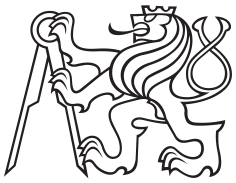
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Rozšiřování knihovny pro grafové algoritmy

Karolína Rezková

Vedoucí práce: RNDr. Marko Genyk-Berezovskyj
Květen 2017

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Karolína R e z k o v á

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Rozšiřování knihovny pro grafové algoritmy

Pokyny pro vypracování:

1. Proveďte rešerši rozšířených a volně dostupných grafových algoritmických knihoven. Podle své oblasti zájmu si zvolte specifickou knihovnu, seznamte se s její výstavbou a rozšiřte její funkcionalitu.
2. Otestujte důležité algoritmy teorie grafů v knihovně již implementované, vyhodnoťte jejich deklarovanou a skutečnou efektivitu a případně navrhněte a implementujte úpravy vedoucí k časové nebo paměťové úspoře.
3. Identifikujte téma, které v knihovně výrazněji chybí, a implementujte základ pro jeho další rozšiřování buď přímo v samotné knihovně nebo vazbami na další vnější SW řešící úlohy stejného charakteru. Možnými kandidáty jsou planarita, izomorfizmus, uživatelské nebo programátorské rozhraní.
4. Všechny úpravy knihovny doprovoďte programátorskou a uživatelskou dokumentací.

Seznam odborné literatury:

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms, 3rd ed., MIT Press, 2009
[2] J. Matoušek, J. Nešetřil: Kapitoly z diskrétní matematiky, Karolinum, 2010
[3] R. Sedgewick: Algorithms in C Part 5: Graph Algorithms (3rd Edition), Addison-Wesley Professional, 2002
[4] J. Demel: Grafy a jejich aplikace, Praha, Academia, 2002

Vedoucí bakalářské práce: RNDr. Marko Genyk-Berezovskyj

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 18. 12. 2015

Poděkování

Chtěla bych poděkovat všem, kteří mi pomohli při psaní této práce, obzvláště RNDr. Marku Genyku-Berezovskému za umožnění práce na tomto tématu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2017

Abstrakt

Tato bakalářská práce je zaměřena na open-source knihovnu pro grafové algoritmy jGraphT. Nejprve porovnává tuto knihovnu s dalšími dostupnými knihovnami, které se taktéž soustředí na matematické grafy. Následně seznamuje čtenáře s obsahem knihovny a jeho strukturou. Ve své poslední části rozšiřuje obsah knihovny o test, který detekuje rovinnost zadaného grafu. Za tímto účelem je popsán a implementován Hopcroft-Tarjan algoritmus, který rozhoduje o rovinnosti grafu v lineárním čase.

Klíčová slova: teorie grafů, rovinnost, Hopcroft-Tarjan algoritmus, grafová knihovna

Vedoucí práce: RNDr. Marko Genyk-Berezovskyj

Abstract

This bachelor thesis is focused on the open-source library for graph algorithms jGraphT. At first, it compares this library with similar available libraries, which also focus on mathematical graphs. Then it introduces the contents of the library and its structure. In its last part, the content of the library is extended by the test that detects the planarity of the specified graph. For this purpose, the Hopcroft-Tarjan algorithm is described and implemented. It is able to determine the planarity of the given graph in linear time.

Keywords: graph theory, planarity, Hopcroft-Tarjan algotithm, graph library

Title translation: Expansion of a graph algorithms library

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Pro svou práci jsem si vybrala grafovou knihovnu `JGraphT` [Nav]. Tato volně dostupná knihovna je napsaná v jazyce Java a zabývá se v podstatě vším, co se týká grafů. Umožňuje nejen jejich vizualizaci, ale také aplikaci již implementovaných jednodušších i složitějších postupů. Knihovna je již nyní poměrně rozsáhlá. Přesto se stále rozvíjí a její obsah se zvětšuje. I když se její autoři snaží přistupovat ke grafům co nejobecněji, práce s knihovnou není příliš složitá. V kódu je užíváno generických datových typů, což umožňuje jeho univerzální použití.

Ačkoliv se to na první pohled nezdá, matematické grafy jsou hojně užívané v praxi. Pomocí grafů lze modelovat rozmanité struktury a zkoumat jejich vlastnosti, čehož lze využít při řešení mnoha rozličných problémů spadajících do běžného života i do různých vědních oborů, které se zdánlivě grafové teorie vůbec netýkají. Své uplatnění najdou například při modelování dopravních sítí, vyhledávání ve strukturách nebo plánování činností systémů.

Knihovna je tématicky rozdělena do několika balíčků, jako jsou generování grafů, složitější grafové algoritmy, obarvování grafů, prostředky pro procházení grafů a jiné. V této práci se zaměřím na ty části, které se věnují grafovým algoritmům. Nejprve definuji termíny, které budu v první části práce užívat. Implementované algoritmy se pokusím popsat, některé z nich otestovat a ohodnotit. Knihovnu jako takovou porovnám z hlediska obsahu a funkčnosti s dalšími dostupnými knihovnami, které se zabývají tímtéž nebo podobným tématem. Následně se budu věnovat implemetaci algoritmu pro testování rovinnosti orientovaných a neorientovaných grafů, který v knihovně prozatím chybí. Představím Hopcroft-Tarjan algoritmus, objasním specifika své verze jeho implementace a demonstuji práci s nově naprogramovaným algoritmem.

Kapitola 2

Přehled užité terminologie

Pro jednodušší orientaci v pojmech, které budu nadále běžně užívat, uvádím jejich seznam s definicemi.

Definice 2.1 (Graf, vrchol, hrana)**.** *Graf* je uspořádaná dvojice $G = (V, E)$, kde V označuje množinu *vrcholů* (*uzlů* - *vertices*) a E množinu *hran* grafu (*edges*), přičemž existuje zobrazení $f : E \to V \times V$. [MN07]

Definice 2.2 (Sjednocení grafů)**.** *Sjednocením* dvou grafů $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$ rozumíme graf $G = (V, E)$, kde $V = V_1 \cup V_2$ a $E = E_1 \cup E_2$. [Har]

Poznámka. Opakovaným užitím definice lze dosáhnout sjednocení spočetného množství grafů.

Definice 2.3 (Podgraf)**.** Mějme graf $G = (V, E)$. Graf $G_1 = (V_1, E_1)$ nazveme *podgrafem* grafu G, pokud $V_1 \subseteq V$ a $E_1 \subseteq E$. [Dem02]

Definice 2.4 (Bipartitní graf)**.** Graf $G = (V, E)$ nazveme *bipartitní*, pokud existují dvě disjunktní množiny (tzv. partity) V_1, V_2 splňující $V = V_1 \cup V_2$ a všechny hrany z E se zobrazují do dvojice vrcholů v_1 a v_2 tak, že $v_1 \in V_1$ a $v_2 \in V_2$. [MN07]

Definice 2.5 (Orientovaný a neorientovaný graf)**.** Graf G nazveme *orientovaným*, je-li zobrazení f tvaru $e \to \{v_1, v_2\}$, kde $e \in E$ a $v_1, v_2 \in V$. Vrchol v_1 nazveme *počáteční* vrchol hrany e, vrchol v_2 nazveme *koncový* vrchol hrany e. Temto vrcholům se říká *krajní vrcholy* hrany e. Pro vrchol v definujeme přirozená čísla *vstupní* (respektivě *výstupní*) *stupeň vrcholu*, jakožto počet hran, jejichž koncovým (respektivě počátečním) vrcholem je vrchol v. Je-li zobrazení f tvaru $e \to (v_1, v_2)$, nazveme graf *neorientovaný*. Vrcholy v_1 a v_2 opět nazveme *krajními* vrcholy hrany e (nebo můžeme říci, že tyto vrcholy jsou *incidentní* s hranou e). Pro vrchol v definujeme přirozené číslo *stupeň vrcholu*, jakožto počet hran, pro něž je v krajním vrcholem. [MN07] [Dem02]

Definice 2.6 (Ohodnocený graf)**.** Graf nazveme *ohodnocený* (nebo *s ohodnocením hran*) je-li každé jeho hraně přiřazeno nějaké reálné číslo. [ČRK04]

Definice 2.7 (Úplný graf)**.** Neorientovaný graf nazveme *úplný*, jsou-li v něm všechny dvojice vrcholů propojeny hranou. [MN07]

Poznámka. Úplný graf, s množinou vrcholů o mohutnosti n, značíme K_n . Úplný bipartitní graf, jehož partity mají mohutnost n a m, značíme $K_{n,m}$.

Definice 2.8 (Multigraf a jednoduchý graf)**.** Graf nazveme *multigraf*, pokud obsahuje dvě hrany, jejichž krajní vrcholy se shodují (v případě orientovaného grafu se musí shodovat vstupní vrchol se vstupním a výstupní s výstupním). V opačném případě nazveme graf *jednoduchý* (*prostý*). [Dem02]

Definice 2.9 (Pseudorgaf)**.** *Pseudograf* je neorientovaný multigraf, v němž se objevují smyčky. [Har]

Definice 2.10 (Indukovaný podgraf)**.** Mějme graf $G = (V, E)$. Podgraf G_1 grafu G nazveme *indukovaný* množinou $A \subseteq V$, pokud A je množina vrcholů grafu G_1 a množina hran grafu G_1 obsahuje všechny hrany grafu G s koncovými vrcholy v množině A. [Dem02]

Definice 2.11 (Druhá mocnina grafu)**.** *Druhá mocnina* orientovaného grafu $G = (V, E)$ je graf $G_1 = (V, E_1)$, kde pro každou hranu $e_1 \in E_1$ existují právě dvě hrany e_a a e_b tak, že počáteční vrchol hrany e_1 je zároveň počátečním vrcholem hrany e_a, koncový vrchol hrany e_1 je koncový vrchol hrany e_b a koncový vrchol hrany e_a se shoduje s počátečním vrcholem hrany e_b. [Eks07]

Definice 2.12 (Tranzitivní uzávěr grafu)**.** *Tranzitivní uzávěr* orientovaného grafu G je orientovaný graf G_1 takový, že má shodnou množinu vrcholů a hrana (u, v) patří množině hran G_1, když existuje v grafu G orientovaná cesta z u do v. [ČRK04]

Definice 2.13 (Sousední vrcholy)**.** Dvojici vrcholů nazveme *sousední*, pokud jsou spojeny hranou (jsou incidentní s toutéž hranou). [Dem02]

Definice 2.14 (Izomorfní grafy)**.** Dva grafy jsou *izomorfní*, pokud existuje vzájemně jednoznačné zobrazení $g : V \to V_1$ a $h : E \to E_1$ takové, že $e = (u, v) \Leftrightarrow h(e) = (g(u), g(v))$ pro orientované grafy a obdobně potom také $e = \{u, v\} \Leftrightarrow h(e) = \{g(u), g(v)\}$ pro grafy neorientované. [MN07]

Definice 2.15 (Orientovaný a neorientovaný sled)**.** Posloupnost vrcholů a hran $v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k$ nazveme *orientovaný sled* (respektive *neorientovaný sled*), je-li $f(e_i) = (v_i, v_{i+1})$ (respektive $f(e_i) = \{v_i, v_{i+1}\}$). [Dem02]

Definice 2.16 (Tah)**.** Sled nazveme *tahem*, pokud se v něm neopakují hrany. [Dem02]

Definice 2.17 (Cesta)**.** Tah nazveme *cestou* , pokud se neopakují ani vnitřní vrcholy. *Délkou cesty* rozumíme počet jejích hran. [Dem02]

Definice 2.18 (Uzavřený tah)**.** Tah je *uzavřený*, pokud se shoduje jeho první a poslední vrchol. [Dem02]

Definice 2.19 (Eulerův tah)**.** Tah nazveme *Eulerův*, pakliže prochází každou hranou grafu. [MN07]

Definice 2.20 (Kružnice)**.** Uzavřená cesta bývá označena termínem *kružnice*. [Dem02]

Definice 2.21 (Hamiltonovská kružnice)**.** Řekneme, že graf obsahuje *Hamiltonovskou kružnici* , pokud v grafu existuje kružnice obsahující všechny jeho vrcholy. [MN07]

Definice 2.22 (Vrcholové pokrytí)**.** *Vrcholové pokrytí* grafu je podmnožina vrcholů splňující vlastnost, že každá hrana grafu je incidentní s alespoň jedním

vrcholem této množiny. [MN07]

Definice 2.23 (Párování)**.** *Párování grafu* je podmnožina hran grafu taková, že žádné dvě hrany z této množiny nemají společný vrchol. [MN07]

Definice 2.24 (Cyklický a acyklický graf)**.** Graf nazveme *cyklický*, pokud obsahuje kružnici. V opačném případě jej nazveme *acyklický*. [Dem02]

Definice 2.25 (Diametr)**.** *Diametr* nebo též *průměr* grafu nazveme přirozené číslo $\max d(u, v \| u, v \in V)$, kde d značí délku cesty mezi příslušnými vrcholy. [MN07]

Definice 2.26 (Obarvení)**.** Mějme graf $G = (V, E)$ a k přirozené číslo. Zobrazení $g : V \to \{1, 2, \ldots, k\}$ nazýváme *obarvením* grafu G k barvami, pokud pro hranu s krajními vrcholy u, v platí $g(u) \neq g(v)$. [MN07]

Definice 2.27 (Strom)**.** Souvislý graf bez kružnic nazýváme *strom*. [MN07]

Definice 2.28 (Les)**.** Sjednocení více stromů nazýváme *les*. [MN07]

Poznámka. Ojediněle bývá les také považován za strom.

Definice 2.29 (Kořen)**.** Mějme orientovaný strom $G(V, E)$. Vrchol $v \in V$ je *kořenem* stromu G, pokud z v vede orientovaná cesta do všech vrcholů grafu G. [MN07]

Definice 2.30 (Kořenový strom)**.** Strom, který má kořen, nazveme *kořenový strom*. [MN07]

Definice 2.31 (Hladina)**.** Řekneme, že vrchol $u \in V$ kořenového stromu leží v *hladině* výšky k, má-li cesta od kořene tohoto stromu k vrcholu v délku k. [Dem02]

Definice 2.32 (Spojitý graf)**.** Řekneme, že graf je *spojitý* (*souvislý, slabě souvislý*), pokud mezi každou dvojicí jeho vrcholů existuje neorientovaná cesta. [MN07]

Definice 2.33 (Silně souvislý graf)**.** Orientovaný graf $G = (V, E)$ je *silně souvislý*, pokud existuje orientovaná cesta z vrcholu u do vrcholu v a $\forall u, v \in V$. [Dem02]

Definice 2.34 (Komponenta souvislosti)**.** Množina vrcholů A grafu G je *komponenta slabé souvislosti* tohoto grafu, je-li největším slabě souvislým podgrafem tohoto grafu (obdobně se silnou souvislostí). [MN07]

Definice 2.35 (Předchůdce, následník, rodič)**.** Vrchol v orientovaného kořenového stromu je *předchůdcem* (respektivě *následníkem*) vrcholu u, existuje-li orientovaná cesta z v do u (respektivě z u do v). Předchůdce vrcholu v, z něhož vede orientovaná cesta k vrcholu v délky 1, se nazývá *rodič* vrcholu v. [MN07]

Definice 2.36 (Kostra)**.** *Kostra* souvislého grafu $G = (V, E)$ je podgraf grafu G obsahující všechny vrcholy V, který je zároveň stromem. [MN07]

Definice 2.37 (Minimální kostra)**.** Jako *minimální kostru* grafu s ohodnocenými hranami definujeme kostru s nejmenší možnou cenou. [MN07]

Definice 2.38 (Vrcholově 2-souvislý graf a artikulace)**.** Graf $G = (V, E)$ je *(vrcholově) 2-souvislý*, pokud graf $G \setminus U$ je souvislý pro každou $U \in V$ takovou, že $|U| < 2$. Vrchol, po jehož vyjmutí se graf rozpadne na komponenty souvislosti, nazveme *artikulace*. [MM96].

Definice 2.39 (Klika). *Klika* je úplný podgraf nějakého grafu. [Dem02]

Definice 2.40 (Separátor). *Separátor* grafu $G = (V, E)$ je podmnožina $U \in V$ tohoto grafu taková, že graf $(V \setminus U, E)$ není souvislý. [Dem02]

Definice 2.41 (Tok v orientovaném grafu). Pojmem *tok* v orientovaném grafu $G = (V, E)$ nazýváme ohodnocení hran reálnými čísly $g : E \to \mathbb{R}$, které pro každý vrchol splňuje Kirchhoffův zákon. Označíme-li po řadě $I(v)$, $O(v)$ množiny hran, pro něž je v počátečním (koncovým) vrcholem, musí platit:

$$\sum_{e \in I(v)} g(e) = \sum_{e \in O(v)} g(e).$$

[Dem02]

Definice 2.42 (Řezy grafu). (*Hranový*) *řez* grafu $G = (V, E)$ je množina hran $F \in E$ taková, že graf $G_1 = (V, E \setminus F)$ není souvislý, s-t *řez* je množina hran K taková, že každý z vrcholů s, t leží v různých komponentách souvislosti grafu $(V, E \setminus K)$. [Dem02]

Kapitola 3

Dostupné grafové knihovny

Při podrobnějším hledání a volbě vhodné grafové knihovny narazí její budoucí uživatel pravděpodobně na problém s výběrem. Na trhu je dostupných poměrně mnoho knihoven zabývajících se problematikou grafů pro různé programovací jazyky. Často se odlišují v rozsahu a zaměření (některé grafové knihovny se soustředí hlavně na kvalitní vizualizaci grafů, jiné zase na implementaci více či méně známých grafových algoritmů). Já jsem mezi nimi vybrala 9 dalších a pokusila se je porovnat s knihovnou `jGraphT`[Nav] - 5 z nich je implementováno v jazyce `Java`, zbylé 4 v dalších jazycích.

Při bližším zkoumání mě překvapilo, že všechny mnou vybrané knihovny podporují práci s orientovanými i neorientovanými grafy. Mezi knihovnami v `Javě` jsem objevila pouze jednu podobnou `jGraphT`, jedná se o knihovnu `grph`[Hog]. Tyto dvě knihovny si mohou konkurovat, neboť jejich obsah je podobně rozsáhlý. Nutno však podotknout, že knihovna `grph` má výrazně méně informací v javadocu. Pokud chce uživatel najít nějaké bližší údaje k algoritmům a implementaci, je třeba hledat v dokumentaci dostupné na webu. Knihovna `JUNG`[JUN] sice obsahuje méně implementovaných algoritmů, ale její vnitřní členění je intuitivnější - pro nezasvěceného člověka je přehlednější, tedy se v ní rychleji zorientuje. Knihovna `JDSL`[JDS] se grafům věnuje jen v malé míře, mezi zkoumanými knihovnami je výrazně nejslabší. Komerční knihovna `yWorks`[yWo] má v sobě implementovány některé algoritmy, má ale také rozšířené funkce pro grafické zobrazení, které vyčnívají nad zbývajícími knihovnami. Na knihovně `GraphStream`[Gra] mne nejvíce zaujala možnost generovat zajímavé grafy (jako například čtvercovou síť toru) a uživatelská přístupnost.

Knihovna `igraph`[igr] je implementována v různých programovacích jazycích, její rozsah odpovídá rozsahu knihovny `JGraphT`, nabízí i mnoho možností exportu dat dojiných formátů.

V jazyce `C++` jsou napsány dvě knihovny: `LEDA`[GA] a `BGL`[Boo]. První zmíněná knihovna není tak důkladně propracovaná, chybí jí pro uživatele podstatné ukázky využití a její dokumentace je dle mého názoru méně přehledná. Druhá knihovna je součástí rozsáhlejší univerzální knihovny `BOOST`, která je často aktualizována.

V jazyce `Python` je implementována knihovna `NetworkX`[Net], která je také kvalitně zpracována z algoritmického hlediska. Možnosti jejích vizualizací jsou

ovšem omezené zatímco dokumentace je na vysoké úrovni.

Při snaze porovnat knihovny jsem zpracovala tabulku, která shrnuje některé jejich vlastnosti. Jelikož mým cílem je zkoumat především algoritmickou část, vybrala jsem několik používaných algoritmů a snažila se je v těchto knihovnách najít. Tabulku uvádím na následujících stránkách:

Název knihovny	**jGraphT**	**Graph Stream**	**grph**
Jazyk	Java	Java	Java
Rozsah	velký	střední	velký
Volná dostupnost	ano	ano	ano - bez vlastních úprav
Udržovanost	ano	ano	ano
Zaměření	grafová teorie	modelování	grafová teorie
Podpora multigrafů	ano	ano	ano
Generátor grafu	ano	ano	ano
Souvislost grafu	ano	ano	ano
Nejkratší cesta	ano	ano	ano
Minimální kostra	ano	ano	ano
Maximální tok	ano	ne	ano
Minimální řez	ano	ne	ano
Izomorfismus grafů	ano	ne	ano
A*	ne	ne	ne
Vizualizace	přídavná knihovna	ano	pomocí exportu
Demo	ano	ano	ano
Export	ano	ano	omezený

Tabulka 3.1: Porovnání knihoven jGraphT, GraphStream a grph.

Název knihovny	jGraphT	yWorks	igraph
Jazyk	Java	Java a další	C, R, Python
Rozsah	velký	střední	velký
Volná dostupnost	ano	ne	ano
Udržovanost	ano	ano	ano
Zaměření	grafová teorie	modelování	grafová teorie
Podpora multigrafů	ano	ano	ano
Generátor grafu	ano	ano	ano
Souvislost grafu	ano	ano	ano
Nejkratší cesta	ano	ano	ano
Minimální kostra	ano	ano	ano
Maximální tok	ano	ano	ano
Minimální řez	ano	ano	ano
Izomorfismus grafů	ano	ne	ano
A*	ne	ne	ne
Vizualizace	přídavná knihovna	dobrá	pomocí exportu
Demo	ano	ano	ano
Export	ano	ano	ano

Tabulka 3.2: Porovnání knihoven jGraphT, yWorks a igraph.

Název knihovny	jGraphT	JUNG	JDSL
Jazyk	Java	Java	Java
Rozsah	velký	střední	malý
Volná dostupnost	ano	ano	ano - pro nekomerční účely
Udržovanost	ano	ne	ne
Zaměření	grafová teorie	modelování	základní manipulace
Podpora multigrafů	ano	ano	ano
Generátor grafu	ano	ano	ne
Souvislost grafu	ano	okrajově	ne
Nejkratší cesta	ano	ano	ano
Minimální kostra	ano	ano	ano
Maximální tok	ano	ano	ne
Minimální řez	ano	ano	ne
Izomorfismus grafů	ano	ne	ne
A*	ne	ne	ne
Vizualizace	přídavná knihovna	dobrá	ne
Demo	ano	ano	ano
Export	ano	ano	ne

Tabulka 3.3: Porovnání knihoven jGraphT, JUNG a JDSL.

Název knihovny	jGraphT	LEDA	BGL
Jazyk	Java	C++	C++
Rozsah	velký	střední	velký
Volná dostupnost	ano	omezeně	ano
Udržovanost	ano	ne	ano
Zaměření	grafová teorie	grafová teorie	grafová teorie
Podpora multigrafů	ano	ano	ano
Generátor grafu	ano	ano	ano
Souvislost grafu	ano	ano	ano
Nejkratší cesta	ano	ano	ano
Minimální kostra	ano	ano	ano
Maximální tok	ano	ano	ano
Minimální řez	ano	ano	ano
Izomorfismus grafů	ano	ano	ano
A*	ne	ne	ano
Vizualizace	přídavná knihovna	pomocí exportu	základní
Demo	ano	ne	ano
Export	ano	omezeně	omezeně

Tabulka 3.4: Porovnání knihoven jGraphT, LEDA a BGL.

Název knihovny	jGraphT	NetworkX
Jazyk	Java	Python
Rozsah	velký	velký
Volná dostupnost	ano	ano
Udržovanost	ano	ano
Zaměření	grafová teorie	grafová teorie
Podpora multigrafů	ano	ano
Generátor grafu	ano	ano
Souvislost grafu	ano	ano
Nejkratší cesta	ano	ano
Minimální kostra	ano	ano
Maximální tok	ano	ano
Minimální řez	ano	ano
Izomorfismus grafů	ano	ano
A*	ne	ne
Vizualizace	přídavná knihovna	základní
Demo	ano	ano
Export	ano	omezeně

Tabulka 3.5: Porovnání knihoven jGraphT a NetworkX.

3.1 Test knihoven jGraphT a GraphStream

Za účelem prověření funkčnosti a efektivity běhu knihoven jsem provedla jednoduchý praktický test, na jehož základě porovnám knihovny `jGrapht` a `GraphStream`.

Aby bylo dosaženo co nejvěrohodnějších výsledků, všechny testy byly několikrát opakovány za shodných vnějších podmínek. Z naměřených hodnot byl následně vybrán medián. Testy byly provedeny pro grafy s počtem vrcholů $0, 3, 10, 30, 100, 300, 1000$ a 3000 (hodnoty se zvyšují o polovinu řádu, při testech velkého množství grafů s vyšším počtem vrcholů některé algoritmy nedoběhly ani po desítkách minut - to však bylo nejspíše způsobeno omezením hardwarových možností). Pakliže test pro graf daného počtu vrcholů proběhl za dobu kratší než $0,1$ vteřiny, bylo za účelem vyšší přesnosti provedeno měření času tisíce opakování tohoto testu na různých grafech. Jako výsledek testu pak byl brán průměrný čas jedné iterace.

Provedeme porovnání metod, které generují pseudonáhodný graf o daném počtu vrcholů a hran a testu spojitosti grafu na dvou typech grafů - hustém a řídkém. Jako zástupce množiny řídkých grafů vybereme graf, jehož počet hran je roven počtu jeho vrcholů. Z množiny hustých grafů zvolíme graf s počtem hran

$$\left\lfloor \frac{n \cdot (n-1)}{4} \right\rfloor,$$

kde n je počet vrcholů testovaného grafu. Tato hodnota přibližně odpovídá faktu, že každý vrchol je hranou spojen s průměrně

$$\left\lfloor \frac{n-1}{2} \right\rfloor$$

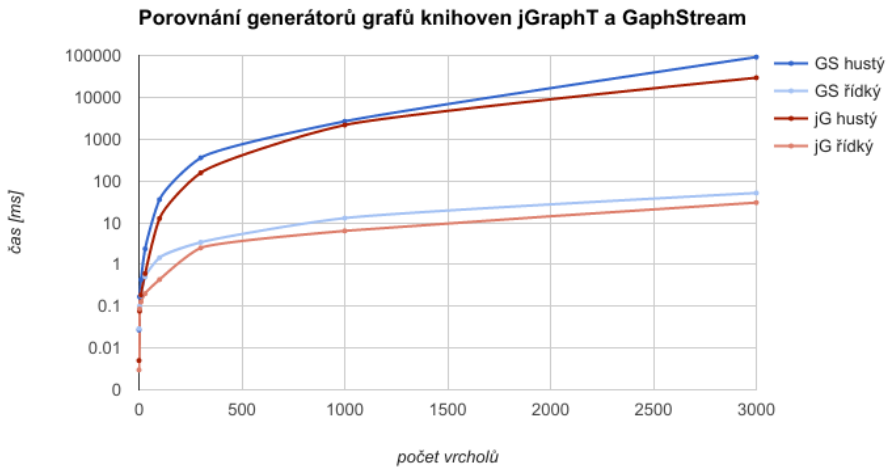
vrcholy.

Výsledné hodnoty zaneseme dografu, na vodorovné ose s lineární stupnicí lze vyčíst počet vrcholů testovaného grafu. Na svislé ose s logaritmickou stupnicí lze vyčíst medián délky trvání testu na jedné instanci grafu v miliseknundách.

V knihovně `jGrapht` užijeme generátor, jehož vstupní parametry jsou počet vrcholů a počet hran grafu. V knihovně `GraphStream` jsou vstupními parametry generátoru počet vrcholů grafu a průměrný počet hran vedoucích z vrcholu grafu.

Z grafu na obrázku3.1 můžeme usoudit, že asymptotická časová složitost generování pseudonáhodného grafu je v obou knihovnách stejná. Knihovna `jGrapht` dosahuje lepších výsledků než knihovna `GraphStream`. Tvary křivek ve vizualizaci odpovídají faktu, že implementace grafových generátorů v obou testovaných knihovnách mají polynomiální časovou náročnost.

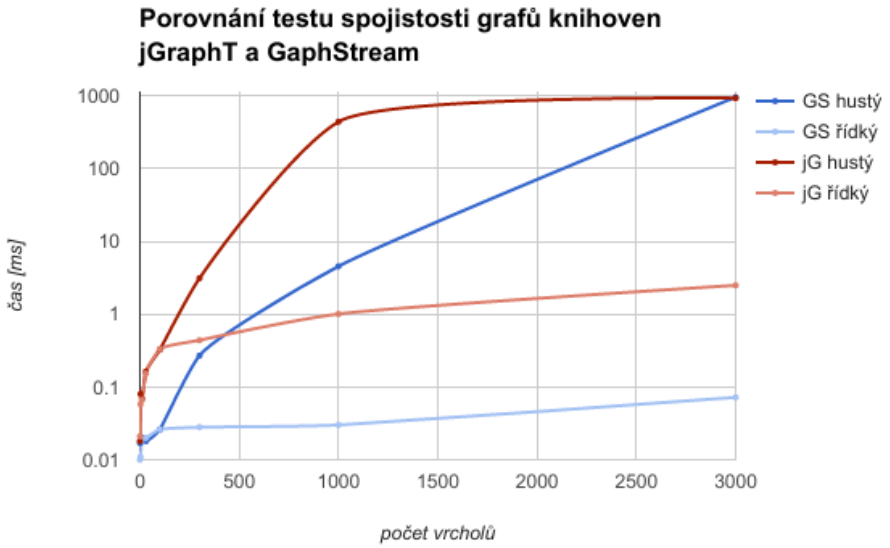
Graf na obrázku 3.2 vykresluje výsledky pro test spojitosti řídkých a hustých grafů v obou knihovnách. V knihovně `jGraphT` tuto funkci zastává metoda `isGraphConnected()` obsažená ve třídě `ConnectivityInspector` v balíčku `org.jgraph.alg`. Její návratová hodnota je typu `boolean`. Metoda nejprve nalezne množiny vrcholů, které jsou spojité, a kontroluje jejich počet. Základem tohoto algoritmu je prohledávání grafu do šířky, jehož časová složitost by



Obrázek 3.1: Porovnání generátorů grafů jGraphT a GraphStream.

měla být $O(E + V)$. V knihovně `GraphStream` je za tímto účelem imlementována metoda `getConnectedComponentsCount()`, jejíž návratový typ je `int`. Tento test probíhal formou vygenerování určitého množství testovacích grafů, které jsme dopředu uložili do struktury `ArrayList`. V testu pak docházelo k měření nepřerušených opakování volání metody pro detekci spojitého grafu. Knihovna `jGrapht` v tomto testu dosahuje výrazně horších výsledků než knihovna `GraphStream`.

Podle vizualizace na obrázku 3.2 odhaduji, že testy spojitosti řídkých grafů v obou knihovnách budou skutečně probíhat v čase $O(E + V)$. Křivky vykreslující časový průběh testu na hustých grafech se mi nezdají přesvědčivé.



Obrázek 3.2: Porovnání testu spojitosti grafů jGraphT a GraphStream.

Kapitola 4
Knihovna jGraphT

4.1 Výstavba knihovny jGraphT

Jak již bylo zmíněno v úvodu, knihovna `JGraphT` se skládá z několika balíčků jejichž obsah specifikujeme v této kapitole. Nespornou výhodou této knihovny je univerzalita jejího využití. Vrcholy grafů mohou být libovolného typu - celá čísla, slova, seznamy objektů, nebo cokoliv jiného, dokonce i samotné grafy.

4.1.1 org.jgrapht

Tento balíček obsahuje několik rozhraní, která definují metody příslušné určitým typům grafů. Kupříkladu rozhraní `Graph` definuje metody pro přidávání a odstraňování hran a vrcholů grafu, metodu kontrolující zda graf obsahuje hranu či vrchol a metody, které vrací množinu vrcholů či hran grafu (případně množinu hran incidentních s konkrétním vrcholem). Toto rozhraní dříve rozšiřovalo několik dalších (specifičtějších) rozhraní popisující nové metody pro konkrétní typy grafů - stupeň vrcholu pro neorientovaný graf, vstupní a výstupní stupeň vrcholu pro graf orientovaný a přiřazení konkrétních vah hranám grafu s ohodnocením hran. Od tohoto přístupu se však nyní upustilo.

Balíček dále obsahuje rozhraní umožňující tvorbu hran a vrcholů, rozhraní specifikující práci s cestami v grafu (s metodami vracejícími počáteční a koncový vrchol cesty, případně seznam jejích hran či vrcholů) a rozhraní umožňující pracovat se vzájemně jednoznačným zobrazením dvou grafů (s metodami pro určení odpovídajících si vrcholů či hran), abstraktní třídu `Graphs`, která již konkrétně uvádí metody umožňující základní operace s grafy (jako například test incidence vrcholu s hranou, obrácení hran orientovaného grafu či jeho přetvoření v neorientovaný, seznam sousedních vrcholů v neorientovaném grafu, resp. předchůdců a následníků v grafu orientovaném a metody pro přidání hran a vrcholů ke grafu). V neposlední řadě je třeba zmínit abstraktní třídu `GraphTests`, která shromažďuje testy různých vlastností grafu. Do něho jsem také přidala test pro zjištění planarity grafu.

4.1.2 org.jgrapht.graph a org.jgrapht.graph.builder

Balíček `org.jgrapht.graph` obsahuje mnoho tříd, které implementují rozhraní `org.jgrapht`. Tyto třídy udávají, jak se budou chovat konkrétní typy grafů, jako pseudografy, multigrafy, jednoduché grafy a další. Dále obsahuje třídy umožňující realizovat sjednocení grafů a podgrafy.

Druhý zmíněný balíček, tedy `org.jgrapht.graph.builder`, obsahuje třídy, které realizují tvorbu různých typů grafů.

4.1.3 org.jgrapht.generate

Tento balíček obsahuje třídy, jejichž metody umí vytvářet grafy konkrétních vlastností, jako jsou prázdný graf, uplný graf, kružnice dané velikosti, nebo náhodný graf o daném počtu vrcholů a hran, případně o daném počtu vrcholů a pravděpodobnosti existence hrany.

4.1.4 org.jgrapht.alg a jeho součásti

Tento balíček je hlavní algoritmický balíček celé knihovny. V jedné ze svých částí `org.jgrapht.alg.cycle` definuje několik způsobů pro hledání silných komponent v grafech - pro orientované grafy jsou implementovány `Tarjanův`, `Tiernanův`, `Johnsonův` a `Szwarcfiter-Lauerův` algoritmus, pro neorientované `Patonův` algoritmus. Označíme-li V počet vrcholů grafu, počet hran grafu E a počet jednoduchých kružnic v grafu C, potom časová náročnost algoritmů v nejhorších případech je udána po řadě $O(V \cdot E \cdot C)$, $O(V \cdot c^V)$, $O((V + E)C)$, $O(V + EC)$ a $O(V^3)$.

V dalších částech jsou implementována různá řešení jednotlivých algoritmických problémů.

Pro orientované grafy je uveden algoritmus detekující cykly (kružnice), na základě prohledáváni do hloubky je schopen najít všechny vrcholy obsažené v cyklu, případně může být tento cyklus specifikován konkrétním vrcholem.

Další třída je schopna rozhodnout, zda je orientovaný či neorientovaný graf spojitý. Tuto skutečnost určuje porovnání počtu komponent slabé souvislosti grafu a jedničky. Prohledávání do šířky rozloží množinu vrcholů v komponenty slabé souvislosti, dovede také rozhodnout, zda existuje cesta mezi dvěma vrcholy.

Pro úplný graf s ohodnocenými hranami, v němž platí trojúhelníková nerovnost, poskytuje přibližné minimální ohodnocení `Hamiltonovské` kružnice (tato aproximace by neměla přesáhnout dvojnásobek ohodnocení optimálního řešení).

Z neorientovaného grafu je možné vrátit `Eulerův` uzavřený tah a na základě toho rozhodnout, zda je graf eulerovský.

Pro orientovaný graf je možno rozhodnout, zda je silně spojitý, dále získat seznam množin vrcholů jeho silných komponent, či seznam jeho silně spojitých podgrafů.

Dále knihovna obsahuje třídu, která rozhoduje, zda je neorientovaný graf vrcholově 2-souvislý, poskytuje seznam všech artikulací, nebo komponenty

vrcholové 2-souvislosti obsahující zadaný vrchol.

Další třída zacházející s neorientovanými grafy určuje horní odhad počtu barev při obarvení grafu hladovým algoritmem a vrací též jeho obarvení ve formě pole barev.

Pro neorientované grafy je také možno hledat jejich vrcholové pokrytí, jedna ze tříd poskytuje metody hledající aproximaci minimálního vrcholového pokrytí na základě hladového algoritmu, nebo jiný odhad který opět nepřesáhne dvojnásobek optimálního výsledku.

Další třída iterativně hledá tranzitivní uzávěr orientovaného grafu a udává horní odhad počtu iterací tohoto postupu.

Mezi problémy řešené touto knihovnou patří i hledání nejkratší cesty mezi dvěma vrcholy grafu s ohodnocením hran. Za tímto účelem bylo implementováno několik algoritmů: `Dijkstrův` algoritmus (vyžadující kladné ohodnocení hran), `Floyt Warshallův` algoritmus (nesmí obsahovat cykly záporné délky, ale je schopen najít všechny nejkratší cesty, určit jejich počet a diametr grafu), `Bellman Fordův` algoritmus (ten již povoluje i záporná ohodnocení) a jeho rozšíření hledající k nejkratších cest (i tento algoritmus požaduje na vstupu graf bez kružnic záporné váhy).

Další běžnou úlohou teorie grafů je hledání minimální kostry grafu které je možno realizovat na základě tří algoritmů - `Kruskalova` hladového algoritmu (s časovou náročností $O(E \cdot \log(E))$), `Borůvkova` nebo `Primova` algoritmu (ten je někdy také nazýván `Jarníkův` a jeho implementace podporuje neorientované grafy s ohodnocenými hranami).

Pro kořenové stromy přináší tento balíček také řešení problému hledání společného předka nejnižší vrstvy dvou vrcholů. I tato úloha přináší dvě různá řešení, naivní prohledávání předků vrcholů a `Tarianův` algoritmus.

Další problém řešený tímto balíčkem se týká párování grafu. Hledáním maximálního párování se zabývá `Hopcroft Karpův` algoritmus (vhodný pro neorientované jednoduché biparitní grafy, jehož časová složitost dosahuje hodnoty $O(E \cdot \sqrt{V})$ - opět se jedná o úpravu hladového algoritmu) a `Edmondův` algoritmus (s časovou náročností $O(V^4)$, jakožto úprava prohledávání do šířky). Pro úplné biparitní grafy s nezáporným ohodnocením hran je implementován `Kuhn Munkresův` algoritmus (také zvaný maďarský algoritmus) který hledá perfektní párování nejnižší ceny grafu. Pracuje s časovou náročností $O(V^3)$.

Dalším podstatným problémem teorie grafů je hledání maximálního toku či minimálního řezu grafu. Za pomoci `Edmont Karpova` algoritmu je uživatel schopen spočítat maximální tok sítě. Uvedená horní hranice časové náročnosti je $O(VE^2)$. Zadaná síť musí být popsána orientovaným grafem s nezáporným ohodnocením hran. Minimální řez neorientovaného grafu s ohodnocenými hranami hledá rekurzivní `Stoer Wagnerův` algoritmus, jeho uvedená časová náročnost implementace v této knihovně je $O(VE \cdot \log(E))$. Pro orientovaný graf s ohodnocenými hranami lze nalézt minimální s-t řez za pomoci `Edmont Karpova` algoritmu pro nalezení maximálního toku v kombinaci se zpětným prohledáváním hran.

Detekci klik grafů řeší `Bron Kerboschův` algoritmus, úprava rekurzivního prohledávání do hloubky, hledání minimálního separátoru rozkladu na kliky

řeší takzvaný "MCS-M+ algoritmus".

Balíček org.jgrapht.alg.color implementuje algoritmy související s obarvováním grafu - různá řazení vrcholů za účelem obarvení, počet použitých barev v obarvení a horní a dolní odhad počtu těchto barev.

Nově je obsažen i balíček, zabývající se isomorfizmy grafů za pomoci VF2 algoritmu.

4.1.5 org.jgrapht.experimental

Tento balíček obsahuje algoritmy, které jsou ve fázi vývoje, nejsou tedy považovány za hotové. Aktuálně obsahuje jedinou třídu implementující Brownův algoritmus využívající backtrackingu.

4.1.6 Další balíčky knihovny

Knohovna dále obsahuje několik menších balíčků, které zajišťují její správný chod.

Balíček org.jgrapht.traverse obsahuje iterátory, které procházejí grafy pomocí několika různých technik - do hloubky, do šířky apod.

Balíček org.jgrapht.specifics obsahuje implementaci specifických metod pro konkrétní typy grafů.

Balíček org.jgrapht.event obsahuje třídy, jež spravují události (Events) a jejich posluchače (Listeners).

Balíčky org.jgrapht.ext a org.jgrapht.io specifikuje nástroje pro export grafů do formátů podporovaných jinými programy, jako třeba Matlab.

Balíček org.jgrapht.util slouží pro technické záležitosti, které nesouvisejí s grafovou teorií (např. výpočet faktoriálu).

Poslední balíček org.jgrapht.demo obsahuje několik ukázek použití knihovny JGraphT, které novému uživateli usnadňují začátek práce s touto knihovnou.

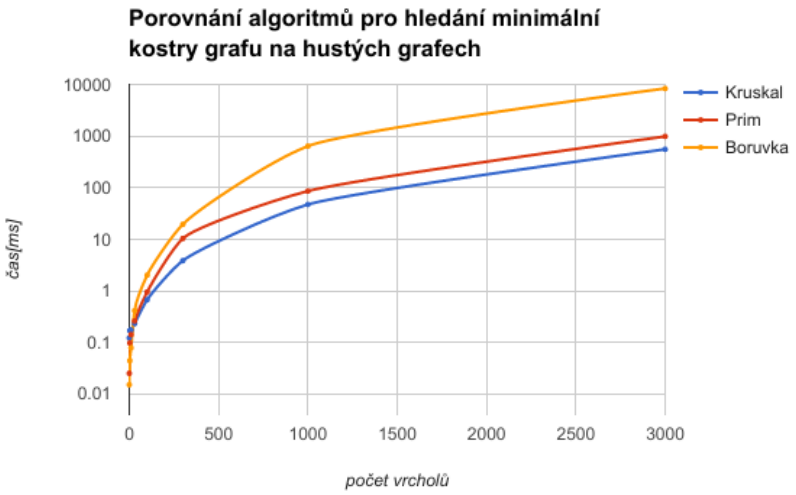
4.2 Aplikace knihovny

Proveďme nyní obdobné testy jako při porovnání knihovny jGraph s knihovnou GraphStream. Zaměřme se na test algoritmů hledajících minimální kostru grafu.

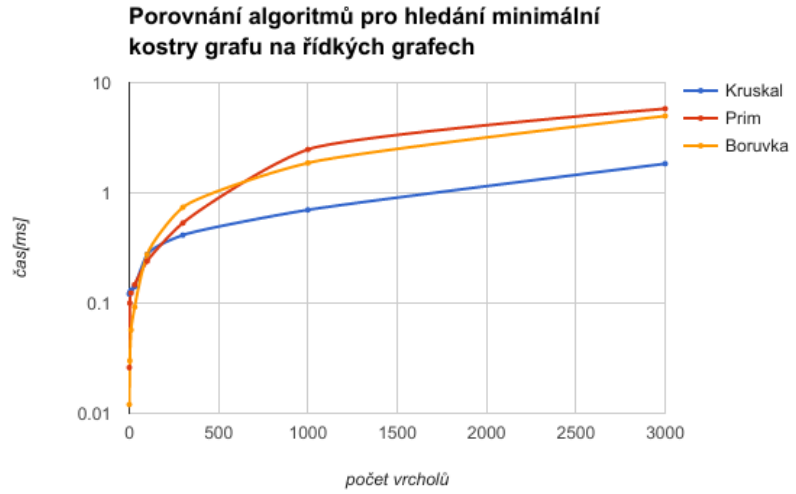
4.2.1 Hledání minimální kostry

Pro hledání minimální kostry grafů jsou v knihovně implementovány tři algoritmy - Primův, Borůvkův a Kruskalův. Pro jejich porovnání jsem si vygenerovala náhodné grafy obdobně jako v předešlém případě, přičemž závislost počtu hran na počtu vrcholů jsem zvolila také obdobně. Čas potřebný k provedení těchto výpočtů je zachycen v grafech 4.1 a 4.2.

Dosažené výsledky ukazují, že Kruskalův algoritmus dosahuje nejlepších výsledků jak pro řídké, tak pro husté grafy. Měli-li bychom porovnat zbylé



Obrázek 4.1: Porovnání testu algoritmů pro hledání minimální kostry hustých grafů.



Obrázek 4.2: Porovnání testu algoritmů pro hledání minimální kostry řídkých grafů.

dva algoritmy, pak `Primův` je vhodný pro husté grafy, oproti tomu `Borůvkův` dosahuje lepších výsledků na grafech řídkých. Jejich časová složitost se však zdá být stejná. Deklarovaná časová složitost algoritmů pro hledání minimální kostry je $O(E \cdot log(E))$, kde E značí počet hran grafu. Výsledek vizualizace tuto skutečnost potvrzuje.

Kapitola 5

Rovinnost grafů

Přestože je knihovna `jGraphT` obsáhlá, chybí v ní implementace algoritmu, který testuje rovinnost grafů. Z definice lze snadno vyvodit, že každý rovinný graf lze zakreslit do roviny tak, aby se žádná dvojice jeho hran nekřížila. Rovinné grafy mají vlastnosti, které lze prakticky aplikovat v různých oborech. Pomocí grafů je obecně možné modelovat vztahy mezi částmi nějakého celku. Je-li třeba přehledně zakreslit schéma tohoto celku, je vhodné využít právě nakreslení bez křížení hran, kterého lze dosáhnout jen u rovinného grafu.

- V matematice můžeme rovinnosti využít při zkoumání mnohostěnů, neboť jejich grafy jsou vždy rovinné.
- Rovněž v chemii najdeme využití pro poznatky o rovinnosti při znázorňování molekulárních grafů.
- Při navrhování pozemní dopravní síťe je také výhodné zvažovat rovinnou variantu, protože křížení cest s sebou může nést riziko kolize či nutnost vyšší finanční investice.
- V elektrotechnice si své uplatnění najde například při navrhování integrovaných obvodů.

Pro detekci rovinných grafů existuje více různých algoritmů. Velice známé je následující tvrzení [MN07], [Cha96], [Pat13].

Tvrzení 5.1 (nutná podmínka pro rovinnost grafu). *Nechť $G = (V, E)$ je rovinný graf s minimálně třemi vrcholy. Potom $|E| \leq 3 \cdot |V| - 6$.*

V roce 1930 byla publikována následující věta:

Věta 5.2 (Kuratowski). *Graf G je rovinný, právě když žádný jeho podgraf není isomorfní dělení grafu $K_{3,3}$ ani dělení grafu K_6.*

Kuratowského věta, uvedena například v [MM96], sice udává nutnou a postačující podmínku pro rovinnost grafů, není však vhodná pro její praktické testování.

Později vznikly jiné algoritmy s kubickou či kvadratickou časovou náročností vzhledem k počtu vrcholů grafu. Byly založené na postupné konstrukci grafu.

V roce 1974 byl vědci představen první z několika algoritmů, které jsou schopny klasifikovat graf v lineárním čase. V následující podkapitole tento algoritmus rozebereme a teoreticky ověříme jeho časovou složitost.

5.1 Hopcroft-Tarjan algoritmus

5.1.1 Zavedení pojmů

Pro jednodušší orientaci v následujícím textu nejprve uveďme definice používaných termínů s několika poznámkami pro jejich lepší pochopení.

Definice 5.3. Mějme graf $G = (V, E)$. *Nakreslením grafu* G rozumíme zobrazení $f : (V \cup E) \to \mathbb{R}^2$, které každému vrcholu $v \in V$ přiřazuje bod $f(v)$ roviny a každé hraně $e = (v_1, v_2) \in E$ oblouk $f(e)$ v rovině s koncovými body $f(v_1), f(v_2)$. Předpokládejme, že žádný z obrazů vrcholů není nekoncovým bodem žádného oblouku $f(e)$.

Nakreslení grafu G, v němž se žádná dvojice neprotíná v nekoncovém bodě žádného oblouku, se nazývá *rovinné nakreslení grafu* G.

Má-li graf G rovinné nakreslení, řekneme, že je *rovinný*.

Souvislou otevřenou oblast množiny $\mathbb{R}^2$ nazveme *stěna rovinného nakreslení grafu* G.

Neomezenou stěnu rovinného nakreslení grafu G nazveme *vnější stěnou*. [Pat13]

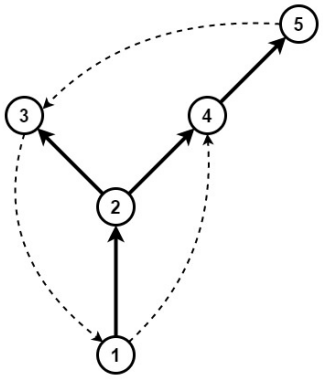
Poznámka. Každý graf má právě jednu vnější stěnu.

Definice 5.4 (klasifikace hran grafu)**.** Mějme orientovaný graf G. Aplikujme na graf G algoritmus prohledávání do hloubky. Hrany grafu, pomocí kterých jsme v průběhu algoritmu nalézali doposud nenavštívené vrcholy grafu, tvoří tzv. *DFS-strom grafu* G. Tyto hrany nazveme *stromové*. Je-li počáteční vrchol hrany e předchůdcem jejího koncového vrcholu v DFS-stromě a hrana e není stromová, nazveme ji *dopřednou*. Je-li naopak počáteční vrchol hrany e následníkem jejího koncového vrcholu v DFS-stromě, označíme hranu jako *zpětnou*. Pokud žádný krajní vrchol hrany e není předchůdcem druhého krajního vrcholu, říkáme, že hrana je *příčná*. [MM96]

Na obrázku 5.1 vidíme orientovaný graf. Vrcholy jsou označeny číslem, které popisuje pořadí nalezení daného vrcholu. Vrchol 1 je kořenem DFS-stromu. Hrany $(1, 2), (2, 3), (2, 4)$ a $(4, 5)$ jsou stromové. Hrana $(3, 1)$ je zpětná, hrana $(1, 4)$ dopředná a hrana $(5, 3)$ příčná.

Poznámka. Podobně lze klasifikovat i hrany grafu neorientovaného. Pak se z každé neorientované hrany stane orientovaná a celý graf bude též orientovaný. Množiny dopředných a příčných hran tohoto grafu však budou prázdné. Množinu stromových hran grafu budeme nadále značit T, množinu zpětných hran B.

Pro následující definice uvažujme existenci neorientovaného 2-souvislého grafu $G = (V, E)$, na němž proběhl algoritmus prohledávání do hloubky. Vrcholy tohoto grafu budeme značit přirozenými čísly podle pořadí, v jakém byly objeveny při prohledávání grafu. Hranám grafu byla určena orientace podle předešlé poznámky a definice. Povšimněme si, že takový graf obsahuje nejvýše jednu hranu vycházející ze svého kořene. Tuto skutečnost lze jednoduše odůvodnit. Pokud je graf tvořen pouze jedním vrcholem, neobsahuje žádné hrany. Pokud obsahuje více vrcholů, musí existovat hrana vedoucí z jeho



Obrázek 5.1: Klasifikace hran grafu.

kořene (jinak by graf nemohl být 2-souvislý). Pokud by existovala další hrana vedoucí z jeho kořene, pak by při odebrání kořene z množiny vrcholů graf již nebyl souvislý (kdyby byl souvislý, algoritmus DFS by nemohl proběhnout korektně). To by však opět znamenalo spor s 2-souvislostí.

Zaveďme následující značení. Sled s počátečním vrcholem u a koncovým vrcholem v o minimální délce 1, jehož hrany náleží množině X, budeme značit $u \xrightarrow[X]{} v$. Sled s počátečním vrcholem u a koncovým vrcholem v o minimální délce 0, jehož hrany náleží množině X, budeme značit $u \xrightarrow[X]{*} v$. Je-li délka sledu skutečně 0, musí nastat rovnost $u = v$.

Definice 5.5 ($V(e), low, low_2$)**.** Mějme stromovou hranu $e = (u, v) \in E$. Definujme množinu vrcholů $V(e)$ následovně:

$$V(e) = \left\{ x \middle| x \in V, v \xrightarrow[T]{*} x \right\}.$$

Dále definujme vrchol

$$low(v) = min\left(\left\{ w \middle| (u, w) \in B, u \in V(e) \right\} \cup \left\{ v \right\} \right)$$

a

$$low_2(v) = min\left(\left(\left\{ w \middle| (u, w) \in B, u \in V(e) \right\} \cup \left\{ v \right\} \right) \setminus \left\{ low(v) \right\} \right).$$

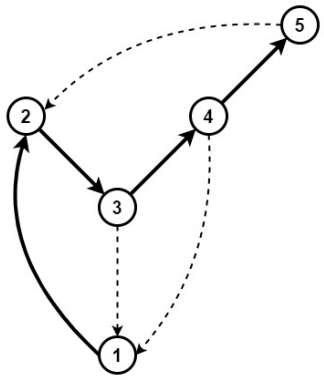
[Pat13], [MM96]

Poznámka. Množinu $V(e)$ chápeme jako množinu všech vrcholů grafu G, do nichž vede orientovaný sled libovolného množství stromových hran.

Vrcholy $low(v)$ a $low_2(v)$ můžeme díky vastnostem 2-souvislého grafu chápat jako vrcholy s nejnižším označením, do nichž vede orientovaný sled libovolného množství stromových hran ukončený nejvýše jednou zpětnou hranou.

Lze si všimnout, že každý 2-souvislý graf obsahující alespoň jednu hranu určitě obsahuje hranu $e = (1, 2)$. To plyne přímo z podstaty principu výstavby DFS-stromu. Hodnoty $low(2)$ a $low_2(2)$ jsou v tom případě pořadě 1 a 2. Je zřejmé, že musí existovat vrchol v, z něhož vede zpětná hrana do vrcholu 1. Kdyby tomu tak nebylo, po odebrání vrcholu 2 z grafu by se porušila jeho spojitost, což je ve sporu s 2-souvislostí grafu. Graf je nutně souvislý a jelikož z jeho kořenu (vrcholu 1) vede pouze jedna stromová hrana, a to do vrcholu 2, musí být vrchol v potomkem vrcholu 2. Nyní už je zřejmé, že existuje sled s počátečním vrcholem 2 a koncovým vrcholem 1, jehož poslední hrana $(v, 1)$ je zpětná a ostatní jsou stromové, a proto můžeme prohlásit, že $low(2) = 1$. Hodnotu $low_2(2)$ lze odvodit z definice - množina, jejíž minimum hledáme, neobsahuje vrchol 1, ale obsahuje vrchol 2. Menší hodnoty tedy nelze dosáhnout.

Pro lepší porozumění zavedených pojmů uveďme následující příklad. V tabulce 5.1 jsou uvedeny vlastnosti hran grafu znzorněného na obrázku 5.2.



Obrázek 5.2: Určení vlastností hran grafu.

Stromová hrana e	$V(e)$	$low(e)$	$low_2(e)$
$(1, 2)$	$\{2, 3, 4, 5\}$	1	2
$(2, 3)$	$\{3, 4, 5\}$	1	3
$(3, 4)$	$\{4, 5\}$	1	4
$(4, 5)$	$\{5\}$	2	5

Tabulka 5.1: Vlastnosti stromových hran grafu z obrázku 5.2.

Definice 5.6. Nechť $e = (u, v) \in E$ je hrana grafu G. *Cyklus* pak definujeme následujícím vzorcem:

$$C(e) = \begin{cases} low(v) \xrightarrow[T]{*} w \cup (w, low(v)), & \text{pro } e \in T; \\ (u, v) \cup v \xrightarrow[T]{*} u, & \text{pro } e \in B. \end{cases}$$

Označme vrcholy cyklu $C(e)$: $C(e) = w_0 \underset{T}{\rightarrow} w_1 \underset{T}{\rightarrow} \cdots \underset{T}{\rightarrow} w_n \underset{B}{\rightarrow} w_0$. Pro část cesty $C(e)$ začínající vrcholem w_0 a končící vrcholem u zaveďme termín

hlavní část cyklu $C(e)$. Je-li e stromová, pak cestu začínající vrcholem v a končící vrcholem w_n označme pojmem *vedlejší část cyklu* $C(e)$. Řekneme, že hrana (x, y) *vystupuje z cyklu* $C(e)$, pokud cyklus $C(e)$ obsahuje vrchol x, ale neobsahuje hranu (x, y). [MM96]

Poznámka. Cyklus $C(e)$ můžeme interpretovat jako kružnici, která splňuje následující tři podmínky:

- obsahuje hranu e,
- obsahuje vrchol $low(v)$,
- obsahuje právě jednu zpětnou hranu.

V případě, že e je stromová hrana, cyklus nemusí být určen jednoznačně. Hlavní část cyklu nikdy není prázdná, vždy obsahuje alespoň vrchol w_0.

Pro lepší srozumitelnost následující definice označme symblem $\tilde{G}(e)$ graf $\tilde{G}(e) = (V_e, E_e)$ podgraf grafu G, kde $V_e = V(e) \cup \Big\{v|(u, v) \in B, u \in V(e)\Big\}$ a $E_e = \Big\{(u, v)|(u, v) \in B, u \in V(e)\Big\}$.

Definice 5.7. Nechť $e = (u, v) \in E$ je hrana grafu G. Potom definujeme *segment* $S(e)$ podle vzorce

$$S(e) = \begin{cases} C(e) \cup G(V(e)) \cup \tilde{G}(e), & \text{pro } e \in T; \\ (u, v) \cup v \underset{T}{\rightarrow} u, & \text{pro } e \in B. \end{cases}$$

Řekneme, že segment $S(e)$ má *silně rovinné nakreslení*, pokud hlavní část cyklu $C(e)$ leží na hranici vnější stěny. Takový segment potom označíme jako *silně rovinný*. [Pat13], [MM96]

Poznámka. Segment $S(e)$ je vždy 2-souvislý graf. Narozdíl od cyklu je jednoznačně určen. Lze ho chápat jako sjednocení všech kružnic v grafu G, které splňují dvě podmínky:

- obsahuje hranu e,
- obsahují právě jednu zpětnou hranu.

Lze si také povšimnout, že každý segment $S(e)$ je sjednocením cyklu $C(e)$ a všech segmentů $S(\bar{e})$, kde $\bar{e}$ jsou hrany, které vystupují z cyklu $C(e)$.

Definice 5.8 (Připojení)**.** Mějme cyklus $C(e)$. Nechť hrana $\bar{e} = (u, v)$ vystupuje z cyklu $C(e)$. Definujme *připojení* $A(\bar{e})$ jako množinu vrcholů

$$A(\bar{e}) = \begin{cases} \{u\} \cup \Big\{y \Big| y \notin V(\bar{e}), \exists (x, y) \in B : x \in V(\bar{e})\Big\}, & \text{pro } \bar{e} \in T; \\ \{u, v\}, & \text{pro } \bar{e} \in B. \end{cases}$$

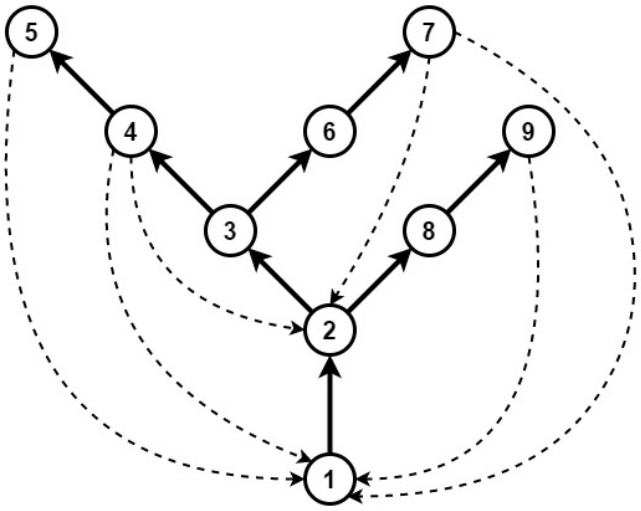
[MM96], [Pat13]

Poznámka. Připojení $A(\bar{e})$ obsahuje vrcholy, pomocí nichž je segment $S(\bar{e})$ napojen na cyklus $C(e)$, z něhož hrana $\bar{e}$ vystupuje.

Pomocí obrázku 5.3 se opět pokusíme ukázat nové pojmy na konkrétním grafu. Existuje pouze jediný cyklus hrany $(4,2)$, neboť tato hrana je zpětná, $C\Big((4,2)\Big) = (2,3,4)$. Hrana $(8,9)$ je stromová, ale i přesto pro ni také existuje pouze jeden cyklus, $C\Big((8,9)\Big) = (1,2,8,9)$. Pro hranu $(3,4)$ existují dva cykly: $C\Big((3,4)\Big) = (1,2,3,4)$ a $C\Big((3,4)\Big) = (1,2,3,4,5)$. Povšimněme si, že $(2,3,4)$ není cyklus hrany $(3,4)$ - neobsahuje totiž vrchol $low(4) = 1$.

Segment hrany $(2,3)$ je celý graf po odebrání vrcholů 8 a 9. Segment hrany $(2,8)$ je pouze kružnice $(1,2,8,9)$. Segment hrany $(1,2)$ je celý graf - toto tvrzení platí pro všechny 2-souvislé grafy a lze jednoduše dokázat z definice segmentu, neboť $V\Big((1,2)\Big) = V$, a tedy $G\Big(V\Big((1,2)\Big)\Big) = G(V)$, což je celý původní graf.

Zvolme si cyklus $C = C\Big((2,3)\Big) = (1,2,3,4)$. Vedlejší část cyklu C je cesta $(3,4)$. Hrany vystupující z cyklu C jsou $(3,6)$ a $(4,5)$. Pro tyto hrany nalezneme připojení $A\Big((3,6)\Big) = \{1,2,3\}$ a $A\Big((4,5)\Big) = \{1,5\}$.

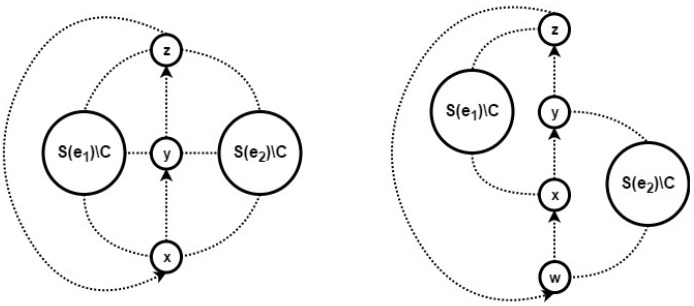


Obrázek 5.3: Graf pro demonstraci pojmů segment, cyklus a připojení

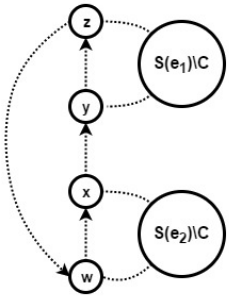
Definice 5.9 (konfliktní a kompatibilní segmenty)**.** Mějme cyklus C, z něhož vystupují dvě různé hrany e_1 a e_2. Segmenty $S(e_1)$ a $S(e_2)$ označíme jako *konfliktní*, pokud je splněna alespoň jedna z následujících podmínek:

- $|A(e_1) \cap A(e_2)| > 2$ (viz obrázek 5.4 vlevo),
- $\exists w, y \in A(e_1) \wedge \exists x, z \ \in A(e_2) : w < x < y < z$ (viz obrázek 5.4 vpravo).

V opačném případě segmenty označíme jako *kompatibilní*. Příklad kompatibilních segmentů vykresluje obrázek 5.5. [Pat13]



Obrázek 5.4: Grafy s konfliktními segmenty



Obrázek 5.5: Graf s kompatibilními segmenty

5.1.2 Popis algoritmu

Uveďme nejprve v krátkosti hlavní myšlenkové kroky algoritmu, později objasníme jejich význam [GX88], [MM96], [HT74], [Cha96], [Pat13].

1. Rozdělme graf na komponenty 2-souvislosti.
2. V každé komponentě 2-souvislosti přiřaďme hranám vhodnou orientaci.
3. Uspořádejme hrany grafu do vhodného pořadí pro testování rovinnosti.
4. Zvolme počáteční hranu, následně cyklus, který tuto hranu obsahuje. Testujme silnou rovinnost segmentů hran vystupujících z daného cyklu.
5. Testujme, zda lze sloučit silně rovinnná nakreslení segmentů tak, aby výsledné nakreslení grafu zůstalo silně rovinné.

První krok algoritmu je motivován následujícím tvrzením zmíněném v [GX88].

Tvrzení 5.10 (Postačující podmínka pro rovinnost grafu)**.** *Mějme graf G. Jsou-li jeho komponenty 2-souvislosti rovinné, potom je graf G rovinný.*

Poznámka. Tvrzení lze dokázat za pomoci indukce podle počtu 2-souvislých komponent grafu G, které bychom postupně přidávali k nakreslení. Ve skutečnosti platí toto tvrzení v podobě ekvivalence, jako je zmíněno v [GX88]. Obrácená implikace, zmíněná v modifikované podobě jako *Věta III.2* v [Zel77], by šla jednoduše dokázat sporem, ale pro tento algoritmus není důležitá.

Předešlé tvrzení jinými slovy říká, že pokud se podaří dokázat rovinnost každé 2-souvislé komponenty zadaného grafu, je rovinný celý graf G. Již bylo zmíněno, že segmentem hrany vedoucí z kořene DFS-stromu 2-souvislého grafu G je celý graf G. `Hopcroft-Tarjan` algoritmus na komponenty 2-souvislosti grafu pohlíží jako na segmenty těchto hran a testuje jejich silnou rovinnost, která je pro rovinnost postačující podmínkou.

Aby bylo možné provést druhý krok algoritmu, je třeba si uvědomit, že rovinnost grafu nezávisí na orientaci jeho hran. `Hopcroft-Tarjan` algoritmus vyžaduje, aby graf G obsahoval pouze stromové a zpětné hrany, čehož můžeme dosáhnout při použití algoritmu prohledávání do hloubky.

Podle [MM96] je v dalším kroku z důvodu efektivity algoritmu potřeba setřídit hrany grafu, tentýž zdroj poskytuje částečné uspořádání hran daného grafu:

- hranu (u, v) uvažujeme dříve než hranu (x, y), pakliže $u > x$,
- v případě rovnosti $u = x$ uvažujeme hranu (u, v) dříve, pokud platí $min(A(u, v)) < min(A(x, y))$,
- v případě rovnosti $min(A(u, v)) = min(A(x, y))$ uvažujeme hranu (u, v) dříve, pokud $|A(u, v)| = 2$ a zároveň $|A(x, y)| > 2$,
- mají-li obě připojení mohutnost 2, nebo je-li mohutnost obou připojení ostře větší než 2, pak tyto dvě hrany nelze porovnat.

Uvažujme následující zobrazení [HT74]:

$$\Phi((u,v)) = \begin{cases} 2 \cdot v, & \text{pro } (u,v) \in B; \\ 2 \cdot low(v), & \text{pro } (u,v) \in T, low_2(v) \geq u; \\ 2 \cdot low(v) + 1, & \text{pro } (u,v) \in T, low_2(v) < u. \end{cases}$$

Setřídíme-li hrany s týmž počátečním vrcholem vzestupně podle hodnoty zobrazení $\Phi()$, dosáhneme uspořádání popsaného výše.

Ve čtvrtém kroku algoritmu zvolíme hranu e_0 vedoucí z kořene DFS jako počáteční hranu. Cyklus $C(e_0)$ konstruujeme tak, aby hrana e_{i+1} měla nejnižší hodnotu $\Phi(e_{i+1})$ z hran, jejichž počátečním vrcholem je koncový vrchol hrany e_i. Pro test silné rovinnosti segmentů hran vystupujících z tohoto cyklu zavoláme algoritmus rekurzivně. Pokud v libovolné části algoritmu narazíme na segment, který není silně rovinný, okamžitě můžeme prohlásit, že zadaný graf není rovinný.

V posledním kroku algoritmu je třeba rozhodnout, zda lze silně rovinné segmenty grafu spojit do výsledného rovinného grafu. Je třeba zvolit, zda daný segment budeme kreslit dovnitř či vně cyklu C. Za tímto účelem budeme

testovat bipartitu grafu $IG(C)$ definovaného v [MM96], kde $C = C(e_0)$. Vrcholy grafu $IG(C)$ jsou segmenty původního grafu. Mezi dvěma vrcholy grafu $IG(C)$ existuje hrana právě tehdy, když jim příslušné segmenty původního grafu G jsou konfliktní. Je-li graf $IG(C)$ bipartitní, je celý graf G rovinný. Test bipartity grafu $IG(C)$ spočívá v postupném přidávání jednotlivých segmentů k již zkonstruované části grafu. Komponenty souvislosti grafu $IG(C)$ lze (vzestupně) seřadit na základě vrcholů jednotlivých segmentů, jimiž se připojují k cyklu C. Tyto hodnoty se pro jednotlivé komponenty souvislosti neprolínají, což je zřejmé z definice $IG(C)$. Segmenty ke grafu $IG(C)$ přidáváme podle pořadí hran definovaného za pomoci zobrazení $\Phi()$. Segment lze k již zkonstruované části grafu připojit bez porušení bipartity, pokud je nejmenší prvek připojení tohoto segmentu větší nebo roven maximálnímu prvku připojení alespoň jedné z partit každé komponenty souvislosti grafu $IG(C)$. Právě k této partitě potom nový segment přiřadíme. Předtím, než budeme induktivně přidávat další segment ke grafu $IG(C)$, bude třeba aktualizovat seznam jeho komponent souvislosti. Všechny komponenty, jejichž maximální prvek připojení je menší než minimální prvek připojení posledního připojeného segmentu, se nově spojí do jedné komponenty souvislosti [MM96], [HT74].

5.1.3 Implementace algoritmu

Do knihovny `jGraphT` jsem přidala balíček `org.jgrapht.alg.planarity`, jehož obsahem je implementace `Hopcroft-Tarjan` algoritmu pro zjištění rovinnosti grafu. Balíček obsahuje třídy `HopcroftTarjanPlanarityInspector`, `HopcroftTarjanStrongPlanarityInspector` a `VertexProperties`. Nyní si přiblížíme obsah tohoto balíčku.

V některých pasážích následujícího textu bude popis metod doplněn jejich pseudokódem. Poznamenejme, že symbol $\leftarrow$ vyjadřuje přiřazení a zápis $\{\}$ značí prázdný seznam.

třída `HopcroftTarjanPlanarityInspector`

První zmíněná třída je velice jednoduchá, obsahuje pouze jediný atribut `graph`, který uchovává zadaný graf, dále konstruktor a metodu `isGraphPlanar()`, jejíž návratový typ je `boolean`. Třída zajišťuje jednoduché předzpracování vstupního grafu a prvotní ořezávání na základě nutné podmínky pro rovinnost grafu z tvrzení 5.1.

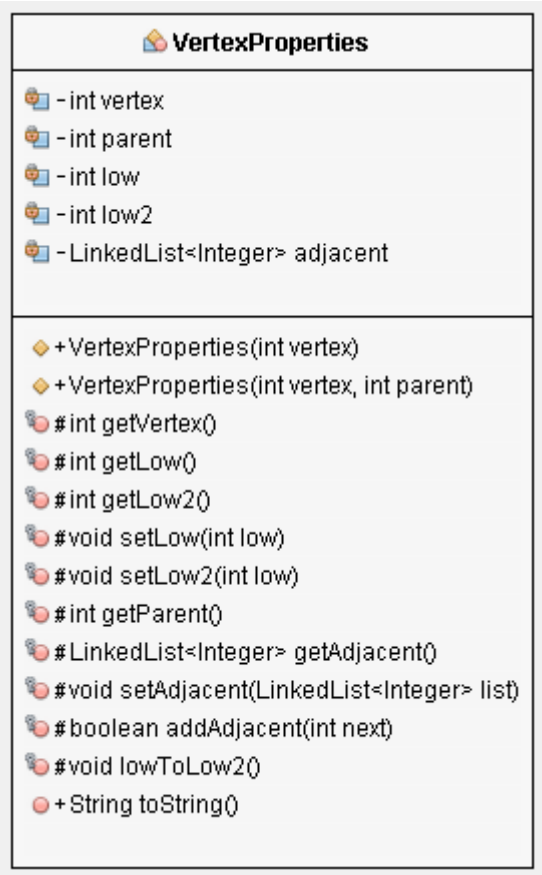
Parametrem konstruktoru je graf, jehož rovinnost je třeba otestovat. Pokud je graf orientovaný, převede jej pro své budoucí potřeby na neorientovaný.

Metoda `isGraphPlanar()` provede zmíněné ořezávání. Dále zavolá již naimplementovanou metodu, která postupně vrací všechny komponenty 2-souvislosti grafu, pro každou komponentu vytvoří instanci třídy pro testování silné rovinnosti a zavolá test silné rovinnosti. V případě, že narazí na komponentu, která není silně rovinná, vrací `false`. Pakliže jsou všechny komponenty souvislosti silně rovinné, vrací hodnotu `true`.

Při implementaci poslední zmíněné metody jsem si všimla, že metoda `getBiconnectedVertexComponents()` nepracuje vždy správně. Chyby vznikaly u grafů, které nebyly spojité. Proto v grafu nejprve najdu komponenty souvislosti a komponenty 2-souvislosti poté hledám u jednotlivých komponent souvislosti zvlášť.

třída `VertexProperties`

Třída `VertexProperties` napomáhá uchovávat informace o vrcholech grafu G. Její struktura je naznačena v Obrázku 5.6.



Obrázek 5.6: Diagram třídy VertexProperties

Třída má pět atributů, z nichž čtyři jsou celočíselné. Jedná se o atribut `vertex`, který uchovává pořadí vrcholu v DFS-stromu, dále `parent`, který uchovává pořadí přímého předchůdce tohoto vrcholu (v případě kořenu DFS-stromu uchovává hodnotu -1), a atributy `low` a `low2`, které uchovávají odpovídající charakteristiku stromové hrany vedoucí do daného vrcholu (v případě kořenu DFS-stromu jsou tyto hodnoty nastaveny na nulu). Posledním atributem je spojový seznam `adjacent`, který uchovává vrcholy, do nichž vede z daného vrcholu nějaká orientovaná hrana.

Také metody třídy jsou jednoduché. Hlavní konstruktor třídy má dva celočíselné parametry, které určují hodnoty atributů `vertex` a `parent`. Druhý

konstruktor má pouze jeden celočíselný parametr. Je to konstruktor pro kořen DFS-stromu. Pro každý atribut je implementován getter (tj. metoda, která vrátí hodnotu atributu konkrétní instance třídy). Pro atributy `low`, `low2` a `adjacent` jsou implementovány settery (tj. metody, které umožují změnit hodnotu atributu dané instance). Dále je definována metoda `to String()`, jejíž návratovou hodnotou je řetězec s výpisem hodnot atributů konkrétní instance této třídy; metoda `lowToLow2()`, která zkopíruje hodnotu atributu `low` do atributu `low2`, a metoda `addAdjacent()` s jedním celočíselným parametrem, která tento parametr přidá na konec spojového seznamu `adjacent`.

třída `HopcroftTarjanStrongPlanarityInspector`

Poslední navržená třída má dva atributy typu `ArrayList`. Jedná se o seznam `vertexList`, v němž se uchovávají vrcholy zadaného grafu, a seznam `vertexProperties`, který uchovává objekty typu `VertexProperties` nesoucí informace o vrcholech ze seznamu `vertexList` v odpovídajícím pořadí.

Při zavolání jediného konstruktoru této třídy proběhne inicializace obou atributů třídy. Proběhne sestavení DFS-stromu grafu. Pro každý vrchol se spočítají hodnoty low, low_2 a sestaví se spojový seznam vrcholů, do nichž vede nějaká orientovaná hrana grafu. Tento seznam se následně setřídí za pomoci funkce $\Phi()$.

Nejdůležitější metodou této třídy je metoda `isSegmentStronglyPlanar()`. Tato metoda má dva celočíselné parametry, které popisují hranu určující testovaný segment, jejím návratovým typem je `LinkedList`. Metoda vrací upravený seznam připojení, pokud je segment silně rovinný, v opačném případě vrací `null`. Pseudokód její implementace je popsán v algoritmu 1.

Metoda `isComponentPlanar()` s návratovým typem `boolean` vyhodnocuje, zda je konkrétní komponenta 2-souvislosti rovinná. Slouží pro inicializaci algoritmu. Po zavolání metody se volá `isSegmentStronglyPlanar()` s parametry odpovídajícími hraně vedoucí z kořene DFS-stromu. Následně metoda vyhodnotí výsledek a vrátí hodnotu `false` v případě, že komponenta 2-souvislosti rovinná není, v opačném případě vrátí hodnotu `true`.

Metoda `phi()` má dva celočíselné parametry, které popisují hranu grafu, vrací hodnotu funkce Φ pro zadanou hranu (její návratový typ je tedy `int`).

Metoda `getListMax()` má jako parametr seznam typu `LinkedList`, jehož prvky jsou přirozená čísla. Metoda vrací maximální prvek tohoto seznamu, nebo -1, pokud je seznam prázdný. Jejím návratovým typem je opět `int`.

Metoda `radsort` slouží k setřídění sousedních vrcholů daného vrcholu. Má dva vstupní parametry: celé číslo, které určuje daný vrchol, a spojový seznam sousedních vrcholů. Vrací setříděný spojový seznam sousedních vrcholů. Pro seřazení souseních vrcholů konkrétního vrcholu byl použit algoritmus `radix sort`. Tento algoritmus však požaduje omezenou množinu klíčů vstupního seznamu. Seřazení souseních vrcholů probíhá na základě hodnoty funkce Φ, jejíž obor hodnot je skutečně omezený - jeho minimem je dvojnásobek hodnoty nejnižšího označení vrcholu grafu a jeho maximem je dvojnásobek hodnoty nejnižšího označení vrcholu grafu zvětšený o 1. Díky tomu je skutečně možné vybraný algoritmus použít.

Algoritmus 1 Určení silné rovinnosti segmentu

```
function isSegmentStronglyPlanar(S, T)
    result ← {}
    aLefts ← {}
    aRights ← {}
    if (počet vrcholů grafu ≤ 4) then
        return result
    end if
    spine ← vedlejší část cyklu C((S,T)) pozpátku
    if (hrana S,T je zpětná) then
        result připoj T
    else
        result připoj poslední prvek spine
    end if
    while (spine není prázdný) do
        source ← vyjmi poslední prvek spine
        actAdjacent ← vedlejší vrcholy vrcholu source kromě prvního
        for all tagret ∈ actAdjacent) do
            A ← isSegmentStronglyPlanar(source, target)
            if (A = null) then
                return null
            end if
            if (bipartityTestAndComponentsUpdate = false) then
                return null
            end if
        end for
        parent ← rodič vrcholu source
        odstraň parent z posledního seznamu v aLefts
        odstraň parent z posledního seznamu v aRights
    end while
    w1 ← low(T)
    while (aLefts není prázdný) do
        alb ← vyjmi poslední prvek aLefts
        arb ← vyjmi poslední prvek aRights
        if ( (max(alb) ≥ w1)  &  ((max(arb)≥ w1)) then
            return null
        end if
        if (max(alb) ≥ w1) then
            result připoj seznam arb
            result připoj seznam alb
        else
            result připoj seznam alb
            result připoj seznam arb
        end if
    end while
    return result
end function
```

Metoda `maxComponentAttachment` má dva parametry typu `LinkedList`, které zastupují seřazené seznamy připojení určité komponenty souvislosti grafu $IG(C)$. Vrací větší z maximálních prvků těchto dvou seznamů, nebo hodnotu -1, pokud jsou oba seznamy prázdné.

Metoda `attachmentSort` slouží k seřazení spojového seznamu připojení, který je také jejím vstupním parametrem. Za tímto účelem opět využívá algoritmu `radix sort`, což je možné, neboť hodnoty jednotlivých připojení mohou nabývat pouze hodnot označení vrcholů, a tedy jen omezeného množství různých hodnot. Vrací vzestupně seřazený spojový seznam připojení.

Metoda `bipartityTestAndComponentsUpdate` je poslední pomocnou metodou této třídy. Jejími vstupními parametry jsou tři spojové seznamy a celé číslo. Jedná se o dva seznamy, jejichž prvky jsou opět spojové seznamy připojení komponent souvislosti jednotlivých partit grafu, dále spojový seznam připojení právě zpracovaného segmentu a hodnota jeho minimálního prvku připojení. Návratový typ této metody je `boolean`. Metoda vrací `true`, pakliže právě zpracovávaný segment lze zařadit do jedné z komponent souvislosti již zpracované části grafu $IG(C)$, v opačném případě vrací `false`. Její implementace je přiblížena pseudokódem v algoritmu 2.

5.1.4 Příklad použití implementovaného algoritmu

Na závěr uveďme příklad kódu, který vygeneruje náhodný graf s deseti vrcholy a dvaceti hranami a otestuje jeho rovinnost:

```
GnmRandomGraphGenerator gg
    = new GnmRandomGraphGenerator(10, 20);
Graph g = new SimpleGraph(DefaultEdge.class);
VertexFactory<Integer> vf
    = new VertexFactory<Integer>() {
        int counter = 0;
        @Override
        public Integer createVertex() {
            return counter++;
        }
    };
gg.generateGraph(g, vf, null);
HopcroftTarjanPlanarityInspector pi
    = new HopcroftTarjanPlanarityInspector(g);
pi.isGraphPlanar();
```

Algoritmus 2 Test bipartity dosavadní části grafu $IG(C)$ a aktualizace jeho komponent

```
function BIPARTITYTESTANDCOMPONENTSUPDATE(m, A, aLefts, aRights)
    if (graf IG(C) zatím nemá žádné komponenty) then
        aLefts přidej prvek A
        aRights přidej prvek {}
        return true
    end if
    ar ← {}
    al ← A
    temp ← poslední z aLefts spoj s posledním z aRights
    while (aLefts není prázdný & (max(temp) > m)) do
        if (max(poslední z aLefts) > m) then
            prohoď poslední aLefts a poslední aRights
        end if
        if (max(poslední z aLefts) > m) then
            return false
        end if
        al přidej všechny prvky posledního aLefts
        ar přidej všechny prvky posledního aRights
        odstraň poslední prvek aRights
        odstraň poslední prvek aLefts
    end while
    al ← attachmentSort(al)
    ar ← attachmentSort(ar)
    aLefts přidej prvek al
    aRights přidej prvek ar
    return true
end function
```

Kapitola 6

Závěr

V rámci řešení své bakalářské práce jsem se podrobně seznámila se strukturou grafové knihovny `jGraphT`, naučila jsem se s ní pracovat a následně provedla několik testů již implementovaných algoritmů.

Dále jsem se pokusila vyhledat co největší množství jiných knihoven, které se zabývají grafovou teorií, a tyto knihovny porovnat.

Nastudovala jsem `Hopcroft-Tarjan` algoritmus, pomocí něhož lze určit, zda je zadaný graf rovinný. Algoritmus jsem se snažila řádně vysvětlit, k objasnění významu některých použitých pojmů jsem vytvořila ilustrační obrázky (matematické grafy byly vytvořeny v softwaru dostupném na https://www.draw.io/). V [Pat13] je uvedeno, že tento algoritmus nedosahuje nejvyšší rychlosti mezi známými algoritmy pro testování rovinnosti, dle autorů je však schopen otestovat rovinnost v čase $O(n)$, kde n značí počet vrcholů grafu. Algoritmus jsem naprogramovala a začlenila ho do knihovny `jGraphT`. Následně jsem se pokusila otestovat jeho správnost. Systematicky jsem prošla všechny grafy, jejichž množina vrcholů má mohutnost nejvýše šest. Tyto grafy vyhodnotila má verze algoritmu korektně. Poté jsem algoritmus testovala na rozsáhlejších mnou zadaných nebo náhodně vygenerovaných rovinných i nerovinných grafech, ani v tomto případě jsem neobjevila chybu. Systematické testování pro grafy s množinou vrcholů mohutnosti 7 by bylo složité, neboť by bylo třeba otestovat 2^{21}, což je více než 2000000 různých grafů. Implementaci jsem doprovodila stručnou dokumentací formou javadocu a podrobným popisem uvedeným v tomto textu.

Své rozšíření pro knihovnu přikládám na CD. Po rozšíření dokumentace v angličtině plánuji svou prací rozšířit repozitář knihovny `jGraphT` na serveru `GitHub`.

V budoucnu je možné rozšířit hlavní třídu o metodu, která vrací konkrétní vnoření grafu popisující jeho rovinné nakreslení.

Příloha A

Rejstřík

Symbols

A

B

C

D

E

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Z

Příloha B

Literatura

[Boo] *Boost c libraries*, `http://boost.org/doc/libs/1_60_0/libs/graph/doc/index.html`.

[Cha96] Siddhartha Chatterjee, *Comp 122: Algorithms and analysis testing graph planarity (part 1 of 2)*.

[ČRK04] Roman Čada, Zdeněk Ryjáček, and Tomáš Kaiser, *Diskrétní matematika*, Západočeská univerzita, 2004.

[Dem02] Jiří Demel, *Grafy a jejich aplikace*, Academia, 2002.

[Eks07] Jan Ekstein, *Hamiltonian cycles in the square of a graph with block graph homeomorphic to a star*.

[GA] *Graph algorithms*, `http://algorithmic-solutions.info/leda_manual/Graph_Algorithms.html`.

[Gra] *Graphstream - graphstream - a dynamic graph library*, `http://graphstream-project.org/`.

[GX88] David Gries and Jinyun Xue, *The hopcroft-tarjan planarity algorithm, presentation and improvements*, Tech. report, Cornell University, 1988.

[Har] Frank Harary, *Graph theory. 1969*.

[Hog] Luc Hogie, *High performance graph library for java*, `http://i3s.unice.fr/~hogie/grph`.

[HT74] John Hopcroft and Robert Tarjan, *Efficient planarity testing*, Journal of the ACM (JACM) **21** (1974), no. 4, 549–568.

[igr] *igraph – the network analysis package*, `http://igraph.org/`.

[JDS] *Data structures library in java*, `http://cs.brown.edu/cgc/jdsl/`.

[JUN] *Jung*, `http://jung.sourceforge.net/`.

[MM96] Kurt Mehlhorn and Petra Mutzel, *On the embedding phase of the hopcroft and tarjan planarity testing algorithm*, Algorithmica **16** (1996), no. 2, 233–242.

[MN07] Jiří Matoušek and Jaroslav Nešetřil, *Kapitoly z diskrétní matematiky*, Karolinum, 2007.

[Nav] Barak Naveh, *Jgrapht*, `http://jgrapht.org/`.

[Net] *Overview — networkx*, `http://networkx.github.io/`.

[Pat13] Maurizio Patrignani, *Planarity testing and embedding.*, 2013.

[yWo] yWorks, *yfiles diagramming libraries*, `http://yworks.com/`.

[Zel77] Bohdan Zelinka, *Rovinné grafy*, Mladá fronta, 1977.