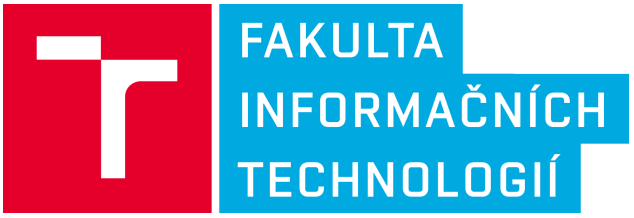



Vysoké učení technické v Brně

GAL

Grafové algoritmy

---

# Projektová dokumentace
## Testování rovinnosti grafu

---

*Autor:* Petr Buchal
*Login:* xbucha02
Vladan Kudláč
xkudla15

15. prosince 2019

# 1 Úvod

Graf je rovinný právě tehdy, když pro něj existuje takové rovinné nakreslení, ve kterém se žádná z hran nekříží. Motivací nalezení takového nakreslení je přehledná vizualizace, použití pro návrh plošných spojů, apod [1].

Existují dva základní teorémy, které říkají, kdy je graf rovinný. Kuratowski's theorem, který říká, že graf je rovinný právě tehdy, když graf neobsahuje podgraf, který vznikl dělením hran z grafu K5 nebo $K_{3,3}$. Wagner's theorem říká, že graf je rovinný právě tehdy, když graf neobsahuje podgraf, který by byl minor grafu K5 nebo $K_{3,3}$. Implemetace těchto teorémů má exponenciální asymptotickou časovou složitost.

Dále existuje Eulerův vztah $E \leq 3V - 6$, který platí pro rovinné grafy s alespoň třemi hranami. Podmínka je efektivní, ale není postačující. Je vhodné otestovat graf touto podmínkou a poté spustit úplný algoritmus. Podmínku nevyužíváme, aby nedošlo ke zkreslení výsledků.

# 2 Left-right algoritmus

Left-right algoritmus (1982) je založen na prohledávání do hloubky (DFS), někdy též Fraysseix–Rosenstiehl planarity criterion. Na rozdíl od jiných algoritmů nevyžaduje složité datové struktury, implementovat lze s využitím polí. Algoritmus si poradí s biconnected komponentami i nesouvislým grafem. Algoritmus lze využít nejen k zjištění rovinnosti ale i k získání rovinného nakreslení. Časová složitost DFS průchodů je O(E). Mezi průchody dochází k řazení, výsledná složitost závisí na řadicím algoritmu. Při použití bucket search lze dosáhnout lineární časové složitosti [2].

## 2.1 Vstupní tvar grafu

Left-right algoritmus předpokládá neorientovaný graf. Pokud je na vstupu orientovaný graf, je nutné jej nejprve symetrizovat. Pokud je mezi 2 vrcholy více hran, lze uvažovat pouze jednu, v případě rovinného nakreslení by hrany byly rovnoběžné a neovlivnily by rovinnost. Smyčky nemohou ovlivnit rovinnost grafu, navíc nejsou v neorientovaném grafu povoleny, je tedy nutné je odstranit před spuštěním LR algoritmu. Graf nemusí být spojitý a může obsahovat vrcholy stupně 1. Graf může být uložen jako seznam sousedů i maticí sousednosti, volba uložení ovlivní pouze způsob procházení v 1. DFS. Aktuálně je algoritmus implementován se vstupním grafem zadaným seznamem sousedů.

## 2.2 Algoritmus

Algoritmus se skládá ze tří průchodů grafem modifikovanými DFS algoritmy. Při prvním průchodu vytvoří orientovaný graf obsahující stromové a zpětné (konfliktní) hrany. K uzlům si ukládá výšku (vzdálenost od kořene) a ukládá si kořenové uzly. U hran zjistí hodnotu zanoření (hrany ležící na cyklu procházejícím kořenovým uzlem mají nejnižší hodnotu, hrany cyklů dále od kořene mají hodnotu vyšší) a výšku nejnižšího uzlu v cyklu (lowpoint).

Při druhém průchodu prochází graf od kořenů postupně po hranách seřazených vzestupně dle hodnoty lowpoint. Algoritmus se snaží najít Left/Right uspořádání zpětných hran – zpětné hrany vedou směrem ke kořenu buď nalevo nebo napravo od stromové hrany. Při objevení nové zpětné hrany je vložena na seznam konfliktních dvojic. Při navracení se konfliktní dvojice slučují do dvojice konfliktních intervalů.

Při třetím průchodu vytváří rovinné nakreslení. Pro testování rovinnosti není třetí průchod potřebný. [2][3]

# 3 Metoda přidávání cest

Hopcroft-Tarjanův algoritmus neboli metoda přidávání cest (angl. Path addition method) je prvním algoritmem, který dokázal testovat planaritu algoritmu s lineární časovou složitostí O(E) [5]. Vytvořen byl v roce 1974.

## 3.1 Vstupní tvar grafu

Během testování planarity Hopcroft-Tarjanovým algoritmem počítáme s určitou podobou grafu. Prvním vlastností testovaného grafu je to, že je neorientovaný. Můžeme testovat i orientovaný graf, ale ten je třeba nejdříve symetrizovat. O symetrizaci grafu se stará metoda *symetrize* ve třídě *Graph*. Dále je třeba, aby algoritmus neobsahoval více než jednu hranu mezi dvěma vrcholy. O tuto podmínku se implicitně stará třída *Graph*. Dále graf nemůže obsahovat vrcholy stupně 1, ty jsou před použitím algoritmu vymazány metodou *remove_vertices_of_degree_1*. A nakonec co se vstupního tvaru grafu týče, graf nemůže obsahovat smyčky, ty se odstraňují metodou *remove_self_loops*. Algoritmus zároveň vyžaduje uložení grafu pomocí seznam sousedů [2].

## 3.2 Algoritmus

Po převedení grafu na požadovaný tvar podle vstupních podmínek se pomocí BFS (metoda *get_disconnected_components*) kontroluje jestli je graf souvislý, pokud není,

zkoumá se u planarita u každé komponenty zvlášť. Je-li alespoň jedna komponenta neplanární, celý graf není planární. Následně se pro zkoumanou komponentu (dále jen graf) pomocí DFS zkoumá jestli je bi-souvislý (bi-connected). O to se stará metoda *get_biconnected_components*, které poskytuje artikulační body metoda *DFS*. Pokud není bi-souvislý, lze každou bi-souvislou komponentu testovat na planaritu zvlášť a je-li alespoň jedna bi-souvislá komponenta neplanární, celý graf není planární. Pokud je graf bi-souvislý testuje se na planaritu celý [4]. Následně algoritmus uspořádá hrany grafu do vhodného pořadí pro testování rovinnosti. Poté zvolíme počáteční hranu, z níž vytvoříme cyklus a následně testujeme silnou rovinnost segmentů hran vystupujících z daného cyklu. Nakonec algoritmus testuje zda-li lze sloučit nakreslení grafů, tak aby výsledek zůstal silně rovinný [3]. Pseudokód algoritmu je na obrázku 1.

**Algorithm 1** Planarity by Hopcroft and Tarjan

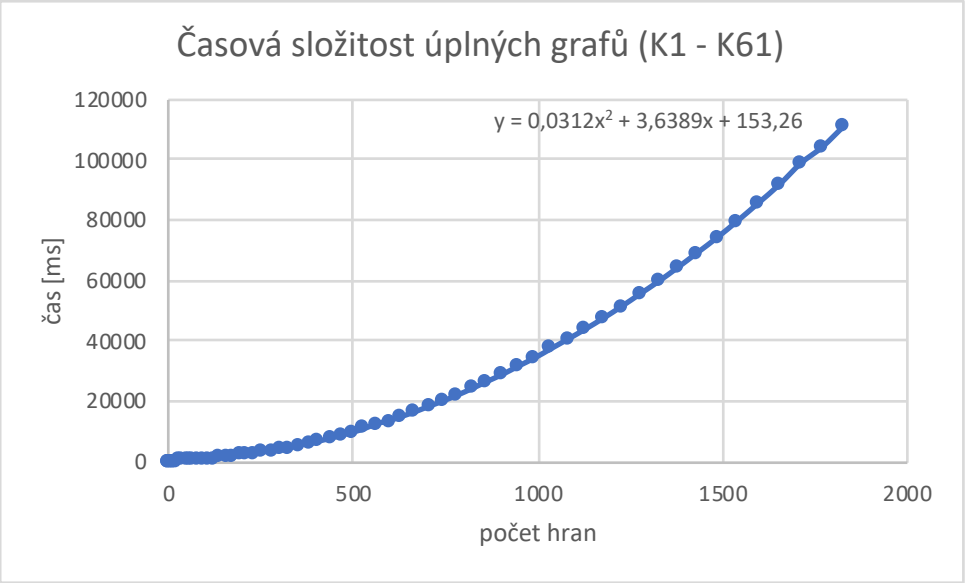
```
1:  procedure PLANARITY(G)
2:      E ← 0;
3:      for each edge of G do
4:          E ← E + 1;
5:          if E > 3V − 3 then
6:              goto nonplanar;
7:      divide G into biconnected components;
8:      for each biconnected component G do
9:          run DFS on G for numbering;
10:         transform G into palm tree P;
11:         find a cycle c in P;
12:         construct planar representation for c;
13:         for each segment after deletion of c do
14:             apply recursively to all segments;
15:             if segment plus cycle c is planar and
16:             segment can be added to embedding
17:             then
18:                 add segement to planar embedding;
19:             else
20:                 goto nonplanar;
```

Obrázek 1: Pseudokód algoritmu přidávání cest [1].

# 4 Závěr

Pro testování časové složitosti na hustých grafech jsme používali úplné grafy K1 až K4, které jsou rovinné a K4 až K61, které nejsou rovinné. Obrázek 2 ukazuje, že implementovaný algoritmus LR není lineární vůči počtu hran. Po proložení

výsledků vyplývá z rovnice spojnice asymptotická složitost $O(n^2)$. Odchylka od lineární asymptotické složitosti je dána řadicím algoritmem.




Obrázek 2: Časová složitost LR algoritmu

# 5 Zdroje

1. SARVOTTAMANANDA, Swami. Planarity Testing of Graphs [online]. Ramakrishna Mission Vivekananda University, 2011 [cit. 2019-12-15]. Dostupné z: cs.rkmvu.ac.in/~sghosh/public_html/nitp_igga/slides/shreesh-planarity-patna.pdf
2. BRANDES, Ulrik. The Left-Right Planarity Test [online]. In: . 2009 [cit. 2019-12-15]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.217.9208
3. EMERY, Dylan. Plane and Simple: Characterizing and Testing Planarity [online]. In: . 2018 [cit. 2019-12-15]. Dostupné z: https://scholarship.tricolib.brynmawr.edu/bitstream/handle/10066/20807/2018EmeryD.pdf?sequence=1&isAllowed=y

# Reference

[1] Planarity testing by path addition : Martyn g taylor.

[2] Planarity testing of graphs: Swami sarvottamananda.

[3] Rozšiřování knihovny pro grafové algoritmy: Karolína rezková.

[4] Testování rovinnosti grafu: Jaroslav ciml.

[5] Tung Hoang. Graph planarity and path addition method of hopcroft-tarjan for planarity testing, Nov 2018.