# Caminho Mínimo entre Todos os Pares de Vértices

**Letícia Rodrigues Bueno**

UFABC

# Problemas de Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;

# Problemas de Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;

# Problemas de Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;
- **Caminho mínimo entre quaisquer vértices *u* e *v*:** algoritmo de Dijsktra;

# Problemas de Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;
- **Caminho mínimo entre quaisquer vértices *u* e *v*:** algoritmo de Dijsktra;
- **Caminho mínimo em grafos com pesos negativos:** algoritmo de Bellman-Ford;

# Problemas de Caminho Mínimo

- **Caminho mínimo de fonte única:** algoritmo de Dijsktra;
- **Caminho mínimo de destino único:** inverta a direção das arestas e aplique algoritmo de Dijstra;
- **Caminho mínimo entre quaisquer vértices $u$ e $v$:** algoritmo de Dijsktra;
- **Caminho mínimo em grafos com pesos negativos:** algoritmo de Bellman-Ford;
- **Caminho mínimo entre todos os pares de vértices:** algoritmo de Floyd-Warshall em tempo $O(n^3)$.

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:**

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra *n* vezes (uma para cada vértice);

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra *n* vezes (uma para cada vértice);
- **Quanto custaria?**

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n)$

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n)$

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n)$ $= O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;
- **Se pesos negativos:**

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;
- **Se pesos negativos:** executar algoritmo de Bellman-Ford $n$ vezes (uma para cada vértice);

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;
- **Se pesos negativos:** executar algoritmo de Bellman-Ford $n$ vezes (uma para cada vértice);
- **Quanto custaria?**

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;
- **Se pesos negativos:** executar algoritmo de Bellman-Ford $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O(nm)$

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;
- **Se pesos negativos:** executar algoritmo de Bellman-Ford $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O(nm) = O(n^2 m)$

# Caminho Mínimo entre Todos os Pares de Vértices

**Opções:**

- **Se pesos são não negativos:** executar algoritmo de Dijkstra $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O((n+m)\log n) = n \times O(m \log n) = O(nm \log n)$ sendo $O(n^3 \log n)$ para um grafo denso;
- **Se pesos negativos:** executar algoritmo de Bellman-Ford $n$ vezes (uma para cada vértice);
- **Quanto custaria?** $n \times O(nm) = O(n^2 m)$ sendo $O(n^4)$ para um grafo denso;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;

# Caminho Mínimo entre Todos os Pares de Vértices

**Algoritmo de Floyd-Warshall:**

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:

# Caminho Mínimo entre Todos os Pares de Vértices

**Algoritmo de Floyd-Warshall:**

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:
  - $W[i, j] = 0$ se $i = j$;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:
  - $W[i,j] = 0$ se $i = j$;
  - $W[i,j] = p(i,j)$ se $i \neq j$ e $(i,j) \in E(G)$ onde $p(i,j)$ é o peso da aresta $(i,j)$;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:
  - $W[i,j] = 0$ se $i = j$;
  - $W[i,j] = p(i,j)$ se $i \neq j$ e $(i,j) \in E(G)$ onde $p(i,j)$ é o peso da aresta $(i,j)$;
  - $W[i,j] = \infty$ se $i \neq j$ e $(i,j) \notin E(G)$;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:
  - $W[i,j] = 0$ se $i = j$;
  - $W[i,j] = p(i,j)$ se $i \neq j$ e $(i,j) \in E(G)$ onde $p(i,j)$ é o peso da aresta $(i,j)$;
  - $W[i,j] = \infty$ se $i \neq j$ e $(i,j) \notin E(G)$;
- **matriz** *pai*:

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:
  - $W[i,j] = 0$ se $i = j$;
  - $W[i,j] = p(i,j)$ se $i \neq j$ e $(i,j) \in E(G)$ onde $p(i,j)$ é o peso da aresta $(i,j)$;
  - $W[i,j] = \infty$ se $i \neq j$ e $(i,j) \notin E(G)$;
- **matriz** *pai*:
  - $pai[i,j] = null$ se $i = j$ ou se não existe caminho de $i$ para $j$;

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

- algoritmo de programação dinâmica;
- Pode ter pesos negativos, sem ciclos de pesos negativos;
- Grafo orientado $G$ onde $V(G) = 1, 2, 3, \ldots, n$ e subconjunto $1, 2, 3, \ldots, k$;
- grafo representado por **matriz de adjacências** $W$ onde:
  - $W[i,j] = 0$ se $i = j$;
  - $W[i,j] = p(i,j)$ se $i \neq j$ e $(i,j) \in E(G)$ onde $p(i,j)$ é o peso da aresta $(i,j)$;
  - $W[i,j] = \infty$ se $i \neq j$ e $(i,j) \notin E(G)$;
- **matriz** *pai*:
  - $pai[i,j] = null$ se $i = j$ ou se não existe caminho de $i$ para $j$;
  - $pai[i,j]$: predecessor de $j$ em caminho mínimo a partir de $i$.

# Caminho Mínimo entre Todos os Pares de Vértices

## Algoritmo de Floyd-Warshall:

# Caminho Mínimo entre Todos os Pares de Vértices

**Algoritmo de Floyd-Warshall:**

- Para $i, j \in V(G)$, considera todos caminhos de $i$ a $j$ em que vértices intermediários pertencem a $1, 2, 3, \ldots, k$, onde $p$ é o mais curto de todos;

# Caminho Mínimo entre Todos os Pares de Vértices

**Algoritmo de Floyd-Warshall:**

- Para $i, j \in V(G)$, considera todos caminhos de $i$ a $j$ em que vértices intermediários pertencem a $1, 2, 3, \ldots, k$, onde $p$ é o mais curto de todos;
- Analisa caminho $p$ e caminhos mais curtos de $i$ a $j$ com todos vértices intermediários em $1, 2, 3, \ldots, k-1$;

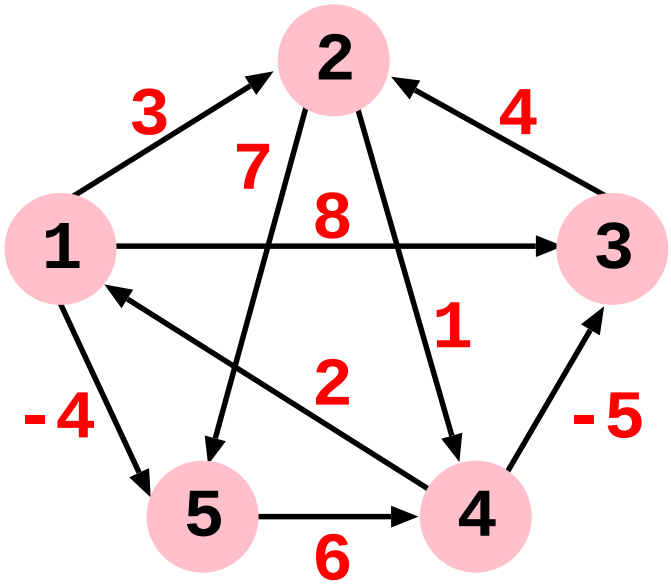
# Caminho Mínimo entre Todos os Pares de Vértices

**Algoritmo de Floyd-Warshall:**

- Para $i, j \in V(G)$, considera todos caminhos de $i$ a $j$ em que vértices intermediários pertencem a $1, 2, 3, \ldots, k$, onde $p$ é o mais curto de todos;
- Analisa caminho $p$ e caminhos mais curtos de $i$ a $j$ com todos vértices intermediários em $1, 2, 3, \ldots, k-1$;
- Análise depende de $k$ ser vértice intermediário de $p$;

# Caminho Mínimo entre Todos os Pares de Vértices

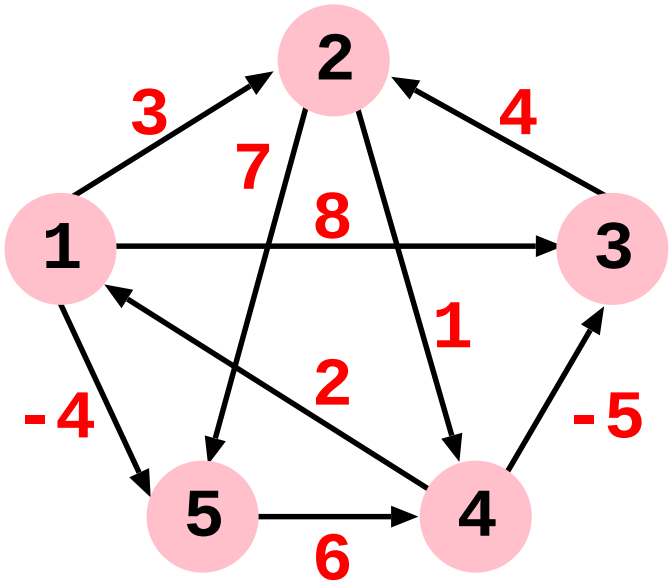
- Algoritmo de Floyd-Warshall:




$$dist = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad pai = \begin{bmatrix} null & 1 & 1 & null & 1 \\ null & null & null & 2 & 2 \\ null & 3 & null & null & null \\ 4 & null & 4 & null & null \\ null & null & null & 5 & null \end{bmatrix}$$

# Caminho Mínimo entre Todos os Pares de Vértices

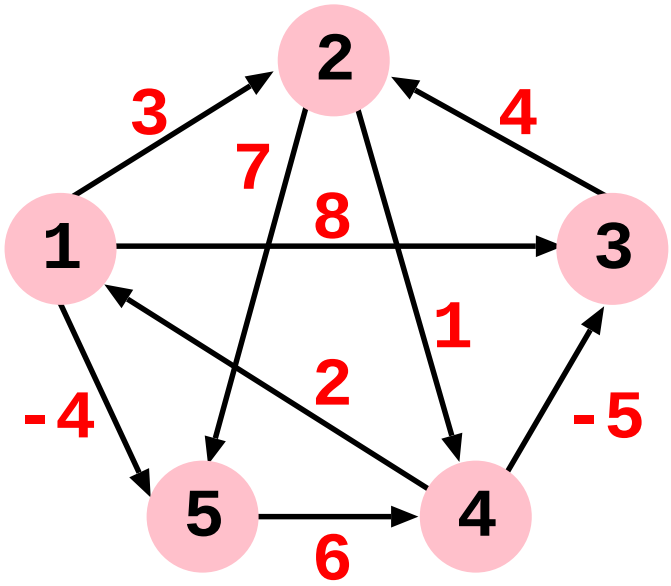
- Algoritmo de Floyd-Warshall:




$$dist = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad pai = \begin{bmatrix} null & 1 & 1 & null & 1 \\ null & null & null & 2 & 2 \\ null & 3 & null & null & null \\ 4 & 1 & 4 & null & 1 \\ null & null & null & 5 & null \end{bmatrix}$$

# Caminho Mínimo entre Todos os Pares de Vértices

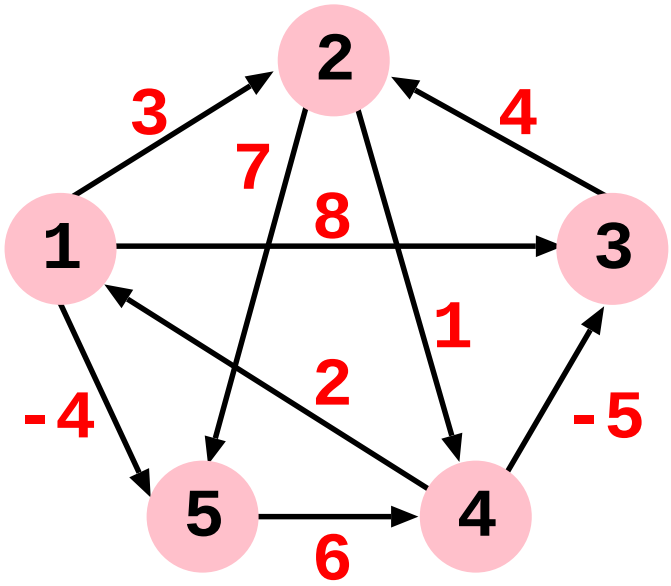
- Algoritmo de Floyd-Warshall:




$$dist = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad pai = \begin{bmatrix} null & 1 & 1 & 2 & 1 \\ null & null & null & 2 & 2 \\ null & 3 & null & 2 & 2 \\ 4 & 1 & 4 & null & 1 \\ null & null & null & 5 & null \end{bmatrix}$$

# Caminho Mínimo entre Todos os Pares de Vértices

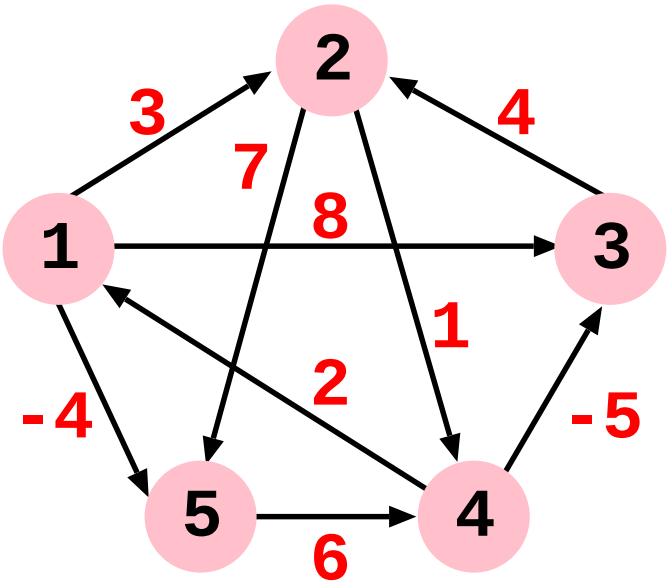
- Algoritmo de Floyd-Warshall:




$$dist = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \qquad pai = \begin{bmatrix} null & 1 & 1 & 2 & 1 \\ null & null & null & 2 & 2 \\ null & 3 & null & 2 & 2 \\ 4 & 3 & 4 & null & 1 \\ null & null & null & 5 & null \end{bmatrix}$$

# Caminho Mínimo entre Todos os Pares de Vértices

- Algoritmo de Floyd-Warshall:

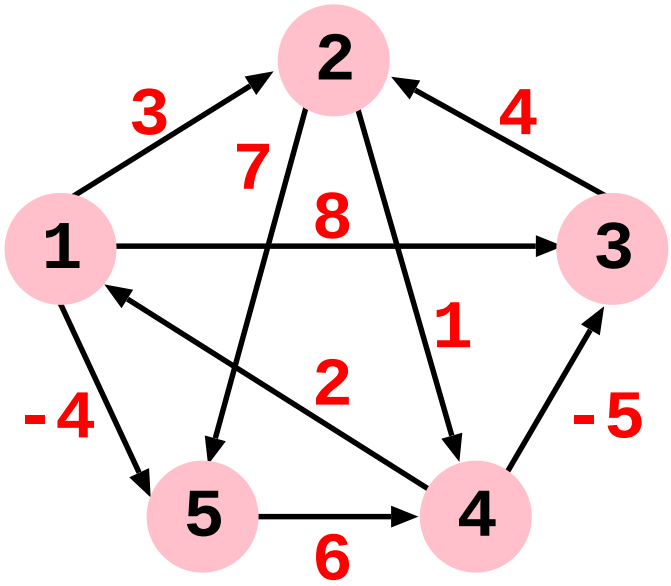



$$dist = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{bmatrix}$$

$$pai = \begin{bmatrix} null & 1 & 4 & 2 & 1 \\ 4 & null & 4 & 2 & 1 \\ 4 & 3 & null & 2 & 1 \\ 4 & 3 & 4 & null & 1 \\ 4 & 3 & 4 & 5 & null \end{bmatrix}$$

# Caminho Mínimo entre Todos os Pares de Vértices

- Algoritmo de Floyd-Warshall:




$$dist = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$pai = \begin{bmatrix} null & 3 & 4 & 5 & 1 \\ 4 & null & 4 & 2 & 1 \\ 4 & 3 & null & 2 & 1 \\ 4 & 3 & 4 & null & 1 \\ 4 & 3 & 4 & 5 & null \end{bmatrix}$$

# Complexidade do Algoritmo de Floyd-Warshall

```
1   floyd-warshall(W):
2       dist = W
3       para k=1 a n faça
4           para i=1 a n faça
5               para j=1 a n faça
6                   dist[i,j]=min(d[i,j],d[i,k]+d[k,j])
7       retorne dist;
```

# Complexidade do Algoritmo de Floyd-Warshall

```
1   floyd-warshall(W):
2       dist = W
3       para k=1 a n faça
4           para i=1 a n faça
5               para j=1 a n faça
6                   dist[i,j]=min(d[i,j],d[i,k]+d[k,j])
7       retorne dist;
```

**Complexidade:**

# Complexidade do Algoritmo de Floyd-Warshall

```
1   floyd-warshall(W):
2       dist = W
3       para k=1 a n faça
4           para i=1 a n faça
5               para j=1 a n faça
6                   dist[i,j]=min(d[i,j],d[i,k]+d[k,j])
7       retorne dist;
```

**Complexidade:**

- $O(n^3)$;

# Bibliografia Utilizada

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to Algorithms*, 3ª edição, MIT Press, 2009.