VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

GRAFOVÉ ALGORITMY

2020/2021

Téma 27 - Porovnání Testování rovinnosti grafu

Daniel Dolejška (xdolej08)
Tomáš Odehnal (xodehn08)

Brno, 17. prosince 2020

1 Úvod

Cílem tohoto projektu byla implementace a experimentální porovnání algoritmů pro testování rovinnosti grafů. Rovinnost grafu je vlastnost definována jako možnost zakreslení grafu do roviny tak, že se žádné dvě hrany daného grafu nebudou křížit. [1] Pro porovnání byly zvoleny Left-Right Planarity Test a Hopcroft-Tarjan algoritmus.

2 Implementace a použité knihovny

Implementace programu byla provedena v jazyce C++. Použita byla pouze knihovna OGDF[1]. Tato knihovna poskytuje značné množství již implementovaných algoritmů, a to i včetně algoritmů pro testování rovinnosti grafů, avšak využito bylo pouze metod, algoritmů a struktur pro základní operace s grafy. Především bylo využito náhodné generování, načítání a ukládání reprezentací grafů, datových struktur pro práci s grafy v programu a jednoduchých operací typu vytvoření indukovaného podgrafu, apod.

Využítím knihovny OGDF má program možnost pracovat s grafy ve formátech `GML`, `GML/XML` či `GraphML`. To umožňuje zobrazení, generování i úpravu grafů v oblíbených grafových editorech.

3 Spuštění programu

Zobrazení nápovědy programu:

```
$ ./gal_projekt -h
```

Vygenerování rovinného grafu s 10 uzly a 20 hranami:

```
$ ./gal_projekt -g -p -n 10 -m 20 -o Graphs/planar10_20.gml
```

Počet hran se může rovnat maximálně $3|V| - 6$. Pokud bude zadána větší hodnota, tak počet hran bude omezen na hodnotu odpovídající předchozímu výrazu.

Vygenerování nerovinného grafu s 10 uzly (počet hran je automaticky vypočítán podle výrazu $3|V| - 6$):

```
$ ./gal_projekt -g -n 10 -o Graphs/non_planar10.gml
```

Otestování, jestli je daný graf rovinný pomocí algoritmu Left-Right:

```
$ ./gal_projekt -a left-right -f Graphs/planar10_20.gml
```

Výstup programu je informace, jestli je graf rovinný.

Otestování, jestli je daný graf rovinný pomocí algoritmu Hopcroft-Tarjan:

```
$ ./gal_projekt -a hopcroft-tarjan -f Graphs/planar10_20.gml
```

Výstup programu je informace, jestli je graf rovinný.

Získání času (v sekundách) doby testování grafu:

```
$ ./gal_projekt -a hopcroft-tarjan -t -f Graphs/planar10_20.gml
```

[1] `https://ogdf.uos.de/`

4 Algoritmus Hopcroft-Tarjan

Jedná se o první publikovaný algoritmus, který dokázal ověřit rovinnost grafu v lineárním čase v závislosti na počtu uzlů a hran grafu — $\mathcal{O}(|V| + |E|)$. [1]

Algoritmus na vstupu vyžaduje 2-souvislý graf. V případě, kdy graf není 2-souvislý, je v lineárním čase rozložen na 2-souvislé komponenty (podgrafy), které jsou postupně ověřeny nezávisle na sobě. Pokud jsou všechny 2-souvislé podgrafy rovinné, pak je i původní graf rovinný. Implementace algoritmu dále předpokládá neorientovaný graf. Všechny grafy jsou ovšem reprezentovány jako orientované — orientace hran je ovšem algoritmem přímo ignorována a všechny hrany jsou považovány za neorientované. [2, 3, 1]

Při testování jednotlivých komponent je nejdříve sestrojen strom DFS-průchodu grafem, přiřazeny DFS-identifikátory uzlů, vypočteny dva nejnižší identifikátory dosažitelných uzlů a určeny typy (stromové, zpětné) jednotlivých hran. Následně algoritmus vyhledá veškeré existující cykly v daném DFS stromu — cyklem je myšlena posloupnost stromových hran zakončená hranou zpětnou. Kritickou částí algoritmu je správné seřazení hran v seznamech sousednosti všech uzlů grafu. Algoritmus poté zvolí první existující hranu, nalezne odpovídající cyklus, kterého je hrana součástí, a rekurzivně kontroluje rovinnost jednotlivých segmentů. V průběhu kontroly je "sestrojován" rovinný podgraf postupným přidáváním ověřených segmentů grafu. Hrany uzlů, které nejsou součástí právě kontrolovaného cyklu jsou postupně přidávány buď na pravou či levou stranu daného cyklu, kde se na základě cílového uzlu hrany určuje, zda-li je možné hranu do aktuálního rovinného podgrafu vložit tak, aby podgraf zůstal rovinný či ne. Pokud je možné všechny jednotlivé hrany postupně přidat, je graf rovinný, při selhání kontroly segmentu je algoritmus ukončen a zdrojový graf označen za nerovinný. [2, 3, 1]

Jak rozklad na 2-souvislé komponenty, modifikovaná aplikace DFS, tak i rekurzivní průchod přes existující hrany a uzly je lineární. [1]

5 Left-Right algoritmus

Left-Right algoritmus na testování rovinnosti grafu, také označovaný jako de Fraysseix–Rosenstiehl planarity criterion je založen na prohledávání grafu do hloubky (DFS). Algoritmus na rozdíl od jiných algoritmů na testování rovinnosti grafu nevyžaduje žádné složité datové struktury (např. algoritmus Booth and Lueker 1976). Lze implementovat pomocí polí a zásobníku. Dokáže testovat i nesouvislé grafy.[4]

Left-right algoritmus na testování rovinnosti grafu staví na skutečnosti, že cykly v grafu mohou mít za následek, že se hrany budou křížit. Cyklus uzavírá oblast grafu a tím tak vytváří dva disjunktní regiony plochy (na kterou bychom graf nakreslili). Musíme řešit, jestli zbývající části grafu se nachází uvnitř nebo vně oblasti, kterou ohraničuje cyklus. Pokud graf není rovinný, tak dochází k částečnému překrývání cyklů v grafu. Testování rovinnosti grafu pomocí tohoto algoritmu je určování, jestli jsou cykly v grafu konzistentní. To znamená, že všechny zpětné hrany vycházející ze stromových hran, jsou na stejné straně. K rychlému nalezení cyklů se používá DFS. [4, 5]

Asymptotická časová složitost tohoto algoritmu je $\mathcal{O}(|V| + |E|)$, kde $|E|$ musí splňovat podmínku $|E| \leq 3|V| - 6$. Pokud tato podmínka není splněna, graf nemůže být planární.

První průchod grafem pomocí DFS má časovou složitost $\mathcal{O}(|V| + |E|)$. Tato fáze algoritmu vytvoří z neorientovaného grafu orientovaný a určí typy hran (stromové nebo zpětné). Po této fázi získáme ohodnocení ke každé hraně tzv. *nesting_depth*.

Před druhým průchodem grafu je potřeba vzestupně seřadit odchozí hrany všech uzlů podle hodnot *nesting_depth* pro hrany, které jsou přímo svázané s hodnotami *lowpoint*. *Lowpoint* hodnota hrany je hodnota nejnižšího návratového bodu (uzel s nejmenší vzdáleností od kořene DFS stromu). V [4] je uvedeno, že lze použít Counting sort nebo Bucket sort. Vzhledem k tomu, že přímo v pseudokódu algoritmu bylo uvedeno, že lze použít Counting sort, tak jsme použili Counting sort. Řazení má lineární časovou složitost $\mathcal{O}(|V| + |E|)$.

Druhý průchod grafem má stejnou stejnou asymptotickou časovou složitost jako první průchod grafem. Nyní se ale graf (hrany) prochází vzhledem k uspořádání, které bylo provedeno v předchozím kroku. Graf je procházen od kořene (kořenů) DFS stromu a algoritmus se snaží najít Left-Right rozdělení hran, což je dáno rozdělením zpětných hran. Algoritmus vkládá hrany do konfliktních dvojic, což reprezentuje Left-Right rozdělení. A graf je planární pokud respektuje Left-Right rozdělení.

Algoritmus ještě obsahuje třetí průchod grafem, kde se vytváří rovinná reprezentace grafu. To již ale není obsahem zadání (máme pouze testovat rovinnost). Tedy tuto část jsme již neimplementovali. [4]

Výše je uvedeno, že algoritmus lze implementovat pouze pomocí polí a zásobníku. My jsme vzhledem k ulehčení implementace použili třídy pro intervaly hran a páry konfliktních intervalů.

6 Testování a experimenty

Oba algoritmy jsme otestovali pomocí skriptu `test_algorithm.sh`, který se spouští se dvěma argumenty:

1. První argument je určení, který algoritmus chceme testovat. Povolené hodnoty: `left-right` a `hopcroft-tarjan`.
2. Druhý argument je určení, na kterých grafech chceme zvolenou metodu testovat. Povolené hodnoty: `planar` a `non-planar`.

Skript vygeneruje graf a otestuje, jestli zvolený algoritmus správně rozhoduje tento graf. Následně provede spuštění algoritmu za účelem získání času, který algoritmus pro testování rovinnosti potřeboval.

Pro experimenty (a testování) jsme vygenerovali rovinné a nerovinné grafy, které měly počet uzlů od 10 do 1000 a počet těchto uzlů jsme inkrementovali po 10. Počet hran u každého grafu (rovinného i nerovinného) byl maximální možný, aby byla splněna podmínka $|E| \leq 3|V| - 6$. Proto je počet hran u každého grafu vypočítán z hodnoty z počtu uzlů podle vzorce $3|V| - 6$. (Například rovinný graf s 10 uzly má 24 hran a tento počet hran je stejný jako u nerovinného grafu s 10 uzly.) Vygenerovali jsme tedy nakonec 100 rovinných a 100 nerovinných grafů. Pro získání času jsme algoritmus spustili na každém grafu 100-krát. Toto jsme udělali z důvodu, že jednotlivé běhy algoritmu na stejném grafu mohou trvat různě dlouhou dobu, ale měly by se pohybovat kolem nějaké střední hodnoty. Z těchto 100 hodnot času jsme vypočítali průměr, který jsme následně použili v grafech.

Z důvodu, že soubory vygenerovaných grafů mohou být poměrně obsáhlé (např. soubor pro graf s 1000 uzlů má velikost přes 1MB.

Oba algoritmy byly tedy otestovány na grafech se stejným počtem uzlů a hran, aby nám získané časy lépe tyto algoritmy porovnaly.

Pro otestování a získání časových hodnot do souboru spustíme skript s následujícími parametry a přesměrujeme standardní výstup do csv souboru:

```
$ ./test_algorithm.sh <algorithm> <graph> 1> <algorithm>_<graph>.csv
```

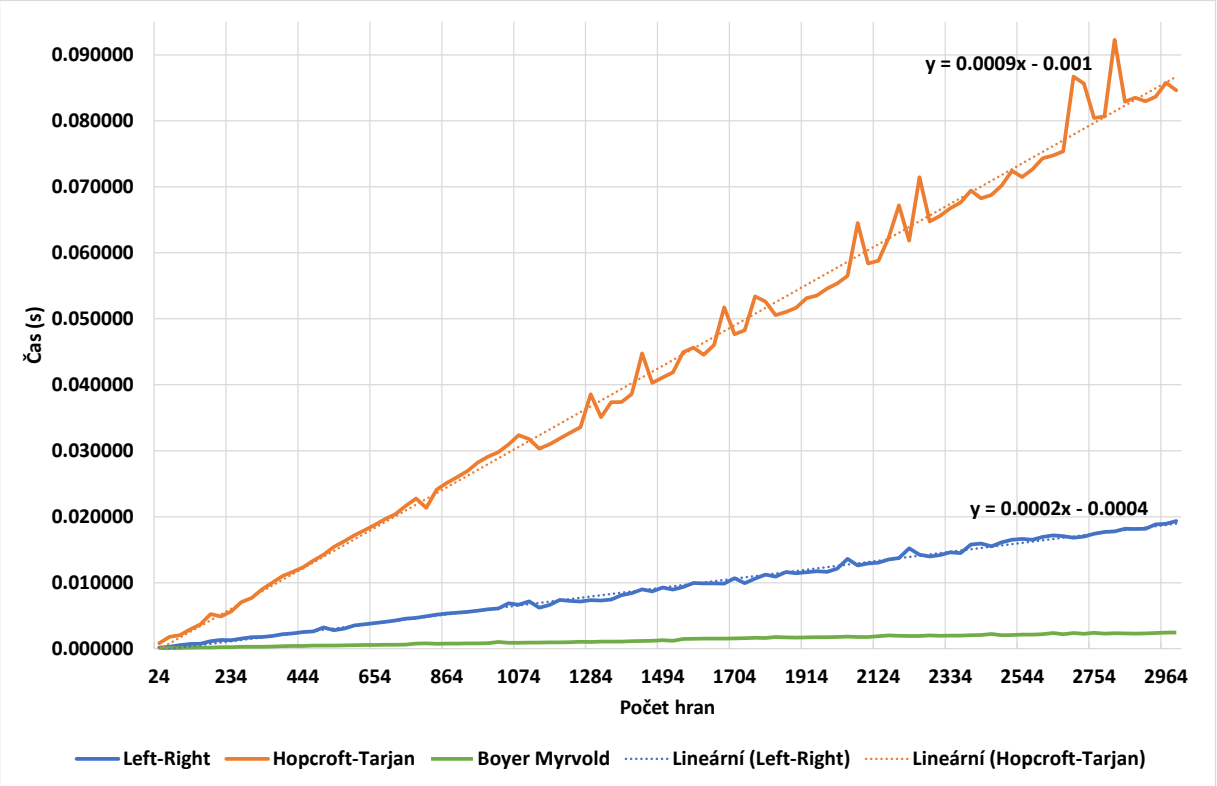
Kde `<algorithm>` je daný zvolený algoritmus a `<graph>` je typ testovaných grafů (viz výše).

Na obhajobě projektu jsme dostali připomínku, že jsme mohli srovnat implementované algoritmy s algoritmem z knihovny OGDF. Knihovna OGDF používá jako implcitní algoritmus Boyer-Myrvold (viz [6]) (dále referenční algoritmus). Tedy stejné testy a experimenty, které jsme provedli pro naše implementované algoritmy jsme použili i pro tento algoritmus. Výsledky jsme zahrnuli do grafů níže. Experimenty pro tento algoritmus lze spustit pomocí skriptu výše, kde `<algorithm>` je rovno `ref`.

6.1 Výsledky experimentů pro rovinné grafy

Jak lze vidět z obrázku č. 1 naměřené časy u obou algoritmů vzrůstají lineárně vzhledem k rostoucímu počtu hran (a i uzlů). Tedy teoretická lineární časová složitost odpovídá naměřeným časům. Navíc protože pro rovinné grafy musí oba algoritmy projít úplně celý graf dá se říci, že časy odpovídají teoretické složitosti $\Theta(|V| + |E|)$. Ovšem i když oba algoritmy mají lineární časovou složitost, algoritmust Left-Right ji má méně vzrůstající než algoritmus Hopcroft-Tarjan.

Referenční algoritmus dosahuje lepších výsledků než naše implementované algoritmy.



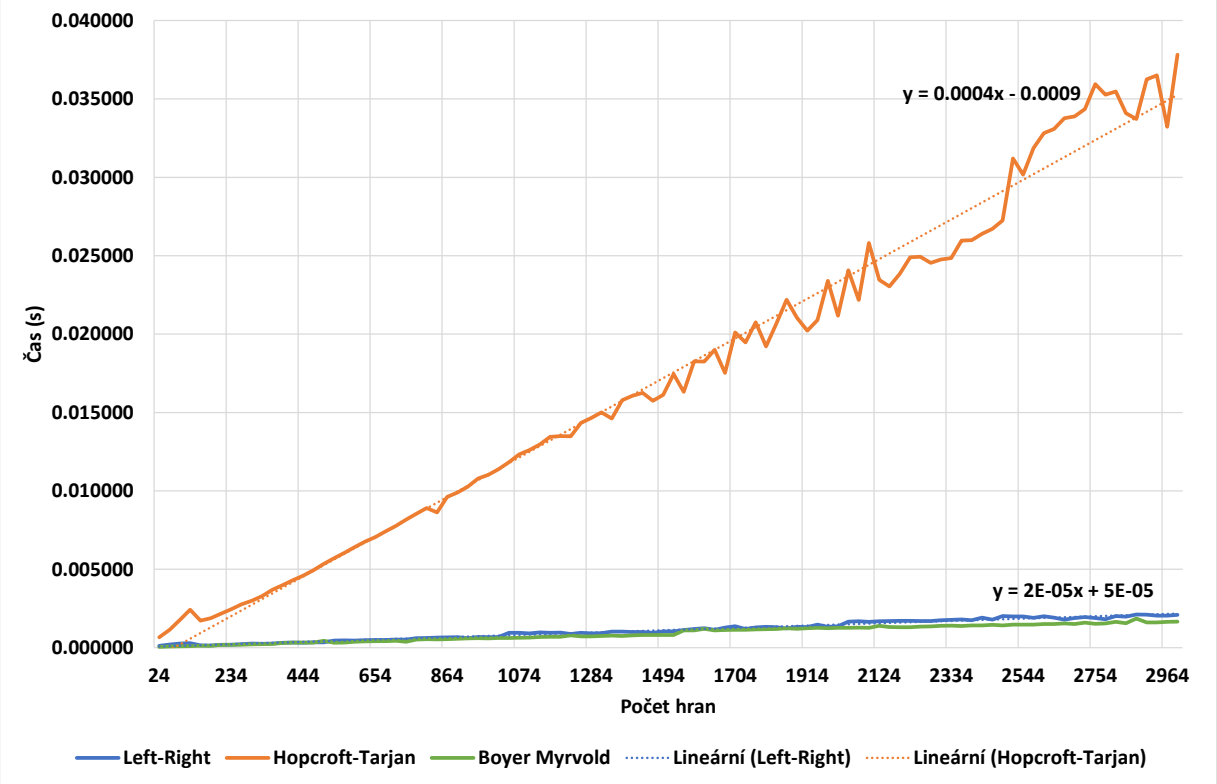
Obrázek 1: Provnání LR a Hopcroft-Tarjan algoritmů pro rovinné grafy.

6.2 Výsledky experimentů pro nerovinné grafy

Jak lze viděž z obrázku č. 2 naměřené časy u obou algoritmů (stejně jako v případě rovinných grafů) vzrůstají lineárně vzhledem k rostoucímu počtu hran. Tedy i zde teoretická lineární časová složitost odpovídá naměřeným časům. Podle očekávání jsou u obou algoritmů naměřené časové hodnoty menší než, hodnoty z experimentů pro rovinné grafy. Je tomu proto, protože algoritmy nemusí procházet celý graf, jak tomu je u rovinných grafů. Jakmile totiž zjistí, že není splněna některá z jejich interních podmínek, prohlásí, že graf není planární a algoritmus končí. Časová složitost u nerovinných grafů je $\mathcal{O}(|V|+|E|)$.

Zde nás překvapilo, že naměřené časy u algoritmu Left-Right jsou menší o celý řád. U algoritmu Hopcroft-Tarjan jsou naměřené časy menší jen o víc než dvojnásobek.

Referenční algoritmus dosahuje lepších výsledků než algoritmus Hopcroft-Tarjan, ale Left-Right algoritmus dosahuje přibližně stejných výsledků.



Obrázek 2: Provnání LR a Hopcrof-Tarjan algoritmů pro nerovinné grafy.

7 Závěr

Experimenty ukázaly, že oba implementované algoritmy měly pro rovinné a nerovinné grafy naměřené časy lineární vzrůst. Tedy lze oba algoritmy v praxi použít. Ovšem z výsledků experimentů je zřejmé, že lepší je použít algoritmus Left-Right, protože doba běhu algoritmu má menší vzrůstající tendenci vzhledem k počtu hran (uzlů) v grafu.

Referenční algoritmus pro planární grafy dosahuje lepších výsledků, ovšem pro neplanární grafy dostahuje velice podobných výsledků jako Left-Right algoritmus.

Reference

[1] Frank Rubin. A search procedure for hamilton paths and circuits. *Journal of the ACM (JACM)*, 21(4):576–580, 1974.

[2] Kurt Mehlhorn. An implementation of the hopcroft and tarjan planarity test and embedding algorithm. Research Report MPI-I-93-151, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, October 1993.

[3] Swami Sarvottamananda. Planarity testing of graphs. `http://cs.rkmvu.ac.in/~sghosh/public_html/nitp_igga/slides/shreesh-planarity-patna.pdf`, 2011. (Accessed on 11/19/2020).

[4] Ulrik BRANDES. The left-right planarity test. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.9208&rep=rep1&type=pdf`, 2009. (Accessed on 11/19/2020).

[5] Dylan EMERY. Plane and simple: Characterizing and testing planarity. `https://scholarship.tricolib.brynmawr.edu/bitstream/handle/10066/20807/2018EmeryD.pdf?sequence=1&isAllowed=y`, 2018. (Accessed on 11/19/2020).

[6] The open graph drawing framework. `https://ogdf.uos.de/doc/index.html`, 2020. (Accessed on 12/17/2020).