Přednáška 3

Orientované grafy a binární relace

V této části se seznámíme s pojmy:

- acyklický graf, testování acykličnosti, topologické uspořádání uzlů/hran orientovaného grafu
- graf binární relace na množině, graf složení relací, složení grafů, tranzitivní uzávěr grafu

Skripta odstavec 2.2, str. 33 - 36

Co víme: silně souvislý graf má každou hranu v nějakém cyklu

Jak vypadá opačný extrém ?

acyklický graf = orientovaný graf bez cyklů

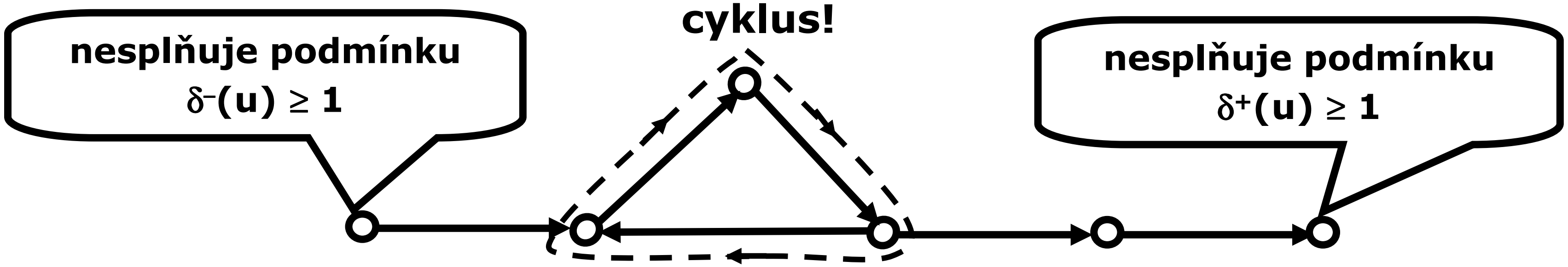
Jak nejlépe testovat, zda je graf acyklický ?

??? Hledáním cyklů ???

<u>Zjištění:</u> Pokud pro uzly orientovaného grafu G platí

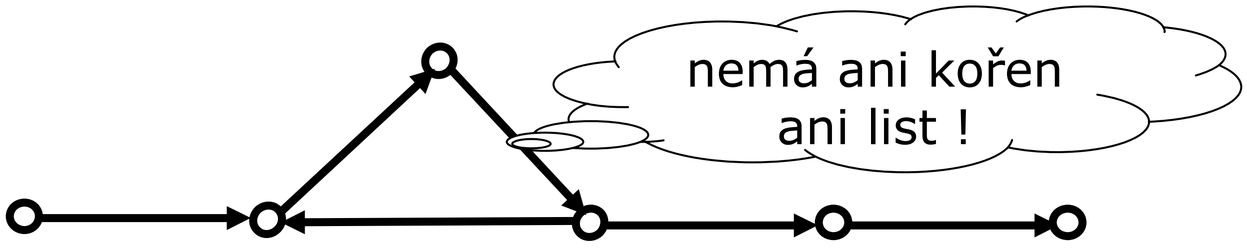
$\forall$ **u**$\in$**U:** δ^{+}**(u)** $\geq$ **1**   **nebo** $\forall$ **u**$\in$**U:** δ^{-}**(u)** $\geq$ **1 ,**

potom graf G obsahuje alespoň jeden cyklus.



Naše zjištění představuje podmínku postačující, nikoliv nutnou!

$\Rightarrow$ **Pro testování acykličnosti se nehodí.**



<u>Nové zjištění:</u> Orientovaný graf G je acyklický $\Leftrightarrow$

pro jeho libovolný kořen nebo list **u** je graf **G - {u} acyklický.**

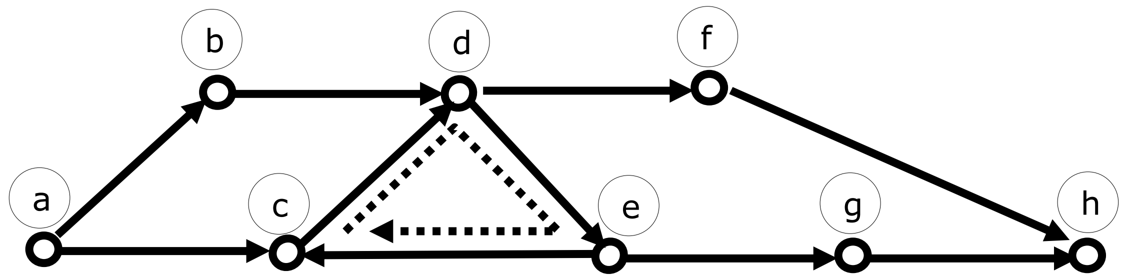
Teď už můžeme formulovat

ALGORITMUS TESTOVÁNÍ ACYKLIČNOSTI
postupným odebíráním kořenů/listů

K čemu je dobrý acyklický graf ?

Plánujeme pořadí provádění nějakých akcí, např.:

$a < b$, $a < c$, $b < d$, $c < d$, $d < e$, $d < f$, $e < g$, $e < c$, $f < h$, $g < h$



Tyto akce NELZE reálně naplánovat. Proč?

Odpovídající graf není acyklický

Topologické uspořádání uzlů (obyčejného) orientovaného grafu je posloupnost $u_1, u_2, \ldots, u_n$ taková, že každá hrana (u_i, u_j) má $i<j$.

Topologické uspořádání hran (obyčejného) orientovaného grafu je posloupnost $h_1, h_2, \ldots, h_m$ taková, že každá dvojice **navazujících** hran h_i, h_j má $i<j$ (co jsou to "navazující" hrany ?)

Jak bychom našli topologické uspořádání uzlů ?

Budeme postupně odebírat kořeny grafu ($\delta^-(u)=0$) jako při testu acykličnosti. Jak to efektivně zařídit ?

- **pro každý uzel spočítáme**
  - **$\delta^-(u)$ … vstupní stupeň (je-li =0, je to kořen)**
  - **$\Gamma(u)$ … množinu následníků**
- **při každém vypuštění kořene** upravíme $\delta^-(u)$ pro jeho následníky, při poklesu na 0 zařadíme mezi kořeny.

Pořadí odebrání uzlů je jejich topologickým uspořádáním.

! Později uvedeme ještě jednodušší algoritmus !

Vztah orientované grafy :: binární relace

Binární relace na množině X : $R \subseteq X \times X$

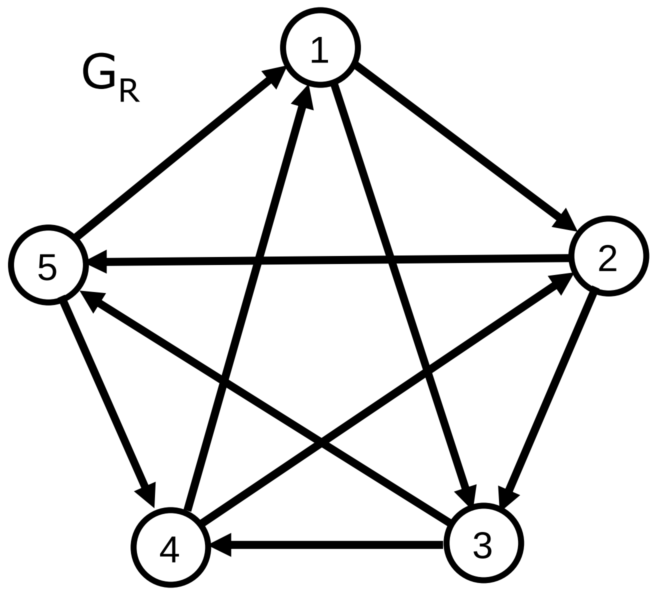
Prostý OG s množinou uzlů U : $H \subseteq U \times U$

(orientovaný) **graf binární relace** R ... G_R :

h = (u, v) vyjadřuje platnost u R v

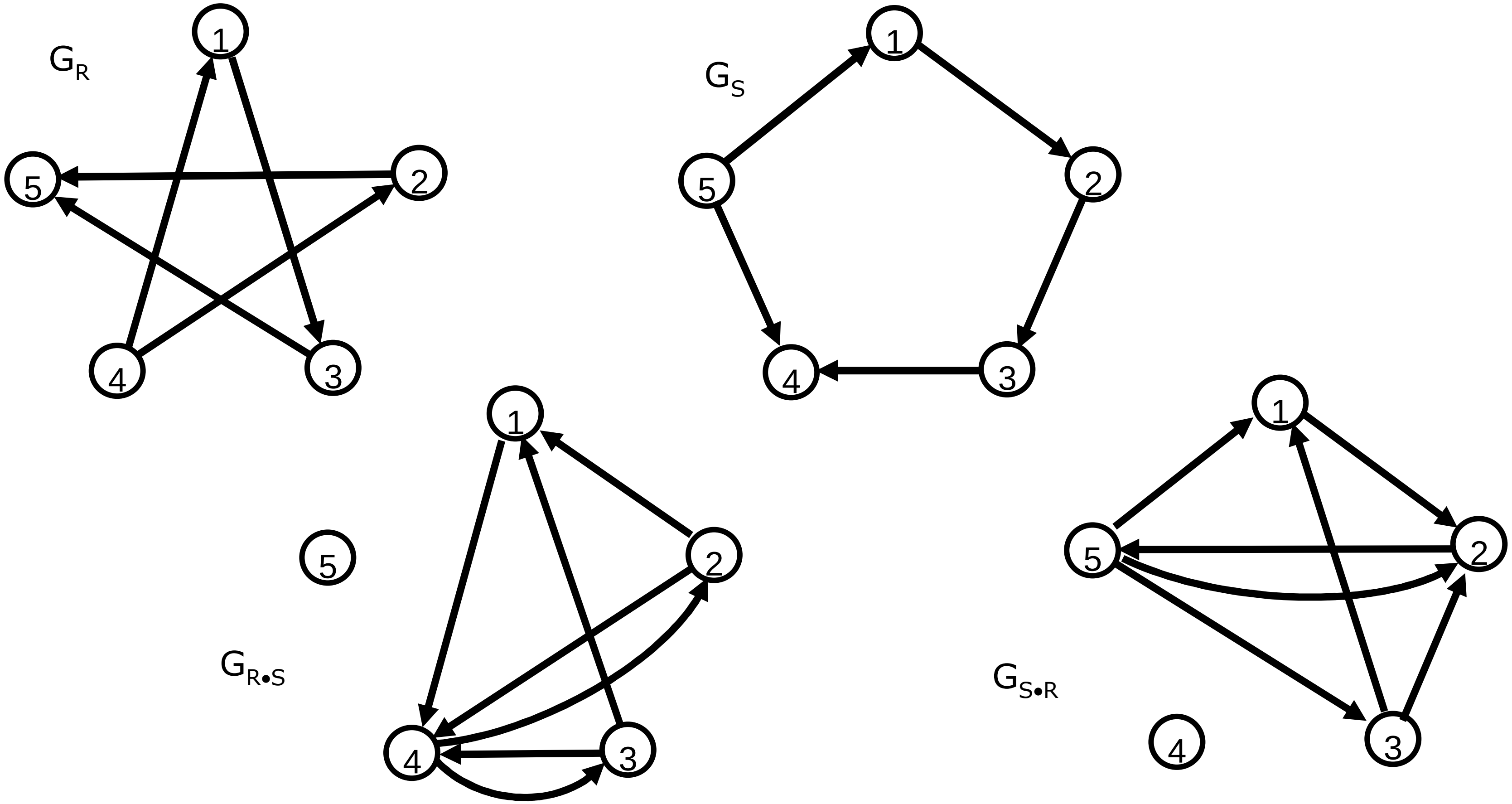
X = {1,2,3,4,5}

R = {<1,2>, <1,3>,
<2,3>, <2,5>,
<3,4>, <3,5>,
<4,1>, <4,2>,
<5,1>, <5,4>}



Vztah orientované grafy :: binární relace

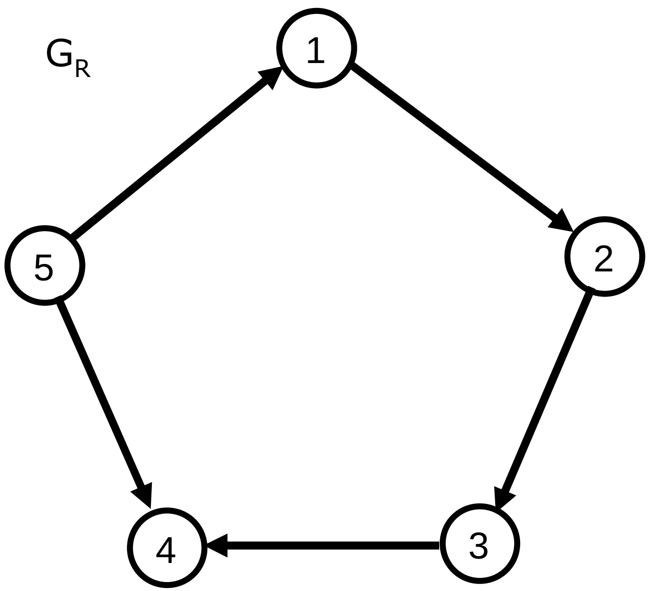
Složení grafů $G_R \bullet G_S = G_{R \bullet S}$

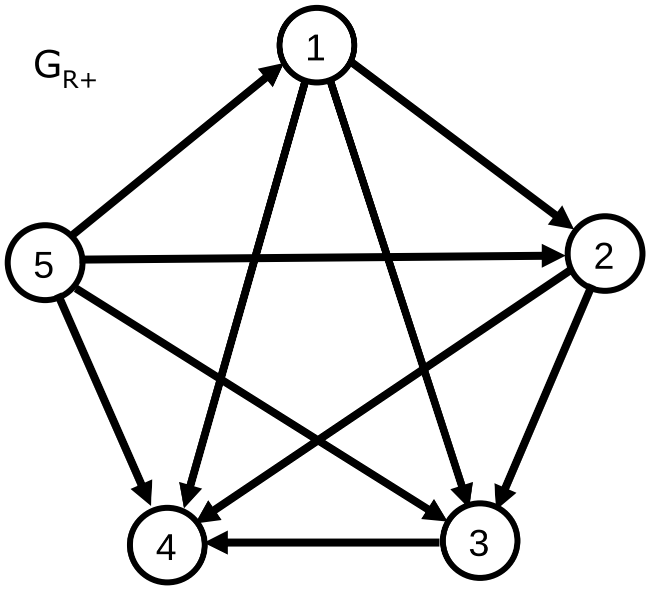


Vztah orientované grafy :: binární relace

(RE), (SY), (TR), (ANS), (AS), (IR) - jak se projeví v grafu ??

Tranzitivní uzávěr grafu G





Kontrolní otázky

3.1 **Navrhněte algoritmus topologického očíslování hran acyklického orientovaného grafu.**

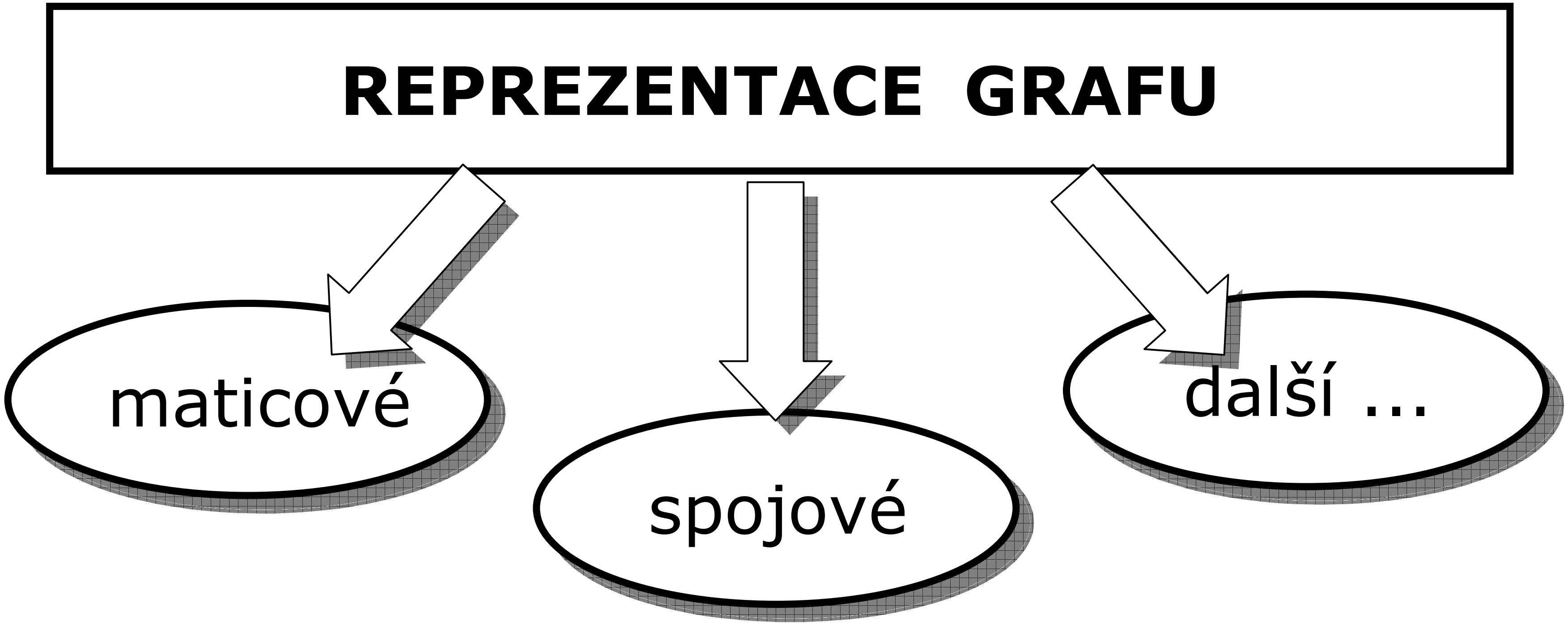
3.2 **Zdůvodněte, proč pro testování acykličnosti (resp. hledání topologického uspořádání uzlů) orientovaného grafu stačí vypouštět jenom kořeny (nebo jenom listy).**

3.3 **Je topologické uspořádání uzlů (hran) orientovaného grafu určeno jednoznačně ?**

3.4 **Kolika různými způsoby lze orientovat úplný neorientovaný graf o n uzlech K_n tak, aby byl výsledný graf acyklický ?**

3.5 **Popište strukturu obyčejného orientovaného grafu s n uzly, který má pro danou hodnotu k ($1 \leq k \leq n-1$) přesně k! . (n-k)! různých topologických uspořádání uzlů.**

3.6 **Popište strukturu grafu binární relace, která je reflexivní (resp. symetrická, antisymetrická, asymetrická, tranzitivní, ireflexivní).**



V této části se seznámíme s pojmy:

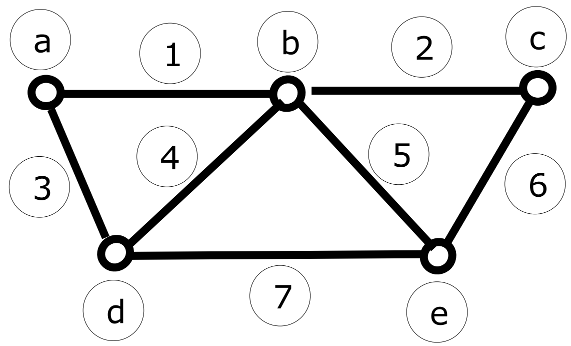
- matice incidence NG/OG, matice sousednosti NG/OG
- (základní) spojová reprezentace NG/OG

Skripta odstavec 4.1, str. 65 - 74

Matice incidence NG

A = **[** a_{ik} **]** obdélníková matice typu |U| x |H| **nad tělesem mod 2** (pozor na základní operace!)

$$a_{ik} = \begin{cases} 1 & \text{.... hrana } h_k \text{ inciduje s uzlem } u_i \\ 0 & \text{.... jinak} \end{cases}$$



	1	2	3	4	5	6	7
a	1	0	1	0	0	0	0
b	1	1	0	1	1	0	0
c	0	1	0	0	0	1	0
d	0	0	1	1	0	0	1
e	0	0	0	0	1	1	1

? Co nám říká matice incidence o grafu ?

? Smyčky, rovnoběžné hrany ?

Poznáme podle matic incidence, zda jsou dva grafy izomorfní?
Např.: $G_1 \cong G_2$ **?? právě když ??** $\mathbf{A}_1 = \mathbf{A}_2$

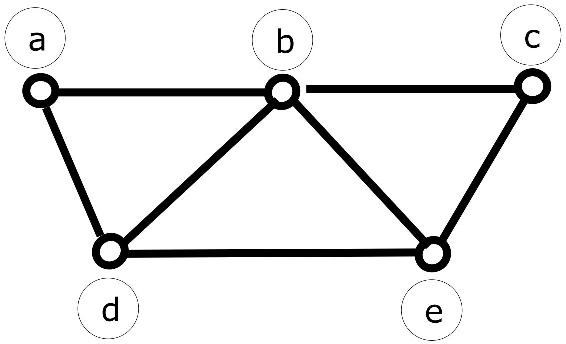
Zjištění:

- **součet (mod 2) ve sloupci je 0 (vždy dvě jedničky!) $\Rightarrow$ řádky jsou lineárně závislé, takže hodnost matice A ...**
- **h(A) $\leq$ |U| - 1 (rovnost platí pro souvislé grafy)**
- **h(A) = |U| - p obecný vztah pro graf s p komponentami**

Matice sousednosti NG

V = **[** v_{ij} **]** čtvercová matice typu |U| x |U| **nad okruhem celých čísel:**

v_{ij} = počet hran mezi uzly u_i a u_j



	a	b	c	d	e
a	0	1	0	1	0
b	1	0	1	1	1
c	0	1	0	0	1
d	1	1	0	0	1
e	0	1	1	1	0

? Co nám říká matice sousednosti o grafu ?

? Smyčky, rovnoběžné hrany ?

Poznáme podle matic sousednosti, zda jsou dva grafy izomorfní?

Např.: $G_1 \cong G_2$ **?? právě když ??** $\mathbf{V}_1 = \mathbf{V}_2$

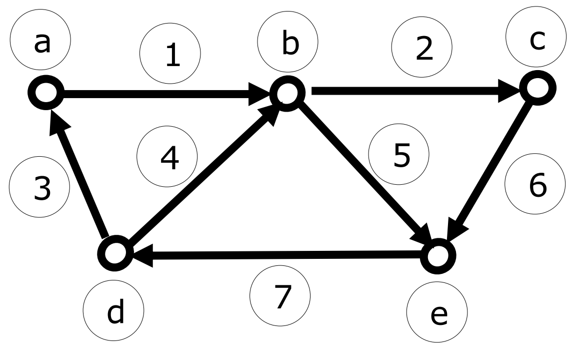
<u>Zjištění:</u>

- $\mathbf{V} = \mathbf{V}^T$
- $\mathbf{V}^{\mathbf{r}} = [v_{ik}^{(r)}]$ ... **počet sledů** délky **r** mezi u_i a u_k
- $\mathbf{A} \,.\, \mathbf{A}^T = \mathbf{V} + \mathbf{D}$, $\mathbf{D} = [\, d_{ii} \,]$, kde $d_{ii} = \delta(u_i)$

Matice incidence OG

A = **[** a_{ik} **]** obdélníková matice typu |U| x |H| **nad okruhem celých čísel:**

$$a_{ik} = \begin{cases} 1 & \text{... hrana } h_k \text{ vychází z uzlu } u_i \\ -1 & \text{... hrana } h_k \text{ končí v uzlu } u_i \\ 0 & \text{... jinak} \end{cases}$$



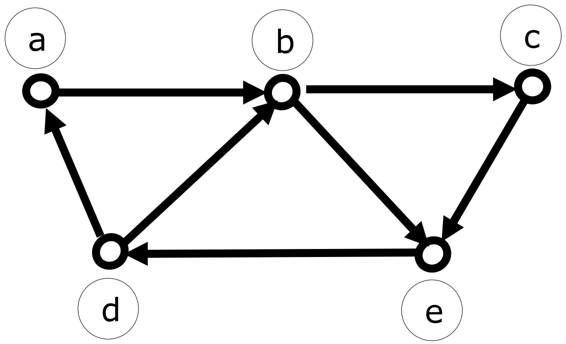
	1	2	3	4	5	6	7
a	1	0	-1	0	0	0	0
b	-1	1	0	-1	1	0	0
c	0	-1	0	0	0	1	0
d	0	0	1	1	0	0	-1
e	0	0	0	0	-1	-1	1

Vlastnosti podobné jako pro NG

Matice sousednosti OG

V = **[** v_{ij} **]** čtvercová matice typu |U| x |U| **nad okruhem celých čísel:**

v_{ij} = počet hran z uzlu u_i do uzlu u_j



	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	0	1
c	0	0	0	0	1
d	1	1	0	0	0
e	0	0	0	1	0

	a	b	c	d	e
a	0	1	0	1	0
b	1	0	1	1	1
c	0	1	0	0	1
d	1	1	0	0	1
e	0	1	1	1	0

? Poznáme izomorfní orientované grafy ?

? V = V^T ?

Zjištění:

- $\mathbf{V^r} = [v_{ik}^{(r)}]$ ... **počet spojení** délky **r** z u_i do u_k
- $\mathbf{V^*} = \sum \mathbf{V^i}$, $i=0, \dots d$, kde $d = \min(|H|, |U|-1)$

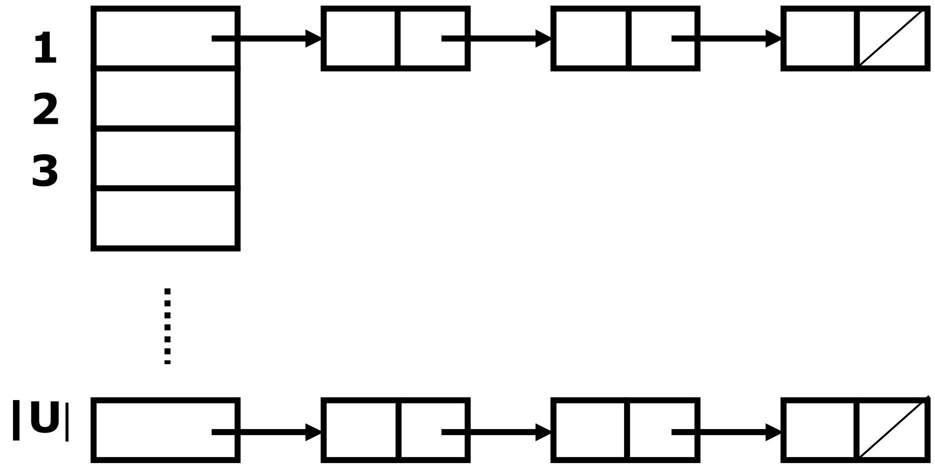
?? Co asi říká o grafu tato matice V* ??

- $\mathbf{A . A^T = D - V - V^T}$, $\mathbf{D} = [\, d_{ii} \,]$, kde $d_{ii} = \delta^+(u_i) + \delta^-(u_i)$

Spojová reprezentace grafu

NG - seznamy sousedů
OG - seznamy následníků

Adj[u]:



<u>Srovnání paměťové složitosti:</u>

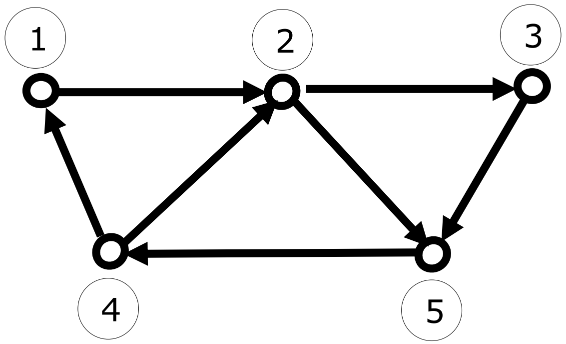
NG A: |U|.|H| (bitů!) V: |U|.|U| (integer ? Boolean)

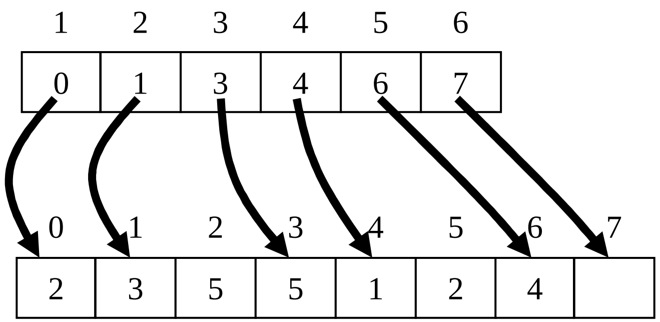
Adj: |U| + 2.|H| **OG** Adj: |U|+|H|

Základní spojovou reprezentaci mohou ještě doplňovat:

- seznamy předchůdců uzlů (pro OG)
- u každého prvku v seznamu následníků/předchůdců lze uvést i odpovídající označení (číslo) hrany
- přiřazení dvojic (uspořádaných dvojic) uzlů hranám (ρ a σ)
- ohodnocení/označení uzlů a/nebo hran
- atd.

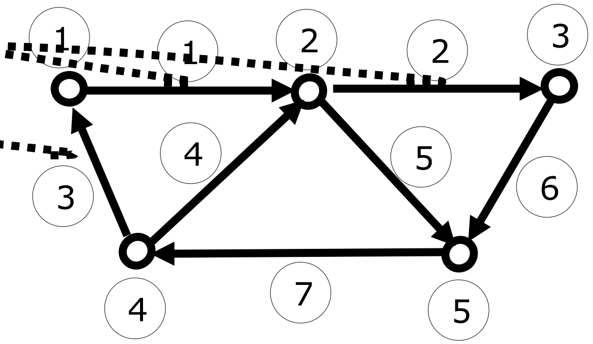
Místo spojové reprezentace seznamů následníků/předchůdců je též možno použít např. uložení v poli:





Až dosud jsme měli na mysli **vnitřní reprezentaci grafů**, ale jsou také různé **vnější reprezentace**, jimiž lze zadat graf na vstupu nějakého programu, např.:

```
// počet uzlů a hran
5  7
// hrany jako dvojice uzlů
1  2
2  3
4  1
4  2
2  5
3  5
5  4
```



Vzpomeňte si na další možnosti ...

Vlastní reprezentaci mohou mít některé speciální typy grafů, jako např.:

- kořenové stromy
- pravidelné stromy
- atd.

Tím jsme se již zabývali na prvním prosemináři ...

Kontrolní otázky

3.7 Jak se z matice incidence neorientovaného grafu určí množina sousedů zadaného uzlu? Jaká bude časová složitost této operace?

3.8 Jak se z matice incidence orientovaného grafu určí množina předchůdců zadaného uzlu? Jaká bude časová složitost této operace?

3.9 Přesně popište, jaký bude vztah matice sousednosti (obecného) neorientovaného grafu G a matice sousednosti grafu G', který vznikl nějakou orientací hran grafu G.

3.10 V jakém orientovaném grafu bude r-tá mocnina V^r matice sousednosti V obsahovat počty různých orientovaných cest mezi jednotlivými uzly ?

3.11 Jak se z matice sousednosti neorientovaného grafu určí množina sousedů zadaného uzlu? Jaká bude časová složitost této operace?

3.12 Jak se z matice sousednosti orientovaného grafu určí množina předchůdců zadaného uzlu? Jaká bude časová složitost této operace?

3.13 Navrhněte algoritmus převodu matice incidence A neorientovaného grafu na jeho matici sousednosti V.

3.14 Navrhněte algoritmus převodu matice sousednosti V orientovaného grafu na jeho matici incidence A.

Prohledávání grafů

V této části probereme témata

- strom prohledání do šířky (BF-strom)
- časová složitost prohledání do šířky
- rozklad neorientovaného grafu na komponenty

Skripta odstavec 4.2, str. 74 - 79

Prohledávání grafu do šířky

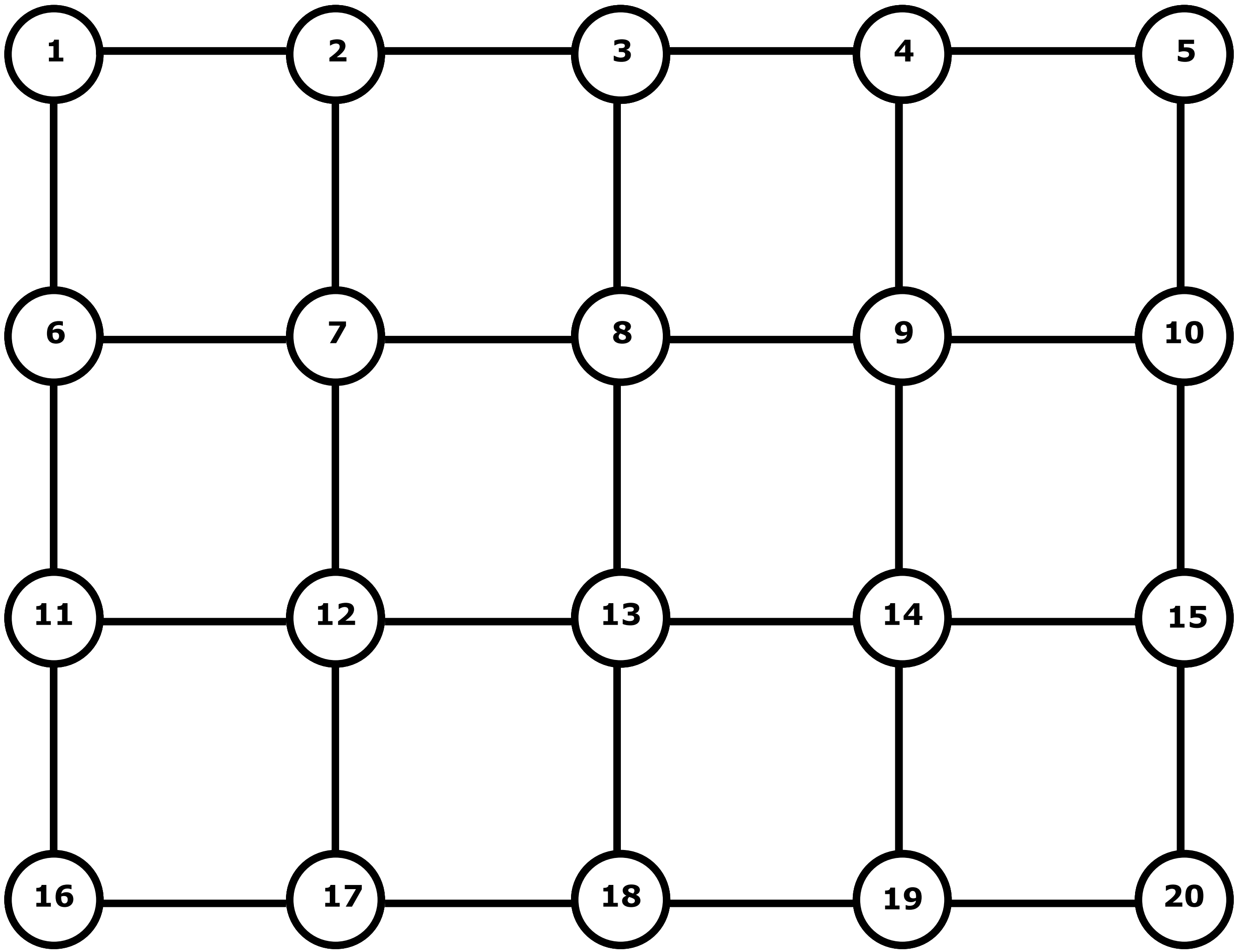
BFS - Breadth-First Search

Je zadán graf $G = \langle H,U,\sigma \rangle$ (není podstatné, zda NG nebo OG) a jeho uzel $s \in U$.

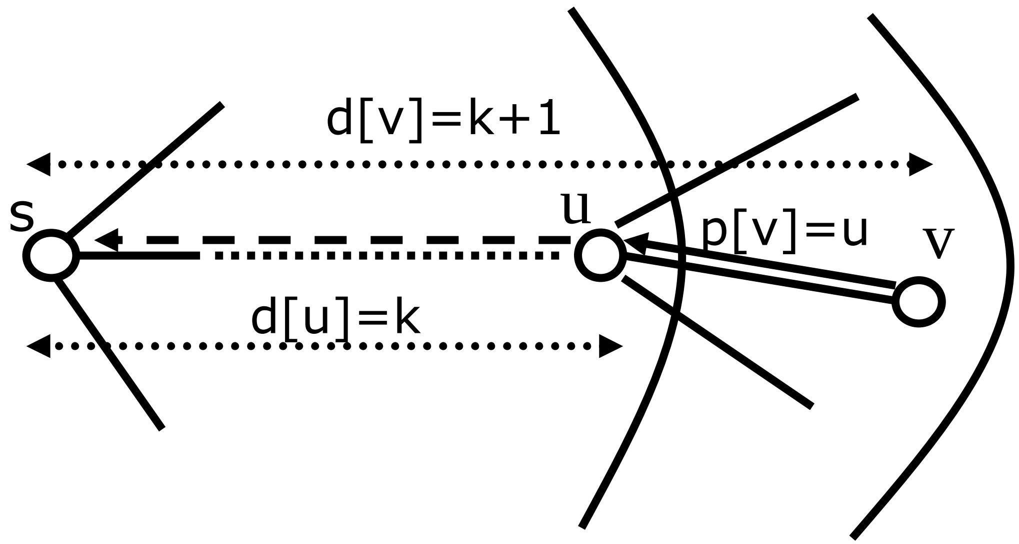
Prohledáním do šířky dostaneme strom (nejkratších) $s \rightarrow u$ cest pro všechny uzly u dostupné z uzlu s (BF-strom)

<u>Stavy uzlů</u>:

FRESH	-	nový (dosud neobjevený) uzel
OPEN	-	právě objevený ("nadějný") uzel
CLOSED	-	vyčerpaný uzel


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

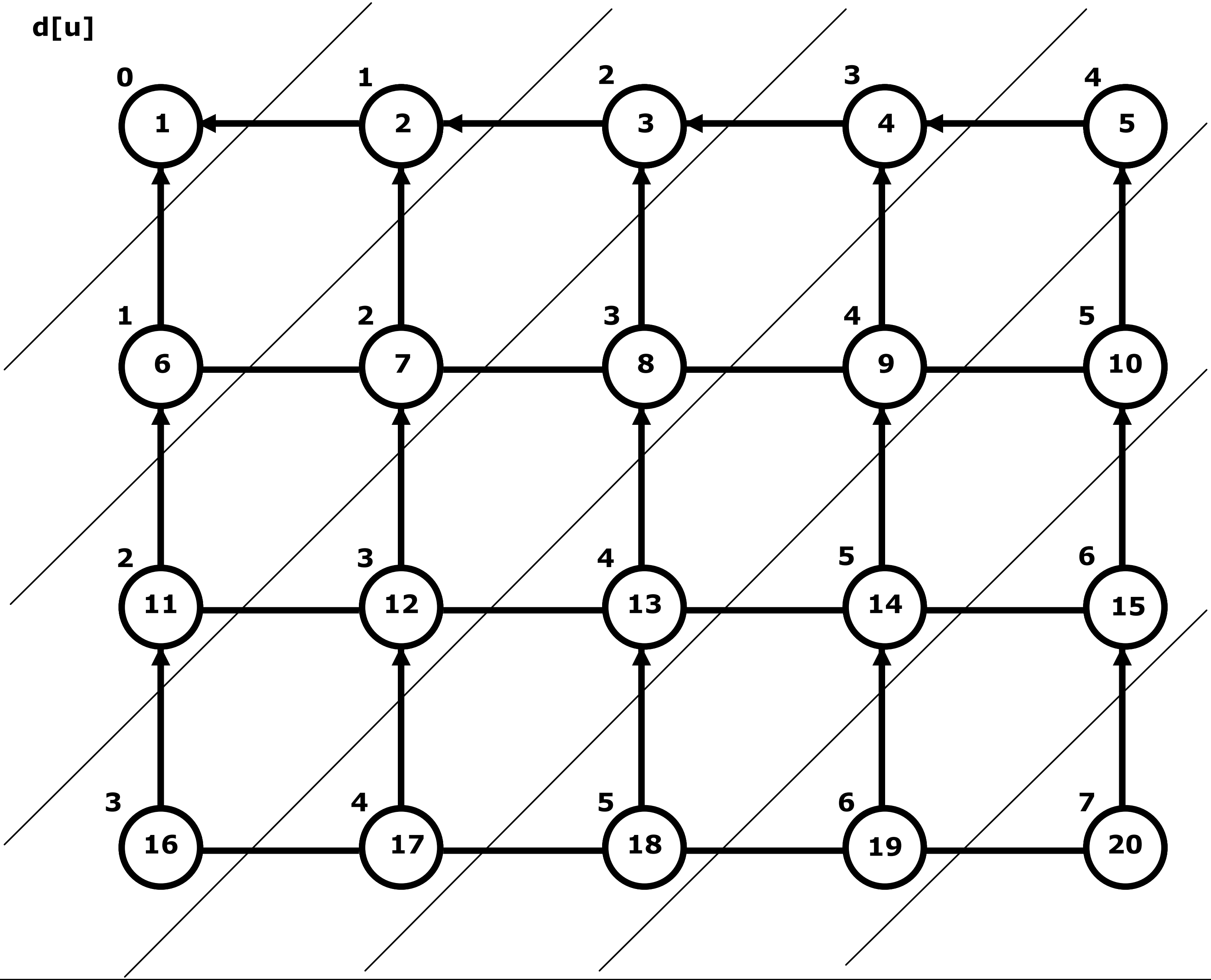
Průběh prohledávání do šířky



Použité datové struktury:

stav[u] - FRESH / OPEN / CLOSED
d[u] - zjištěná vzdálenost s $\rightarrow$ u
p[u] - předchůdce uzlu u (viz $\longrightarrow$)
Queue - fronta OPEN uzlů

```
void BFS (Graph G, Node s) {   //  pseudokód
  for (Node u in U(G)-s)
    { stav[u] = FRESH; d[u] = ∞; p[u] = null; }
  stav[s] = OPEN; d[s] = 0; p[s] = null;
  Queue.Init(); Queue.Push(s);
  while (!Queue.Empty()) {
    u = Queue.Pop();
    for (v in Adj[u]) {
      if (stav[v] == FRESH) {
        stav[v] = OPEN; d[v] = d[u]+1;
        p[v] = u; Queue.Push(v);
    } }
    stav[u]=CLOSED;
 }
```


d[u]
0
1
1
2
2
3
3
4
4
5
1
6
2
7
3
8
4
9
5
10
2
11
3
12
4
13
5
14
6
15
3
16
4
17
5
18
6
19
7
20

Jak složitý je algoritmus BFS?

- cykl na řádku 1 a 2 ... |U|
- operace s frontou O(1) na uzel $\Rightarrow$ celkem **O(|U|)**
- cykly 5-13 a 7-11 pro každého souseda $\Rightarrow$ **O(|H|)**

$$\Rightarrow \mathbf{O(|U| + |H|)}$$

Zjištění:

- pokud fronta obsahuje uzly v_1, v_2, ..., v_r, potom platí

  $$\mathbf{d[v_r]} \le \mathbf{d[v_1] + 1}$$

  $$\mathbf{d[v_i]} \le \mathbf{d[v_{i+1}]} \text{ pro } i=1,2,\ldots,r-1$$

- BFS nalezne nejkratší **s** $\rightarrow$ **v** cestu pro každý uzel **v** dosažitelný z uzlu **s** a **(p[v],v)** určuje její poslední hranu
- všechny tyto hrany tvoří tzv. **BF-strom**

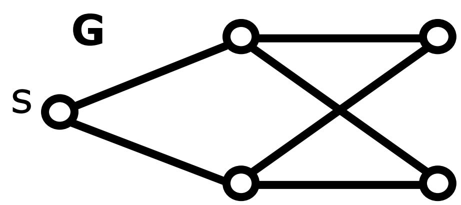
Kontrolní otázky

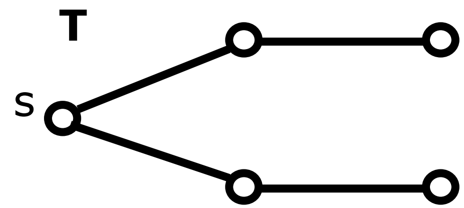
3.15 Změní se nějak chování či výsledek algoritmu BFS, pokud příkaz na řádku 12 umístíme bezprostředně za řádek 6?

3.16 Jak budou výsledky algoritmu BFS (tzn. vytvořený BFS-strom a hodnoty d[u]) ovlivněny změnou pořadí uzlů v seznamech sousedů ?

3.17 Změní se nějak složitost algoritmu BFS, pokud namísto spojové reprezentace použijeme k vyjádření struktury grafu jeho matici incidence A (resp. matici sousednosti V) ?

3.18 Zdůvodněte, proč nelze následující strom T získat jako BFS-strom při prohledání grafu G do šířky pro žádné uspořádání uzlů v seznamech sousedů uzlů, přestože strom T představuje jeden z možných stromů nejkratších cest z uzlu s do všech ostatních uzlů.





3.19 Jak vypadá neorientovaný graf, jehož BFS strom má při libovolném uspořádání uzlů v seznamu sousedů tvar hvězdice ?

3.20 Upravte algoritmus BFS tak, aby určoval počet a strukturu (tj. skupiny uzlů) jednotlivých komponent neorientovaného grafu.