Distância em grafos / digrafos ponderados

→ Dados um (di) grafo G, uma função $w: E(G) \to \mathbb{R}$ e dois vértices $u, v \in V(G)$:

$$dist_G^w(u,v) = \begin{cases} \text{peso de um } uv\text{-caminho de menor peso} & \text{se há } uv\text{-cam.} \\ \infty & \text{c.c.} \end{cases}$$

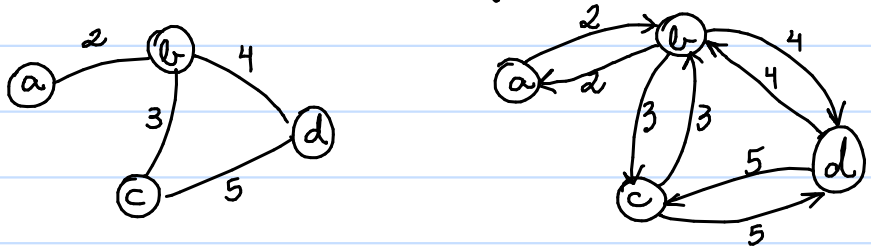
→ Um uv-caminho é mínimo se tem comprimento / peso igual à $dist_G^w(u,v)$

Problema ***Caminhos Mínimos Entre Todos os Pares***

Entrada: $\langle D, w \rangle$, onde D é um digrafo e w é uma função de peso nos arcos.
Saída: Valor $\text{dist}_D^w(u,v)$ para todo par $u, v \in V(D)$.

Grafos ou digrafos?

→ Todo grafo G tem um digrafo associado $D(G)$



Um uv-caminho mínimo em G é mínimo em $D(G)$ e vice-versa.

→ Resolver os problemas de caminhos em qualquer digrafo implica em resolver em digrafos que são digrafos associados a grafos
∴ consideraremos apenas digrafos

→ Qualquer algoritmo atual (e talvez que venha a ser criado) só resolve esses dois problemas se o digrafo <u>não tiver ciclo</u> com peso <u>total</u> negativo.

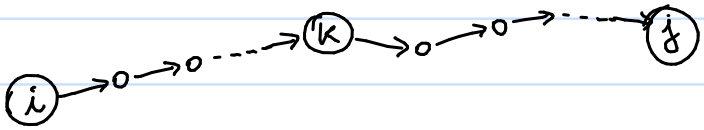
Observações sobre caminhos mínimos

→ Se $P = (v_1, \dots, v_k)$ é um v_1v_k-caminho mínimo, então qualquer subcaminho $(v_i, \dots, v_j)$ de P é um v_iv_j-caminho mínimo.

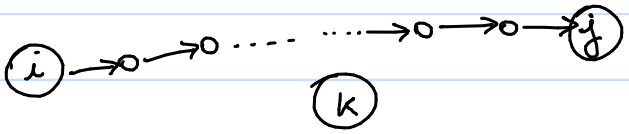
(apresentam subestrutura ótima)

Como resolver esse problema?

→ Para construir um ij-caminho, com $i, j \in V(D)$ quaisquer, "só" resta escolher os vértices internos
↳ Seja X o conj. dos disponíveis p/ serem internos
↳ $k \in X$ pode ser usado ou não.



Se usamos, podemos construir um ik-caminho e um kj-caminho e uni-los.



Se não usamos, podemos construir um ij-caminho usando vértices de $X \setminus \{k\}$ como internos.

→ Caso base: $|X| = 0$.

① Qual k escolher?

Vamos considerar $V(D) = \{1, \dots, n\}$, com $n = |V(D)|$.

→ Podemos escolher $k \in N^+(i)$ com menor $w(ik)$?
Ou $k \in N^-(j)$ com menor $w(kj)$?

→ Não importa, pois todo vértice em X será escolhido em alguma chamada recursiva.

→ Para simplificar a implementação, vamos escolher o vértice que tiver o maior rótulo.

② Esse algoritmo é ótimo?

Defina $V^k = \{1, \dots, k\}$. Se P_{ij}^k é o peso de um ij-caminho mínimo cujos vértices internos pertencem a V^k, então

$$P_{ij}^k = \min \{ P_{ij}^{k-1}, P_{ik}^{k-1} + P_{kj}^{k-1} \} \quad \text{se } 1 \leq k \leq n$$

e

$$P_{ij}^0 = \begin{cases} 0 & \text{se } i = j \\ w(ij) & \text{se } i \neq j \text{ e } ij \in E(D) \\ \infty & \text{se } i \neq j \text{ e } ij \notin E(D) \end{cases}$$

Isso vale pois caminhos mínimos contêm caminhos mínimos.

→ Todo subproblema pode ser descrito por (i, j, k), que indica que queremos encontrar ij-caminho tendo k vértices disponíveis.
↳ Assim, $1 \leq i \leq n$, $1 \leq j \leq n$ e $0 \leq k \leq n$

→ Então uma matriz $D[1..n][1..n][0..n]$ consegue guardar todos
↳ Em $D[i][j][k]$ teremos o custo do menor ij-caminho que usa vértices em V^k.
↳ Queremos calcular $D[i][j][n]$ para todo $i, j \in V(D)$.

PD para Caminhos Mínimos entre Todos os Pares (Bottom-Up)

```
FLOYD-WARSHALL-BOTTOMUP(D, w)
1  Seja W[1..n][1..n][0..n] uma matriz
2  para i = 1 até n, incrementando faça
3      para j = 1 até n, incrementando faça
4          se i == j então
5              W[i][j][0] = 0
6              j.pred[i] = i
7          senão se ij ∈ E(D) então
8              W[i][j][0] = w(ij)
9              j.pred[i] = i
10         senão
11             W[i][j][0] = ∞
12             j.pred[i] = null
13 para k = 1 até n, incrementando faça
14     para i = 1 até n, incrementando faça
15         para j = 1 até n, incrementando faça
16             nao_usa_k = W[i][j][k − 1]
17             usa_k = W[i][k][k − 1] + W[k][j][k − 1]
18             se nao_usa_k < usa_k então
19                 W[i][j][k] = nao_usa_k
20             senão
21                 W[i][j][k] = usa_k
22                 j.pred[i] = j.pred[k]
23 devolve W
```

notar como a 3ª dimensão não é tão essencial

PD para Caminhos Mínimos entre Todos os Pares (Bottom-Up)

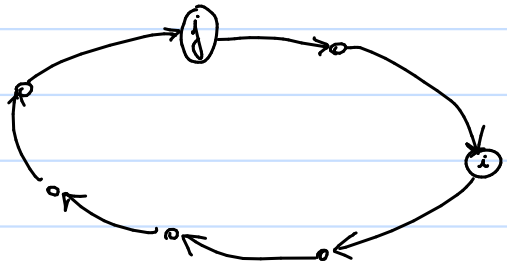
```
FLOYD-WARSHALL-MELHORADO(D, w)
1  Seja W[1..n][1..n] uma matriz
2  para i = 1 até n, incrementando faça
3      para j = 1 até n, incrementando faça
4          se i == j então
5              W[i][j] = 0
6              j.pred[i] = i
7          senão se ij ∈ E(D) então
8              W[i][j] = w(ij)
9              j.pred[i] = i
10         senão
11             W[i][j] = ∞
12             j.pred[i] = null
13 para k = 1 até n, incrementando faça
14     para i = 1 até n, incrementando faça
15         para j = 1 até n, incrementando faça
16             se W[i][j] > W[i][k] + W[k][j] então
17                 W[i][j] = W[i][k] + W[k][j]
18                 j.pred[i] = j.pred[k]
19 devolve W
```

→ versão só com duas dimensões

Ciclos negativos

→ O algoritmo executa normalmente sobre dígrafos que possuem ciclos negativos.
↳ Porém ele <u>não</u> calcula distâncias corretamente

→ Se algum $W[i][i] < 0$ ao fim, existe ciclo negativo em D.



$W[i][j] + W[j][i]$

Problema do Caminho mínimo entre todos os pares

```
RESOLVECAMINHOSENTRETODOSPARES(D, w)
1 W = FLOYD-WARSHALL-BOTTOMUP(D, w)
2 para i = 1 até v(G), incrementando faça
3     se W[i][i] < 0 então
4         devolve null
5 devolve W
```

PD para Caminhos Mínimos entre Todos os Pares (Top-Down)

```
FLOYD-WARSHALL-TOPDOWN(D, w)
1 para i = 1 até n, incrementando faça
2     para j = 1 até n, incrementando faça
3         para k = 0 até n, incrementando faça
4             W[i][j][k] = ∞
5 para i = 1 até n, incrementando faça
6     para j = 1 até n, incrementando faça
7         W[i][j][n] = FLOYD-WARSHALLREC-TOPDOWN(D, w, n, i, j)
8 devolve W
```

```
FLOYD-WARSHALLREC-TOPDOWN(D, w, k, i, j)
1  se W[i][j][k] == ∞ então
2      se k == 0 então
3          se i == j então
4              W[i][j][0] = 0
5              j.pred[i] = i
6          senão se ij ∈ E(D) então
7              W[i][j][0] = w(ij)
8              j.pred[i] = i
9          senão
10             W[i][j][0] = ∞
11             j.pred[i] = null
12     senão
13         nao_usa_k = FLOYD-WARSHALLREC-TOPDOWN(D, w, k − 1, i, j)
14         usa_k = FLOYD-WARSHALLREC-TOPDOWN(D, w, k − 1, i, k) +
             FLOYD-WARSHALLREC-TOPDOWN(D, w, k − 1, k, j)
15         se nao_usa_k < usa_k então
16             W[i][j][k] = nao_usa_k
17         senão
18             W[i][j][k] = usa_k
19             j.pred[i] = j.pred[k]
20 devolve W[i][j][k]
```