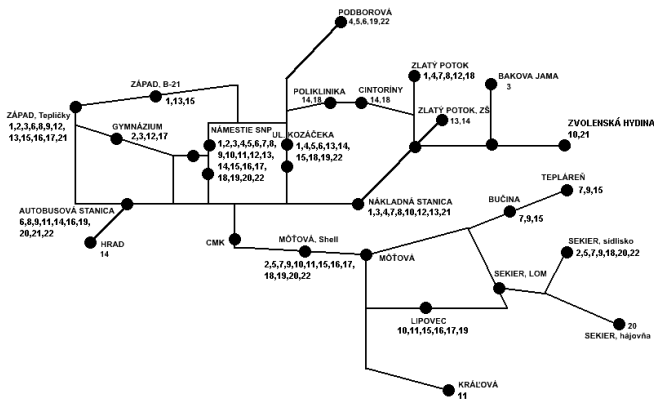




8. prednáška (17.4.2012)



Grafy a grafové algoritmy

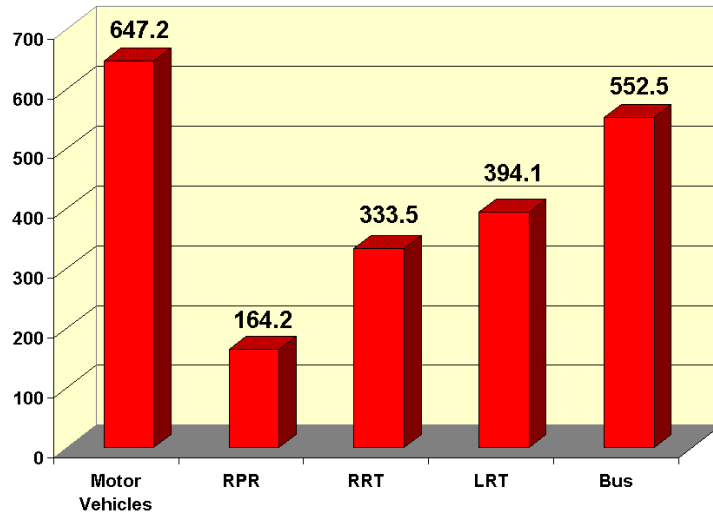


alebo

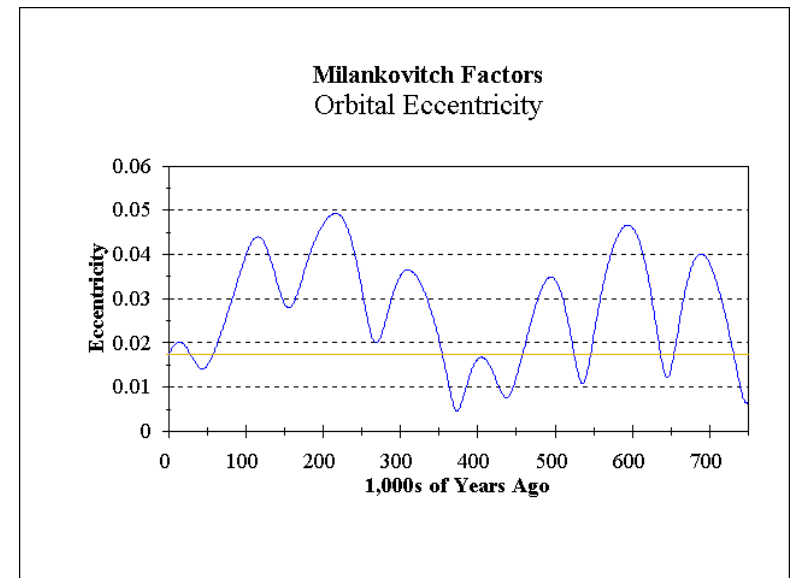
Graphs are everywhere



Čo nie sú grafy ...

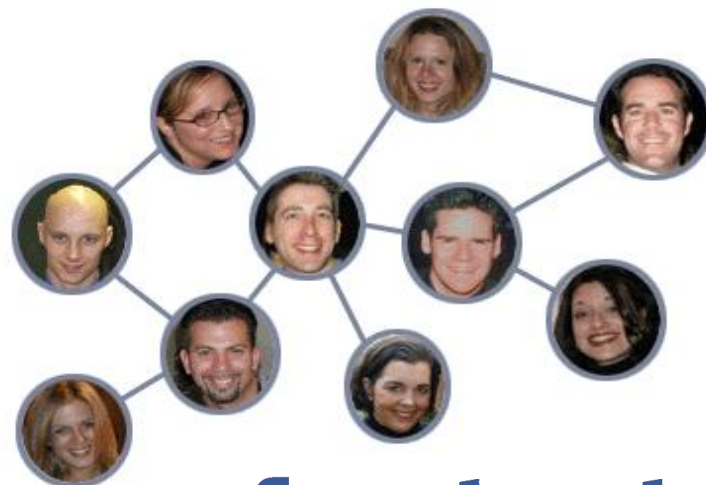
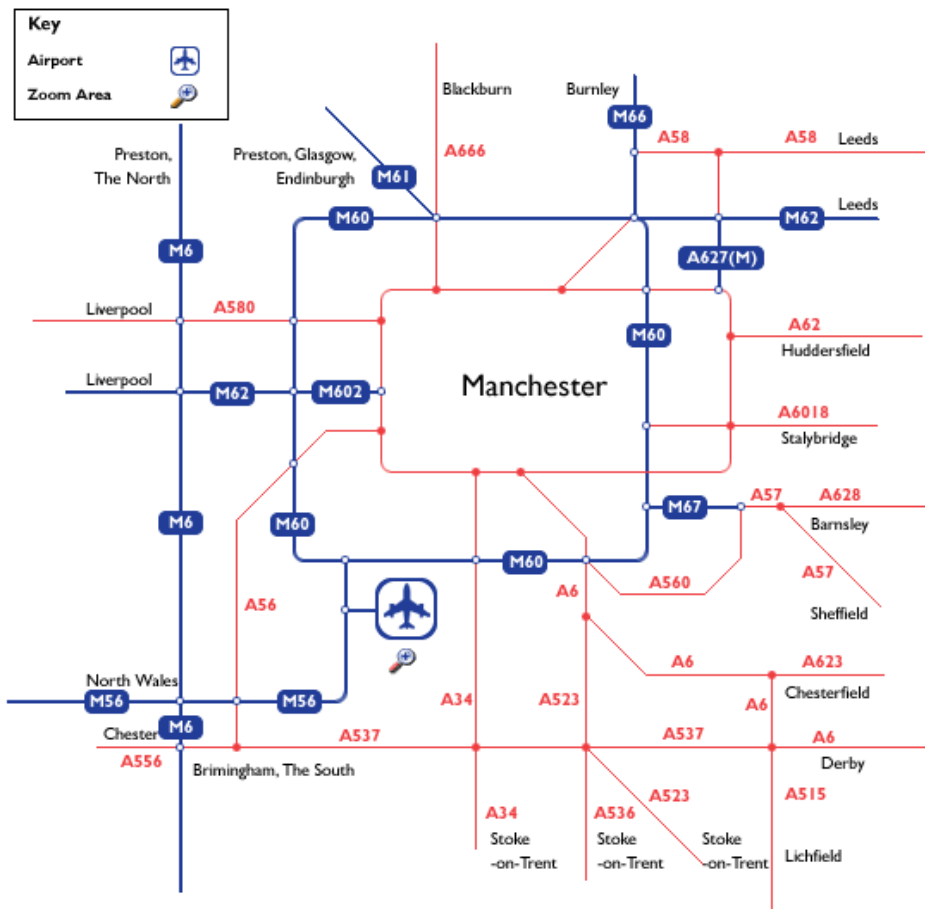


Takto vnímajú
pojem graf bežní
ľudia ...





Graphs are everywhere!



facebook®

Takto vnímajú
pojem graf
informatici a
matematici



Grafy ...

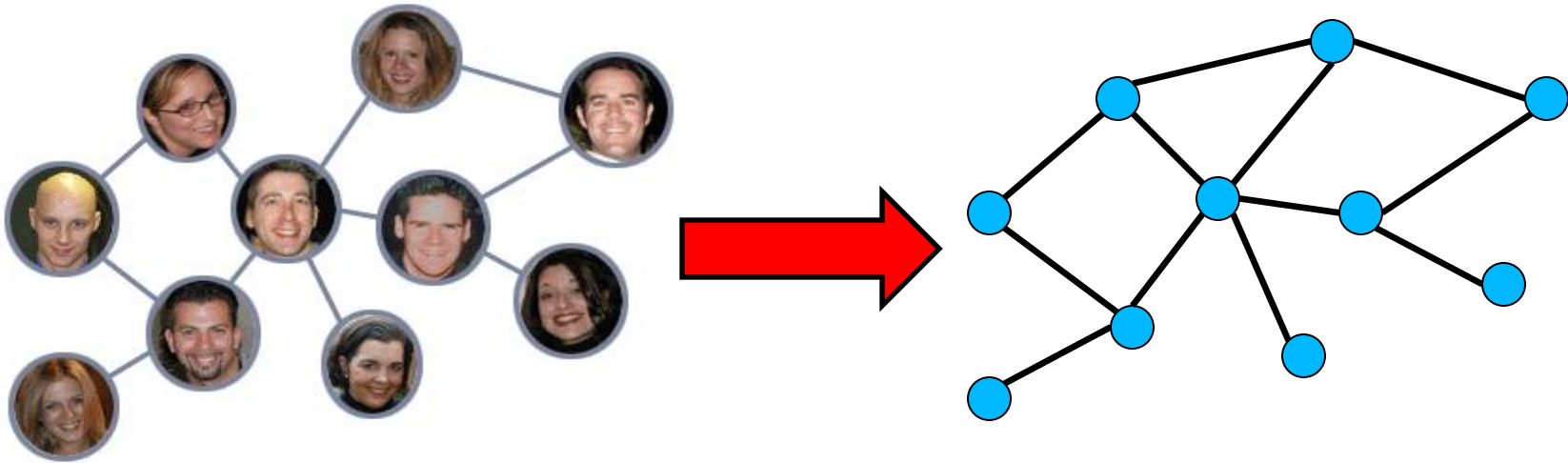
- Svet okolo nás je plný vzťahov - binárnych **relácií medzi objektmi**



- Príklady:
 - objekt: človek, relácia: poznať sa
 - objekt: mesto, relácia: byť spojený priamou cestou
 - objekty: osoby a mestá, relácia: bývať v meste
 - relácia je iba medzi objektmi rôzneho typu (osoba-mesto, nie osoba-osoba alebo mesto-mesto)
 - objekty: študenti a prednášky, relácia: študent sa zúčastnil prednášky



Zovšeobecnenie relácií ...

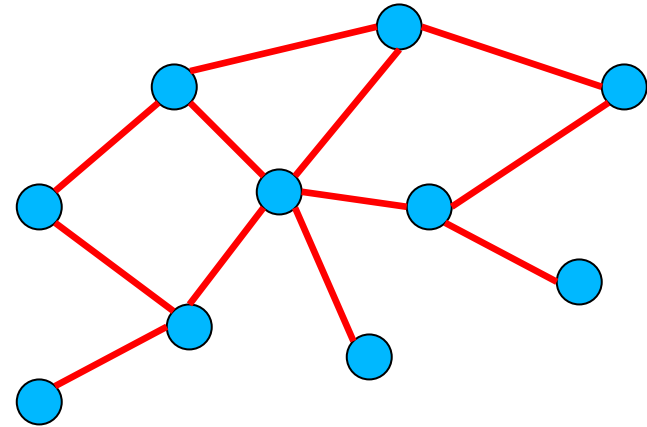


Graf = krúžky a čiary (alebo krúžky a šípky)



Grafy – formálnejšie

- **Grafom** G nazývame dvojicu (V, E) , kde:
 - V je množina **vrcholov grafu** (krúžky)
 - E ($E \subseteq V \times V$) je množina **hrán grafu** (čiary, resp. šípky)

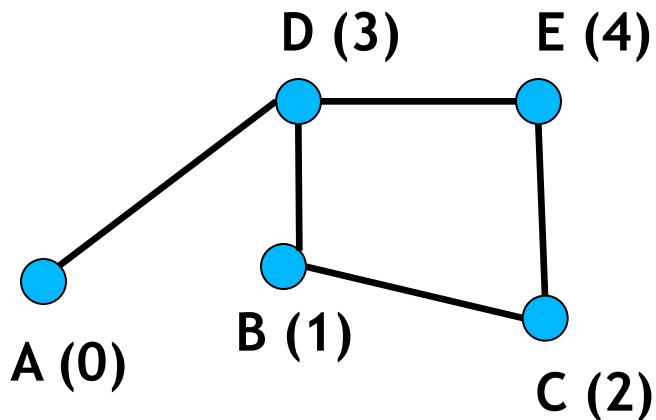


- **Príklady:**
 - vrcholy = mestá, hrany = priame cesty medzi mestami
 - vrcholy = používatelia Facebook-u, hrany = priateľstvo medzi používateľmi FB



Ako uložit' graf v programe?

Matica susednosti:



```
boolean[][] graf =
    new boolean[n][n]
```

	A	B	C	D	E
A	F	F	F	T	F
B	F	F	T	T	F
C	F	T	F	F	T
D	T	T	F	F	T
E	F	F	T	T	F

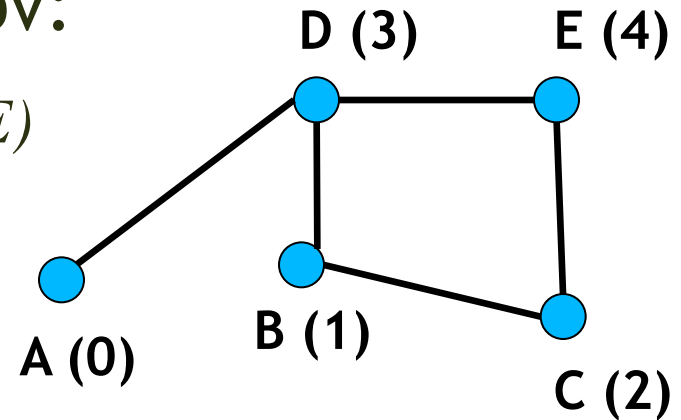
graf[u][v] = medzi vrcholmi u a v je hrana



Ako uložit' graf v programe?

- zoznam hrán + zoznam vrcholov:

- $(A, D), (B, D), (D, E), (B, C), (C, E)$
- (A, B, C, D, E)
- `List<Hrana> + List<Vrchol>?`



- **Incidenčná matica** pre graf s n vrcholmi a m hranami:

- `boolean[][] graf = new boolean[n][m]`
- `graf[u][e] = true`, práve vtedy keď vrchol u je jedným z koncových vrcholov hrany e
- hrana $e = (u, v)$ je **incidentná** s vrcholmi u a v



Ako uložit' graf v programe?

- Objektový prístup (knížnica *graph.jar*)
- Základné objekty:
 - **Graph** - reprezentuje graf
 - **Vertex** - reprezentuje vrchol grafu
 - **Edge** - reprezentuje hranu v grafe


```
Graph fb = new Graph();  
Vertex u = fb.addVertex("Janko");  
Vertex v = fb.addVertex("Marienka");  
Edge priatelstvo = fb.addEdge(u, v);  
System.out.println(fb);
```



Prehľadávanie grafov



Sú Košice odrezané od sveta?

SPOJENIE ODCHODY ZASTÁVKOVÉ CP SPOJE

Cestovný p.:  Lietadlá

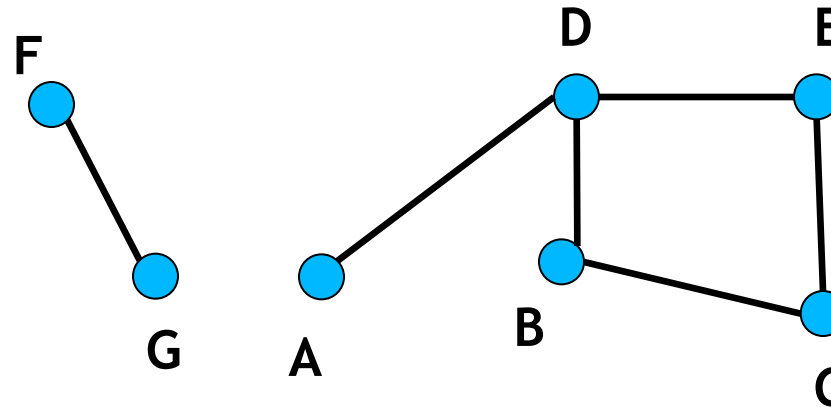
Odkiaľ: Košice

Kam: Silicon Valley

Len priame spojenia  [+ Pridať prestupné miesta](#) 



Súvislosť grafu



● Problém:

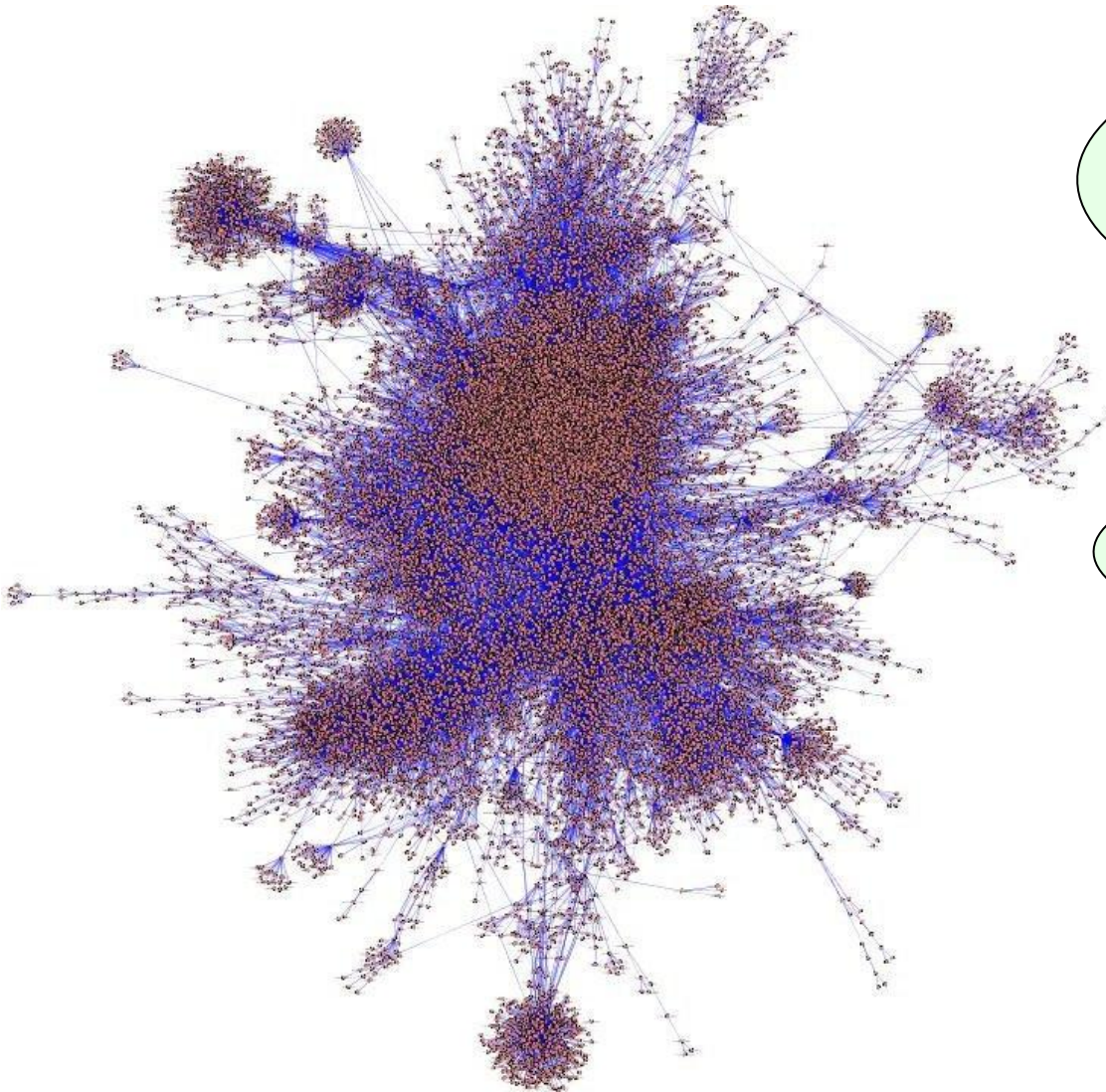
- Ako zistiť, či je graf **súvislý**, t.j., či existuje spojenie medzi každými 2 vrcholmi grafu?

● *Interpretácia:*

- súvislosť cestnej/komunikačnej siete
- letecké spojenie medzi všetkými mestami
- izolované sociálne skupinky v kolektíve



Súvislosť: Pozriem a vidím?



Pozriem a vidím
„nefunguje“ pre
veľké grafy.

„Pozriem a vidím“
v tabuľke?

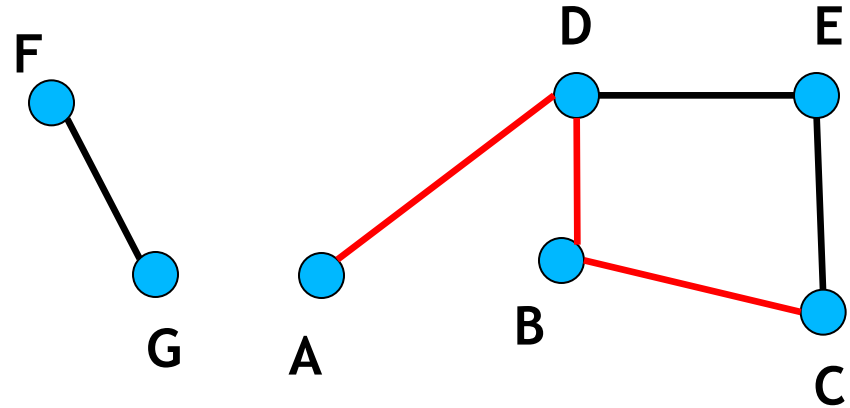
	A	B	C	D	E
A	F	F	F	T	F
B	F	F	T	T	F
C	F	T	F	F	T
D	T	T	F	F	T
E	F	F	T	T	F



Terminológia (neformálne)

● Podgraf

- ľubovoľná podmnožina vrcholov a podmnožina s nimi incidentných hrán



● Cesta

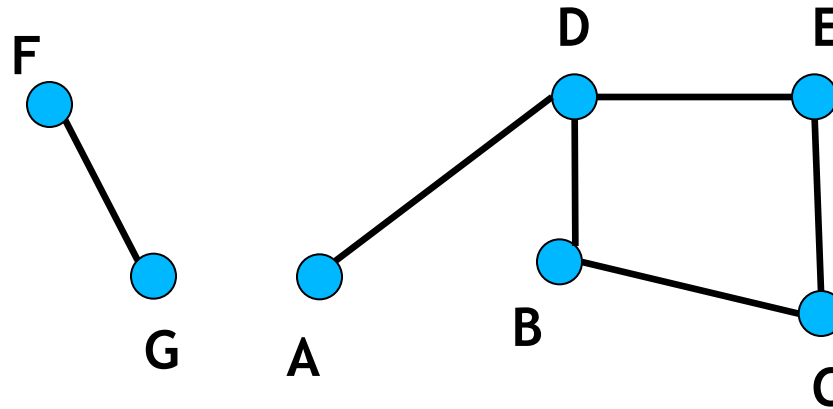
- postupnosť vrcholov grafu bez opakovania, v ktorej každé 2 za sebou idúce vrcholy sú spojené hranou
- Príklad cesty: A, D, B, C

● Súvislý graf

- medzi každými dvoma vrcholmi existuje cesta



Súvislosť grafu - idea



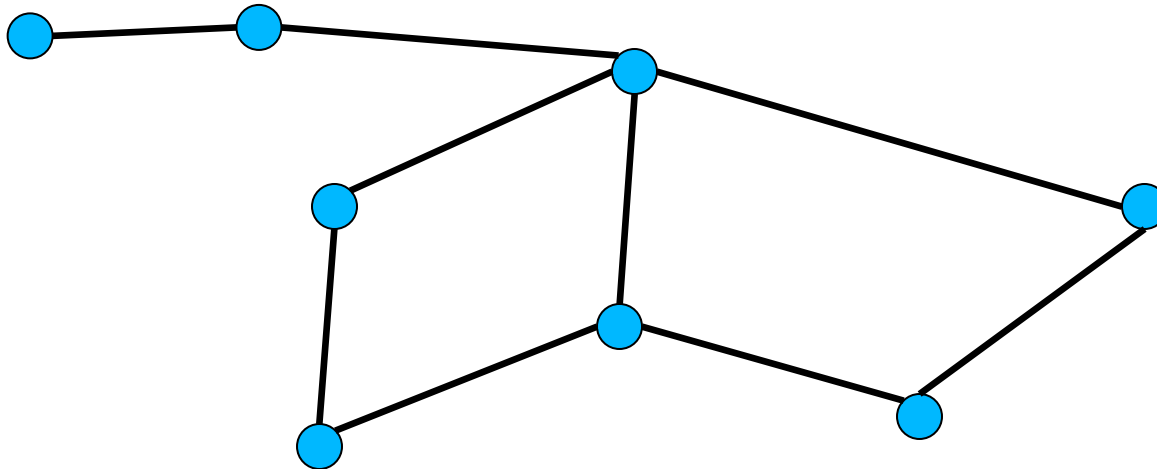
- Idea algoritmu:

- zistiť, či sa z nejakého vrcholu grafu vieme dostať do všetkých ostatných - **systematické prehľadávanie grafu**
- „implementácia“: po navštívení každého vrcholu navštívime aj všetkých jeho susedov



Prehľadávanie do šírky

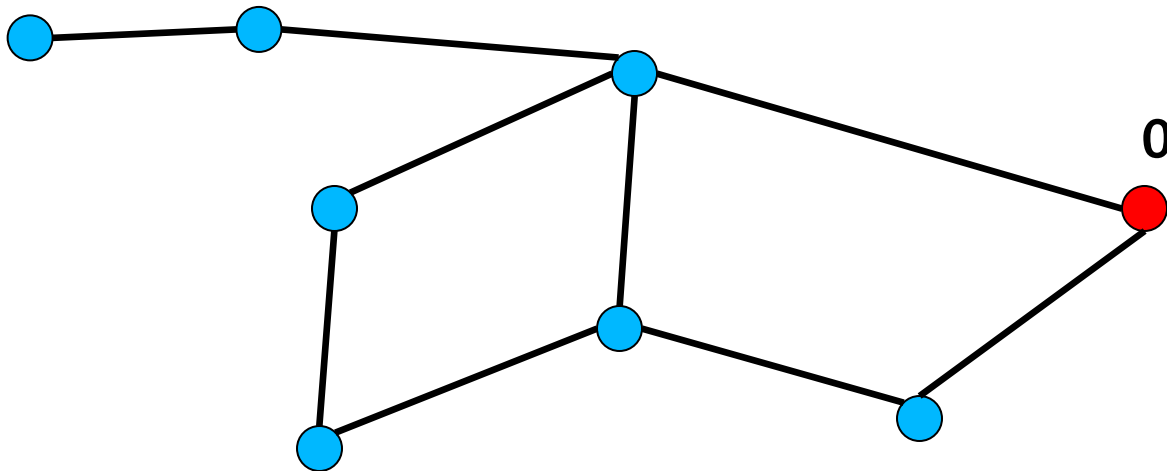
- Pracuje vo fázach:
 - v každej fáze navštívime **nenavštívených susedov**, tých vrcholov, ktoré boli po prvý krát navštívené v predchádzajúcej fáze





Prehľadávanie do šírky

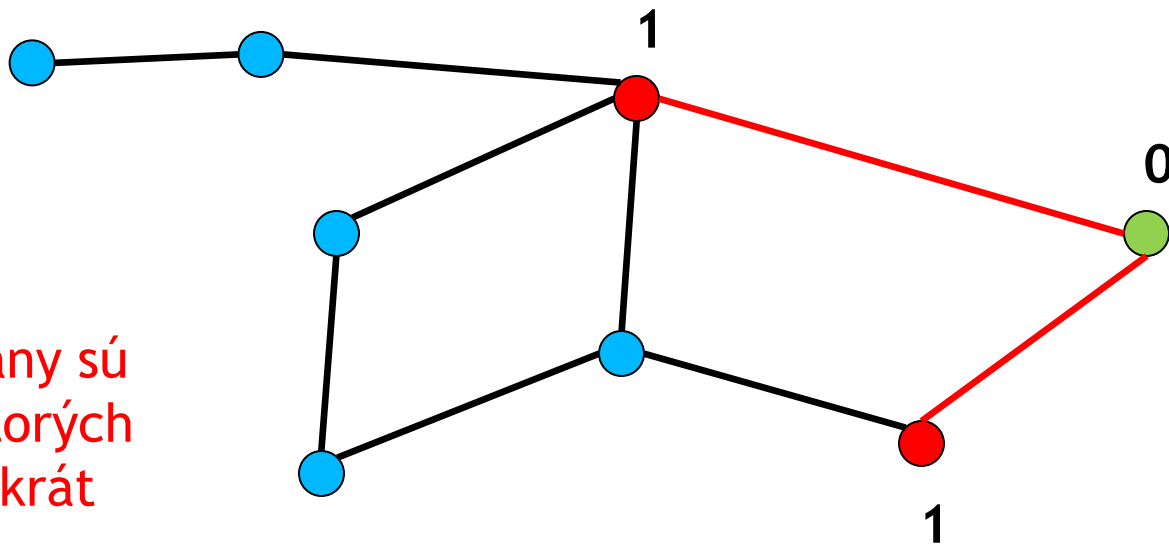
- Pracuje vo fázach:
 - v každej fáze navštívime **nenavštívených susedov**, tých vrcholov, ktoré boli po prvý krát navštívené v predchádzajúcej fáze





Prehľadávanie do šírky

- Pracuje vo fázach:
 - v každej fáze navštívime **nenavštívených susedov**, tých vrcholov, ktoré boli po prvý krát navštívené v predchádzajúcej fáze

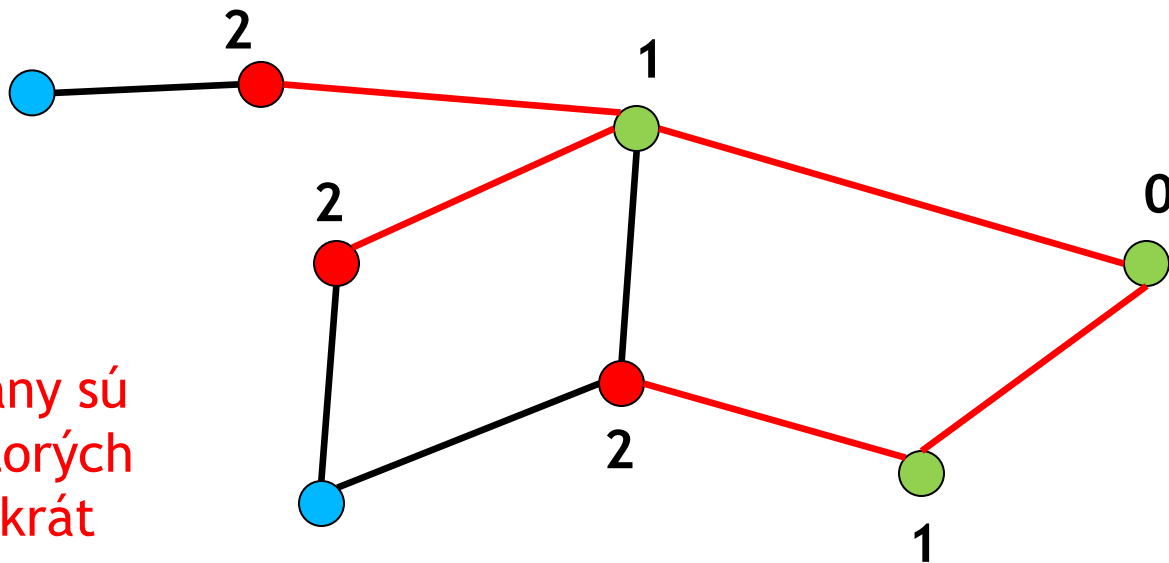


Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime



Prehľadávanie do šírky

- Pracuje vo fázach:
 - v každej fáze navštívime **nenavštívených susedov**, tých vrcholov, ktoré boli po prvý krát navštívené v predchádzajúcej fáze

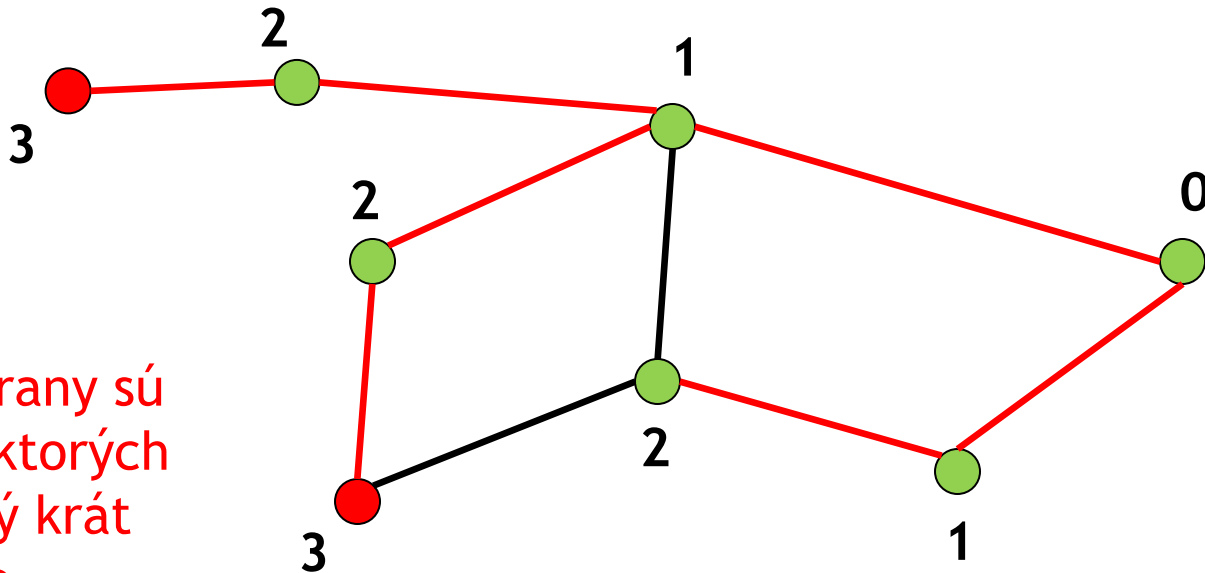


Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime



Prehľadávanie do šírky

- Pracuje vo fázach:
 - v každej fáze navštívime **nenavštívených susedov**, tých vrcholov, ktoré boli po prvý krát navštívené v predchádzajúcej fáze



Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime



Prehľadávanie do šírky - BFS

- Prehľadávanie do šírky
 - **Breadth-first search (BFS)**
- Použijeme rad
 - obsahuje len navštívené vrcholy, ktorých susedov sme ešte z tohto vrcholu navštívili
- Pri použití knižnice *graph.jar*, pomocou *setValue/getXYZValue* vieme uložiť/získať do/z vrcholov a hrán dodatočné údaje



Prehľadávanie do šírky - BFS

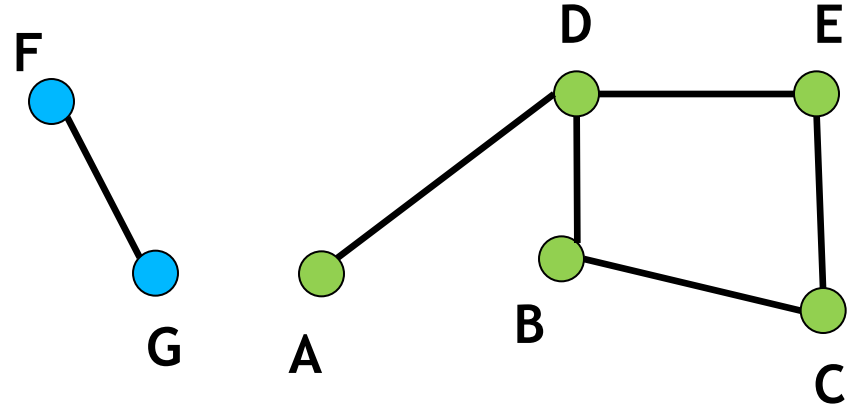
```
public static Map<Vertex, Boolean> bfs(Graph g, Vertex start) {  
    Map<Vertex, Boolean> navstiveny = new HashMap<Vertex, Boolean>();  
    for (Vertex v : g.getVertices())  
        navstiveny.put(v, false);  
  
    Queue<Vertex> rad = new LinkedList<Vertex>();  
    navstiveny.put(start, true); rad.offer(start);  
  
    while (!rad.isEmpty()) {  
        Vertex v = rad.poll();  
        for (Vertex sused : v.getOutNeighbours())  
            if (!navstiveny.get(sused)) {  
                navstiveny.put(sused, true);  
                rad.offer(sused);  
            }  
    }  
  
    return navstiveny;  
}
```

Navštívime
nenavštívených susedov a
pridáme ich do radu, aby
sme nezabudli navštívit aj
ich susedov.



Prehľadávanie do šírky - BFS

- BFS spustený z A , B , C , D alebo E nikdy nenavštívi vrchol F a G
 - graf nie je súvislý



dva komponenty súvislosti

- **Komponent grafu**
 - maximálny súvislý podgraf
 - množina vrcholov, ktoré sú navzájom prepojené cestami v grafe
 - na hľadanie komponentov ide použiť napr. BFS (ale aj iné algoritmy na testovanie súvislosti)



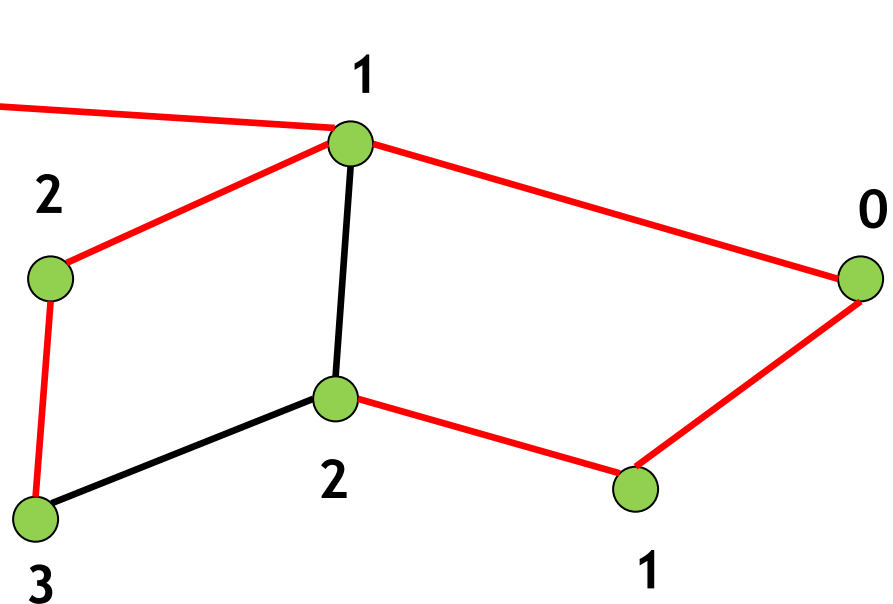
Prehľadávanie do šírky - BFS

- Graf je **súvislý**, ak BFS prechod **navštívi všetky vrcholy** grafu

- BFS vieme použiť na výpočet **najkratšej cesty** z daného vrcholu do všetkých ostatných

- Červené hrany (objaviteľské hrany) vytvárajú **súvislý podgraf bez cyklov**:

- minimálna množina hrán, ktoré musíme zachovať, aby graf ešte ostal súvislý





Kostra grafu

- **Kostra grafu** je taká **podmnožina T hrán** grafu G , že platí:
 1. Medzi každými 2 vrcholmi grafu **existuje cesta** využívajúca len hrany kostry T
 2. **Odobratím** ľubovoľnej hrany kostry už vlastnosť 1 **nebude platiť**
- Kostra grafu – minimálna množina hrán grafu, ktorá „drží graf pokope“
 - graf môže mať veľa kostier
- Objaviteľské hrany v BFS definujú **BFS kostru**





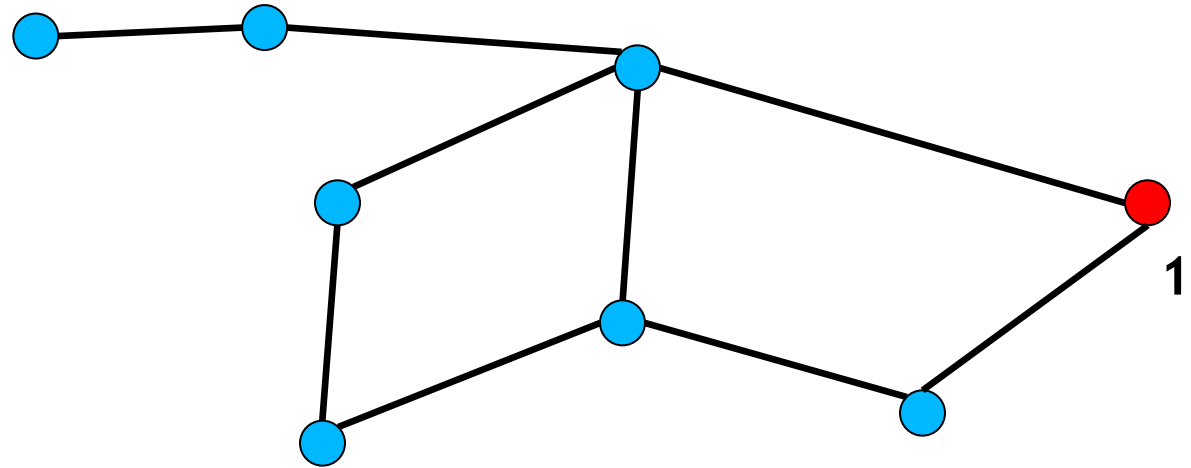
Aplikácie BFS

- **Sociálna sieť** - nájdenie sociálnej vzdialenosti medzi osobami
 - fenomén malého sveta - six degrees of separation
- **Sieť dopravných spojení** - nájdenie spojenia s minimálnym počtom prestupov
 - vrchol - stanica, hrana - dopravné spojenie
- **Komunikáčna sieť**
 - nájdenie minimálnej množiny spojení na upgrade, aby medzi každými 2 uzlami bolo spojenie po upgradovaných (optických) linkách



Prehľadávanie do hĺbky

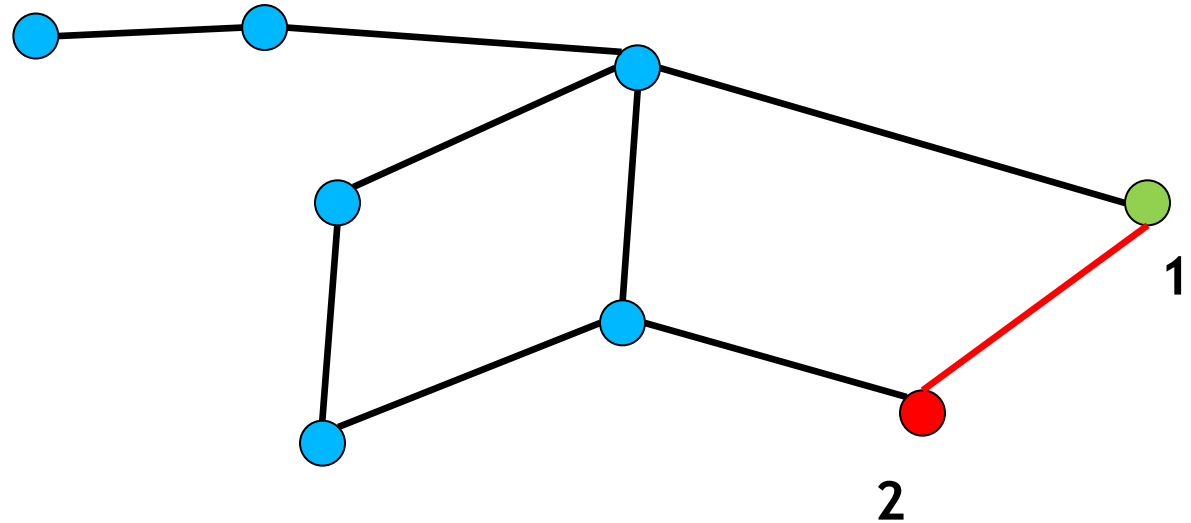
- Prehľadávanie do hĺbky
 - **depth-first search DFS**
- **Stratégia návštevy vrcholu:**
 - **označ** vrchol ako navštívený
 - postupne **navštív** všetky jeho susedné nenavštívené vrcholy
 - **vrát' sa** do vrcholu, z ktorého si sem prišiel
- Narozdiel od BFS („vlna šíriaca sa v grafe“) je DFS predstaviteľné ako „**putovanie v grafe**“



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



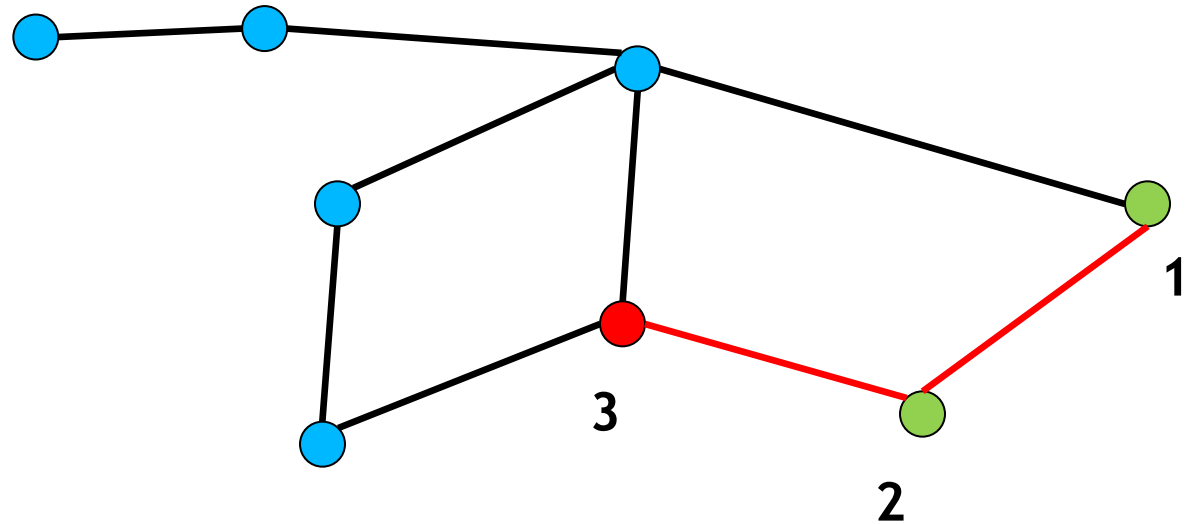
DFS



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



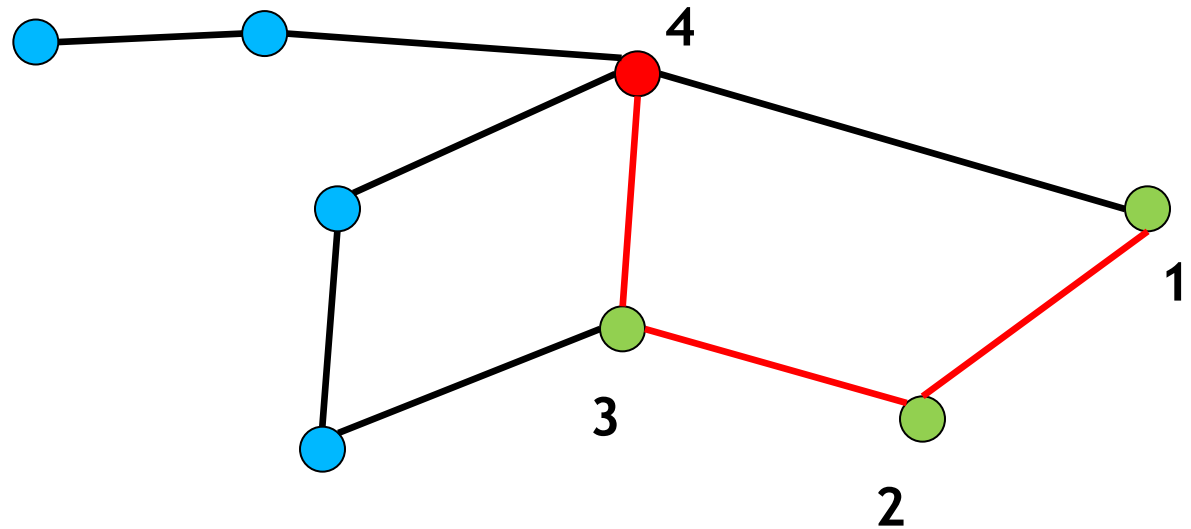
DFS



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



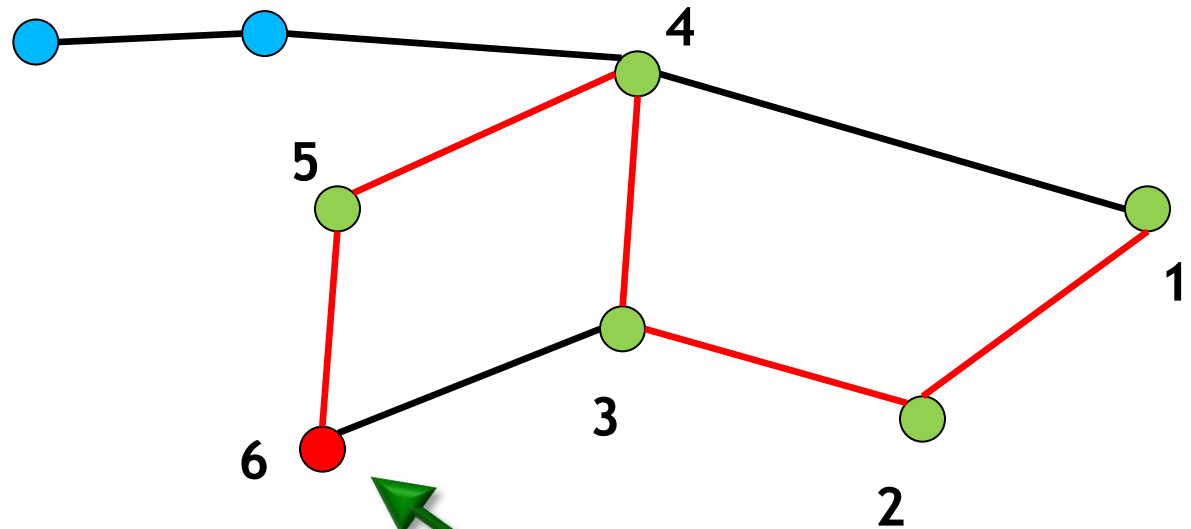
DFS



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)

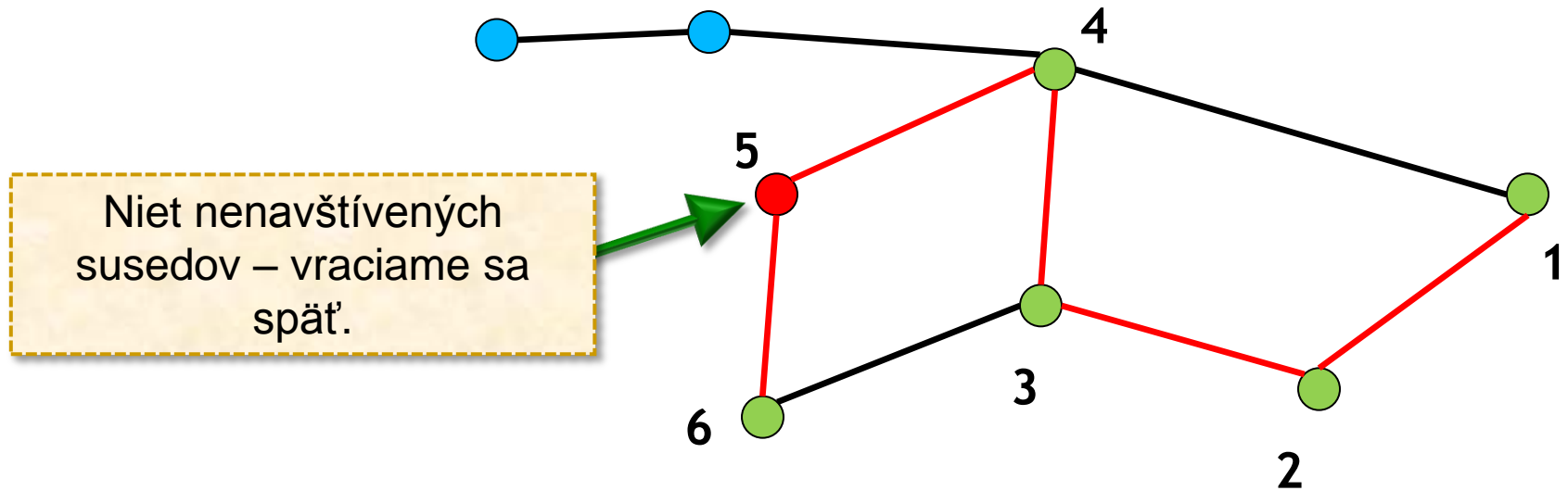


DFS

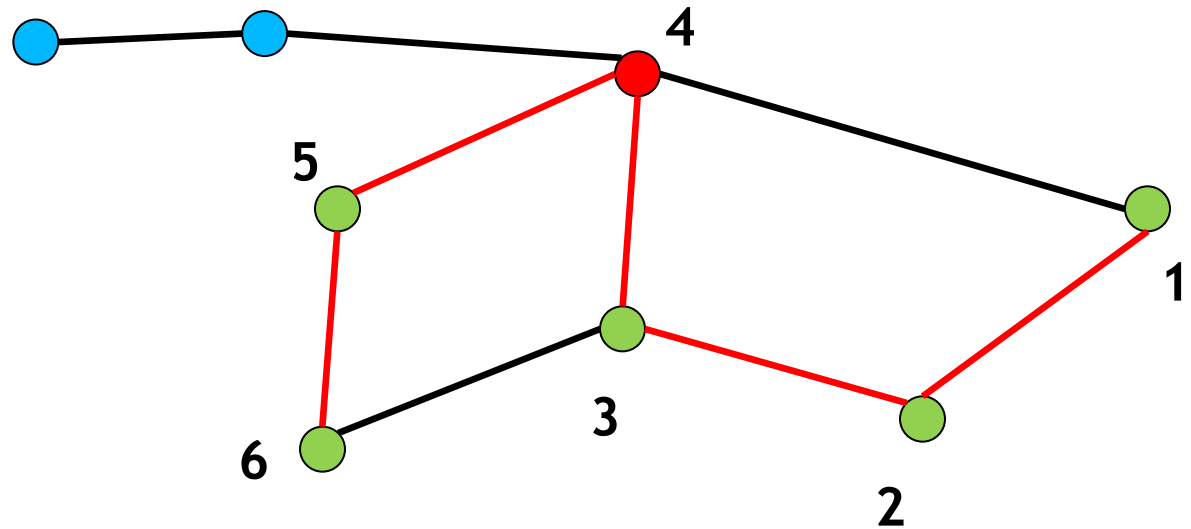


- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)

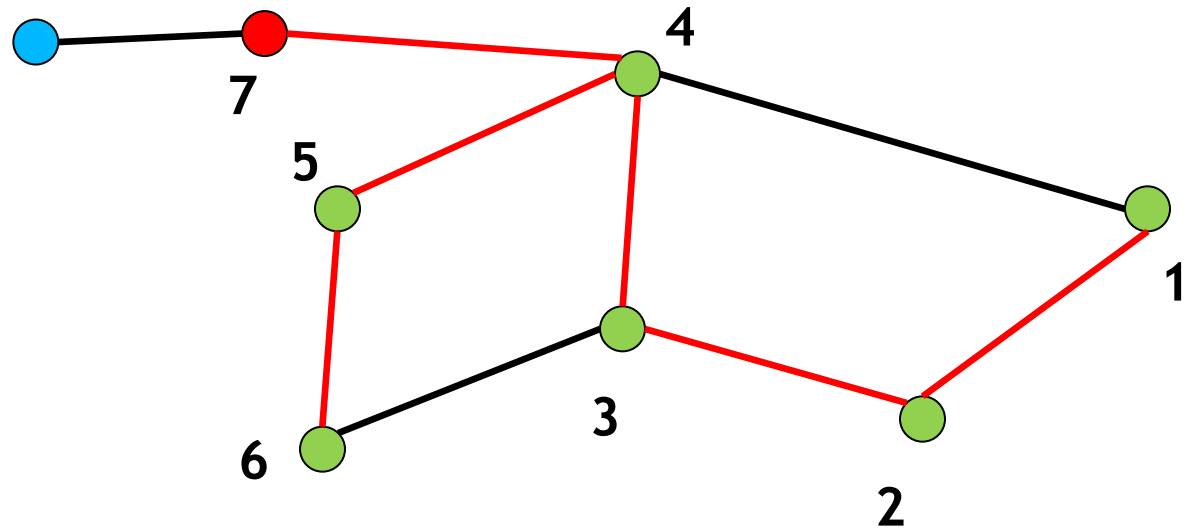
Niet nenavštívených susedov – vraciame sa späť.



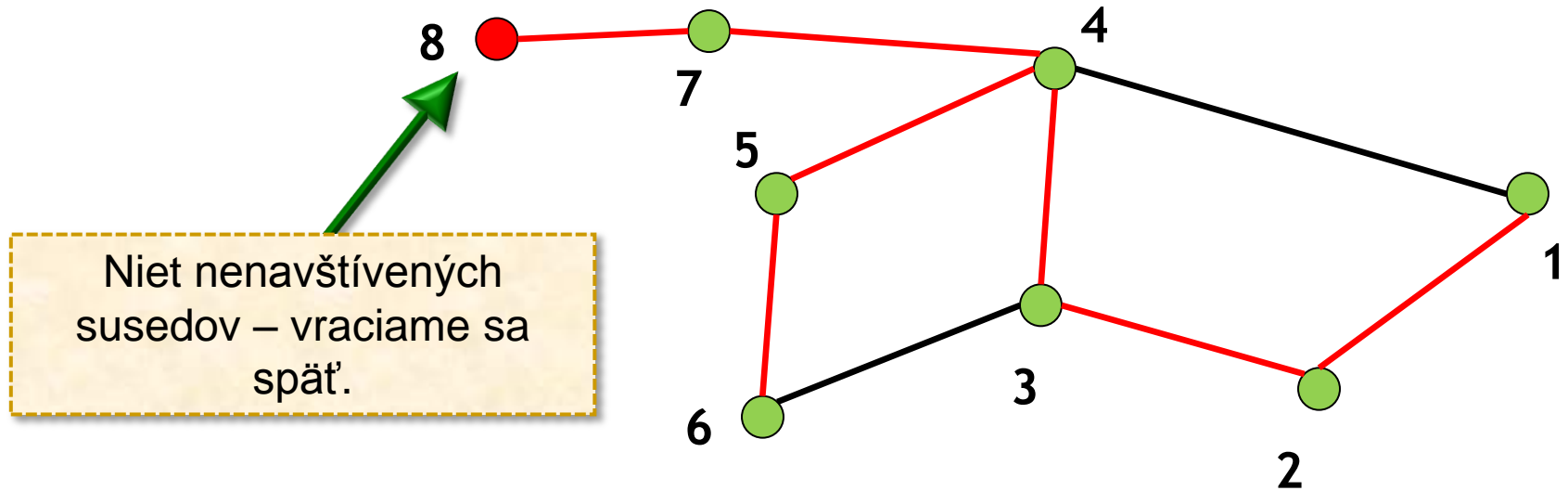
- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



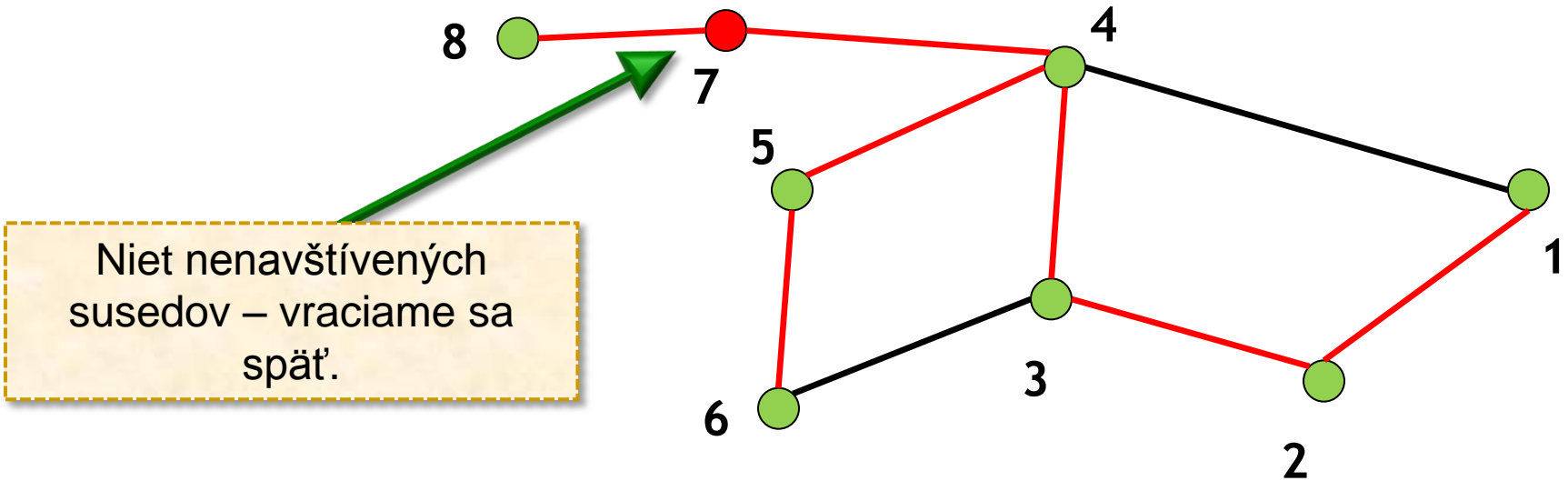
- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



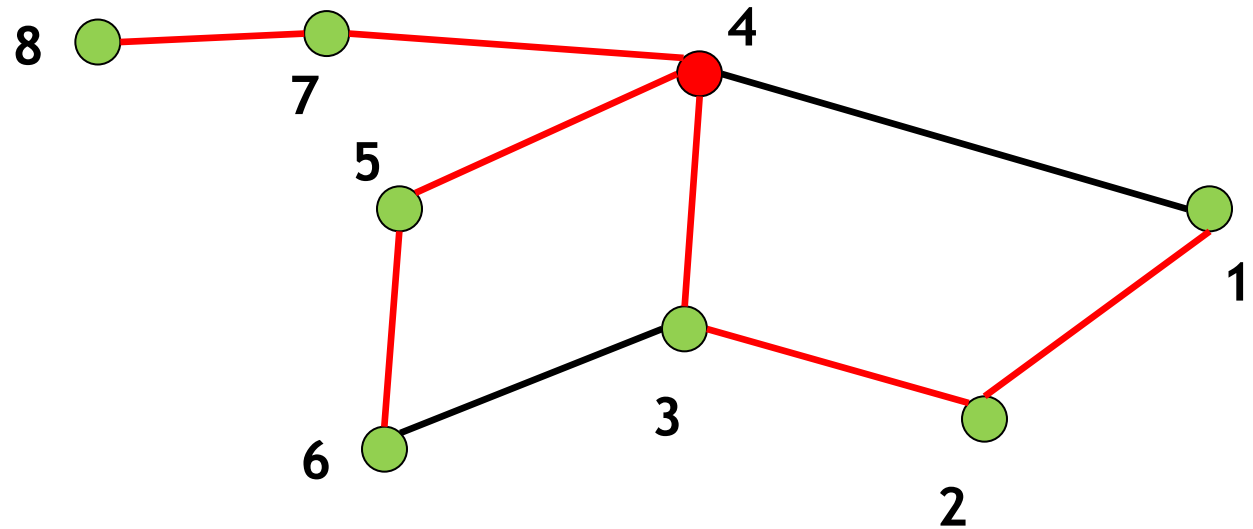
- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



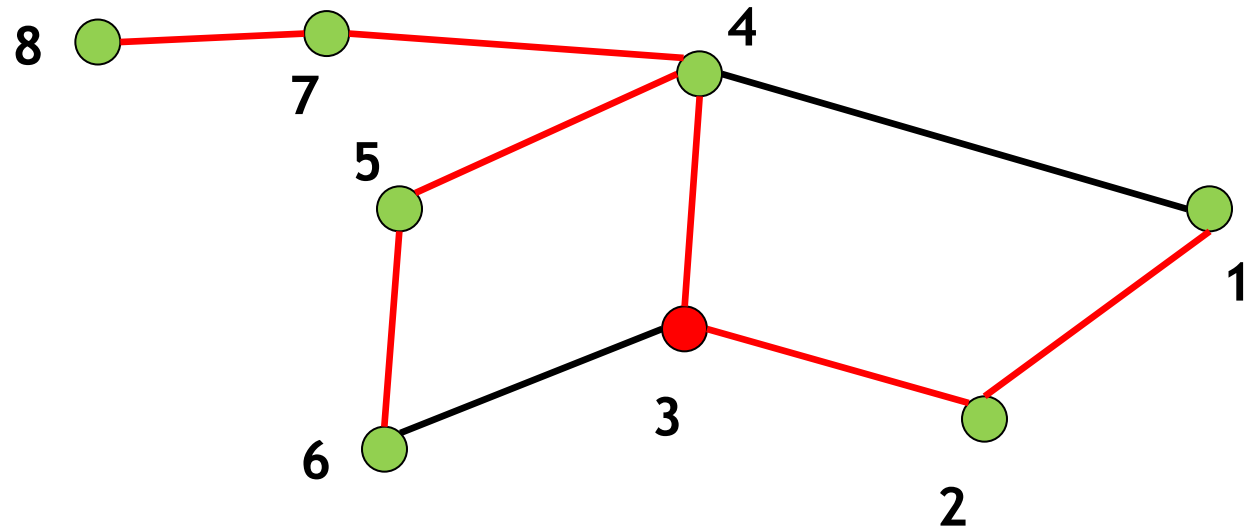
DFS



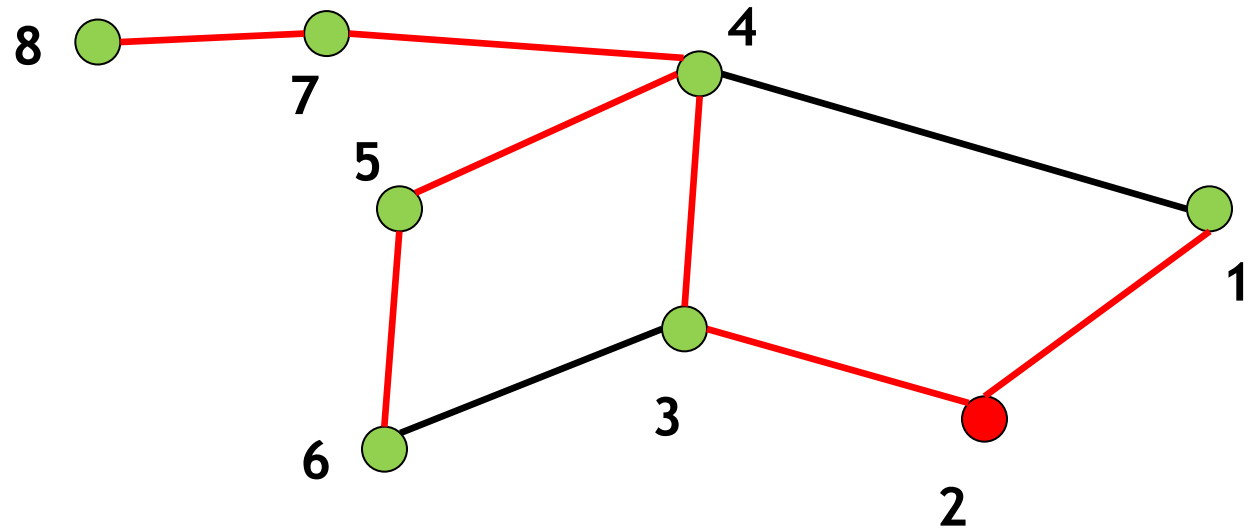
- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



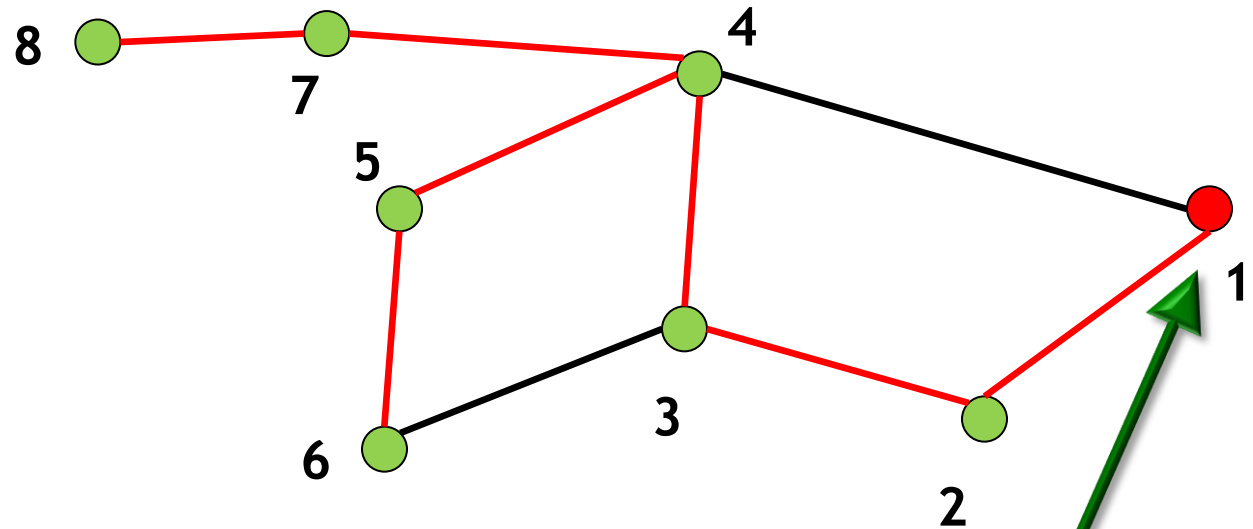
- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)



DFS



- Červené hrany sú hrany, po ktorých vrchol prvý krát navštívime (objaviteľské hrany)

Niet nenavštívených susedov a sme tam, kde sme začali: **končíme!**



Rekurzívne DFS

```

public void dfs(Vertex v, Map<Vertex, Boolean> navstiveny) {
    navstiveny.put(v, true);
    for (Vertex sused : v.getOutNeighbours())
        if (!navstiveny.get(sused))
            dfs(sused, navstiveny);
}

```

Poznačíme návštevu

Postupne navštívime
nenavštívených
susedov

```

public Map<Vertex, Boolean> dfsRekurzivne(Graph g, Vertex start) {
    Map<Vertex, Boolean> navstiveny =
        new HashMap<Vertex, Boolean>();
    for (Vertex v : g.getVertices())
        navstiveny.put(v, false);
    dfs(start, navstiveny);
    return navstiveny;
}

```



Nerekurzívne DFS

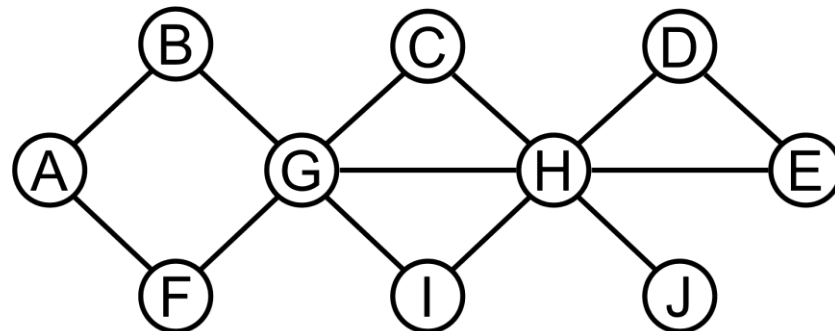
```
public static Map<Vertex, Boolean> dfsNerekurzivne(Graph g, Vertex start) {  
    Map<Vertex, Boolean> navstiveny = new HashMap<Vertex, Boolean>();  
    for (Vertex v : g.getVertices())  
        navstiveny.put(v, false);  
  
    Stack<Vertex> zasobnik = new Stack<Vertex>();  
    navstiveny.put(start, true); zasobnik.push(start);  
  
    while (!zasobnik.isEmpty()) {  
        Vertex v = zasobnik.pop();  
        for (Vertex sused : v.getOutNeighbours())  
            if (!navstiveny.get(sused)) {  
                navstiveny.put(sused, true);  
                zasobnik.push(sused);  
            }  
        }  
    }  
    return navstiveny;  
}
```



Vlastnosti DFS

- Objaviteľské hrany (hrany, po ktorých sme po prvý krát navštívili vrchol) v DFS definujú **DFS kostru**
- DFS kostra - má isté užitočné „grafárske“ vlastnosti týkajúce sa nekostrových
 - napr. efektívne hľadanie **artikulácií** (vrcholov, po ktorých odstránení sa graf „rozpadne“ = nebude súvislý) v grafe

Odstránením *G* alebo *H* sa graf rozpadne.





BFS vs. DFS

- Časová náročnosť oboch algoritmov je pri vhodnej reprezentácii grafu $O(n + m)$, kde n je počet vrcholov a m je počet hrán
 - pri použití matice susednosti majú oba algoritmy časovú zložitost' $O(n^2)$
 - každý z n vrcholov navštívime len raz, v každom vrchole preskúmame nanajvyš n jeho susedov
- DFS vo všeobecnosti vyžaduje menej pamäte (pri vhodnej implementácii)
- BFS navyše hľadá aj najkratšie cesty



Grafová terminológia

● Excentricita vrcholu:

- vzdialenosť od neho k najvzdialenejšieho vrcholu
 - algoritmus: BFS prehľadávanie

● Centrum grafu:

- vrchol s minimálnou excentricitou

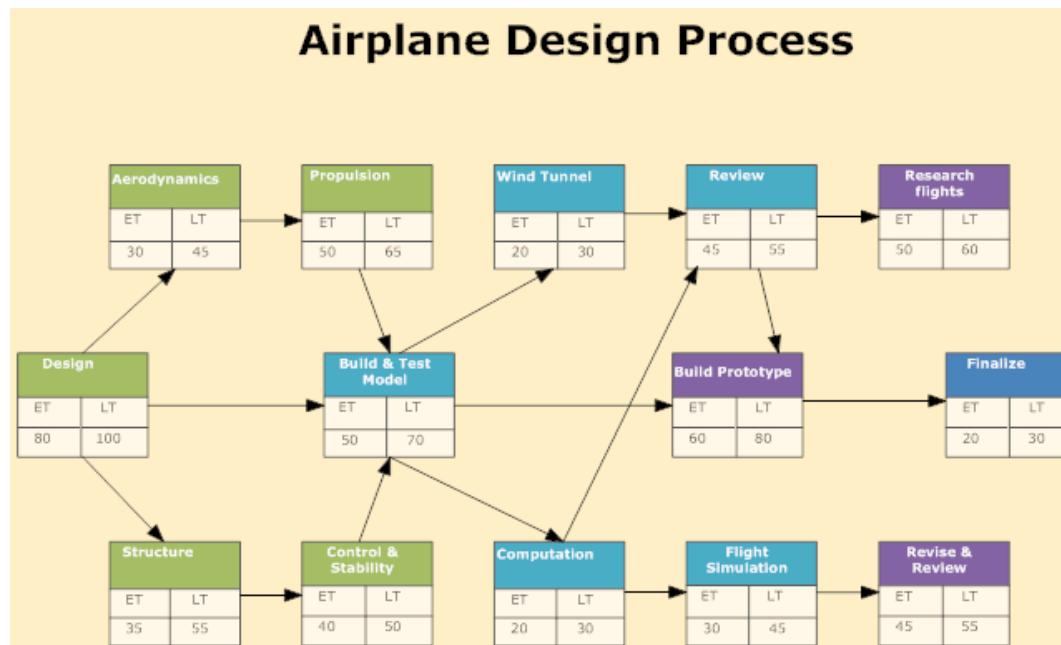
● Priemer grafu:

- maximálna vzdialenosť medzi 2 vrcholmi grafu
 - priemer grafu = maximálna excentricita v grafe (spomedzi všetkých vrcholov grafu)



Topologické triedenie

a grafy, kde hrany majú orientáciu





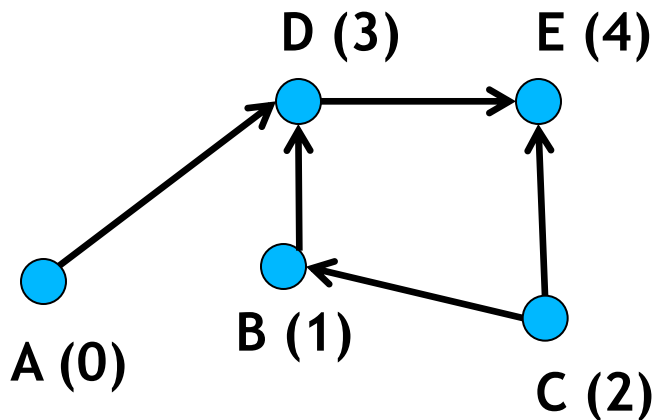
Orientovaný graf

- Niekedy vzťahy (relácie) **nie sú symetrické**:
 - Osoba A má rada osobu B, ale osoba B nemá rada A
- Nesymetrická relácia „**začať pred**“
 - PAZ1a musí byť absolvované pred PAZ1b
- Nesymetrické vzťahy (relácie) modelujeme **orientovaným grafom** (directed graph)
 - orientovanú hranu voláme **šíp**
 - hrany nie sú čiary, ale šípky



Orientovaný graf v programe

Matica susednosti:



```
boolean[][] graf =
    new boolean[n][n]
```

	A	B	C	D	E
A	F	F	F	T	F
B	F	F	F	T	F
C	F	T	F	F	T
D	F	F	F	F	T
E	F	F	F	F	F

graf[u][v] = z u do v ide hrana v grafe



Algoritmy pre orient. grafy

- DFS a BFS prechody vieme použiť na **nájdenie vrcholov dostupných** zo zadaného vrcholu po orientovaných cestách
 - orientovaná cesta = cesta určená šípami
- **Topologické usporiadanie:**
 - nájsť takú postupnosť vrcholov, aby ak z u do v je v grafe orientovaná hrana, tak vrchol u je v postupnosti pred vrcholom v

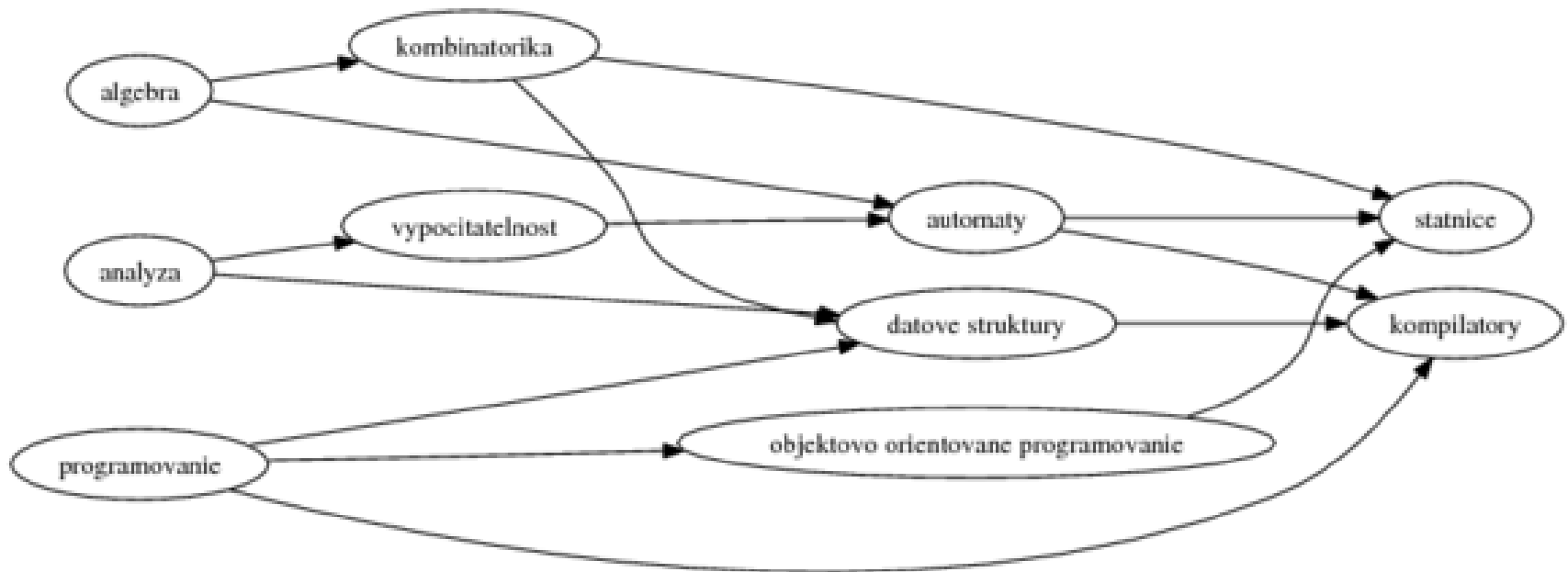


Topologické usporiadanie

- **Motivácia:**
 - množina činností
 - vieme, čo musí **byť spravené skôr**:
 - **A** musí byť spravená pred **B**, pretože činnosť **B** potrebuje použiť výsledok činnosti **A**:
 - košeľa musí byť oblečená skôr ako kabát
 - PAZ1a musí byť spravený pred PAZ1b
 - vodoinštaláciu môžem t'ahať, až keď sú hotové múry
- **Problém:** nájsť takú postupnosť vykonávania činností, aby boli splnené všetky podmienky



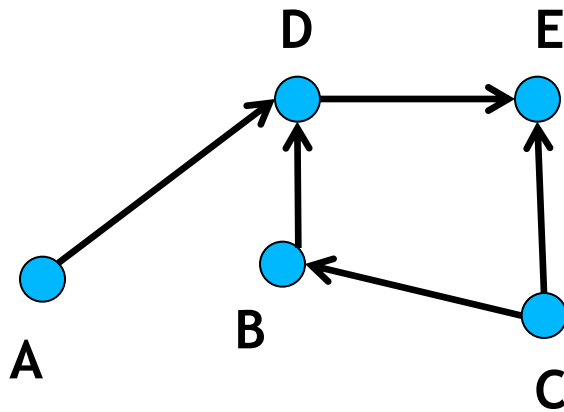
Topologické usporiadanie





Topologické usporiadanie

- Topologické usporiadanie:
 - nájsť takú postupnosť vrcholov, aby ak z u do v je v grafe orientovaná hrana, tak u je v postupnosti pred vrcholom v



Orientovaný graf môže mať **viacero** topologických usporiadaní.

Topologicky usporiadané vrcholy:

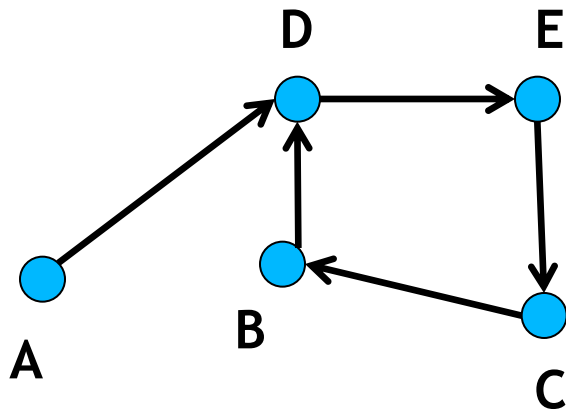
A, C, B, D, E

C, B, A, D, E



Topologické usporiadanie

- Topologické usporiadanie:
 - nájsť takú postupnosť vrcholov, aby ak z u do v je v grafe orientovaná hrana, tak u je v postupnosti pred vrcholom v



Existujú grafy, ktoré nemožno topologicky usporiadať.



TopSort - algoritmus

● Idea:

● kým sa dá, opakuj:

- vyber ľubovoľný vrchol, do ktorého **nevchádza žiadna** orientovaná **hrana**, „vypíš ho“ a **odstráň ho**

- ak sa skončilo s prázdny grafom, máme topologické usporiadanie vrcholov

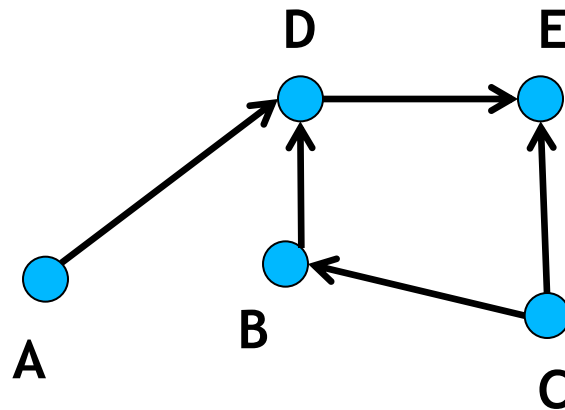
- ak sa skočilo s neprázdny grafom, graf nemá topologické usporiadanie

● Prečo to funguje?

- Dôkaz indukciou na veľkosť grafu (cvičenia)



TopSort – vizualizácia



A C B D E



Topologické usporiadanie

- Užitočný algoritmus pri manažovaní projektov
 - metóda PERT
- Časová zložitosť $O(n^2)$:
 - n odstraňovaných vrcholov
 - nájdanie vrcholu bez predchodcu $O(n)$
 - odstránenie vrcholu a s ním incidentných hrán $O(n)$
 - pri vhodnej reprezentácii grafu s n vrcholmi a m hranami $O(n+m)$
- Možno použiť na **nájdanie** orientovaných **cyklov** v grafe (viac na cvičeniach)



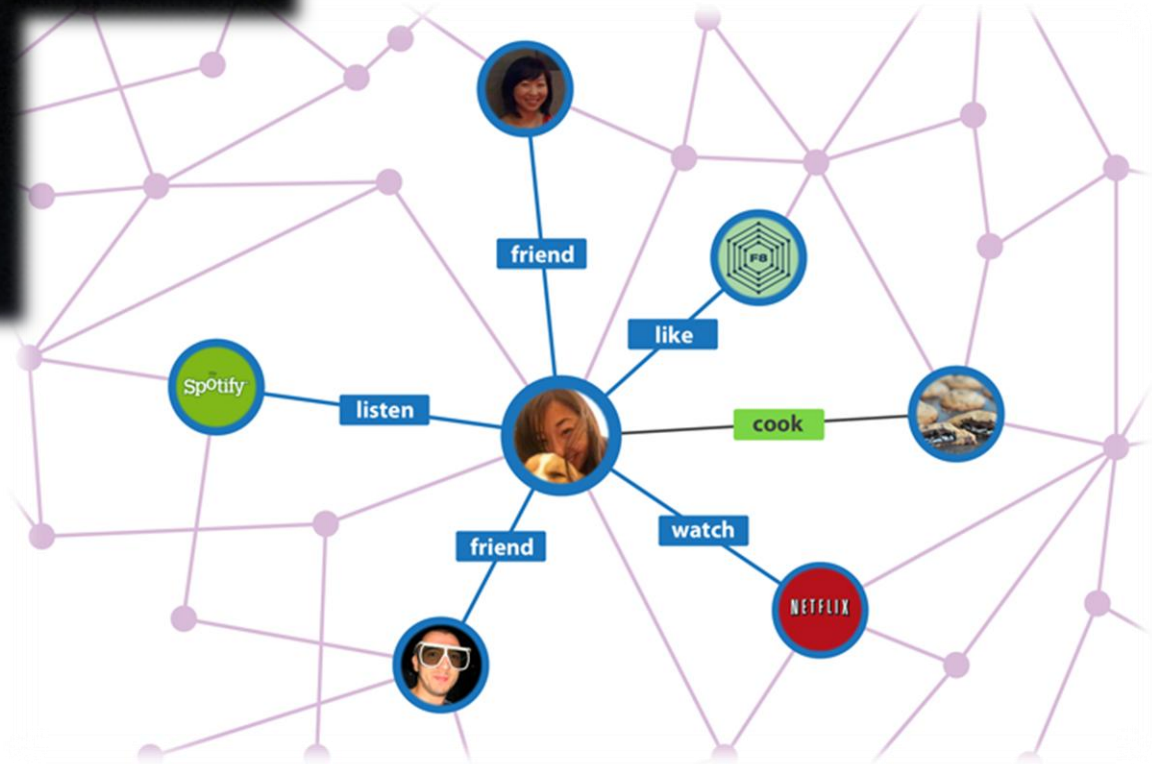
Sumarizácia

- **Graf**
 - prostriedok na zachytenie vzťahov medzi objektami
- **Grafy a grafové algoritmy**
 - neuveriteľné množstvo aplikácií a možností použitia
- **Prehľadávanie grafov**
 - overenie súvislosti, najkratšie cesty, kostry, ...
- **Topologické triedenie**
 - usporiadanie vrcholov orientovaného grafu, hľadanie orientovaných cyklov, ...



„Graf náš každodenný“

Open Graph A new class of apps





ak nie sú otázky...

Ďakujem za pozornosť!

