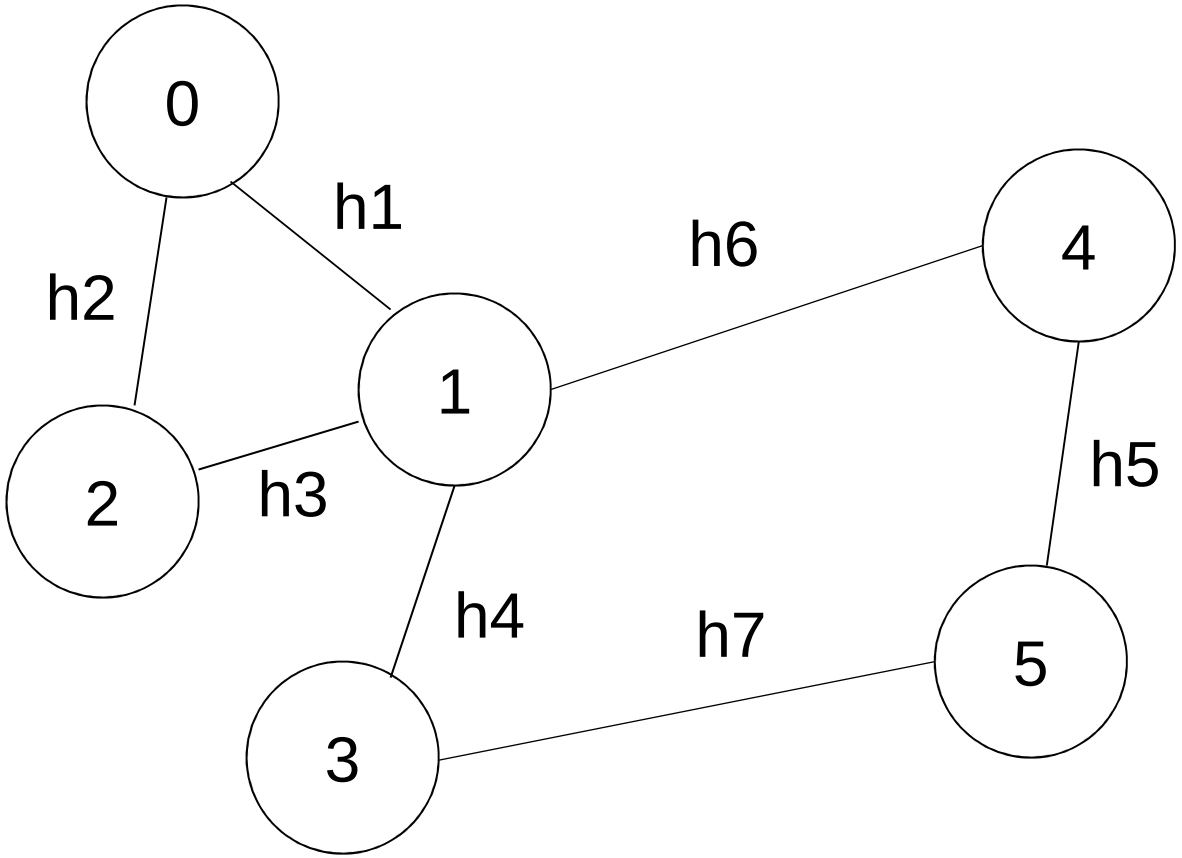
Vzdálenosti a grafy

Vzdálenost uzlů v neorientovaném grafu

Je dán neorientovaný neohodnocený graf G = (V,E,I)

- vzdálenost uzlů u a v v neorientovaném souvislém grafu G je délka nejkratší cesty spojující tyto dva uzly
  - značíme ji $d_G(u, v)$
  - délka cesty = počet hran v cestě

Graf



$d_G(0, 5) = 3$

$d_G(0, 3) = 2, d_G(3, 0) = 2$

$d_G(0, 2) = 1$

Vlastnosti vzdálenosti

- vzdálenost $d(u, v)$ je celé nezáporné číslo
- $d(u, v) \geq 0$; přitom $d(u, v) = 0$, právě když $u = v$
- symetrie: $d(u, v) = d(v, u)$
- trojúhelníková nerovnost:

$$d(u, v) \leq d(u, z) + d(z, v)$$

- je-li $d(u, v) > 1$, pak platí:

$$\exists z \in V, (u \neq z \neq v \wedge d(u, v) = d(u, z) + d(z, v))$$

Vzdálenost uzlů v orientovaném grafu

Je dán orientovaný neohodnocený graf

G = (V,E,I)

- vzdálenost $d(u, v)$ uzlů u a v v orientovaném souvislém grafu G je délka nejkratší orientované cesty (= počet hran) spojující tyto dva uzly
  - není symetrická
  - $d(u, v)$ je definována pokud existuje mezi u a v orientovaná cesta, jinak je $d(u, v) = \infty$
  - $d(u, v)$ je definována pro všechny dvojice uzlů pouze v případě silně souvislého grafu

Vzdálenost uzlů v orientovaném a ohodnoceném grafu

- S = ($h_1, h_2, \ldots, h_k$) je spojení orientovaneho grafu G = (V,E,I) s reálným ohodnocením hran $w : E \rightarrow R$
- w-délkou spojení S nazýváme reálné číslo $d_w(S)$ definované

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

Vzdálenost uzlů v orientovaném a ohodnoceném grafu

- w-vzdálenost $d_w(u, v)$ z uzlu *u* do uzlu *v* je nejmenší w-délka spojení z *u* do *v*, pokud takové spojení v grafu G existuje
- je-li uzel *v* z uzlu *u* nedostupný, pokládáme

$$d_w(u, v) = \infty$$

Vzdálenost uzlů v orientovaném a ohodnoceném grafu

- problém:
  - pokud je v grafu cyklus se záporným ohodnocením hran, neexistuje minimální spojení
    - opakovaným zařazením cyklu stále snižujeme délku spojení, tedy minimum je - ∞
- jestliže žádná orientovaná cesta z *u* do *v* neprochází cyklem grafu se zápornou nebo nulovou w-délkou, pak bude každé minimální spojení určující w-vzdálenost $d_w(u, v)$ orientovanou cestou

Vlastnosti vzdálenosti

- vzdálenost $d(u, v)$ je celé nezáporné číslo
  - pouze pro nezáporná celočíselná ohodnocení
- $d(u, v) \geq 0$; přitom $d(u, v) = 0$, právě když $u = v$
  - pouze po kladné ohodnocení hran

Vlastnosti vzdálenosti

- symetrie
  - neexistuje u orientovaných grafů
- trojúhelníková nerovnost:

$$d(u, v) \leq d(u, z) + d(z, v)$$

  - platí

Vlastnosti vzdálenosti

- nechť G = (V,E,l) je orientovaný graf s ohodnocením hran w : E → R a buď $L = (u_1, u_2, \ldots, u_k)$ cesta určující vzdálenost $d_w(u_1, u_k)$ z uzlu u_1 do uzlu u_k
- pro libovolná $i, j : 1 \leq i \leq j \leq k$ potom platí

$$d_w(u_1, u_k) = d_w(u_1, u_i) + d_w(u_i, u_j) + d_w(u_j, u_k))$$

Základní úloha

- nalezení nejkratších cest z jednoho zadaného uzlu $s \in V$ do všech ostatních uzlů $u \in V$ grafu G

Varianty základní úlohy

1. nalezení nejkratších cest z každého uzlu $u \in V$ do pevně zadaného cílového uzlu t
   – lze řešit jako základní úlohu v opačně orientovaném grafu G^-

Varianty základní úlohy

2. nalezení jedné nejkratší cesty mezi jedinou zadanou (uspořádanou) dvojicí uzlů $u, v \in V$.
   - řešíme-li základní úlohu s výchozím uzlem u, vyřešíme tím i tuto úlohu
   - není znám žádný algoritmus, který by měl asymptotickou složitost v nejhorším případě lepší než algoritmus pro základní úlohu

Varianty základní úlohy

3. nalezení nejkratší cesty pro všechny (uspořádané) dvojice uzlů $u, v \in V$
   - lze řešit s použitím algoritmu pro základní úlohu - použijeme jej postupně pro každý uzel grafu v roli výchozího uzlu
   - existují i maticové algoritmy

Princip algoritmu pro základní úlohu

- vychází z algoritmu prohledávání grafu do šířky
- u uzlu *s* (výchozí) nastaví vzdálenost na hodnotu 0 a u ostatních na nekonečno
- prochází graf do šířky a přepočítává vzdálenosti na základě tzv. **„relaxace hrany**“

Inicializace

```
INIT-PATHS(G,s,w)
// s je počáteční uzel,  w je ohodnocení hran
{
  for každý uzel u ∈ V
  { // všechny uzly grafu mají od s
    // nekonečnou vzdálenost
    d[u] = ∞;
    // p[u] je přechůdce u na nejkratší cestě
    p[u] = NULL;
  }
  d[s] = 0;
}
```

Relaxace hrany

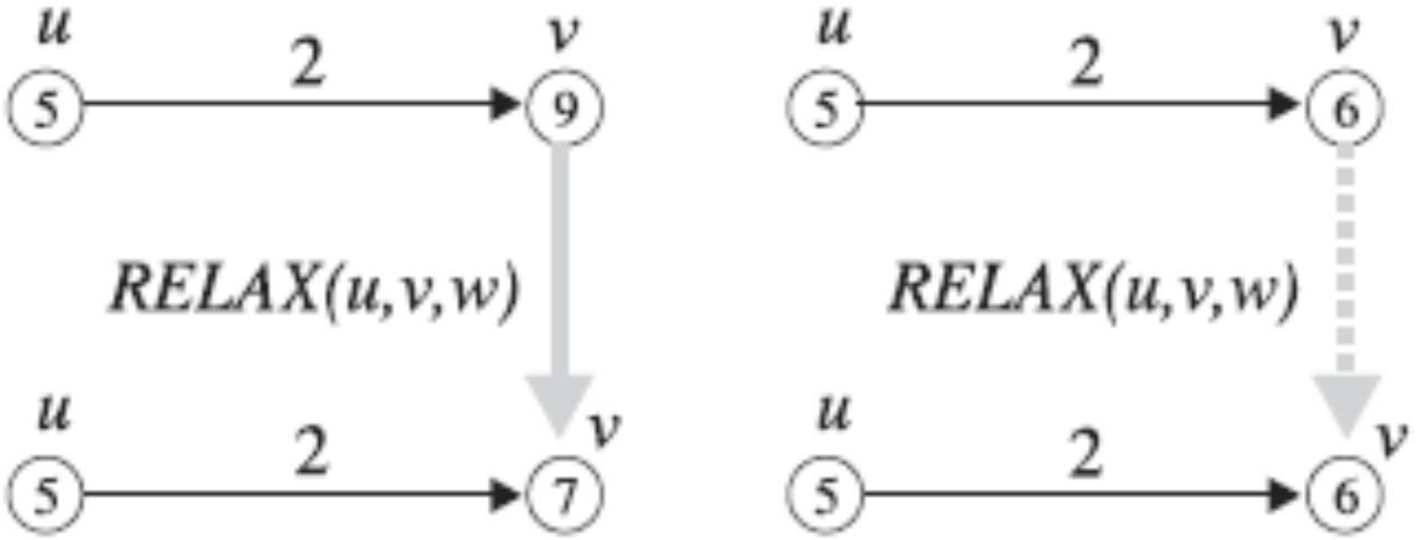
Princip:

- pokud je vzdálenost do uzlu *v* přes uzel *u* kratší než dosavadně spočítaná, přepočtu vzdálenost a opravím předchůdce uzlu *v* na *u*

Relaxace hrany

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

Relaxace hrany



Dijkstrův algoritmus

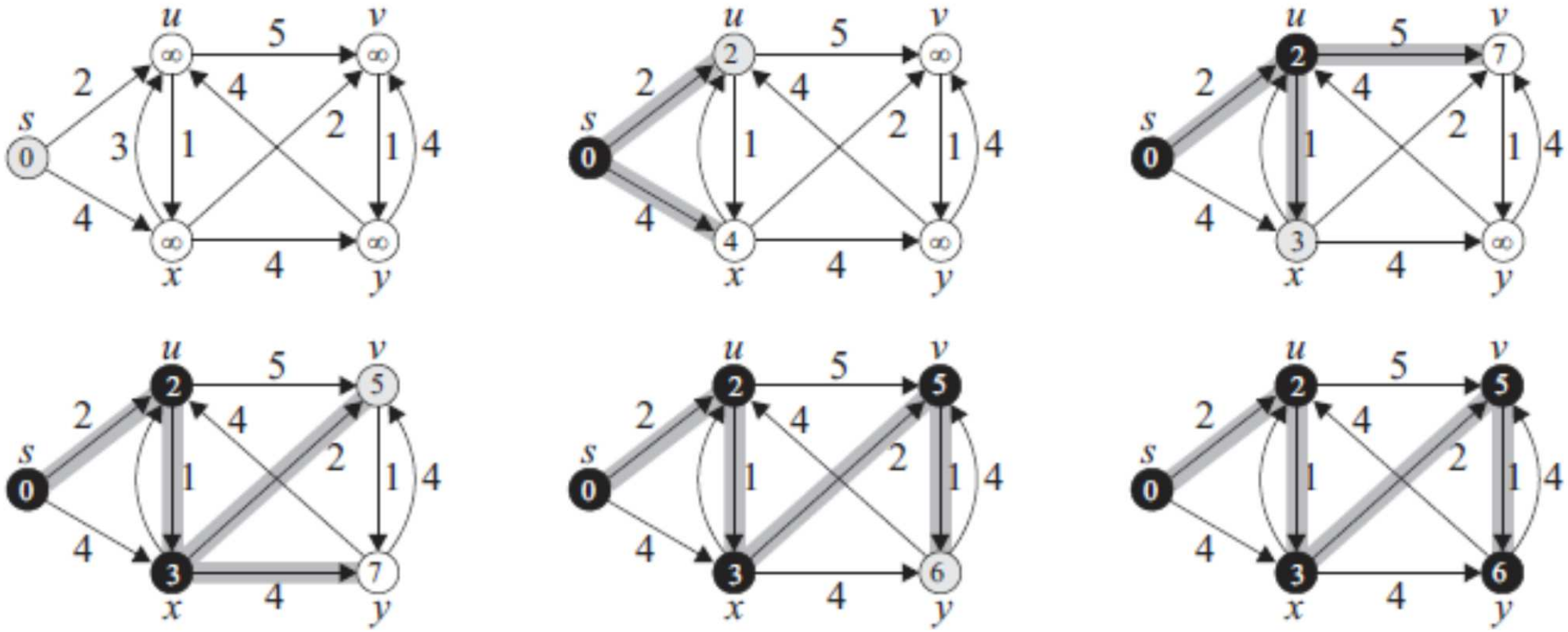
- ohodnocení hran musí být nezáporné
- algoritmus šíří při prohledávání grafu „vlnu“, která pohlcuje uzly, k nímž byla již nalezena nejkratší cesta
- složitost $O(|E| \log_2 |V|)$.

Dijkstrův algoritmus

```
DIJKSTRA(G, s, w)
1   INIT-PATHS(G, s)          Počáteční nastavení d[u], p[u]
2   S := ∅                    a zatím žádný zavřený uzel.
3   INIT-QUEUE(Q)             Všechny uzly grafu
4   for každý uzel u ∈ U      se uloží do fronty
5       do ENQUEUE(Q, u)      seřazené podle d[u]
6   while not EMPTY(Q)        Dokud něco zbývá,
7       do u :=EXTRACT-MIN(Q) vyber uzel nejbližší k s,
8          S := S ∪ {u}       přeřaď jej mezi uzavřené
9          for každý uzel v ∈ Adj[u]  a všem jeho následníkům
10            do RELAX(u,v,w)         uprav vzdálenost od s.
```

zdroj: doc. Kolář: Teoretická informatika, FIT ČVUT

Dijkstrův algoritmus



zdroj: doc. Kolář: Teoretická informatika, FIT ČVUT

Počáteční inicializace

- INIT_PATH
  - d[s]=0, d[u]=∞, d[v]=∞, d[x]=∞, d[y]=∞
    p[s]=NULL, p[u]=NULL, p[v]=NULL,
    p[x]=NULL, p[y]=NULL

- nastavím množinu uzavřených uzlů na prázdnou
  - $S = \emptyset$
- všechny uzly grafu vložím do fronty Q a seřadím dle vzdálenosti *d*
  - Q = { s(0), u(∞), v(∞), x(∞), y(∞) }

- z Q vyberu uzel nejbližší k s a uzavřu jej
  - je to sám uzel s
  - $S = \{s\}$
- pro všechny sousedy uzlu s provedu relaxaci hrany
  - u: $d[u] = 2$, předchůdce $p[u]=s$
  - x: $d[x] = 4$, předchůdce $p[x]=s$
  - $Q = \{u(2), x(4), v(\infty), y(\infty)\}$
- Q není prázdná, z Q vyberu uzel nejbližší k s a uzavřu jej
  - je to uzel u
  - $S = \{s,u\}$

- pro všechny sousedy uzlu u provedu relaxaci hrany
  - v: $d[v] = 5$, předchůdce $p[v]=u$
  - x: $d[x] = 3$, předchůdce $p[x]=u$
    - vzdálenost do x byla přes „za 4“, půjdu-li přes u, je „za 3“
  - $Q = \{v(3), x(5), y(\infty)\}$
- atd.

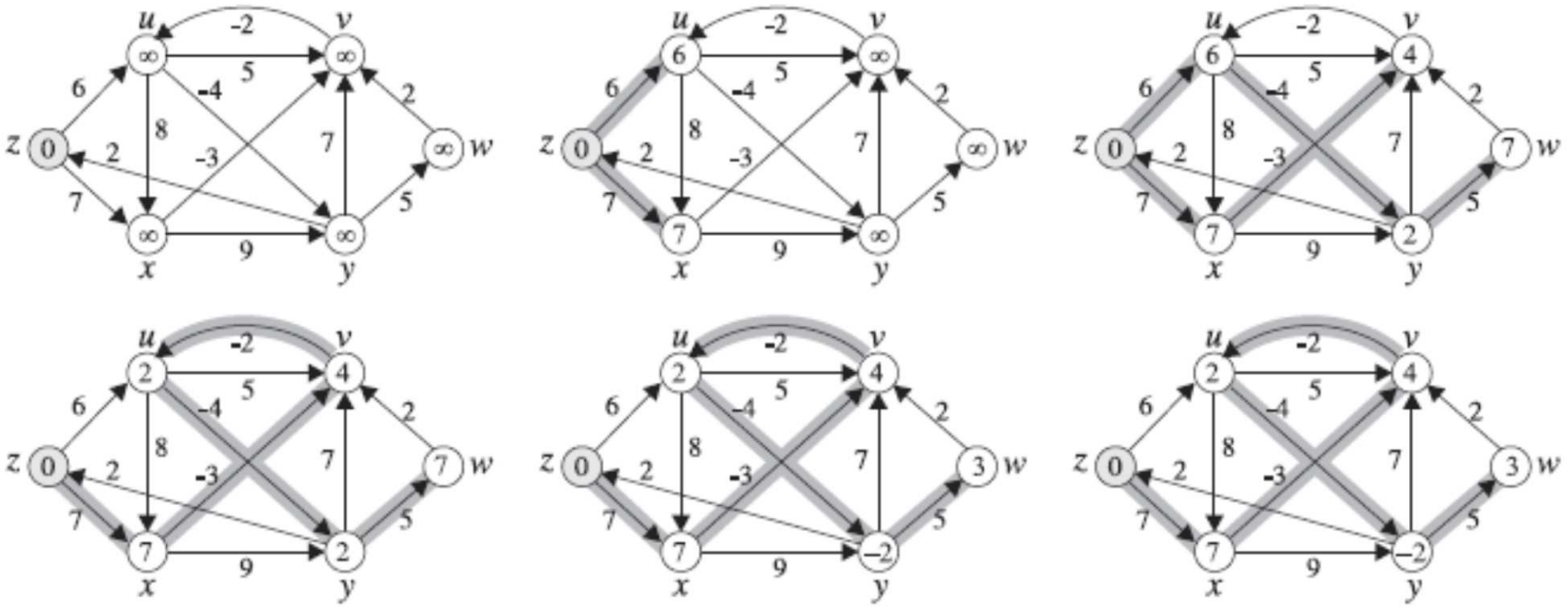
Bellmanův-Fordův algoritmus

- použitelný i pro záporné ohodnocení hran
- Dijkstrův algoritmus relaxoval vždy jen hrany vycházející z vybraného uzlu, B-F algoritmus opakovaně relaxuje systematicky všechny hrany
- algoritmus zároveň detekuje záporné cykly
  - algoritmus vrací true/false, zda detekoval záporný cyklus

Bellmanův-Fordův algoritmus

```
BELLMAN-FORD(G,s,w)
{
  INIT-PATHS(G, s)
  for i = 1 to |U| − 1 {
   for každou hranu (u, v) ∈ H
      RELAX(u, v,w)
   }
  for každou hranu (u, v) ∈ H
    if (d[v] > d[u] + w(u, v) )
      return FALSE;
  return TRUE;
}
```

Bellmanův-Fordův algoritmus



Upravený Bellmanův-Fordův algoritmus

- vybírá pro relaxaci ty hrany, pro jejichž počáteční uzel se hodnota vzdálenosti v předchozí relaxaci snížila

Upravený Bellmanův-Fordův algoritmus

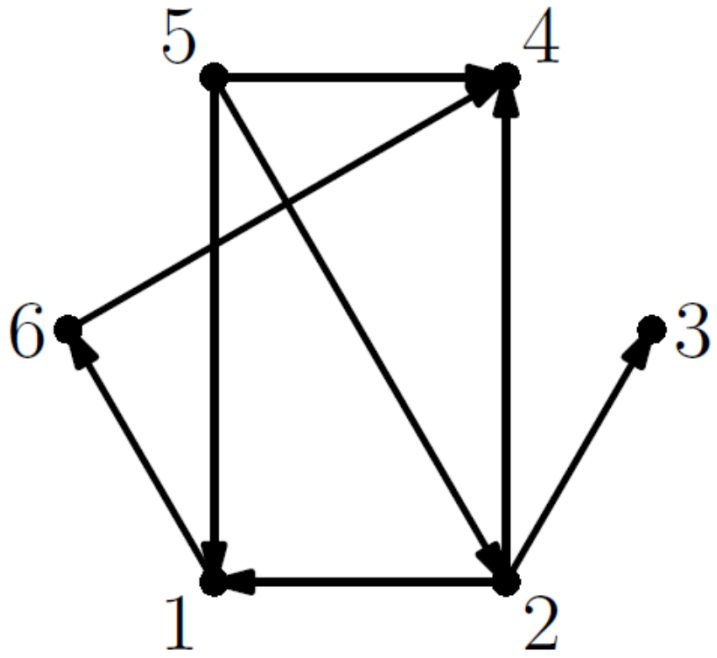
```
BELLMAN-FORD-Q(G, s, w)
1   INIT-PATHS(G, s)                    Počáteční nastavení d[u], p[u]
2   INIT-QUEUE(Q); ENQUEUE(Q, s)        a uložení s do fronty Q.
3   while not EMPTY(Q)                  Dokud je něco ve frontě,
4       do u :=QUEUE-FIRST(Q)           vyber další uzel
5           for každý uzel v ∈ Adj[u]   a všechny vycházející hrany
6               do RELAX-Q(u, v, w)     relaxuj.

RELAX-Q(u, v, w)
1   if d[v] > d[u] + w(u, v)            Lze-li d[v] zmenšit,
2     then d[v] := d[u] + w(u, v)       pak to provedeme,
3          p[v] := u                    upravíme uzlu v předchůdce
4          ENQUEUE(Q, v)                a uložíme jej do fronty.
```

Acyklický graf

- graf, který nemá cykly
- uzly lze topologicky uspořádat
  - je zachováno pořadí předchůdce – následník, tj. uzly lze zapsat do posloupnosti takové, že pro každé dva uzly platí, že „dříve“ zapsaný uzel je v grafu předchůdce „později“ zapsaného
- při hledání nejkratších cest stačí procházet uzly v topologickém pořadí a relaxovat hrany směrem k následníkům

Acyklický graf



- topologické uspořádání uzlů:

  5, 2, 3, 1, 6,4

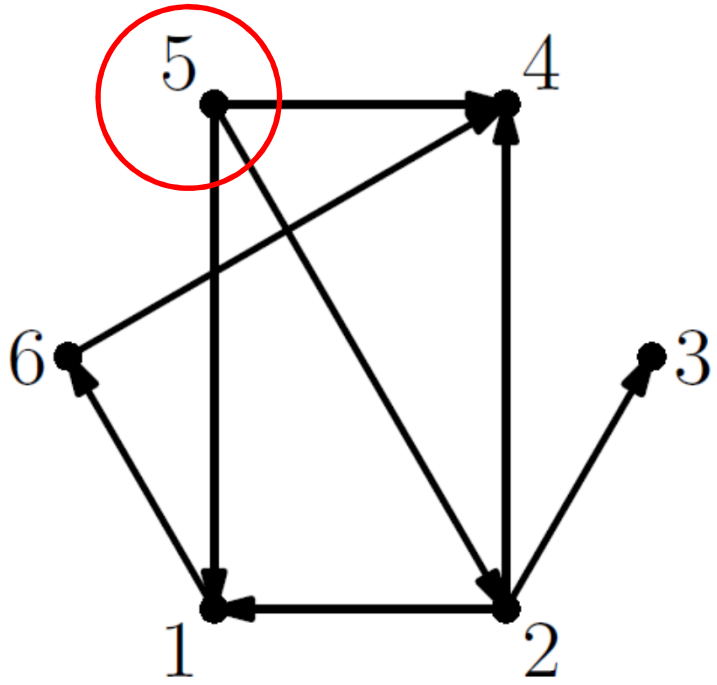
Acyklický graf

- graf je acyklický právě tehdy, když tento graf a každý jeho neprázdný podgraf obsahuje vstupní uzel
  - vstupní uzel
    - uzel, jehož vstupní stupeň je 0, tj. nevchází do něj žádná hrana
  - výstupní uzel
    - uzel, jehož výstupní stupeň je 0, tj. nevychází z něj žádná hrana
- stejná věta platí i pro výstupní uzel

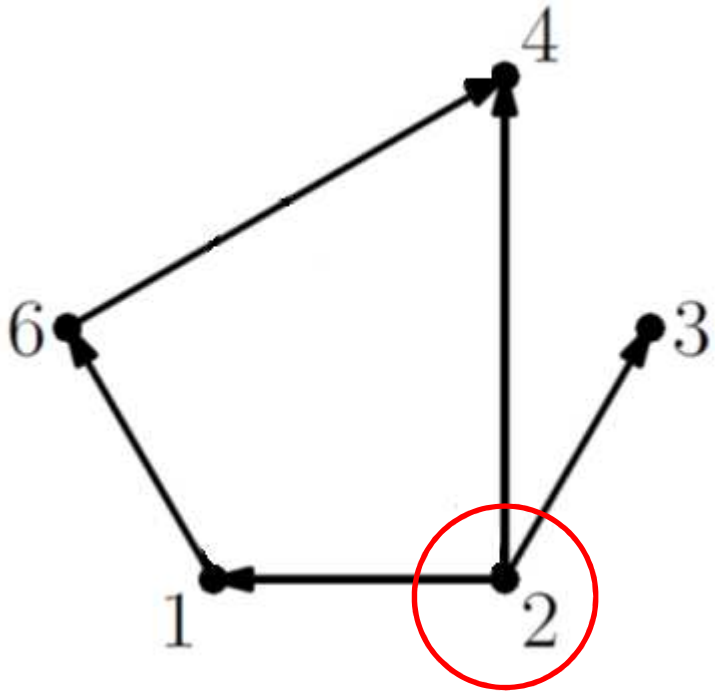
Test acykličnosti grafu

```
int je_acyklicky(G)
{
  while (G je neprázdný)
  {
    if (neexistuje vstupní uzel)
    { graf není acyklický
      return 0;
    }
  u = vstupní uzel
  //odejmi uzel u z grafu a incidující hrany
  G = G – {u}
  přidej uzel u do posloupnosti
  }
  return 1;
```

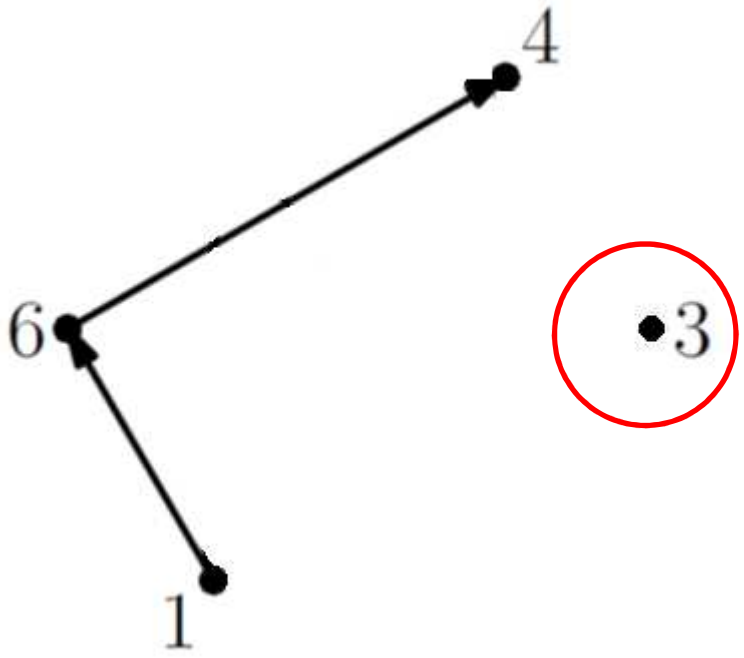
Test acykličnosti grafu



Test acykličnosti grafu

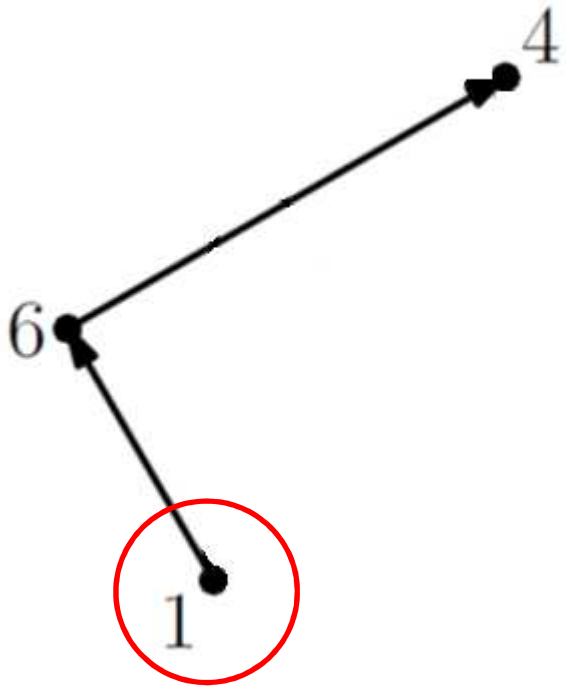


Test acykličnosti grafu



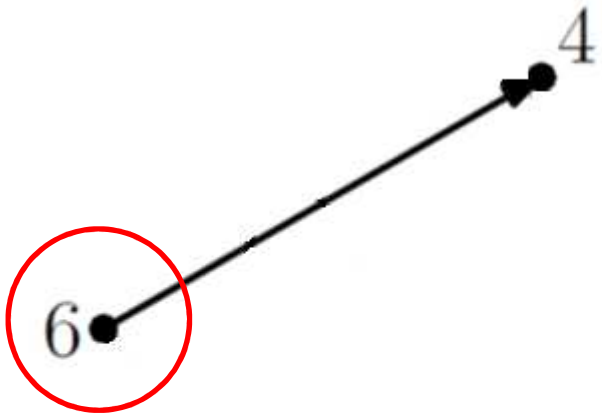
5, 2

Test acykličnosti grafu



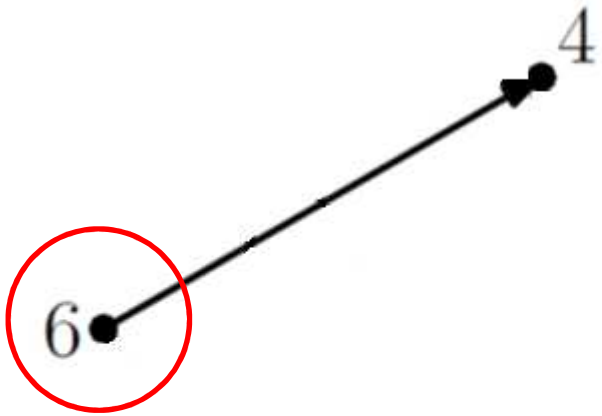
5, 2, 3

Test acykličnosti grafu



5, 2, 3, 1

Test acykličnosti grafu



5, 2, 3, 1

Test acykličnosti grafu

5, 2, 3, 1, 6

Test acykličnosti grafu

5, 2, 3, 1, 6, 4

Hledání vzdáleností v acyklickém grafu

```
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)
```

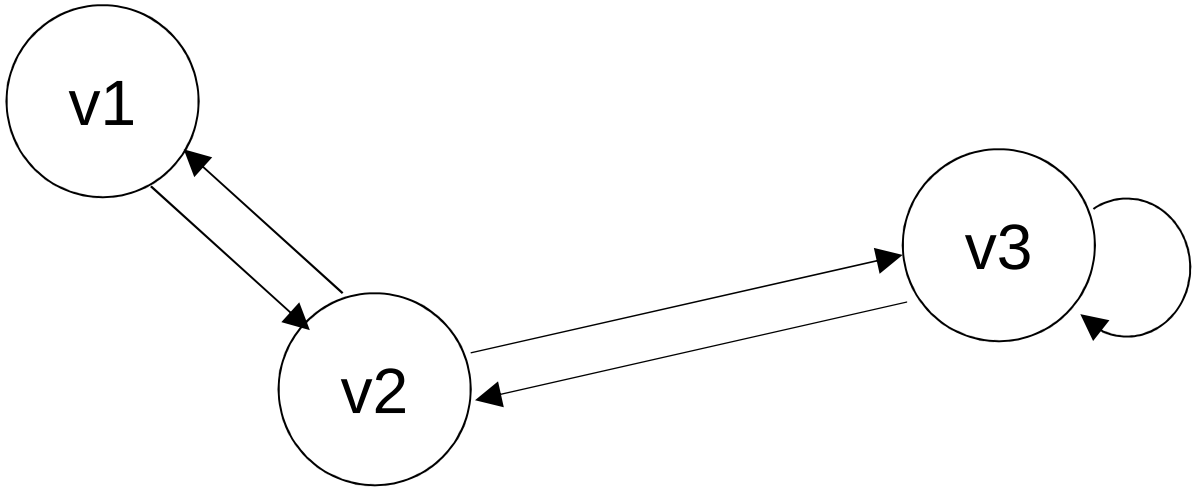
Nejkratší cesty mezi všemi uzly
Maticové operace

Je dán graf G = (V,E,I) a jeho matice sousednosti A

Prvek $a_{i,j}^{(k)}$ k-té mocniny matice sousednosti A^k na pozici *i,j* určuje počet sledů délky právě k

- je-li prvek na diagonále, existuje v grafu uzavřený sled délky k

Příklad:



$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad A^3 = \begin{pmatrix} 0 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{pmatrix}$$

Nejkratší cesty mezi všemi uzly
Maticové operace

- pokud použijeme operace v tělese mod 2, jsou v mocninách matic pouze 1
  - vyjadřují existenci sledů, nikoliv počet
- protože délka nejkratšího sledu je délkou nejkratší cesty, můžeme odvodit:
  - pokud se poprvé objeví nenulový prvek na pozici i,j v k-té mocnině, nejkratší cesta má délku k
  - v grafu o *n* uzlech může existovat nejdelší cesta maximálně délky n-1 – hledáme-li cesty, stačí spočítat max. (n-1) mocninu