

① Abych našel takovou orientaci, že neorientovaný souv. graf bez mostů se stal orientovaným, silně souvislým, můžeme ~~zkusit~~ opravit DFS algoritmus tak, aby my jsme šli do hloubky (ke každému sousedu) a dělali jsme orientaci od souseda k sousedu pokud neprojdeme celkový graf a uděláme pro každou hrachu orientaci v obě strany (silně souv. graf)

Příklad algoritmu

SALOMUB

ILLIA

Orient - graf

$(\text{Drať}(V, e) \leftarrow$

$\text{libovolný-vrchol} \leftarrow$

~~Make-orient~~ $\text{Make-orient}(\text{lib. vrchol})$

$\text{Make-orient}(V - \text{vrchol})$

Pokud $\text{Jestli-čís-orient}() = \text{True}$

then **VYSTUP orient-graf**

Pro sousedy vrcholu V :

Pokud (V, soused) nejsou

in **orient-graf**:

and (soused, V) nejsou

in **orient-graf**:

then



orient-graf.Add(V, soused)

orient-graf = **Make-orient**
(**Soused**)

VYSTUP orient-graf

Testli-^u orient (vrchol):

Pro sousedy vrcholy Vrchol:

Pokud (vrchol, souse^d) nejsou in
orient-graf

hebo

(souse^d, vrchol) nejsou in orient:

VYSTUP FALSE

VYSTUP True

Zdruhodnime Správne^t Algoritmu:

Algoritmus dostane^t graf
bez mostu a je
přes vrcholy, začíná^t
z libovolného vrcholu:
Potom v funkci Make_orient^t
jdeme přes sousedy vrcholu
a v graf orient. graf
budeme zapisovat nové
orientované hrany
(v , $soused$), a také
DFS znovu ale pro sousedy,
s ním minulý vrchol udělá
novou orientační hranu
($soused$, v) která není
v orient. graf.

Funkce `Testli_už_orient`
kontroluje, má vrchol už
všechny možné hrany s `soused`
nebo seštěně.

Když algoritmus skončí,
tak vypíše nám orient-graf
že všema orientovanými hranami.

Algoritmus předpokládá,
že na vstupu bude
neorientovaný graf
bez mostek

② Abych dokazal že
pro graf G ~~je~~ topolog.
uspořádání určeno jednoznačně
jenom když G se DAG
musíme dokázat 2 věci:

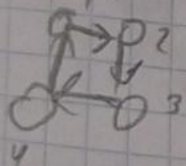
1. G má topologické uspořádání
2. G se DAG

① 1) Necht' graf má cyklus.
Můžeme udělat uspoř.:

$$v_1 \leq v_2 \leq \dots \leq v_k \leq v_1$$

2) Pro vše vrcholy musí platit
definice o otevírání a zavírání
vrcholu, ~~je~~

Napr. $v \in$



$$\text{out } i \in (1) \leq \text{in } 2 < \text{in } 3 < \text{in } 4$$

$$\text{out}(1) > \text{out}(2) > \text{out}(3) > \text{out}(4)$$

$\text{out}(4) > \text{out}(1)$ neplatí,

✓ topologickém usp. $\text{out}(1) \dots$ musí
byť rostoucí

Dostali jsme cyklus,
který
je možné udelat v
top. uspořádání.

② G je DAG

Podmínka: Pro důkaz existence
trození, že v každém
DAGu \exists zdroj (vrchol)
ke kterému nemají hrany
(Abych dokazal to můžeme
použít z libovolního
vrcholu proti směru hran,
na konci cesty bude zdroj)

Z podmínky rozumíme,
že v DAGu (acyklickém grafu)
jestli my odebereme zdroj,
tak on opět bude DAG
(Protože z definice on nikdy acyklický)

3) Abych rozmístil to nejmenší
strážníky tak, aby každá
ulice byla z alespoň jedné
strany hlídána musíme
napsat algoritmus v stylu DFS
(hledání do hloubky) protože
městečko je stom ~~Registry~~

Kod

$G(v, e)$

STRAŽNÍKY $\leftarrow 0$

kořen.stražník $\leftarrow \text{True}$

Pro vše vrcholy $v \in G$:

$v.\text{star} \leftarrow \text{nenalezený}$

goto ALGORITMUS(kořen)

potom

goto POČET(STRAŽNÍKY)

WSTUP(STRAŽNÍKY)

ALGORITMUS(v)

$v.\text{star} \leftarrow \text{otevřený}$

Pro vše děti vrcholu v :

Pokud $v.\text{star} \neq \text{zavřený}$:

$v.\text{stražník} = \text{True}$

goto ALGORITMUS(děti)

$v.\text{star} \leftarrow \text{zavřený}$

Počít(STRAŽNÍKY):

Pro vše vrcholy v G:

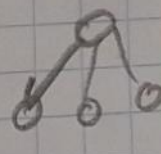
Pokud: v. strážník: True

$$\text{STRAŽNÍKY} + 1$$

VÝSTUP STRAŽNÍKY

Co dělá kod?

Každý vrchol má podmínku
v. strážník, která říká, musí
mít vrchol strážníka (True) nebo
ne (FALSE)

Předpokládáme, že graf 
musí mít právě 1 strážníka
v kořene.

V funkci Algoritmus procházíme
přes vrcholy a jejich děti;
jestli vrchol má děti, podmínka
strážník bude True, jestli
vrchol nemá děti, on nebude
mít strážníka.

Funkce POČET

počítá vše vrcholy které

ma v. strážník = True

a jejich počet

je minimální

počet strážníků

ve grafu LD