Pontifícia Universidade Católica
do Rio de Janeiro

Estruturas de Dados Avançadas- INF1010

Introdução Grafos
Busca em profundidade e busca em largura

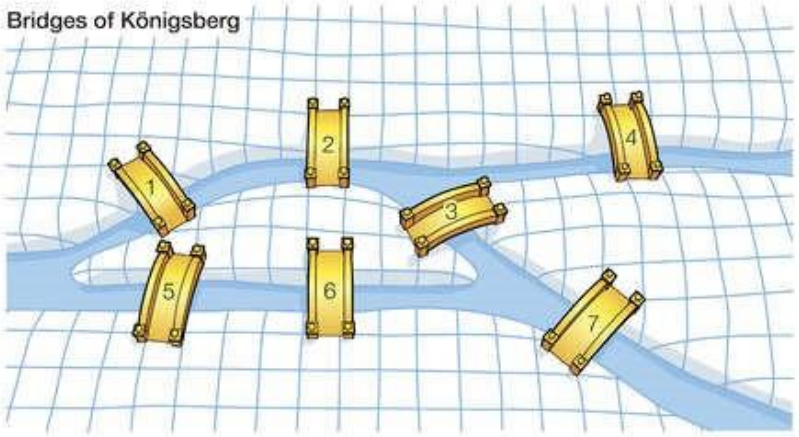
DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

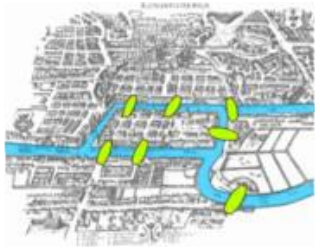
Introdução aos Grafos

- São estruturas matemáticas que podemos **implementar** através de estruturas de dados.
- Primeira aplicação conhecida: Sete pontes de Königsberg (1736)

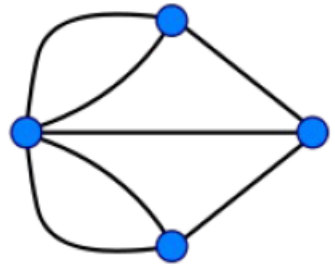
  – saindo de um ponto de Königsberg, é possível atravessar todas as pontes exatamente uma vez e retornar ao ponto inicial?

  – O problema foi resolvido por Leonhard Euler



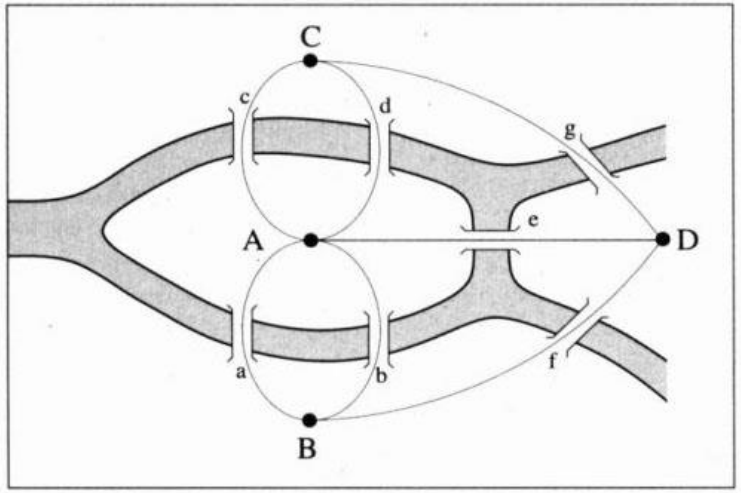


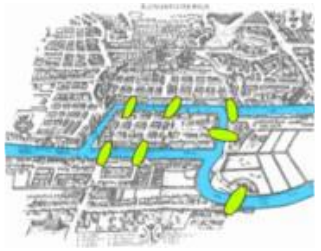
Introdução aos Grafos



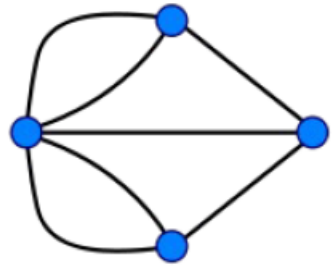
- Discutia-se nas ruas da cidade de Königsberg(atual Kaliningrado) a possibilidade de atravessar todas as sete pontes sem repetir nenhuma.

- Euler transformou os caminhos em linhas e suas intersecções em pontos
  - Foi a primeira representação de um grafo





Introdução aos Grafos

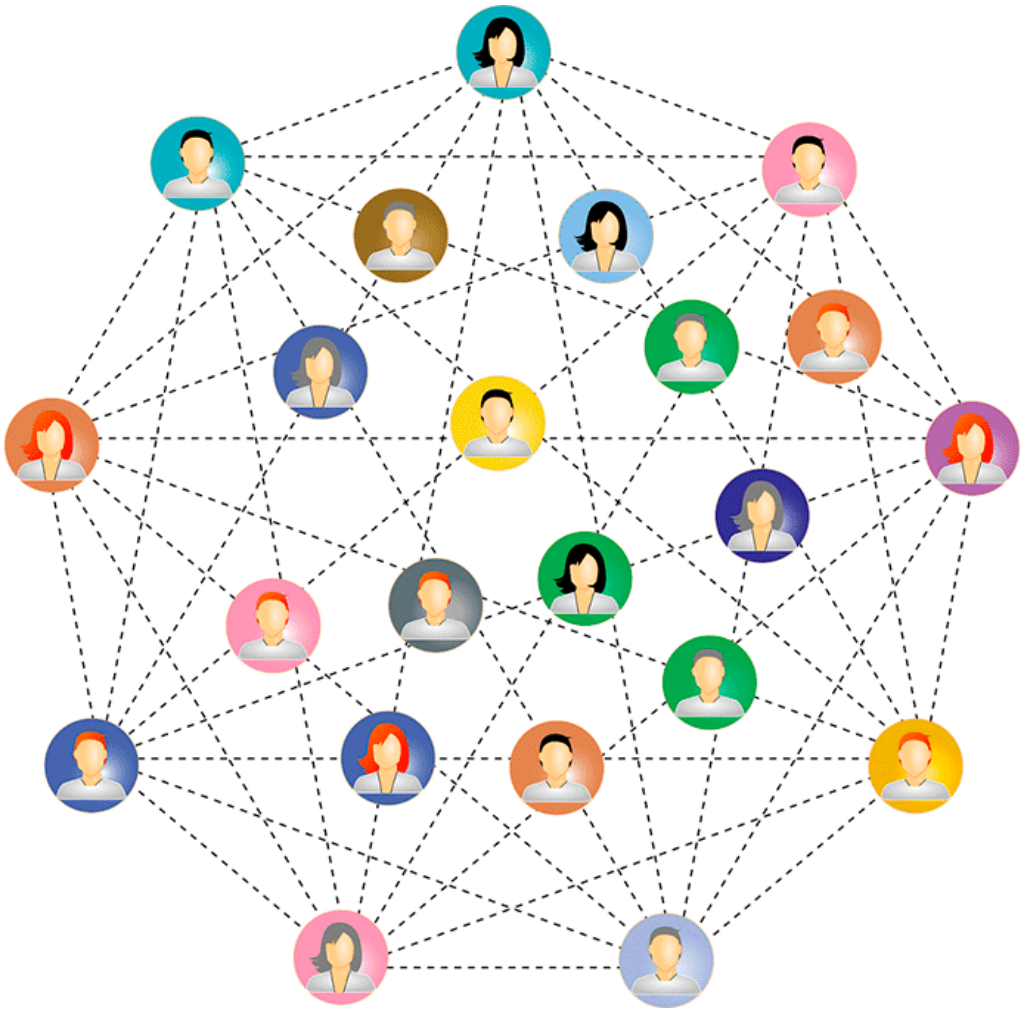


- Discutia-se nas ruas da cidade de Königsberg (atual Kaliningrado) a possibilidade de atravessar todas as sete pontes sem repetir nenhuma.

- Euler transformou os caminhos em linhas e suas intersecções em pontos
  - Foi a primeira representação de um grafo

- Ele provou que não existia caminho que possibilitasse tais restrições
  - Só seria possível se houvesse exatamente zero ou dois pontos de onde saísse um número ímpar de caminhos (inicio e fim do percurso).

Aplicações de Grafos



Aplicações de Grafos

Aplicações de Grafos

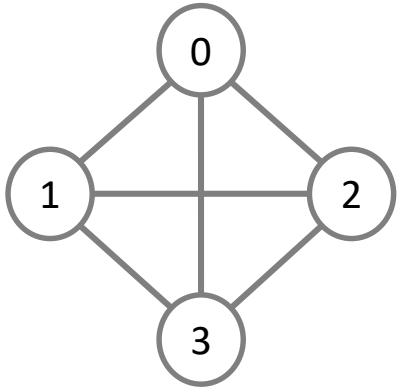
Grafo	Vértices	Arestas
Cronograma	tarefas	restrições de preferência
Malha viária	interseções de ruas	ruas
Rede de água (telefônica,…)	edificações (telefones,…)	canos (cabos,…)
Redes de computadores	computadores	conexões
Software	funções	chamadas de função
Web	páginas Web	links
Redes Sociais	pessoas	relacionamentos
...		

Problemas comuns

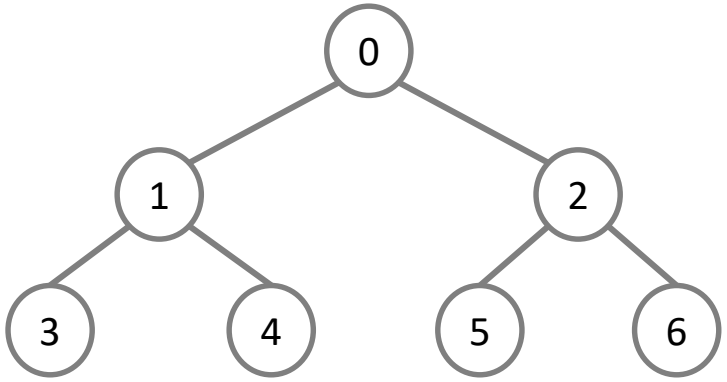
- Achar o caminho mínimo (mais curto, com menor custo, …)
- Determinar a ordem de execução de tarefas interdependentes
- Determinar o fluxo máximo de uma rede

Grafo Não Dirigido

- Par G = (V,E), onde
  - V é um conjunto de *n* **nós** ou **vértices**
  - E é um conjunto de *m* **arestas** (conexões entre dois vértices)



vértices: V = {0,1,2,3}
arestas: E = {{0,1},{0,2},{0,3},
{1,2},{1,3},{2,3}}



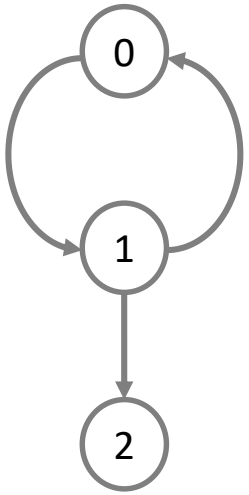
vértices : V = {0,1,2,3,4,5,6}
arestas: E = {{0,1},{0,2},{1,3},
{1,4},{2,5},{2,6}}

Grafo Dirigido (Orientado, Digrafo)

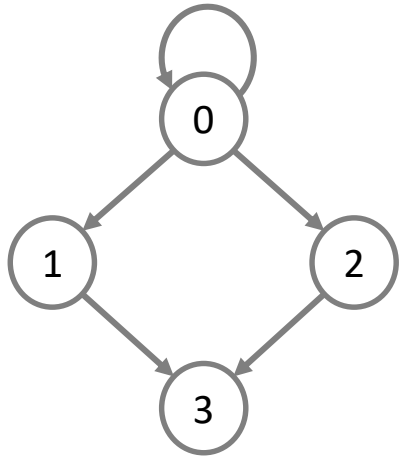
- Par G = (V,E), onde
  - V é um conjunto de *n* **nós** ou **vértices**
  - E é um conjunto de *m* **arcos** (conexões direcionadas entre dois vértices)

vértices: V = {0,1,2}
arcos: E = {(0,1), (1,0), (1,2)}

pares ordenados de vértices

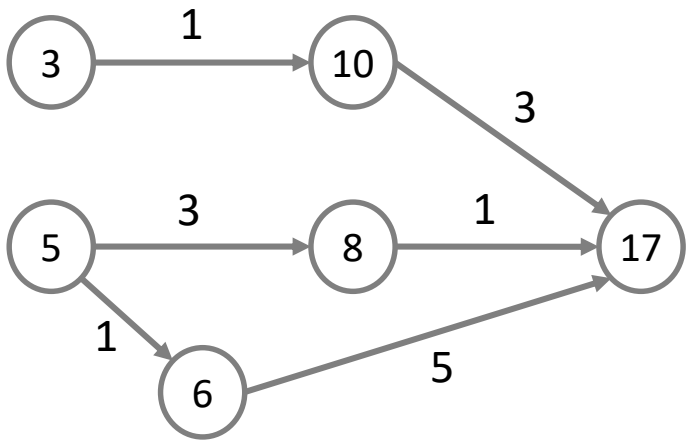


vértices: V = {0,1,2,3}
arestas: E = {{0,0},{0,1},{0,2},
{1,3},{2,3}}



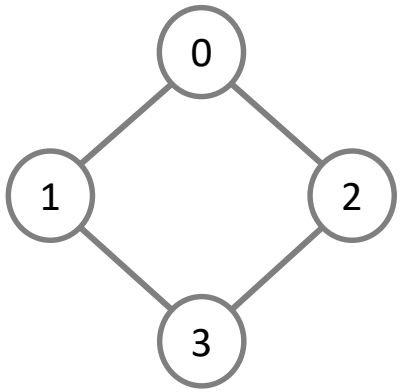
Grafo Ponderado

- Tripla G = (V,E,p), onde
  - V é um conjunto de *n* **nós** ou **vértices**
  - E é um conjunto de *m* **arcos**
  - p é uma função que atribui um **peso** a cada arco

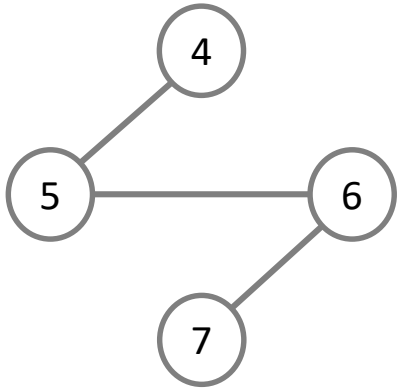


Vértices Adjacentes

- Vértices conectados por arestas

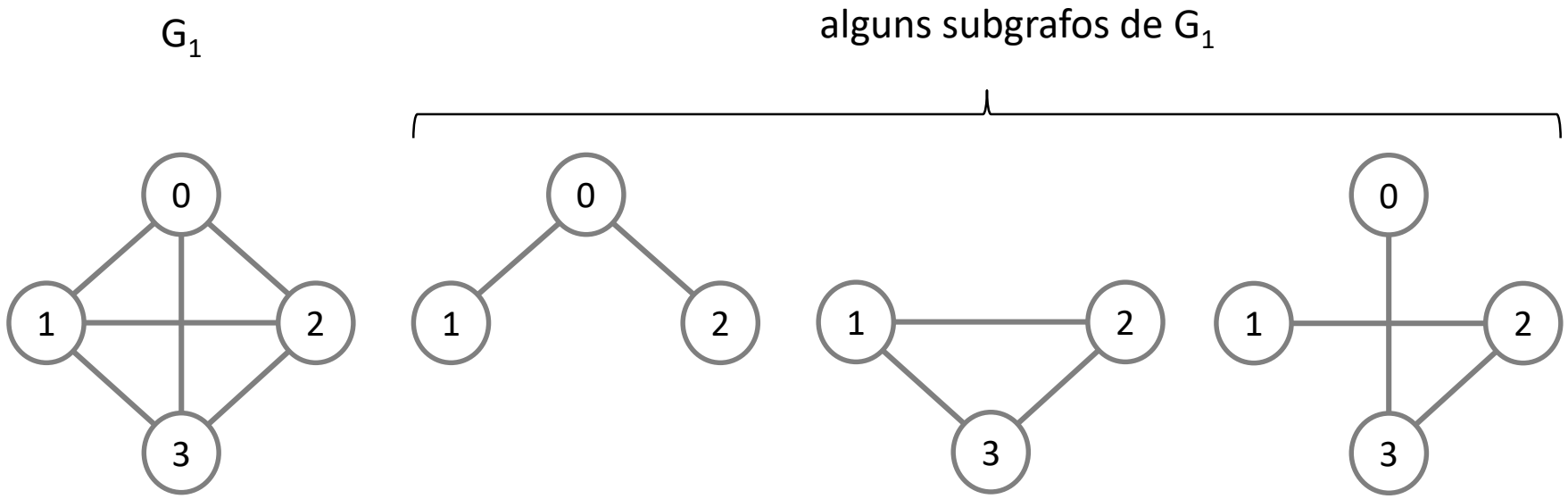


0 e 1
0 e 2
1 e 3
2 e 3

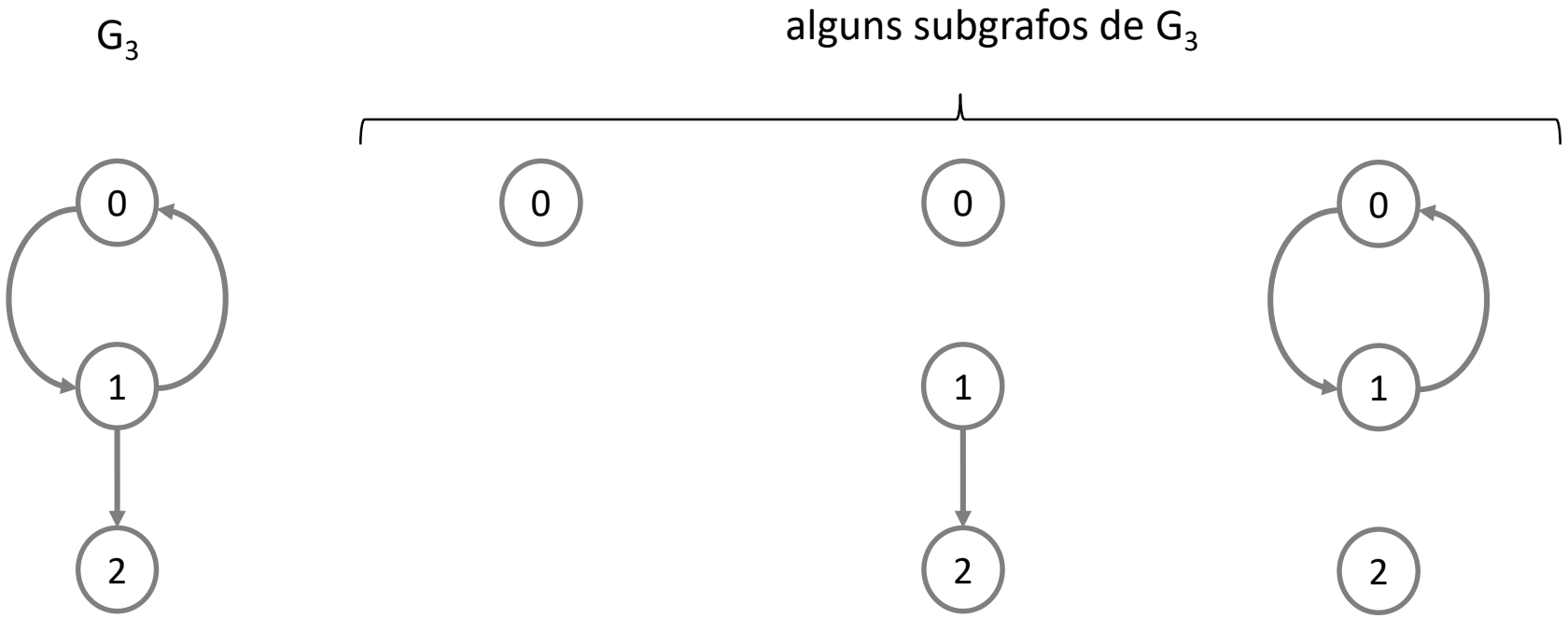


4 e 5
5 e 6
6 e 7

Subgrafo

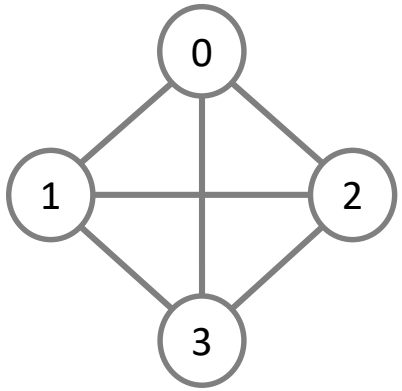


Subgrafo



Grafo Completo

- Um grafo não direcionado é **completo** sse cada vértice está conectado a cada um dos outros vértices

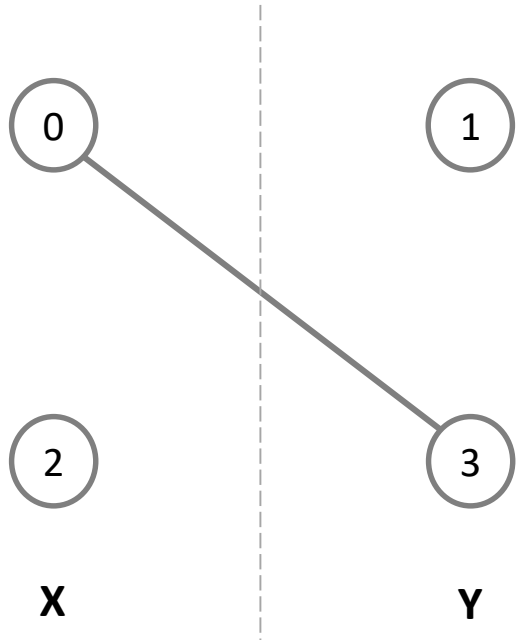


Exemplo de grafo completo: K_4

- Notação: K_n
- Um grafo completo de *n* vértices possui $n(n-1)/2$ arestas

Grafo Bipartido

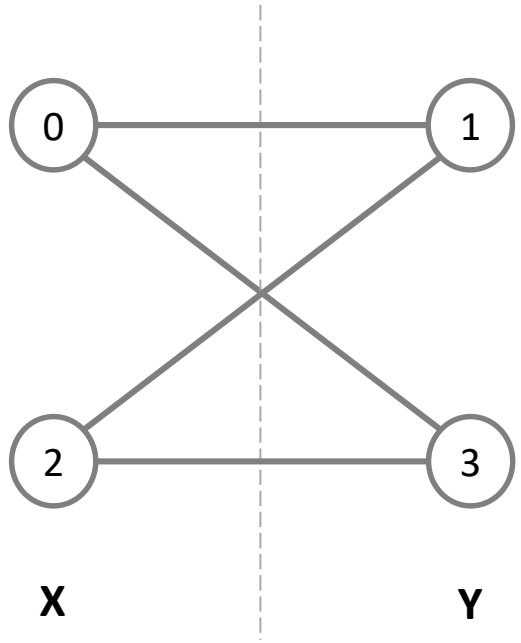
- Um grafo é chamado **bipartido** (bipartite) se o seu conjunto de vértices puder ser particionado em dois subconjuntos X e Y, tal que cada aresta tenha uma extremidade em X e a outra em Y.



Grafo Bipartido Completo

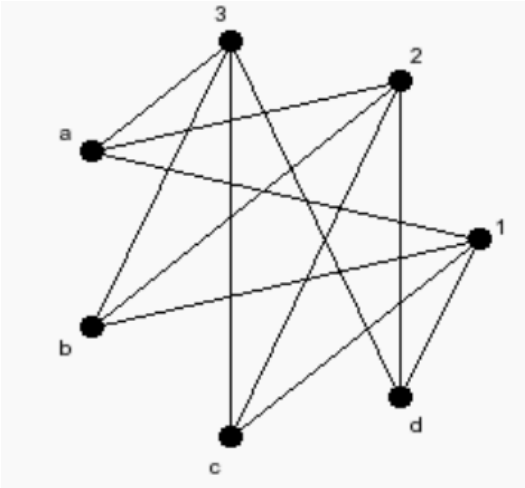
- Em um grafo **bipartido completo**, todos os vértices de X estão ligados a todos os vértices de Y (por meio de uma única aresta para cada par de vértices).

- Notação: $K_{n,m}$

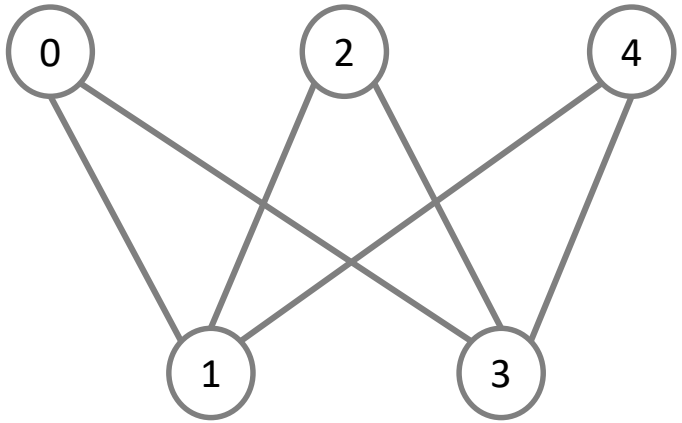


Exemplo de grafo bipartido completo: $K_{2,2}$

Exemplos Grafo Bipartido Completo



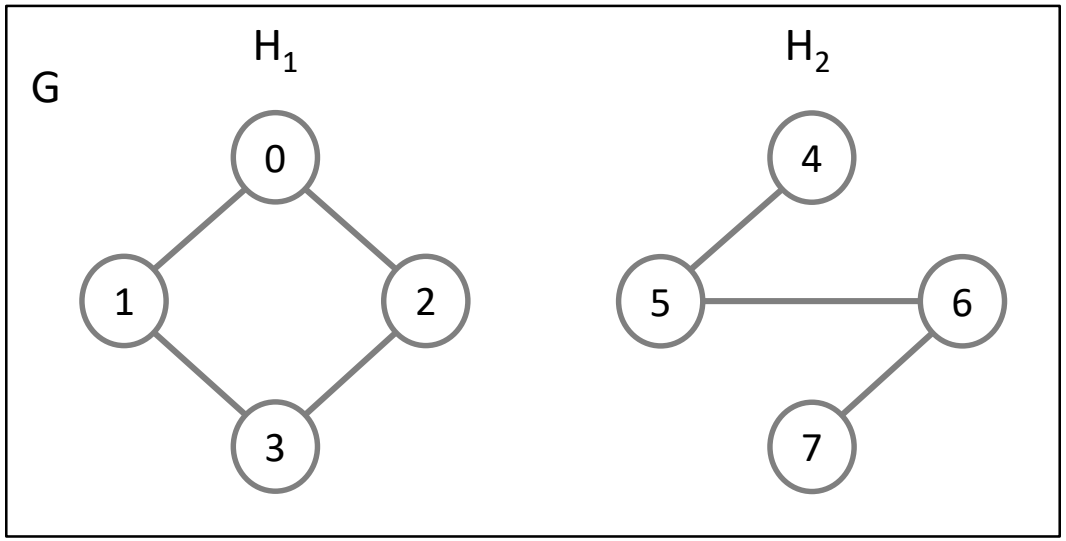
bipartido completo: $K_{3,4}$



bipartido completo: $K_{2,3}$

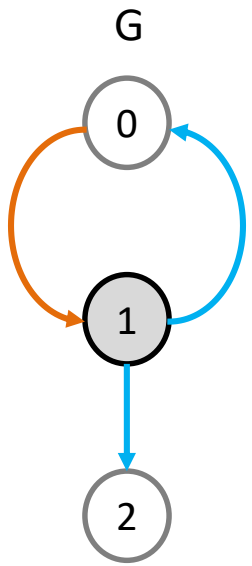
Grafo Conectado

- Um grafo não direcionado é **conectado** ou **conexo** sse existe um **caminho** entre quaisquer dois vértices
  - componentes **conexos** de um grafo

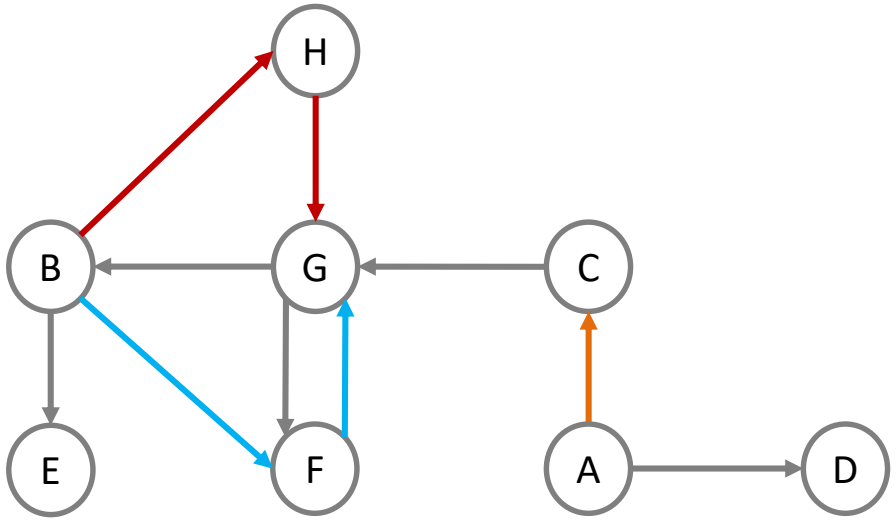


Grau

- Um vértice *v* possui **grau** $\delta(v) = n$ se há exatamente *n* arestas nele incidentes
  - grau do vértice 1: $\delta(1) = 3$
  - grau de entrada do vértice 1: $\delta^{-}(1) = 1$
  - grau de saída do vértice 1: $\delta^{+}(1) = 2$



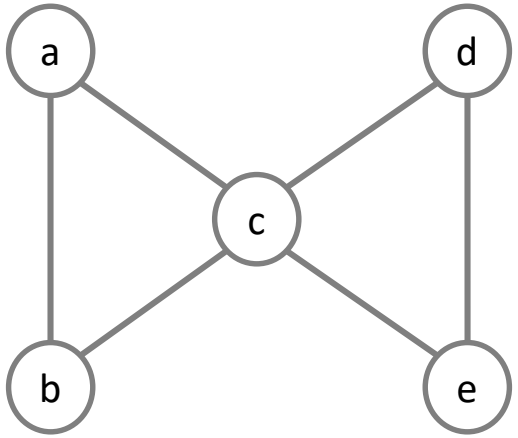
Caminhos



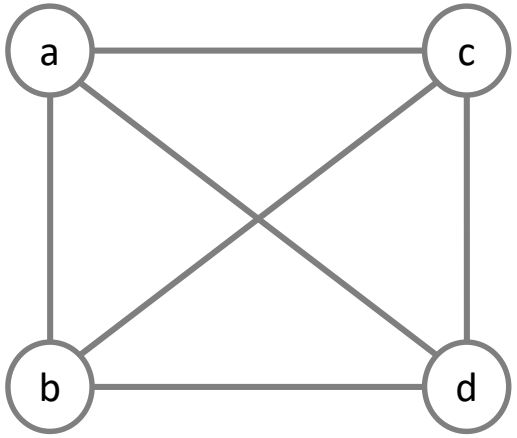
- caminho de comprimento 1 entre A e C
- caminho de comprimento 2 entre B e G, passando por H
- caminho de comprimento 2 entre B e G, passando por F
- caminho de comprimento 3 entre A e F

Grafo Euleriano

- Um **caminho euleriano** em um grafo G é um passeio fechado que contém todas as arestas de G, uma de cada vez.
  - Um grafo conexo G admite **caminho euleriano** sse todos os seus **vértices** forem de **grau par**.



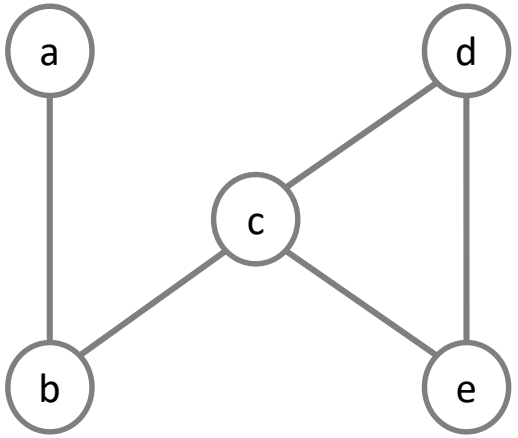
Grafo é chamado de euleriano
pois possui caminho euleriano, exemplo: abcdeca



Não é um grafo euleriano
Há vértices que têm grau ímpar

Caminho Euleriano

- Um **caminho euleriano** em um grafo G é um passeio fechado que contém todas as arestas de G, uma de cada vez.
  - Um grafo conexo G admite **caminho euleriano aberto** sse existirem **dois vértices** de **grau ímpar**.

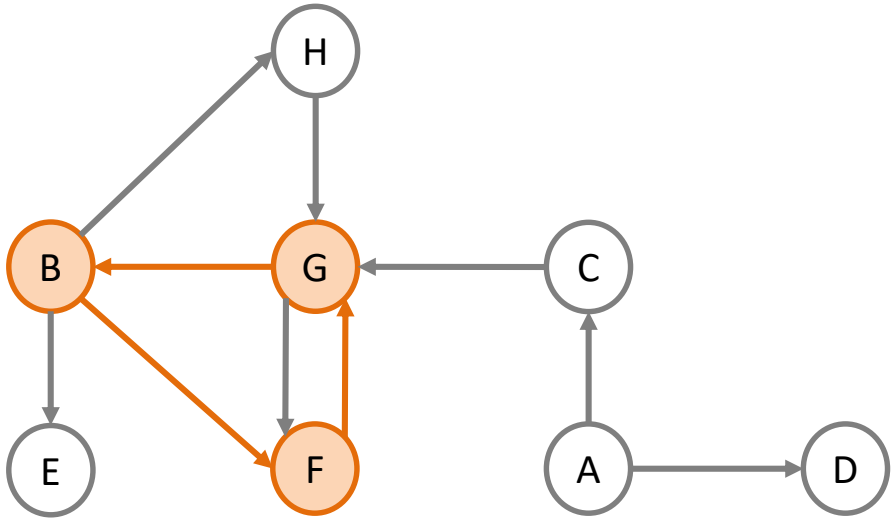


Há 2 vértices de grau ímpar ou seja há um caminho euleriano aberto.

NÃO é um grafo euleriano, pois contém vértices de grau ímpar
abcdec é um exemplo de caminho euleriano aberto

Ciclos

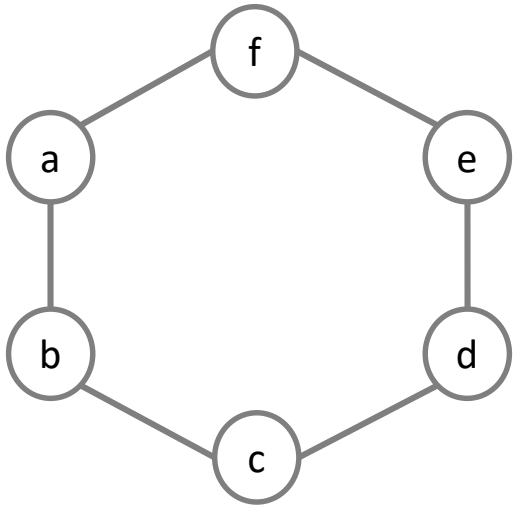
- Um ciclo é um caminho de um vértice a ele mesmo
  - grafo **cíclico** contem um ou mais ciclos
  - grafo **acíclico** não contem ciclos



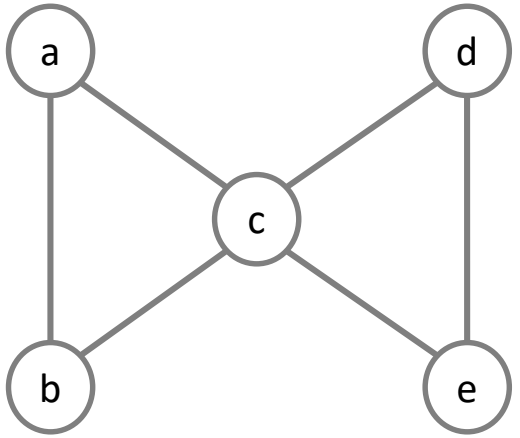
B – F – G – B

Grafo Hamiltoniano

- Um **ciclo hamiltoniano** em um grafo G é um ciclo que contém todos os vértices de G e cada vértice aparece apenas uma vez.



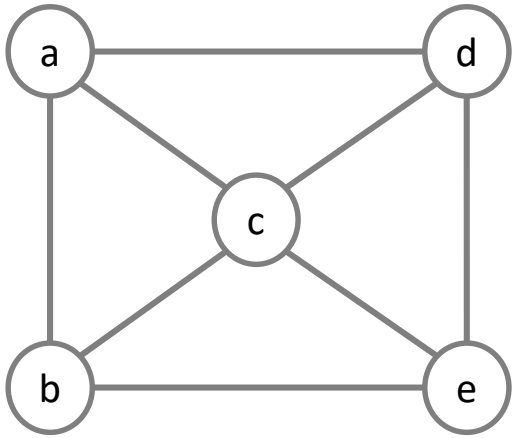
Grafo é chamado de hamiltoniano
pois possui ciclo hamiltoniano, exemplo: abcdefa



Não é um grafo hamiltoniano
Não possui ciclo hamiltoniano

Grafo Hamiltoniano

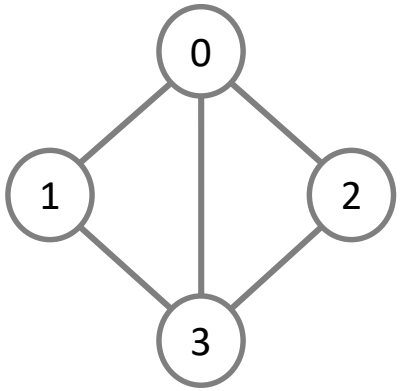
- Se G é um grafo simples e ($n_{vértices} \geq 3$) e ($\delta(G) \geq n_{vértices}/2$) então G é hamiltoniano.
  - Onde $\delta(G)$ é o grau mínimo do grafo G



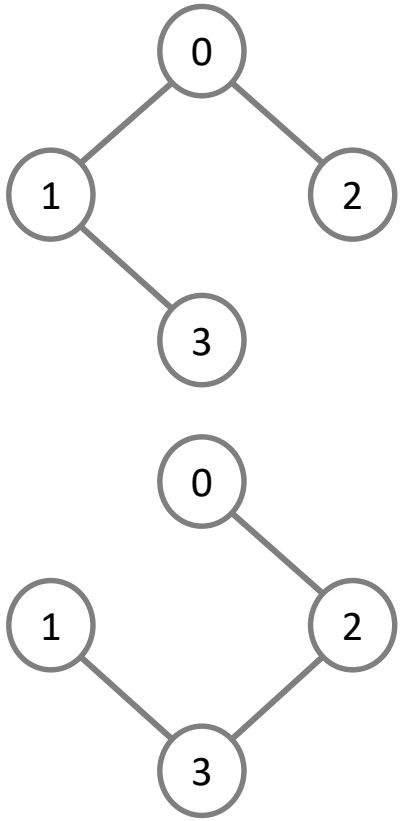
Grafo é hamiltoniano
pois $n_{vértices} = 5$, $\delta(G) = 3$ e $3 \geq 2{,}5$

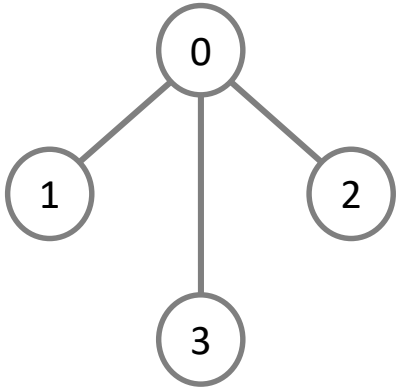
Árvore Geradora

- Subgrafo **acíclico** que contem todos os vértices, com caminhos entre quaisquer dois vértices



Grafo G





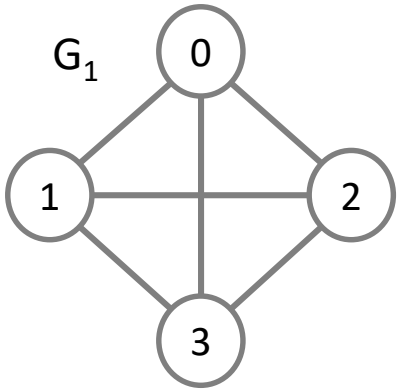
Representações de Grafos

- Matriz de adjacências ($V \times V$)
  - cada elemento m[i][j] representa uma aresta de v_i a v_j
  - espaço O(V^2): grafos densos ($E \sim V^2$)

- Lista de adjacências
  - vetor de vértices, cada vértice tem lista de arestas
  - espaço O($V + E$): grafos esparsos ($E \ll V^2$)

Matriz de Adjacências

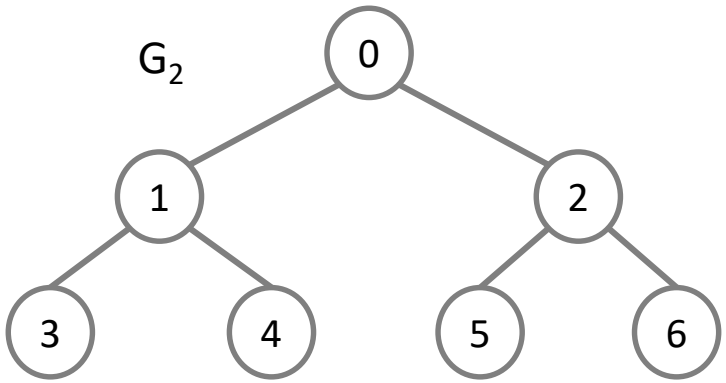
$$mat[i][j] = \begin{cases} 1, \text{ se houver uma aresta do nó i para o nó j} \\ 0, \text{ caso contrário} \end{cases}$$



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

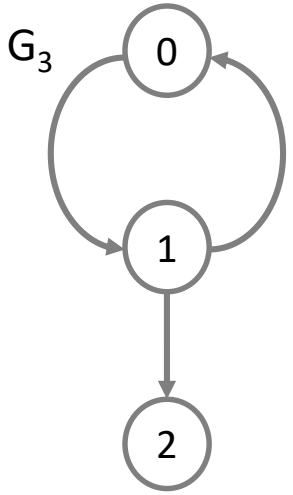
- matrizes simétricas para grafos não direcionados

Matriz de Adjacências



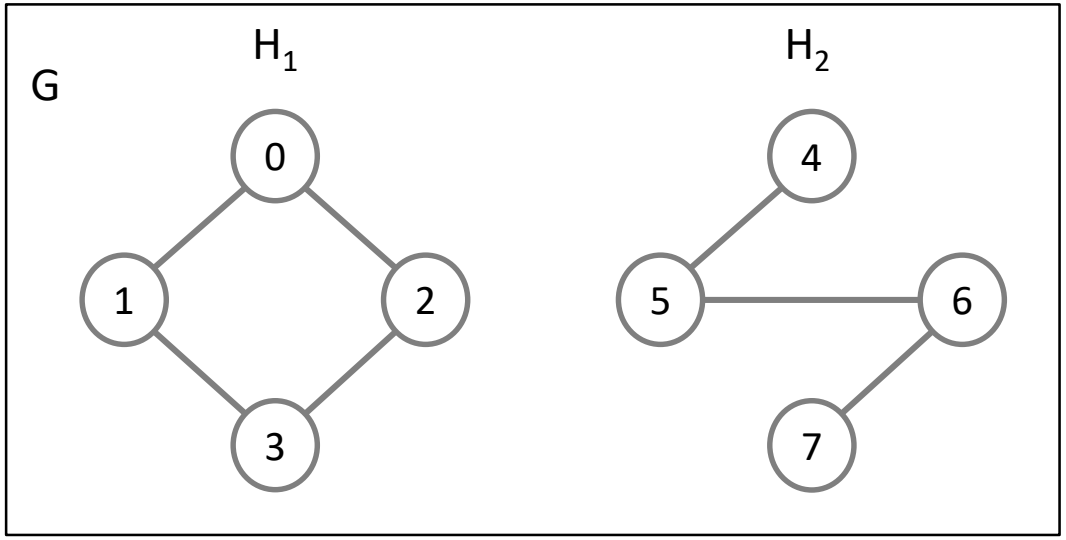
	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	0	1	1	0	0
2	1	0	0	0	0	1	1
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0
5	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0

Matriz de Adjacências



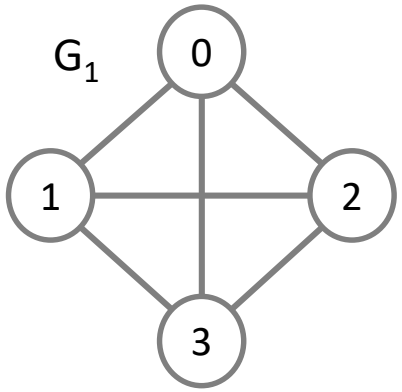
	0	1	2
0	0	1	0
1	1	0	1
2	0	0	0

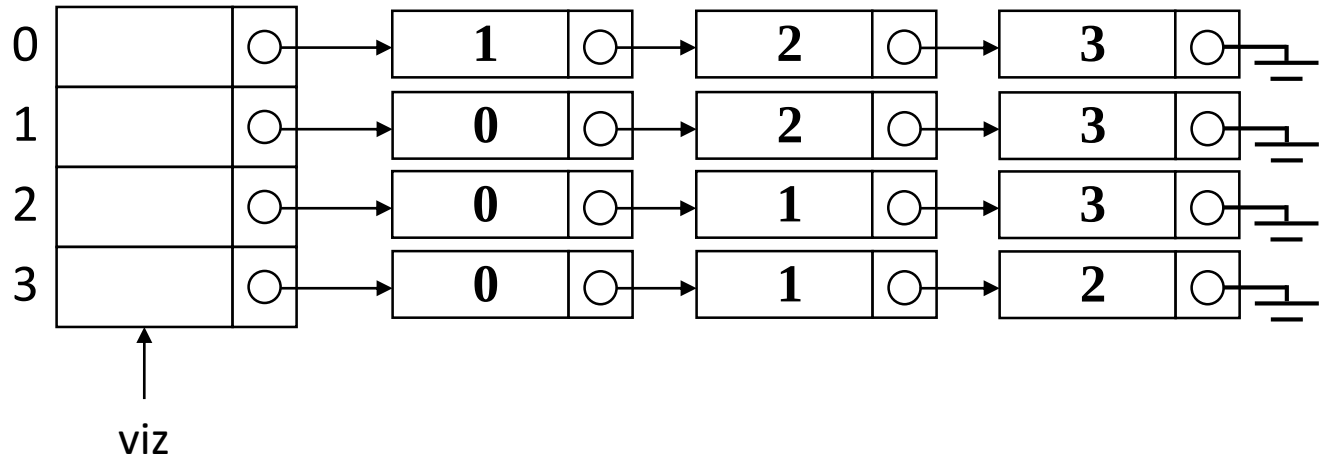
Matriz de Adjacências



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

Listas de Adjacências






```
struct _grafo {
   int nv;     /* numero de nos ou vertices */
   int na;     /* numero de arestas */
   Viz** viz; /* viz[i] aponta para a lista
                 de arestas incidindo em i */
};
```

```
typedef struct _viz Viz;
struct _viz {
   int noj;
   float peso;
   Viz* prox;
};
```

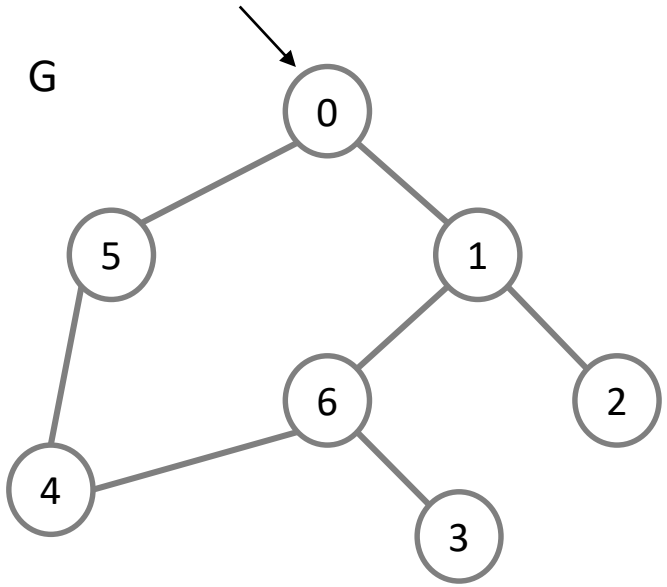
Criando um Grafo

```
static Viz* criaViz(Viz* head, int noj, float peso) {
/* insere vizinho no inicio da lista */
   Viz* no = (Viz*) malloc(sizeof(Viz));
   assert(no);
   no->noj = noj;
   no->peso = peso;
   no->prox = head;
   return no;
}

Grafo* grafoCria(int nv, int na) {
   int i;
   Grafo* g = (Grafo *) malloc(sizeof(Grafo));
   g->nv = nv;
   g->na = na;
   g->viz = (Viz **) malloc(sizeof(Viz *) * nv);
   for (i = 0; i < nv; i++)
      g->viz[i] = NULL;
   return g;
}
...
grafo->viz[no1] = criaViz(grafo->viz[no1], no2, peso);
grafo->viz[no2] = criaViz(grafo->viz[no2], no1, peso);
```

Percurso em Grafos

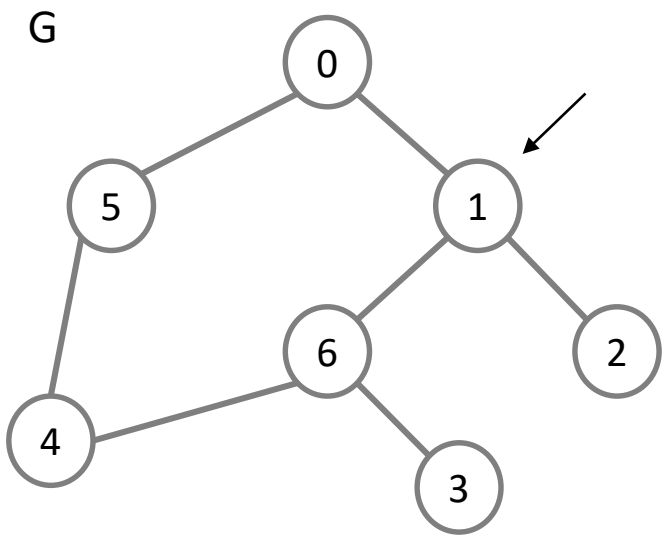
- Em profundidade (*depth-first search* – dfs)
  - arestas que partem do vértice visitado por último
- Em largura (*breadth-first search* – **bfs**)
  - arestas que partem do vértice visitado primeiro
- Guloso (*greedy*)
  - arestas de menor custo
  - tipicamente procurando caminho mínimo

dfs com recursão



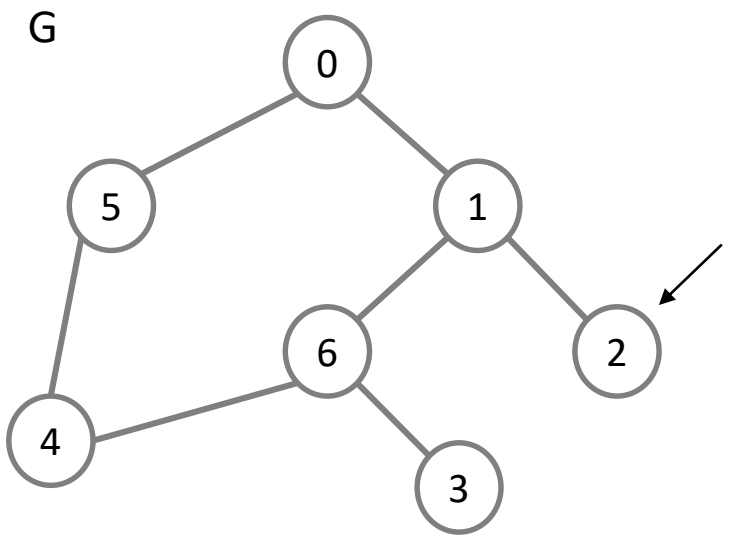
dfs(0)

dfs com recursão



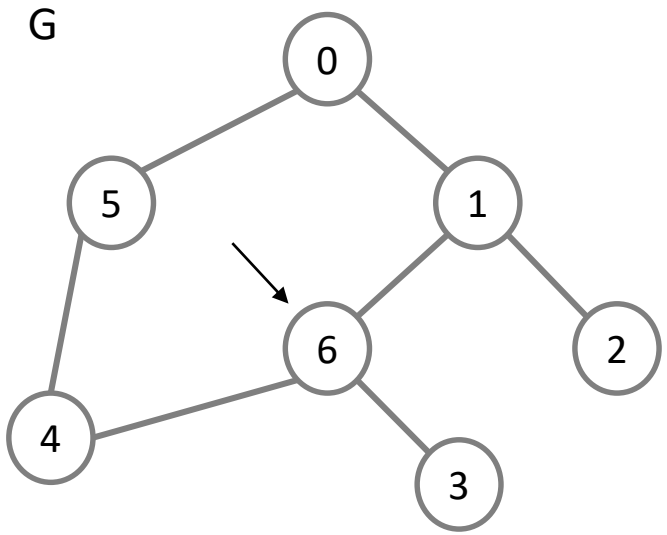
dfs(0)
  dfs(1)

dfs com recursão



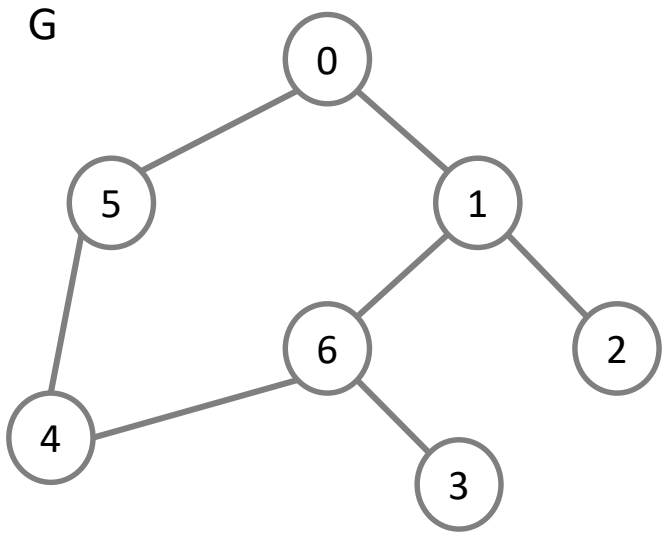

```
dfs(0)
  dfs(1)
    dfs(2)
```

dfs com recursão




```
dfs(0)
  dfs(1)
    dfs(2)
    dfs(6)
```

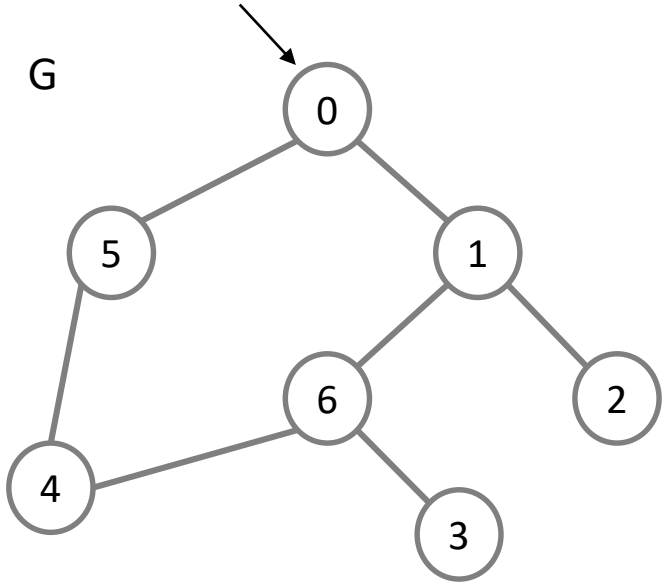
dfs com recursão



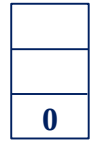

```
dfs(0)
  dfs(1)
    dfs(2)
    dfs(6)
      dfs(3)
      dfs(4)
        dfs(5)
```

Estado fica na pilha de chamadas recursivas!

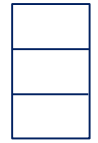
dfs com Pilha

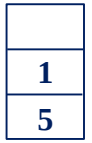


dfs(0)
empilha 0

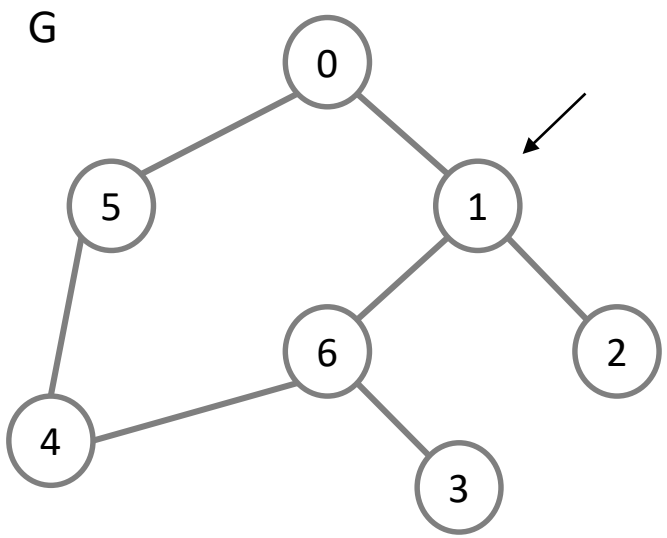


desempilha 0
visita 0
empilha 5, 1

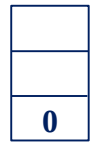




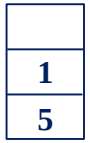
dfs com Pilha



dfs(0)
empilha 0

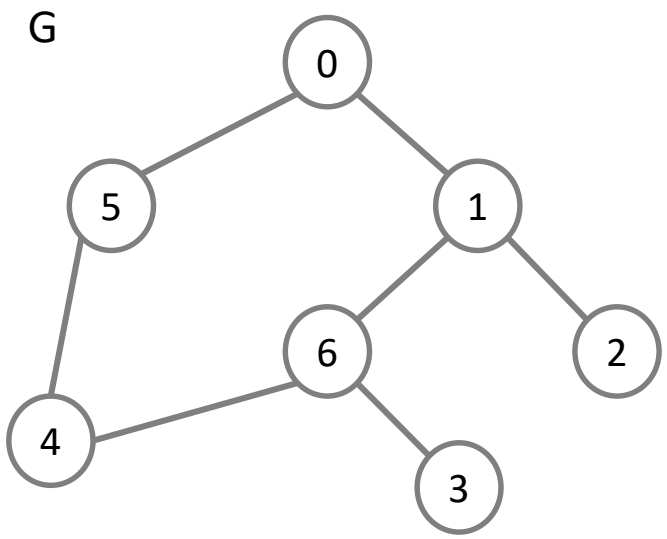


desempilha 0
visita 0
empilha 5, 1



desempilha 1
visita 1
empilha 6:2

dfs com Pilha



dfs(0)

empilha 0	[0]
desempilha 0	[ ]
visita 0	
empilha 5, 1	[1 5]
desempilha 1	[5]
visita 1	
empilha 6, 2	[2 6 5]
desempilha 2	[6 5]
visita 2	
desempilha 6	[5]
visita 6	
empilha 4, 3	[3 4 5]
desempilha 3	[4 5]
visita 3	
desempilha 4	[5]
visita 4	
desempilha 5	[ ]
visita 5	

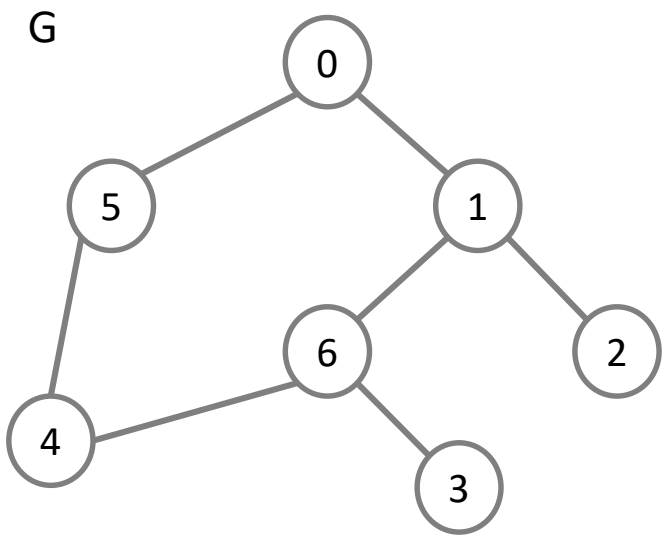
dfs com recursão

```
/* grafo como matriz de adjacências */
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

void dfs(int v) {
   int w;
   printf("%3d", v);
   visited[v] = 1;
   for (w = 0; w < MAX_VERTICES; w++)
      if (graph[v][w] && !visited[w]) dfs(w);
}
```

dfs com recursão

```
/* grafo como listas de adjacências */
typedef struct _listNode ListNode;
struct _listNode {
   int vertex;
   ListNode* link;
};

ListNode* graph[MAX_VERTICES];
int visited[MAX_VERTICES];

void dfs(int v){
   ListNode* w;
   visited[v] = 1;
   printf("%3d", v);
   for (w = graph[v]; w != NULL; w = w->link)
      if (!visited[w->vertex]) dfs(w->vertex);
}
```

Percurso bfs



bfs(0)
  visita 0
    visita 1 ⟶ arestas de 0
    visita 5

    visita 2 ⟶ arestas de 1
    visita 6

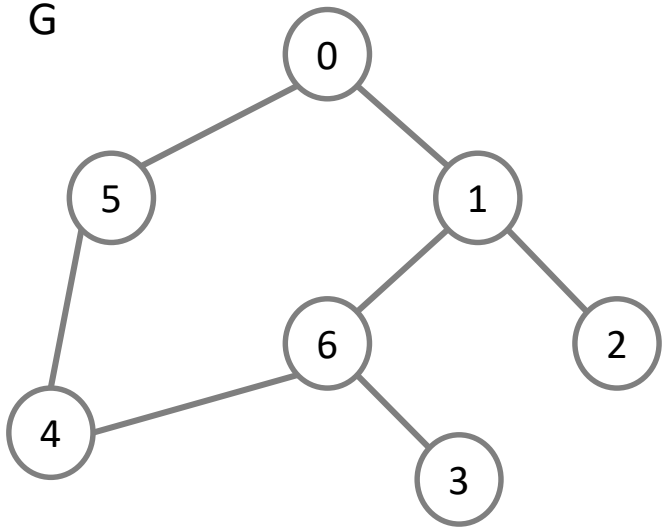
    visita 4 ⟶ arestas de 5

    visita 3 ⟶ arestas de 6

Como registrar os nós pendentes?

Percurso bfs

G



bfs(0)
  visita 0
    visita 1 ⟶ arestas de 0
    visita 5

    visita 2 ⟶ arestas de 1
    visita 6

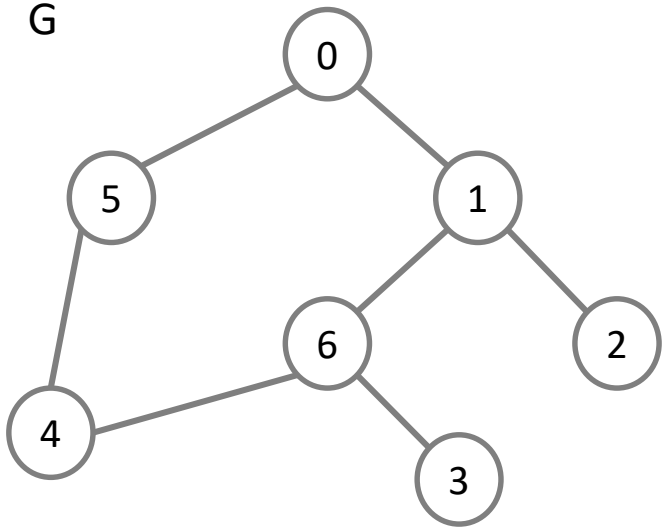
    visita 4 ⟶ arestas de 5

    visita 3 ⟶ arestas de 6

Como registrar os nós pendentes?
  Fila de nós

bfs com Fila Auxiliar

G



bfs(0)
visita 0
  -> enfileira 1, 5 [1,5]

visita 1
  -> enfileira 2, 6 [5,2,6]

visita 5
  -> enfileira 4 [2,6,4]

visita 2
[6,4]

visita 6
  -> enfileira 3 [4,3]

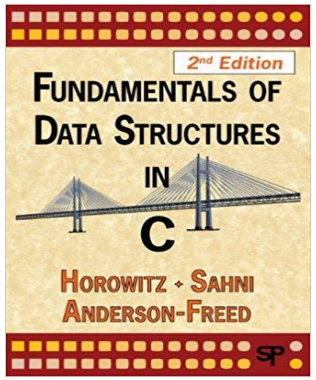
visita 4
[3]

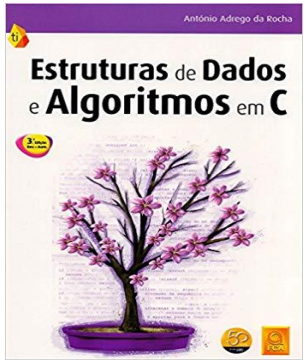
visita 3
[]

TAD Grafo

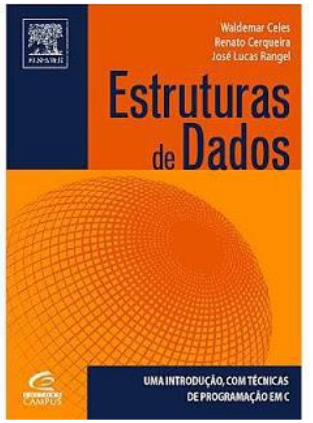
```
typedef struct graph Graph;

Graph* graph_create(int initial_size);

Graph* graph_destroy(Graph* g);

int graph_insert_vertex(Graph* g, void* info);

void graph_insert_edge(Graph* g, int v1, int v2, int weight);

void* graph_get_vertex_info(Graph* g, int idx);

void depth_first(Graph* g, int idx, int max_hops, void(*cb_fn)(void*));

unsigned int graph_shortest_distance(Graph* g, int v1, int v2);

void graph_print(Graph* g, void(*cb_fn)(void*));

int graph_num_components(Graph* g);
```

Leitura Complementar

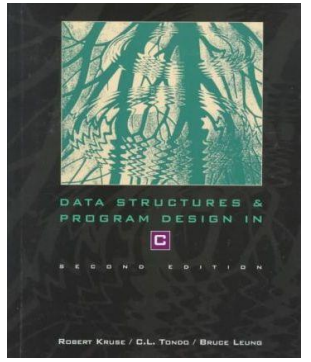




- Celes, W., Cerqueira, R., Rangel, J.L., **Introdução a Estruturas de Dados – Uma introdução com técnicas de programação em C**,
  Ed. Campus, 2004
  - **Capítulo 22 – Grafos**

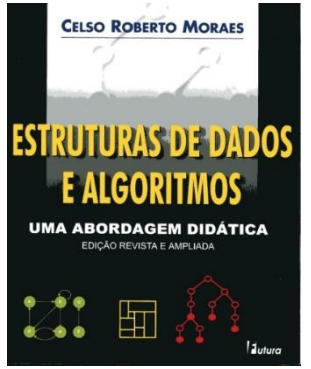


- Horowitz. E.; Sahni, S.; Anderson-Freed, S. **Fundamentals of Data Structures in C**, 2nd edition. Silicon Press, 2008.
  - Capítulo 6: Graphs



- Kruse, R.; Tondo, C.; Leung, B.; Mogalla, S.; **Data Structures and Program Design in C**, 2nd edition. Pearson, 1996.
  - Capítulo 11: Graphs

- ROCHA, A.; **Estruturas De Dados E Algoritmos Em C**, 3ª Edição, Ed. FCA, 2014
  - Capítulo 10: Grafos



- MORAES, C.; **Estruturas De Dados E Algoritmos: Uma Abordagem Didática**, Ed. Futura, 2003
  - Capítulo 11: Grafos