



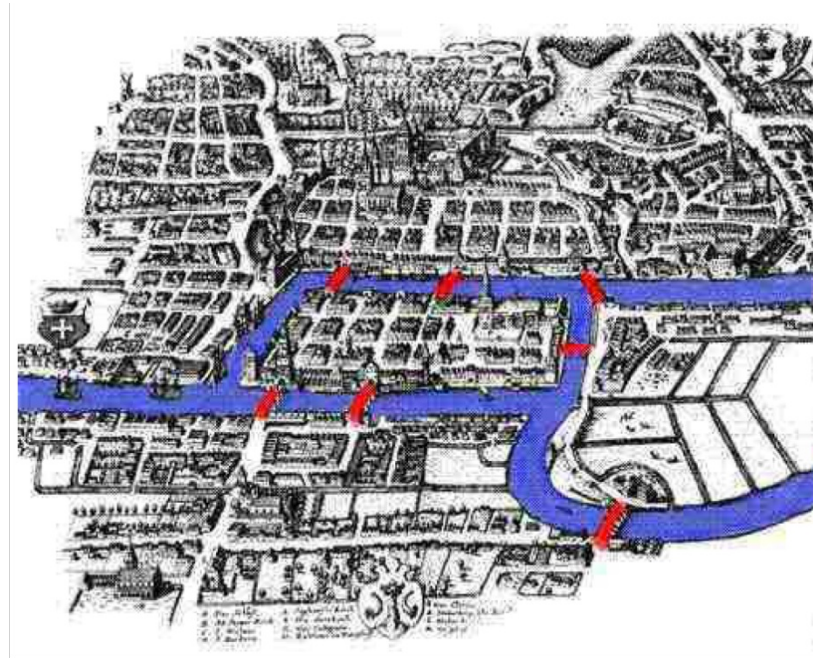
# INF 1010

## Estruturas de Dados Avançadas

### Grafos

# Primeiro uso conhecido

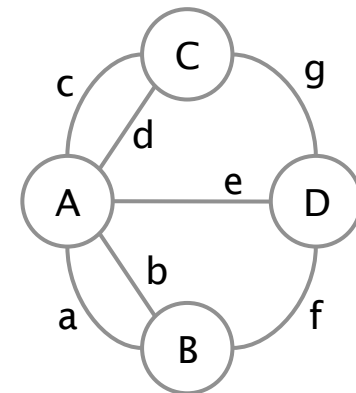
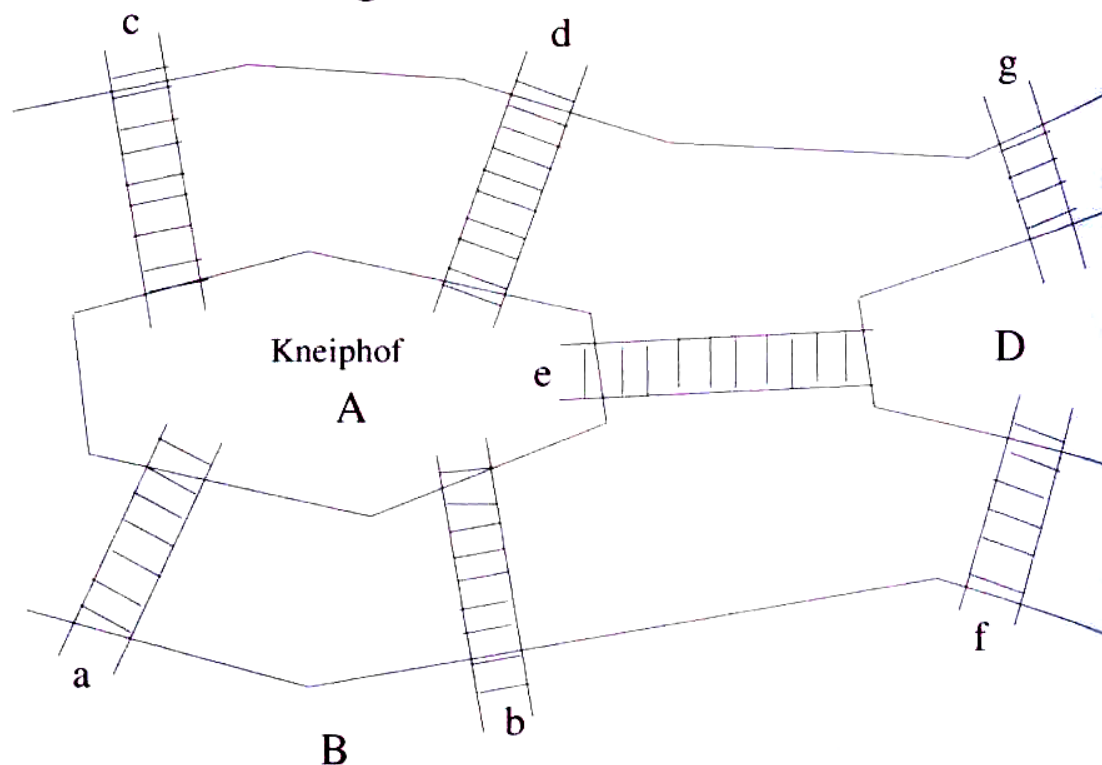
## 1736 Euler: pontes de Königsberg



# Primeiro uso conhecido

## 1736 Euler: pontes de Königsberg

Saindo de uma margem, é possível atravessar todas as pontes exatamente uma vez e retornar à mesma margem?



# Caminho Euleriano

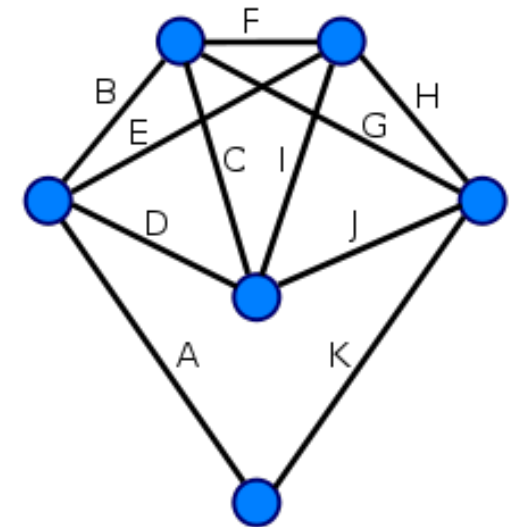
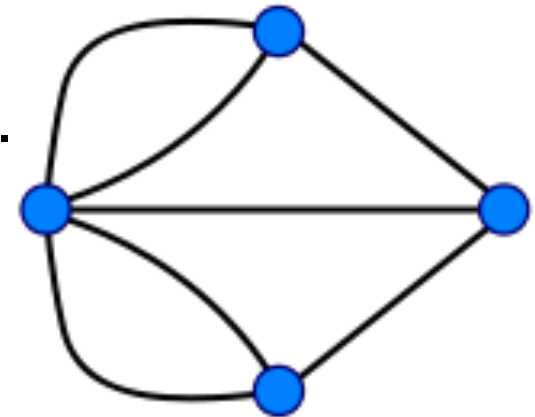
Caminho Euleriano:

caminho que visita cada aresta apenas uma vez.

Circuito Euleriano:

caminho Euleriano que começa e termina no mesmo vértice (**Grafos Eulerianos**).

Um grafo conexo possui caminho euleriano se e somente se ele tem exatamente zero ou dois vértices de grau ímpar.

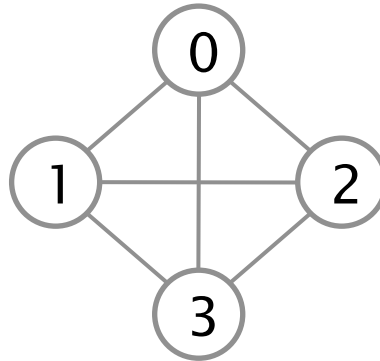


# Definição

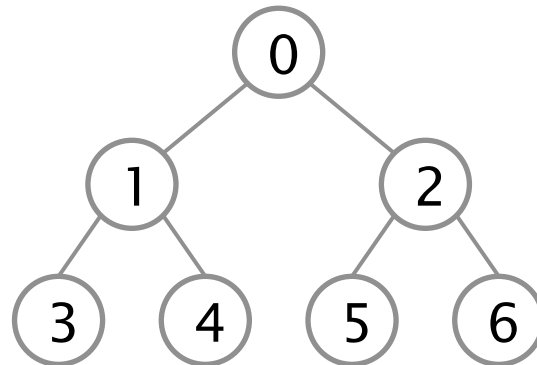
$$G = (V, E)$$

um conjunto  $V$  de  $n$  nós (vértices, *vertices*) e  
um conjunto  $E$  de  $m$  arcos (arestas, *edges*)

exemplos



vértices:  $V = \{0, 1, 2, 3\}$   
arestas:  $E = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)\}$   
 $n = 4$   
 $m = 6$



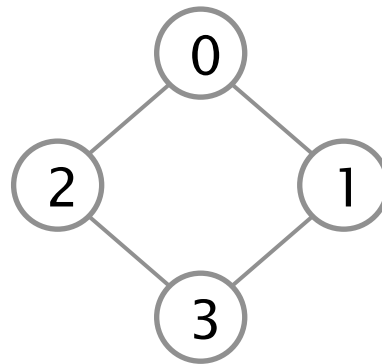
vértices :  $V = \{0, 1, 2, 3, 4, 5, 6\}$   
arestas:  $E = \{(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)\}$   
 $n = 7$   
 $m = 6$

# Usos de grafo

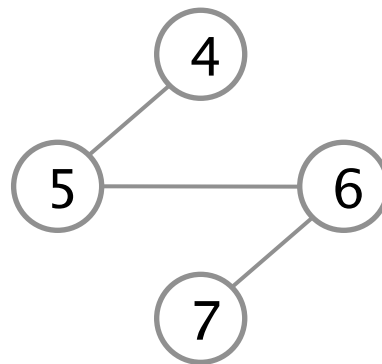
grafo	vértices	arestas
transporte	interseções de ruas	ruas
comunicação	computadores	cabos
Web	páginas Web	links
sociedade	pessoas	relacionamentos
software	funções	chamadas de função
cronograma	tarefas	restrições de preferência
...		

# Vértices adjacentes

vértices conectados por arestas



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



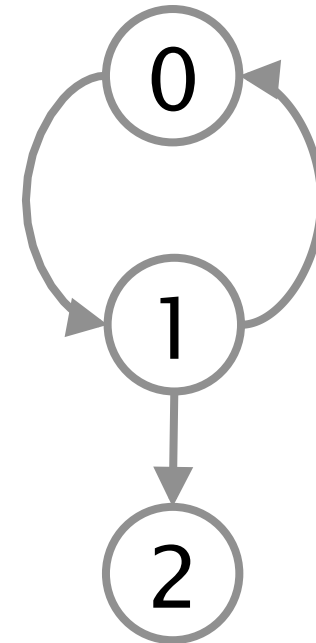
4 e 5  
5 e 6  
6 e 7

# Digrafo: grafo direcionado (orientado)

vértices:  $V = \{0, 1, 2\}$

arestas (pares ordenados):

$E = \{ \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 2 \rangle \}$

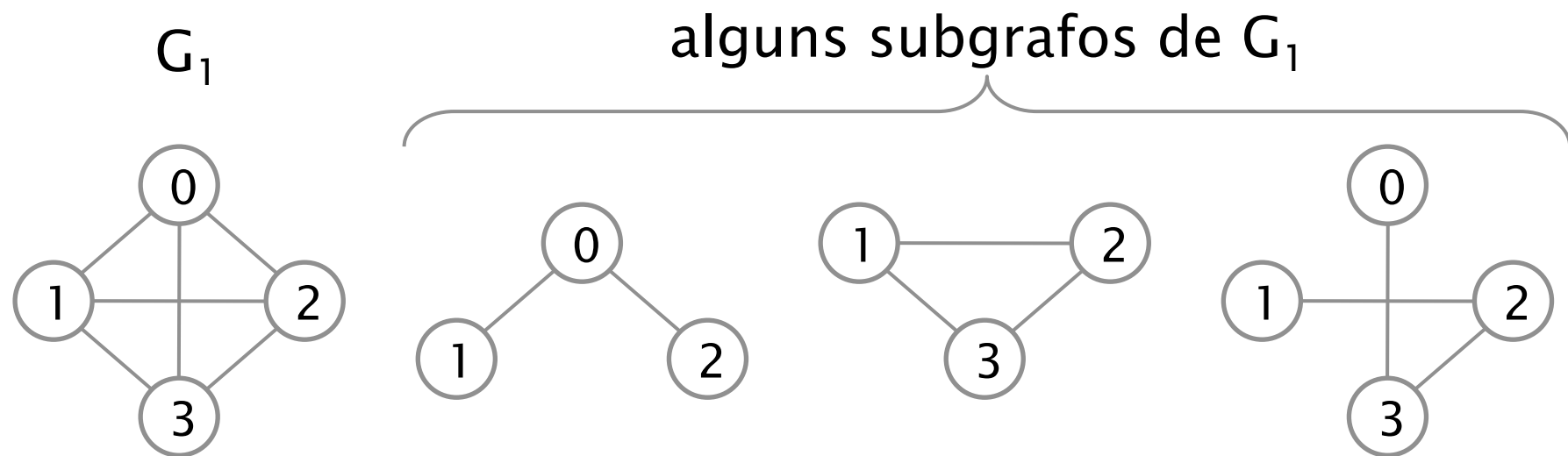


Qual é o número máximo de arestas em um grafo direcionado de  $n$  vértices?

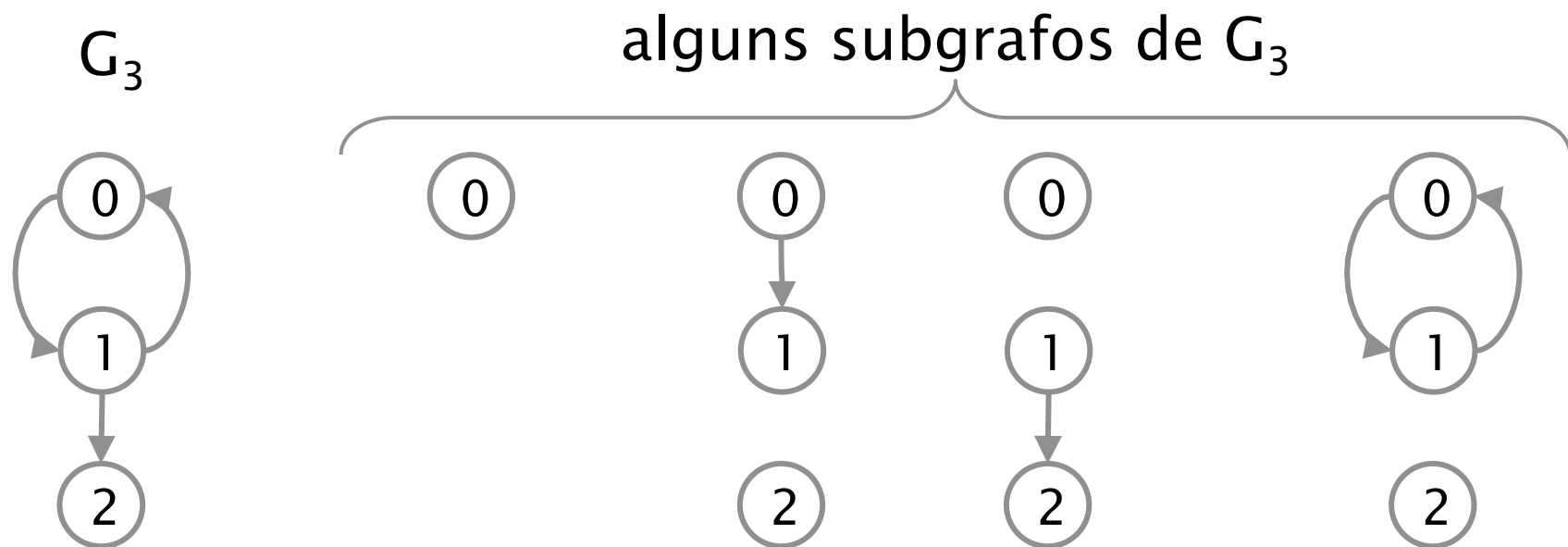
$n(n-1)$



# Subgrafo

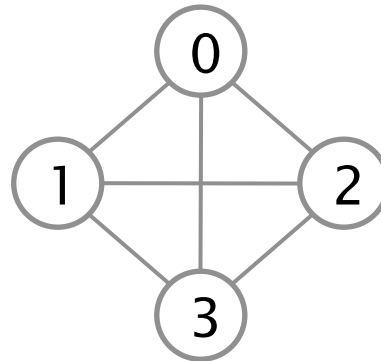


# Subgrafo



# Grafo completo

grafo não direcionado, em que cada vértice está conectado com cada um dos outros vértices por apenas uma aresta



Quantas arestas há em um grafo completo de  $n$  vértices?

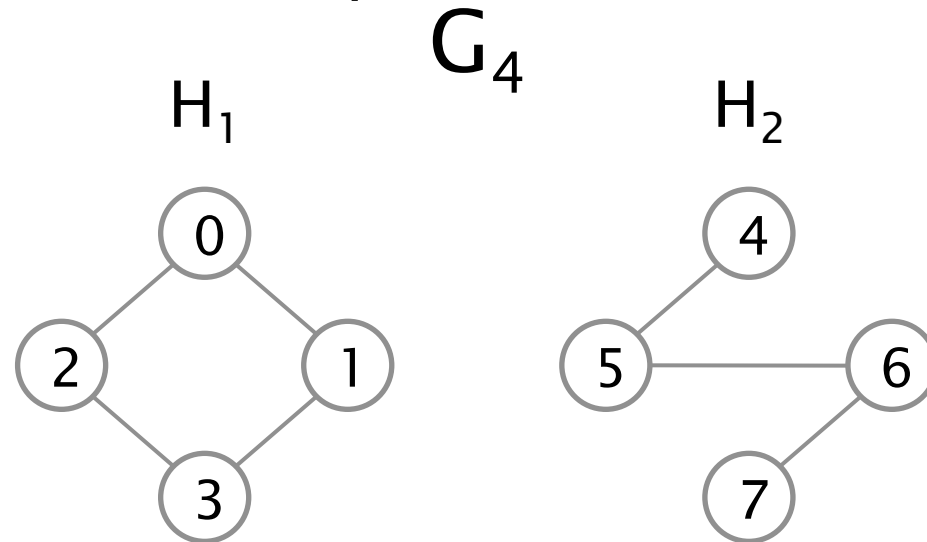
$$n(n-1)/2$$

# Grafo conectado (ou conexos)

existe um caminho entre quaisquer dois  
vértices

exemplo

grafo com 2 componentes conectados



# Grau

número de arestas incidentes em um vértice  
exemplo:

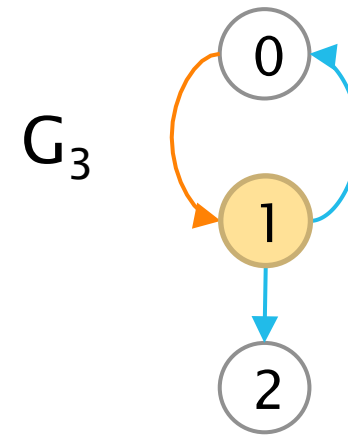
grau do vértice (1): 3

grau de **entrada** do vértice (1): 1

grau de **saída** do vértice (1): 2

voltando ao problema da  
ponte de Königsberg

seria possível somente se o grau de todos os nós  
fosse par



# Caminhos e ciclos

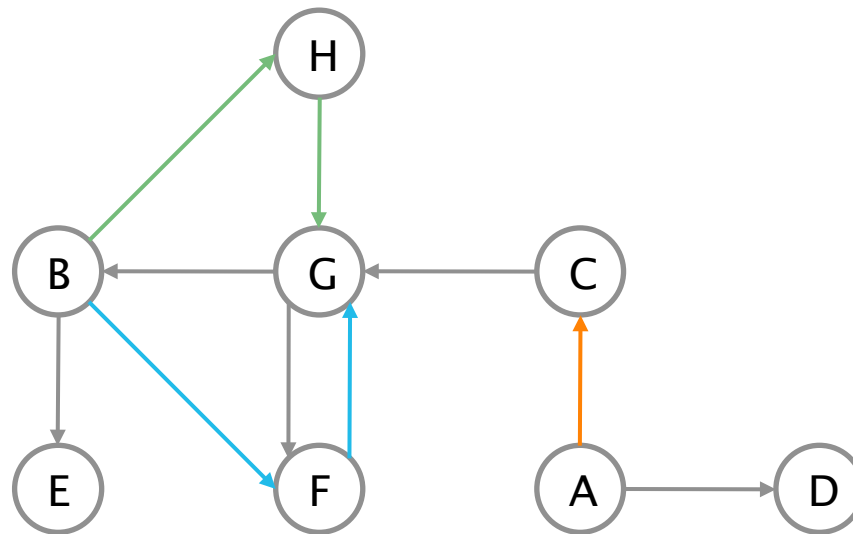
## caminhos

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



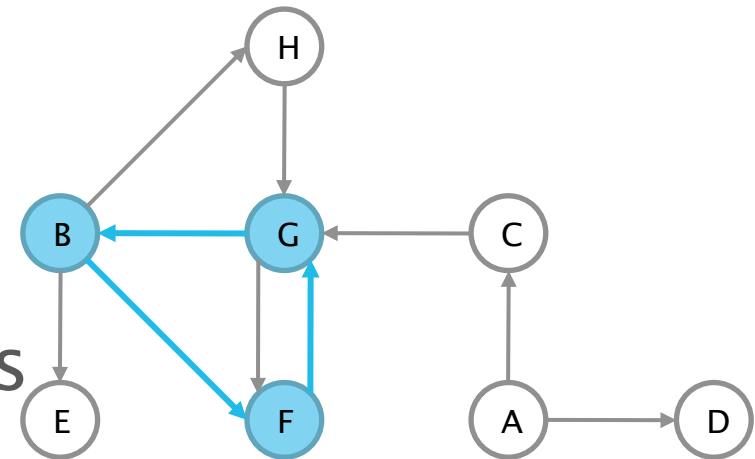
# Ciclos

caminho de um nó para si mesmo

exemplo: B-F-G-B

grafo cíclico

contém um ou mais ciclos

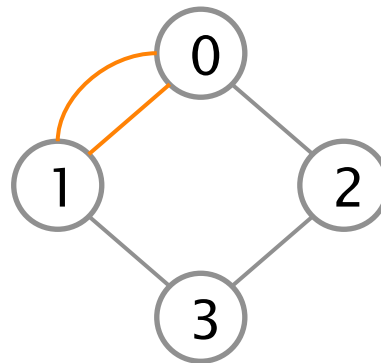


grafo acíclico

não contém ciclos

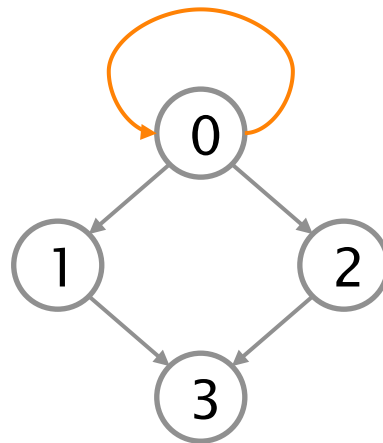
# Multigrafo

dois nós podem estar conectados por mais de uma aresta





# Digrafo com auto-aresta



# dúvidas

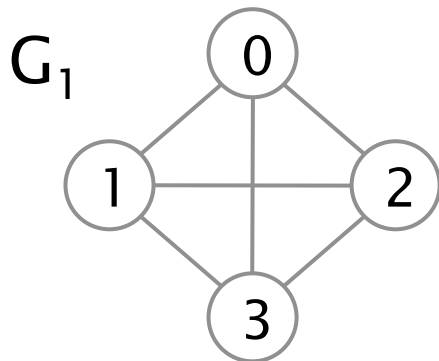
## sobre definições de grafo?

# Representações de grafo

- matriz de adjacências
- listas de adjacências (incidências)

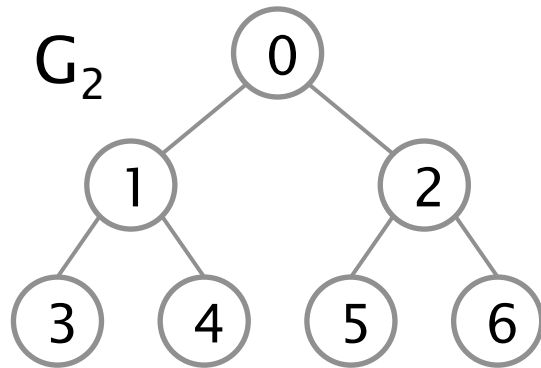
# Matriz de adjacências

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



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

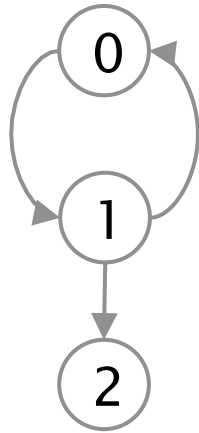
# Matriz de adjacências



	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

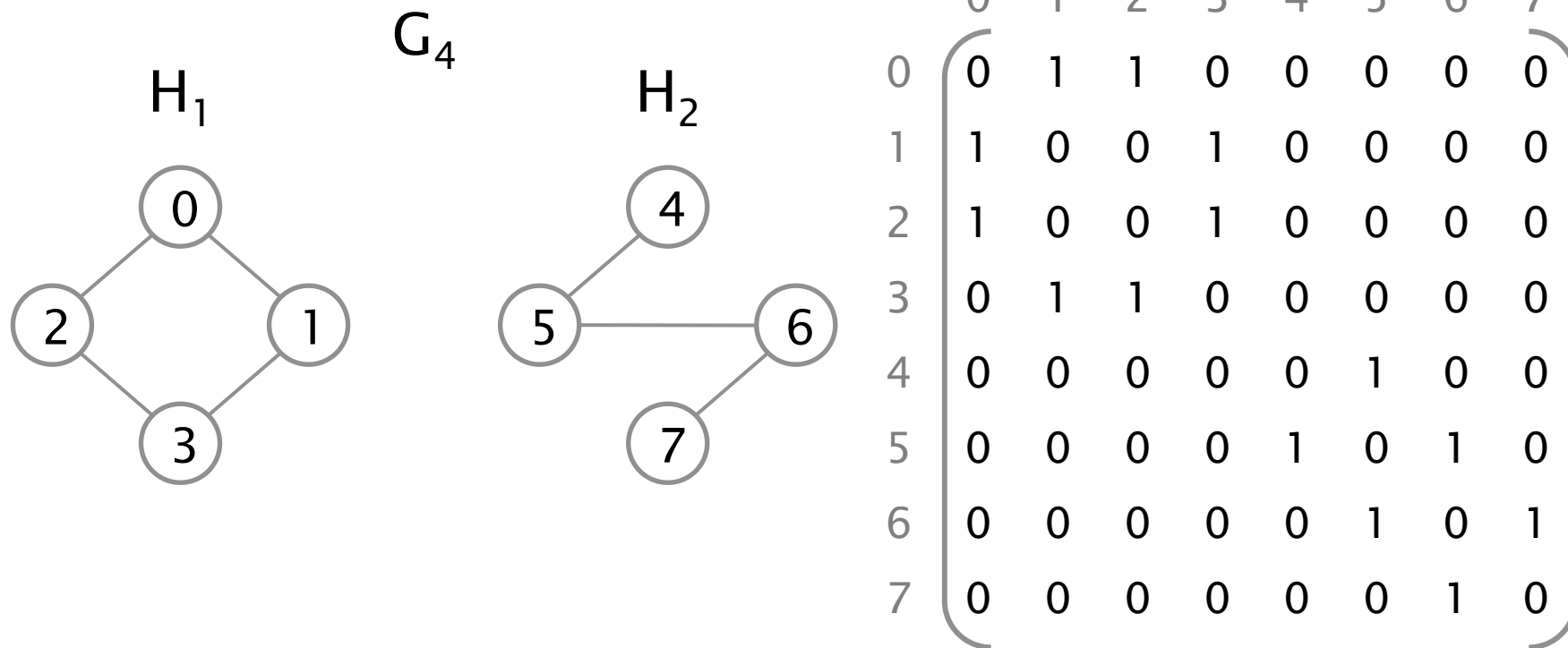
# Matriz de adjacências

$G_3$



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

# Matriz de adjacências



# Percursos em grafos

em profundidade (*depth-first search*)

arestas que partem do vértice mais recente  
(visitado por último)

em largura (*breadth-first search*)

arestas que partem do vértice mais antigo  
(visitado primeiro)

guloso (*greedy*)

arestas de menor custo, menor caminho



# Percursos em grafos

Cada vértice examinado deve ser marcado como visitado.

Por quê?

# Percurso em profundidade

## – *depth first search*

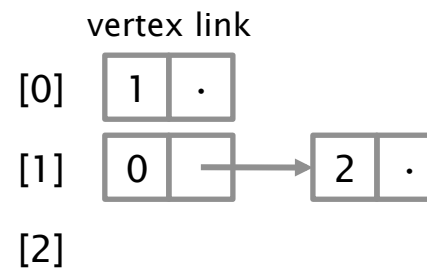
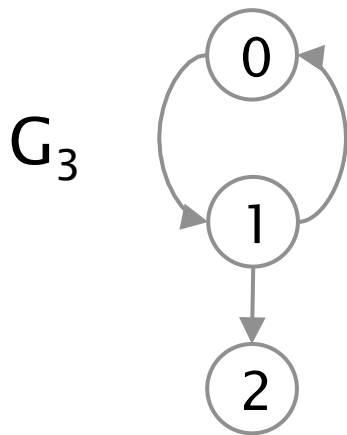
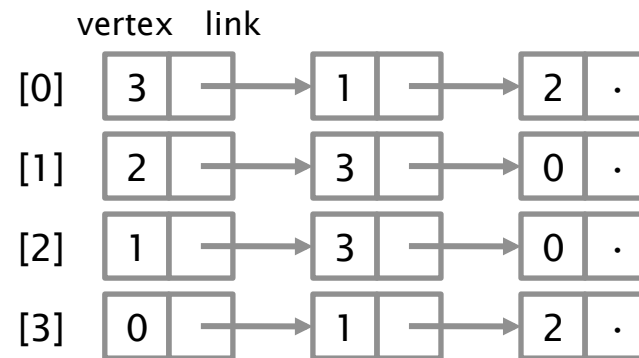
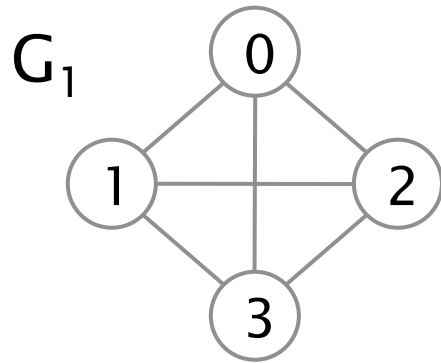
```
/* grafo como matriz de adjacências */
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

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);
}
```

# dúvidas

sobre o percurso dfs em um  
grafo representado como  
matriz de adjacências?

# Listas de adjacências



# 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];

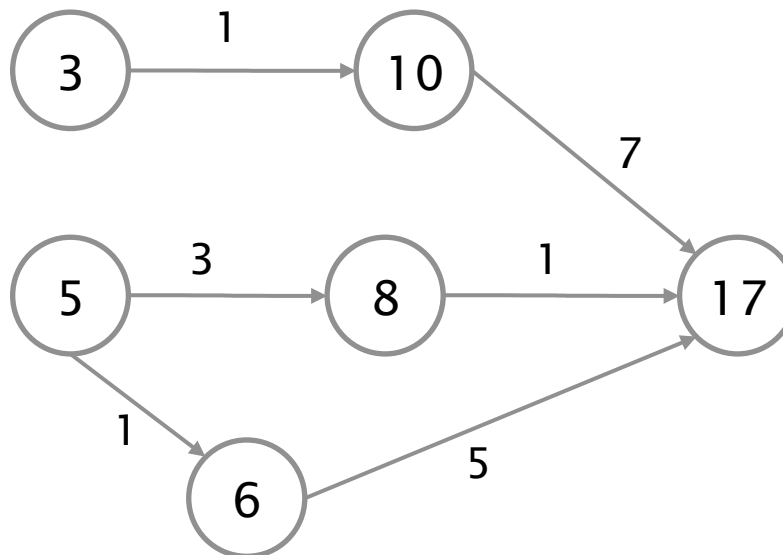
void dfs(int v){
    ListNode* w;
    visited[v] = 1;
    printf("%3d", v);
    for (w = graph[v]; w != NULL; w = w->link)
        if (!visited[w->vertex]) dfs(w->vertex);
}
```

## Percurso em largura – *breadth first search*

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

# Grafo ponderado

cada aresta possui um peso (*weight*) ou custo (*cost*)



# Menor caminho entre um nó de origem e um de destino – algoritmo de Dijkstra

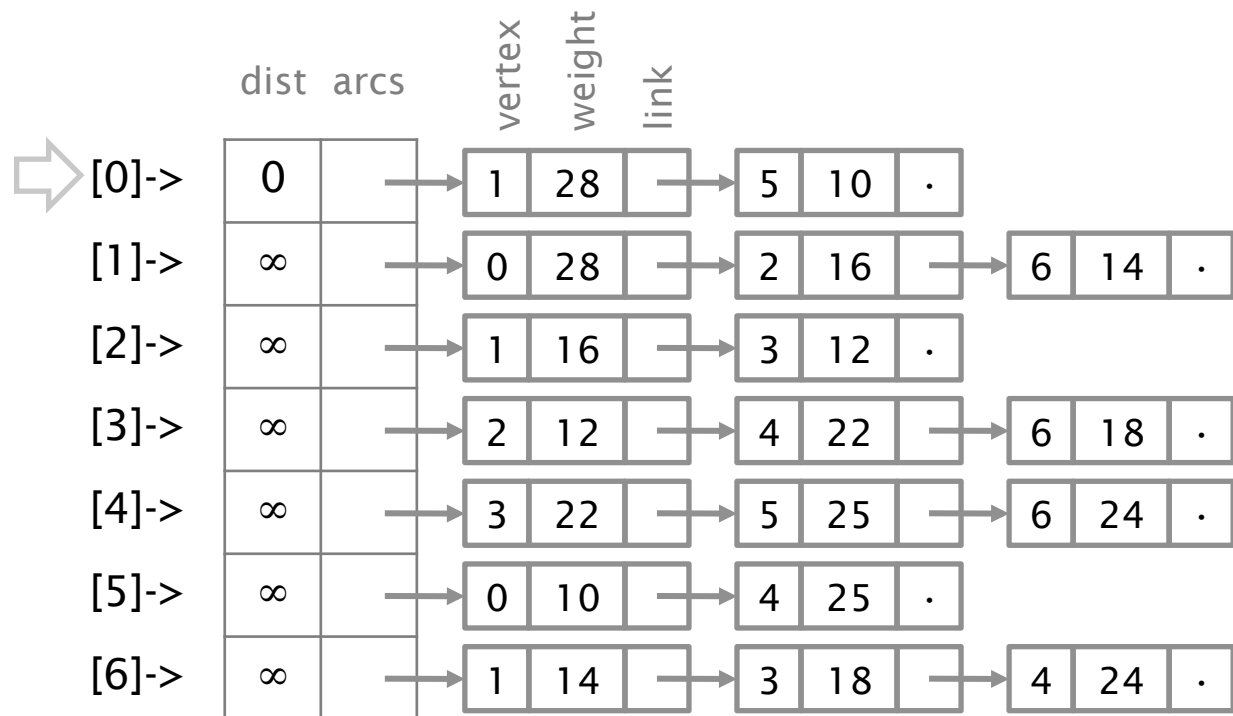
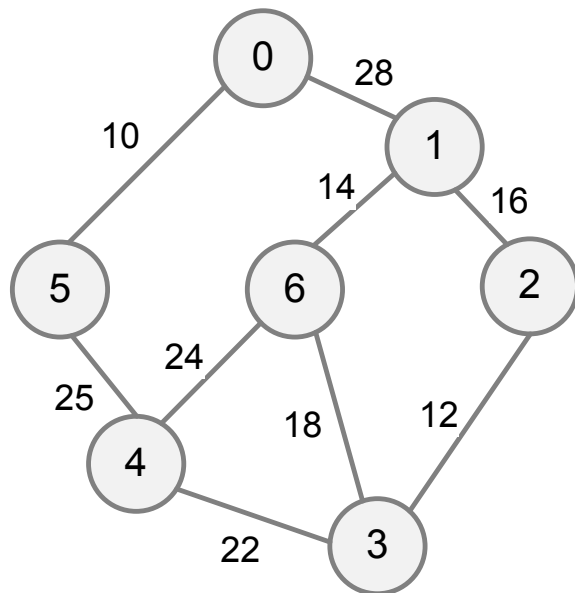
1. Defina o nó de origem
2. Atribua todos os nós um valor de distância ao nó de origem: valor zero ao nó de origem, e *infinito* para todos os outros nós.
3. Marque todos os demais nós como não visitados e o nó origem como corrente (A).
4. Considere a distância de todos os nós vizinhos não visitados ao nó corrente e calcule uma distância deles ao nó origem através do nó corrente. Por exemplo, se o nó atual A tiver distância 6, e houver uma aresta de peso 2 conectando-o com um outro nó B, a distância de B através de A será 8. Se essa distância for menor do que a distância registrada anteriormente (infinito, na primeira rodada; zero para o nó de origem), sobrescreva a distância de B.
5. Ao terminar de considerar todos os vizinhos do nó atual A, marque-o como visitado. Um nó visitado não será mais verificado; sua distância registrada agora é final e mínima.
6. Se todos os nós tiverem sido visitados, termine. Caso contrário, marque o nó não visitado com a menor distância (do nó de origem) como o próximo "nó corrente", e repita a partir do passo 4.

Ao final, tem-se a menor distância entre o nó de origem e cada um dos nós do grafo.



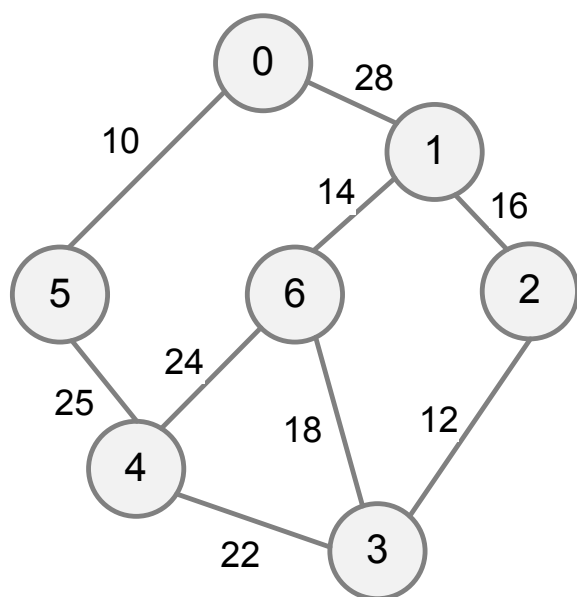
# Dijkstra - exemplo

considerando como vértice inicial 0



# Dijkstra - exemplo

modifica distâncias dos vértices 1 e 5



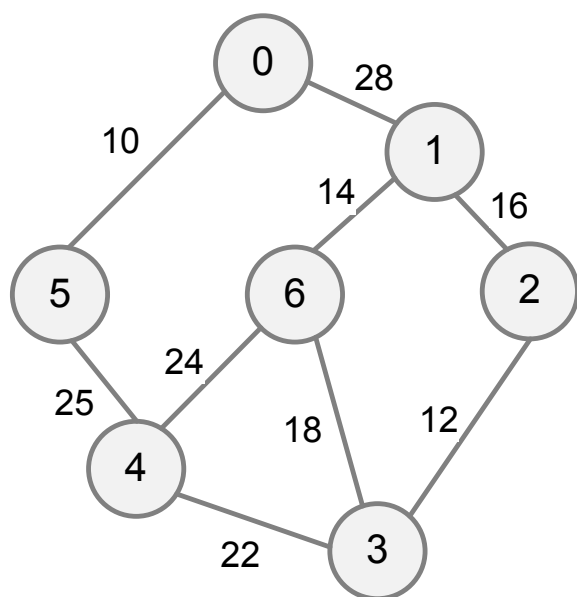
	dist	arcs	vertex	weight	link
[0]->	0		1	28	→ 5 10 .
[1]->	28		0	28	→ 2 16 → 6 14 .
[2]->	∞		1	16	→ 3 12 .
[3]->	∞		2	12	→ 4 22 → 6 18 .
[4]->	∞		3	22	→ 5 25 → 6 24 .
[5]->	10		0	10	→ 4 25 .
[6]->	∞		1	14	→ 3 18 → 4 24 .

# Dijkstra - exemplo

marca o vértice 0 como visitado

seleciona o vértice 5 (o de menor distância)

ignora vértice 0 (já visitado); modifica distância do vértice 4



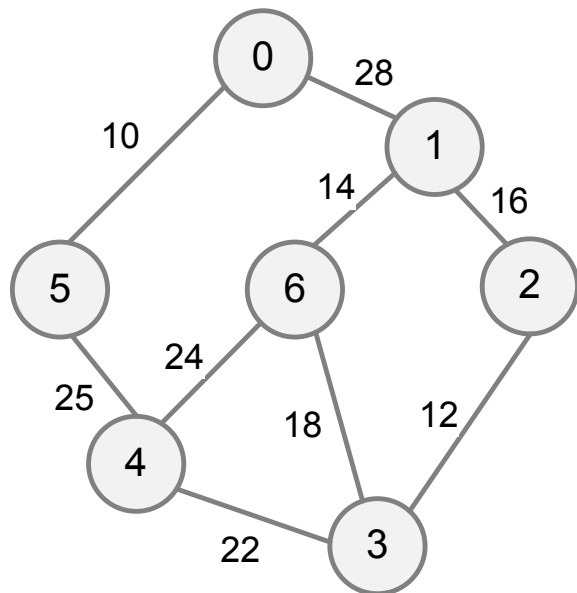
	dist	arcs	vertex	weight	link
[0]->	0		1	28	→ 5 10 .
[1]->	28		0	28	→ 2 16 → 6 14 .
[2]->	∞		1	16	→ 3 12 .
[3]->	∞		2	12	→ 4 22 → 6 18 .
[4]->	35		3	22	→ 5 25 → 6 24 .
[5]->	10		0	10	→ 4 25 .
[6]->	∞		1	14	→ 3 18 → 4 24 .

# Dijkstra - exemplo

marca o vértice 5 como visitado

seleciona o vértice 1 (o de menor distância)

ignora vértice 0; modifica distâncias dos vértices 2 e 6



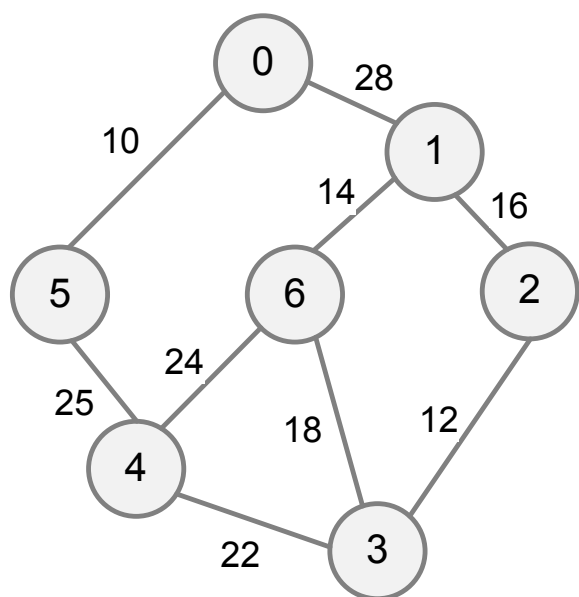
	dist	arcs	vertex	weight	link
[0]->	0		1	28	→ 5 10 .
[1]->	28		0	28	→ 2 16 → 6 14 .
[2]->	44		1	16	→ 3 12 .
[3]->	∞		2	12	→ 4 22 → 6 18 .
[4]->	35		3	22	→ 5 25 → 6 24 .
[5]->	10		0	10	→ 4 25 .
[6]->	42		1	14	→ 3 18 → 4 24 .

# Dijkstra - exemplo

marca o vértice 1 como visitado

seleciona o vértice 4 (o de menor distância)

modifica distância do vértice 3; ignora vértice 5;  
mantém a distância do vértice 6



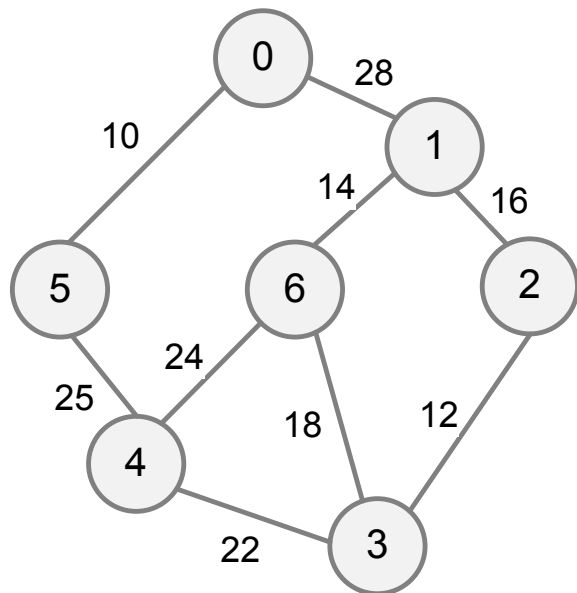
	dist	arcs	vertex	weight	link
[0]->	0		1	28	→ 5 10 .
[1]->	28		0	28	→ 2 16 → 6 14 .
[2]->	44		1	16	→ 3 12 .
[3]->	57		2	12	→ 4 22 → 6 18 .
[4]->	35		3	22	→ 5 25 → 6 24 .
[5]->	10		0	10	→ 4 25 .
[6]->	42		1	14	→ 3 18 → 4 24 .

# Dijkstra - exemplo

marca o vértice 4 como visitado

seleciona o vértice 6 (o de menor distância)

ignora vértice 1; mantém distância do vértice 3; ignora vértice 4



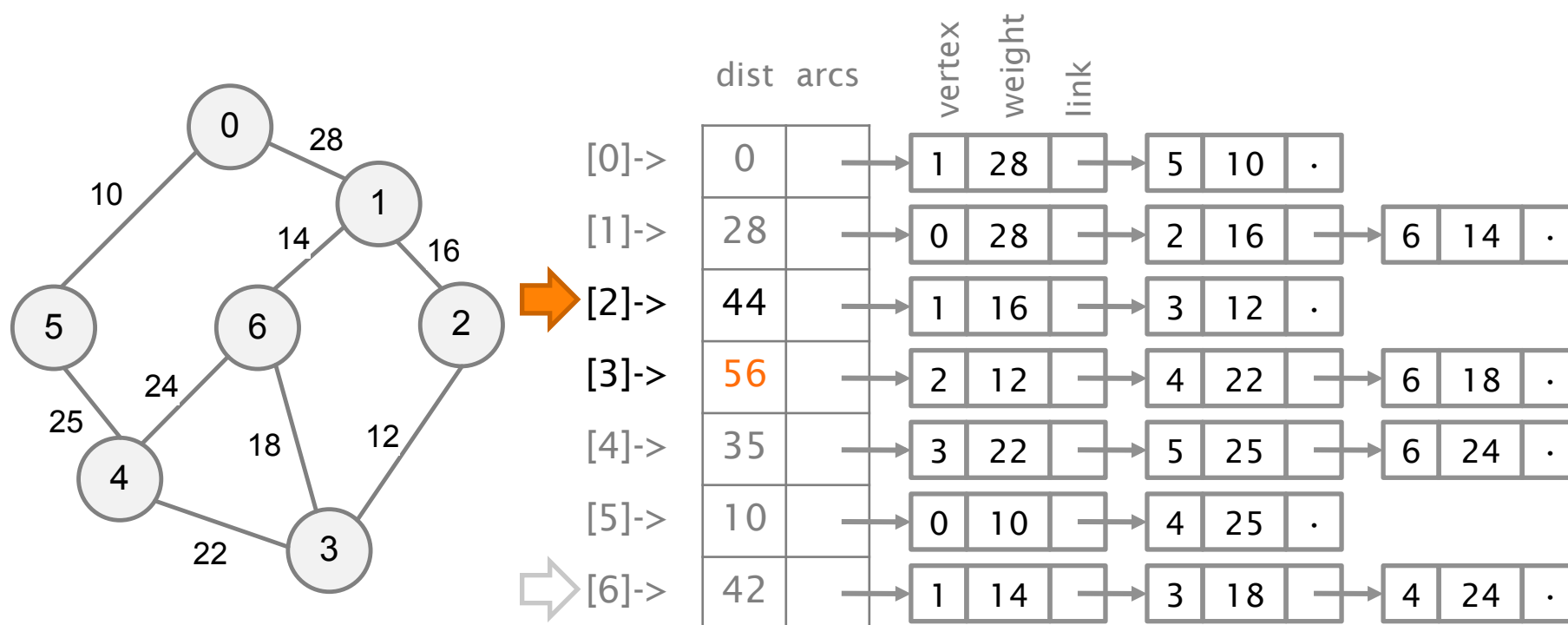
	dist	arcs	vertex	weight	link
[0]->	0		1	28	→ 5 10 .
[1]->	28		0	28	→ 2 16 → 6 14 .
[2]->	44		1	16	→ 3 12 .
[3]->	57		2	12	→ 4 22 → 6 18 .
[4]->	35		3	22	→ 5 25 → 6 24 .
[5]->	10		0	10	→ 4 25 .
[6]->	42		1	14	→ 3 18 → 4 24 .

# Dijkstra - exemplo

marca o vértice 6 como visitado

seleciona o vértice 2 (o de menor distância)

ignora vértice 1; modifica distância do vértice 3

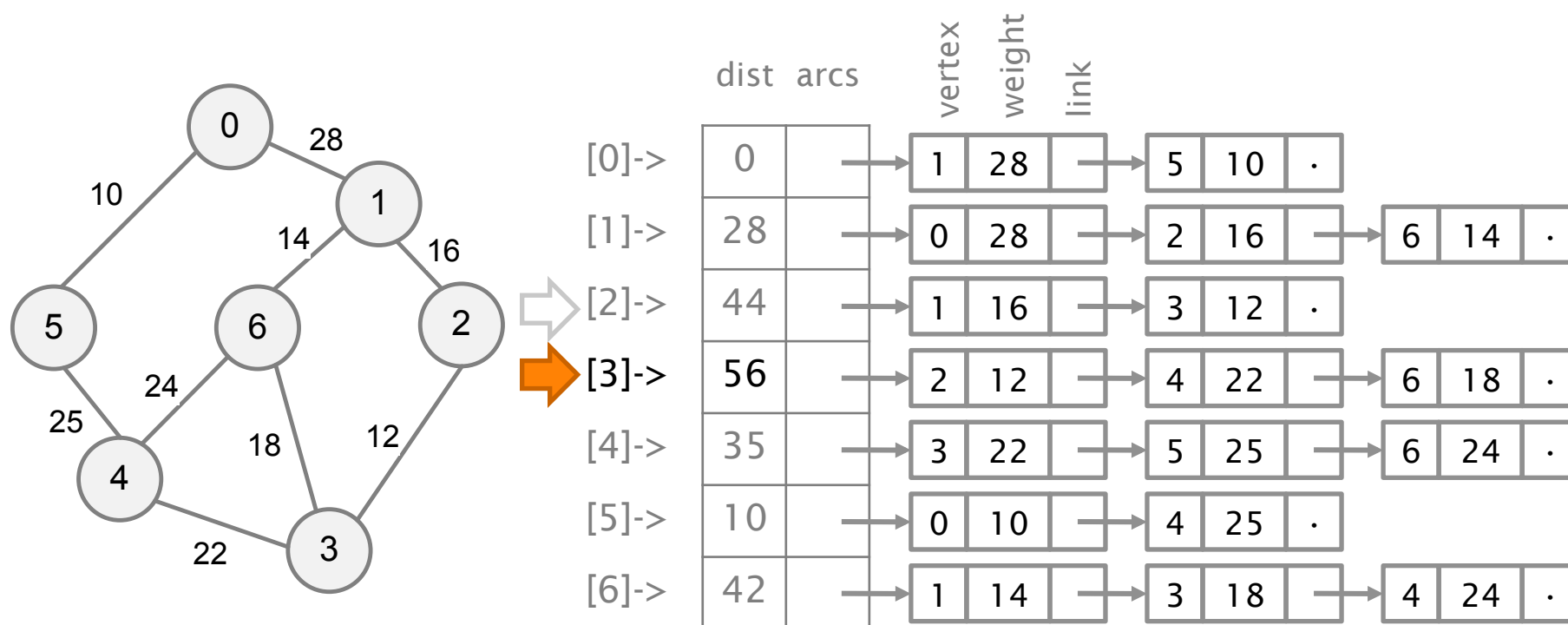


# Dijkstra - exemplo

marca o vértice 2 como visitado

seleciona o vértice 3 (o de menor distância)

ignora vértices 2, 4 e 6

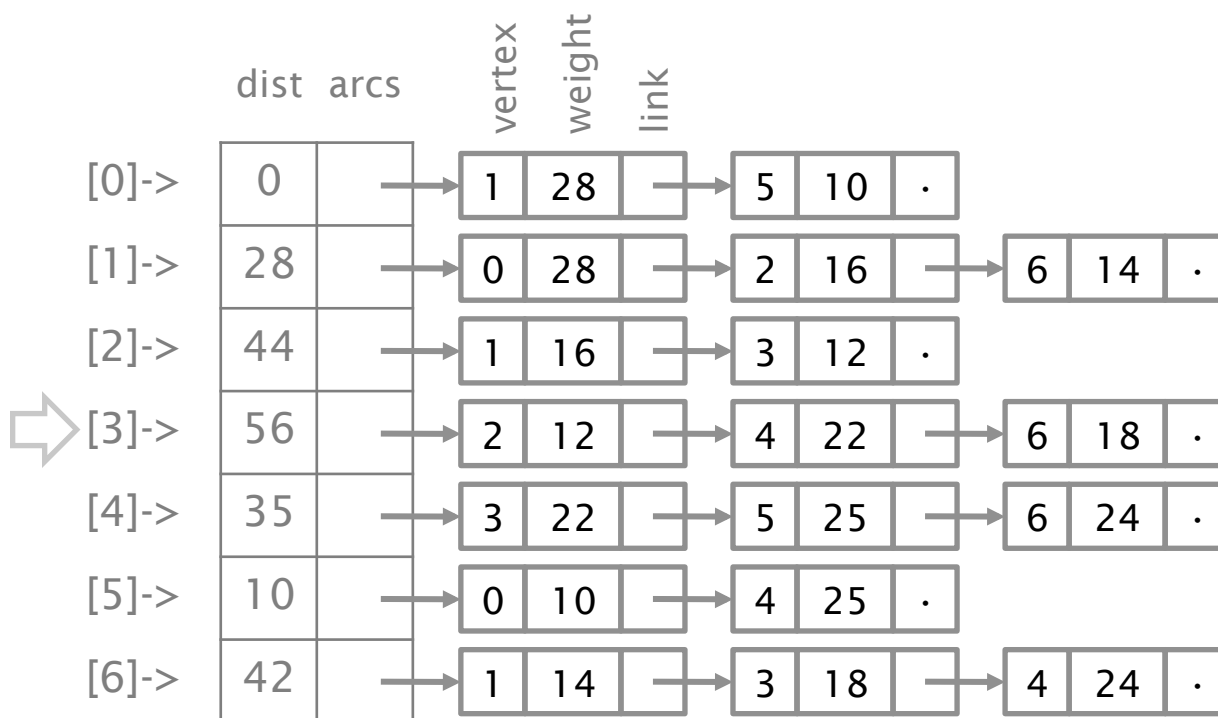
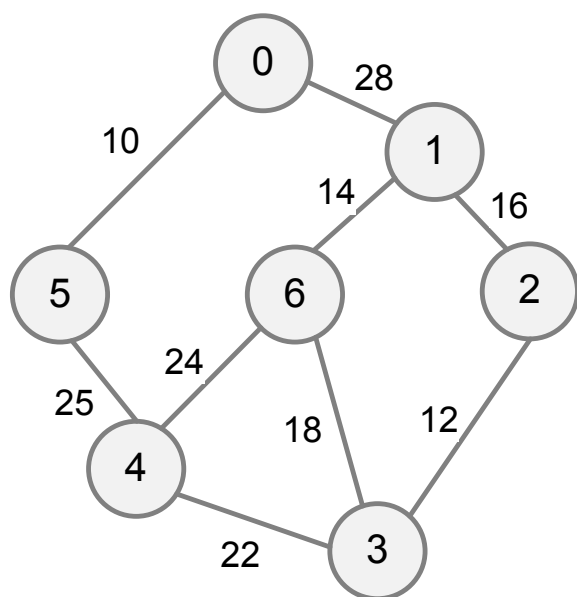




# Dijkstra - exemplo

marca o vértice 3 como visitado

não há mais vértices não visitados – FIM!



# dúvidas

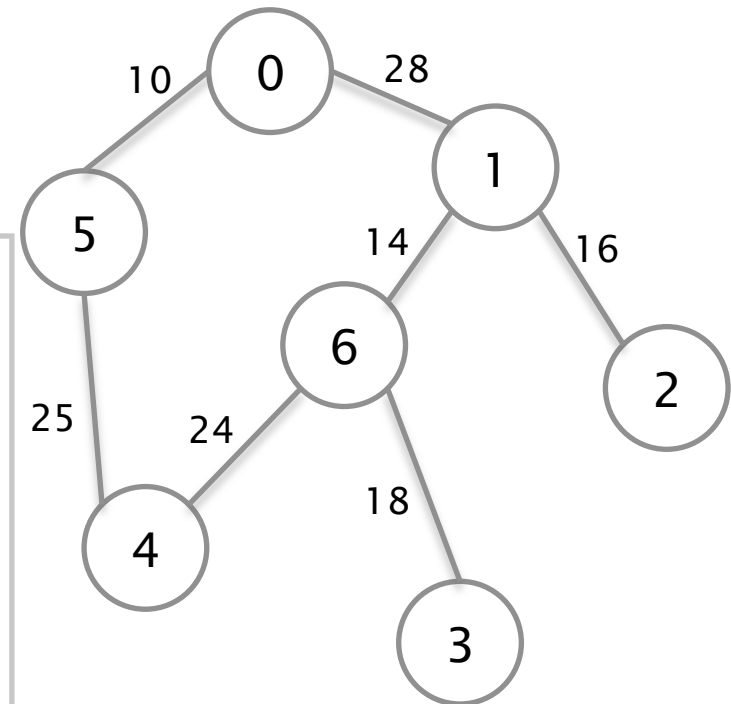
## sobre o algoritmo de Dijkstra?

# 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);
```

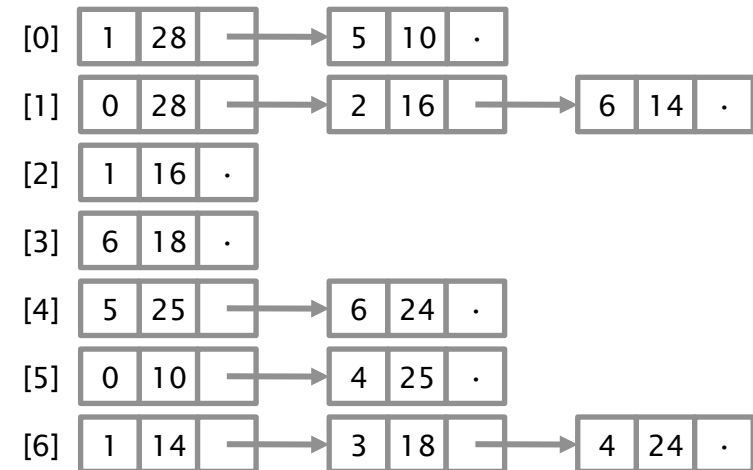
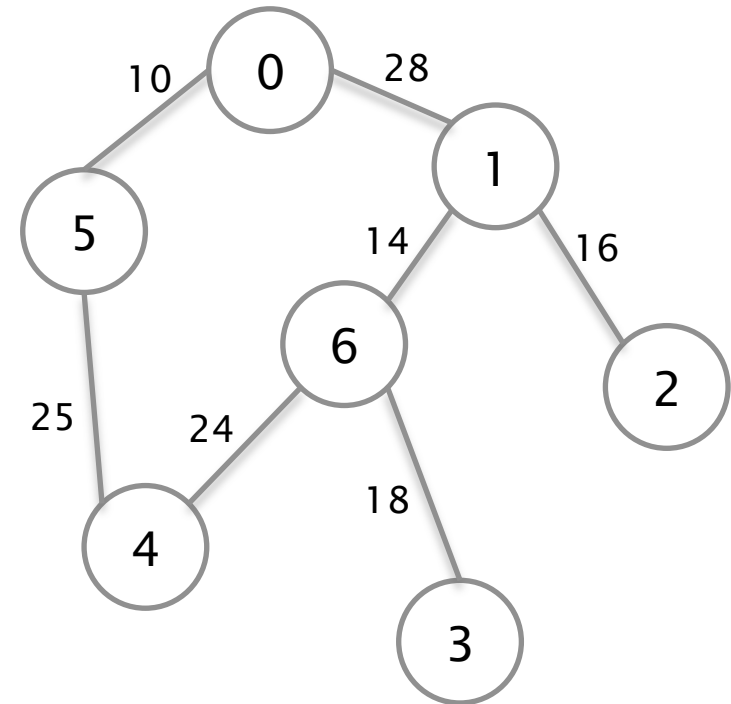
# Exemplo

```
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;
}
```

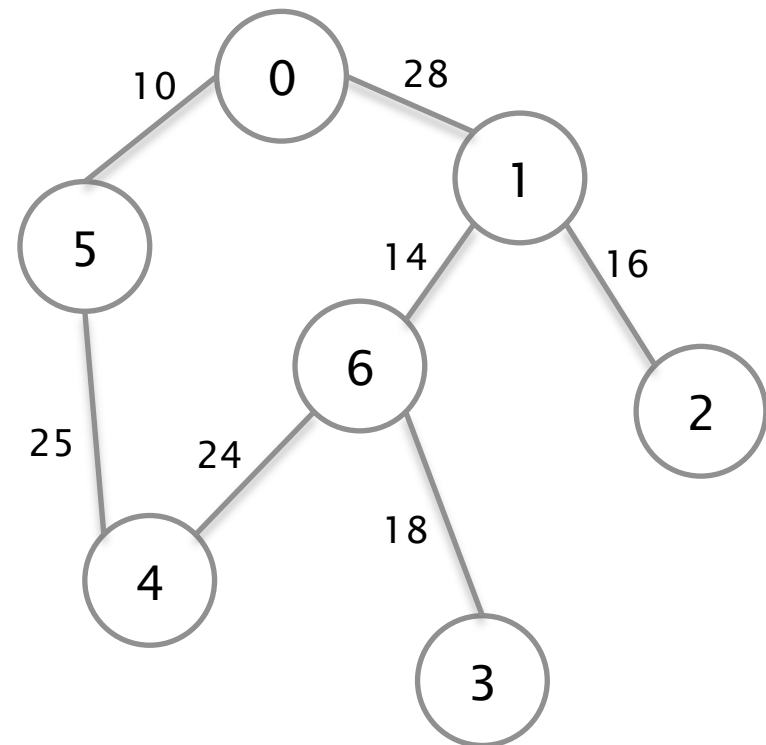


# **dfs**

## **(depth-first search)**

# dfs iniciando em 0

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



# **bfs**

## **(breadth-first search)**



# 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)

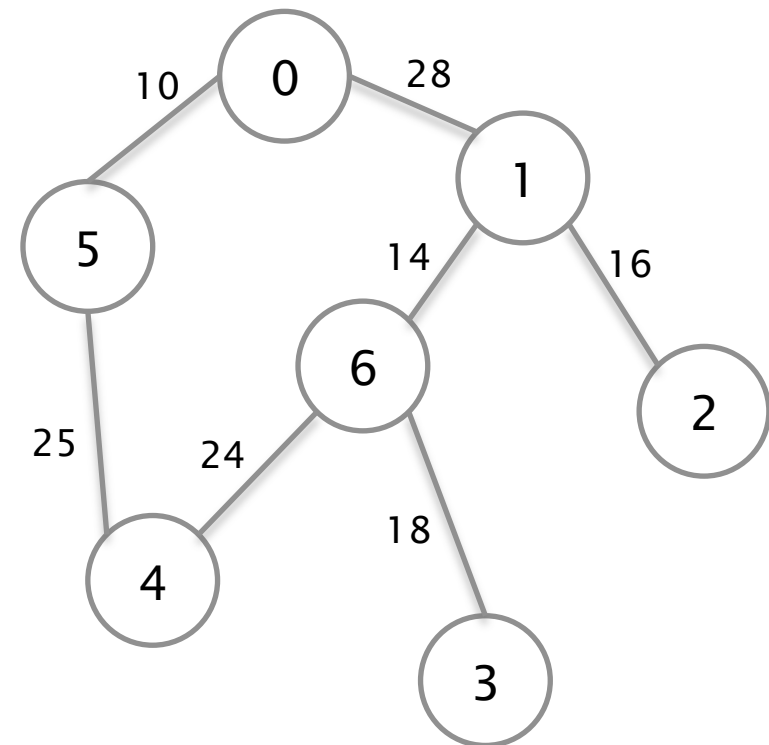
-> enfileira 3      [4,3]

bfs(4)

[3]

bfs(3)

[]

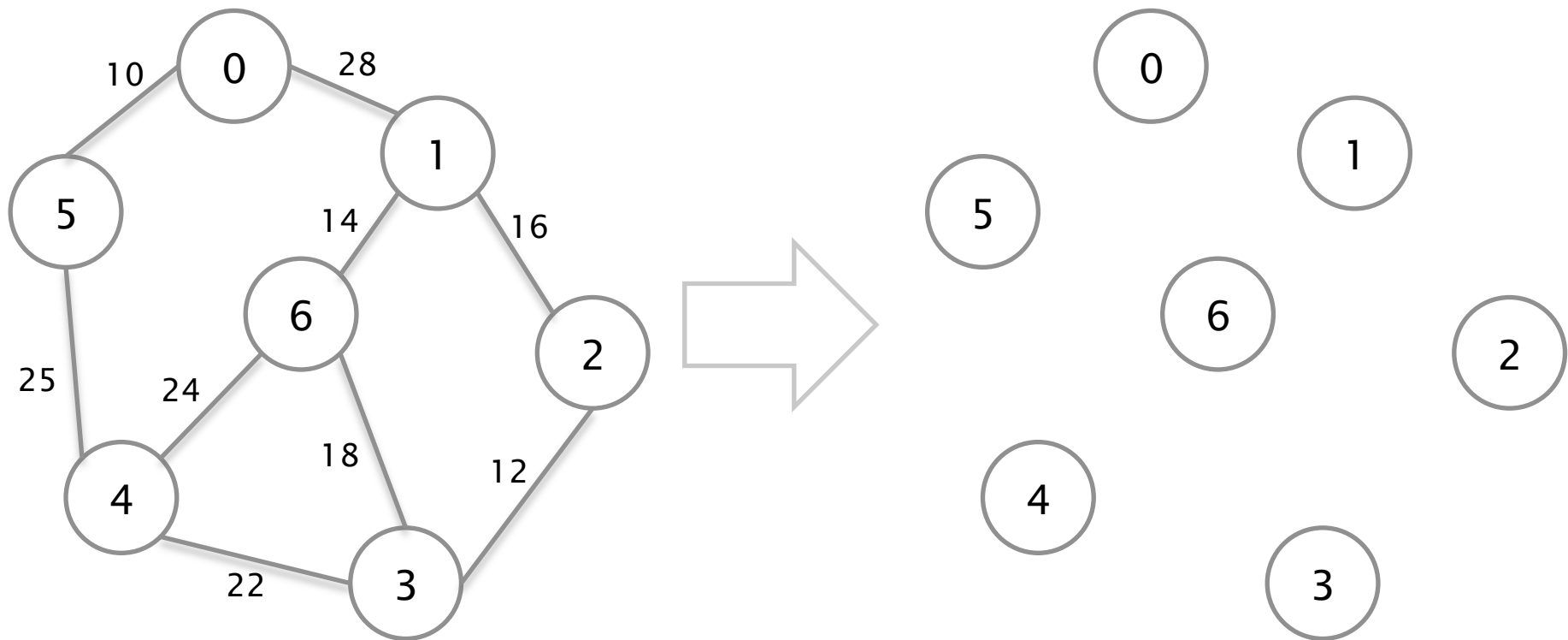


# dúvidas?

# árvore geradora

# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

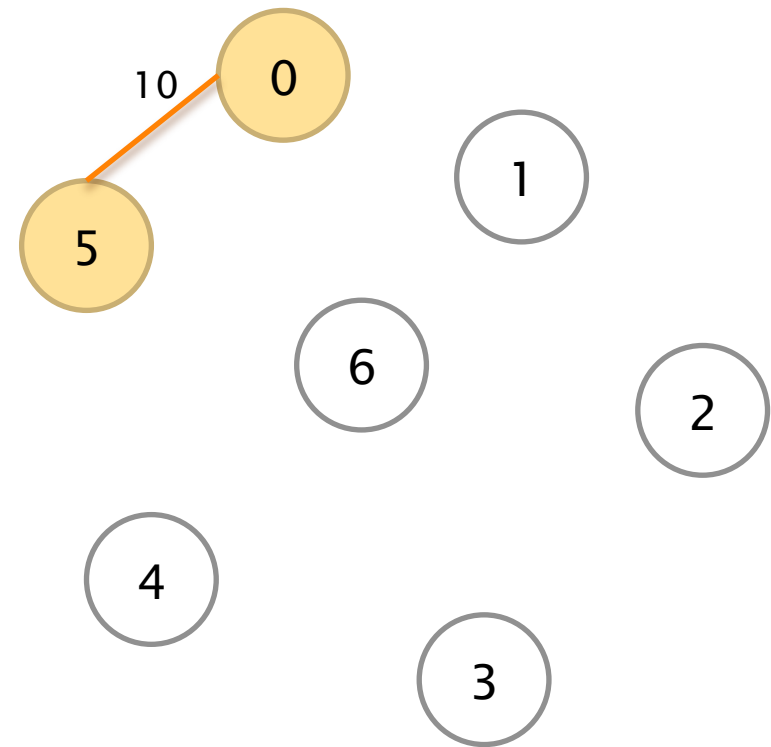
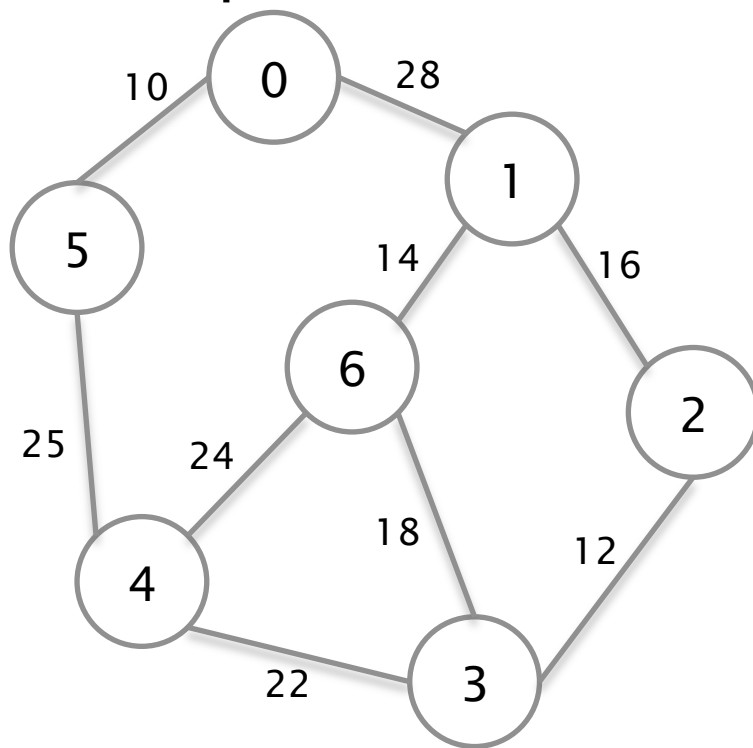
1. Considere cada nó uma árvore (formando uma floresta)



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

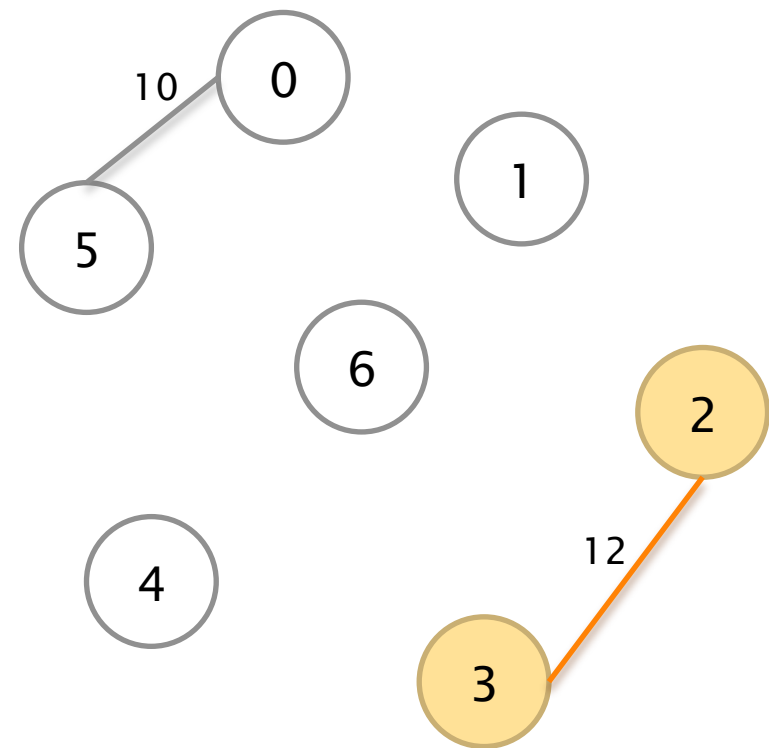
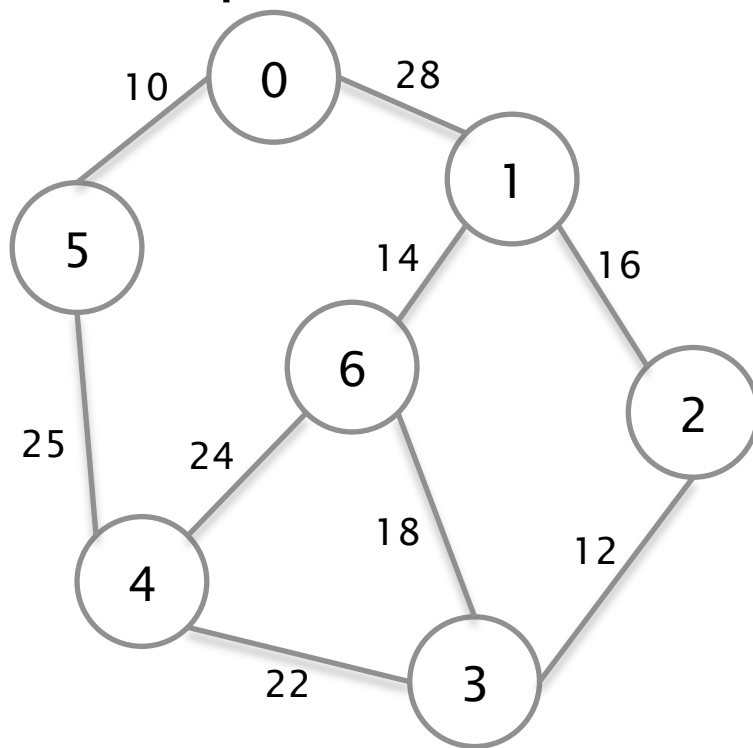
3. Repita até todos os nós estarem conectados.



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

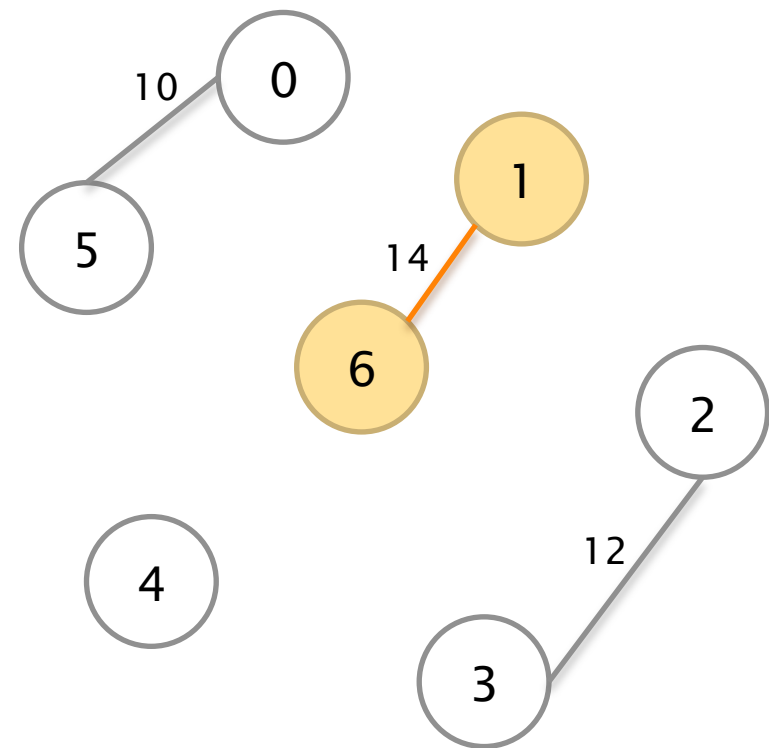
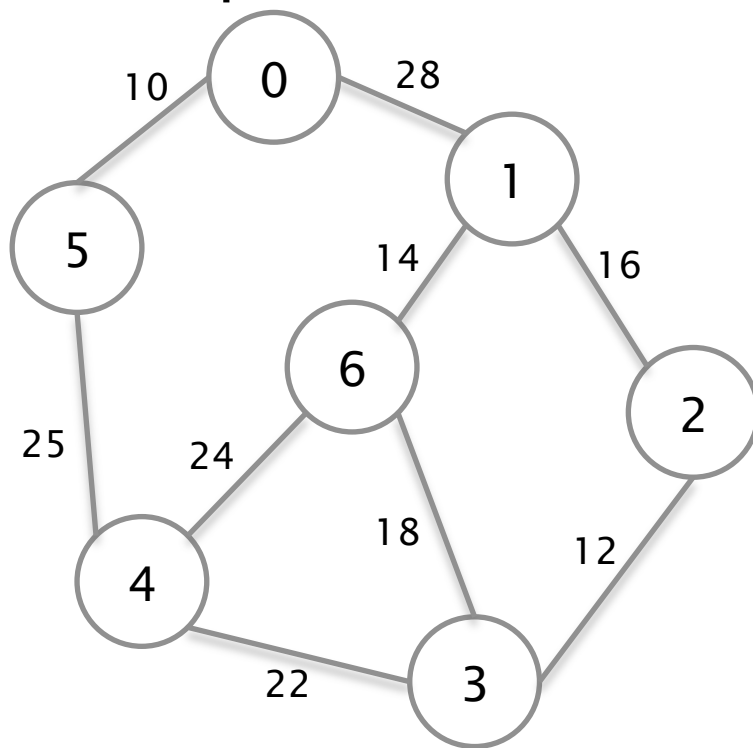
3. Repita até todos os nós estarem conectados.



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

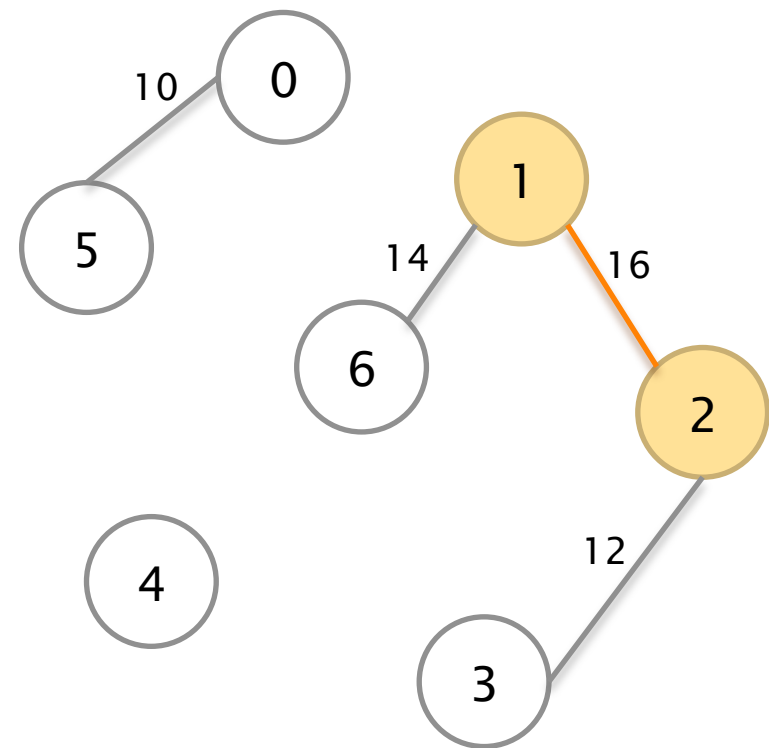
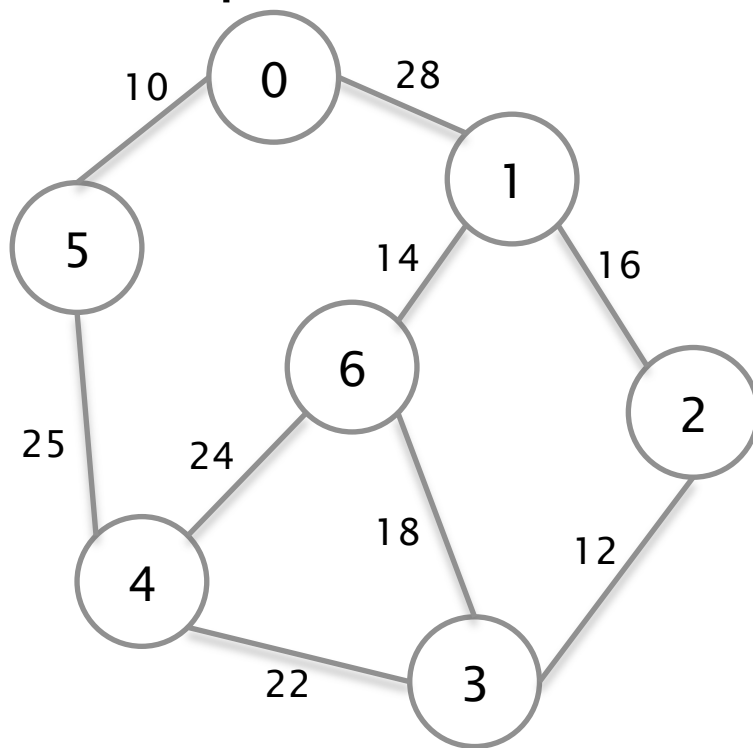
3. Repita até todos os nós estarem conectados.



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

3. Repita até todos os nós estarem conectados.

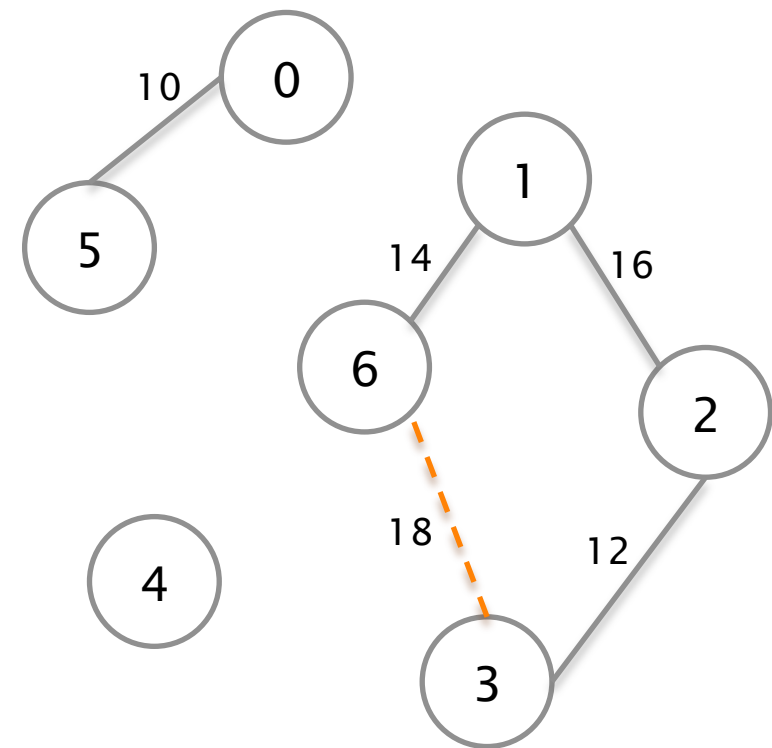
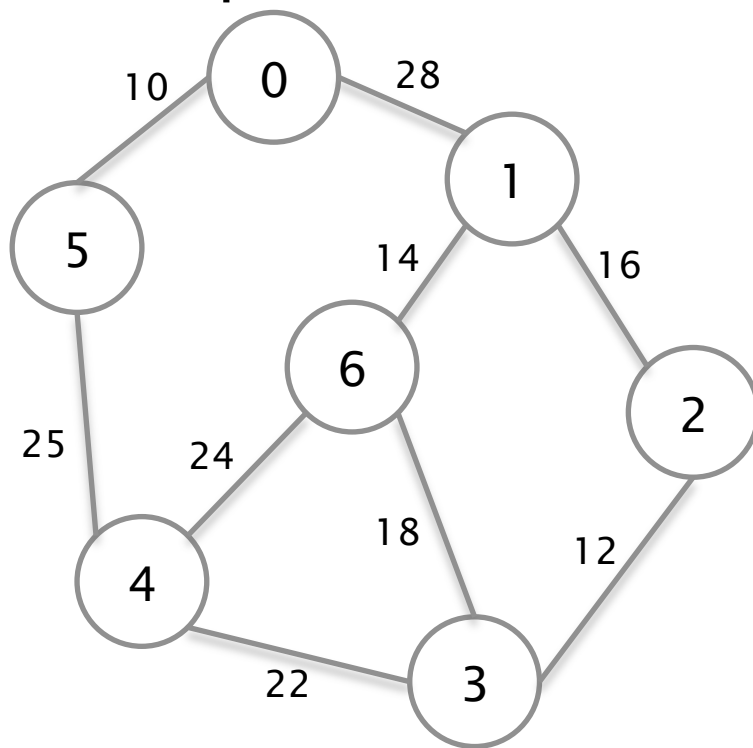




# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

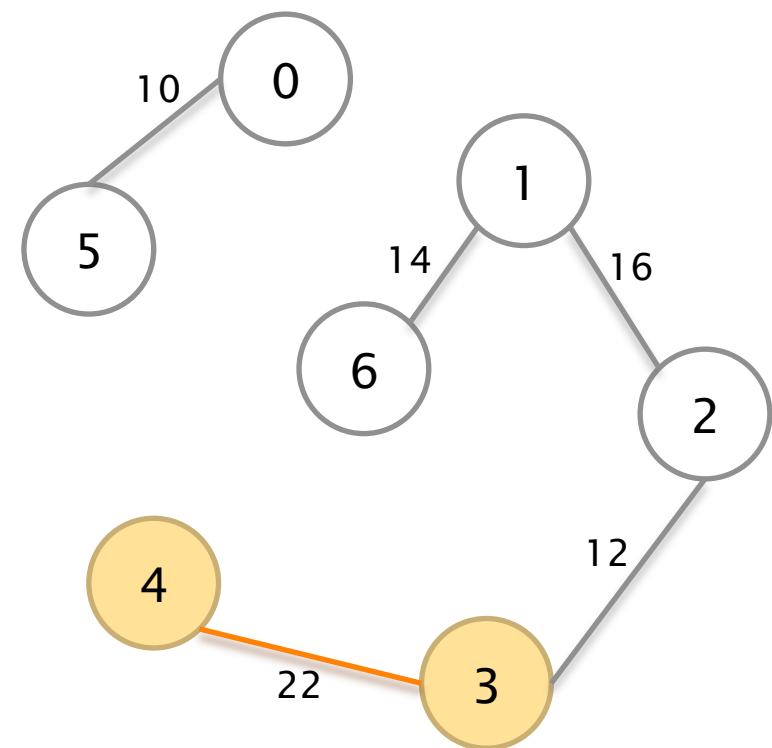
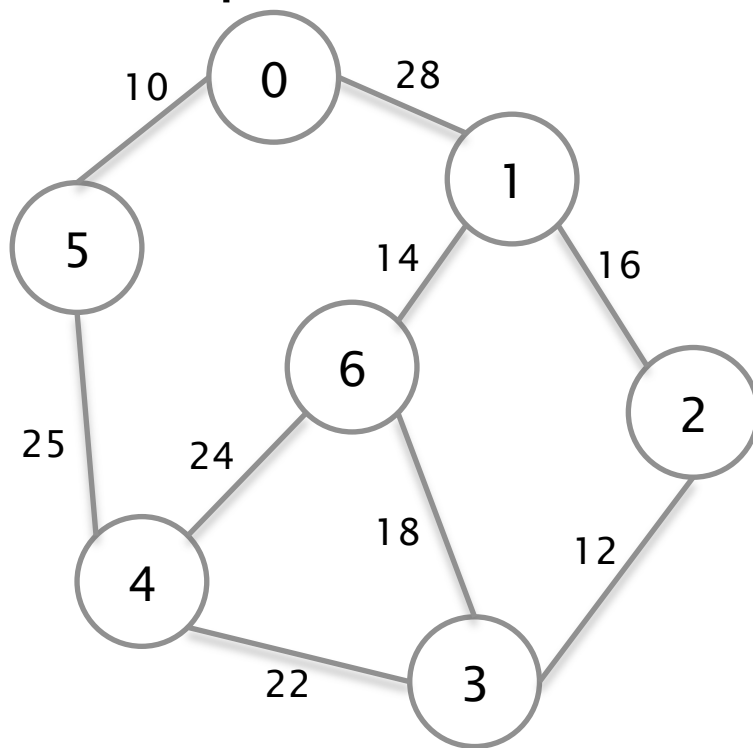
3. Repita até todos os nós estarem conectados.



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

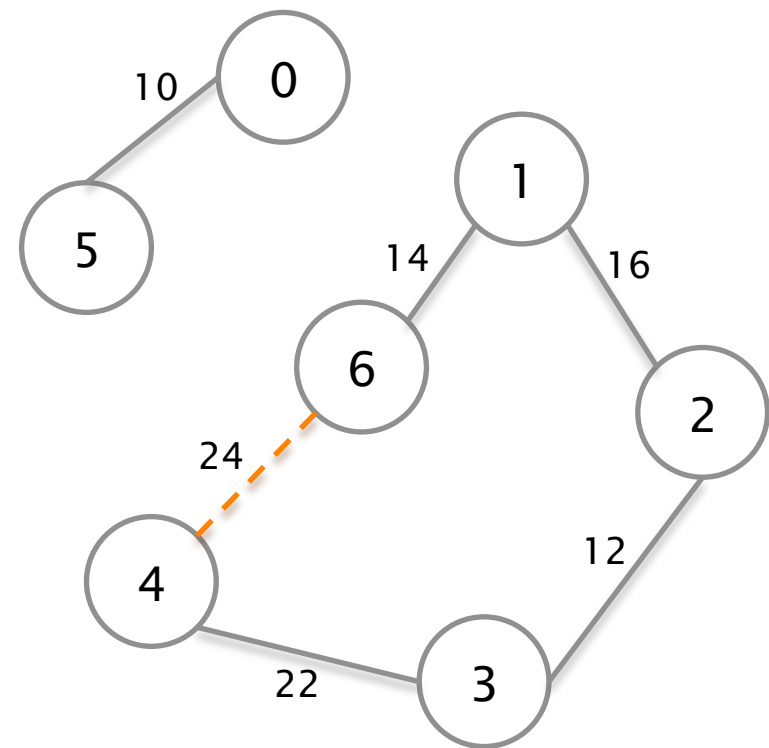
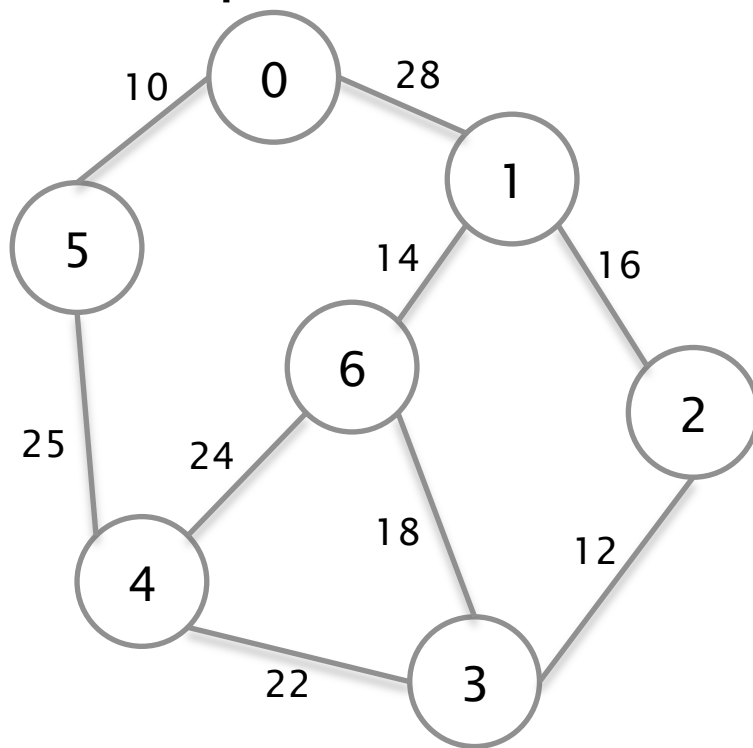
3. Repita até todos os nós estarem conectados.



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

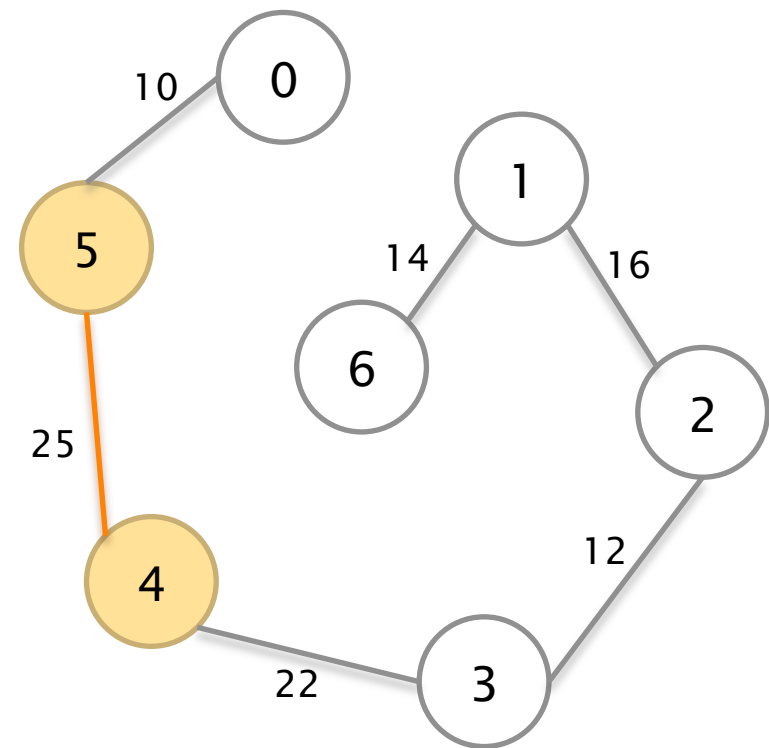
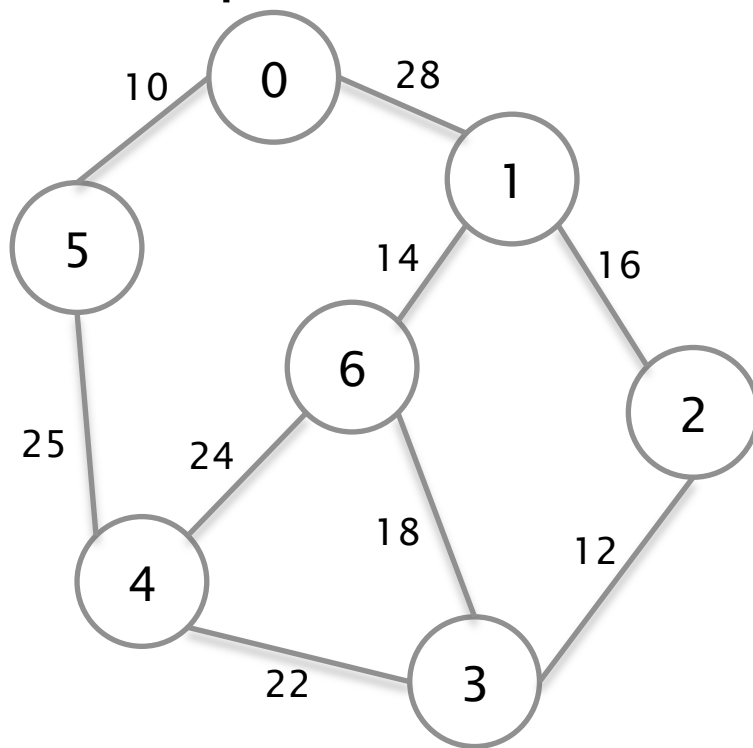
3. Repita até todos os nós estarem conectados.



# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.

3. Repita até todos os nós estarem conectados.



# dúvidas

## sobre o algoritmo de Kruskal?