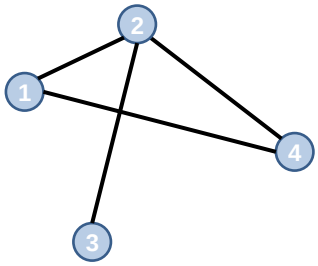
GRAFOS

Prof. André Backes

Definição

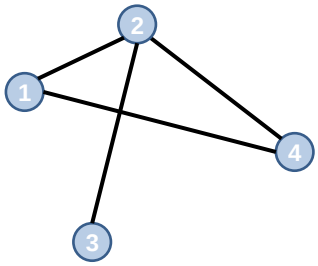
- Como representar um conjunto de objetos e as suas relações?
  - Diversos tipos de aplicações necessitam disso
  - Um grafo é um modelo matemático que representa as relações entre objetos de um determinado conjunto.



Definição

3

- Grafos em computação
  - Forma de solucionar problemas computáveis
  - Buscam o desenvolvimento de algoritmos mais eficientes
    - Qual a melhor rota da minha casa até o restaurante?
    - Duas pessoas tem amigos em comum?



Definição

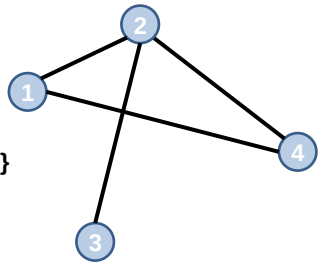
4

- Um grafo G(V,A) é definido por dois conjuntos
  - Conjunto V de vértices (não vazio)
    - Itens representados em um grafo;
  - Conjunto A de arestas
    - Utilizadas para conectar pares de vértices, usando um critério previamente estabelecido.

G(V,A)

V = {1,2,3,4}

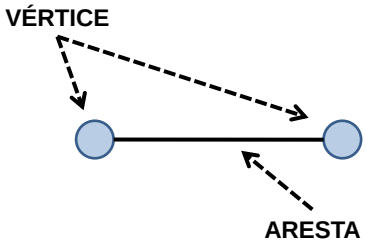
A = {{1,2},{1,4},{2,3},{2,4}}



Conceitos básicos

5

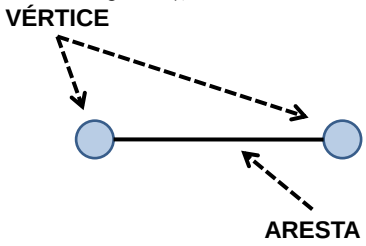
- Vértice é cada um dos itens representados no grafo.
  - O seu significado depende da natureza do problema modelado
    - Pessoas, uma tarefa em um projeto, lugares em um mapa, etc.



Conceitos básicos

6

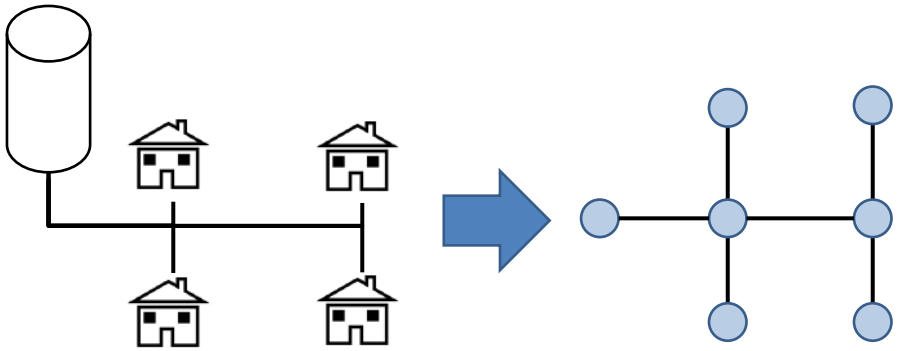
- Aresta (ou arco) liga dois vértices
  - Diz qual a relação entre eles
  - Dois vértices são **adjacentes** se existir uma aresta ligando eles.
    - Pessoas (parentesco entre elas ou amizade), tarefas de um projeto (pré-requisito entre as tarefas), lugares de um mapa (estradas que existem ligando os lugares), etc.



Conceitos básicos

7

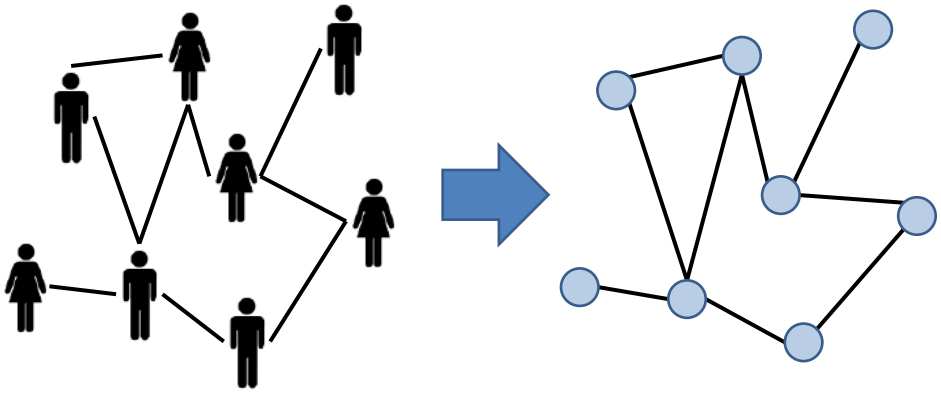
- Praticamente qualquer objeto pode ser representado como um grafo.
  - Exemplo: sistema de distribuição de água



Conceitos básicos

8

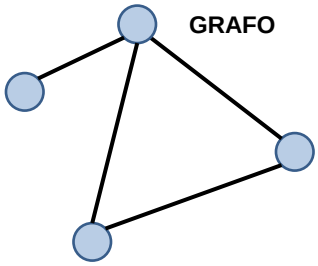
- Praticamente qualquer objeto pode ser representado como um grafo
  - Exemplo: rede social

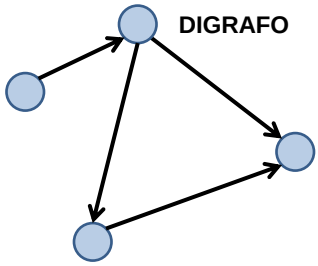


Conceitos básicos

9

- As arestas podem ou não ter direção
  - Existe um orientação quanto ao sentido da aresta
    - Em um grafo direcionado ou **digrafo**, se uma aresta liga os vértices **A** a **B**, isso significa que podemos ir de **A** para **B**, mas não o contrário





Conceitos básicos

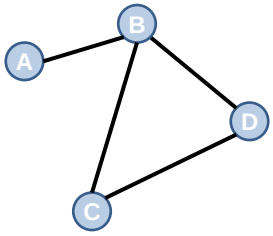
10

- Grau
  - Indica o número de arestas que conectam um vértice do grafo a outros vértices
    - número de vizinhos que aquele vértice possui no grafo (que chegam ou partem dele)
  - No caso dos dígrafos, temos dois tipos de grau:
    - **grau de entrada**: número de arestas que chegam ao vértice;
    - **grau de saída**: número de arestas que partem do vértice.

Conceitos básicos

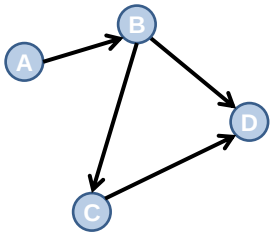
11

GRAFO



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

DIGRAFO

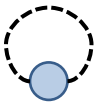


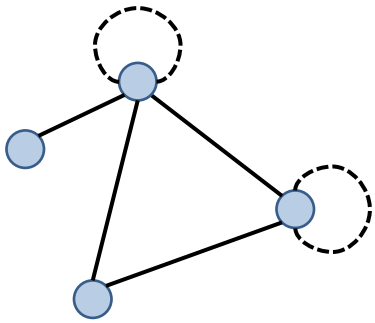
Grau Entrada	Grau Saída
G(A) = 0	G(A) = 1
G(B) = 1	G(B) = 2
G(C) = 1	G(C) = 1
G(D) = 2	G(D) = 0

Conceitos básicos

12

- Laço
  - Uma aresta é chamada de laço se seu vértice de partida é o mesmo que o de chagada
    - A aresta conecta o vértice a ele mesmo

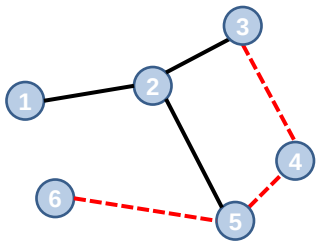




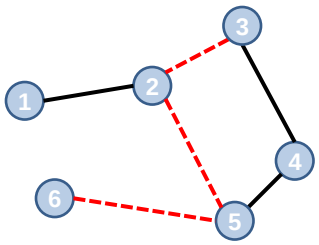
Conceitos básicos

13

- Caminho
  - Um caminho entre dois vértices é uma sequência de vértices onde cada vértice está conectado ao vértice seguinte por meio de uma aresta.



CAMINHO: 3-4-5-6

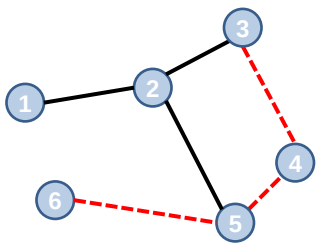


CAMINHO: 3-2-5-6

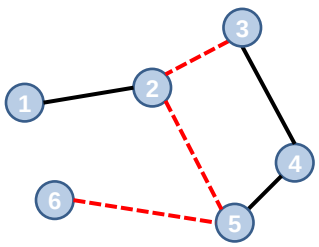
Conceitos básicos

14

- Caminho
  - Comprimento do caminho: número de vértices que precisamos percorrer de um vértice até o outro



CAMINHO: 3-4-5-6

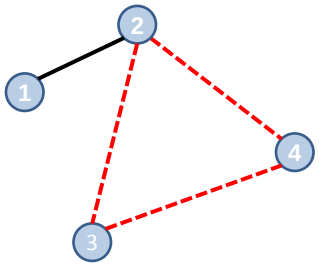


CAMINHO: 3-2-5-6

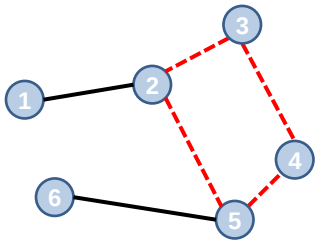
Conceitos básicos

15

- Ciclo
  - Caminho onde o vértice inicial e o final são o mesmo vértice.
    - Note que um ciclo é um caminho fechado sem vértices repetidos



CICLO: 2-3-4



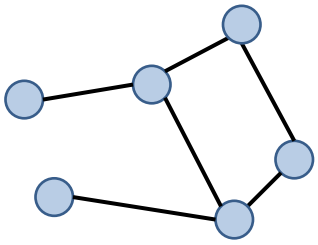
CICLO: 2-3-4-5

Tipos de Grafos

16

- Grafo trivial
  - Possui um único vértice e nenhuma aresta
- Grafo simples
  - Grafo não direcionado, sem laços e sem arestas paralelas (multigrafo)

TRIVIAL

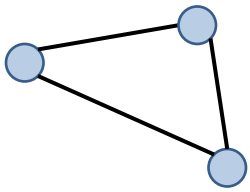


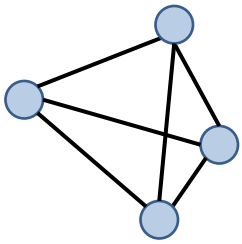
SIMPLES

Tipos de Grafos

17

- Grafo completo
  - Grafo simples onde cada vértice se conecta a todos os outros vértices do grafo.

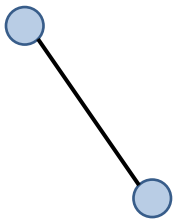




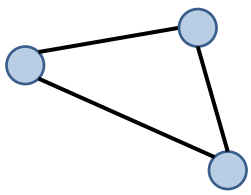
Tipos de Grafos

18

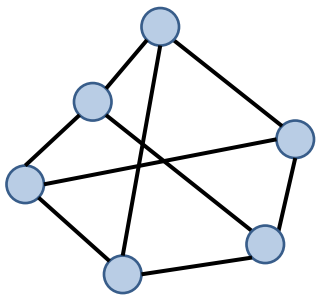
- Grafo regular
  - Grafo onde todos os seus vértices possuem o mesmo grau (número de arestas ligadas a ele)
  - Todo grafo completo é também regular



Grau = 1



Grau = 2

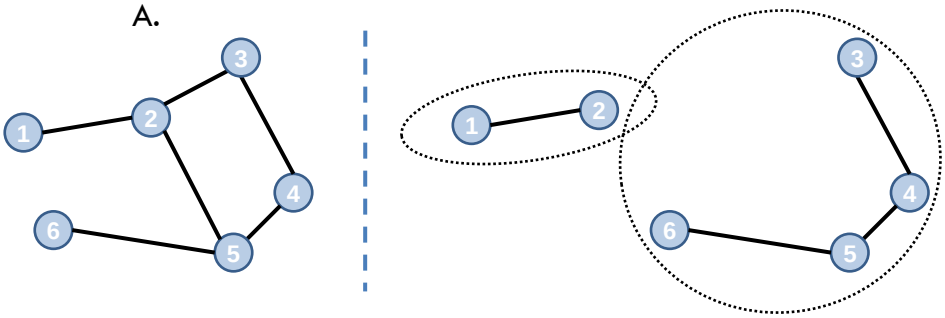


Grau = 3

Tipos de Grafos

19

- Subgrafo
  - Gs(Vs,As) é um subgrafo de G(V,A) se o conjunto de vértices Vs for um subconjunto de V, Vs ⊆ V, e se o conjunto de arestas As for um subconjunto de A, As ⊆ A.

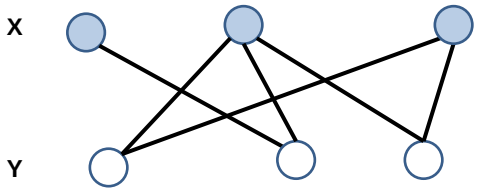


GRAFO SUBGRAFOS

Tipos de Grafos

20

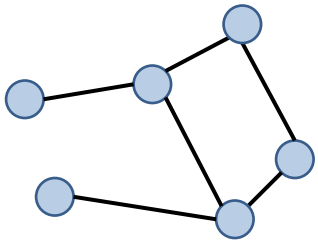
- Grafo bipartido
  - Um grafo G(V,A) onde o seu conjunto de vértices pode ser divididos em dois subconjuntos X e Y sem intersecção.
    - As arestas conectam apenas os vértices que estão em subconjuntos diferentes



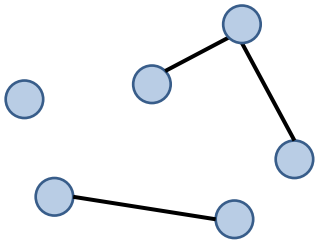
Tipos de Grafos

21

- Grafo conexo e desconexo
  - **Grafo conexo**: existe um caminho ligando quaisquer dois vértices.
  - Quando isso não acontece, temos um **grafo desconexo**



GRAFO CONEXO



GRAFO DESCONEXO

Tipos de Grafos

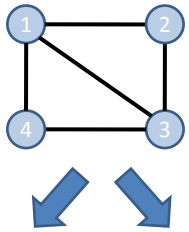
22

- 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.
    - Em outras palavras, dois grafos são isomorfos se existe uma função ***f*** onde, para cada dois vértices **a** e **b** adjacentes no grafo G1, **f(a)** e **f(b)** também são adjacentes no grafo G2.

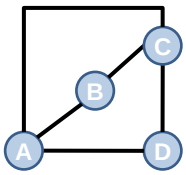
Tipos de Grafos

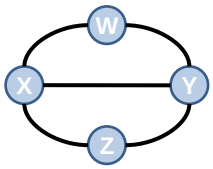
23

- Grafos isomorfos



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



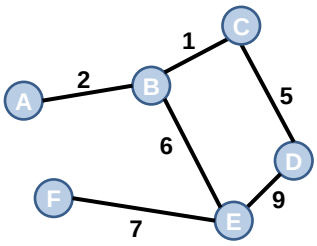


Grau
f(1) = X
f(2) = W
f(3) = Y
f(4) = Z

Tipos de Grafos

24

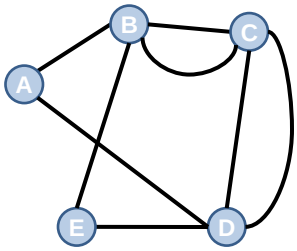
- Grafo ponderado
  - É um grafo que possui **pesos** (valor numérico) associados a cada uma de suas arestas.



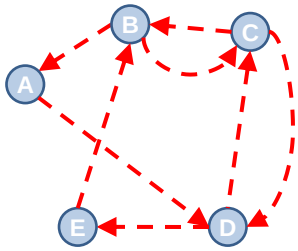
Tipos de Grafos

25

- Grafo Euleriano
  - Grafo que possui um **ciclo** que visita todas as suas arestas apenas uma vez, iniciando e terminando no mesmo vértice.



GRAFO EULERIANO

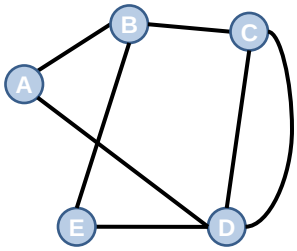


CICLO EULERIANO
C-D-C-B-A-D-E-B-C

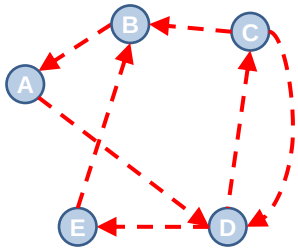
Tipos de Grafos

26

- Grafo Semi-Euleriano
  - Grafo que possui um **caminho** aberto (não é um ciclo) que visita todas as suas arestas apenas uma vez.



GRAFO
SEMI-EULERIANO

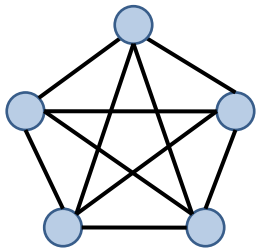


CAMINHO EULERIANO
C-D-C-B-A-D-E-B

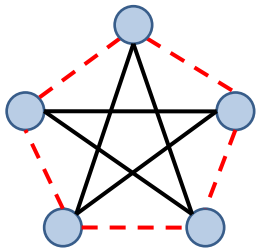
Tipos de Grafos

27

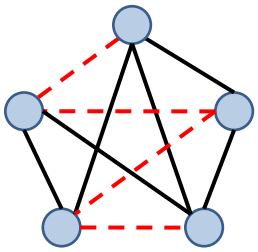
- Grafo Hamiltoniano
  - Grafo que possui um caminho que visita todos os seus vértices apenas uma vez.
  - Pode ser um ciclo



GRAFO HAMILTONIANO



CICLO HAMILTONIANO



CAMINHO HAMILTONIANO

Tipos de representação

28

- Como representar um grafo no computador?
  - Existem duas abordagens muito utilizadas:
    - Matriz de Adjacência
    - Lista de Adjacência
  - Qual a representação que deve ser utilizada?
    - Depende da aplicação!

Tipos de representação

29

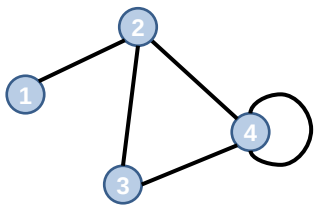
- Matriz de adjacência
  - Utiliza uma matriz **N x N** para armazenar o grafo, onde **N** é o número de vértices
    - Alto custo computacional, $O(N^2)$
  - Uma aresta é representada por uma marca na posição **(i , j)** da matriz
    - Aresta liga o vértice **i** ao **j**

Tipos de representação

30

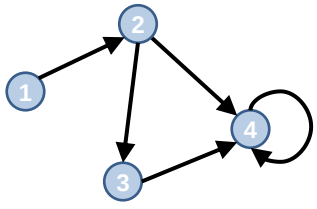
- Matriz de adjacência

GRAFO



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

DIGRAFO



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

Tipos de representação

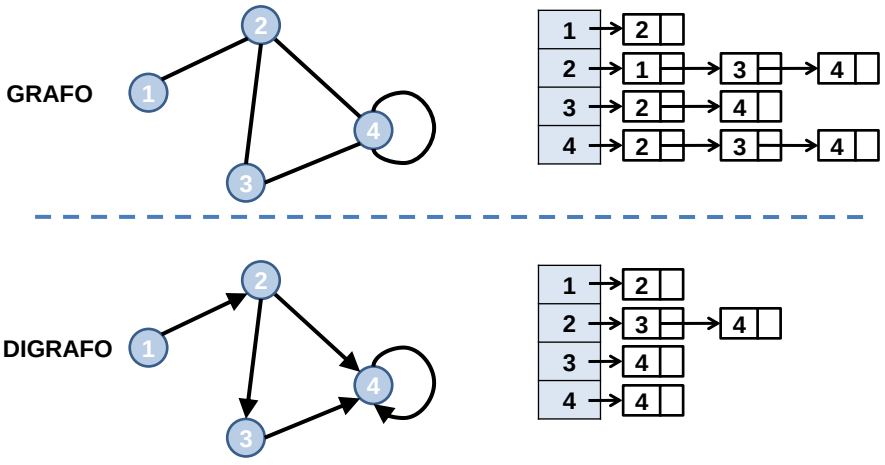
31

- Lista de adjacência
  - Utiliza uma lista de vértices para descrever as relações entre os vértices.
    - Um grafo contendo **N** vértices utiliza um array de ponteiros de tamanho **N** para armazenar os vértices do grafo
    - Para cada vértice é criada uma lista de arestas, onde cada posição da lista armazena o índice do vértice a qual aquele vértice se conecta

Tipos de representação

32

- Lista de adjacência



Tipos de representação

33

- Qual representação utilizar?
  - Lista de adjacência é mais indicada para um grafo que possui muitos vértices mas poucas arestas ligando esses vértices.
  - A medida que o número de arestas cresce e não havendo nenhuma outra informação associada a aresta (por exemplo, seu peso), o uso de uma matriz de adjacência se torna mais eficiente

TAD Grafo

34

- Vamos usar uma **lista de adjacência**
  - Lista de arestas: lista sequencial

```
3  //Arquivo Grafo.h
4  typedef struct grafo Grafo;
5
6  //Arquivo Grafo.c
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "Grafo.h" //inclui os Protótipos
10 //Definição do tipo Grafo
11 struct grafo{
12     int eh_ponderado;
13     int nro_vertices;
14     int grau_max;
15     int** arestas;
16     float** pesos;
17     int* grau;
18 };
19 //Programa principal
20 Grafo *gr;
```

Tamanho das listas

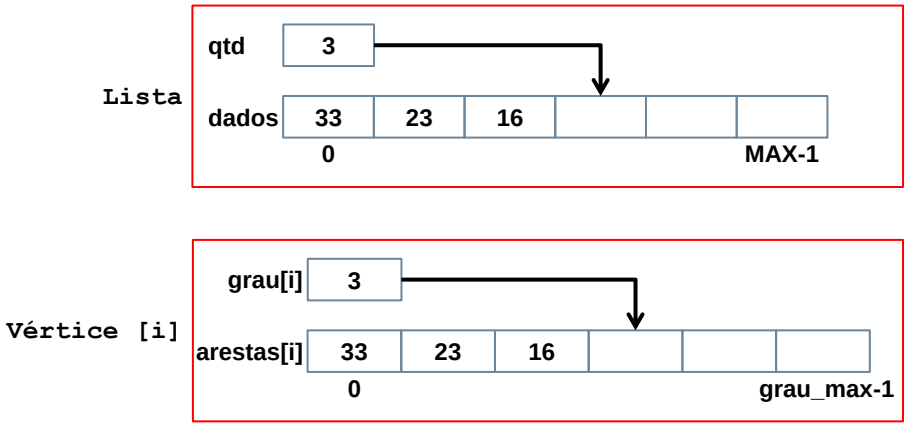
Array de listas

Qtd de elementos em cada lista

TAD Grafo

35

- Cada vértice funciona como uma lista estática



TAD Grafo

36

- Criando um grafo

```
5   "Criando um grafo"
6   //Arquivo Grafo.h
7   Grafo *cria_Grafo(int nro_vertices, int grau_max,
8                     int eh_ponderado);
9
10  //Programa principal
11  Grafo *gr;
12  gr = cria_Grafo(10, 7, 0);
13
```

TAD Grafo

37

- Criando um grafo

```
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;
}
```

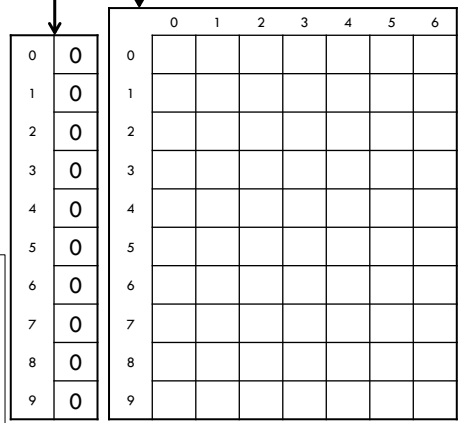
Cria matriz arestas

Cria matriz pesos

TAD Grafo

38

```
struct grafo{
    int eh_ponderado;
    int nro_vertices;
    int grau_max;
    int** arestas;
    float** pesos;
    int* grau;
};
Grafo *gr;
gr = cria_Grafo(10, 7, 0);
```



Cria um grafo de 10 vértices. Cada vértice se conecta com até outros 7 vértices
- **Matriz 10x7 para as arestas**
- **Vetor "grau" guarda o número de conexões de cada um dos 10 vértices**

TAD Grafo

39

- Liberando o grafo

```
5       "Liberando um grafo"
6       //Arquivo Grafo.h
7       void libera_Grafo(Grafo* gr);
8
9       //Programa principal
10      Grafo *gr;
11      gr = cria_Grafo(10, 7, 0);
12      .
13      .
14      .
15      libera_Grafo(gr);
16
```

TAD Grafo

40

- Liberando o grafo

```
1   void libera_Grafo(Grafo* gr){
2       if(gr != NULL){
3           int i;
4           for(i=0; i<gr->nro_vertices; i++)
5               free(gr->arestas[i]);
6           free(gr->arestas);
7
8           if(gr->eh_ponderado){
9               for(i=0; i<gr->nro_vertices; i++)
10                  free(gr->pesos[i]);
11              free(gr->pesos);
12          }
13          free(gr->grau);
14          free(gr);
15      }
16  }
```

Libera matriz arestas

Libera matriz pesos

TAD Grafo

41

- Inserindo uma aresta

```
5       "Inserindo uma aresta no grafo"
6       //Arquivo Grafo.h
7       int insereAresta(Grafo* gr, int orig, int dest,
8                        int eh_digrafo, float peso);
9       //Programa principal
10      Grafo *gr;
11      gr = cria_Grafo(10, 7, 0);
12      insereAresta(gr, 0, 1, 0, 0);
13      insereAresta(gr, 1, 3, 0, 0);
14      .
15      .
16      .
17      libera_Grafo(gr);
18
```

TAD Grafo

42

- Inserindo uma aresta

```
1   int insereAresta(Grafo* gr, int orig, int dest,
2                    int eh_digrafo, float peso){
3       if(gr == NULL)
4           return 0;
5       if(orig < 0 || orig >= gr->nro_vertices)
6           return 0;
7       if(dest < 0 || dest >= gr->nro_vertices)
8           return 0;
9
10      gr->arestas[orig][gr->grau[orig]] = dest;
11      if(gr->eh_ponderado)
12          gr->pesos[orig][gr->grau[orig]] = peso;
13      gr->grau[orig]++;
14
15      if(eh_digrafo == 0)
16          insereAresta(gr,dest,orig,1,peso);
17      return 1;
18  }
```

Verifica se vértice existe

Insere no final da linha

Insere outra aresta se NÃO for digrafo

TAD Grafo

43

```
insereAresta(gr,0,1,0,0);
```

Antes da inserção

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							

Após a inserção

0	1
1	1
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0						
2							
3							
4							
5							
6							
7							
8							
9							

TAD Grafo

44

```
insereAresta(gr,0,1,0,0);
insereAresta(gr,1,3,0,0);
```

Antes da inserção

0	1
1	1
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0						
2							
3							
4							
5							
6							
7							
8							
9							

Após a inserção

0	1
1	2
2	0
3	1
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0	3					
2							
3	1						
4							
5							
6							
7							
8							
9							

TAD Grafo

45

```
insereAresta(gr,0,1,0,0);
insereAresta(gr,1,3,0,0);
insereAresta(gr,3,2,0,0);
```

Antes da inserção

0	1
1	2
2	0
3	1
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0	3					
2							
3	1						
4							
5							
6							
7							
8							
9							

Após a inserção

0	1
1	2
2	1
3	2
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0	3					
2	3						
3	1	2					
4							
5							
6							
7							
8							
9							

TAD Grafo

46

```
insereAresta(gr,0,1,0,0);
insereAresta(gr,1,3,0,0);
insereAresta(gr,3,2,0,0);
insereAresta(gr,6,1,0,0);
```

Antes da inserção

0	1
1	2
2	1
3	2
4	0
5	0
6	0
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0	3					
2	3						
3	1	2					
4							
5							
6							
7							
8							
9							

Após a inserção

0	1
1	3
2	1
3	2
4	0
5	0
6	1
7	0
8	0
9	0

	0	1	2	3	4	5	6
0	1						
1	0	3	6				
2	3						
3	1	2					
4							
5							
6	1						
7							
8							
9							

Material Complementar

47

- Vídeo Aulas
  - Aula 56: Grafos – Definição:
  - youtu.be/gJvSmrxekDo
  - Aula 57: Grafos – Propriedades:
  - youtu.be/qvSbkbUkZjo
  - Aula 58: Grafos – Tipos de Grafos (Parte 1):
  - youtu.be/5saF2Dg6slc
  - Aula 59: Grafos – Tipos de Grafos (Parte 2):
  - youtu.be/LsLK04bWgy4
  - Aula 60: Grafos – Representação de Grafos (Parte 1):
  - youtu.be/k9DJn-COtKg
  - Aula 61: Grafos – Representação de Grafos (Parte 2):
  - youtu.be/-dAxrWDufa8