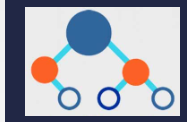


Estrutura de Dados 2

Aula 07 – Grafos: Conceitos e implementação – parte 1

Antonio Angelo de Souza Tartaglia
angelot@ifsp.edu.br

Estrutura de Dados 2

Grafos

- Definição:
  - É um modelo matemático que representa relações entre objetos;
  - São utilizados na definição e/ou resolução de problemas de diversas áreas.

- Grafos em computação:
  - É uma forma de solucionar problemas computáveis;
  - Buscam o desenvolvimento de algoritmos mais eficientes.

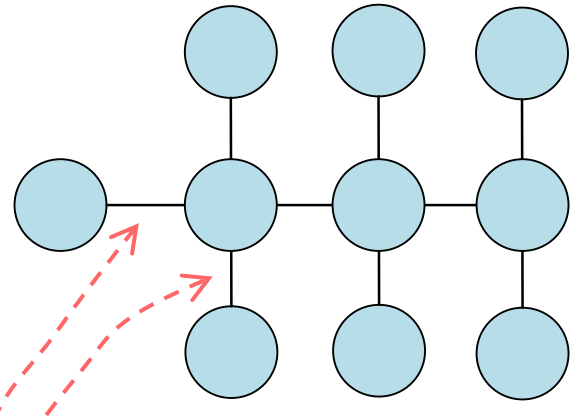
- Exemplos:
  - Qual a melhor rota do IFSP – Guarulhos até a sua casa?
  - Em uma rede social, duas pessoas têm um amigo em comum?

Estrutura de Dados 2

Grafos

- Exemplos modelados com grafos:

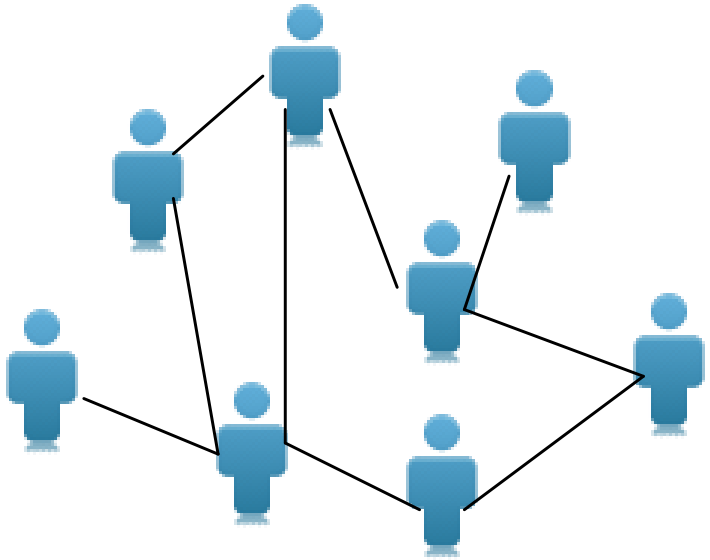
Cada casa e a central de abastecimento, representam um vértice.



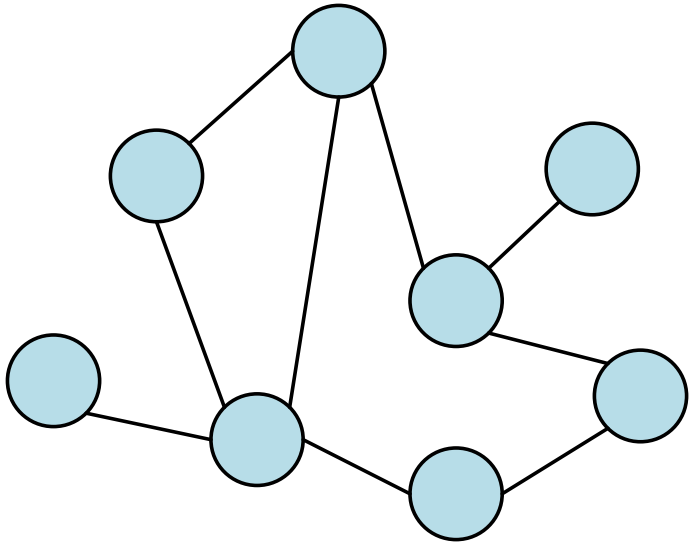
O sistema de transporte de água é representado pelas ligações ou arestas.

Estrutura de Dados 2

Grafos



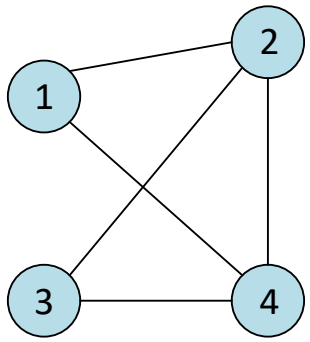
Conjunto de amigos, cada um deles representando um vértice em um grafo.



Os relacionamentos são representados pelas arestas.

Grafos

- O Grafo é sempre definido como um conjunto de vértices e um conjunto de arestas que conectam qualquer par de vértices.
  - G = (V, A)
- Onde:
  - V – é o conjunto de vértices (não pode ser vazio);
  - A – conjunto de arestas



G(V, A)

V = {1, 2, 3, 4}

A = {{1, 2}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}

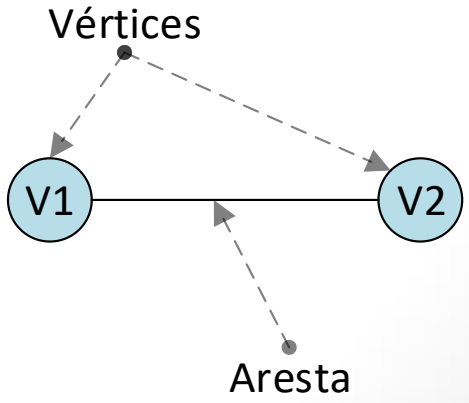
Estrutura de Dados 2

Grafos

Sempre será uma representação do objeto que é alvo do problema.

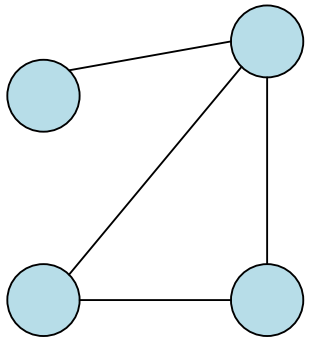
- Vértice:
  - É cada uma das entidades representadas em um grafo;
  - Dependem da natureza do problema. Podem ser pessoas, casas, cargos de uma empresa, esquinas de uma cidade em um mapa, etc.
  - Dois vértices são adjacentes se existir uma aresta ligando-os

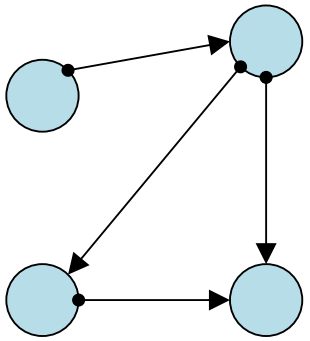
- Aresta:
  - Também chamada de **Arco**;
  - Está associada a dois vértices (V1, V2);
  - Faz a ligação entre eles, ou seja, diz qual a relação entre eles.



Grafos

- Direção das arestas
  - Grafo direcionado ou dígrafo:
    - Existe uma orientação quanto ao sentido da Aresta;
    - Se uma Aresta liga “A” e “B”, podemos ir de “A” para “B”, mas não ao contrário.
  - Grafo não direcionado:
    - Não existe nenhuma orientação quanto ao sentido da Aresta. Podemos ir de “A” para “B” ou de “B” para “A”.





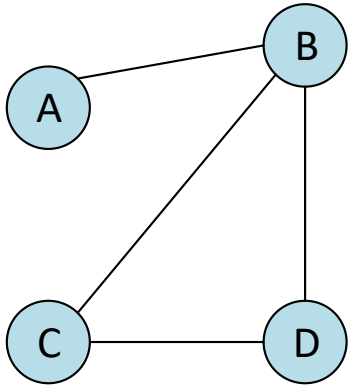
Grafos

- Grau de um Vértice
  - É o número de Arestas que chegam ou partem dele.
- No caso dos dígrafos, temos:
  - Grau de entrada: Arestas que chegam ao Vértice;
  - Grau de saída: Arestas que partem do Vértice.

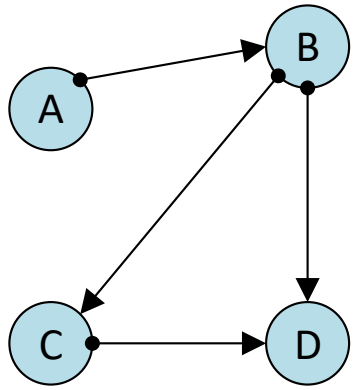
Aresta

Grau

G(A) = 1
G(B) = 3
G(C) = 2
G(D) = 2



Grafo



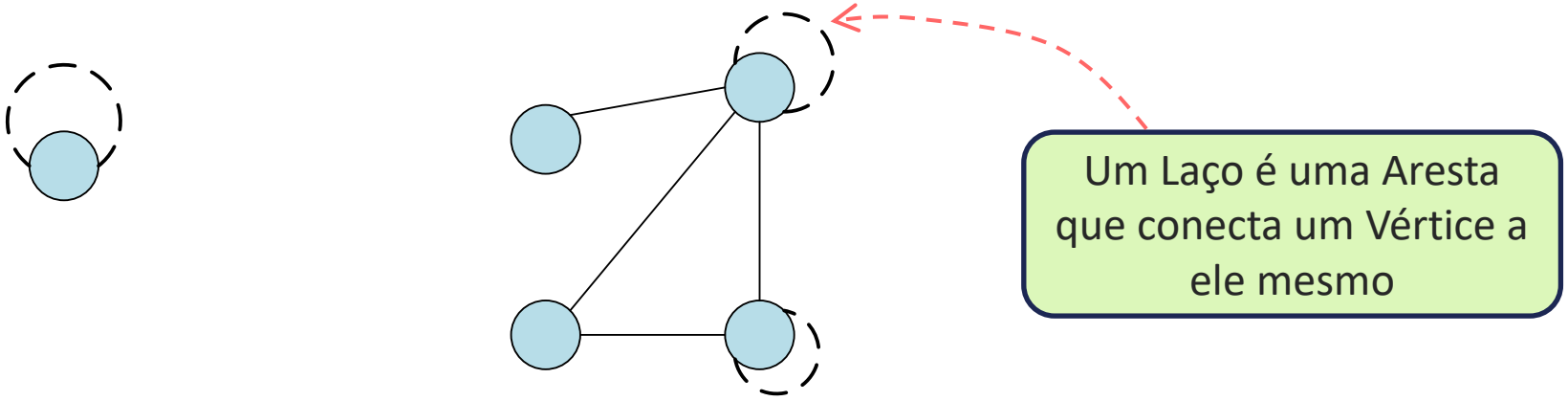
Dígrafo

Grau de entrada	Grau de saída
G(A) = 0	G(A) = 1
G(B) = 1	G(B) = 2
G(C) = 1	G(C) = 1
G(D) = 2	G(D) = 0

Estrutura de Dados 2

Grafos

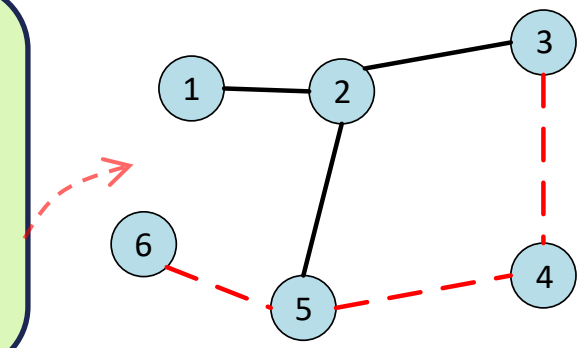
- Laço
  - Uma Aresta é chamada de **Laço** se seu Vértice de partida é o mesmo que o de chegada, ou seja, a Aresta conecta o Vértice com ele mesmo (v, v).

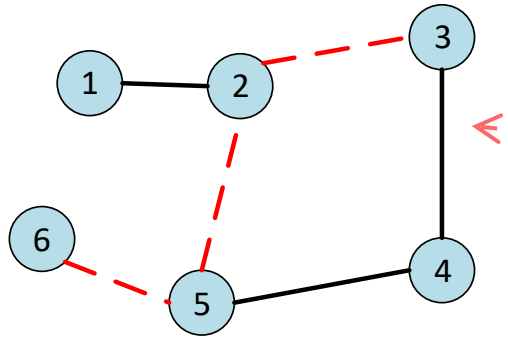


Grafos

- Caminho:
  - É uma sequencia de Vértices de modo que existe sempre uma Aresta ligando o Vértice anterior com o seguinte.
- Caminho Simples:
  - Nenhum dos Vértices no caminho se repete.
- Comprimento do Caminho:
  - É o número de Arestas que o Caminho usa.

Usando como exemplo um mapa, é possível traçar uma rota de estradas, saindo do ponto 3, indo até o ponto 6.

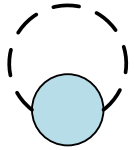




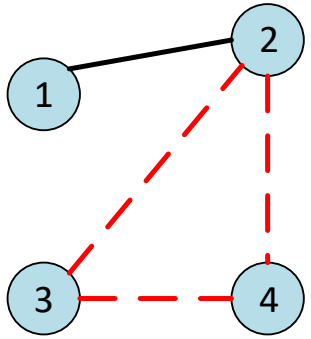
É possível ter mais de um caminho no Grafo.

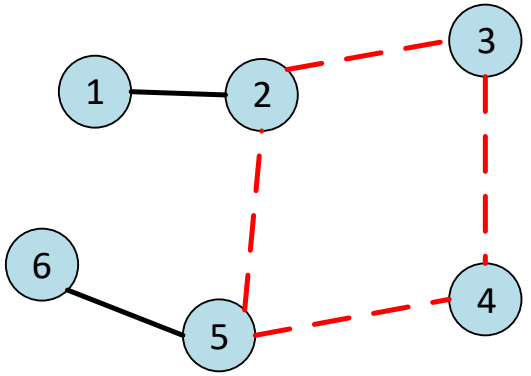
Grafos

- Ciclo:
  - É um caminho que começa e termina no mesmo Vértice;
  - Um **Laço** é um ciclo de comprimento 1.



- Grafo Aciclico:
  - Não contém ciclos simples, onde cada Vértice aparece apenas uma vez, diferentemente os Vértices podem aparecer mais de uma vez.

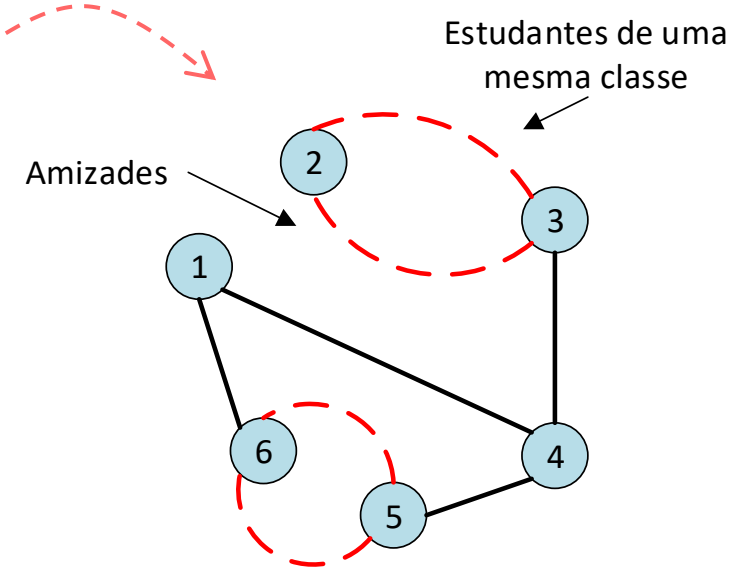




Grafos

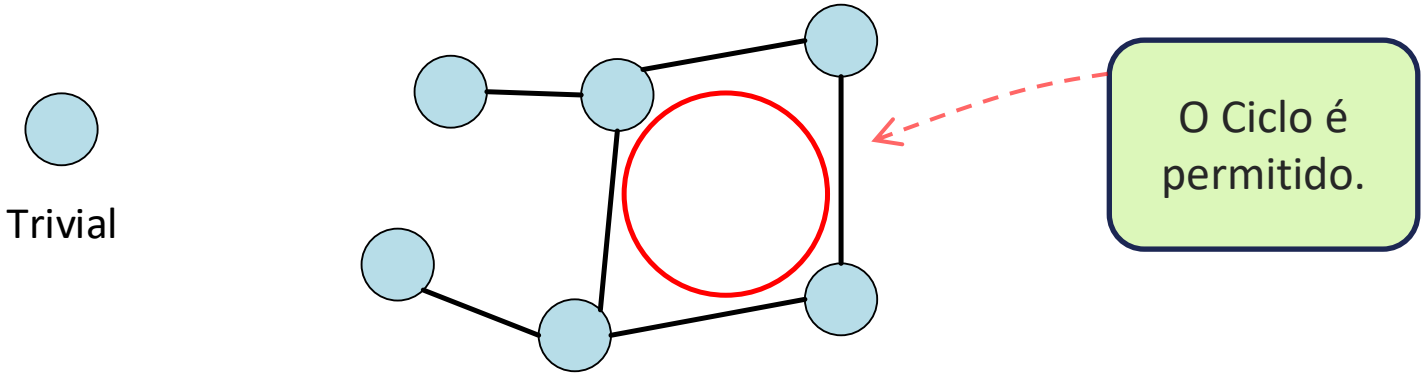
- Arestas Multiplas:
  - Também chamado de **Multigrafo**;
  - É um Grafo que permite mais de uma aresta conectando o mesmo par de Vértices;
  - Neste caso, as Arestas são ditas **“Paralelas”**.

Usado quando se modela duas características diferentes no mesmo Grafo



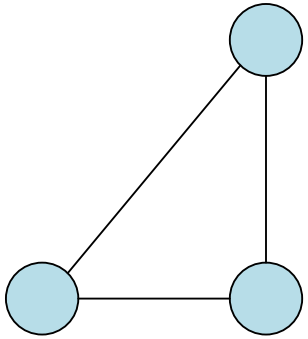
Tipos de Grafos

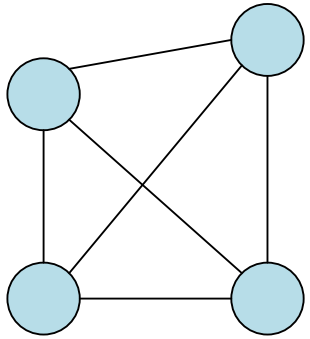
- Grafo Trivial
  - É um Grafo com um único Vértice e sem Arestas.
- Grafo Simples
  - É um Grafo não direcionado, sem Laços e sem Arestas Paralelas.



Tipos de Grafos

- Grafo Completo
  - É um Grafo simples onde cada Vértice se conecta a todos os outros Vértices.

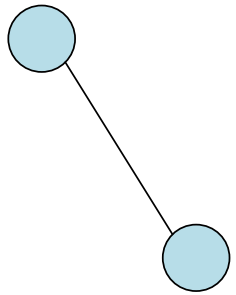




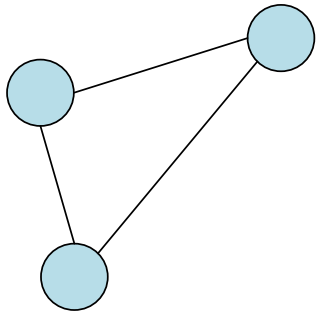
Tipos de Grafos

- Grafo Regular
  - É um Grafo onde todos os Vértices possuem o mesmo Grau.

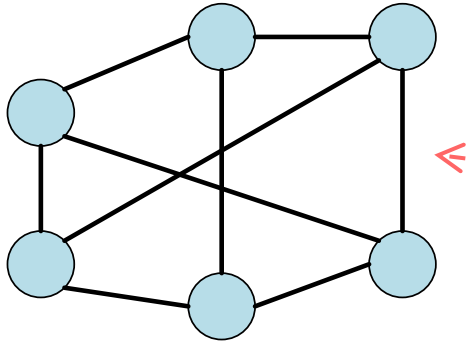
Um Grafo completo é sempre Regular



Grau 1



Grau 2



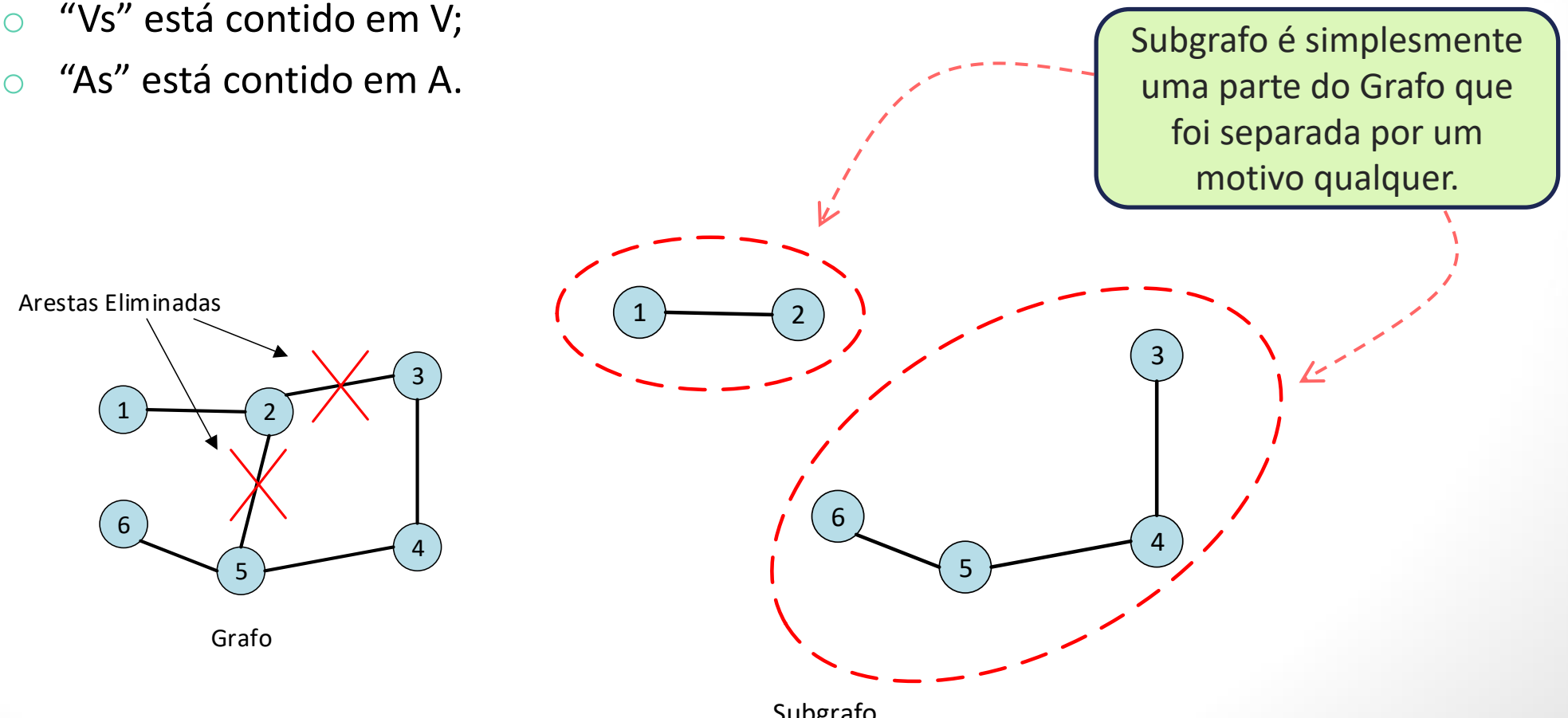
Grau 3

Grafo Regular: todos os Vértices possuem o mesmo número de conexões.

Regular: Todos os Vértices têm 3 Arestas, mas não é completo: os Vértices não estão todos interligados entre si.

Tipos de Grafos

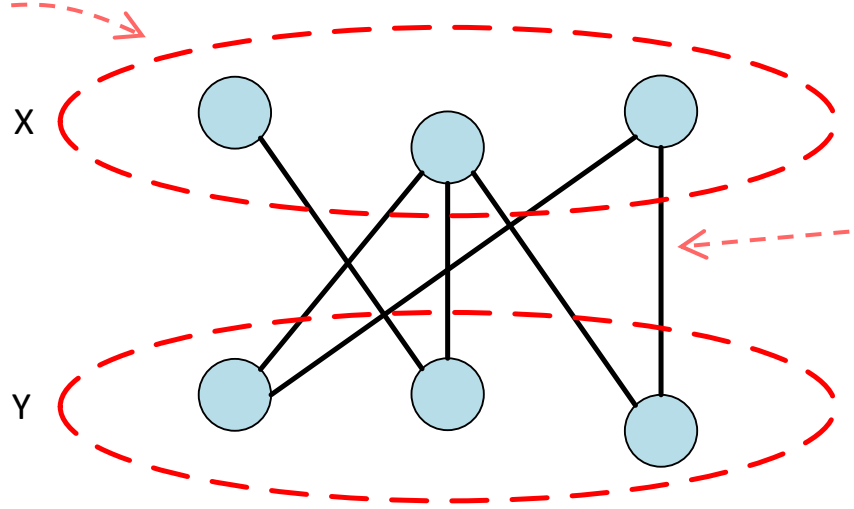
- Subgrafo
  - Um Grafo Gs(Vs, As), é chamado de Subgrafo de G(V, A) se:
    - “Vs” está contido em V;
    - “As” está contido em A.



Tipos de Grafos

- Grafo Bipartido
  - É um Grafo cujos Vértices podem ser divididos em 2 conjuntos;
  - Nesse caso, as Arestas ligam os Vértices que estão em conjuntos diferentes, nunca ligando Vértices do mesmo conjunto.

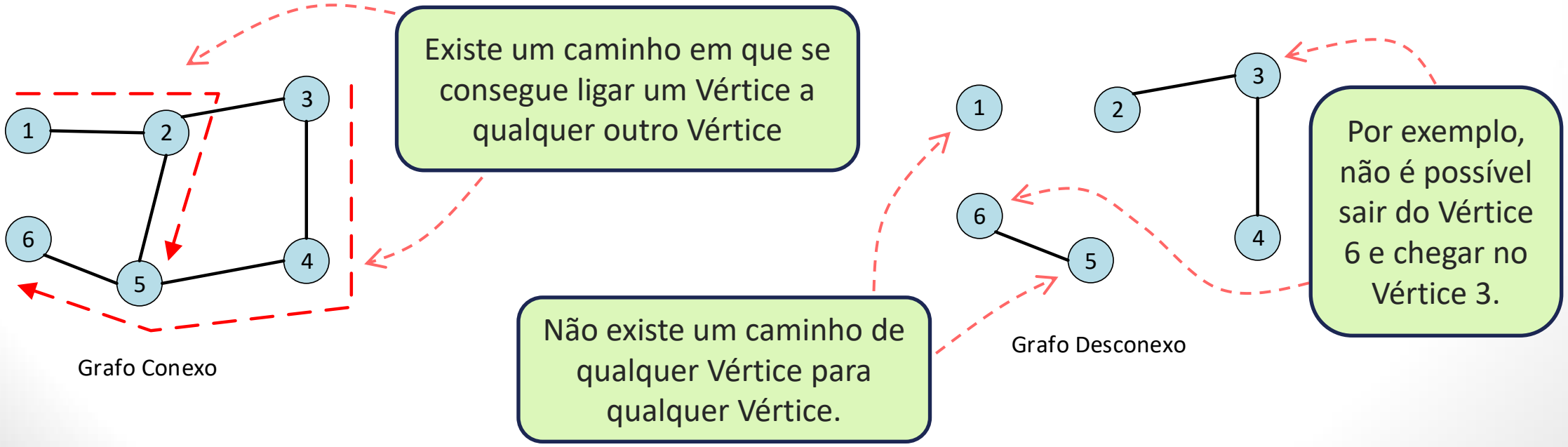
Note que não há Arestas ligando 2 Vértices que estejam no mesmo conjunto



As Arestas vão apenas ligar os Vértices de um conjunto aos Vértices do outro conjunto.

Tipos de Grafos

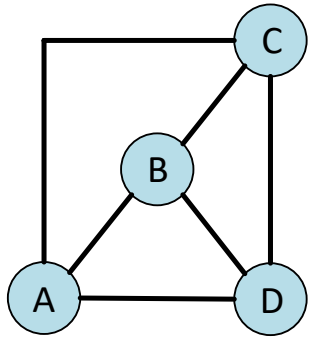
- Grafo Conexo:
  - Existe um caminho partindo de qualquer Vértice até qualquer outro Vértice do Grafo.
- Grafo Desconexo:
  - Não existe caminho ligando dois Vértices.

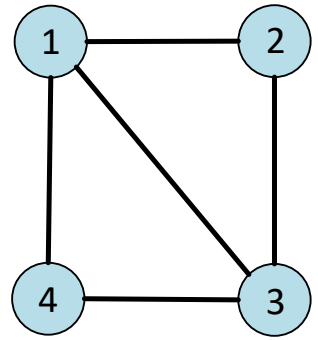


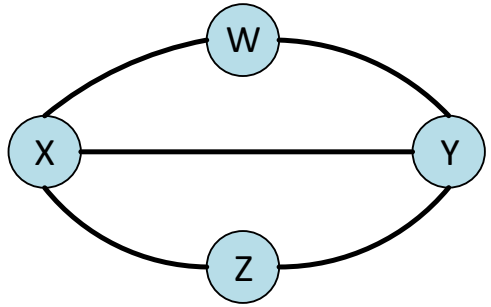
Tipos de Grafos

- Grafos Isomorfos:
- Dois Grafos, G1(V1, A1) e G2(V2, A2), são ditos **Isomorfos** se existe uma função que faça o mapeamento de Vértices e Arestas de modo que os dois Grafos se tornem coincidentes.

Grau
F(1) = A
F(2) = B
F(3) = C
F(4) = D





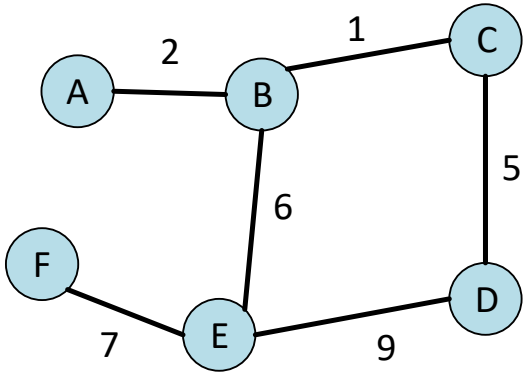


Grau
F(1) = X
F(2) = W
F(3) = Y
F(4) = Z

São aparentemente diferentes, porém são iguais. São os mesmos Grafos, mas estão nomeados de formas diferentes. Só é necessário relacionar os Vértices entre os Grafos para dizer que são Grafos iguais.

Tipos de Grafos

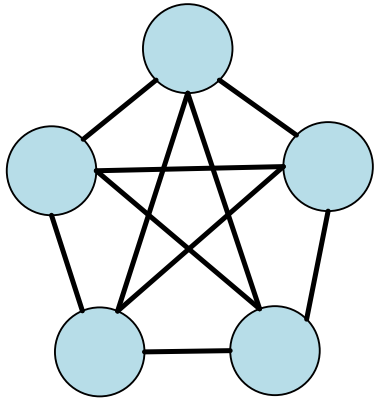
- Grafo Ponderado:
  - É o Grafo que possui **Pesos** associados a cada uma de suas Arestas.



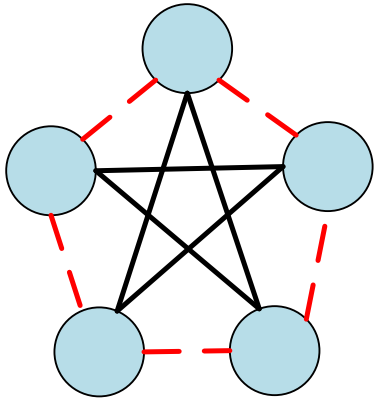
Exemplo: Em um mapa o peso pode definir a distância entre dois pontos...

Tipos de Grafos

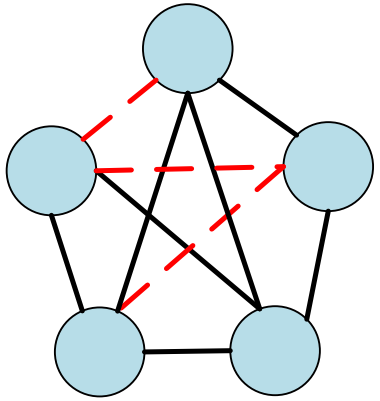
- Grafo Hamiltoniano:
  - É o Grafo que possui um caminho que visita cada Vértice apenas uma vez;
  - Sua detecção é uma tarefa extremamente árdua;
  - O Ciclo Hamiltoniano é o Ciclo que visita cada Vértice apenas uma vez, e volta para a origem.



Grafo Hamiltoniano



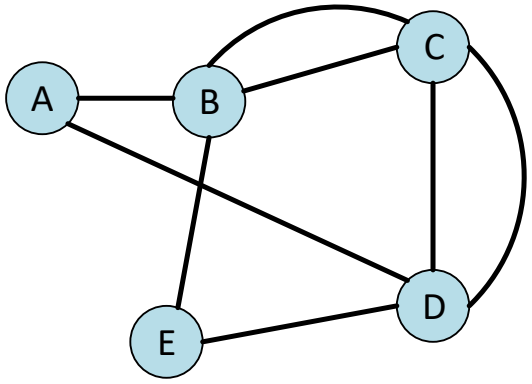
Ciclo Hamiltoniano



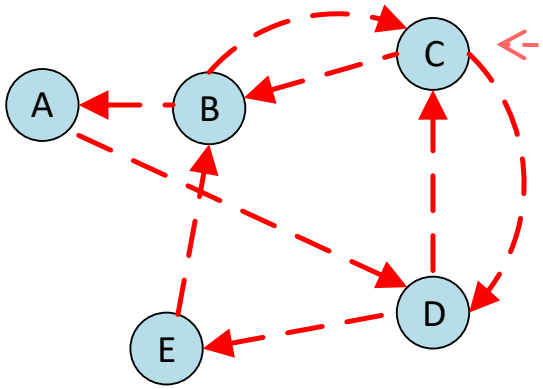
Caminho Hamiltoniano

Tipos de Grafos

- Grafo Euleriano:
  - É o Grafo que possui um **ciclo** que visita cada Aresta **apenas uma vez**.



Grafo Euleriano

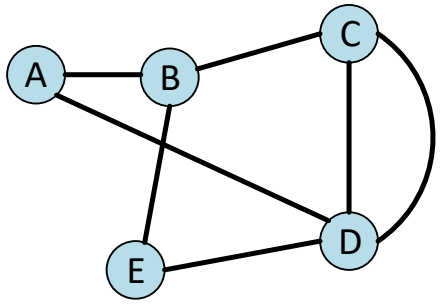


Ciclo Euleriano
CDCBADEBC

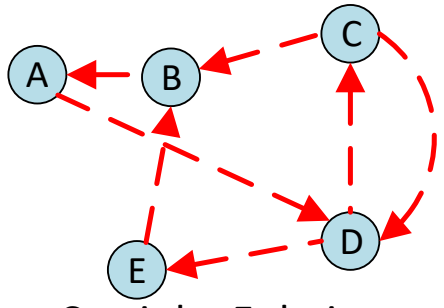
É possível passar por um Vértice mais de uma vez, mas pela Aresta apenas uma vez somente, e percorrendo todo o Caminho.

Tipos de Grafos

- Grafo Semi-Euleriano:
  - É o Grafo que possui um **Caminho** que visita cada Aresta apenas uma vez.



Grafo Semi-Euleriano



Caminho Euleriano
CDCBADEB

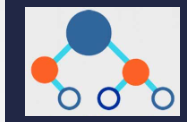
Representação de Grafos

- Como representar um Grafo no Computador?
  - Duas abordagens são muito utilizadas:
    - Matriz de Adjacências;
    - Lista de Adjacências.

Para grafos muito conectados, com muitas arestas.

Utilizada para grafos muito esparsos

- A representação a ser utilizada depende muito da aplicação.

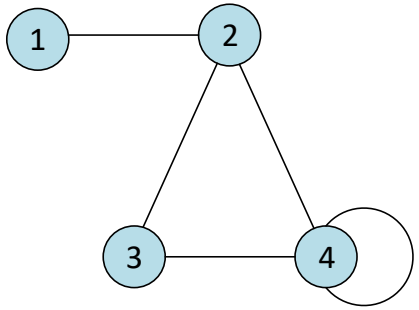


Representação de Grafos

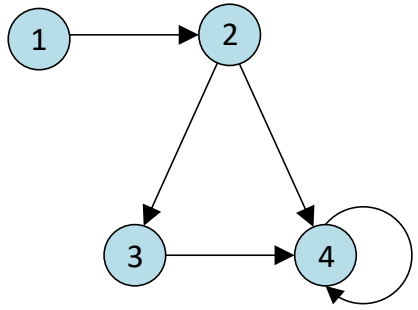
- Matriz de Adjacências:
  - Uma matriz N x N é utilizada para armazenar o Grafo, onde N é o número de Vértices;
    - Alto custo computacional, $O(n^2)$.
  - Uma Aresta é representada por uma **marca** na posição (i, j) da matriz;
    - Aresta liga o Vértice i ao j.

Representação de Grafos

- Matriz de Adjacências:



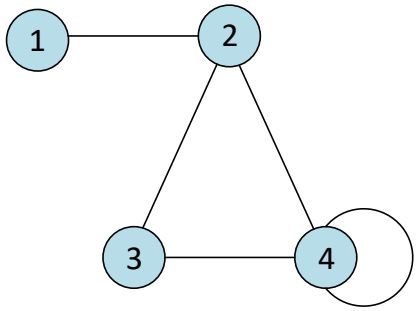
	1	2	3	4
1	0	1	0	0
2	1	0	1	1
3	0	1	0	1
4	0	1	1	1

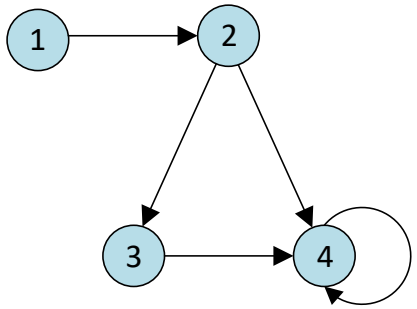


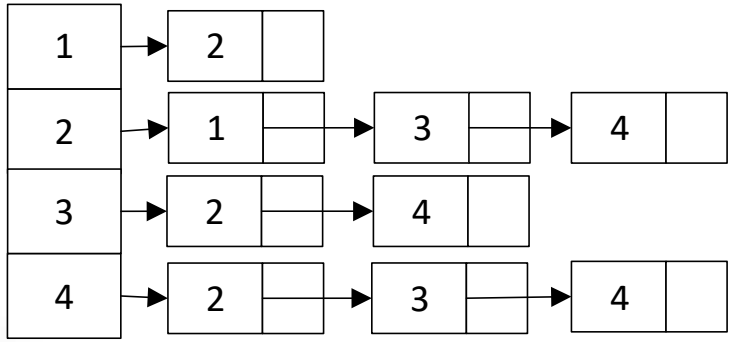
	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	0	0	0	1

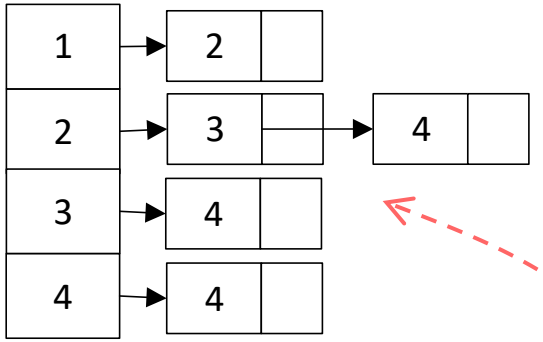
Representação de Grafos

- Lista de Adjacências:









A Lista pode ser estática ou dinâmica

Definindo o TAD Grafo usando Lista de Adjacências

- grafo.h – Definir:
  - Os protótipos das funções;
  - O ponteiro Grafo;

- grafo.c - Definir:
  - O tipo de dado grafo. Para isso será definida a lista de adjacências.
  - Implementar as suas funções

Definindo o TAD Grafo usando Lista de Adjacências

```c
//No arquivo grafo.h
typedef struct grafo Grafo;
```

```c
//No programa principal
Grafo *gr;
```

```c
//No arquivo grafo.c
#include <stdio.h>
#include <stdlib.h>
#include "grafo.h"

//definição do tipo grafo
struct grafo{
    int eh_ponderado;
    int nro_vertices;
    int grau_max;
    int **arestas;
    int **pesos;
    int *grau;
};
```

Matriz dinâmica para armazenar as conexões

Matriz dinâmica para pesos caso queira trabalhar com Grafos Ponderados

Define para cada Vértice, quantas Arestas ele já possui.

Definindo o TAD Grafo usando Lista de Adjacências

```c
//No arquivo grafo.h
Grafo *cria_Grafo(int nro_vertices, int grau_max, int eh_ponderado);
```

```c
//No programa principal
Grafo *gr;
gr = cria_Grafo(10, 7, 0);
```

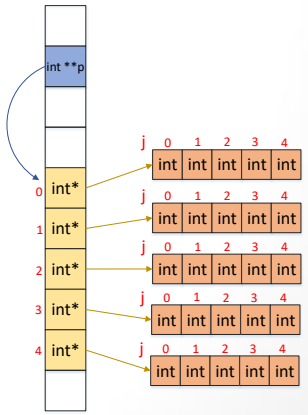
```c
//No arquivo grafo.c
Grafo *cria_Grafo(int nro_vertices, int grau_max, int eh_ponderado){
    Grafo *gr = (Grafo*) malloc(sizeof(struct grafo));
    if(gr != NULL){
        int i;
        gr->nro_vertices = nro_vertices;
        gr->grau_max = grau_max;
        gr->eh_ponderado = (eh_ponderado != 0)? 1 : 0;
        gr->grau = (int*) calloc(nro_vertices, sizeof(int));
        gr->arestas = (int**) malloc(nro_vertices * sizeof(int*));
        for(i = 0; i < nro_vertices; i++){
            gr->arestas[i] = (int*) malloc(grau_max * sizeof(int));
        }
        if(gr->eh_ponderado){
            gr->pesos = (float**) malloc(nro_vertices * sizeof(float*));
            for(i = 0; i < nro_vertices; i++){
                gr->pesos[i] = (float*) malloc((grau_max * sizeof(float)));
            }
        }
    }
    return gr;
}
```

Verificação para garantir que a informação sempre será 0 ou 1, V ou F

calloc para entrar tudo zerado

Matriz de Arestas

Matriz de Pesos



As verificações de alocação foram suprimidas por questão de espaço, devem ser implementadas.

Definindo o TAD Grafo usando Lista de Adjacências

- Liberando um Grafo

```c
//No arquivo grafo.h
void libera_grafo(Grafo *gr);
```

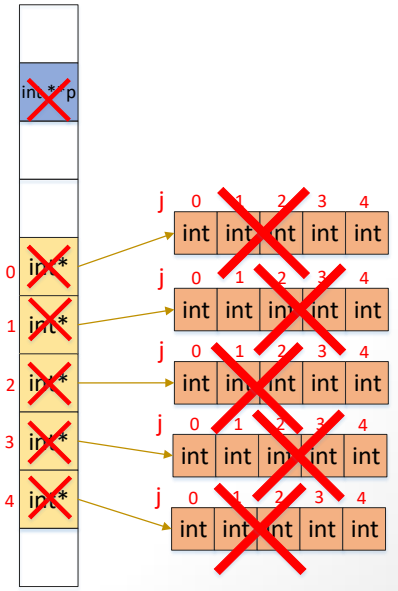
```c
//No arquivo grafo.c
void libera_grafo(Grafo *gr){
    if(gr != NULL){
        int i;
        for(i = 0; i < gr->nro_vertices; i++){
            free(gr->arestas[i]);
        }
        free(gr->arestas);

        if(gr->eh_ponderado){
            for(i = 0; i < gr->nro_vertices; i++){
                free(gr->pesos[i]);
            }
            free(gr->pesos);
        }
        free(gr->grau);
        free(gr);
    }
}
```

```c
//No programa principal
Grafo *gr;
gr = cria_Grafo(10, 7, 0);
....
....
....
libera_grafo(gr);
```

Libera Matriz de Arestas

Libera Matriz de Pesos



Definindo o TAD Grafo usando Lista de Adjacências

- Inserindo uma Aresta no Grafo

```
//No arquivo grafo.h
int insere_Aresta(Grafo *gr, int orig, int dest, int eh_Digrafo, float peso);
```

```
//No programa principal
Grafo *gr;
gr = cria_Grafo(10, 7, 0);
insere_Aresta(gr, 0, 1, 0, 0);
insere_Aresta(gr, 1, 3, 0, 0);
....
....
....
libera_grafo(gr);
```

```
//No arquivo grafo.c
int insere_Aresta(Grafo *gr, int orig, int dest, int eh_digrafo, float peso){
    if(gr == NULL)
        return 0;
    if(orig < 0 || orig >= gr->nro_vertices)
        return 0;
    if(dest < 0 || dest >= gr->nro_vertices)
        return 0;
    gr->arestas[orig][gr->grau[orig]] = dest;
    if(gr->eh_ponderado){
        gr->pesos[orig][gr->grau[orig]] = peso;
    }
    gr->grau[orig]++;
    if(eh_digrafo == 0){
        insere_Aresta(gr, dest, orig, 1, peso);
    }
    return 1;
}
```

Verifica se existem e se são Vértices válidos

Insere no final da linha

Insere outra Aresta se não for Digrafo, depois chama recursivamente passando a ordem inversa, e informa que agora é Digrafo

Definindo o TAD Grafo usando Lista de Adjacências

- Removendo uma Aresta no Grafo

```
//No arquivo grafo.h
int remove_aresta(Grafo *gr, int orig, int dest, int eh_digrafo)
```

```
//No programa principal
Grafo *gr;
gr = cria_Grafo(10, 7, 0);
insere_Aresta(gr, 0, 1, 0, 0);
insere_Aresta(gr, 1, 3, 0, 0);
remove_aresta(gr, 0, 1, 0);
....
....
....
libera_grafo(gr);
```

Definindo o TAD Grafo usando Lista de Adjacências

- Removendo uma Aresta no Grafo

```c
//No arquivo grafo.c
int remove_aresta(Grafo *gr, int orig, int dest, int eh_digrafo){
    if(gr == NULL)
        return 0;
     if(orig < 0 || orig >= gr->nro_vertices)
        return 0;
    if(dest < 0 || dest >= gr->nro_vertices)
        return 0;
    int i = 0;
    while(i < gr->grau[orig] && gr->arestas[orig][i] != dest){
        i++;
    }
    if(i == gr->grau[orig])
        return 0;
    gr->grau[orig]--;
    gr->arestas[orig][i] = gr->arestas[orig][gr->grau[orig]];
    if(gr->eh_ponderado){
        gr->pesos[orig][i] = gr->pesos[orig][gr->grau[orig]];
    }
    if(eh_digrafo == 0){
        remove_aresta(gr, dest, orig, 1);
    }
    return 1;
}
```

Verifica se existem e se são Vértices válidos

Procura a Aresta

Elemento não existe

Remove a outra Aresta se não for Digrafo

São passados invertidos

Estrutura de Dados 2

Atividade

- Após a montagem parcial do TAD Grafo, entregue-o no Moodle como Atividade Grafo