(2.4) Grafové algoritmy a jejich složitost:

■ **PROHLEDÁVÁNÍ DO HLOUBKY:** (DFS) každý uzel má 2 časové značky OPEN, CLOSE

$\Theta(|U|+|H|)$

(1.) INICIALIZACE; // všechny uzly na FRESH

(2.) for ∀u ∈ U do
  if (stav[u] = FRESH) then DFS_PROJDI(u)

(3.) DFS_PROJDI(u)
  stav[u] = OPEN; ← časová značka
  i = i++; d[u] = i;
  for ∀v = nasl(u) do
    if (stav[v] = FRESH) then
      p[v] = u
      DFS_PROJDI(v);
  stav[u] = CLOSED;
END;

REKURZE

↳ zde je možno detekovat typ hrany (stromová, zpětná, dopředná ...)

(1) Inicializace uzlů
stav[s] = OPEN,
(2) INIT_QUEUE(s),
(3) while NOT EMPTY_QUEUE do {
  u := QUEUE.first();
  for ∀v ∈ nasl[u] do {
    if (v = FRESH) { stav[v] = OPEN
      ENQUEUE(v) }
  }
  DEQUEUE;
  stav[v] = CLOSED;
}

■ **PROHLEDÁVÁNÍ DO ŠÍRKY** (BFS) každý uzel má 3 časové značky ← FRESH, OPEN, CLOSE

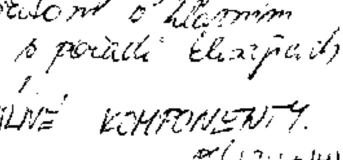
$\Theta(|U|+|H|)$ – výsledkem jsou max. cesty z poč. uzlu.

■ **TOPOLOGICKÉ USPOŘÁDÁNÍ:**

↳ uspořádání uzlů do jedné řady a orientované hrany směřují jedním směrem
↳ pouze u acyklických grafů (cyklické grafy nemají topolog. uspořádání)

(1) Vytvoř prázdný seznam uzlů
(2) Pomocí DFS(G) spočítej časové značky uzavření značek.
(3) V okamžiku uzavření ulož ∀ uzel na začátek seznamu.

} => TOP SORT



$\Theta(|U|+|H|)$

■ **Silná souvislost.**

• platí pro libovolnou dvojici (u,v) ≡ spojem z uzlu u do uzlu v a z uzlu v do uzlu u.
↳ silná komponenta: ← každý max. silně souvislý podgraf

(1) Provedl DFS(G) => značky
(2.) vytvoř opačně orientovaný graf G^-
(3.) proveď DFS(G), přičemž v hlavním cyklu probíhej uzly v pořadí klesajících značek z kroku 1.
↳ ROZKLAD NA SILNÉ KOMPONENTY. $\Theta(|U|+|H|)$

■ **MINIMÁLNÍ KOSTRA GRAFU** (BORŮVKA - KRUSKAL) (Jarník - PRIM)

$|H| = |U| - 1$

A := ∅
for ∀u ∈ U do Make-Set(u)
uspořádej hrany podle váhy w
for (∀h ∈ H) do
  if (FIND-SET(u) ≠ FIND-SET(v)) then {
    A := A ∪ {(u,v)}
    UNION(u,v)
  }
return A

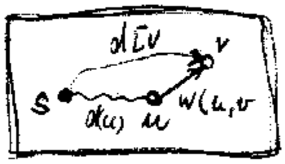
Postupně přidávám hranu s nejmenší váhou => slučování stromů

Prioritní fronta Q uspořádaná podle klíčů

Q = U for ∀u ∈ Q do d[u] = ∞;
d[x] = 0, p[x] = NIL;
while Q ≠ 0 do {
  u := EXTRACT-MIN(Q)
  for ∀v ∈ nasl[u] do {
    if (v ∈ Q) and (w(u,v) < d[v]) then
      p[v] := u,
      d[v] = w(u,v).
  }
}

extract min.

DIJKSTRŮV ALGORITMUS NEJKRATŠÍCH CEST: $(w_i > 0)$



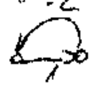
(1) Init Path(); // (d[u] = ∞) d[s] := 0;
Init QUEUE();
(2) for ∀u ∈ U do ENQUEUE(Q,u)
(3) while not Empty(Q) do
    u := EXTRACT-MIN(Q)
    S := S ∪ {u}
    for ∀v ∈ nasl[u] do RELAX(u,v,w)
}

RELAX(u,v,w){
  if d[v] > d[u] + w(u,v) then
    d[v] := d[u] + w(u,v)
    p[v] := u;
}

$O(|H| \ln |U|)$

BELLMANŮV - FORDŮV ALGORITMUS NEJKRATŠÍCH CEST

– detekuje záporný cyklus



(1) Init_Path()
(2) for i=1 to |U|-1 do
    for ∀(u,v) ∈ H do RELAX(u,v,w)
(3) for ∀(u,v) ∈ H do
    if d[v] > d[u] + w(u,v) then return false; => záp cyklus
    else return true;

$O(|U| \cdot |H|)$

Pro Acyklický graf:

1. TOPSORT - topolog. uspoř.
2. Init-Path
3. for ∀u ∈ TopSort do
   for ∀v ∈ nasl[u] do Relax(u,v,w)

NEJKRATŠÍ CESTY MEZI VŠEMI UZLY:

- opakovaným použitím DIJKSTRY nebo B-Forda. (n^3)
- Matice w-délek $W = [w_{ij}]$
- Matice vzdáleností $D = [d_{ij}]$
- Matice předchůdců $P = [p_{ij}]$

$D^1 = W \quad D^2 = W^2 \quad \dots \quad W^{n-1}$ max.

$O(n^3 \log n)$ $n = U$.

DYNAMICKÉ PROGRAMOVÁNÍ:

- rozložení problému na podproblémy, ty vyřešit a zkombinovat (např. FIBBONACCI … pamatuji si poslední 2 prvky)
- řešení ZDOLA NAHORU => lineární nebo kvadratická složitost

HLADOVÝ ALGORITMUS

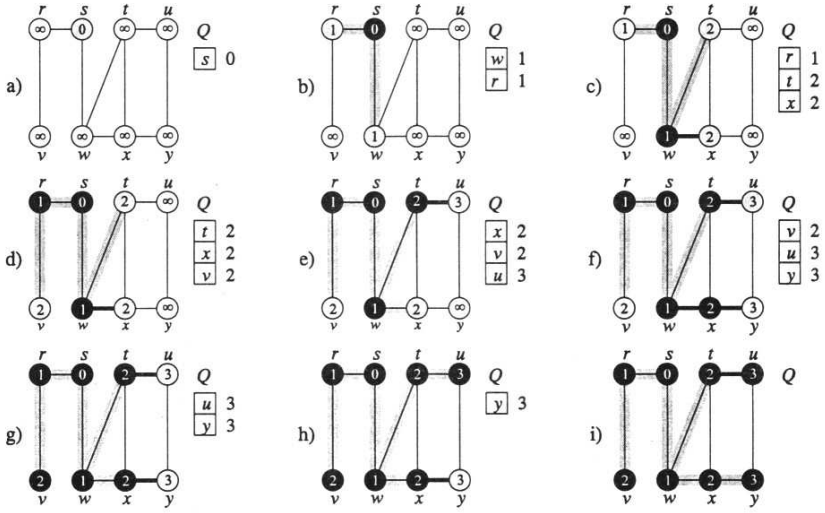
Vlastnost: ke globálnímu optimálnímu výběru lze dospět prostřednictvím (lokálně) optimistických hladových výběrů.
Podmínka výběru nezávisí na řešení podproblému, ale jen na stavu.
↳ např. algoritmus (JARNIK - PRIM).

7. Grafové algoritmy a jejich složitost

Prohledávání

Do šířky (BFS Breadth-First Search) - *O(|U| + |H|)*

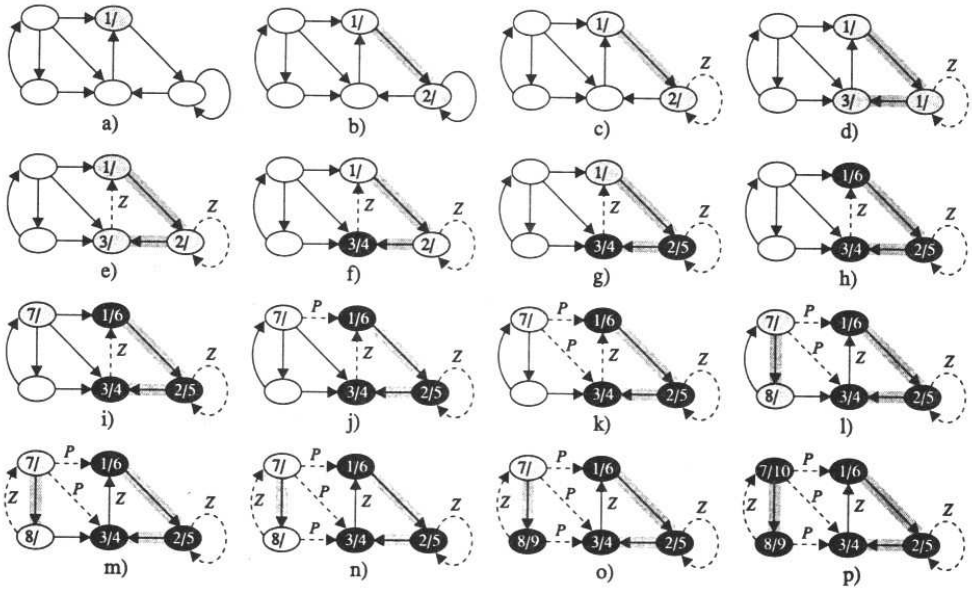
- Každý uzel má 3 časové značky (FRESH, OPEN, CLOSE)



Obrázek 4.7: Prohledávání do šířky

Do hloubky(DFS = Depth-First Search) - *O(|U| + |H|)*

- Každý uzel má 2 časové značky (OPEN, CLOSE)
- Rekurze



Obrázek 4.8: Prohledávání do hloubky

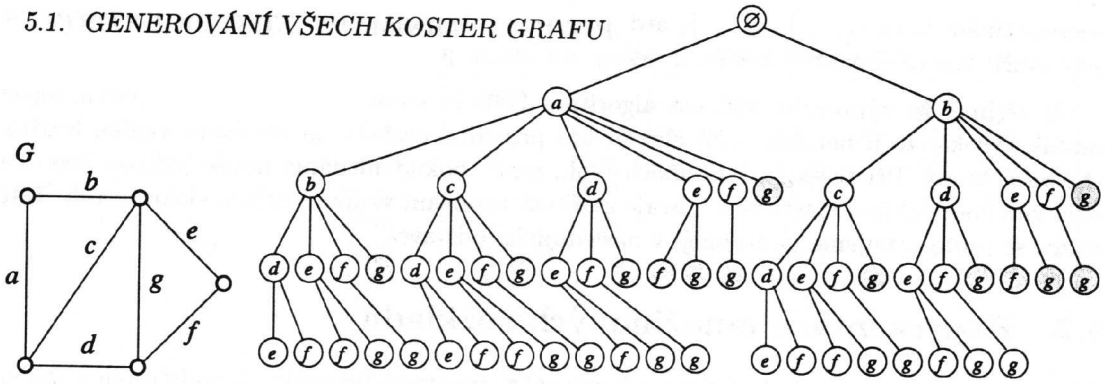
Silná souvislost - O(|*U*| + |*H*|)

Algoritmus 4.31 Rozklad na silné komponenty

S-COMP(*G*)

1 proveď DFS(*G*), který určí časovou značku uzavření $f[u]$ každého uzlu u
2 vytvoř opačně orientovaný graf G^-
3 proveď DFS(G^-), přitom v hlavním cyklu DFS probírej uzly v pořadí klesající hodnoty $f[u]$ (získané v kroku 1)
4 rozklad množiny uzlů podle jednotlivých stromů DF-lesa získaného v kroku 3 určuje rozklad na silné komponenty

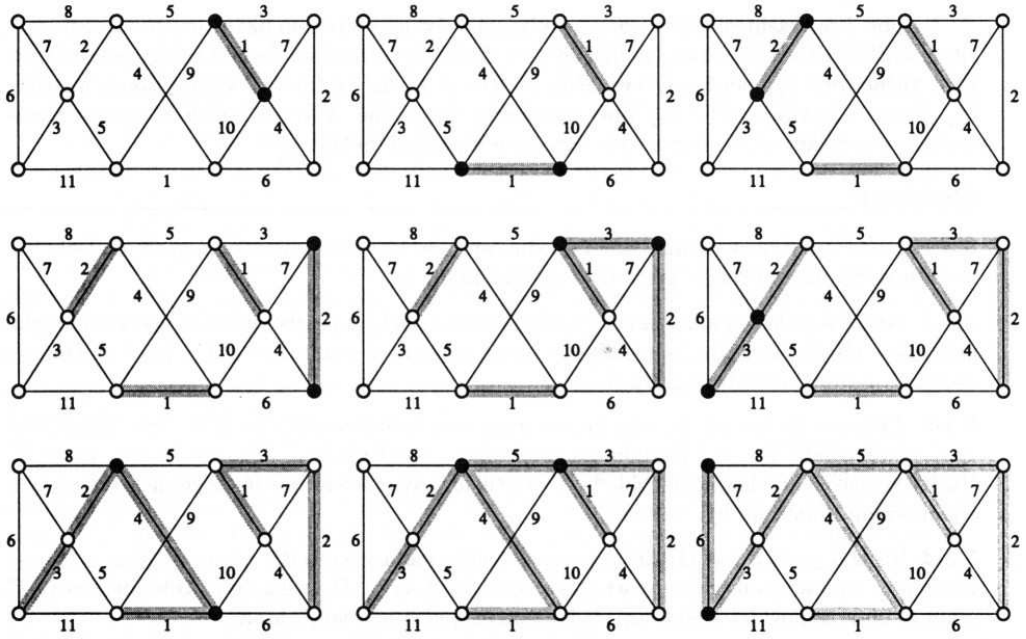
Všechny kostry grafu



Obrázek 5.1: Generování koster grafu

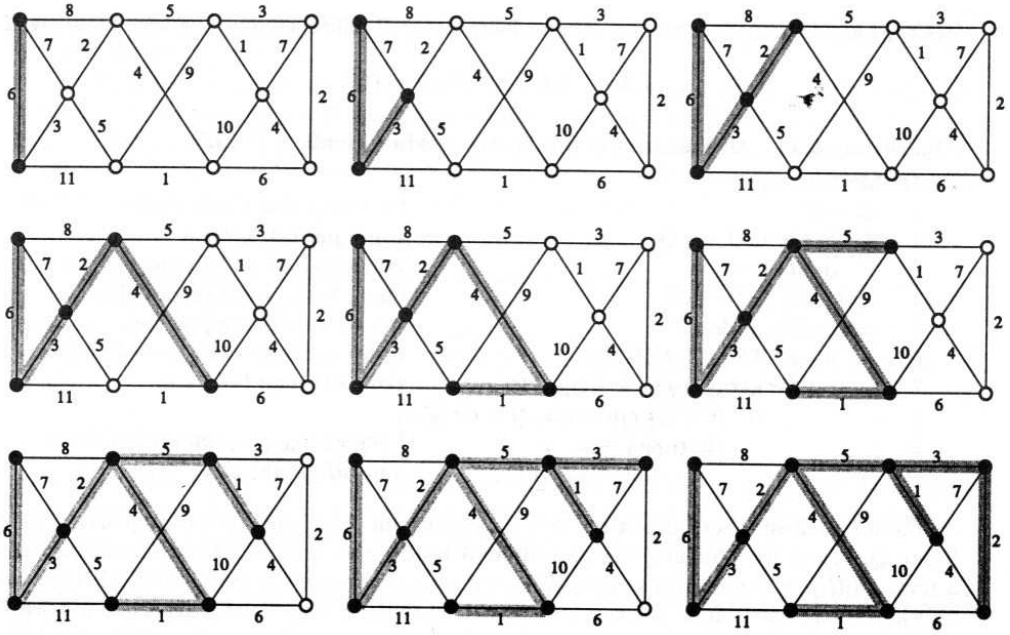
Minimální kostra grafu

Borůvka – Kruskal (O(|*H*| log(|*H*|)))



Obrázek 5.6: Provádění Borůvkova-Kruskalova algoritmu

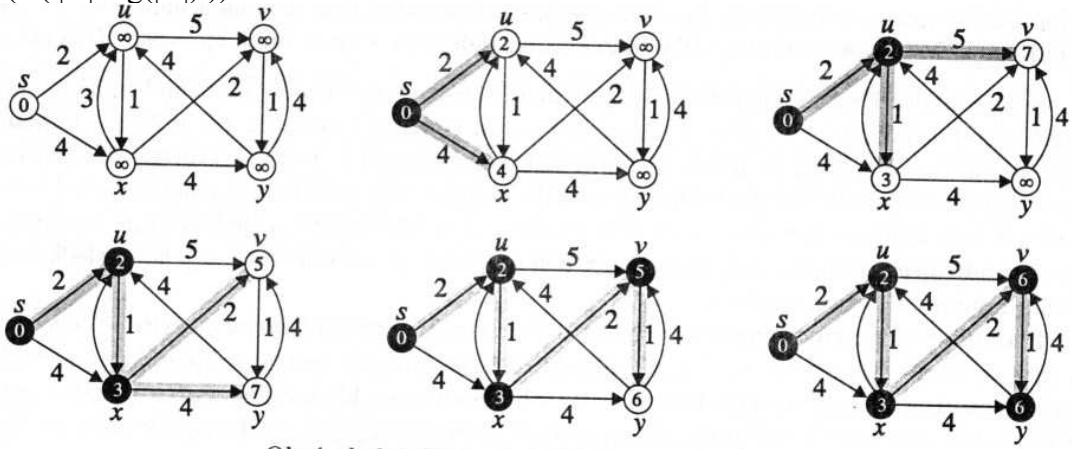
Jarník – Prim (O($|H|$ log($|U|$)))



Obrázek 5.7: Provádění Jarníkova-Primova algoritmu

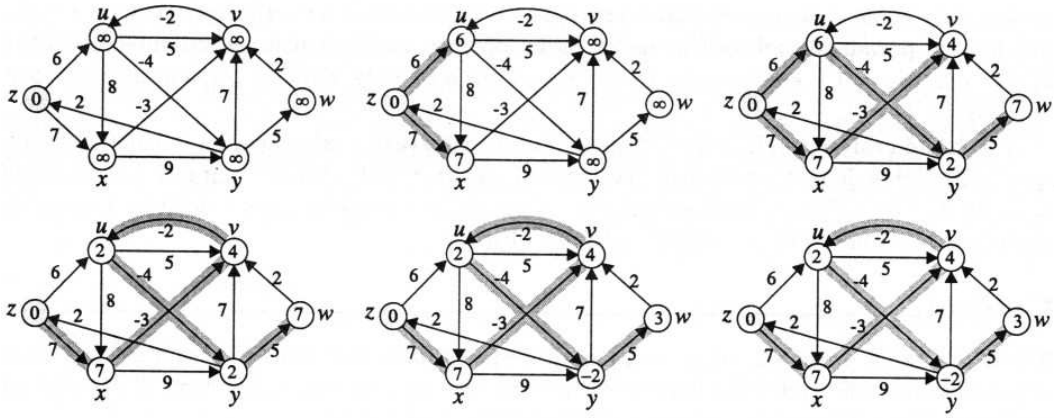
Nejkratší cesty

Dijkstra (O($|H|$ log($|U|$)))



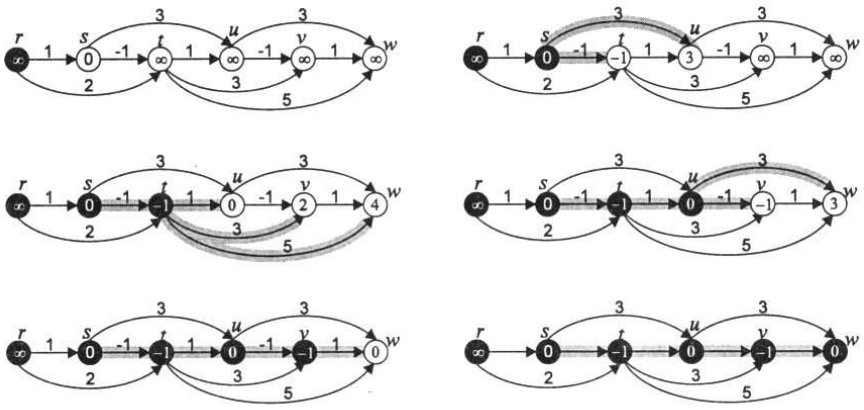
Obrázek 6.4: Provedení Dijkstrova algoritmu

Belman Ford (O($|U|$ * $|H|$))



Obrázek 6.6: Provedení Bellmanova-Fordova algoritmu

Nejkratší cesty pro acyklický graf



Obrázek 6.7: Hledání nejkratších cest na acyklickém grafu

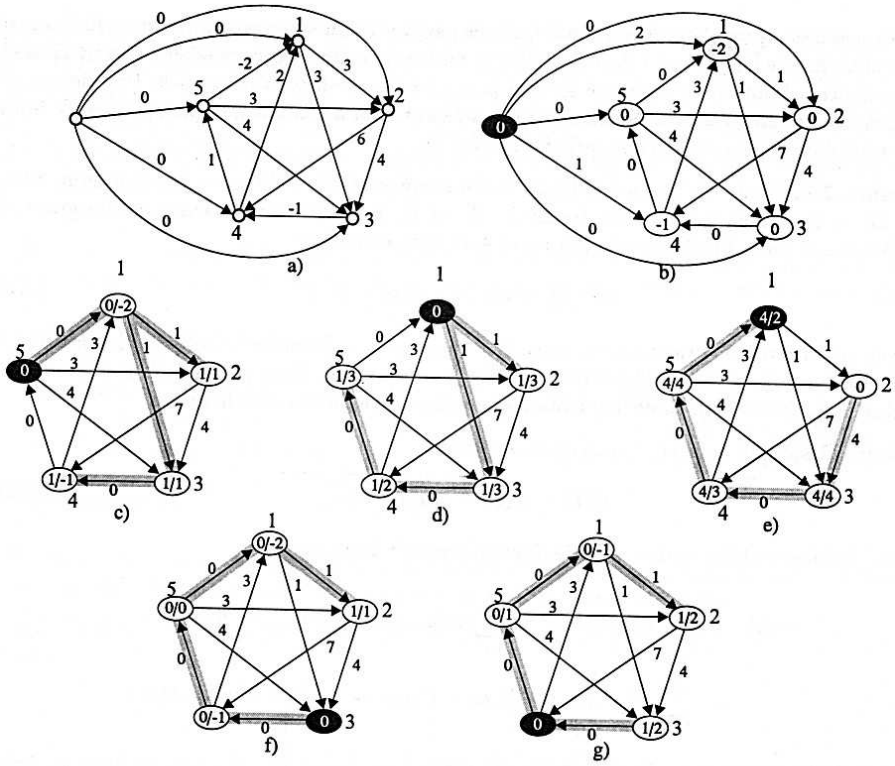
Algoritmus 6.13 Nejkratší cesty pro acyklický graf

```
DAG-PATHS(G, s, w)
1  Topologické uspořádání uzlů grafu G
2  INIT-PATHS(G, s)
3  for každý uzel u v pořadí jeho topologického uspořádání
4      do for každé v ∈ Adj[u]
5          do RELAX(u, v, w)
```

- pokud existuje orientovaná hrana z uzlu *u* do *v*, pak se uzel *u* nachází před uzlem *v* v topologickém uspořádání
- Topologické uspořádání na řádce 1 trvá $|U| + |H|$. Řádky 3-5 se provádějí pro každý uzel právě jednou a výběr uzlu u trvá $O(1)$. Ve vnitřním cyklu relaxuje každá hrana taktéž jednou.
- Složitost O(**|U| + |H|**)

Nejkratší cesty mezi všemi

Johnsonův (O($n^2 \log(n) + n|H|$))



Obrázek 7.6: Provádění Johnsonova algoritmu

Algoritmus 7.7 **Nejkratší cesty v řídkém grafu**

JOHNSON(G)

1 Rozšíření grafu G na graf G' — Přidání uzlu s a hran (s, u).
2 **if not** BELLMAN-FORD(G', w, s) — Našly se v G' cykly
3 **then** write "graf obsahuje cykl záporné délky" — záporné w-délky?
4 **else** **for** každý uzel $u \in U'$
5 **do** $h(u) := d(s, u)$ — S výsledky B-F algoritmu
6 **for** každou hranu $(u, v) \in H'$ — proveď přehodnocení hran.
7 **do** $\widehat{w}(u, v) := w(u, v) + h(u) - h(v)$
8 **for** každý uzel $u \in U$ — Pro každý uzel původního grafu
9 **do** DIJKSTRA($G, \widehat{w}, u$) — spočti $\widehat{w}$-vzdálenosti $\widehat{d}(u, v)$
10 **for** každý uzel $v \in U$ — a uprav je na w-vzdálenosti.
11 **do** $d_{u,v} := \widehat{d}(u, v) + h(v) - h(u)$
12 **return D**

Floyd-Warshall (O(n^3))

$D^{(0)}$

0	3	6	œ	œ
œ	0	2	6	œ
œ	œ	0	-3	œ
1	œ	œ	0	1
-2	3	4	œ	0

$D^{(1)}$

0	3	6	œ	œ
œ	0	2	6	œ
œ	œ	0	-3	œ
1	4	7	0	1
-2	1	4	œ	0

$D^{(2)}$

0	3	5	9	œ
œ	0	2	6	œ
œ	œ	0	-3	œ
1	4	6	0	1
-2	1	3	7	0

$D^{(3)}$

0	3	5	2	œ
œ	0	2	-1	œ
œ	œ	0	-3	œ
1	4	6	0	1
-2	1	3	0	0

$D^{(4)}$

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

$P^{(0)}$

/	1	1	/	/
/	/	2	2	/
/	/	/	3	/
4	/	/	/	4
5	5	5	/	/

$P^{(1)}$

/	1	1	/	/
/	/	2	2	/
/	/	/	3	/
4	1	1	/	4
5	1	5	/	/

$P^{(2)}$

/	1	2	2	/
/	/	2	2	/
/	/	/	3	/
4	1	2	/	4
5	1	2	2	/

$P^{(3)}$

/	1	2	3	/
/	/	2	3	/
/	/	/	3	/
4	1	2	/	4
5	1	2	3	/

$P^{(4)}$

/	1	2	3	4
4	/	2	3	4
4	1	/	3	4
4	1	2	/	4
5	1	2	3	/

Obrázek 7.4: Matice vzdáleností a přechůdců na nejkratších cestách

FLOYD-WARSHALL(W)

1 $\mathbf{D}^{(0)} := \mathbf{W}$ — Počáteční nastavení hodnotami w_{ij}
2 **for** $k := 1$ **to** n **do** — Pro narůstající množinu
3 **for** $i := 1$ **to** n **do** — vnitřních uzlů nejkratších cest
4 **for** $j := 1$ **to** n **do** — spočti další matici $\mathbf{D}^{(k)}$.
5 $d_{ij}^{(k)} := \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
6 **return** $\mathbf{D}^{(n)}$