Aula 13 - Algoritmo de Floyd-Warshall

Luís Felipe

UFF

12 de Maio de 2026

Luís Felipe

12/05/26

Algoritmo de Floyd-Warshall

Problema:

- Dado um dígrafo G com pesos nas arestas, determinar a distância entre todos os pares de vértices.
- Ou seja, calcular:

$$dist(u, v) \quad \forall u, v \in V(G)$$

Pergunta:

Executar Dijkstra para cada vértice é eficiente?

- Complexidade: $O(n(n+m)\log n)$
- Para grafos densos: $O(n^3 \log n)$

Resposta:

Algoritmo de Floyd-Warshall se torna mais eficiente do que a execução do Dijkstra para cada vértice.

Luís Felipe

12/05/26

Ideia do algoritmo

- Vamos construir a solução gradualmente.
- A ideia é permitir vértices intermediários no caminho.
- Definimos:

$$D_k(u, v)$$

- menor distância de u a v usando apenas vértices $\{1, \ldots, k\}$ como intermediários. Obs.: Cada inteiro k define um conjunto de 1 até k.
- Queremos calcular:

$$D_n(u, v)$$

Recorrência fundamental

- Para cada par (u, v):

$$D_k(u, v) = \min\{D_{k-1}(u, v),\ D_{k-1}(u, k) + D_{k-1}(k, v)\}$$

- Ou seja:
  - ou não usamos k;
  - ou usamos k como intermediário.

Ideia:

testar se passar por k melhora o caminho

Inicialização

- Definimos:

$$D_0(u, v) = \begin{cases} 0 & \text{se } u = v \\ W(u, v) & \text{se existe aresta} \\ \infty & \text{caso contrário} \end{cases}$$

- Ou seja:
  - matriz de pesos do grafo

Luís Felipe

12/05/26

Algoritmo de Floyd-Warshall

```
1  Entrada: matriz W
2
3  D <- W
4
5  para k = 1 ate n faca
6      para u = 1 ate n faca
7          para v = 1 ate n faca
8              se D[u][k] + D[k][v] < D[u][v] entao
9                  D[u][v] <- D[u][k] + D[k][v]
```

Luís Felipe

12/05/26

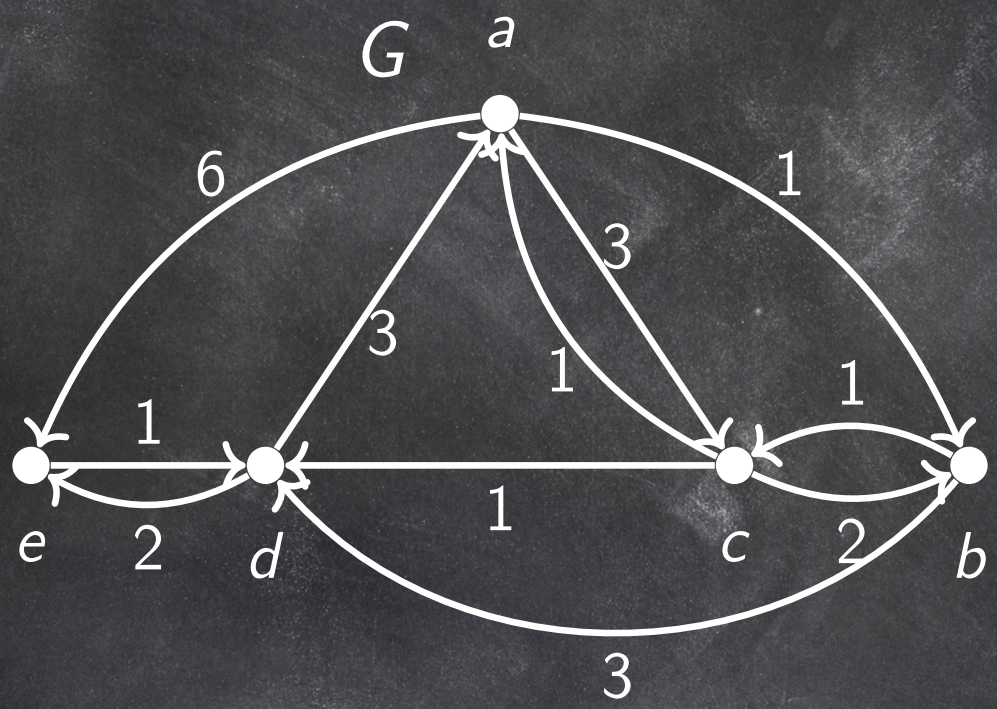
Floyd-Warshall em Python

```python
def floyd_warshall(W):

    n = len(W)

    D = [row[:] for row in W]        # cria uma copia da matriz de pesos

    for k in range(n):
        for i in range(n):
            for j in range(n):
                if D[i][k] + D[k][j] < D[i][j]:
                    D[i][j] = D[i][k] + D[k][j]

    return D
```

Luís Felipe

12/05/26

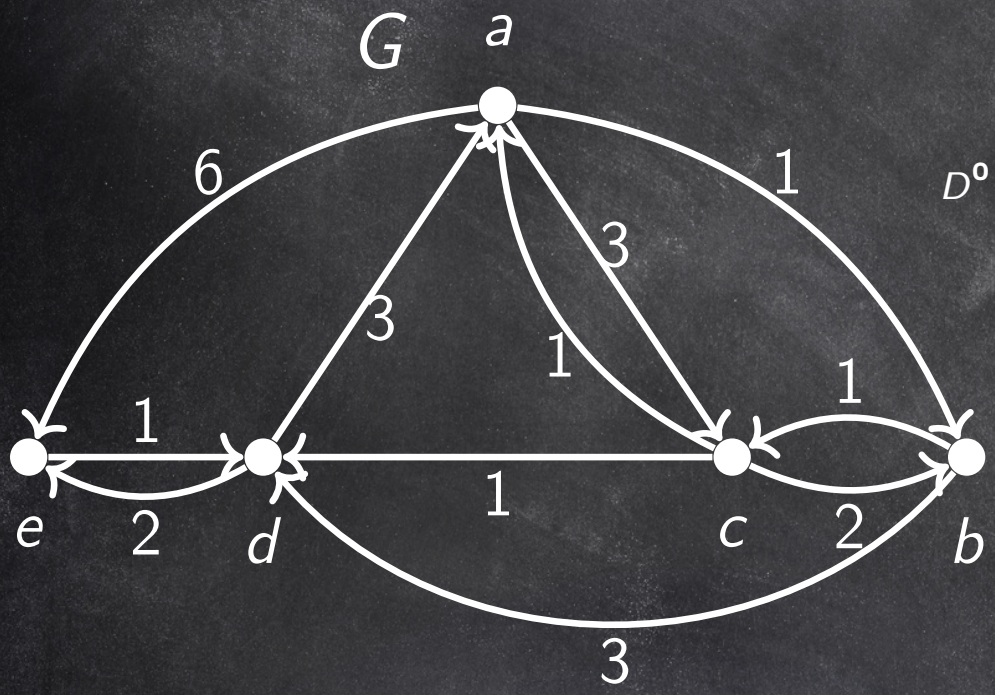
Exemplo



Luís Felipe

12/05/26

Exemplo: matriz inicial $D^0 = W$



$D^0 =$

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

- $D^0(i,j)$ representa o custo de ir de i até j sem usar vértices intermediários.
- Portanto, D^0 é exatamente a matriz de pesos W.

Exemplo: cálculo das matrizes D^k

- Em cada etapa k, liberamos o vértice k como possível intermediário.
- Para cada par (i, j), testamos:

$$D_k(i,j) = \min\{D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j)\}.$$

- Ou seja, perguntamos:

passar por k melhora o caminho de i até j?

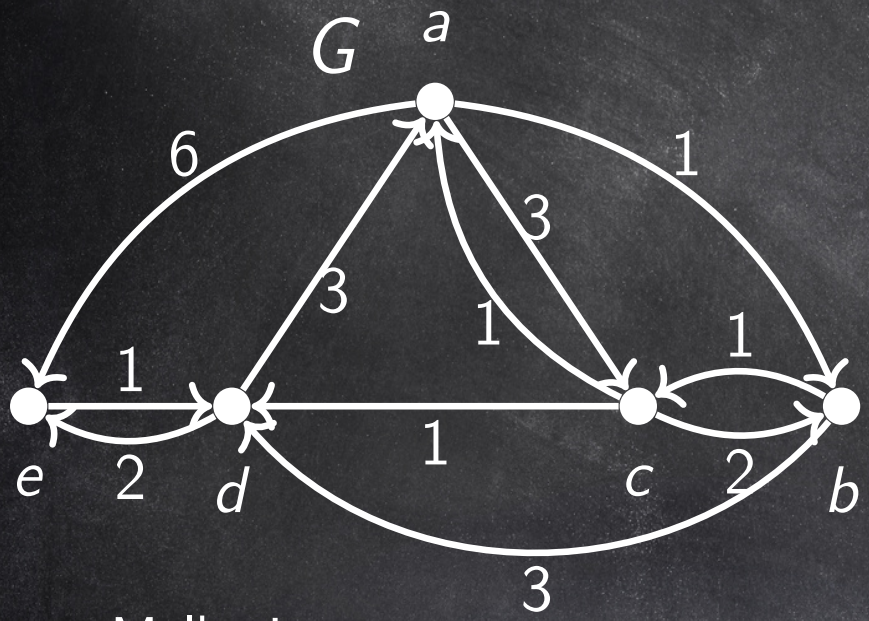
Luís Felipe

12/05/26

Exemplo: matriz D^1

Sejam os vértice a, b, c, d, e classificados como $1, 2, 3, 4, 5$.

- Agora o vértice 1 pode ser usado como intermediário.



$D^0 =$

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

$D^1 =$

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

- Melhorias:

$$D_1(3,5) = D_0(3,1) + D_0(1,5) = 1 + 6 = 7.$$

$$D_1(4,2) = D_0(4,1) + D_0(1,2) = 3 + 1 = 4.$$

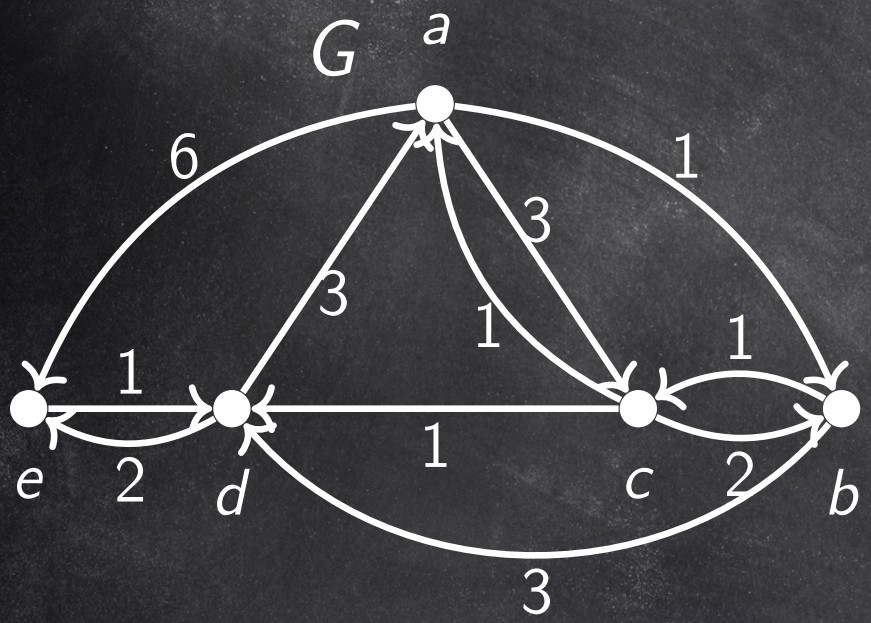
$$D_1(4,3) = D_0(4,1) + D_0(1,3) = 3 + 3 = 6.$$

Luís Felipe

12/05/26

Exemplo: matriz D^2

- Agora os vértices $\{1, 2\}$ podem ser usados como intermediários.



$D^1 =$

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

$D^2 =$

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

- Melhorias:

$$D_2(1,3) = D_1(1,2) + D_1(2,3) = 1 + 1 = 2.$$

$$D_2(1,4) = D_1(1,2) + D_1(2,4) = 1 + 3 = 4.$$

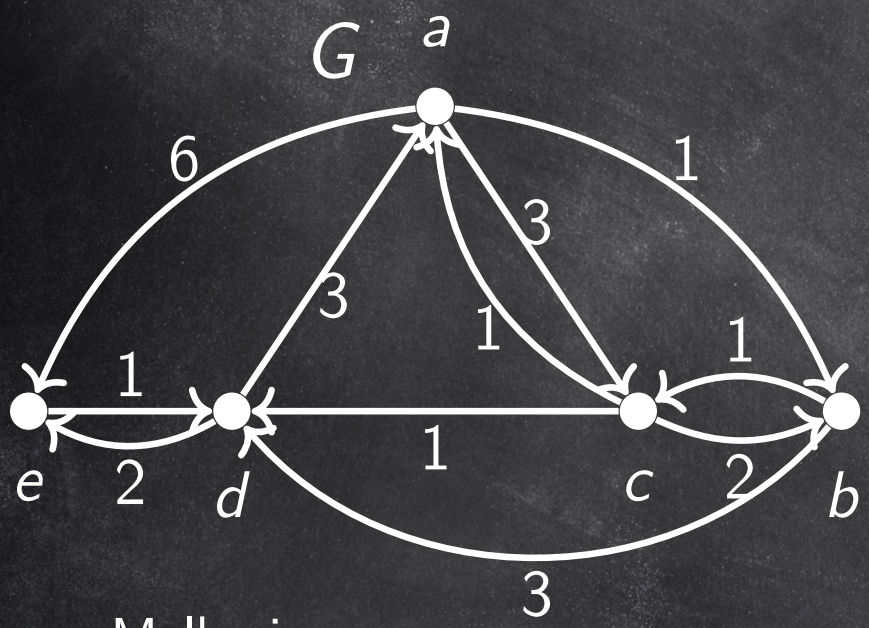
$$D_2(4,3) = D_1(4,2) + D_1(2,3) = 4 + 1 = 5.$$

Luís Felipe

12/05/26

Exemplo: matriz D^3

- Agora os vértices $\{1, 2, 3\}$ podem ser usados como intermediários.



$D^2 =$

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

$D^3 =$

	1	2	3	4	5
1	0	1	2	3	6
2	2	0	1	2	8
3	1	2	0	1	7
4	3	4	5	0	2
5	∞	∞	∞	1	0

- Melhorias:

$$D_3(1,4) = D_2(1,3) + D_2(3,4) = 2 + 1 = 3.$$

$$D_3(2,1) = D_2(2,3) + D_2(3,1) = 1 + 1 = 2.$$

$$D_3(2,4) = D_2(2,3) + D_2(3,4) = 1 + 1 = 2.$$

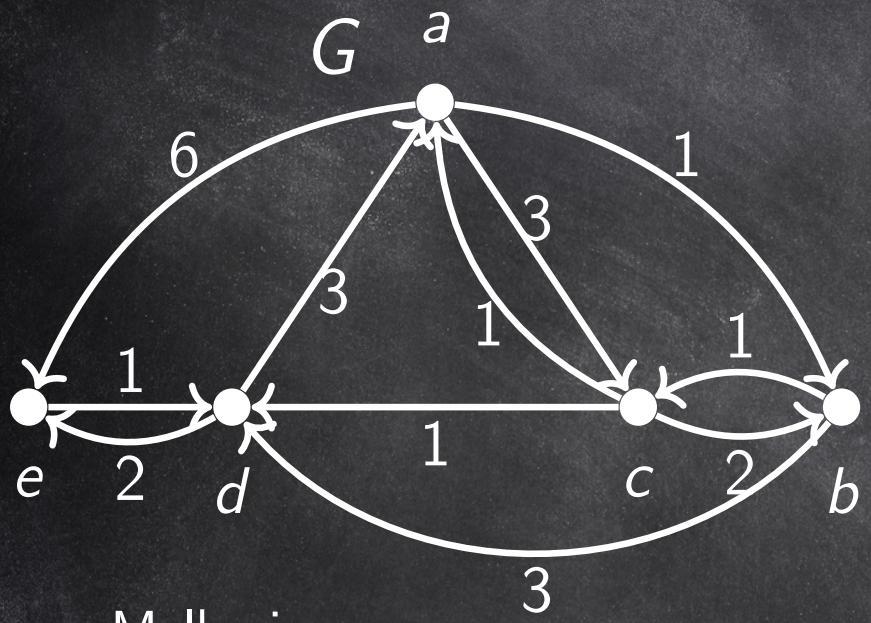
$$D_3(2,5) = D_2(2,3) + D_2(3,5) = 1 + 7 = 8.$$

Luís Felipe

12/05/26

Exemplo: matriz D^4

- Agora os vértices $\{1, 2, 3, 4\}$ podem ser usados como intermediários.



$D^3 =$

	1	2	3	4	5
1	0	1	2	3	6
2	2	0	1	2	8
3	1	2	0	1	7
4	3	4	5	0	2
5	∞	∞	∞	1	0

$D^4 =$

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

- Melhorias:

$$D_4(1,5) = D_3(1,4) + D_3(4,5) = 3 + 2 = 5.$$

$$D_4(2,5) = D_3(2,4) + D_3(4,5) = 2 + 2 = 4.$$

$$D_4(3,5) = D_3(3,4) + D_3(4,5) = 1 + 2 = 3.$$

$$D_4(5,1) = D_3(5,4) + D_3(4,1) = 1 + 3 = 4.$$

$$D_4(5,2) = D_3(5,4) + D_3(4,2) = 1 + 4 = 5.$$

$$D_4(5,3) = D_3(5,4) + D_3(4,3) = 1 + 5 = 6.$$

Exemplo: matriz final D^5

- Agora todos os vértices $\{1, 2, 3, 4, 5\}$ podem ser usados como intermediários.

$D^4 =$

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

$D^5 =$

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

- Nenhum valor melhora ao liberar o vértice 5.
- Portanto, D^5 é a matriz final de distâncias mínimas.

Como recuperar os caminhos mínimos?

- A matriz D fornece apenas os custos dos caminhos mínimos.
- Para recuperar os caminhos, usamos uma matriz auxiliar P.
- Definimos:

$$P^k(i, j)$$

- $P^k(i, j)$ é o penúltimo vértice do caminho mínimo de i até j usando apenas vértices de $\{1, \ldots, k\}$ como intermediários.
- A ideia é guardar informação suficiente para reconstruir o caminho depois.

Luís Felipe

12/05/26

Matriz inicial P^0

- Como inicialmente só consideramos arestas diretas:

$$P^0(i,j) = \begin{cases} i, & \text{se existe aresta } i \to j, \\ \text{null}, & \text{caso contrário.} \end{cases}$$

$P^0 =$

	1	2	3	4	5
1	null	1	1	null	1
2	null	null	2	2	null
3	3	3	null	3	null
4	4	null	null	null	4
5	null	null	null	5	null

Luís Felipe

12/05/26

Atualização da matriz P

- A atualização de P acompanha a atualização de D.
- Se:

$$D_{k-1}(i,j) \leq D_{k-1}(i,k) + D_{k-1}(k,j),$$

então o caminho antigo continua melhor.

$$P^k(i,j) = P^{k-1}(i,j)$$

- Caso contrário, o melhor caminho passa por k.

$$P^k(i,j) = P^{k-1}(k,j)$$

- Isso ocorre porque, no novo caminho:

$$i \rightsquigarrow k \rightsquigarrow j,$$

o penúltimo vértice antes de j é o mesmo do caminho mínimo de k até j.

Luís Felipe

12/05/26

Matrizes finais

$D^5 =$

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

$P^5 =$

	1	2	3	4	5
1	null	1	2	3	4
2	3	null	2	3	4
3	3	3	null	3	4
4	4	1	2	null	4
5	4	1	2	5	null

D^5 dá os custos; P^5 permite reconstruir os caminhos.

Luís Felipe

12/05/26

Exemplo: caminho mínimo de 5 até 3

- Queremos reconstruir o caminho mínimo de 5 até 3.
- Pela matriz final:

$$D^5(5,3) = 6.$$

- Agora usamos P^5 para descobrir os predecessores.

$$P^5(5,3) = 2$$

- Logo, o vértice anterior a 3 no caminho é 2:

$$5 \rightsquigarrow 2 \to 3.$$

$$P^5(5,2) = 1$$

$$5 \rightsquigarrow 1 \to 2 \to 3.$$

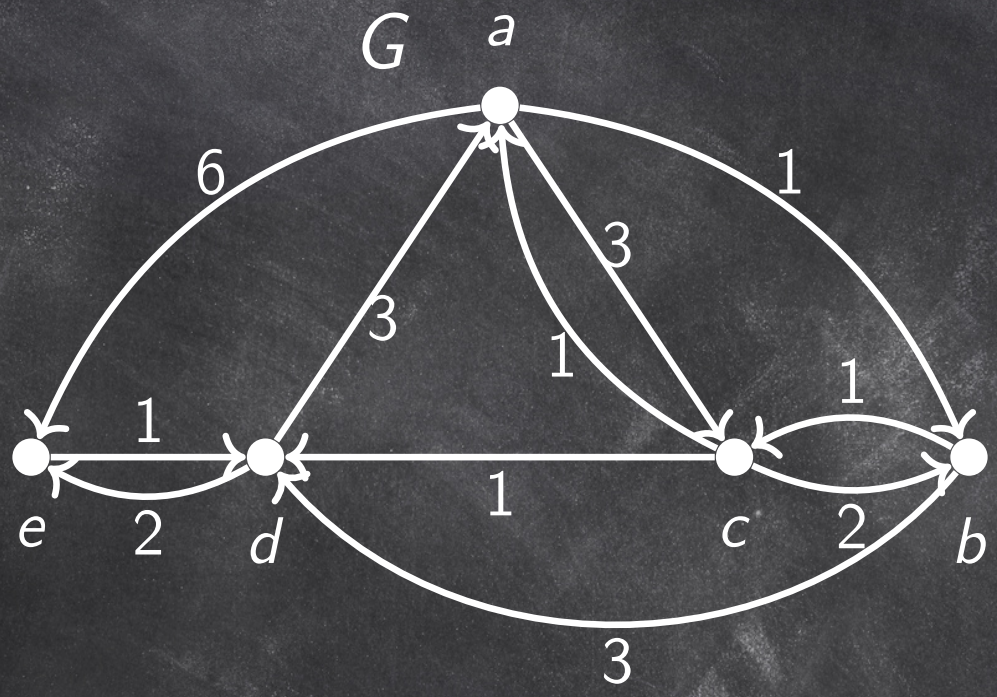
$$P^5(5,1) = 4$$

$$5 \rightsquigarrow 4 \to 1 \to 2 \to 3.$$

Luís Felipe

12/05/26

Caminho mínimo reconstruído



$$5(e) \to 4(d) \to 1(a) \to 2(b) \to 3(c)$$

- Custo do caminho:

$$W(5,4) + W(4,1) + W(1,2) + W(2,3) = 1 + 3 + 1 + 1 = 6$$

- Esse valor coincide com: $D^5(5,3) = 6$.

Luís Felipe

12/05/26

Corretude do Algoritmo de Floyd-Warshall

Ideia: Após a k-ésima iteração, $D[u][v]$ é a menor distância de u até v usando apenas vértices de $\{1, \ldots, k\}$ como intermediários.
Prova por indução em k:
Base: $k = 0$.

- $D[u][v] = W(u, v)$;
- correto, pois nenhum vértice intermediário é permitido.

Hipótese: Após a iteração $k - 1$, $D[u][v]$ é correto usando apenas intermediários em $\{1, \ldots, k - 1\}$.
Passo: Na iteração k, para cada par (u, v):

- ou o caminho mínimo de u até v não usa o vértice k;
- ou usa k como intermediário.

No primeiro caso, o custo é $D[u][v]$; no segundo, o custo é $D[u][k] + D[k][v]$. Portanto, atualizamos:
$D[u][v] \leftarrow \min\{D[u][v],\ D[u][k] + D[k][v]\}$.
Conclusão: Ao final da iteração n, $D[u][v] = dist(u, v)$ para todo par (u, v).

Luís Felipe

12/05/26

Complexidade

- Três loops aninhados:

$$O(n^3)$$

- Independe do número de arestas.
- Muito eficiente para grafos densos.

Floyd-Warshall é programação dinâmica

- O algoritmo define subproblemas:

$$D_k(u, v)$$

- Usa a recorrência:

$$D_k(u, v) = \min\{D_{k-1}(u, v),\ D_{k-1}(u, k) + D_{k-1}(k, v)\}$$

- Resolve os subproblemas em ordem crescente de k.
- Portanto, é um algoritmo de programação dinâmica.

Luís Felipe

12/05/26

Floyd-Warshall como Programação Dinâmica

```
Entrada: matriz de pesos W

# D_k[u][v] = menor distancia de u ate v
# usando apenas vertices {1,...,k} como intermediarios

D_0[u][v] <- W(u,v)

para k = 1 ate n faca
    para todo par (u,v) faca
        D_k[u][v] <- min(
            D_{k-1}[u][v],
            D_{k-1}[u][k] + D_{k-1}[k][v]
        )

retornar D_n
```

Observação:

- Esta é a forma clássica de programação dinâmica;
- a versão usual do algoritmo reutiliza a mesma matriz D.

Luís Felipe

12/05/26

Floyd-Warshall como PD em python

```python
def floyd_warshall_pd(W):
    n = len(W)

    # D[k][u][v] = menor distancia de u ate v
    # usando apenas vertices {0, ..., k-1} como intermediarios
    D = [[[float('inf')] * n for _ in range(n)] for _ in range(n + 1)]

    # caso base: D[0] = W
    for u in range(n):
        for v in range(n):
            D[0][u][v] = W[u][v]

    # programacao dinamica
    for k in range(1, n + 1):
        for u in range(n):
            for v in range(n):
                D[k][u][v] = min(
                    D[k - 1][u][v],
                    D[k - 1][u][k - 1] + D[k - 1][k - 1][v]
                )

    return D[n]
```

Luís Felipe

12/05/26

Detecção de ciclos negativos

- Após o algoritmo:
- Se existe v tal que:

$$D(v, v) < 0$$

- então existe ciclo negativo.
- pois encontramos caminho de v para ele mesmo com custo negativo.

Luís Felipe

12/05/26

Comparação dos algoritmos

- BFS $O(n+m)$
  - sem pesos
  - distância de um vértice aos demais

- Dijkstra $O((n+m)\log n)$
  - pesos não negativos
  - distância de um vértice aos demais
  - rápido

- Bellman-Ford $O(nm)$
  - permite pesos negativos
  - detecta ciclos negativos alcançáveis a partir da raiz
  - distância de um vértice aos demais

- Floyd-Warshall $O(n^3)$
  - permite pesos negativos
  - detecta ciclos negativos
  - distância entre todos os pares
  - programação dinâmica