Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Knihovna pro analýzu grafových struktur

Plzeň 2014 Jakub Žáček

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 25. června 2014

Jakub Žáček

Poděkování

Rád bych poděkoval Ing. Richardu Lipkovi Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Abstract

This bachelor thesis deals with the construction of precedence graphs from a list of actions, their prerequisites and followers. You will find here the basics of graph theory, description of several existing libraries for graph analysis in Java programming language, a detailed comparison of two selected libraries. Futhermore it describes the development of a new library, which constructs graphs from the action list mentioned above. This work also contains an example of solution of this problem based on real data.

Abstrakt

Tato bakalářská práce se zabývá tvorbou grafů ze seznamu akcí, jejich prerekvizit a následníků. Naleznete zde základy grafové teorie, popis některých existujících knihoven pro práci s grafy v programovacím jazyce Java, podrobné srovnání dvou z těchto knihoven a popis vývoje nové knihovny, která tvoří grafy z výše uvedeného seznamu akcí. Dále je zde i ukázka řešení daného problému na reálných datech.

Obsah

1 Úvod **1**

2 Grafy **2**
2.1 Teorie grafů . . . 2
2.1.1 Metriky grafů . . . 2
2.2 Implementace grafů . . . 3
2.2.1 Implementace maticí . . . 3
2.2.2 Implementace spojovou strukturou . . . 3
2.3 Grafové algoritmy . . . 4
2.3.1 Procházení grafů . . . 4
2.3.2 Hledání nejkratších cest . . . 4
2.3.3 Hledání koster grafů . . . 5
2.3.4 Některé další algoritmy . . . 5
2.4 Vizualizace grafů . . . 6

3 Existující implementace **8**
3.1 JGraph a JGraphT . . . 8
3.2 JSL . . . 9
3.3 JUNG . . . 9
3.4 Grph . . . 10
3.5 FlexiGraph . . . 11
3.6 GraphStream . . . 11
3.7 JDSL . . . 12
3.8 yFiles for Java . . . 12

4 Podrobný rozbor knihoven **13**
4.1 Knihovna JUNG . . . 13
4.1.1 Vizualizace . . . 13
4.1.2 Práce se soubory . . . 15
4.2 Knihovna GraphStream . . . 15
4.2.1 Vizualizace . . . 16

4.2.2 Práce se soubory 17
4.3 Srovnání obou knihoven 17
4.3.1 Vizualizace grafů 17
4.3.2 Základní algoritmy 22
4.3.3 Práce se soubory 23
4.3.4 Generování grafů 26

5 Návrh knihovny pro sestavení precedenčního grafu 27
5.1 Načtení akcí ze souboru 27
5.2 Vytvoření výsledného grafu 27
5.2.1 Definice Akce 27
5.2.2 Algoritmus 28
5.3 Prezentace výsledku uživateli 31

6 Implementace knihovny 32
6.1 Použití knihovny 34
6.2 Testování knihovny 36

7 Závěr 39

A Uživatelská dokumentace 43

B Seznam použitých zkratek 49

1 Úvod

Cílem této práce je vytvořit novou knihovnu v programovacím jazyce Java, která bude vytvářet grafy z řady akcí, jejich prerekvizit[1] a následníků tak, aby všechny tyto vazby byly splněny. Akce může reprezentovat libovolnou událost. Tato knihovna bude také obsahovat jednoduché grafické rozhraní umožňující uživateli vybrat potřebné údaje a následně zobrazit vizualizaci výsledného grafu. K práci s grafy a jejich vizualizaci bude využita některá z již existujících knihoven pod svobodnou licencí.

Výsledná knihovna bude mít využití v plánování testovacích scénářů, které se budou skládat z jednotlivých akcí či segmentů scénáře. Dalším možným využitím je generování precedenčních grafů[2]. Jiným možným využitím je jednoduché plánování různých projektů a všeobecně hledání správného sledu různých akcí.

V první části této práce budou rozebrány základy grafů a grafové teorie, jejich základní implementace v počítači, algoritmy a různé druhy vizualizace. Dále zde bude stručný popis některých stávajících knihoven v Javě pro vizualizaci grafů a práci s nimy a následně podrobné srovnání výběru některých z těchto knihoven.

Druhá část této práce se bude zabývat návrhem samotné knihovny (výběr vhodných algoritmů, konstrukcí atd.). Knihovna bude velmi univerzální – nebude vyžadovat žádný konkrétní formát vstupních souborů. Dále pak její implementací a testováním. Také zde bude popis jednoduché aplikace (uživatelského rozhraní) používající tuto knihovnu.

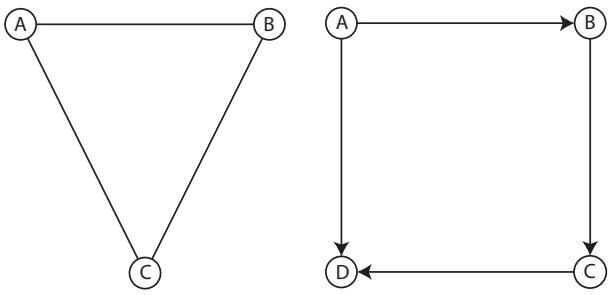
[1] akce, které musí nastat před danou akcí

[2] orientovaný graf neobsahující žádné smyčky

2 Grafy

2.1 Teorie grafů

Graf[1] je reprezentací množiny objektů a jejich spojení – skládá se tedy z vrcholů/uzlů (reprezentace objektů) a hran (jejich propojení). Graf je uspořádaná dvojice $G = \langle V, E \rangle$, kde V je množina vrcholů a E množina hran. Prvky množiny hran E jsou definovány jako $E \subseteq \{\{u, v\} | u, v \in V, u \neq v\}$ u neorientovaného grafu a $E \subseteq V \times V$ u grafu orientovaného. U orientovaných grafů záleží na směru propojení (tzv. orientované hrany). U neorientovaných naopak na směru nezáleží (viz obr. 2.1). U některých grafů mohou být též ohodnocené hrany.



Obrázek 2.1: Neorientovaný (vlevo) a orientovaný graf (vpravo)

2.1.1 Metriky grafů

- **počet vrcholů** – celkový počet vrcholů v grafu (V)
- **počet hran** – celkový počet hran v grafu (E)
- **stupeň vrcholu** – počet hran přímo spojených s daným vrcholem
- **vrcholy N-tého stupně** – vrcholy, které jsou spojeny s právě N hranami
- **vzdálenost dvou vrcholů** – nejkratší vzdálenost mezi dvěma vrcholy (nejmenší počet hran v případě neohodnoceného grafu, nejmenší součet ohodnocení v případě grafu ohodnoceného)

2.2 Implementace grafů

Existují dva základní způsoby, jak grafy implementovat – maticí (dvourozměrným polem) a spojovou strukturou. Popisuji zde jen základní implementaci, řešení může být v závislosti na konkrétním případě jiné.

2.2.1 Implementace maticí

Celý graf je reprezentován jedním dvourozměrným polem o velikosti $N \times N$, kde N je počet vrcholů v grafu. V případě neohodnoceného grafu může být pole typu `boolean` (`true` pokud hrana mezi vrcholy existuje, `false` pokud neexistuje). Graf může být pole typu `int`, a daná hodnota určuje ohodnocení hrany mezi dvěma vrcholy a v případě neexistence hrany může být hodnota např. -1.

Výhodou tohoto řešení je jednoduchost při programování. Dále se vyplatí paměťově u hustých grafů (počet hran se blíží druhé mocnině počtu vrcholů). Přidání a odebrání jednotlivých hran jsou velmi rychlé operace. Také operace zjištění, zda spolu dva vrcholy sousedí, či nikoliv, je velmi rychlá.

Nevýhodou je, že i u řídkých grafů (počet hran je řádově menší než druhá mocnina počtu vrcholů) je matice stále stejně velká. Přidávání nových vrcholů si žádá zvětšení celé matice.

2.2.2 Implementace spojovou strukturou

Graf se skládá z instancí třídy reprezentující graf, ve které je pole typu `Vrchol` obsahující sousední vrcholy grafu. V případě ohodnoceného grafu zde může být asociativní pole typu `Vrchol` a `int`, kde `int` je ohodnocení hrany. Samotný graf je pak uložen v třídě `Graf` obsahující jednorozměrné pole typu `Vrchol` se všemi vrcholy v daném grafu.

Velkou výhodou tohoto řešení je jeho dynamičnost a oproti předchozímu velmi malá paměťová náročnost (závislá zejména na počtu hran, nikoliv vrcholů). U řídkých grafů je pak výhodou nižší paměťová náročnost než u matice sousednosti. Dále lze snadno zjistit stupeň vrcholu. Přidávání nových uzlů je též snazší než u implementace maticí.

Nevýhodou je vyšší paměťová náročnost u hustých grafů (může být dokonce i vyšší než u matice sousednosti.

	Matice	**Seznam**
Jsou U a V sousedi	$O(1)$	$O(E/N)$
Nalezení potomka	$O(N)$	$O(E/N)$
Nalezení předka	$O(N)$	$O(N+E)$
Zjištění stupně vrcholu	$O(N)$	$O(1)$
Paměťová náročnost	$O(N^2)$	$O(N+E)$

Tabulka 2.1: Složitosti operací u jednotlivých implementací[2]

2.3 Grafové algoritmy

U grafů existuje široká škála algoritmů, které jsou schopny zjišťovat např. nejkratší cesty, kostry grafů a mnoho dalších informací. Uvedu zde jen ty, které jsou nejčastěji podporovány knihovnami popsanými v následující kapitole.

2.3.1 Procházení grafů

- **Prohledávání do šířky (Breadth-First Search – BFS)** – Tento algoritmus slouží k prohledání všech vrcholů grafu a následně nalezení nejkratší cesty (za předpokladu neohodnoceného grafu).
- **Prohledávání do hloubky (Depth-First Search – DFS)** – Používá se k nalezení cesty mezi dvěma vrcholy, přičemž tato cesta nemusí být vždy nejkratší. Ale její nalezení je rychlejší než u BFS.

2.3.2 Hledání nejkratších cest

- **Dijkstrův algoritmus** – Algoritmus je používán pro hledání nejkratších cest z jednoho vrcholu do všech ostatních v grafech s nezáporně ohodnocenými hranami.
- **Algoritmus A* (A star)** – Velmi podobný Dijkstrovu algoritmu (hledání nejkratších cest v nezáporně ohodnocených grafech) s tím, že přidává navíc heuristický prvek.

- **Floyd-Warshallův algoritmus** – Opět algoritmus umožňující nalezení nejkratších vzdáleností, ale v tomto případě již mohou být hrany záporně ohodnoceny. Graf však nesmí obsahovat cykly záporné délky.
- **Bellman-Fordův algoritmus** – Podobně jako Floyd-Warshallův algoritmus hledá cesty s nejnižším ohodnocením. Avšak zatímco Floyd-Warshallův algoritmus hledá nejkratší cesty ze všech vrcholů do všech ostatních vrcholů, Bellman-Fordův algoritmus vyhledává pouze nejkratší cestu z daného vrcholu do ostatních.

2.3.3 Hledání koster grafů

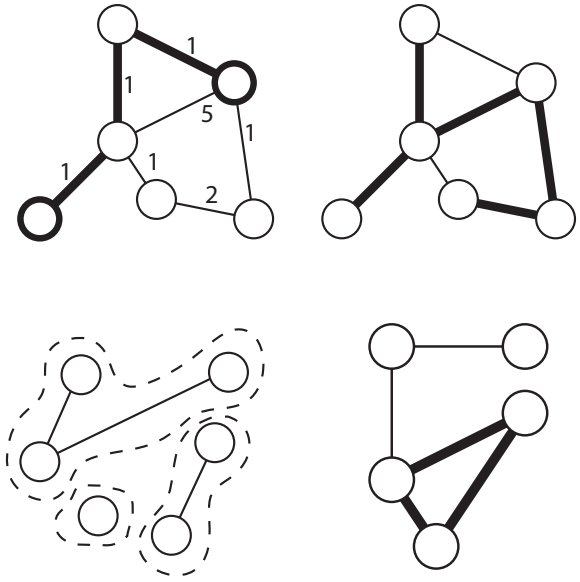
Kostra grafu je podgraf[1], který neobsahuje žádné cykly. Tj. libovolné dva vrcholy lze spojit vždy jen jednou cestou – jedná se o strom.

- **Kruskalův algoritmus** – Postupným vybíráním hran od nejkratších, které netvoří s již vybranými hranami žádnou kružnici, postupně nalezne minimální kostru grafu s nezáporně ohodnocenými hranami.
- **Borůvkův algoritmus** – Hledá minimální kostru grafu s nezáporně a různě ohodnocenými hranami.
- **Jarníkův algoritmus** – Hledá minimální kostru ohodnoceného grafu.

2.3.4 Některé další algoritmy

- **Hledání klik grafů** – Klika je takový podgraf nějakého grafu, který je úplným grafem, tj. všechny jeho vrcholy jsou spojeny hranou se všemi ostatními.
- **Hledání komponent grafů** – Komponenta grafu je takový souvislý podgraf, jenž není obsažen v žádném větším souvislém podgrafu.

[1] podgraf grafu je graf, v němž chybí některé hrany či vrcholy oproti původnímu grafu



Obrázek 2.2: Znázornění nejkratší cesty v grafu, kostry grafu, komponent grafu a kliky grafu (zleva)

2.4 Vizualizace grafů

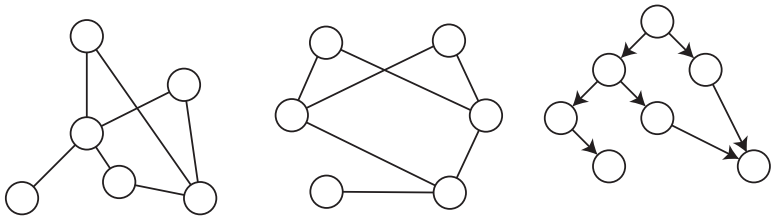
Protože graf jako takový je ryze abstraktní záležitost, je vhodné jej nějakým způsobem vizualizovat, aby s ním mohl člověk snáze pracovat. Nejčastějším způsobem vizualizace je zobrazení vrcholů jako bodů, kruhů nebo obdélníků a hran jako čar, případně šipek u orientovaných grafů (viz obr. 2.1). Tento způsob je nejjednodušší a mnohdy plně dostačující.

V některých případech jsou vrcholy různě velké, čímž je možno rozlišit váhu jednotlivých vrcholů, stejně tak tloušťky čar znázorňující hrany mohou být různé v závislosti na ohodnocení dané hrany. Někdy jsou navíc vrcholy či hrany znázorněny barevně. Vše záleží na konkrétním grafu a na tom, co reprezentuje.

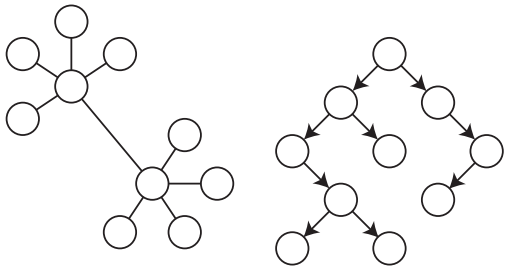
V takto vizualizovaných grafech mohou být vrcholy organizovány různými způsoby. Níže je popis a názorné obrázky (2.3 a 2.4) těch nejznámějších:

- **náhodné/ruční uspořádání** – vrcholy jsou náhodně/ručně (uživatelem) rozmístěny – nejjednodušší, ale nejméně přehledné
- **kruhové uspořádání** – vrcholy se nacházejí na pomyslné kružnici (např. u vizualizace WWW stránek)
- **hierarchické uspořádání** – u orientovaných grafů, vrcholy předků jsou výše než vrcholy potomků (např. vizualizace průběhu práce – workflow diagram)
- **organické uspořádání** – jednotlivé vrcholy mají navzájem odpudivé síly v závislosti na síle vazeb mezi sebou (např. vizualizace sociálních sítí)
- **stromové uspořádání** – podobné jako hierarchiské s tím rozdílem, že každý vrchol má jen jednoho předka (např. vizualizace dědičnosti jednotlivých tříd v javě)

Někdy jsou vizualizované grafy i animované, kdy je lépe vidět např. vývoj v průběhu času.



Obrázek 2.3: Náhodné/ruční, kruhové a hierarchické uspořádání (zleva)



Obrázek 2.4: Organické (vlevo) a stromové uspořádání (vpravo)

3 Existující implementace

Zde je přehled některých knihoven do programovacího jazyka Java, které umožňují vytvářet, reprezentovat a/nebo podporují různé algoritmy nad grafy. Souhrnou

3.1 JGraph a JGraphT

Knihovna JGraphT[3], která je stále aktivně vyvíjena, umožňuje reprezentaci orientovaných i neorientovaných grafů. Hrany v těchto grafech mohou být ohodnocené, neohodnocené nebo popsané.

Tato knihovna také podporuje širokou škálu algoritmů, například Floyd-Warshallův, Bellman-Fordův, Dijkstrův a přibližně dvacet dalších algoritmů. Je velmi dobře dokumentována na své webové stránce a v JavaDoc dokumentaci.

Naproti tomu knihovna JGraph[4] (nově JGraphX) je určena k vizualizaci a editaci grafů. Je také stále aktivně vyvíjena. Tato knihovna podporuje značné množství metod, jimiž lze různé grafy zobrazit nebo upravit. Existuje adaptér, jenž umožňuje v knihovně JGraph vizualizovat data reprezentovaná knihovnou JGraphT.

Název	JGraphT
Licence	GNU-LGPLv2, Eclipse Public License
Poslední verze	28. 4. 2014
Algoritmy	Dijkstra, Floyd-Warshall, Bellman-Ford, Bron-Kerbosch, detekce cyklů, Edmonds-Blossom, Edmonds-Karp, Hopcroft-Karp, Kruskal, K-shortest path, Kuhn-Munkres, Prim, Stoer-Wagner, Tarjan
Vizualizace	✗
Web	`http://www.jgrapht.org`

Tabulka 3.1: JGraphT

Název	JGraph
Licence	BSD License
Poslední verze	5. 5. 2014
Algoritmy	✗
Vizualizace	organické, kruhové, stromové, hierarchické uspořádání
Web	`http://github.com/jgraph/jgraphx`

Tabulka 3.2: JGraph

3.2 JSL

JSL[5] podporuje v původní verzi pouze algoritmy prohledávání do šířky, do hloubky a algoritmus A*, ale je možné jednoduše naprogramovat i další algoritmy díky jednoduchému rozhraní. Vývoj této knihovny byl ukončen v roce 2003, tudíž JSL již není aktivně vyvíjen. Obsahuje také velmi kvalitně popsaný JavaDoc.

Název	Java Search Library
Licence	Mozilla Public License
Poslední verze	15. 12. 2003
Algoritmy	BFS, DFS, A*
Vizualizace	✗
Web	`http://jsl.sourceforge.net`

Tabulka 3.3: JSL

3.3 JUNG

Knihovna JUNG[6], jejíž podpora skončila v roce 2010, podporuje orientované, neorientované grafy, grafy s paralelními hranami a hypergrafy. JUNG podporuje širokou škálu algoritmů od grafové teorie, přes data mining po statistickou analýzu. JUNG má kvalitní dokumentaci na stránkách vygenerovaných Mavenem a JavaDoc.

JUNG také podporuje vizualizaci grafů.

Název	Java Universal Network/Graph Framework
Licence	BSD License
Poslední verze	25. 1. 2010
Algoritmy	BFS, Dijkstra, Prim, shluková analýza, transformace grafů
Vizualizace	kruhové, stromové, kruhové-stromové, organické uspořádání
Web	`http://jung.sourceforge.net`

Tabulka 3.4: JUNG

3.4 Grph

Momentálně aktivně vyvíjená knihovna, která především nabízí snadné použití a malé paměťové nároky. Podporuje orientované i neorientované grafy. Mimo široké škály podporovaných algoritmů podporuje také vizualizaci grafů. Grph[7] má velmi podrobný uživatelský manuál.

Název	Grph
Licence	GNU-LGPL
Poslední verze	22. 4. 2014
Algoritmy	BFS, DFS, Dijkstra, Floyd-Warshall, Bellman-Ford, K-nejkratší cesta... (celkem více než 100, celý seznam na `http://www.i3s.unice.fr/~hogie/grph/grph-algos.pdf`)
Vizualizace	organické uspořádání
Web	`http://www.i3s.unice.fr/~hogie/grph`

Tabulka 3.5: Grph

3.5 FlexiGraph

Vývoj knihovny FlexiGraph[8] byl ukončen v roce 2007. Podporuje reprezentaci grafů a základní algoritmy jako DFS a BFS. FlexiGraph je doplněn o slušně popsaný JavaDoc.

Název	FlexiGraph
Licence	Apache 2.0 License
Poslední verze	20. 10. 2010
Algoritmy	BFS, DFS, hledání stromů, shluků, řazení podle stupňů
Vizualizace	kruhové, organické, náhodné uspořádání
Web	`https://code.google.com/p/flexigraph`

Tabulka 3.6: FlexiGraph

3.6 GraphStream

Aktivně vyvíjená knihovna univerzitou Le Havre podporující širokou škálu algoritmů a vizualizačních metod.

GraphStream[9] podporuje algoritmy jako A*, Dijkstrův, Bellman-Forsův, dále umožňuje generovat grafy různými algoritmy a řadu dalších funkcí. Použití každé funkce je podrobně zdokumentováno na vlastní webové stránce.

GraphStream podporuje vizualizační metody, které poskytují prakticky neomezené možnosti přizpůsobení.

Název	GraphStream
Licence	GNU-LGPL
Poslední verze	25. 4. 2014
Algoritmy	Dijkstra, Bellman-Ford, A*, Kruskal, Prim, Welsch-Powell, stupně vrcholů, hustota grafu, koeficienty shluků, spojené komponenty
Vizualizace	organické, ruční, hierarchické uspořádání
Web	`http://graphstream-project.org`

Tabulka 3.7: GraphStream

3.7 JDSL

Knihovna[10] vytvořená na univerzitě Brown již není vyvýjena od roku 2005. Podporuje algoritmy DFS, Dijkstra, Prim a topologické řazení. Má kvalitní dokumentaci a JavaDoc na svých webových stránkách.

Licencí je Brown University License (zdarma pro nekomerční využití při uvedení autora).

Název	Java Data Structures Library
Licence	Brown University License (zdarma pro nekomerční použití)
Poslední verze	1. 9. 2005
Algoritmy	DFS, Dijkstra, Prim, Topologické řazení
Vizualizace	✗
Web	`http://cs.brown.edu/cgc/jdsl`

Tabulka 3.8: JDSL

3.8 yFiles for Java

Komerční knihovna yFiles[11] je stále aktivně vyvíjena. Podporuje širokou škálu algoritmů i vizualizačních layoutů. Má velmi podrobnou a kvalitní dokumentaci na svých webových stránkách.

Název	yFiles
Licence	komerční
Poslední verze	2. 4. 2014
Algoritmy	BFS, DFS, Bellman-Ford, Dijkstra, K-nejkratší cesta, Kruskal, Prim, hledání cyklů, shluků (cca 70, celý seznam na `http://docs.yworks.com/yfiles/doc/api/y/algo/package-summary.html`)
Vizualizace	kruhové, organické, stromové ortogonální, hierarchické, radiální uspořádání
Web	`http://www.yworks.com/`

Tabulka 3.9: yFiles

4 Podrobný rozbor knihoven

4.1 Knihovna JUNG

Knihovnu JUNG[6] jsem zvolil pro podrobnější rozbor hlavně kvůli jednoduchému používání a velké škále algoritmů a vizualizačních uspořádání, která knihovna podporuje.

JUNG se skládá celkem z 5 základních balíků jar (reprezentace, algoritmy, vstup/výstup ze souboru, vizualizace, 3D vizualizace). Dále však obsahuje cca dalších 10 podpůrných balíků jar. Dohromady mají celkem přes 6MB.

Vytvoření jednoduchého grafu v knihovně JUNG je poměrně snadné. Nejprve je nutné vybrat vhodnou třídu reprezentující graf. Jsou zde rozlišeny reprezentace pro husté a řídké grafy, pro singlegrafy a multigrafy a pro orientované či neorientované grafy. Například pro neorientovaný řídký multigraf je zde třída `UndirectedSparseMultigraph<V,E>` (všechny grafové třídy se nachází v balíku `edu.uci.ics.jung.graph` a dědí od abstraktní třídy `AbstractGraph<V,E>`). Všechny třídy grafů jsou generické, kde `V` značí datový typ vrcholů a `E` datový typ hran. V knihovně neexistuje žádný předdefinovaný datový typ pro vrcholy či hrany, lze tedy využít jakýkoliv datový typ. Díky tomu je možné ukládat k vrcholům i hranám vlastní data, ale u jednoduchých grafů bohatě stačí využití základních datových typů jako je `Integer` nebo `String`.

Nový vrchol se do grafu přidá metodou `addVertex()` požadující jediný argument a tím je instance zvoleného datového typu pro vrchol (`V`). Přidání hrany se provádí metodou `addEdge()`, kde je nutné zadat instanci datového typu hrany (`E`), a instance obou propojených vrcholů.

4.1.1 Vizualizace

Přizpůsobení vzhledu jednotlivých vrcholů a hran se prování za použití tzv. transformerů, což jsou abstraktní generické třídy `Transformer<I,O>` (`I` je datový typ hrany či vrcholu a `O` je výstupní datový typ, např.: `Color`, `Shape` nebo `Stroke`) obsahující metodu `transform()`, která v závislosti na daném vrcholu či hraně vrací příslušný tvar, barvu, čáru atd..

Následné vykreslení grafu je o něco složitější. Nejprve je nutné vytvořit instanci zvoleného uspořádání, kterým chceme graf vykreslit. Například `SpringLayout<V,E>` představuje organické uspořádání (všechna třídy uspořádání se nachází v balíku `edu.uci.ics.jung.algorithms.layout`). Jediným parametrem kontruktoru je instance grafu. Opět se jedná o generickou třídu potřebující definici typu vrcholů a hran. Dále je nutné vytvořit samotnou komponentu vykreslující graf. Ta je reprezenotvána třídou `BasicVisualizationServer<V,E>`. Paremetrem konstruktoru je instance uspořádání. Tato komponenta lze následně vložit do jakékoliv komponenty grafického prostředí `Swing`.

Další podporovaná uspořádání:

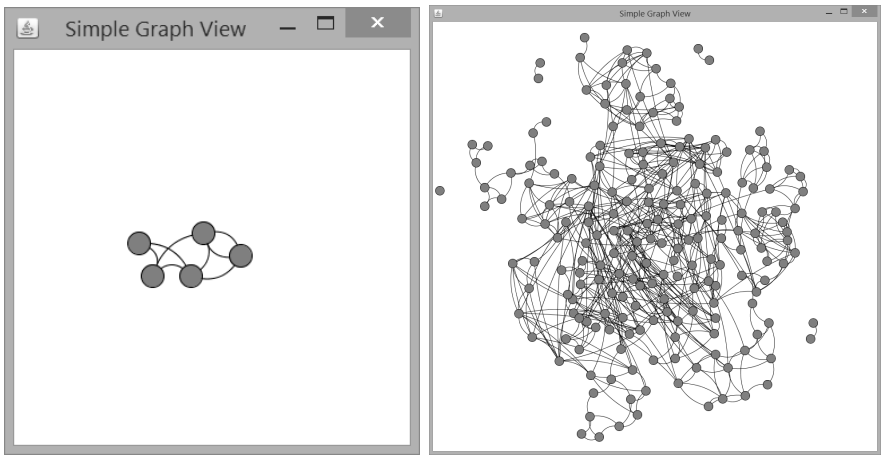
`CircleLayout<V,E>` – kruhové uspořádání

`StaticLayout<V,E>` – statické uspořádání

`TreeLayout<V,E>` – stromové uspořádání

`SpringLayout<V,E>` – organické uspořádání

Ukázky kódu jednoduché vizualizace pro knihovnu JUNG i GraphStream lze nalézt na přiloženém CD (podadresář `ukazky-kodu`).



Obrázek 4.1: Ukázka vizualizace knihovny JUNG, vlevo ručně vytvořený graf, vpravo náhodně vygenerovaný graf o 200 vrcholech; organické uspořádání

4.1.2 Práce se soubory

V základu umí JUNG importovat a exportovat grafy pouze z/do formátu GraphML[12]. Nicméně by neměl být problém imlementovat vlastní formáty.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
      http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="Tutorial 1" edgedefault="undirected">
    <node id="0">
    </node>
    <node id="1">
    </node>
    <edge id="0_1" source="0" target="1" directed="false">
    </edge>
  </graph>
</graphml>
```

Zdrojový kód 4.1: Ukázka grafu reprezentovaného formátem GraphML

4.2 Knihovna GraphStream

Knihovnu GraphStream[9] jsem pro podrobnější rozbor zvolil kvůli velkému množství podporovaných algoritmů a široké škále možností přizpůsobení vzhledu grafů, vrcholů a hran. Knihovna je také stále aktivně vyvíjena.

Knihovna je rozdělena na tři základní části – `core`, `algo`, `ui`.

`core` – obsahuje jádro knihovny, nezbytné pro jakékoliv použití

`algo` – obsahuje implementaci jednotlivých algoritmů

`ui` – obsahuje implementaci dodatečných grafických rozhraní

Jádro aplikace je velmi malé a kompaktní (necelých 900 kB) v jediném souboru *.jar. Výhodou je, že není nutné použít rozšiřující knihovny `ui` (přes 7 MB) a `algo` (přes 300 kB), a tím nemusí být výsledná aplikace

zbytečně velká. Samotná knihovna je velmi snadná na obsluhu základních funkcí a zároveň poskytuje širokou škálu možností jak funkcionalitu rozšířit. Vytvoření jednoduchého grafu je tedy otázkou několika řádek kódu, stejně jako jeho vykreslení, případně vygenerování (knihovna obsahuje generátory náhodných, mřížkových, eukleidovských a dalších grafů).

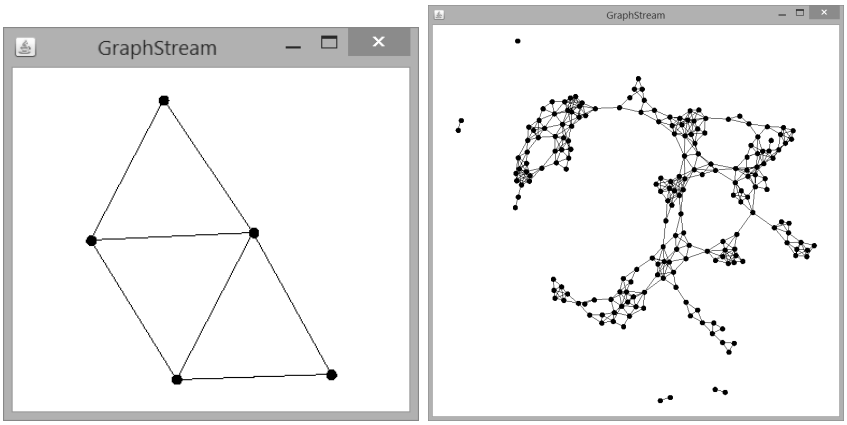
Základem každého grafu je třída implementující rozhraní `Graph`. Ta uchovává veškerá data o daném grafu. Základní dvě třídy implementující toto rozhraní jsou `SingleGraph` (umožňující pouze jednu hranu mezi dvěma vrcholy) a `MultiGraph` (podporuje více hran mezi dvěma vrcholy). Konstruktor požaduje jediný parametr a tím je unikátní identifikátor daného grafu (`string`).

Vrcholy se do grafu přidávají metodou `addNode()`, která opět požaduje pouze unikátní textový identifikátor vrcholu. Metoda vrátí vrchol s daným identifikátorem. Pokud vrchol s daným identifikátorem již existuje, metoda jej pouze vrátí a nevyhodí žádnou výjimku. Přidání hrany se provádí metodou `addEdge()`.

4.2.1 Vizualizace

Veškeré další vlastnosti (libovolná uživatelská data) jednotlivých vrcholů a hran se mění metodami `addAttribute()`, která umožňuje přidávat a měnit různé atributy jednotlivých položek. Používá se především pro změnu stylu daného objektu. Knihovna podporuje změny tvaru, velikosti (síle) i barvy jednotlivých vrcholů a hran. Tato metoda také umožňnuje uložit vlastní data (libovolného typu) k danému vrcholu či hraně.

Graf lze následně vykreslit metodou `display()`, která má nepovinný parametr určující, zda se má poloha jednotlivých vrcholů animovat a „odpuzovat“ od ostatních nesousedních vrcholů. Vrcholy lze samozřejmě rozmístit i manuálně určením absolutních souřadnic (atributy `x` a `y`). GraphStream podporuje i stylování grafů kaskádovými styly CSS.



Obrázek 4.2: Ukázka vizualizace knihovny GraphStream, vlevo ručně vytvořený graf, vpravo náhodně vygenerovaný graf o 200 vrcholech; organické uspořádání; použity stejné grafy jako u knihovny JUNG

4.2.2 Práce se soubory

Graphstream umožnuje import a export z/do formátů GraphML[12], DGS[13] (nativní formát GraphStream), DOT[14], GML[15] a SVG[16] (jen vizualizace).

4.3 Srovnání obou knihoven

Obě knihovny jsou velmi kvalitně naprogramované a robustní. Na svých webových stránkách mají kvalitní dokumentaci a JavaDoc i řadu příkladů.

Knihovna GraphStream je stále aktivně vyvíjena, kdežto vývoj knihovny JUNG skončil v roce 2010. I přesto je knihovna JUNG stále funkční. Nenarazil jsem ani u jedné na žádné chyby či neošetřené výjimky.

4.3.1 Vizualizace grafů

Obě knihovny umožňují poměrně kvalitní vizualizaci grafů. Zatímco GraphStream podporuje pouze organické a statické uspořádání. Knihovna JUNG podporuje mnohem širší škálu vizualizačních uspořádání. Na obrázcích 4.3 - 4.7 je srovnána vizualizace obou knihoven pro různé počty vrcholů a hran.

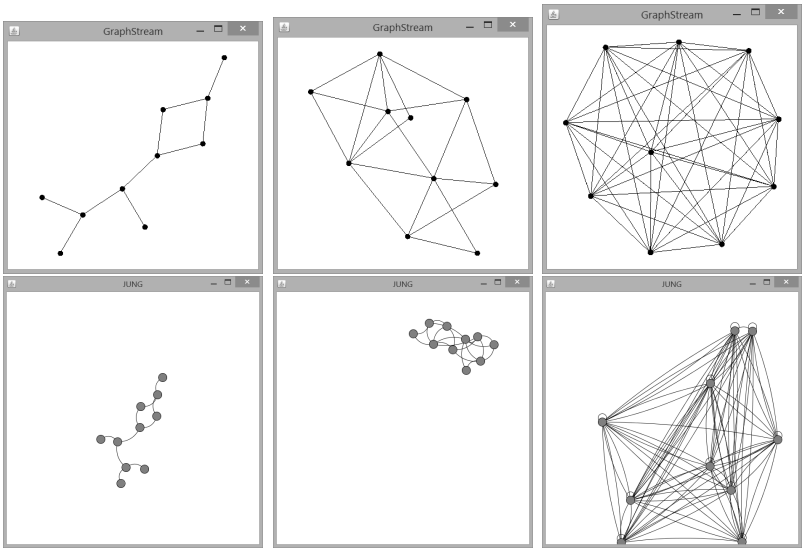
U knihovny GraphStream se nejprve vykreslí všechny vrcholy přes sebe a následně se postupně odpudivou silou přesunou na správné místo, kde se ustálí. Naproti tomu JUNG spočítá přibližnou polohu vrcholů už před zobrazením grafu a následně už se vrcholy příliš nepřemisťují. Nicméně v mnohých případech pohyb vrcholů vůbec neustane.

GraphStream není schopen v základním nastavení vykreslit vícenásobné hrany (více hran mezi dvěma vrcholy) a hrany v rámci jednoho vrcholu. JUNG je schopen vizualizovat oba případy.

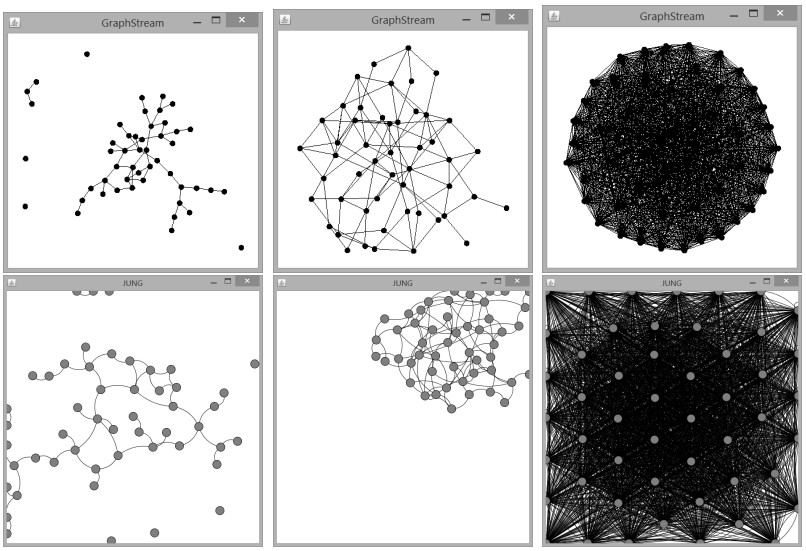
Bez dalšího programování nelze s vrcholy ve vizualizaci knihovny JUNG nijak manipulovat. U knihovny GraphStream lze vrcholy přesouvat myší.

U menších počtů vrcholů má knihovna JUNG problémy s umístěním grafů, snaží se je umisťovat ke kraji, takže ve výsledku je plátno prázdné a graf je umístěn u kraje. U větších počtů se pak vrcholy roztahují po celém plátně. GraphStream grafy vždy centruje a snaží se vrcholy uspořádat do tvaru kruhu.

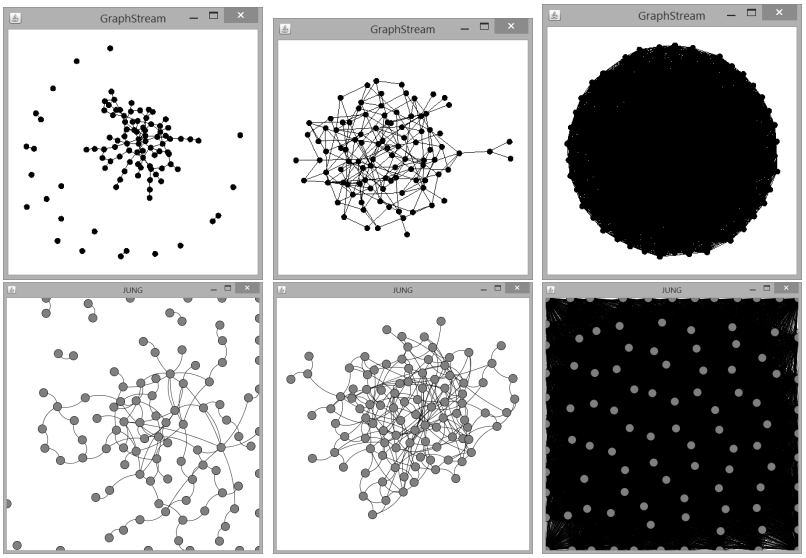
Všechny grafy jsou dostupné na přiloženém CD ve formátu GraphML (podsložka `grafy`; první číslo v názvu souboru je počet vrcholů, druhé je počet hran v daném grafu). Výsledky vizualizace se mohou lišit – obě knihovny vrcholy rozmisťují náhodně, tj. je-li stejný graf vykreslen stejnou knihovnou dvakrát, výsledné vizualizace mohou být rozdílné.



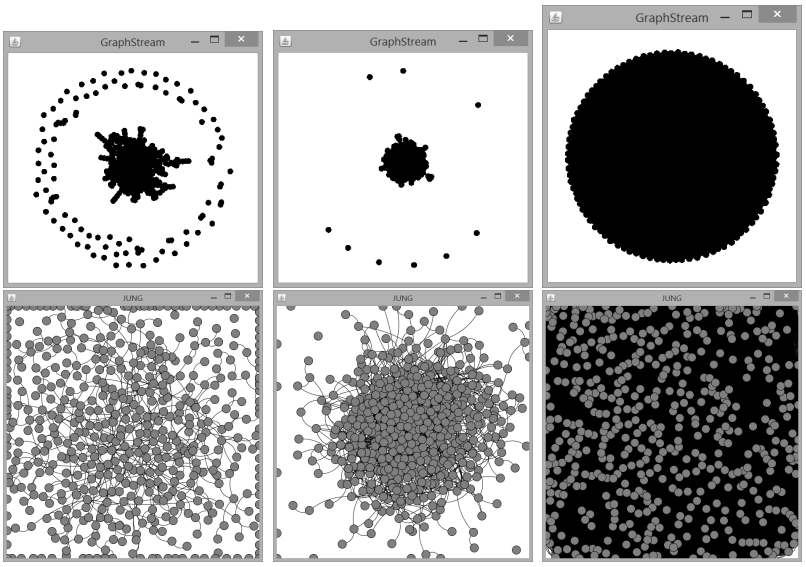
Obrázek 4.3: Náhodný graf o 10 vrcholech a 10, 20 a 95 hranách; organické uspořádání; nahoře GraphStream, dole JUNG



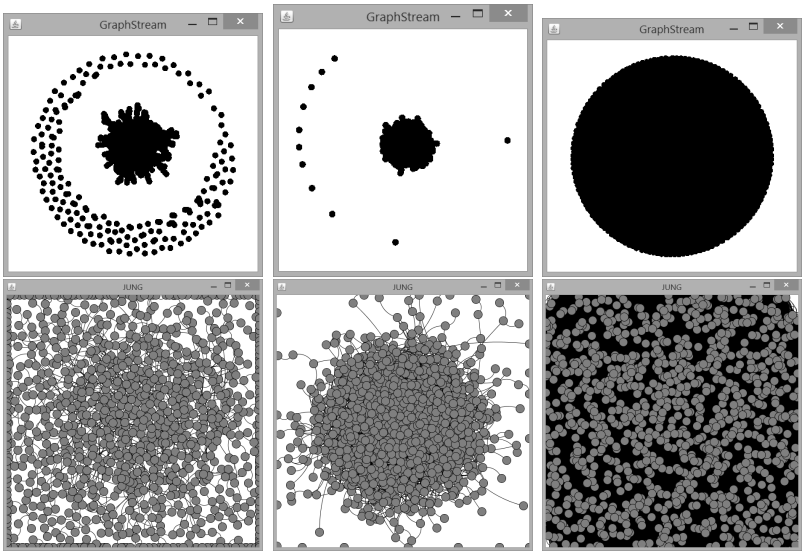
Obrázek 4.4: Náhodný graf o 50 vrcholech a 50, 100 a 2 495 hranách; organické uspořádání; nahoře GraphStream, dole JUNG



Obrázek 4.5: Náhodný graf o 100 vrcholech a 100, 200 a 9 995 hranách; organické uspořádání; nahoře GraphStream, dole JUNG



Obrázek 4.6: Náhodný graf o 500 vrcholech a 500, 1 000 a 249 995 hranách; organické uspořádání; nahoře GraphStream, dole JUNG



Obrázek 4.7: Náhodný graf o 1 000 vrcholech a 1 000, 2 000 a 999 995 hranách; organické uspořádání; nahoře GraphStream, dole JUNG

Rychlost vizualizace

U knihovny GraphStream je měřen čas od načtení grafu po ustálení vizualizovaných vrcholů. U knihovny JUNG je čas měřen od načtení grafu do zobrazení vizualizace, kromě několika výjimek, kdy se vrcholy po zobrazení ještě přesunuly o značný kus. V tomto případě je čas počítán do chvíle než se pohyb vrcholů subjektivně nezpomalí. Protože ve většině případů pohyb vrcholů u knihovny JUNG neskončil, mohou být tyto časy nepřesné.

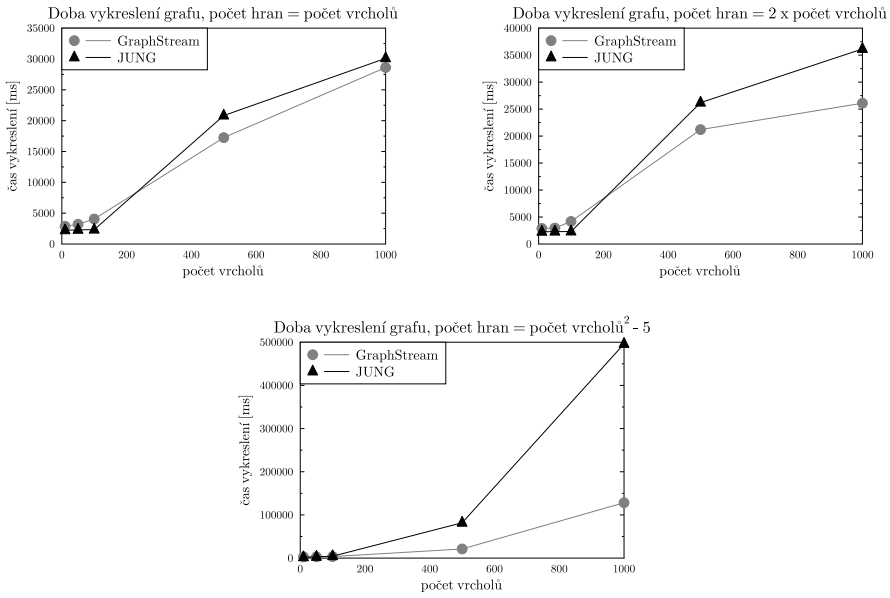
V	**10**	**50**	**100**	**500**	**1000**
$E = V$	2870 ms	3200 ms	4050 ms	17249 ms	28620 ms
$E = 2 \times V$	2882 ms	2955 ms	4185 ms	21199 ms	26094 ms
$E = V^2 - 5$	2882 ms	3001 ms	3527 ms	21140 ms	128245 ms

Tabulka 4.1: Rychlost vizualizace knihovnou GraphStream; V – počet vrcholů, E – počet hran

V	**10**	**50**	**100**	**500**	**1000**
$E = V$	2226 ms	2302 ms	2329 ms	20825[1] ms	30106[1] ms
$E = 2 \times V$	2272 ms	2293 ms	2315 ms	26193[1] ms	36144[1] ms
$E = V^2 - 5$	1871 ms	2853 ms	4410 ms	81800 ms	495995 ms

Tabulka 4.2: Rychlost vizualizace knihovnou JUNG; V – počet vrcholů, E – počet hran

[1] subjektivní čas snížení rychlosti pohybu vrcholů



Obrázek 4.8: Znázornění vizualizačních časů pro různé počty hran

Z výše uvedených grafů a tabulek je vidět, že u menších grafů dosahuje lepších výsledků knihovna JUNG, ale u těch větších dosahuje větší rychlosti vykreslení knihovna GraphStream.

4.3.2 Základní algoritmy

GraphStream i JUNG podporují širokou škálu algoritmů ať už pro hledání nejkratších cest či hledání koster grafů.

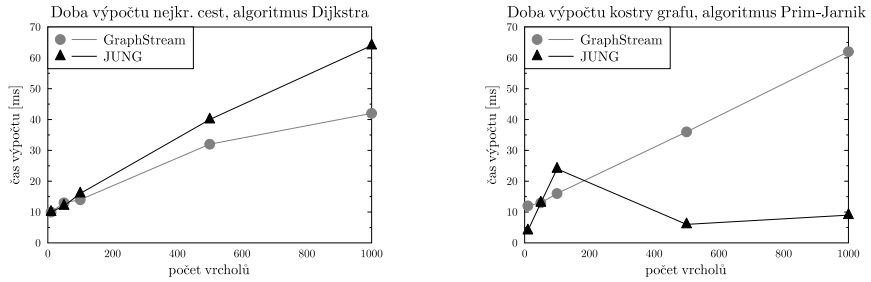
Níže je srovnání obou knihoven pro algoritmy Dijkstra a Prim-Jarnik.

V	**10**	**50**	**100**	**500**	**1000**
GraphStream	10 ms	13 ms	14 ms	32 ms	42 ms
JUNG	10 ms	12 ms	16 ms	40 ms	64 ms

Tabulka 4.3: Rychlost nalezení nejkratších vzdáleností algoritmem Dijkstra; V – počet vrcholů, počet hran = 2V

V	**10**	**50**	**100**	**500**	**1000**
GraphStream	12 ms	13 ms	16 ms	36 ms	62 ms
JUNG	4 ms	13 ms	24 ms	6 ms	9 ms

Tabulka 4.4: Rychlost nalezení minimální kostry grafu algoritmem Prim-Jarnik; V – počet vrcholů, počet hran = 2V



Obrázek 4.9: Znázornění časů výpočtů pro algoritmy Dijkstra (vlevo) a Prim-Jarnik (vpravo)

Z grafů je zřejmé, že u algoritmu Dijkstra dosahuje lepších výsledků knihovna GraphStream. U algoritmu Prim-Jarnik dosahuje ve většině případů rapidně lepších výsledků knihovna JUNG. V některých ovšem naopak zaostává za GraphStream.

4.3.3 Práce se soubory

Obě knihovny podporují import a export ve formátu GraphML. Proto následující srovnání porovnává načtení a uložení grafů právě v tomto formátu.

Pro testování byly použity stejné grafy jako u vizualizace.

GraphStream navíc podporuje načítání i ukládání v řadě dalších formátů jako např.: GraphML, DGS, DOT, GML a SVG.

V	**10**	**50**	**100**	**500**	**1000**
$E = V$	60 ms	73 ms	95 ms	234 ms	361 ms
$E = 2 \times V$	70 ms	86 ms	119 ms	313 ms	479 ms
$E = V^2 - 5$	86 ms	422 ms	737 ms	2778 ms	10278 ms

Tabulka 4.5: Rychlost načtení grafu knihovnou GraphStream; V – počet vrcholů, E – počet hran

V	**10**	**50**	**100**	**500**	**1000**
$E = V$	58 ms	63 ms	76 ms	133 ms	190 ms
$E = 2 \times V$	57 ms	69 ms	82 ms	162 ms	239 ms
$E = V^2 - 5$	68 ms	231 ms	381 ms	1776 ms	5952 ms

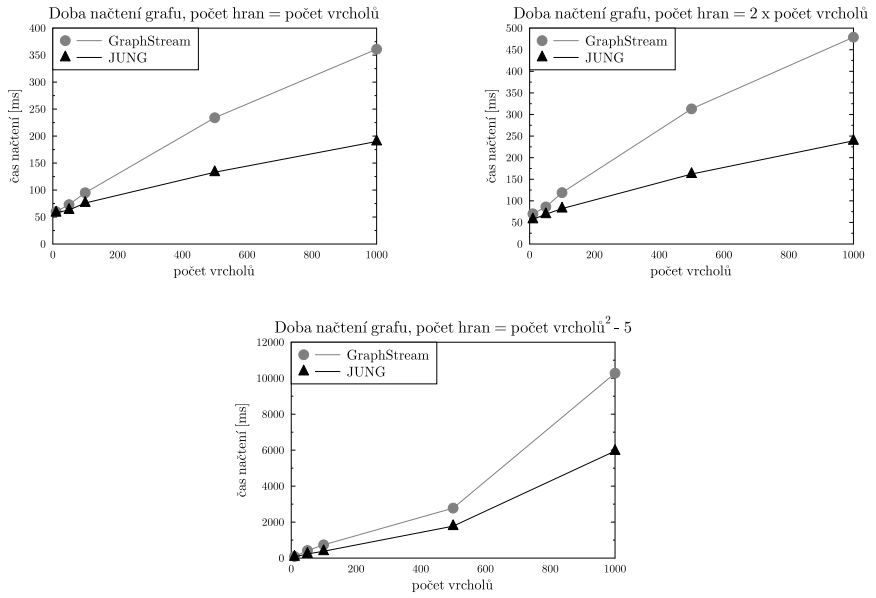
Tabulka 4.6: Rychlost načtení grafu knihovnou JUNG; V – počet vrcholů, E – počet hran

V	**10**	**50**	**100**	**500**	**1000**
$E = V$	13 ms	21 ms	32 ms	93 ms	224 ms
$E = 2 \times V$	20 ms	28 ms	43 ms	150 ms	297 ms
$E = V^2 - 5$	24 ms	324 ms	339 ms	2026 ms	5166 ms

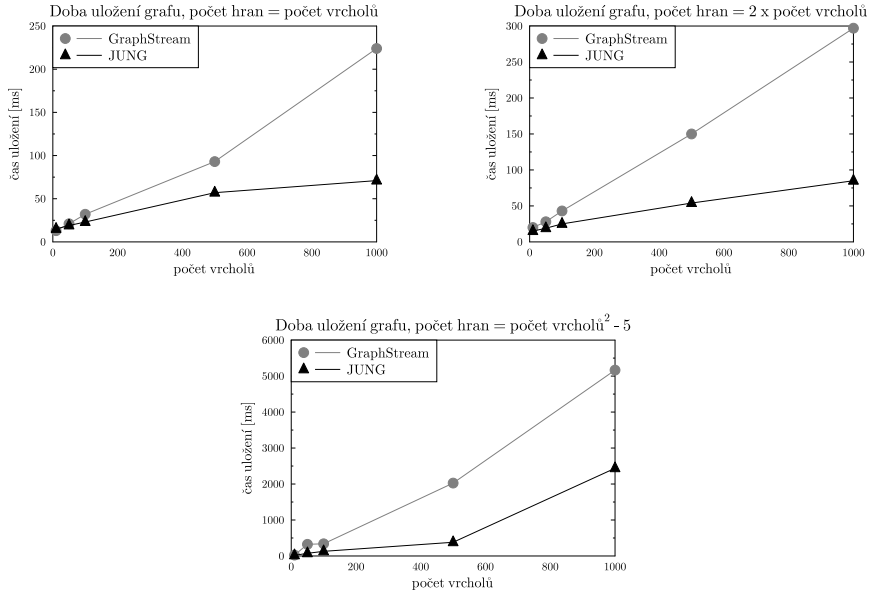
Tabulka 4.7: Rychlost uložení grafu knihovnou GraphStream; V – počet vrcholů, E – počet hran

V	**10**	**50**	**100**	**500**	**1000**
$E = V$	15 ms	19 ms	23 ms	57 ms	71 ms
$E = 2 \times V$	15 ms	19 ms	25 ms	54 ms	85 ms
$E = V^2 - 5$	17 ms	74 ms	129 ms	381 ms	2438 ms

Tabulka 4.8: Rychlost uložení grafu knihovnou JUNG; V – počet vrcholů, E – počet hran



Obrázek 4.10: Znázornění časů načtení grafů pro různé počty hran



Obrázek 4.11: Znázornění časů uložení grafů pro různé počty hran

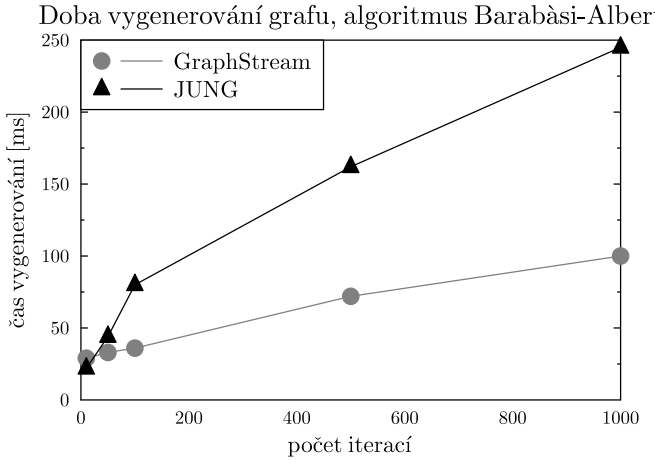
Z výše uvedených grafů a tabulek je zřejmé, že obě knihovny provádí mnohem rychleji ukládání než načítání. Téměř ve všech případech pak knihovna JUNG dosáhla výrazně lepších výsledků než GraphStream.

4.3.4 Generování grafů

Obě knihovny podporují generování grafů. GraphStream má implementováno 13 různých algoritmů pro vygenerování grafů[17]. JUNG jich podporuje jen 5 (Barabàsi-Albert, Eppstein-Powel-Law, Erdos-Renyi, Kleinberg Small World generator, Mixed random graph generator). Jediný společný algoritmus je Barabàsi-Albert[18], který je použit pro srovnání rychlosti vygenerování grafů pro obě knihovny. Všechny vygenerované grafy jsou ve formátu GraphML uloženy v podsložce `vygenerovane` na přiloženém disku CD.

Iterací	**10**	**50**	**100**	**500**	**1000**
GraphStream	29 ms	33 ms	36 ms	72 ms	100 ms
JUNG	22 ms	44 ms	80 ms	162 ms	245 ms

Tabulka 4.9: Rychlost vygenerování grafu; algoritmus Barabàsi-Albert



Obrázek 4.12: Znázornění časů generování grafů pro různé počty iterací

Zde lepších výsledků dosáhla knihovna GraphStream, ve které je i samotné vygenerování snazší na naprogramování.

5 Návrh knihovny pro sestavení precedenčního grafu

Samotná knihovna se bude skládat ze tří částí. První část bude použita pro načtení množiny akcí ze souboru, druhá část na vytvoření grafu a třetí na prezentaci výsledků uživateli.

Mou hlavní snahou bude, aby knihovna byla co nejvíce univerzální a aby mohla být použita velmi rychle, bez zbytečného programování a aby si ji mohl pokročilý uživatel snadno přizpůsobit dle svého.

5.1 Načtení akcí ze souboru

Základní verze knihovny bude podporovat pouze načítání dat ze souboru typu XML[19]. Nicméně nebude nijak specifikován sled ani názvy jednotlivých elementů XML souboru.

Uživatel bude moci nadefinovat které elementy reprezentují akci, její data, prerekvizity či následníky.

Knihovna bude obsahovat také rozhraní, jehož implementací bude uživatel schopen přidat podporu pro libovolný formát souborů.

5.2 Vytvoření výsledného grafu

Hlavní funkcionalita celé knihovny se bude nacházet v této části. Konkrétně jde o algoritmus, který sestaví ze všech akcí výsledný graf.

5.2.1 Definice Akce

Akce bude jednoznačně identifikována unikátním identifikátorem ID, dále může obsahovat libovolná data.

Dále každá akce může obsahovat seznam ID prerekvizit, těsných prerekvizit, následníků a těsných následníků. Těsné prerekvizity jsou akce, které musí nastat bezprostředně před danou akcí, těsní následníci jsou akce, jež musí nastat bezprostředně po dané akci. Prerekvizity mohou nastat kdykoliv před danou akcí a následníci kdykoliv po dané akci.

5.2.2 Algoritmus

Protože jsem pro řešení daného problém nenalezl žádný vhodný algoritmus, rozhodl jsem se vytvořit vlastní.

Nejprve je nutné zjistit, zda některé akce nemají reference (prerekvizity, následníci), které se odkazují na neexistující akce. Dále je možné spojit akce obsahující těsné prerekvizity a těsné následníky.

V přípravě algoritmu se převedou všichni následníci a těsní následníci na prerekvizity a těsné prerekvizity. Např: A má následníka B. => B má prerekvizitu A.

Samotný algoritmus se skládá ze dvou fází:

- **seřazení akcí** – v této fázi bude všem akcím přiřazena číselná hodnota „čas“. Ta udává v jakém kroce může být akce provedena. Akce s časem 0 budou provedeny jako první, následovat budou akce s časem 1, 2 atd... Více akcí může mít stejný čas, pak mohou být provedeny zároveň. V této fázi je možné detekovat neexistující řešení (A je prerekvizita B, B je prerekvizita A).
- **spojení akcí** – Od nultého času budou postupně propojovány akce tak, aby vždy byly splněny všechny prerekvizity.

Algoritmus je znázorněn pseudokódem níže.

```
Algorithm vytvorGraf(A):
Input: seznam akci A
Output graf akci

Graf G := novy graf
pridejVrcholy(IDAkci(A))

if (not overReferenceAkci()) then Chyba(Neplatne reference)

prevedNasledníkyNaPrerekvizity(A)

integer cas := 0
Seznam akci S := novy seznam(A)
Asociativni pole<integer, seznam Akci> P := nove asociativni
    pole
boolean zmena

while(S not empty) do
  zmena = false
  H : for(X : vsechny akce v S) do
    if (cas = 0) then
      if (X.prerekvizity is empty and X.tesnePrerekvizity is
    empty) then
        P.set(cas, PridejDoSeznamu(X))
        S.remove(X)
        zmena := true
      end
    end else
      for (Y : X.prerekvizity a X.tesnePrerekvizity) do
        if (P.getKey(Y) = null or P.getKey(Y) >= cas) goto(H)
      end
      P.set(cas, PridejDoSeznamu(X))
      S.remove(X)
      zmena := true
    end
  end
  cas++
  if (not zmena) then Chyba(Neexistuje reseni)
end

for (X : vsechny akce v P podle casu; cas > 0) do
  PridejHrany(G, z Vsechny Akce s nizsimi casy zminene v X.
    prerekvizity (postupne od nejblizsich nizsich casu), do X)
end

return G
```

Zdrojový kód 5.1: Pseudokód algoritmu

Tento algortimus sice dokáže zjistit, že dané zadání nemá řešení, ale není schopen odhalit problematické vazby a případně je napravit. Tento problém jsem vyřešil mírně upraveným Tarjanovým algoritmem[20].

Tarjanův algoritmus používá upravené prohledávání grafu do hloubky k nalezení silných komponent v orientovaném grafu neboli k nalezení smyček. Smyčka v orientovaném grafu znamená, že se lze dostat z jednoho vrcholu do toho samého přes jednu či více hran. Graf bez smyček je strom.

Nejprve si vytvořím ze seznamu Akcí graf tak, že jednotlivé Akce představují vrcholy a jejich vazby jsou hranami v grafu. Např.: A je prerekvizitou B bude v grafu jako hrana A -> B; A je následníkem B bude v grafu jako B -> A. U každé hrany si uložím o jakou vazbu (prerekvizitam, následník...) se jedná. Samotný algoritmus je pak znázorněn pseudokódem níže:

```
1  List: findCycles(graf G)
     Set navstiveneVrcholy = new Set
3    Set uzavreneVrcholy = new Set
     List problematickeVazby = new List
5
     for (X : vsechny vrcholy v G) do
7      findCycles1(G, null, X, navstiveneVrcholy, uzavreneVrcholy,
       problematickeVazby)
     end
9    return problematickeVazby
   end
11
   findCycles1(graf G, hrana H, vrchol V, Set navstiveneVrcholy,
       Set uzavreneVrcholy, List problematickeVazby)
13   if (not (H = null) and navstiveneVrcholy.contains(V)) then
       if (uzavreneVrcholy.contains(V)) then return
15     problematickeVazby.add(H)
       return
17   end

19   navstiveneVrcholy.add(V)

21   for (X : vsechny sousedni vrcholy V) do
       findCycles1(G, hrana(V, X), X, navstiveneVrcholy,
       uzavreneVrcholy, problematickeVazby)
23   end

25   uzavreneVrcholy.add(V)
   end
```

Zdrojový kód 5.2: Pseudokód upraveného Tarjanova algoritmu

5.3 Prezentace výsledku uživateli

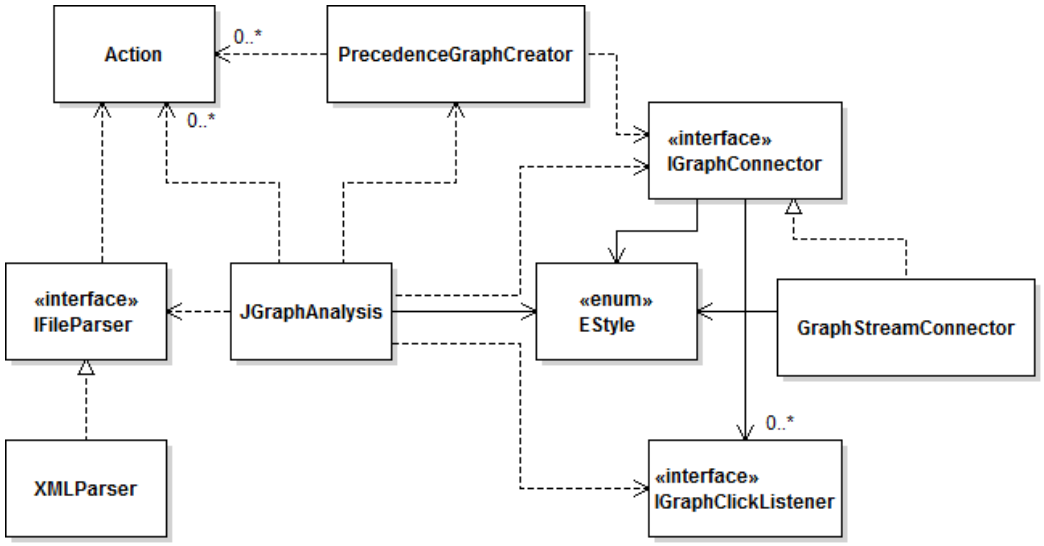
K reprezentaci grafu a jeho vizualizaci bude použita již existující knihovna s otevřenou licencí.

Z výše uvedeného srovnání knihoven jsem zvolil knihovnu GraphStream, protože podporuje vše potřebné. Je stále ve vývoji a je jednoduchá na obsluhu. Výsledná knihovna nicméně bude umožňovat uživateli použití i jiné knihovny s minimálním dopisováním zdrojového kódu.

Uživatel bude moct zvolit jednotlivé akce a zobrazit jejich podrobnosti (prerekvizity, následníky, data). Propojení jednotlivých akcí bude znázorněno vizualizací grafu (výstup knihovny GraphStream).

6 Implementace knihovny

Jádro celé knihovny je poměrně jednoduché a obsahuje pouze několik tříd (viz UML diagram na obr. 6.1).



Obrázek 6.1: UML diagram jádra knihovny

Většina těchto tříd celé knihovny je generických. `I` představuje datový typ identifikátoru akce a `D` je datový typ dat akce.

Hlavní třídou celé knihovny je třída `JGraphAnalysis<I, D>` v balíku `cz.dawon.java.library`. Tato knihovna spojuje všechny části knihovny dohromady.

Ve stejném balíku se nachází také knihovna `PrecedenceGraphCreator<I, D>`, která obsahuje samotný algoritmus na propojení akcí.

Třída `Action<I, D>` představuje jednu třídu spolu s jejím identifikátorem, daty a jednotlivými vazbami na ostatní třídy.

Dále je zde balík `cz.dawon.java.library.parsers` obsahující třídy potřebné pro zpracování souboru.

Nejdůležitější je zde rozhraní `IFileParser<I, D>` definující jedinou metodu `parse()`. Implementací tohoto rozhraní může uživatel zpracovávat sou-

bory s Akcemi ve vlastním formátu.

Dále je zde třída `XMLParser`, která je implementací rozhraní `IFileParser`. Tato třída je schopná načítat soubory ve formátu XML. V této třídě je ještě definována podtřída `NodeSelector` umožňující přesnou specifikaci jednotlivých elementů či atributů v XML souboru.

Další balík je `cz.dawon.java.library.graphConnectors`. Zde je zásadní rozhraní `IGraphConnector<I>`. Jeho implementací může uživatel použít libovolnou knihovnu pro práci s grafy, která umí základní operace jako je přidání a odebrání vrcholů a hran a uložení dat k nim, získání Swing komponenty obsahující vykreslený graf atd. Pokud dokáže knihovna i reagovat na kliknutí do vykresleného grafu, je zde ještě rozhraní `IGraphClickListener<I>`. Jedná se o listener poslouchající události kliknutí na určitý vrchol. Výčet `EStyle` obsahuje různé styly pro vykreslené vrcholy či hrany, takže lze změnit tvar, velikost či barvu vybrané Akce, jejích prerekvizit, následníků atd.

Implementací rozhraní `IGraphConnector<I>` pro knihovnu GraphStream je třída `GraphStreamConnector`. Toto je jediné místo, kde je knihovna GraphStream propojena s hlavní knihovnou.

Výše je popsáno celé jádro knihovny, které je schopno samostatně fungovat. Třídy `XMLParser` a `GraphStreamConnector<I>` v něm nejsou nutné, uživatel si může implementovat vlastní třídy.

Další částí knihovny je průvodce. Ten k práci potřebuje třídu `XMLParser`. Samotný průvodce se nachází v balíku `cz.dawon.java.gui.parserSetup`. Zde je hlavní přístupový bod k průvodci třída `JGraphAnalysisSetup`. Zavoláním metody `showDialog()` se zobrazí samotný průvodce jehož dialog je reprezentován třídou `MainWindow`. Také zde lze přidat listener `IJGraphAnalysisSetupListener` metodou `addJGraphAnalysisSetupListener()`, který čeká na dokončení průvodce. Po dokončení vrátí tento posluchač uživateli instanci třídy `JGraphAnalysis<String, String>`.

Balík `cz.dawon.java.gui.parserSetup.cards` obsahuje jednotlivé obrazovky průvodce. Zde je hlavní rozhraní `ICard` definující všechny metody potřebné k zobrazení a obsluze jedné obrazovky.

Nakonec balík `cz.dawon.java` obsahuje ukázkovou aplikaci s hlavním třídou `Main`.

6.1 Použití knihovny

Nejjednodušší možnost využití knihovny je ukázkovou aplikací. Tu stačí pouze spustit. Pokud chce uživatel zakomponovat knihovnu do vlastní aplikace, stačí vložit následující úsek kódu:

```
JGraphAnalysis<String, String> jga = new JGraphAnalysis<String,
    String>();

XMLParser parser = new XMLParser();
parser.setActionSelector(new NodeSelector("action", null, "
    actions"));
parser.setActionIdSelector(new NodeSelector(null, "id", null));
parser.setActionDataSelector(new NodeSelector("data", null, null
    ));
// ... zde by nasledovaly NodeSelectory pro prerekvizity,
    nasledniky atd.
jga.setParser(parser); // misto XMLParser lze vyuzit libovolnou
    tridu implementujici IFileParser

GraphStreamConnector graph = new GraphStreamConnector();
graph.createGraph("Test");
jga.setGraphConnector(graph); // misto GraphStreamConnector lze
    vyuzit libovolnou tridu implementujici IGraphConnector

JPanel vyslednyGraf = new JPanel();
vyslednyGraf.add(graph.getComponent()); // komponenta do niz
    bude vykreslen graf

try {
  jga.parse("c:/cesta/k/xml/souboru.xml"); // nacteni souboru
} catch (IOException e) {
  // chyba pri cteni souboru
} catch (ParseException e) {
  // chyba pri parsovani souboru
}
jga.addVertices(); // pridani vrcholu do grafu
try {
  jga.process(); // samotne zpracovani souboru
} catch (InvalidAlgorithmParameterException e) {
  // neexistujici reseni
} catch (NoSuchElementException e) {
  // neplatna vazba
}
```

Zdrojový kód 6.1: Základní použití knihovny

Dále lze využít průvodce, kdy se kód ještě o něco zkrátí:

```
private void start() {
  JGraphAnalysisSetup setup = new JGraphAnalysisSetup(null);
  setup.addJGraphAnalysisSetupListener(new
    IJGraphAnalysisSetupListener() {
     @Override
     public void setupDone(JGraphAnalysis<String, String> jga) {
       onDone(jga); // pri dokonceni pruvodce
     }
  });
  setup.showDialog(); // zobrazeni pruvodce
}

private void onDone(JGraphAnalysis<String, String> jga) {
  GraphStreamConnector graph = new GraphStreamConnector();
  graph.createGraph("Test");
  jga.setGraphConnector(graph); // misto GraphStreamConnector
    lze vyuzit libovolnou tridu implementujici IGraphConnector

  JPanel vysledny Graf = new JPanel();
  vysledny Graf.add(graph.getComponent()); // komponenta do niz
    bude vykreslen graf
  // nacteni souboru a nastaveni parseru provede pruvodce
  jga.addVertices(); // pridani vrcholu do grafu
  try {
    jga.process(); // samotne zpracovani souboru
  } catch (InvalidAlgorithmParameterException e) {
    // neexistujici reseni
  } catch (NoSuchElementException e) {
    // neplatna vazba
  }
}
```

Zdrojový kód 6.2: Základní použití knihovny s průvodcem

V případě, že uživatel potřebuje přesně detekovat a opravit cykly v zadání, stačí do kódu přidat následující řádky:

```
//... nastaveni parseru a nacteni Akci
jga.addVertices(); // pridani vrcholu
PrecedenceGraphCreator<String, String> pgc = jga.init();
// zde se musi jednat o jinou instanci GraphStreamConnector,
    nejedna se o vysledny graf!
GraphStreamConnector graph1 = new GraphStreamConnector();
graph1.createGraph("Temp");

try {
  // samotne vyhledani cyklu; pokud je druhy parametr true, bude
     provedena i oprava
  List<ReferenceIdentificator<String>> badReferencies = pgc.
    findAndFixCycles(graph1, true);
  // badReferencies nyni obsahuje identifikatory vsech vadnych
    vazeb
  jga.process(pgc); // samotne zpracovani souboru, zde je nutny
    parametr
} catch (InvalidAlgorithmParameterException e) {
  // neexistujici reseni, pokud je vyse nastavena i oprava
    chybnych vazeb, k teto vyjimce by dojit nemelo
} catch (NoSuchElementException e) {
  // neplatna vazba
}
```

Zdrojový kód 6.3: Nalezení problematických vazeb a jejichoprava

Všechny třídy a metody celé knihovny jsou okomentovány JavaDocem. Celá knihovna včetně zdrojového kódu se nachází na přiloženém CD ve složce `knihovna`. Ukázkové soubory XML se nachází ve složce `knihovna/xml`.

6.2 Testování knihovny

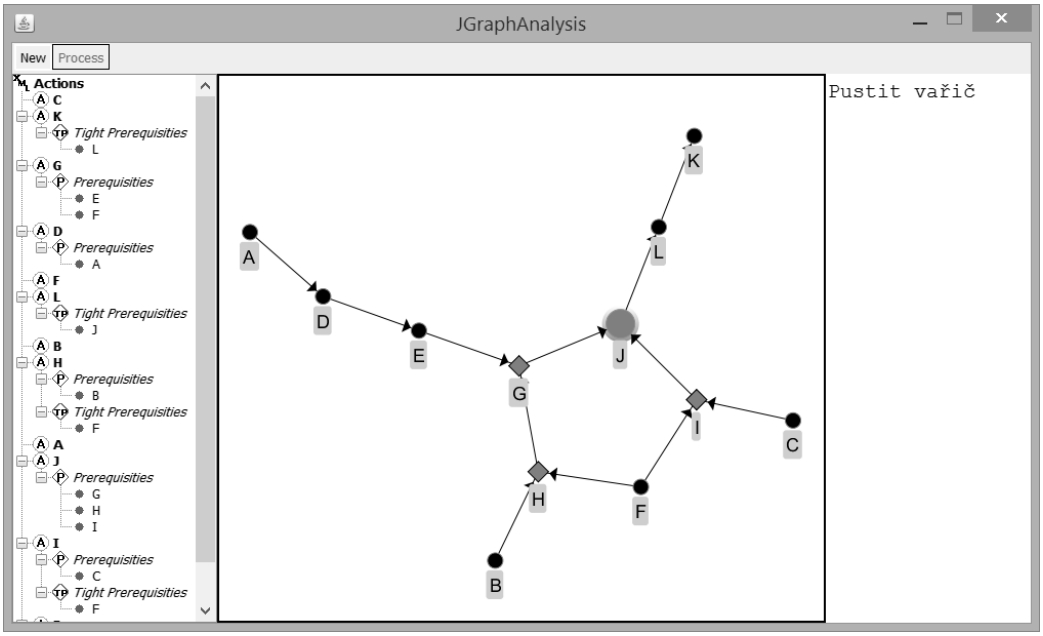
Knihovnu jsem testoval na několika souborech jak s validními vstupy, tak se vstupy obsahujícími neplatná data. Všechny testované soubory se nachází ve složce `knihovna/xml`. Soubory s příponou `*.xml` jsou zdrojová data a soubory s příponou `*.gas` jsou exportovaná nastavení ukázkového programu knhovny.

Popis jednotlivých souborů:

- **valid1_1.xml; valid1_2.xml; valid2_1.xml; potatoes.xml** – validní soubory, které by program měl zpracovat bez problému
- **with_cycles1_2.xml; with_cycles2_2.xml** – soubory obsahující jednu či více smyček. Program by měl zobrazit varování, že soubory obsahují smyčky a měl by smazat vazby způsobující smyčky.
- **invalid_referencies_2.xml** – soubor obsahující jednu či více vazeb na neexistující Akce. Program by měl zobrazit chybové hlášení, že se nepodařilo dané Akce najít.

Čísla za podtržítky souborů označují názvy souborů `*.gas`, které lze použít pro import nastavení pro daný soubor.

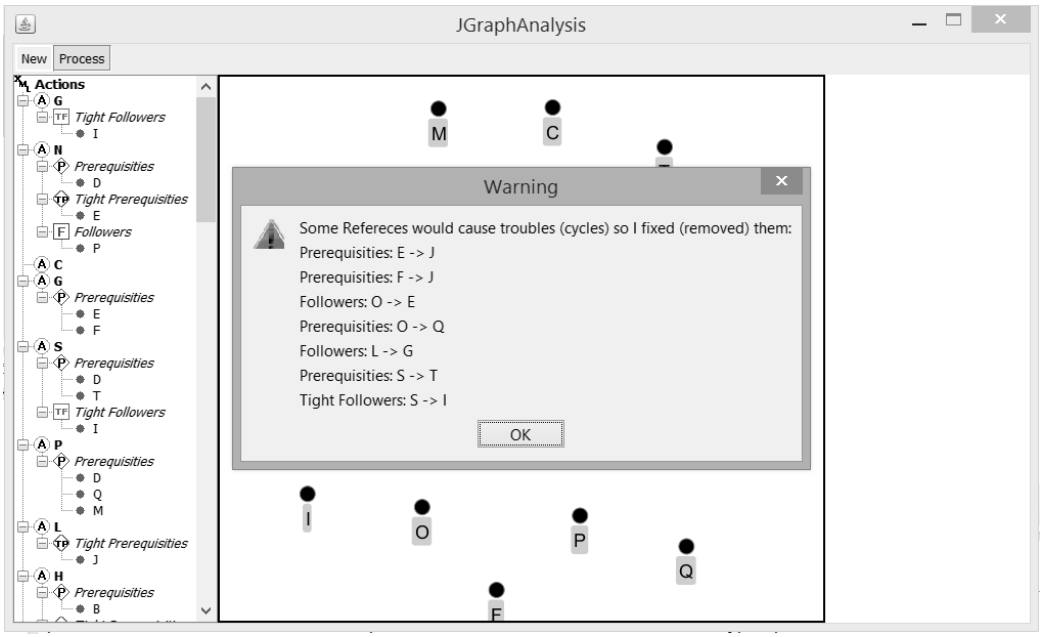
Soubor `potatoes.xml` obsahuje sled akcí při obyčejné přípravě brambor. U každé Akce je v elementu data uložen popis daného úkonu (koupit brambory, pustit vařič, atd.).



Obrázek 6.2: Testování aplikace – příprava brambor

Na obrázku 6.2 lze vidět, že Akce `A`, `B`, `C` a `F` jsou úkony, které mohou být provedeny zároveň (nákup surovin a napuštění vody do hrnce). Akce `K` je pak výstupní (slití vody), kdy už jsou brambory připraveny.

Naopak soubor `with_cycles2_2.xml` obsahuje neplatná data, proto nemůže být zpracován tak, jak je.



Obrázek 6.3: Testování aplikace – cyklické vazby

Po načtení souboru a kliknutí na tlačítko `Process` se zobrazí upozornění, že některé vazby nemohly být zpracovány a byly odebrány (viz obr. 6.3). Následně však bude soubor zpracován a nalevo v okně budou problematické vazby označeny červeně.

Při načtení souboru `invalid_referencies_2.xml` se při pokusu o zpracování zobrazí hláška o neexistující Akci s daným ID a soubor nebude zpracován.

7 Závěr

V první části této práce jsem provedl zběžné srovnání více různých knihoven pro práci s grafy a následně důkladné srovnání dvou lepších knihoven (JUNG a GraphStream) včetně porovnání jejich výkonu při načítání, vizualizaci, zpracování a generování grafů.

Výsledná knihovna JGraphAnalysis je schopna vytvořit graf ze seznamu zadaných Akcí, jejich následníků a prerekvizit. Její součástí je průvodce a ukázková aplikace demonstrující veškerou funkčnost knihovny. Knihovna je schopna odhalit i neexistenci řešení a případně vazby způsobující neexistenci řešení.

Mou hlavní snahou bylo, aby měla knihovna co nejuniverzálnější použití a zároveň aby byla co nejjednodušší na obsluhu. Výsledná knihovna by se pak dala přirovnat ke skládačce, kdy uživatel může využít jen ty části knihovny, které potřebuje. Pokud využije jen jádro knihovny, může zpracovávat soubory libovolného formátu a pro vizualizaci využít libovolnou knihovnu. Pokud využije navíc průvodce, stačí jen pár řádek kódu a je možno knihovnu zakomponovat do výsledné aplikace. Pokud využije ukázkovou aplikaci, není nutné nic programovat.

Knihovna by mohla být rozšířena dále tak, aby podporovala další parametry plánování (například přidání doby trvání jednotlivých Akcí – pak by knihovna mohla zjišťovat kolik času by zabralo splnění všech Akcí).

Knihovna včetně ukázkové aplikace bude uvolněna k volnému použití pod licencí GNU Lesser General Public License[21] a zveřejněna na internetu.

Literatura

[1] BONDY, John Adrian a Uppaluri Siva Ramachandra MURTY. *Graph theory with applications*. 5. vyd. New York: Elsevier Science Publishing Co. Inc., 1982, 264 s. ISBN 04-441-9451-7.

[2] ČERNÝ, Jakub. KAM, MFF UK. *Základní grafové algoritmy* [online]. 2010, 24.11.2010 [cit. 2014-06-12]. Dostupné z: `http://kam.mff.cuni.cz/~kuba/ka/ka.pdf`

[3] NAVEH, Barak. *JGraphT* [online]. 2011 [cit. 2014-06-21]. Dostupné z: `http://www.jgrapht.org`

[4] Jgraph/jgraphx. *GitHub* [online]. 2014 [cit. 2014-06-21]. Dostupné z: `https://github.com/jgraph/jgraphx`

[5] ZEIGERMANN, Oliver a Henrik HEINE. *JSL: Java Search Library* [online]. 2004 [cit. 2014-06-21]. Dostupné z: `http://jsl.sourceforge.net`

[6] O'MADADHAIN, Joshua, Danyel FISHER a Tom NELSON. *JUNG: Java Universal Network/Graph Framework* [online]. 2010 [cit. 2014-06-21]. Dostupné z: `http://jung.sourceforge.net`

[7] HOGIE, Luc. *Grph: The high performance graph library for Java* [online]. 2014 [cit. 2014-06-22]. Dostupné z: `http://www.i3s.unice.fr/~hogie/grph`

[8] JIM.ANDREOU. *Flexigraph: A Java graph algorithms library* [online]. 2010 [cit. 2014-06-22]. Dostupné z: `https://code.google.com/p/flexigraph`

[9] BALEV, Stefan, Julien BAUDRY, Antoine DUTOT, Yoann PIGNÉ a Guilhelm SAVIN. *GraphStream: A Dynamic Graph Library* [online]. 2014 [cit. 2014-06-22]. Dostupné z: `http://graphstream-project.org`

[10] *The Data Structures Library in Java* [online]. 2000 [cit. 2014-06-22]. Dostupné z: http://cs.brown.edu/cgc/jdsl/home.html

[11] YFiles for Java. YWORKS. *YWorks: the diagramming comapny* [online]. 2014 [cit. 2014-06-22]. Dostupné z: http://www.yworks.com/en/products_yfiles_about.html

[12] Home. *The GraphML File Format* [online]. 2013 [cit. 2014-06-05]. Dostupné z: http://graphml.graphdrawing.org/

[13] The DGS File Format Specification. *GraphStream* [online]. 2013 [cit. 2014-06-05]. Dostupné z: http://graphstream-project.org/doc/Advanced-Concepts/The-DGS-File-Format/

[14] The DOT Language. *Graphviz - Graph Visualization Software* [online]. 2014 [cit. 2014-06-05]. Dostupné z: http://www.graphviz.org/content/dot-language

[15] Projects. *Universität Passau* [online]. 2010 [cit. 2014-06-05]. Dostupné z: http://www.fim.uni-passau.de/en/fim/faculty/chairs/theoretische-informatik/projects.html

[16] DAHLSTRÖM, Erik, Patrick DENGLER, Anthony GRASSO, Chris LILLEY, Cameron MCCORMACK, Doug SCHEPERS, Jonathan WATT, Jon FERRAIOLO a Dean JACKSON. Scalable Vector Graphics (SVG). *W3C* [online]. 5. vyd. 2008, 7.2.2013 [cit. 2014-06-20]. Dostupné z: http://www.w3.org/TR/SVG

[17] Overview of generators. *GraphStream* [online]. 2013 [cit. 2014-06-05]. Dostupné z: http://graphstream-project.org/doc/Generators/Overview-of-generators_1.0/

[18] ALBERT, Réka a Albert-László BARABÁSI. Emergence of Scaling in Random Networks. *Emergence of Scaling in Random Networks* [online]. 1999 [cit. 2014-06-05]. DOI: 10.1126/science.286.5439.509. Dostupné z: http://www.sciencemag.org/content/286/5439/509

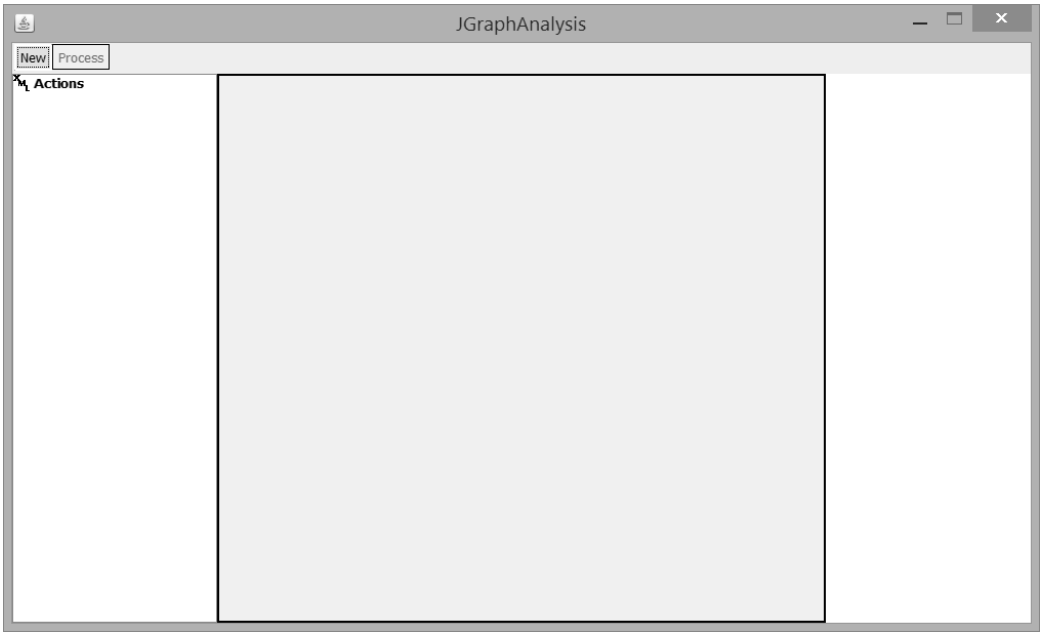
[19] BRAY, Tim, Jean PAOLI, C. M. SPERBERG-MCQUEEN, Eve MALER a François YERGEAU. Extensible Markup Language (XML). *W3C* [online]. 5. vyd. 2008, 7.2.2013 [cit. 2014-06-20]. Dostupné z: http://www.w3.org/TR/REC-xml

[20] EPPSTEIN, David. ICS 161: Design and Analysis of Algorithms: Strongly connected components. EPPSTEIN, David. UNIVERSITY OF CALIFORNIA, Irvine. *David Eppstein* [online]. 1996 [cit. 2014-06-16]. Dostupné z: `http://www.ics.uci.edu/~eppstein/161/960220.html#sca`

[21] GNU LESSER GENERAL PUBLIC LICENSE. FREE SOFTWARE FOUNDATION, Inc. *GNU Operating System* [online]. 3. vyd. 2007, 29.6.2007 [cit. 2014-06-24]. Dostupné z: `http://www.gnu.org/licenses/lgpl.html`

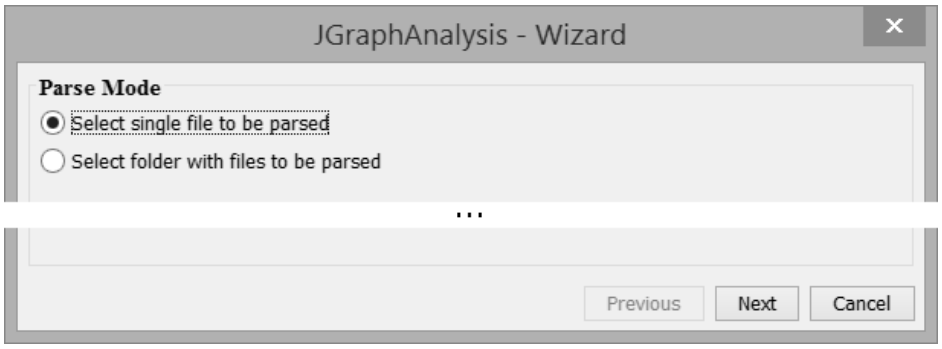
A Uživatelská dokumentace

Po spuštění spustitelného souboru *.jar se uživateli zobrazí hlavní okno aplikace. Po levé straně je výpis všech akcí a jejich referecní. Uprostřed je prostor pro výsledný graf a napravo prostor pro zobrazení dat daných akcí.



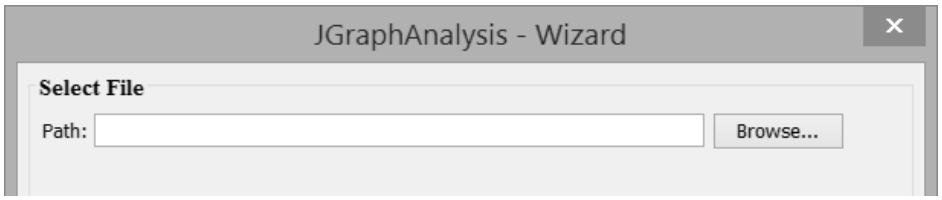
Obrázek A.1: Hlavní okno programu

Po kliknuní na tlačítko `New` se zobrazí průvodce načtením dat Akcí. V něm je možné vybrat, zda jsou akce uloženy v jednom souboru nebo ve více souborech.



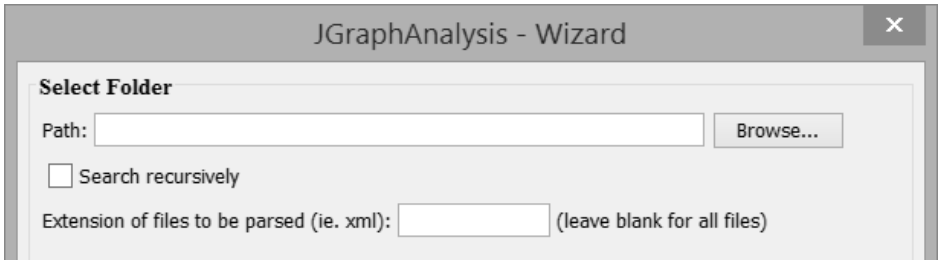
Obrázek A.2: Výběr zda akce budou v jednom nebo více souborech

V případě zvolení první možnosti (vstup z jediného souboru) se zobrazí pole pro výběr souboru. Buď může uživatel zadat cestu ručně, nebo může použít tlačítko `Browse...` pro výběr daného souboru.



Obrázek A.3: Volba vstupního souboru

V opačném případě dostane uživatel možnost zvolit adresář obsahující dané soubory, vybrat zda se má adresář procházet rekurzivně a příponu hledaných souborů. V případě nevyplnění přípony jsou brány všechny soubory.

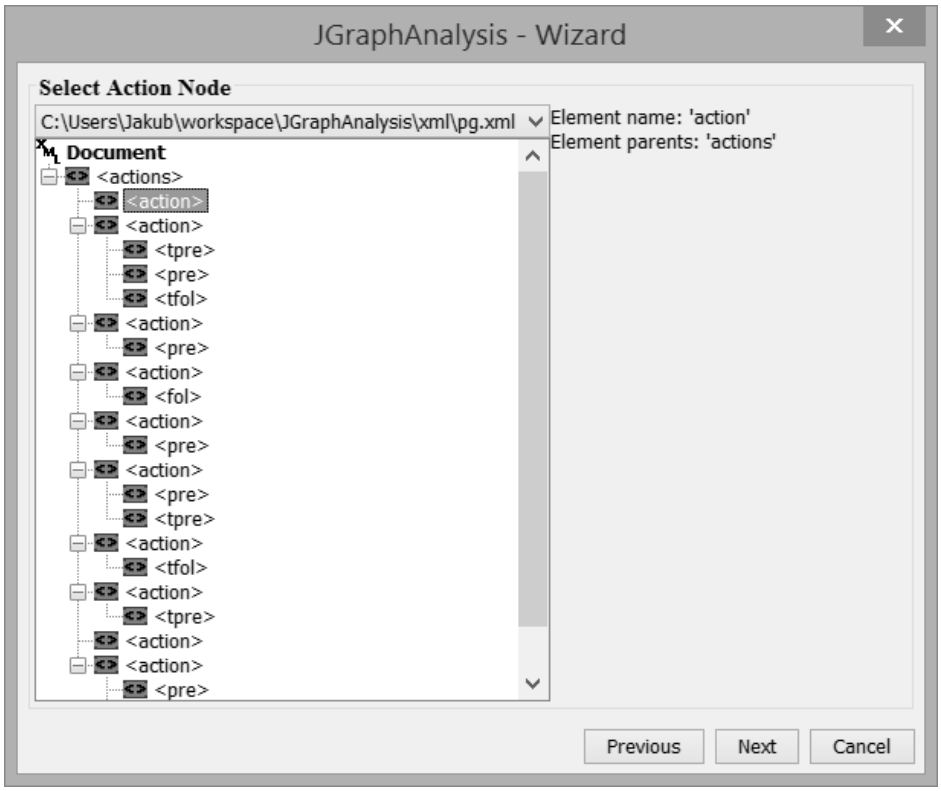


Obrázek A.4: Volba vstupního adresáře a parametrů souborů

V dalším kroce si uživatel může zvolit, zda zvolí potřebné elementy v XML souboru ručně, nebo načte nastavení ze souboru. Pokud zvolí import, bude po kliknutí na tlačítko `Next` dotázán na soubor obsahující nastavení a průvodce skončí.

Obrázek A.5: Výběr ručního zadání či importu nastavení ze souboru

Pokud uživatel zvolil ruční výběr, bude zobrazen strom elementů XML dokumentu. Nahoře může vybrat soubor (pokud je zvolena možnost načítání z více souborů). Klepnutím na položku stromu musí zvolit element reprezentující jednu akci.

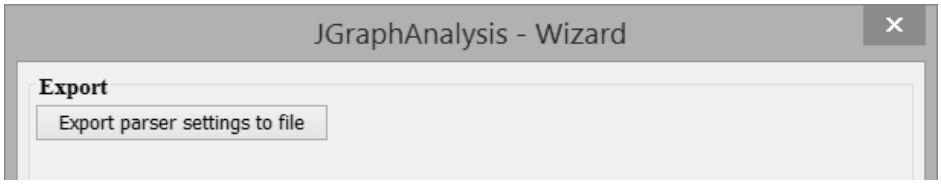


Obrázek A.6: Výběr elementu akce

Další krok je velmi podobný předchozímu, jen jsou ve stromě zobrazeny kromě elementů i jejich atributy z XML souboru. Uživatel zde musí vybrat element či atribut reprezentující ID akce. Následuje ještě pět identických kroků, kdy uživatel může (pokud nezvolí nebude daný parametr načítán) zvolit atributy či elementy dat, prerekvizit, těsných prerekvizit, následníků a těsných následníků.

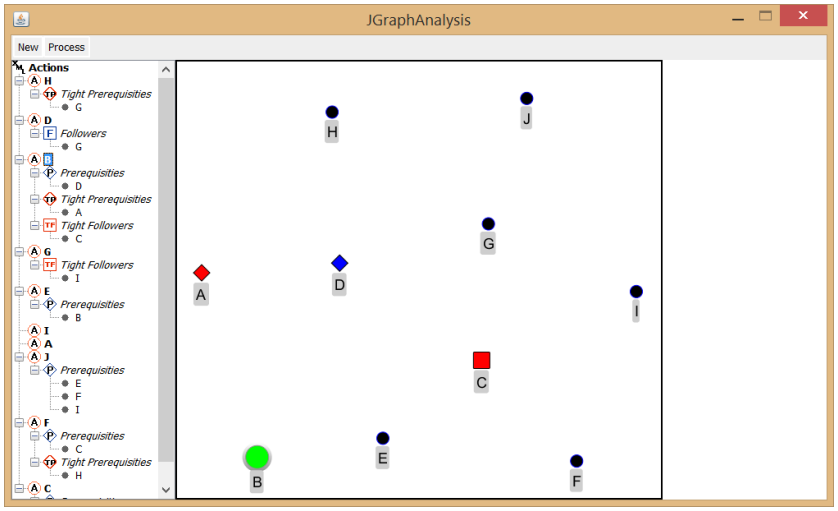
Obrázek A.7: Výběr elementu či atributu identifikátoru akce

Nakonec má uživatel možnost právě zvolená nastavení uložit do souboru.



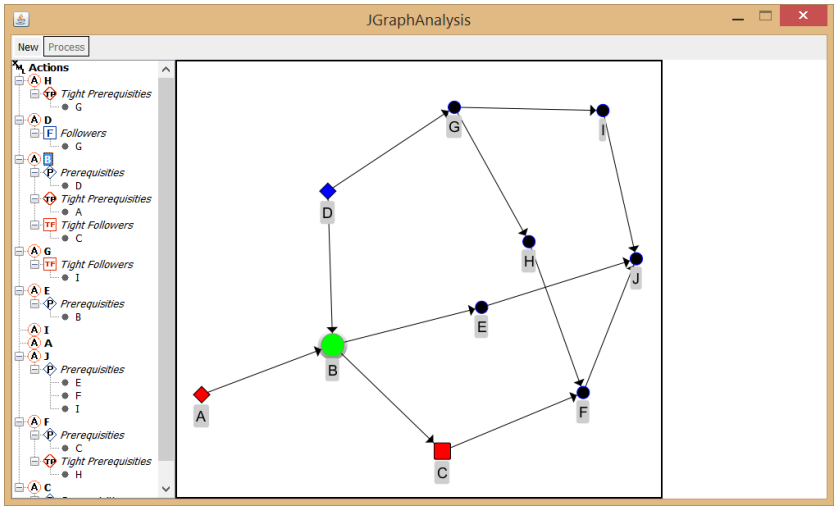
Obrázek A.8: Možnost uložení nastavení

Po ukončení průvodce se v hlavní okně zobrazí strom akcí a jejich referencí a v místě pro graf se zobrazí zatím nepropojené akce.



Obrázek A.9: Hlavní okno s načtenými akcemi

Tlačítkem `Process` lze nyní jednotlivé akce propojit.



Obrázek A.10: Hlavní okno s propojenými akcemi

Kliknutím na akci ve stromu, nebo přímo v grafu ji lze vybrat. Vybraná akce se označí v grafu zeleným kolečkem. Její prerekvizity jsou označeny kosočtvercem a následníci čtvercem. Těsní následníci a prerekvizity mají červenou barvu, následníci a prerekvizity mají modrou barvu. Pokud má akce nějaká data, jsou zobrazena napravo od grafu.

Pokud algoritmus odhalí nemožnost řešení a odebere některé vazby, pak uživatele na tuto skutečnost upozorní varováním a tyto vazby ve stromě označí červenou barvou textu.

B Seznam použitých zkratek

BFS ..... Breadth-First Search

CD ..... Compact Disk

CSS ..... Cascading Style Sheets

DFS ..... Depth-First Search

GML ..... Graph Modeling Language

GNU ..... anglicky „pakůň“

GPL ..... General Public License

ID ..... Identifier

SVG ..... Scalable Vector Graphics

UML ..... Unified Modeling Language

XML ..... Extensible Markup Language

Seznam obrázků

2.1 Neorientovaný (vlevo) a orientovaný graf (vpravo) 2

2.2 Znázornění nejkratší cesty v grafu, kostry grafu, komponent grafu a kliky grafu (zleva) 6

2.3 Náhodné/ruční, kruhové a hierarchické uspořádání (zleva) . . 7

2.4 Organické (vlevo) a stromové uspořádání (vpravo) 7

4.1 Ukázka vizualizace knihovny JUNG, vlevo ručně vytvořený graf, vpravo náhodně vygenerovaný graf o 200 vrcholech; organické uspořádání 14

4.2 Ukázka vizualizace knihovny GraphStream, vlevo ručně vytvořený graf, vpravo náhodně vygenerovaný graf o 200 vrcholech; organické uspořádání; použity stejné grafy jako u knihovny JUNG 17

4.3 Náhodný graf o 10 vrcholech a 10, 20 a 95 hranách; organické uspořádání; nahoře GraphStream, dole JUNG 18

4.4 Náhodný graf o 50 vrcholech a 50, 100 a 2 495 hranách; organické uspořádání; nahoře GraphStream, dole JUNG 19

4.5 Náhodný graf o 100 vrcholech a 100, 200 a 9 995 hranách; organické uspořádání; nahoře GraphStream, dole JUNG 19

4.6 Náhodný graf o 500 vrcholech a 500, 1 000 a 249 995 hranách; organické uspořádání; nahoře GraphStream, dole JUNG 20

4.7 Náhodný graf o 1 000 vrcholech a 1 000, 2 000 a 999 995 hranách; organické uspořádání; nahoře GraphStream, dole JUNG 20

4.8 Znázornění vizualizačních časů pro různé počty hran 22

4.9 Znázornění časů výpočtů pro algoritmy Dijkstra (vlevo) a Prim-Jarnik (vpravo) 23

4.10 Znázornění časů načtení grafů pro různé počty hran 25

4.11 Znázornění časů uložení grafů pro různé počty hran 25

4.12 Znázornění časů generování grafů pro různé počty iterací . . . 26

6.1 UML diagram jádra knihovny 32

6.2 Testování aplikace – příprava brambor 37

6.3 Testování aplikace – cyklické vazby 38

A.1 Hlavní okno programu 43

A.2 Výběr zda akce budou v jednom nebo více souborech 44

A.3 Volba vstupního souboru 44

A.4 Volba vstupního adresáře a parametrů souborů 44

A.5 Výběr ručního zadání či importu nastavení ze souboru 45

A.6 Výběr elementu akce 45

A.7 Výběr elementu či atributu identifikátoru akce 46

A.8 Možnost uložení nastavení 46

A.9 Hlavní okno s načtenými akcemi 47

A.10 Hlavní okno s propojenými akcemi 47

Seznam tabulek

2.1 Složitosti operací u jednotlivých implementací[2] 4

3.1 JGraphT 8

3.2 JGraph 9

3.3 JSL 9

3.4 JUNG 10

3.5 Grph 10

3.6 FlexiGraph 11

3.7 GraphStream 11

3.8 JDSL 12

3.9 yFiles 12

4.1 Rychlost vizualizace knihovnou GraphStream; V – počet vrcholů, E – počet hran 21

4.2 Rychlost vizualizace knihovnou JUNG; V – počet vrcholů, E – počet hran 21

4.3 Rychlost nalezení nejkratších vzdáleností algoritmem Dijkstra; V – počet vrcholů, počet hran = 2V 22

4.4 Rychlost nalezení minimální kostry grafu algoritmem Prim-Jarnik; V – počet vrcholů, počet hran = 2V 23

4.5 Rychlost načtení grafu knihovnou GraphStream; V – počet vrcholů, E – počet hran 24

4.6 Rychlost načtení grafu knihovnou JUNG; V – počet vrcholů, E – počet hran 24

4.7 Rychlost uložení grafu knihovnou GraphStream; V – počet vrcholů, E – počet hran 24

4.8 Rychlost uložení grafu knihovnou JUNG; V – počet vrcholů, E – počet hran 24

4.9 Rychlost vygenerování grafu; algoritmus Barabàsi-Albert . . . 26