INF 1010
Estruturas de Dados Avançadas

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	linhas
Software	funções	chamadas de função
Web	páginas Web	links
Redes Sociais	pessoas	relacionamentos
…		

Grafos

... não são estruturas de dados, e sim estruturas matemáticas que implementamos com estruturas de dados…

Grafo não dirigido

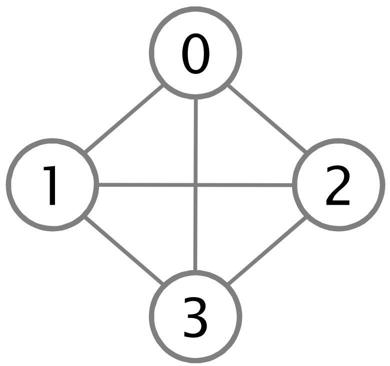
Um *grafo não dirigido* é um par G = (V,E), onde

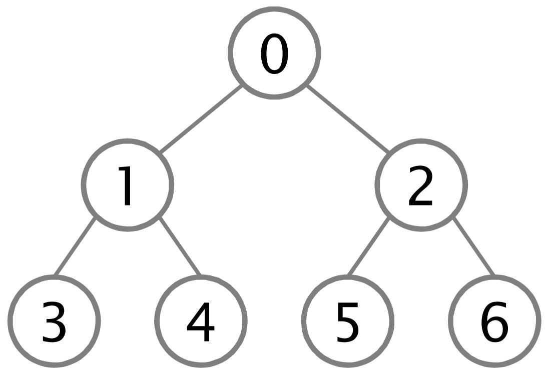
V é um conjunto de *nós* ou *vértices* e

E é um conjunto de *arestas*

uma *aresta* é um conjunto de 2 vértices

Exemplos





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}}

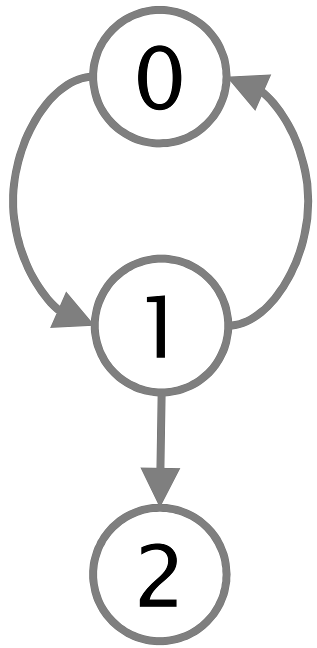
Grafo dirigido (orientado, ou Digrafo)

Um *grafo dirigido* é um par G = (V,E), onde
V é um conjunto de *n nós* ou *vértices* e
E é um conjunto de *m arcos*

- um *arco* é um par ordenado de vértices

Exemplo

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

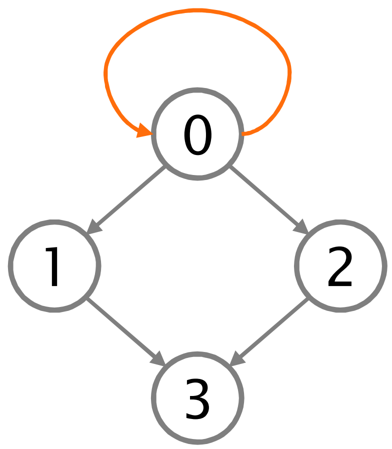


Grafo dirigido (orientado, ou digrafo)

Exemplo - Digrafo com auto-arco

vértices: V = {0,1,2,3}

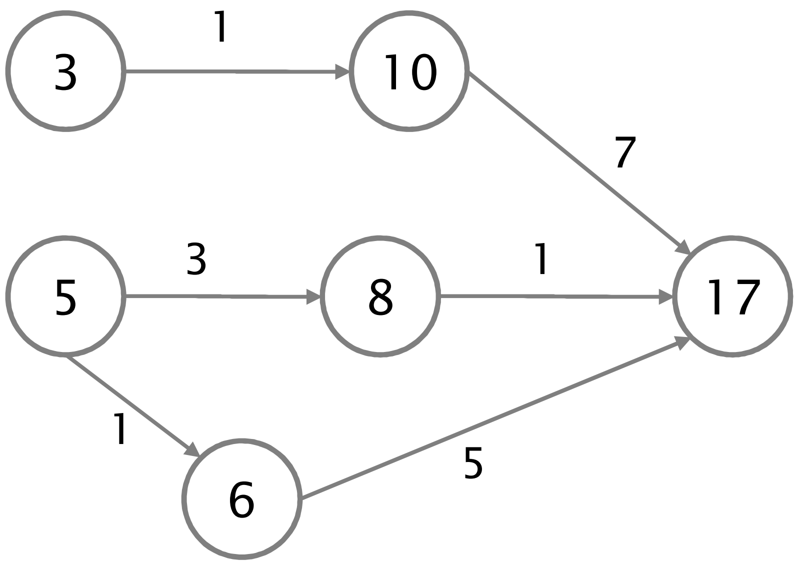
arcos: E = {(0,0), (0,1), (0,2), (1,3), (2,3)}



Grafo ponderado

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

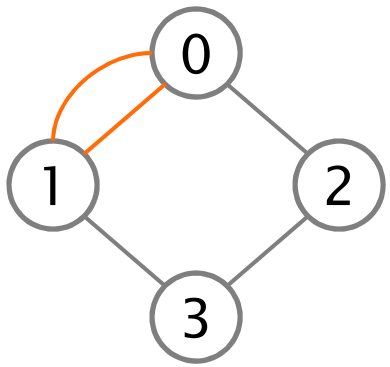
Exemplo



Multigrafo

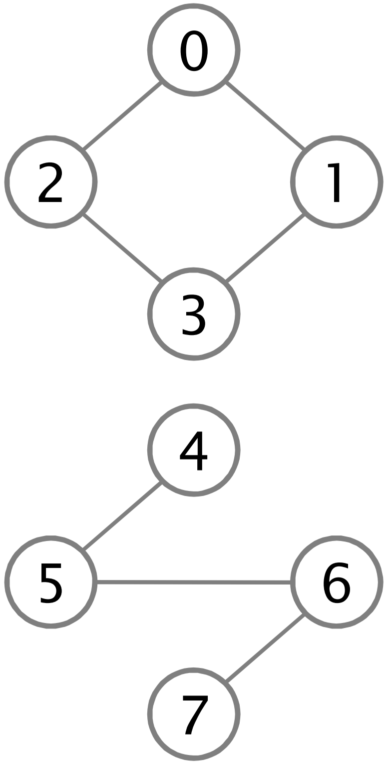
Um *multigrafo* é um grafo onde
dois nós podem estar conectados por mais de uma aresta

Exemplo



Vértices adjacentes

Vértices conectados por arestas

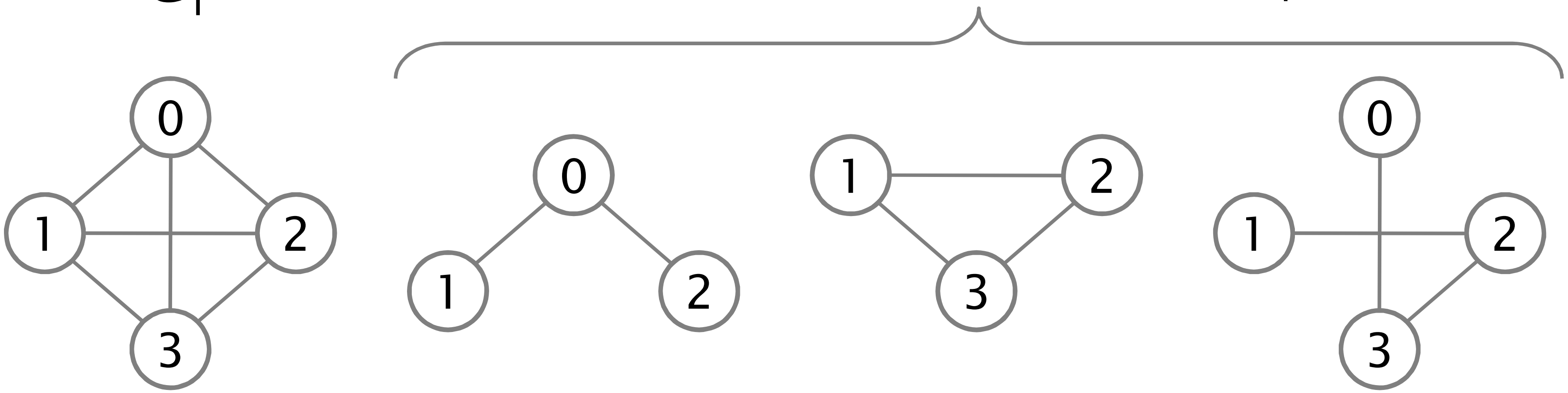


0 e 1
0 e 2
1 e 3
2 e 3

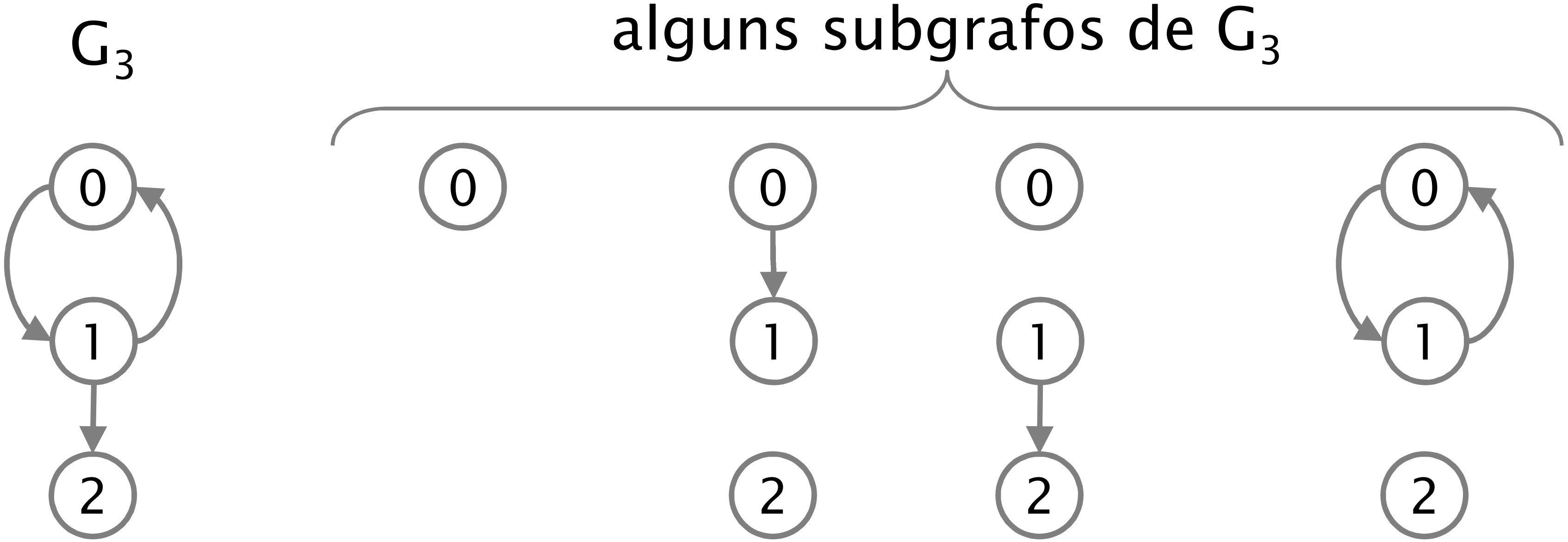
4 e 5
5 e 6
6 e 7

Subgrafo

G_1 alguns subgrafos de G_1

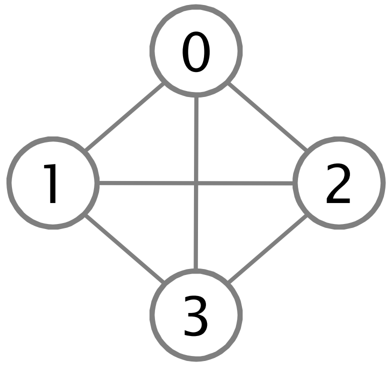


Subgrafo



Grafo completo

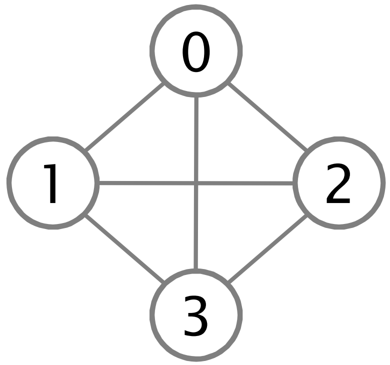
Um grafo não direcionado é *completo* sse cada vértice está conectado a cada um dos outros vértices por uma aresta



Quantas arestas há em um grafo completo de n vértices?

Grafo completo

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



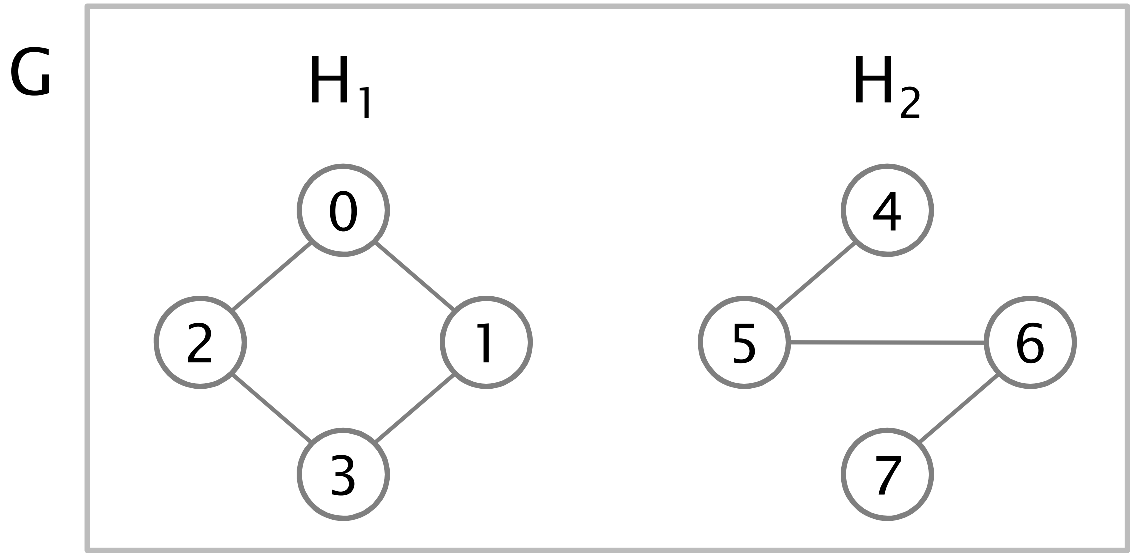
Quantas arestas há em um grafo completo de n vértices?

$n\,(n-1)/2$

Grafo conectado

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

Componente conexa de um grafo



Grau

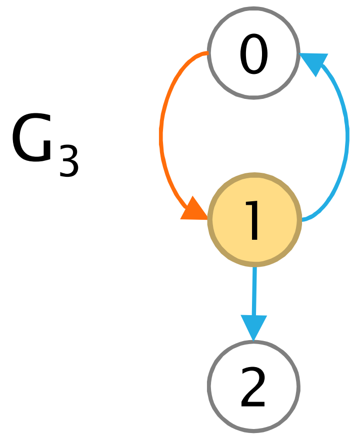
Um vértice possui *grau n* sse
há exatamente *n* arestas incidentes ao vértice

Exemplo:

grau do vértice 1: 3

grau de entrada do vértice 1: 1

grau de saída do vértice 1: 2



Caminhos e ciclos

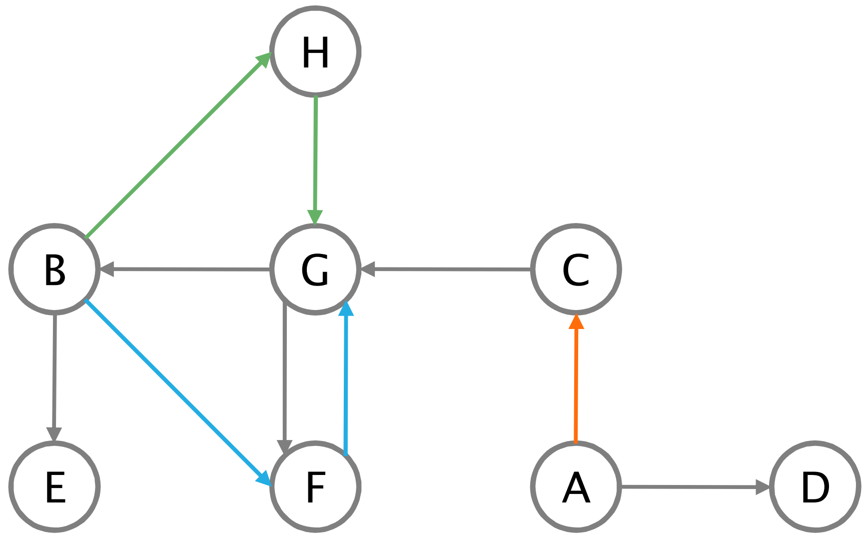
Caminho

de comprimento 1 entre A e C

de comprimento 2 entre B e G, passando por H

de comprimento 2 entre B e G, passando por F

de comprimento 3 de A a F



Ciclos

Um *ciclo* é um caminho de um nó para ele mesmo

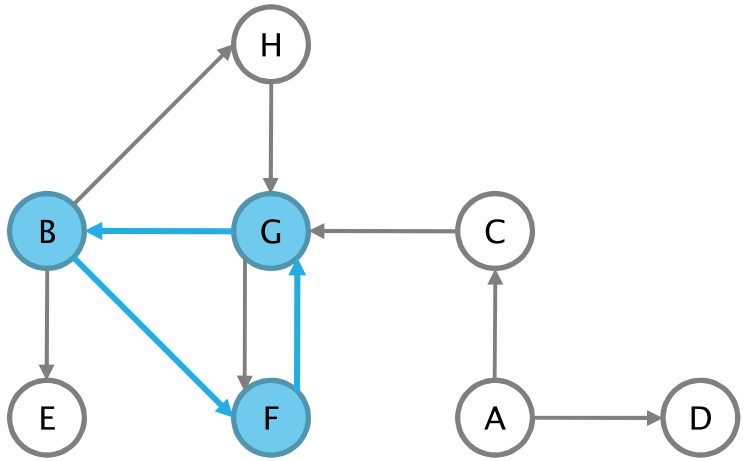
exemplo: B-F-G-B

Grafo cíclico

contém um ou mais ciclos

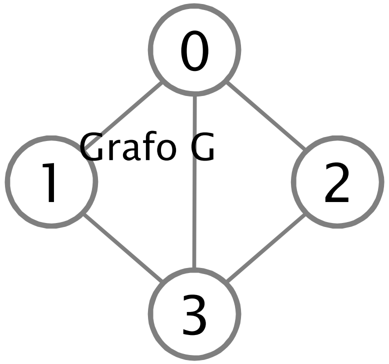
Grafo acíclico

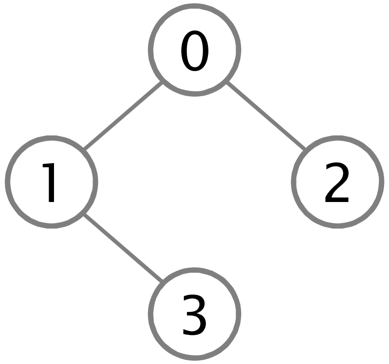
não contém ciclos

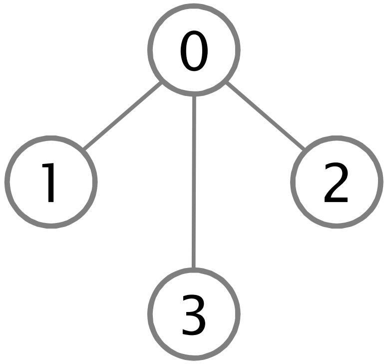


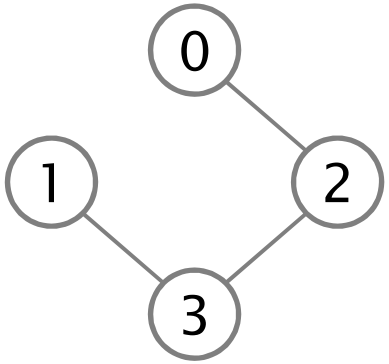
árvore geradora

subgrafo acíclico contendo todos os vértices com caminhos entre quaisquer 2 vértices









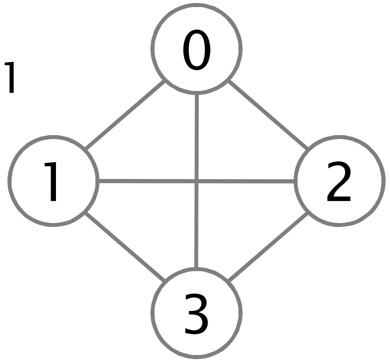
Representações de grafo

- Matriz de adjacências
- Listas de adjacências (incidências)

Matriz de adjacências

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

G_1

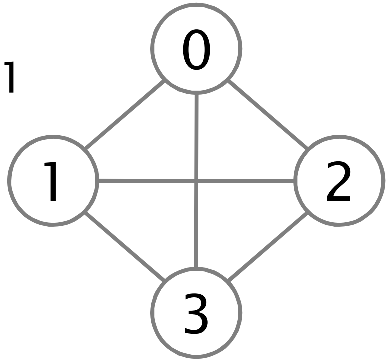


$$
\begin{array}{c c}
 & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}
\end{array}
$$

Matriz de adjacências

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

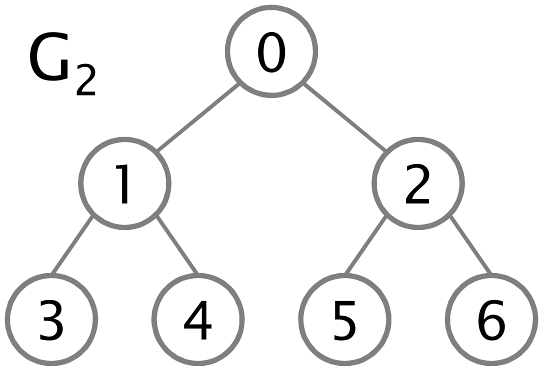
G_1



$$\begin{array}{c c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

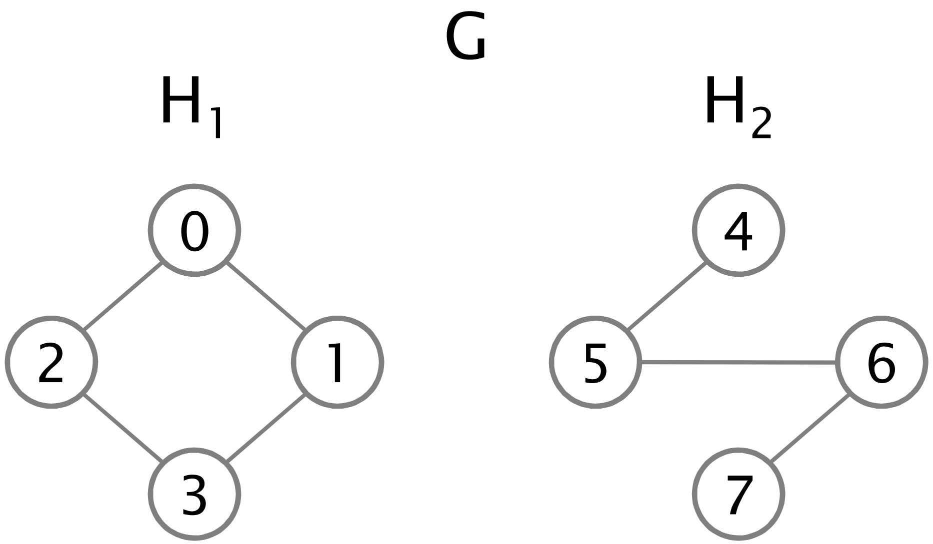
matrizes simétricas para grafos não direcionados

Matriz de adjacências



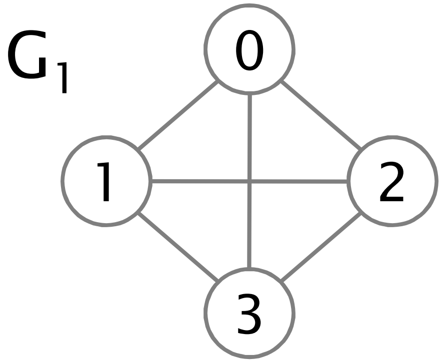
$$
\begin{array}{c@{\,}c}
 & \begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} &
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

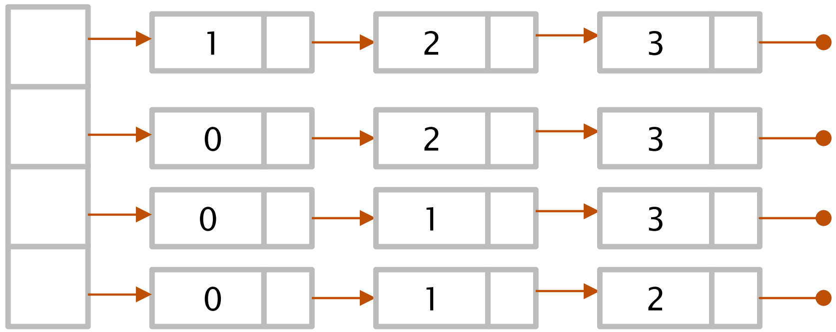
Matriz de adjacências



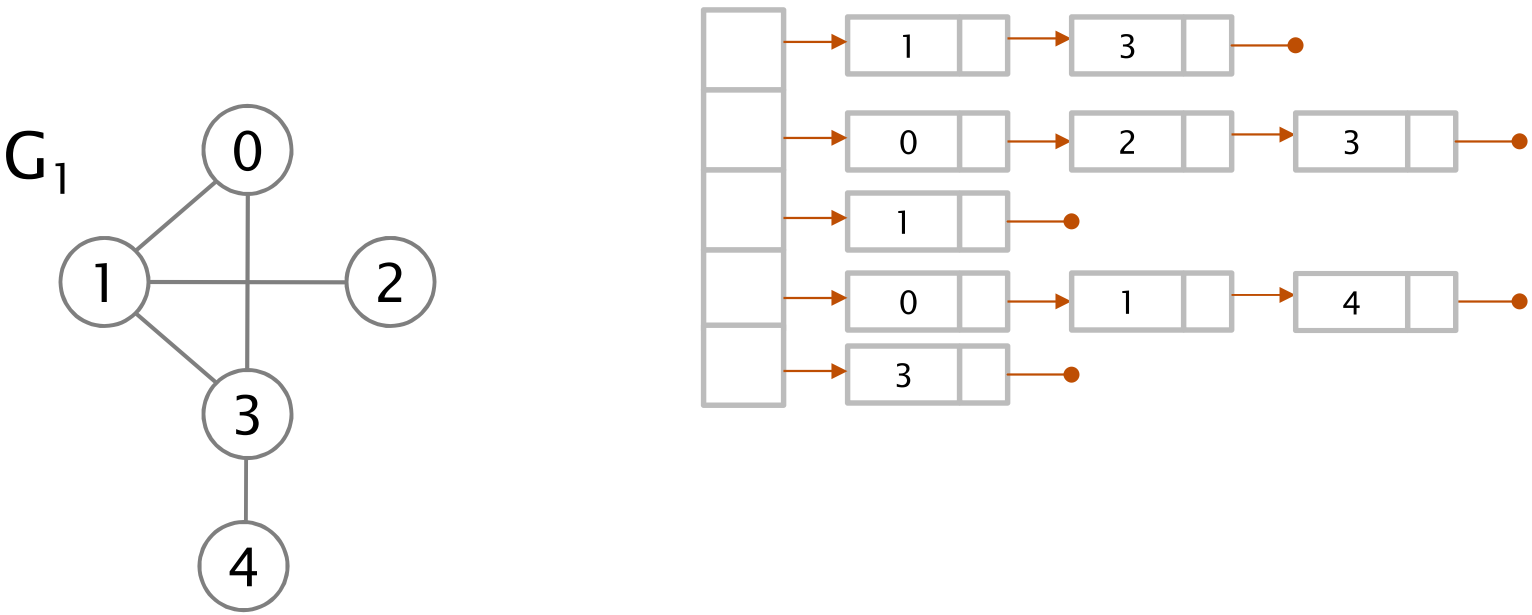
$$
\begin{array}{c c}
 & \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\end{array}
$$

Listas de adjacências





Listas de adjacências



Percursos em grafos

passeios percorrendo todos os nós de um grafo

Percursos 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 menor caminho)

Percursos em grafos

Cada vértice examinado deve ser marcado como visitado

Por quê?

Percurso em profundidade – *depth first search*

```
/* 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);
}
```

Percurso em profundidade
– *depth first search*

```
/* 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);
}
```

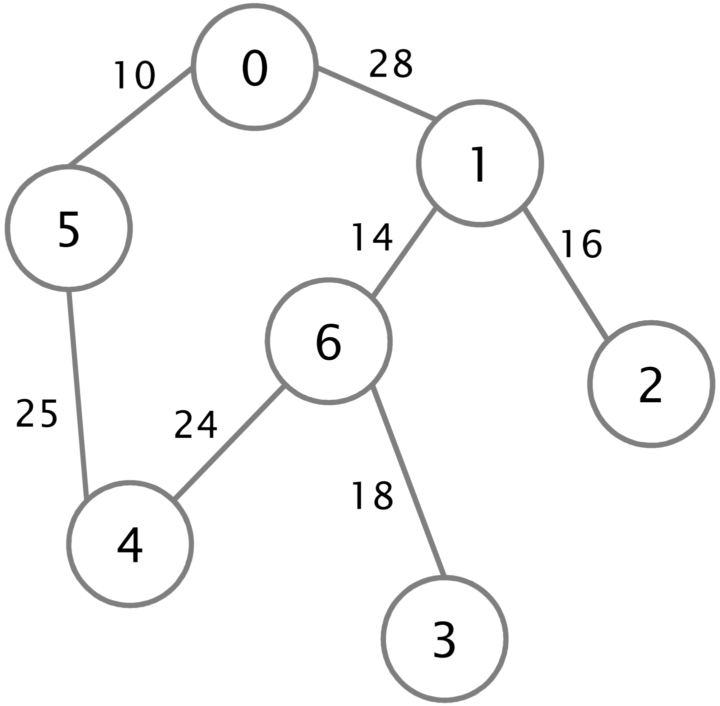
Percurso em largura
– *breadth first search*

Semelhante ao percurso por nível em uma árvore (registrando os nós visitados)

TAD Grafo

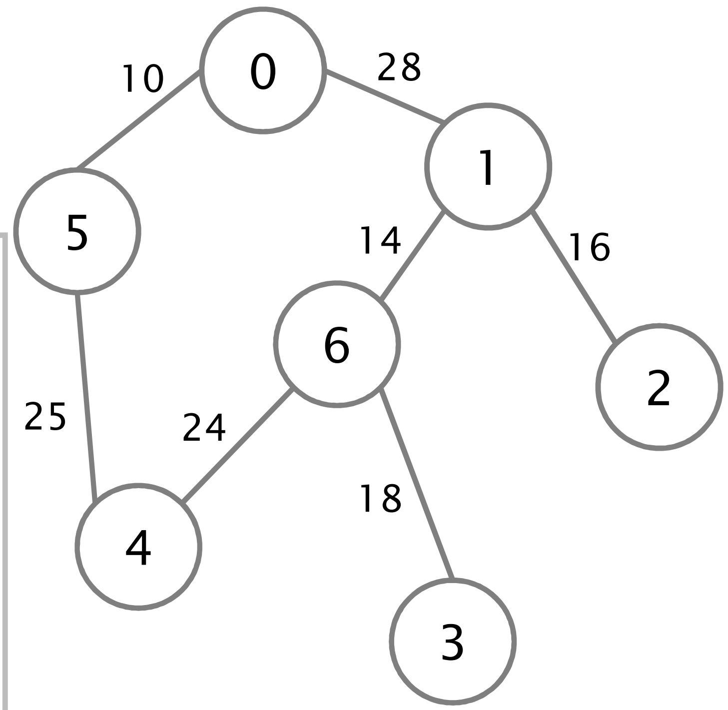
```
typedef struct graph Graph;

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);
```

Exemplo

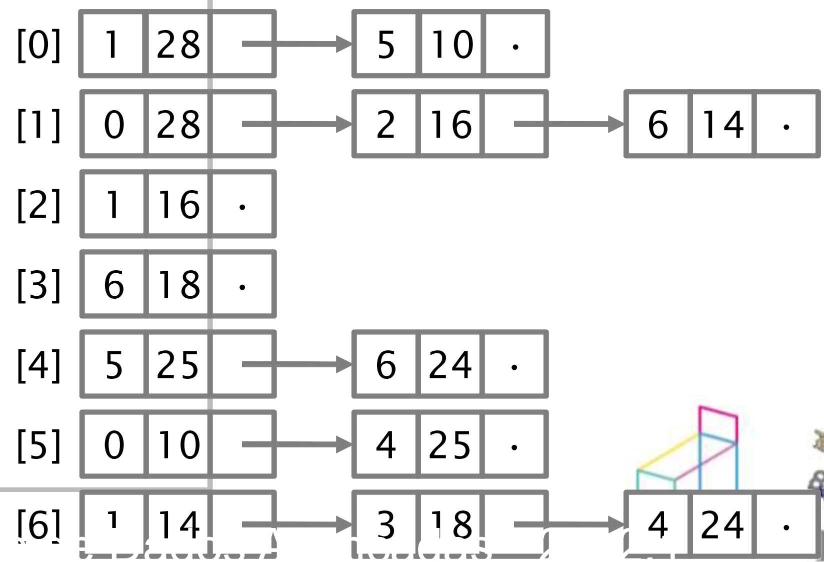



```
int main(void)
{
   Graph* g = graph_create(10);
   graph_insert_vertex(g, NULL); /* vertex 0 */
   graph_insert_vertex(g, NULL); /* vertex 1 */
   graph_insert_vertex(g, NULL); /* vertex 2 */
   graph_insert_vertex(g, NULL); /* vertex 3 */
   graph_insert_vertex(g, NULL); /* vertex 4 */
   graph_insert_vertex(g, NULL); /* vertex 5 */
   graph_insert_vertex(g, NULL); /* vertex 6 */
   graph_insert_edge(g, 0, 1, 28);
   graph_insert_edge(g, 0, 5, 10);
   graph_insert_edge(g, 1, 2, 16);
   graph_insert_edge(g, 1, 6, 14);
   graph_insert_edge(g, 3, 6, 18);
   graph_insert_edge(g, 4, 5, 25);
   graph_insert_edge(g, 4, 6, 24);
   graph_destroy(g);
   return 0;
}
```

Exemplo

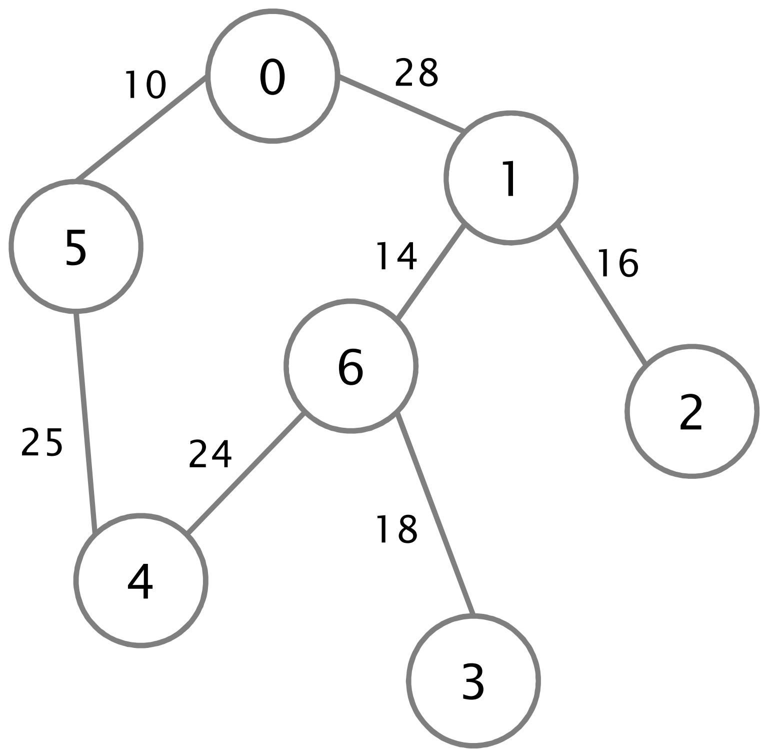



```
int main(void)
{
   Graph* g = graph_create(10);
   graph_insert_vertex(g, NULL); /* vertex 0 */
   graph_insert_vertex(g, NULL); /* vertex 1 */
   graph_insert_vertex(g, NULL); /* vertex 2 */
   graph_insert_vertex(g, NULL); /* vertex 3 */
   graph_insert_vertex(g, NULL); /* vertex 4 */
   graph_insert_vertex(g, NULL); /* vertex 5 */
   graph_insert_vertex(g, NULL); /* vertex 6 */
   graph_insert_edge(g, 0, 1, 28);
   graph_insert_edge(g, 0, 5, 10);
   graph_insert_edge(g, 1, 2, 16);
   graph_insert_edge(g, 1, 6, 14);
   graph_insert_edge(g, 3, 6, 18);
   graph_insert_edge(g, 4, 5, 25);
   graph_insert_edge(g, 4, 6, 24);
   graph_destroy(g);
   return 0;
}
```



dfs iniciando em 0

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



bfs iniciando em 0

bfs(0)
-> enfileira 1, 5 [1,5]
bfs(1)
-> enfileira 2, 6 [5,2,6]
bfs(5)
-> enfileira 4 [2,6,4]
bfs(2)
[6,4]
bfs(6)
-> enfileira 3 [4,3]
bfs(4)
[3]
bfs(3)
[]