GRAFOVÉ ALGORITMY

2007/8 - 1. termín

1. Artikulace a bloky

Uvažujeme neorientované grafy. ***Artikulace*** je takový vrchol, že po jeho odstranění (včetně hran s ním incidentních) se zvětší počet souvislých komponent. ***Blok*** je (vzhledem k inluzi) maximální množina vrcholů vzhledem k následující vlastnosti : je alespoň dvouprvková a příslušný indukovaný podgraf je souvislý a nemá artikulaci.

Níže máte uvedeny dva algoritmy. První je varianta DFS, druhý by měl počítat artikulace a bloky souvislého grafu.

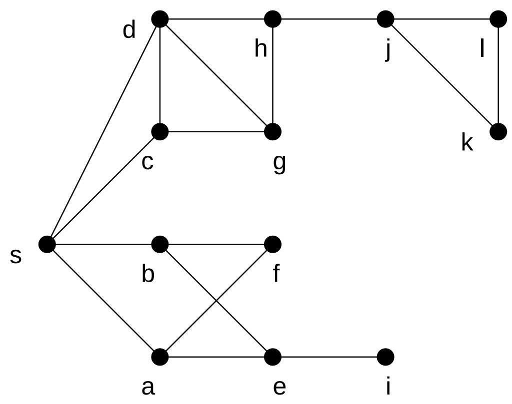
Komentář k proměnným :
$G = (V, E)$ je neorientovaný graf, $s \in V$,
$nr(v)$ je pořadí objevení vrcholu v,
$p(v)$ je předchůdce vrcholu v,
$u(e)$ je příznak objevení hrany e, přitom píšeme $e = uv = vu$ pro $e = \{u, v\}$,
S je zásobník,
C je množina všech artikulací,
L je funkce low,
k je počet bloků,
$B_1, \ldots, B_k$ jsou bloky.
Přitom $L(v)$ je definováno jako :

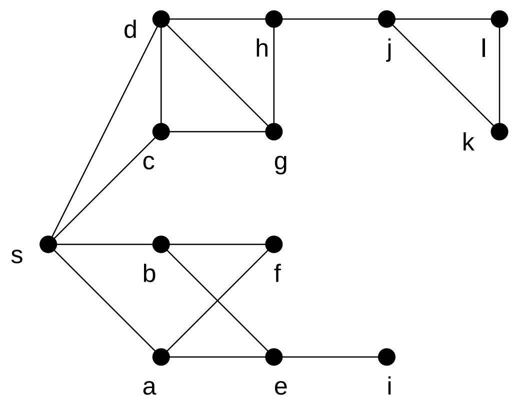
$$\min\{\ nr(v),\ \min\{\ nr(u) \mid \exists\, w \in V \text{ a } v{-}w{-}\text{cesta ze stromových hran a zpětná hrana } wu\ \}\ \}$$

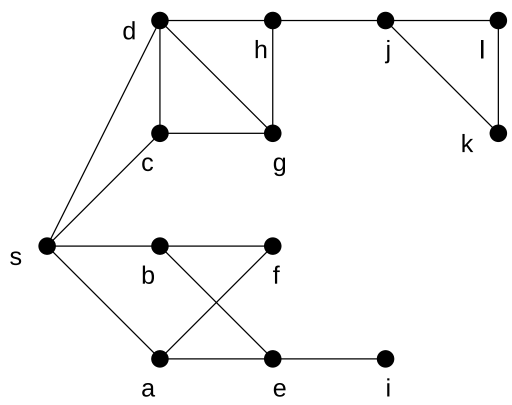
Na dvě místa do druhého algoritmu doplňte přiřazení/podmínku týkající se funkce L.
Dále doplňte :
Vrchol $v \neq s$ je artikulace právě když ....(v termínech L).

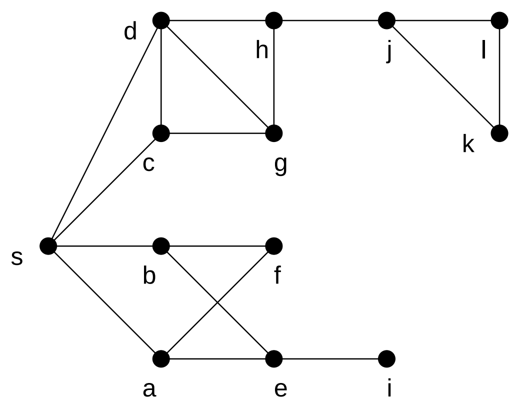
Vrchol s je artikulace právě když ....

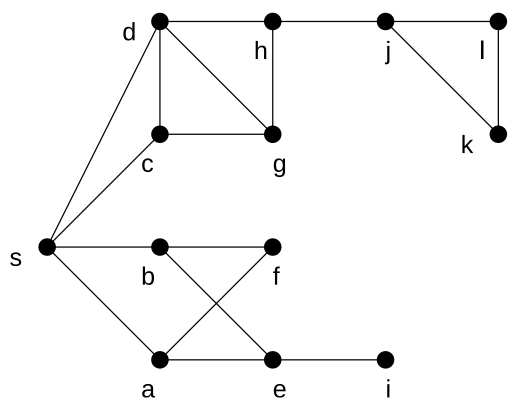
Průběh části výpočtu druhého algoritmu na níže uvedený graf uveďte do připravené tabulky. Pokud máme někde více možností, řídíme se abecedou. Přitom :
krok = vypořádání se s novou hranou nebo návrat po stromové hraně,
typ kroku : t = objevení stromové hrany,
b = objevení zpětné hrany,
n = návrat po stromové hraně.
Uveďte stav zásobníku S a proměnné L po každém kroku. Váš první krok : jeden krok před prvním návratem. Váš poslední krok : druhá stromová z s.
Po skončení výpočtu :
$C =$ ....
$k =$ ....
$B_1 =$ ....
$B_2 =$ ....
....
Do diagramu též pro každý vrchol v uveďte $nr(v)/L(v)$.

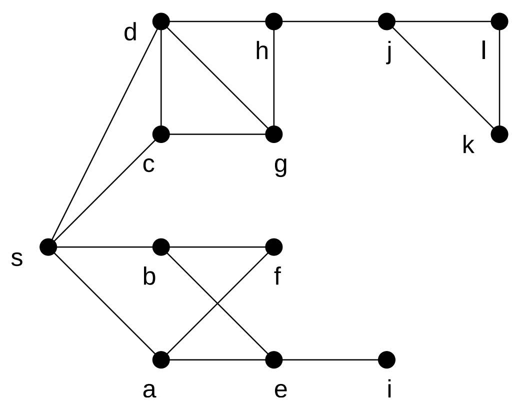












První algoritmus na souvislém grafu má složitost $O(|E|)$, neboť kromě inicializace pracujeme s každou hranou v každém směru právě jednou a vždy provedeme konstantní počet kroků. Ja je to pro druhý algoritmus ?

....

Dokažte, že druhý algoritmus správně počítá bloky (alespoň pro případ $p(v)$ je artikulace, $p(v) \neq s$).

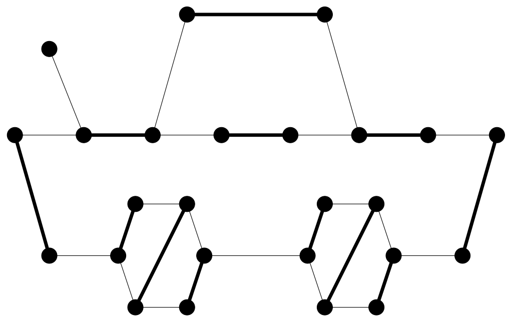
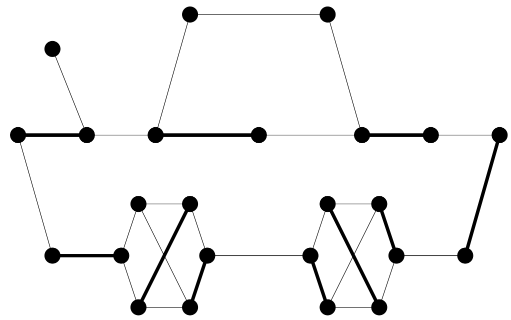
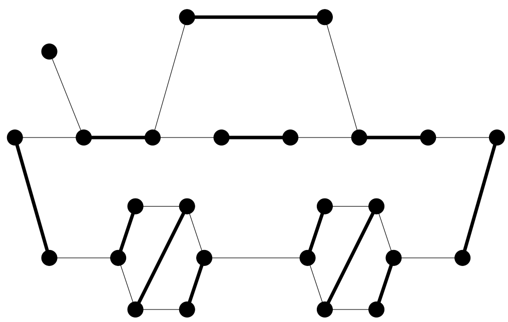
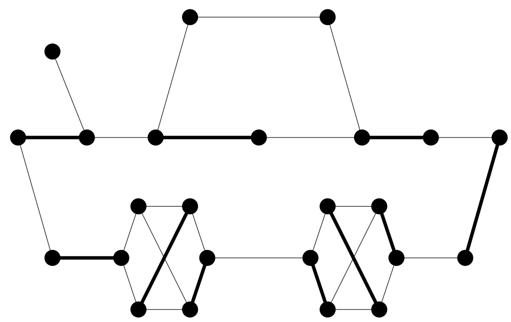
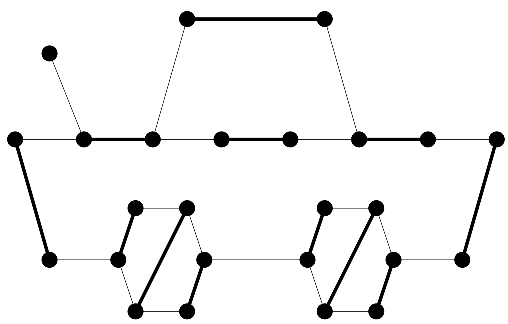
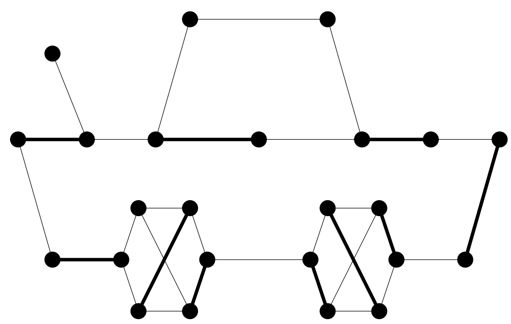
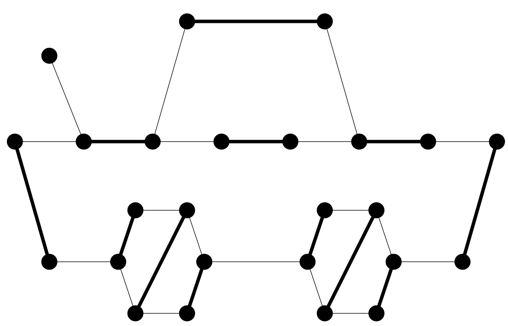
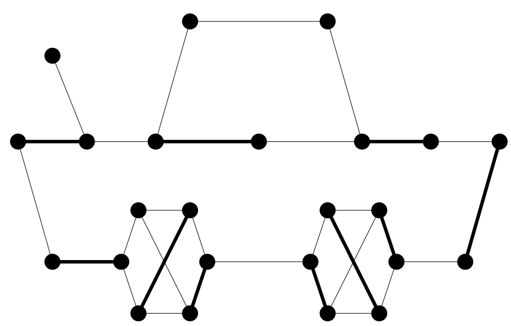
....

2. Párování

Grafy G a H spolu s jejich párováními jsou zadány obrázky níže. O každém z nich rozhodněte, zda je bipartitní a své rozhodnutí dokažte.

Zjistěte zda daná párování jsou maximální. Pro oba grafy najděte jejich maximální párování a dokažte maximalitu.

Můžete např. použít alternaci volných cest s tím, že když z daného vrcholu taková cesta neexistuje, nemusíme ho dále uvažovat.

GRAFOVÉ ALGORITMY

2007/8 - 2. termín

1. Druhá nejlepší minimální kostra

Nechť $G = (V, E)$ je souvislý neorientovaný graf, nechť $w : E \to \mathbb{R}$ je na různých hranách různá. Již víme, že existuje jediná minimální kostra (stručně MST). Kostra T je SBMST, jestliže má mezi všemi kostrami druhé nejmenší ohodnocení (takových koster může být více).

a) Nechť T je MST a T' je SBMST. Dokažte, že existují hrany $(u, v) \in T$ a $(x, y) \notin T$ tak, že $T' = (T \setminus \{(u, v)\}) \cup \{(x, y)\}$.

Důkaz : Uvědomte si, že libovolná kostra má $|V| - 1$ hran.

1. Nechť $T' \setminus T = \{(x, y)\}$. Pak $T \setminus T' =$ ....

a platí ....

2. Nechť $|T' \setminus T| \geq 2$. Pak i $|T \setminus T'| \geq 2$. Nechť (u, v) je minimální váhy z $T \setminus T'$. Pak $T' \cup \{(u, v)\}$ má cyklus c a na něm existuje hrana $(x, y) \in$

....

Sporem dokážeme, že $w(x, y) > w(u, v)$. V $T \cup \{(x, y)\}$ máme cyklus c'. Ten obsahuje $(u', v') \in$

....

Nyní $T'' = (T \setminus \{(u', v')\}) \cup \{(x, y)\}$ je kostra a tedy $w(u', v') <$ ....

Celkem

....

a to je spor s volbou

....

Konečně položme $T''' = (T' \setminus \{(x, y)\}) \cup \{(u, v)\}$. Pak $w(T''') <$ ....

Přitom $T''' \neq T$, neboť

....

A už konečný máme spor, neboť

....

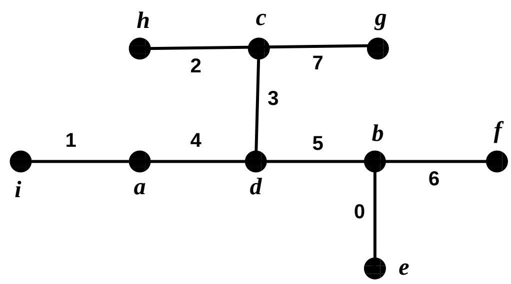
b) Následující algoritmus pro kostru $T \subseteq E$ počítá, pro všechna $u, v \in V$, maximálně ohodnocenou hranu na (jediné) u-v-cestě v T.

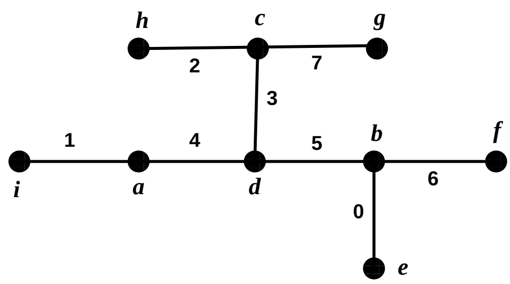
```
BFS-Fill-Max(T, w)
 1  for each vertex u ∈ V
 2       do for each vertex v ∈ V
 3               do {
 4                   max[u, v] ← NIL
 5                   }
 6          Q ← ∅
 7          Enqueue(Q, u)
 8          while Q ≠ ∅
 9               do {
10                   x ← Dequeue(Q)
11                   for each v ∈ Adj[x]
12                        do {
13                            if max[u, v] = NIL and .....
14                               then if x = u or .....
15                                       then max[u, v] ← (x, v)
16                                       else  max[u, v] ← max[u, x]
17                                    Enqueue(Q, v)
18                            }
19                   }
20  return max
```

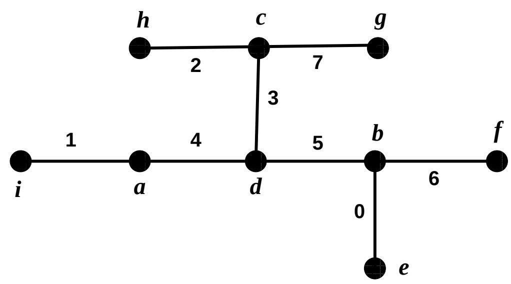
b1) Doplňte dvě chybějící formulky.

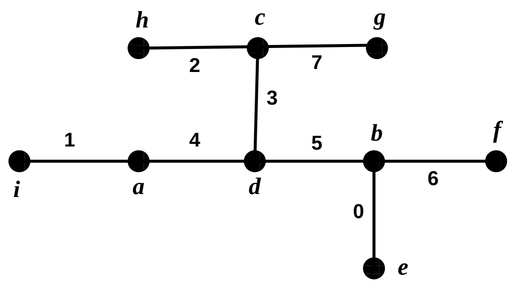
b2) Odhadněte složitost (při důkazu se můžete odvolat na BFS).

b3) Aplikujte algoritmus na následující strom. Seznamy sousedů jsou podle abecedy. Pro $u = a$ uveďte postupně všechny změny proměnných $Q, x, v,$ max společně s číslem řádku, kde k tomu došlo.









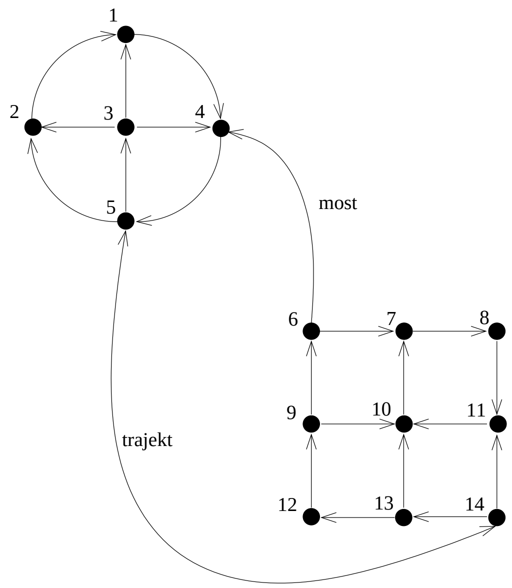
c) Je dána MST T a matice $(\max[u, v])_{u,v \in V}$. Jak spočítáte nějakou SBMST T' ?
Nalezneme $(x, y) \notin T$ minimalizující
....
a položíme
$T' =$ ....

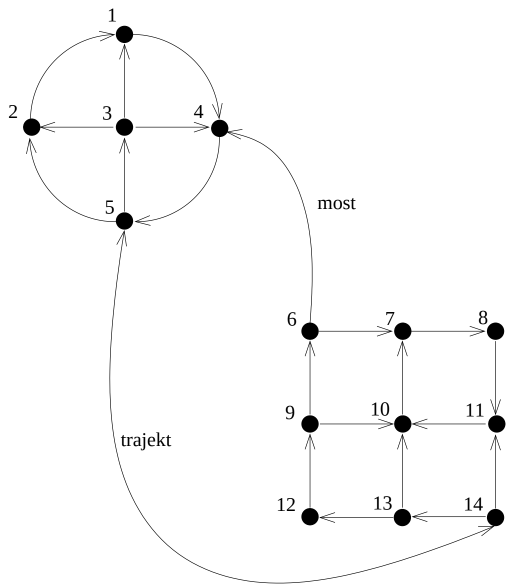
Důkaz korektnosti :
Podle bodu a) existují
....
tak, že
$T' =$ ....
přitom
$w(T') = w(T) -$ ....

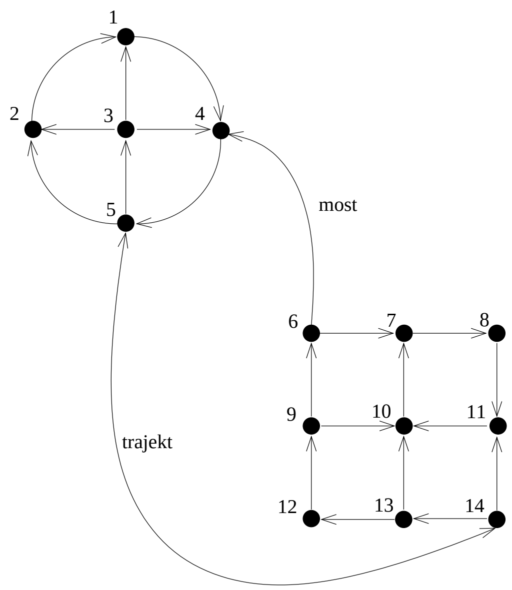
Pro dané (x, y) je (u, v) minimalizující $w(T')$ právě
....

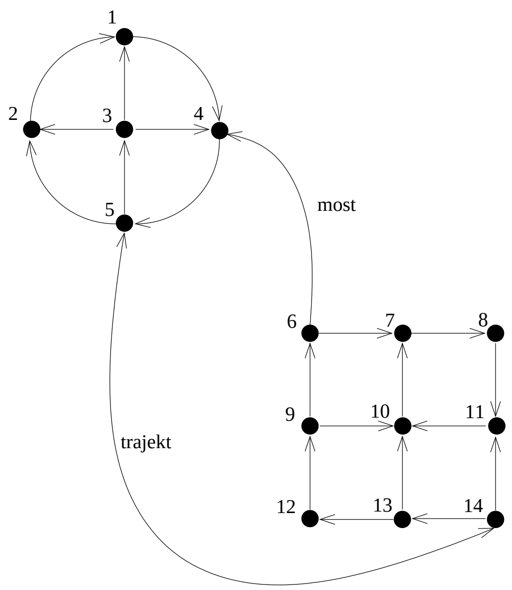
2. Silně souvislé komponenty

V městě Autokarově řešili problém s dopravou. Silnice jsou tu úzké a nestačí pro obousměrný provoz. Bylo potřeba z každé silnice udělat jednosměrku. Řešení strážníka Hlavičky vidíte na přiloženém plánku. Jediná obousměrná cesta je trajektová linka mezi body 5 a 14. Vaším úkolem je pomocí vhodného algoritmu spočítat silně souvislé komponenty takto vzniklého grafu (a navrhnout minimální počet a umístění autovrakovišť, které potřebuje Autokarov po změnách strážníka Hlavičky :)).









GRAFOVÉ ALGORITMY

2007/8 - 3. termín

1. st-očíslování

Je dán neorientovaný graf $G = (V, E)$ s $n \geq 2$ vrcholy, který je 2-souvislý (t.j. je souvislý a po odstranění libovolného vrcholu se spolu s ním incidentními hranami zůstává souvislý).

Máme najít bijekci STN : $V \to \{1, \ldots, n\}$, píšeme $v = v_i$ pro $\mathrm{STN}(v) = i$, tak, aby :

(i) $\{v_1, v_n\} \in E$,

(ii) pro každé $j \in \{2, 3, \ldots, n-1\}$ existují $i, k \in \{1, \ldots, n\}$ tak, že $i < j < k$ a $\{v_i, v_j\}, \{v_j, v_k\} \in E$.

Předpokládáme, že již proběhl průzkum do hloubky. Čas objevení $(1, 2, \ldots)$ vrcholu v nechť je DFN(v), jeho otec je FATH(v).

Hodnota funkce LOW(u) je definováno jako :

$$\min\{\ \mathrm{DFN}(u),\ \min\{\ \mathrm{DFN}(w) \mid \exists\, x \in V \text{ a } u{-}x{-}\text{cesta ze stromových hran a zpětná } (x, w)\ \}\ \}\ .$$

Dále předpokládáme, že máme napočítanou i funkci LOW spolu s příslušnými cestami.

Pro $v \in V$ se cesta PATH(v) určuje takto :

Případ 1. Existuje nová zpětná hrana (v, w). Prohlásíme ji za starou a položíme $\mathrm{PATH}(v) = (v, w)$.

Případ 2. Existuje nová stromová hrana (v, u). Nechť $(u = u_0, u_1, \ldots, u_k, w)$ je cesta dávající $\mathrm{LOW}(u) = \mathrm{DFN}(w)$. Proč je $u \neq w$?

....

Položíme $\mathrm{PATH}(v) = (v, u_0, u_1, \ldots, u_k, w)$ a všechny vrcholy a hrany na ní prohlásíme za staré.

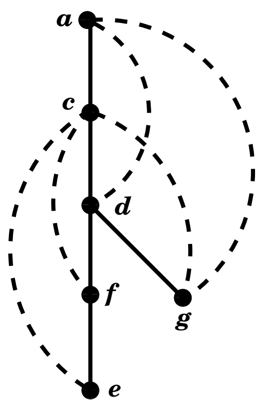
Případ 3. Existuje nová zpětná hrana (u, v). Nechť $(u = u_0, u_1, \ldots, u_k, w)$ je cesta nahoru po stromových hranách do prvního starého w. Položíme $\mathrm{PATH}(v) = (v, u_0, u_1, \ldots, u_k, w)$ a všechny vrcholy a hrany na ní prohlášíme za staré.

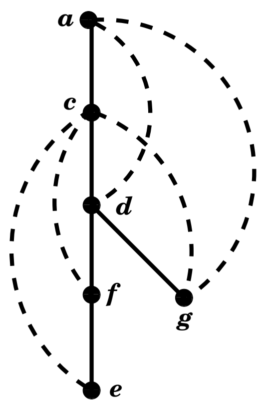
Případ 4. Všechny hrany incidentní s v jsou staré. Položíme $\mathrm{PATH}(v) = \emptyset$.

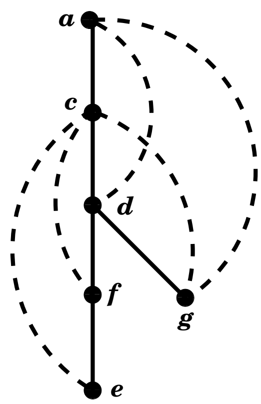
ST-Number(G)

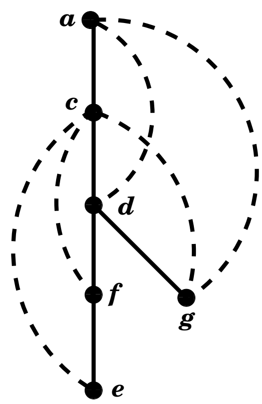
1 **begin**
2 mark s, t and (s,t) “old” and all the other vertices and edges “new”;
3 push down t and s into stack S in this order;
4 $\{s$ is over $t\}$
5 COUNT $\leftarrow$ 1;
6 pop up the top entry v from S;
7 **while** $v \neq t$
8 **do begin**
9 **if** PATH$(v) = \emptyset$
10 **then begin**
11 STN$(v) \leftarrow$ COUNT;
12 COUNT $\leftarrow$ COUNT $+ 1$
13 **end**
14 **else begin**
15 let PATH$(v) = (v, u_1, u_2, \cdots, u_k, w)$;
16 push down the vertices u_k, u_{k-1}, $\ldots$, u_1, v into S in this order
17 $\{v$ is a top entry of $S\}$
18 **end**
19 **fi**;
20 pop up the top entry v from S
21 **end**;
22 STN$(t) \leftarrow$ COUNT
23 **end**.

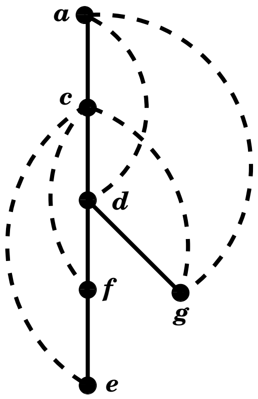
Algoritmus demonstrujte na níže uvedených diagramech. Zastavujeme se vždy na ř. 19. Nad diagram uveďte případ (1 až 4), podle něhož se našla cesta. “Old” vrcholy a hrany vyznačte modře, nalezenou (orientovanou) cestu červeně. Pod diagram uveďte stav zásobníku S a změny proměnné STN od posledního kroku. Začínáme s modře vyznačenými $a, c, \{a, c\}$ a zásobníkem c, a (odshora).

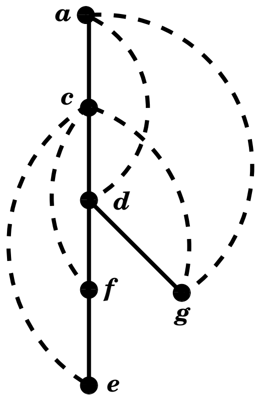


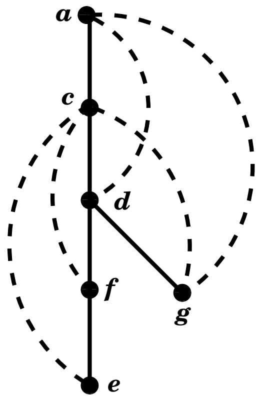


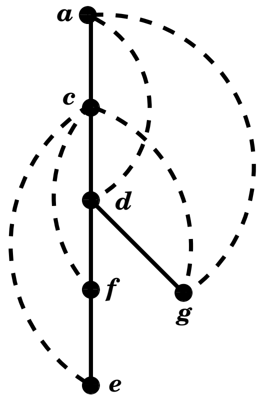


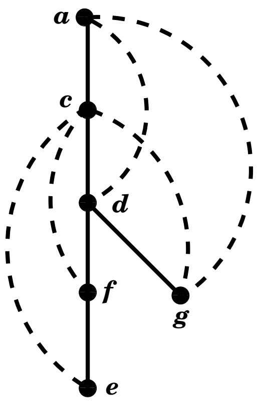


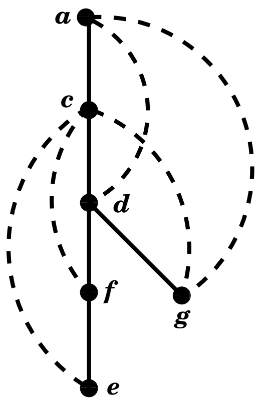


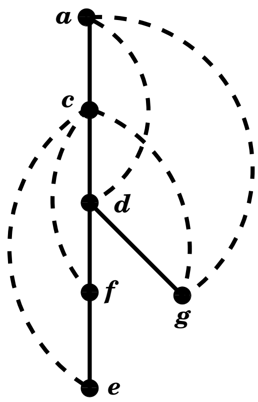


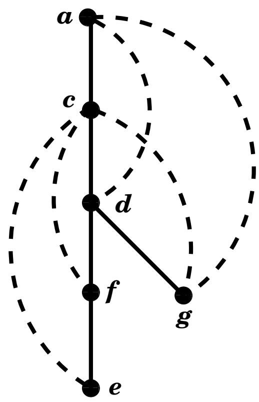












Dostali jste skutečně st-očíslování ? Ke každému i připište v_i a namalujte všechny hrany množiny E. (Druhý diagram máme pro ty co pokazí první.)

Korektnost :

Proč nastane některý z případů 1 - 4 ?

....

Libovolný v je označen jako starý a definitivně odstraněn z S pouze v případě, že všechny hrany s ním incidentní jsou staré.

G je 2-souvislý a proto je lib. v dosažitelné z s

....

a tedy každý vrchol se dostane do S a je prohlášen za starý předtím, než je t odstraněn.

Zřejmě $\mathrm{STN}(s) = 1$, $\mathrm{STN}(t) = n$ a $\{s, t\} \in E$.

Vrchol $u \neq s, t$ jde poprve do S jako vnitřní vrchol nějaké cesty $(v, u_1, \ldots, u_i = u, \ldots, w)$. Přitom w je starý, neboť

....

a w je stále v S, neboť jsme objevili novou hranu, která

....

V zásobníku je tedy v jistém okamžiku u nad w a pod v.

Konečně si uvědomíme, že je-li někdy x nad y, je STN ....

Proč ?

....

Jaká je **složitost** našeho algoritmu ? Zdůvodněte.

....

2. Obchodní cestující.

Obchodní cestující Kliment Stíhačka má zítra ráno v plánu několik obchodních jednání po České i Slovenské republice. Bydlí v Českých Budějovicích a má navštívit Brno, Cheb, Liberec, Nitru a Žilinu, v libovolném pořadí. Chtěl by mít služební cestu co nejkratší, ale vybrat takovou je (NP-)těžké a na to do rána nemá čas. Můžete mu však pomoci následujícím 2-aproximativním algoritmem.

Uvažujme úplný neorientovaný graf s vrcholy tvořenými výše jmenovanými městy a váhami hran danými vzdáleností měst v kilometrech:

	Brno	Č. B.	Cheb	Liberec	Nitra
Žilina	203	381	550	379	142
Nitra	174	339	541	406	
Liberec	235	236	246		
Cheb	374	230			
Č. B.	182				

1. Najděte nejlacinější kostru T tohoto grafu.
2. Zdvojte v T každou hranu a nakreslete takto vznilký obrazec jedním tahem tak, abyste začali i skončili ve vrcholu České Budějovice.
3. Vypište vrcholy v pořadí, v jakém jste je poprvé potkali při kreslení. To bude plán cesty pro Klimenta.

Ve druhém kroce máte možnost volby, jak obrazec nakreslíte. Proveďte kroky 2 a 3 pro jinou volbu nakreslení než poprvé. Určete, která z výsledných cest je kratší.

Bodování:

Nalezení minimální kosty: 12 bodů maximálně.

Vypsání cest: 7+7 bodů maximálně.

Určení kratší z nich: 4 bodů.