Cvičení týden 10

Nejkratší cesty

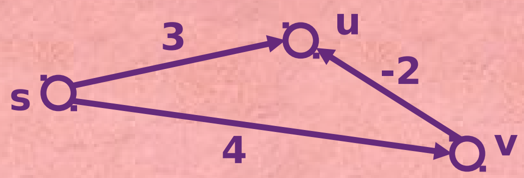
Používané pojmy:

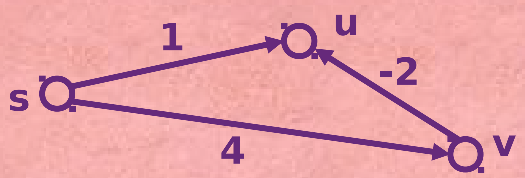
- nejkratší cesty 1 $\rightarrow$ n, relaxace :
  - Dijkstrův algoritmus, Bellmanův-Fordův algoritmus
- nejkratší cesty n $\rightarrow$ n :
  - Floydův-Warshallův algoritmus, Johnsonův algoritmus, zobecněný F-W algoritmus

Kontrolní otázky

7.1 Která část Dijkstrova algoritmu je podstatně závislá na předpokladu nezáporného ohodnocení hran? Ukažte na jednom příkladu, že pro záporně ohodnocené hrany může Dijkstrův algoritmus dát špatný výsledek, a na jiném příkladu, že může dát správný výsledek.

Uzavření uzlu poté, co byl jako uzel s minimálním d[u] vybrán z prioritní fronty.





w(s,u)=3, w(s,v)=4, w(v,u)=-2 : Dijkstra dá d(s,u)=3, má být 2

w(s,u)=1, w(s,v)=4, w(v,u)=-2 : Dijkstra dá správně d(s,u)=1

7.2 Je možné prohlásit, že Dijkstrův algoritmus bude fungovat správně i při záporném ohodnocení hran, pokud bude zadaný graf acyklický?

Není to pravda – viz předchozí příklad.

7.3 Je možné prohlásit, že Dijkstrův algoritmus bude fungovat správně i při záporném ohodnocení hran, pokud bude použit k určení vzdáleností z kořene do ostatních uzlů kořenového stromu?

V kořenovém stromu existuje pro každý uzel u právě jedna cesta z kořene do u, jejíž w-délku algoritmus správně určí.

Upravíme Dijkstrův algoritmus pro kořenové stromy

```
void InitPaths(Graph G, Node s) {  // inicializace
  for (Node u in U(G))
     { d[u] = +∞ ; p[u] = null; }
  d[s] = 0; }
```

```
void Relax (Node u, Node v, Weights w) {
  if (d[v] > d[u]+w(u,v)) // příp.úprava délky cesty
     { d[v] = d[u]+w(u,v);  p[v] = u; } }
```

```
void Dijkstra(Graph G, Node s, Weights w) {
  InitPaths(G,s);
  S = ⊘; Queue.Init();
  for (Node u in U(G)) Queue.Push(u);
  while (!Queue.Empty()) {
 u = Queu            S∪{u};
         Queue.Pop();
 for (Node
           Relax(u,v,w);
} } }
```

Kontrolní otázky

7.4 Navrhněte časově efektivní algoritmus pro určení celkového počtu různých orientovaných cest v acyklickém grafu.

(Návod: Inspirujte se algoritmem DAG-Paths a za hodnotu d[u] berte počet cest končících v uzlu u.)

Změna INIT-PATHS : `for (Node u in U(G)) d[u]=0;`

Změna RELAX : `d[v] += d[u] + 1;`

```
   void InitPaths(Graph G, Node s) {  // inicializace
     for (Node u in U(G))
          { d[u] = +∞ ; p[u] = null; }
     d[s] = 0;
   }
```

```
   void Relax (Node u, Node v, Weights w) {
     if (d[v] > d[u]+ w(u,v)) // příp.úprava délky cesty
       { d[v] = d[u]+ w(u,v);  p[v] = u; }
   }
```

```
   DAG-Paths - nejkratší cesty pro acyklické grafy
     "Topologicky uspořádáme uzly grafu G"
     InitPaths(G,s);
     for (Node u in U(G) v pořadí top. uspořádání)
       for (Node v in Adj[u]) Relax(u,v,w);
```

Kontrolní otázky

7.4 **Navrhněte časově efektivní algoritmus pro určení celkového počtu různých orientovaných cest v acyklickém grafu.**
(Návod: Inspirujte se algoritmem DAG-Paths a za hodnotu d[u] berte počet cest končících v uzlu u.)

Změna INIT-PATHS : `for (Node u in U(G)) d[u]=0;`

Změna RELAX : `d[v] += d[u] + 1;`

7.5 **Navrhněte algoritmus, který určí vzdálenost ze všech uzlů do uzlu s v acyklickém orientovaném grafu G. Určete potřebné datové struktury a časovou složitost navrženého algoritmu.**

1. **Převod G na opačně orientovaný graf G' se "stejným" ohodnocením w.**
2. **DAG-PATHS pro G' (tj. jediný průchod všemi uzly v topologickém pořadí s relaxací vystupujících hran)**

Složitost je lineární O(|U|+|H|).

DAG-Paths - nejkratší cesty pro acyklické grafy

```
"Topologicky uspořádáme uzly grafu G"
InitPaths(G,s);
for (Node u in U(G) v pořadí top. uspořádání) {
  for (Node v in Adj[u]) Relax(u,v,w);
}
```

Kontrolní otázky

7.4 **Navrhněte časově efektivní algoritmus pro určení celkového počtu různých orientovaných cest v acyklickém grafu.**

(Návod: Inspirujte se algoritmem DAG-Paths a za hodnotu d[u] berte počet cest končících v uzlu u.)

Změna INIT-PATHS : `for (Node u in U(G)) d[u]=0;`

Změna RELAX : `d[v] += d[u] + 1;`

7.5 **Navrhněte algoritmus, který určí vzdálenost ze všech uzlů do uzlu s v acyklickém orientovaném grafu G. Určete potřebné datové struktury a časovou složitost navrženého algoritmu.**

1. Převod G na opačně orientovaný graf G' se "stejným" ohodnocením w.

2. DAG-PATHS pro G' (tj. jediný průchod všemi uzly v topologickém pořadí s relaxací vystupujících hran)

Složitost je lineární O(|U|+|H|).

DAG-Paths - nejkratší cesty pro acyklické grafy

```
"Topologicky uspořádáme uzly grafu G"
InitPaths(G,s);
for (Node u in U(G) v pořadí top. uspořádání) {
  for (Node v in Adj[u]) Relax(u,v,w);
}
```

Kontrolní otázky

7.4 **Navrhněte časově efektivní algoritmus pro určení celkového počtu různých orientovaných cest v acyklickém grafu.**
(Návod: Inspirujte se algoritmem DAG-Paths a za hodnotu d[u] berte počet cest končících v uzlu u.)

Změna INIT-PATHS : `for (Node u in U(G)) d[u]=0;`

Změna RELAX : `d[v] += d[u] + 1;`

7.5 **Navrhněte algoritmus, který určí vzdálenost ze všech uzlů do uzlu s v acyklickém orientovaném grafu G. Určete potřebné datové struktury a časovou složitost navrženého algoritmu.**

1. Převod G na opačně orientovaný graf G' se "stejným" ohodnocením w.

2. DAG-PATHS pro G' (tj. jediný průchod všemi uzly v topologickém pořadí s relaxací vystupujících hran)

Složitost je lineární O(|U|+|H|).

7.6 **Navrhněte alg**
daného uzlu d

DAG-PATHS s u

```
if ( (d[u] !=
  d[v] = d[u]+
  p[v] = u;
}
```

DAG-Paths - nejkratší cesty pro acyklické grafy

```
"Topologicky uspořádáme uzly grafu G"
InitPaths(G,s);
for (Node u in U(G) v pořadí top. uspořádání) {
  for (Node v in Adj[u]) Relax(u,v,w);
}
```

Kontrolní otázky

7.4 **Navrhněte časově efektivní algoritmus pro určení celkového počtu různých orientovaných cest v acyklickém grafu.**

(Návod: Inspirujte se algoritmem DAG-Paths a za hodnotu d[u] berte počet cest končících v uzlu u.)

Změna INIT-PATHS : `for (Node u in U(G)) d[u]=0;`

Změna RELAX : `d[v] += d[u] + 1;`

7.5 **Navrhněte algoritmus, který určí vzdálenost ze všech uzlů do uzlu s v acyklickém orientovaném grafu G. Určete potřebné datové struktury a časovou složitost navrženého algoritmu.**

1. Převod G na opačně orientovaný graf G' se "stejným" ohodnocením w.
2. DAG-PATHS pro G' (tj. jediný průchod všemi uzly v topologickém pořadí s relaxací vystupujících hran)

Složitost je lineární O(|U|+|H|).

7.6 **Navrhněte algoritmus lineární složitosti pro hledání nejdelších cest z daného uzlu do všech ostatních uzlů v acyklickém grafu.**

DAG-PATHS s úpravou RELAX:

```
if ( (d[u] != ∞) && ((d[v] == ∞) || (d[v] < d[u]+w(u,v))) ) {
  d[v] = d[u]+w(u,v);
  p[v] = u;
}
```

Kontrolní otázky

7.7 Doplňte Dijkstrův a Bellman-Fordův algoritmus o výpočet hodnoty r[u], která představuje počet hran nejkratší cesty z uzlu s do uzlu u.

(Návod: Stačí vhodně upravit operace InitPaths a Relax.)

Změna INIT-PATHS : `for (Node u in U(G)) { d[u]=+∞; r[u]=0; p[u] … }`

Změna RELAX : `{ d[v]=d[u]+w(u,v); r[v] = r[u]+1; p[v] … }`

Společné operace pro základní varianty algoritmů:

```
   void InitPaths(Graph G, Node s) {  // inicializace
     for (Node u in U(G))
         { d[u] = +∞ ; p[u] = null; }
     d[s] = 0;
   }
```

```
   void Relax (Node u, Node v, Weights w) {
     if (d[v] > d[u]+w(u,v)) // příp.úprava délky cesty
       { d[v] = d[u]+w(u,v);  p[v] = u; }
   }
```

Kontrolní otázky

8.1 Podobně jako je při provádění Floyd-Warshallova algoritmu možné počítat matici P předchůdců uzlů na nejkratších cestách, je možné také počítat matici Q následníků uzlů na nejkratších cestách. Určete pravidlo, podle něhož se nastaví počáteční hodnoty prvků $q_{ij}^{(0)}$ této matice, a pravidlo pro přechod od (k-1)-ní ke k-té iteraci hodnot q_{ij}.

$q_{ij}^{(0)}$ = 0/null pokud i=j nebo w(i,j)=+∞, $q_{ij}^{(0)}$ = j v ostatních případech

$q_{ij}^{(k)} = q_{ij}^{(k-1)}$ pokud $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

$q_{ij}^{(k)} = q_{ik}^{(k-1)}$ pokud $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Algoritmus Floyda-Warshalla

```
D(0) = W;
for (k=1; k<=n; k++) {
  for (i=1; i<=n; i++)   {
     for (j=1; j<=n; j++) {
          dij(k) = min (dij(k-1) , dik(k-1) + dkj(k-1));
} } }
return D(n);
```

Výpočet matice předchůdců:

$p_{ij}^{(0)}$ = 0/null pro i=j nebo $w_{ij} = \infty$

= i jinak (pro $(u_i,u_j) \in H$)

$p_{ij}^{(k)} = p_{ij}^{(k-1)}$ pro $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

$= p_{kj}^{(k-1)}$ pro $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Kontrolní otázky

8.2 Dalším rozšířením Floyd-Warshallova algoritmu zajistěte, aby po ukončení výpočtu byl znám počet hran na nejkratších cestách mezi všemi uzly (opět ve formě matice označené např. R). (Návod: Inspirujte se řešením obdobného problému pro algoritmus Dikstrův a Bellman-Fordův.)

pom = $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$;

if ($d_{ij}^{(k-1)} >$ pom) {

$d_{ij}^{(k)}$ = pom; $r_{ij}^{(k)} = r_{ik}^{(k-1)} + r_{kj}^{(k-1)}$;

}

Nastavení počátečních hodnot matice R:

$r_{ij}^{(0)}$ = 0/null pro i=j nebo $w_{ij} = +\infty$

1 jinak (pro $(u_i,u_j) \in H$)

5 **$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$;**

Výpočet matice předchůdců:

$p_{ij}^{(0)}$ = **0/null** pro i=j nebo $w_{ij} = +\infty$

i jinak (pro $(u_i,u_j) \in H$)

$p_{ij}^{(k)}$ = $p_{ij}^{(k-1)}$ pro $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

$p_{kj}^{(k-1)}$ pro $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Kontrolní otázky

8.3 **Zdůvodněte, proč je provádění Floyd-Warshallova algoritmu možné všechny iterace matice $D^{(k)}$ uchovávat v jediném poli.**

(Návod: Ověřte, že vnitřní dva cykly neméní hodnotu prvků v k-tém řádku a k-tém sloupci, na nichž závisí hodnoty prvků v nové iteraci.)

$$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

Tělo vnitřního cyklu pro k-tý řádek (pro i=k)

$d_{kj}^{(k)} = \min (d_{kj}^{(k-1)}, d_{kk}^{(k-1)} + d_{kj}^{(k-1)}) = d_{kj}^{(k-1)}$ **neboť** $d_{kk}^{(k-1)} = 0$

Tělo vnitřního cyklu pro k-tý sloupec (pro j=k)

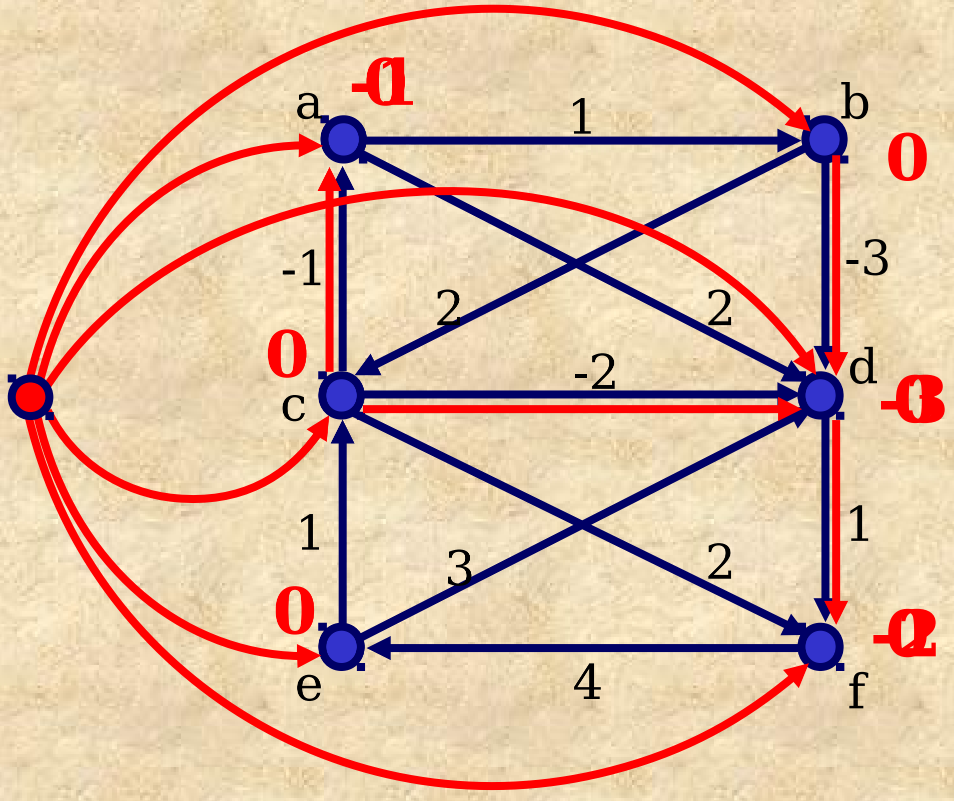
$d_{ik}^{(k)} = \min (d_{ik}^{(k-1)}, d_{ik}^{(k-1)} + d_{kk}^{(k-1)}) = d_{ik}^{(k-1)}$ **neboť** $d_{kk}^{(k-1)} = 0$

8.4 **Jak se při použití Floyd-Warshallova algoritmu zjistí případná existence záporných cyklů v grafu?**

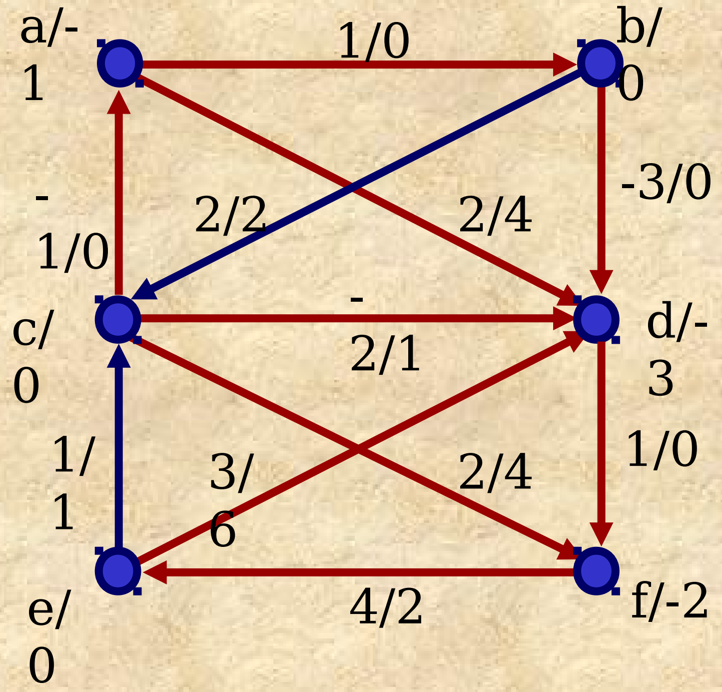
Některé diagonální prvky matice D obsahují záporná čísla.

Kontrolní otázky

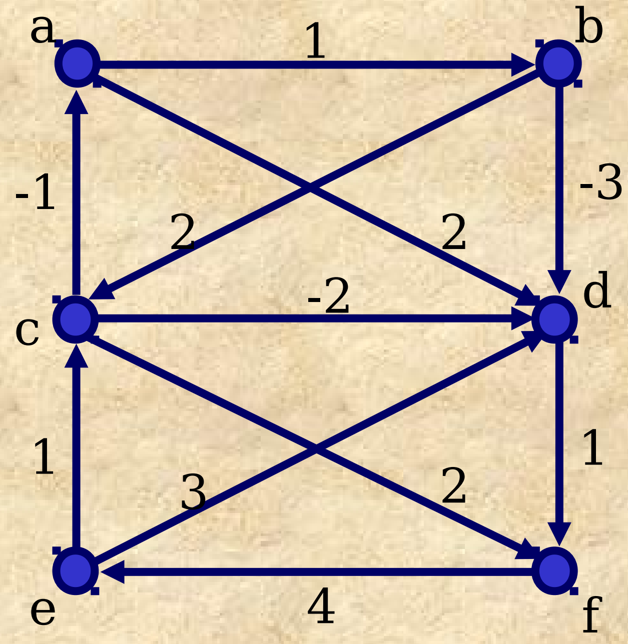
8.5 **Pomocí Johnsonova algoritmu určete matici vzdáleností pro následující orientovaný graf :**



Řešení 8.6

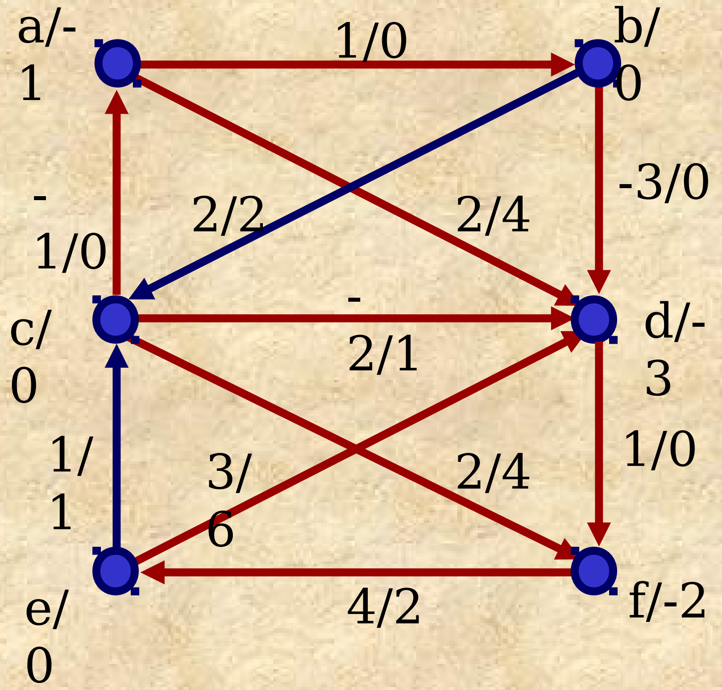


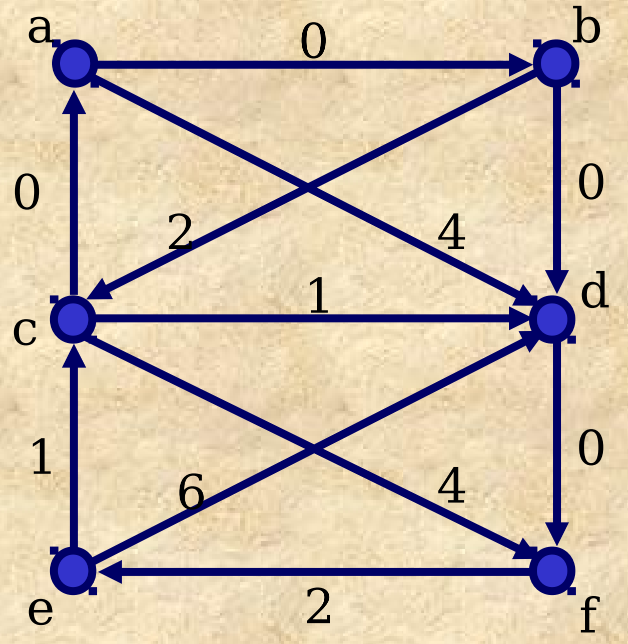
a/-1
1/0
b/0
-1/0
2/2
2/4
-3/0
c/0
-2/1
d/-3
1/1
3/6
2/4
1/0
e/0
4/2
f/-2



a
1
b
-1
2
2
-3
c
-2
d
1
3
2
1
e
4
f

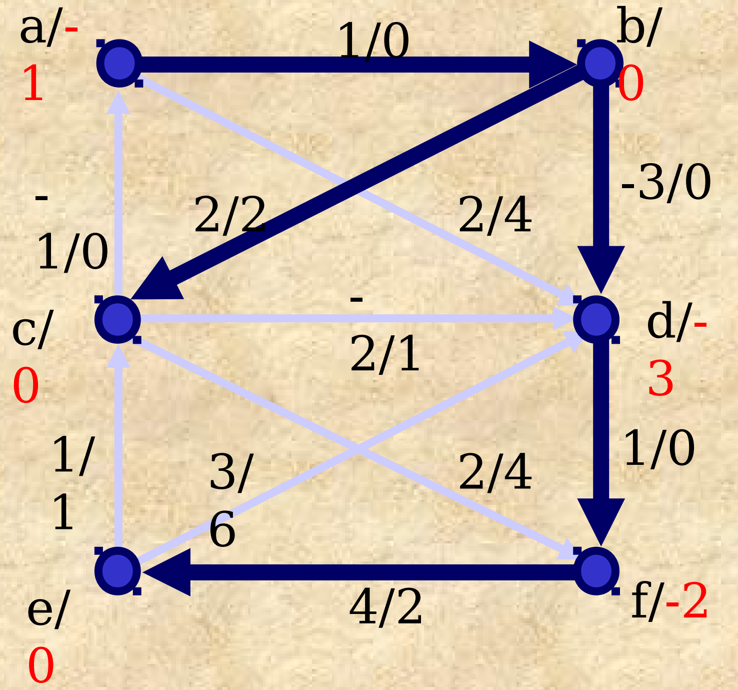
Řešení 8.6





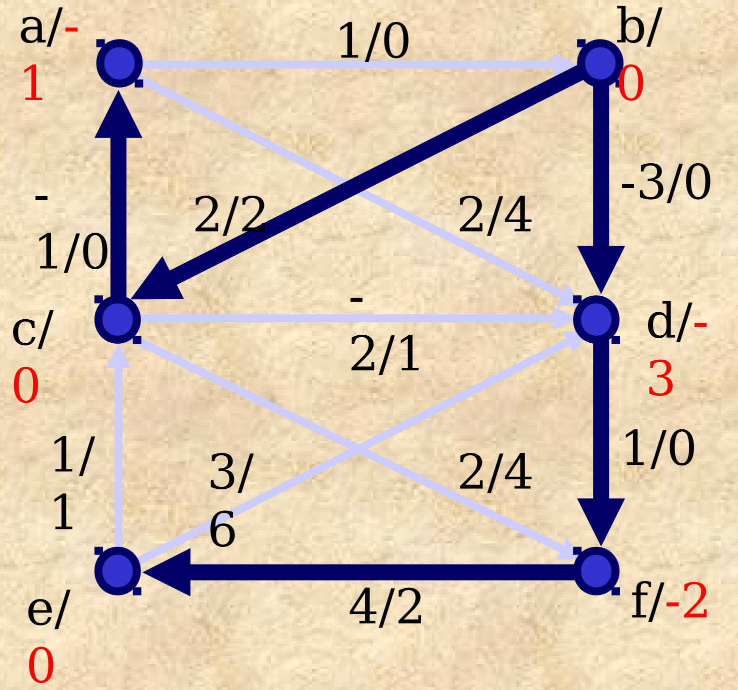
A teď opakovaně provádíme Dijkstrův algoritmus …

Řešení 8.6



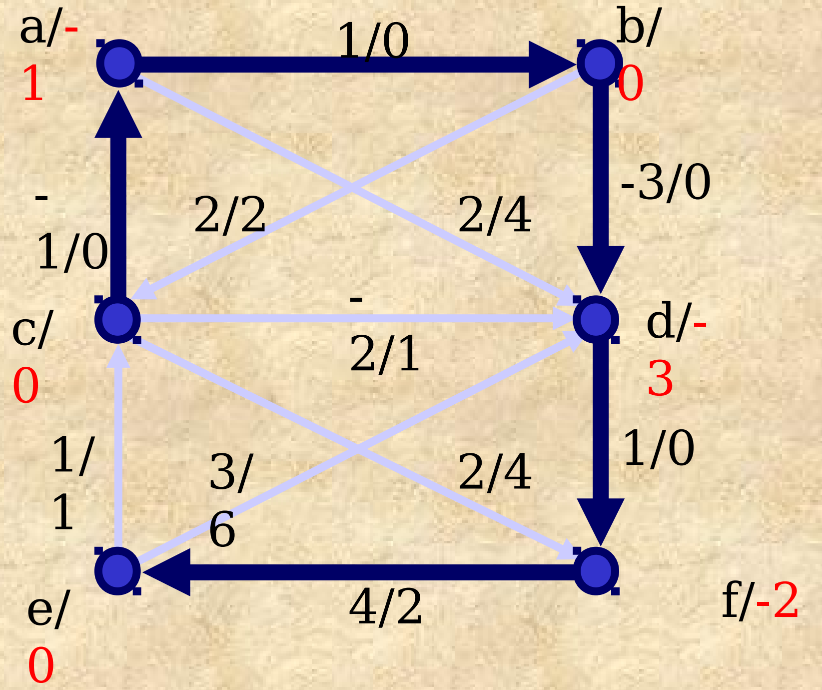
	a	b	c	d	e	f
a	0	1	3	-2	3	-1
b	1	0	2	-3	2	-2
c	-1	0	0	-3	2	-2
d	5	6	6	0	5	1
e	0	1	1	-2	0	-1
f	4	5	5	2	4	0

Řešení 8.6



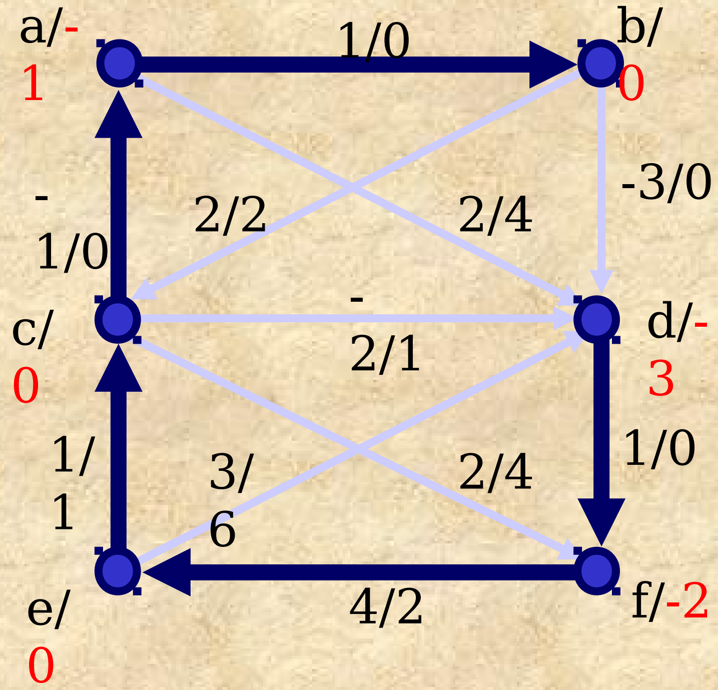
	a	b	c	d	e	f
a	0	1	3	-2	3	-1
b	1	0	2	-3	2	-2
c	-1	0	0	-3	2	-2
d	5	6	6	0	5	1
e	0	1	1	-2	0	-1
f	4	5	5	2	4	0

Řešení 8.6



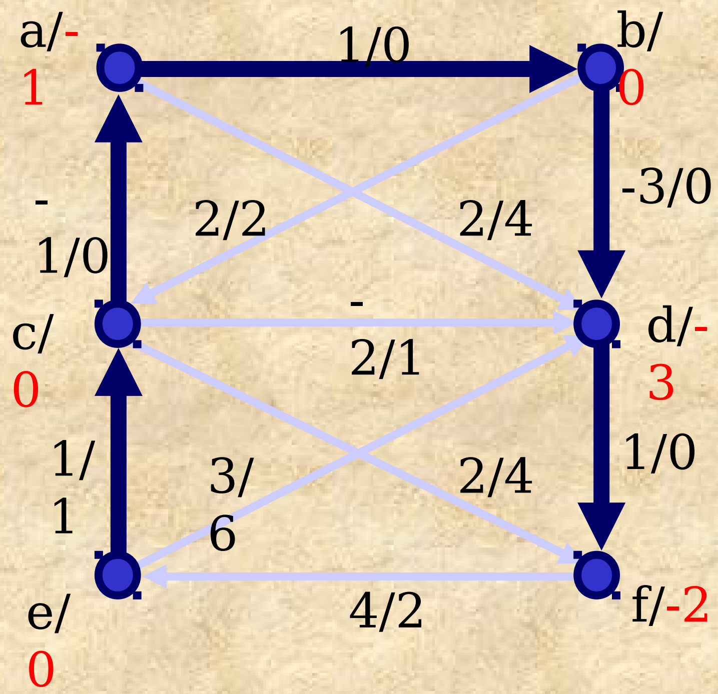
	a	b	c	d	e	f
a	0	1	3	-2	3	-1
b	1	0	2	-3	2	-2
c	-1	0	0	-3	2	-2
d	5	6	6	0	5	1
e	0	1	1	-2	0	-1
f	4	5	5	2	4	0

Řešení 8.6



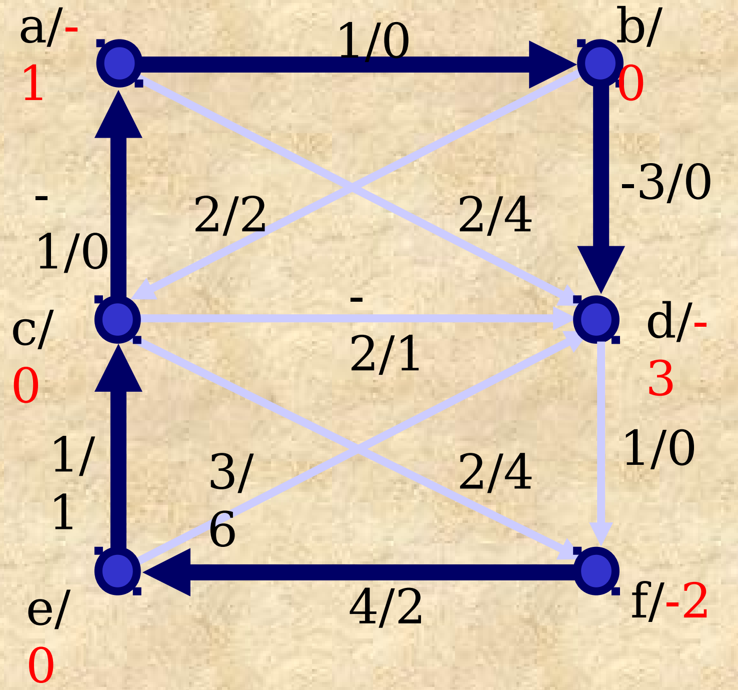
	a	b	c	d	e	f
a	0	1	3	-2	3	-1
b	1	0	2	-3	2	-2
c	-1	0	0	-3	2	-2
d	5	6	6	0	5	1
e	0	1	1	-2	0	-1
f	4	5	5	2	4	0

Řešení 8.6



	a	b	c	d	e	f
a	0	1	3	-2	3	-1
b	1	0	2	-3	2	-2
c	-1	0	0	-3	2	-2
d	5	6	6	0	5	1
e	0	1	1	-2	0	-1
f	4	5	5	2	4	0

Řešení 8.6



	a	b	c	d	e	f
a	0	1	3	-2	3	-1
b	1	0	2	-3	2	-2
c	-1	0	0	-3	2	-2
d	5	6	6	0	5	1
e	0	1	1	-2	0	-1
f	4	5	5	2	4	0

Kontrolní otázky

8.6 Jaký vztah platí mezi ohodnoceními w(u,v) a w'(u,v), pokud jsou hodnoty w(u,v) ≥ 0 pro všechny hrany (u,v) ?

w'(u,v) = w(u,v) pro všechny hrany

8.7 Jak je třeba definovat operace ⊕ a ⊗ a nosič (tj. množinu P) v odpovídajícím polookruhu, aby zobecněný Floyd-Warshallův algoritmus určil počet různých spojení mezi jednotlivými dvojicemi uzlů?

Nosičem bude množina přirozených čísel (včetně nuly) doplněná o nevlastní prvek ∞, operace ⊕ bude skoro obyčejné sčítání celých čísel a operace ⊗ bude skoro obyčejné násobení celých čísel – obě operace musí umět adekvátně přičíst a násobit nekonečno. Inicializace použije matici sousednosti, 0* = 1, a* = ∞ pro a>0.

P = ‹P, ⊕, ⊗, 0, 1› :

a ⊕ (b ⊕ c) = (a ⊕ b) ⊕ c	‹P, ⊕, 0›
a ⊕ 0 = 0 ⊕ a = a	je komutativní
a ⊕ b = b ⊕ a	monoid
a ⊕ a = a	idempotence

a ⊗ (b ⊗ c) = (a ⊗ b) ⊗ c	‹P, ⊗, 1›
a ⊗ 1 = 1 ⊗ a = a	je monoid
a ⊗ 0 = 0 ⊗ a = 0	s nulou

a ⊗ (b ⊕ c) = (a ⊗ b) ⊕ (a ⊗ c)	distributivnost	uzávěr
(b ⊕ c) ⊗ a = (b ⊗ a) ⊕ (c ⊗ a)	zleva a zprava	a*=1 ⊕ a ⊕ a⊗a ⊕ a⊗a⊗a ⊕ …