5 Prohledávání grafů

V této přednášce se budeme zabývat metodou systematického prohledávání kombinatorických struktur. Tato činnost je známa programátorům, kteří používají metodu hrubé síly, tzv. *backtracking* při prohledávání stavového prostoru možných řešení té–které úlohy.

Naším cílem je prozkoumat systematické prohledávání grafů (multigrafů). Navrhneme optimální lineární algoritmy (v počtu hran) a na příkladech ukážeme, že tato jednoduchá metoda vyřeší na první pohled komplikované úlohy z teorie grafů.

Zkoumáním labyrintů se zabývalo mnoho matematiků, a už v roce 1895 Tarry vyslovil následující 2 pravidla pro průchod labyrinty (grafy, resp. multigrafy):

1. Každou hranou můžeme projít v 1 směru maximálně jedenkrát — (zabraňuje cyklení)

2. Hranou, kterou do uzlu vstoupíme poprvé, se můžeme vrátit, až když už není jiná možnost pokračování.

Z těchto pravidel můžeme vyvodit následující vlastnosti:
— Prohledávání je konečné. To plyne z pravidla 1.
— Časová náročnost je $O(n+m)$, kde n je počet vrcholů a m je počet hran.
— Nelze–li aplikovat žádné z pravidel 1 nebo 2, znamená to, že jsme opět na začátku a každou hranou jsme prošli právě $2\times$.

Ještě v 19. století přidal k pravidlům 1 a 2 Trámaux další pravidlo:

3 Hranou, kterou jsme vstoupili do již navštíveného uzlu, se ihned vracíme zpět.

Tolik tedy historická poznámka.

5.1 Moderní algoritmy na prohledávání grafů

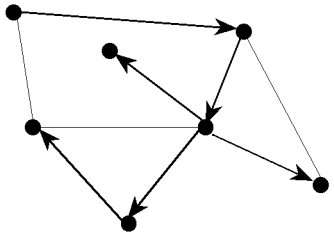
Nejpoužívanější způsoby, používané pro prohledávání grafů, jsou dva:

1. Prohledávání do hloubky (DFS - depth first search).

2. Prohledávání do šířky (BFS - breadth first search).

Předpokládejme, že prohledávaný graf je zadán takto: každý vrchol v ukazuje na seznam Adj(v) svých sousedů. Obecné zásady průchodu labyrintem pak implementujeme následujícím způsobem: navštěvované vrcholy číslujeme $1, 2, \ldots, n$ v tom pořadí jak je objevujeme. V případě, že jsme všechny vrcholy nevyčerpali, pokračujeme od prvního neobjeveného vrcholu (tj. v jiné komponentě souvislosti grafu). Při tomto postupu dostane každá hrana orientaci, šipka označuje směr průchodu. Hrany grafu můžeme rozdělit do dvou tříd: hrany *stromové*, označíme T – to jsou ty, po kterých jsme objevili nové vrcholy. Ostatní hrany nazveme *zpětné*, značíme B, viz Obr. 8.

Nyní si algoritmus popíšeme podrobněji. Začneme prohledáváním do hloubky viz algoritmus Alg. 6



Obr. 8: Průchod do hloubky. Tenké hrany jsou z B.

Hlavní program
for $x \in V$ **do** num(x) := 0;
$i := 0$; $T := B := \emptyset$
for $x \in V$ **do**
  if num(x) = 0 **then** *DFS*(x)

procedure *DFS*(v)
begin
  $i := i + 1$; num(v) := i;
  for $w \in Adj(v)$ **do**
    if num(w) = 0 **then**
      begin
        $T := T \cup (v, w)$
        DFS(w)
      end
    else if num(w) $\leq num(v)$**then**
      $B := B \cup (v, w)$
end

Alg. 6: Prohledávání grafu do hloubky

Uvedená rekurzivní procedura projde do hloubky daný graf v čase $O(n+m)$, kde n je počet vrcholů a m je počet hran.

V jistém smyslu opačný postup k prohledávání do hloubky je prohledávání do šířky. Zde nejdříve zkoumáme všechny hrany z průběžně navštíveného vrcholu v, pak po řadě všechny hrany ze sousedů vrcholu v, atd. V podstatě jde o rozklad vrcholové množiny do „pater“ podle vzdálenosti od startovacího vrcholu v, viz Obr. 9.

5.2 Aplikace metody DFS

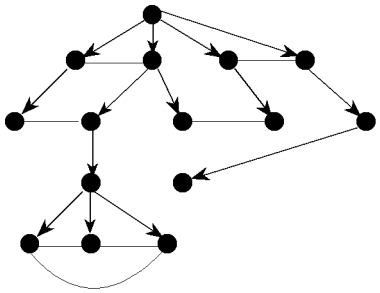
1. Hledání komponent souvislosti

2. Toplogické třídění

3. Hledání 2-souvislých komponent (vrcholově nebo hranově)

4. Hledání silně souvislých komponent

Nejjednodušší aplikací metody prohledávání do hloubky je určení počtu komponent souvislosti daného grafu G. Stačí



Obr. 9: Průchod do šířky.

si uvědomit, že množina T tvoří les v G. Jako ilustraci očíslujeme vrcholy číslem komponenty viz algoritmus Alg. 7.

```
Hlavní program
for v ∈ V do
    comp(v) := 0
c := 0
for v ∈ V do
    if comp(v) = 0 then
        c := c + 1; COMP(v)

procedure COMP(v)
begin
  comp(v) := c
  for w ∈ Adj(v) do
      if comp(w) = 0 then COMP(w)
end
```

Alg. 7: Určování komponent souvislosti

Další jednoduchou aplikací DFS nalezneme v orientovaných grafech. Zde se samozřejmě vydáváme jen po orientovaných hranách grafu G. Úlohou topologického třídění je očíslovat daný orientovaný graf tak, aby platilo: Je-li $i \to j$ hrana G pak $i < j$. Samozřejmě topologicky utřídit lze jen acyklické orientované grafy. Algoritmus Alg. 8 lehce zjistí, zda daný graf je acyklický. Používá dvojí číslování label(·)num(·).

```
Hlavní program
for x ∈ V do
    num(x) :=label(x) := 0
j := n + 1; i := 0
for x ∈ V do
    if num(x) = 0 then
        TOPSORT(x)

procedure TOPSORT(v)
begin
  i := i + 1;num(v) := i
  for w ∈ Adj(v) do
      if num(w) = 0 then
          TOPSORT(w)
      else if label(w) = 0 then
          Gje cyklický!
  j := j − 1; label(v) := j
end
```

Alg. 8: Topologické třídění

Složitější aplikací je hledání 2–souvislých komponent. Nechť $G = (E, V)$ je souvislý graf. Bod $v \in V$ nazveme artikulací, jestliže jeho vyjmutí z G zvýší počet komponent souvislosti grafu G.

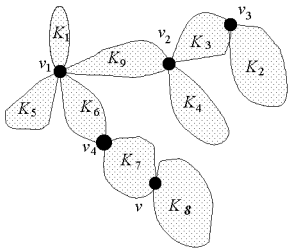
2-vrcholově-souvislá komponenta (block) K je maximální množina hran z E taková, že každé dvě hrany z K leží na prosté kružnici.

K tomu, abychom určili artikulace, využijeme následující zřejmé lemma.

Lemma 5.1 *Vrchol v je artikulací právě když existují dva vrcholy x a y a každá cesta z x do y prochází vrcholem v.*

□

Základní myšlenku algoritmu pochopíme na Obr. 10. Zde



Obr. 10: Schematické znázornění 2–souvislosti

je nakreslen graf s devíti 2–souvislými komponentami a pěti artikulacemi. Začneme–li například DFS ve vrcholu v komponentě K_9, můžeme se přes artikulaci v_2 dostat do komponenty K_4. Pak ale musíme projít celou K_4, abychom se dostali zpět do v_2. Odtud pokračujeme do další komponenty, řekněme do K_3. Tady už je situace komplikovanější, přes artikulaci v_3 můžeme pokračovat do K_2, aniž bychom prošli všechny hrany komponenty K_3. Naštěstí však procházíme strukturou, která má stromový tvar (srv. Obr. 10) a s výhodou můžeme použít zásobník na procházené hrany. Až se dostaneme zpět do komponenty K_3 budou na vrcholu již navštívené hrany K_3. Až budeme komponentu K_3 opouštět naposledy, budou v zásobníku všechny její hrany. Stačí tedy umět rozpoznávat při průchodu do hloubky artikulace.

Při průchodu budeme počítat novou funkci low(v). Její hodnotu definujeme jako nejmenší číslo num(x), kde x je předchůdce v ve stromu T a přitom existuje zpětná hrana z nějakého následníka v T vrcholu v do x (vrchol v je

sám sobě předchůdcem i následníkem). Induktivně lze low($\cdot$ $\cdot$) spočítat takto: low$(v) = \min(x, y, z)$, kde

$$\begin{aligned} x &= \min\{\text{low}(w) \mid (v,w) \in T\} \\ y &= \min\{\text{num}(w) \mid (v,w) \in B\} \\ z &= \text{num}(v). \end{aligned}$$

Nyní je zřejmé, že vrchol a je artikulace právě tehdy, když existují vrcholy $v, w \in V$ takové, že $(a,v) \in T, w \neq a, w$ není následníkem v v T a low$(v) \geq$ num(a). Jediné co tedy potřebujeme, je spočítat funkci low při průchodu do hloubky. Díky její rekurzivní definici je to již snadná záležitost viz algoritmus Alg. 9.

```
Hlavní program
i := 0; S :=prázdný zásobník
for x ∈ V do num(x) := 0;
for x ∈ V do
    if num(x) = 0 then BICON(x)

procedure BICON(v)
begin
  i := i + 1; num(v) :=low(v) := i;
  for w ∈Adj(v) do
      if num(w) = 0 then
          begin
            Push(S, (v, w));
            BICON(w);
            low(v) := min(low(v),low(w));
            if low(w) ≥num(v)then
            v je artikulace nebo kořen T,
            založíme novou komponentu,
            ze zásobníku odebereme její hrany
            až po hranu (v, w), nebo konec zásobníku
          end
      else if num(w) <num(v) then
          begin
            Push(S, (v, w));
            low(v) := min(low(v),num(w))
          end
end
```

Alg. 9: Hledání dvousouvislých komponent, výpočet funkce *Low*.

Cvičení 5.1 Most (bridge) je hrana, jejímž vyjmutím z G se zvětší počet komponent souvislosti grafu G.

2-hranově-souvislá komponenta (bridge-block) K, je maximální množina vrcholů z V taková, že každé dva vrcholy z K jsou spojeny dvěma hranově disjunktními cestami.

Modifikujte algoritmus tak, aby hledal 2-hranově souvislé komponenty, (popřípadě vyhledával i mosty).

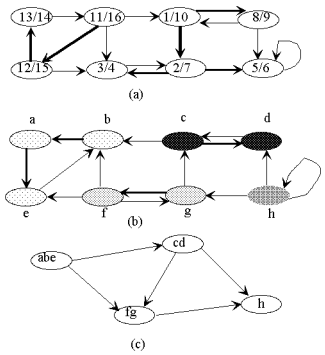
Poslední aplikací, kterou uvedeme, bude hledání silně souvislých komponent v orientovaném grafu G. Silně souvislá komponenta je maximální množina hran s vlastností, že pro libovolnou dvojici vrcholů v a w existuje orientovaná cesta z v do w a naopak z w do v. Definujeme graf G^T, který z grafu G vznikne tak, že obrátíme orientaci všech hran. Zřejmě G i G^T mají totožné silně souvislé komponenty.

Uvažme algoritmus Alg. 10, který je založený na prohledávání do hloubky. Představme si že čísluje vrcholy dvojicí čísel $(x(\cdot), y(\cdot))$, srv. (a) na Obr. 11, takto: Nechť je k dispozici inkrementální čítač, který se zvyšuje vždy po navštívení vrcholu. Poprvé když objevíme vrchol v, v přiřadíme hodnotu čítače jako první číslo $x(v)$, opouštíme–li naposledy vrchol v, přiřadíme mu druhé číslo $y(v)$.

```
begin
  Prohledej graf G do hloubky a každý
  vrchol v očísluj (x(v), y(v))
  Spočti G^T
  Prohledej do hloubky graf G^T v pořadí
    klesající posloupnosti y(·), spočítané
    v kroku 1
  Stromy v lese T jsou hledané
  silně souvislé komponenty grafu G
end
```

Alg. 10: Hledání silně souvislých komponent

Graficky je algoritmus znázorněn na Obr. 11. V části (a) silné šipky ukazují DFS les T, každý uzel v je očíslován dvojicí $(x(v), y(v))$. V části Obr. 11 je zobrazen duální graf G^T. Různými druhy rastru jsou vyznačeny stromy DFS lesa T v G^T. Část (c) ukazuje acyklický kondenzovaný graf, který vznikl ze silných komponent grafu G.



Obr. 11: Hledání silně souvislých komponent

Klíčovým pozorováním, na kterém je algoritmus založen, je fakt, že žádná cesta mezi dvěma vrcholy silně souvislé komponenty, tuto komponentu neopouští (kondenzovaný graf je acyklický). Při průchodu do hloubky tedy opouštíme silně souvislou komponentu až když ji celou projdeme a to v uzlu, ve kterém jsme do ní vstoupili. Pomocí těchto pozorování lze dokázat správnost algoritmu indukcí podle počtu nalezených stromů v lese T v grafu G^T.

Nejdůležitější aplikací prohledávání do hloubky je (dvouprůchodový) algoritmus testující rovinnost grafu, který jej

případně vnoří do roviny. Tento algoritmus by si však vzhledem ke své komplikovanosti vyžadoval samostatnou kapitolu. Prozatím nám bude stačit, že pracuje v čase lineárním v počtu vrcholů grafu (multigrafu).

5.3 Terminologie algoritmu testování rovinnosti

Prohledávání do hloubky každé hraně původně neorientovaného grafu přiřadí orientaci, zároveň jsou prohledáváním hrany každé komponenty rozděleny na stromové a zpětné.

Základní vlastností stromových hran je, že tvoří strom orientovaný z vrcholu, v němž začalo prohledávání, každá zpětná hrana (a, b) má vlastnost, že z její hlavy (z vrcholu b) vede stromová orientovaná cesta do jejího ocasu (do vrcholu a). (Takovému orientovanému grafu s rozdělením hran na stromové a zpětné, se říká palmový strom.)

Pro prvek x palmového stromu (x je hrana nebo vrchol) definujeme množinu následníků $\mathrm{desc}(x)$. Vrchol y je následníkem prvku x ($y \in \mathrm{desc}(x)$), pokud existuje orientovaná stromová cesta z x do y. Hrana (y, z) je následníkem prvku x, pokud $y \in \mathrm{desc}(x)$ nebo $(y, z) = x$. (Definice je komplikovaná, protože chceme aby zpětné hrany byly následníky. Jinak bychom mohli mluvit o podstromu.)

Pro prvek x dále definujeme množinu dotyků $\mathrm{att}(x)$. Vrchol y je dotykem x pokud je y hlavou nějaké hrany, která je následníkem x a navíc je x následníkem y. Hrana (y, z) je dotykem x, jedná-li se o zpětnou hranu a navíc je x následníkem z. Množinu $\mathrm{att}(x)$ dělíme na množiny vrcholů $\mathrm{att}^v(x)$ a hran $\mathrm{att}^e(x)$.

V prvním prohledávání do hloubky algoritmus testování rovinnosti spočítá pro každou zorientovanou hranu e při vynořování se z rekurence hodnoty $\mathrm{low}_1(e) = \min\{\mathrm{num}(y) \mid y \in \mathrm{att}^v(e)\}$ a $\mathrm{low}_2(e) = \min\{num(y) \mid y \in \mathrm{att}^v(e) \wedge \mathrm{num}(y) \neq \mathrm{low}_1(e)\}$.

Vzhledem k tomu, že chceme dosáhnout času $O(n)$, kde n značí počet vrcholů, nemůžeme si dovolit procházet všechny hrany, pokud je graf příliš hustý. Z Eulerovy věty ale víme, že v rovinném grafu nemůže být víc než $3n$ hran. Proto v průběhu algoritmu počítáme navštívené hrany, a pokud kdykoli bude trojnásobek počtu navštívených vrcholů menší než počet navštívených hran, ukončíme předčasně algoritmus s tím, že graf není rovinný. Tímto máme garantovaný čas $O(n)$ pro první fázi.

Pro druhou fázi algoritmu je potřeba připravit vhodné pořadí hran vycházejících z jednotlivých vrcholů. Ukazuje se, že vhodné pořadí je pořadí podle funkce $\varphi((a, b)) = 2\mathrm{num}((a, b)) - (\mathrm{low}_2((a, b)) \geq \mathrm{num}(a))$, kde předpokládám běžné konvence pro výsledek porovnání (pravda = 1, nepravda = 0). V mezifázi je potřeba pro každý vrchol a setřídit hrany s ocasem a vzestupně podle φ. Nechť e_1^a, e_2^a, ... je výsledný seznam, kde $\varphi(e_1^a) \leq \varphi(e_2^a) \leq \cdots$. Formálně bychom za tyto orientované hrany mohli do seznamu sousedů vrcholu a přidat všechny orientované hrany s hlavou a. Tak bychom dostali reprezentaci původního neorientovaného grafu. Při prohledávání do hloubky takto reprezentovaného grafu dostaneme stejný palmový strom jako při prvním prohledávání (cvičení).

Vzhledem k tomu, že se snažíme o algoritmus složitosti $O(n)$, nezbývá nám čas na třídění pomocí porovnávání (viz kapitola o dolních odhadech). Řešením je naalokovat si pole velikosti $2n$ (rozsah možných hodnot funkce φ), a v tomto poli na místě i uchovávat ukazatel na spojový seznam hran, kde $\varphi = i$. Postupně zařadíme všechny hrany do seznamů podle φ, poté spojíme seznamy a dostaneme tak seznam S seřazený podle φ (raději sestupně). Zatím jsme potřebovali čas $O(n)$ pro zatřídění hran do seznamů a $O(n)$ na spojení $2n$ seznamů (včetně seznamů prázdných). Abychom dostali seznamy e_i^a pro jednotlivé ocasy a, naalokujeme pole velikosti n, a v tomto poli na místě i budeme uchovávat ukazatele na spojový seznam hran s ocasem a kde $\mathrm{num}(a) = i$. Každý spojový seznam již bude setříděný podle φ. Po zařazení všech prvků seznamu S budeme mít vše připraveno pro spuštění druhého prohledávání grafu do hloubky.

Při druhém prohledávání do hloubky již budeme testovat možnost vnoření do roviny (případně tvořit vnoření). Zaveďme nyní terminologii nutnou pro druhé prohledávání. Situaci si zjednodušíme tím, že budeme testovat rovinnost zvlášť pro každou komponentu vrcholové dvousouvislosti.

Začneme definicí pojmu první cesta z prvku x. Je-li x vrchol, je první cesta x složena z x a z první cesty z e_1^x. Je-li (y, z) stromová hrana, pak je první cesta (y, z) složena z (y, z) a z první cesty ze z. Je-li (y, z) zpětná hrana, pak první cesta z (y, z) obsahuje pouze tuto hranu. Důležitým pojmem druhého prohledávání je základní cyklus $\mathrm{cycle}(x)$ daného prvku x. Základní cyklus prvku x je tvořen zpětnou hranou (y, z), do níž vede první cesta z x, a stromovou cestou ze z do y (cvičení:jednoznačnost). Cestě ze z do y se říká páteř cyklu. Páteř cyklu $\mathrm{cycle}(x)$ obsahuje první cestu z x (dvousouvislost,cvičení).

V průběhu druhého prohledávání, při práci s hranou $e = (u, v)$ budeme vytvářet strukturovanou informaci o množině $\mathrm{att}^e(e)$ na základě strukturovaných informací o $\mathrm{att}^e(e_i^v)$. Informace $\mathrm{att}^e(e)$ bude rozdělena na bloky vnoření, což jsou množiny hran z $\mathrm{att}^e(e)$, které nutně leží na stejné straně cyklu $\mathrm{cycle}(e)$. Bloky vnoření budou sdružovány do antagonistických dvojic bloků. Dva bloky jsou antagonistické, pokud hrany jednoho bloku nutně v libovolném rovinném vnoření leží na opačné straně cyklu $\mathrm{cycle}(e)$ než hrany druhého bloku. (Pokud jsou dvojice bloků (B_1, B_2) a (B_1, B_3) antagonistické, je nutně B_2 a B_3 tentýž blok.)

Bloky vnoření budeme udržovat ve struktuře, která umožní nalezení $\mathrm{first}(B) = \min\{\mathrm{num}(v) \mid (u, v) \in B\}$ a $\mathrm{last}(B) = \max\{\mathrm{num}(v) \mid (u, v) \in B\}$, spojení dvou bloků, a odstranění nespecifikované hrany, kde num hlavy je rovno $\mathrm{last}(B)$, vše v konstantním čase na operaci (k tomu stačí hrany bloku udržovat v oboustranném spojovém seznamu s rostoucím num hlav hran).

Struktura $\mathrm{att}^e(e)$ je tvořena seznamem dvojic bloků $(b(e)_1^1, b(e)_2^1)$, $(b(e)_1^2, b(e)_2^2)$, ..., $(b(e)_1^k, b(e)_2^k)$. Seznam dvojic bloků je uspořádán podle rostoucího $\mathrm{last}(b(e)_1^j)$. Dvojice bloků $(b(e)_1^j, b(e)_2^j)$ je standardizována, pokud $\mathrm{last}(b(e)_1^j) \geq \mathrm{last}(b(e)_2^j)$.