SCC 503 - ALGORITMOS E ESTRUTURAS DE DADOS II

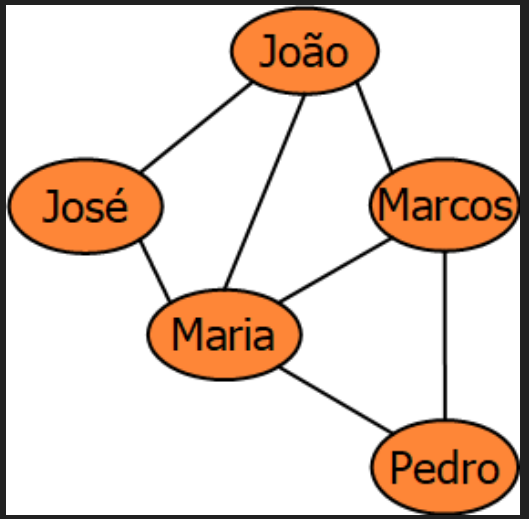
Grafos - Mais Conceitos e Lista Ligada

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

*Material baseado em aulas dos professores: Elaine Parros Machado de Souza, Gustavo Batista, Robson Cordeiro, Moacir Ponti Jr., Maria Cristina Oliveira e Cristina Ciferri.

Antes de tudo, uma breve revisão

Definições



- ➔ Exemplo de Grafo
  - ◆ Rede social de amizade
    - Cada vértice é uma pessoa
    - Cada aresta representa uma amizade entre pessoas

Definições

➔ Se eu sou seu amigo, isso significa que você é meu amigo?

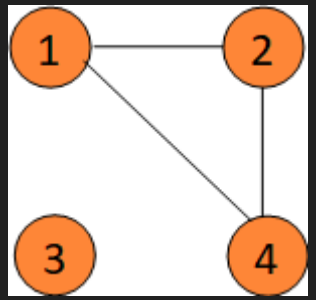
- Se aresta (x,y) sempre implica em (y,x)
  - Grafo não-direcionado
- Caso contrário
  - Grafo direcionado (dígrafo)
- Como seria um grafo sobre “escreveu um artigo com”?

Grafo

- ➔ Um grafo não direcionado G é um par (V,A), em que o conjunto de arestas A é constituído de pares de vértices não ordenados
- ➔ Os pares (u,v) e (v,u) são considerados como uma única aresta
- ➔ A relação de adjacência é simétrica

Grafo

- G = (V,A)
  - V = {1, 2, 3, 4}
  - A = {(1,2),(1,4),(2,4)}

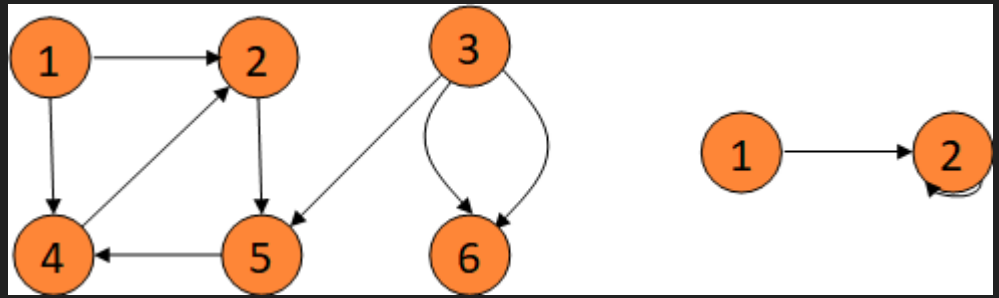


Dígrafo

- Um grafo direcionado (grafo orientado ou Dígrafo) G é um par (V,A), em que
  - V é um conjunto finito de vértices
  - A é uma relação binária ordenada em V
- Uma aresta (*u,v*) sai do vértice u (origem) e chega no vértice v (destino)
  - O vértice *v* é adjacente ao vértice *u*
  - O vértice *u* é incidente ao vértice *v*
  - A existência de (*u,v*) não implica a existência de (*v,u*)

Dígrafo

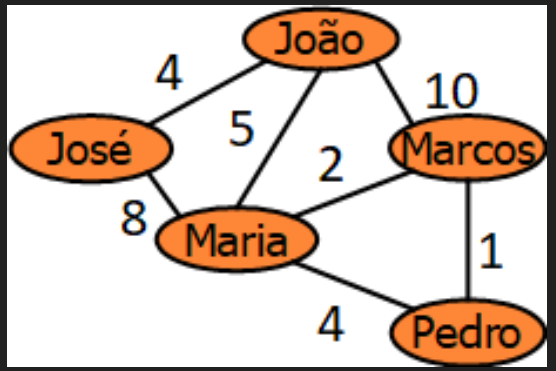
- ➔ Podem existir arestas de um vértice para ele mesmo
  - ◆ Self-loops
- ➔ Arestas múltiplas
  - ◆ Arestas com mesma origem e mesmo destino



Grafo

➔ O quanto você é meu amigo?

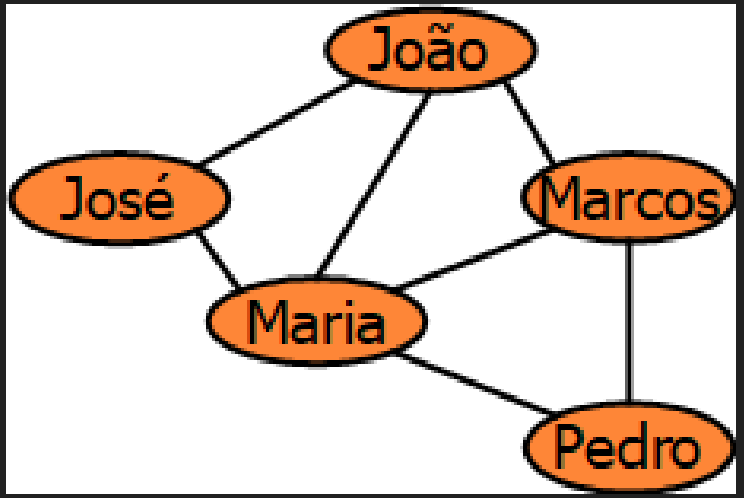
- ◆ Grafo ponderado
  - As arestas possuem um peso (valor numérico) associado



Grafo

➔ Grafo não ponderado

- ◆ Todas as arestas possuem um mesmo peso



Mais Definições

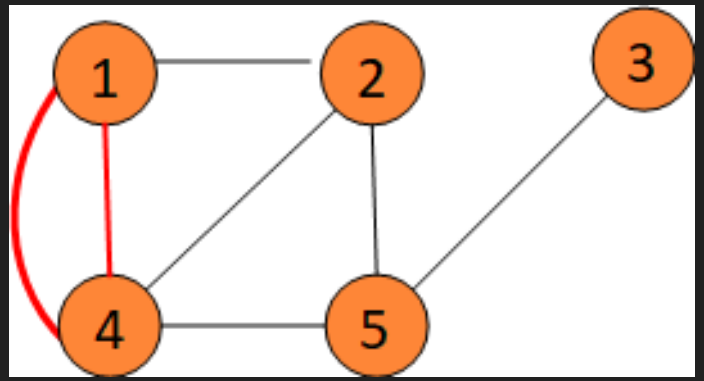
Definições

- ➔ Eu sou amigo de mim mesmo?
  - ◆ Aresta (x,x)
    - ● Laço ou self-loop
- ➔ Eu posso ser seu amigo diversas vezes?
  - ◆ Relação modelada com arestas múltiplas ou paralelas (multigrafo)
  - ◆ Por exemplo, o grafo das pontes de Könisberg...

Grafo

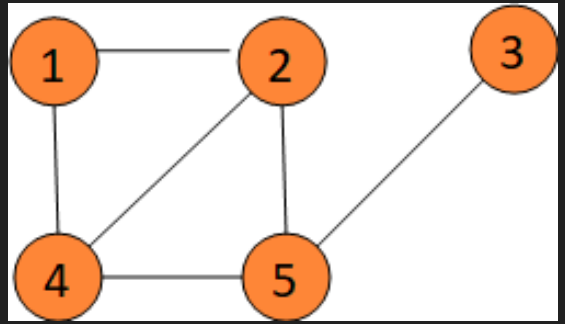
➔ Grafo com arestas múltiplas: G =(V,A)

- V = {1,2,3,4,5}
- A = {(1,2), (1,4), (1,4),(2,5),(4,2),(5,4),(3,5)}



Grafo

- ➔ Grafo simples
  - ◆ Não possui laços nem arestas múltiplas
- ➔ G = (V,A)
  - ◆ V = {1,2,3,4,5} e
  - ◆ A = {(1,2),(1,4),(2,5),(4,2),(5,4),(3,5)}



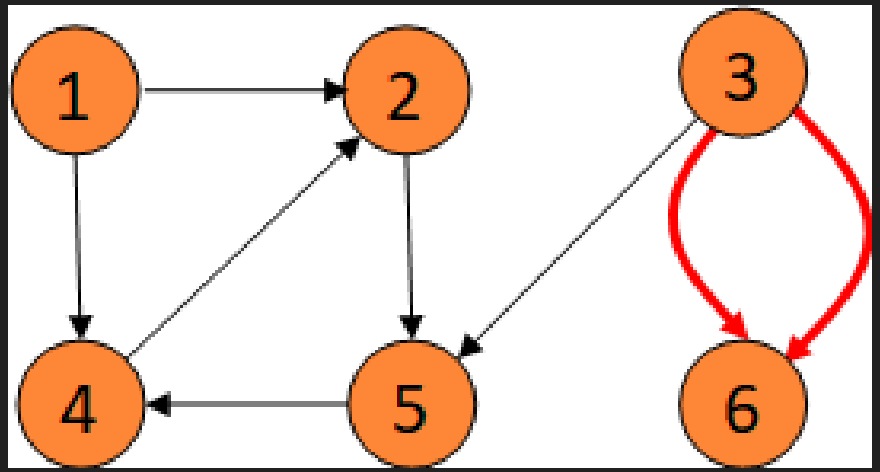
Dígrafo

- G = (V,A)
  - V = (1,2) e
  - A = {(1,2),(2,2)}

Dígrafo

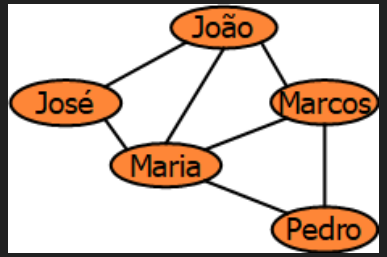
➔ G =(V,A)

- V = {1,2,3,4,5,6} e
- A = {(1,2),(1,4),(2,5),(4,2),(5,4),(3,5),(3,6), (3,6)}



Grau de um Vértice

Grafo



➔ Quem possui mais (ou menos) amigos?

- Quantidade de relacionamentos (conexões)
- Grau do vértice
  - Número de vértices adjacentes a ele (não direcionado)
  - Pessoa mais popular tem o vértice de maior grau do grafo
  - Vértices de grau zero?

Caminhos em Grafos

Caminhos em Grafos

➔ Será que eu estou ligado a uma celebridade por uma cadeia de amigos?

- ◆ Existe um caminho entre eu e alguma celebridade?
- ◆ Caminho entre dois vértices
  - Existe uma sequência de arestas que conectam os dois vértices

Caminho de um Grafo

➔ Um caminho de comprimento **k** de um vértice *x* a um vértice *y* em um grafo G = (V,A) é uma sequência de vértices (v_0 ,v_1 ,v2 ,....,v_k) tal que:

- x = v_0
- y = v_k
- (v_{i-1}, v_i) ∈ A para i = 1,2,...,k

Caminho de um Grafo

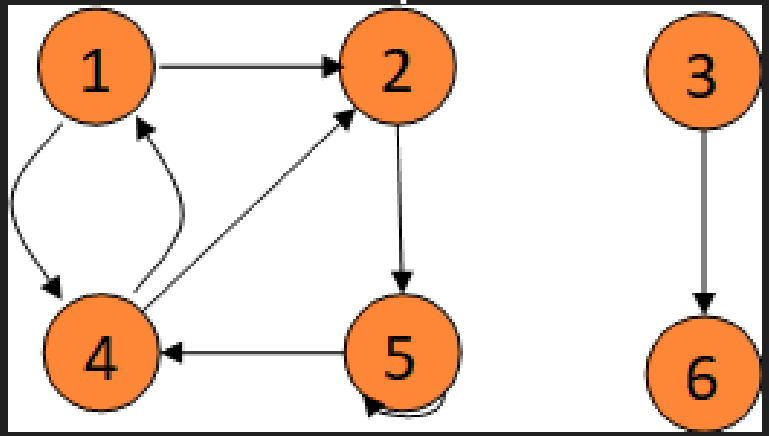
- O comprimento de um caminho é dado pelo número de arestas nele, isto é, o caminho contém:
  - Os vértices v_0 , v_1 , v_2 ,...., v_k
  - As arestas $(v_0 , v_1),(v_1 , v_2) ,...., (v_{k-1} , v_k)$

Caminho de um Grafo

- ➔ Se existir um caminho **c** de *x* a *y* então *y* é alcançável a partir de *x* via c.
- ➔ Caminho simples
  - ◆ Todos os vértices do caminho são distintos
- ➔ Caminho Hamiltoniano
  - ◆ Caminho que passa por todos os vértices de um grafo exatamente uma vez

Caminho de um Grafo - Exemplo

- ➔ Caminho (1,2,5,4)
  - ◆ Simples, com comprimento 3
- ➔ Caminho (1,4,1,2)
  - ◆ Não é simples, com comprimento 3



Características - Matriz de Adjacência

- ➔ Para que categoria de grafo usar?
  - ◆ Grafos densos, onde |A| é próximo de $|V|^2$
- ➔ Tempo necessário para acessar um elemento é independente de |V| e |A|
  - ◆ O(1)
- ➔ Muito útil para algoritmos que precisam saber com rapidez se...
  - ◆ Existe uma aresta ligando dois vértices

Características - Matriz de Adjacência

- ➔ Desvantagem
  - ◆ Espaço
    - O($|V|^2$)
  - ◆ Ler ou percorrer a matriz
    - Complexidade de tempo O($|V|^2$)
- ➔ Matriz é simétrica para grafos não direcionados
  - ◆ Cerca de metade do espaço pode ser economizado representando a matriz triangular superior ou inferior

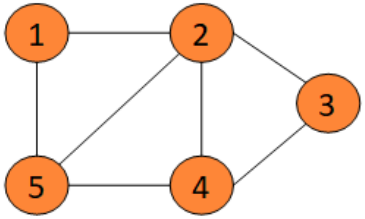
Lista de Adjacência

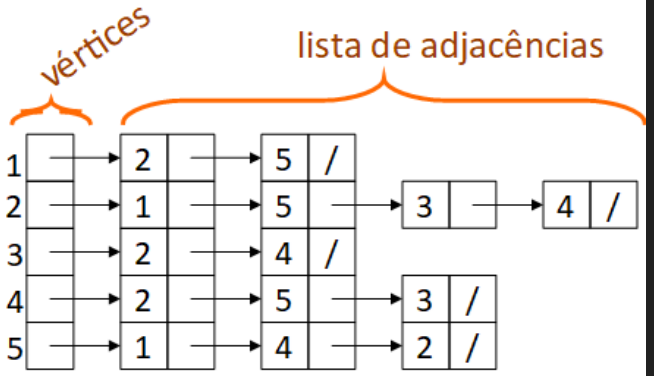
Lista de Adjacências

- ➔ Cada linha da matriz de adjacências é representada por uma lista ligada
- ➔ Nós na lista do vértice *u* representam os vértices que são adjacentes a *u*
- ➔ Cada lista ligada é apontada por um nó cabeça.
- ➔ Nós cabeça organizados sequencialmente em um vetor
  - ◆ Acesso rápido a qualquer lista ligada

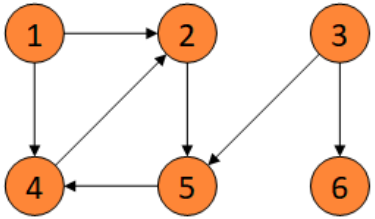
Lista de Adjacências

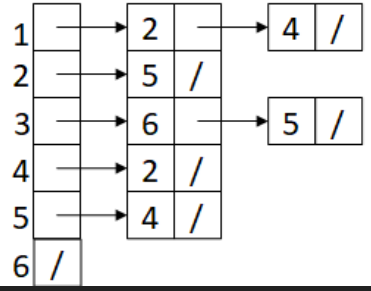
grafo não direcionado





grafo direcionado





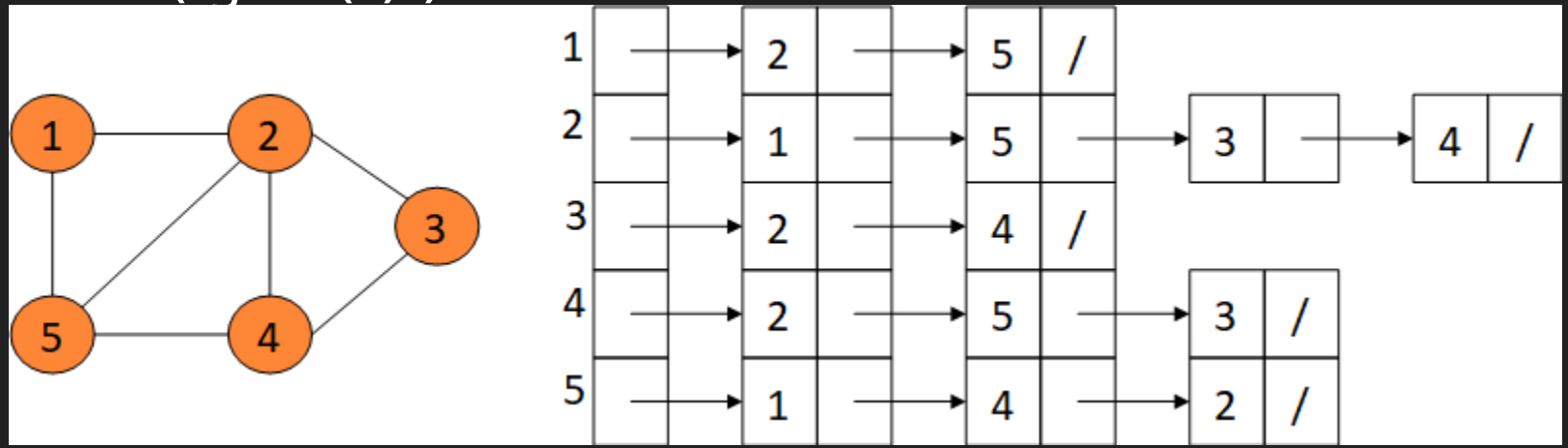
Lista de Adjacências

- ➔ Vértices em geral armazenados em uma ordem arbitrária.
- ➔ Complexidade de espaço $O(|V|+|A|)$
- ➔ Indicada para grafos esparsos
  - ◆ $|A|$ é muito menor do que $|V|^2$

Lista de Adjacências

➔ Complexidade para determinar se existe uma aresta entre o vértice v e o vértice u?

- O(|grau(v)|)



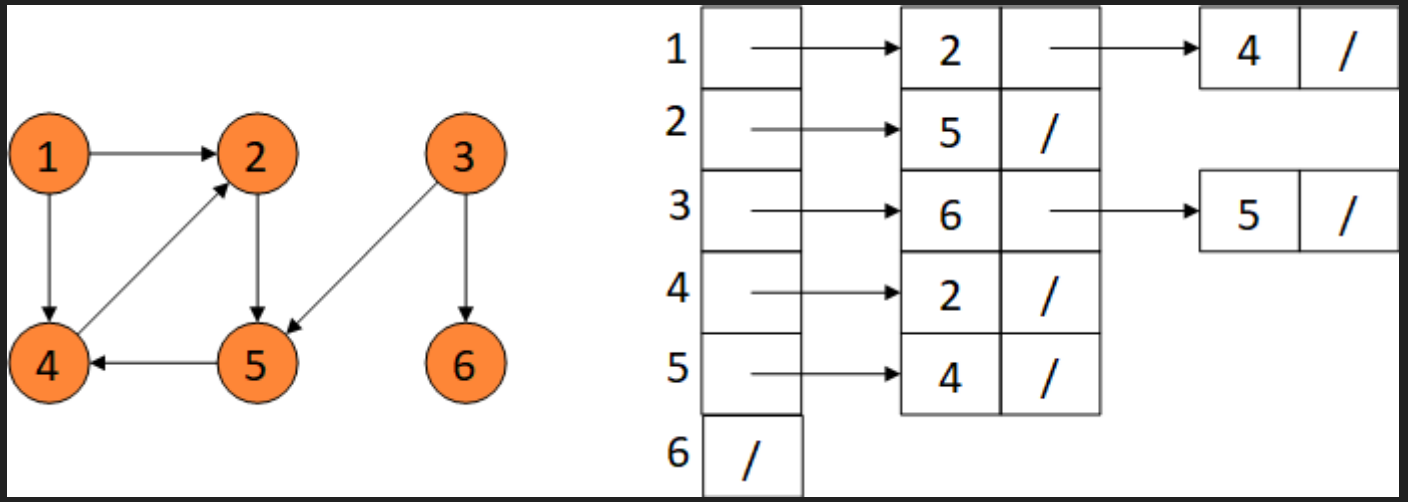
Lista de Adjacências

➔ Complexidade para determinar se existe uma aresta entre o vértice v e o vértice u?

- O(|grau(v)|)
- grafos não direcionados
  - Grau(v) = grau do vértice v
- grafos direcionados
  - Grau(v) = grau de saída do vértice v
- Grau(v) ~= |V| para vértices com muitas arestas

Lista de Adjacências

- ➔ Grafos direcionados
  - ◆ Como determinar o grau de entrada de um vértice v?
  - ◆ Requer percorrer toda a estrutura do grafo



Comparação entre Matriz e Listas

Comparações entre Matriz e Listas

Operação	Matriz	Listas
Inicializa	$O(\|V\|^2)$	**$O(\|V\|)$**
InsereAresta	$O(1)$	$O(1)$
ExisteAresta	**$O(1)$**	$O(grau(v))$
RetiraAresta	**$O(1)$**	$O(grau(v))$
LiberaGrafo	**$O(1)$**	$O(\|V\|+\|A\|)$
ExisteAdj	$O(\|V\|)$	**$O(1)$**
PrimeiroAdj	$O(\|V\|)$	**$O(1)$**
ProxAdj	$O(\|V\|)$	**$O(1)$**

Comparações entre Matriz e Listas

Operação	**Matriz**	**Listas**
Percorrer um grafo	$O(\|V\|^2)$	**$O(\|V\|+\|A\|)$**
Determinar o grau de um vértice em um grafo não direcionado	$O(\|V\|)$	**O(grau(v))**
Determinar o grau de um vértice em um grafo direcionado	$O(\|V\|)$	O(grau(v)) (*out-degree*) $O(\|V\|+\|A\|)$ (*in-degree*)

Possível implementação E.D. Grafo

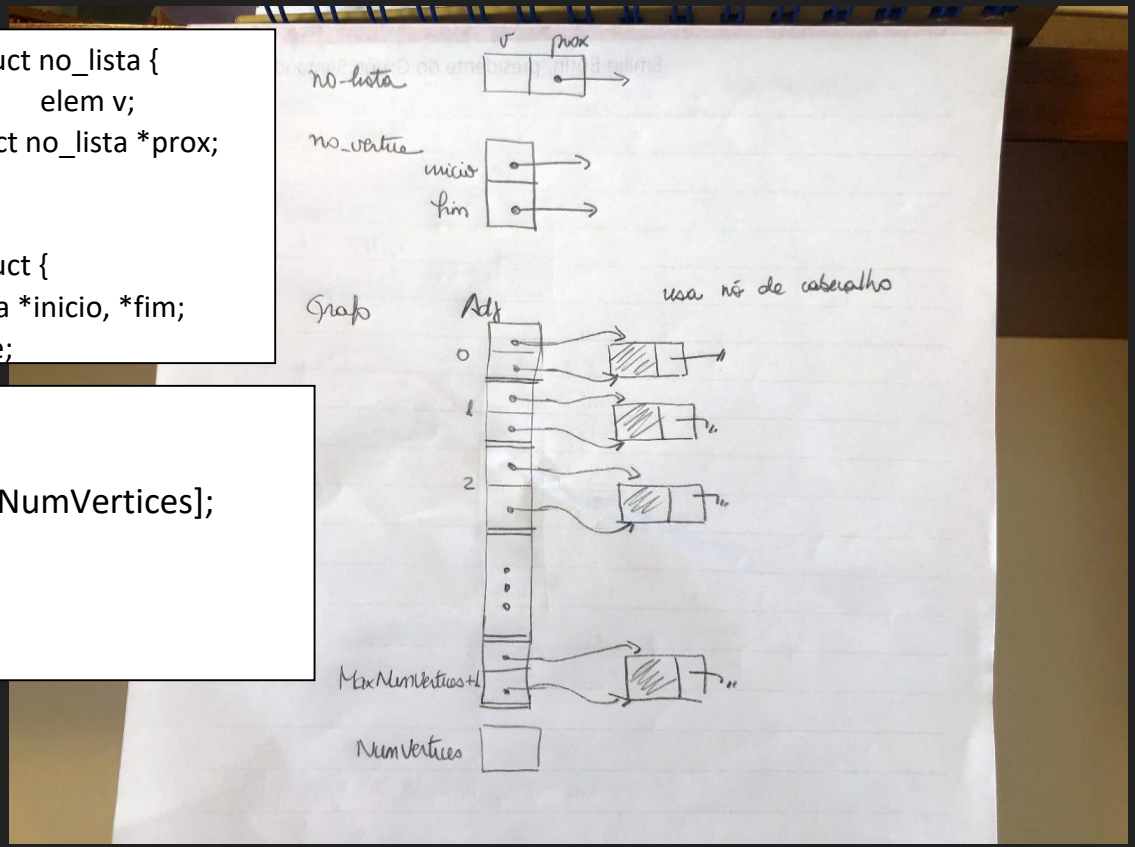
```
define MaxNumVertices 100

typedef int elem;

typedef struct no_lista {
        elem v;
        struct no_lista *prox;
} no_lista;
```

```
typedef struct {
        no_lista *inicio, *fim;
} no_vertice;


typedef struct {
        no_vertice Adj[MaxNumVertices];
        int NumVertices;
} Grafo;
```

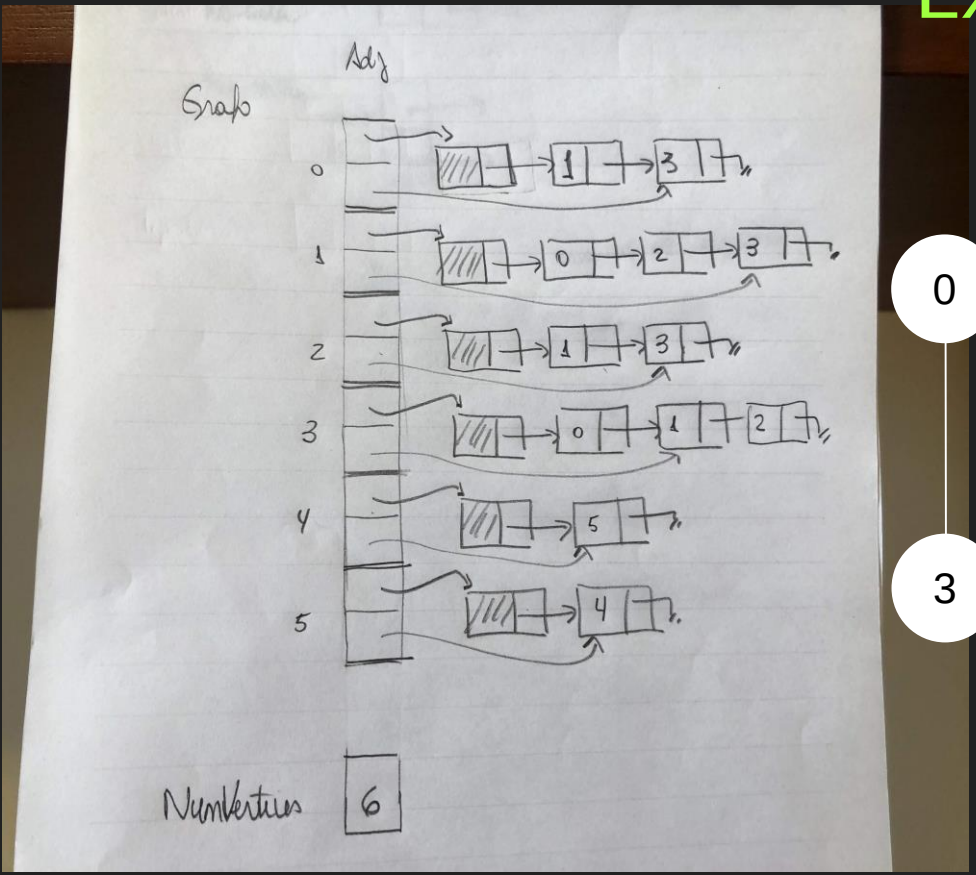
Possível implementação E.D. Grafo

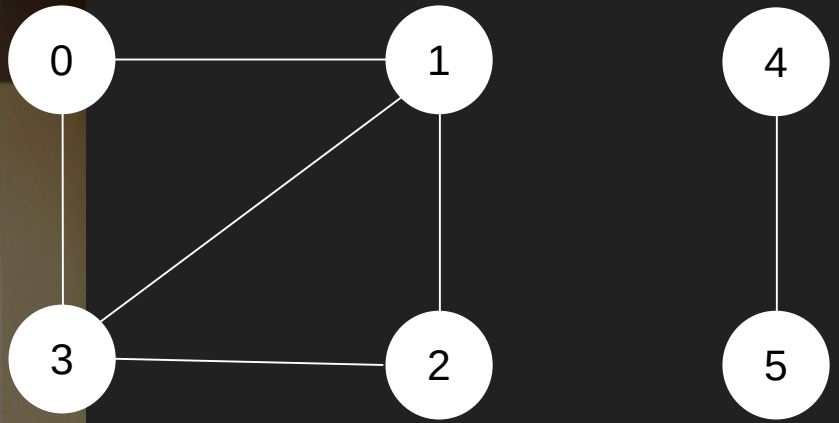
```
typedef struct no_lista {
                    elem v;
            struct no_lista *prox;
} no_lista;

typedef struct {
        no_lista *inicio, *fim;
} no_vertice;
```

```
typedef struct {
        no_vertice Adj[MaxNumVertices];
        int NumVertices;
} Grafo;
```



Exemplo





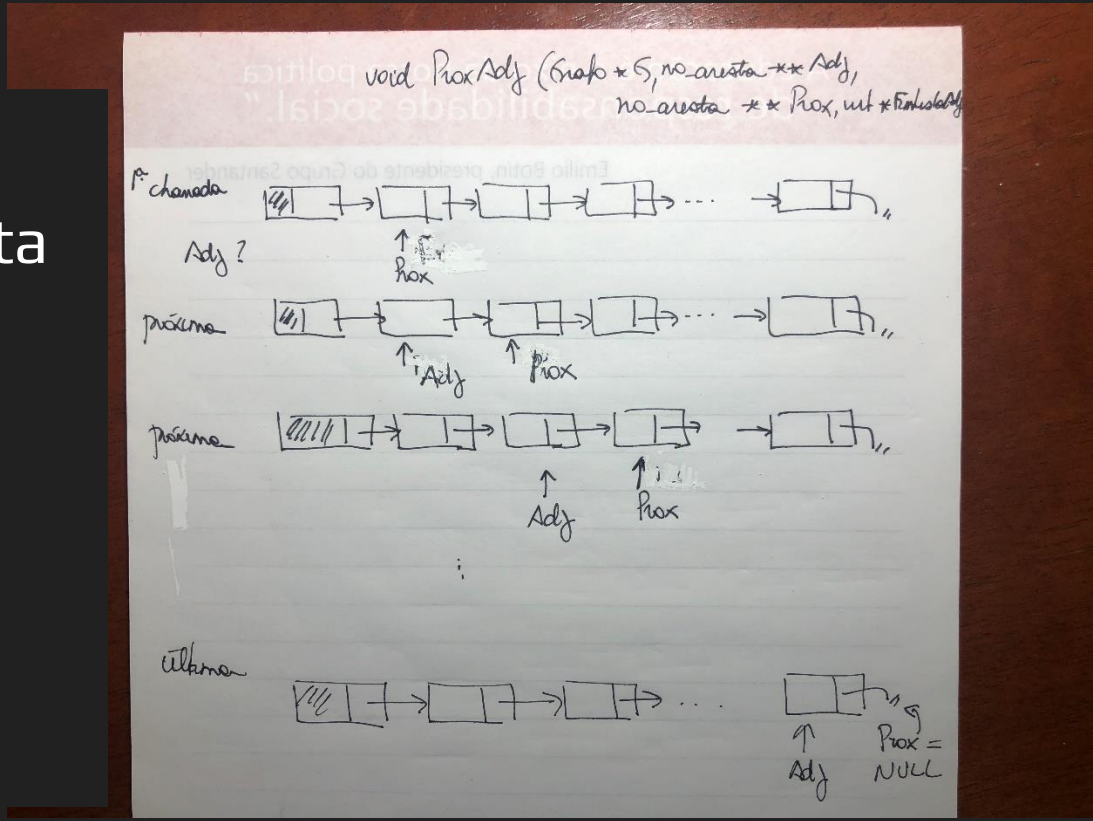
Função: percorrer Lista de Adjacências do vértice V

```
/* retorna o (endereço do) primeiro vértice na lista de adjacentes a V  */

no_aresta* PrimeiroListaAdj(Grafo *G, int *V, int *erro) {
   if (*V >= G->NumVertices) {
      *erro= 1;
      return(NULL);
   }
   else {
       *erro= 0;
       return(G->Adj[*V].inicio->prox);
   }
}
```

Função: retorna vértice corrente a Lista de Adjacências do vértice V e avança para o próximo

```
/* na lista de adjacentes de V: retorna o (endereço do) vértice atualmente
apontado por Prox (na variável Adj) e  já posiciona Prox no próximo vértice
da lista; FimListaAdj retorna 1 se chegou no final da lista */

void ProxAdj(Grafo *G, no_aresta **Adj, no_aresta **Prox, int *FimListaAdj)
{
    *Adj= *Prox;
    *Prox= (*Prox)->prox;

    if (*Prox == NULL)
      *FimListaAdj= 1;
}
```

```
void ProxAdj(Grafo *G,
no_aresta **Adj, no_aresta
**Prox, int *FimListaAdj) {
    *Adj= *Prox;
    *Prox= (*Prox)->prox;

    if (*Prox == NULL)
      *FimListaAdj= 1;
}
```



Referências

- WIRTH,N. Algorithms and Data Structures, Englewood Cliffs, Prentice-Hall, 1986.
- CORMEN, H.T.; LEISERSON, C.E.; RIVEST, R.L. Introduction to Algorithms, MIT Press, McGraw-Hill, 1999.
- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.
- SZWARCFITER,J.L. Grafos e Algoritmos Computacionais. Editora Campus, 1983.