Estruturas de Dados Avançadas (INF1010)

Grafos

Grafos

Estruturas matemáticas que **implementamos** com estruturas de dados...

Primeira aplicação conhecida: pontes de Königsberg (1736)

- saindo de um ponto de Königsberg, é possível atravessar todas as pontes exatamente uma vez e retornar ao ponto inicial?
- resposta: não (apenas se o *grau* de cada vértice for par: entrar por uma aresta e sair por outra, ambas não visitadas antes)

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

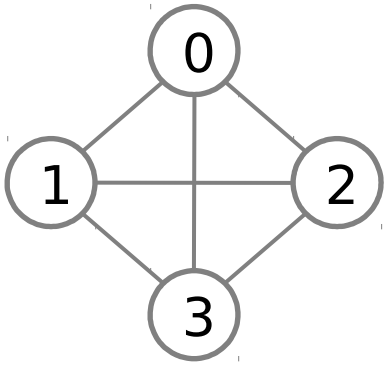
...

Grafo Não Dirigido

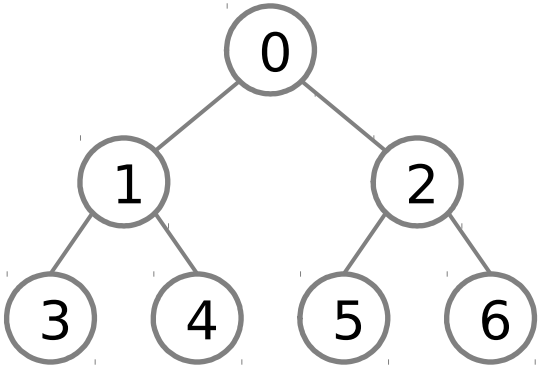
G = (V,E)

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

→ E é um conjunto de **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)

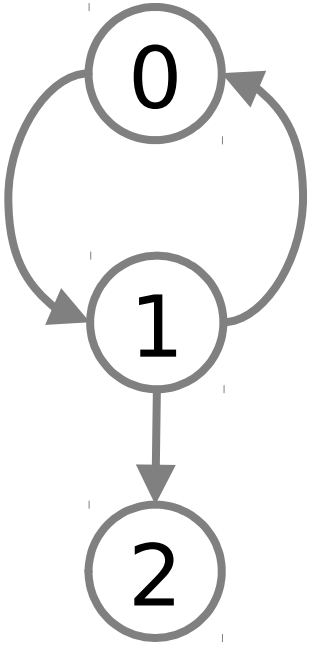
G = (V,E)

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

→ E é um conjunto de **arcos** (conexões direcionadas entre dois vértices)

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

pares ordenados de vértices



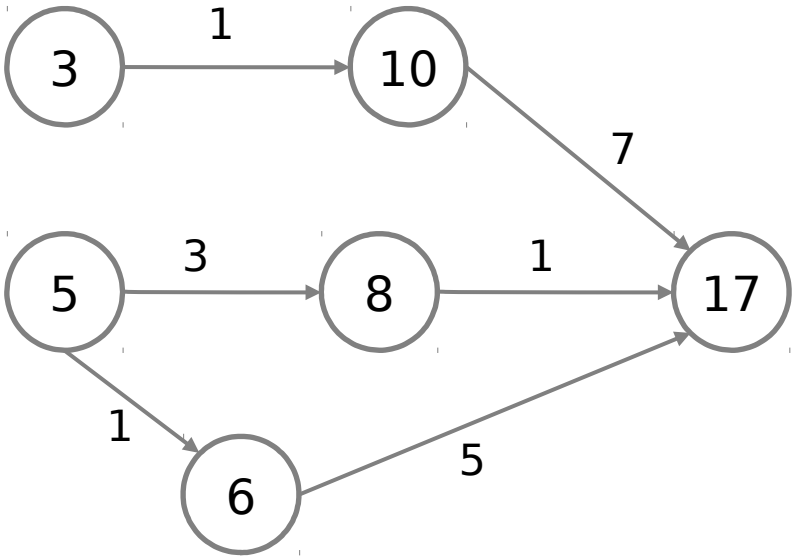
Grafo Ponderado

G = (V,E,p)

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

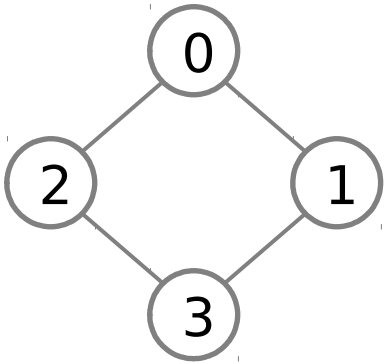
→ E é um conjunto de **arcos**

→ p é uma função que atribui um **peso** a cada arco

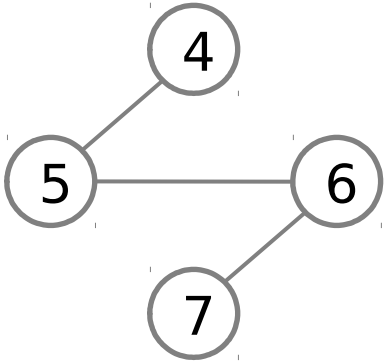


Vértices Adjacentes

Vértices conectados por arestas



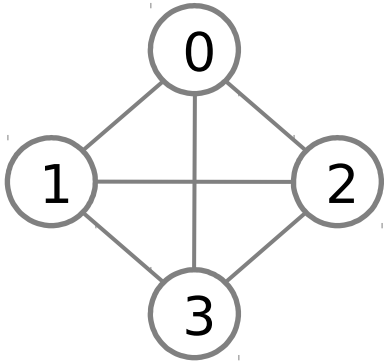
0 e 1
0 e 2
1 e 3
2 e 3



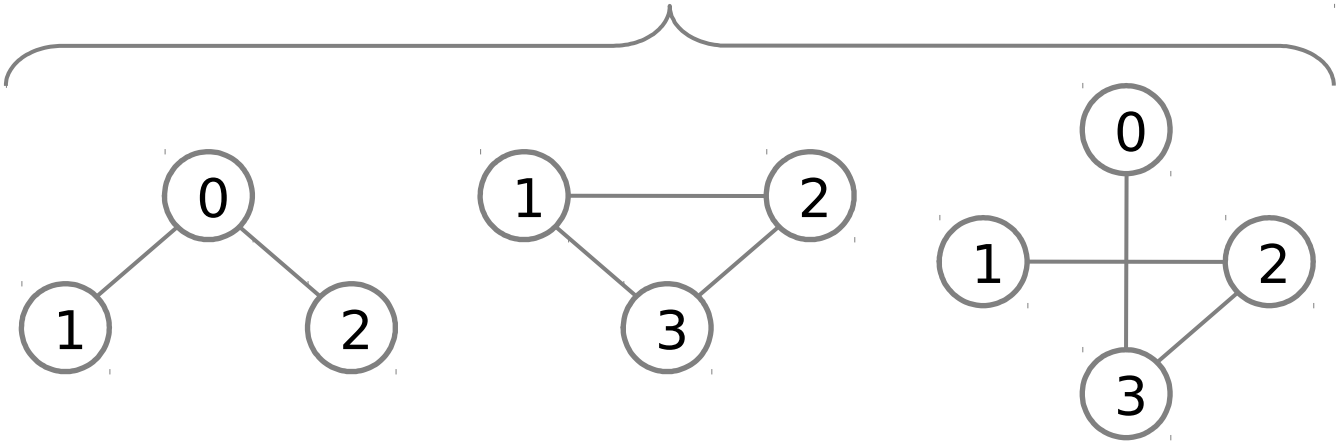
4 e 5
5 e 6
6 e 7

Subgrafo

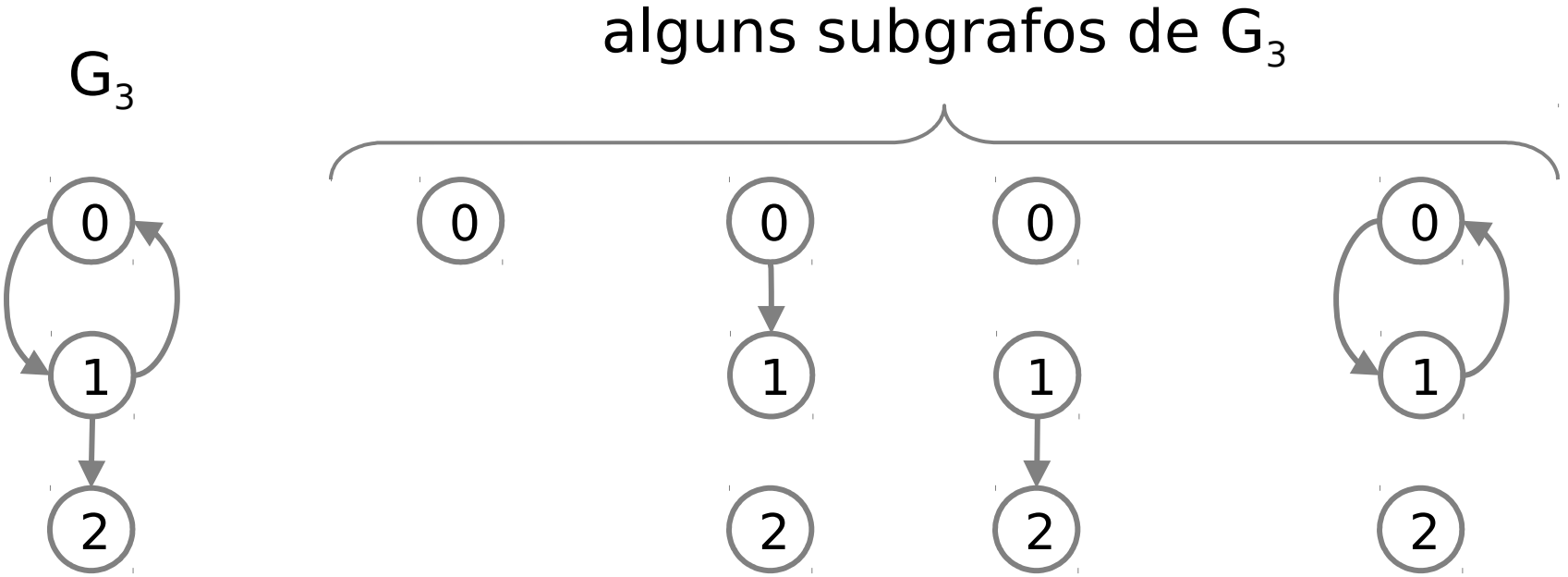
G_1



alguns subgrafos de G_1

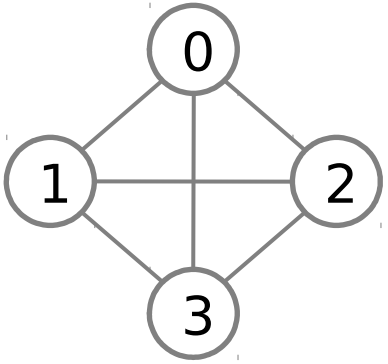


Subgrafo



Grafo Completo

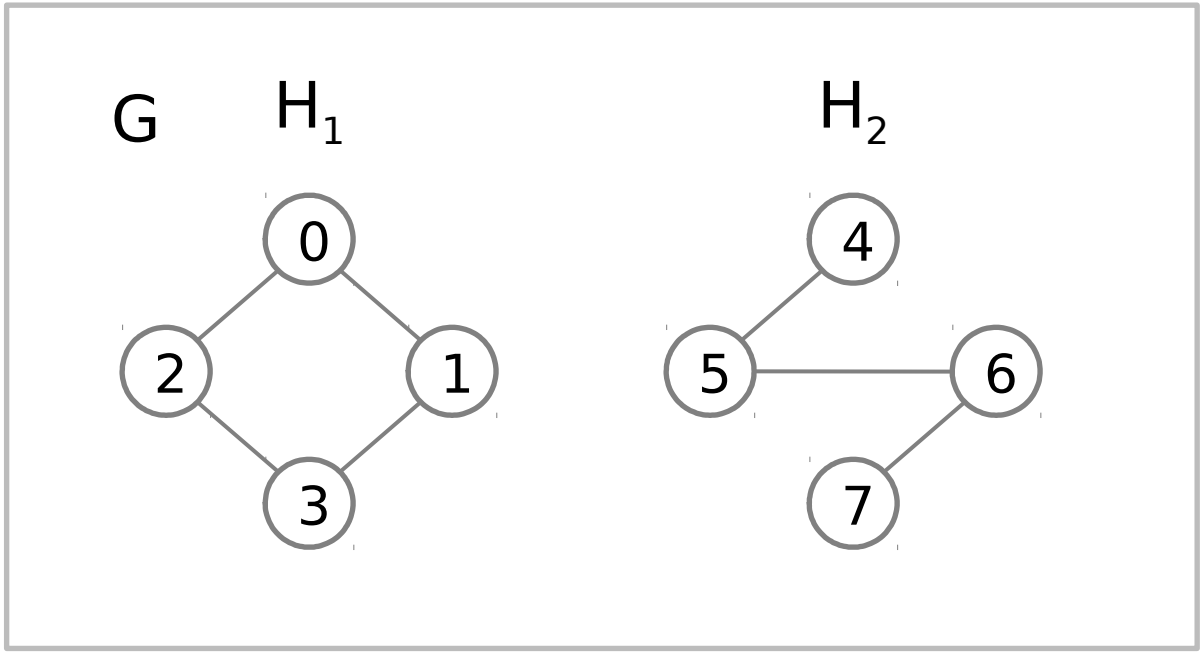
Um grafo não direcionado é **completo** sse cada vértice está conectado a cada um dos outros vértices



Grafo Conectado

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

→ componentes **conexos** de um grafo

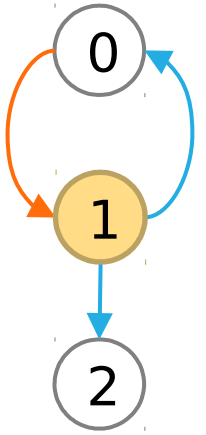


Grau

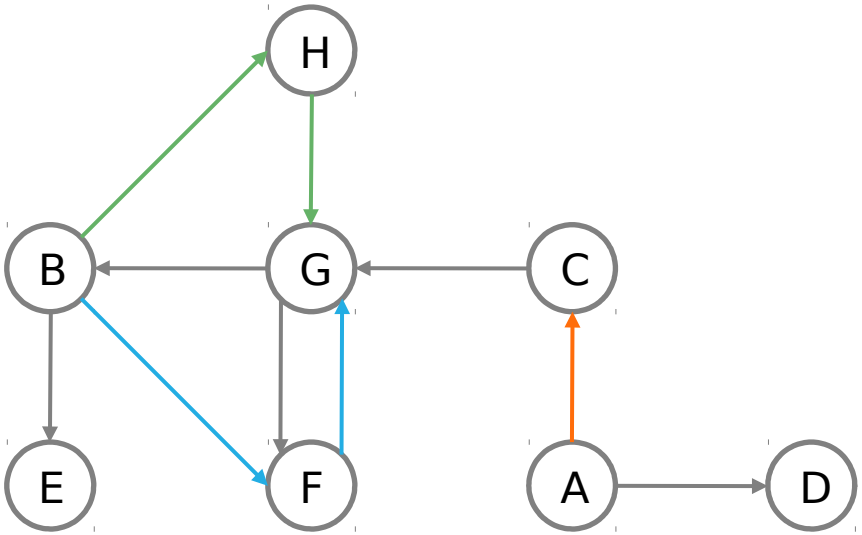
Um vértice possui **grau** *n* se há exatamente *n* arestas nele incidentes

grau do vértice 1: 3
grau de entrada do vértice 1: 1
grau de saída do vértice 1: 2

G



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

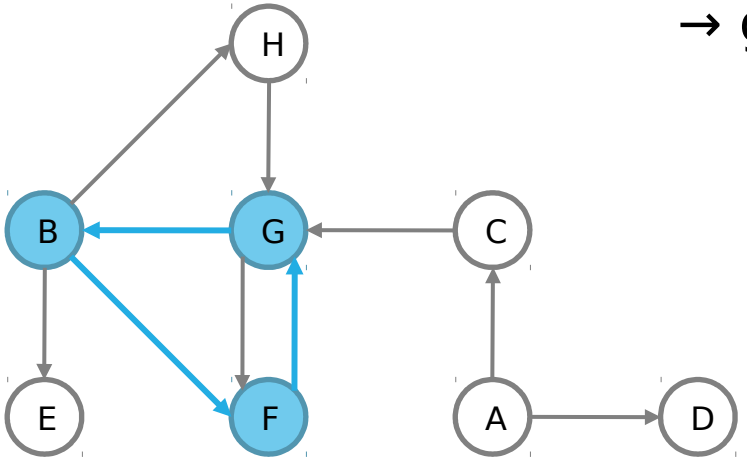
→ caminho de comprimento 3 entre A e F

Ciclos

Um ciclo é um caminho de um vértice a ele mesmo

→ grafo **cíclico** contem um ou mais ciclos

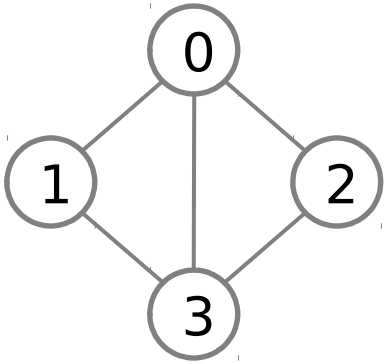
→ grafo **acíclico** não contem ciclos



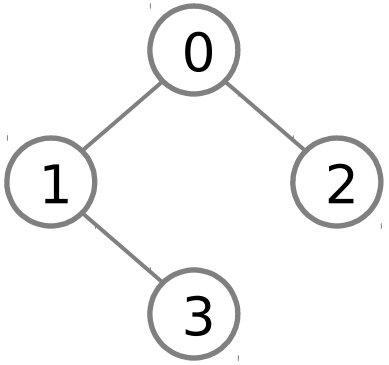
B-F-G-B

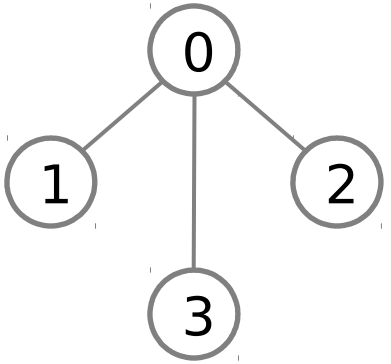
Árvore Geradora

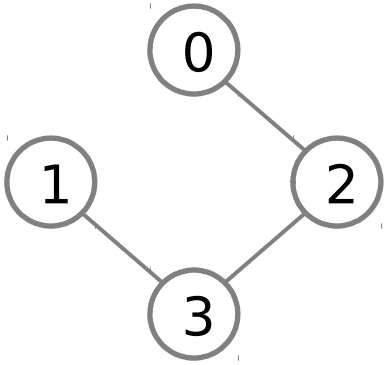
Subgrafo **acíclico** que contem todos os vértices, com caminhos entre quaisquer dois vértices



Grafo G







Representações de Grafos

Matriz de adjacências (V x V)

→ cada elemento m[i][j] representa uma aresta de v_i a v_j

→ espaço $O(V^2)$: grafos densos ($E \sim V^2$)

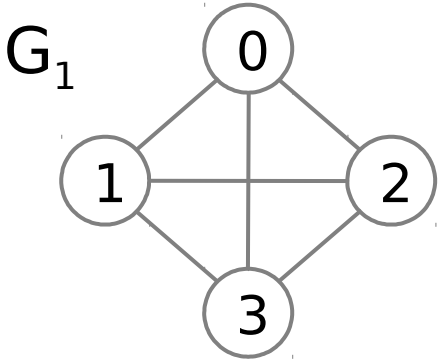
Lista de adjacências

→ vetor de vértices, cada vértice tem lista de arestas

→ espaço $O(V + E)$: grafos esparsos ($E << V^2$)

Matriz de Adjacências

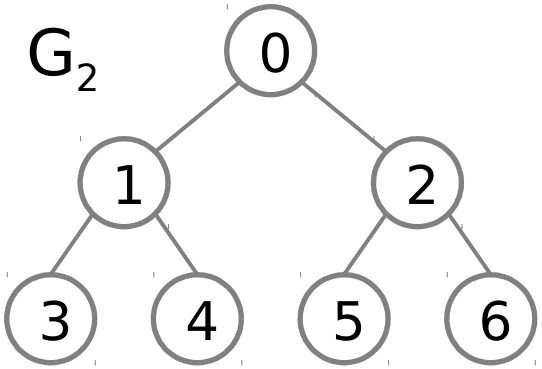
$$mat[i][j] = \begin{cases} 1, \text{ se houver uma aresta do nó i para o nó j} \\ 0, \text{ caso contrário} \end{cases}$$



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

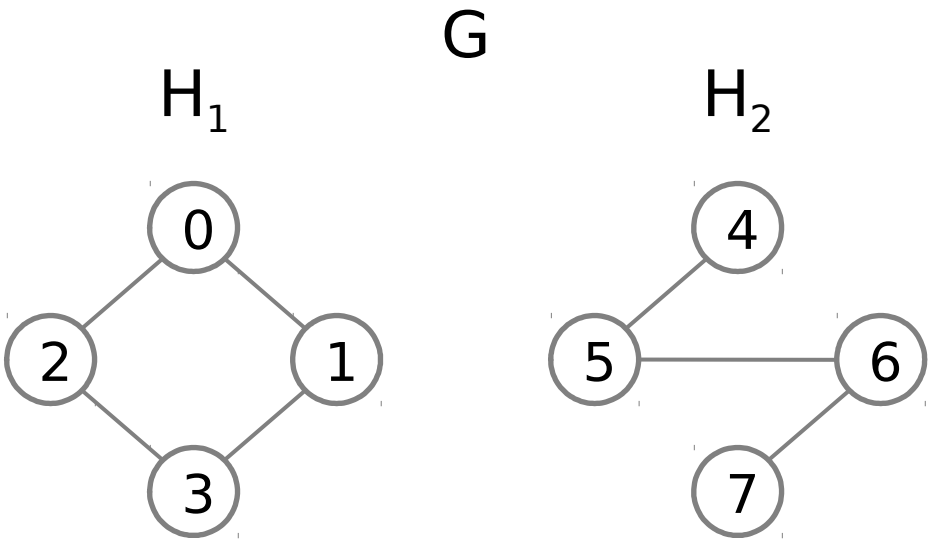
→ matrizes simétricas para grafos não direcionados

Matriz de Adjacências



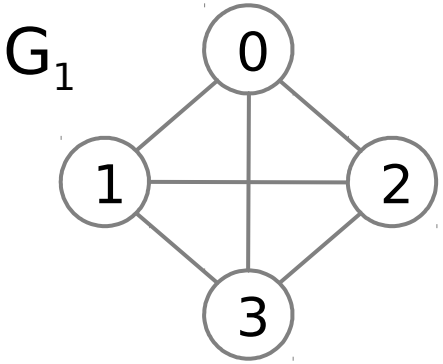
	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	0	1	1	0	0
2	1	0	0	0	0	1	1
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0
5	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0

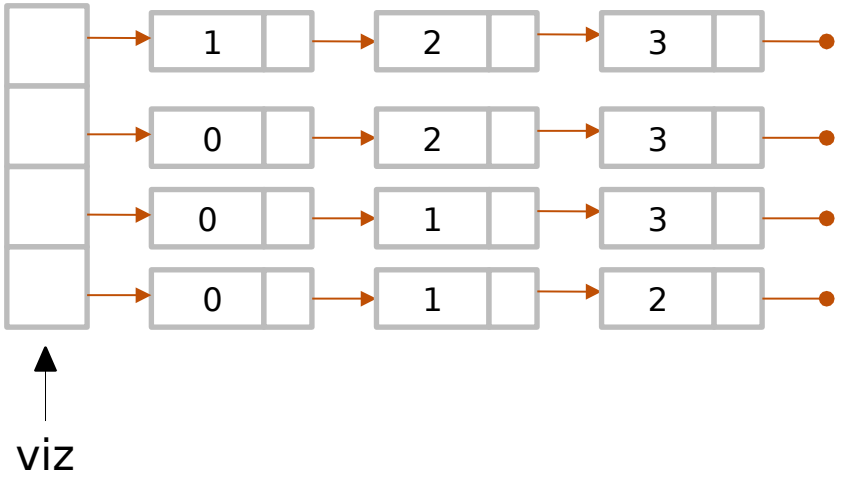
Matriz de Adjacências



	0	1	2	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**)

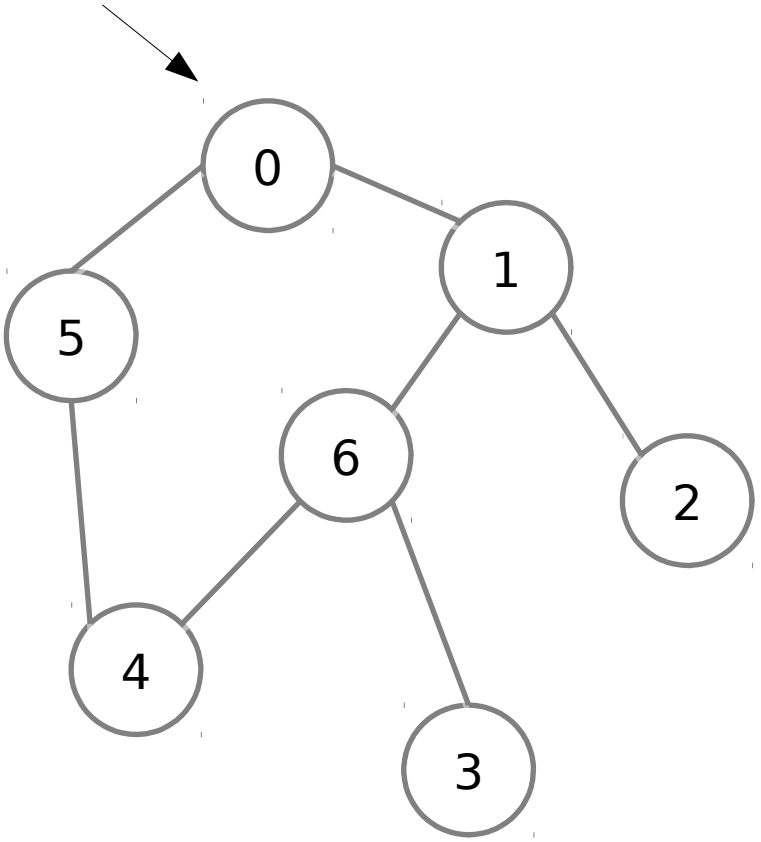
→ arestas que partem do vértice visitado primeiro

Guloso (*greedy*)

→ arestas de menor custo

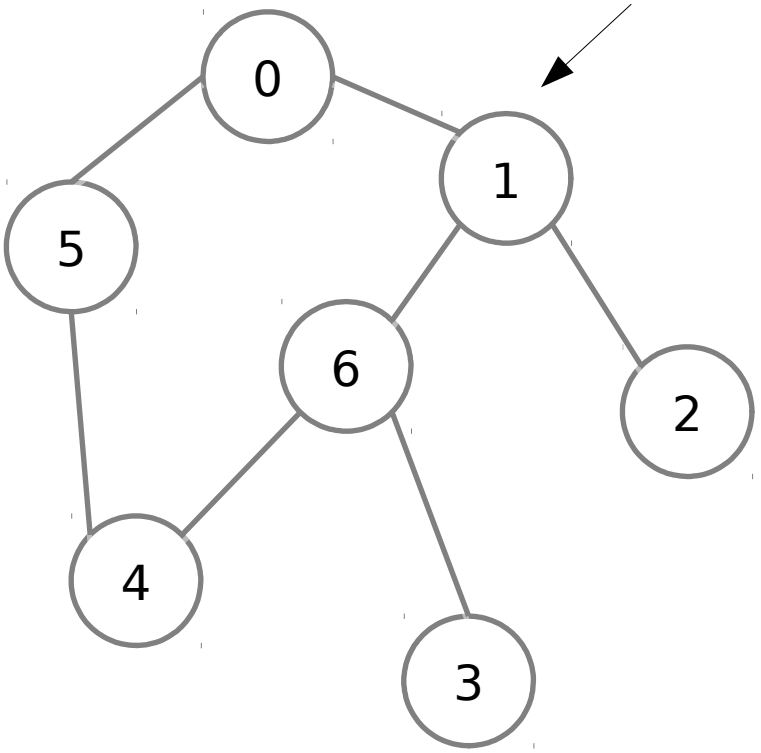
→ tipicamente procurando caminho mínimo

dfs com recursão



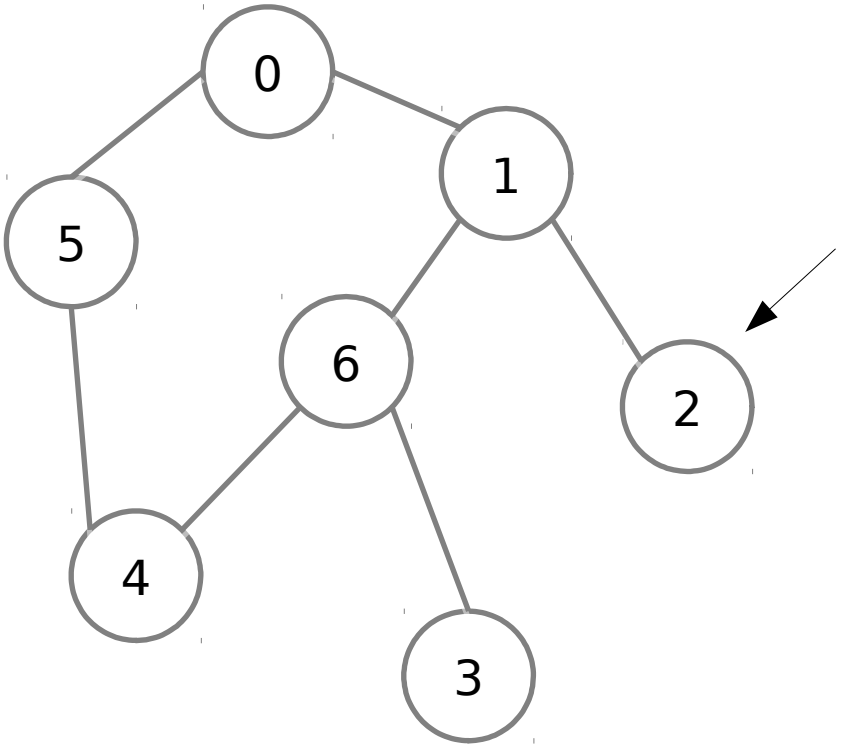
dfs(0)

dfs com recursão



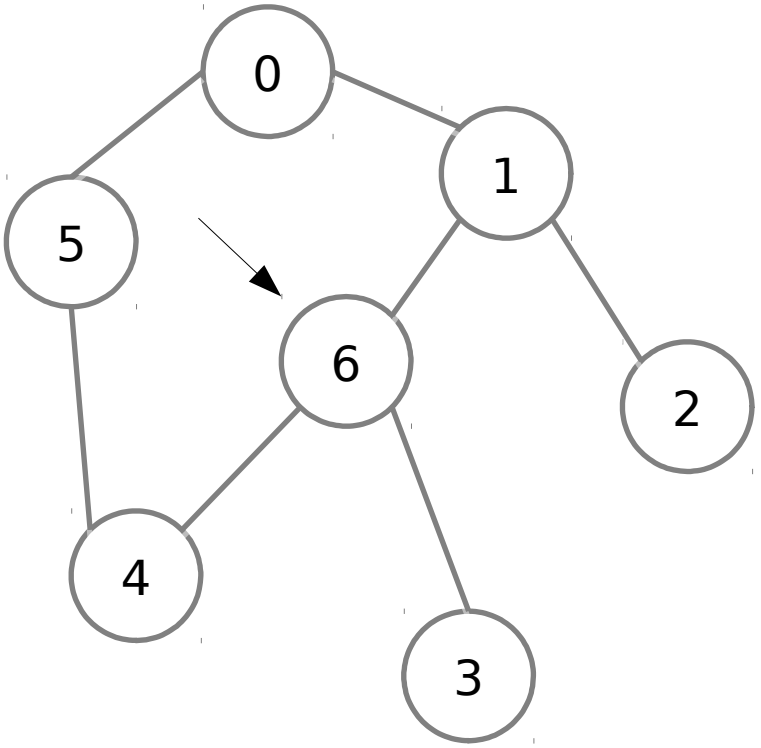
dfs(0)
    dfs(1)

dfs com recursão



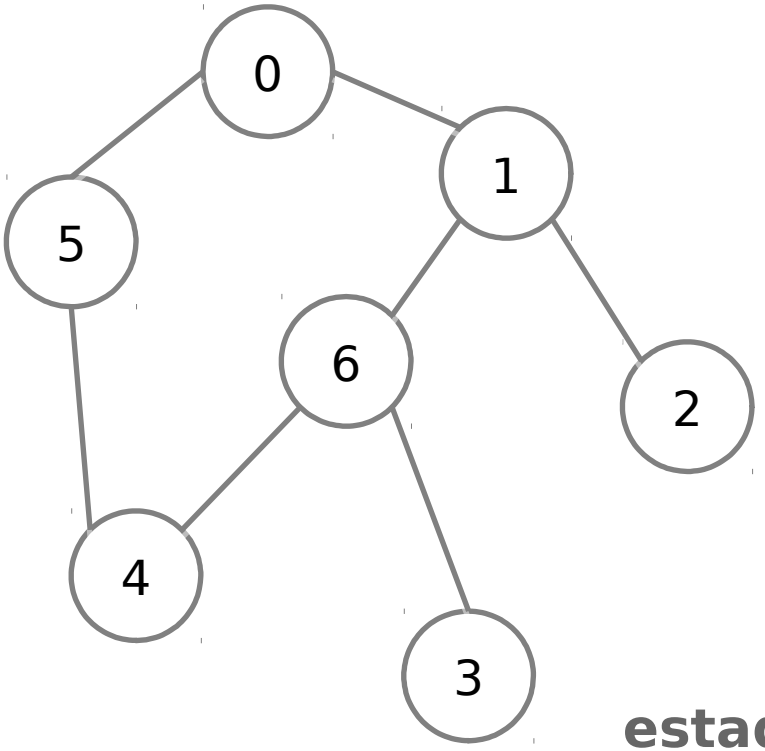

```
dfs(0)
    dfs(1)
        dfs(2)
```

dfs com recursão




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

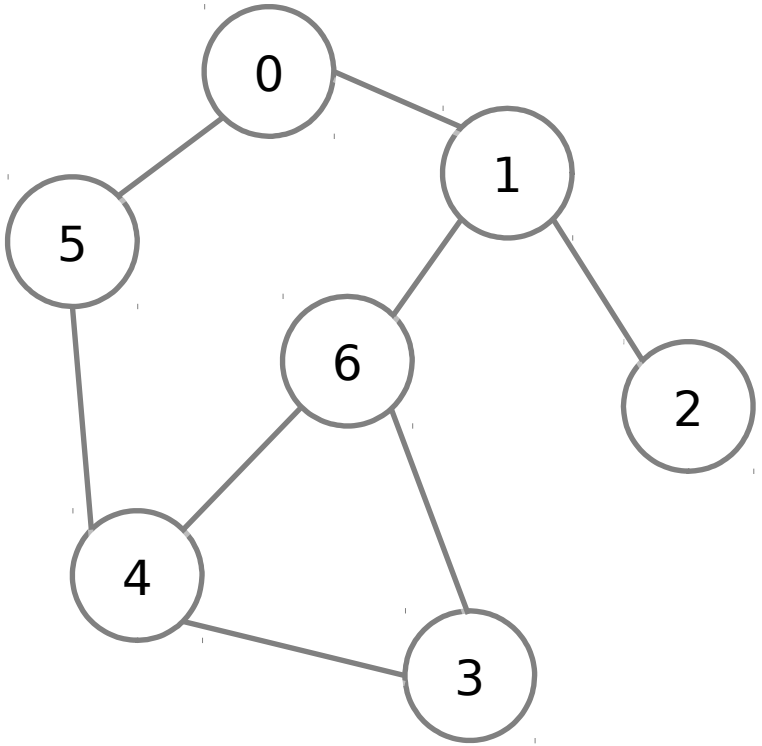
dfs com recursão




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

estado fica na pilha de chamadas recursivas!

dfs com recursão: outro exemplo

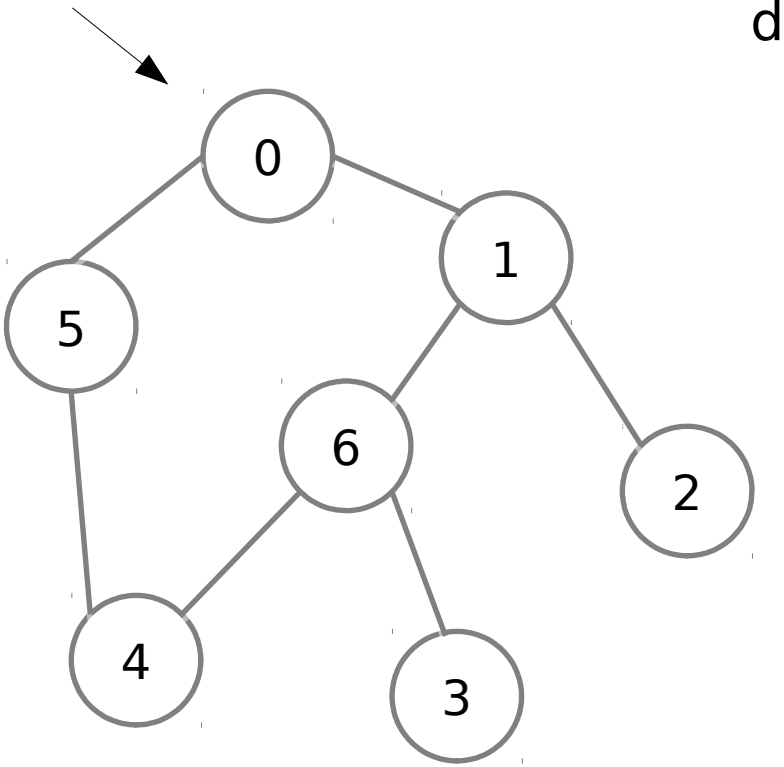



```
dfs(0)
  dfs(1)
    dfs(2)
    dfs(6)
      dfs(3)
      dfs(4)
        dfs(6)
        dfs(5)
        dfs(0)
```

já foram visitados!

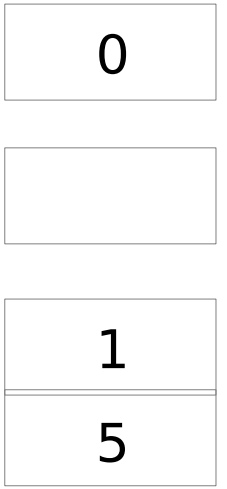
estrutura de marcação indica nós já visitados!

dfs com Pilha

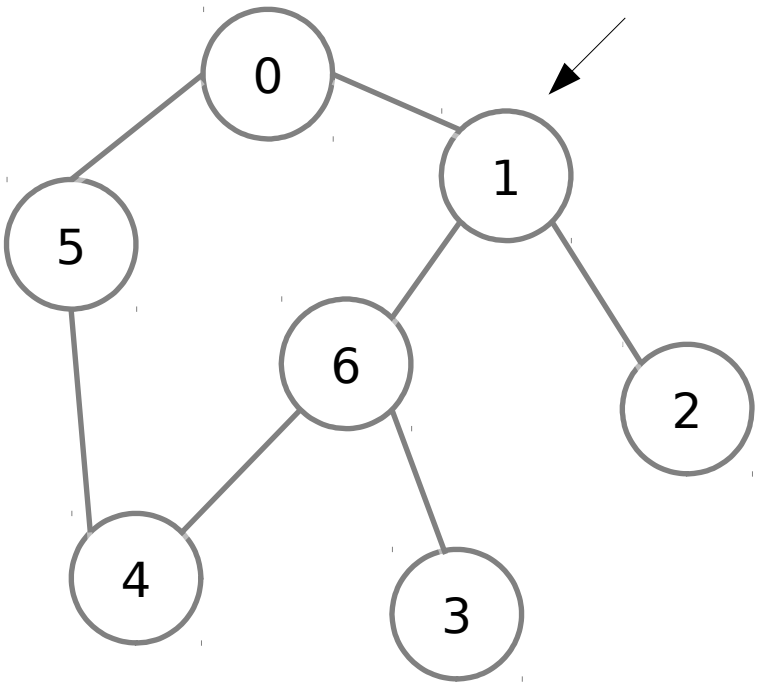


dfs(0)
empilha 0

desempilha 0
visita 0
empilha: 5:1



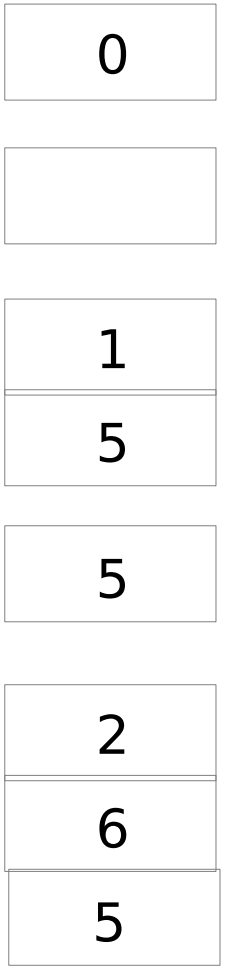
dfs com Pilha



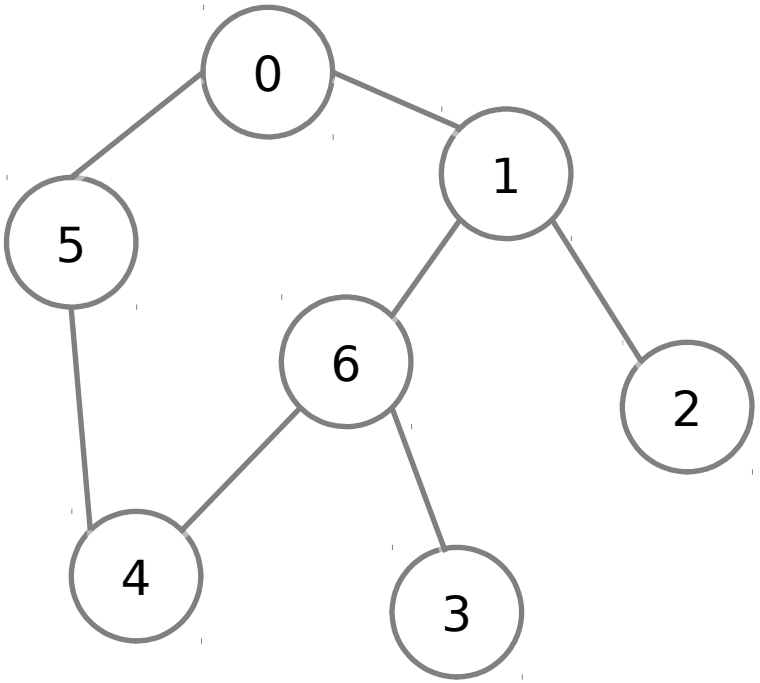
dfs(0)
empilha 0

desempilha 0
visita 0
empilha: 5:1

desempilha 1
visita 1
empilha 6:2



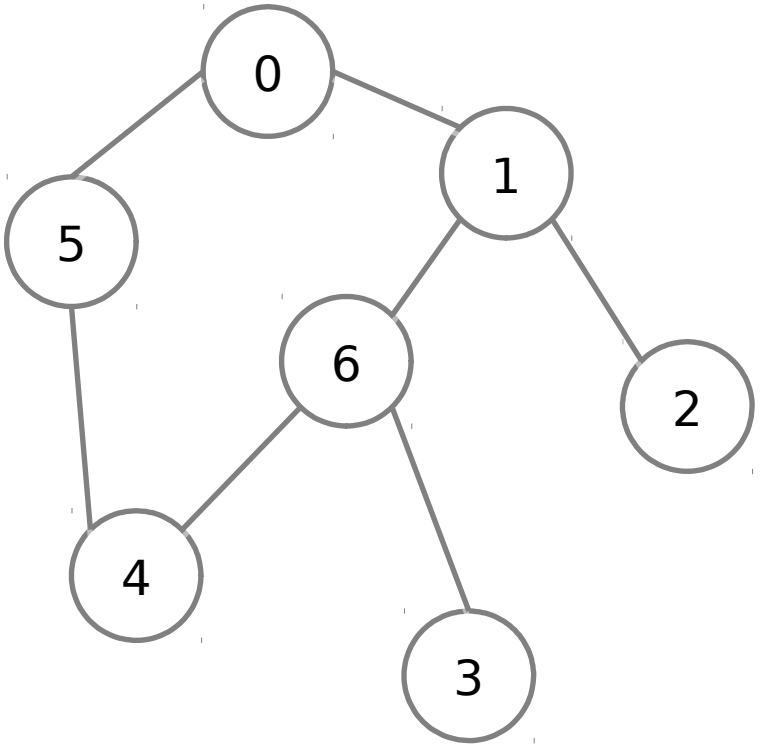
dfs com Pilha



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	

Percurso bfs



bfs(0)

visita 0
visita 1 → arestas de 0
visita 5

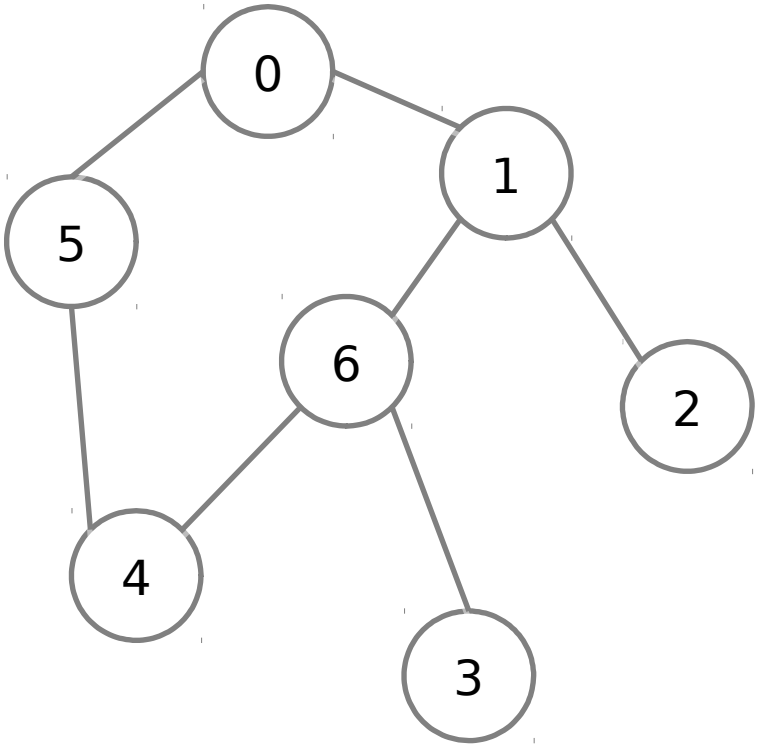
visita 2 → arestas de 1
visita 6

visita 4 → arestas de 5

visita 3 → arestas de 6

como registrar os nós pendentes?

Percurso bfs



bfs(0)

visita 0
visita 1 → arestas de 0
visita 5

visita 2 → arestas de 1
visita 6

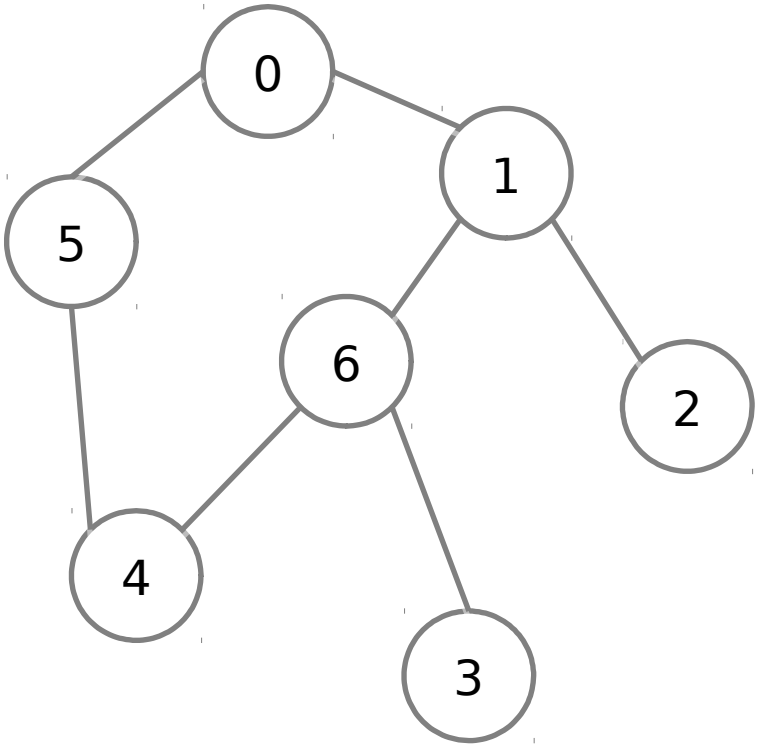
visita 4 → arestas de 5

visita 3 → arestas de 6

como registrar os nós pendentes?

→ **fila** de nós

bfs com Fila Auxiliar



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
[]