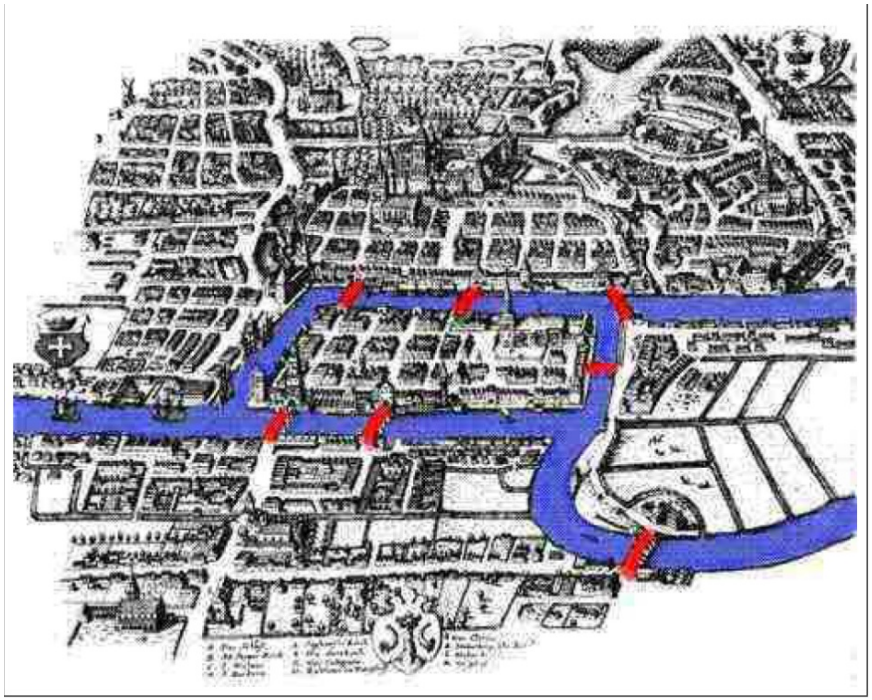

# INF 1010
# Estruturas de Dados Avançadas

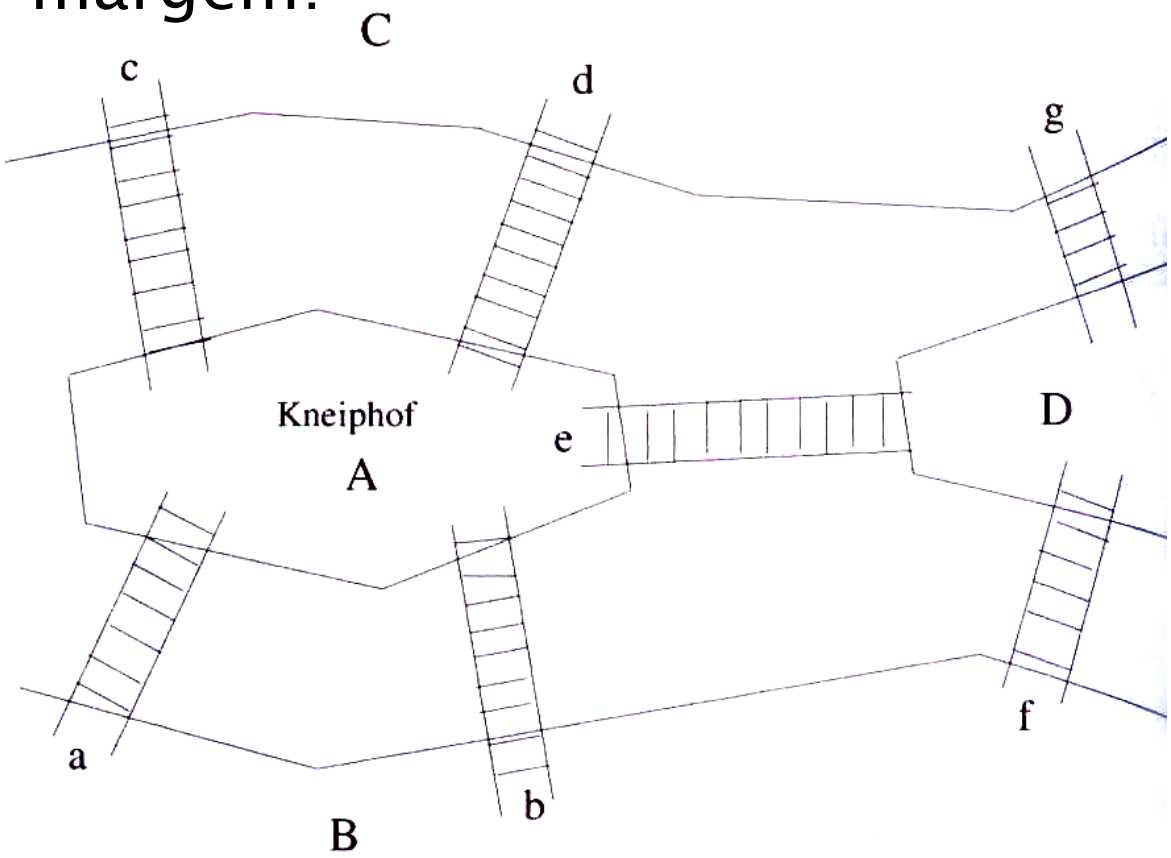
## Grafos

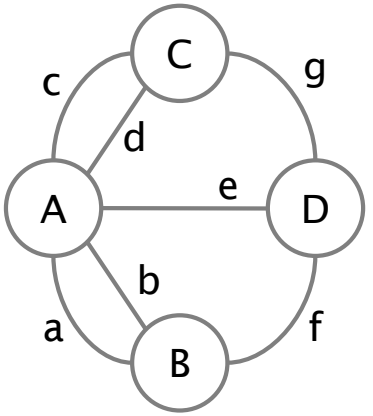
# Primeiro uso conhecido
# 1736 Euler: pontes de Königsberg



# 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?



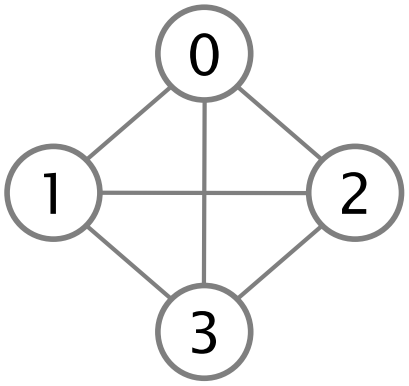



# Definição

G = (V,E)

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

exemplos



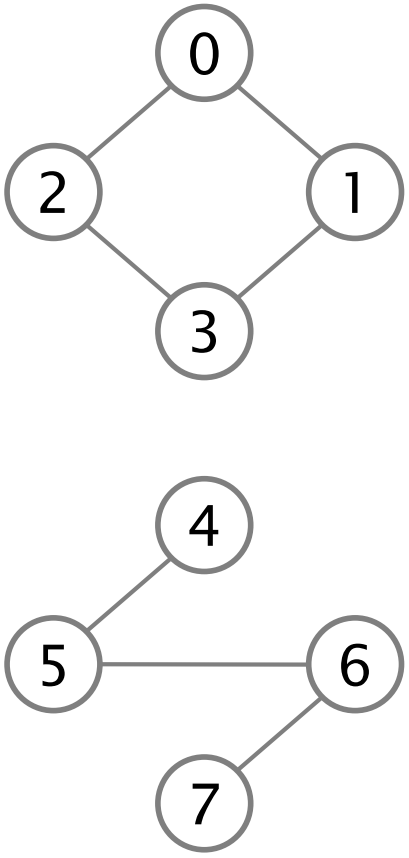

vértices: V = {0,1,2,3}
arestas: E = {(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)}
n = 4
m = 6

# 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



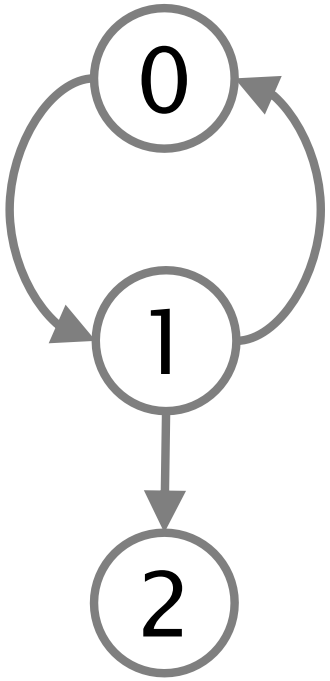

0 e 1
0 e 2
1 e 3
2 e 3
4 e 5
5 e 6
6 e 7

# Digrafo: grafo direcionado (orientado)

vértices: V = {0,1,2}

arestas (pares ordenados):

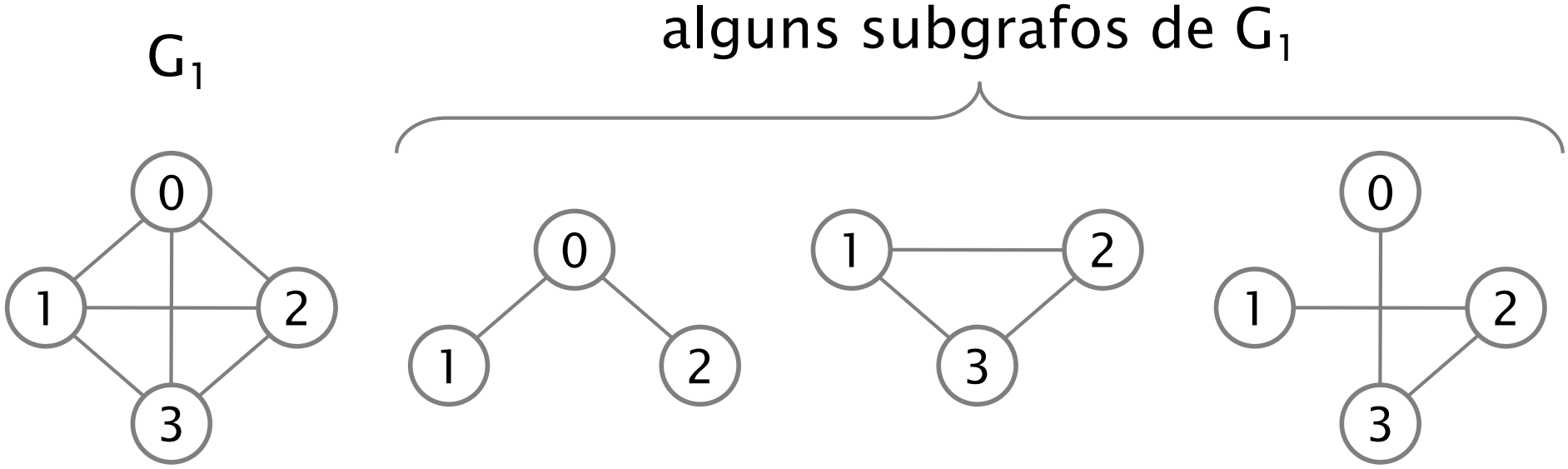
E = {<0,1>,<1,0>,<1,2>}



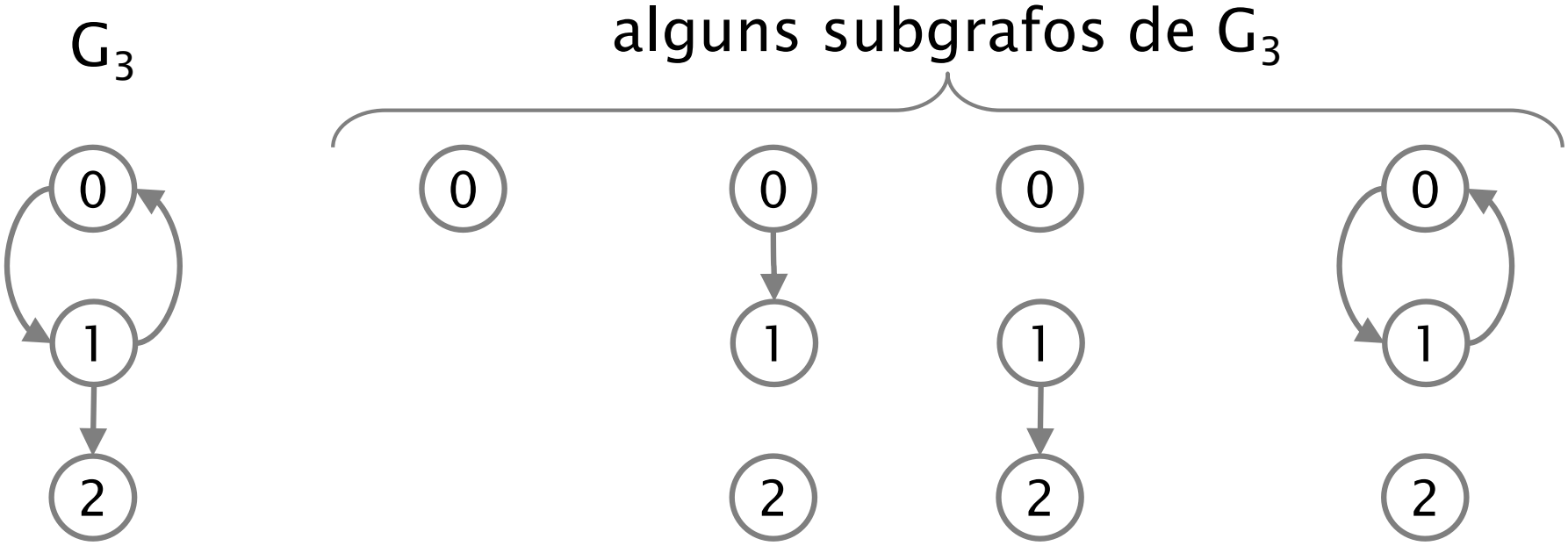

Qual é o número máximo de arestas em um grafo direcionado de *n* vértices?

n (n-1)

# Subgrafo

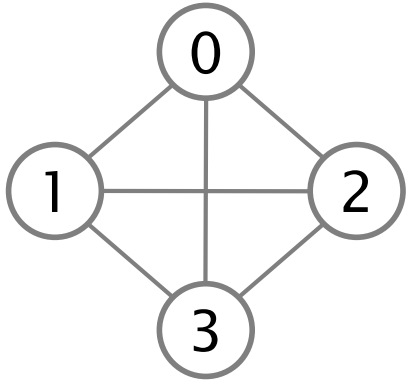


# Subgrafo



# Grafo completo

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



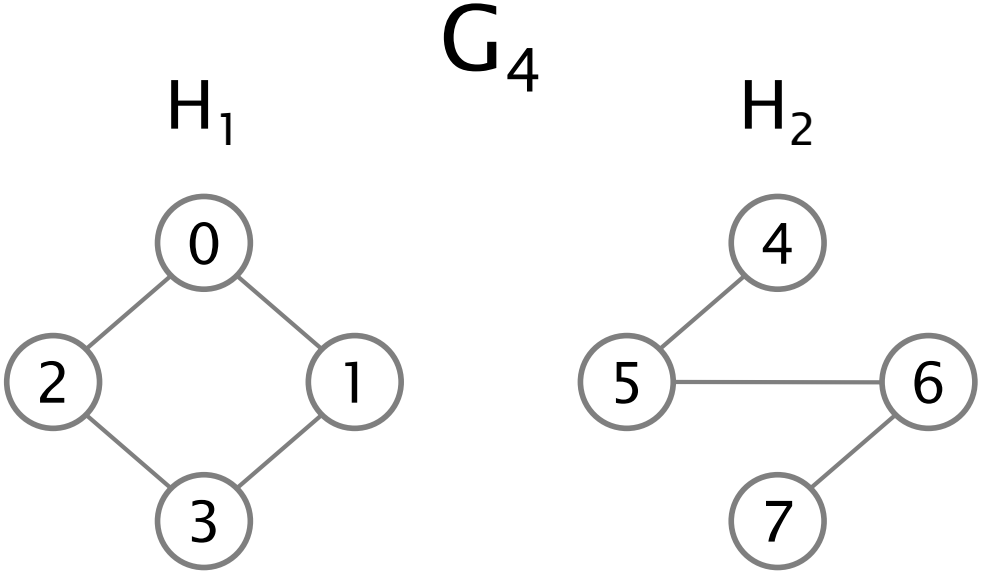

Quantas arestas há em um grafo completo de n vértices? n (n-1)/2

# Grafo conectado

existe um caminho entre quaisquer dois vértices

exemplo

grafo com 2 componentes conectados



# Grau

número de arestas incidentes em um vértice

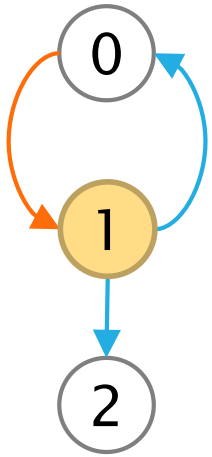
exemplo:

grau do vértice (1): 3

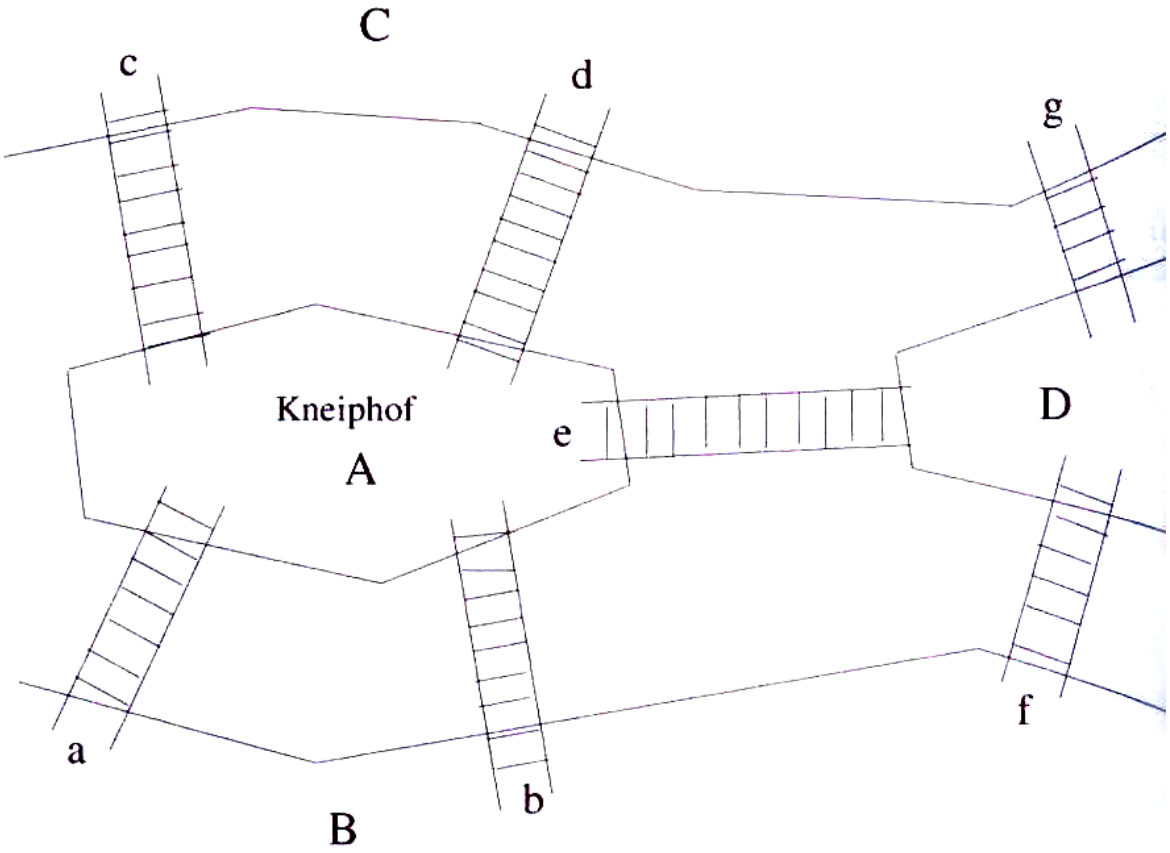
grau de entrada do vértice (1): 1

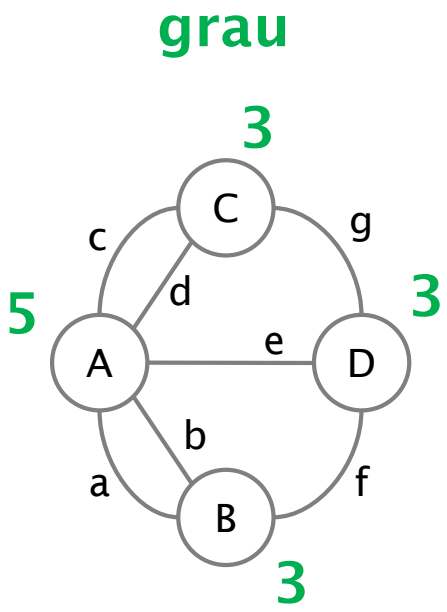
grau de saída do vértice (1): 2

$G_3$



# Voltando ao problema das pontes de Königsberg



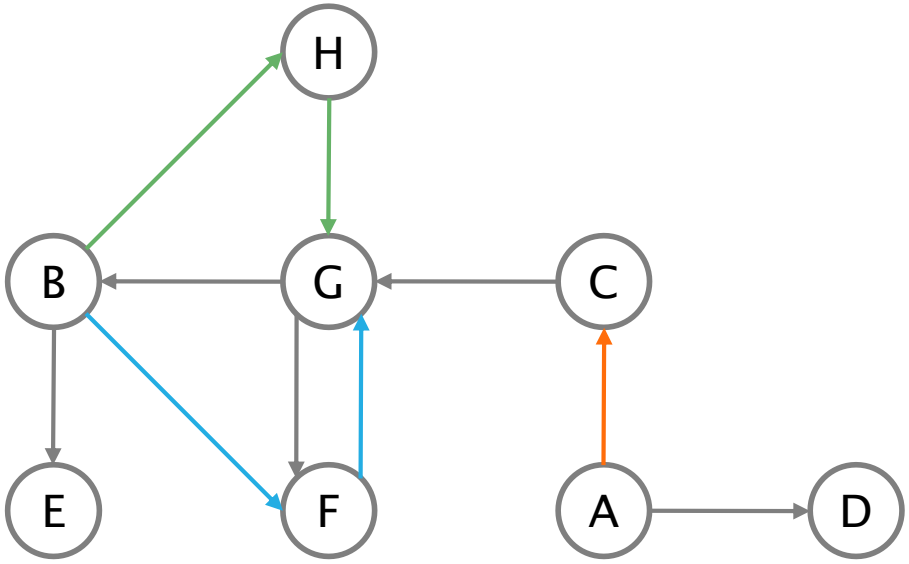




**seria possível somente se o grau de todos os nós fosse par (entrada e saída).**

# 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

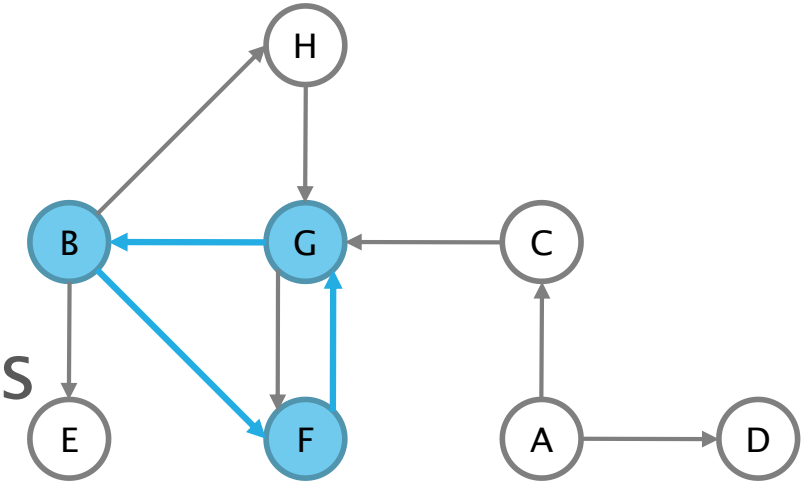
exemplo: B-F-G-B

grafo cíclico

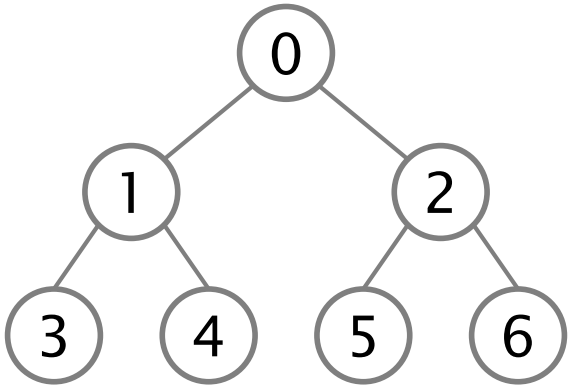
contém um ou mais ciclos

grafo acíclico

não contém ciclos



# Árvores



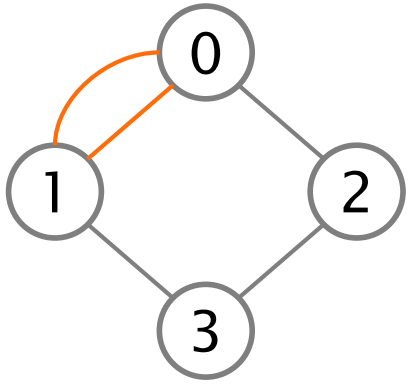

vértices : V = {0,1,2,3,4,5,6}
arestas:
E = {(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)}
n = 7
m = 6

**São um caso particular de grafos acíclicos.**

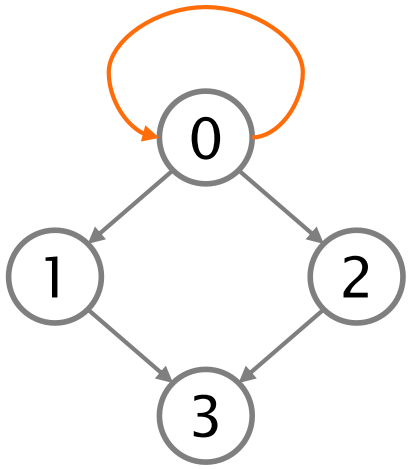
**Além de acíclicos cada nó tem um só pai, menos a raiz (que não tem pai).**

# Multigrafo

dois nós podem estar conectados por mais de uma aresta



# Digrafo com auto-aresta



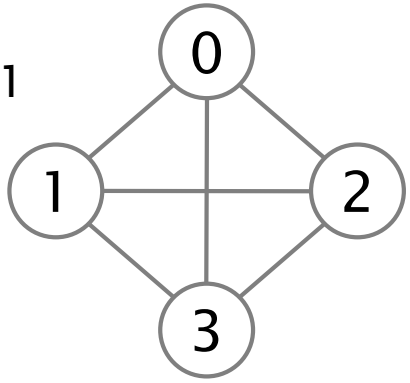
# Representações de grafo

- matriz de adjacências, W
- matriz de incidências, A
- listas de adjacências (incidências)

# Matriz de adjacências, [W]

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

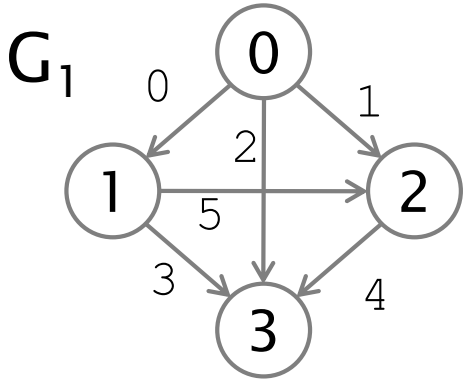
$G_1$




| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 0 |

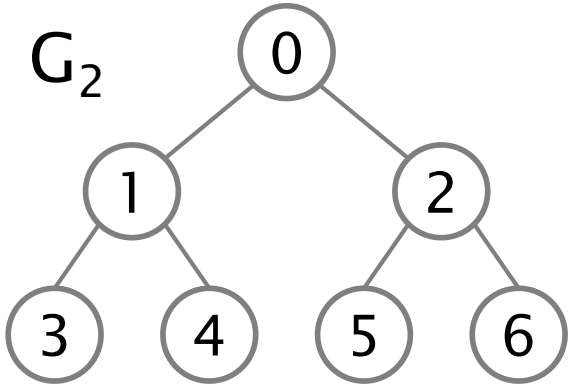
# Matriz de Incidênca

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 0 |




$$[A] = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

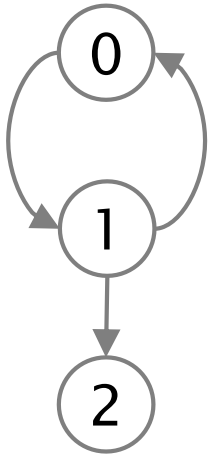
# Matriz de adjacências




| | 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

$G_3$

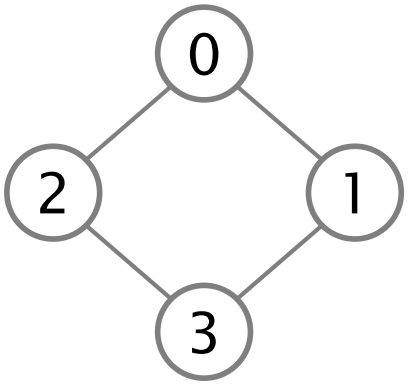



| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 2 | 0 | 0 | 0 |

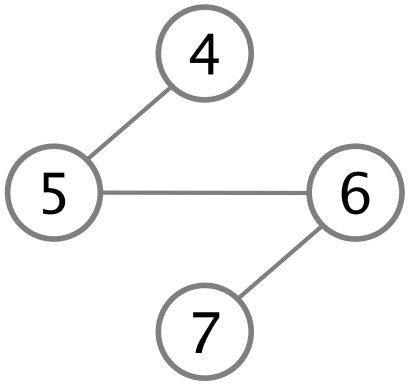
# Matriz de adjacências

$G_4$

$H_1$



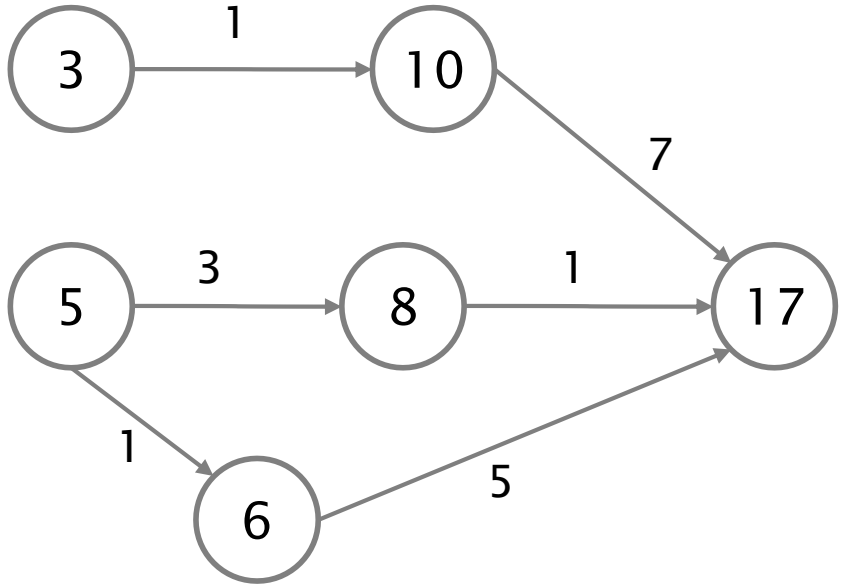

$H_2$




| | 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 |

# Grafo ponderado

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



# 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

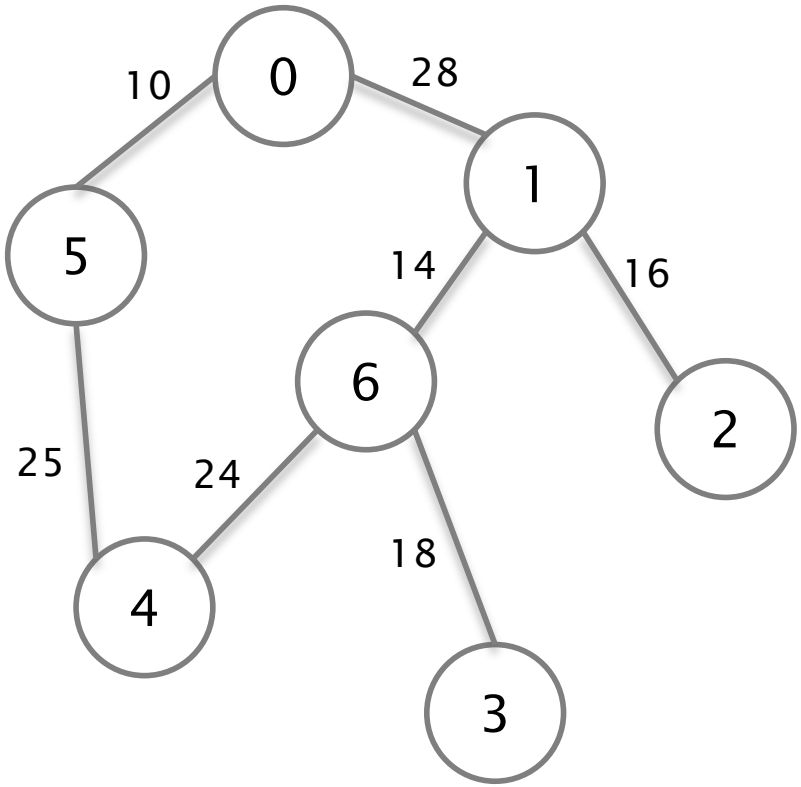
Cada vértice examinado deve ser marcado como visitado.

Por quê?

# dfs
# (depth-first search)

# dfs iniciando em 0

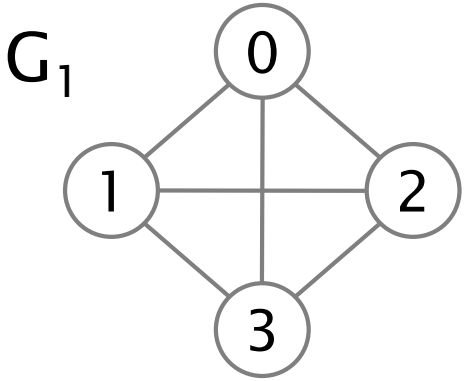
```
dfs(0)
    dfs(1)
     dfs(2)
     dfs(6)
          dfs(3)
          dfs(4)
               dfs(5)
```

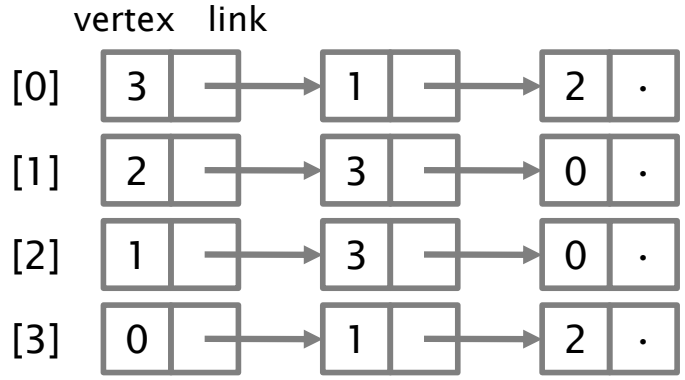


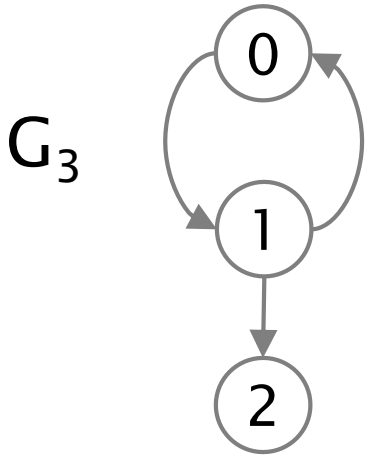
# Percurso em profundidade – *depth first search*

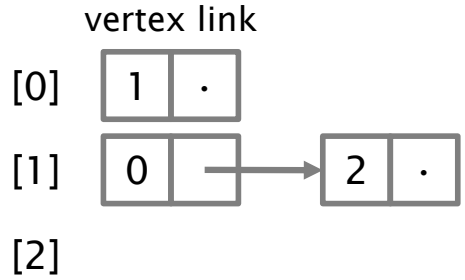
```
/* grafo como matriz de adjacências */
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

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);
}
```

# 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);
}
```

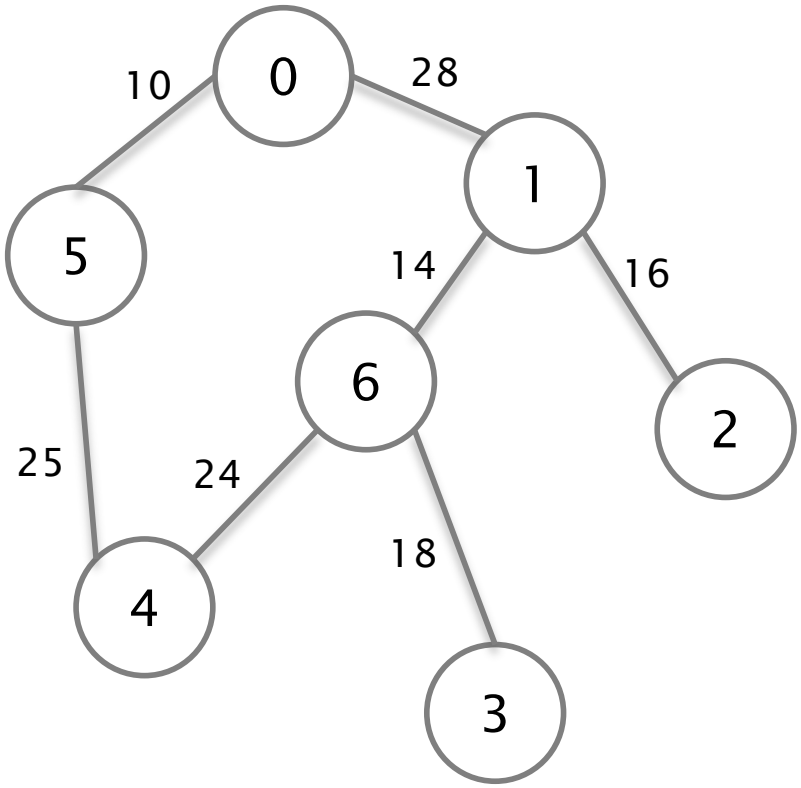
# bfs
# (breadth-first search)

# Percurso em largura
## – *breadth first search*

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

# 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)
     []
```



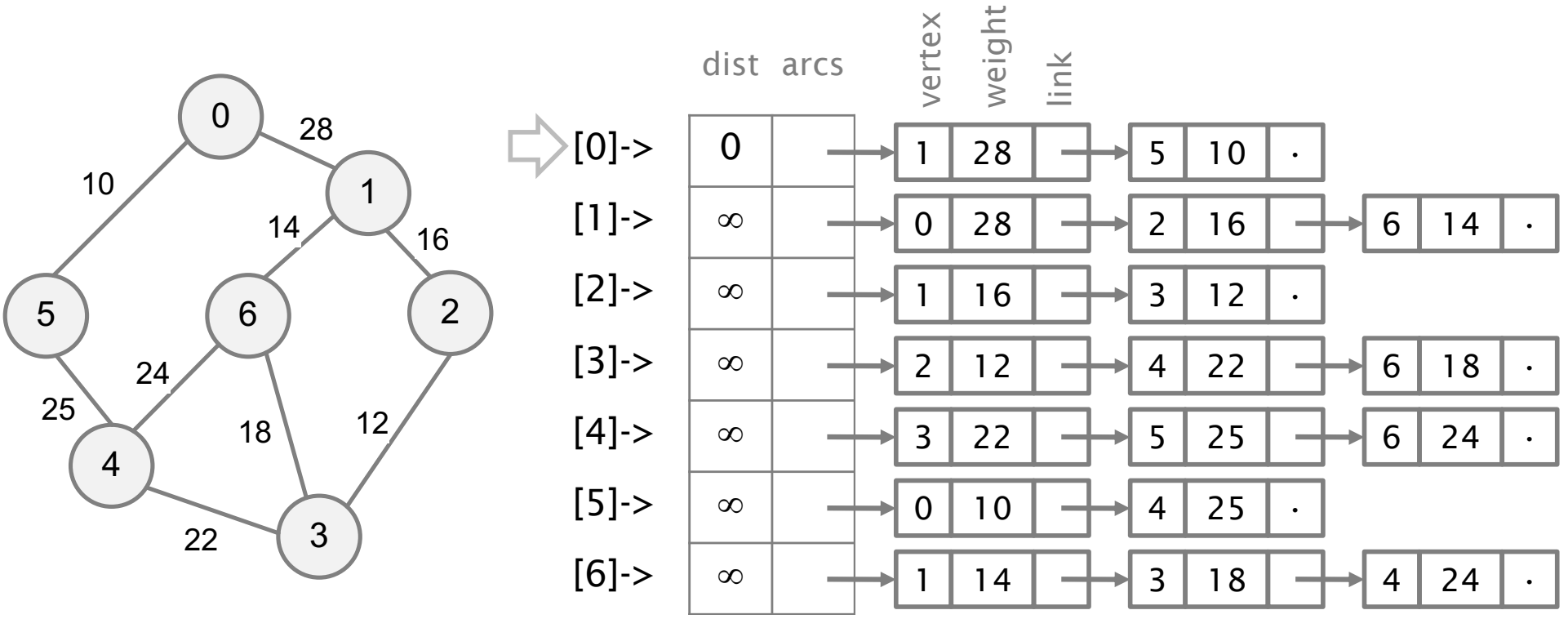
# Menor caminho entre um nó de origem e um de destino – algoritmo de Dijkstra

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.

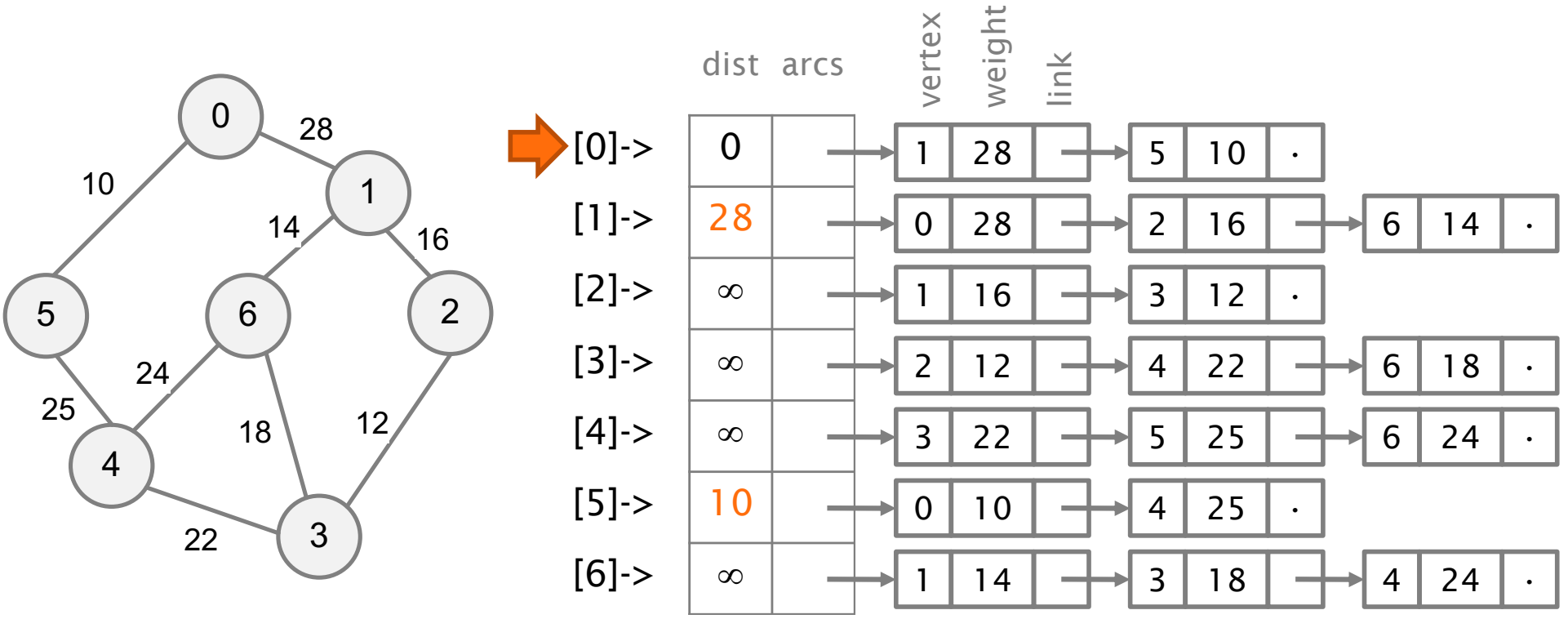
# Dijkstra - exemplo

considerando como vértice inicial 0



# Dijkstra - exemplo

modifica distâncias dos vértices 1 e 5

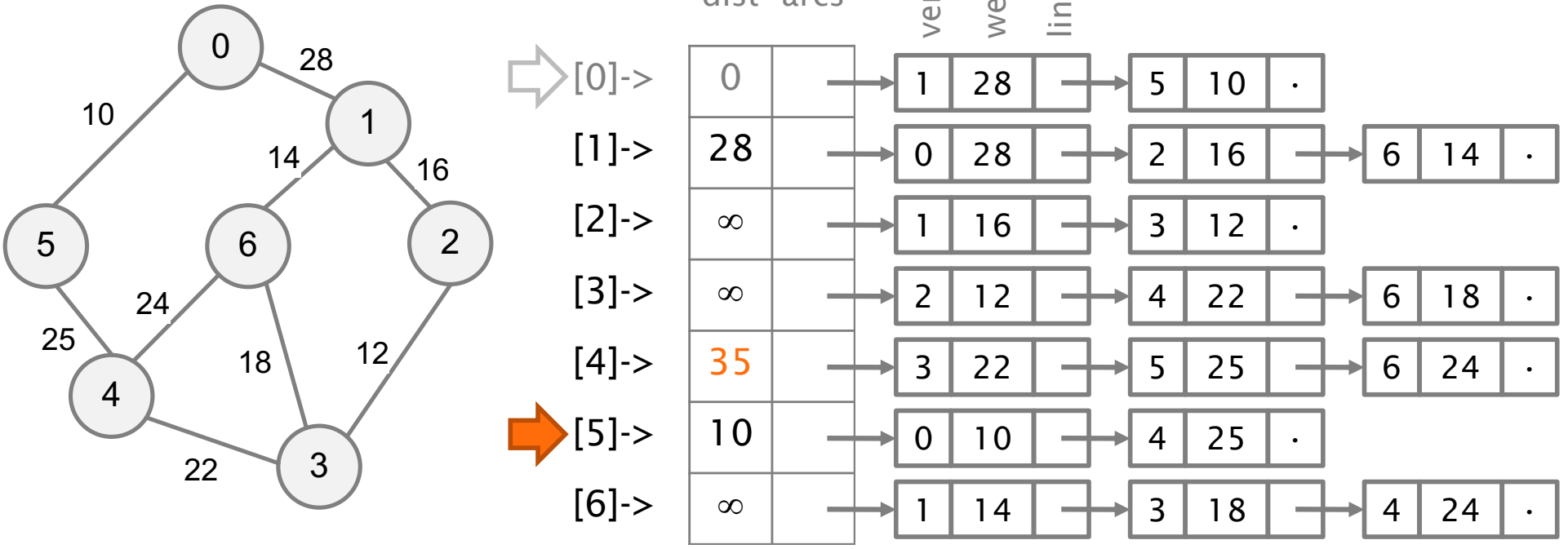


# Dijkstra - exemplo

marca o vértice 0 como visitado

seleciona o vértice 5 (o de menor distância)

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

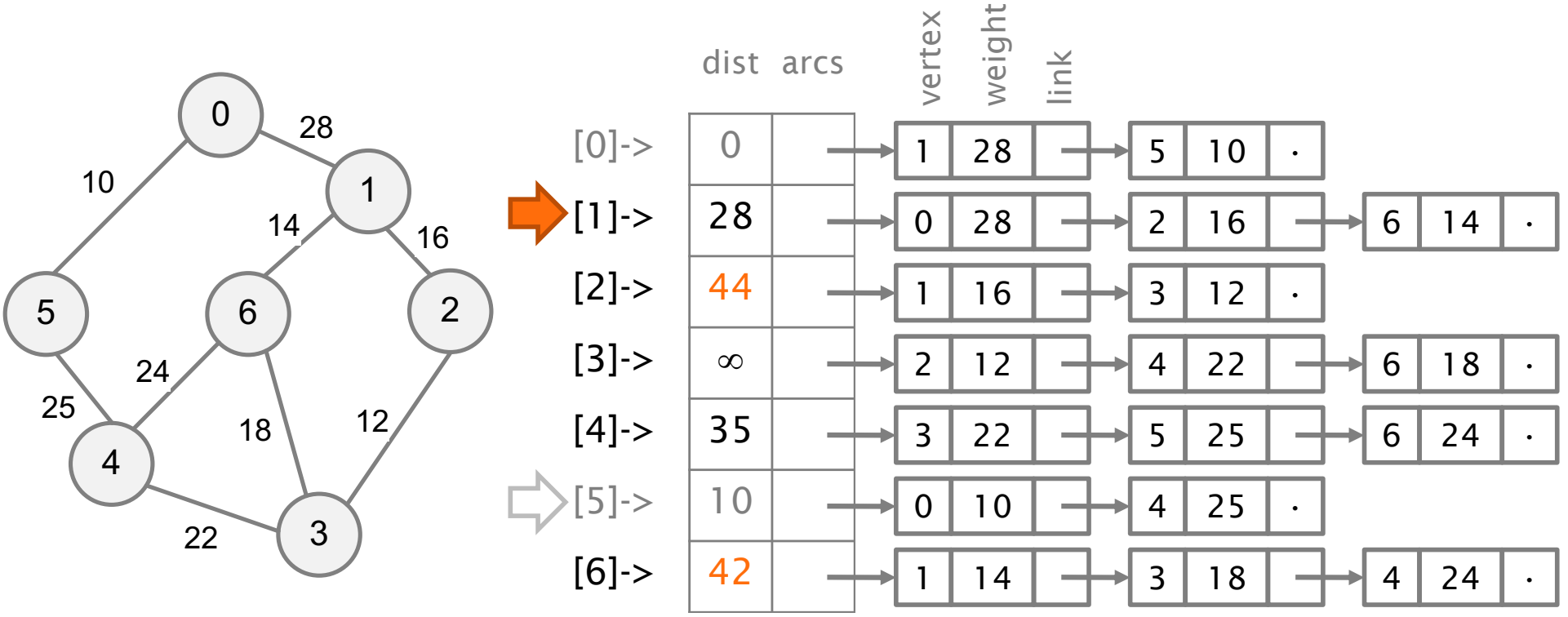


# Dijkstra - exemplo

marca o vértice 5 como visitado

seleciona o vértice 1 (o de menor distância)

ignora vértice 0; modifica distâncias dos vértices 2 e 6

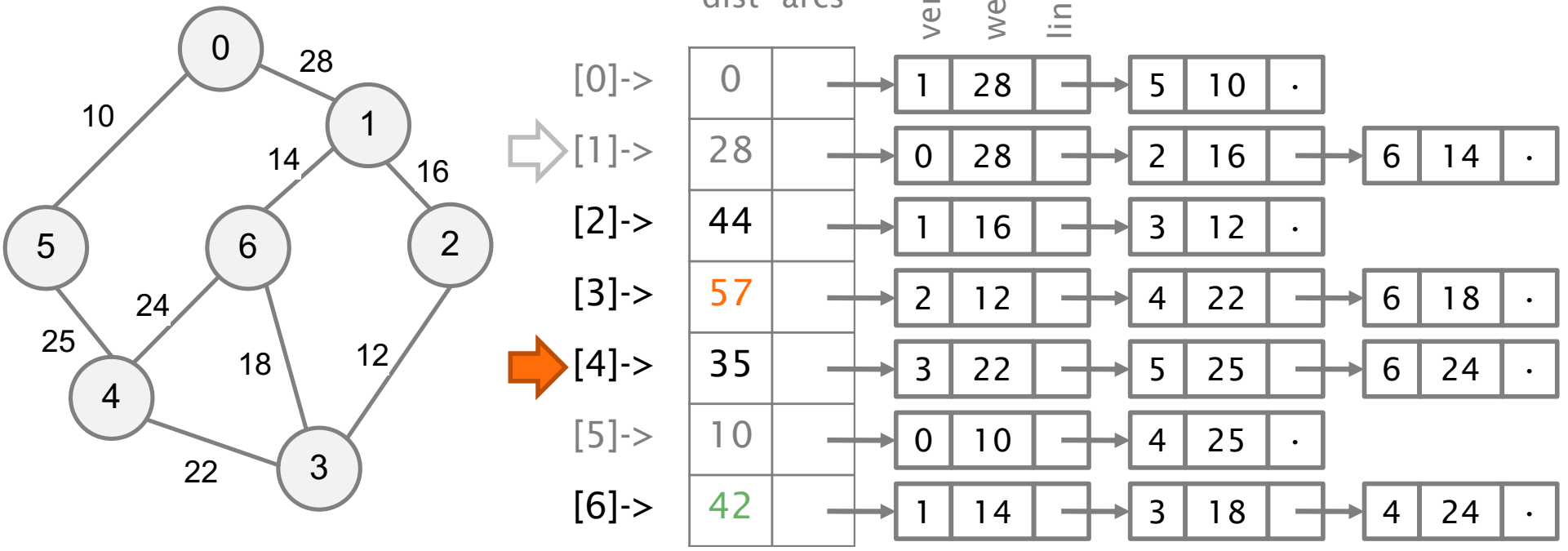


# Dijkstra - exemplo

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

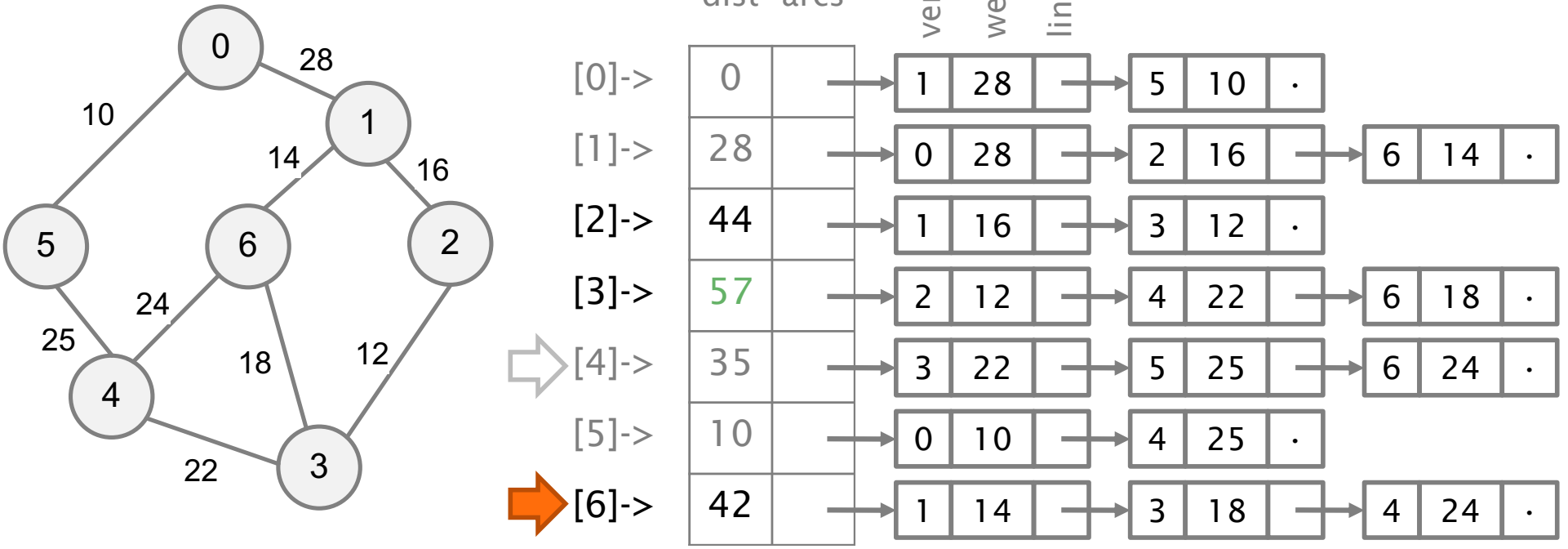


# Dijkstra - exemplo

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

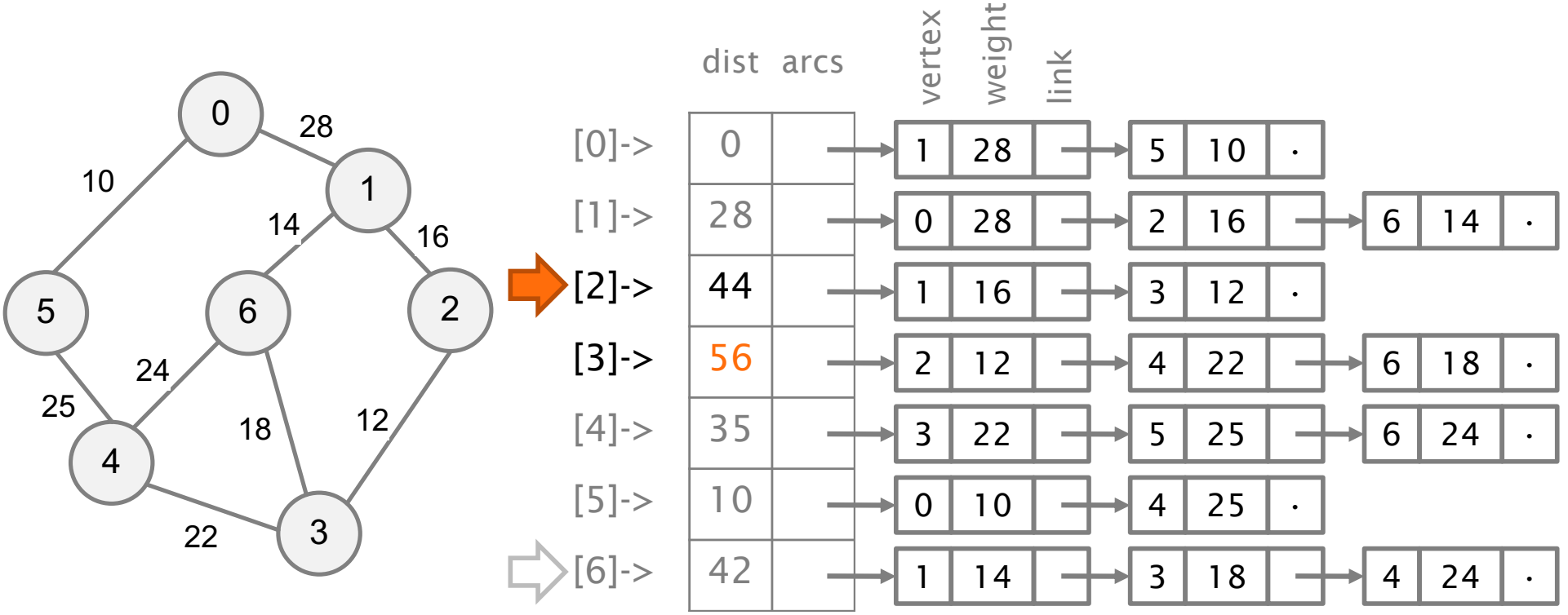


# Dijkstra - exemplo

marca o vértice 6 como visitado

seleciona o vértice 2 (o de menor distância)

ignora vértice 1; modifica distância do vértice 3

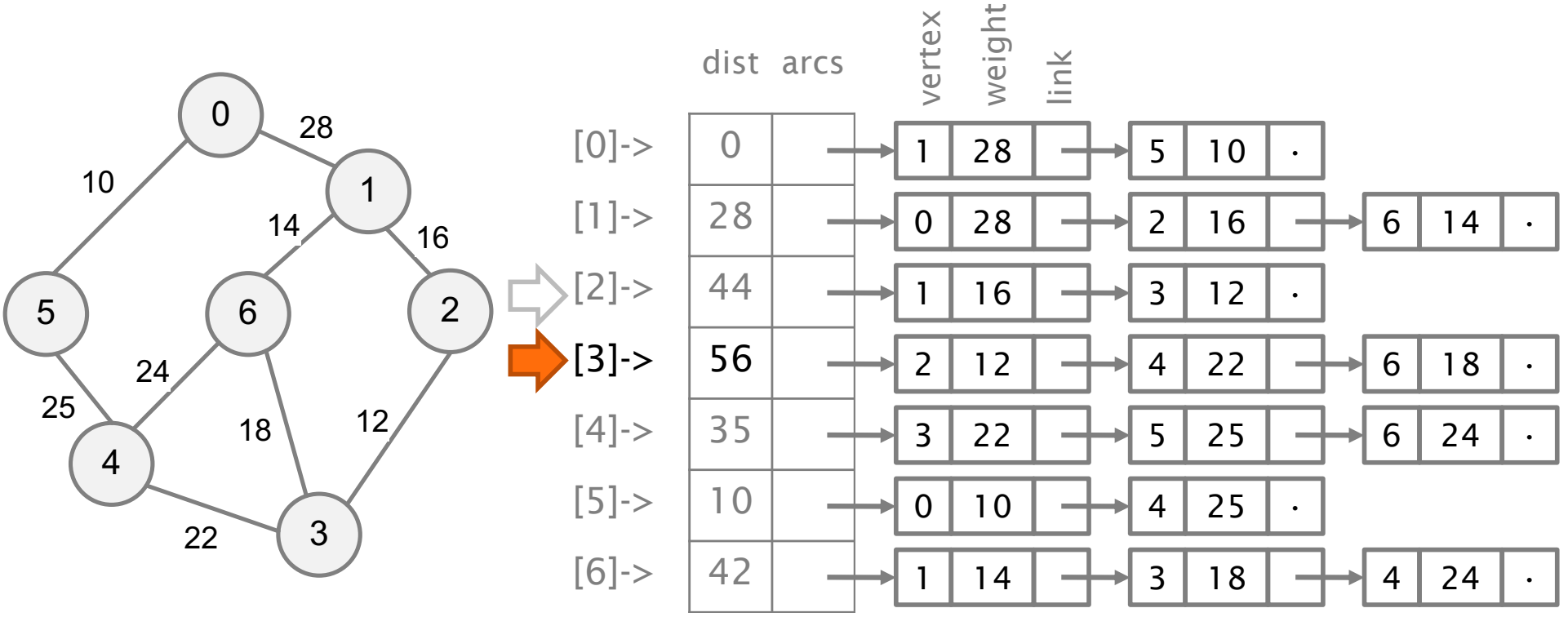


# Dijkstra - exemplo

marca o vértice 2 como visitado

seleciona o vértice 3 (o de menor distância)

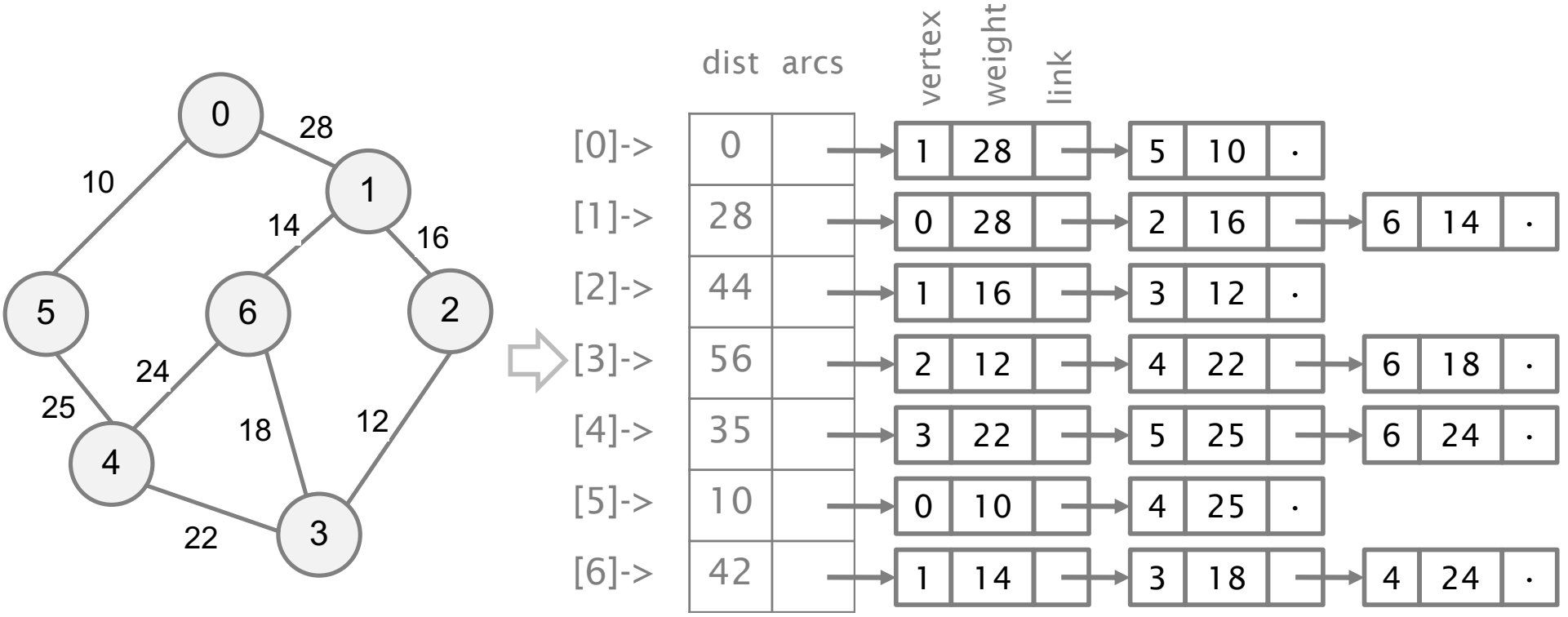
ignora vértices 2, 4 e 6



# Dijkstra - exemplo

marca o vértice 3 como visitado

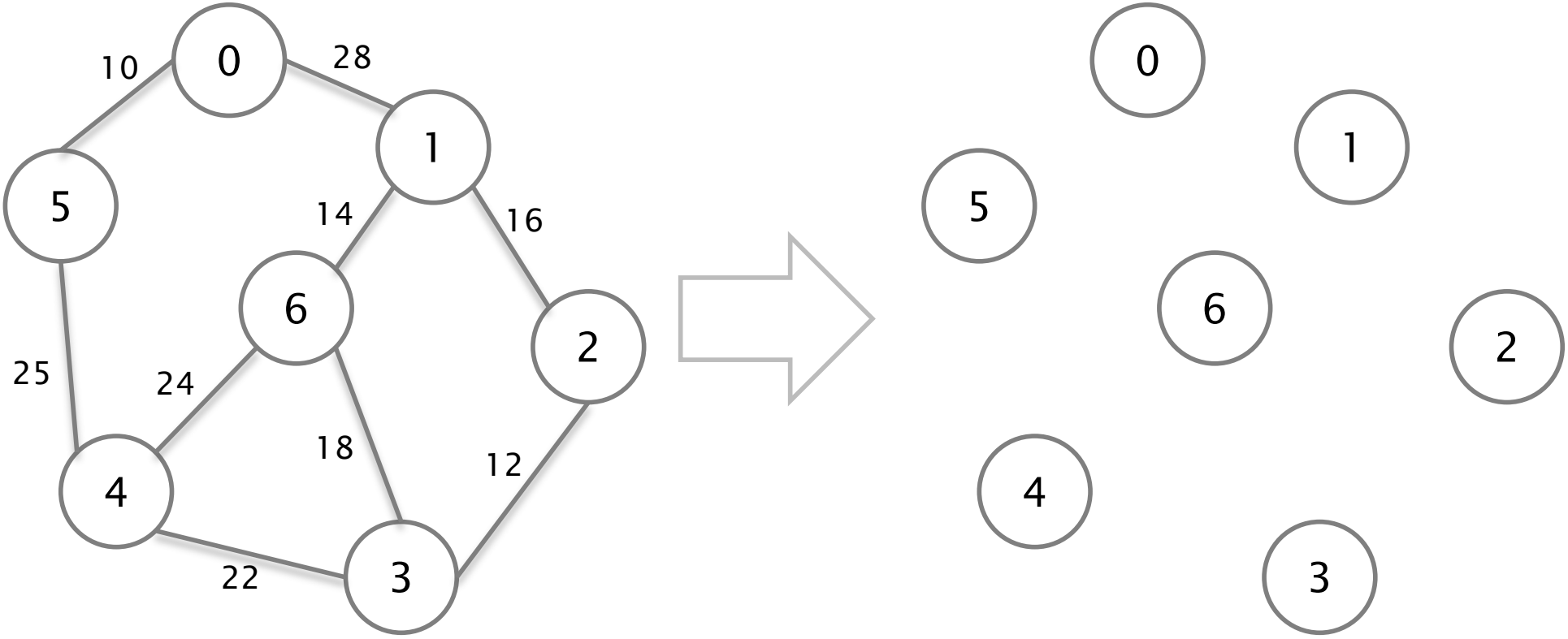
não há mais vértices não visitados – FIM!



# árvore geradora

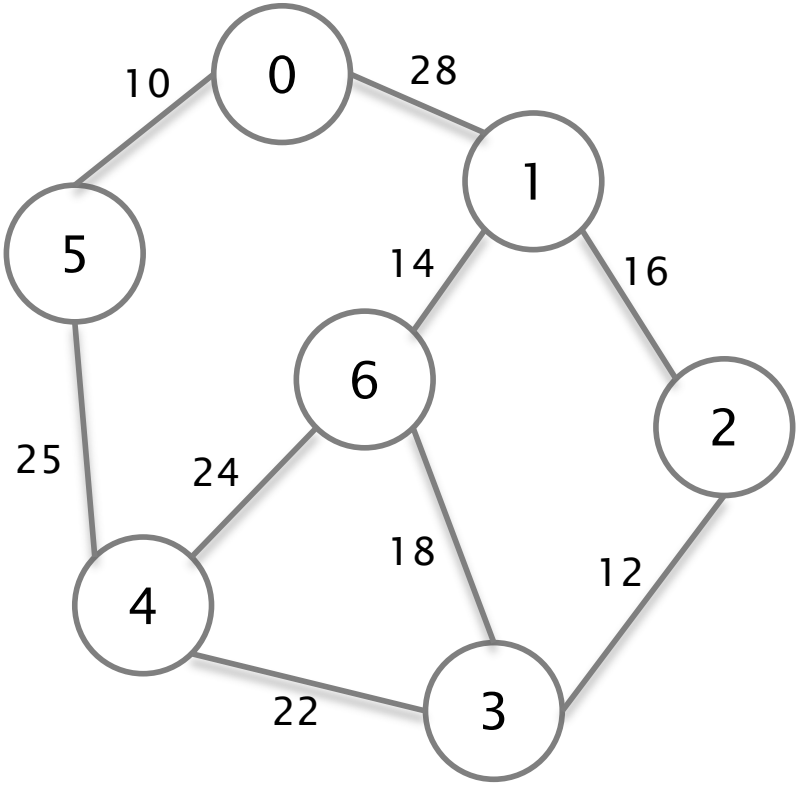
# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

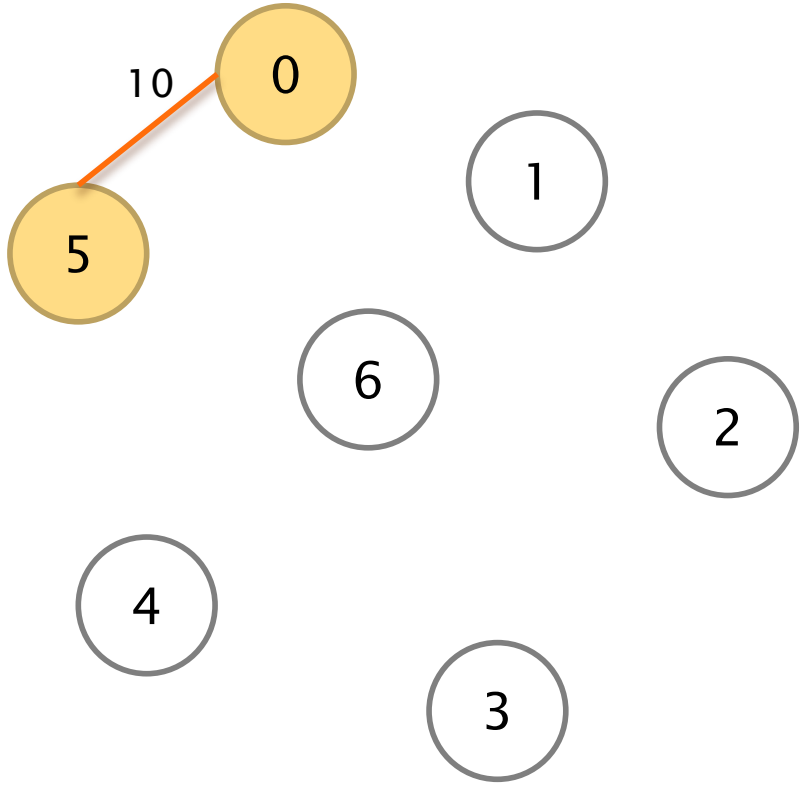
1. Considere cada nó uma árvore (formando uma floresta)



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

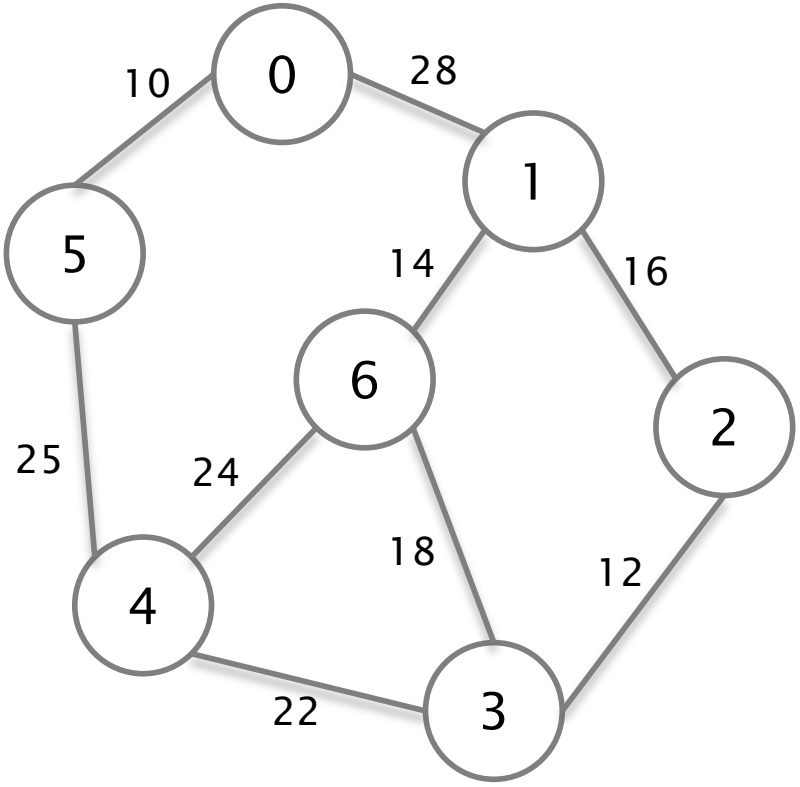
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

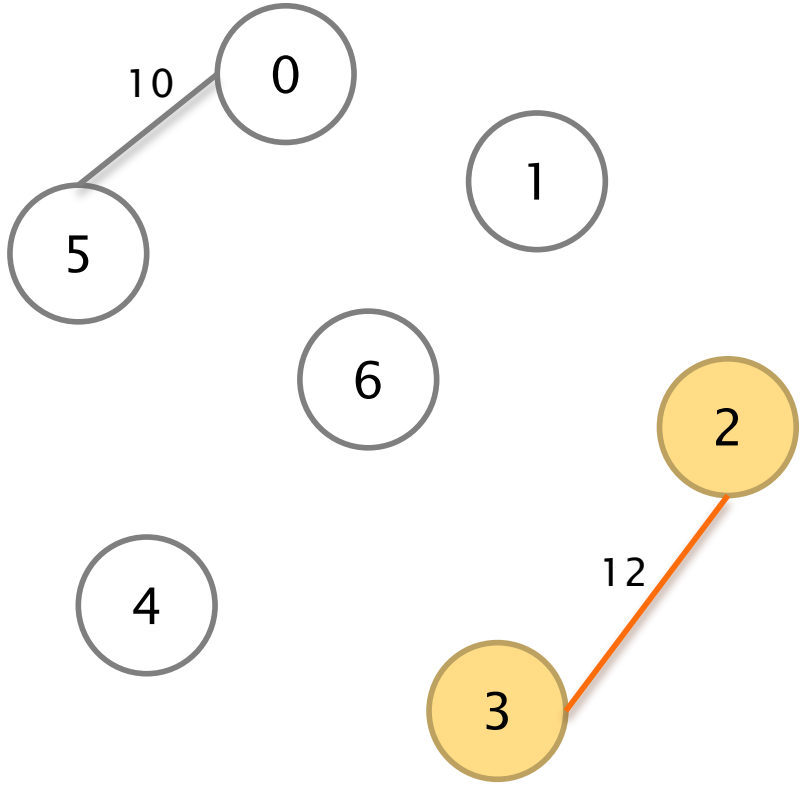


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

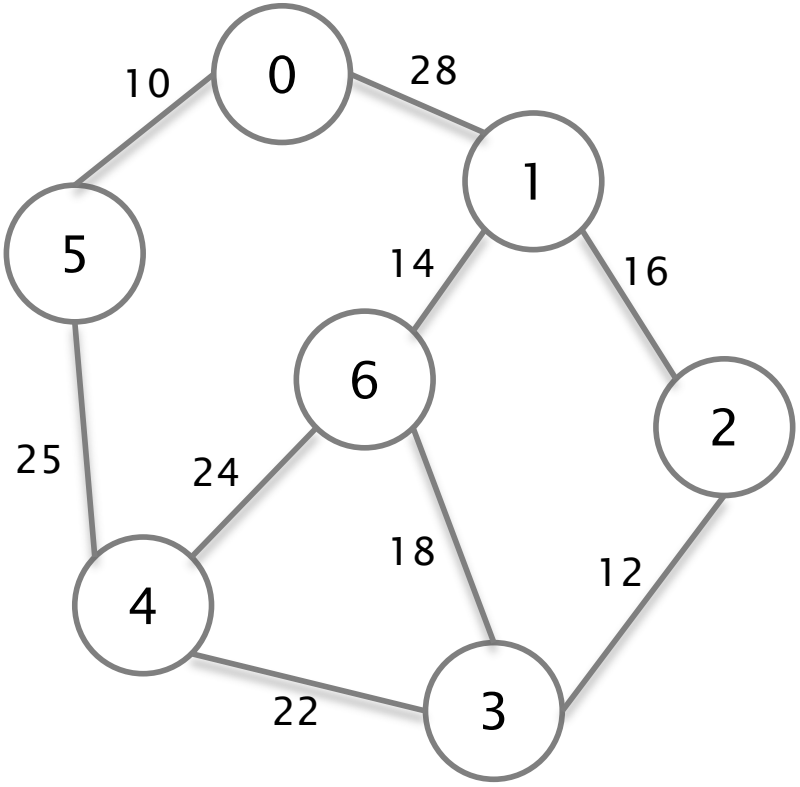
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

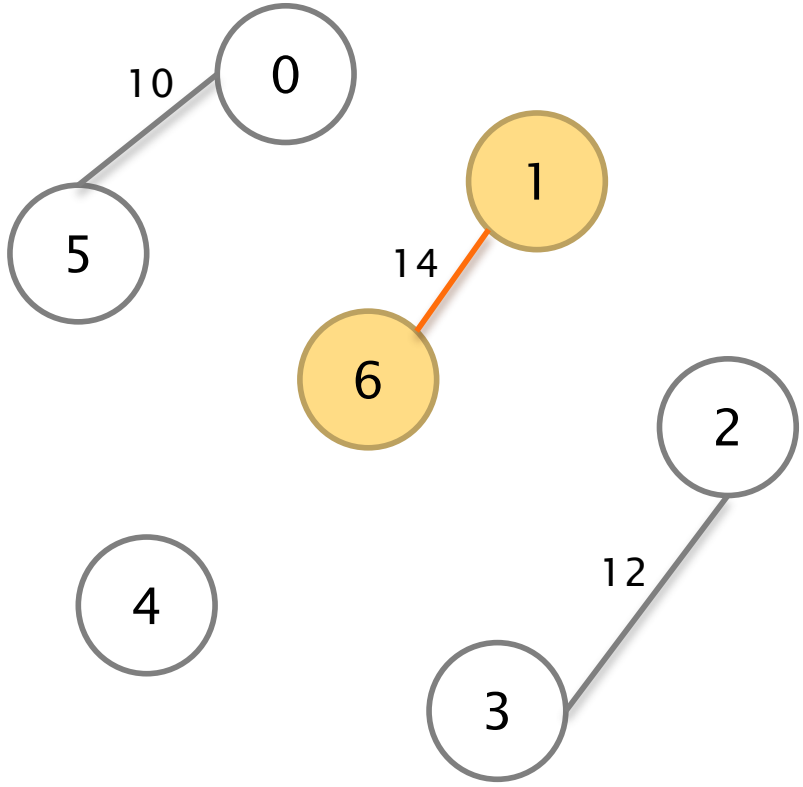


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

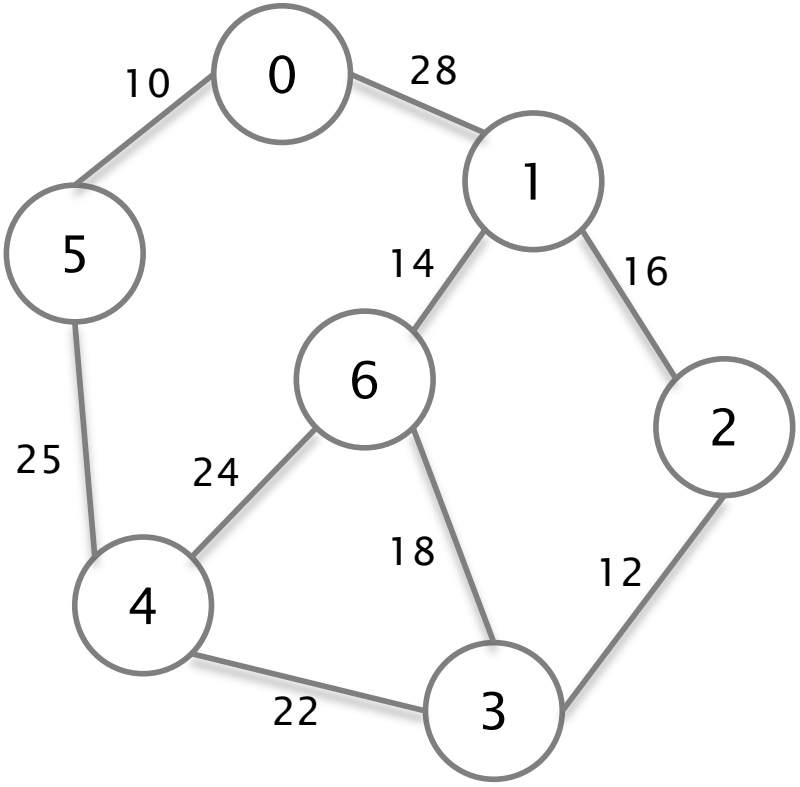
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

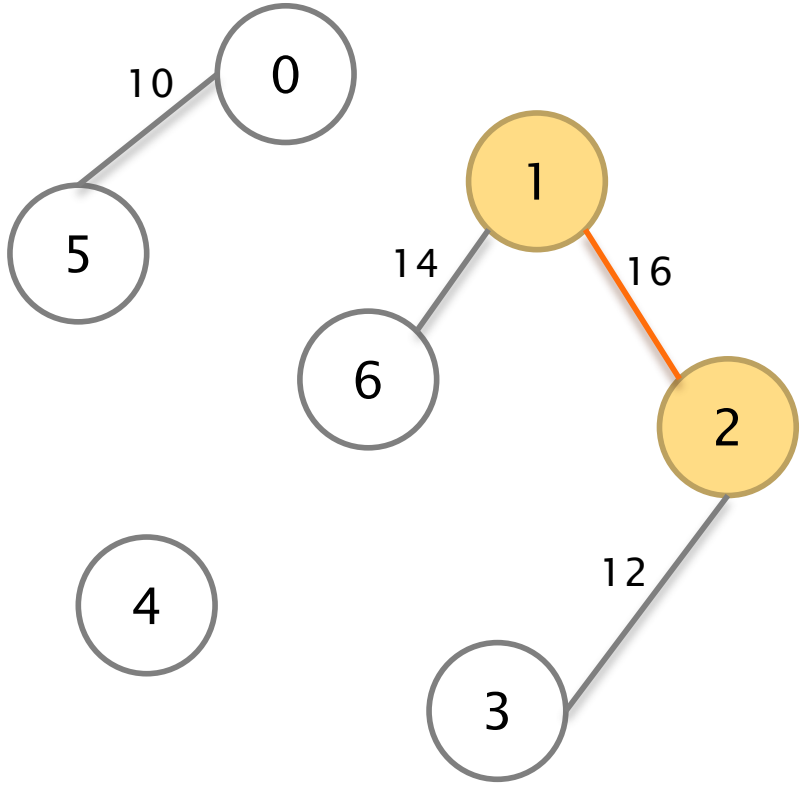


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

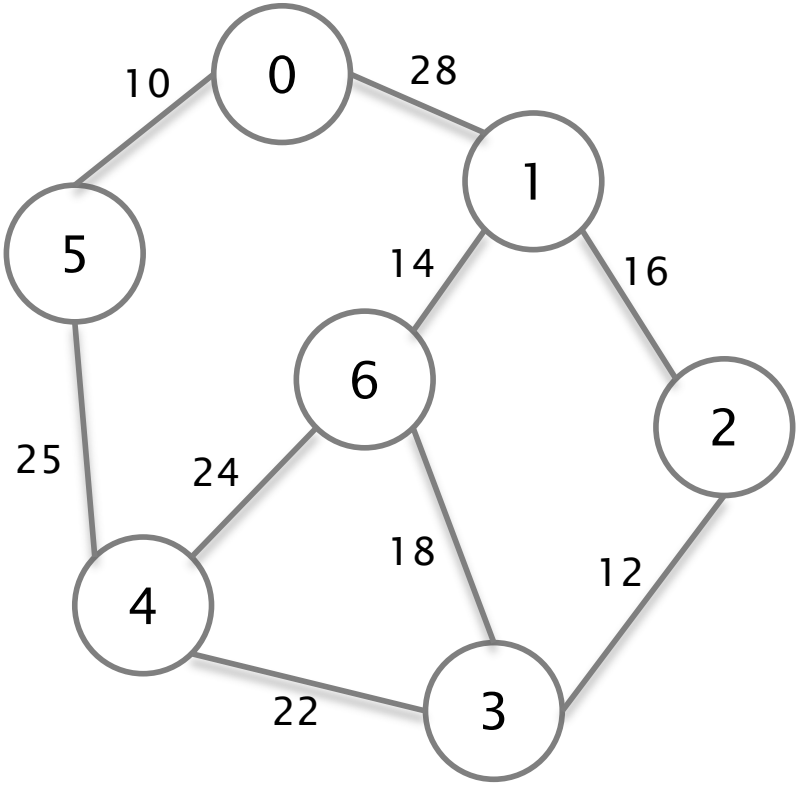
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

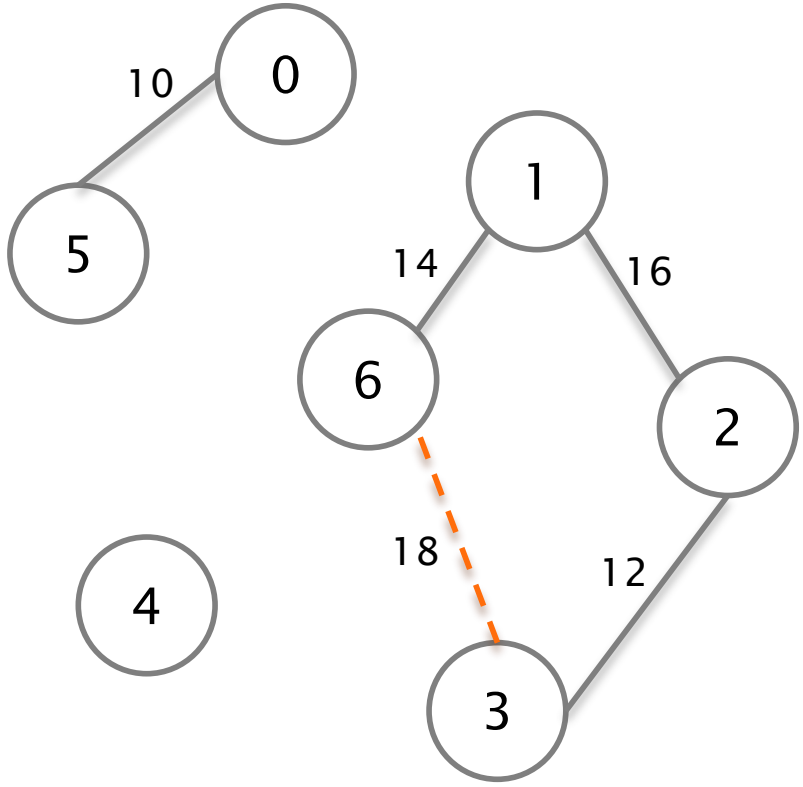


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

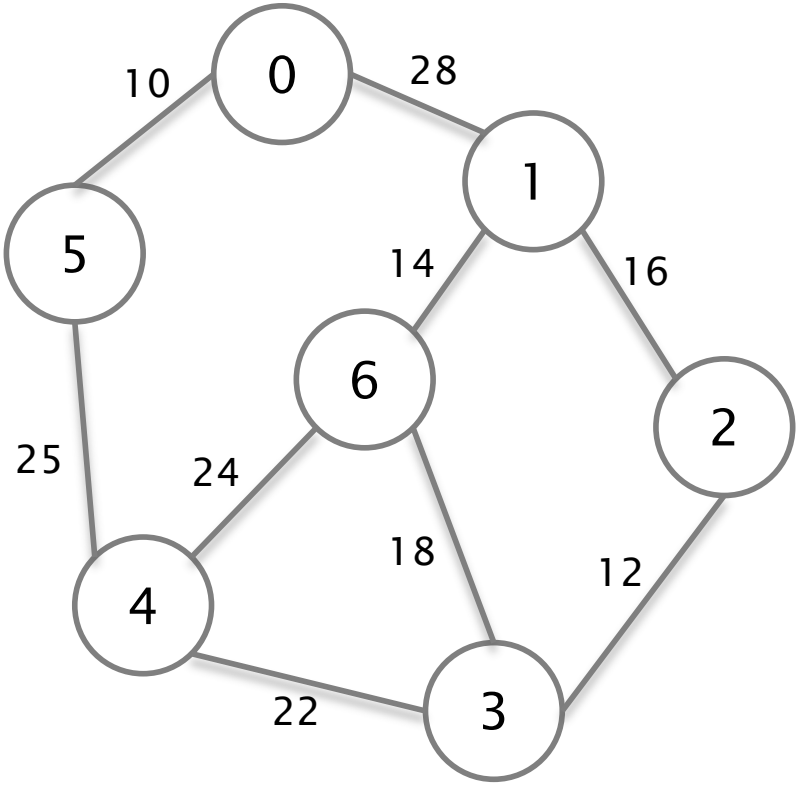
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

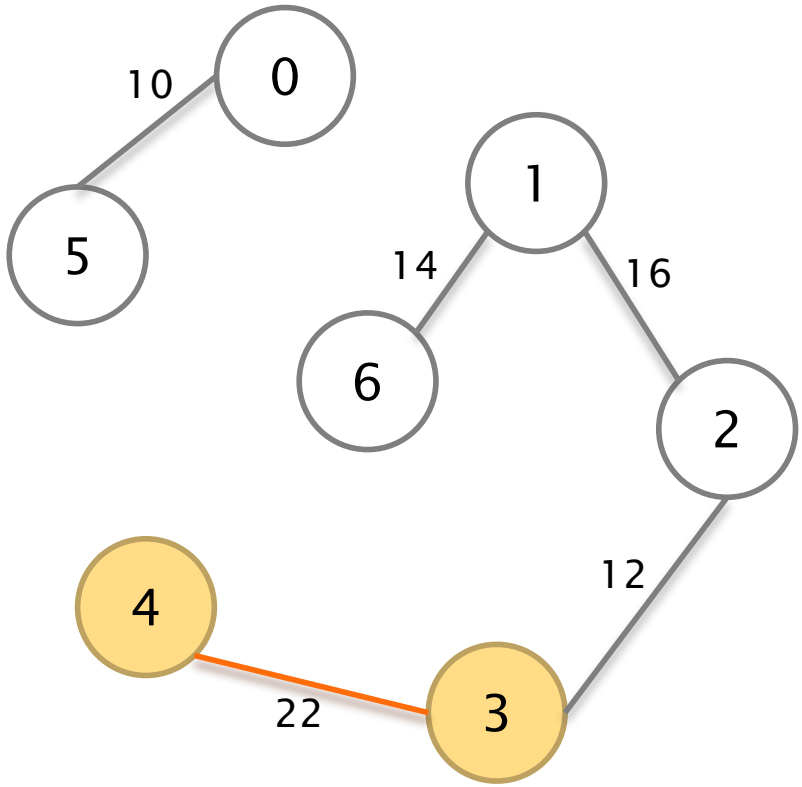


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

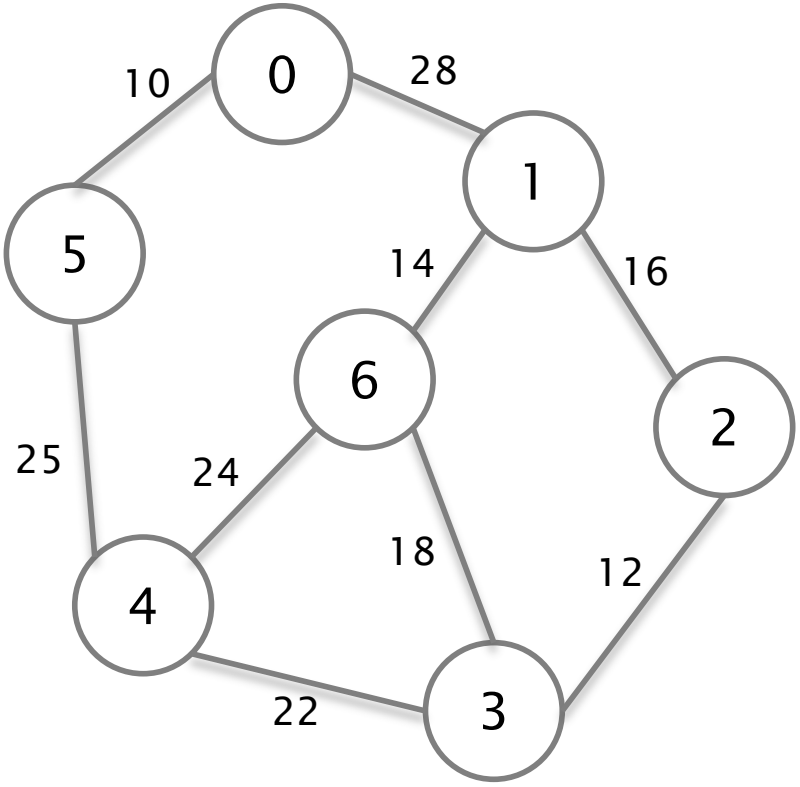
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

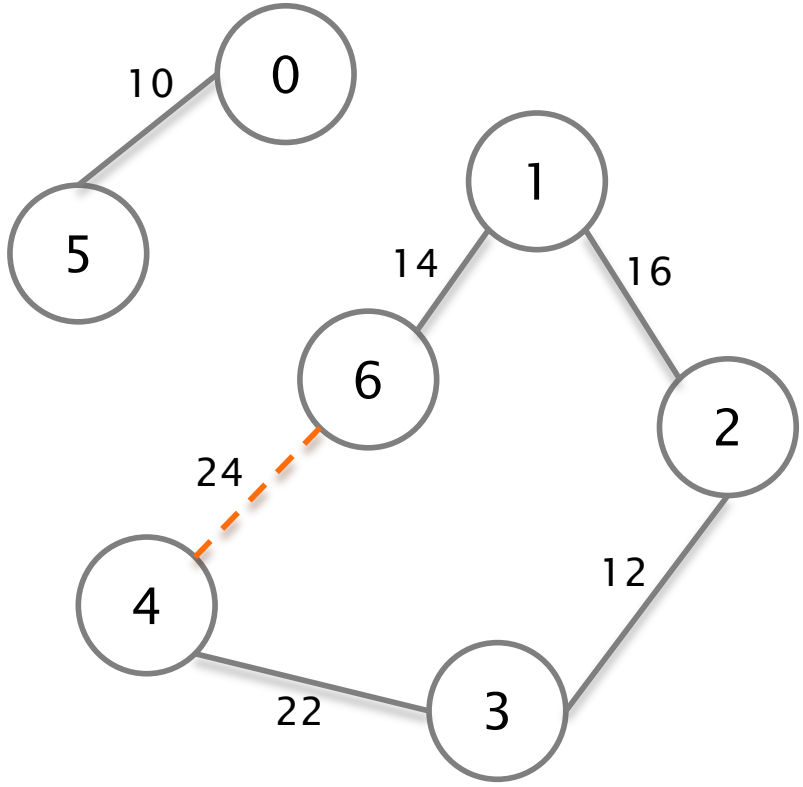


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

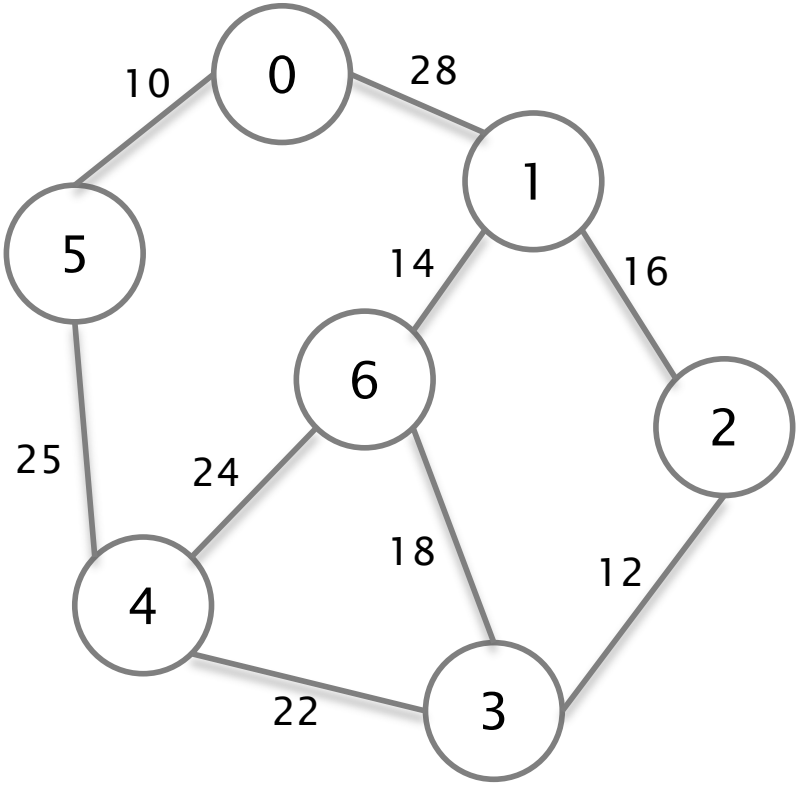
2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.

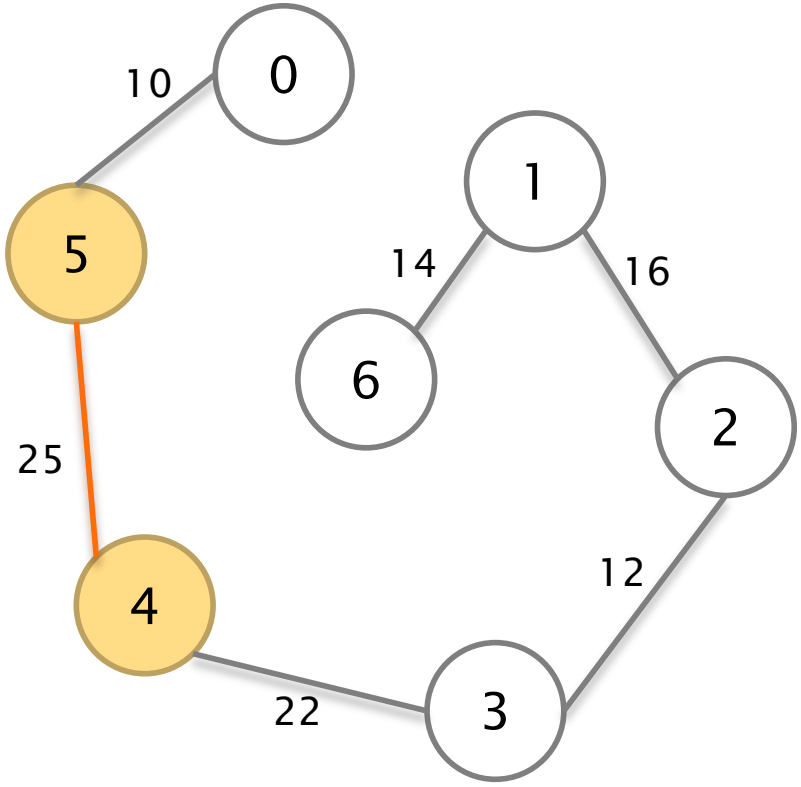


# Árvore geradora (*spanning tree*) de custo mínimo – algoritmo de Kruskal

2. Examine a aresta de menor custo. Se ela unir duas florestas, inclua-a.
3. Repita até todos os nós estarem conectados.



# Algoritmo

1. criar uma floresta F (um conjunto de árvores), onde cada vértice no grafo é uma árvore separada
2. criar um conjunto S contendo todas as arestas no gráfico
3. enquanto S é não vazio e F ainda não está cobrindo
   1. remover uma aresta com peso mínimo de S
   2. se essa aresta conecta duas árvores diferentes, combine as duas árvores em uma única árvore
   3. caso contrário descarte essa aresta.