DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Grafos

Professores: Wagner Meira Jr
Eder Figueiredo

Grafos

Um grafo é uma representação abstrata de um conjunto de objetos e das relações existentes entre eles. É definido por um conjunto de **vértices** e por ligações que conectam pares de vértices, chamadas **arestas**. Um grafo *G* é denotado da forma:

$$G(V, E)$$

Onde **V** é o conjunto de vértices e **E** o conjunto de arestas.

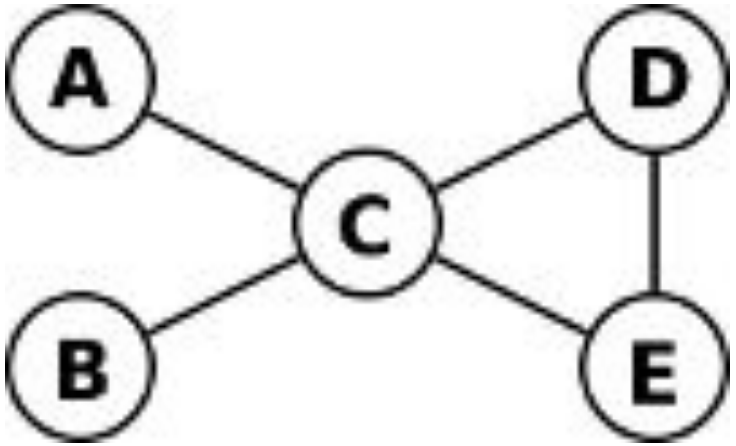
Grafos - representação

Um grafo pode ser representado exibindo a descrição de ambos conjuntos V e E, no entanto a representação mais comum é a gráfica.

Representação por conjuntos:

V = {A, B, C, D, E}
E = {(A,C), (B,C), (C,D), (D, E), (C, E)}

Representação gráfica:



Grafos - algumas definições

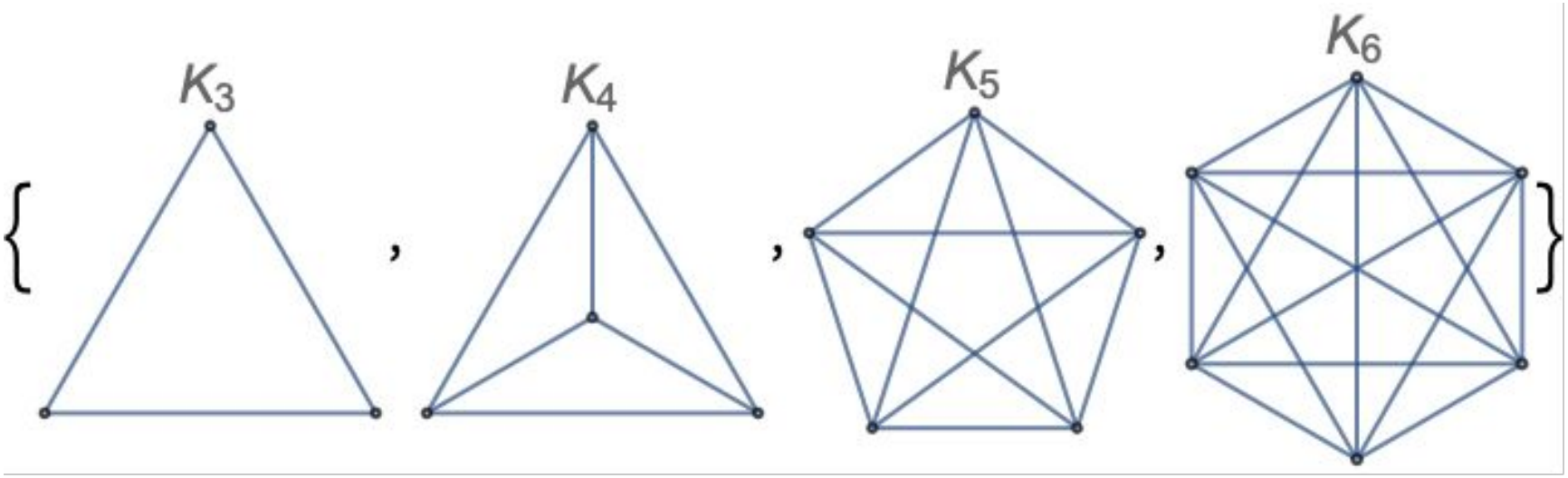
- Dois vértices *u* e *v* são ditos **adjacentes** ou **vizinhos** se existe a aresta *uv*.
- Um **caminho** é uma sequência de vértices $v_1 v_2 \ldots v_k$ de forma que $v_i v_{i+1}$ são vizinhos para todo 0 < i < k.
- O **grau** de um vértice *v*, denotado por *d(v)* é a quantidade de vizinhos que *v* possui. O grau máximo de um grafo é denotado $\Delta(G)$ e o grau mínimo por $\delta(G)$.

Grafos - algumas definições

- Um **laço** é uma aresta que conecta um vértice a ele mesmo.
- Uma aresta que conecta *u* a *v* é dita **paralela** se existe uma outra aresta que conecta *u* a *v*.
- Um grafo é **simples** se **não** possui laços nem arestas paralelas. Os grafos que veremos nesta seção serão todos simples.

Grafos - algumas definições

- Um grafo é **completo** se para todo par de vértices *u* e *v*, a aresta *uv* existe. Em outras palavras é um grafo que possui todas as arestas possíveis. Um grafo completo de *n* vértices é chamado de K_n.



Grafos - representação computacional

Mas como representar um grafo no computador? Basicamente temos uma lista de vértices e uma lista de arestas. Existem várias maneiras de representar um grafo como uma estrutura de dados, mas as duas principais são:

- Matriz de adjacência
- Lista de adjacência

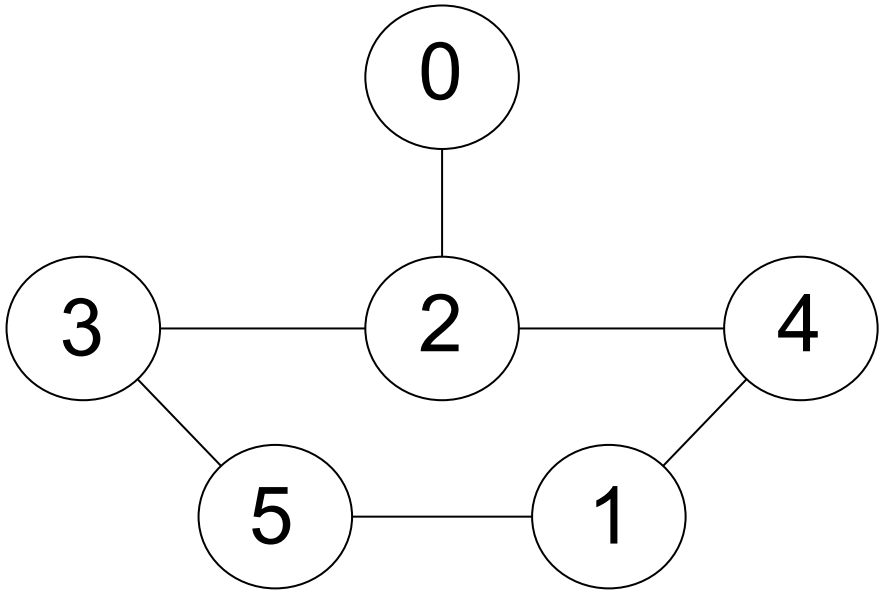
Grafos - Matriz de adjacência

Seja *G* um grafo com *n* vértices. Vamos construir uma matriz quadrada *A* com *n* dimensões que satisfaz a seguinte propriedade:

A[i][j] = 1 se e somente se existe uma aresta que conecta o vértice *i* com o vértice *j*.

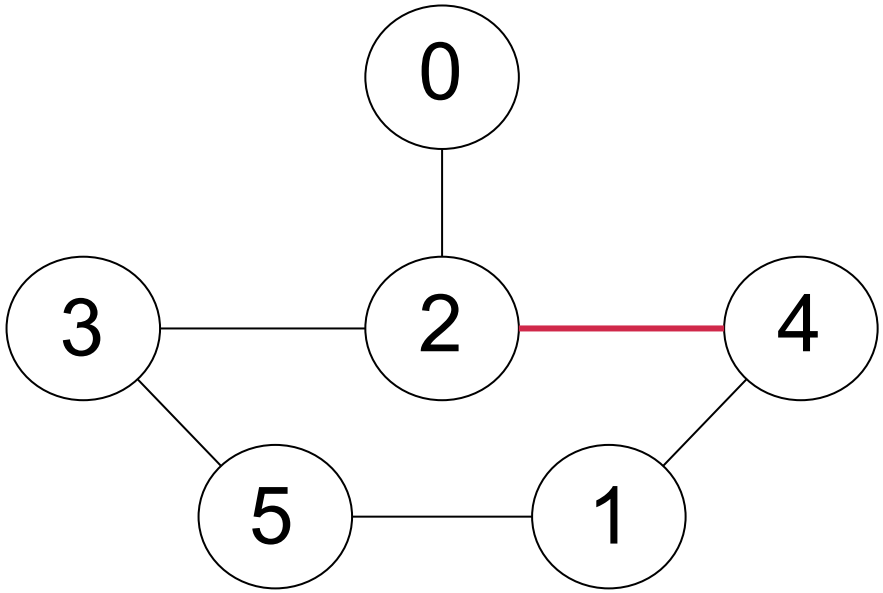
A[i][j] = 0 caso contrário.

Matriz de adjacência - Exemplo



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

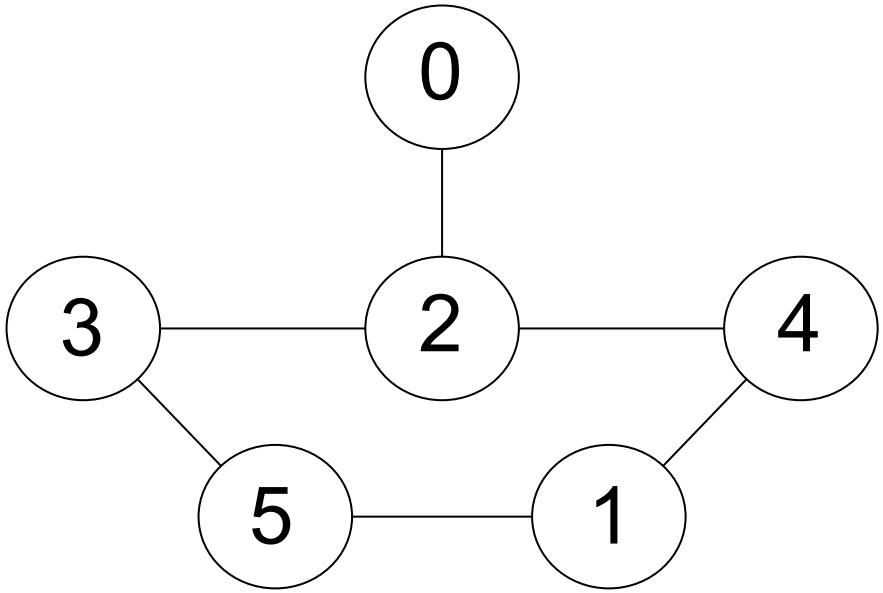
Matriz de adjacência - Exemplo



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

Para verificar a existência de uma aresta basta consultar a célula correspondente da matriz. esta operação pode ser realizada em tempo *O(1)*.

Matriz de adjacência - Exemplo



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

Observe que a matriz de adjacência é simétrica, uma vez que a aresta que conecta *u* com *v* também conecta *v* com *u*.

Matriz de adjacência - Exemplo

Vantagens:

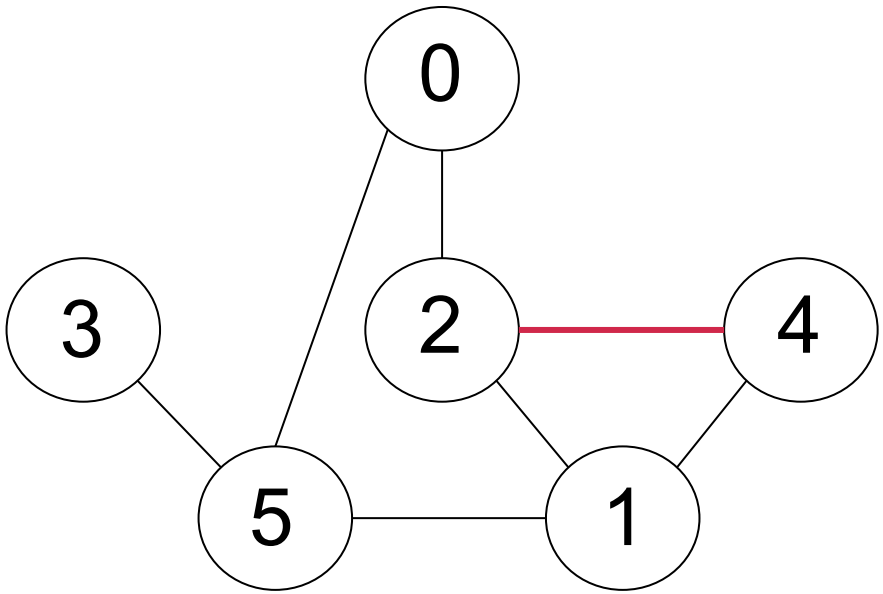
- Acesso rápido a arestas. A existência de uma aresta pode ser conferida em $O(1)$.

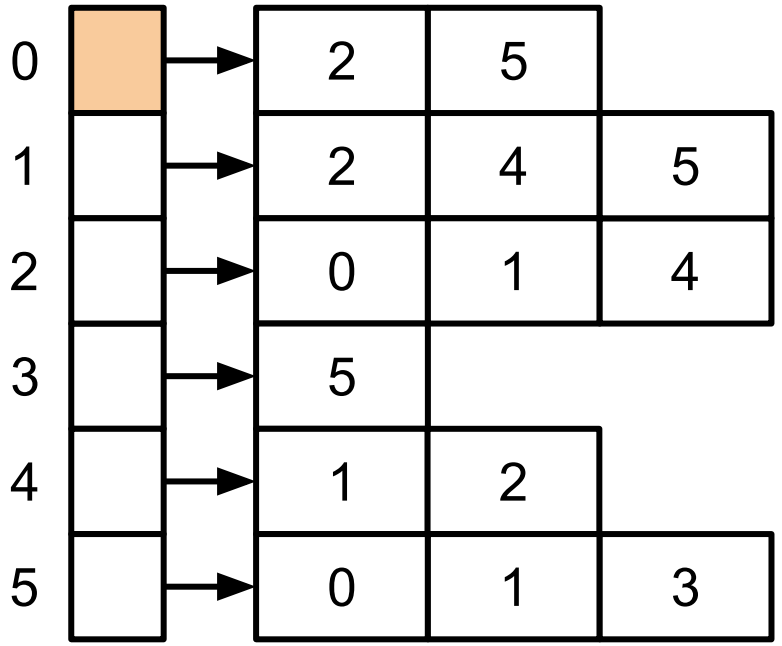
Desvantagens:

- Consultar a vizinhança de um vértice é feita em $\theta(n)$, pois sempre será necessário varrer a linha toda da matriz.
- Ocupa espaço $O(n^2)$ independente de quantas arestas o grafo possui.

Grafos - Lista de adjacência

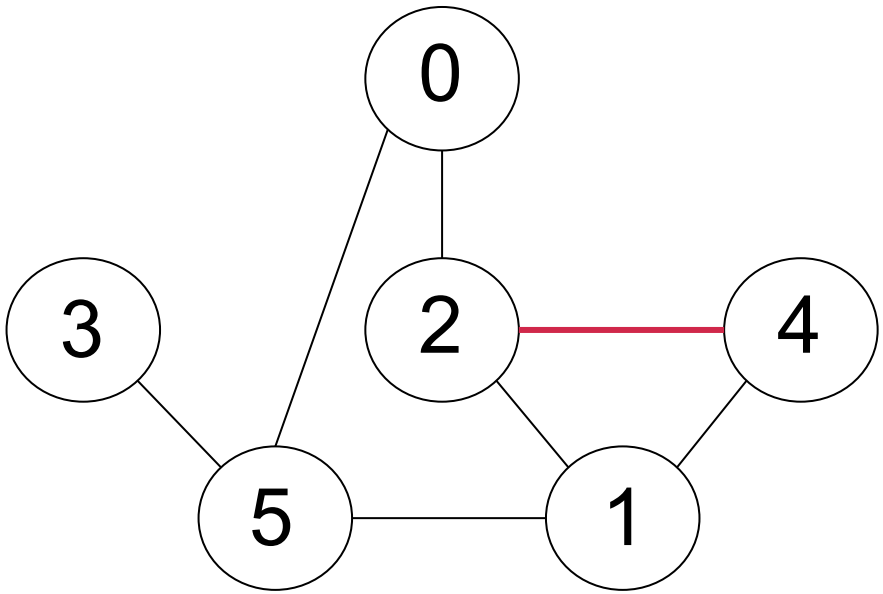
Para verificar a existência de uma aresta agora é necessário percorrer as listas.

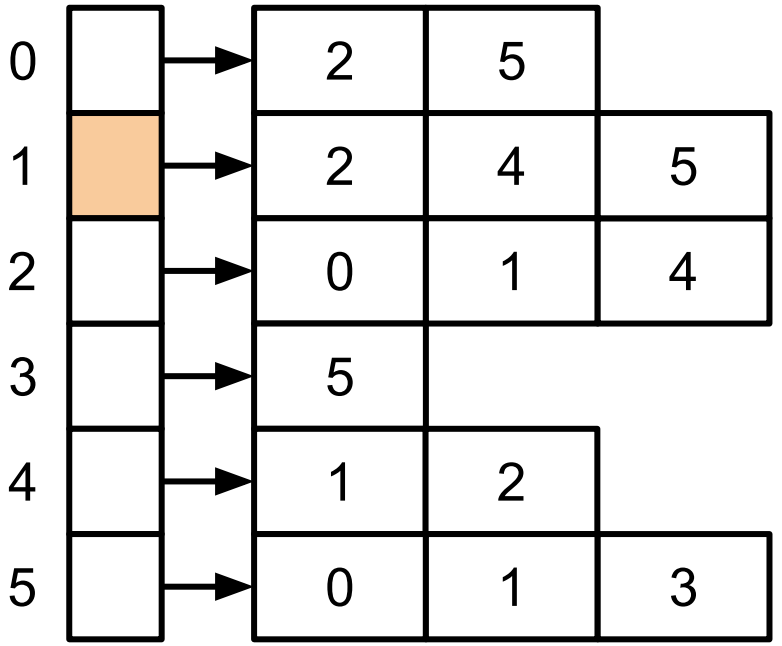




Grafos - Lista de adjacência

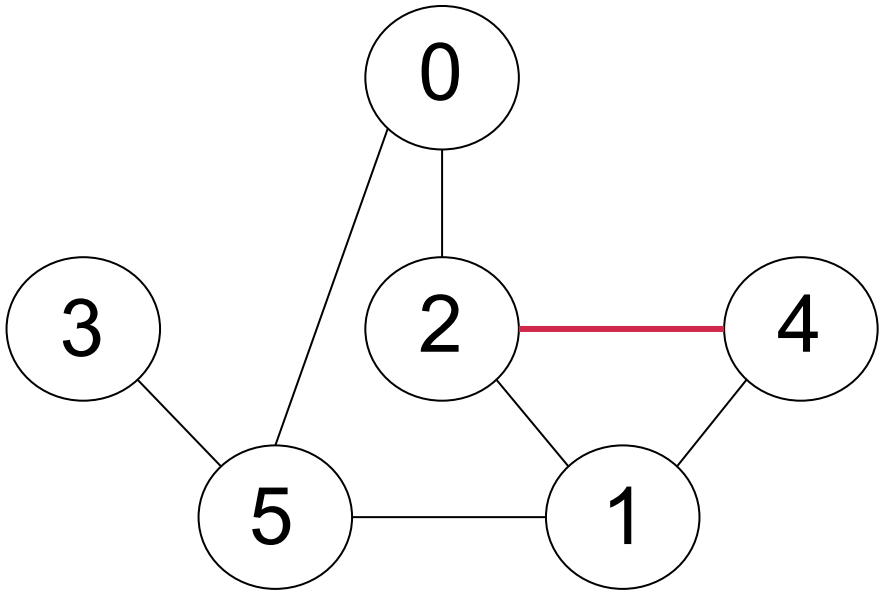
Para verificar a existência de uma aresta agora é necessário percorrer as listas.

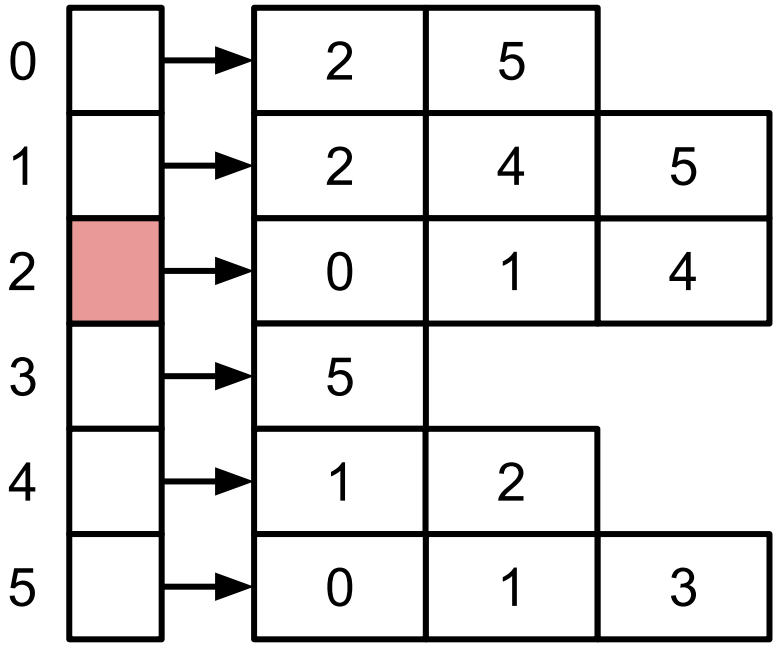




Grafos - Lista de adjacência

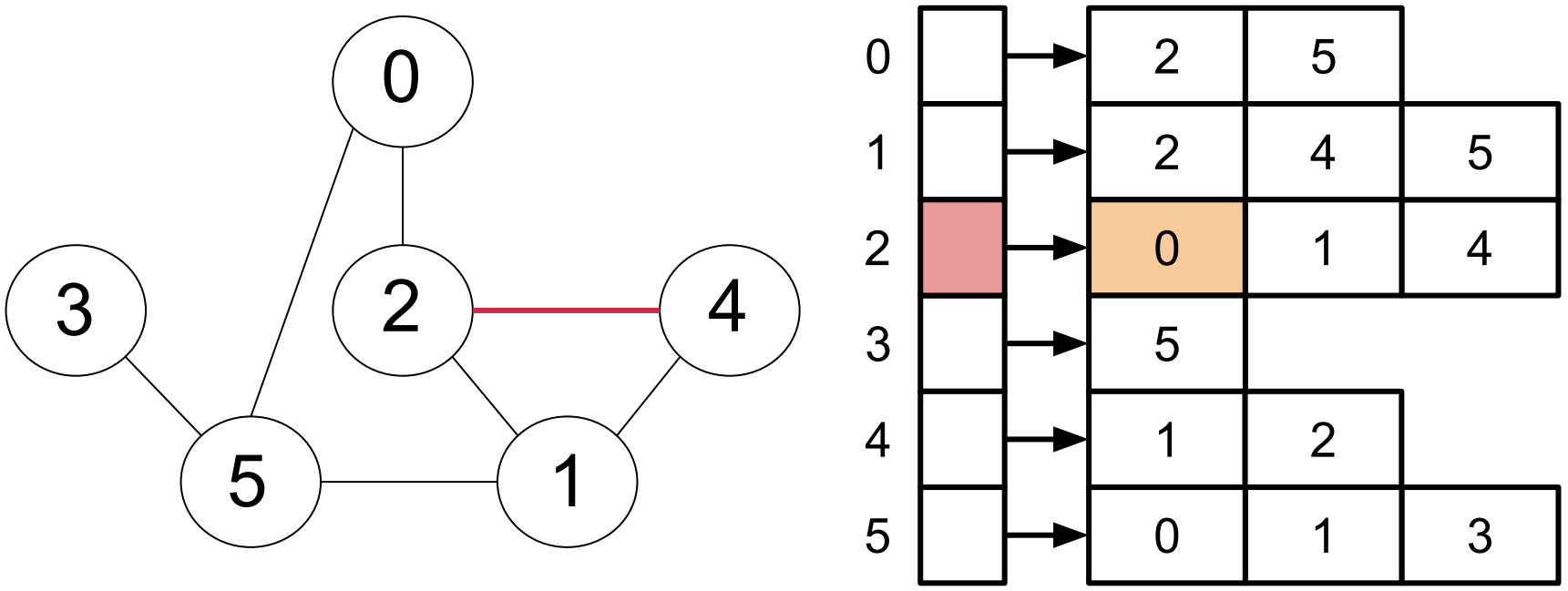
Para verificar a existência de uma aresta agora é necessário percorrer as listas.





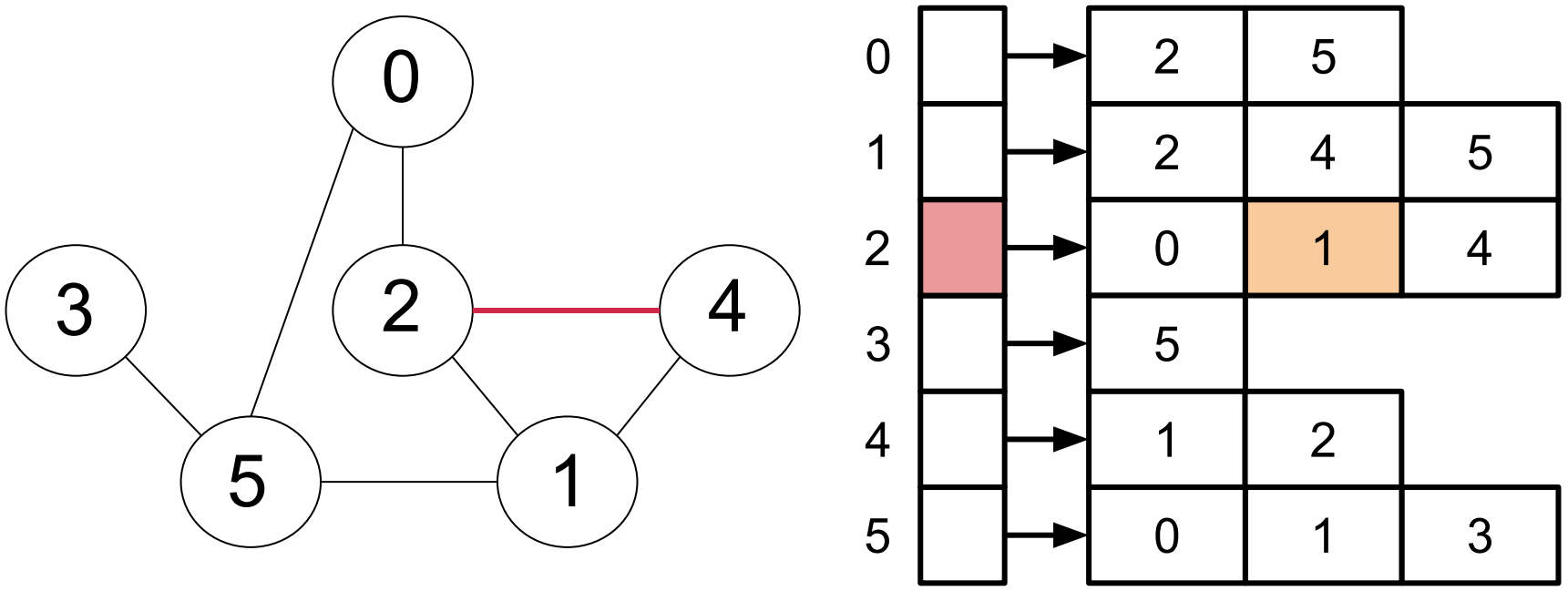
Grafos - Lista de adjacência

Para verificar a existência de uma aresta agora é necessário percorrer as listas.



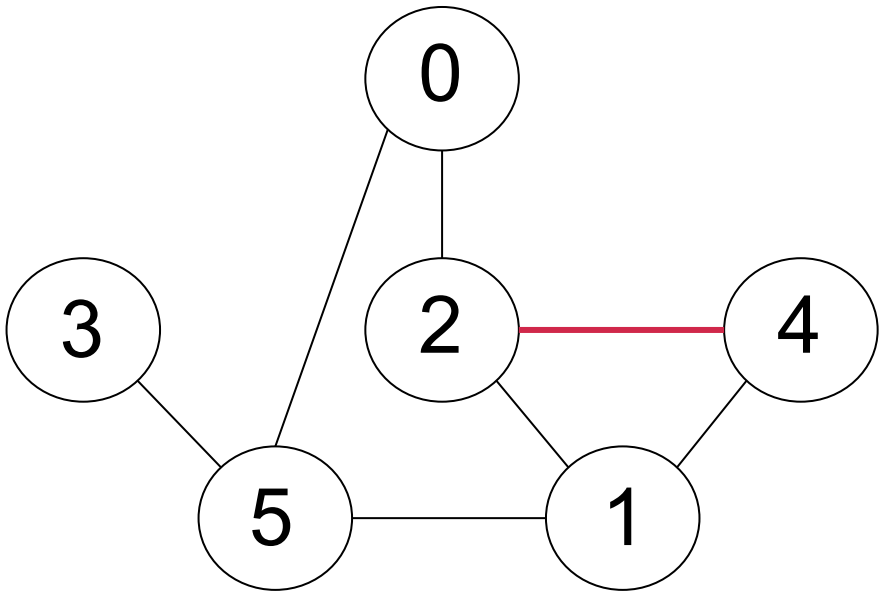
Grafos - Lista de adjacência

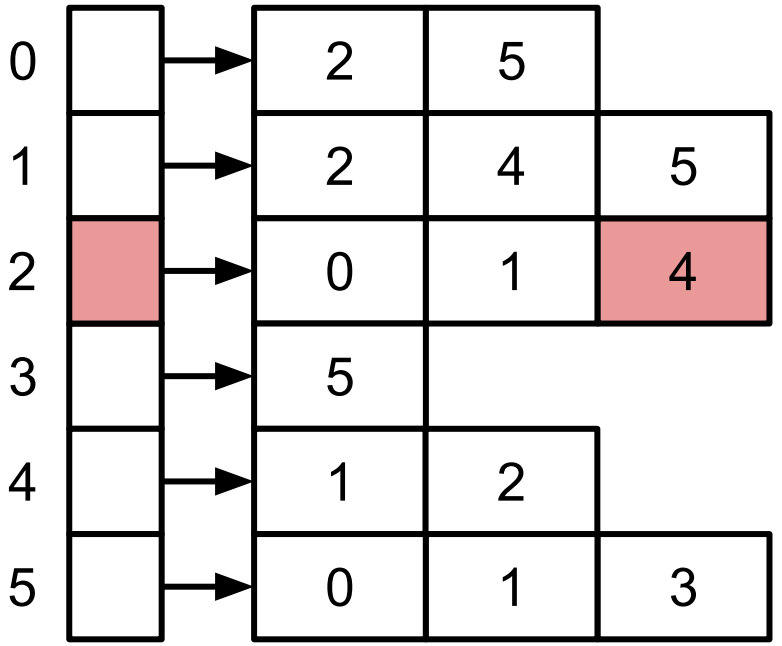
Para verificar a existência de uma aresta agora é necessário percorrer as listas.



Grafos - Lista de adjacência

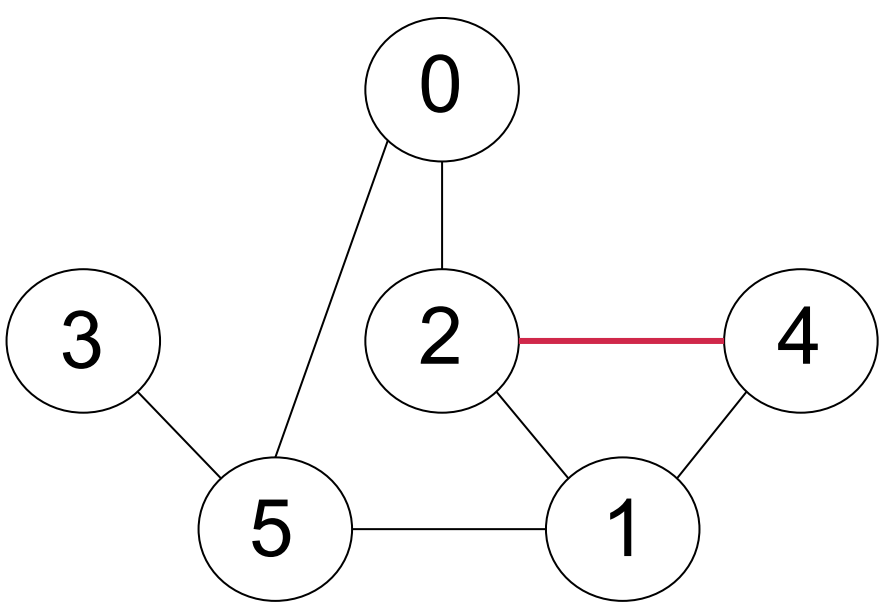
Para verificar a existência de uma aresta agora é necessário percorrer as listas.

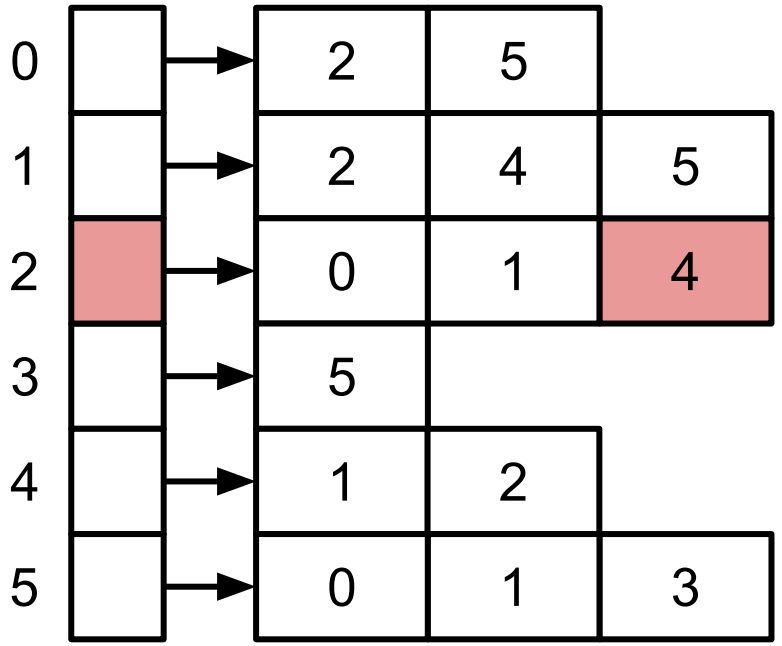




Grafos - Lista de adjacência

Se as listas forem implementadas como listas encadeadas esse processo é realizado em tempo *O(n)*.





Lista de adjacência

Vantagens:

- Ocupa espaço *O(* |V|+|E| *)*. Note que |E| é $O(n^2)$, então no pior caso a lista ocupa espaço proporcional a de uma matriz de adjacência.
- Consultar a vizinhança de um vértice é feita em *O(n)* no pior caso.

Desvantagens:

- Verificar a existência de uma aresta custa *O(n)*.

Roteiro da atividade

Nesta atividade você deverá implementar um programa que:

1. Recebe uma operação a ser realizada com o grafo pela linha de comando. As operações são:
   - "-d" Dados básicos: Deve imprimir na tela, um valor por linha: a quantidade de vértices e de arestas do grafo, o grau mínimo e o máximo.
   - "-n" Vizinhanças: Deve imprimir os vizinhos de cada um dos vértices. Todos os vizinhos de um vértice devem estar na mesma linha separados por um espaço em branco e encerrando com uma quebra de linha.
   - "-k" : Deve imprimir 1 caso o grafo de entrada seja um grafo completo e 0 caso contrário.
2. Recebe os dados de um grafo pela entrada padrão.

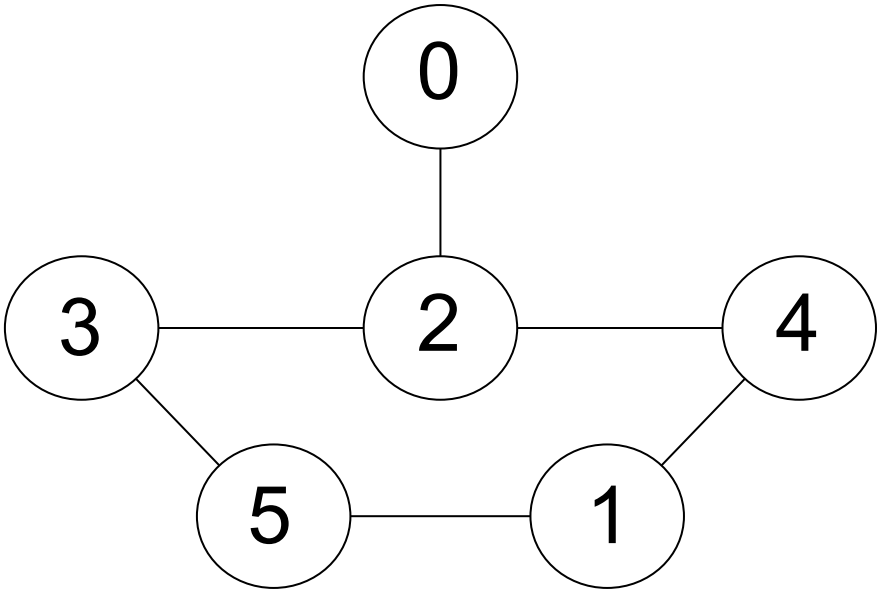
Leitura do grafo

A leitura dos dados do grafo se dará da seguinte forma:

1. Um inteiro *n* indicando quantos vértices o grafo possui.
2. As próximas *n* linhas contém as vizinhos de cada vértice. Um inteiro *m* indicando quantos vizinhos o vértice possi seguidos de *m* inteiros indicando cada vizinho.

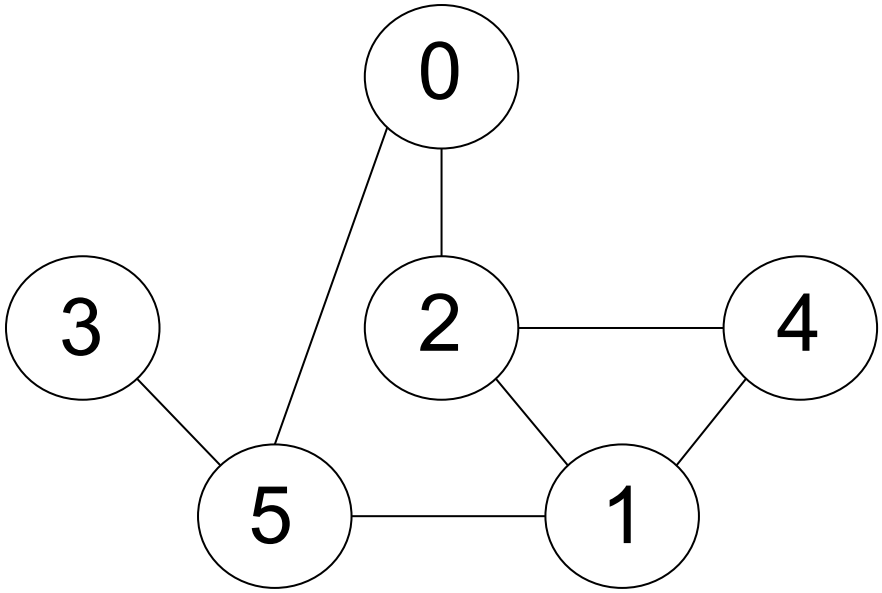
Exemplo de leitura

```
6
1 2
2 4 5
3 0 3 4
2 2 5
2 1 2
2 1 3
```



Exemplo de leitura

```
6
2 2 5
3 2 4 5
3 0 1 4
1 5
2 1 2
3 0 1 3
```

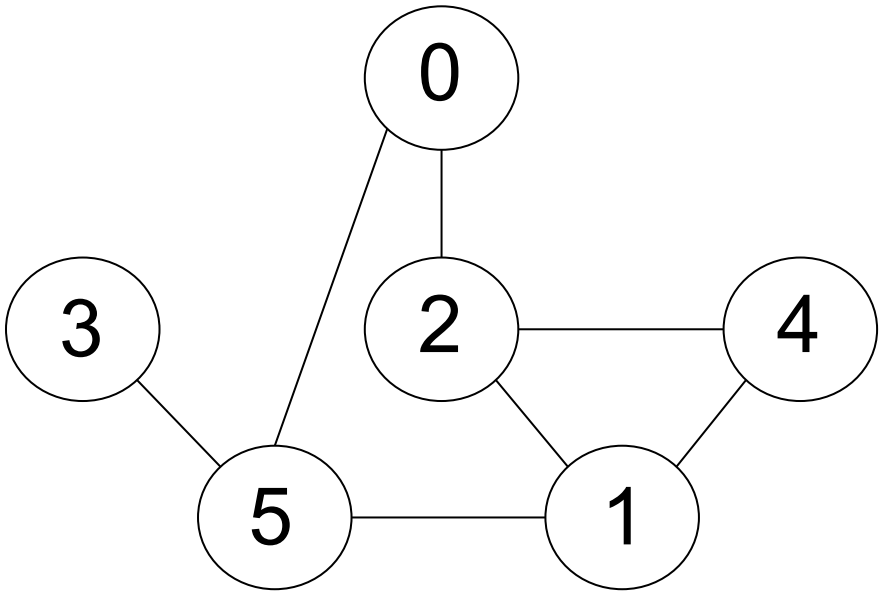


Exemplo de saídas

Suponha que o programa foi executado da seguinte forma:

```
./pa6.out -d
```

Grafo de entrada:



Saída esperada:

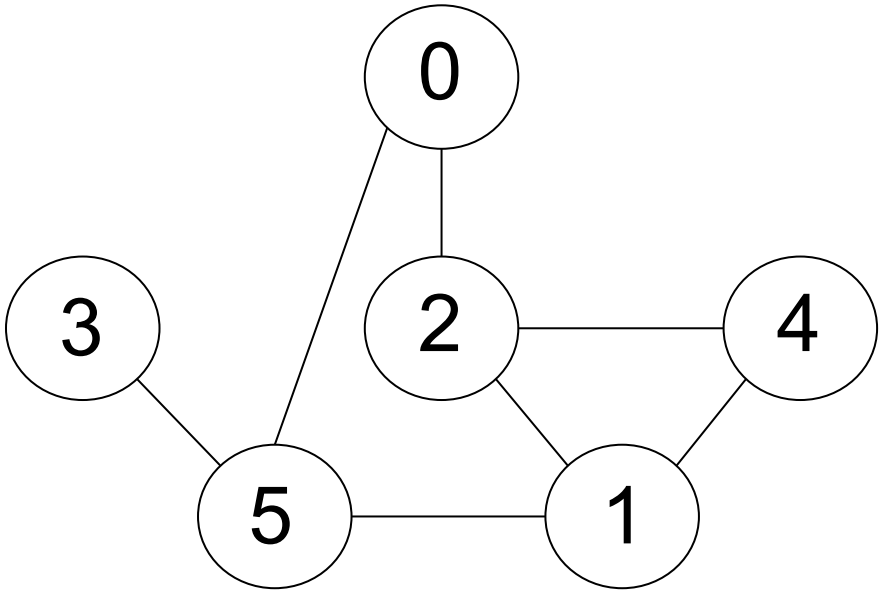
```
6

7

1

3
```

Exemplo de saídas

Suponha que o programa foi executado da seguinte forma:

```
./pa6.out -d
```

Grafo de entrada:



Saída esperada:

```
6
7
1
3
```

Quantidade de vértices

Quantidade de arestas

Grau mínimo

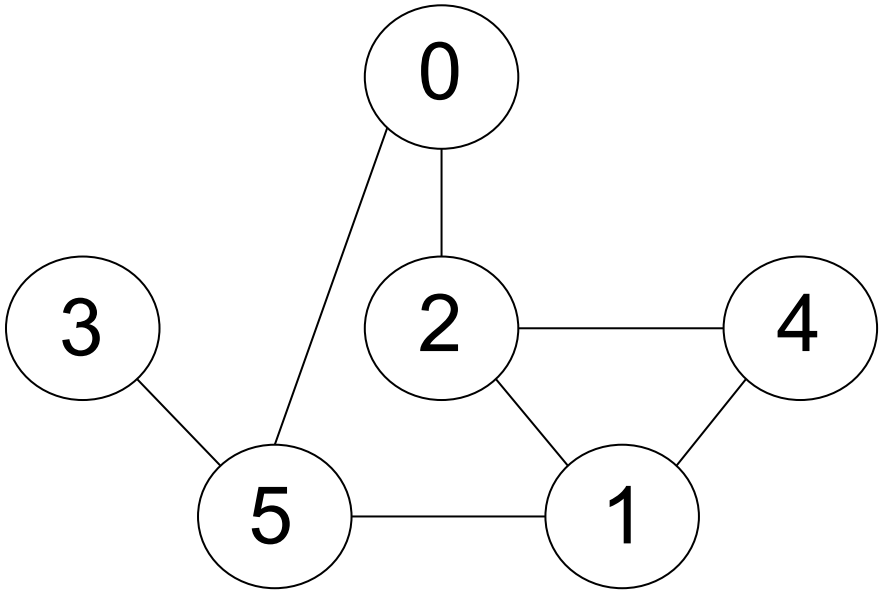
Grau máximo

Exemplo de saídas

Suponha que o programa foi executado da seguinte forma:

```
./pa6.out -n
```

Grafo de entrada:



Saída esperada:

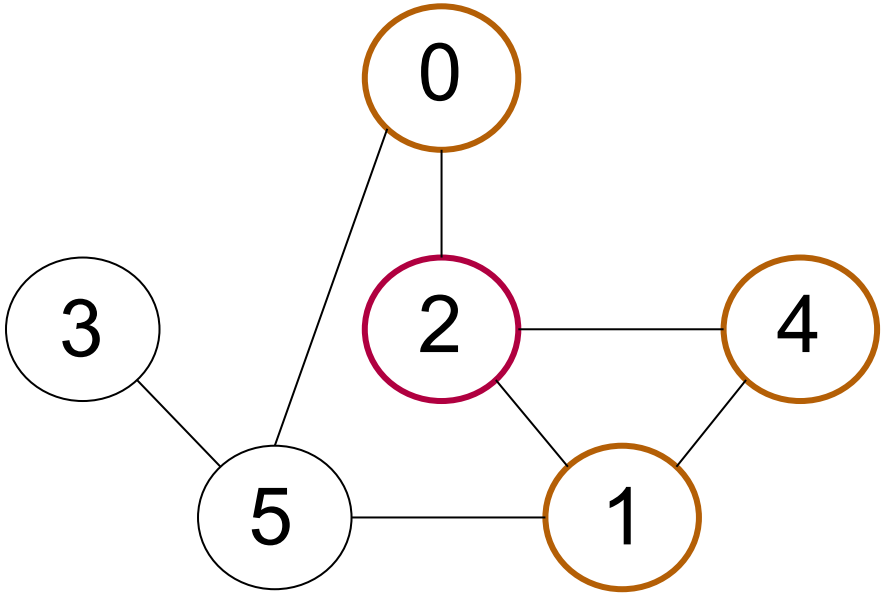
```
2 5

2 4 5

0 1 4

5

1 2

0 1 3
```

Exemplo de saídas

Suponha que o programa foi executado da seguinte forma:

```
./pa6.out -n
```

Grafo de entrada:



Saída esperada:

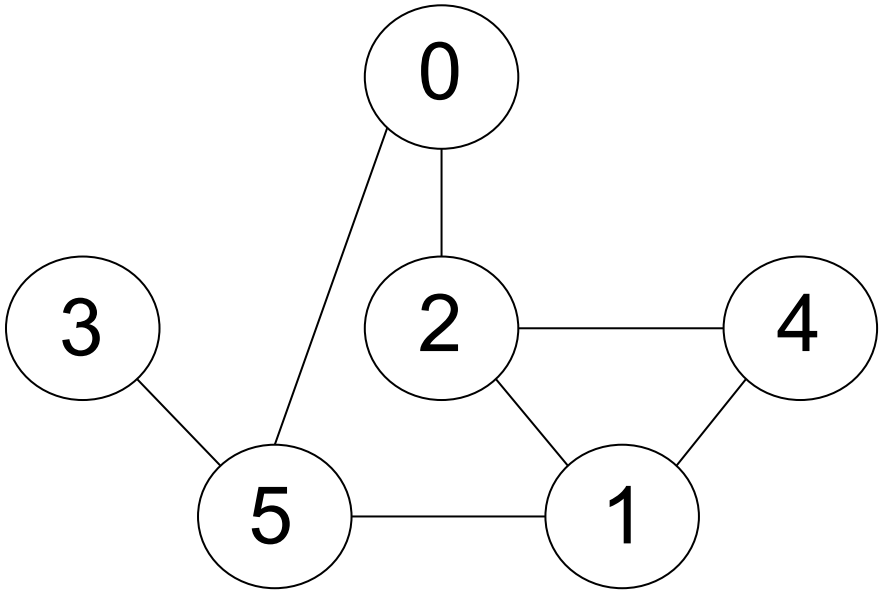
```
2  5
2  4  5
0  1  4
5
1  2
0  1  3
```

Vizinhos do vértice 2

Exemplo de saídas

Note que a saída das vizinhanças deve ser dada na mesma ordem em que os vizinhos aparecem na entrada.

Grafo de entrada:



Saída esperada:

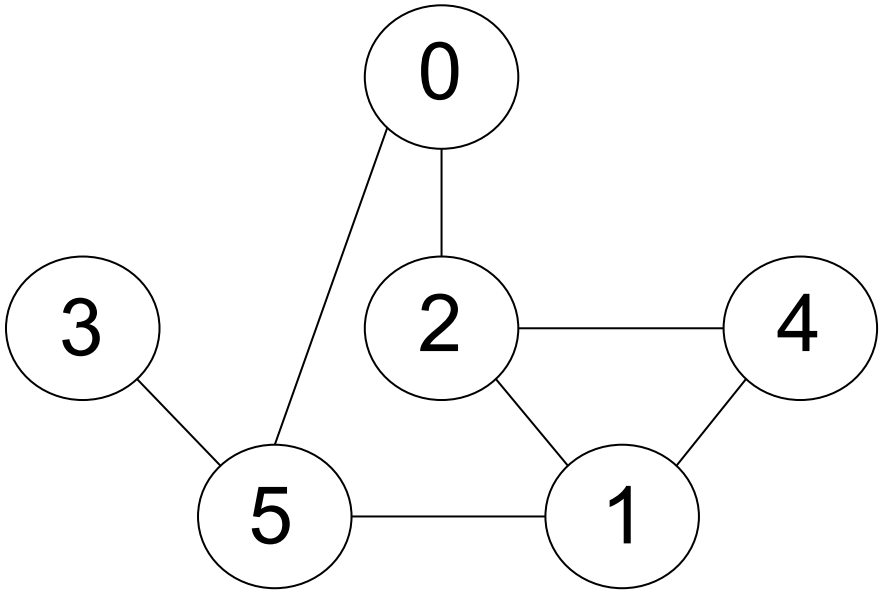
```
2 5
2 4 5
0 1 4
5
1 2
0 1 3
```

Exemplo de saídas

Suponha que o programa foi executado da seguinte forma:

```
./pa6.out -k
```

Grafo de entrada:



Saída esperada:

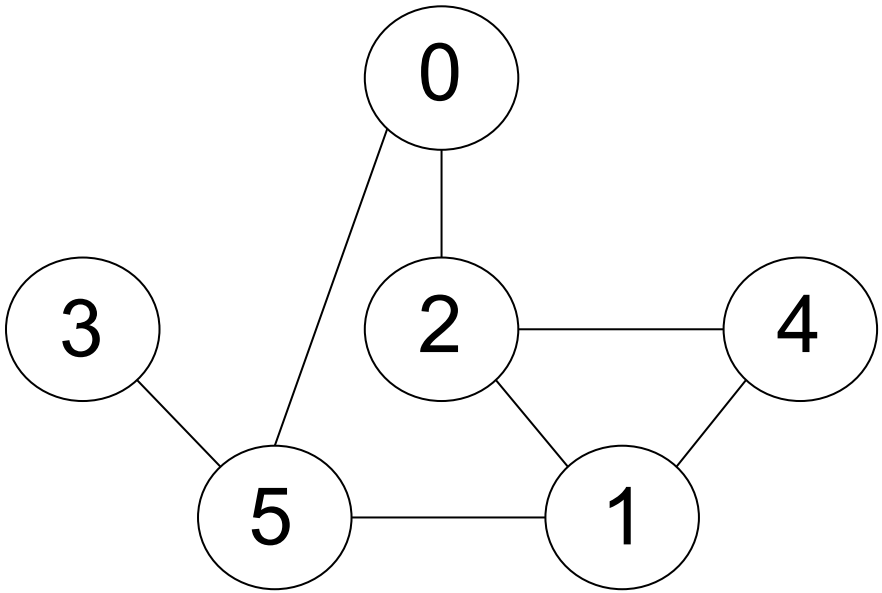
```
0
```

Exemplo de saídas

Suponha que o programa foi executado da seguinte forma:

```
./pa6.out -k
```

Grafo de entrada:



Saída esperada:

```
0
```

O grafo de entrada não é um grafo completo

Roteiro da atividade

Observações sobre a implementação do grafo:

- **Em C:** Você deve implementar o TAD Grafo bem como as funções propostas no arquivo Grafo.h
- **Em C++:** Você deve implementar a classe do arquivo Grafo.hpp.
- Em ambos os casos você não deve alterar a interface das funções ou métodos(você não pode adicionar campos ou métodos a classe, nem modificar as assinaturas dos métodos ou funções), mas pode adicionar os includes de seus TADs.
- O grafo deve ser implementado utilizando uma lista de adjacência.
- Sua lista de adjacência deve ser implementada utilizando Listas encadeadas.

Grafo.h - Interface do TAD Grafo e suas funções

```
typedef struct s_grafo Grafo;

Grafo* NovoGrafo();
void DeletaGrafo(Grafo* g);

void InsereVertice(Grafo* g);
void InsereAresta(Grafo* g, int v, int w);

int QuantidadeVertices(Grafo* g);
int QuantidadeArestas(Grafo* g);

int GrauMinimo(Grafo* g);
int GrauMaximo(Grafo* g);

void ImprimeVizinhos(Grafo* g, int v);
```

Grafo.hpp - Interface da classe Grafo

```
class Grafo{
    public:
        Grafo();
        ~Grafo();
        void InsereVertice();
        void InsereAresta(int v, int w);
        int QuantidadeVertices();
        int QuantidadeArestas();
        int GrauMinimo();
        int GrauMaximo();
        void ImprimeVizinhos(int v);
    private:
        ListaAdjacencia vertices;
};
```

Lista encadeada

Observações sobre a implementação das listas encadeadas:

- Você pode se basear no TAD visto em sala.
- Note que nem todas as funções do TAD visto em aula serão necessárias nessa prática, e talvez você precise implementar funcionalidades novas para seu TAD.

Submissão

- A submissão será feita por **VPL**. Certifique-se de seguir as instruções do tutorial disponibilizado no moodle.
- O seu arquivo executável **DEVE** se chamar **pa6.out** e deve estar localizado na pasta **bin**.
- Seu código será compilado com o comando:

  make all
- Você **DEVE** utilizar a estrutura de projeto abaixo junto ao Makefile :

```
- PA6
   |- src
   |- bin
   |- obj
   |- include
   Makefile
```