

Nejkratší cesty grafem

Nejkratší cesty mezi dvojicemi uzelů. Dijkstra. Nejkratší cesty mezi všemi dvojicemi uzelů. Floyd-Warshall.

Tomáš Bayer | bayertom@fsv.cvut.cz

Katedra geomatiky, fakulta stavební ČVUT.

Obsah přednášky

1 Úvod

2 Nejkratší cesta mezi 2 uzly

- Relaxace hrany
- Dijkstra algoritmus

3 Nejkratší cesty mezi všemi páry uzelů

- Maticová reprezentace problému
- Floyd-Warshallův algoritmus

1. Nejkratší cesta grafem

Často řešená úloha v geoinformatici, logistice, dopravě.

Optimalizace průchodu grafem tak, aby délka cesty byla minimální.

Výsledná cesta je orientovaná.

Varianty problému:

- ① nejkratší cesta ze zadaného uzlu do cílového uzlu,
- ② nejkratší cesta ze zadaného uzlu do každého uzlu,
- ③ nejkratší cesty ze všech uzelů do zadaného uzlu,
- ④ nejkratší cesty mezi všemi dvojicemi uzelů.

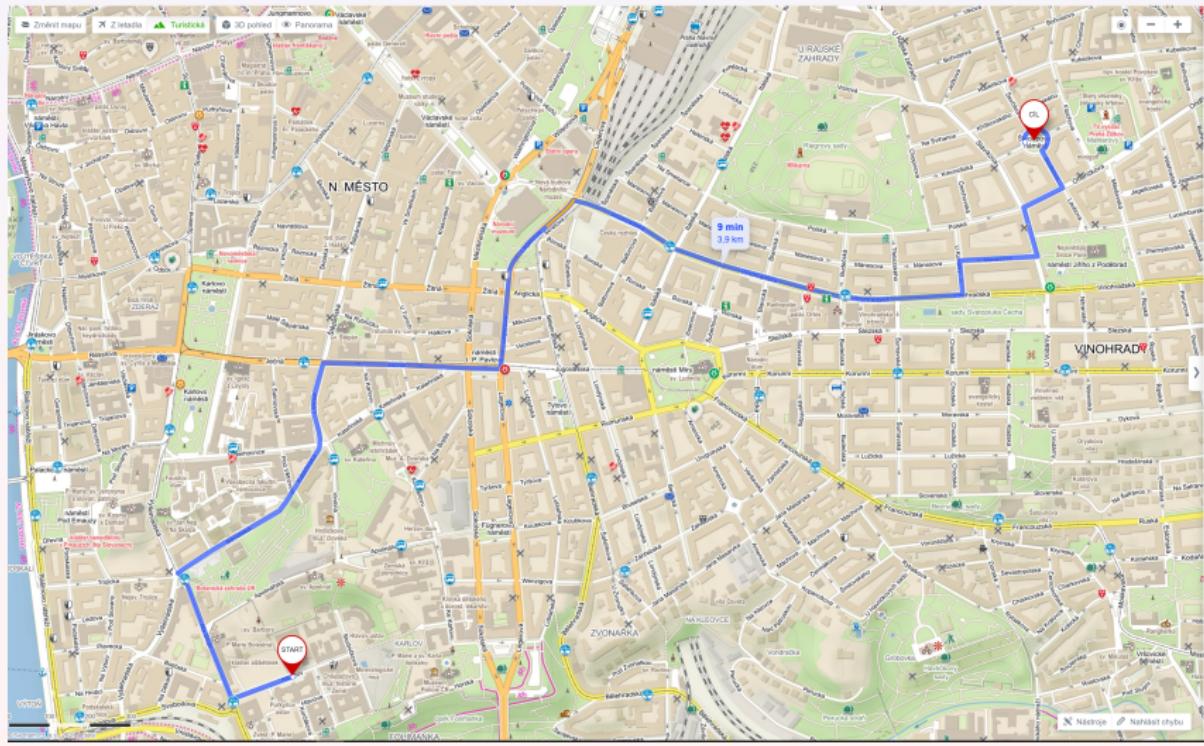
Variantu ad 1) lze poměrně jednoduše převést na ad 2) nebo ad 4).

Variantu ad 3) lze změnou orientace hran převést na ad 2).

Lze aplikovat na orientované/neorientované grafy.

Předpoklad: hrany grafů mají kladné ohodnocení, $w > 0$.

2.Ukázka nejkratší cesty mezi 2 uzly



3. Nejkratší cesta z 1 uzlu

Nejčastěji řešená “dopravní” úloha.

Hledání nejkratší cesty z uzlu s do k v G .

Předpoklady pro G :

Orientovaný, neorientovaný, souvislý, konečný.

Popis G - spojový seznam (úspornější).

Většina metod vychází z BFS: prohledávání do šířky.

Provádí opakovanou relaxaci, netřeba značkovat.

Uzly uloženy v prioritní frontě (místo běžné fronty).

Ohodnocení hran $w > 0$ (obecně $w \in \mathbb{R}$).

Interpretace w : vzdálenost, čas jízdy, náklady, spotřeba, ...

Lze hledat nejkratší, nejlevnější či jinou cestu.

Přehled algoritmů:

- Dijkstra ($w \in \mathbb{R}^+$), běžně používán.
- Bellman+Ford ($w \in \mathbb{R}$), specializované případy.

Použití: navigační SW, logistika, doprava.

4. W -délka a w -vzdálenost

Cesta $C = \langle h_1, \dots, h_k \rangle$ v grafu $G = \langle H, U, \rho \rangle$ tvořena k hranami h s w -délkou d_w

$$d_w(C) = \sum_{i=1}^k w(h_i).$$

W -vzdálenost $d_w(u, v)$ uzelů u, v grafu G je nejmenší w -délka cesty z u do v

$$d_w(u, v) = \min_{\forall C} d_w(C)$$

Nejkratší cesta z vrcholu u do v neexistuje: $d_w(u, v) = \infty$.

Pro každé $u, v, x \in G$ platí(?) axiomy:

- (1) $d_w(u, v) \geq 0$, ("nezápornost" vzdálenosti),
- (2) $d_w(u, v) = 0 \Leftrightarrow u = v$, (identita),
- (3) $d_w(u, v) = d_w(v, u)$, (symetrie, pro neorientované G),
- (4) $d_w(u, v) \leq d_w(u, x) + d_w(x, v)$, (trojúhelníková nerovnost).

Věta o nejkratší cestě:

Pokud $d_w(u, v)$ nejkratší cestou z u do v přes x , pak $d_w(u, x)$ nejkratší cestou z u do x a $d_w(x, v)$ nejkratší cestou z x do v .

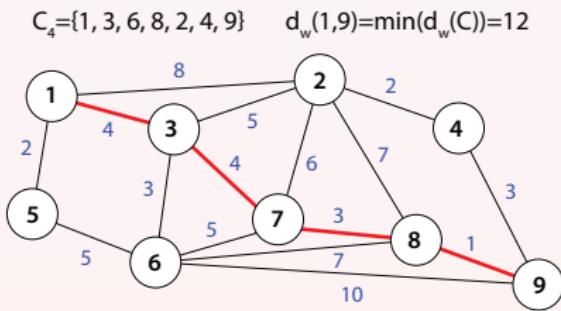
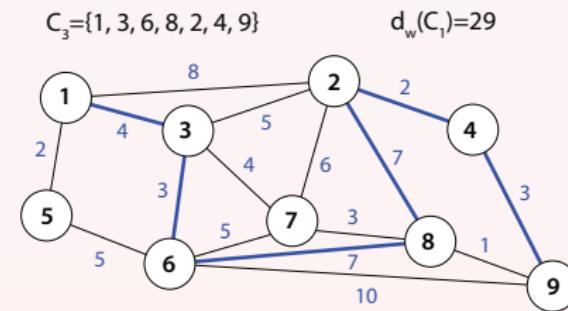
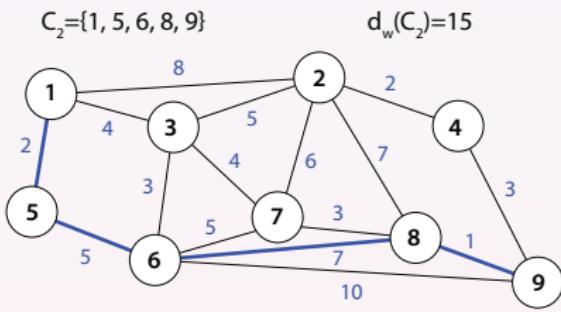
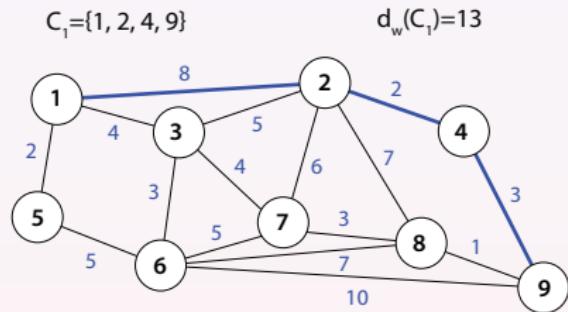
$$d_w(u, v) = d_w(u, x) + d_w(x, v) = d_w(u, x) + w(x, v).$$

Libovolná část nejkratší cesty je též nejkratší, důsledek TN.

5. W–délka vs. w– vzdálenost

Mezi uzly 1, 9 mnoho cest C s různými w –délkami (modré).

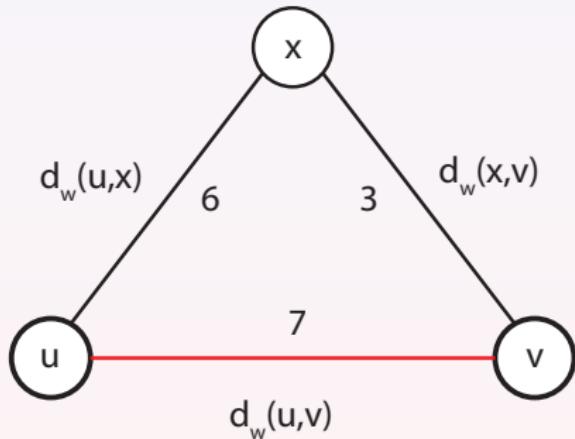
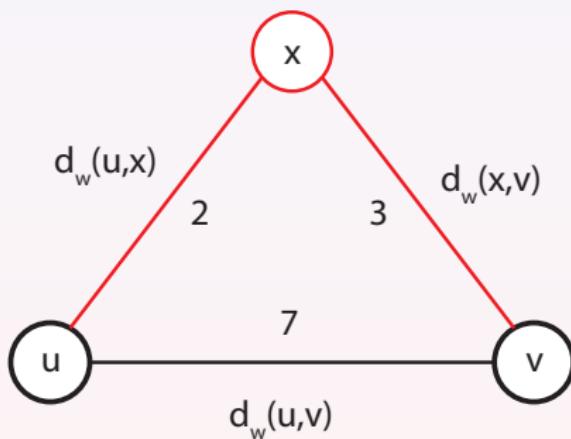
Jedna z nich nejkratší (červeně), definuje w –vzdálenost, $d_w(1, 9) = 12$.



6. Ukázka trojúhelníkové nerovnosti

$$d_w(u,v) = d_w(u,x) + d_w(x,v) = 5$$

$$d_w(u,v) < d_w(u,x) + d_w(x,v) = 7$$



Vlevo, platí věta o nejkratší cestě.

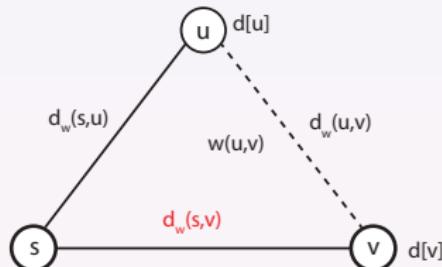
Spojení obou případů

$$d_w(u, v) \leq d_w(u, x) + d_w(x, v).$$

7. Relaxace hrany (u, v)

Nalezení nejkratší cesty z $s \rightarrow v$ + aktualizace předchůdce v .

2 varianty: $s \rightarrow v$ (přímá) nebo $s \rightarrow u \rightarrow v$ (delší).



Hodnoty $d[u]$, $d[v]$ horními odhady $d_w(s, u)$, $d_w(s, v)$.

Trojúhelníková nerovnost platí pro w vzdálenosti, ale i pro horní odhady.

$$d_w(s, v) \leq d_w(s, u) + d_w(u, v) \leq d_w(s, u) + w(u, v) \leq d[v] \leq d[u] + w(u, v),$$

Vlastní relaxace probíhá v opačném směru: opakovánou aktualizací odhadu $d[v]$ hledáme $d_w(s, v)$.

Odhad $d[v]$ se průběžně snižuje.

Aneb první nalezení v neznamená, že jsme k němu došli nejkratší cestou.

Pokud

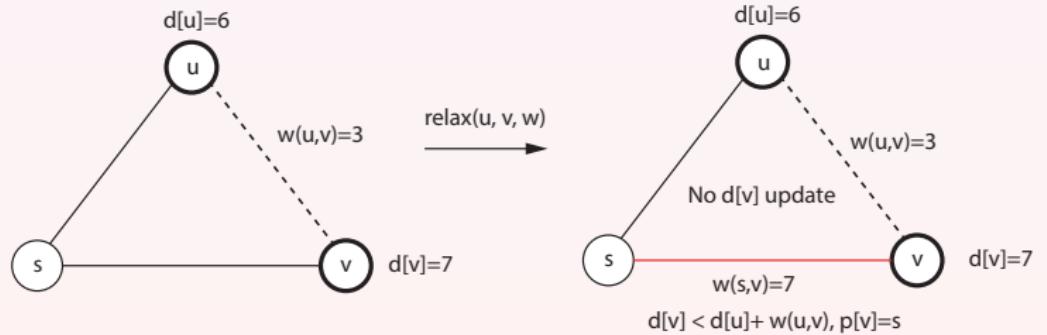
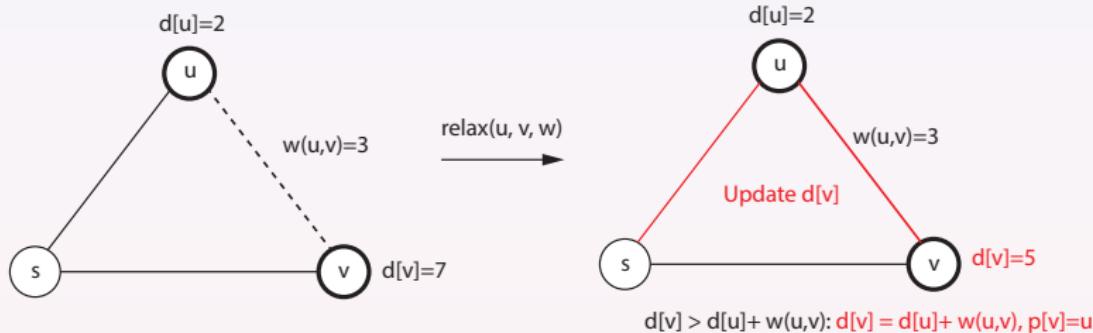
$$d[v] > d[u] + w(u, v),$$

nejkratší cesta vede přes u a $p[v] = u$.

```
def relax(u, v, w):
    if d[v] > d[u]+w(u,v):      #Is path (s, u, v) shorter than (s, v)?
        d[v] = d[u]+w(u,v)       #Update d[v]
        p[v] = u                  #Update predecessor
```

8. Ukázka relaxace hrany (u, v)

Výhodnější cesta $s \rightarrow u \rightarrow v$ nebo $s \rightarrow v$?



9. Dijkstra algoritmus

Autor Edsger W. Dijkstra (1956).

Nejznámější algoritmus pro hledání nejkratších cest mezi 2 uzly.

Ve skutečnosti nejkratší cesta z s do všech ostatních uzelů G .

Zobecňuje strategii BFS + relaxace + prioritní fronta.

Uzly netřeba značkovat.

Předpoklady:

- nezáporné ohodnocení w ,
- G je souvislý.

Snaha o co nejmenší prodloužení cesty.

Realizuje se opakovanou relaxací hrany (u, v) .

Využívá *Greedy strategii*:

Heuristická optimalizace, zde úspěšná (obecně nemusí být).

Hledá globální minimum tak, že v každém kroku hledáme lokální.

Jednoduchá implementace.

10. Princip Dijkstru algoritmu

Hledána nejkratší cesta mezi uzly s a k .

Využívá postupného zpřesňování odhadu nejkratší délky od s do k .

Stávající nejkratší cestu se snažíme co nejméně prodloužit (+ 1 uzel).

Hodnota $d[u]$: aktuální odhad nejkratší vzdálenosti $d_w(s, u)$ k uzlu u .

Hodnota $d[v]$: aktuální odhad nejkratší vzdálenosti $d_w(s, v)$ k uzlu v .

Použití relaxace:

Pokud

$$d[v] > d[u] + w[u][v],$$

pak

$$d[v] = d[u] + w[u][v],$$

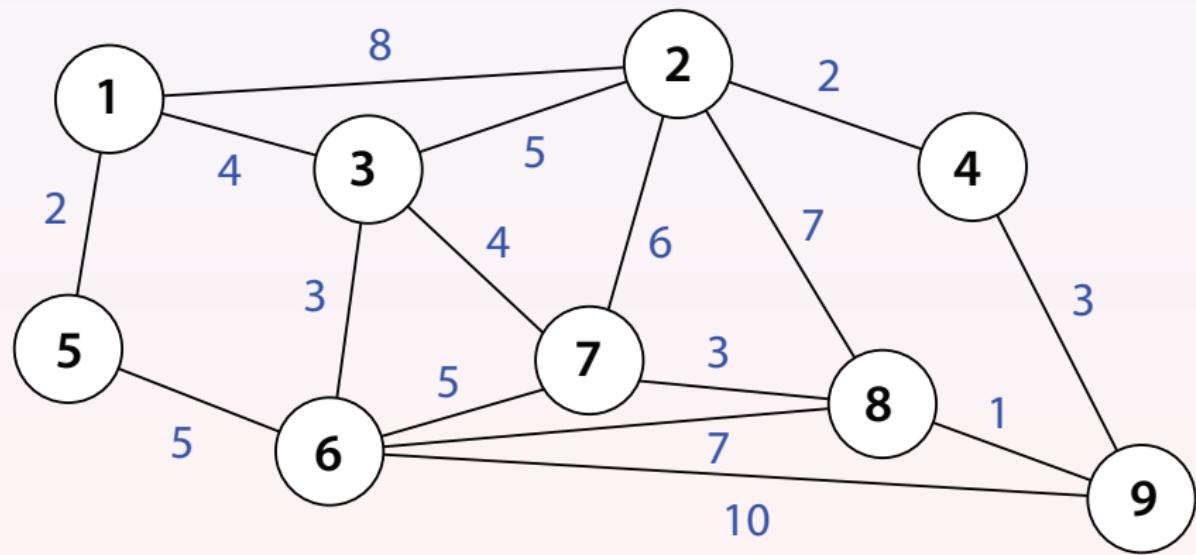
je novým odhad $d[v]$ a

$$p[v] = u.$$

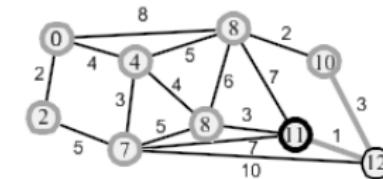
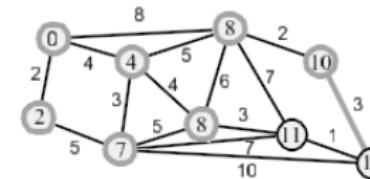
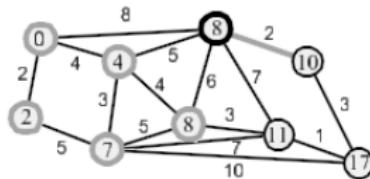
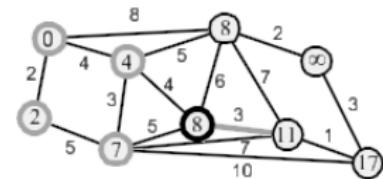
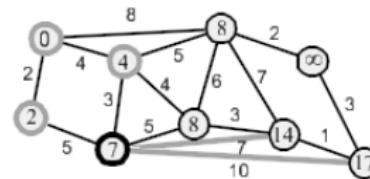
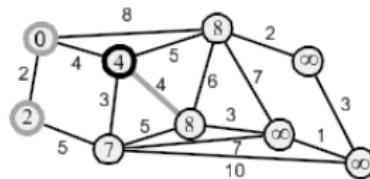
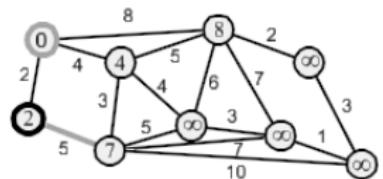
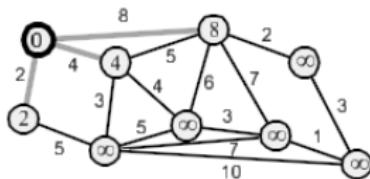
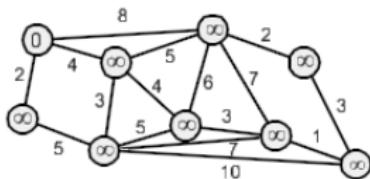
V každém kroku vybíráno uzel u s nejmenší hodnotu $d[u]$.



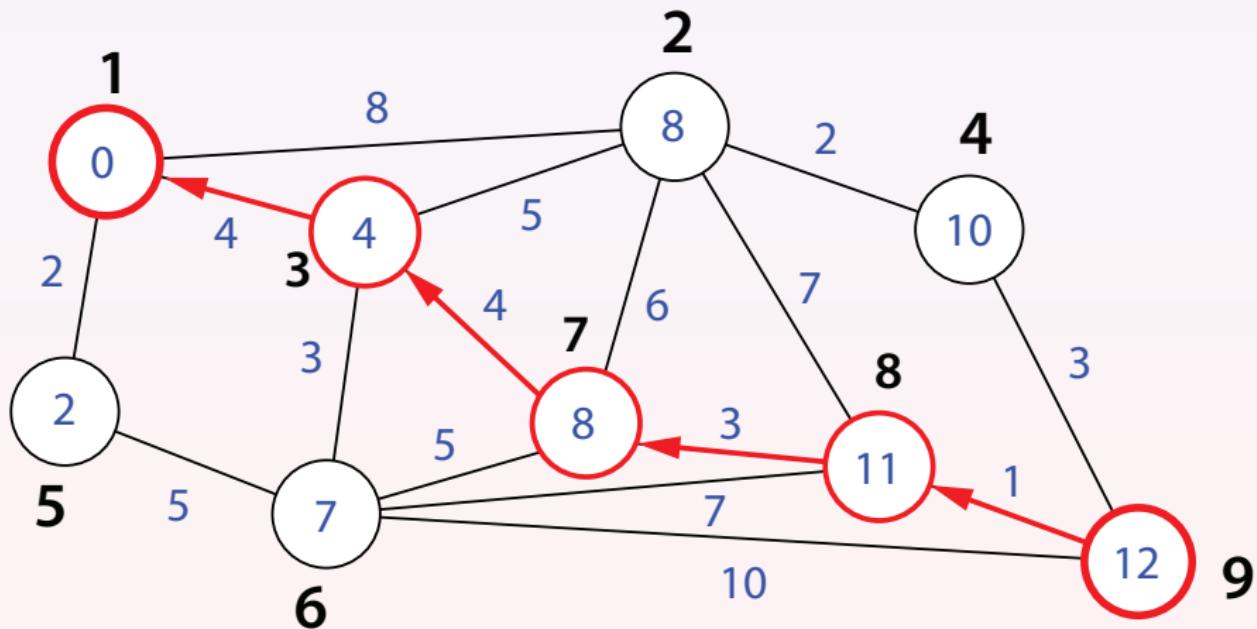
11. Ukázka vstupního grafu



12. Ukázka Dikjstra algoritmu



13. Výsledek Dijkstryho algoritmu



14. Popis Dijkstra algoritmu

U každého uzlu uchováváno: $d[u]$, $p[u]$.

Uzly uloženy v prioritní frontě Q

$$Q = \{d[u], u\}.$$

Netřeba značkovat uzly.

Cesta rekonstruována ze seznamu předchůdců zpětně.

Startovní uzel s , jednotlivé fáze:

1 *Inicializační fáze:*

Inicializace vstupních hodnot: $d(u) = \infty$, $p(u) = -1$.

Nastavení $d[s] = 0$. Přidání $\langle d[s], s \rangle$ do Q .

2 *Iterativní fáze:*

Dokud Q není prázdná:

- Z Q vybrán uzel s nejmenší hodnotou $d(u)$.
- Relaxaci z u na všechny sousedy v :
 - Pokud nové $d[v]$ menší než původní, aktualizujeme.
 - Aktualizujeme předchůdce: $p[v] = u$.
 - Přidáme $\langle d[v], v \rangle$ do Q .

Opakujeme (2), dokud Q není prázdná, tj. existuje nějaký otevřený uzel.

Snadná implementace, analogie BFS.

15. Datový model grafu s ohodnocením

Spojová reprezentace.

V Pythonu použit Dictionary

$$\langle K, V \rangle .$$

Klíč K : uzel.

Hodnota V : seznam incidujících vrcholů s ohodnocením hran.

```
G={V1 : {V1:W1, V2:W2, ..., Vk:Wk},  
  V2 : {V1:W1, V2:W2, ..., Vk:Wk},  
  ...  
  Vk : {V1:W1, V2:W2, ..., Vk:Wk}}
```

Ukázka popisu G :

```
G3 = {  
    1 : {2:8, 3:4, 5:2},  
    2 : {1:8, 3:5, 4:2, 7:6, 8:7},  
    3 : {1:4, 2:5, 6:3, 7:4},  
    4 : {2:2, 9:3},  
    5 : {1:2, 6:5},  
    6 : {3:3, 5:5, 7:5, 8:7, 9:10},  
    7 : {2:6, 3:4, 6:5, 8:3},  
    8 : {2:7, 6:7, 7:3, 9:1},  
    9 : {4:3, 6:10, 8:1}  
}
```

16. Implementace Dijkstra algoritmu

Implementace s prioritní frontou.

```
def dijkstra(G, start, end):
    d = [inf] * (len(G) + 1)                      #Set infinite distance
    p = [-1] * (len(G) + 1)                         #No predecessors
    Q = queue.PriorityQueue()                       #Priority queue
    Q.put((0, start))                             #Add start vertex
    d[start] = 0                                    #Start d[s] = 0
    while not Q.empty():
        du, u = Q.get()                            #Pop first element
        for v, wuv in G[u].items():                #Relaxation, all (u,v)
            if d[v] > d[u] + wuv:                  #We found a better way
                d[v] = d[u] + wuv                   #Update distance
                p[v] = u                           #Update predecessor
                Q.put((d[v], v))                  #Add to Q
```

Ukázka:

```
dijkstra(G, 1, 9)
path(p, 1, 9)
>> 1 3 7 8 9
```

17. Nejkratší cesty mezi všemi páry uzelů

Hledání nejkratších cest mezi všemi dvojicemi uzelů v G .
Méně častá úloha než pro dvojice uzelů.

Pro orientované i neorientované grafy.

Předpoklad: graf neobsahuje záporné w .

Metody:

- *Opakování Dijkstry*

Pro všechny kombinace uzelů.

Neefektivní, používáno jen pro řídké grafy.

- *Specializované algoritmy*

Floyd-Warshal, prodlužování w -délky.

Přechod od spojové reprezentaci k maticové: D , W , P .

“Kombinace” matice sousednosti a ohodnocení.

Matice w -délék a w -vzdáleností, předchůdců.

18. Matice w-délek W

Graf $G = \langle H, U, \rho \rangle$, ohodnocení $w : H \mapsto \mathbb{R}$.

Čtvercová matice $W = [w_{i,j}]$ řádu n je maticí w -délek

$$w_{ij} = \begin{cases} 0, & \text{pokud } i = j, \\ w(u_i, u_j), & \text{pokud } i \neq j, (u_i, u_j) \in H, \\ \infty, & \text{pokud } i \neq j, (u_i, u_j) \notin H. \end{cases}$$

Pro G lze snadno sestavit.

Pro neorientovaný graf symetrická, kombinuje incidenci a ohodnocení.

Výchozí vstupní matice pro výpočty.

Reprezentace v Pythonu: 2D seznam

```
W=[  
    [w11, w12, ..., w1n],  
    [w21, w22, ..., w2n],  
    ...  
    [wn1, wn2, ..., wnn],  
]
```

19. Matice w-vzdáleností D

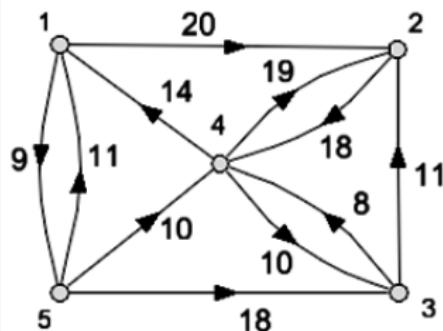
Čtvercová matice $D = [d_{i,j}]$ řádu n je maticí w vzdáleností

$$d_{ij} = d_w(u_i, u_j).$$

Nejkratší vzdálenosti mezi páry uzelů.

Pro neorientovaný G symetrická.

Získána následným výpočtem, výsledek grafových algoritmů.



$$W = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 20 & \infty & \infty & 9 \\ 2 & \infty & 0 & \infty & 19 & \infty \\ 3 & \infty & 11 & 0 & 10 & \infty \\ 4 & 14 & 18 & 8 & 0 & \infty \\ 5 & 11 & \infty & 18 & 10 & 0 \end{pmatrix}$$

$$D = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 20 & 27 & 19 & 9 \\ 2 & 33 & 0 & 27 & 19 & 42 \\ 3 & 24 & 11 & 0 & 10 & 33 \\ 4 & 14 & 18 & 8 & 0 & 23 \\ 5 & 11 & 28 & 18 & 10 & 0 \end{pmatrix}$$

Ilustruje dosažitelnost uzelů z jiných uzelů (konečné vzdálenosti).

20. Matice předchůdců P

Čtvercová matice $P = [p_{i,j}]$ řádu n je maticí předchůdců

$$p_{ij} = \begin{cases} 0, & \text{pokud } i = j \text{ nebo } \nexists \text{ min. cesta } u_i \rightarrow u_j, \\ u_k, & u_k \text{ předchůdce } u_j, \exists \text{ min. cesta } u_i \rightarrow u_j. \end{cases}$$

Analogie s předchůdcem uzlu ve spojové reprezentaci.

Prvek p_{ij} obsahuje analogickou informaci jako 1D pole předchůdců.

Umíme přímo určit její approximaci $P^{(0)}$ pro 0 vnitřních uzelů:

$$P^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 0 & 5 & 5 & 0 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 5 & 5 & 1 \\ 4 & 0 & 4 & 2 & 1 \\ 4 & 3 & 0 & 3 & 1 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 4 & 5 & 5 & 0 \end{bmatrix}.$$

Matice P získána následným výpočtem, výsledek grafových algoritmů.

Umožňuje rekonstrukci *nejkratší cesty*.

Provádí se zpětně, od koncového bodu k počátečnímu: rekurze, iterace,



21. Rekonstrukce nejkratší cesty, nerekurzivní

Od posledního uzlu j postupujeme k prvnímu uzlu i .

Nutno prohodit počáteční a koncový uzel.

```
def path(P, i, j):                      #Path from i to j
    print(j)                            #print end vertex j
    while i != j:                      #Until we start=stop
        j = P[i][j]                     #Predecessor P[i][j]
        if j == 0:                       #No predecessor, stop
            print("No Path.")
            return
    print(j)                            #Print point
```

22. Floyd-Warshallův algoritmus

Nejčastější metoda hledání nejkratších cest mezi páry uzelů.

Předpoklad: G nemá hrany $w < 0$.

Postupné zpřesňování rozšiřováním množiny vnitřních uzelů.

Vnitřní uzel cesty.

Vnitřní uzel cesty

$$P = \langle u_1, u_2, u_3, \dots, u_{k-1}, u_k \rangle ,$$

je její libovolný nekrajní uzel

$$\{u_2, u_3, \dots, u_{k-1}\} .$$

Princip F-W algoritmu.

Postupné rozšiřování vnitřních uzelů na všechny uzly v grafu.

V každém kroku se jeden uzel stává vnitřním a hledáme cesty přes něj.

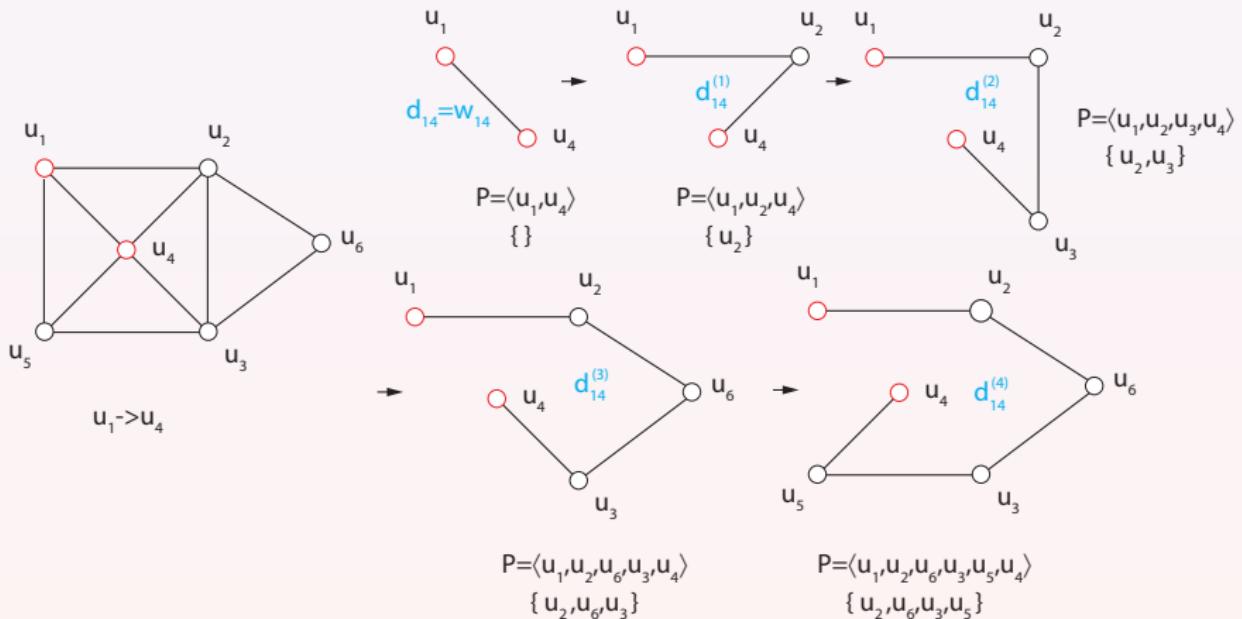
Snadná implementace, trojice vnořených cyklů.

Složitost $O(\|U\|^3)$.

Řídké grafy: n -násobné použití Dijkstry + B-F.

23. Rozšiřování vnitřních uzlů

Postupným rozšiřováním P přes všechny vnitřní uzly neminejme případnou nejkratší cestu. Rozšiřujeme pouze, pokud cesta přes vnitřní uzel kratší.



24. Matice $D^{(k)}$ w-délek

Modifikovaná matice w-délek $D^{(k)}$

$$D^{(k)} = [d_{ij}^{(k)}],$$

kde $d_{ij}^{(k)}$ je nejkratší w-délka mezi uzly (u_i, u_j) tvořená k vnitřními uzly

$$\{u_1, u_2, u_3, \dots, u_k\}.$$

Platí

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j}, & k = 0, \\ \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}), & k \geq 1. \end{cases}$$

Nejkratší cesta $u_i \rightarrow u_j$ s k vnitřními uzly:

- nejde přes u_k , pak $d_{i,j}^{(k)} = d_{i,j}^{(k-1)}$,
- jde přes u_k , lze rozdělit na $u_i \rightarrow u_k$ a $u_k \rightarrow u_j$.

Ověření, zda cesta $u_i \rightarrow u_j$ není horší než cesta z uzlu $u_i \rightarrow u_j$ přes u_k .

Pokud cesta přes u_k výhodnější, aktualizován předchůdce $p_{ij} = p_{kj}$, obdoba relaxace.

Cíl F-W algoritmu:

Postupný převod $D^{(0)} \equiv W \rightarrow D^{(k)} \equiv D$.

$$D^{(0)} = \begin{bmatrix} 0 & 20 & \infty & \infty & 9 \\ \infty & 0 & \infty & 18 & \infty \\ \infty & 11 & 0 & 10 & \infty \\ 14 & 19 & 8 & 0 & \infty \\ 11 & \infty & 18 & 10 & 0 \end{bmatrix}, \quad D^{(5)} = \begin{bmatrix} 0 & 20 & 27 & 19 & 9 \\ 33 & 0 & 26 & 18 & 41 \\ 24 & 11 & 0 & 10 & 33 \\ 14 & 19 & 8 & 0 & 23 \\ 11 & 29 & 18 & 10 & 0 \end{bmatrix} = D.$$

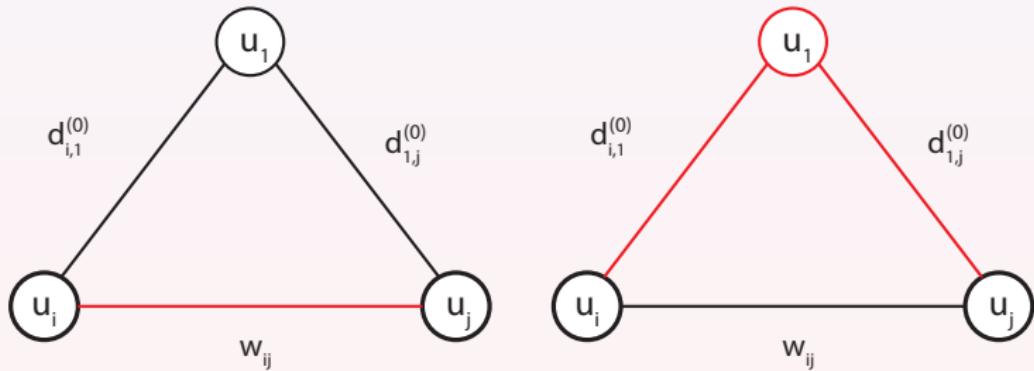
25. Princip F-W algoritmu

Varianta 1: $P_{i,j} = \langle u_i, u_j \rangle, \{\emptyset\}, d_{ij}^{(0)} = w_{i,j}$.

Varianta 2: $P_{i,j} = \langle u_i, u, u_{1j} \rangle, \{u_1\}, d_{i,j}^{(1)} = d_{i,1}^{(0)} + d_{1,j}^{(0)}$.

Nejkratší cesta minimum z obou variant

$$d_{i,j}^{(1)} = \min \left\{ d_{i,j}^{(0)}, d_{i,1}^{(0)} + d_{1,j}^{(0)} \right\}.$$



Lze zobecnit pro k vnitřních uzlů

$$d_{i,j}^{(k)} = \min \left\{ d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \right\}.$$

26. Matice $P^{(k)}$

Modifikovaná matice předchůdců s k vnitřními uzly

$$P^{(k)} = [p_{ij}].$$

Prvek p_{ij} předchůdcem u_j na cestě $\langle u_i, u_j \rangle$.

Matice $P^{(0)}$

$$p_{i,j}^{(0)} = \begin{cases} 0, & i = j \vee w_{i,j} = \infty, \\ i, & \text{jinak.} \end{cases}$$

Žádný vnitřní uzel, předchůdcem uzlu u_j hrany $\{u_i, u_j\}$ uzel u_i .

$$P^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 0 & 5 & 5 & 0 \end{bmatrix}, \quad P^{(5)} = \begin{bmatrix} 0 & 1 & 5 & 5 & 1 \\ 4 & 0 & 4 & 2 & 1 \\ 4 & 3 & 0 & 3 & 1 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 4 & 5 & 5 & 0 \end{bmatrix} \equiv P.$$

Aktualizace $P^{(k)}$ sleduje aktualizační pravidlo $D^{(k)}$

$$p_{i,j}^{(k)} = \begin{cases} p_{i,j}^{(k-1)}, & d_{i,j}^{(k-1)} \leq d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}, \\ p_{k,j}^{(k-1)}, & d_{i,j}^{(k-1)} > d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}. \end{cases}$$

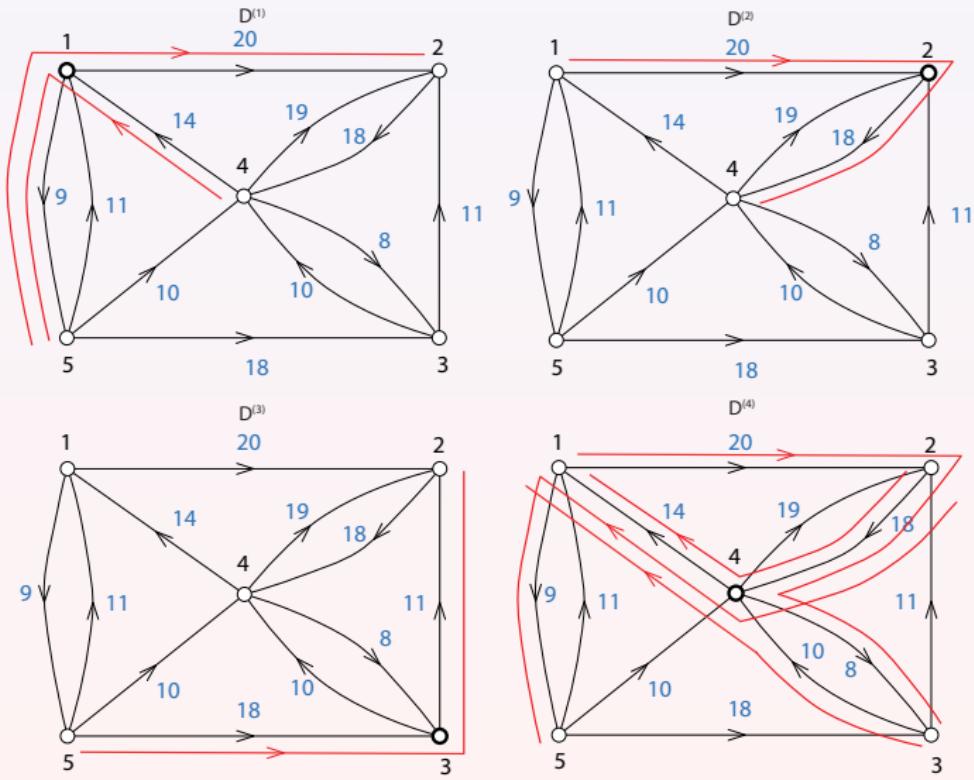
Pokud výhodnější cesta přes u_k , aktualizace předchůdce uzlu u_j na u_k .

Rekonstrukce cesty $u_i \rightarrow u_j$ zpětně od koncového uzlu u_j : procházíme i -tý řádek P

Cíl F-W algoritmu:

Postupný převod $P^{(0)} \rightarrow P^{(k)} \equiv P$.

27. Ukázka funkcionality Floyd-Warshall



28. Implementace Floyd-Warshall

Snadná implementace.

Tři vnořené cykly + neúplná podmínka.

```
def fw(P, W):
    D = W
    n = len(W)
    for k in range(0, n):
        for i in range(0, n):
            for j in range(0, n):
                if D[i][j] > D[i][k] + D[k][j]:
                    D[i][j] = D[i][k] + D[k][j]
                    P[i][j] = P[k][j]
    return P, D
```

Doba běhu $O(\|U\|^3)$.

Neefektivní pro velké řídké grafy.

29. Výsledky, Floyd-Warshall, 1/2

Matice W, P :

```
W = [
    [0, 20, inf, inf, 9],
    [inf, 0, inf, 18, inf],
    [inf, 11, 0, 10, inf],
    [14, 19, 8, 0, inf],
    [11, inf, 18, 10, 0],
]
P = [
    [0, 1, 0, 0, 1],
    [0, 0, 0, 2, 0],
    [0, 3, 0, 3, 0],
    [4, 4, 4, 0, 0],
    [5, 0, 5, 5, 0],
]
```

Volání funkce:

```
P,D = fw(P, W)
```

Matice $D^{(2)} - D^{(5)}$:

$$D^{(1)} = \begin{bmatrix} 0 & 20 & \infty & \infty & 9 \\ \infty & 0 & \infty & 18 & \infty \\ \infty & 11 & 0 & 10 & \infty \\ 14 & 19 & 8 & 0 & 23 \\ 11 & 31 & 18 & 10 & 0 \end{bmatrix}, D^{(2)} = \begin{bmatrix} 0 & 20 & \infty & 38 & 9 \\ \infty & 0 & \infty & 18 & \infty \\ \infty & 11 & 0 & 10 & \infty \\ 14 & 19 & 8 & 0 & 23 \\ 11 & 31 & 18 & 10 & 0 \end{bmatrix}, D^{(3)} = \begin{bmatrix} 0 & 20 & \infty & 38 & 9 \\ \infty & 0 & \infty & 18 & \infty \\ \infty & 11 & 0 & 10 & \infty \\ 14 & 19 & 8 & 0 & 23 \\ 11 & 29 & 18 & 10 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 20 & 46 & 38 & 9 \\ 32 & 0 & 26 & 18 & 41 \\ 24 & 11 & 0 & 10 & 33 \\ 14 & 19 & 8 & 0 & 23 \\ 11 & 29 & 18 & 10 & 0 \end{bmatrix}, D^{(5)} = \begin{bmatrix} 0 & 20 & 27 & 19 & 9 \\ 33 & 0 & 26 & 18 & 41 \\ 24 & 11 & 0 & 10 & 33 \\ 14 & 19 & 8 & 0 & 23 \\ 11 & 29 & 18 & 10 & 0 \end{bmatrix} = D.$$

30. Výsledky, Floyd-Warshall, 2/2

Postup:

$D^{(1)}$: hledáme všechny $u_i \rightarrow u_j$ přes u_1

1 → (1) → 2, 1 → (1) → 3, 1 → (1) → 4, 1 → (1) → 5: 2 → (1) → 1, 2 → (1) → 3, 2 → (1) → 4, 2 → (1) → 5.

3 → (1) → 1, 3 → (1) → 2, 3 → (1) → 4, 3 → (1) → 5: 4 → (1) → 1, 4 → (1) → 2, 4 → (1) → 3, 4 → (1) → 5.

5 → (1) → 1, 5 → (1) → 2, 5 → (1) → 3, 5 → (1) → 4: 4 → (1) → 1, 4 → (1) → 2, 4 → (1) → 3, 4 → (1) → 5.

$D^{(2)}$: hledáme všechny $u_i \rightarrow u_j$ přes u_2

1 → (2) → 2, 1 → (2) → 3, 1 → (2) → 4, 1 → (2) → 5: 2 → (2) → 1, 2 → (2) → 3, 2 → (2) → 4, 2 → (2) → 5.

3 → (2) → 1, 3 → (2) → 2, 3 → (2) → 4, 3 → (2) → 5: 4 → (2) → 1, 4 → (2) → 2, 4 → (2) → 3, 4 → (2) → 5,

5 → (2) → 1, 5 → (2) → 2, 5 → (2) → 3, 5 → (2) → 4: 4 → (2) → 1, 4 → (2) → 2, 4 → (2) → 3, 4 → (2) → 5.

Matice $P^{(0)} - P^{(5)}$:

$$P^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 0 & 5 & 5 & 0 \end{bmatrix}, \quad P^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 1 & 5 & 5 & 0 \end{bmatrix}, \quad P^{(2)} = \begin{bmatrix} 0 & 1 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 1 & 5 & 5 & 0 \end{bmatrix},$$

$$P^{(3)} = \begin{bmatrix} 0 & 1 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 3 & 5 & 5 & 0 \end{bmatrix}, \quad P^{(4)} = \begin{bmatrix} 0 & 1 & 4 & 2 & 1 \\ 4 & 0 & 4 & 2 & 1 \\ 4 & 3 & 0 & 3 & 1 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 4 & 5 & 5 & 0 \end{bmatrix}, \quad P^{(5)} = \begin{bmatrix} 0 & 1 & 5 & 5 & 1 \\ 4 & 0 & 4 & 2 & 1 \\ 4 & 3 & 0 & 3 & 1 \\ 4 & 4 & 4 & 0 & 1 \\ 5 & 4 & 5 & 5 & 0 \end{bmatrix} = P.$$