Hledáme efektivní řešení úloh na grafu

Mějme dán graf $G = (V, E)$, chceme algoritmicky vyřešit následující úlohy:

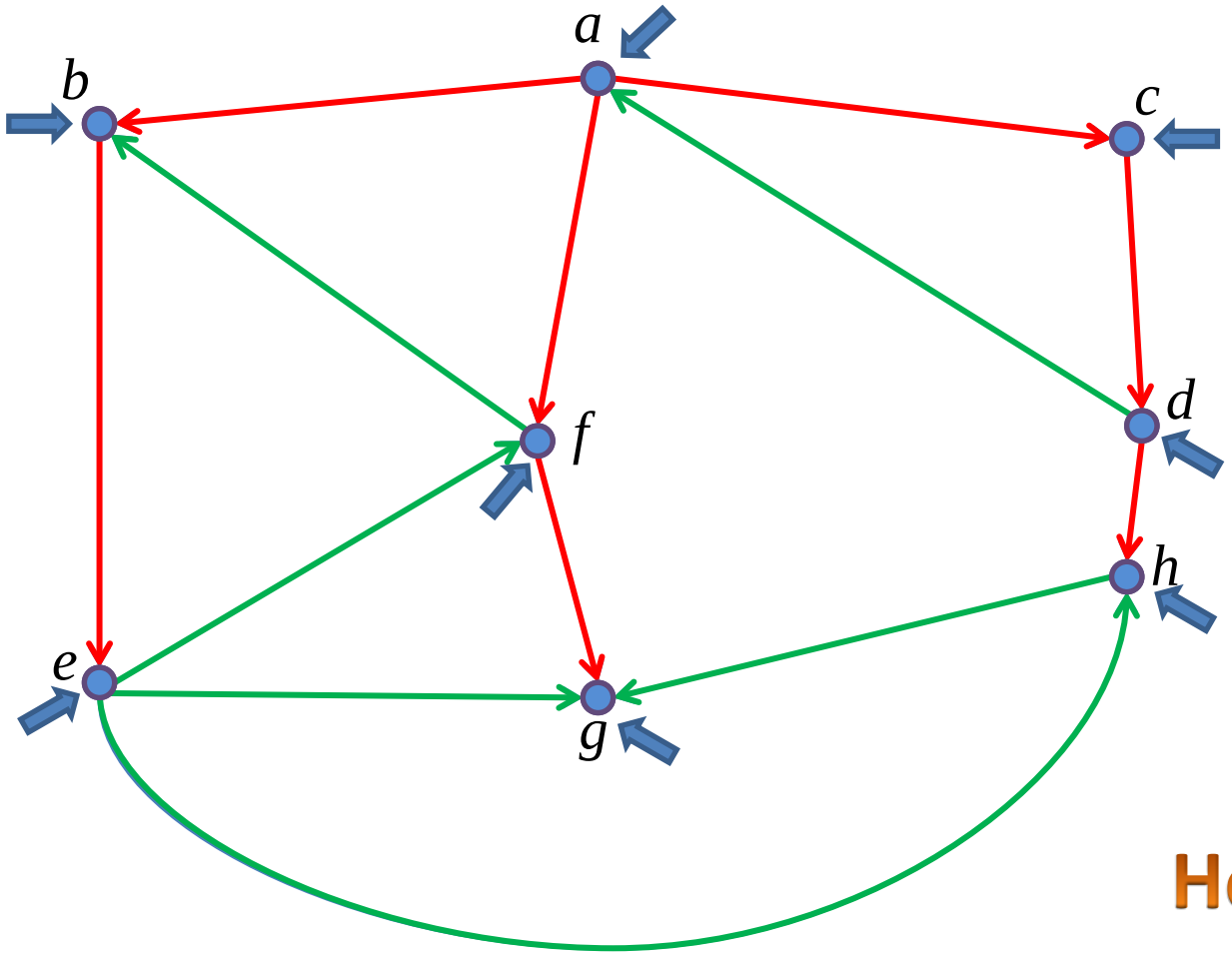
- Je daný vrchol t dosažitelný z vrcholu s? Pokud ano, jaká nejkratší cesta tyto vrcholy spojuje?
- Jaké má graf komponenty souvislosti?
- Obsahuje graf cyklus? Pokud ne, jak topologicky setřídíme vrcholy?
- Jak nalezneme silné komponenty?
- Jak nakreslíme graf jedním tahem?

Systematické průchody grafem

1. Prohledávání do šířky – BFS (breadth-first search)
2. Prohledávání do hloubky – DFS (depth-first search)

- Zobecnění průchodu stromem na obecný graf

Průchod do šířky – ukázka



Hotovo

Průchod do šířky

```
BFS(G,s): // G=(V,E)
    // inicializace, používáme pole D, P a frontu F
    for each v in V do
        D[v]:=-1; P[v]:=NULL;
    done
    D[s]:=0; F->push(s);
    while not F->empty() do
        v:=F->pop();
        for each soused w vrcholu v do
            if D[w]=-1 then
                D[w]:=D[v]+1; P[w]=v; F->push(w);
            endif
        done
    done
```

Průchod do šířky

Pozorování:

• v průběhu algoritmu obsahuje fronta F jen vrcholy, které mají D[*v*] rovno *d* nebo *d+1*, pro nějaké *d*

• D[*v*] obsahuje délku nejkratší cesty z *s* do *v*

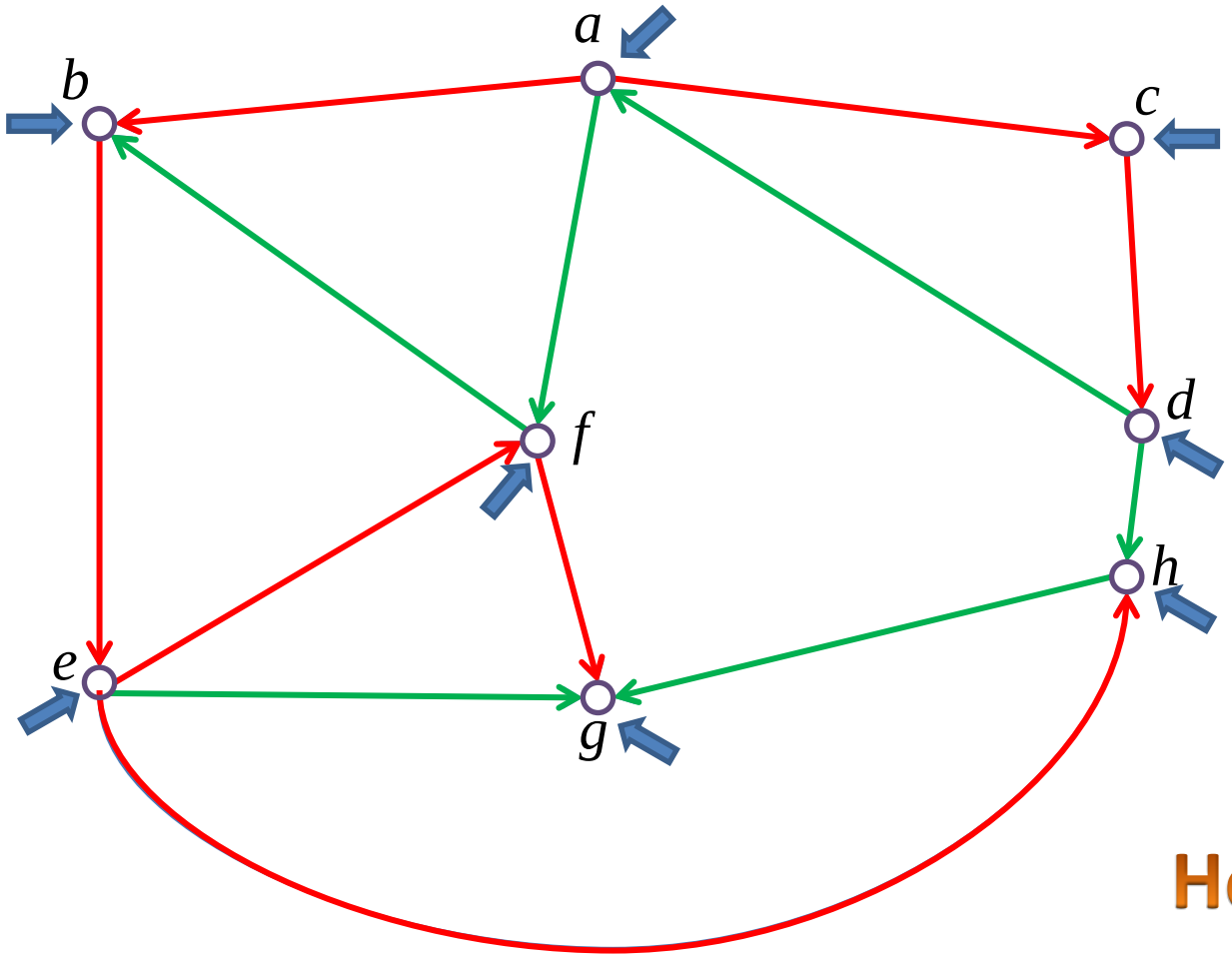
Časová složitost $O(|V|+|E|)$.

• odpovídá velikosti reprezentace G spojovým seznamem

Počítání komponent

```
SpocitejKomponenty(G): // G=(V,E) - neorient. graf
    pocet_komp:=0;
    // modifikuj BFS, inicializuj D a P vně BFS2
    for each v in V do D[v]:=-1; P[v]:=NULL; done
    for each v in V do
        if D[v]=-1 then
            BFS2(G,v);
            pocet_komp++;
        endif
    done
    return pocet_komp;
```

Průchod do hloubky – ukázka



Hotovo

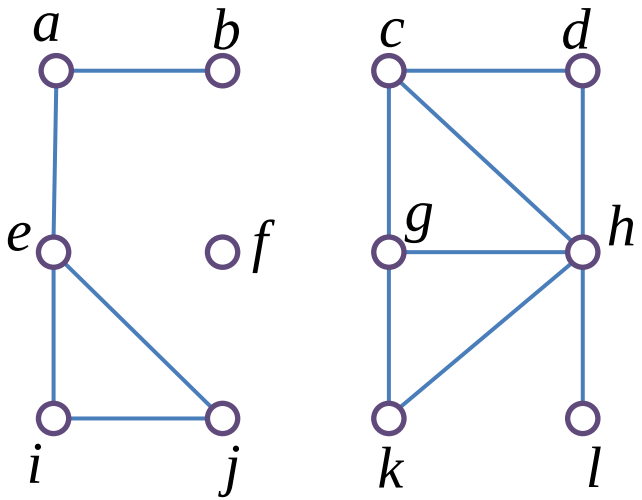
Průchod do hloubky

```
DFS(G): // G=(V,E)
    for each v in V do
        stav[v]:=FRESH;
        P[v]:=NULL;
    done
    time:=0;
    for each v in V do
        if stav[v]=FRESH then
            DFS_Projdi(s);
        endif
    done
```

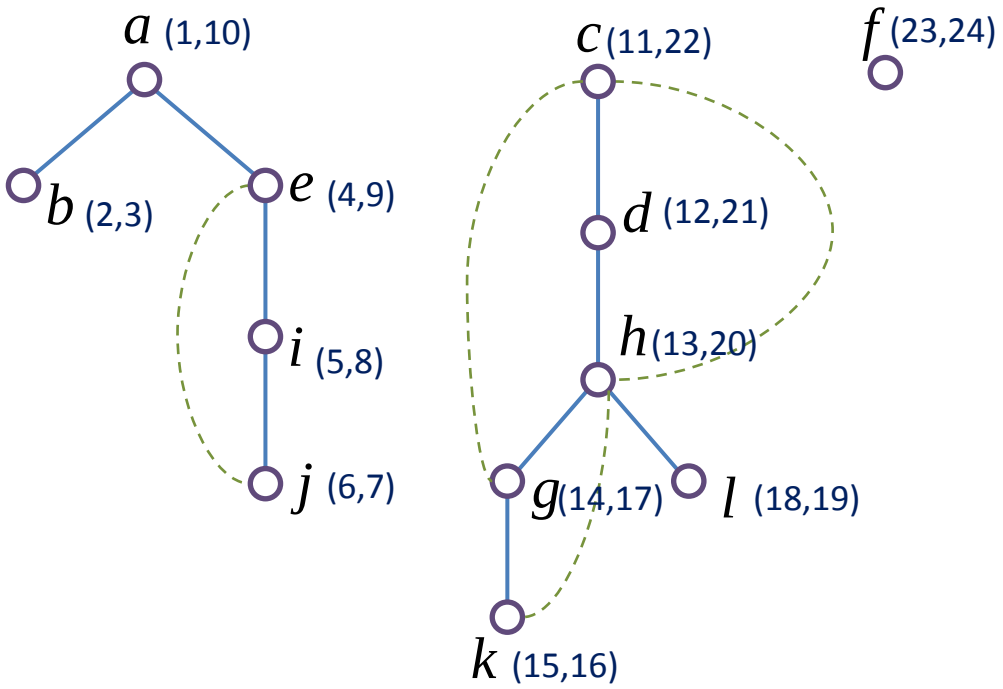
```
DFS_Projdi(v):
    stav[v]:=OPEN;
    time:=time+1;
    in[v]:=time;
    for each w in Sousede[v] do
        if stav[w]=FRESH then
            P[w]:=v;
            DFS_Projdi(w);
        endif
    done
    stav[v]:=CLOSED;
    time:=time+1;
    out[v]:=time;
```

Průchod do hloubky – příklad č. 1

vstupní graf:

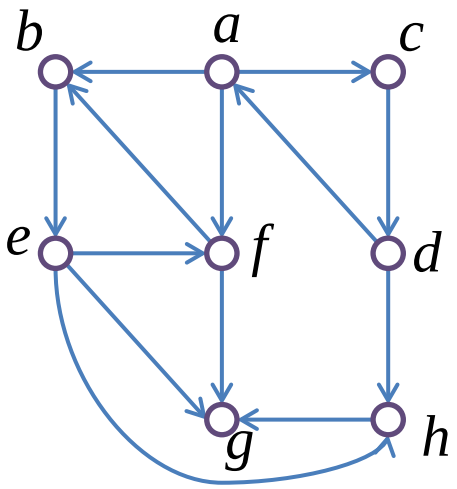


DFS stromy (DFS les):

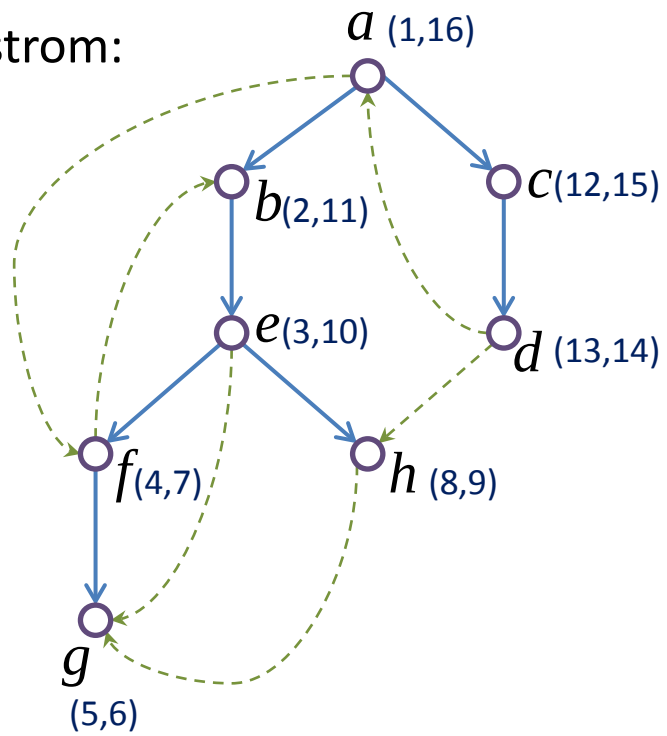


Průchod do hloubky – příklad č. 2

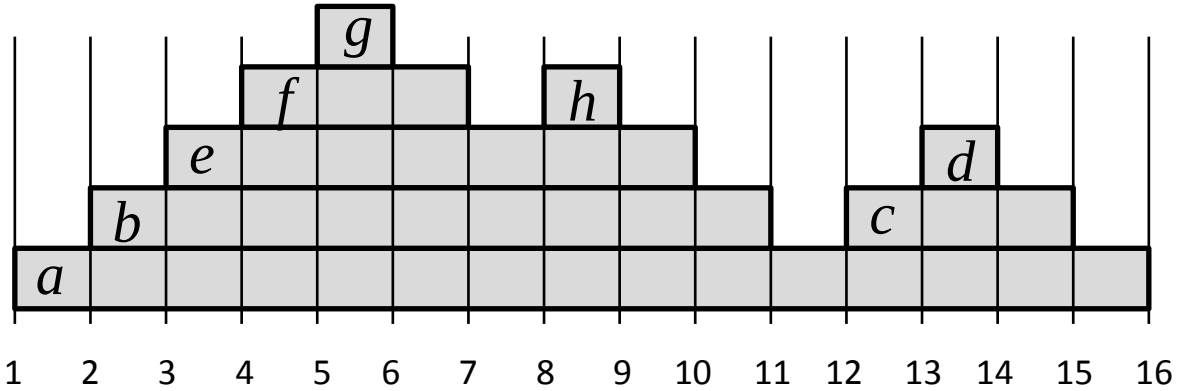
vstupní graf:



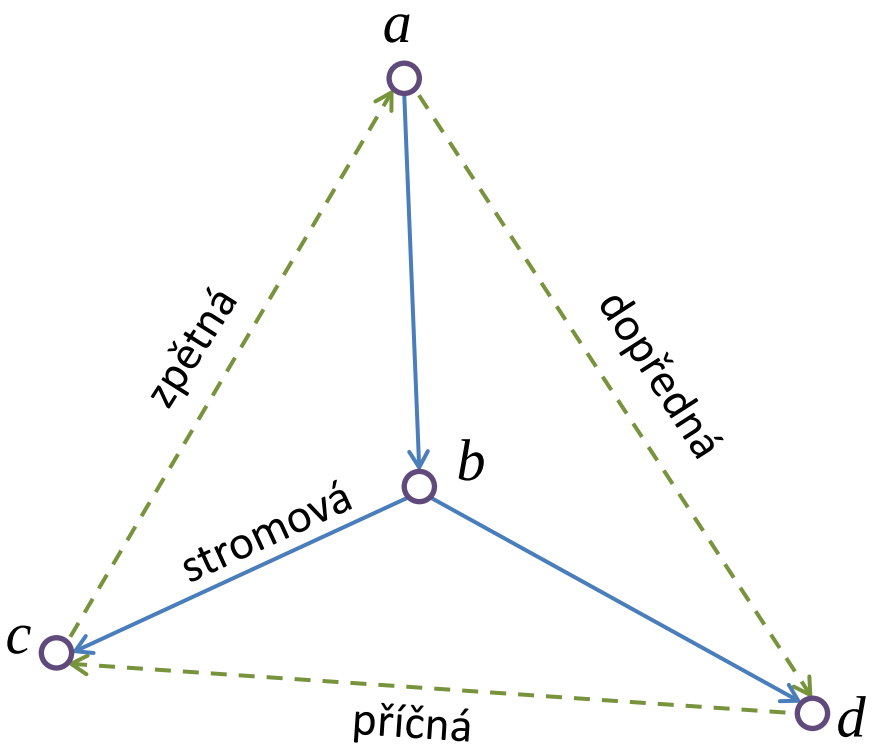
DFS strom:



časová osa:



Typy hran při DFS pro orientovaný graf



Určení typu pro hranu *(u,v)* podle stavu dosaženého vrcholu *v*:

1. *v* je FRESH → stromová
2. *v* je OPEN → zpětná
3. *v* je CLOSED → dopředná nebo příčná

Typy hran při DFS pro orientovaný graf

Pozorování:

1. Hrana *(u,v)* je stromová nebo dopředná právě když

$$(in[u], out[u]) \supset (in[v], out[v])$$

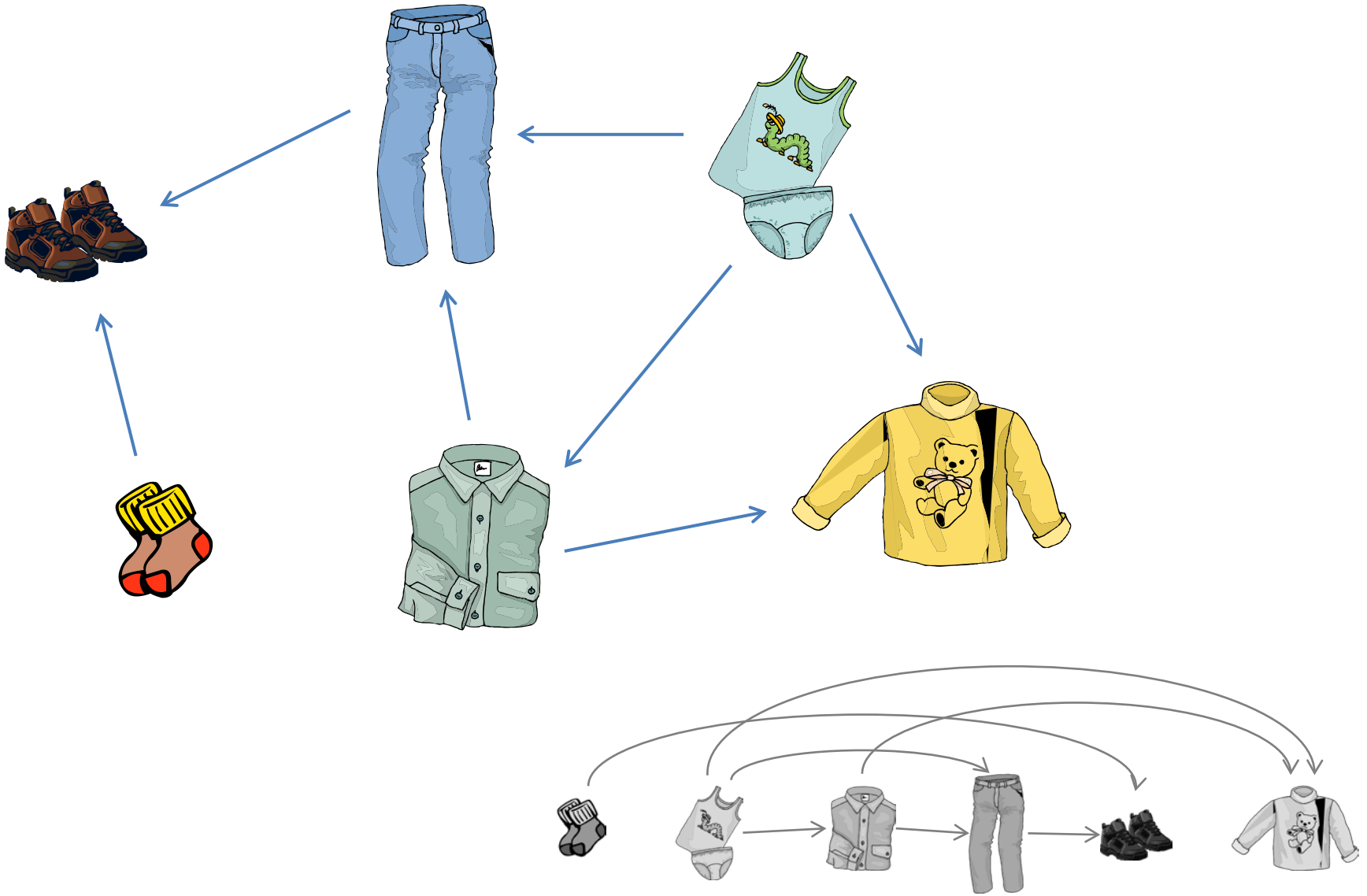
2. Hrana *(u,v)* je zpětná právě když

$$(in[u], out[u]) \subset (in[v], out[v])$$

3. Hrana *(u,v)* je příčná právě když

$$in[v] < out[v] < in[u] < out[u]$$

Topologické uspořádání



Topologické uspořádání

Věta: Orientovaný graf G obsahuje cyklus právě když DFS nalezne zpětnou hranu.

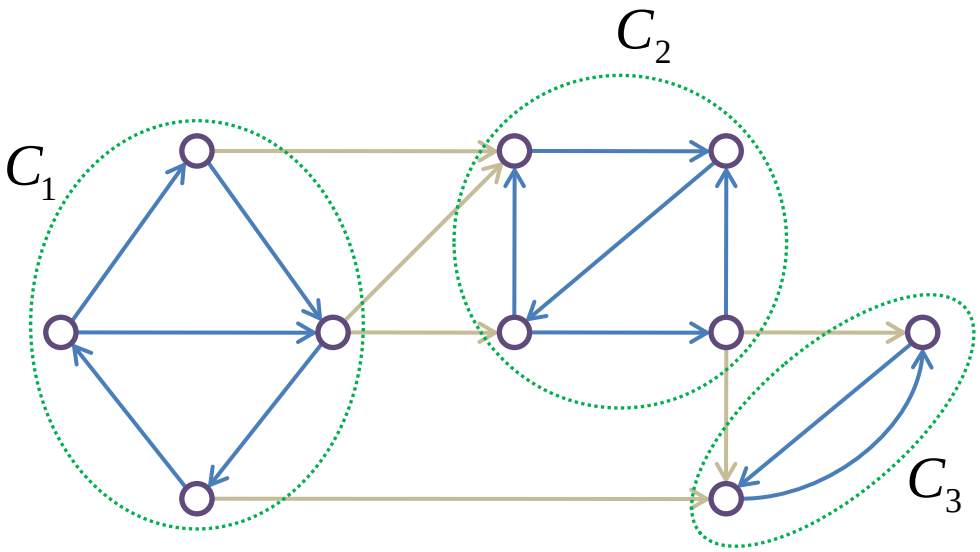
Věta: Uspořádání vrcholů orientovaného acyklického grafu G podle klesajících hodnot *out[.]* je topologické.

Důkaz:

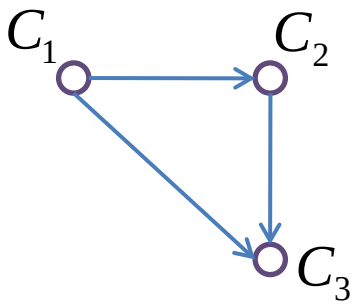
- Je-li (u,v) stromová nebo dopředná, pak $out[u] > out[v]$
- Je-li (u,v) příčná, pak též $out[u] > out[v]$

Tj., orientace hran v uvedeném uspořádání je korektní.

Hledání silných komponent



kondenzace grafu:

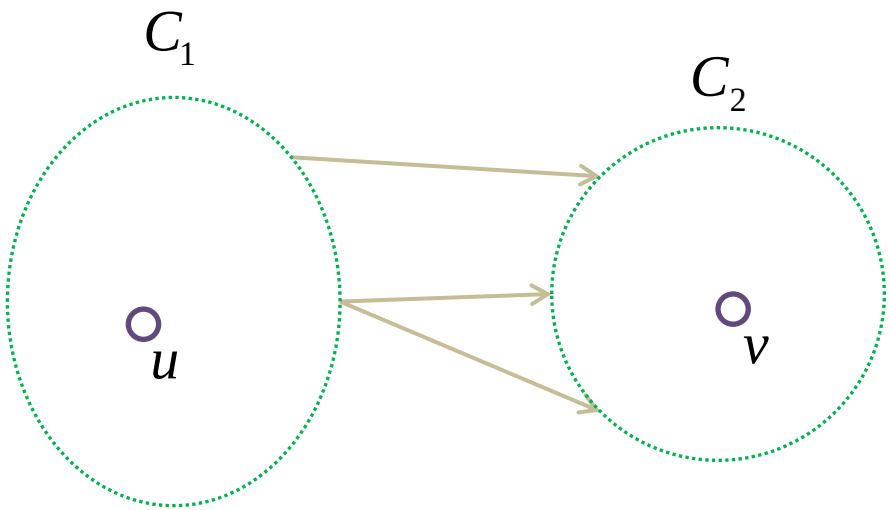


Při použití DFS potřebujeme komponenty projít v pořadí: C_3, C_2, C_1

Obecně, v opačném pořadí vůči nějakému topologickému uspořádání.

Hledání silných komponent

Pozorování:



$$out[u] > out[v]$$

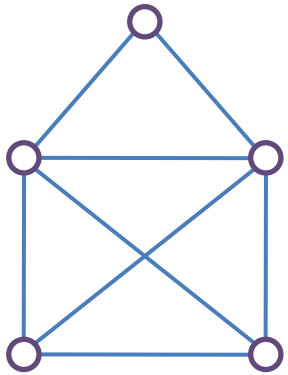
Pro libovolné $u \in C_1$ a $v \in C_2$.

Hledání silných komponent - algoritmus

Vstup: orientovaný graf G

1. vytvoř G^{-}, graf opačně orientovaný ke G
2. proveď DFS na G^{-} a ulož posloupnost $out[.]$ v sestupném pořadí
3. proveď DFS na G, vrcholy procházej v pořadí podle klesajících hodnot $out[.]$

Kreslení grafu jedním tahem



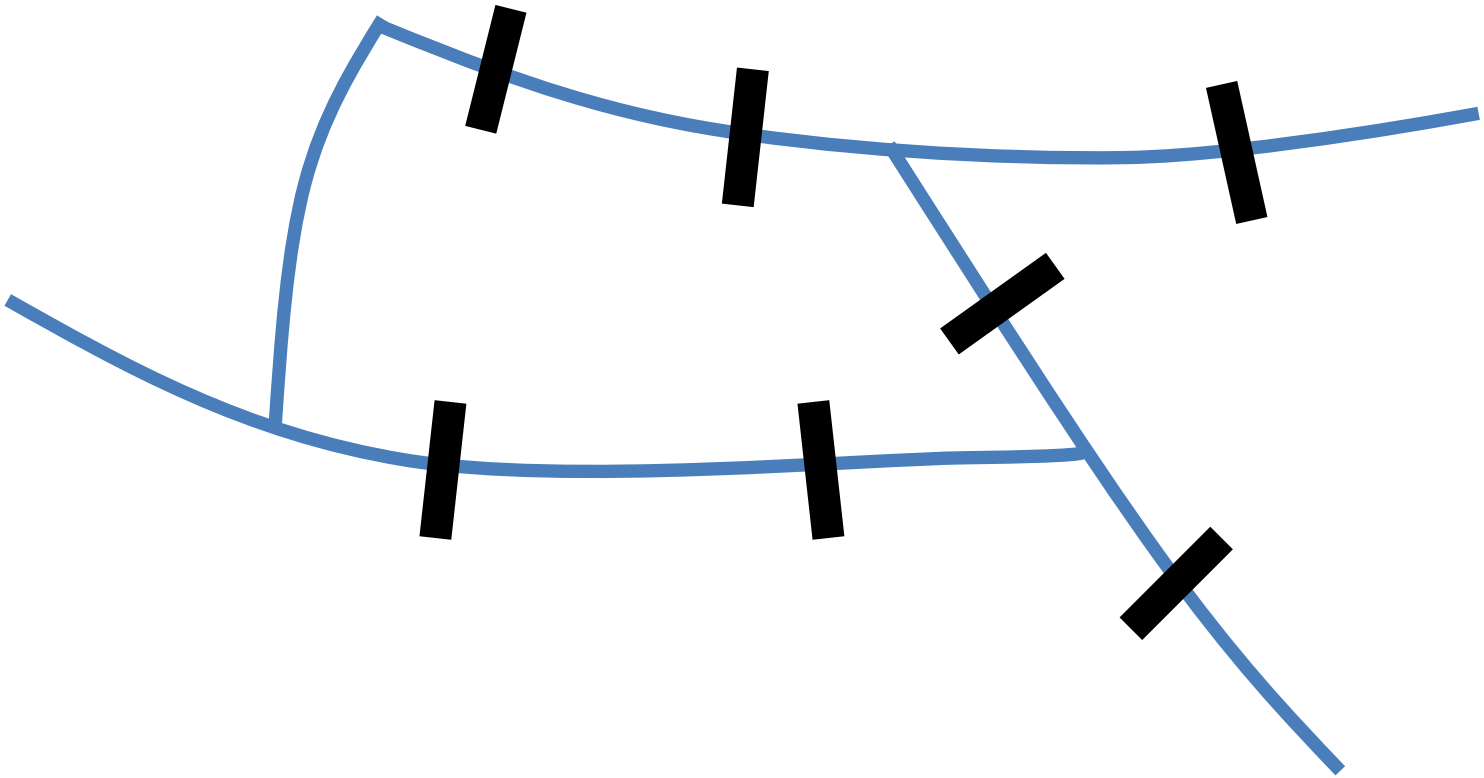
Def.: Sled nazýváme tahem, pokud se v něm neopakují hrany (vrcholy se opakovat mohou).

Def.: Eulerovský tah je každý tah, který obsahuje všechny hrany.

Def.: Graf *G* nazýváme Eulerovský, pokud obsahuje alespoň jeden uzavřený Eulerovský tah.

Věta: Souvislý graf *G* je Eulerovský právě když všechny jeho vrcholy mají sudý stupeň.

Sedm mostů města Královce



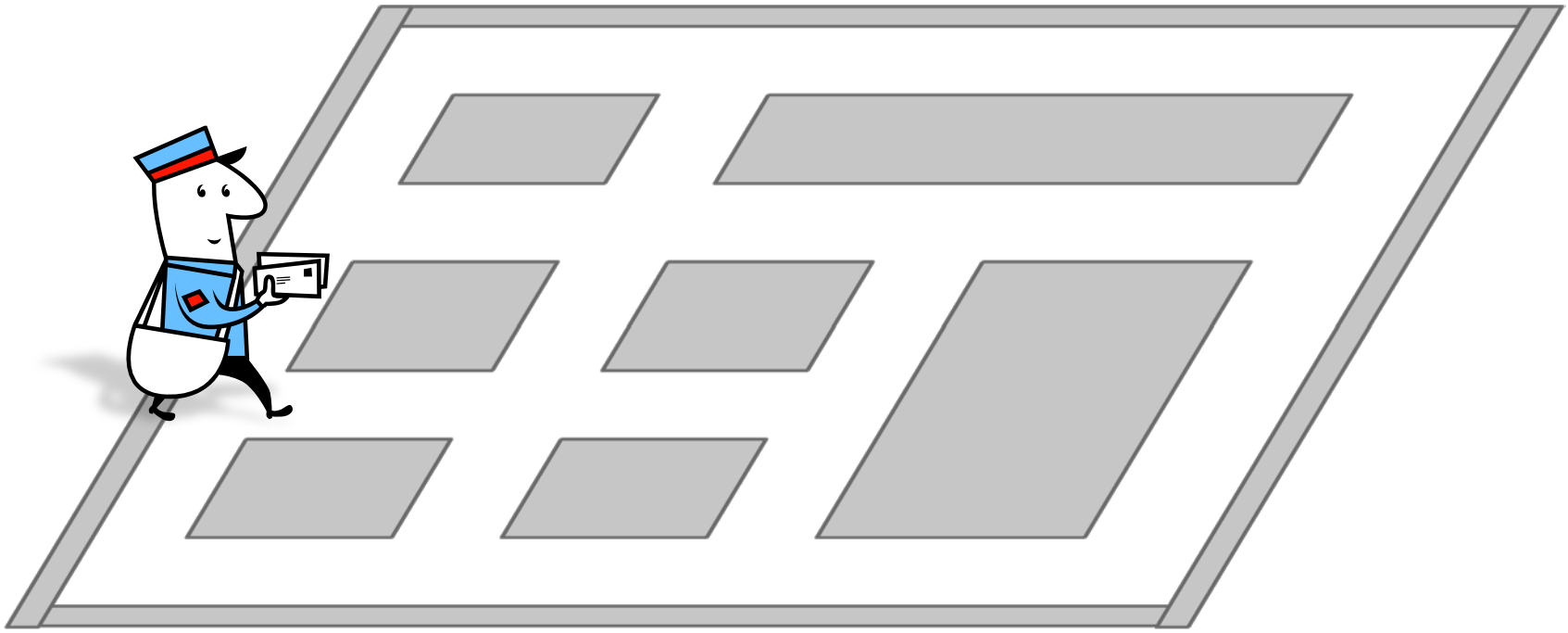
Eulerovský graf

```
Euler(v):
    while existuje neprojitá hrana {v,w} do
        označ hranu {v,w};
        Euler(w);
        vypiš v;
    done
```

- podobné jako DFS, vrcholy ale procházíme vícekrát

Pošťákův problém

Jak obejít všechny ulice a vrátit se do výchozího bodu tak, aby celková trasa byla minimální?



Pošťákův problém

Lemma: Nechť *S* je sled, který je optimálním řešením pošťákova problému v grafu *G*. Potom každá hrana grafu je v *S* obsažena maximálně dvakrát.

H .. podgraf *G,* sestává z hran, které jsou v *S* dvakrát

T .. množina vrcholů v *G* lichého stupně

Lemma: *H* se skládá z $|T|/2$ hranově disjunktních cest, jejichž konce leží v *T*.

Pošťákův problém - algoritmus

1. Určíme množinu *T* vrcholů lichého stupně.
2. Nalezneme $|T|/2$ cest, které spárují vrcholy v *T* a budou obsahovat co nejméně hran (využívá se algoritmus na hledání maximálního párování minimální ceny). Získáme tak graf *H*.
3. V multigrafu *G*+*H* nalezneme uzavřený Eulerovský graf.

Pošťákův problém