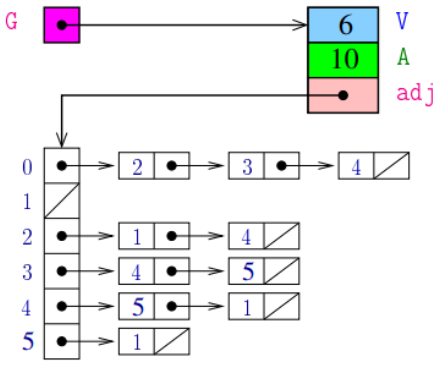
Exemplo: Dado o grafo G abaixo:



Uma sugestão de estrutura de dados com matriz de Adjacência:



Uma sugestão de estrutura de dados com listas:




```
struct digraph {
    int V;
    int A;
    link *adj;
};
```

A estrutura digraph representa um dígrafo:

V contém o número de vértices
A contém o número de arcos do digrafo
adj é um ponteiro para vetor de listas de adjacência

Um objeto do tipo Digraph contém o endereço de um digraph:
typedef struct digraph *Digraph;

A lista de adjacência de um vértice v é composta por nós do tipo node. Um link é um ponteiro para um node. Cada nó da lista contém um vizinho w de v e o endereço do nó seguinte da lista:

```
typedef struct node *link;
struct node {
Vertex w;
link next;
};
```

Exercício: Implemente um programa que dado um grafo qualquer (que pode ser direcionado), apresente a lista de adjacência e o grau de cada nó.

.**Vetor de adjacência**:

Neste tipo de representação são utilizados dois vetores, V e W, com dimensões n e m, respectivamente. Cada elemento i do vetor V registra o grau do vértice i. Os elementos de W correspondem aos vizinhos dos vértices representados em V. Os elementos W[1] até W[V[1]] correspondem aos vizinhos do vértice 1, os elementos W[V[1]+1] até W[V[1]+V[2]] correspondem aos vizinhos do vértice 2, e daí por diante. Generalizando, dado um vértice j > 1, seus adjacentes encontram-se em W[V[1]+...+V[j-1] +1] a W[V[1]+...+V[j-1]+V[j]].

Portanto, a estrutura de representação vetorial é similar a da lista. Para o vetor, a variável tem o endereço base, onde começa

o vetor na memória e cada elemento possui um tamanho fixo. A busca por um determinado elemento dentro do vetor é mais rápida se comparada com a busca utilizando listas, tendo em vista que para se acessar uma determinada posição que está dentro da lista, tem-se que percorrer a lista até a posição desejada!

A figura 2.5 ilustra os dois tipos de estrutura de dados e a representação do vetor de adjacência.

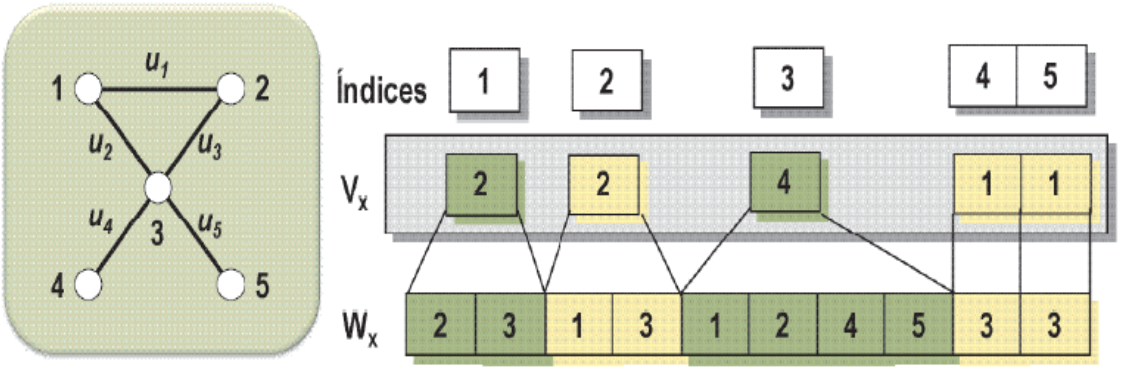


Fig. 2.5 – vetor de adjacência

A estrutura mais adequada depende do problema que se deseja resolver e do algoritmo a ser implementado.