Zápočtová práce z Algoritmů a Datových Struktur I (NTIN060)

Hledání komponent silné souvislosti

David Pěgřímek

http://davpe.net

Orientovaný graf G
Je uspořádaná dvojice (V,E) kde V je množina vrcholů a E množina orientovaných hran. Počet vrcholů si značíme n, počet hran m.

Silná souvislost
Graf G je silně souvislý, pokud mezi každými dvěma vrcholy grafu vede orientovaná cesta.

Tvrzení: Nechť $x, y \in V$. Pak relace R na množině V *existuje orientovaná cesta z x do y a zároveň z y do x* je ekvivalence.
Důkaz:
(reflexivita) Pro každý $x \in V$ existuje orientovaná cesta do x a to délky 0.
(symetrie) Plyne z definice relace, neboť pro $x, y \in V$ platí $xRy \wedge yRx$.
(tranzitivita) Mějme $x, y, z \in V$. Z definice relace plyne pokud $xRy \wedge yRz$, pak i xRz.

Tuto relace můžeme rozložit na třídy ekvivalence, kterým říkáme *komponenty silné souvislosti* (v následujícím textu jim budu zkráceně říkat jen *komponenty*).

Algoritmus na hledání silně souvislých komponent publikoval v roce 1972 americký informatik Robert Tarjan [1]. Je založen (stejně jako většina algoritmů na hledání silně souvislých komponent) na prohledávání do hloubky. Pro jeho pochopení si zavedeme pár pojmů a tvrzení.

pole `dfsn`
Hodnota `dfsn`[v] je pořadové číslo, ve kterém DFS vrchol v objevilo.

Předchůdce vrcholu v
Vrchol w, pro který platí `dfsn`[w] $\leq$ `dfsn`[v]. Analogicky se definuje *následník* vrcholu v.

Kořen silně souvislé komponenty
Vrchol r takový, že všechny ostatní vrcholy v téže komponentě jsou následníci vrcholu r. Vrchol r má navíc nejmenší hodnotu `dfsn` ze všech vrcholů s touto vlastností.

pole `mark`
Hodnota `mark`[v] je 0, když je vrchol v neobjeven, -1 když je v na komponentovém zásobníku. Hodnota > 0 určuje pořadové číslo komponenty, ve které leží v.

pole `link`
Hodnota `link`[v] je minimum z `dfsn`[v] a množiny `dfsn`[w] vrcholů w takové, že z následníka vrcholu v vede hrana do w, který už byl objeven. Navíc kořen r komponenty obsahující w je předchůdce vrcholu v. Pole `link` slouží k nalezení kořenů komponent.

Lemma: Vrchol v je kořen silně souvislé komponenty právě tehdy když `link`[v] = `dfsn`[v].
Důkaz:
$\rightarrow$ (sporem)
Nechť v je kořen komponenty, z definice `link`u víme, že `link`[v] $\leq$ `dfsn`[v]. Nechť pro spor `link`[v] $<$ `dfsn`[v]. Z definice `link`u rovněž existují vrcholy r a w, že r je předchůdce w a zároveň w je předchůdce v. Máme tedy nerovnosti `dfsn`[r] $\leq$ `dfsn`[w] $\wedge$ `dfsn`[w] $<$ `dfsn`[v], z čehož vyplývá, že `dfsn`[r] $<$ `dfsn`[v] a vrchol r je tedy předchůdce vrcholu v. Navíc, tyto dva vrcholy jsou ve stejné komponentě, neboť vede cesta z r do v a zároveň z v do r (přes w). Čili v nemůže být kořen komponenty (neboť má jako předchůdce vrchol r), což je spor.
$\leftarrow$ (sporem)
Nechť `link`[v] = `dfsn`[v] a pro spor v není kořenem komponenty. Pak existuje vrchol r, který je kořenem komponenty a vrchol v je jeho následník a tedy existuje cesta P z v do r. Podívejme se na první hranu cesty P, která vede od nějakého následníka vrcholu v do vrcholu w, který není následník v. Vrchol w byl navštívený dříve než v, proto `dfsn`[w] $<$ `dfsn`[v]. Vrchol w navíc leží ve stejné komponentě jako r,

neboť w leží na cestě z v do r a rovněž musí existovat cesta z r do v (neboť r je kořenem komponenty). Čili `link`[v] $\leq$ `dfsn`[w] $\wedge$ `dfsn`[w] $<$ `dfsn`[v] a tedy `link`[v] $<$ `dfsn`[v], což je spor.

Tarjanův algoritmus: popis
Pro každý vrchol v, který nemá přiřazené číslo komponenty se provede prohledání do hloubky při kterém se vrcholy ukládají na komponentový zásobník. Pokud narazíme na vrchol, který už jsme jednou viděli, ale který nemá přiřazenou komponentu, tak si DFS číslo tohoto vrcholu poznamenáme do pole `link`. Při návratu z DFS propagujeme nejmenší hodnotu `link`u. Pokud při návratu narazíme na kořen r komponenty, vypíšeme celou komponentu určenou kořenem r díky zásobníku.

Tarjanův algoritmus: pseudokód

```
component_idx ← 1
dfs_idx ← 1
mark[1..n] ← 0

for i = 1 to n do
  if mark[i] = 0 then
    FindSCC(i)
  end if
end for
return

FindSCC(v)

    link[v] ← dfsn[v] ← dfs_idx++
    Stack.push(v)
    mark[v] ← −1

    for {w | (v,w) je hrana grafu G} do
      if mark[w] = 0 then
        FindSCC(w)
        link[v] = Min(link[v], link[w])
      else if mark[w] = -1 then
        link[v] = Min(link[v], dfsn[w])
      end if
    end for

    if link[v] = dfsn[v] then
      repeat
        x ← Stack.pop()
        mark[x] ← component_idx
      until x=v
      component_idx++
    end if
```

Časová složitost
Funkce FindSCC se zavolá pro každý vrchol právě jednou. Vyprazdňování komponentového zásobníku nás v celém algoritmu stojí $O(n)$, neboť každý vrchol je přidán do zásobníku právě jednou. Probírání všech sousedů vrcholu v, trvá $O(deg(v))$, což je v celém algoritmu $O(m)$. Celková časová složitost Tarjanova algoritmu je tedy $O(n + m)$.

Paměťová složitost
V algoritmu používáme zásobník pro vrcholy (každý se v zásobníku objeví právě jednou) a dále si pro každý vrchol potřebujeme pamatovat jeho značku, dfs index a link. Celkově spotřebujeme $O(n)$ paměti.

Správnost
a) Pořadí vrcholů v komponentovém zásobníku je korektní
Poté, co při návratu z prohledávaní do hloubky narazíme na kořen r (z lemmatu víme, že to skutečně kořen je) přiřadíme do jedné komponenty všechny vrcholy, které jsou v zásobníku nad r. V grafu jsou to následníci vrcholu r. Tyto vrcholy neleží v žádné jiné komponentě (kdyby už měly přiřazenou komponentu, nebyly by na zásobníku).
b) Hodnoty `link` jsou spočteny správně
Hodnoty `link`[v] počítáme podle definice: jakmile nějaký následník vrcholu v vede do vrcholu w, který je na zásobníku, poznamenáme si jeho `dfsn` a propagujeme do vrcholu v nejmenší hodnotu `dfsn` všech takových vrcholů w.

Reference

[1] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146-160, 1972