# KSI

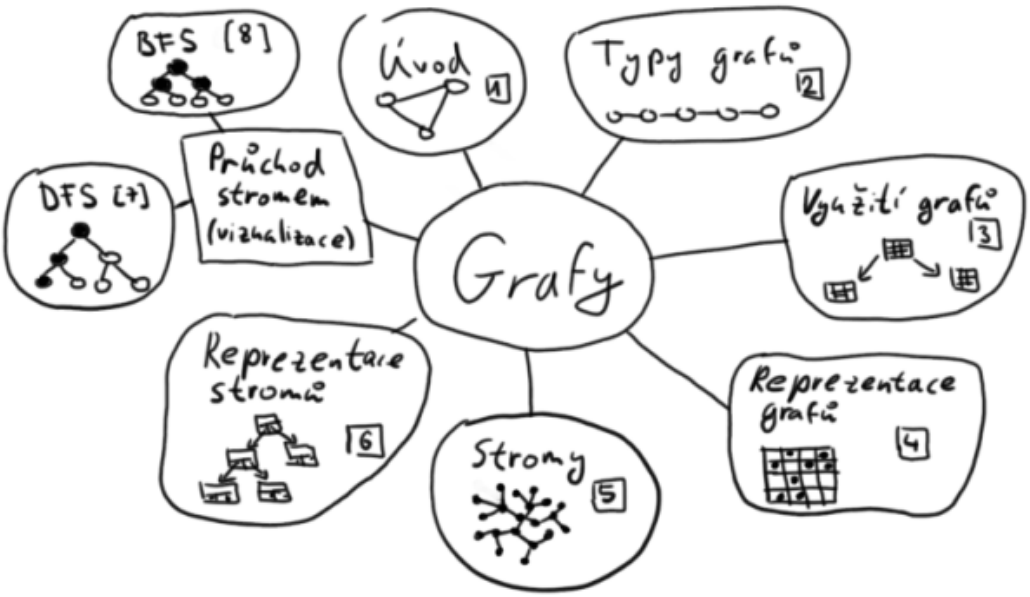
## Korespondenční Seminář z Informatiky

Milí řešitelé,

děkujeme vám za zaslaná řešení druhé sady KSI. V té třetí se budeme zabývat stromy. Někomu by se možná mohlo zdát, že strom nemá s informatikou mnoho společného, ale opak je pravdou.

## Grafy a stromy

Velkou část úvodníku jsme pro Vás tentokrát připravili formou videí, jejichž přehled vidíte na tomto grafu:




Rozcestník k videím naleznete na webové stránce:
`http://effaacademy.xf.cz/informatika`

- Úvod do teorie grafů[1]
- Typy grafů[2]
- Využití grafů[3]
- Reprezentace grafů v paměti[4]
- Stromy[5]
- Reprezentace stromů v paměti[6]
- Průchod stromu do hloubky (DFS) - vizualizace[7]
- Průchod stromu do šířky (BFS) - vizualizace[8]

### Pojmy

**Graf** – dvojice skládající se z množiny vrcholů (označené $V$) a množiny hran (označené $E$), kde každá hrana spojuje dva vrcholy.

[1] `http://www.youtube.com/watch?v=cZdlDgDhm70`
[2] `http://www.youtube.com/watch?v=2FLi8dV7DO8`
[3] `http://www.youtube.com/watch?v=Dp9mD2QPO_g`
[4] `http://www.youtube.com/watch?v=02vfx_crXSw`
[5] `http://www.youtube.com/watch?v=6kYYOc7VJw0`
[6] `http://www.youtube.com/watch?v=rZTQQbguPLw`
[7] `http://www.youtube.com/watch?v=0uz3Nj8ZnUY`
[8] `http://www.youtube.com/watch?v=a6Mx8vhHvS4`

**Neorientovaný graf** – hrany spojují své koncové vrcholy a není určeno, který vrchol je začátek hrany a který konec. Důležité je, že jsou dva vrcholy spojeny hranou. Hrana z $A$ do $B$ je totéž jako hrana z $B$ do $A$.

**Orientovaný graf** – hrany jsou šipky z jednoho vrcholu do druhého. Hrana z $A$ do $B$ je jiná hrana než hrana z $B$ do $A$.

**Sousední vrcholy** v grafu – jsou spojeny hranou.

**Podgraf** – obdoba pojmu podmnožina. Podgraf vznikne vymazáním některých vrcholů původního grafu, všech hran do těchto vrcholů zasahujících a případně některých dalších hran.

**Ohodnocený graf** – graf, u něhož jsou hrany označeny nějakou informací, typicky číslem (např. vzdálenost mezi městy).

**Cesta délky n** – $n$ vrcholů, které jsou uspořádány za sebou, z každého vrcholu kromě posledního vede hrana do následujícího vrcholu (celkem $(n-1)$ hran).

**Cyklus (kružnice)** – skoro jako cesta, s tím rozdílem, že první vrchol je stejný, jako poslední vrchol.

**Acyklický graf** – neobsahuje žádný cyklus.

**Souvislý graf** – takový (neorientovaný) graf, v němž platí, že pro každé dva vrcholy $x$, $y$ existuje alespoň jedna cesta z $x$ do $y$.

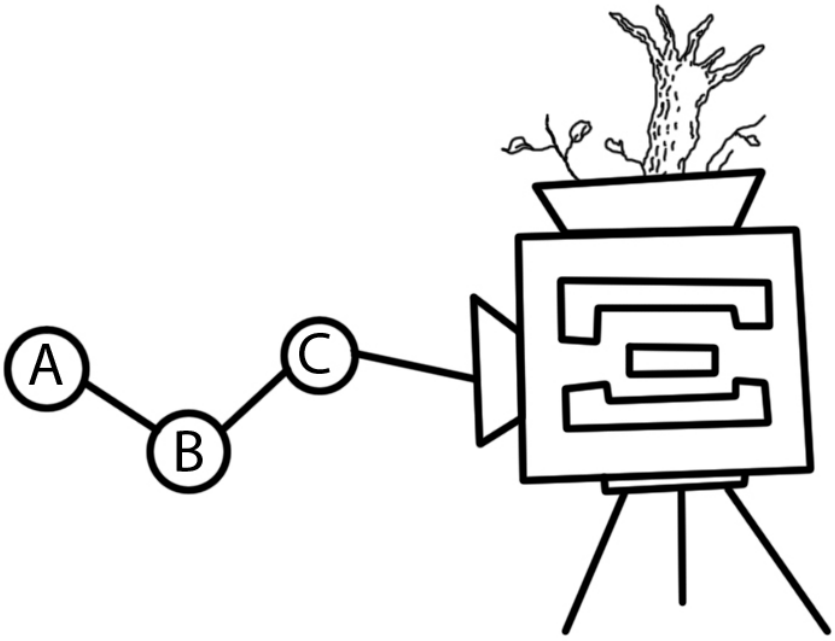
**Strom** – souvislý graf neobsahující cykly.

**Úplný graf** – každý z vrcholu tohoto grafu je spojen hranami se všemi ostatními vrcholy.

**Smyčka** – hrana z vrcholu do sebe samého.

**Matice sousednosti** – tabulka vyjadřující, které vrcholy spolu sousedí. Konkrétně buňka $M_{i,j}$ obsahuje hodnotu `true` právě tehdy, když je hrana mezi vrcholem $i$ a $j$. Pokud reprezentujeme ohodnocený graf, buňky místo hodnot `true` obsahují hodnoty příslušných hran.

**Seznam sousednosti** – obsahuje pro každý vrchol seznam jeho sousedů.




**Procházení grafu**

Představme si, že máme graf a potřebujeme najít nejkratší cestu z vrcholu A do vrcholu B. Nebo chceme jen zjistit, jestli mezi těmito vrcholy nějaký cesta existuje. Nebo chceme zjistit, jestli je daný graf stromem, jinými slovy, jestli obsahuje nějaký cyklus. Ve všech případech musíme jakýmsi způsobem procházet grafem, abychom našli odpověď na naše otázky. Představíme si proto dva standardní způsoby procházení, které nám na většinu problémů vystačí. Jedná

se o *procházení do hloubky*, aneb DFS (Depth-first Search) a *procházení do šířky*, aneb BFS (Breadth-first Search).

Obě metody jsou v jádru podobné a jejich myšlenka je velice jednoduchá. V obou případech používáme jakousi pomocnou datovou strukturu pro ukládání vrcholů pro pozdější zpracování. Na začátku si to této struktury vložíme jeden, daný počáteční vrchol a dále pokračujeme jednoduchým cyklem, jak je znázorněno v následujícím pseudokódu:

**Algorithm 1** Průchod

```
Vstup: vrchol v
  vlož v do datové struktury
  while struktura není prázdná do
    x ← vyber vrchol ze struktury
    označ x za navštívený
    vlož nenavštívené sousedy x do struktury
  end while
```

Rozdíl mezi průchodem do šířky a průchodem do hloubky je pouze ve volbě oné datové struktury. V případě průchodu do šířky používáme tzv. *frontu*. Frontu si představte jako obyčejnou frontu (např. lidí) – když přidáme prvek do struktury, zařadí se na konec; pokud chceme vybrat prvek ze struktury, vybereme zepředu. Použití fronty způsobí, že nejprv projedeme všechny sousedy počátečního vrcholu, pak sousedy jeho sousedů atd. V podstatě tedy vrcholy procházíme podle vzdálenosti od počátečního vrcholu (tento způsob tedy může být výhodný při hledání nejkratší cesty).

V případě procházení do hloubky používáme *zásobník*. Zásobník je datová struktura, kde prvky vložené jako poslední budou odebrány první. Při vkládání prvku do zásobníku ho zařadíme na konec, při odebrání prvku ovšem odebíráme také z konce. Tato metoda má hlavní výhodu v implementaci, neboť lze výhodně využít rekurzi. Díky rekurzi nemusíme zásobník nijak implementovat, protože využijeme zásobník pro volání funkcí. Prohledávání do hloubky pak může vypadat takto:

**Algorithm 2** Průchod do hloubky

```
Vstup: vrchol v
  označ v jako navštívený
  for all soused s vrcholu v do
    if s není označen jako navštívený then
      PrůchodDoHloubky(s)
    end if
  end for
```

Pomocí průchodu do hloubky lze snadno označit všechny dosažitelné vrcholy či zjistit, jestli je v (neorientovaném) grafu nějaký cyklus (pokud v cyklu foreach najdeme vrchol, který již byl navštívený, můžeme říct, že zde je cyklus, neboť jsme se do něj prve museli dostat jiným způsobem).

**Procházení stromu**

Podívejme se nyní na to, jak vypadá standardní procházení do šířky a do hloubky na stromech.

Při procházení do šířky procházíme vrcholy podle vzdálenosti od počátečního vrcholu. Pokud si tedy představíme začáteční vrchol jako kořen stromu, bude procházení do šířky odpovídat procházení po vrstvách. Začneme v kořeni, pak pokračujeme jeho přímými potomky, poté potomky jeho potomků atd.

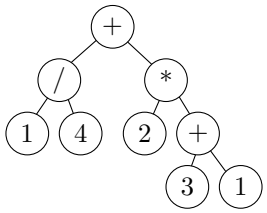
Na procházení do hloubky není nic speciálního. Situace je dokonce jednodušší než na normálních grafech. Jelikož zde nejsou cykly, nemůže se nám stát, že by nějaký z následníků zpracovávaného vrcholu již byl navštíven dříve. Průchod stromu do hloubky lze tedy jednoduše implementovat pomocí rekurzivní funkce:

**Algorithm 3** Průchod stromu

**Vstup:** vrchol $v$
  **for all** potomek $p$ vrcholu $v$ **do**
    PrůchodStromu($p$)
  **end for**

Při procházení grafu typicky provádíme v každém vrcholu nějakou akci (pouhý průchod vrcholu totiž k ničemu dobrý není). U binárních stromů rozlišujeme tři způsoby průchodu do hloubky podle toho, ve kterém bodě akci provedeme:
**Preorder** proveď akci; projdi levý podstrom; projdi pravý podstrom.
**Inorder** projdi levý podstrom; proveď akci; projdi pravý podstrom.
**Postorder** projdi levý podstrom; projdi pravý podstrom; proveď akci.

Prakticky si to ukážeme na aritmetických výrazech. Každý výraz lze reprezentovat jako binární strom - operátory +,-,*,/ budou ve vnitřních vrcholech stromu, v listech jsou čísla. Např: 1/4+2*(3+1)




Představme si, že chceme takto reprezentovaný aritmetický výraz vyhodnotit. Nabízí se jednoduché řešení prohledáváním do hloubky:

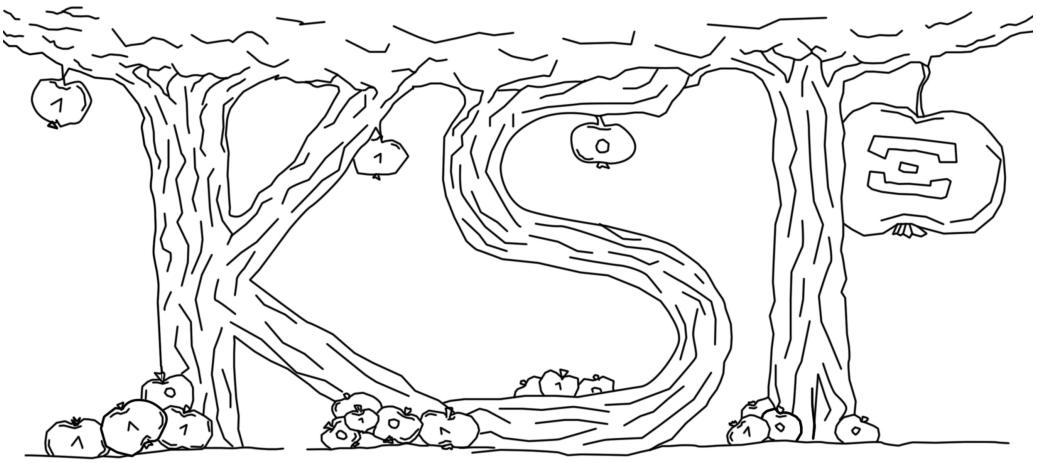
**Algorithm 4** Vyhodnoť

**Vstup:** vrchol $v$
  **if** $v$ je list **then**
    **return** $v$
  **else**
    Vyhodnoť(levý potomek $v$)
    Vyhodnoť(pravý potomek $v$)
    **return** hodnota levého potomka +,-,*,/ hodnota pravého potomka (podle typu vnitřního vrcholu)
  **end if**

Zde tedy nejprv vyhodnotíme oba podstromy a pak provedeme akci (tj. vyhodnocení příslušné aritmetické operace s již známými hodnotami) a jedná se proto o metodu postorder. Pokud bychom tento výraz naopak chtěli vypsat v běžné matematické notaci, provedli bychom to pomocí tohoto průchodu do hloubky:

Zde jsme akci provedli *mezi* průchody podstromů a jedná se tedy o způsob *inorder*. Poznamenejme, že tomuto způsobu zápisu matematických výrazů se říká *infixová* notace - operátor se zapisuje mezi operandy. Pokud bychom místo toho použili *preorder* nebo *postorder*, dostali bychom *prefixovou*, resp. *postfixovou* notaci. Výše zmíněný výraz pak vypadá následovně: 1 4 / 2 3 1 + * + (postfixová), + / 1 4 * 2 + 3 1 (prefixová).

Postfixová notace byla dříve často používána v kalkulačkách. Má některé zajímavé výhody - např. nejsou potřeba závorky (rozmyslete si proč). Prefixová notace nachází aplikaci především



**Algorithm 5** Vypiš

```
Vstup: vrchol v
  if v je list then
    vypiš číslo ve vrcholu v
  else
    Vypiš(levý potomek v)
    Vypiš typ operace ve vrcholu v
    Vypiš(pravý potomek v)
  end if
```

v informatice, či v matematice při zápisu funkcí. Jako první vždy píšeme jméno funkce a za ní následuje seznam jejich parametrů. Tedy například *Plus(Děleno(1,4), Krát(2, Plus(3, 1)))*.

Tímto náš letmý úvod do tajů stromů a grafů končí. Přejeme vám hodně zdaru při řešení úloh 3. sady!

## Zadání 3. sady úloh KSI (termín odevzdání: 6. 1. 2013)

*Řešení zasílejte pomocí internetového systému na adrese* `http://ksi.fi.muni.cz`.

**Příklad 1: Říční byznys** (10 bodů)

Povodí řeky Stromovky je velmi rozsáhlé, vlévá se do ní spousta přítoků, do nichž se vlévá další spousta přítoků, do nichž se zase vlévá spousta přítoků a tak dále. Nejsou na ní však žádné ostrovy, jezírka ani slepá ramena, pouze soutoky a úseky. Soutok je zde definován jako místo, kde se slévají dva a více přítoků (úseků). Jako úsek pak označujeme část vodní plochy mezi dvěma soutoky. Dva úseky spolu sousedí právě tehdy, když jsou odděleny jedním soutokem. Hlavní tok Stromovky pro nás není důležitý, všechny úseky v povodí jsou stejně významné. Protože povodí je jednou z nejkrásnějších oblastí na světě, turistický zájem o něj je obrovský, a tak se našla spousta společností pořádajících vyhlídkové plavby, které by zde rády dělaly byznys. Aby nedocházelo k problémům, antimonopolní úřad vydal pravidlo, podle nějž žádná společnost nemůže plavby provozovat na dvou sousedních úsecích (to však neznamená, že nemůže pořádat vyhlídkové plavby na více úsecích, které spolu nesousedí). Kolik (obecně) nejméně výletních společností musí v povodí podnikat, aby nezůstal žádný úsek nevyužitý?

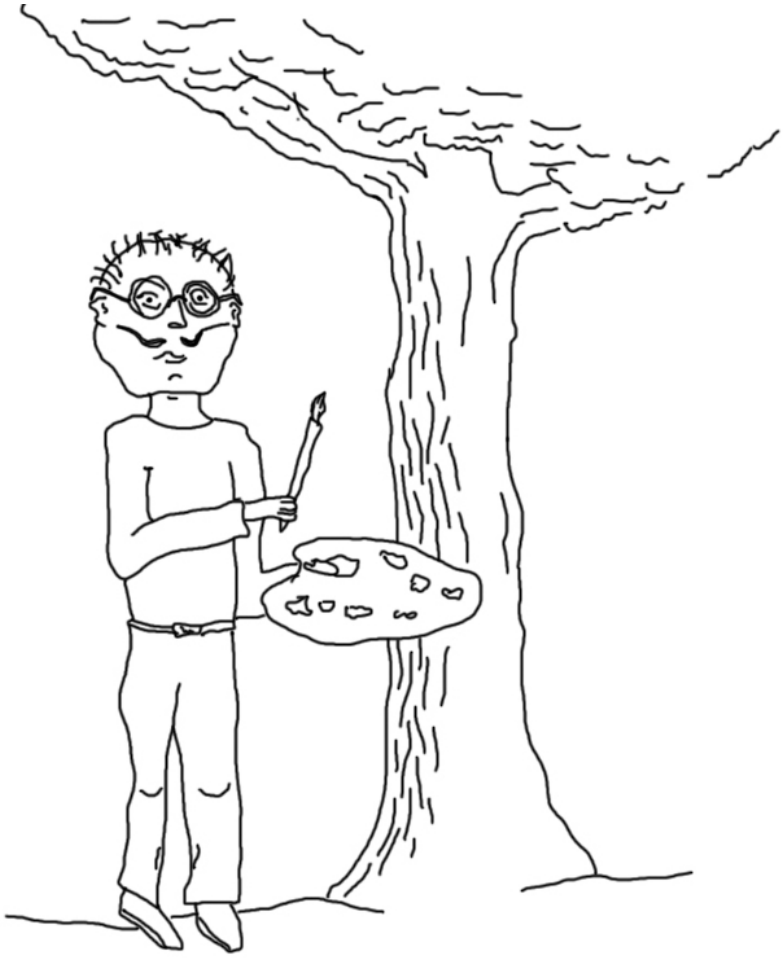
Povodí však není zajímavé jen z turistického pohledu, ale také z rybářského. Vyskytuje se zde jedinečný druh ryb *Piscis Informatica*, který žije pouze v soutocích, nikoli na jednotlivých úsecích. Po tomto objevu se do oblasti nahrnula spousta rybářských společností, proto musel opět zasáhnout antimonopolní úřad. Zavedl podobné pravidlo – žádná společnost nesmí okupovat dva soutoky, mezi nimiž vede právě jeden říční úsek. Kolik nejméně rybářských společností musí v povodí podnikat, aby byly využity všechny soutoky?

Tím však naše podnikání zdaleka nekončí! Výletní společnosti se totiž rozhodly, že koupí společnosti rybářské a spojí tak tato dvě řemesla. To ovšem znamená spojení obou pravidel – pokud podnikají na soutoku, nesmí zároveň podnikat na žádném z úseků vedoucím do nebo z tohoto soutoku a pokud podnikají na říčním úseku, nesmí podnikat na soutoku na jeho začátku ani konci. Kolik nejméně takovýchto spojených společností musí na řece podnikat nyní, pokud opět chceme využít všechny soutoky a úseky?

U každé odpovědi uveďte i důkaz, proč je právě vaše odpověď správná.

**Příklad 2: Vyhodnocování výrazů** (10 bodů)

V obálce jste spolu se zadáním dostali kartičky (pokud ne, tak si je můžete stáhnout z webu). Na těchto kartičkách jsou zapsány matematické výrazy. Vašim úkolem je tyto zápisy výrazů přepsat do normální, čitelné podoby. Jako řešení posílejte seznam dvojic *číslo karty - výraz* seřazený od nejnižšího čísla karty po nejvyšší.



**Příklad 3: Farma** (10 bodů)

Ferdinand sa rozhodol, že si doma postaví počítačovú farmu a konečne vyráta tie mimozmeské signály zo SETI, naráta si kopu bitcoinov, hrubou silou sa nabúra do všetkých bezpečnostných agentúr, a vôbec, čo už sa tak dá robiť s takou počítačovou farmou. I nakúpil výhodne v zľave hromadu nových strojov.

Lenže keď mu ich doviezli, zistil, že už nemá dosť káblov na to, aby ich rozumne poprepájal. A to konkrétne, doviezli mu $n$ strojov, ale on má len $n-1$ káblov. Poprepájal teda svoje počítače aspoň týmito $n-1$ káblami tak, aby medzi každými dvoma viedla káblová cesta (čiže zapojil ich do stromu.)

Teraz ale potrebuje, aby jeho stroje vedeli spolu nejak rozumne komunikovať.

Každý stroj má svoje unikátne id, a každý má 1 alebo viac výstupov. Na druhom konci každého takého výstupu sedí iný stroj Ferovej farmy. Hneď po zapojení a zapnutí každý stroj vie o svete len to, aké je jeho vlastné ID a koľko má výstupov. Svet z pohľadu vrchola teda vyzerá napríklad takto:


Krúžok predstavuje samotný vrchol a výstupky sú jeho výstupy, na druhom konci ktorých sú iné stroje, nevie však ešte, ktoré.

Na konci chceme, aby si stroje vedeli posielať správy navzájom - čiže vytvoriť routovacie tabuľky pre každý vrchol. V takej tabuľke bude mať každý stroj napísané údaje potrebné k tomu, aby keď k nemu príde správa určená pre konkrétny iný vrchol siete vedel, ktorým výstupom ju má poslať ďalej. Pre potreby tohto príkladu nie je podstatné, akú zložitosť bude mať samotný výpočet toho, kam správu preposlať s pomocou tejto tabuľky.

Vašou úlohou je napísať jeden program, ktorý spustíme na všetkých strojoch naraz, a ktorý nám zabezpečí, že si každý vrchol vytvorí takúto routovaciu tabuľku.

Na každom zo strojov sú nastavené premenné ID (ID tohto stroja) a PortNum(počet portov tohto stroja, porty sú očíslované 1 až PortNum). Program môže okrem štandardných vecí ako cykly, polia, etc... používať aj príkazy:

send(port, sprava) - pošle správu <sprava> výstupom <port>

sprava := get(port) - dostane správu z výstupu <port> a priradi ju premennej sprava. Pokiaľ na výstupe <port> žiadna správa práve nečaká, bude premenná sprava prázdna.

stop - ukončí celý program vtedy, ak si je už istý, že pozná celú routovaciu tabuľku. Ak vrchol zavolá stop, nemôže už ani preposielať správy ostatných vrcholov, ktoré ešte stop nezavolali.

Okrem toho, či algoritmus zvládne naozaj vytvoriť routovaciu tabuľku v každom vrchole nás ešte zaujíma celkový počet správ, ktoré sa prenesú. Vašim cieľom je samozrejme, aby sa dokopy v celej sieti prenieslo čo najmenej správ, kým nebudú mať všetky vrcholy správnu tabuľku. Nie je vyslovene nutné, aby každý vrchol zavolal stop.

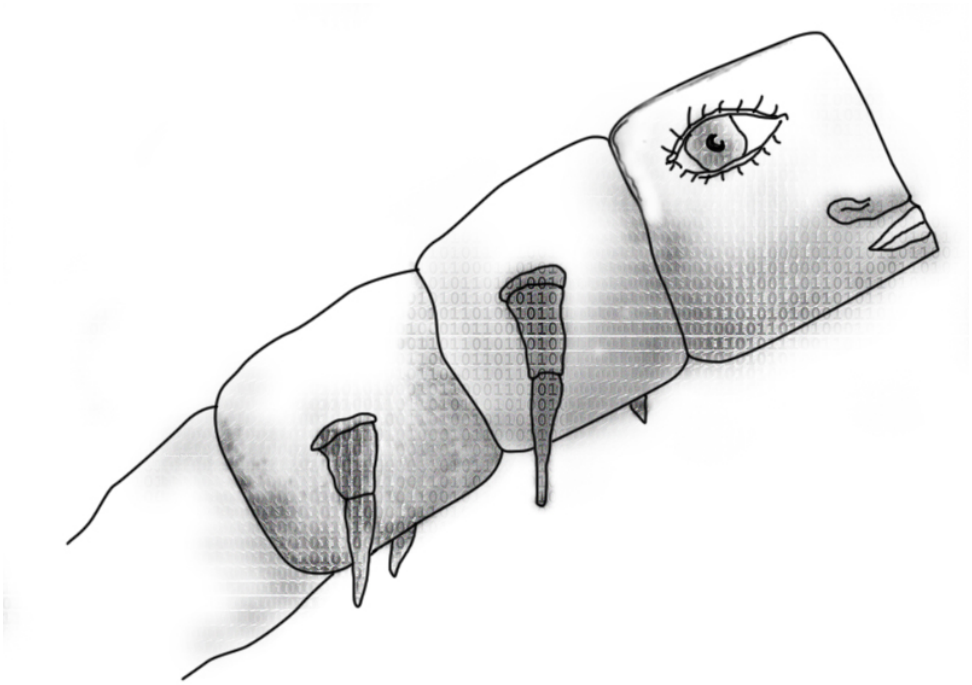
Keďže toto si asi nevyskúšate napísať naživo pre reálnu farmu počítačov, vašou úlohou je napísať preudokód a najmä dobrý popis riešenia a toho, ako bude výsledná tabuľka vyzerať.

**Příklad 4: Taxonomie havěti** (10 bodů)

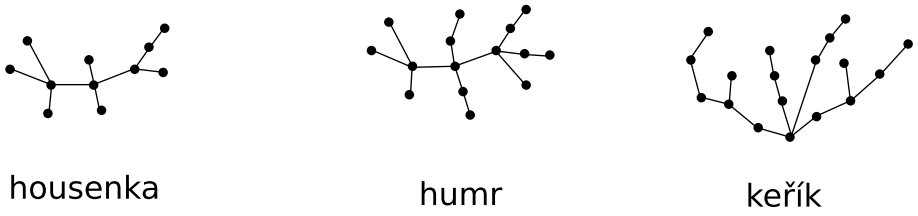
Po mořském dně pobíhá spousta acyklické grafové havěti. Na svých četných, ale krátkých nožkách kmitají mezi korálovými útesy housenky a za svou každodenní potravou se rozvážně posouvají i pestrobarevní humři. Sem tam je to ovšem jenom obyčejný korálový keřík.

V tom aby se vyznal leda tak odborník na havěť. Třeba Ty. Na vstupu dostaneš jednu acyklickou souvislou grafovou potvůrku ve tvaru seznamu sousedů pro každý vrchol. Tvým úkolem je rozhodnout, o koho se jedná. Na našem korálovém útesu žijí:

- **housenka** – housenka má páteř (cestu délky alespoň dvě hrany), ze které vyrůstají v libovolném počtu a z libovolných vrcholů páteře její nožičky (cesty délky jedna hrana)
- **humr** – humr má také páteř (cestu délky alespoň dvě hrany) a jeho nožičky jsou cesty délky jedna nebo dvě hrany. Pokud by graf mohl být podle této definice i humrem i housenkou (má jen krátké nožičky), pak je to housenka.



- **obyčejný korálový keřík** – nesplňuje podmínky ani na housenku ani na humra.




**Příklad 5: Vysokohorský problém** (10 bodů)

Istá veľká firma, agresívny finančný žralok, kúpila nádherné Veľhory. Keďže ide o pásmo z najvyššou ochranou, rozhodli sa, že tu vybudujú sieť hotelov, aby sem prilákali turistov a poriadne si naplnili vrecká. Avšak istá skupina nadšencov sa pripútavala o miestne porasty tak dlho, až boli do oblasti vyslaní kontrolóri s úlohou zistiť, či firma neničí toto chránené územie... Aspoň nie príliš výrazne.

Veľhory sú tvorené pozemkami, tie sú navzájom prepojené cestičkami. Tieto cestičky sa krížia iba na pozemkoch. Vo veľhorách je cestičiek pomerne málo a tak v miestnej cestnej sieti nie sú žiadne cykly.

Firma vynaložila značné finančné prostriedky ktoré oslabili pamäť kontrolórov natoľko, že si „pamätajú “iba jeden pozemok a to ten, z ktorého práve odišli. Kontrolóri sa pri kontrole náhodne pohybujú po cestičkách a pozemkoch Veľhôr, preto treba zabrániť tomu, aby sa stalo, že kontrolór príde na zastavaný pozemok z iného zastavaného pozemku.

Firma má teda problém a tak sa obracia na vás. Navrhnite pre firmu algoritmus, ktorý zistí, na ktorých pozemkoch môže firma postaviť svoje hotely tak, aby žiadne dva nesusedili priamo cestičkou, teda aby medzi každými dvoma hotelmi bol aspoň jeden nezastavaný pozemok. Samozrejme firma požaduje aby postavených hotelov bolo čo najviac.

---

A to je z třetí sady KSI vše. Přejeme ti hodně úspěchů při řešení, a když budeš mít jakékoliv otázky, neváhej se na nás obrátit e-mailem na adresu `ksi@fi.muni.cz` nebo v diskuzním fóru na webových stránkách.

**Termín odevzdání 3. sady úloh KSI: 6. 1. 2013**

`http://ksi.fi.muni.cz`