Pontifícia Universidade Católica do Rio de Janeiro

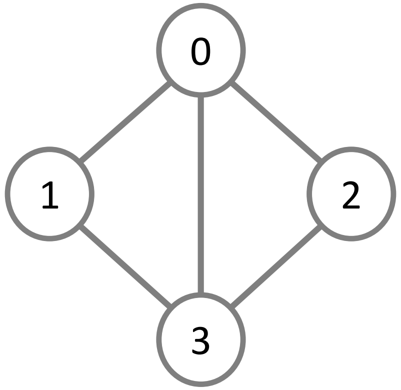
Estruturas de Dados Avançadas- INF1010

Árvore Geradora Mínima
(Algoritmo de Kruskal)

Departamento de Informática PUC-Rio

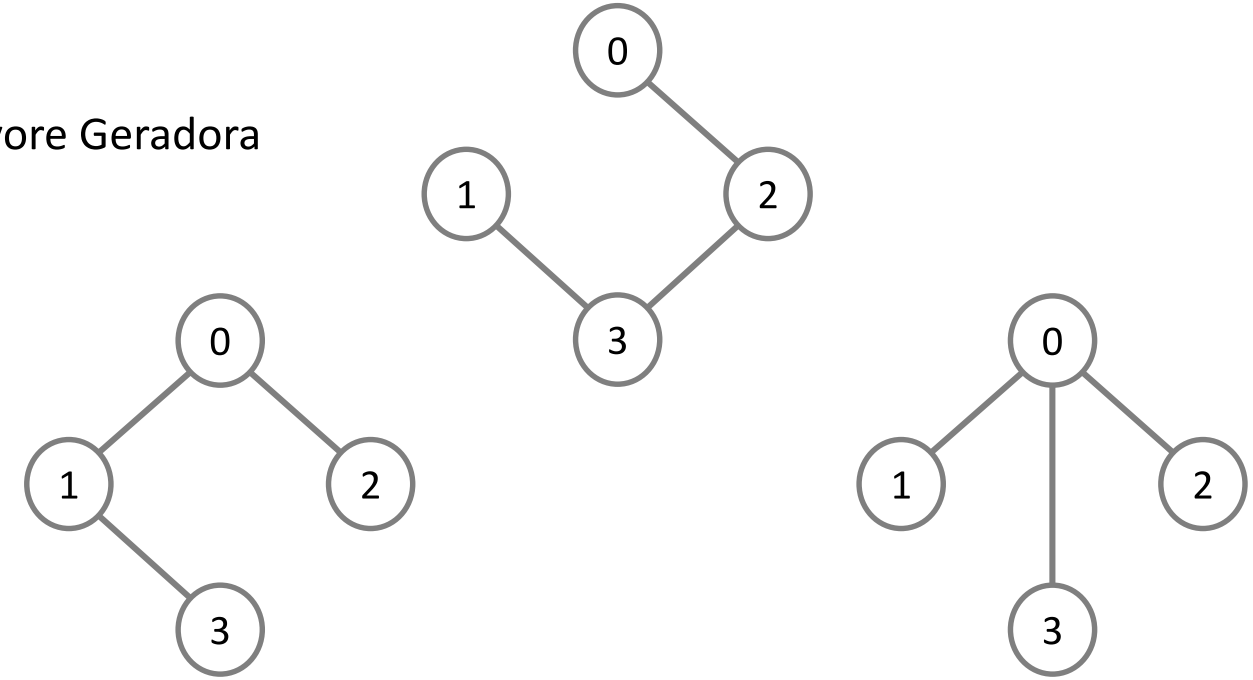
Árvore Geradora (*Spanning Tree*)

- A árvore geradora é um subgrafo **acíclico** de um grafo conectado, que contem todos os vértices, com caminhos entre quaisquer dois vértices (subconjunto de arestas)



Grafo G

Árvore Geradora



Árvore Geradora Mínima (*Minimum Spanning Tree*)

- Dado um **grafo ponderado G = (V,E,p)**, uma árvore geradora de custo mínimo é uma árvore geradora de G tal que a soma dos pesos das arestas é mínima, dentre todas as árvores geradoras de G.

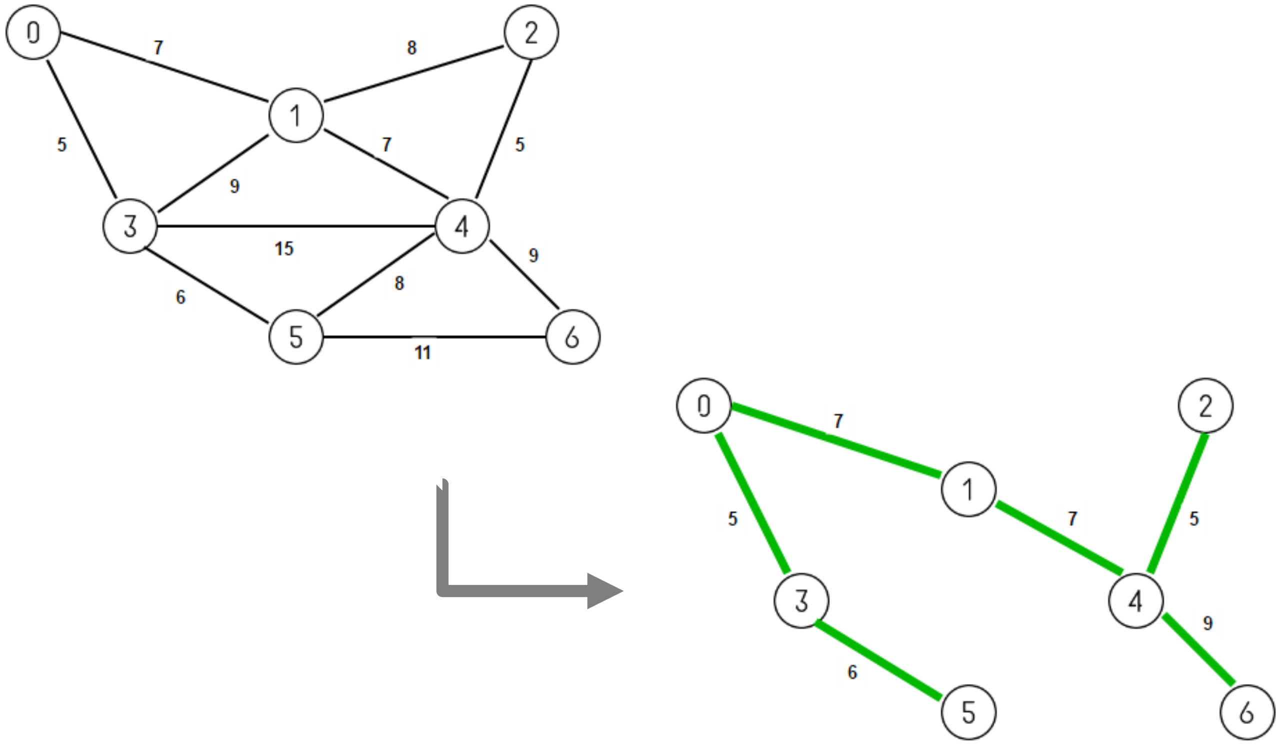
Diversas aplicações:

estradas, circuitos eletrônicos, redes de computadores, controle de processos, ...

Algoritmos: Kruskal, Prim ou Borůvka

os três algoritmos adotam um método “guloso”

construção da solução em etapas/passos, com a seleção de um item em cada passo

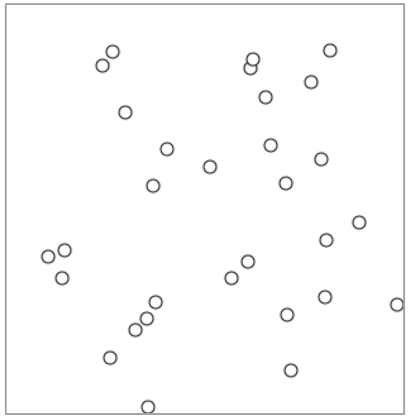


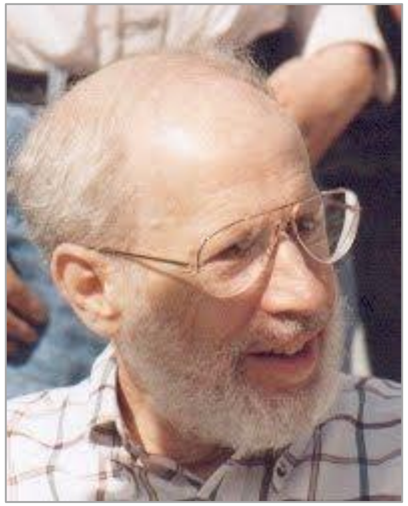
Algoritmo de Kruskal

- Entrada:
  - um grafo ponderado G = (V,E,p)

- Saída:
  - árvore geradora de custo mínimo

- Algoritmo:
  - 1. Considere cada nó em V como uma árvore separada (formando uma "floresta")
  - 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a.
  - 3. Repita o passo 2 até todos os nós estarem conectados.





Joseph Bernard Kruskal, Jr

"On the shortest spanning subtree of a graph and the traveling salesman problem", Proceedings of the American Mathematical Society, pp. 48–50 1956

Algoritmo de Kruskal

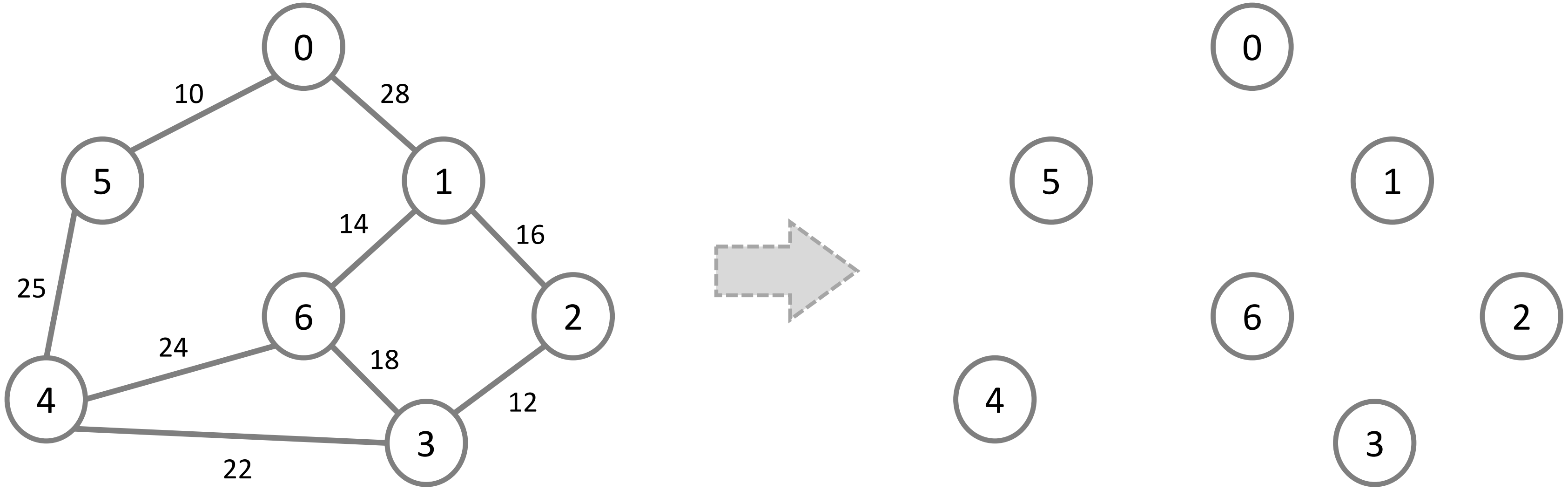
```
Algoritmo Kruskal(G):
    A := ∅
    para cada v ∈ G.V faça:
        MAKE-SET(v)

    para cada (u, v) em G.E ordenado pelo custo(u, v), faça
        Se FIND-SET(u) ≠ FIND-SET(v) então
           A := A ∪ {(u, v)}
           UNION(FIND-SET(u), FIND-SET(v))

    retorna A
```

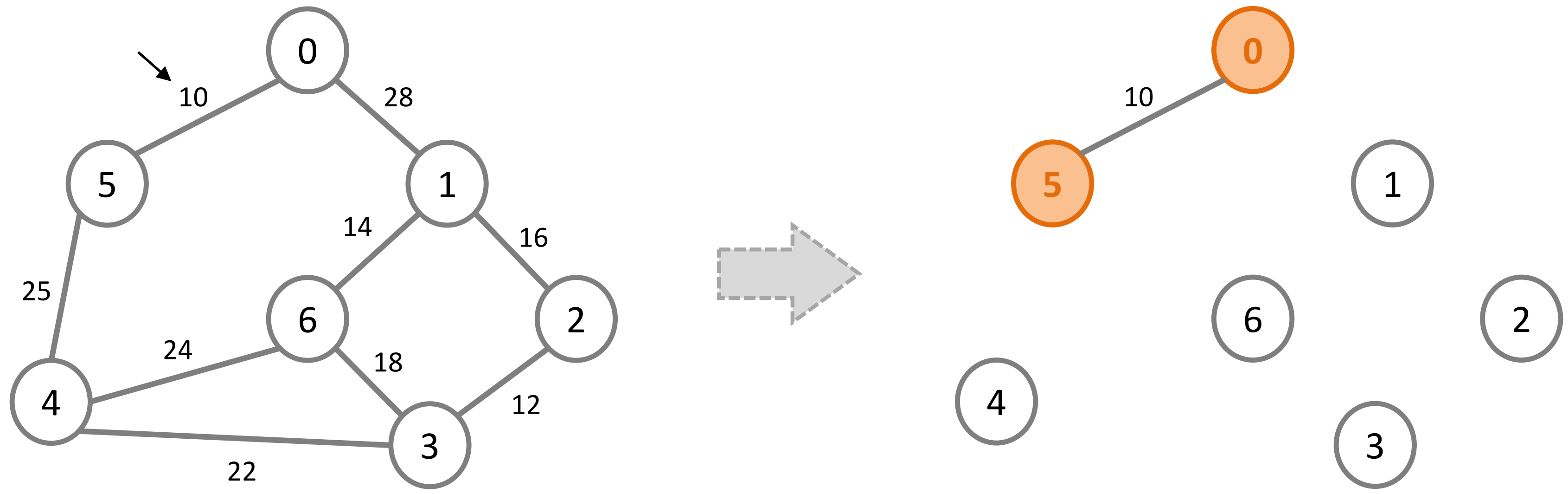
Algoritmo de Kruskal

- 1. Considere cada nó como uma árvore separada (formando uma floresta – Make-Set)



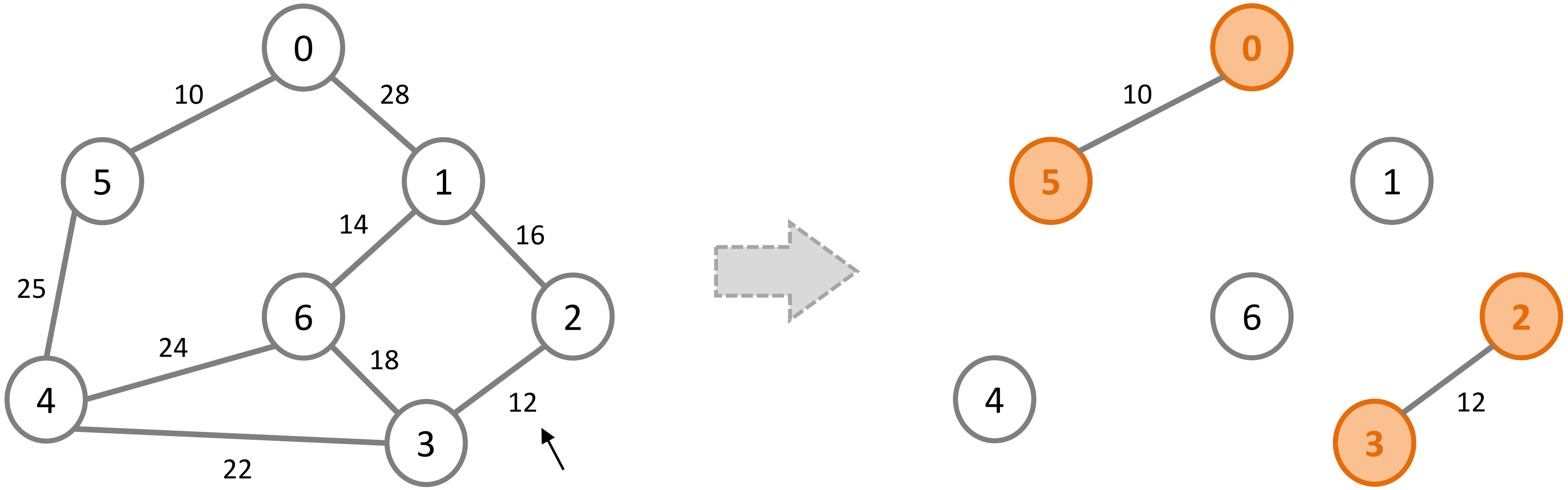
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a
- 3. Repita o passo 2 até todos os nós estarem conectados



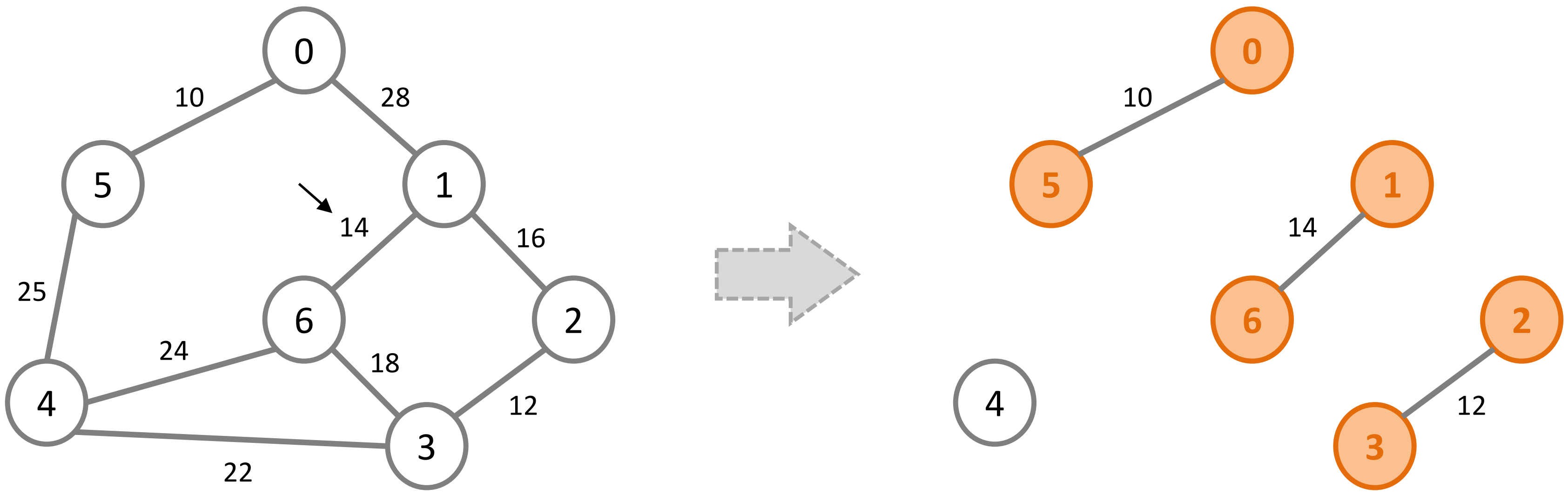
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a
- 3. Repita o passo 2 até todos os nós estarem conectados



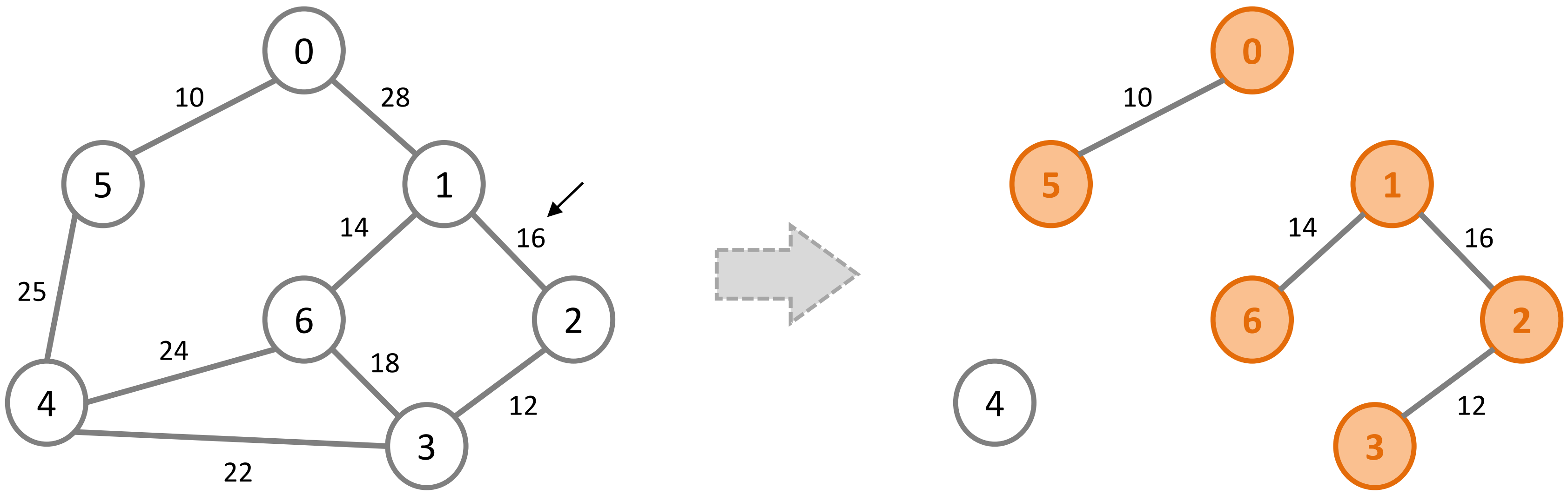
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a
- 3. Repita o passo 2 até todos os nós estarem conectados



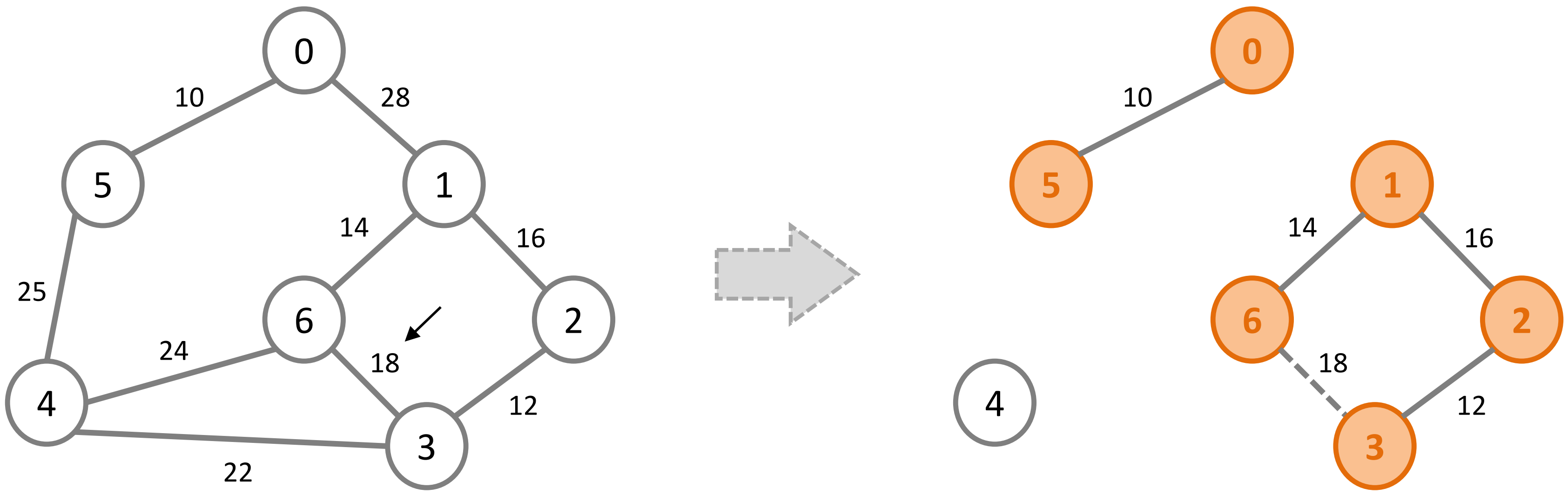
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a
- 3. Repita o passo 2 até todos os nós estarem conectados



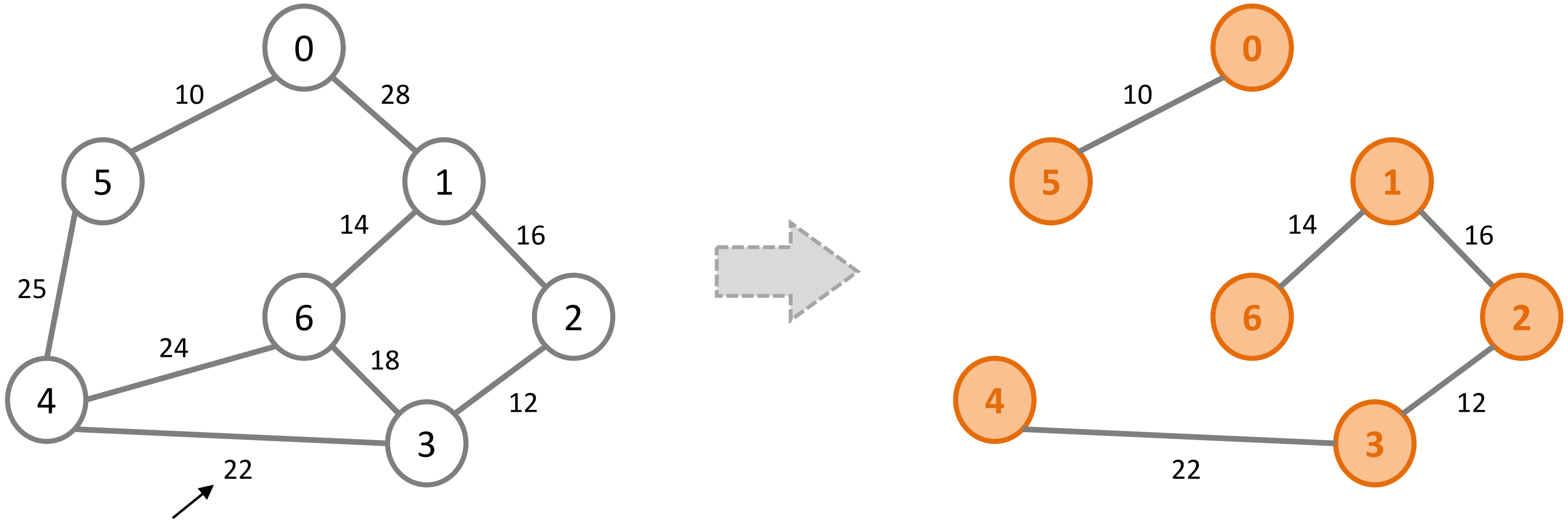
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. **Se ela unir duas árvores na floresta, inclua-a**
- 3. Repita o passo 2 até todos os nós estarem conectados



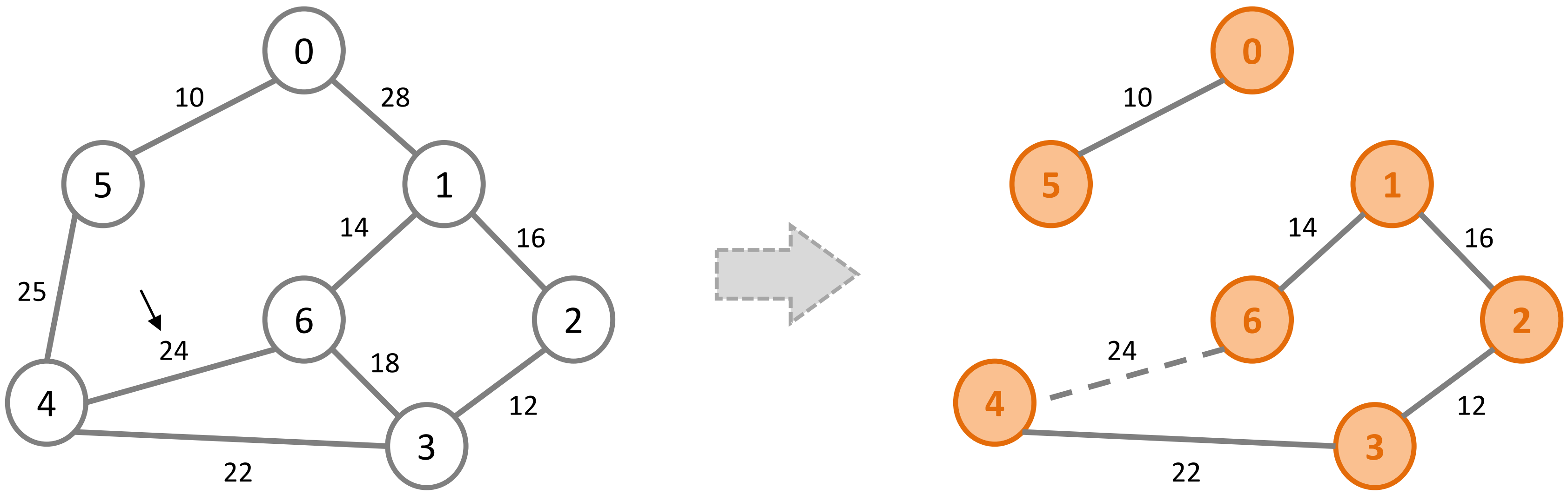
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a
- 3. Repita o passo 2 até todos os nós estarem conectados



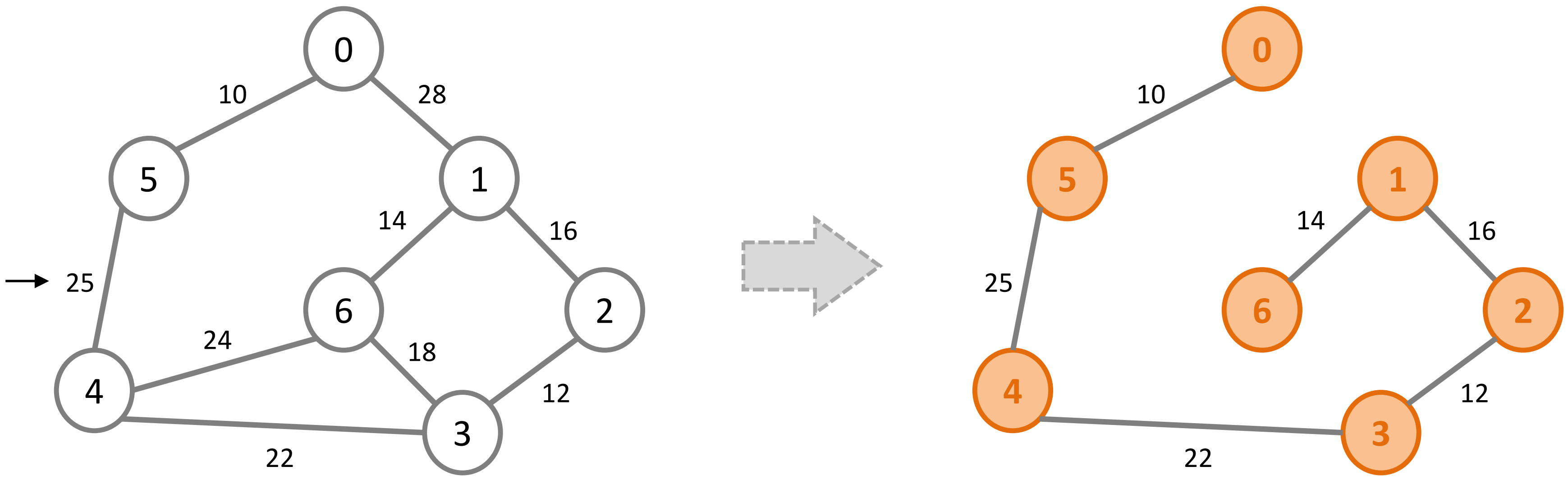
Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. **Se ela unir duas árvores na floresta, inclua-a**
- 3. Repita o passo 2 até todos os nós etarem conectados



Algoritmo de Kruskal

- 2. Examine a aresta de menor custo. Se ela unir duas árvores na floresta, inclua-a
- 3. Repita o passo 2 **até todos os nós estarem conectados**



Pontifícia Universidade Católica do Rio de Janeiro

Estruturas de Dados Avançadas- INF1010

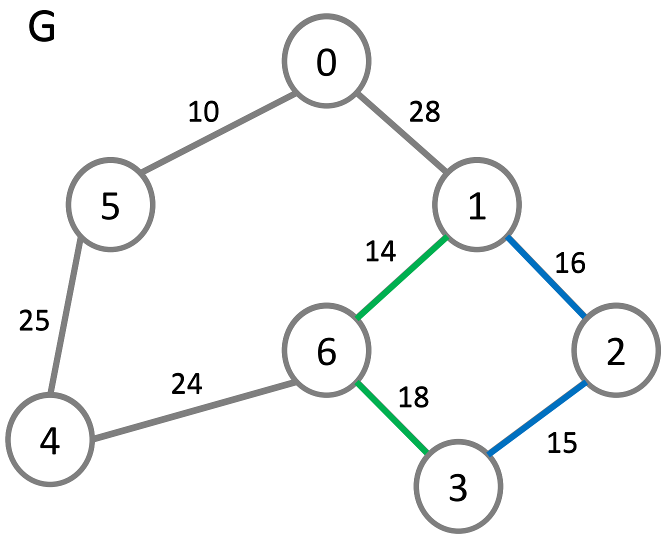
Caminho mais curto
(Algoritmo de Dijkstra)

2023.1

Departamento de Informática PUC-Rio

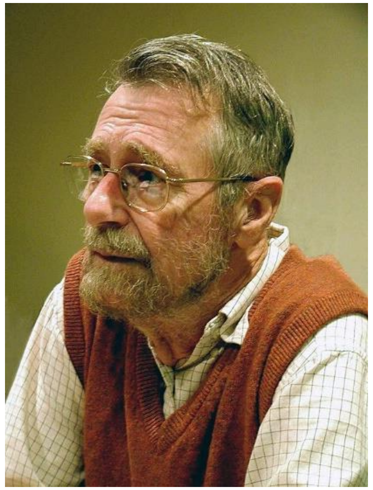
Introdução

- O problema do caminho mais curto (ou caminho mínimo) procura minimizar o custo de um percurso entre dois vértices
  - O custo do percurso é a soma dos pesos de cada aresta percorrida.
  - Calcula-se o caminho entre nós ***i*** (origem) e ***j*** (destino) com menor peso total.



Aplicações: peso → custo, tempo, ...
- roteamento em redes
- deslocamento de caminhões
- desenho de chips
- roteamento de mensagens em telecomunicações
- jogos
- ...

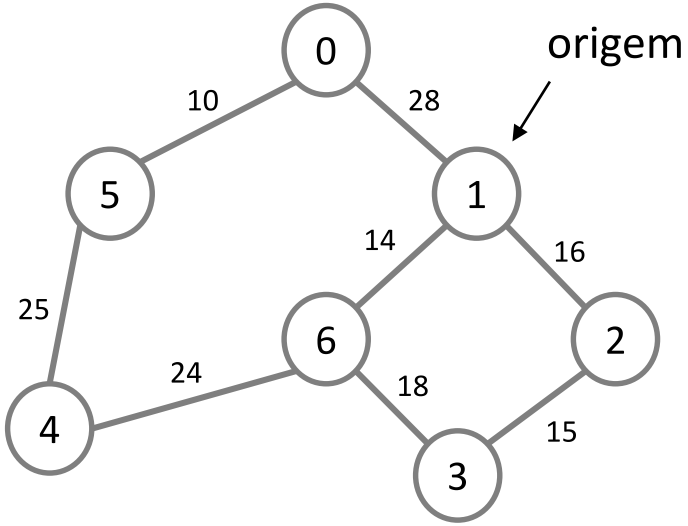
Algoritmo de Dijkstra



Edsger W. Dijkstra

- **Entradas:**
  - Um grafo ponderado G = (V,E,p)
  - Um vértice V do grafo
- **Saída:**
  - Menor caminho entre V e cada um dos nós do grafo
- **Ideia básica:**
  - construir os caminhos mínimos na ordem crescente de distâncias dos vértices ao vértice “origem” (i.e., começando pelos nós mais “próximos”)

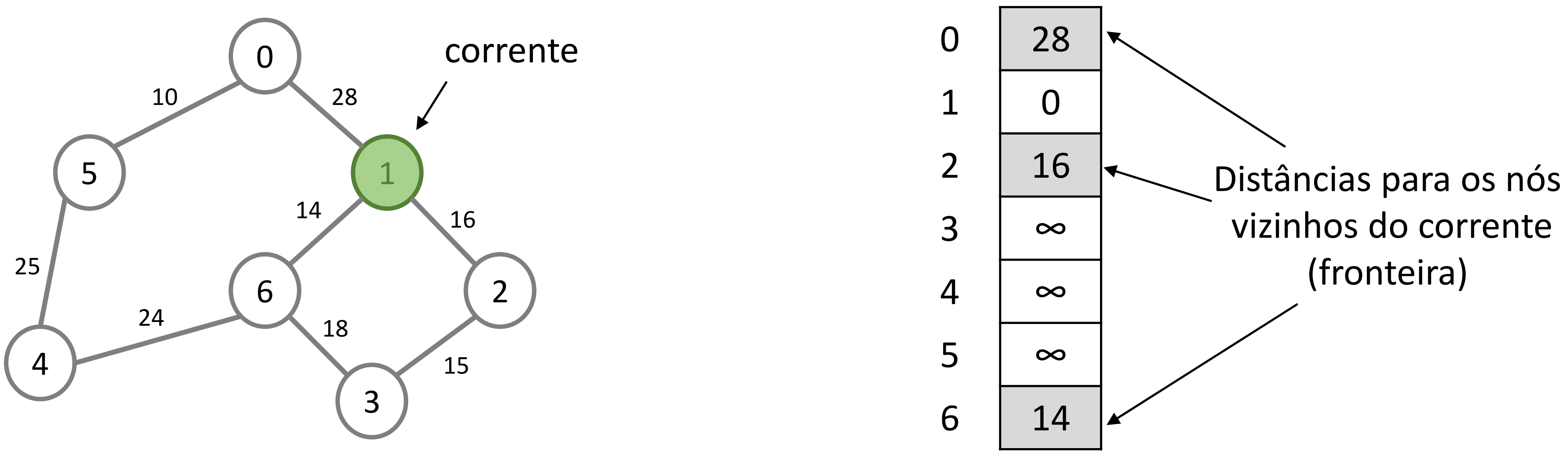
Algoritmo de Dijkstra



0	∞
1	0
2	∞
3	∞
4	∞
5	∞
6	∞

1. Defina o nó de origem
2. Atribua a todos os nós um valor de distância ao nó de origem: valor zero ao nó de origem, e *infinito* para todos os outros nós.

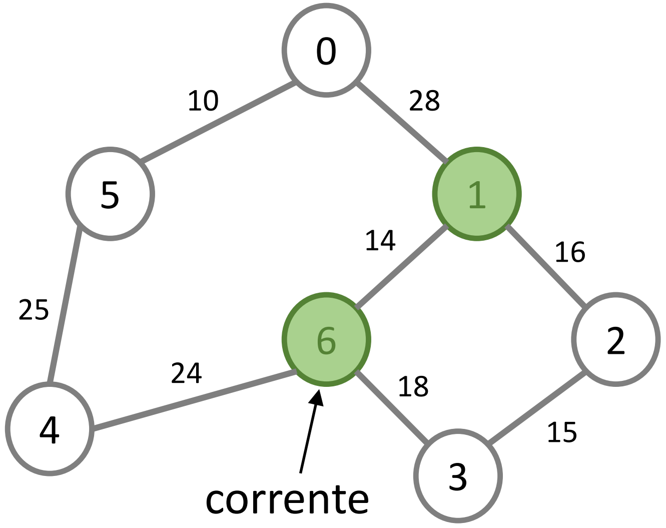
Algoritmo de Dijkstra



3. Marque todos os demais nós como **não visitados** e o nó origem como corrente.

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.

Algoritmo de Dijkstra

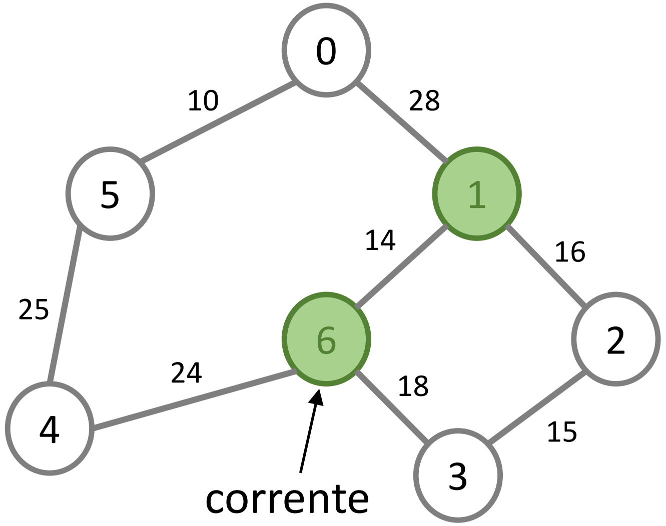


0	28
1	0
2	16
3	∞
4	∞
5	∞
6	14

Menor Distância

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.

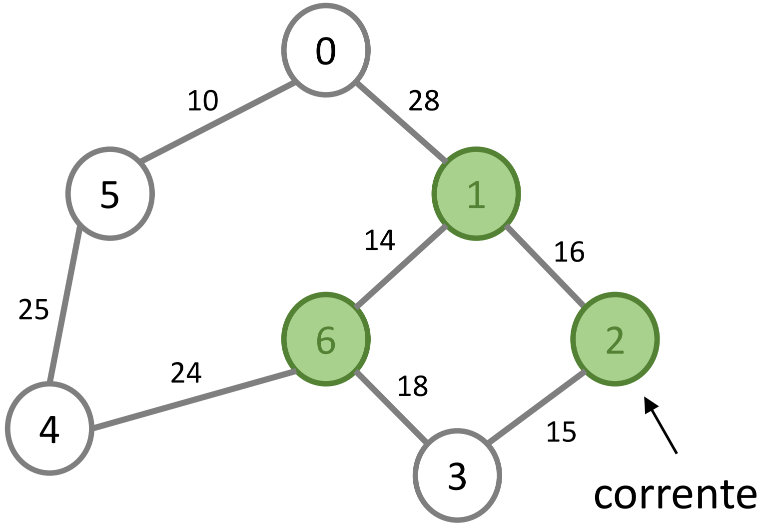
Algoritmo de Dijkstra



0	28	28
1	0	0
2	16	16
3	∞	32
4	∞	38
5	∞	∞
6	14	14

Distâncias para os nós vizinhos do corrente (fronteira)

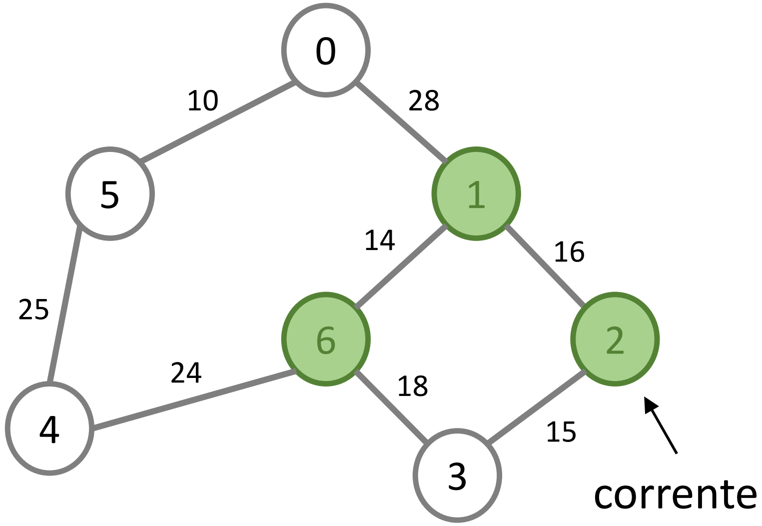
Algoritmo de Dijkstra



0	28
1	0
2	16
3	32
4	38
5	∞
6	14

Menor Distância

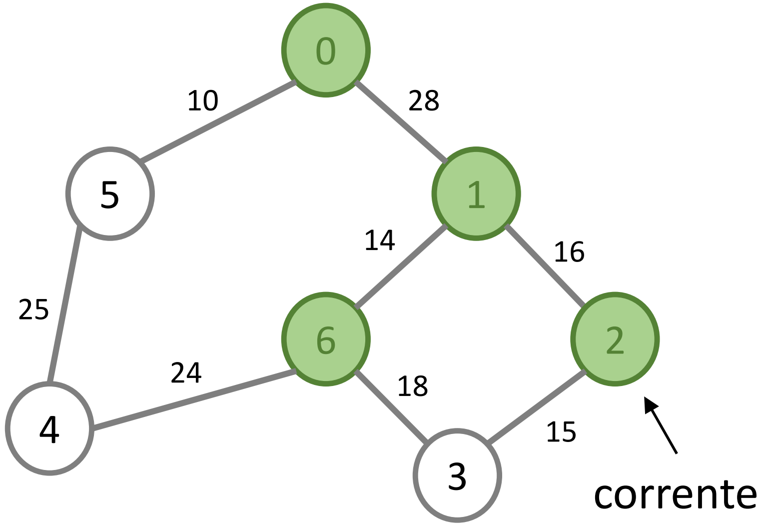
Algoritmo de Dijkstra



0	28		0	28
1	0		1	0
2	16		2	16
3	32		3	31
4	38		4	38
5	∞		5	∞
6	14		6	14

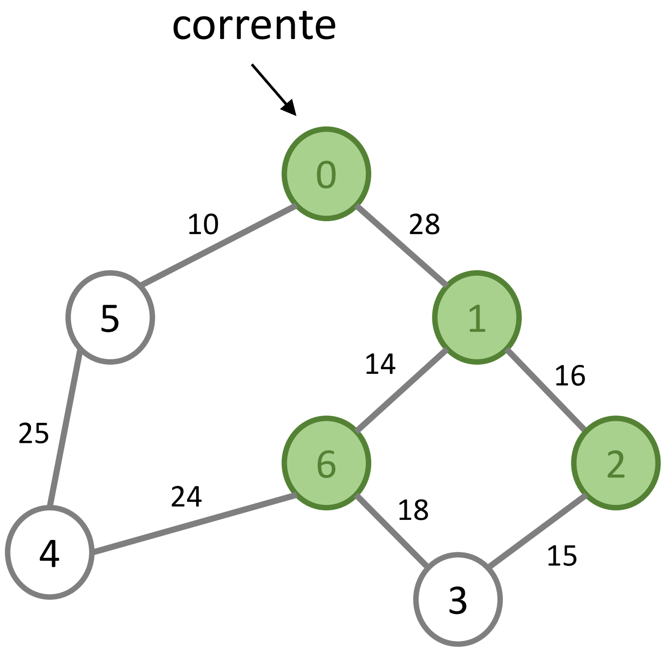
Distâncias para os nós vizinhos do corrente (fronteira)

Algoritmo de Dijkstra



0	28	← Menor Distância
1	0	
2	16	
3	31	
4	38	
5	∞	
6	14	

Algoritmo de Dijkstra

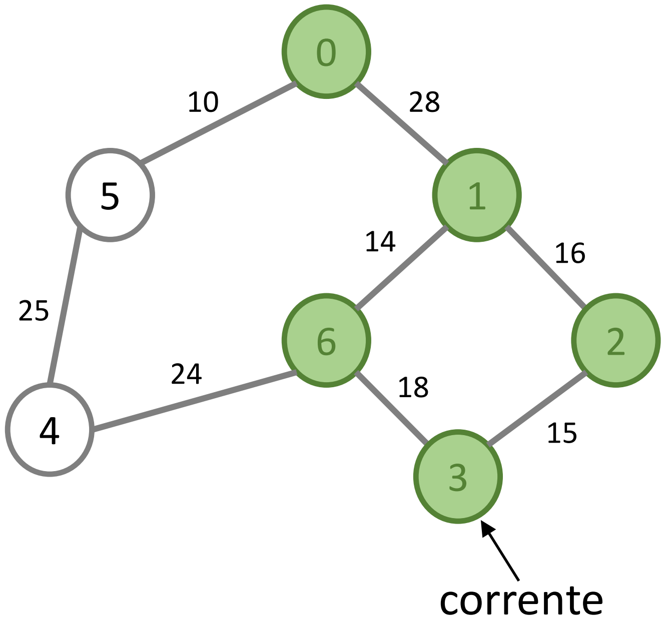


0	28
1	0
2	16
3	31
4	38
5	∞
6	14

0	28
1	0
2	16
3	31
4	38
5	38
6	14

Distâncias para os nós vizinhos do corrente (fronteira)

Algoritmo de Dijkstra

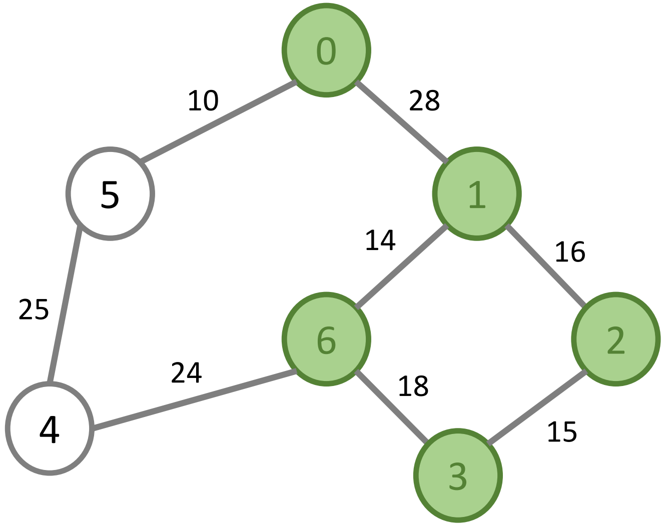


0	28
1	0
2	16
3	31
4	38
5	38
6	14

Menor Distância

não tem vizinhos não visitados!

Algoritmo de Dijkstra



0	28
1	0
2	16
3	31
4	38
5	38
6	14

não há caminhos mais curtos para encontrar

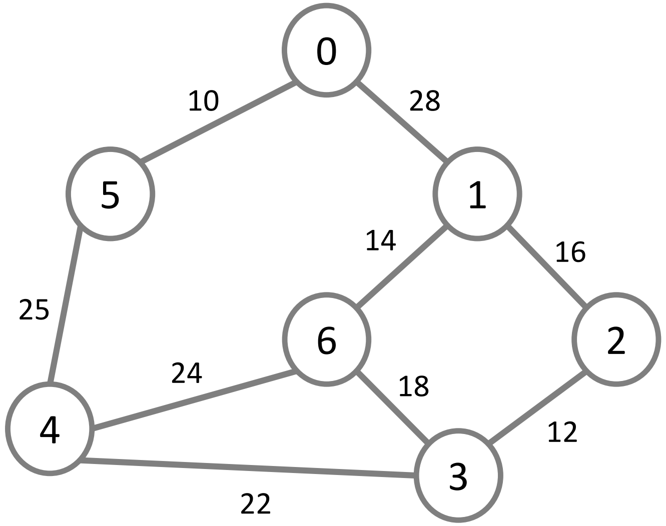
Algoritmo de Dijkstra

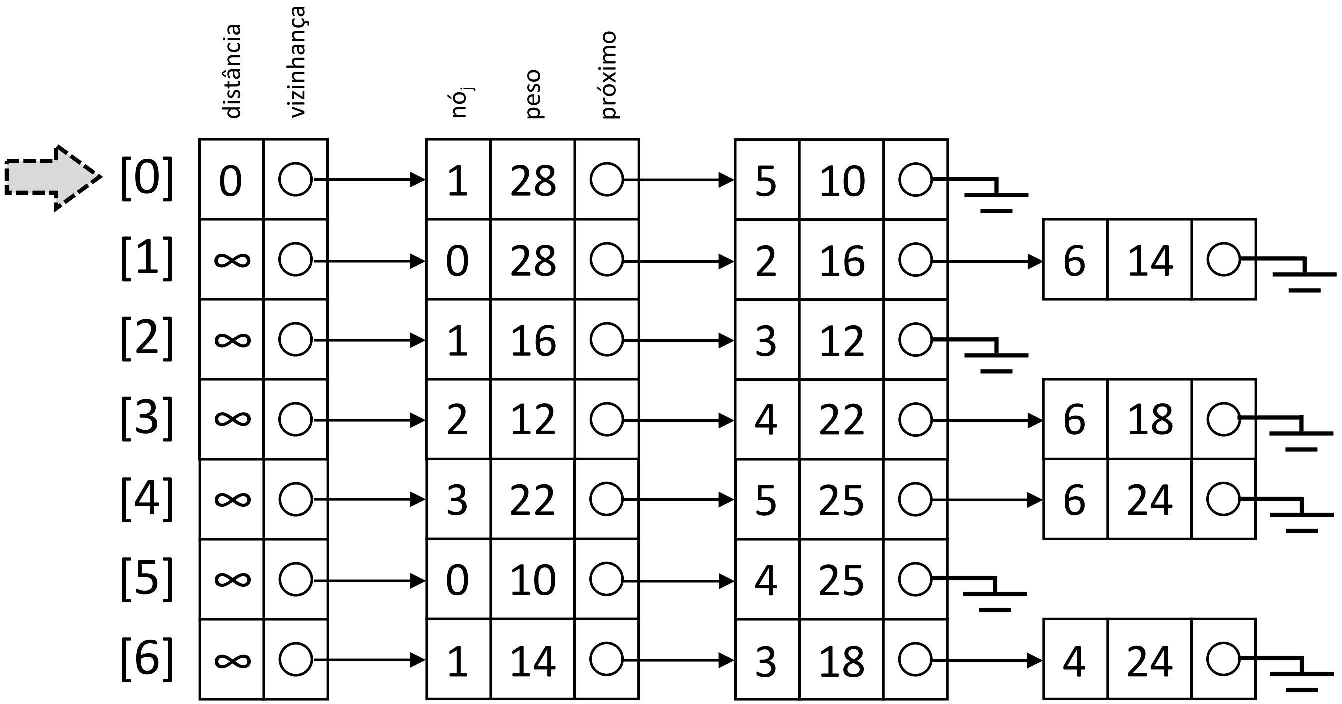
1. Defina o nó de origem
2. Atribua a todos os nós um valor de distância ao nó de origem: valor zero ao nó de origem, e *infinito* para todos os outros nós.
3. Marque todos os demais nós como não visitados e o nó origem como corrente (A).
4. Considere a distância de todos os nós vizinhos não visitados ao nó corrente e calcule uma distância deles ao nó origem através do nó corrente.
   1. 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.
   2. 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 (ao 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

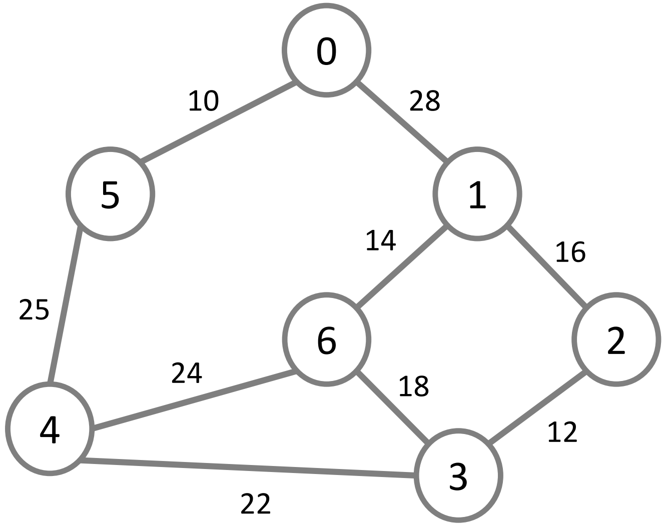
- Vértice Inicial: 0

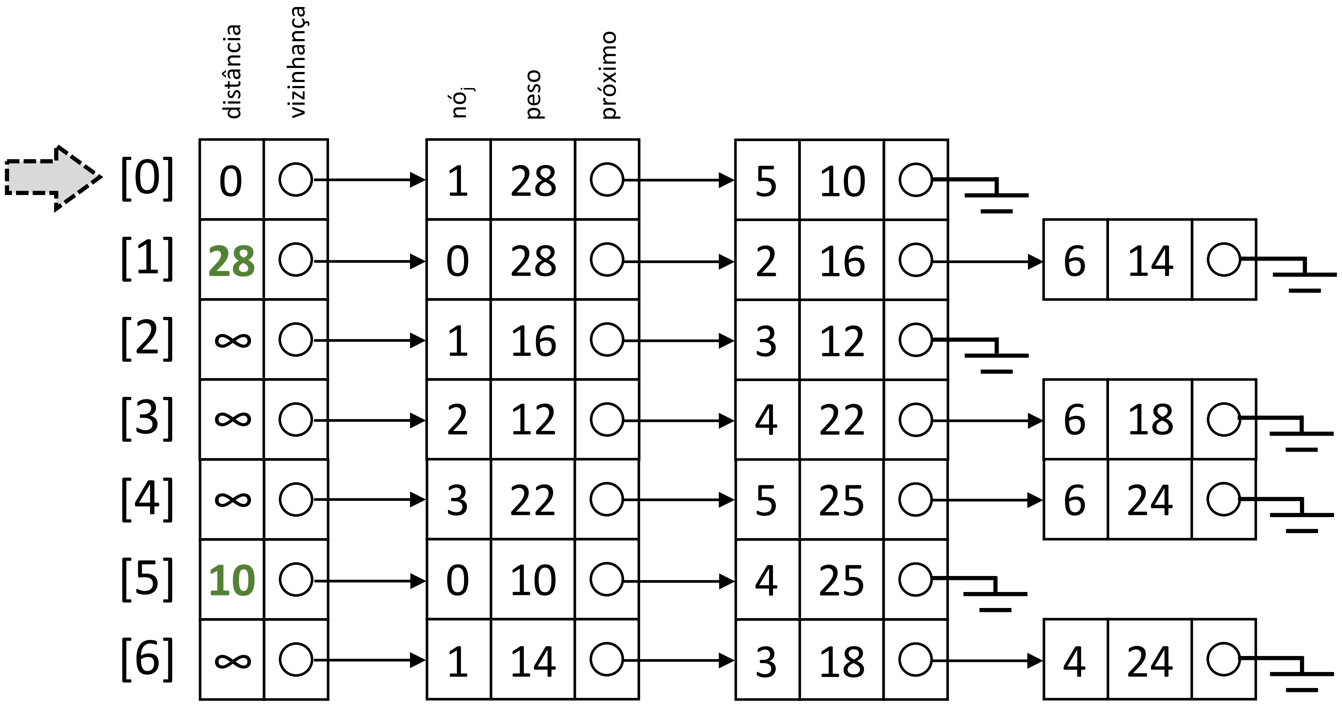




Dijkstra - exemplo

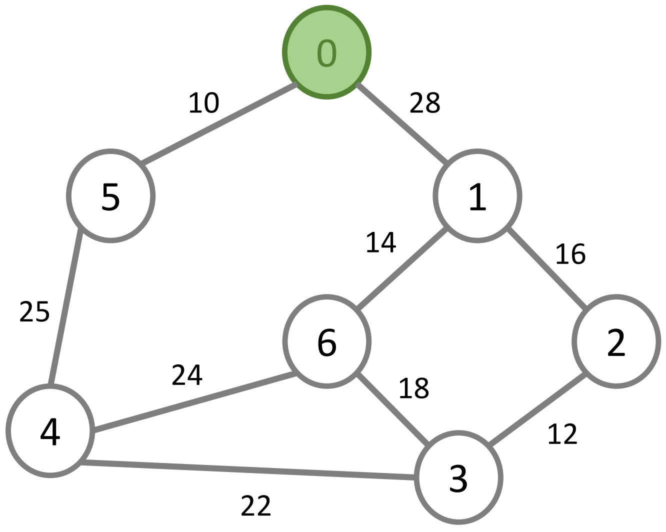
- Modifica distâncias dos vértices 1 e 5

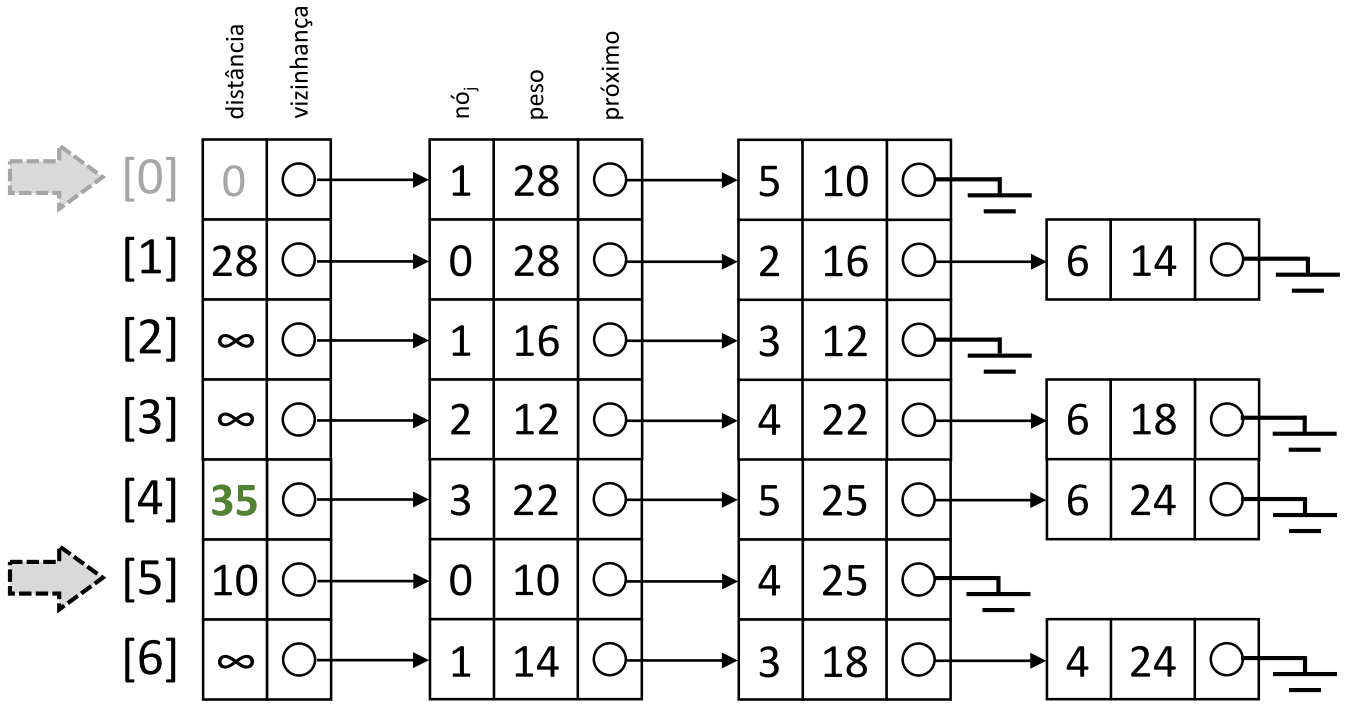




Dijkstra - exemplo

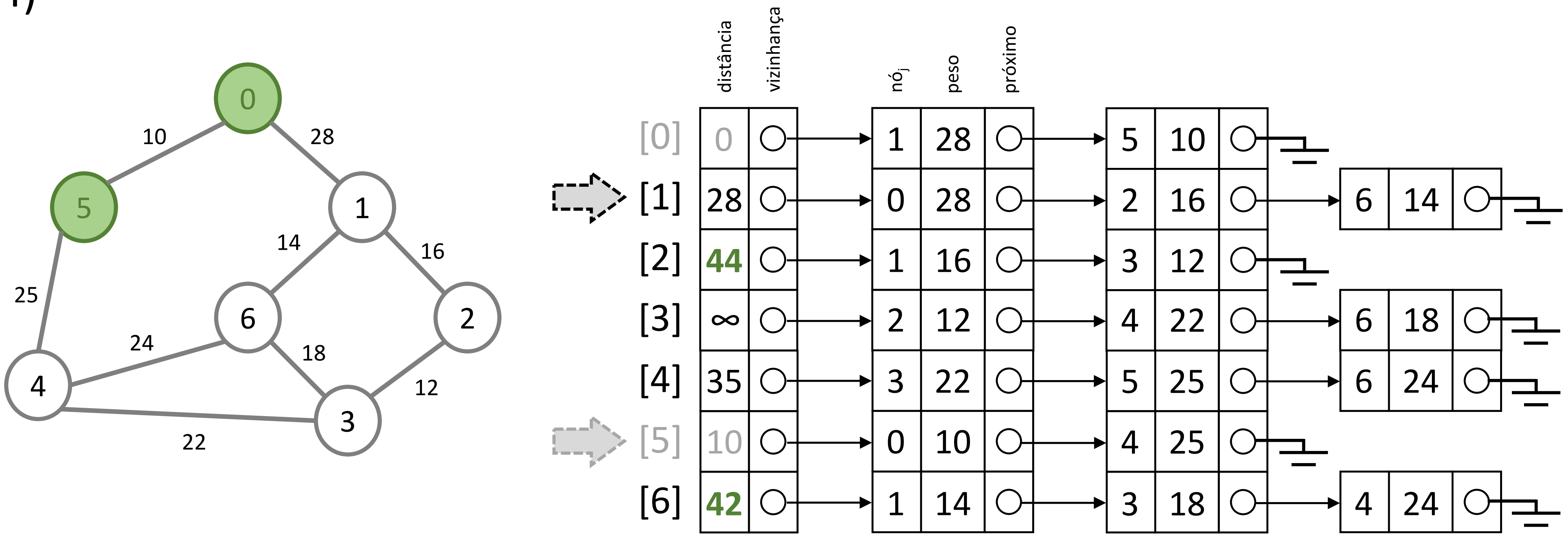
- Marca o vértice 0 como visitado
- Seleciona o vértice 5 (vértice não visitado de menor distância)
- Ignora vértice 0 (já visitado) e modifica distância do vértice 4 (10 + 25)





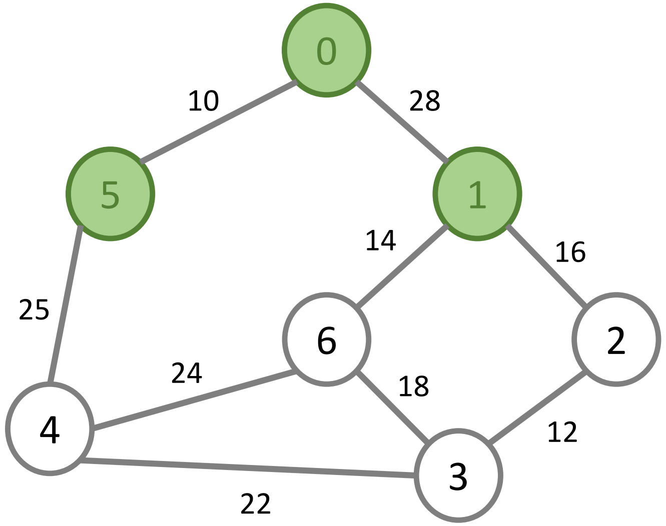
Dijkstra - exemplo

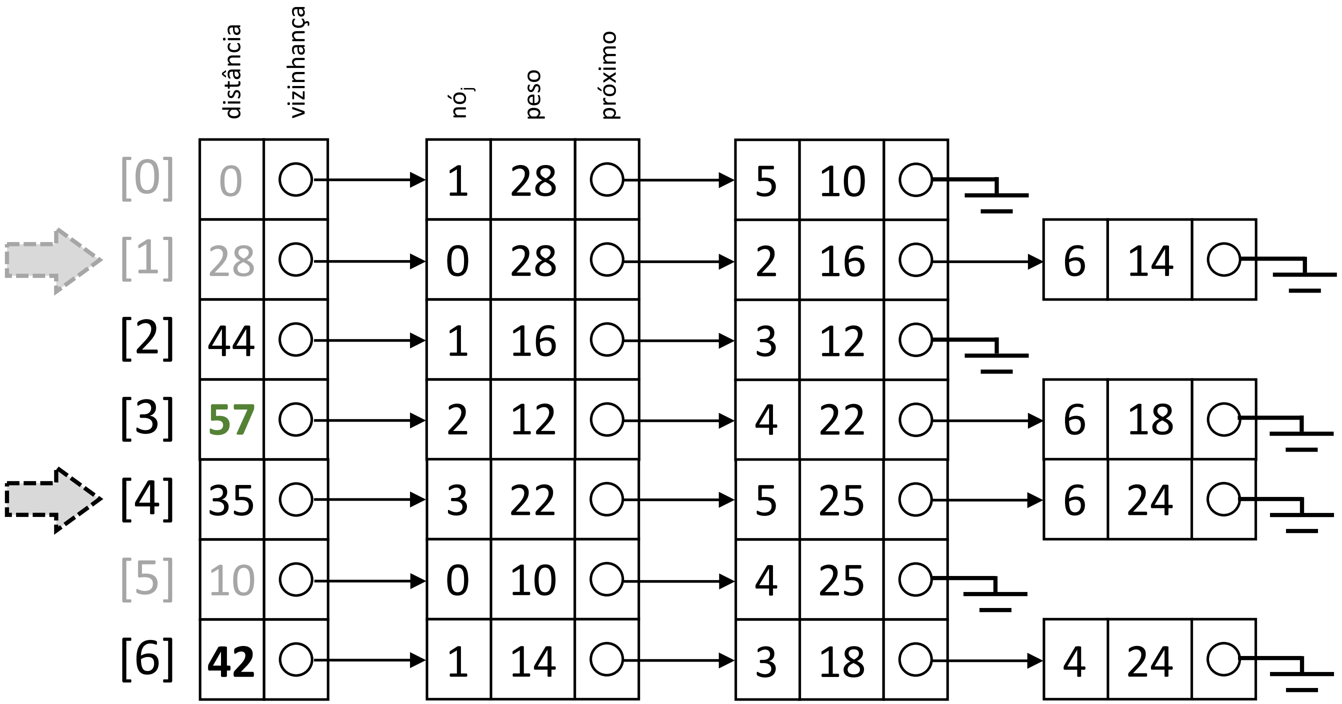
- Marca o vértice 5 como visitado
- Seleciona o vértice 1 (vértice não visitado de menor distância)
- Ignora vértice 0 (já visitado) e modifica distância do vértice 2 (28 + 16) e 6 (28 + 14)



Dijkstra - exemplo

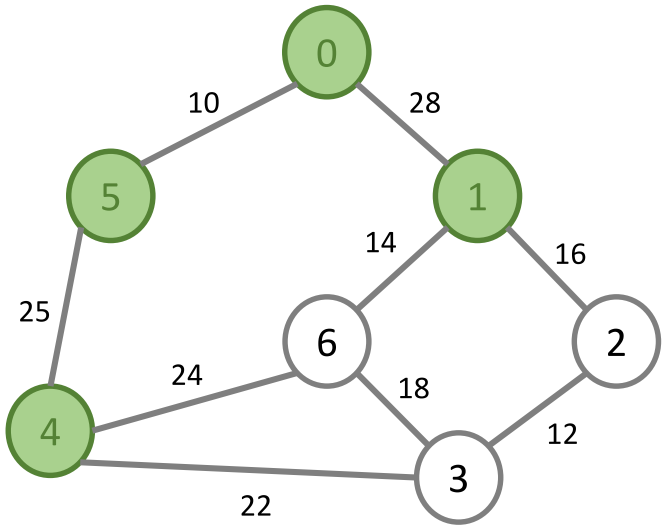
- Marca o vértice 1 como visitado
- Seleciona o vértice 4 (vértice não visitado de menor distância)
- Modifica distância do vértice 3 (35 + 22); ignora vértice 5; mantém a distância do vértice 6 (35 + 24)

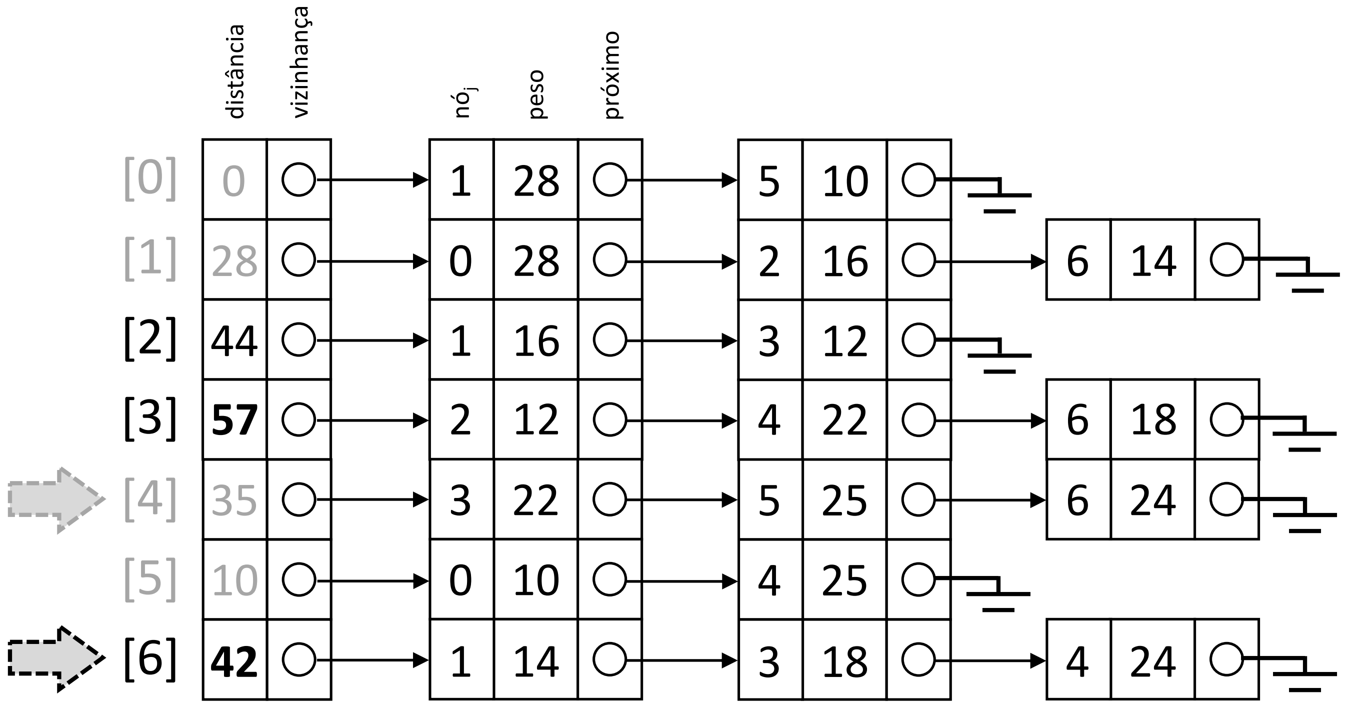




Dijkstra - exemplo

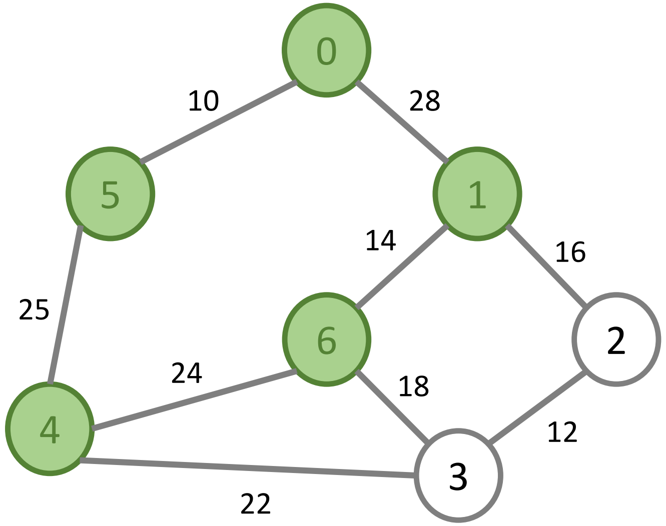
- Marca o vértice 4 como visitado
- Seleciona o vértice 6 (vértice não visitado de menor distância)
- Ignora vértice 1; mantém distância do vértice 3 (42 + 18); ignora vértice 4

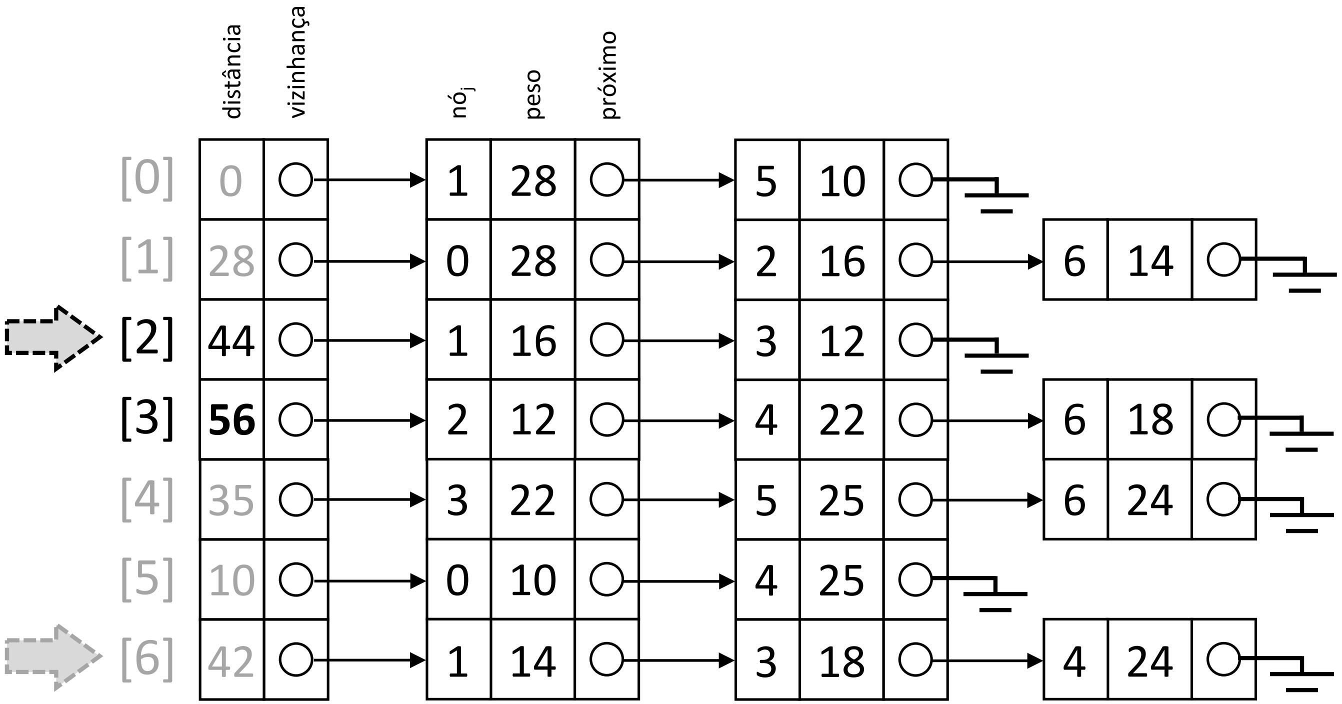




Dijkstra - exemplo

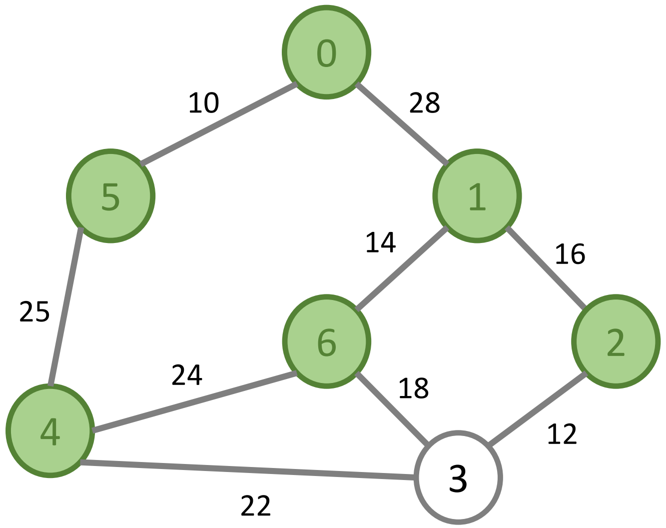
- Marca o vértice 6 como visitado
- Seleciona o vértice 2 (vértice não visitado de menor distância)
- ignora vértice 1; modifica distância do vértice 3 (44 + 12)

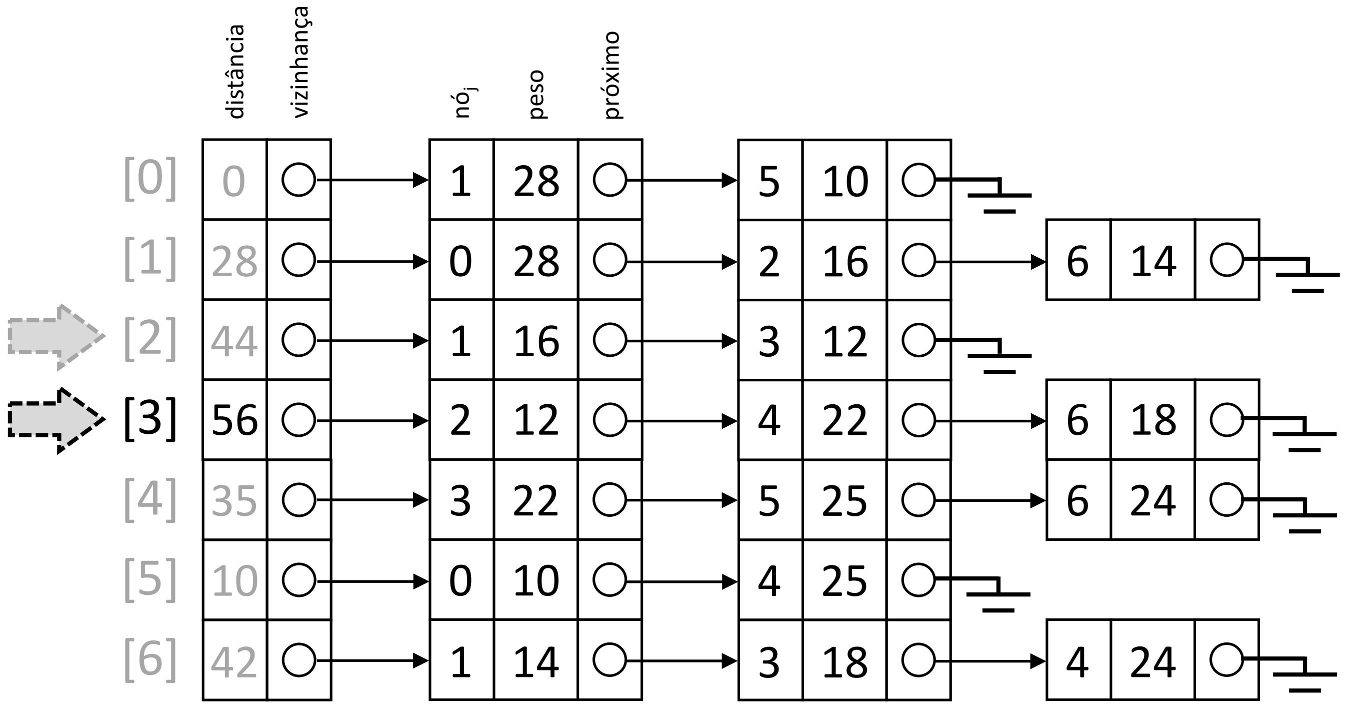




Dijkstra - exemplo

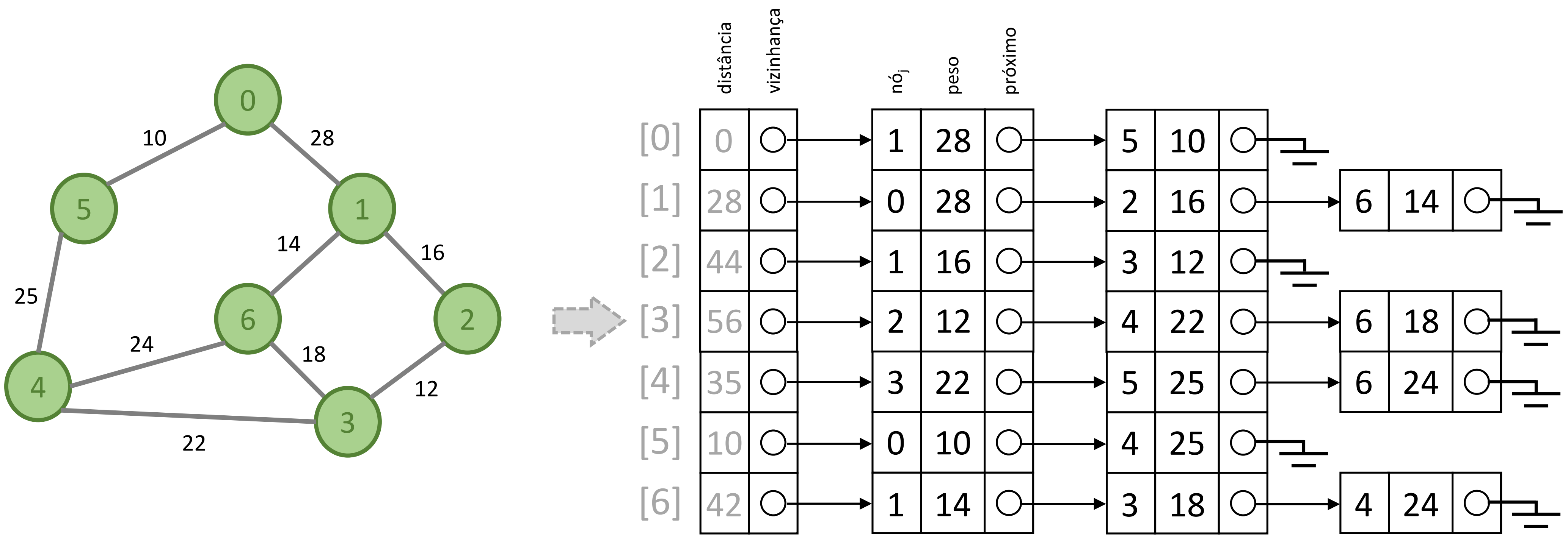
- Marca o vértice 2 como visitado
- seleciona o vértice 3 (vértice não visitado de menor distância)
- Ignora vértices 2, 4 e 6



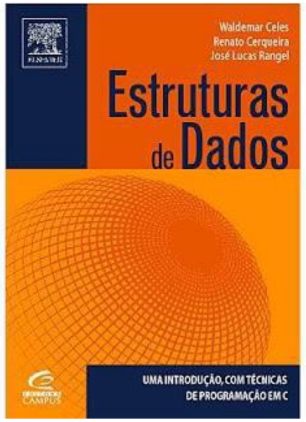


Dijkstra - exemplo

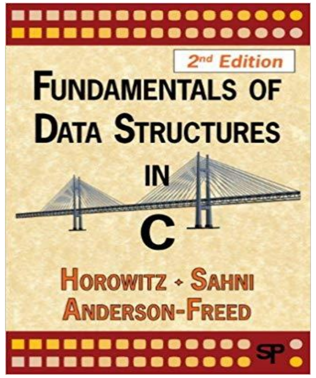
- Marca o vértice 3 como visitado
- Não há mais vértices não visitados – FIM!



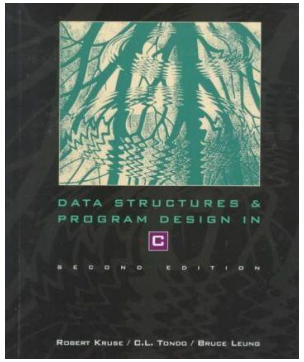
Leitura Complementar



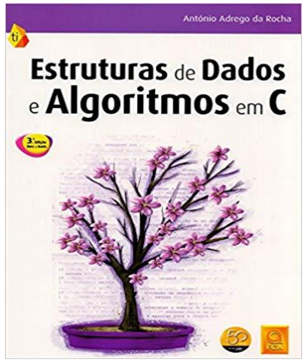
- Celes, W., Cerqueira, R., Rangel, J.L., **Introdução a Estruturas de Dados – Uma introdução com técnicas de programação em C**,
  Ed. Campus, 2004
  - **Capítulo 22 – Grafos; 22.5 – Caminho mínimo**



- Horowitz. E.; Sahni, S.; Anderson-Freed, S. **Fundamentals of Data Structures in C**, 2nd edition. Silicon Press, 2008.
  - Capítulo 6: Graphs; 6.4 – Shortest Paths and Transitive Closure



- Kruse, R.; Tondo, C.; Leung, B.; Mogalla, S.; **Data Structures and Program Design in C**, 2nd edition. Pearson, 1996.
  - Capítulo 11: Graphs; 11.5 – Djkistra’s Algorith: Shortest Paths



- ROCHA, A.; **Estruturas De Dados E Algoritmos Em C**, 3ª Edição, Ed. FCA, 2014
  - Capítulo 10: Grafos; 10.9.2 – Caminho Mais Curto; 10.9.3 – Todos os Pares de Caminhos mais Curtos