Aula 8:
Grafos: Algoritmos de Dijkstra e Tarjan

***Disciplina*: Maratona de Programação 1**

Profs. Edmilson Marmo e Luiz Olmes

edmarmo@unifei.edu.br, olmes@unifei.edu.br

Nas aulas anteriores...

- **O QUE JÁ ESTUDAMOS?**
  - Introdução à Maratona
  - Problemas ad hoc
  - Standard Template Library (STL)
  - Grafos: DFS e BFS

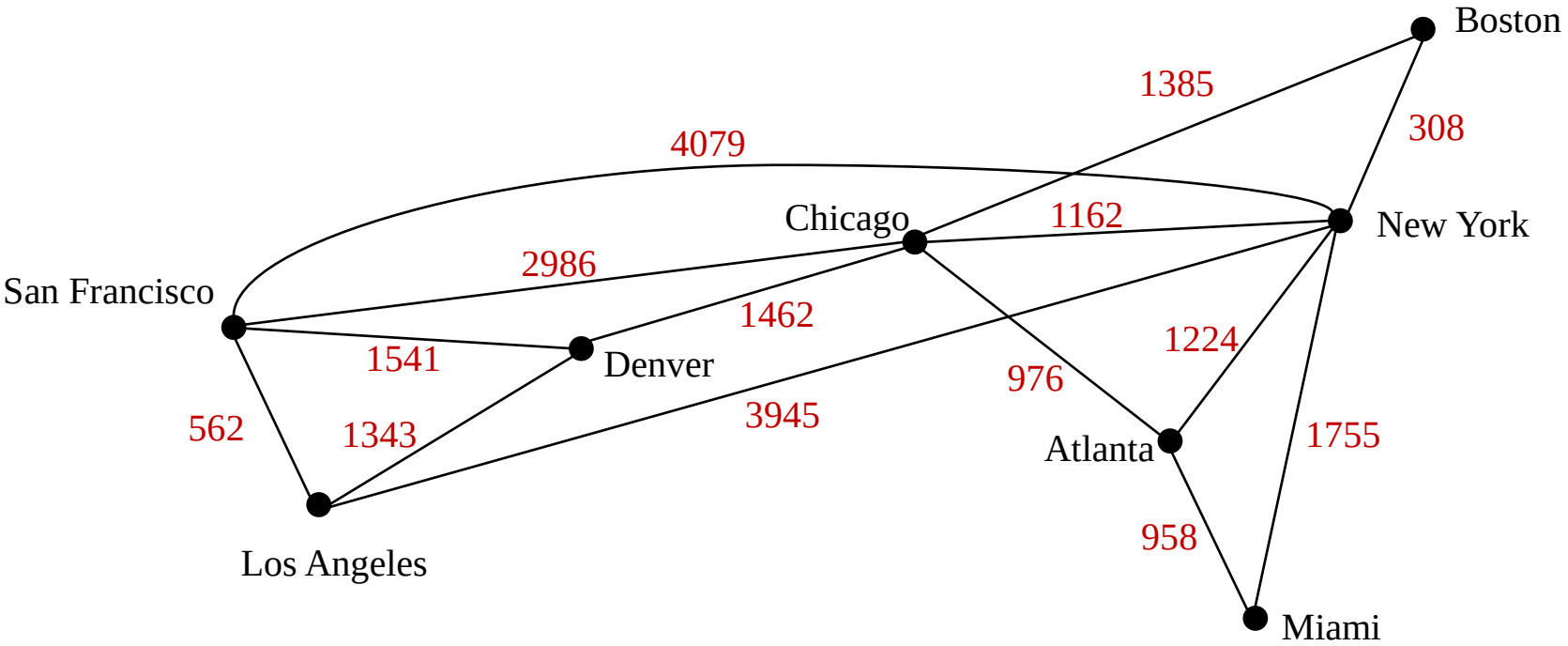
- **OBJETIVOS:**
  - Single-Source Shortest Path:
    - O problema do Caminho Mínimo
    - Algoritmo de Dijkstra
  - Componentes Fortemente Conexas:
    - Algoritmo de Tarjan

Caminho Mínimo

- Uma das primeiras intuições que temos em relação a caminhos mínimos é em relação à distâncias físicas.
  - Aplicativo de navegação: minimização de distâncias.

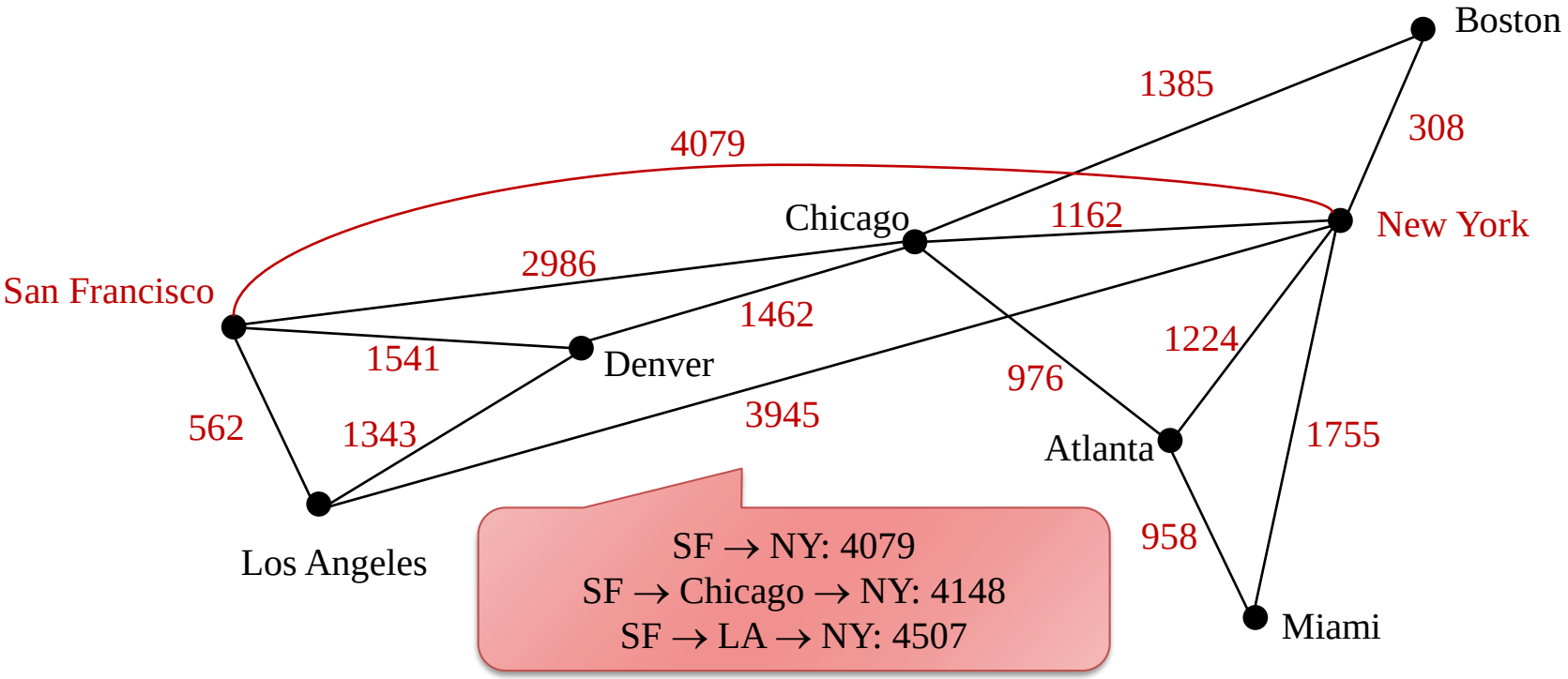
Caminho Mínimo

- Uma das primeiras intuições que temos em relação a caminhos mínimos é em relação à distâncias físicas.



Caminho Mínimo

- Uma das primeiras intuições que temos em relação a caminhos mínimos é em relação à distâncias físicas.

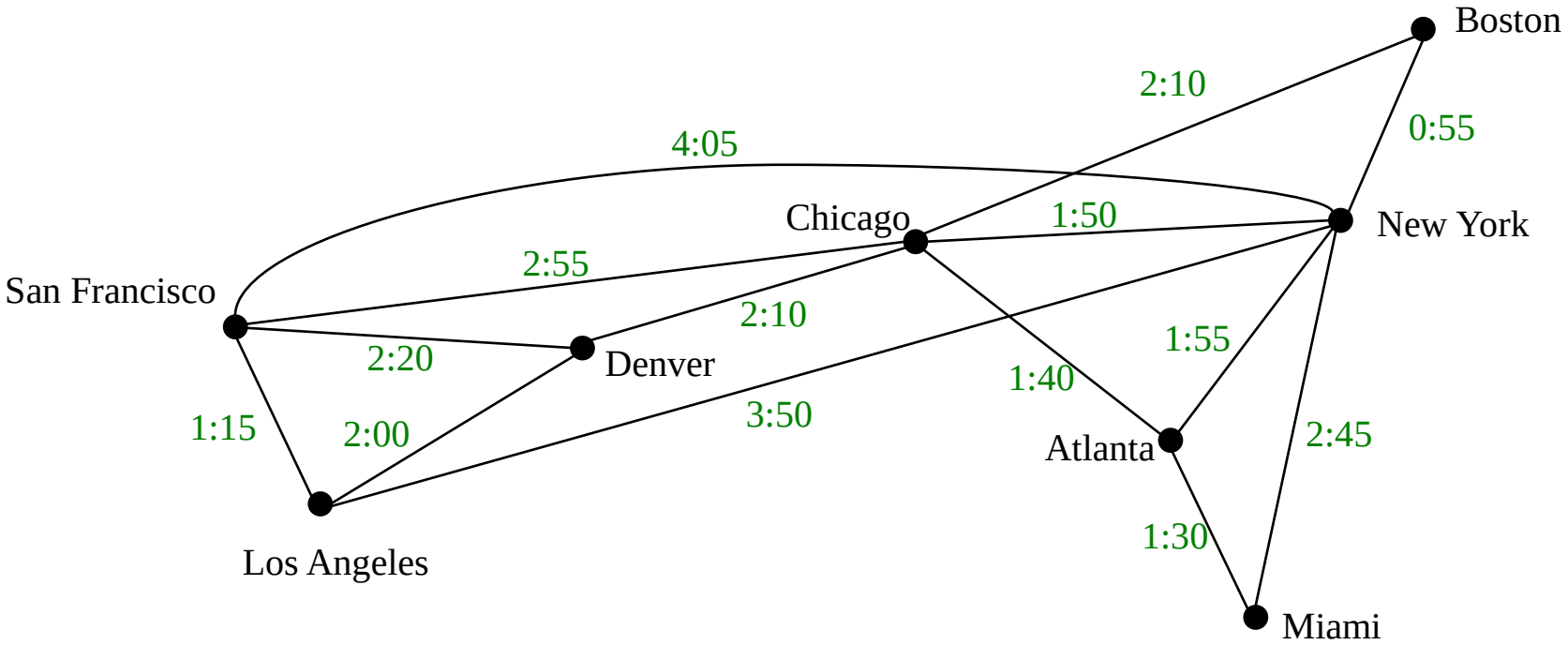


Caminho Mínimo

- Entretanto, um caminho em um grafo pode representar outros aspectos:

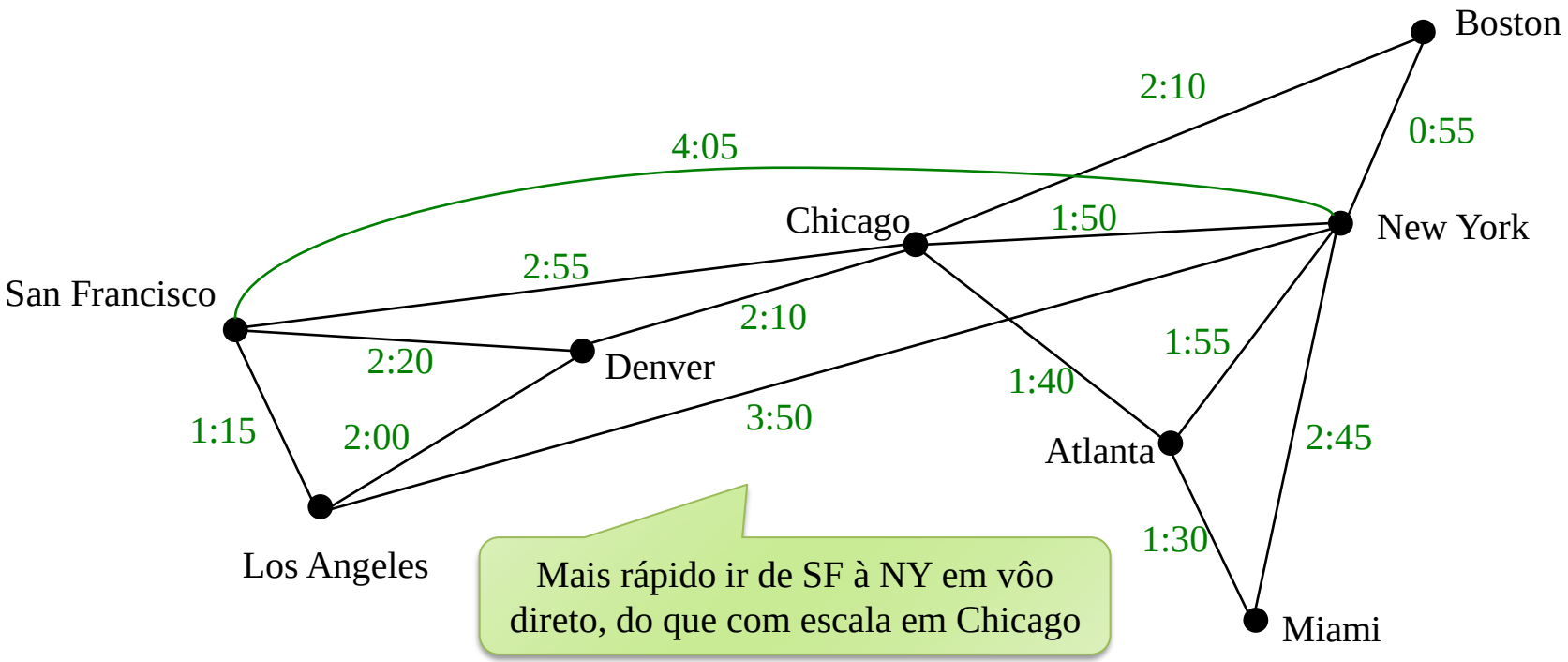
Caminho Mínimo

- Entretanto, um caminho em um grafo pode representar outros aspectos:
  - Tempo de vôo



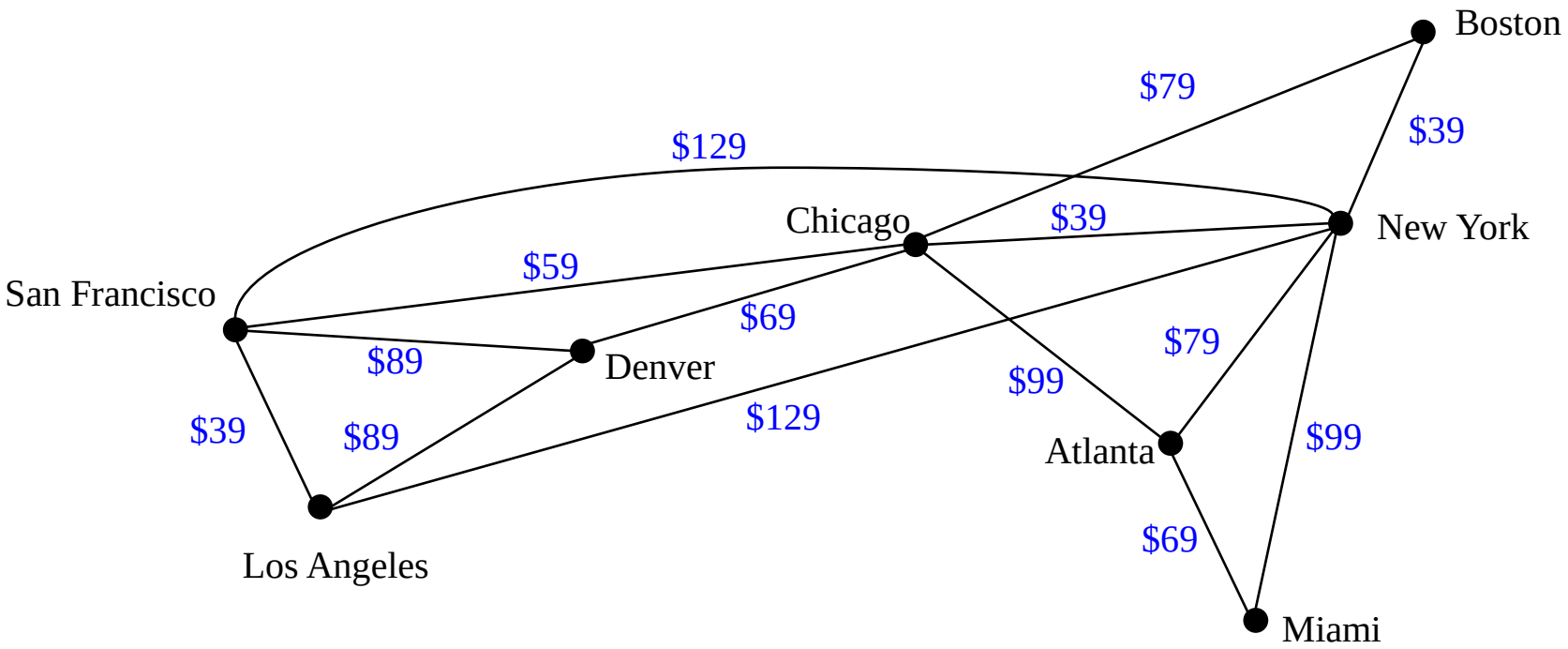
Caminho Mínimo

- Entretanto, um caminho em um grafo pode representar outros aspectos:
  - Tempo de vôo



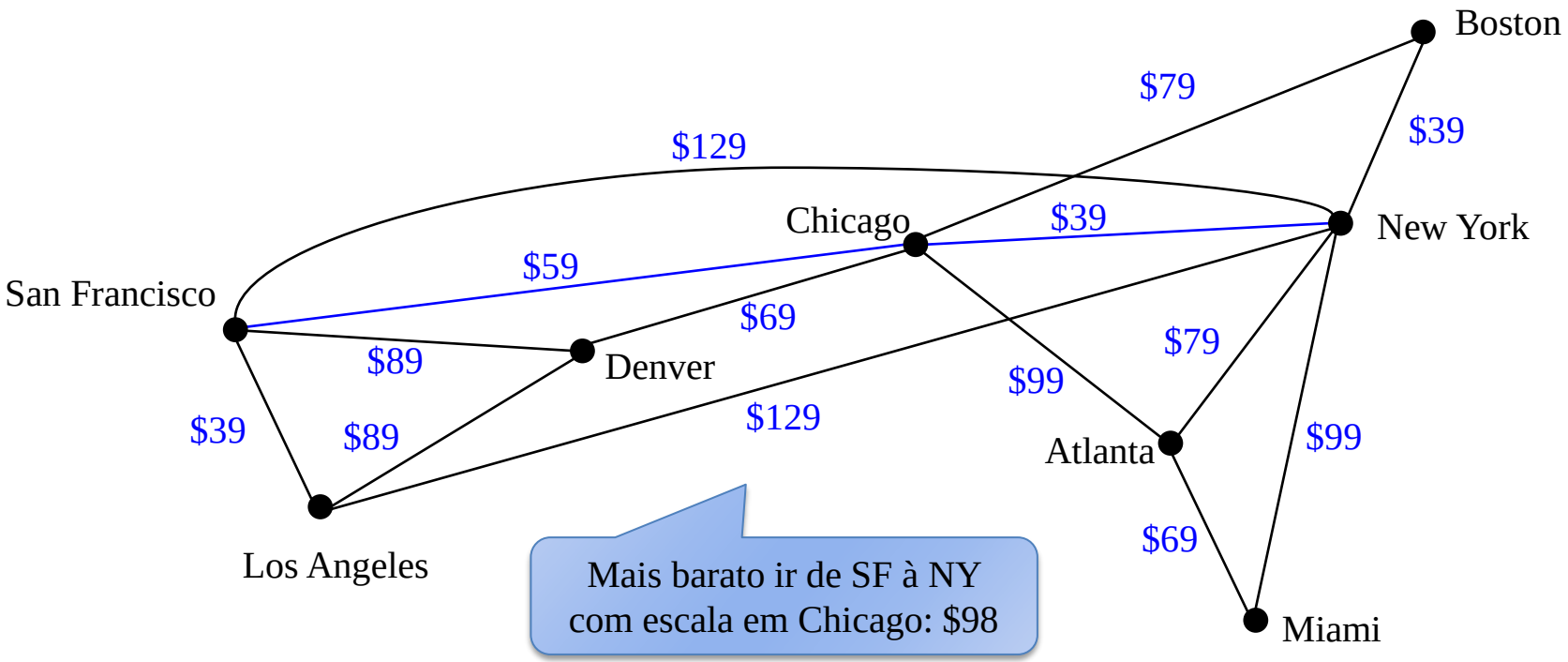
Caminho Mínimo

- Entretanto, um caminho em um grafo pode representar outros aspectos:
  - Preço da passagem



Caminho Mínimo

- Entretanto, um caminho em um grafo pode representar outros aspectos:
  - Preço da passagem



Problema do Caminho Mínimo

- Definição: dado um grafo, o problema do caminho mínimo consiste em encontrar uma rota de um vértice de origem *S* (*source*) até um vértice de destino *T* (*target*) tal que nenhuma outra rota de *S* para *T* tenha custo menor que aquela encontrada.

Problema do Caminho Mínimo

- Este problema se aplica a grafos não ponderados e a grafos ponderados.

Problema do Caminho Mínimo

- Este problema se aplica a grafos não ponderados e a grafos ponderados.

- Grafos não ponderados: neste caso, é suficiente usar uma BFS (Busca em Largura – *aula passada*), ao custo de O(V + A), para encontrar um caminho mínimo entre dois vértices.
  - Problema: Duende Perdido (Beecrowd: 2294)

Problema do Caminho Mínimo

- Este problema se aplica a grafos não ponderados e a grafos ponderados.

- Grafos não ponderados: neste caso, é suficiente usar uma BFS (Busca em Largura – *aula passada*), ao custo de O(V + A), para encontrar um caminho mínimo entre dois vértices.
  - Problema: Duende Perdido (Beecrowd: 2294)

Problema do Caminho Mínimo

- Este problema se aplica a grafos não ponderados e a grafos ponderados.

- Grafos não ponderados: neste caso, é suficiente usar uma BFS (Busca em Largura – *aula passada*), ao custo de O(V + A), para encontrar um caminho mínimo entre dois vértices.
  - Problema: Duende Perdido (Beecrowd: 2294)

- Grafos ponderados: quando as arestas possuem pesos, uma BFS não executa corretamente. Para grafos ponderados, tem-se as duas seguintes variantes do problema:
  - Single-Source Shortest Path
  - All-Pairs Shortest Path.

Problema do Caminho Mínimo

- Single-Source Shortest Path (SSSP): caminho mínimo de um vértice de origem S...
  - ... para um vértice de destino T (S $\rightarrow$ T), ou...
  - ... para todos os outros vértices do grafo (S $\rightarrow$ A, S $\rightarrow$ B, S $\rightarrow$ C, S $\rightarrow$ D, etc.).
  - O problema do SSSP pode ser resolvido através de algoritmos como Dijkstra e Bellman-Ford.

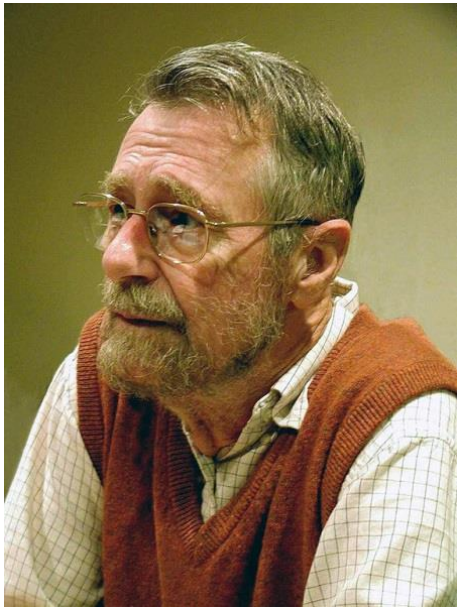
Problema do Caminho Mínimo

- Single-Source Shortest Path (SSSP): caminho mínimo de um vértice de origem S...
  - ... para um vértice de destino T (S → T), ou...
  - ... para todos os outros vértices do grafo (S → A, S → B, S → C, S → D, etc.).
  - O problema do SSSP pode ser resolvido através de algoritmos como Dijkstra e Bellman-Ford.

- All-Pairs Shortest Path (APSP): caminhos mínimos de todos os vértices para todos os outros vértices:
  - A → B, A → C, A → D, ..., B → A, B → C, B → D, etc.
  - O problema do APSP é resolvido através do algoritmo de Floyd-Warshall.

Algoritmo de Dijkstra

- Proposto pelo cientista da computação holandês Edsger Wybe Dijkstra, em 1959.
  - Pronúncia aproximada em português: *déikstra*
- É um dos algoritmos para cálculo de caminho mínimo.
- Aplicável a grafos direcionados e não direcionados.
- Algoritmo restrito a grafos ponderados, com arestas de pesos não-negativos.



☆ 11/05/1930
✞ 06/08/2002

Algoritmo de Dijkstra

- O algoritmo de Dijkstra inicia o processamento com a condição padrão de todo algoritmo SSSP:
  - Dado um vértice de origem S, a distância até S é zero: `dist[s] = 0`
  - A distância para todos os outros vértices U do grafo é infinita: $\forall$`u, dist[u] =` ∞
- A estrutura de fila de prioridade é usada para armazenar cada vértice e a sua distância no caminho: `(dist[u], u)`
  - Útil para manter uma ordenação por distâncias.
  - Em C++: a estrutura `priority_queue` devolve o elemento de maior valor. Como queremos a menor distância, basta inseri-la negativa na fila.
- A cada iteração, o algoritmo processa o vértice inserido na fila de prioridade que possui o menor valor de distância: `u`

Algoritmo de Dijkstra

- Dado o vértice `u`, que no início corresponde ao vértice de origem, pois este possui a menor distância (`dist[s] = 0`), para cada vértice `v` vizinho de `u`, o algoritmo aplica a chamada condição de relaxamento do vértice:

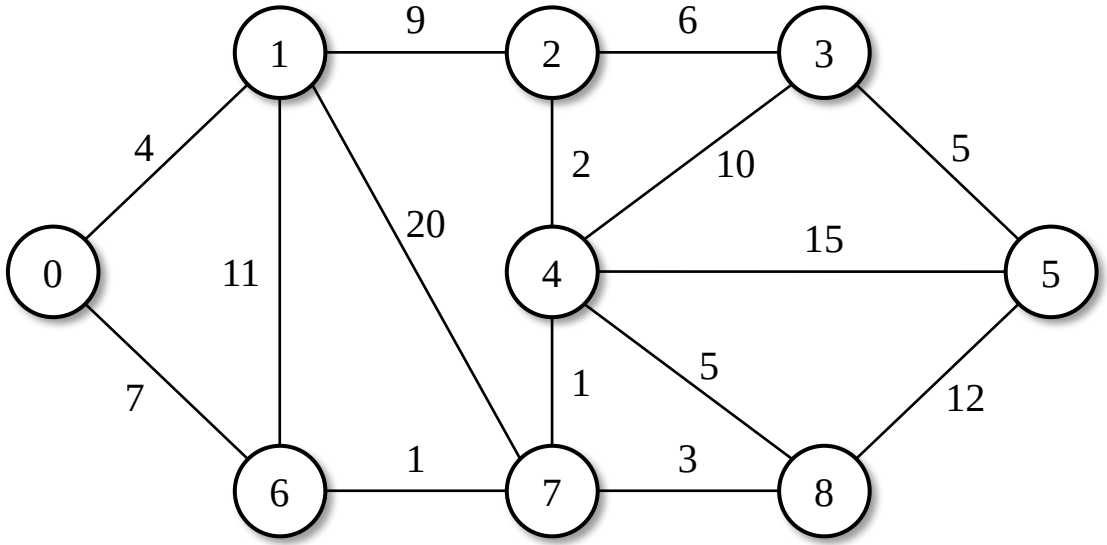
- $dist[v] = min(dist[v], dist[u] + w_{uv})$

- Que pode ser interpretada como: "*A distância da origem até o vértice `v` é o mínimo entre...*"
  - O próprio valor da distância até `v`.
  - A distância da origem até `v`, passando por `u`.

- Esta atualização permite encontrar as menores distâncias no caminho.
  - As informações na fila de prioridades são atualizadas.
  - O predecessor do vértice também é atualizado.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.

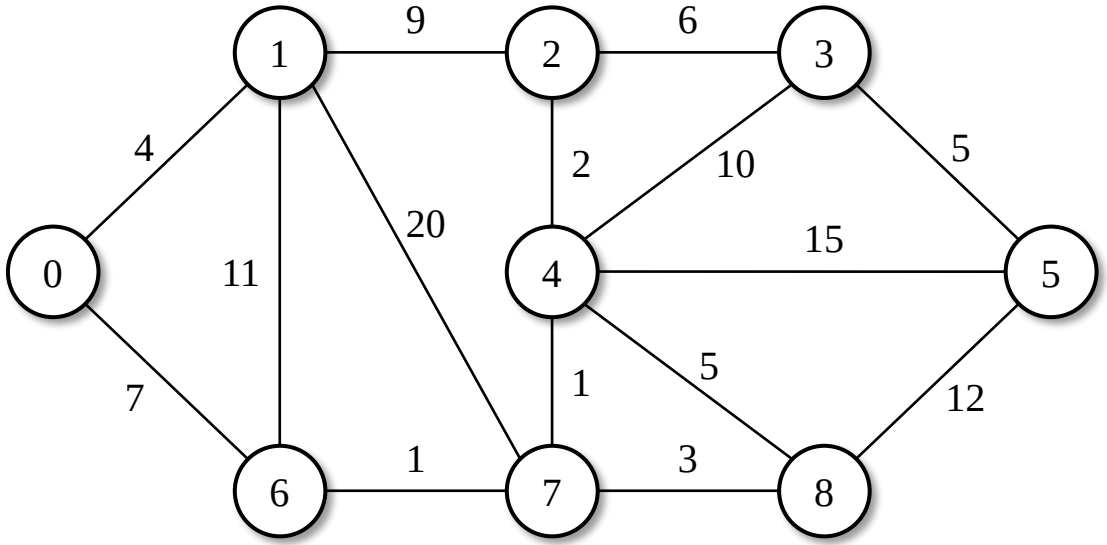


Nó	Dist	Pred
0		
1		
2		
3		
4		
5		
6		
7		
8		

Fila Prio
d[u] (u)

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



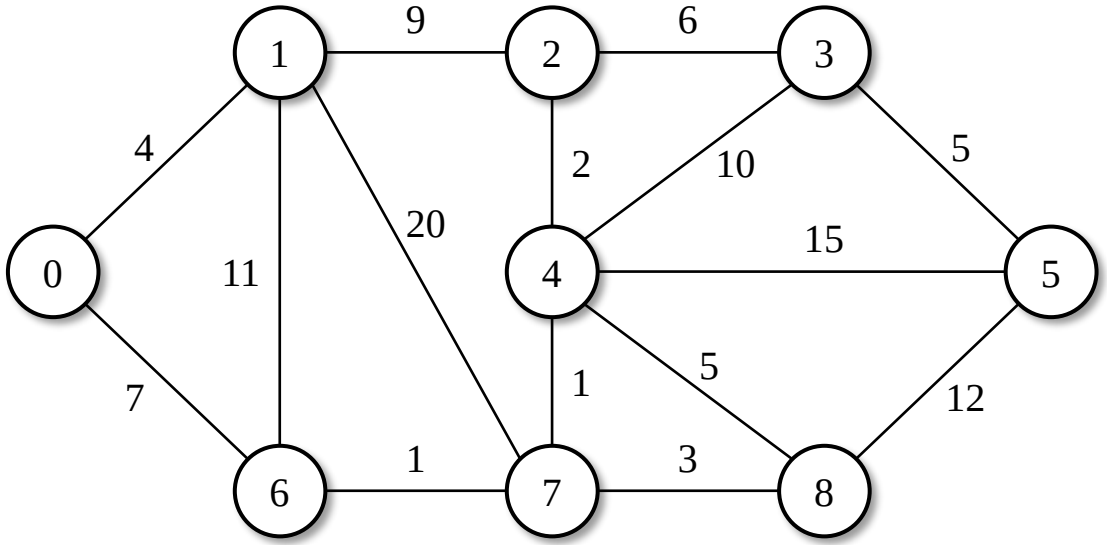
Nó	Dist	Pred
0	inf	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)

Inicialização do algoritmo: as distâncias para todos os vértices são marcadas como infinitas.
Os predecessores de todos de todos os vértices são inexistentes.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



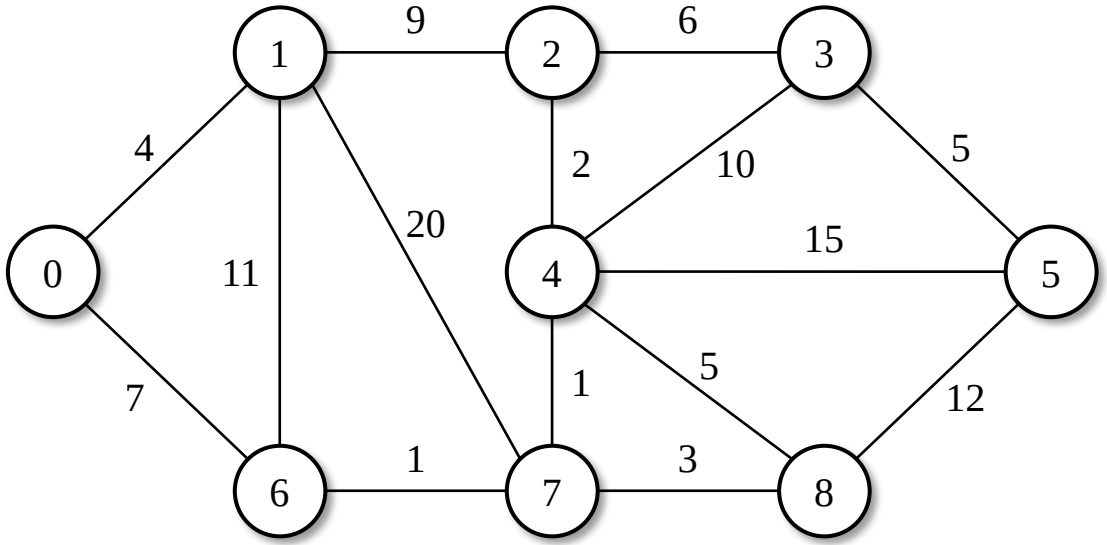
Nó	Dist	Pred
0	0	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)
0 (0)

Definição da origem: a distância do vértice de origem é marcada como zero.
O vértice de origem e sua distância são inseridos na fila de prioridade (prioridade = menor distância).

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



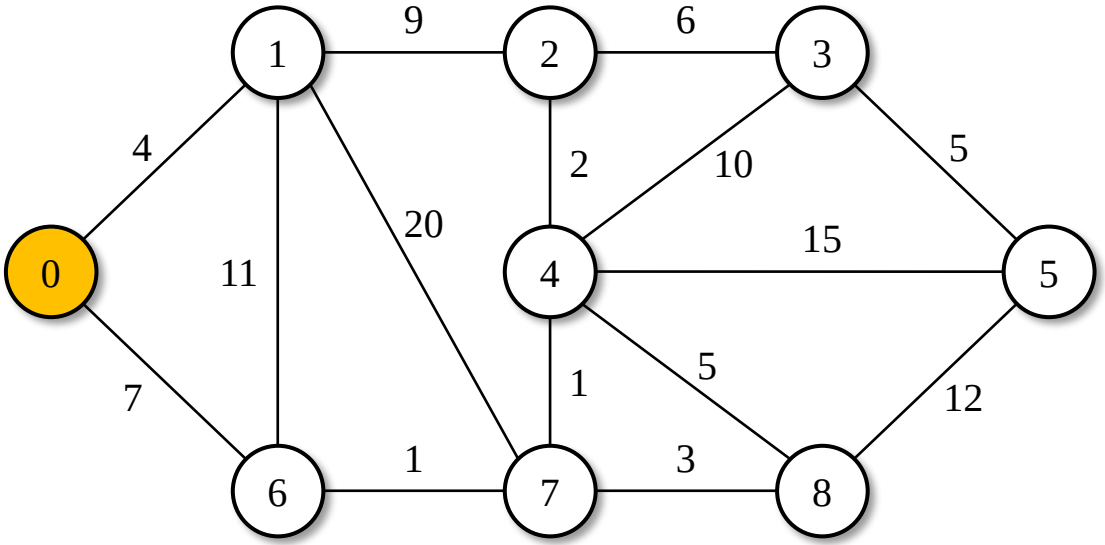
Nó	Dist	Pred
0	0	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)
0 (0)

Loop do algoritmo: enquanto a fila não estiver vazia, remove um elemento, marca ele como visitado e processa os seus vizinhos ainda não visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



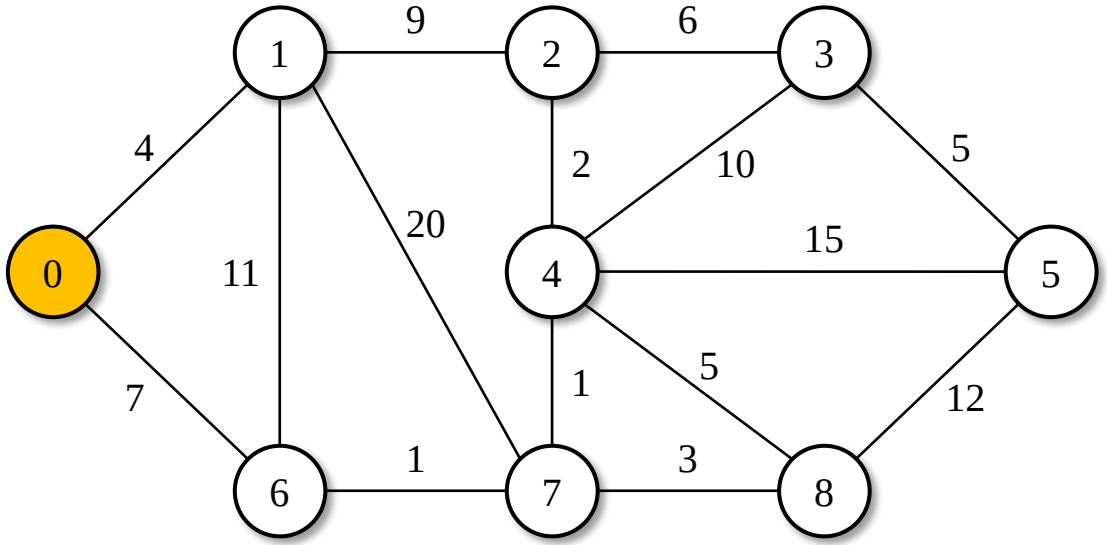
Nó	Dist	Pred
0	0	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)

Loop do algoritmo: remove da fila o vértice 0, com distância 0, e o marca como visitado. (visitado = vermelho na tabela)

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



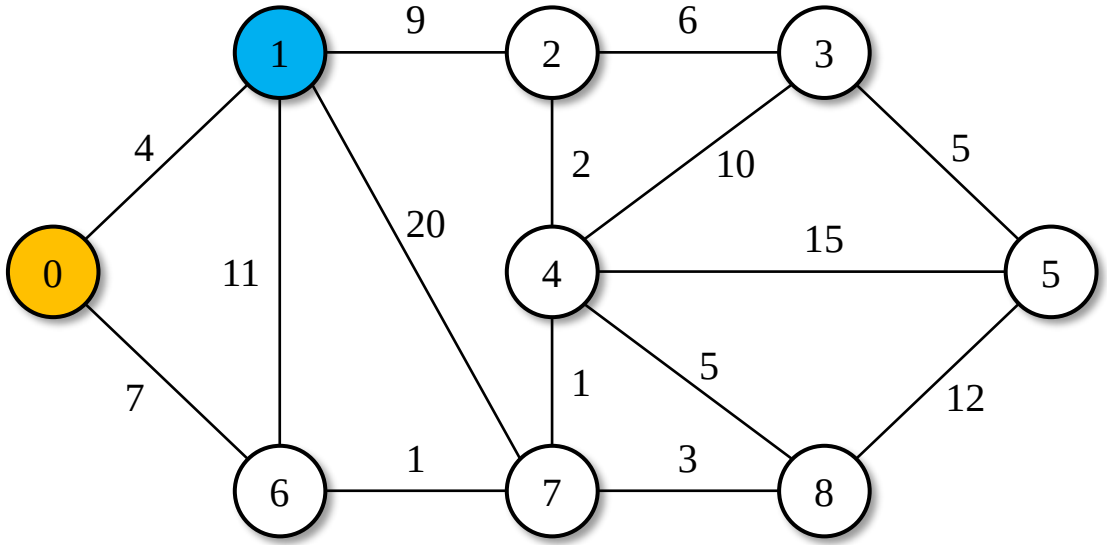
Nó	Dist	Pred
0	0	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)

Análise dos vizinhos: para cada vizinho de 0, verifica a condição de relaxamento.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

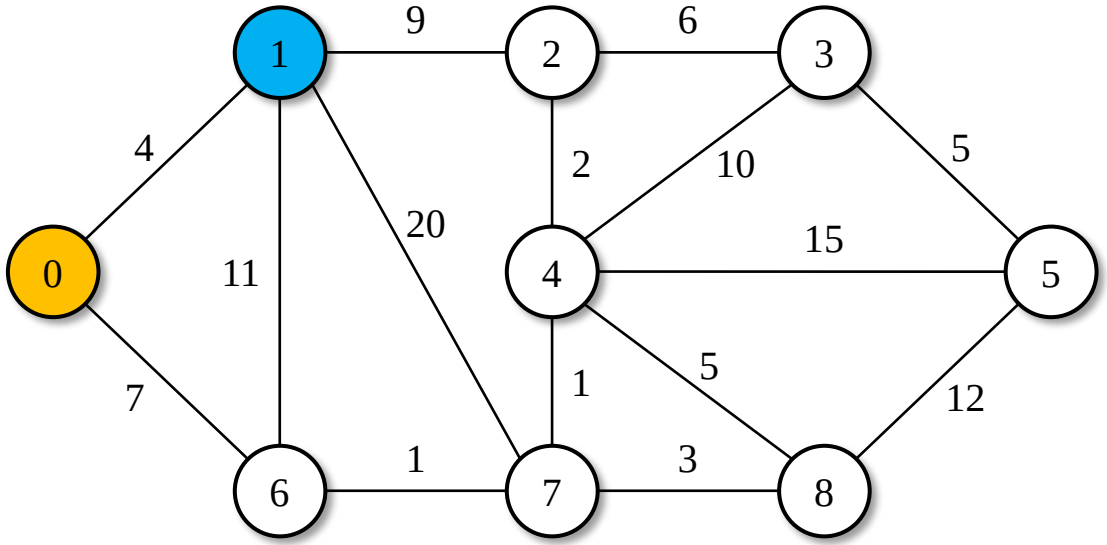
Fila Prio
d[u] (u)

Análise dos vizinhos: para o vértice 1: `dist[0] + dist[0][1] < dist[1]`?

`0 + 4 < inf`?

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	inf	-1
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

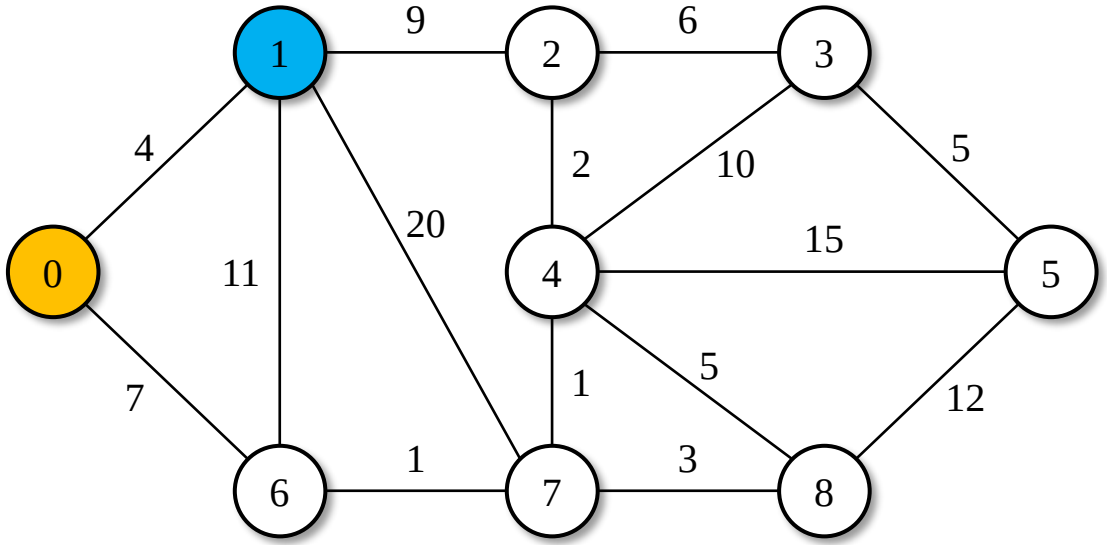
Fila Prio
d[u] (u)

Análise dos vizinhos: para o vértice 1: `dist[0] + dist[0][1] < dist[1]`?

`0 + 4 < inf`? Sim! Recalcula a distância para 1 e o insere na fila.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

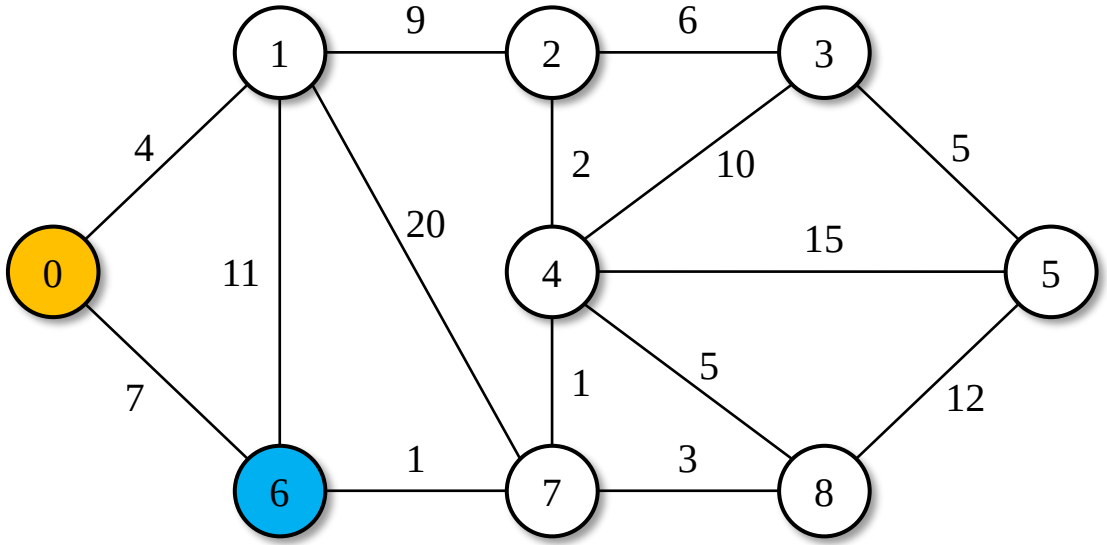
Fila Prio
d[u] (u)
4 (1)

Análise dos vizinhos: para o vértice 1:

```
dist[1] = min(dist[1], dist[0] + dist[0][1])
dist[1] = min(inf, 0 + 4)
dist[1] = 4
```

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

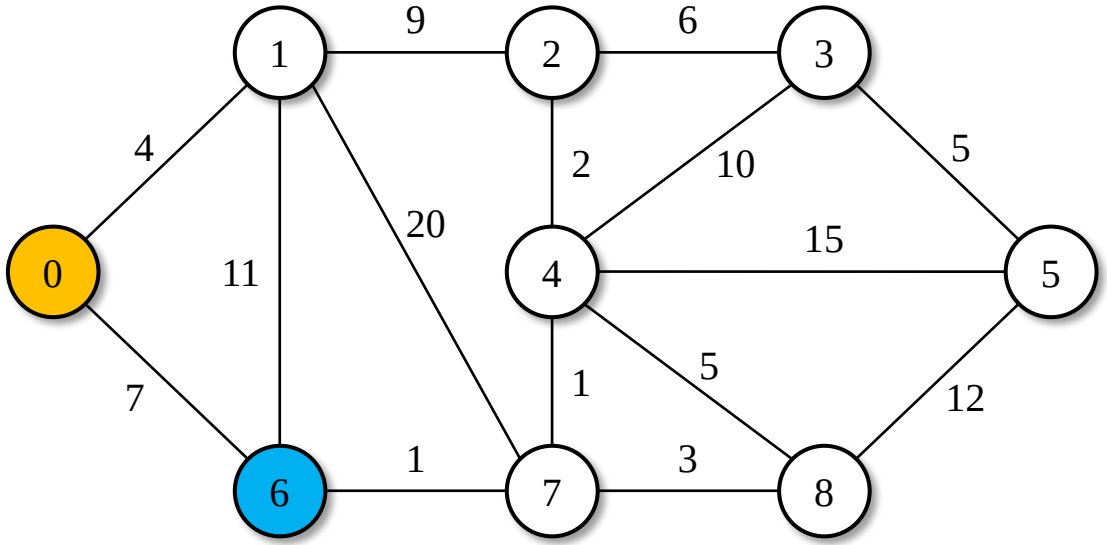
Fila Prio
d[u] (u)
4 (1)

Análise dos vizinhos: para o vértice 6: `dist[0] + dist[0][6] < dist[6]`?

`0 + 7 < inf`?

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	inf	-1
7	inf	-1
8	inf	-1

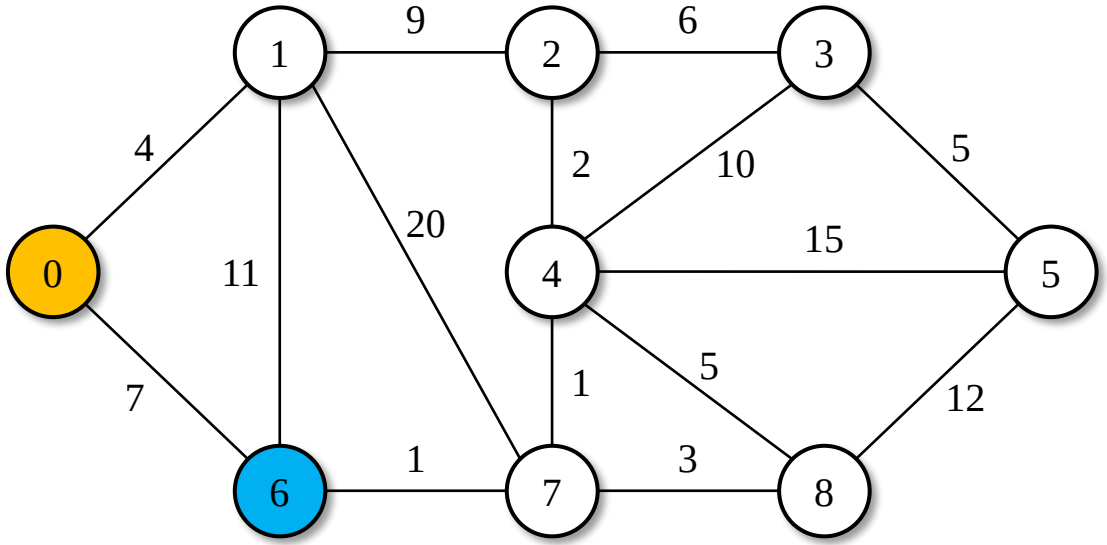
Fila Prio
d[u] (u)
4 (1)

Análise dos vizinhos: para o vértice 6: `dist[0] + dist[0][6] < dist[6]`?

`0 + 7 < inf`? Sim! Recalcula a distância para 6 e o insere na fila.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	inf	-1
8	inf	-1

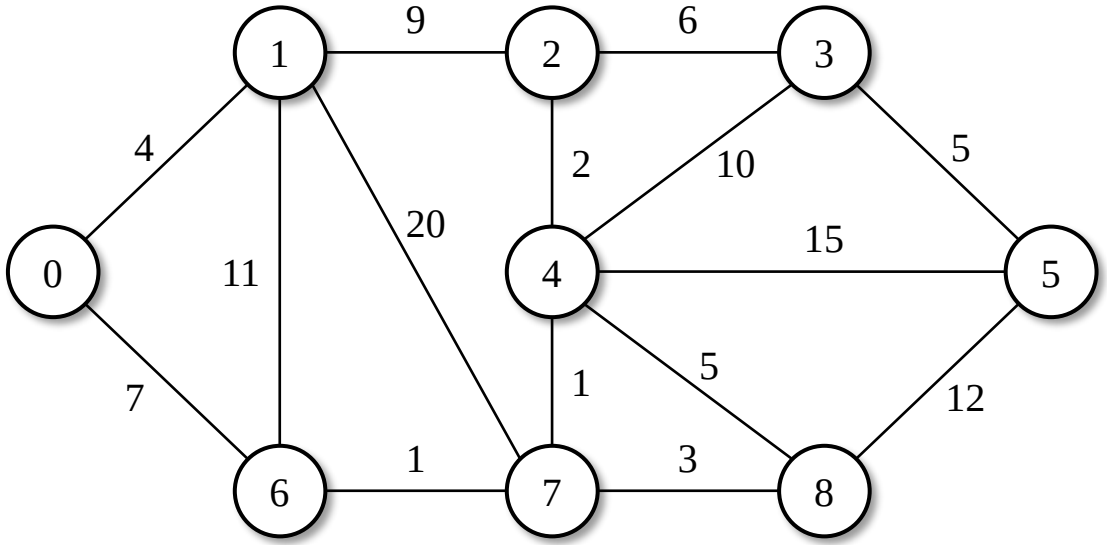
Fila Prio
d[u] (u)
4 (1)
7 (6)

Análise dos vizinhos: para o vértice 6:

```
dist[6] = min(dist[6], dist[0] + dist[0][6])
dist[6] = min(inf, 0 + 7)
dist[6] = 7
```

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



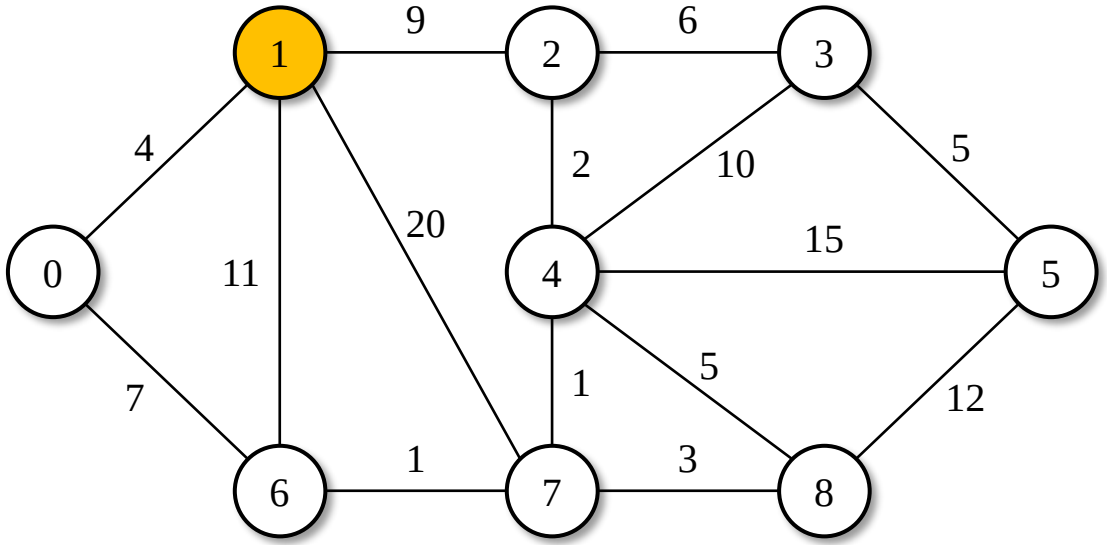
Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)
4 (1)
7 (6)

Loop do algoritmo: enquanto a fila não estiver vazia, remove um elemento, marca ele como visitado e processa os seus vizinhos ainda não visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



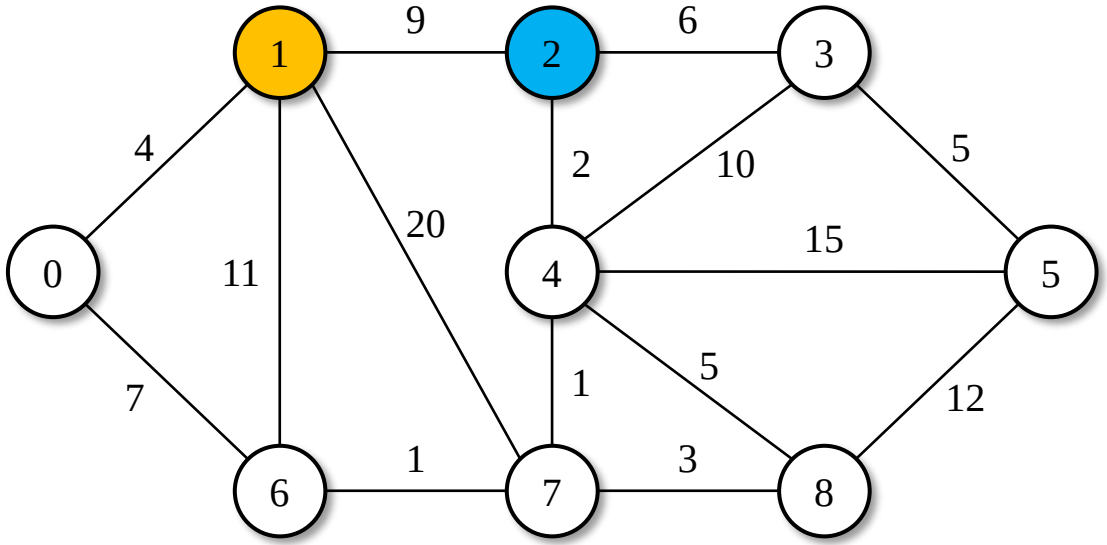
Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	inf	-1
8	inf	-1

Fila Prio
d[u] (u)
7 (6)

Loop do algoritmo: remove da fila o vértice 1, com distância 4, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 1, verifica a condição de relaxamento.
No caso, apenas 2, 6 e 7, pois 0 já foi visitado.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	inf	-1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	inf	-1
8	inf	-1

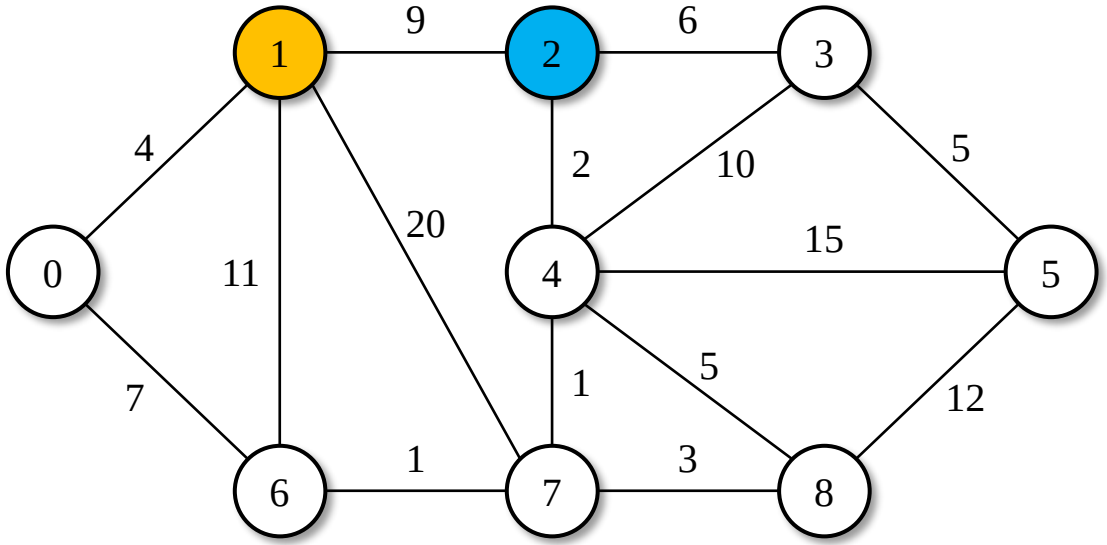
Fila Prio
d[u] (u)
7 (6)

Análise dos vizinhos: para o vértice 2: `dist[1] + dist[1][2] < dist[2]`?

`4 + 9 < inf`?

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	inf	-1
8	inf	-1

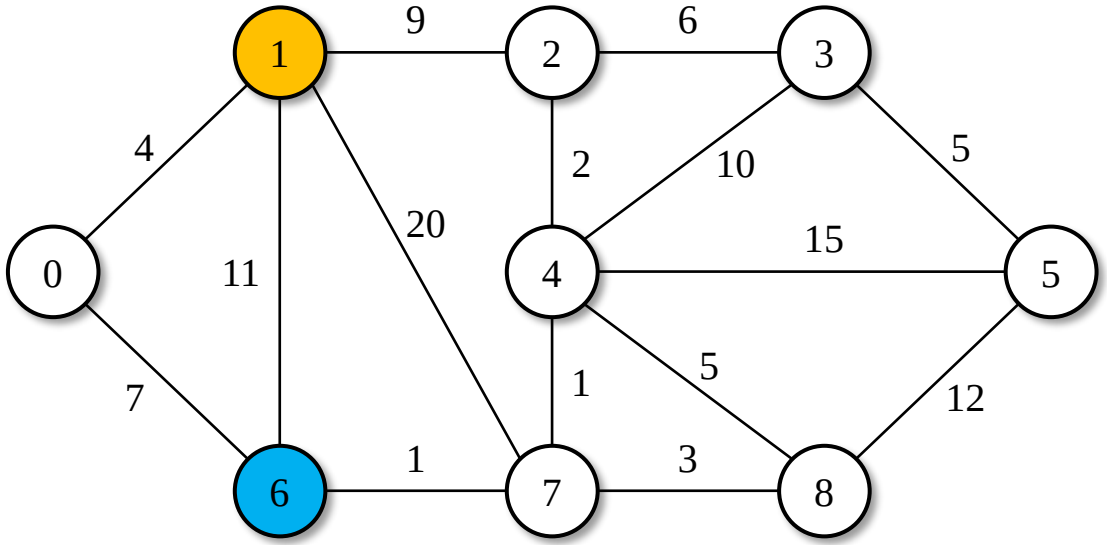
Fila Prio
d[u] (u)
7 (6)
13 (2)

Análise dos vizinhos: para o vértice 2:

```
dist[2] = min(dist[2], dist[1] + dist[1][2])
dist[2] = min(inf, 4 + 9)
dist[2] = 13
```

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	inf	-1
8	inf	-1

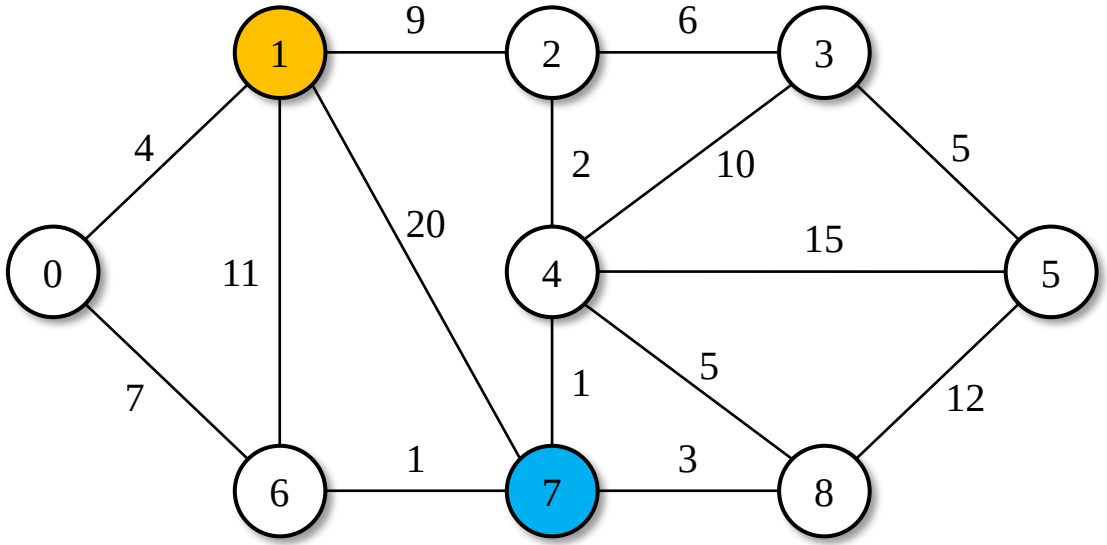
Fila Prio
d[u] (u)
7 (6)
13 (2)

Análise dos vizinhos: para o vértice 6: `dist[1] + dist[1][6] < dist[6]`?

`4 + 11 < 7`? Não! Não atualiza distância e não insere na fila.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	24	1
8	inf	-1

Fila Prio
d[u] (u)
7 (6)
13 (2)
24 (7)

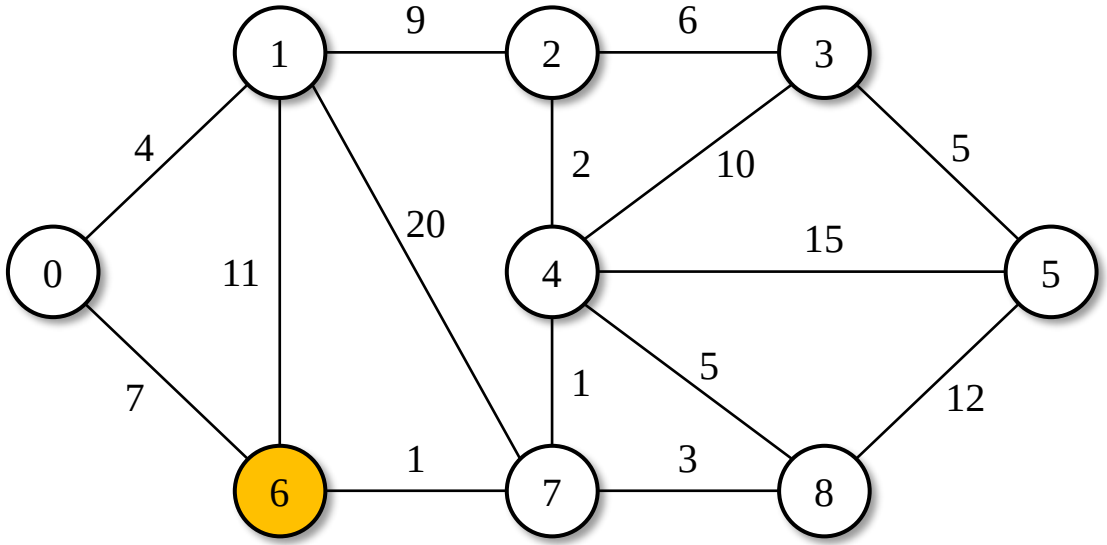
Análise dos vizinhos: para o vértice 7: `dist[1] + dist[1][7] < dist[7]`?

`4 + 20 < inf`? Sim! Atualiza e insere na fila.

`dist[7] = 24`

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



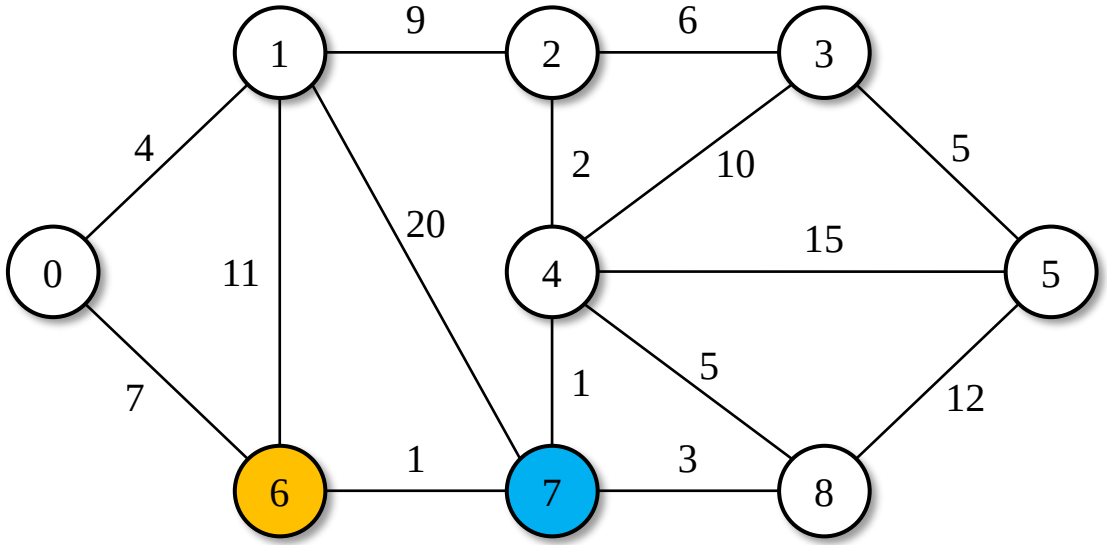
Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	24	1
8	inf	-1

Fila Prio
d[u] (u)
13 (2)
24 (7)

Loop do algoritmo: remove da fila o vértice 6, com distância 7, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 6, verifica a condição de relaxamento.
No caso, apenas 7, pois 0 e 1 já foram visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	24	1
8	inf	-1

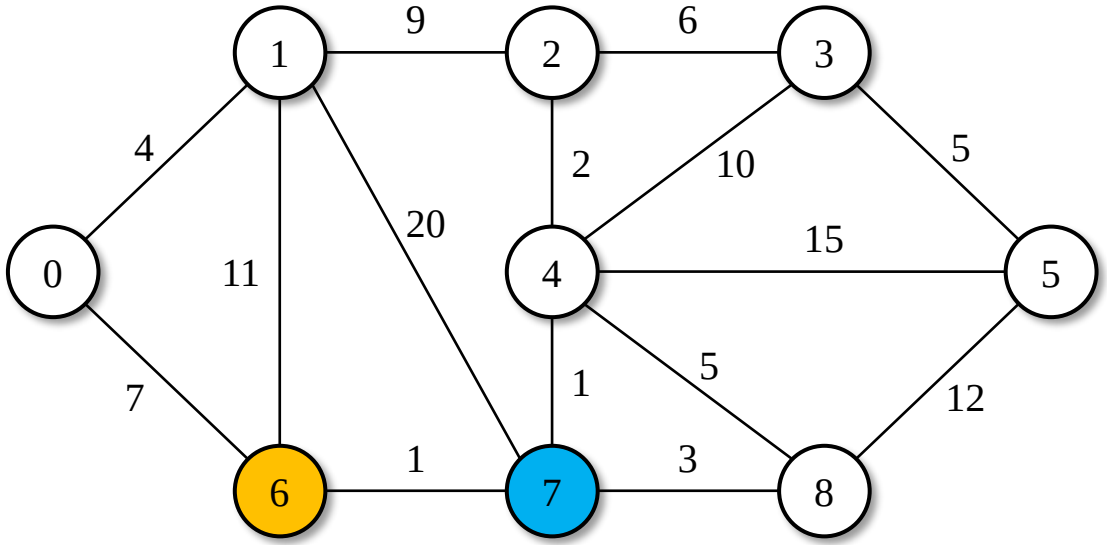
Fila Prio
d[u] (u)
13 (2)
24 (7)

Análise dos vizinhos: para o vértice 7: `dist[6] + dist[6][7] < dist[7]`?

`7 + 1 < 24`? Sim!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	24	1
8	inf	-1

Fila Prio
d[u] (u)
13 (2)
24 (7)

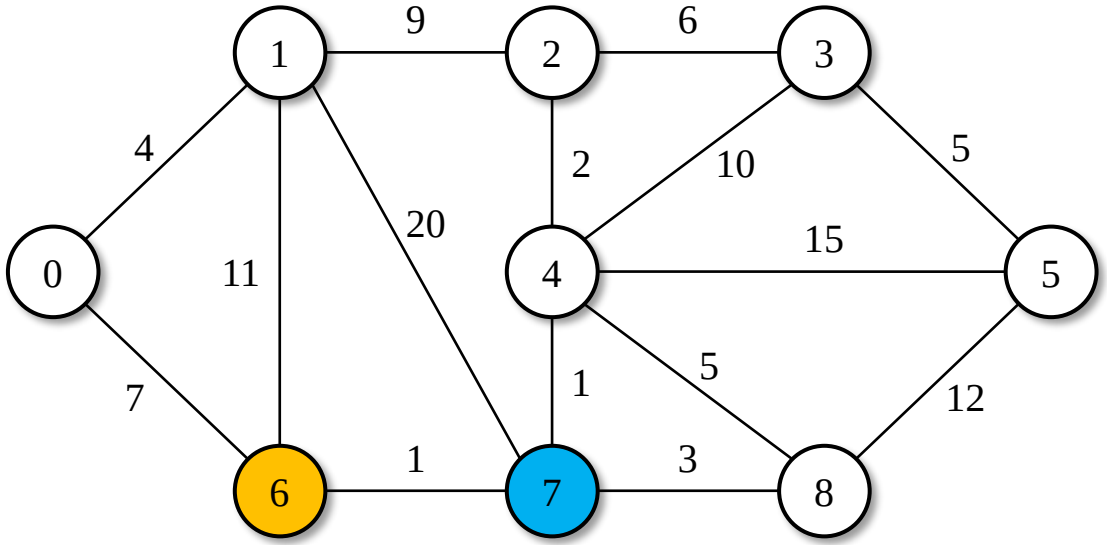
Análise dos vizinhos: para o vértice 7: `dist[6] + dist[6][7] < dist[7]`?

`7 + 1 < 24`? Sim!

Isso significa que a distância da origem até 7, passando por 6, é menor! Atualiza a distância e a fila.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	8	6
8	inf	-1

Fila Prio
d[u] (u)
13 (2)
8 (7)

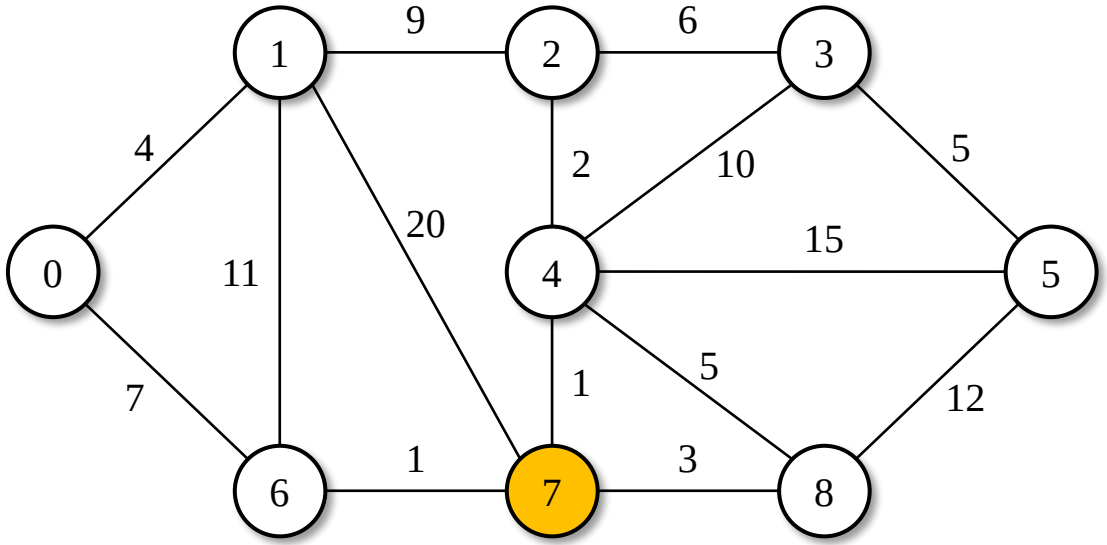
Análise dos vizinhos: para o vértice 7: `dist[6] + dist[6][7] < dist[7]`?

`7 + 1 < 24`? Sim!

`dist[7] = 8`

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



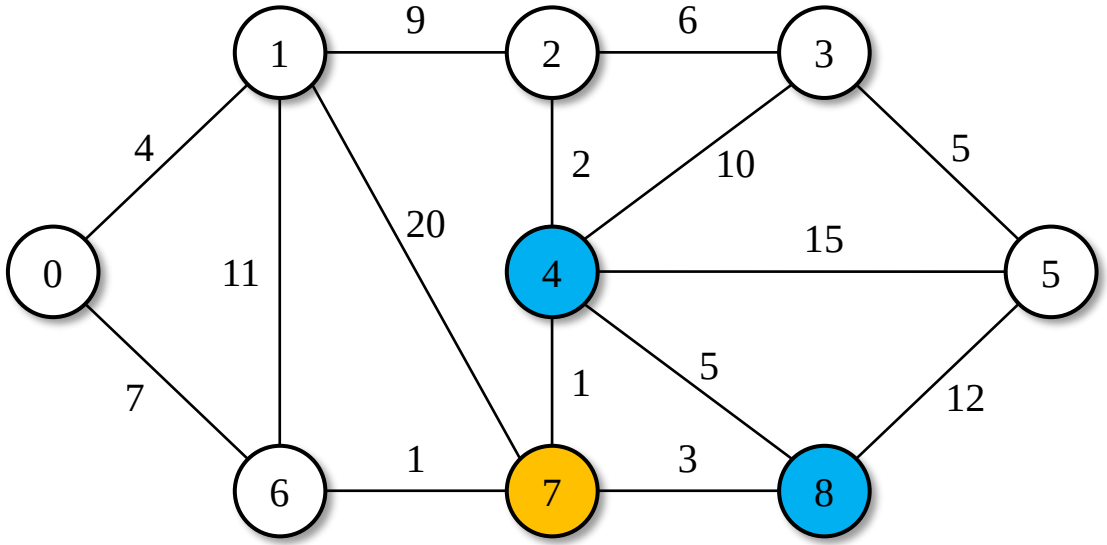
Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	8	6
8	inf	-1

Fila Prio
d[u] (u)
13 (2)

Loop do algoritmo: remove da fila o vértice 7, com distância 8, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 7, verifica a condição de relaxamento.
No caso, apenas 4 e 8, pois 1 e 6 já foram visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	inf	-1
5	inf	-1
6	7	0
7	8	6
8	inf	-1

Fila Prio
d[u] (u)
13 (2)

Para o vértice 4:

`dist[7] + dist[7][4] < dist[4]?`

`8 + 1 < inf?` Sim!

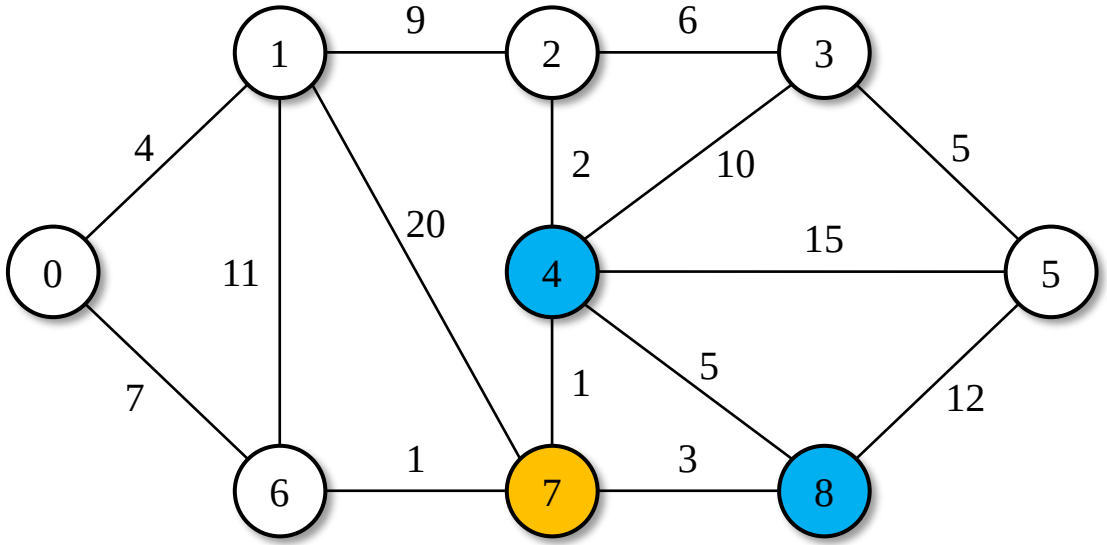
Para o vértice 8:

`dist[7] + dist[7][8] < dist[8]?`

`8 + 3 < inf?` Sim!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	9	7
5	inf	-1
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
13 (2)
9 (4)
11 (8)

Para o vértice 4:

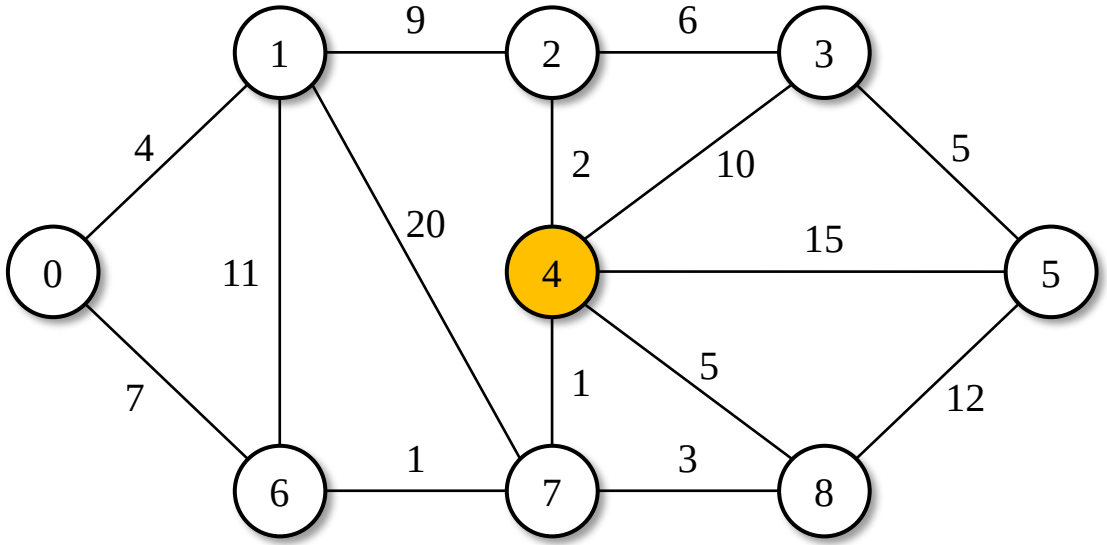
```
dist[7] + dist[7][4] < dist[4]?
8 + 1 < inf? Sim!
dist[4] = 9
```

Para o vértice 8:

```
dist[7] + dist[7][8] < dist[8]?
8 + 3 < inf? Sim!
dist[8] = 11
```

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



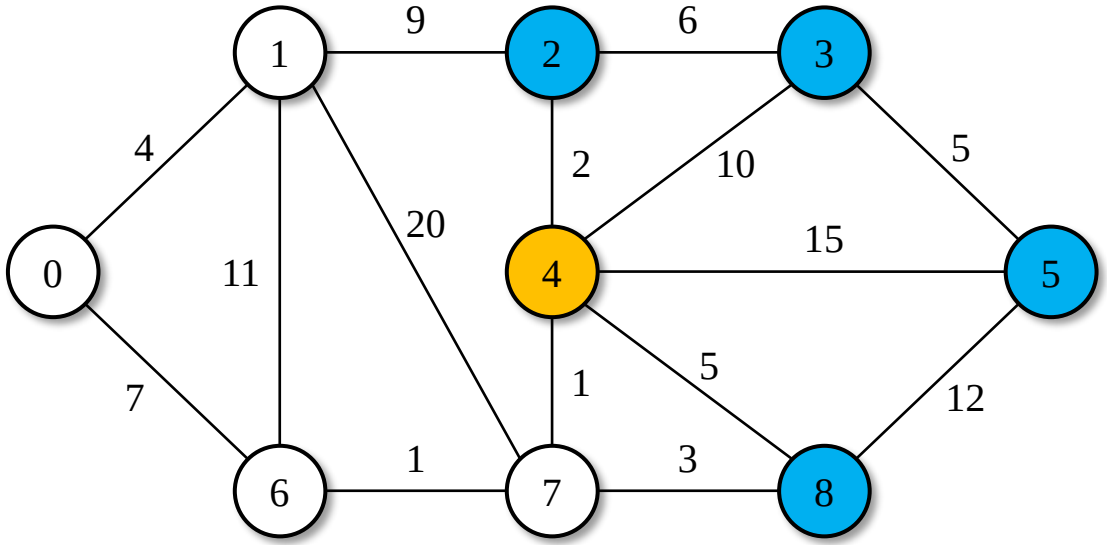
Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	9	7
5	inf	-1
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
13 (2)
11 (8)

Loop do algoritmo: remove da fila o vértice 4, com distância 9, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 4, verifica a condição de relaxamento.
No caso, apenas 2, 3, 5 e 8, pois 7 já foi visitado.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	13	1
3	inf	-1
4	9	7
5	inf	-1
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
13 (2)
11 (8)

Para o vértice 2:
dist[4] + dist[4][2] < dist[2]
9 + 2 < 13? Sim!

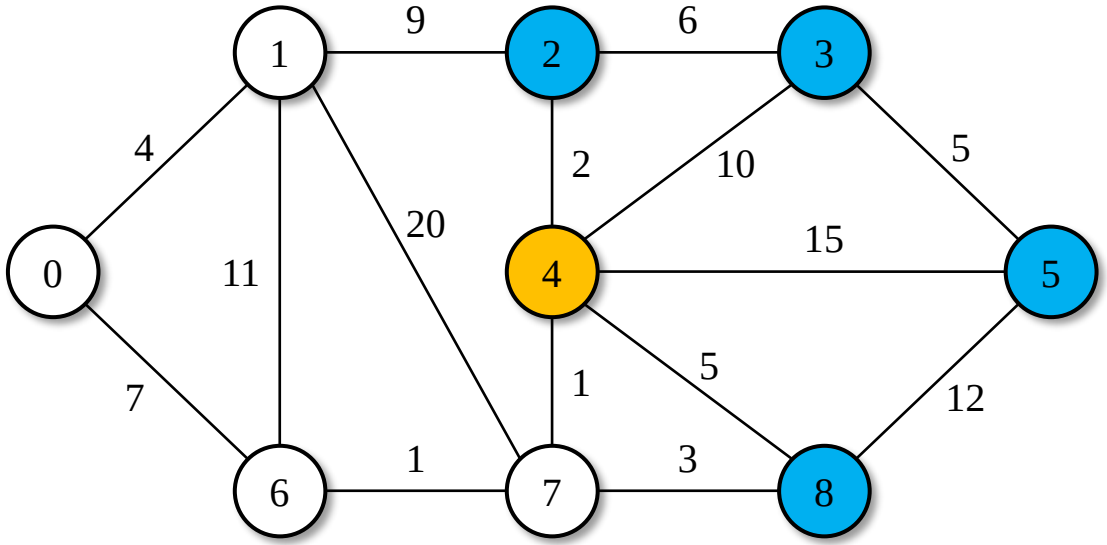
Para o vértice 3:
dist[4] + dist[4][3] < dist[3]
9 + 10 < inf? Sim!

Para o vértice 5:
dist[4] + dist[4][5] < dist[5]
9 + 15 < inf? Sim!

Para o vértice 8:
dist[4] + dist[4][8] < dist[8]
9 + 5 < 11? Não!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	19	4
4	9	7
5	24	4
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
11 (2)
11 (8)
19 (3)
24 (5)

Para o vértice 2:
dist[4] + dist[4][2] < dist[2]
9 + 2 < 13? Sim!
dist[2] = 11

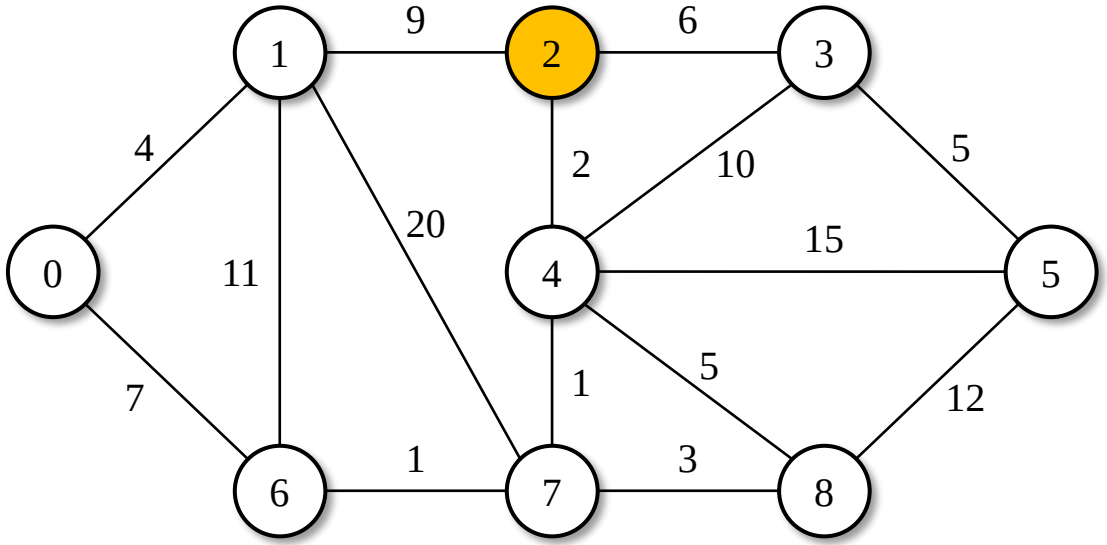
Para o vértice 3:
dist[4] + dist[4][3] < dist[3]
9 + 10 < inf? Sim!
dist[3] = 19

Para o vértice 5:
dist[4] + dist[4][5] < dist[5]
9 + 15 < inf? Sim!
dist[5] = 24

Para o vértice 8:
dist[4] + dist[4][8] < dist[8]
9 + 5 < 11? Não!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



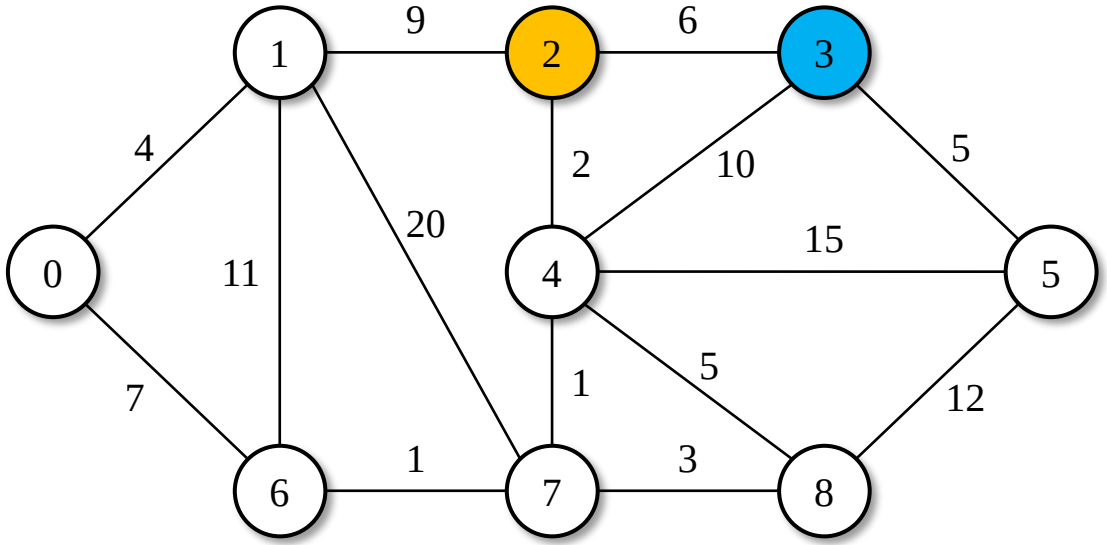
Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	19	4
4	9	7
5	24	4
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
11 (8)
19 (3)
24 (5)

Loop do algoritmo: remove da fila o vértice 2, com distância 11, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 2, verifica a condição de relaxamento.
No caso, apenas 3, pois 1 e 4 já foram visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	19	4
4	9	7
5	24	4
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
11 (8)
19 (3)
24 (5)

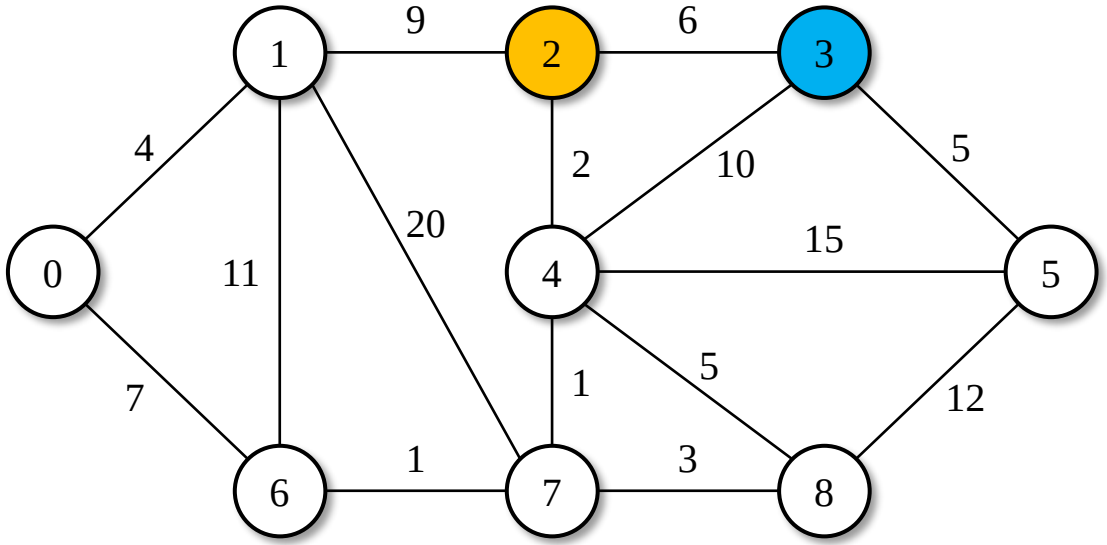
Para o vértice 3:

`dist[2] + dist[2][3] < dist[3]`?

`11 + 6 < 19`? Sim!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	24	4
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
11 (8)
17 (3)
24 (5)

Para o vértice 3:

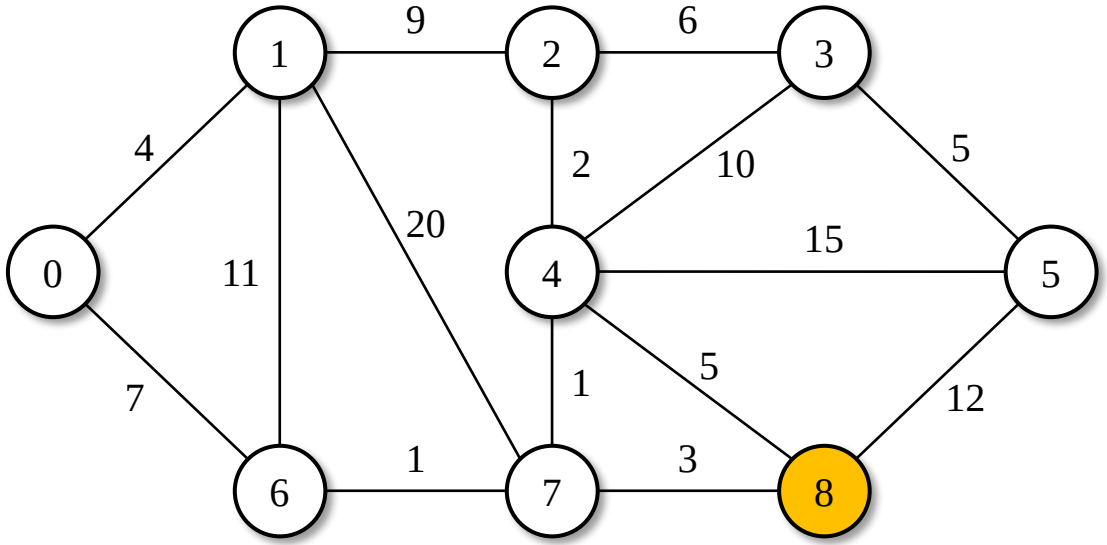
`dist[2] + dist[2][3] < dist[3]`?

`11 + 6 < 19`? Sim!

`dist[3] = 17`

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



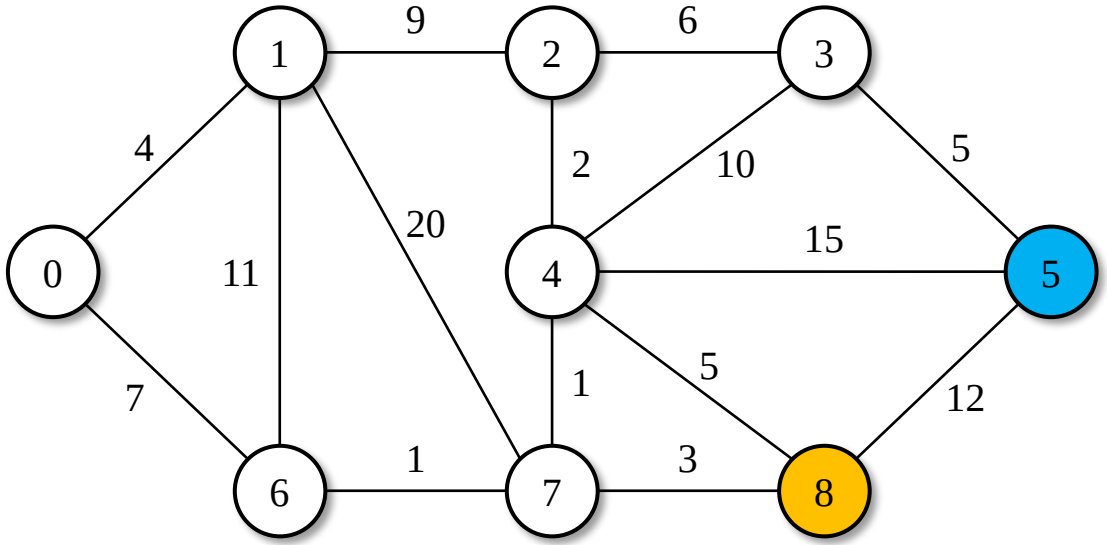
Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	24	4
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
17 (3)
24 (5)

Loop do algoritmo: remove da fila o vértice 8, com distância 11, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 8, verifica a condição de relaxamento.
No caso, apenas 5, pois 5 e 7 já foram visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	24	4
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
17 (3)
24 (5)

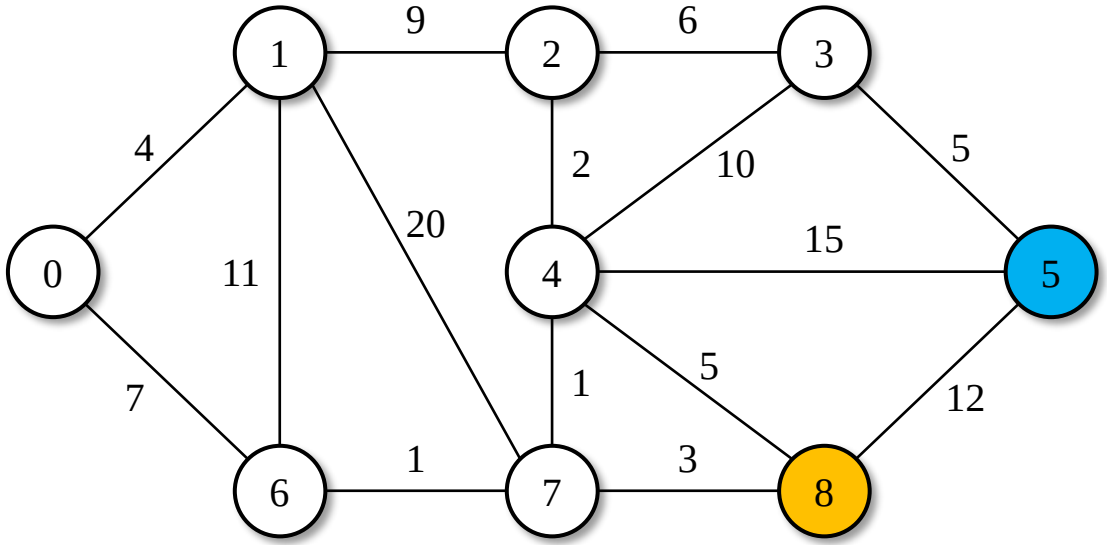
Para o vértice 5:

`dist[8] + dist[8][5] < dist[5]`?

`11 + 12 < 24`? Sim!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	23	8
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
17 (3)
23 (5)

Para o vértice 5:

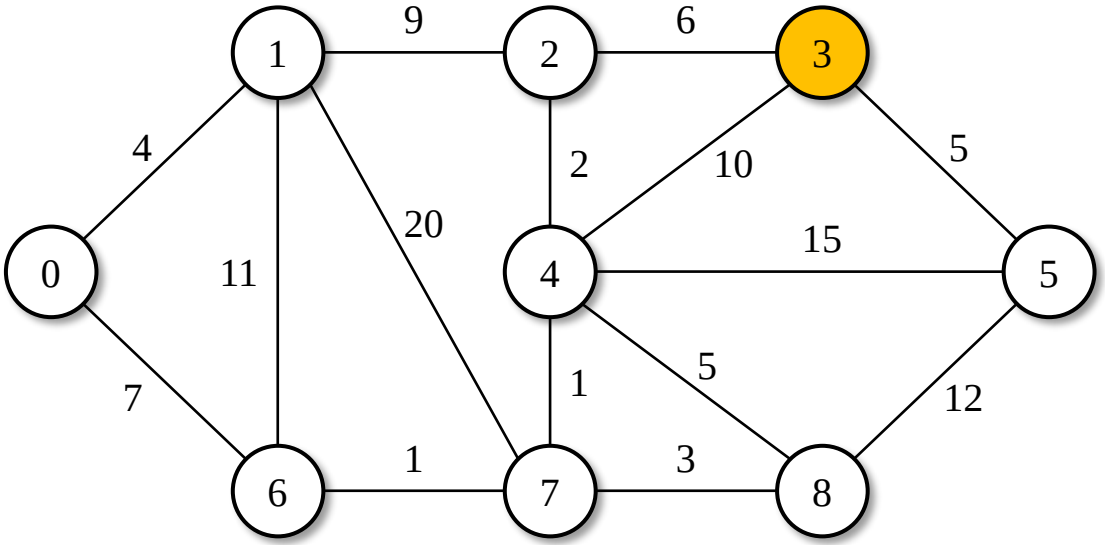
`dist[8] + dist[8][5] < dist[5]`?

`11 + 12 < 24`? Sim!

`dist[5] = 23`

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



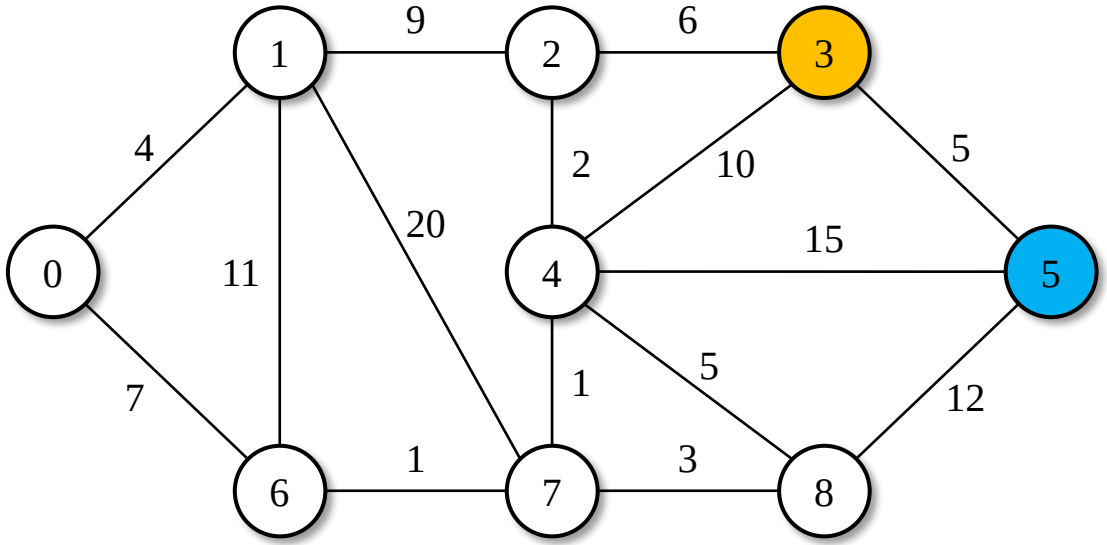
Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	23	8
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
23 (5)

Loop do algoritmo: remove da fila o vértice 3, com distância 17, e o marca como visitado.
Análise dos vizinhos: para cada vizinho de 3, verifica a condição de relaxamento.
No caso, apenas 5, pois 2 e 4 já foram visitados.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	23	8
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
23 (5)

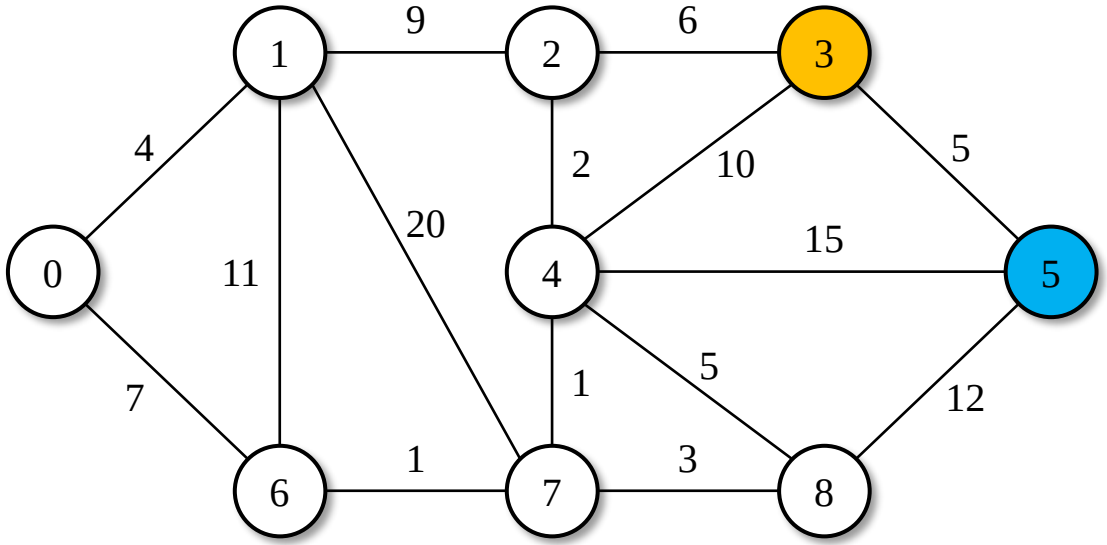
Para o vértice 5:

`dist[3] + dist[3][5] < dist[5]`?

`17 + 5 < 24`? Sim!

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	22	3
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)
22 (5)

Para o vértice 5:

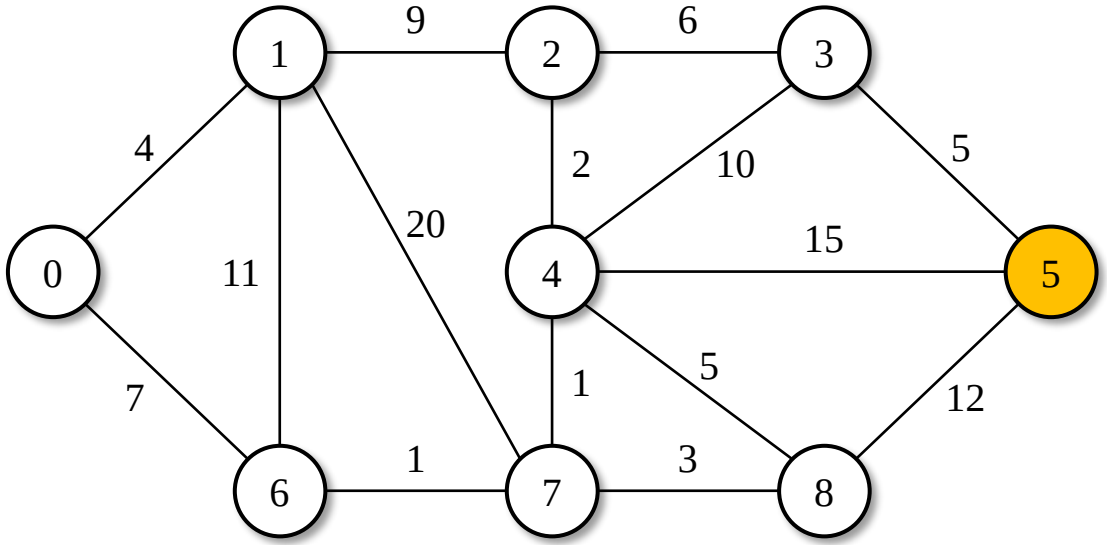
`dist[3] + dist[3][5] < dist[5]`?

`17 + 5 < 24`? Sim!

`dist[5] = 22`

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



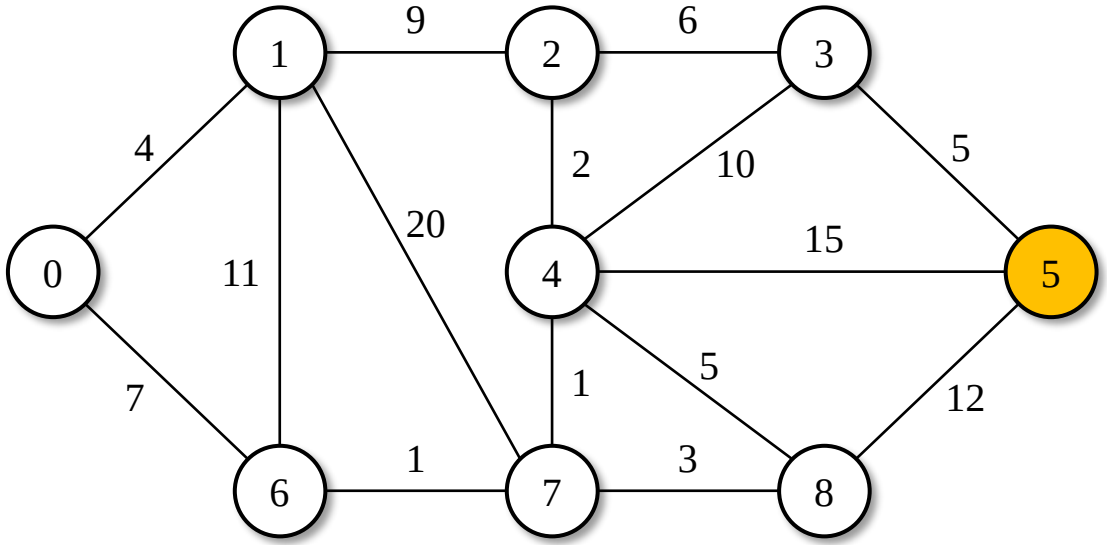
Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	22	3
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)

Loop do algoritmo: remove da fila o vértice 5, com distância 22, e o marca como visitado.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



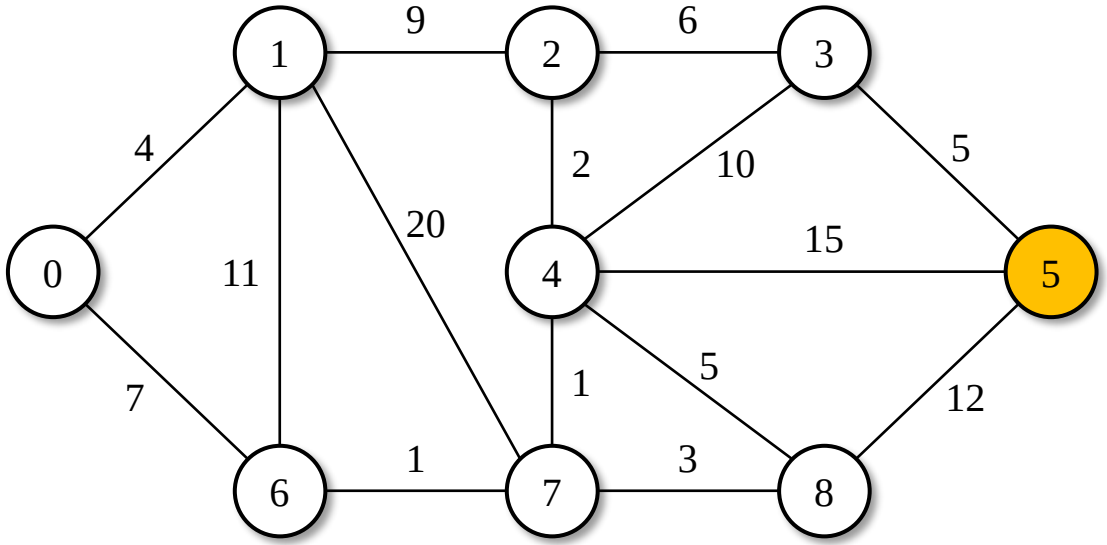
Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	22	3
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)

Encerramento: ao remover o vértice de destino, o algoritmo pode ser encerrado, pois o menor caminho até este vértice já foi encontrado.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



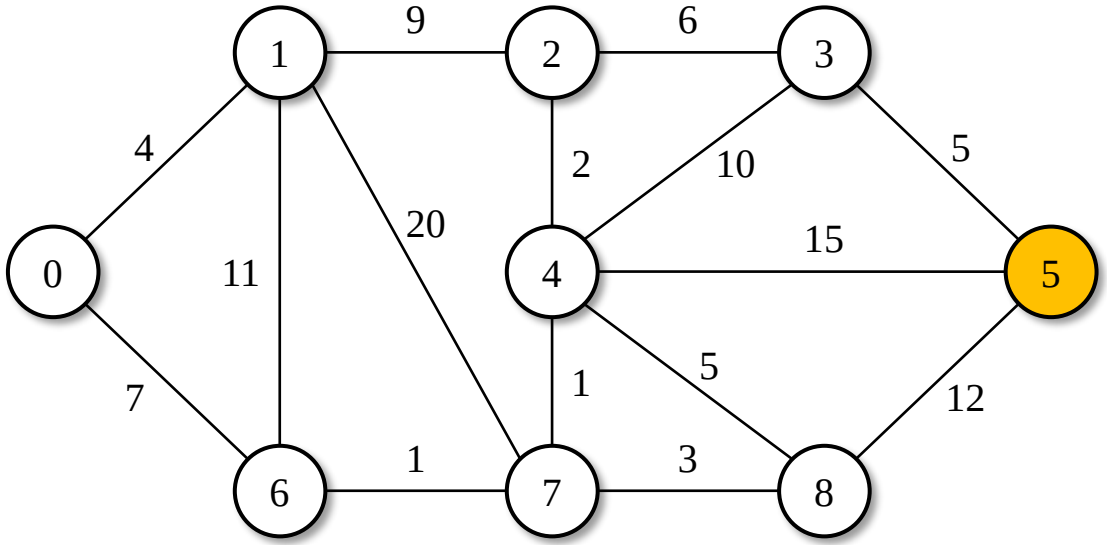
Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	22	3
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)

Caminho mínimo: o caminho mínimo de 0 até 5 tem custo 22.

Algoritmo de Dijkstra

- Exemplo: computar o menor caminho de 0 a 5.



Nó	Dist	Pred
0	0	-1
1	4	0
2	11	4
3	17	2
4	9	7
5	22	3
6	7	0
7	8	6
8	11	7

Fila Prio
d[u] (u)

Rota: a partir de 5 (destino), basta verificar os predecessores, até chegar em 0 (origem).
5 – 3 – 2 – 4 – 7 – 6 – 0.

Dijkstra: implementação com lista de adjacência

- Complexidade: O(A + V log V)

```
1. // Definicoes gerais
2. #define INF 0x3f3f3f3f // infinito
3.
4. int N = 9; // numero de vertices do grafo
5.
6. vector<int> dist(N, INF); // vetor de distancias
7. vector<int> pred(N, -1); // vetor de predecessores
8. vector<int> visitado(N, 0); // marca se o vertice foi visitado
9. priority_queue<pair<int, int>> q; // fila: (distancia, vertice)
10.
11.// prototipo da funcao
12.void dijkstra(int s); // a funcao recebe o vertice de origem
13.
```

Dijkstra: implementação com lista de adjacência

```
14.void dijkstra(int s)
15.{
16.    dist[s] = 0;
17.    q.push( {0, s} );
18.
19.    while(!q.empty())
20.    {
21.        int a = q.top().second;
22.        q.pop();
23.
24.        if(visitado[a]) continue;
25.
26.        visitado[a] = 1;
27.
28.        for(auto u : adj[a])
29.        {
30.            int b = u.first;
31.            int w = u.second;
32.
33.            if(dist[a] + w < dist[b])
34.            {
35.                dist[b] = dist[a] + w;
36.                pred[b] = a;
37.                q.push( {-dist[b], b} );
38.            }
39.        } // for
40.    } // while
41.} // dijkstra
```

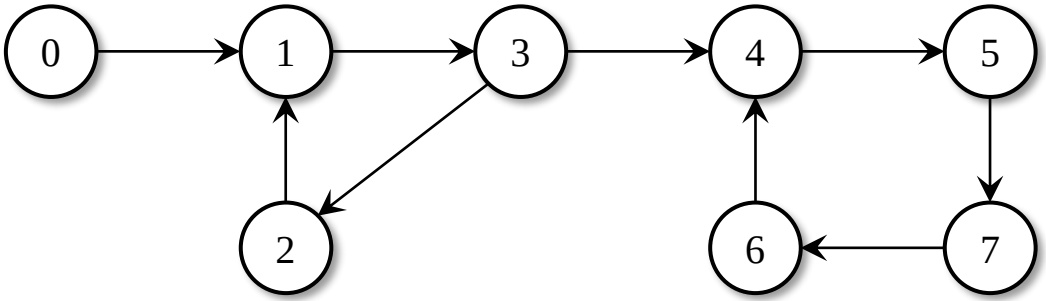
Componentes Fortemente Conexas

- O problema da conectividade está relacionado à passagem de um vértice do grafo a outro vértice, de acordo com as ligações existentes.
- Esta passagem diz respeito à atingibilidade.
- Exemplos práticos:
  - Em uma rede de computadores, um servidor é capaz de enviar mensagens de dados para um cliente específico?
  - É possível ir de carro da cidade X para a cidade Y?
- Estes problemas são resolvidos de forma similar ao percurso em grafos.

Componentes Fortemente Conexas

- O algoritmo de Busca em Profundidade pode ser empregado como base para a identificação de Componentes Fortemente Conexas (*Strongly Connected Components*) em grafos.

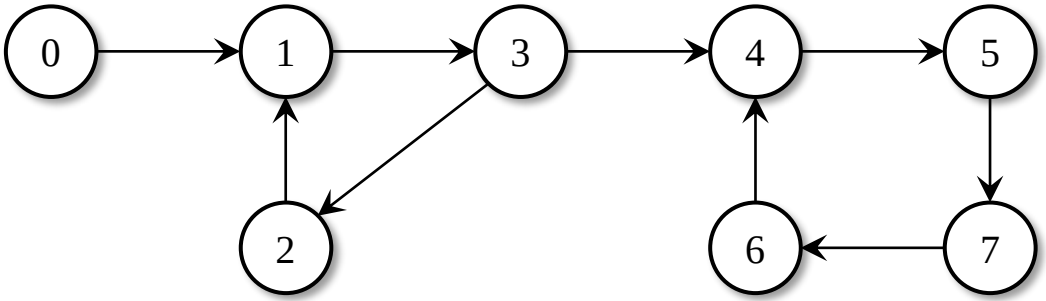
- Para grafos não direcionados, a DFS permite obter facilmente Componentes Conexas. Entretanto, quando o grafo é direcionado, sua identificação nem sempre é trivial:



Componentes Fortemente Conexas

- O algoritmo de Busca em Profundidade pode ser empregado como base para a identificação de Componentes Fortemente Conexas (*Strongly Connected Components*) em grafos.

- Para grafos não direcionados, a DFS permite obter facilmente Componentes Conexas. Entretanto, quando o grafo é direcionado, sua identificação nem sempre é trivial:

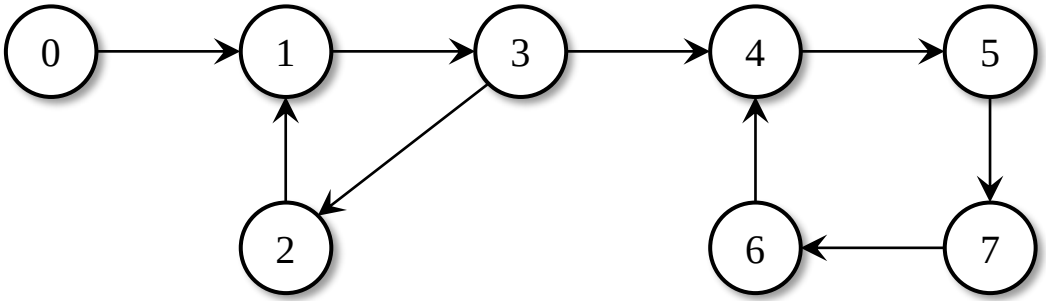


Parece ter 1 componente conexa: `dfs(0)` alcança todos os vértices.

Componentes Fortemente Conexas

- O algoritmo de Busca em Profundidade pode ser empregado como base para a identificação de Componentes Fortemente Conexas (*Strongly Connected Components*) em grafos.

- Para grafos não direcionados, a DFS permite obter facilmente Componentes Conexas. Entretanto, quando o grafo é direcionado, sua identificação nem sempre é trivial:



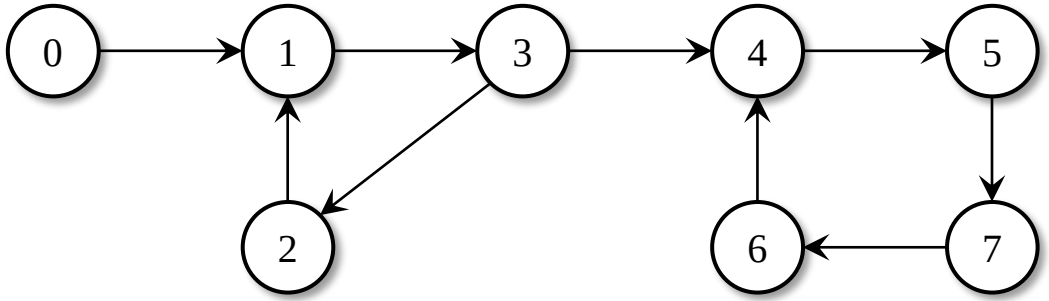
Parece ter 1 componente conexa: `dfs(0)` alcança todos os vértices.

Não é uma SCC: `dfs(1)` não alcança o vértice 0.

Componentes Fortemente Conexas

- Definição: em uma SCC, para qualquer par de vértices *u* e *v*, existe um caminho de *u* para *v* e vice-versa.

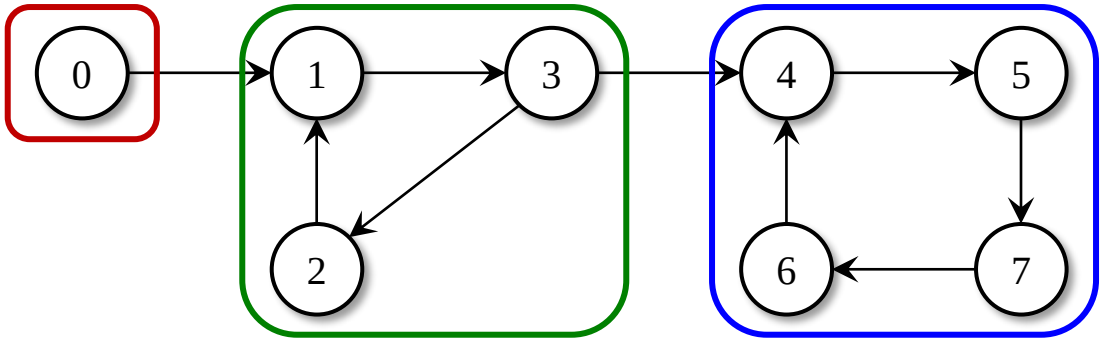
- Por exemplo, o grafo anterior possui 3 Componentes Fortemente Conexas:



Componentes Fortemente Conexas

- Definição: em uma SCC, para qualquer par de vértices *u* e *v*, existe um caminho de *u* para *v* e vice-versa.

- Por exemplo, o grafo anterior possui 3 Componentes Fortemente Conexas:



Componentes Fortemente Conexas

- Existem, ao menos, dois algoritmos conhecidos para encontrar Componentes Fortemente Conexas em grafos direcionados.

- O Algoritmo de Kosaraju executa duas DFS sobre o grafo:
  - A primeira execução constrói a lista de nós, de acordo com a ordem em que a DFS os visita. A segunda DFS identifica as SCC no grafo.

- O Algoritmo de Tarjan execute uma DFS sobre os vértices de modo que as sub-árvores dos Componentes Fortemente Conectados são removidas assim que forem encontradas.

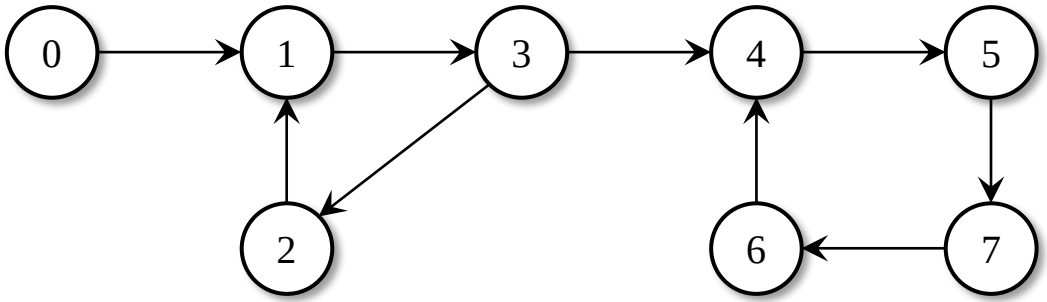
- Ambos possuem complexidade temporal proporcional a O(V + A), porém, os fatores constantes de Tarjan são bem menores do que os de Kosaraju.

Algoritmo de Tarjan

- Marca todos os vértices do grafo como não-visitados.
- Inicia a DFS. Ao visitar um nó, atribui a ele um valor numérico (num) e um valor mínimo de ligação (*low-link*). Marca os nós como visitados e os adiciona em uma pilha.
  - num: valor sequencial, a partir de 0, com a ordem em que os nós foram visitados.
- No retorno da chamada da DFS, se o nó anterior estiver na pilha, então o seu *low-link* é o mínimo entre si e o *low-link* do último nó.
  - Permite que um *low-link* se propague através de um ciclo.
- Depois de visitar todos os vizinhos, se o nó atual iniciou um SCC, retire os nós da pilha até que o nó atual seja alcançado.
  - Um nó inicia uma SCC quando seu valor numérico é igual ao seu *low-link*.

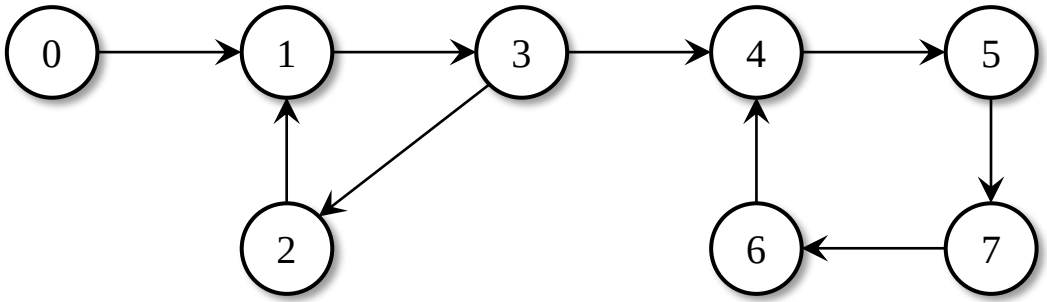
Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 0

UNVISITED: – 1

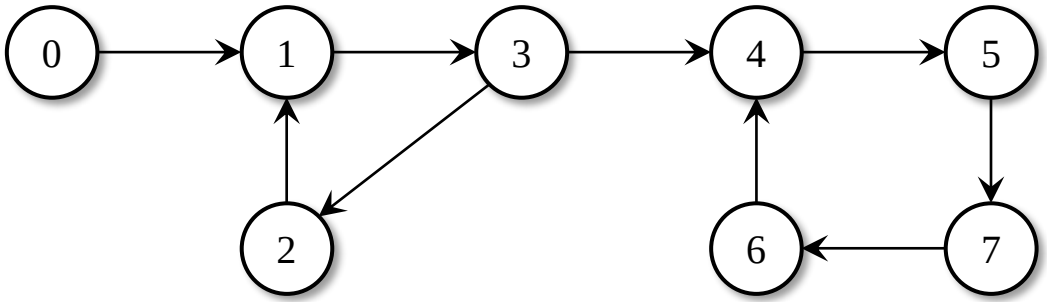
Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	-1	0
7	-1	0

Pilha

Inicialização do algoritmo: todos os vértices são marcados com num = –1 (UNVISITED).
O low-link de todos os vértices é zero. A pilha está vazia.

Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 0

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	-1	0
7	-1	0

Pilha

Execução da DFS: inicia a DFS a partir de um vértice *u* qualquer, desde que num[*u*] = UNVISITED.

Algoritmo de Tarjan

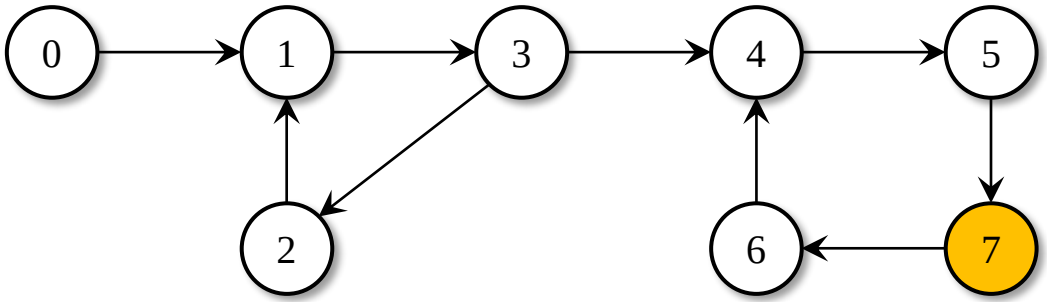
- Exemplo: encontrar as SCC do grafo.

SEQUENCIAL: 1

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	-1	0
7	0	0

Pilha
7



0 low

Execução da DFS: inicia a DFS a partir de um vértice *u* qualquer, desde que num[*u*] = UNVISITED.
Iniciando de 7. Adiciona 7 na pilha. Seta num[7] = 0 e low[7] = 0 (sequencial). Seta Nó[7] como visitado. Incrementa o valor SEQUENCIAL.

Algoritmo de Tarjan

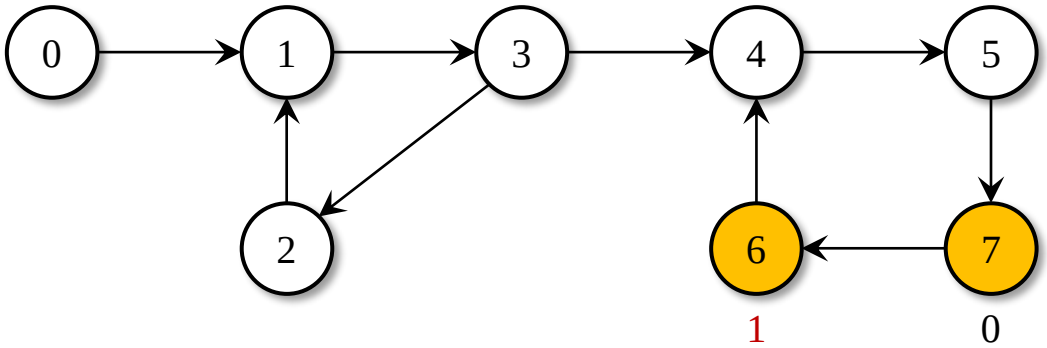
- Exemplo: encontrar as SCC do grafo.

SEQUENCIAL: 2

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	1	1
7	0	0

Pilha
6
7



DFS a partir de 7: vai para o nó 6. Adiciona 6 na pilha. Seta num[6] = 1 e low[6] = 1 (sequencial). Marca Nó[6] como visitado. Incrementa o valor SEQUENCIAL.

Algoritmo de Tarjan

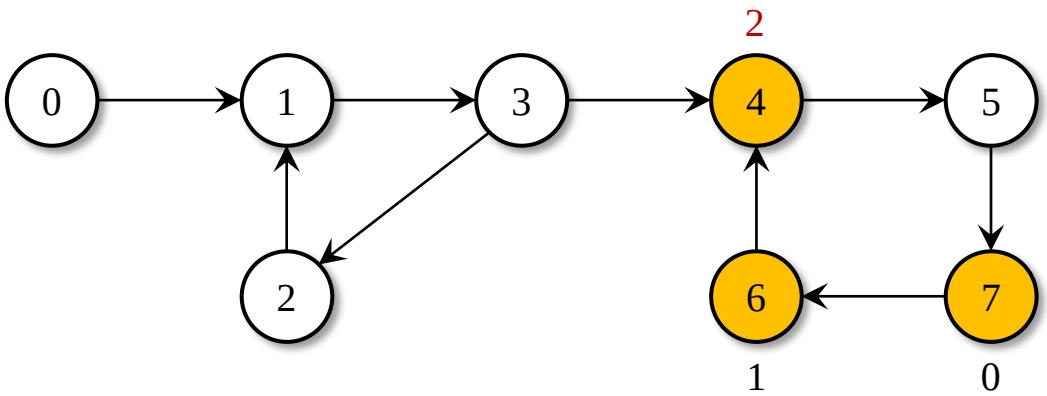
- Exemplo: encontrar as SCC do grafo.

SEQUENCIAL: 3

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	2
5	-1	0
6	1	1
7	0	0

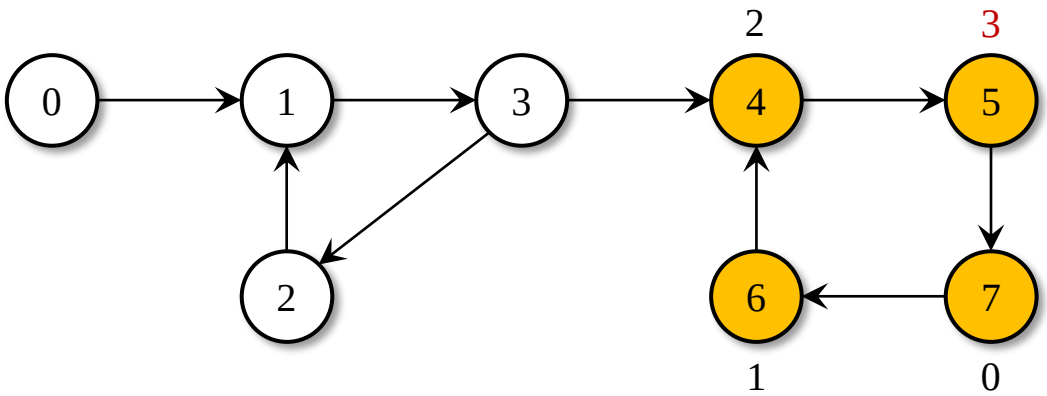
Pilha
4
6
7



DFS a partir de 6: vai para o nó 4. Adiciona 4 na pilha. Seta num[4] = 2 e low[4] = 2 (sequencial). Marca Nó[4] como visitado. Incrementa o valor SEQUENCIAL.

Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 4

UNVISITED: – 1

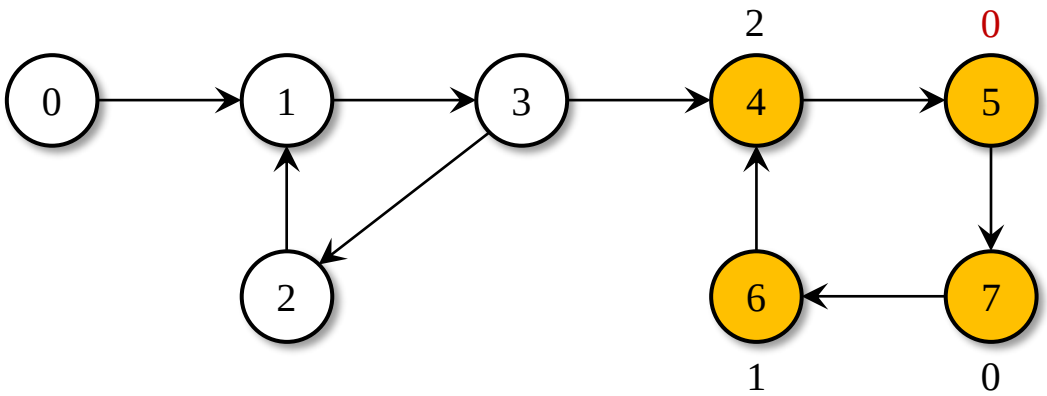
Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	2
5	3	3
6	1	1
7	0	0

Pilha
5
4
6
7

DFS a partir de 4: vai para o nó 5. Adiciona 5 na pilha. Seta num[5] = 3 e low[5] = 3 (sequencial). Marca Nó[5] como visitado. Incrementa o valor SEQUENCIAL.

Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 4

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	2
5	3	0
6	1	1
7	0	0

Pilha
5
4
6
7

DFS a partir de 5: o único vizinho (7) já foi visitado na DFS. O algoritmo irá retroceder da recursão. Como 7 está na pilha, atualiza low[5] com o mínimo entre low[5] e low[7].

Algoritmo de Tarjan

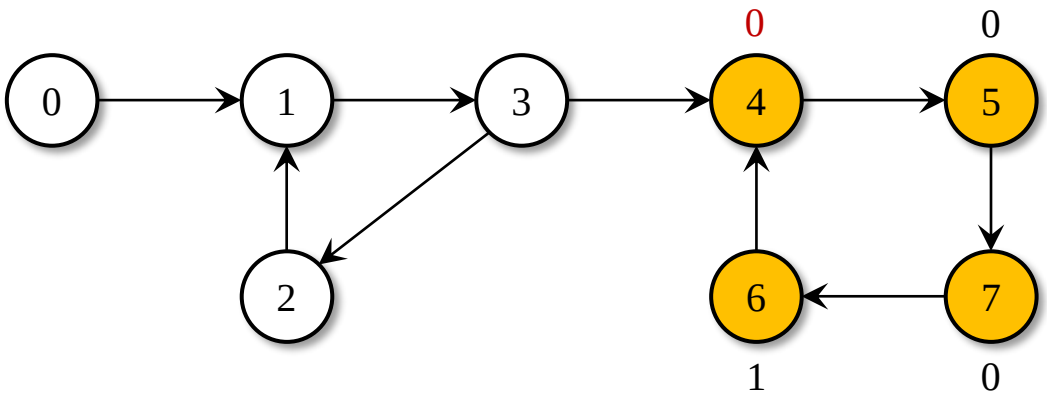
- Exemplo: encontrar as SCC do grafo.

SEQUENCIAL: 4

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	0
5	3	0
6	1	1
7	0	0

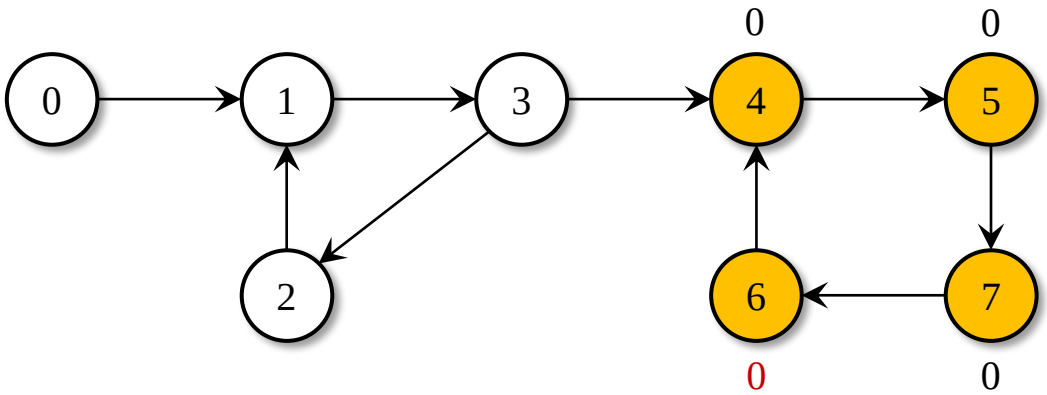
Pilha
5
4
6
7



Retrocede para 4 (veio de 5): os vizinhos de 4 já foram visitados na DFS.
Como 5 está na pilha, atualiza low[4] com o mínimo entre low[4] e low[5].

Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 4

UNVISITED: – 1

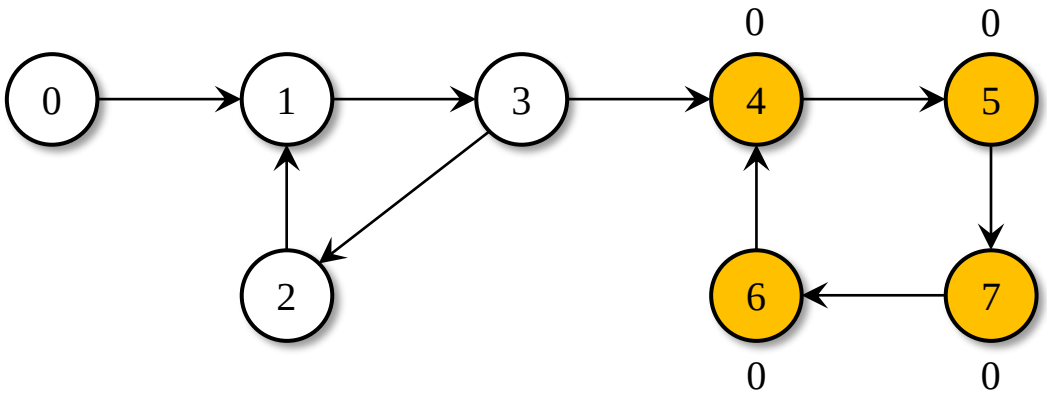
Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	0
5	3	0
6	1	0
7	0	0

Pilha
5
4
6
7

Retrocede para 6 (veio de 4): os vizinhos de 6 já foram visitados na DFS.
Como 4 está na pilha, atualiza low[6] com o mínimo entre low[6] e low[4].

Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 4

UNVISITED: – 1

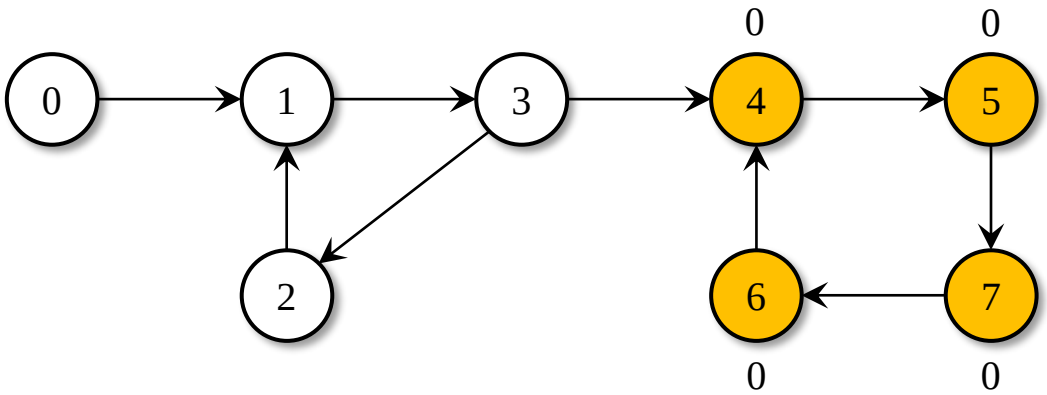
Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	0
5	3	0
6	1	0
7	0	0

Pilha
5
4
6
7

Retrocede para 7 (veio de 6): os vizinhos de 7 já foram visitados na DFS.
Como 6 está na pilha, atualiza low[7] com o mínimo entre low[7] e low[6].

Algoritmo de Tarjan

- Exemplo: encontrar as SCC do grafo.



SEQUENCIAL: 4

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	0
5	3	0
6	1	0
7	0	0

Pilha

Identificação de SCC: os vizinhos de 7 já foram visitados na DFS. Como num[7] = low[7], o algoritmo encontrou uma SCC. Desempilha todos os valores até encontrar o 7.

Algoritmo de Tarjan

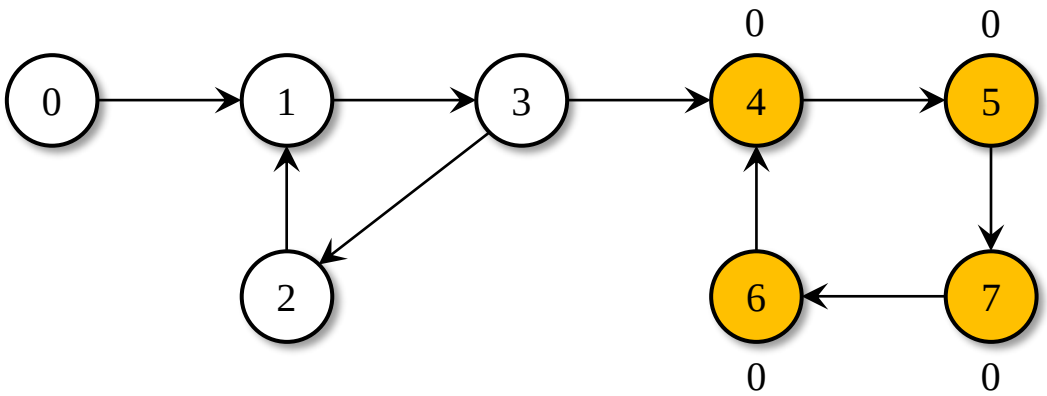
- Exemplo: encontrar as SCC do grafo.

SEQUENCIAL: 4

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	0
5	3	0
6	1	0
7	0	0

Pilha



Continuação do algoritmo: escolhe outro vértice qualquer marcado com num = –1 (UNVISITED).
Repete o processo, até que não haja mais vértices onde num = –1 (UNVISITED).

Algoritmo de Tarjan

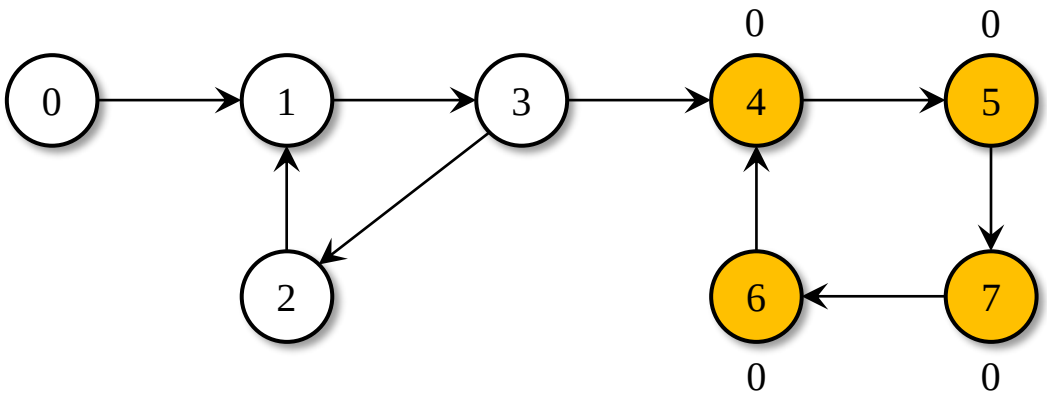
- Exemplo: encontrar as SCC do grafo.

SEQUENCIAL: 4

UNVISITED: – 1

Nó	num	low
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	2	0
5	3	0
6	1	0
7	0	0

Pilha



Finalização: ao completar a execução, o vetor low contém, para cada vértice, a identificação da SCC que ele faz parte.

Tarjan: implementação com lista de adjacência

- Complexidade: O(V + A)

```
1. // Definicoes gerais
2. #define UNVISITED -1
3. int N = 8; // numero de vertices do grafo
4. int dfsNumberCounter = 0, numSCC = 0;
5.
6. vector<int> dfs_num(N, UNVISITED); // vetor de id’s (nums sequenciais)
7. vector<int> dfs_low(N, 0); // vetor de low-links (SCC)
8. vector<int> visited(N, 0); // marca se o vertice foi visitado na DFS
9. stack<int> st; // pilha
10.
11.vector<vector<int>> AL; // lista de adjacencia do grafo
```

Tarjan: implementação com lista de adjacência

```
1. // dentro da funcao principal:
2.
3. for(int u = 0; u < N; u++)
4. {
5.     if(dfs_num[u] == UNVISITED)
6.     {
7.         tarjan(u);
8.     }
9. }
```

Tarjan: implementação com lista de adjacência

```
1. // Algoritmo de Tarjan
2. void tarjan(int u)
3. {
4.     dfs_low[u] = dfs_num[u] = dfsNumberCounter;
5.     dfsNumberCounter++;
6.     st.push(u);
7.     visited[u] = 1;
8.
9.     for(auto v : AL[u])
10.    {
11.        if (dfs_num[v] == UNVISITED) tarjan(v);
12.        if (visited[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
13.    }
14.
```

Tarjan: implementação com lista de adjacência

```
14.
15.     if(dfs_low[u] == dfs_num[u])
16.     {
17.         ++numSCC;
18.         while(1)
19.         {
20.             int v = st.top(); st.pop(); visited[v] = 0;
21.             if(u == v) break;
22.         }
23.     }
24.}
```

Dúvidas?

Aula 8:
Grafos: Algoritmos de Dijkstra e Tarjan

***Disciplina*: Maratona de Programação 1**

Profs. Edmilson Marmo e Luiz Olmes

edmarmo@unifei.edu.br, olmes@unifei.edu.br