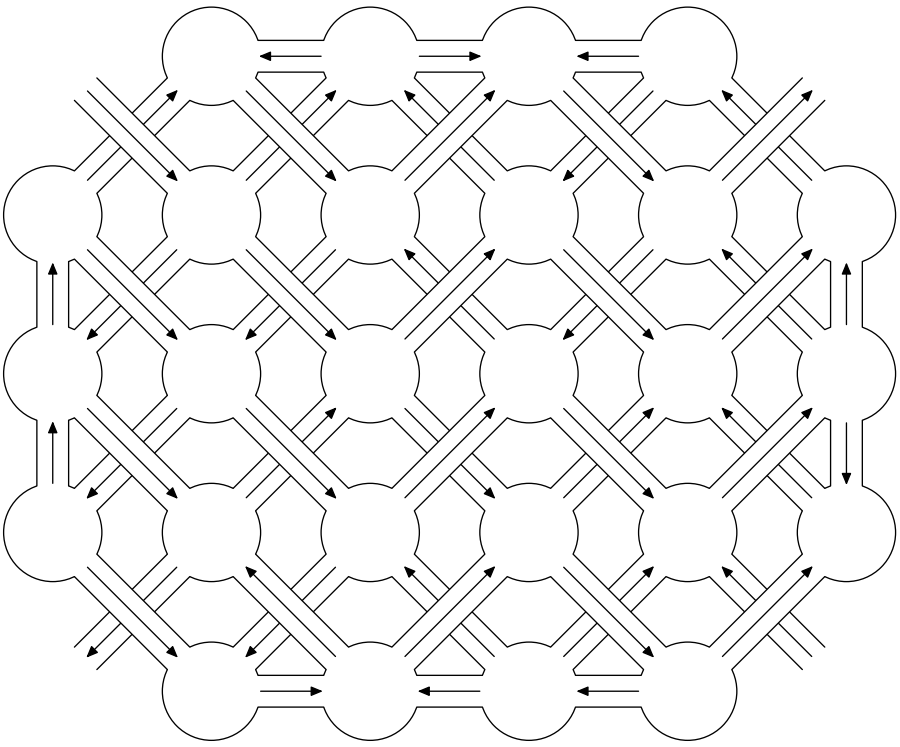
Kapitola 1

Průchod grafu



Na obrázku je speciální bludiště. Skládá se z několika místností, které jsou propojeny chodbami. Každá chodba se dá projít jen ve směru šipky, protože je zavřená dveřmi, které mají z jedné strany kliku a z druhé strany kouli. Dokážete projít bludištěm? Jestli se Vám to zdá jednoduché, tak zkuste tohle: Do 10 minut zjistěte, jak projít celé bludiště tak, abyste každou místnost navštívili právě jednou. Pokud to do 10 minut nestihnete, nebo některou místnost projdete dvakrát, tak spustíte alarm.

1.1 Efektivní průchod grafu

Průchod bludištěm je pro lidi výzvou už po několik století. Vzpomeňme si například na řecké báje a pověsti, na pověst o Mínotaurovi. Theseus chtěl zachránit Ariadnu a proto musel projít bludiště, najít Minutaura, zabít ho a zase se vrátit zpátky. Aby se v bludišti neztratil, dostal od Ariadny niť, pomocí které si označoval cestu. Když

postupoval do neznámých míst bludiště, tak niť odmotával. Když se vracel, tak niť namotával zpátky na klubko. Namotávání niťě ho vyvedlo zpátky před vchod do bludiště.

Asi každý by dokázal pomocí nitě a křídy projít bludiště, ale jak to naučit počítač?[1] Bludiště si v počítači reprezentujeme jako graf. Vrcholy, do kterých se lze dostat z výchozího místa nazveme *dosažitelné*. Pro jednoduchost předpokládejme, že všechny vrcholy bludiště jsou dosažitelné. Později si ukážeme, jak se tohoto předpokladu zbavit.

Efektivní průchod grafu musí splňovat následující body:

- každou hranou projdeme maximálně jednou (jednou tam a jednou zpět)
- hranou se vrátíme až když z vrcholu nevede další cesta
- hranou vedoucí do navštíveného vrcholu se ihned vrátíme

Algoritmus splňující výše uvedené body je konečný a korektní. Konečnost plyne z prvního bodu, protože v každém kroku projdeme po jedné hraně, každou hranu jen jednou a hran je jen konečně mnoho. Korektnost algoritmu v našem případě znamená, že projdeme všechny vrcholy a hrany, které jsou dosažitelné z výchozího vrcholu. Pokud by existoval algoritmem nenavštívený, ale jinak dosažitelný, vrchol v, tak se podívejme na cestu P vedoucí z výchozího vrcholu do v. Taková cesta musí existovat, protože vrchol v je dosažitelný. Protože začátek cesty P byl navštívený a konec nebyl, tak na cestě existuje neprošlá hrana vedoucí z navštíveného vrcholu do nenavštíveného. To je ale spor s druhým bodem.

Časová složitost algoritmu je $\mathcal{O}(n + m)$, protože čas za průchod naúčtujeme[2] hranám, které procházíme, a případnou práci ve vrcholech vrcholům.[3]

Efektivní průchod grafu má dvě možné implementace:

1. **Průchod do hloubky (DFS z anglického Depth First Search)** – podle tohoto algoritmu postupuje i Theseus. Před bludištěm si uváže jeden konec nitě na strom a vstoupí dovnitř. V prvním vrcholu si vybere jednu možnou cestu/hranu a projde po ní do dalšího vrcholu. Aby Theseus neměl zmatek v tom, které hrany už prošel, tak si všechny hrany, které prochází označuje křídou — a to na obou koncích. V každém vrcholu, do kterého Theseus dorazí, provede následující:

   - Pokud na zemi najde položenou niť, tak ví, že už ve vrcholu byl a že se do něj při namotávání nitě zase vrátí. Odloží tedy další prozkoumávání tohoto vrcholu na později, provede čelem vzad a začne namotávat niť na klubko. To ho dovede zpátky do předchozího vrcholu.
   - Pokud na zemi žádnou niť nenajde, tak se vydá první možnou neprošlou hranou. Pokud by taková hrana neexistovala, tak je vrchol zcela prozkoumán. V tom případě Theseus neztrácí čas a začne namotávat niť na klubko. Tím se dostane zpátky do předchozího vrcholu.

[1] Někteří lidé si myslí, že pro průchod zahradního bludiště stačí použít následující pravidlo pravé ruky: Při průchodu bludištěm se budeme pravou rukou neustále držet zdi. Takto podél zdi projdeme celé bludiště a zase se vrátíme na začátek. Mají pravdu, že se vrátíme zpátky na začátek bludiště, ale bohužel ho neprojdeme celé. Dovedete najít příklad bludiště, kde pravidlo pravé ruky selže?

[2] Viz počítání amortizované časové složitosti v kapitole **??**.

[3] Práce ve vrcholech se může vykonávat při prvním vstupu do vrcholu, mezi návratem z jedné hrany a odchodem do další hrany nebo při posledním opouštění vrcholu. Podle toho označujeme práci ve vrcholu jako *preorder, inorder* a *postorder*. Čas za práci preorder a postorder naúčtujeme vrcholům a čas za práci vykonávanou inorder hranám, které následně projdeme.

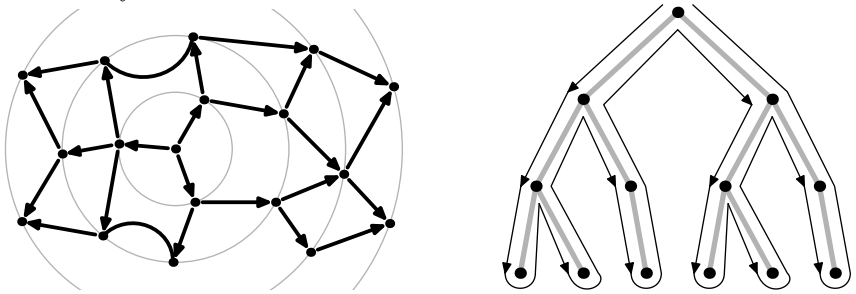
Tímto postupem prozkoumá celé bludiště a nakonec se vrátí do výchozího vrcholu.

Jak to provést v počítači? Křídu potřebujeme proto, abychom se nezacyklili. Abychom každou hranu prošli jen jednou. Místo křídy si pro každou hranu zřídíme proměnnou označující, jestli jsme hranu prošli. Klubíčko a niť potřebujeme proto, abychom našli cestu ven z bludiště. Položená niť nám vyznačuje cestu z výchozího vrcholu do aktuálního vrcholu. V počítači nám bude úplně stačit, když si tuto cestu budeme pamatovat jako posloupnost vrcholů na této cestě. Pro uložení této cesty použijeme zásobník. Odmotávání nitě z klubka při postupu do dalšího vrcholu odpovídá přidání vrcholu na zásobník. Namotávání nitě na klubko při návratu zpět odpovídá odebrání vrcholu ze zásobníku.

Stejně postupuje i *backtracking* (to je ta metoda „Hrr, na ně! A když to nepůjde, tak coufnem."). Backtracking je metoda hrubé síly, která vyzkouší všechny možnosti.

2. **Průchod do šířky (BFS z anglického Breadth First Search)** – tento průchod (prohledání grafu) si můžeme představit tak, že se do výchozího vrcholu postaví miliarda Číňanů a všichni naráz začnou prohledávat graf. Když se cesta rozdělí, tak se rozdělí i dav řítící se hranou. Předpokládáme, že všechny hrany jsou stejně dlouhé. Graf prozkoumáváme „po vlnách". V první vlně se všichni Číňané dostanou do vrcholů, do kterých vede z výchozího vrcholu hrana. V druhé vlně se dostanou do vrcholů, které jsou ve vzdálenosti 2 od výchozího vrcholu. Podobně v k-té vlně se všichni Číňané dostanou do vrcholů ve vzdálenosti k od výchozího vrcholu. Kvůli těmto vlnám se někdy průchodu do šířky říká *algoritmus vlny.* V počítači vlnu nasimulujeme tak, že při vstupu do nového vrcholu uložíme všechny možné cesty do fronty. Frontu průběžně zpracováváme.

Na následujícím obrázku nalevo je znázorněn průchod do šířky a napravo průchod do hloubky.



Který průchod grafu je lepší? Jak kdy a jak na co. To záleží na tom, jak vypadá graf, který prohledáváme. Někdy je výhodnější jedna metoda o proti druhé, protože budeme potřebovat výrazně méně paměti na uložení zásobníku/fronty. Jindy je to naopak.

Příklad: Dostaneme bludiště, které má tvar čtvercové sítě o rozměrech $n \times n$, a jsou v něm pouze obvodové zdi. Je to takové fotbalové hřiště s mantinely. Někde v bludišti jsme ztratili míč, protože se udělala hrozná mlha. V mlze je vidět jen o malinko víc než na jedno políčko bludiště. Chtěli bychom projít celé bludiště, navštívit každé políčko, a míč najít.[4] Vchod do bludiště je pravém horním rohu.

Při použití DFS, ve kterém z každého prozkoumávaného políčka zkoušíme cesty v pořadí nahoru, doprava, dolů a doleva, bude neustále možné pokračovat dále. A

[4] Úlohu můžeme přeformulovat i tak, že chceme obarvit všechna políčka bludiště tím, že na první políčko vylijeme plechovku barvy. Barva se rozlije po všech dostupných políčkách. Proto tomuto způsobu barvení říkáme záplavové vyplňování (floodfill).

to tak dlouho, dokud nezaplníme skoro všechna políčka. Jednoduše řečeno, průchod bludištěm bude vypadat jako dlouhá cesta naskládaná zig-zag po celém bludišti. (Pokud to není jasné, tak si zkuste průchod odkrokovat.) Všechna políčka budou na zásobníku a proto budeme potřebovat zásobník velikosti $\mathcal{O}(n^2)$. Při použití BFS budeme bludiště procházet v soustředných vlnách a bude nám stačit fronta velikosti $4n$ (nejvýše obvod bludiště). Ve fotbalové analogii to odpovídá vytvoření rojnice.

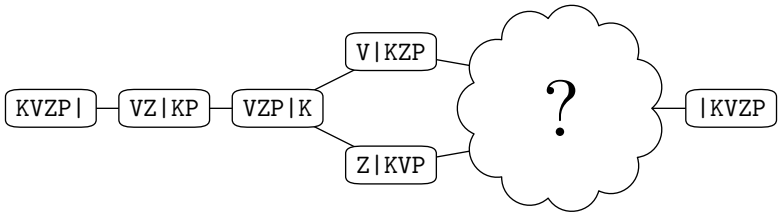
Takto se zdá, že z paměťových důvodů je lepší použít BFS. Toto zdání ale klame. Podívejme se na další příklad.

Příklad: Jsme na dovolené a bydlíme v hotelu. Hotel má 5 pater, do kterých se dostaneme výtahem. Na každém patře je dlouhá chodba se dveřmi do 20 pokojů. Potřebovali bychom zjistit, na kterém pokoji bydlí ta hezká slečna, co jsme ji viděli u snídaně, ale nevíme jak se jmenuje. Rozhodneme se proto projít celý hotel a slečnu najít. Potřebujeme projít všechny místnosti hotelu a v každé nechat vzkaz. Pokud použijeme DFS, vystačíme si se zásobníkem velikosti 3 (výtah, chodba, pokoj). Při použití BFS budeme potřebovat tak velkou frontu, kolik je v hotelu pokojů, tedy 100.

Výhodnou BFS je, že prochází vrcholy v pořadí podle nejkratší vzdálenosti od počátku. Pokud tedy hledáme nejkratší cestu, nejmenší počet tahů apod., tak je výhodnější použít BFS.

Příklad: (Koza, vlk a zelí) Na břehu řeky má převozník kozu, vlka a zelí. Do loďky se mu vejde jen jedno zvíře nebo zelí. Jak má převézt kozu, vlka a zelí na druhý břeh, když na žádném břehu nesmí nechat nehlídaného vlka s kozou (vlk by sežral kozu) ani kozu se zelím (koza by snědla zelí)?

Vrcholy grafu jsou stavy které označují, kdo je na kterém břehu. Například `VZ|KP` značí, že na prvním břehu zůstal vlk se zelím a na druhém břehu převozník s kozou. Přechod po hraně grafu odpovídá převozu zvířete, zelí nebo přejezdu převozníka na protější břeh. Například na začátku jsme se převozem kozy dostali ze stavu `KVZP|` do `VZ|KP`. Řešení úlohy lze najít průchodem tohoto grafu a nalezením cesty do vrcholu `|KVZP`.



Při hledání dalších stavů a kreslení obrázku nevidíme, jak to bude vypadat dál. Chová se to jako opravdové bludiště. Takovéto úlohy se vyskytují celkem často. Graf je „skrytý“ a odpovídá možným větvením programu.

Prohledáváme takzvaný *„stavový prostor“* programu. Pozor, často se necháme zmýlit tím, co je „stav“. Stav zachycuje celou situaci. V předchozí úloze nebylo stavem jen to zvíře, které převážíme přes řeku (koza nebo vlk či zelí). Stav musí obsahovat polohu všech zvířátek a zelí, včetně převozníka.

Podobnou úlohou je proskákání šachovnice šachovým koněm tak, abychom každé políčka navštívili právě jednou. V této úloze není stavem jen poloha naposledy položeného koně a počet umístěných koní. Stavem je „fotka“ celé šachovnice. To je poloha všech dosud umístěných koní.

V řadě úloh nám ujasnění si toho, co je stav, pomůže k nalezení řešení.

1.2 DFS na neorientovaném grafu

V implementaci DFS můžeme místo zásobníku využít rekurze. Ta je ve skutečnosti implementována zase pomocí zásobníku, ale je příjemnější pro programátora. Abychom si ulehčili výklad, tak budeme během průchodu grafu barvit vrcholy. Vrcholy, které jsme ještě nenavštívili budou mít bílou barvu. Vrcholy, které zpracováváme (už jsme se do nich dostali, ale ještě jsme neprozkoumali všechny cesty z nich vedoucí) barvu šedou a hotové vrcholy barvu černou. Šedé jsou přesně ty vrcholy, které jsou na zásobníku.

V polích `in[v]` a `out[v]` si budeme pro každý vrchol pamatovat čas, kdy jsme do něj poprvé vstoupili a kdy jsme ho nadobro opustili.

Při praktickém použití není nutné rozlišovat všechny tři barvy, ani si pamatovat `in[v]` a `out[v]`. Stačí, když budeme schopni rozlišit, které vrcholy jsme už navštívili a které ještě ne.

Graf si reprezentujeme seznamem sousedů, takže seznam `sousedi[v]` je seznam sousedních vrcholů vrcholu v.

```
Projdi(v):
    color[v]=GREY
    time=time+1
    in[v]=time
    for each w ∈ Sousedi[v] do
        if color[w]=WHITE then
            Projdi(w)
    color[v]=BLACK
    time=time+1
    out[v]=time
```

Procedura `Projdi(v)` projde tu část grafu, které je dosažitelná z výchozího vrcholu. Pokud by graf měl několik oddělených částí, tak musíme do každé části převézt Thesea helikoptérou. Postupně v každé části zvolíme výchozí místo a z něj graf projdeme. To obstará následující procedura DFS.

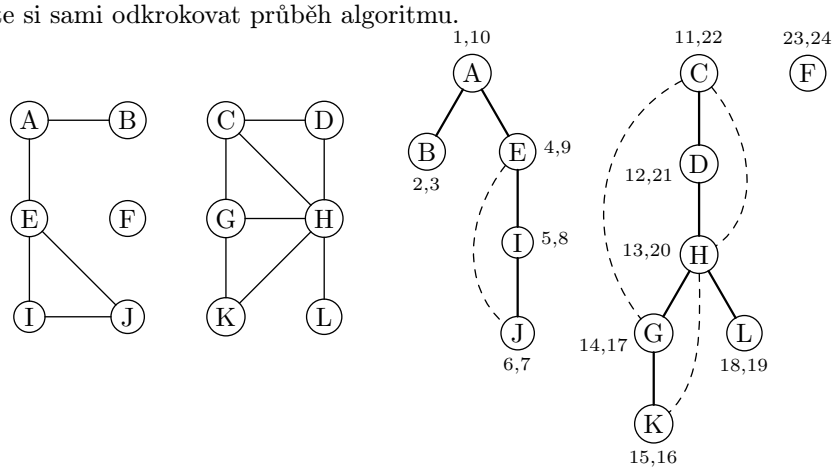
```
DFS():
    ∀v ∈ V : color[v]=WHITE
    time=0
    for each v ∈ V do
        if color[w]=WHITE then
            Projdi(w)
```

Podívejme se nejprve na tu část grafu, kterou jsme prošli procedurou `Projdi(v)`. Výchozí vrchol v nazveme *kořen*. Hrany, po kterých jsme v průběhu algoritmu prošli do nenavštíveného bílého vrcholu tvoří strom. Proto tyto hrany nazveme *stromové* a jim odpovídající strom nazveme *DFS stromem* (stromem průchodu do hloubky). Ostatním hranám budeme říkat *zpětné,* protože vedou zpět do již navštívených vrcholů. Stromy průchodu všech oddělených částí dohromady tvoří *DFS les.*

Pokud při prozkoumávání vrcholu v projdeme stromovou hranou do vrcholu u, tak řekneme, že u je *synem* v. Naopak v je *otcem* u. Všechny synové vrcholu v, synové synů a tak dále jsou *potomci* vrcholu v. Obráceně se dá říci, že v je jejich *předchůdce.* Někdy místo potomci říkáme *následníci.* Ještě jednou a přesněji, u je následník vrcholu v v zakořeněném stromě, pokud v leží na jednoznačné cestě z kořene do u.

Na obrázku budeme syny každého vrcholu kreslit pod jejich otce a v pořadí zleva doprava tak, jak jsme na ně při průchodu do hloubky narazili.

Příklad: Na následujícím obrázku vlevo je graf G, který projdeme pomocí DFS. Pokaždé, když si algoritmus můžete vybrat z několika vrcholů, tak vybereme abecedně nejmenší vrchol. Na obrázku vpravo je strom průchodu grafu G do hloubky. Čísla u každého vrcholu v jsou hodnoty `in[v]`, `out[v]`, tj. časy příchodu a odchodu. Zkuste si sami odkrokovat průběh algoritmu.



1.3 Komponenty souvislosti

Motivace: Máme ohromné bludiště, které vzniklo tak, že někdo v ohromné hale postavil z cihel spoustu zdí. Do haly vede několik vchodových dveří. Chtěli bychom zjistit, kolik oddělených bludišť v hale je (několik vchodů může vést do stejného bludiště). Dále chceme zjistit, kolik nejméně zdí musíme probourat, abychom dostali jedno velké bludiště.

Definice: Graf je *souvislý*, pokud mezi každýma dvěma vrcholy vede cesta. Pokud graf není souvislý, tak se skládá z jednotlivých částí, které už souvislé jsou. Těmto částem budeme říkat *komponenty souvislosti.* Jinak řečeno, komponenta souvislosti je maximální souvislý podgraf.

Pro nalezení komponent souvislosti použijeme průchod do hloubky s malou úpravou. Víme, že procedura `Projdi(`v`)` projde všechny vrcholy dosažitelné z v tj. vrcholy komponenty obsahující vrchol v. Proto stačí, když si při tomto průchodu budeme vrcholy označovat číslem komponenty.

Časová složitost nalezení komponent souvislosti stejná jako časová složitost DFS a to je $\mathcal{O}(n + m)$.

1.4 Komponenty 2-souvislosti

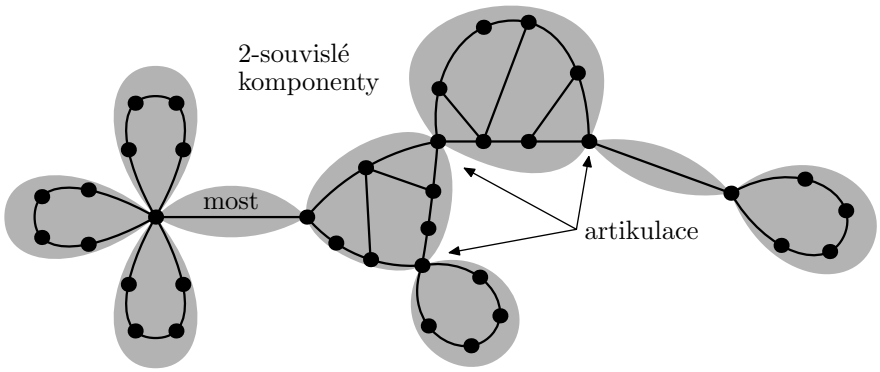
Motivace: Představme si situaci za války, kdy partyzáni znemožňovali postup vojsk tak, že vyhodili do povětří vhodný most a tím přerušili silniční spojení. Dostaneme graf silniční sítě a ptáme se, jestli v ní existuje vrchol (křižovatka) nebo hrana (most) taková, že se po jejím vyhození graf rozpadne na několik komponent.

Pokud se Vám zdá motivace příliš zastaralá, tak místo partyzánů představte teroristy a místo silniční sítě síť počítačovou.

Definice: Graf je 2-*souvislý*, pokud po vyhození libovolného vrcholu zůstane souvislý. Platí věta, že graf je 2-souvislý právě tehdy, když každé dva vrcholy leží na společné kružnici. 2-*souvislá komponenta* je maximální 2-souvislý podgraf. *Artikulace* je vrchol, po jehož vyhození se graf rozpadne na komponenty souvislosti.

Podobně *most* je hrana, po jejímž vyhození se graf rozpadne. Most je také 2-souvislá komponenta.

Na následujícím obrázku jsou 2-souvislé komponenty znázorněny šedými obláčky. Jedna artikulace může patřit do několika 2-souvislých komponent.



Úkol: Dostaneme graf a chtěli bychom najít všechny jeho artikulace, případně vypsat 2-souvislé komponenty.

Triviálním řešením je postupně zkusit vyhodit každý vrchol a testovat souvislost výsledného grafu. To by nám ale zabralo čas $\mathcal{O}(n(n+m))$. Podívejme se na lepší řešení pomocí DFS:

Pozorování 1 *v je artikulace $\iff$ v je kořen DFS stromu a má alespoň dva potomky nebo není kořen a má syna w takového, že z žádného jeho potomka (včetně w) nevede zpětná hrana do předchůdce v.*

Přípustná cesta je cesta vedoucí po stromových hranách směrem od kořene, která může končit jedním skokem zpět po zpětné hraně. Pro každý vrchol grafu spočítáme funkci $\texttt{LOW}(v) = \min\{\texttt{in}[w] \mid \text{z } v \text{ do } w \text{ vede přípustná cesta}\}$. Funkci $\texttt{LOW}(v)$ můžeme počítat už při průchodu do hloubky, protože při opuštění vrcholu v známe všechny potřebné hodnoty `in[w]`. Funkci spočítáme jako $\texttt{LOW}(v) = \min\{x, y, z\}$, kde $x = \texttt{in}[v]$, $y = \min\{\texttt{in}[w] \mid vw \text{ je zpětná hrana}\}$ a $z = \min\{\texttt{LOW}[w] \mid vw \text{ je stromová hrana}\}$.

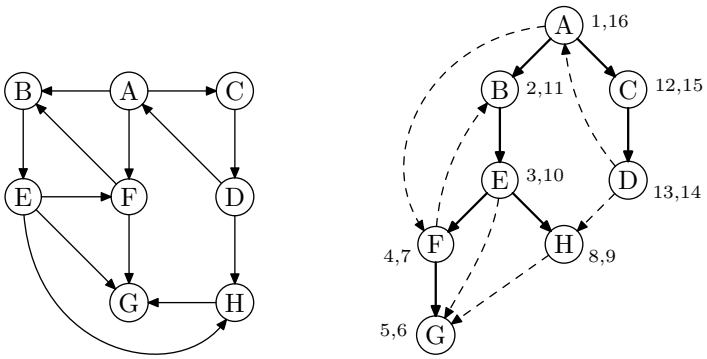
Předchozí pozorování můžeme přeformulovat pomocí funkce `LOW`.

Pozorování 2 *v je artikulace $\iff$ v je kořen DFS stromu a má alespoň dva potomky nebo není kořen a má syna w s* $\texttt{LOW}(w) \geq \texttt{in[v]}$

Na základě pozorování poznáme už při průchodu grafu do hloubky, které vrcholy jsou artikulace. Dodejme, že artikulace nemůžeme vypisovat rovnou, protože bychom mohli některé vypsat dvakrát. Musíme si u každého vrcholu pamatovat, jestli je artikulace, a výsledek vypsat až nakonec. Pokud bychom chtěli vypsat i hrany každé 2-souvislé komponenty, tak stačí během DFS ukládat procházené hrany na druhý zásobník. Při návratu do artikulace vypíšeme všechny hrany, které přibyly na zásobníku od posledního opuštění artikulace (stačí vhodně uříznout vršek zásobníku).

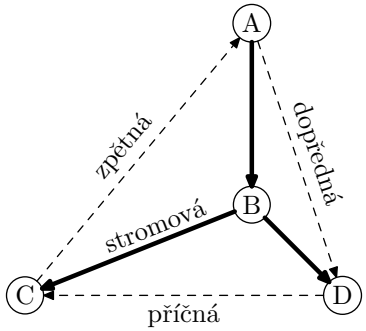
1.5 DFS na orientovaném grafu

Průchod do hloubky bude beze změny fungovat i pro orientované grafy. Na následujícím obrázku je nalevo orientovaný graf a napravo jeho strom průchodu do hloubky. Opět, pokud jsme během DFS měli na výběr z několika vrcholů, tak jsme si vybrali abecedně nejmenší vrchol.



V případě neorientovaných grafů jsme rozlišovali mezi stromovými a zpětnými hranami. U orientovaných grafů budeme muset rozlišit více případů nestromových hran. Typ hrany určíme ve chvíli, kdy hranu procházíme pomocí DFS a to podle barvy vrcholu, do kterého hrana vede, případně podle časů příchodu a odchodu z koncových vrcholů hrany.

- **stromové hrany** – vedou z právě prozkoumávaného vrcholu do nenavštíveného (bílého) vrcholu; tvoří DFS strom.
- **zpětné hrany** – vedou z právě prozkoumávaného vrcholu do šedivého vrcholu, tj. do předchůdce v DFS stromě.
- **dopředné hrany** – vedou z právě prozkoumávaného vrcholu do černého vrcholu v témže podstromě, tj. do svého potomka.
- **příčné hrany** – vedou z právě prozkoumávaného vrcholu do černého vrcholu v jiném podstromě, tj. mezi dvěmi různými podstromy téhož grafu.



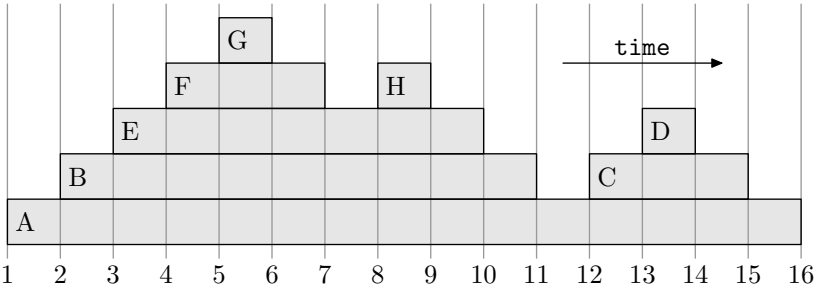
Hloubavý čtenář vidí, že barvy vrcholů ve skutečnosti nepotřebujeme a že jsme schopni barvu vrcholu určit pomocí hodnot $\texttt{in}[v]$, $\texttt{out}[v]$. Lepší charakterizaci hran při DFS dává následující pozorování.

Pozorování 3 *Intervaly $(in[v], out[v])$ pro všechny $v \in V$ tvoří dobré uzávorkování. To znamená, že dva intervaly jsou buď disjunktní a nebo je jeden podmnožinou druhého. Pro každou hranu $uv \in E$ platí jedna z následujících možností:*

- *uv je stromová nebo dopředná hrana $\iff (in[v], out[v]) \subseteq (in[u], out[u])$*
- *uv je zpětná hrana $\iff (in[u], out[u]) \subseteq (in[v], out[v])$*
- *uv je příčná hrana $\iff in[v] < out[v] < in[u] < out[u]$ (intervaly jsou disjunktní)*

Interval $(in[v], out[v])$ znázorňuje dobu, po kterou byl vrchol v uložen na zásobníku. Vlastnost korektního uzávorkování plyne přímo z toho, jak pracujeme se zásobníkem. Vrchol neopustíme a neodebereme ze zásobníku dříve, než jsou zpracováni všichni jeho potomci.

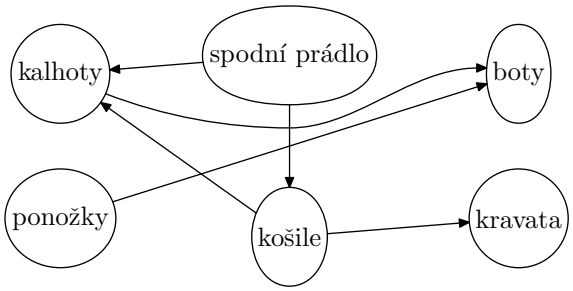
Průběh zaplnění zásobníku pro graf ze začátku sekce 1.5 je na následujícím obrázku. Stavu zásobníku odpovídá jen jeden sloupeček. Obrázek zachycuje stav zásobníku v různých časech. Vrchol X je vložen na zásobník v čase $\texttt{in}[X]$ a je ze zásobníku odebrán v čase $\texttt{out}[X]$.



Zkuste si rozmyslet, v jakém vztahu jsou intervaly $(in[u], out[u])$ a $(in[v], out[v])$, pokud je u následníkem v.

1.6 Topologické uspořádání

Motivace: Pomocí orientovaného grafu snadno znázorníme závislosti. Orientovanou hranou (šipkou) mezi úkoly vyjádříme, že druhý úkol můžeme začít provádět až po skončení prvního úkolu. Například když se chceme ráno obléknout, tak si ponožky musíme obléci dříve než boty. Dostaneme seznam úkolů a závislosti mezi nimi. Zajímalo by nás, v jakém pořadí úkoly vykonávat, abychom neporušili závislosti. Zkuste úlohu vyřešit pro následující orientovaný graf.

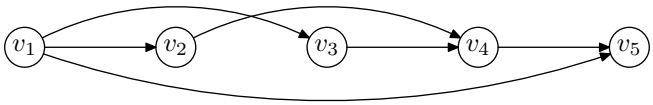


Jiným příkladem je uspořádání látky probírané na přednáškách tak, aby student nejprve slyšel všechny základy a potřebné ingredience a teprve pak si poslechl navazující výsledky.

Podobně funguje Makefile – soubor se závislostmi pro unixový program `make`. Často, když chceme ze vstupních souborů dostat požadované výstupy, musíme spustit více programů nebo utilit. Vstupem jednoho programu může být výstup z jiného programu. To určuje závislosti mezi použitými programy. Abychom dostali správný výstup, musíme programy pouštět ve správném pořadí. Závislosti napíšeme do souboru Makefile. Podle něj `Make` spustí všechny použité programy ve správném pořadí. Při dalším volání, `make` nespouští vše znova, ale podívá se, které vstupy i mezivýsledky se změnily. Potom spustí pouze ty programy, které přepočítají výstupy ke změněným vstupům. To je hlavní výhoda `Make`, která často výrazně zrychlí přepočítávání.

Proč se tomu říká topologické uspořádání? Šipky mezi vrcholy můžeme představit jako relaci '$\succeq$'. Potom hledáme lineární uspořádání '$\geq$', které je rozšířením '$\succeq$'. To znamená, že když $x \succeq y$, pak i $x \geq y$.

Definice: Topologické uspořádání vrcholů orientovaného grafu je seřazení vrcholů do řady v_1, v_2, v_3, ..., v_n tak, že každá hrana vede zleva doprava. Jinými slovy je to takové očíslování vrcholů čísly 1 až n takové, že každá hrana vede z vrcholu s nižším číslem do vrcholu s vyšším číslem.



Úkol: Pro zadaný orientovaný graf nalezněte topologické uspořádání. Přeskládávání vrcholů do topologického uspořádání se někdy říká *topologické třídění.*

Definice: *Orientovaný cyklus* je posloupnost $v_0 \to v_1 \to v_2 \to \cdots \to v_{n-1} \to v_n$ taková, že $v_0 = v_n$ a $v_{i-1}v_i$ je orientovaná hrana pro každé $i \in \{1, \ldots, n\}$. Orientovaným grafům, které neobsahují orientovaný cyklus, říkáme *orientované acyklické grafy.*[5]

Pozorování 4 *Graf G obsahuje orientovaný cyklus $\iff$ DFS najde zpětnou hranu.*

Důkaz: Jeden směr je jednoduchý, pokud je uv zpětná hrana, tak společně s cestou z v do u v DFS stromě tvoří cyklus. Na druhou stranu pokud graf obsahuje orientovaný cyklus, tak označme pomocí r první vrchol cyklu, který byl nalezen při průchodu DFS. Všechny ostatní vrcholy cyklu jsou potomky r, protože do nich z r vede orientovaná cesta. Proto je hrana cyklu, vedoucí do r, zpětná. ∎

Lemma 1 *Topologické uspořádání orientovaného acyklického grafu nalezneme pomocí DFS tak, že každý vrchol vypíšeme v momentě, kdy ho opouštíme. Na závěr výslednou posloupnost obrátíme. Jinými slovy uspořádání vrcholů podle klesajících časů* `out[`v`]` *je topologické.*

Důkaz: K důkazu předchozího tvrzení stačí ukázat, že je ve výsledném pořadí každá hrana zorientována správně. Nechť ij je libovolná orientovaná hrana. Podle klasifikace hran z pozorování 3 je hrana ij buď

- stromová nebo dopředná a potom $\texttt{out}[i] > \texttt{out}[j]$, nebo je
- příčná, ale pak také $\texttt{out}[i] > \texttt{out}[j]$. Hrana ij by ještě mohla být
- zpětná, to ale není podle předchozího pozorování možné, protože by graf obsahoval orientovaný cyklus. ∎

Rozpoznávání acykličnosti a hledání topologického uspořádání můžeme spojit do jednoho průchodu DFS. Budeme hledat topologické uspořádání pomocí průchodu do hloubky. Pokud objevíme zpětnou hranu, tak můžeme skončit a odpovědět, že topologické uspořádání neexistuje. V opačném případě topologické uspořádání najdeme. Časová složitost nalezení topologického uspořádání je $\mathcal{O}(n + m)$.

Poznamenejme na závěr, že topologické uspořádání je pro práci s acyklickými grafy (DAG, directed acyclic graph) stejně důležité, jako třídění pro práci s polem. Řada pěkných algoritmů v acyklických grafech vyžaduje topologicky uspořádané vrcholy. Aplikace najdte ve cvičeních na konci kapitoly.

Alternativním algoritmem pro nalezení topologického uspořádání může být postupné odtrhávání zdrojů (stoků) grafu a jejich vypisování na výstup. *Zdroj* je vrchol, do kterého nevede žádná hrana. Hrany z něj pouze vychází (vytékají). Naopak do *stoku* hrany pouze vedou a žádná hrana z něj nevede ven. Zkuste si rozmyslet, proč tento alternativní algoritmus funguje a jaká může být jeho nejrychlejší implementace.

Který algoritmus byste si vybrali pro řešení úlohy na papíře?

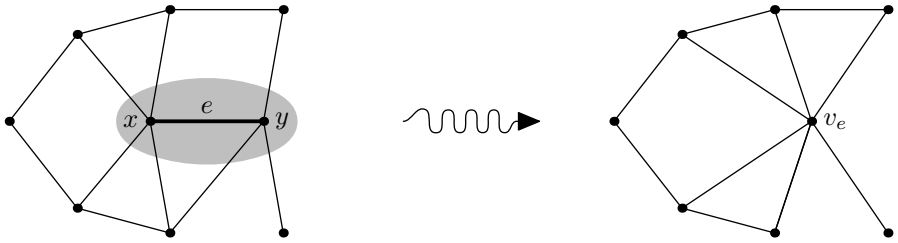
[5] Anglicky se označují jako DAG (Directed Acyclic Graph).

1.7 Intermezzo o kontrakcích

Začneme s kontrakcí hrany v neorientovaném grafu.

Nechť $e = xy$ je hrana grafu $G = (V, E)$. Kontrakci hrany e si můžeme představit tak, že se hrana e zkracuje a zkracuje, až už je tak krátká, že vrcholy x a y splynou v jeden vrchol. Pokud by vznikly násobné hrany nebo smyčky, tak je vyhodíme.

Přesněji řekneme, že graf $G.e$ vznikne kontrakcí hrany e do nového vrcholu v_e tak, že z grafu G vymažeme vrcholy x, y i s hranami, které z nich vedou, a naopak přidáme nový vrchol v_e, který spojíme se všemi zbylými vrcholy grafu G, se kterými byl spojen vrchol x nebo y. Formálněji $G.e = (V', E')$, kde $V' = V \setminus \{x, y\} \cup \{v_e\}$ a $E' = \{uv \in E \mid \{u, v\} \cap \{x, y\} = \emptyset\} \cup \{v_e w \mid xw \in E \setminus \{e\} \text{ nebo } yw \in E \setminus \{e\}\}$.



Nechť $W \subseteq V$ je podmnožina vrcholů taková, že $G[W]$ je souvislý podgraf.[6] Kontrakci množiny W do jednoho nového vrcholu v_W si opět můžeme představit tak, že všechny vrcholy W přibližujeme k sobě natolik, že splynou v jeden vrchol. Opět pokud by vznikly násobné hrany nebo smyčky, tak je vyhodíme.

Přesněji řekneme, že graf $G.W$ vznikne postupnou kontrakcí všech hran mezi vrcholy W. Kontrakce hrany $e = xy$ je v podstatě kontrakcí dvouprvkové množiny vrcholů $\{x, y\}$.

V orientovaných grafech kontrakce funguje stejně a zachovává směr hran.

1.8 Silně souvislé komponenty

Motivace: V jednom (raději) nejmenovaném městě, se starosta rozhodl, že ze všech ulic udělá jednosměrky. Odůvodnil to vznikem většího počtu parkovacích míst. Po jisté době se ale začalo proslýchat, že se z určitých míst nedá dojet do jiných míst ve městě jinak, něž porušením předpisů. Pokud by se to prokázalo, tak by s toho mohl mít starosta průšvih. Rád by to zkontroloval a případně napravil. Protože se rychle blíží volební období, tak už není moc času a nemůžete použít jiný algoritmus, než s lineární časovou složitostí. Znáte vhodný algoritmus, který ověří, že se dá dostat z kteréhokoliv místa ve městě kamkoliv?

Neorientovaný graf G je souvislý, pokud mezi každou dvojicí vrcholů existuje cesta. Jinými slovy, z každého vrcholu $u \in V(G)$ se dostaneme do libovolného jiného vrcholu $v \in V(G)$ po nějaké cestě. Graf G můžeme rozložit na komponenty souvislosti. To jsou největší „kusy“ (podgrafy) grafu G, které jsou souvislé.

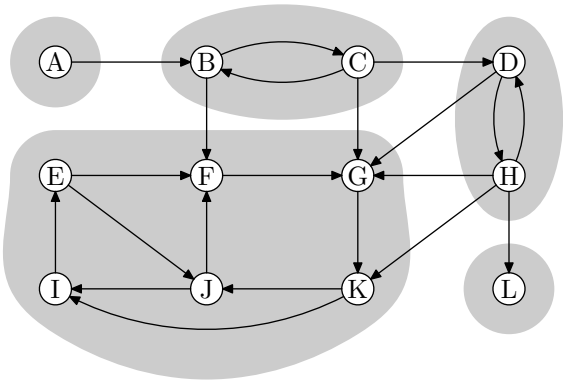
Ale co je to souvislost v orientovaném grafu? Po šipkách můžeme projít jen jedním směrem. Opět bychom chtěli zavést „souvislost“ tak, aby se bylo možné po šipkách dostat z libovolného vrcholu $u \in V(G)$ do libovolného vrcholu $v \in V(G)$. Tentokrát se ale může stát, že orientovaná cesta z u do v povede jinudy než cesta z v do u.

[6] Připomeňme, že $G[W]$ značí podgraf grafu G indukovaný vrcholy W.

Definice: Dva vrcholy $u, v \in V(G)$ jsou spolu *silně propojeny*, pokud existuje orientovaná cesta z u do v a i orientovaná cesta z v do u. Orientovaný graf G je *silně souvislý*, pokud je každá dvojice vrcholů $u, v \in V(G)$ silně propojena.

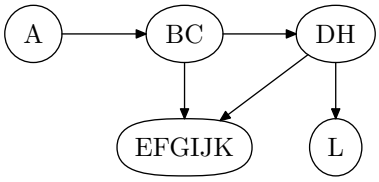
Relace být silně propojen je relace ekvivalence. Třídám této ekvivalence budeme říkat *silně souvislé komponenty.* Jinými slovy, silně souvislá komponenta je maximální podgraf grafu G, který je silně souvislý.

Příklad:



V grafu na obrázku máme 5 silně souvislých komponent.

Kontrakcí každé silně souvislé komponenty do jednoho meta-vrcholu dostaneme meta-graf.



Pozorování 5 *Meta-graf vzniklý kontrakcí silně souvislých komponent je acyklický orientovaný graf.*

Důkaz: Kdyby obsahoval orientovanou kružnici, tak všechny meta-vrcholy na kružnici tvoří jednu silně souvislou komponentu. To je spor s tím, že jsme všechny silně souvislé komponenty zkontrahovali do meta-vrcholů. ∎

Tato struktura orientovaných grafů nám umožňuje topologicky uspořádat silně souvislé komponenty. Silně souvislé komponentě, která odpovídá zdroji/stoku meta-grafu, budeme říkat zdrojová/stoková silně souvislá komponenta.

Při realizaci DFS jsme používali proceduru `Projdi(v)`. Ta projde právě všechny vrcholy, které jsou dostupné z vrcholu v. Pokud pustíme `Projdi(v)` na vrchol, který leží v silně souvislé komponentě odpovídající stoku meta-grafu, tak se dostaneme právě do všech vrcholů této silně souvislé komponenty.

Konkrétně na výše uvedeném příkladu: pokud v G pustíme `Projdi(E)` na vrchol E, tak projdeme právě vrcholy silně souvislé komponenty obsahující $\{E, F, G, I, J, K\}$. Nešlo by toho nějak využít? Ano, ale budeme muset vyřešit dva problémy.

(A) Jak najít vrchol, který určitě leží v silně souvislé komponentě, která je stokem?

(B) Jak pokračovat dále po té, co najdeme první silně souvislou komponentu?

Začneme s problémem (A). Vrchol, který je ve stokové silně souvislé komponentě se přímo najít nedá. Co se ale dá snadno najít, je vrchol, který leží ve zdrojové silně souvislé komponentě.

Pozorování 6 *Vrchol, který při DFS průchodu grafu G dostane největší opouštěcí čas* `out`$[\,\cdot\,]$*, leží ve zdrojové silně souvislé komponentě grafu G.*

Pozorování přímo plyne z následujícího obecnějšího pozorování.

Pozorování 7 *Nechť* `out`$[v]$ *jsou opouštěcí časy vrcholů při DFS průchodu grafu G. Pokud C_1 a C_2 jsou dvě silně souvislé komponenty takové, že z C_1 vede hrana do C_2, tak potom největší hodnota* `out`$[]$ *v první komponentě je větší než největší hodnota* `out`$[\,\cdot\,]$ *ve druhé komponentě.*

Důkaz: Mohou nastat dva případy. V prvním případě DFS navštíví komponentu C_2 jako první. Potom v ní ale zůstane, dokud ji celou neprozkoumá (to je vlastnost procedury `Projdi`()). Teprve pak se DFS dostane do C_2.

Ve druhém případě DFS nejprve navštíví C_1. Nechť v je vrchol, který byl v C_1 navštíven jako první. DFS opustí vrchol v, až když prozkoumá všechny vrcholy, které jsou z v dosažitelné a které nebyly dosud navštíveny. Proto nejprve projde celou komponentu C_2 a pak se teprve naposledy vrátí do v. ∎

Pozorování 7 vlastně říká, že sestupné uspořádání silně souvislých komponent podle jejich největšího čísla `out`$[\,\cdot\,]$ je topologické.

Pozorování 6 nám ukazuje, jak najít vrchol, který leží ve zdrojové silně souvislé komponentě. To je přesně naopak, než potřebujeme! Potřebujeme vrchol ležící ve stokové silně souvislé komponentě. Nevadí, vrchol budeme hledat v grafu G^R. To je v grafu, který vznikne z G tak, že obrátíme všechny hrany. Graf G^R má stejné silně souvislé komponenty jako graf G. Rozmyslete si proč.

Pustíme DFS na graf G^R. Vrchol v s největším `out`$[v]$ bude ležet ve stokové silně souvislé komponentě grafu G. Tím jsme vyřešili problém (A).

Zbývá vyřešit problém (B). Jak pokračovat po tom, co určíme stokovou silně souvislou komponentu? Znovu využijeme pozorování 7. Po té, co z grafu G vymažeme první nalezenou silně souvislou komponentu, tak vrchol w s největším `out`$[w]$ (ze zbylých vrcholů) patří do stokové silně souvislé komponenty zbytku grafu G. Pokud si budeme pamatovat hodnoty `out`$[\,\cdot\,]$, které jsme získali DFS průchodem grafu G^R, tak můžeme z G snadno odtrhnout druhou silně souvislou komponentu, třetí, čtvrtou a tak dál.

Celý algoritmus vypadá následovně:

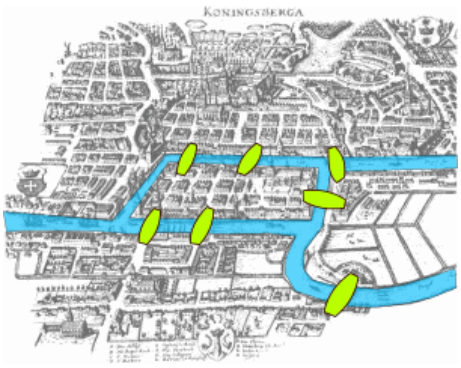
1. Pusť `DFS` na graf G^R a ulož si posloupnost `out`$[\,\cdot\,]$ v klesajícím pořadí.
2. Pusť `DFS` na graf G, ve kterém procházíme vrcholy v pořadí podle klesajících hodnot `out`$[\,\cdot\,]$, a vypisuj nalezené komponenty souvislosti (jako v neorientovaném grafu).

Algoritmu běží v lineárním čase $\mathcal{O}(m+n)$. V podstatě je to dvakrát čas průchodu do hloubky (DFS). (Jak rychle vytvoříme reprezentaci grafu G^R? Vejdeme se do času $\mathcal{O}(m+n)$?)

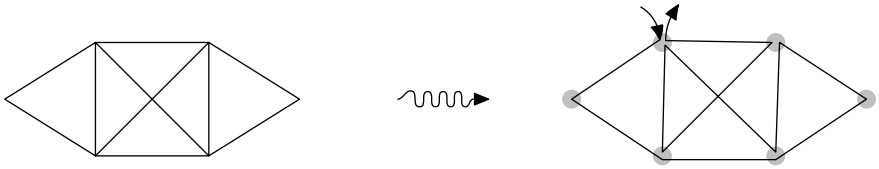
Zkusme algoritmus pustit na grafu G z příkladu. Pokud si DFS může vybrat z více vrcholů, tak si vybere ten abecedně menší. Pořadí vrcholů podle klesajících časů `out`$[\,\cdot\,]$ v průchodu G^R je L, F, G, K, J, I, E, H, D, B, C, A. Silně souvislé komponenty, které algoritmus nalezne, jsou postupně $\{L\}$, $\{E,F,G,I,J,K\}$, $\{D,H\}$, $\{B,C\}$, $\{A\}$.

1.9 Eulerovský tah

Motivace: Městem Královec protéká řeka Préva, na které leží sedm mostů. Stará mapa Královce je na vedlejším obrázku. Šlechtici 18. století už nevěděli coby a tak chtěli během večerní projížďky projet město v kočáře tak, aby přes každý most projeli právě jednou. Nevěděli jak na to a slibovali velkou odměnu tomu, kdo jim ukáže, jak to udělat. Nakonec přišel pan Euler, úlohu vyřešil a odpověděl, že to nejde. Odměnu samozřejmě nedostal.



Jinou motivací pro hledání Eulerovského tahu jsou jednotažky. Dostaneme obrázek a chceme ho nakreslit jedním tahem aniž bychom zvedli tužku z papíru. Známou jednotažkou je například domeček a nebo následující obrázek.



Sled je posloupnost v_0, e_0, v_1, e_1,... v_n taková, že $e_i = \{v_i, v_{i+1}\}$ (neboli každé 2 po sobě jdoucí hrany mají společný vrchol). Pokud se v posloupnosti neopakují hrany, tak ji nazveme *tah* a pokud se neopakují ani vrcholy, tak ji nazveme *cesta.* Uzavřený tah je tah, který začíná i končí ve stejném vrcholu. Uzavřený tah nazveme *cyklus. Uzavřený Eulerovský tah* je tah, který projde všechny hrany grafu a vrátí se do výchozího vrcholu. Graf, který obsahuje alespoň jeden uzavřený Eulerovský tah se nazývá Eulerovský.

Věta 1 *Souvislý graf G je Eulerovský $\iff$ všechny jeho vrcholy mají sudý stupeň.*

Jak najít Eulerovský tah? Na začátku pro jednoduchost předpokládejme, že má každý vrchol sudý stupeň. Pro nalezení uzavřeného Eulerovského tahu použijeme proceduru `Euler(`v`)`. Procedura vypíše vrcholy v pořadí, jak po sobě následují v uzavřeném Eulerovském tahu (vrchol, kterým tah projde k-krát, se na výstupu objeví také k-krát). Procedura `Euler()` funguje hodně podobně jako DFS. Tentokrát můžeme vrcholem projít několikrát, ale každou hranou stále jen jednou.

```
Euler(v):
    while ∃ neprošlá hrana vw do
        označ hranu vw
        Euler(w)
        vypiš(v)
```

Začneme ve vrcholu v a projdeme po libovolné hraně. Každou hranu, po které procházíme, označíme. Vždy když přijdeme do vrcholu různého od v, tak z něj můžeme odejít po neoznačené hraně. To platí proto, že všechny vrcholy mají sudý stupeň a každý průchod přes vrchol označí právě dvě hrany (po jedné jsme přišli a po druhé odešli). Jedinou výjimkou je výchozí vrchol v, protože má navíc označenu hranu, na které jsme tah začali. Po příchodu do výchozího vrcholu se napojíme na začátek tahu a dostaneme cyklus.

Pokud tento cyklus z grafu vyhodíme, dostaneme komponenty souvislosti, které jsou zase Eulerovské (z každého vrcholu jsme vyhodili sudý počet hran). Ve skutečnosti hrany nevyhazujeme, jen je označujeme za prošlé. V neoznačených hranách najdeme stejným způsobem další cyklus. Opakováním postupu nacházíme cykly tak dlouho, dokud jsou v grafu neoznačené hrany.

Slepením nalezených cyklů dohromady dostaneme uzavřený Eulerovský tah. Lepení probíhá podobně, jako když si na výletě s naplánovanou trasou vyrazíte na neplánovaný vyhlídkový okruh. Na chvíli necháte průchodu po naplánované trase, případně odložíte batohy, a vyrazíte na vyhlídkový okruh. Když se vrátíte k batohům, tak pokračujete dále po naplánované trase. Toto lepení za nás v proceduře `Euler(`v`)` provede rekurze.

Jak přesně ta rekurze funguje? Nejprve udělá plán a projde první cyklus (vrcholy naplánovaného tahu jsou uloženy na zásobníku rekurze a hrany cyklu jsou označeny za prošlé). Potom se rekurze začne vracet a vypisovat tah. Když se vrátí do vrcholu, odkud vedou neoznačené hrany, zavolá kamarádku rekurzi, aby zajistila vypsání "výletního" okruhu. Až se kamarádka rekurze vrátí, vypíše se vrchol a pokračuje se v návratu z rekurze.

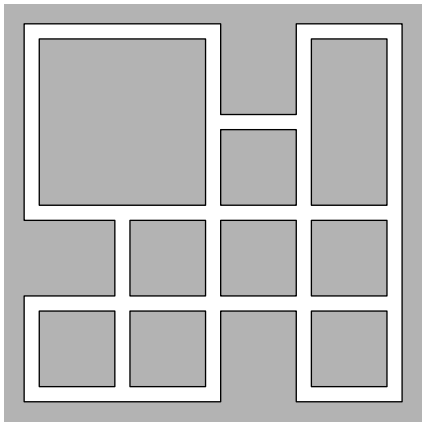
Kdybychom dopředu nevěděli, jestli má každý vrchol grafu sudý stupeň, tak to můžeme během algoritmu snadno zjistit. Pokud během algoritmu přijdeme do vrcholu w (jiného než výchozího vrchol) a už z něj nemůžeme pokračovat dále (všechny hrany vedoucí z vrcholu už jsou označené), tak odpovíme, že w má lichý stupeň a tím pádem víme, že uzavřený Eulerovský tah neexistuje.

Poznámka: K tomu, abychom nakreslili jednotažku jedním tahem, aniž bychom zvedli tužku z papíru, nepotřebujeme uzavřený Eulerovský tah. Stačí otevřený Eulerovský tah. *Otevřený Eulerovský tah* je tah, který projde všechny hrany grafu. Neklademe si žádnou podmínku na to, aby tah skončil ve výchozím vrcholu. Může skončit jinde než začal. Platí věta, která říká, že v souvislém grafu existuje otevřený Eulerovský tah právě tehdy, když všechny vrcholy, až na dva, mají sudý stupeň. Eulerovský tah v jednom lichém vrcholu začne a ve druhém skončí.

Jak by vypadal algoritmus pro nalezení otevřeného Eulerovského tahu?

1.9.1 Pošťákův problém

Motivace: (Problém kropicího vozu) V jednom městě se vzorně starají o své občany. Každý den kropí a zametají všechny ulice, které ve městě mají. Protože je to vzorné město, tak se radní starají i o úsporný rozpočet města. Proto chtějí najít pro řidiče zametacího vozu takovou trasu, aby pokropil a zametl všechny ulice města, ale najezdil co nejméně kilometrů.
Nalezení optimální trasy by městu opravdu pomohlo, protože by ušetřili i při svozu odpadu, rozvozu informačních letáků či volebních lístů. Optimální trasu by ocenila i místní pošta.



Problém se v literatuře označuje jako *pošťákův problém*, protože se problém poprvé studoval v souvisloti s roznášením pošty.[7]

[7] Někdy se mu říká problém čínského pošťáka, protože ho prvně studoval čínský matematik Mei-Ku Kuan v roce 1962.

Úkol: (Pošťákův problém) Dostanete souvislý graf $G = (V, E)$. Nalezněte nejkratší sled, který obsahuje všechny hrany.

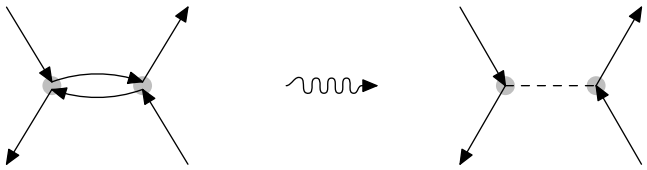
Připomeňme, že *sled* je posloupnost v_0, e_0, v_1, e_1,... v_n taková, že $e_i = \{v_i, v_{i+1}\}$ (neboli každé 2 po sobě jdoucí hrany mají společný vrchol). Sled, který je optimálním řešením úlohy nazveme *pošťákův sled.*

Pokud by v grafu G existoval uzavřený Eulerovský tah, tak je optimálním řešením úlohy. Eulerovský tah projde všechny hrany a každou hranu právě jednou. V takovém případě je řešení jednoduché.

Pokud v grafu neexistuje uzavřený Eulerovský tah, tak budeme muset projít některé hrany vícekrát. Souvislé grafy, které neobsahují uzavřený Eulerovský tah obsahují vrcholy lichého stupně. Nechť $T \subseteq V$ je množina vrcholů lichého stupně. $|T|$ je sudé, protože součet stupňů všech vrcholů je sudé číslo (každá hrana přispěje do tohoto součtu dvojkou).

Lemma 2 *Nechť S je sled, který je optimálním řešením úlohy. Každá hrana grafu je v S použita nejvýše dvakrát.*

Pokud by byla některá hrana uv sledu S použita aspoň třikrát, tak můžeme sled S zkrátit. Na následujícím obrázku je naznačeno zkrácení sledu v případě, kdy po hraně uv vedou 2 průchodu sledu S jdoucí proti sobě (hrana uv zůstane obsažena ve sledu S díky třetímu průchodu). Ještě si rozmyslete možnost, že by sled procházel hranu uv třikrát ve stejném směru.



Lemma 3 *Nechť $H \subseteq G$ je podgraf, který obsahuje právě ty hrany, které optimální sled S projde dvakrát. Vrcholy lichého stupně H jsou právě vrcholy T. Pograf H neobsahuje cykly.*

V multigrafu, který obsahuje každou hranu tolikkrát, kolikkrát ji projde sled S, má každý vrchol sudý stupeň (sled je uzavřený a proto pokaždé když vstoupí do vrcholu, tak z něj i odejde). Po vyhození hran $E(G)$ z multigrafu nám zůstane právě graf H. Lichý vrchol $v \in H$ mohl vzniknout pouze tak, že jsme vyhodili lichý počet hran vedoucích z v. To ukazuje, že liché vrcholy H jednoznačně odpovídají lichým vrcholům v G.

Pokud H obsahuje cyklus, tak ho z H vyhodíme a dostaneme graf H'. V multigrafu, který vznikne součtem G a H', už má každý vrchol sudý stupeň. Proto v něm existuje uzavřený Eulerovský tah, který je také řešením úlohy, a prochází méně hran než sled S. To je spor s tím, že S je optimální sled.

Důsledkem předchozího lemmatu už konečně dostaneme charakterizaci grafu H.

Lemma 4 *H se skládá z $|T|/2$ hranově disjunktních cest, jejichž konce leží v T.*

Teď už máme skoro všechna pozorování potřebná k tomu, abychom úlohu vyřešili. Poslední ingrediencí je algoritmus na maximální párování. Párování M v grafu G je množina vrcholově disjunktních hran (žádné dvě hrany z M nemohou sdílet vrchol).

Algoritmus na maximální párování si zde nevysvětlíme, protože jeho popis by vydal na samostatnou kapitolu. Časová složitost algoritmu je polynomiální. Popis

algoritmu na maximální párování v bipartitním grafu najdete u aplikací toků v sítích v sekci **??**. Popis algoritmu pro obecné grafy čtenář nalezne například v [**?**].

Algoritmus řešící Pošťákův problém:

1. Nalezneme vrcholy T, které mají lichý stupeň v G. Chceme najít $|T|/2$ cest, které spárují vrcholy T a budou obsahovat co nejméně hran. Provedeme to následovně. Vytvoříme pomocný graf G' na vrcholech T. Každé dva vrcholy $u, v \in T$ spojíme hranou uv, kterou odhodnotíme délkou nejkratší cesty mezi u a v. V grafu G' nalezneme maximální párování minimální ceny. Nalezené párování odpovídá hledaným cestám. Ty tvoří optimální graf H.

2. V multigrafu $G + H$ najdeme uzavřený Eulerovský tah. Ten odpovídá sledu, který je řešením úlohy.

Poznámka: V reálných úlohách chceme skutečně najezdit co nejméně. Každá silnice má svojí délku $d(e)$. Chceme najít nejkratší sled, který projde všechny hrany. Délka sledu je součet délek hran, které prochází.

Algoritmus řešící Pošťákův problém v ohodnocených grafech vypadá skoro stejně jako pro neohodnocené grafy, jenom místo maximální párování hledáme maximální párování minimální ceny (cena párování M je součet cen všech hran patřících do M). Maximální párování minimální ceny se dá najít v polynomiálním čase a proto umíme ve stejném čase vyřešit i Pošťákův problém v ohodnocených grafech. Více informací čtenář najde například v [**?**].

Poznámka: Pošťákův problém je orientovaných grafech o něco jednoduší, protože ho můžeme vyřešit pomocí toků v sítích. Viz cvičení **??** na konci sekce **??**.

1.10 BFS, hledání nejkratší cesty

Druhou implementací efektivního průchodu grafu je průchod do šířky (BFS, neboli Bread First Search). Jeho výhodou o proti DFS je, že kromě samotného řešení nalezne i nejkratší cestu, která k řešení vede.

Prohledávání začneme ve vrcholu s a projdeme celou komponentu souvislosti, do které s patří. Délku cesty měříme počtem hran na cestě. Proměnná H[v] bude obsahovat délku nejkratší cesty z s do v. Na začátku bude obsahovat hodnotu -1. V momentě, kdy se při průchodu grafu dostaneme do vrcholu v, nastavíme proměnnou H[v] na správnou hodnotu, což bude číslo větší nebo rovno nule. Proto podle hodnoty H[v] poznáme, jestli jsme už vrcholem v někdy prošli a nebo ještě ne.

```
BFS(s):
   ∀v ∈ V : H[v]:=−1
   H[s]:=0 a přidej s do fronty
   while fronta neprázdná do
      odeber vrchol v z fronty
      for all sousedy w vrcholu v do
         if H[w]=−1 then
            přidej w do fronty
            H[w]:=H[v]+1
```

Pozorování 8 *Nejprve zpracujeme všechny vrcholy se stejným číslem* H[v] *a pak teprve začneme zpracovávat vrcholy s číslem o jedna větším. V průběhu algoritmu jsou ve frontě jen vrcholy, které mají* H[v] *rovno d nebo $d+1$, pro nějaké d.*

Díky tomu, že zpracování vrcholů probíhá po vlnách podle čísla H[v], se algoritmu říká *algoritmus vlny.*

Pozorování 9 H[v] *obsahuje délku nejkratší cesty z s do v.*

Pozorování dokážeme indukcí podle hodnoty H[v]. Vrchol s má H[s]=0 a nejkratší cesta z s do s má rovněž délku nula. Předpokládejme, že pro všechny vrcholy s hodnotou H[v] $\leq k$ je H[v] délka nejkratší cesty z s do v. Potřebujeme ukázat indukční krok a to je, že každý vrchol v s H[v] $= k+1$ leží ve vzdálenosti $k+1$ od s. Pokud ne, tak existuje kratší cesta z s do v a nechť wv je poslední hrana na této cestě. H[w] $< k$. Potom se ale měl algoritmus při zpracovávání vrcholu w podívat na hranu wv a nastavit H[v] na H[w] $+1$. Hodnota H[w] $+1 < k+1$, což je spor.

Podobně jako v DFS tvoří stromové hrany DFS strom, tak i hrany, po kterých při BFS projdeme do nenavštíveného vrcholu, tvoří *BFS strom* (strom průchodu do šířky).

Jednotlivé „vlny“ nazveme *vrstvy* průchodu do šířky. V i-té vrstvě jsou obsaženy právě vrcholy ve vzdálenosti i od počátečního vrcholu s.

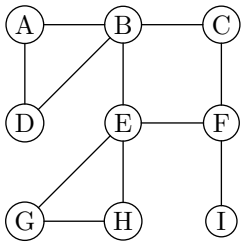
Příklad: Na kolik nejméně skoků doskáčeme šachovým koněm z A1 na A2? Úlohu vyřešíme pomocí průchodu do šířky (algoritmu vlny). Řešit ji můžeme přímo na papíře tak, že budeme do prázdných políček psát na kolik skoků se do nich dostaneme. Začneme na A1, kam napíšeme nulu. Do políček, kam doskočíme z A1 napíšeme jedničky. Ve druhé vlně napíšeme dvojku do všech políček, kam doskočíme z políček s jedničkou, atd. (do popsaných políček už nic nepíšeme). Skončíme ve chvíli, kdy se dostaneme do A2. Číslo napsané na políčku A2 udává nejmenší možný počet skoků.

	A	B	C	D	E	F	G	H
1	0	3	2	3	2	3		
2	3		1	2	3		3	
3	2	1		3	2	3		
4	3	2	3	2	3		3	
5	2	3	2	3		3		
6	3		3		3			
7		3		3				
8								

1.11 Příklady

1.11.1 Přímé procvičení vyložených algoritmů

1. Graf na následujícím obrázku projděte pomocí průchodu do hloubky (DFS). Pokud si v některém kroku můžete vybrat z několika vrcholů, tak si vždy vyberte ten abecedně nejmenší z nich.



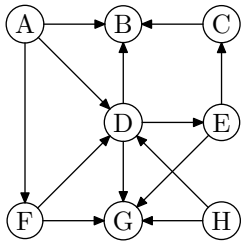
   (a) Rozdělte hrany na stromové a zpětné.

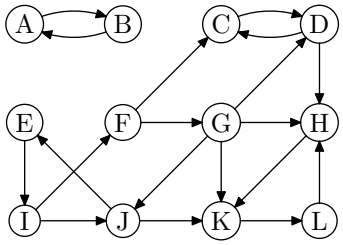
   (b) U každého vrcholu v spočítejte časy příchodu a odchodu – hodnoty `in`$[v]$ a `out`$[v]$.

   (c) Najděte 2-souvislé komponenty grafu.

   (d) Pro každý vrchol v spočítejte hodnotu `LOW`$[v]$.

2. Oba orientované grafy na následujících obrázcích projděte pomocí průchodu do hloubky (DFS). Pokud si v některém kroku můžete vybrat z několika vrcholů, tak si vždy vyberte ten abecedně nejmenší z nich.





   (a) Rozdělte hrany na stromové a zpětné, příčné a dopředné.

   (b) U každého vrcholu v spočítejte časy příchodu a odchodu – hodnoty `in`$[v]$ a `out`$[v]$.

   (c) Najděte topologické uspořádání každého z grafů.

   (d) Vyznačte, které vrcholy jsou zdroje a které stoky.

   (e) Najděte silně souvislé komponenty každého z grafů.

   (f) Vyznačte, které silně souvislé komponenty jsou zdrojové a které stokové.

1.11.2 Průchod grafu do šířky

1. (Cesta tunelem) Rodina tvořená tatínkem, maminkou, dcerou a synem chce projít tunelem. Tatínek projde tunelem za jednu minutu, maminka za dvě, syn za čtyři a dcera za pět. Problém je v tom, že v tunelu je hrozná tma a jejich svíčka vydrží hořet pouze dvanáct minut. Úzkým tunelem mohou procházet naráz nejvýše dva lidé a v žádném případě nesmí jít tunelem nikdo bez svíčky. Poradíte rodině, jak mají tunelem projít?

2. (Přelévání tekutin) Máme 3 nádoby o různých objemech. Žádná nádoba na sobě nemá stupnici a nádoby mají dokonce tak roztodivné tvary, že nám znemožňují odhadování množství vody v nich. Stále ale může naměřit i jiné množství tekutiny. Můžeme přelévat obsah jedné nádoby do druhé tak dlouho, dokud nepřelijeme všechno nebo dokud se druhá nádoba nezaplní. Jaký je nejmenší počet přelití, abychom vyřešili následující úlohy? Ve všech variantách je na počátku největší nádoba plná.

   (a) Máme 3 nádoby o objemech 7l, 5l a 3l. Chceme naměřit 1 litr.

   (b) Máme 3 nádoby o objemech 7l, 4l a 3l. Přeléváním se chceme dostat do stavu, kdy je v jedné nádobě 3l a ve zbylých dvou po 2 litrech.

   (c) Máme 3 nádoby o objemech 8l, 5l a 3l. Přeléváním se chceme dostat do stavu, kdy je počáteční množství rozděleno na dva stejné díly.

3. (Nejkratší cesta)

   (a) V neohodnoceném grafu G najděte nejkratší cestu z vrcholu s (start) do vrcholu c (cíl).

   (b) Všechny hrany v grafu G mají délku 1 nebo 2. Najděte nejkratší cestu z vrcholu s (start) do vrcholu c (cíl).

   (c) Dokázali byste řešení zobecnit i pro grafy s ohodnocením hran 1 až k? Zajímá nás lepší řešení, než pomocí Dijkstrova algoritmu. Chtěli bychom najít řešení v čase $\mathcal{O}(km + n)$.

4. (Cesta kulhavého koně) Náš šachový kůň kulhá. To znamená, že v sudých tazích provede krok jako šachový kůň a v lichých tazích provede krok jako šachový král. Zjistěte, na kolik nejméně kroků se dostane kulhavý kůň z políčka A1 na políčko H8. Až to zjistíte, tak úlohu vyřešte obecně pro libovolné počáteční a cílové políčko.

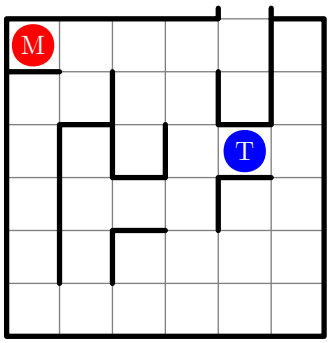
5. (Bludiště) Pomocí matice velikosti $n \times m$ máte zadáno bludiště. X značí zeď a nula volný prostor. Na okraji matice jsou všude zdi, takže jde o uzavřený systém místností a chodeb. V matici se vyskytují i znaky $=poklad a S=místo, kde se zrovna nacházíte. Zjistěte, zda se můžete dostat k pokladu. Předpokládejte, že nejste bílá paní, která by mohla procházet přes zdi.

X	X	X	X	X	X	X	X	X
X	S	0	0	X	0	0	0	X
X	0	X	0	X	0	X	0	X
X	0	0	0	0	0	X	0	X
X	0	0	0	X	0	X	$	X
X	X	X	X	X	X	X	X	X

6. (Bludiště se dveřmi) Máte zadané bludiště jako v předchozí úloze. Navíc se ale v bludišti vyskytují znaky D jako dveře. Dveře jsou ocelové a dají se otevřít pouze dynamitem. Nejste bílá paní, ale jste lupiči, co se chtějí vloupat do sejfu a vykrást banku. Podařilo se vám ukořistit plán sejfu. Pro hladký a bezpečný průběh akce se chcete dostat k penězům na co nejmenší počet kroků, ale hlavně tak, abyste museli odbouchnout co nejméně dveří. Jednou odbouchnutí dveří trvá nesrovnatelně déle než libovolný počet kroků v sejfu. Navíc nadělá ohromný hluk. Vaším úkolem je najít pro lupiče posloupnost pohybů vlevo, vpravo, nahoru, dolů tak, aby se dostali co nejrychleji k penězům. Lupiči už sami pochopí, že když mají projít dveřmi, tak je mají odprásknout.

7. (Průjezd městem) Máte zadaný plán města podobně jako v předchozích úlohách, tj. pomocí matice. Tentokrát jsou v něm pouze silnice široké jeden čtvereček a křižovatky. Vaším úkolem je najít cestu ze startu do cíle (čtverečky označené S a C). Zdá se vám to jednoduché? Tak zkuste najít cestu ze startu do cíle s dodržováním místních dopravních předpisů. V tomto městě je zakázáno na libovolné křižovatce odbočovat vpravo. Vpravo se ale můžete dostat například tak, že projedete rovně přes křižovatku a objedete jeden blok (třikrát zatočíte vlevo).

8. (Theseus a Minotaurus) Zahrajeme si hru ve dvourozměrném bludišti $n \times m$ polí. V bludišti se (samozřejmě kromě zdí, ty se vyskytují v každém pořádném bludišti) nachází Theseus a Minotaurus. Vy budete ovládat Thesea a budete se snažit dostat z bludiště ven, čili dostat se na hranici bludiště a následujícím krokem z bludiště utéct. Ovšem nikdy nesmíte narazit na Minotaura, sic bídně zhynete. Na Minotaura narazíte, pokud s ním sdílíte políčko.



Jeden tah probíhá následovně: nejprve se hýbe Minotaurus a táhne k-krát následujícím způsobem. Pokud nejsou postavy Minotaura a Thesea ve stejném sloupci, chce se Minotaurus pohnout o jedno políčko vlevo nebo vpravo, aby se Theseovi přiblížil. Pokud mu v tom nebrání zeď, skutečně se tam posune. Pokud nejsou obě postavy na stejném řádku bludiště, chce se Minotaurus pohnout nahoru či dolů opět směrem k Theseovi. Opět se na zvolené políčko Minotaurus přesune jen tehdy, není-li tam zeď. V jednom kroku provádí Minotaurus tyto pohyby v zadaném pořadí a může provést oba dva, čili se může dostat na jedno z okolních osmi políček. Po k takovýchto krocích Minotaura se hýbe Theseus, a to na jedno ze čtyř okolních volných polí. Takto se oba střídají na tahu, dokud buď Theseus neuteče z bludiště, nebo dokud Minotaurus nedohoní Thesea. Vaším úkolem bude najít pro Thesea nejkratší posloupnost pohybů vlevo, vpravo, nahoru, dolů, aby se dostal bezpečně z bludiště, případně říci, že to není možné.

9. (Loydova devítka) Napište program, který zjistí, jestli jde Loydova devítka poskládat a řekne vám, jak šoupat čtverečky tak, abyste ji poskládali na nejmenší počet tahů.

1.11.3 Průchod grafu do hloubky

1. (Obrácení hran) Orientovaný graf $G^R = (V, E^R)$ vznikne z orientovaného grafu $G = (V, E)$ obrácením hran. Tedy $E^R = \{xy \mid yx \in E\}$. Navrhněte algoritmus, který spočítá reprezentaci grafu G^R v lineárním čase $\mathcal{O}(n+m)$. Úlohu řešte pro reprezentaci seznamem sousedů. Jak by to bylo pro reprezentaci maticí incidence?

2. (Vstupní a výstupní stupně) Dostaneme orientovaný graf $G = (V, E)$ reprezentovaný seznamem sousedů. Stupeň vrcholu v (značíme $deg\, v$) je počet hran grafu G, které do vrcholu v vedou a nebo které z vrcholu v vychází. Vstupní stupeň vrcholu v (značíme $deg^+ v$) je počet hran grafu G, které do vrcholu v vstupují. Výstupní stupeň vrcholu v (značíme $deg^- v$) je počet hran grafu G, které z vrcholu v vychází. Platí $deg^+ v + deg^- v = deg\, v$.

   (a) Dokažte, že $\sum_{v \in V} deg\, v = 2|E|$.

(b) S využitím části (a) dokažte, že počet vrcholů s lichým stupněm musí být sudé číslo.

(c) Navrhněte algoritmus, který v lineárním čase spočítá vstupní a výstupní stupně všech vrcholů.

3. (Čtverce stupňů sousedů) Dostaneme neorientovaný graf $G = (V, E)$ reprezentovaný seznamem sousedů. Definujeme $\texttt{dvadeg}[v] := \sum_{u \in \texttt{Sousedi}[v]} (deg\, v)^2$. Navrhněte lineární algoritmus, který pro všechny vrcholy spočítá $\texttt{dvadeg}[\,\cdot\,]$ v čase $\mathcal{O}(n + m)$.

4. (Bipartitní graf) Graf $G = (V, E)$ je *bipartitní,* pokud jeho vrcholy můžeme rozdělit na dvě disjunktní množiny V_1, V_2 (tedy $V_1 \cup V_2 = V$ a $V_1 \cap V_2 = \emptyset$) tak, aby všechny hrany vedly z jedné množiny do druhé. (Graf nemůže obsahovat hranu s oběma konci ve stejné množině.)

   (a) Navrhněte lineární algoritmus, který dostane graf G a zjistí, jestli je bipartitní.

   (b) Navrhněte algoritmus, který v lineárním čase zjistí, jestli graf obsahuje lichý cyklus (cyklus liché délky).

   (c) Pokud graf neobsahuje lichý cyklus, dokážete ho obarvit 2 barvami? Obarvení grafu 2 barvami je obarvení jeho vrcholů 2 barvami takové, že každé dva vrcholy spojené hranou mají různou barvu. Je každý takový graf bipartitní?

5. (Vyhodnocení stromu) Dostanete binární zakořeněný strom, jehož každý vrchol v obsahuje číslo $\texttt{z}[v]$. Hodnota $\texttt{zmax}[v]$ je definovaná jako maximum z hodnot $\texttt{z}[w]$, pro všechny potomky w vrcholu v. Spočítejte celé pole $\texttt{zmax}[\,\cdot\,]$ v lineárním čase.

6. (Následník ve stromě) Dostaneme zakořeněný strom. Vrchol u je následníkem vrcholu v, pokud v leží na cestě z kořene do u. Postupně budete dostávat dvojice vrcholů x, y a chtěli bychom co nejrychleji odpovídat na otázku, jestli je x následníkem y?

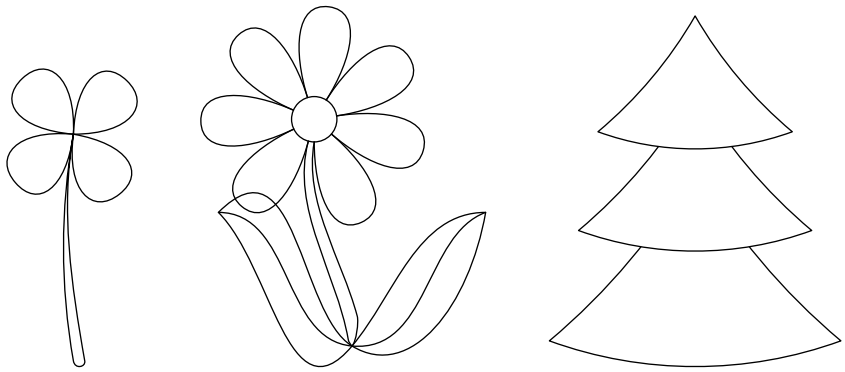
   Pokud si můžete dovolit předzpracování v čase $\mathcal{O}(n)$, zvládnete odpovídat v konstantním čase?

7. (Eulerovský tah) Tah v grafu G je posloupnost v_0, e_0, v_1, e_1,... v_n taková, že $e_i = \{v_i, v_{i+1}\}$ a $v_i \neq v_j$ pro všechna i, j (neboli každé 2 po sobě jdoucí hrany mají společný vrchol a vrcholy se v posloupnosti neopakují). *Uzavřený Eulerovký tah* je tah, který projde všechny hrany grafu a skončí ve stejném vrcholu, ve kterém začal. *Otevřený Eulerovský tah* projde všechny hrany grafu, ale může končit na jiném místě, než začal.

   (a) Navrhněte algoritmus, který v grafu G najde uzavřený Eulerovský tah a vypíše ho. Případně odpovězte, že takový tah neexistuje.

   (b) Navrhněte algoritmus, který v grafu G najde otevřený Eulerovský tah a vypíše ho. Případně odpovězte, že takový tah neexistuje.

   (c) Následující rostlinky nakreslete jedním tahem. Křížení čar považujte za vrcholy.



8. Navrhněte algoritmus, který v grafu G najde libovolnou kostru, případně odpovězte, že neexistuje. Zkuste vymyslet velice jednoduché řešení pracující v čase $\mathcal{O}(n + m)$.

9. Navrhněte algoritmus, který pro graf G vypíše jeho komponenty souvislosti.

10. Navrhněte algoritmus, který pro graf G vypíše jeho 2-souvislé komponenty.

11. Navrhněte algoritmus, který pro orientovaný graf G vypíše jeho silně souvislé komponenty.

12. (Topologické uspořádání) V poušti na Sahaře byly nalezeny zbytky knihovny nějaké dávné civilizace. Civilizace znala písmo a předpokládá se, že měla i abecedu (uspořádání písmenek). Vědci se domnívají, že nalezené spisy byly v knihovně seřazeny lexikograficky.[8] Dostanete názvy spisů, které se zachovaly a to v pořadí jak byly poskládány na poličce. Názvy spisů jsou přepsané tak, že si každý znak označíme jedním písmenkem naší abecedy.

    Například pro českou knihovnu byste dostali seznam: „Alenka v říši divů“, „Alexandr Veliký“, „Baron Prášil“, „Broučci“ „Malý princ“, „Medvídek Pů“. Ze seznamu můžete vyvodit, že v české abecedě je ‘A’ před ‘B’, ale také ‘n’ před ‘x’.

    Podle názvu spisů, které dostanete, zkuste potvrdit nebo vyvrátit teorii o tom, že saharská civilizace měla abecedu. Pokud teorii potvrdíte, tak vypište jednu možnost, jak mohla jejich abeceda vypadat.

13. (Poznámka k topologickému uspořádání) Navrhněte algoritmus, který dostane neorientovaný graf, a zjistí, jestli graf obsahuje kružnici. Zkuste vymyslet řešení, které běží v čase $\mathcal{O}(n)$ (tedy je nezávislé na počtu hran).

14. (Topologické uspořádání) Dokažte nebo vyvraťte: Pokud orientovaný graf obsahuje orientované cykly, tak procedura pro topologické uspořádání nalezne uspořádání vrcholů, které minimalizuje počet „špatných“ hran, vedoucích zprava do leva.

15. (Polosouvislé grafy) Orientovaný graf $G = (V, E)$ je polosouvislý, pokud pro každé dva vrcholy $u, v \in V$ existuje orientovaná cesta z u do v nebo orientovaná cesta z v do u. Navrhněte algoritmus, který zjistí, jestli je graf G polosouvislý. Dokažte jeho správnost a určete jeho časovou složitost.

16. (Jednoznačně souvislé grafy) Orientovaný graf $G = (V, E)$ je jednoznačně souvislý, pokud pro každé dva vrcholy $u, v \in V$ existuje právě jedna orientovaná

[8] To je tak, jak řadíme slova ve slovníku. Česky bychom místo lexikografické uspořádání mohli říci slovníkové uspořádání.

cesta z u do v a právě jedna orientovaná cesta z v do u. Navrhněte algoritmus, který zjistí, jestli je graf G jednoznačně souvislý. Dokažte jeho správnost a určete jeho časovou složitost.

17. (Cyklus obsahující hranu e) Dostanete neorientovaný graf G s vyznačenou hranou e. Navrhněte algoritmus pracující v lineárním čase, který zjistí, jestli v grafu G existuje cyklus obsahující hranu e.

18. Navrhněte efektivní algoritmus, který dostane orientovaný graf $G = (V, E)$ a zjistí, jestli v G existuje vrchol $v \in V$, ze kterého jsou všechny ostatní vrcholy dosažitelné.

19. Navrhněte efektivní algoritmus, který dostane orientovaný acyklický graf $G = (V, E)$ a dvojici vrcholů s, c a vrátí počet různých cest vedoucích z s do c.

20. Dostanete orientovaný acyklický graf. Navrhněte efektivní algoritmus, který zjistí, jestli v grafu existuje cesta, která navštíví každý vrchol právě jednou.

21. Dostanete orientovaný acyklický graf. Navrhněte efektivní algoritmus, který spočítá délku nejdelší orientované cesty.

22. Dostanete orientovaný graf $G = (V, E)$, a ke každému vrcholu $v \in V$ dostanete navíc jeho cenu $\texttt{c}[v]$. Definujeme $\texttt{mcena}[v]$:= cena nejlevnějšího vrcholu dosažitelného z v (včetně v). Naším cílem na vymyslet algoritmus pracující v čase $\mathcal{O}(n + m)$, který vyplní pole $\texttt{mcena}[\,\cdot\,]$.

    (a) Nejprve vymyslete algoritmus, který funguje pro orientované acyklické grafy.
    Nápověda: Zpracovávejte vrcholy v určitém pořadí.

    (b) Rozšiřte předchozí algoritmus tak, aby pracovat na všech orientovaných grafech.
    Nápověda: Připomeňte si strukturu orientovaných grafů (meta-graf, silně souvislé komponenty).

23. Dostanete neorientovaný graf $G = (V, E)$. Zorientujte jeho hrany tak, aby pro každý vrchol v byl $|deg^+ v - deg^- v| \leq 1$.

24. Dostanete neorientovaný graf $G = (V, E)$. Zorientujte jeho hrany tak, aby se po orientovaných cestách dalo dostat odkudkoliv kamkoliv. Případně řekněte, že taková orientace neexistuje.

25. Dostanete neorientovaný graf $G = (V, E)$. Přidejte k němu nejmenší možný počet hran, aby G byl 2-souvislý.

26. Dostanete orientovaný graf $\vec{G} = (V, E)$. Přidejte k němu nejmenší možný počet orientovaných hran tak, aby G byl silně souvislý souvislý.

27. Dostanete rovinný graf $G = (V, E)$. Obarvení grafu je přiřazení barev vrcholům takové, že vrcholy spojené hranou mají různou barvu. Zajímá nás, kolik nejméně barev je potřeba.

    (a) Navrhněte efektivní algoritmus, který obarví G co nejmenším počtem barev.

    (b) Navrhněte lineární algoritmus, který obarví G pomocí 6 barev.

    (c) Jak rychle byste dokázali obarvit G pomocí 5 barev?

    (d) Graf G je k-degenerovaný, pokud každý jeho podgraf (včetně G samotného) obsahuje vrchol stupně menšího nebo rovno k. Ukažte, jak obarvit k-degenerované grafy pomocí $k + 1$ barev. Jak rychle poběží Váš algoritmus?

1.11.4 Úlohy na DFS průchod stavovým prostorem

Tady je pár úloh, jejichž řešení můžete nalézt vyzkoušením všech možností (tak zvaný backtracking). Připomínáme, že zkoušení všech možností, není vždy nejrychlejším řešení, ale někdy nic lepšího neumíme.

Pozor, často se stává, že studenti napíší backtracking nevhodně a vyhodnocují některé stavy vícekrát (viz jak se neefektivně počítali Fibonacciho čísla v sekci **??**).

1. (Proskákání šachovnice koněm) Dostanete šachovnici o rozměrech $n \times m$. Vaším úkolem je zjistit, jestli šachovnici můžeme proskákat šachovým koněm tak, abychom každé políčko navštívili právě jednou.

   Nápověda : Pro urychlení výpočtu zkuste použít následující heuristiku. Vždy když si můžete vybrat, kam skočíte, tak nejdřív zkuste skočit na to políčko, odkud je nejméně možností jak pokračovat.

2. (Rozmístění šachových figurek na šachovnici tak, aby se neohrožovali) Je dána šachovnice o rozměrech $n \times m$. Jedna figurka ohrožuje druhou, pokud ji může skočit jedním šachovým tahem. Vaším úkolem je rozmístit na šachovnici co nejvíce předepsaných figurek tak, aby se navzájem neohrožovali.

   (a) Šachové věže.

   (b) Šachové střelce.

   (c) Šachové koně.

   (d) Šachové dámy.

   (e) Maharadže. Maharádža si v každém tahu může vybrat, jestli bude skákat jako kůň a nebo jako dáma.

   (f) Sultány. Sultán si v každém tahu může vybrat, jestli bude skákat jako kůň a nebo jako střelec.

   Nápověda: Program lze zjednodušit malým trikem. Místo toho, abychom si pamatovali, jestli je určité políčko ohroženo nějakou už umístěnou figurkou, si budeme pamatovat kolika figurkami je dané políčko ohroženo. Zjednoduší nám to návrat do předchozího stavu (odebrání figurky). Trik lze dobře využít například pro šachového koně.

3. (Rozmístění šachových figurek na pneumatiku tak, aby se neohrožovali) Navážeme na předchozí úlohu. Jestli už umíte rozmístit co nejvíce předepsaných šachových figurek na normální šachovnici, tak je můžete zkusit rozmístit na šachovnici, které je nakreslená na pneumatice. Šachovnici na pneumatice dostaneme tak, že běžnou šachovnici nejprve stočíme do válce a válec pak ohneme tak, aby se děravé konce spojili. Předpokládejme, že se kraje původní šachovnice spojí tak, že už ani nepoznáme, kde byly.

   Pozor, diagonály na šachovnici na pneumatice se chovají úplně jinak než na normální šachovnici!

   (a) Pro které rozměry $n \times m$ má šachovnice jen jednu levou a jednu pravou diagonálu?

   (b) Jak poznáme, kolik levých a kolik pravých diagonál má šachovnice o rozměrech $n \times m$?

   (c) Pokud už víte, kolik diagonál má která šachovnice na pneumatice a jak jsou diagonály na šachovnici „namotané“, tak na takovou šachovnici můžete zkusit rozmístit co nejvíce dam tak, aby se vzájemně neohrožovali.

1.11.5 Související úložky z teorie grafů

1. (Struktura 2-souvislých komponent) Pro každou 2-souvislou komponentu vytvoříme zvláštní vrchol. Dva zvláštní vrcholy spojíme hranou právě tehdy, když jim odpovídající 2-souvislé komponenty mají společnou artikulaci. Ukažte, že takto vzniklý graf je strom.

   Nápověda: Ukažte, že vzniklý graf nemůže obsahovat kružnice.

2. Dvě cesty v grafu jsou hranově disjunktní, pokud nemají společnou hranu (ale mohou sdílet vrchol). Ukažte, že v libovolném neorientovaném grafu můžeme spárovat vrcholy lichého stupně a spojit každý pár cestou tak, aby všechny cesty byly hranově disjunktní.

3. (Relace ekvivalence) Nechť S je konečná množina. *Binární relace* R je podmnožina $S \times S$. Jinými slovy R je množina dvojic $(x, y) \in S \times S$. Například pokud by S byla množina lidí, tak $(x, y) \in R$ právě tehdy když „x zná y“.

   Relace je *relací ekvivalence*, pokud splňuje následující vlastnosti

   - (reflexivita): $(x, x) \in R$
   - (symetrie): pokud $(x, y) \in R$, potom i $(y, x) \in R$
   - (tranzitivita): pokud $(x, y) \in R$ a $(y, z) \in R$, potom $(x, z) \in R$

   Například relace „má ve stejný den narozeniny“ je relace ekvivalence a relace “je otcem” není relace ekvivalence.

   Dokažte, že relace ekvivalence rozkládá množinu S na disjunktní skupiny S_1, S_2, …, S_k takové, že

   - Každé dva prvky jedné skupiny jsou v relaci. To je $(x, y) \in R$ pro každé $x, y \in S_i$, pro každé i.
   - Žádné dva prvky z různých skupin nejsou v relaci. Jinými slovy pro každé $i \neq j$ a pro každé $x \in S_i$, $y \in S_j$ platí $(x, y) \notin R$.

   Rozkladem S na skupiny S_1, S_2, …, S_k myslíme, že $S = S_1 \cup S_2 \cup \cdots \cup S_k$ a $S_i \cap S_j = \emptyset$ pro každé $i \neq j$.

   Nápověda: Reprezentujte si relace pomocí orientovaných grafů.

4. (Kotouč) Máme veliký kotouč, který má na obvodu napsány nuly a jedničky. Celý kotouč je skryt v pouzdře, akorát na jedné straně má „okénko“, kterým je vidět k po sobě jdoucích čísel. Pootočením kotouče se nejlevější číslice v okénku schová, ale zprava se objeví nová číslice. Takovýmto pootočením dostaneme následující k-tici. Kolik nejvíc číslic (nul a jedniček) můžeme napsat na obvod kotouče tak, aby se žádná k-tice po sobě jdoucích čísel neopakovala? Umíte takové pořadí nul a jedniček najít?

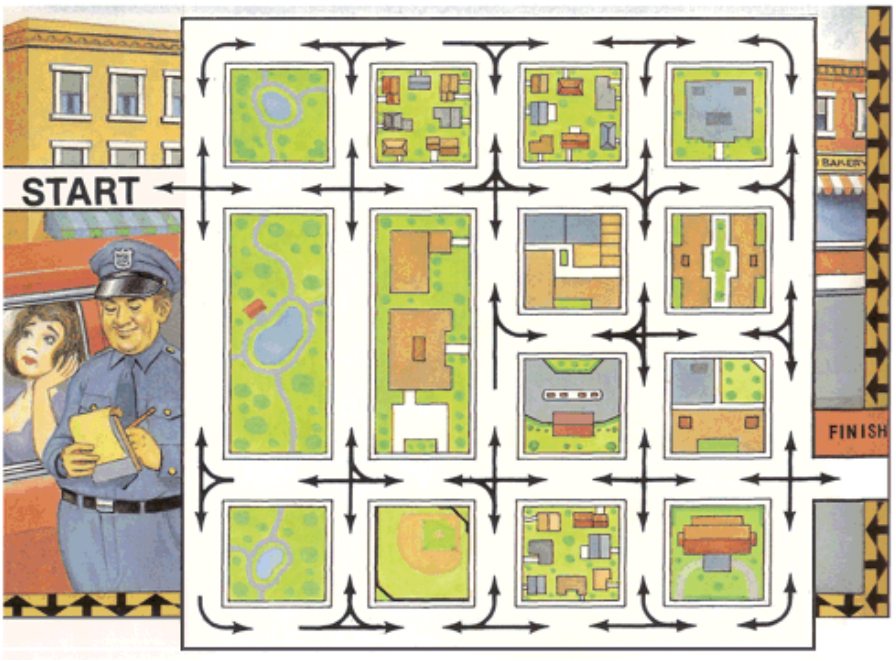
5. (Nekonečné grafy)

   (a) (Kráva před nekonečným plotem) Kráva stojí před nekonečně dlouhým a rovným plotem. Někde v plotu je díra. Kráva se může vydat hledat díru buď doleva nebo doprava. Zjistěte, jak má kráva postupovat, aby se nenachodila moc a zjistila, kde je díra.
   Zkuste najít algoritmus, podle kterého kráva nachodí nejvýše 2 krát tolik než je počáteční vzdálenost mezi krávou a dírou (plus malé epsilon). Takovýmto algoritmům se říká 2-kompetitivní, protože vydají nejvýše 2 krát větší odpověď, než kolik je optimum.

   (b) (Nekonečné bludiště) Stojíte uprostřed nekonečně velkého bludiště. Někde v bludišti je ukryt teleport, kterým se můžete dostat ven. Vymyslete, jak ho s jistotou najít.

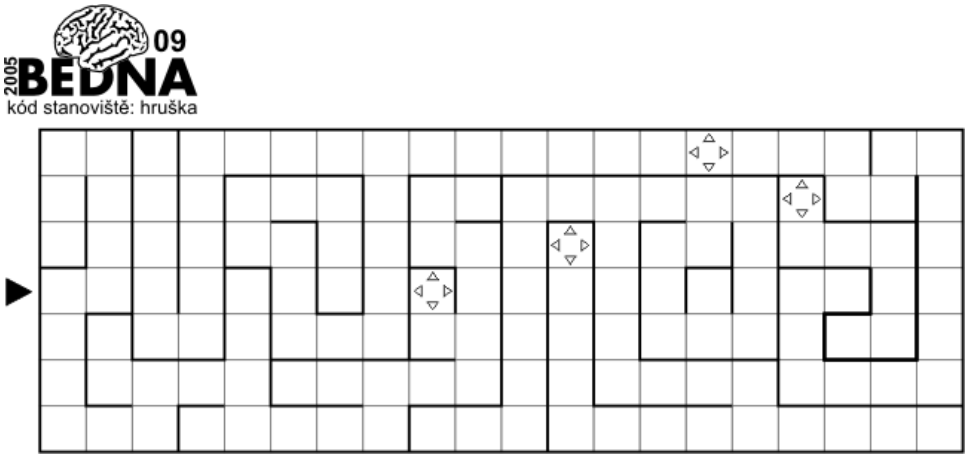
1.11.6 Hravá bludiště

1. (Farmář jede na trh) Následující bludiště pochází od Robert Abbotta a jmenuje se „Farmer goes to market“.[9] Vaším úkolem je poradit farmáři, jak má projet městem tak, aby se ze startu dostal do cíle (finish) a neporušil místní dopravní předpisy. Šipky na každé křižovatce znázorňují, odkud kam se dá křižovatka projet bez porušení dopravních předpisů.

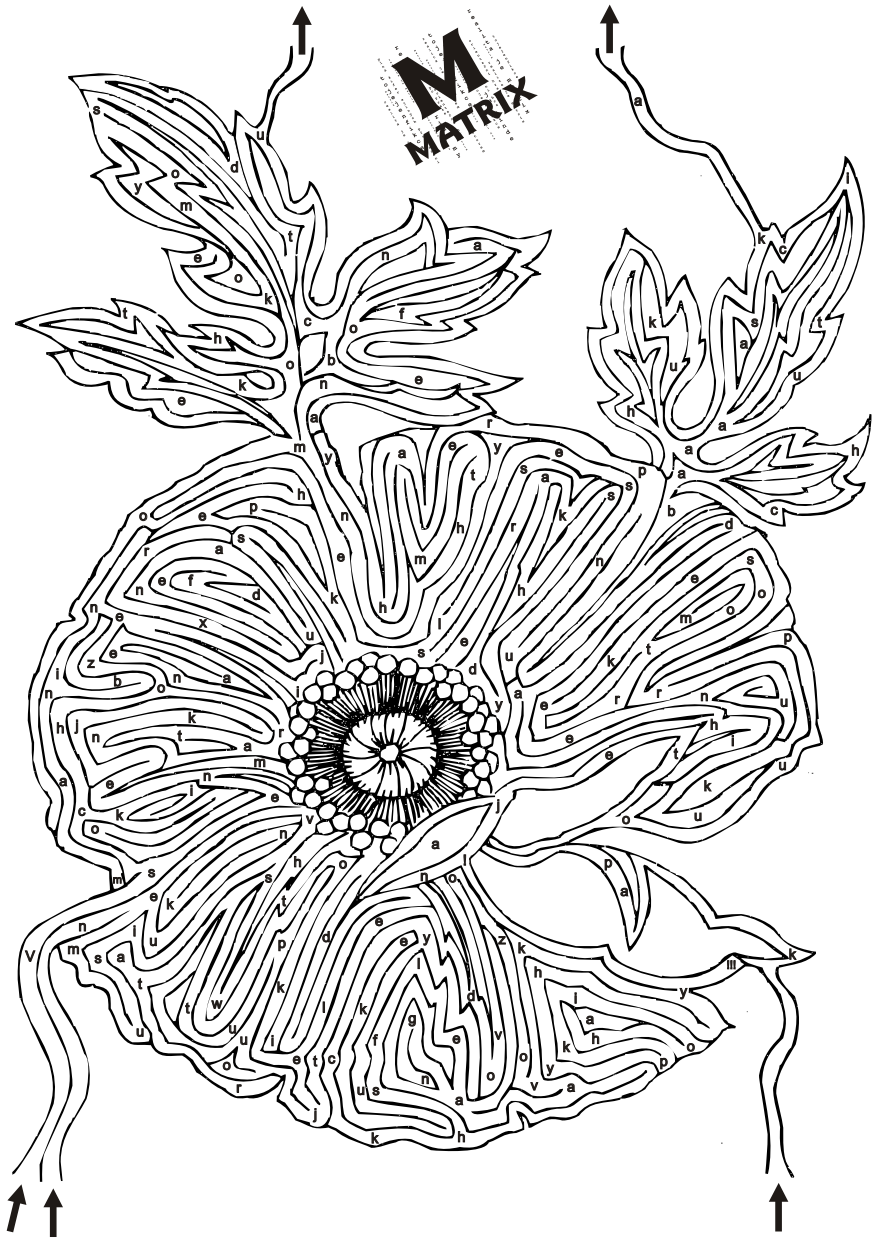


1.11.7 Šifry

Vyřešte následující šifry. První se objevila v šifrovací hře Bedna 2005 a druhá ve hře MATRIX 2005.



[9] Jde o složitější variantu bludiště od Martina Gardnera: Traffic Flow In Floyd's Knob.


M
MATRIX