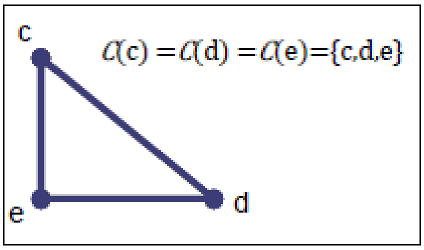
# Neorientované a orientované grafy, jejich reprezentace. Prohledávání grafu (do hloubky a do šířky), topologické uspořádání, souvislost, stromy, minimální kostra

## 1. Graf

= uspořádaná dvojice **vrcholů** V a **hran** E taková, že $E \subseteq \binom{V}{2}$

G = (V, E)
- **orientovaný graf** je uspořádaná dvojice vrcholů
- **neorientovaný graf** je neuspořádaná dvojice vrcholů

-**pojmy**: vážený graf, incidence, stupeň vrcholu deg(u), vstupní stupeň $\deg^+(u)$a výstupní stupeň $\deg^-(u)$, multigraf (vícenásobné hrany)
- **součet stupňů přes všechny vrcholy je sudý**: $\sum_{v \in V} \deg(v) = 2|E|$
- **úplný graf**: pro všechny uzly platí, že deg(u)=|V|-1
- **cesta** = posloupnost vrcholů a hran, kde vrcholy jsou navzájem různé
- **cyklus** = cesta, kde první a poslední uzel jsou totožné
- **souvislost**: pro každé dva vrcholy x a y existuje cesta z x do y
- **podgraf**: graf H je podgrafem G, pokud H vznikl vymazáním některých uzlů případně hran z grafu G
- **komponenta souvislosti** C(u) daná uzlem u je mn. uzlů v, ze kterých existuje cesta do u



Obrázek 1 - Komponenta souvislosti

## 2. Strom

- **souvislý** graf **bez cyklů**
- přidáním libovolné nové hrany vznikne cyklus
- odebráním libovolné hrany přestane být souvislý
- má |V|-1 hran
- každé dva vrcholy jsou spojeny pouze jednou cestou
- pojmy: **list, kořen**

## 3. Reprezentace grafů v paměti

### Matice sousednosti

- čtvercová matice A reprezentující vrcholy $v_1$ až $v_n$

$$a_{i,j} = \begin{cases} 1 & \text{pro } \{v_i, v_j\} \in E \\ 0 & \text{jinak} \end{cases}$$

**Matice vzdáleností**

- čtvercová matice A
reprezentující vrcholy $v_1$ až $v_n$

$$a_{i,j} = \begin{cases} w(\{v_i, v_j\}) & \text{pro } \{v_i, v_j\} \in E \\ 0 & \text{jinak} \end{cases}$$

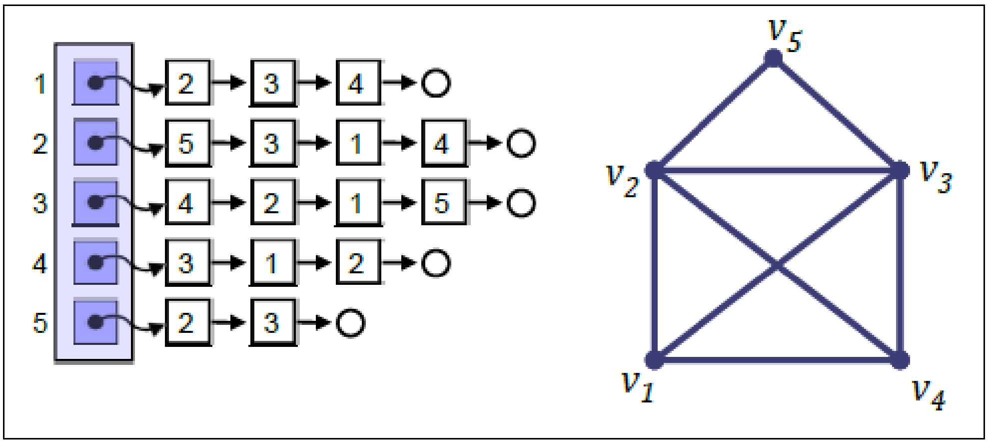
**Matice incidence**

- $|V| = n$, $|E| = m$
- obdélníková matice A o velikosti n krát m

$$(I)_{i,j} = \begin{cases} -1 & \text{pro } e_j = (v_i, *) \\ +1 & \text{pro } e_j = (*, v_i) \\ 0 & \text{jinak} \end{cases}$$

**Seznam sousedů**

- implementováno spojovým seznamem




Obrázek 2 - Seznam sousedů pomocí spojového seznamu

## 4. DFS (Depth First Search)

- algoritmus prohledává graf do hloubky
- nejdřív se prohledají potomci a pak teprve sourozenci
- implementováno pomocí **zásobníku (LIFO) nebo pomocí rekurze** (paměťově náročnější)
- **využití**: zjišťování acykličnost a souvislosti grafu, hledání komponent souvislosti grafu, převod grafu na orientovaný les

```
dfs(Vertex v){
	Stack to_visit = empty;
	Vertices visited = empty;
	to_visit.push(v);
	while( !to_visit.empty() ){
		v = to_visit.pop();
		if( !visited.contains(v) ){
			visited.add(v);
			for( n in v.neighbors() ) {
				to_visit.push(n);
			}
		}
	}
}
```

## 5. BFS (Breadth First Search)

- algoritmus prohledává graf do šířky
- nejdřív se prohledají sourozenci a pak teprve potomci
- implementace úplně stejná jako DFS, akorát se použije místo zásobníku **fronta (FIFO)**

## 6. Topologické uspořádání

- **binární relace** nad grafem G, který je DAG
- relaci R(x,y) můžeme definovat např.: R(x,y) platí, právě když z x vede orientovaná cesta do y
- získáme např. po průchodu DFS
- používá se např. pro plánování na sobě závislých činností. Pokud tyto činnosti vykonáváme v topologickém pořadí, tak daná činnost bude vykonána až po všech činnostech, na kterých závisí

## 7. Minimální kostra grafu

- kostra H grafu G je stromový podgraf takový, že V(G)=V(H)
- min. kostra je taková kostra, která má sumu vah svých hran minimální

### Jarníkův (Primův) algoritmus

- časová složitost je O( |V(G)| * |E(G)| )

```
K = {v0};          // v0 je libovolný vrchol
while( |V(K)| != |V(G)| ){
      vybereme hranu {u,v}∈E(G), kde u∈V(K) a v∉V(K) tak, aby w({u,v}) byla
      minimální
      K = K + {u,v}
}
```

### Borůvkův algoritmus

- časová složitost je O( |E(G)| * log|V(G)| )

```
K = V(G);
while( K má alespoň dvě komponenty souvislosti ){
      pro každou komponentu Ti grafu K vybereme nejlehčí incidentní hranu ti
      všechny hrany ti přidáme do K
}
```

- na začátku každý uzel představuje jednu komponentu
- v každé iteraci se počet uzlů v komponentě minimálně zdvojnásobí, proto se alg. zastaví po maximálně `log(|V(G)|)` iteracích
- každá komponenta si pamatuje dosud nejlehčí hranu

**Union-Find:**

1. každý uzel si pamatuje reprezentanta své komponenty
2. sloučení dvou **komponent** znamená jejich propojení hranou a nastavení společného reprezentanta

**Kruskalův („hladový“) algoritmus**

- časová složitost je O( |E(G)| * log(|V(G)|) )

```
setřídíme všechny hrany vzestupně podle váhy
for(e in seřazené hrany){
      if( K + e je acyklický graf ){
            K = K + e;
      }
}
```

- potřebujeme udržovat komponenty souvislosti, abychom určili, jestli hrana vytvoří cyklus (viz Union-Find)

## 8. Zdroje

[1] Genyk-Berezovskyj, Marko: Přednáška 1 z PAL.
https://cw.felk.cvut.cz/lib/exe/fetch.php/courses/a4m33pal/pal01.pdf

[2] Genyk-Berezovskyj, Marko: Přednáška 2 z PAL.
https://cw.felk.cvut.cz/lib/exe/fetch.php/courses/a4m33pal/pal02.pdf

[3] Mička, Pavel: Topologické uspořádání.
http://www.algoritmy.net/article/1381/Topologicke-usporadani