Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

# BAKALÁŘSKÁ PRÁCE

Demonstrace algoritmů pro kreslení svazů

2015

Adam Drda

Vedoucí práce: Jan Konečný

Studijní obor: Aplikovaná informatika, prezenční forma

**Bibliografické údaje**

| | |
|---|---|
| Autor: | Adam Drda |
| Název práce: | Demonstrace algoritmů pro kreslení svazů |
| Typ práce: | bakalářská práce |
| Pracoviště: | Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci |
| Rok obhajoby: | 2015 |
| Studijní obor: | Aplikovaná informatika, prezenční forma |
| Vedoucí práce: | Jan Konečný |
| Počet stran: | 40 |
| Přílohy: | 1 CD/DVD |
| Jazyk práce: | český |

**Bibliograhic info**

| | |
|---|---|
| Author: | Adam Drda |
| Title: | Demonstration of lattice drawing algorithms |
| Thesis type: | bachelor thesis |
| Department: | Department of Computer Science, Faculty of Science, Palacký University Olomouc |
| Year of defense: | 2015 |
| Study field: | Applied Computer Science, full-time form |
| Supervisor: | Jan Konečný |
| Page count: | 40 |
| Supplements: | 1 CD/DVD |
| Thesis language: | Czech |

## Anotace

*V práci se zabývám automatickou generací diagramů svazu – Hasseových diagramů. Zaměřuji se na tři různé algoritmické metody – úrovňovou, geometrickou a Force-directed metodu, kterými lze diagram vygenerovat. Pro přiblížení funkčnosti metod a problematiky automatického generování grafu jsem vytvořil aplikaci, která postupnými animacemi průběh generování diagramu demonstruje.*

## Synopsis

*The thesis deals with automatic generation of a specific type of graph used for visualization of lattice - Hasse diagram. Mainly I focus on three methods - level, geometric and force-directed method which could be used to create such a diagram. To show how these methods work I have created an application which animates the process of graph creation with each method from beginning to end.*



**Klíčová slova:** uspořádaná množina; svaz; Hasseův diagram; demonstrace algoritmů pro vizualizaci svazu; Java; počítačová aplikace; technická příručka

**Keywords:** ordered set; lattice; Hasse diagram; lattice drawing algorithms; Java; desktop application; documentation

Děkuji Janu Konečnému za skvělou spolupráci a vedení mé práce. Přítelkyni a rodině za psychickou podporu při tvorbě této práce.

*Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.*

datum odevzdání práce podpis autora

# Obsah

# Seznam obrázků



# Seznam tabulek



# Seznam vět

# Seznam zdrojových kódů

# 1 Úvod

Vizualizace uspořádaných množin a obecně libovolných dat grafy je důležitá v mnoha vědních disciplínách. Praktické využití můžeme nalézt například v softwarovém inženýrství, umělé inteligenci, databázových systémech a dalších nejen informatických sektorech.

Vizualizace lze vytvářet ručně – nakreslením grafu „na papír“. Nicméně s rostoucím počtem prvků často klesá čitelnost kresleného grafu a značně roste čas potřebný k jeho vytvoření. Proto je potřeba grafy generovat automaticky, dle předem stanovených algoritmických metod.

Ve své práci se zaměřuji na automatické generování Hasseova diagramu – specifického typu grafu, používaného pro vizualizaci struktury svazu. Nástrojů pro vizualizaci svazu existuje poměrně mnoho, některé dokonce umožňují výběr z různých algoritmických metod při generování diagramu. Mezi ty nejzajímavější nástroje patří programy *JLatVis*, *ToscanaJ* a *EllenaArt*. Z těchto tří je dle mého názoru nejlepší program *JLatVis*, který poskytuje rozsáhlé možnosti vizualizace a umožňuje práci se strukturou svazu. Program *ToscanaJ* se zaměřuje na vizuální podobu diagramu, ale postrádá funkce pro práci se svazem. *EllenaArt* je zaměřen na generování diagramu dle několika algoritmických metod s možností je parametrizovat. Všechny zmíněné nástroje slouží primárně k vytvoření diagramu a práci s ním, ale žádný neumožňuje vizualizovat průběh generování – zobrazit formování grafu různými metodami. Proto se v práci orientuji převážně na průběh tvorby grafu, který je animován mnou vytvořeným programem.

Po tomto úvodu nejprve zmiňuji nutné teoretické základy z oblasti uspořádaných množin, svazů a teorie grafů, které přímo souvisí s funkčností programu. V následující kapitole rozebírám implementované metody, které jsem pro práci vybral a ukazuji příklady vygenerovaných diagramů pomocí jednotlivých metod. Ve čtvrté kapitole jsem se zaměřil na aplikaci LatDra, která byla vytvořena v rámci této práce. Uvádím požadavky, které byly kladeny při vytváření, uživatelský popis jednotlivých funkcí aplikace a její ovládání. Popisuji zvolené technologie, architekturu programu a způsob jakým lze program rozšířit o vlastní metody. Nakonec uvádím programátorský popis tříd.

# 2 Teorie

I přes to, že je téma práce především praktické se čtenář neobejde bez teoretického minima, z kterého program vychází. V celé kapitole předpokládám čtenářovu znalost základních poznatků o množinách a operacích s nimi (podmnožina, kartézský součin apod. ).

Informace k této kapitole jsou čerpány především z [4], [8], [9], některé další pak také z [7].

## 2.1 Relace, uspořádané množiny

**Definice 1 (Relace)**
Nechť jsou $A, B$ neprázdné množiny. Libovolnou podmnožinu $R$ kartézského součinu $A \times B$ nazveme **relací** mezi $A, B$. Jestliže platí $A = B$ pak podmnožinu $R$ nazveme **relací na množině** $A$.

Poznámka 2
Množinu $A$ s relací $R$ budeme značit jako $(A, R)$.

**Definice 3 (Vlastnosti relací)**
Nechť je $(A, R)$ množina s relací. Relaci $R$ nazveme:

- **reflexivní**, pokud $\forall x \in A$ platí, že $\langle x, x\rangle \in R$,
- **antisymetrickou**, pokud $\forall x, y \in A$ platí, že $\langle x, y\rangle \in R \wedge \langle y, x\rangle \in R \Rightarrow x = y$,
- **traznitivní**, pokud $\forall x, y, z \in A$ platí, že $\langle x, y\rangle \in R \wedge \langle y, z\rangle \in R \Rightarrow \langle x, z\rangle \in R$.

**Definice 4 (Relace uspořádání)**
Nechť je $(A, R)$ množina s relací. Relaci $R$ nazveme **uspořádáním**, pokud je $R$ reflexivní, antisymetrická a tranzitivní.

Poznámka 5
Nechť $A$ je množina s relací uspořádání. Pro zápis relace budeme používat symbol $\leq$ a pro zápis náležitosti $\langle x, y\rangle \in R$ použijeme zápis $x \leq y$. Dále, pokud platí $x \leq y \wedge x \neq y$ použijeme zápis $x < y$. Prvky $\langle x, y\rangle \in A$ nazveme **srovnatelnými** pokud platí $x \leq y \vee y \leq x$. Pokud prvky $\langle x, y\rangle \in A$ nejsou srovnatelné, pak je nazveme **nesrovnatelnými** a označíme jako $x \parallel y$.

**Definice 6 (Uspořádaná množina)**
Nechť je $(A, \leq)$ množina s relací uspořádání. Pak tuto množinu nazveme **uspořádaná množina**.

**Definice 7 (Prvky uspořádané množiny)**
Nechť je $(A, \leq)$ uspořádaná množina. Prvek $a \in A$ nazveme:

- **minimální**, pokud $\forall x \in A$ platí, že $x \leq a \Rightarrow x = a$,
- **maximální**, pokud $\forall x \in A$ platí, že $a \leq x \Rightarrow x = a$,
- **nejmenší**, pokud $\forall x \in A$ platí, že $a \leq x$,
- **největší**, pokud $\forall x \in A$ platí, že $x \leq a$.

V každé uspořádané množině $(A, \leq)$ existuje nanejvýš jeden největší a nanejvýš jeden nejmenší prvek. Také platí, že každá uspořádaná množina $(A, \leq)$ obsahuje alespoň jeden minimální prvek a alespoň jeden maximální prvek. Pokud $(A, \leq)$ obsahuje právě jeden největší prvek, pak je to také maximální prvek a jiné maximální prvky v množině $A$ neexistují. Pokud $(A, \leq)$ obsahuje právě jeden nejmenší prvek, pak je to také minimální prvek a jiné minimální prvky v množině $A$ neexistují.

**Definice 8 (Dolní a horní kužel)**
Nechť je $(A, \leq)$ uspořádaná množina a $M \subseteq A$, pak:

- **Dolním kuželem** $L \subseteq A$ podmnožiny $M$ rozumíme množinu všech prvků $a \in A$ takových, že pro $\forall m \in M$ platí $a \leq x$.
- **Horním kuželem** $U \subseteq A$ podmnožiny $M$ rozumíme množinu všech prvků $a \in A$ takových, že pro $\forall m \in M$ platí $a \geq x$.

**Definice 9 (Infimum a supremum)**
Nechť je $(A, \leq)$ uspořádaná množina a $M \subseteq A$, pak:

- $L(M)$ je dolním kuželem neprázdné podmnožiny $M$, pokud v $L(M)$ existuje největší prvek, pak se nazývá **infimum** $M$ a značí se $\inf(M)$.
- $U(M)$ je horním kuželem neprázdné podmnožiny $M$, pokud v $U(M)$ existuje nejmenší prvek, pak se nazývá **supremum** $M$ a značí se $\sup(M)$.

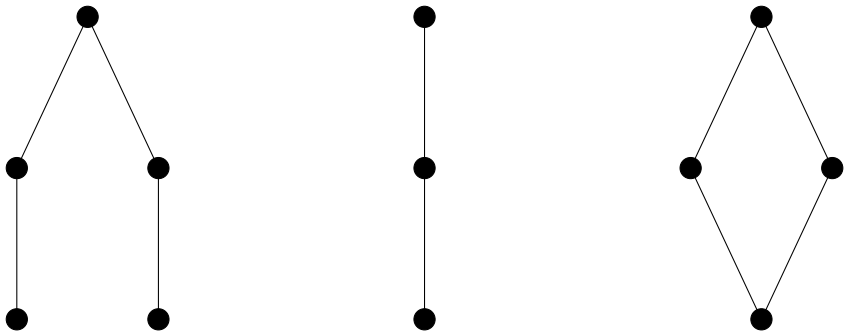
## 2.2 Hasseův diagram

**Definice 10 (Relace pokrytí)**
Nechť je $(A, \leq)$ konečná uspořádaná množina a $x, y \in A$. Jestliže platí $x \leq y$ a pro $\forall z \in A$, pro které platí $x \leq z \leq y$, platí $z = x$ nebo $z = y$, pak řekneme, že je prvek $x$ **pokryt** prvkem (je předchůdcem prvku) $y$ a označujeme jako $x \prec y$. Relaci $\prec$ nazveme **relace pokrytí**.

**Definice 11 (Hasseův diagram)**
Konečnou uspořádanou množinu $(A, \leq)$ lze znázornit v rovině pomocí tzv. **Hasseova diagramu**. Hasseův diagram je orientovaný graf (viz sekci 2.3), který můžeme sestrojit následovně:

- každý prvek $x$ z množiny $A$ budeme reprezentovat bodem v rovině,
- jestliže $x < y$ $(x, y \in A)$, pak bude bod $x$ v rovině umístěn níže než bod $y$,
- jestliže $x \prec y$ $(x, y \in A)$, pak spojíme prvky $x$ a $y$ úsečkou.

Dodávám však, že konstrukce podle výše zmíněného nijak nedefinuje tvar diagramu, takže můžou dva naprosto odlišné diagramy zobrazovat jednu a tu samou uspořádanou množinu. Příklady Hasseových diagramů jsou zobrazeny na obrázku 1.



Obrázek 1: Příklady Hasseových diagramů

**Definice 12 (Speciální případy uspořádaných množin)**
Nechť je $(A, \leq)$ konečná uspořádaná množina, jestliže:

- má každý prvek $x \in A$ právě jednoho předchůdce a právě jednoho následovníka, nazveme $(A, \leq)$ **řetězcem**,
- pro každý prvek $x \in A$ platí, že není srovnatelný s ostatními prvky množiny $A$, nazveme $(A, \leq)$ **antiřetězcem**.

## 2.3 Teorie grafu

Pojem graf je velice obecný, můžeme se s ním setkat v různých odvětvích informatiky, matematiky nebo přírodních věd. My budeme graf chápat jako reprezentaci množiny objektů se znázorněnými vztahy mezi jednotlivými objekty.

Tyto objekty budeme reprezentovat vrcholy a jednotlivé vztahy mezi vrcholy pomocí hran. Graficky jsou vrcholy grafu obvykle znázorněny jako kruhy nebo kružnice, hrany pak většinou jako křivky mezi vrcholy.

**Definice 13 (Orientace hran)**
Hrany můžeme rozdělit na **orientované** a **neorientované**.

- **Neorientovaná** hrana reprezentuje symetrický vztah mezi vrcholy a označuje se neuspořádanou dvojicí.

- **Orientovaná** hrana reprezentuje vztah jednosměrný a označuje se uspořádanou dvojicí vrcholů. Křivka orientované hrany je běžně zakončena šipkou ve směru orientace.

Z tohoto rozdělení hran přímo plyne dělení grafů na **orientované** a **neorientované** grafy.

**Definice 14 (Orientovaný graf)**
**Orientovaný graf** $G$ můžeme formalizovat dvojicí $(V, A)$. $V$ reprezentuje konečnou množinu vrcholů a $A$ je množina orientovaných hran – uspořádaných dvojic vrcholů z $V$.

**Definice 15 (Neorientovaný graf)**
**Neorientovaný graf** $G$ je definován podobně jako graf orientovaný. $G = (V, A)$, kde $V$ reprezentuje konečnou množinu vrcholů a $A$ je množina neorientovaných hran – neuspořádaných dvojic vrcholů z $V$.

Poznámka 16
V sekci 2.2 jsem definoval Hasseův diagram jako orientovaný graf. Konvence kreslení Hasseova diagramu je však lehce odlišná od konvence běžného orientovaného grafu. Pro přehlednost a čistotu výsledného diagramu se při jeho tvorbě vynechávají šipky orientace hran, protože orientace přímo vyplývá z umístění vrcholů. Vynechány jsou také hrany, které plynou z reflexivity a tranzitivy.

## 2.4 Svazy

**Definice 17 (Svaz)**
Nechť je $(A, \leq)$ uspořádaná množina. Jestliže pro libovolné prvky $a, b \in A$ existuje $\sup(a, b)$, označené jako $a \vee b$ a $\inf(a, b)$, označené jako $a \wedge b$, pak uspořádanou množinu $(A, \leq)$ označíme $(A, \vee, \wedge)$ a nazveme **svaz**.

**Definice 18 (Úplný svaz)**
Svaz $(A, \vee, \wedge)$ nazveme **úplným svazem** pokud pro každou $X \subseteq A$ existuje $\sup(X)$ a $\inf(X)$.

**Definice 19 (Nula a jednička svazu)**
Nechť je $(A, \vee, \wedge)$ svaz. Nejmenší prvek svazu nazýváme **nula svazu** (značeno 0) a největší prvek svazu nazýváme **jednička svazou** (značeno 1). Svaz s nulou a jedničkou označujeme jako $(A, \vee, \wedge, 0, 1)$.

# 3 Implementované metody

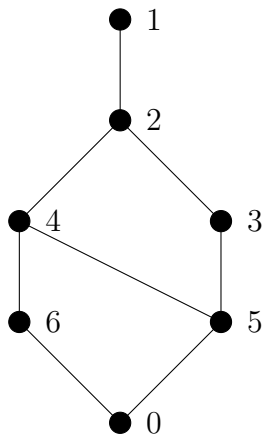
Pro automatickou generaci diagramu svazu existuje poměrně značný počet algoritmických metod. Při výběru implementovaných metod jsem se snažil volit ty, jejichž procesy generování jsou co nejodlišnější, a to ze dvou důvodů:

- aby následná animace generování diagramu nepůsobila vždy stejně či podobně,
- aby program obsahoval co možná největší zásobu nástrojů pro případné rozšíření o další metody.

## 3.1 Úrovňová metoda (Level Method)

Úrovňová metoda je nejjednodušší algoritmus implementovaný v programu. Uživatel může metodu chápat jako určitý úvod do automatické generace Hasseova diagramu. Metoda není založena na hlubších teoretických základech, vrcholy jsou při generování skládány do úrovní dle uspořádání. Průběh metody by šel shrnout následovně:

- Nejdříve je vygenerovaná první úroveň – všechny minimální vrcholy vstupní uspořádané množiny. Vrcholy jsou vygenerovány souměrně ke středu osy $x$.
- V dalším kroku vybere algoritmus následovníky všech vrcholů předchozí úrovně a vytvoří z nich další úroveň, vrcholy jsou opět generovány souměrně ke středu osy $x$. Pokud je mezi následovníky vrchol, který je vygenerován v předchozí úrovni, pak je přesunut do nové, aktuální úrovně. Po vytvoření celé úrovně jsou k vrcholům vygenerovány hrany. Krok je rekurzivně opakován až do chvíle, kdy je množina následovníku předchozí úrovně prázdná, pak generování končí.




Obrázek 2: Diagram vygenerovaný úrovňovou metodou

## 3.2 Geometrická metoda (Geometric Method)

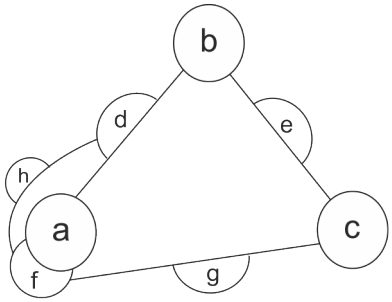
Tato metoda vychází z myšlenky, že dobré rozpoložení prvků diagramu lze získat z jeho geometrické reprezentace. Geometrickou reprezentaci získáme následovně. Představme si svaz jako diagram ve 3D, na který se budeme dívat z jeho nejmenšího bodu směrem nahoru. Nejprve uvidíme následovníky nejmenšího prvku, ty jsou v geometrickém zobrazení reprezentovány kroužky. Poté pokračujeme podle následujících pravidel:

- Prvek s jedním předchůdcem je reprezentován kroužkem, který je částečně překrytý kroužkem tohoto předchůdce.
- Prvek s dvěma předchůdci je reprezentován čarou mezi předchůdci, na které je prvek označen částečně překrytým kroužkem.
- Prvek s třemi předchůdci je reprezentován trojúhelníkem, který předchůdce propojuje. Prvek je zapsán uprostřed trojúhelníku.
- Analogicky, prvek s více jak třemi předchůdci je reprezentován n-úhelníkem, který předchůdce propojuje a prvek je zapsán dovnitř n-úhelníku.

V geometrické reprezentaci se neuvádí nejmenší ani největší prvek. Příklad geometrické reprezentace svazu určeného tabulkou 1 je možné vidět na obrázku 3.

| | |
|---|---|
| 1 | e, g, h |
| h | d, f |
| g | c, f |
| f | a |
| e | b, c |
| d | a, b |
| c | 0 |
| b | 0 |
| a | 0 |
| 0 | |

Tabulka 1: Svaz určený seznamem předchůdců.



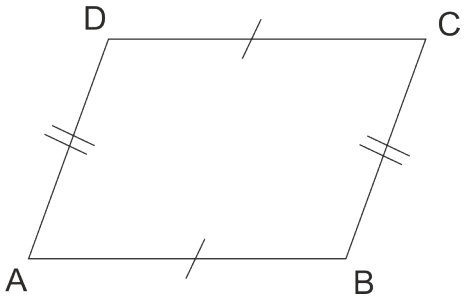

Obrázek 3: Geometrická reprezentace svazu.

Hasseův diagram vzniká z geometrické reprezentace pomocí *pravidla paralelogramu* a *pravidla prodlužování hran*. Obě pravidla představují poměrně jednoduché mechanismy, kterými je v grafu zajištěna určitá geometrická pravidelnost.

Proces vytvoření diagramu z jeho geometrické reprezentace začíná položením následovníků nejmenšího prvku, ty jsou v grafické reprezentaci označeny nepřerušenými kroužky. Následuje vnořování do grafické reprezentace a generování propojených kroužků. Vrcholy, které jsou v grafické reprezentaci propojeny se dvěma a více kroužky jsou položeny dle pravidla paralelogramu. Vrcholy, které jsou propojeny pouze jedním kolečkem jsou položeny dle pravidla prodlužování hran. Příklad diagramu vytvořeného geometrickou metodou je na obrázku 6.

### 3.2.1 Pravidlo paralelogramu

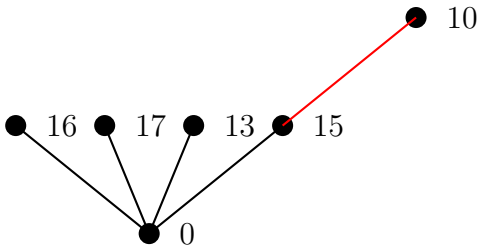
Paralelogram (viz obr. 4) je čtyřúhelník, jehož protilehlé strany jsou rovnoběžné a mají stejnou délku. Pravidlo paralelogramu říká, že vrchol bude v grafu umístěn tak, že spolu s třemi již umístěnými vrcholy, které jsou propojené hranami bude tvořit paralelogram.



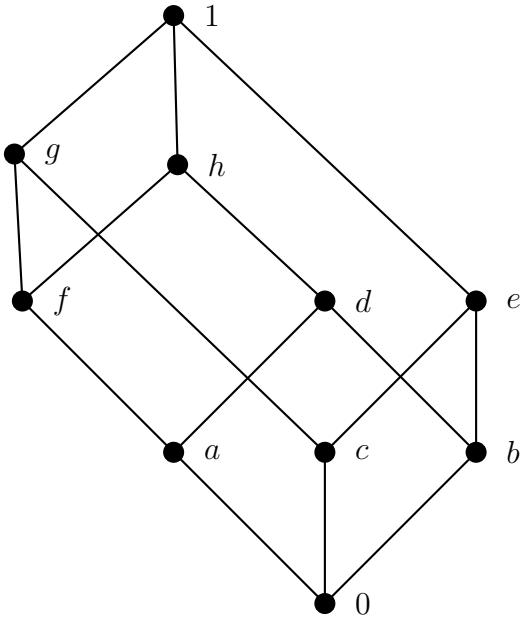

Obrázek 4: Paralelogram

### 3.2.2 Pravidlo prodlužování hran

Pravidlo paralelogramu nelze použít v případě, kdy do daného vrcholu vchází pouze jedna hrana. Pak se užívá pravidla prodlužování hran (viz obr. 5), při kterém je vrchol umístěn na pozici, která vznikne prodloužením předchozí hrany.




Obrázek 5: Příklad užití pravidla prodlužování hran při umístění vrcholu.




Obrázek 6: Diagram vygenerovaný geometrickou metodou.

Informace ke geometrické metodě jsem čerpal z [7] a [6].

## 3.3 Force-directed metoda

Poslední implementovaná metoda je oproti dvěma předchozím značně odlišná. Algoritmus vychází z principů fyziky, graf je chápan jako systém, mezi jehož

prvky působí přitažlivé a odpudivé síly. Vrcholy si lze představit jako elektricky nabité částice, které se navzájem odpuzují a hrany jako pružiny, které částice propojují. Česky by šla metoda přeložit například jako „silou-usměrňovaná“, ale v práci jsem se rozhodl ponechat anglický název *Force-directed*, pod kterým se problematika běžně vyskytuje.

Úrovňová ani geometrická metoda před samotnou generací neuvažuje jakékoliv rozpoložení prvků. Force-directed metoda naopak rozpoložení prvků vyžaduje a výsledek je tímto prvotním rozmístěním vrcholů ovlivněn. V mé implementaci se při volbě Force-directed metody prvky vstupního svazu náhodně rozloží po plátně (kromě největšího a nejmenšího prvku, které se umístí nejvýše, resp. nejníže vůči ostatním prvkům) a povolí se systém *Drag & Drop*, aby uživatel mohl vrcholy libovolně přemístit.

Formování grafu z prvotního rozmístění probíhá v iteracích. V každé iteraci se pro každý vrchol vypočítá síla, která na něj působí a následně je vrchol posunut ve směru síly o její nepatrný podíl. Výpočet síly, která na vrchol $v$ působí, probíhá podle následujícího vzorce:

$$F(v) = \sum_{(u,v)\in V \ \times V} g_{uv} + \sum_{(u,v)\in E} f_{uv},$$

- kde $g_{uv}$ představuje elektrický odpor, který na vrchol $v$ klade vrchol $u$,
- $f_{uv}$ značí sílu, kterou na vrchol $v$ působí pružina mezi $u$ a $v$.

Výpočet elektrického odporu $g_{uv}$:

$$g_{uv_x} = \frac{c_1}{(l(u,v))^2} \cdot \frac{x_v - x_u}{l(u,v)}$$

$$g_{uv_y} = \frac{c_1}{(l(u,v))^2} \cdot \frac{y_v - y_u}{l(u,v)}$$

Výpočet síly $f_{uv}$, která je na vrchol $v$ pružinou kladena:

$$f_{uv_x} = c_2(l(u,v) - c_3) \cdot \frac{x_v - x_u}{l(u,v)}$$

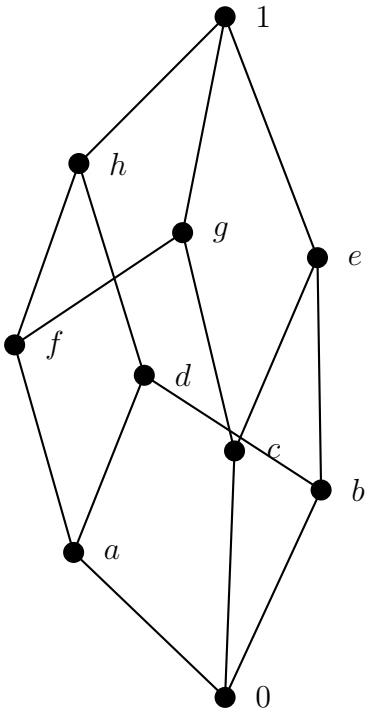
$$f_{uv_y} = c_2(l(u,v) - c_3) \cdot \frac{y_v - y_u}{l(u,v)}$$

Konstanty, které se ve vzorcích užívají představují následující:

- $c_1$ ovlivňuje velikost elektrického odporu, čím větší je hodnota, tím více se vrcholy odpuzují. Při malé hodnotě se vrcholy shlukují.
- $c_2$ představuje tuhost pružin, čím je hodnota konstanty vyšší, tím spíše se vzdálenost mezi $u$ a $v$ rovná $c_3$.

- $c_3$ představuje „neutrální“ vzdálenost mezi vrcholy $u$ a $v$, při které na pružinu mezi vrcholy není vyvíjená žádná síla.

Počet iterací, které se provedou závisí na volbě uživatele. Abych dodržel konvence Hasseova diagram, tak jsem omezil působení sil zafixováním polohy nejmenšího a největšího vrcholu v diagramu a přidal podmínku, podle které se žádný prvek nemůže dostat pod resp. nad jejich polohu. Omezení je však možné vypnout v nastavení programu. Diagram vygenerovaný Force-directed metodou je na obrázku 7.




Obrázek 7: Diagram vygenerovaný force-directed metodou.

Informace k force-directed jsem čerpal z [1].

# 4 Aplikace LatDra

Hlavní náplní práce bylo vytvoření aplikace pro vizualizaci průběhu generování Hasseova diagramu. Název LatDra je zkratkou za Lattice Drawing – kreslení svazů. V kapitole se věnuji uživatelskému a následně i programátorskému popisu aplikace.

## 4.1 Požadavky

### 4.1.1 Povinné požadavky

Povinné požadavky, které jsou dány zadáním bakalářské práce nebo byly kladeny při tvorbě aplikace jsou následující:

- řešení animovaného generování diagramu svazu pomocí vybraných metod,
- implementace vybraných metod,
- možnost přepínání mezi manuálním krokováním průběhu a automatickým generováním,
- neomezený průchod průběhu generování diagramu dopředu i dozadu,
- možnost okamžitého skoku na kompletně vygenerovaný diagram,
- možnost restartovat generování z libovolného kroku,
- výpis informací o aktuálně vykonávaném kroku,
- načtení svazu z XML souboru,
- uložení libovolného kroku generace diagramu do formátu TikZ,
- možnost jednoduchého rozšíření aplikace o další metody.

### 4.1.2 Nepovinné požadavky

Požadavky, které nebyly předmětem zadání bakalářské práce, tyto požadavky jsem při tvorbě program kladl sám sobě:

- možnost změnit rychlost generování nebo vypnout animace vybraných operací (položení vrcholu, přesunutí vrcholu, ... ) v průběhu generování, bez potřeby restartovat proces,
- implementovat Drag & Drop systém vrcholů, který bude dostupný vždy, když jej právě označená metoda povoluje,
- základní možnosti vizualizace – změny velikosti vrcholů, hran, barvy grafu,
- indikace aktuálního stavu generace.

## 4.2 Zvolené technologie

Pro vytvoření aplikace jsem zvolil moderní jazyk Java a grafické rozhraní JavaFX. Volba byla subjektivní, z velké míry ovlivněna předešlými zkušenostmi s programováním v jazyce Java.

Jazyk Java je platformně nezávislý, objektově orientovaný jazyk. Výhodou Javy je dostupnost špičkových vývojových prostředí (Eclipse, IDE NetBeans), jednoduchá přenositelnost nebo podpora paralelismu procesů, kterou jsem při vytváření využil.

Volba grafického rozhraní byla obtíženější. Původně byl program vyvíjen v grafické knihovně Swing, ale kvůli nedostatečné podpoře animace byl později celý předělán pod novější rozhraní JavaFX. JavaFX je moderní framework s širokou podporou animace, CSS stylů a dalších technologií. Uživatelské rozhraní, které se v JavěFX vytváří v *JavaFX scene builderu*, je uloženo ve formátu FXML – formátu odvozeného z XML.

Celý program byl vytvořen ve výše zmíněném vývojovém prostředí Eclipse, které je dostupné zdarma. Prostředí Eclipse obsahuje v základní verzi pouze prostředky pro vývoj standardní Javy. Pro vývoj JavyFX jsem použil rozšíření *e(fx)clipse*.

## 4.3 Instalace a spuštění programu

Jediným požadavkem pro spuštění aplikace je nainstalovaná Java verze *8u51* a vyšší. Aplikace by měla běžet na každém moderním počítači.

## 4.4 Uživatelská příručka

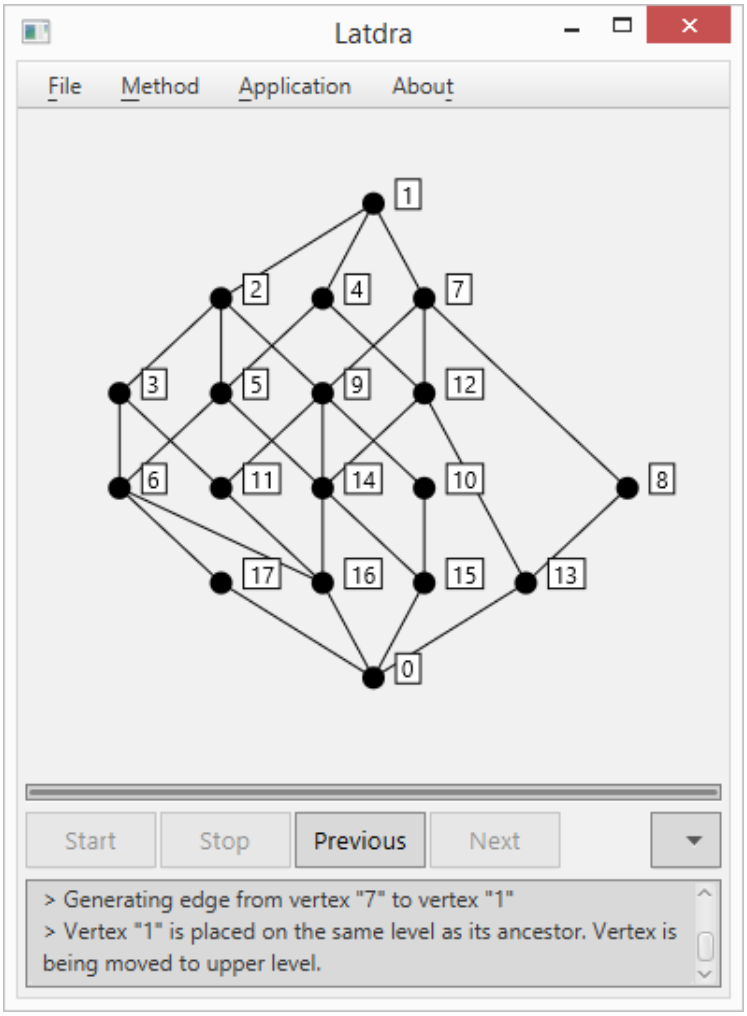
### 4.4.1 Obecný popis

Při tvorbě uživatelského rozhraní jsem se nechal inspirovat klasickým vzhledem přehrávačů videa. Uživatelské rozhraní přehrávače je typicky rozděleno na horní menu, velkou plochu pro video a panel ovládacích prvků spolu s indikátorem aktuální pozice přehrávání. Aplikaci LatDra lze označit za přehrávač průběhu generování diagramu, proto se rozhraní přehrávače výborně hodí. Pro pohodlnou práci s generací jsou komponentům přidány klávesové zkratky.

Vizuální vzhled aplikace zajišťuje CSS soubor, který aplikaci styluje do bílo-šedých barev. Vzhled jsem se snažil zvolit tak, aby nenarušoval uživatelovu práci.

### 4.4.2 Hlavní okno

Program je tvořen jedním hlavním oknem, které lze neomezeně zvětšovat. Okno má nastavenou minimální šířku a výšku, pod kterou nemůže být zmenšeno.




Obrázek 8: LatDra – hlavní okno.

#### 4.4.2.1 Hlavní menu

**File (Alt+F)**

Obsahuje příkazy pro práci se soubory.

**Import**

Zobrazí dialogové okno pro výběr a načtení souboru ve formátu (*.xml). Pokud je vyvoláno zobrazení dialogu a zároveň probíhá generování, pak je průběh pozastaven. Pokud je vybrán soubor a v pozadí pozastaveno generování, pak je generace v pozadí zrušena a nahrazena generací souboru nového.

**Export**

Zobrazí dialogové okno pro uložení aktuálního stavu diagramu do souboru formátu (*.tex).
Položka export je dostupná pouze při pozastavené generaci diagramu.

**Exit**
Ukončí činnost programu.

**Method (Alt+M)**
Obsahuje jednotlivé metody pro generaci diagramu. Položky menu jsou generovány automaticky pro všechny implementované metody.
Označena může být jen jedna metoda. Pokud probíhá generování a zároveň je vybrána nová metoda, pak je průběh generace zahozen a je vytvořena nová generace pro nově vybranou metodu.

**Level method**
Úrovňová metoda 3.1.

**Geometric method**
Geometrická metoda 3.2.

**Force-directed method**
Force-directed metoda 3.3. Při zvolení Force-directed metody jsou prvky vstupního svazu náhodně rozmístěny po plátně (kromě největšího a nejmenšího prvku, které se umístí nejvýše, resp. nejníže vůči ostatním prvkům).

**Application (Alt+A)**
Obsahuje příkazy pro práci s aplikací.

**Preferences**
Otevře dialogové okno s možnostmi pro změnu nastavení generace, vzhledu a exportu diagramu. Otevření okna nepřeruší chod generace.

**About (Alt+T)**
Otevře okno se základními informacemi o aplikaci.

#### 4.4.2.2 Plátno diagramu

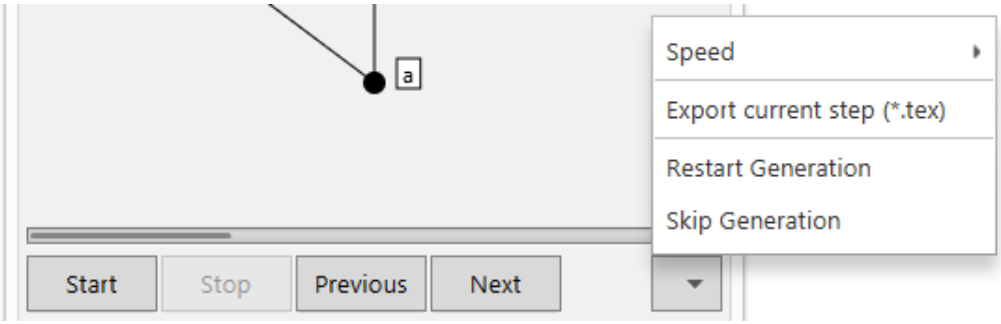
Největší část programu zabírá plátno diagramu. Při pozastavené generaci je možné, pokud to vybraná metoda povoluje, pohybovat s vygenerovanými vrcholy principem *Drag & Drop*. Přesun vrcholu nenaruší chod generování.

#### 4.4.2.3 Indikátor průběhu generace

Těsně pod plátnem je umístěn indikační pruh – *progress bar* aktuálního stavu generování. Pruh je vyplněn procentuálním poměrem k celkovému počtu operací generace. Na začátku generování je pruh prázdný a po poslední operaci je pruh zcela vyplněn.

#### 4.4.2.4 Ovládací panel

Nejdůležitějším ovládacím prvek programu je panel tlačítek umístěný pod indikačním pruhem. Při spuštění programu jsou tlačítka panelu zakázaná a k jejich (částečnému) povolení dojde až po načtení souboru a vybrání metody pro generaci diagramu.




Obrázek 9: Latdra – indikátor stavu a hlavní panel tlačítek.

**Hlavní prvky**

**Start**
Začne automatické generování diagramu, pokud je generace pozastavena, pak ji obnoví. Stisknutím tlačítka dochází k zakázání tlačítek *Previous*, *Next* a povolení tlačítka *Stop*.

**Stop**
Dokončí aktuální operaci a pozastaví generování. Po vykonání aktuální operace dojde k povolení tlačítek *Start*, *Previous* a *Next*.

**Previous (Next)**
Přejde do předešlého (následujícího) stavu generování diagramu. Stisknutím dojde k zakázání všech hlavních tlačítek, po vykonání přechodu dojde k povolení tlačítek *Start*, *Previous* a *Next*.

Pokud je generování pozastaveno v nultém kroku, jsou povolený pouze tlačítka *Start* a *Next*. Pokud je generace dokončena je povoleno jediné tlačítko – *Previous*.

**Vedlejší prvky**

**Speed**
Posuvník rychlosti generování. Rychlost generování se mění bez potřeby restartu.

**Skip generation**
Přeskočí generování na konec – plně vygenerovaný diagram. Položka je dostupná pouze při pozastavené generaci diagramu.

**Restart generation**
Restartuje generování do nultého kroku.

**Export current step (*.tex)**
Stejné jako **File** → **Export**. Zobrazí dialogové okno pro uložení aktuálního stavu diagramu do souboru formátu (*.tex).
Položka Export current step (*text) je dostupná pouze při pozastavené generaci diagramu.

#### 4.4.2.5 Okno pro textový výstup

Při průběhu generování jsou v okně vypisovány informace o právě vykonávaných operacích. Zprávy operací jsou definované každou metodou zvlášť, ale není vyžadováno, aby metody obsahovaly zprávy pro každou operaci.

### 4.4.3 Nastavení programu

Program umožňuje měnit některá nastavení generace, vzhledu a exportu diagramu. Dialogové okno nastavení je vyvoláno z menu **Application** → **Preferences**.

Rozložení prvků okna nastavení je intuitivní. Levý, postranní panel obsahuje odkazy pro jednotlivé skupiny nastavení. Hlavní, největší panel okna se mění v závislosti na vybraném odkazu.
Následuje popis jednotlivých možností nastavení:

**General**
Obsahuje obecná nastavení chodu programu.

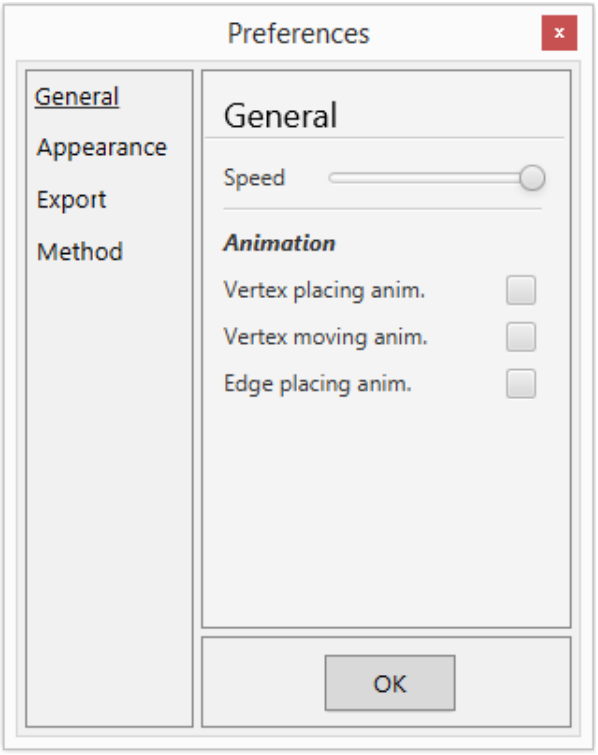
**Speed**
Posuvník rychlosti generování. Funkčností odpovídá posuvníku z panelu tlačítek.

**Vertex placing anim.**
Zakšrtávací tlačítko animace položení vrcholu. Při zaškrtnutí se vrchol postupně objevuje nebo postupně mizí (*Fade In / Out*).

**Vertex moving anim.**
Zaškrtávací tlačítko animace přesunutí vrcholu. Při zaškrtnutí je přesun vrcholu z pozice $(x_1, y_1)$ do pozice $(x_2, y_2)$ postupný – animovaný.

**Edge placing anim.**
Zaškrtávací políčko animace nakreslení hrany. Při zaškrtnutí se hrana vykresluje nebo mizí postupně – animace připomíná kresbu (umazávání) přímky.

**Appearance**
Obsahuje nastavení vzhledu diagramu.




Obrázek 10: Latdra – okno nastavení programu.

**Vertex (Edge) Size**
Rozbalovací seznam umožňující změnit velikost všech vrcholů (hran) generovaného diagramu.

**Vertex (Edge) Color**
Paleta barev umožňující změnit barvu všech vrcholů (hran) generovaného diagramu.

**Export**
Obsahuje nastavení exportu diagramu do souboru (*.tex).

**Method**
Nastavení volitelných parametrů jednotlivých metod.

**Parallelogram highlighting**
Pokud je tlačítko zaškrtnuto, pak se při generování diagramu geometrickou metodou zvýrazňují paralelogramy.

**Rule of Line highlighting**
Při zaškrtnutí se při generování diagramu geometrickou metodou zvýraňuje pravidlo prodlužování hran.

**Fixed minimal (maximal) element**

Při zaškrtnutí zafixuje polohu nejmenšího (největšího) prvku při generování Force-directed metodou.

**Repulsion strength**
Posuvník hodnoty síly odporu vrcholů při generování Force-directed metodou.

### 4.4.4 Import souboru

LatDra přímá soubory v datovém formátu XML. Příkladové soubory jsou k dispozici spolu s programem.

### 4.4.5 Sada řádkových programů

Jako vedlejší produkt aplikace LatDra jsem vytvořil sadu řádkových programů kompatibilních s aplikací JLatVis [5]. Jedná se o program, který lze v JLatVis použít pro vykreslení diagramu svazu. Program pro vstupní textovou strukturu diagramu, kterou obdrží přes standardní vstup ve formátu specifikovaném níže vrátí na standardní výstup pozice prvků diagramu po použití algoritmické metody.

`Příklad vstupní textové struktury diagramu:`

```
"a" 2 3 4
"b" 5 6
"c" 5 7
"d" 6 7
"e" 8
"f" 8
"g" 8
"h"
```

Každý řádek obsahuje prvek svazu v uvozovkách a seznam čísel nižších řádků (řádky jsou počítány od 1), oddělených mezerami.

`Příkladný výstup programu:`

```
"a" 0 0
"b" -60 -60
"c" 60 -60
"d" 0 -60
"e" 0 120
"f" -60 -120
"g" 60 -120
"h" 0 -180
```

Každý řádek obsahuje prvek svazu v uvozovkách a dvojici čísel - souřadnic x a y.

Program obsahuje nápovědu, která je vrácena při spuštění s parametrem *-h*. Nápověda je standardně zobrazena v okně JLatVis. Při spuštění programu s parametrem *-v* je vrácen název programu – *LatDra External Draw*.

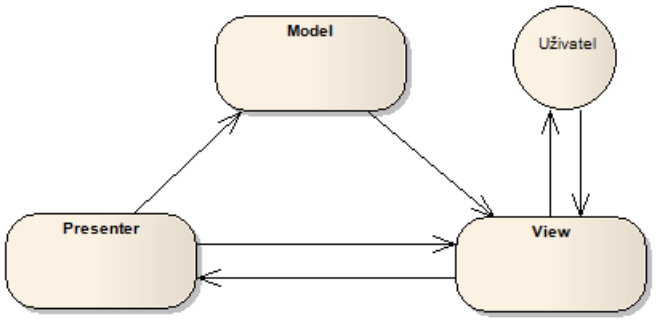
Pokud je program spuštěn bez parametrů, tak je pro vygenerování pozic vstupní struktury použita geometrická metoda. Pro volbu jiné metody je nutné spustit program s parametrem *-m <metoda>*. Kód úrovňové metody je *lm* a Force-directed metody fdm. Při volbě Force-directed metody je navíc možné definovat počet iterací, které se provedou při generování pozic parametrem *-i <pocet>* a sílu odporu vrcholů parametrem *-rs <sila>*.

## 4.5 Programátorský popis

### 4.5.1 Popis architektury programu

Největší podíl na volbě softwarové architektury má požadovaná funkčnost a chování programu. Při návrhu programu LatDra jsem za hlavní požadovanou funkčnost považoval možnost upravit vnitřní strukturu dat pomocí ovládacích prvků uživatelského rozhraní a následně upravená data zobrazit.

Mým požadavkům nejlépe odpovídala architektura (architektonický vzor) typu **Model–View–Presenter (MVP)**, která je derivací populární architektury Model–View–Controller (MVC). Architektura MVP rozděluje program do tří celků – *model, view* a *presenter*. Část *model* reprezentuje data. Část *view* představuje rozhraní, které zobrazuje data *modelu* a zpracovává uživatelský vstup, který deleguje části *presenter*. *Presenter* manipuluje s daty *modelu* čímž aktualizuje *view*.




Obrázek 11: Architektura Model–View–Presenter

Ač se může problematika MVP jevit jednoduše, není tomu tak. Existuje velké množství variací, které často stejné problémy řeší úplně jinak. Literatura v tomhle směru není taky nejpřívětivější, to co se v jedné knize popisuje jako nemyslitelné je v druhé běžně používáno. Případného zájemce o problematiku MVP (MVC) odkazuji na [3] a [2].

### 4.5.2 Logická a vizuální reprezentace svazu

Reprezentaci svazu zastřešují třídy *Vertex, Order* a *OrderedSet*.

**class Vertex** Třída reprezentuje vrchol svazu, obsahuje jméno vrcholu, dvojici souřadnic $(x, y)$ a odkaz na svoji vizuální nadstavbu *Node*, která se vytváří, když se vrchol graficky generuje.

**class Order** Třída reprezentuje uspořádání. Uspořádání je tvořeno dvěma vrcholy – *Vertex from* a *Vertex to*. Třída si udržuje odkaz na svoji vizuální nadstavbu *Edge*, která se vytváří, když se uspořádání graficky generuje.

**class OrderedSet** Třída reprezentuje strukturu svazu. Každý svaz má svůj název a udržuje si odkazy na všechny obsažené vrcholy a uspořádání mezi nimi. Třída obsahuje metody pro práci se svazem, které vychází z popsané teorie.

Při generaci diagramu se pro jednotlivé vrcholy a uspořádání vytvářejí nadstavby *Node* a *Edge*, které poskytují jejich vizuální podobu. S diagramem dále úzce souvisí třída *Tag*, která popisuje popisek vrcholu.

**class Node** Třída reprezentující vizuální podobu vrcholu diagramu. Třída dědí třídu *Circle* z grafického rozhraní JavaFX. Dědičnost umožňuje reprezentovat vrchol kruhem o zadaných souřadnicích a poloměru.

**class Edge** Třida poskytující vizuální reprezentaci uspořádání přímkou (hranou). Objekt třídy Edge dědí třídu *Line* z grafického rozhraní Javy FX. Obsahuje metody pro uchycení hrany mezi dvěma objekty třídy *Node*, uchycení se automaticky přizpůsobuje změně polohy vrcholu.

**class Tag** Třída reprezentující popisek vrcholu diagramu. Obsahuje text popisku, rámeček popisku a odkaz na vrchol, kterému patří. Metody třídy umožňují vhodně přizpůsobovat polohu popisku v diagramu.

### 4.5.3 Systém generace diagramu

Pro vytvoření systému generace diagramu jsem využil návrhový vzor **Command**. Použitím vzoru jsem dokázal rozdělit výpočet generování na jednotlivé operace, jejichž výpočet je ovládán manuálně prvky uživatelského rozhraní. Každá operace výpočtu navíc kromě definice toho, co se vykoná při pohybu dopředu obsahuje definici toho, co se vykoná při vracení se zpět. Tím jsem umožnil neomezený průchod průběhu generování diagramu tam i zpátky.

Návrhový vzor Command zapouzdřuje operaci a parametry operace do objektu tak, aby šla později použít. Každý zapouzdřený objekt implementuje metody, které udává rozhraní. Další objekt nazvaný *Invoker* má zapouzdřené objekty k dispozici a implementované metody spouští. Každá metoda provádí činnost na objektu nazvaném *Reciever*, který je podřízený zapouzdřenému objektu.

#### 4.5.3.1 Rozhraní Command

Jednotlivé operace pro generaci diagramu jsou v mém systému reprezentovány třídami implementujícími rozhraní *Command*. Rozhraní *Command* specifikuje následující metody:

**interface Command**

**void forward()** Specifikuje metodu, kterou operace definuje pohyb generování dopředu.

**void backward()** Specifikuje metodu, kterou operace definuje pohyb generování dozadu.

**void instantForward()** Specifikuje metodu, kterou operace definuje pohyb generování dopředu bez animací nebo časových prodlev. Použití při okamžitém přesunu na vygenerovaný diagram.

#### 4.5.3.2 Implementované operace

Každá z operací pro generaci diagramu pracuje nad objektem plátna diagramu. Objekt plátna, který v mém systému představuje výše zmíněný *Reciever*, je prvek grafického rozhraní knihovny JavaFX. Plátno vykresluje objekty, které metody operace vytvářejí, mění nebo ruší. V programu jsou implementovány následující operace:

**class PlaceVertex** Instancí třídy PlaceVertex je zapouzdřený objekt operace vygenerování vrcholu diagramu. Povinnými parametry pro vytvoření objektu PlaceVertex jsou souřadnice *double x, double y* a vrchol *Vertex v*. Metody, které udává rozhraní *Command* jsou implementovány následovně:

**void forward()** Metoda pro *Vertex v* vytvoří grafickou nadstavbu *Node*. Vytvořený objekt typu *Node* nastaví a pomocí animace vloží do plátna diagramu na souřadnice *x, y*. Po skončení animace vytvoří popisek typu *Tag*, který vloží do plátna diagramu na vhodnou, blízkou pozici ku vrcholu.

**void backward()** Metoda z plátna diagramu vymaže popisek *Tag* a spustí animaci odstranění objektu *Node*.

**void instantForward()** Metoda bez časových prodlev nebo animace vloží do plátna diagramu objekt typu *Node* a jemu příslušný popisek.

Animace pro vložení a odstranění vrcholu z plátna diagramu zajišťuje třída *FadeTransition* grafického rozhraní JavyFX. Rychlost animace je určena v objektu třídy *Preferences*. Animace pro vložení nebo odstranění vrcholu může být vypnuta a nahrazena časovou prodlevou, vypnutí animací určuje již zmíněný objekt třídy *Preferences*.

**class PlaceEdge** Instancí třídy PlaceEdge je zapouzdřený objekt operace vygenerování hrany diagramu. Povinný parametr pro vytvoření objektu PlaceEdge je pouze jeden – uspořádání *Order*. Třída je implementovaná podobně jako *PlaceVertex*, metody rozhraní *Command* fungují následovně:

**void forward()** Metoda pro uspořádání *Order* vytvoří grafickou nadstavbu *Edge*, kterou nastaví a pomocí animace vloží do plátna diagramu.

**void backward()** Metoda animuje odstranění hrany *Edge* z plátna diagramu.

**void instantForward()** Metoda bez časových prodlev nebo animace vloží do plátna diagramu hranu *Edge*.

Vložení a odstranění hrany z plátna diagramu je animováno pomocí třídy *Timeline* JavyFX. Animace může být vypnuta a rychlost, stejně jako při generování vrcholu, určuje objekt třídy *Preferences*.

**class MoveVertex** Instancí třídy MoveVertex je zapouzdřený objekt operace posunutí vrcholu v diagramu. Povinnými parametry pro vytvoření objektu jsou souřadnice nové polohy vrcholu *double x, double y* a vrchol *Vertex v*, který má být posunut. Metody jsou implementovány následovně:

**void forward()** Metoda nejprve uloží původní souřadnice vrcholu a potom vrchol $v$ přesune pomocí animace na nové souřadnice $x$, $y$.

**void backward()** Metoda přesune vrchol $v$ z nových souřadnic na souřadnice původní.

**void instantForward()** Metoda bez časových prodlev nebo animace uloží souřadnice a přesune vrchol $v$ na nové.

Pro přesouvání vrcholů diagramu je použita animace třídy *PathTransition*. Animace může být vypnuta a rychlost, stejně jako při generování vrcholů nebo hran, určuje objekt třídy *Preferences*.

**class MoveVertices** Instancí třídy MoveVertices je zapouzdřený objekt operace posunutí více vrcholů najednou. Třída se zpracováním podobá třídě MoveVertex, pro paralelní vykonání více posunů najednou je použita animace třídy *ParallelTransition*.

#### 4.5.3.3 Zprávy operací

Možnost přidělit operaci zprávu, která si při vykonávání vypíše v okně výstupu, poskytuje třída *CommandOutput*. Objekt třídy je při vytváření operace možné uvést jako nepovinný parametr. Třída obsahuje dva atributy, *forwardOutput* pro zprávu, která se vypíše při generování dopředu a *backwardOutput* pro zprávu, která se vypíše při generování zpět.

#### 4.5.3.4 Generace

Celý průběh generování je v programu reprezentován seznamem operací uvedených výše. K vytvoření seznamu jsem vytvořil třídu *Generator*. Objekt Generator si udržuje seznam operací, který je na počátku prázdný. Metody třídy umožňují vytvoří novou operaci (objekt) a přidat ji do seznamu.

Třída slouží jako „toolbox“ při vytváření metod pro generaci diagramu. Odkloňuje uživatele od manipulace s důležitými objekty, které operace vyžadují a při vytváření operací je dodává sama. Důležitou metodou třídy je metoda *pack()*, která vrací seznam operací.

#### 4.5.3.5 Invoker

Poslední důležitou části systému generace je *Invoker*. Jde o objekt, který vykonává operace uložené v seznamu, který mu poskytne objekt *Generator* metodou *pack()*. *Invoker* si udržuje pozici právě vykonávané operace a má k dispozici následující metody:

**void executeNext** Jestliže se index $i$ aktuální pozice v seznamu nerovná velikosti seznamu, pak se provede metoda *forward* operace na pozici $i$. Po dokončení metody *forward* se navýší $i$ o 1.

**void executePrevious** Jestliže je index $i$ aktuální pozice v seznamu větší než 0, pak se provede metoda *backward* operace na pozici $i$. Po dokončení metody *backward* se sníží $i$ o 1.

**void executeAuto** Invoker bude vykonávat metodu *executeNext* dokud se nenachází na konci seznamu operací nebo není pozastaven.

**void instantExecuteAll** Pro každou operaci ze seznamu se provede metoda *instantForward*.

Objekt je vytvořen při každé změně metody pro generaci svazu nebo načtení nového vstupního souboru. Metody objektu jsou spouštěny prvky uživatelského rozhraní (například tlačítko **Start** při kliknutí zašle zprávu hlavnímu ovladači, který spustí metodu *executeAuto*).

### 4.5.4 Metody generování diagramu svazu

V programu jsou obsaženy tři metody pro generaci diagramu svazu. Metodám odpovídají následující třídy:

**class LevelMethod** Třída implementující úrovňovou metodu generace diagramu.

**class GeometricMethod** Třída implementující geometrickou metodu generace diagramu.

**class ForceDirectedMethd** Třída implementující *Force-directed* metodu generace diagramu.

Každá z metod generace diagramu implementuje rozhraní *Method*, které specifikuje následující metody:

**interface Method**

**List<Command> generate(Generator g, OrderedSet o)** Metoda vrací seznam operací rozhraní Command, který je následně vykonáván objektem třídy *Invoker*. Seznam operací vytváří Generator metodou *pack()*.

**boolean pregeneratedDiagram()** Jestliže je nastaveno na true, pak se při zvolení metody náhodně umístí vrcholy vstupního souboru. Využití u metod, které požadují prvotní rozmístění prvku (Force-directed).

**enabledDDwhileGenerating()** Jestliže je nastaveno na true, pak je při generování diagramu povolen systém Drag & Drop.

#### 4.5.4.1 Rozšíření programu o další metody

Jedním z požadavků při vytváření programu byla možnost program dále rozšířit o vlastní metody. Pro rozšíření je nutné vytvořit třídu, která implementuje rozhraní *Method* a přidat ji ve třídě hlavního ovladače programu – *Presenteru*.

Příklad, jak by mohla třída vypadat je zobrazen ve zdrojovém kódu 1. Následné přidání třídy uvnitř metody *initializeMethods()*, která se nachází ve třídě *Presenter* je ukázáno v kódu 2. Pokud třída implementuje rozhraní Method, pak je přidána do programu a je pro ni automaticky vytvořena položka v menu všech metod.

### 4.5.5 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní programu je definováno soubory ve formátu FXML. Logickou část grafického rozhraní zastřešují tzv. *Controllery* – třídy, které delegují uživatelské podněty hlavnímu ovladači programu – *Presenteru*.

### 4.5.6 Hlavní ovladač programu

Hlavní ovládací třídu programu představuje *Presenter*. Na základě signálů z grafického rozhraní modifikuje data. Objekt se stará o chod programu, zakazuje a povoluje tlačítka, vytváří *Invokery* generací při změně vstupu nebo přepnutí metody generace, povoluje systém Drag & Drop a další.

### 4.5.7 Import souborů

Import souborů zajišťuje třída *Import*. Pro práci s XML soubory je použito objektové rozhraní DOM (Document Object Model). Soubory jsou při importování validovány vůči XSD schématu, které je uloženo v souboru XMLSchema.xsd. Zdrojový kód schématu je zobrazen na 3.

```
1   public class ExampleMethod implements Method {
2
3       @Override
4       public List<Command> generate(Generator g, OrderedSet o) {
5         double x = 10;
6         double y = 10;
7
8         // vygenerovani vrcholu vsech prvku vstupniho souboru na
              pozice <x, y>
9         for (Vertex vertex : o.getVertices()){
10          g.placeVertex(vertex, x, y);
11          x+=10;
12          y+=10;
13        }
14
15        // posunuti vrcholu nejmensiho prvku na pozici <20, 20> a
              vrcholu nejvetsiho prvku na pozici <40, 40>
16        g.moveVertex(o.getLeastElement(), 20, 20);
17        g.moveVertex(o.getGreatestElement(), 40, 40);
18
19        // vygenerovani hran pro vsechny usporadani
20        for (Order order : o.getOrders()){
21          g.placeEdge(order);
22        }
23
24        return g.pack();
25      }
26
27      @Override
28      public boolean pregeneratedDiagram() {
29        return false;
30      }
31
32      @Override
33      public boolean enabledDDwhileGenerating() {
34        return false;
35      }
36  }
```

Zdrojový kód 1: Příklad kódu třídy vlastní metody.

```
1     private void initializeMethods(){
2       // metody, ktere jsou jiz implementovane
3       addMethod(LevelMethod.class, "Level Method");
4       addMethod(GeometricMethod.class, "Geometric Method");
5       addMethod(ForceDirectedMethod.class, "Force-directed Method");
6       // pridana metoda
7       addMethod(ExampleMethod.class, "Příkladová Metoda");
8
9       checkMethodsCorrectness();
10      layoutController.initializeMethodToggleGroup();
11    }
```

Zdrojový kód 2: Přidání metody uvnitř třídy Presenter.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="OrderedSet">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vertices">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Vertex" maxOccurs="unbounded" minOccurs
                  ="1">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="name" use="
                          required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Orders">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Order" maxOccurs="unbounded" minOccurs
                  ="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="from" use="
                          required"/>
                      <xs:attribute type="xs:string" name="to" use="
                          required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="name"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Zdrojový kód 3: XSD Schéma pro XML soubory.

# Závěr

V této práci jsem se zaměřil na problematiku automatického generování diagramu svazu. Existuje několik programů, které umožňují diagram vygenerovat, ale žádný neumí vizualizovat průběh generování a tím přiblížit uživateli jakým způsobem automatická generace probíhá.

Proto jsem vytvořil aplikaci LatDra, která graficky demonstruje průběh generování diagramu a dovoluje jej neomezeně krokovat dopředu i dozadu. Aplikace umožňuje generovat diagram třemi metodami – úrovňovou, geometrickou a Force-directed metodou a je možné jej rozšířit o metody další. Navíc dovoluje exportovat libovolný krok generování do formátu TikZ, měnit vzhled diagramu nebo okamžitě přeskočit na konec generace.

# Conclusions

There are several programs for automatic Hasse diagram generation, but none of them allow visualization of the process of generation. And so I have created such application called Latdra which can graphically demonstrate the process of diagram creation from beginning to end with three implemented methods – level method, geometric method and Force-directed method. LatDra allows you to undo, redo of each process step, export each step into a TeX file, change diagram visual parameters or create diagram without visualization process.

# A Obsah přiloženého CD/DVD

**bin/**
: Obsahuje spustitelný java soubor LatDra a doplňkové soubory.

  **LatDra.jar**
  : Spustitelný soubor aplikace LatDra.

  **examples/**
  : Složka s ukázkovými vstupními soubory ve formátu XML.

  **ext/**
  : Obsahuje LatDra ExternalDraw.jar – sadu řádkových programů pro externí komunikaci s JLatVis.

  **XMLSchema.xsd**
  : Schéma definující XML strukturu vstupních souborů.

**doc/**
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

**src/**
: Kompletní zdrojové texty programu LatDra.

**readme.txt**
: Instrukce pro spuštění programu LatDra, včetně všech požadavků pro jeho bezproblémový provoz.

# Literatura


[1] DI BATTISTA, Giuseppe. Graph drawing: algorithms for the visualization of graphs. Upper Saddle River, N.J.: Prentice Hall, 1999, 397 p. ISBN 0133016153.

[2] FOWLER, Martin. GUI Architectures [online]. 2006-06-18. [cit. 2015-07-22]. Dostupné z: http://martinfowler.com/eaaDev/uiArchs.html.

[3] GREER, Derek. Interactive Application Architecture Patterns [online]. 2007-08-25. [cit. 2015-07-22]. Dostupné z: http://aspiringcraftsman.com/2007/08/25/interactive-application-architecture/.

[4] CHAJDA, Ivan. Algebra III. Skriptum, Univerzita Palackého v Olomouci, 1998.

[5] KANDR, Michal. JLatVis [online]. 2011-02-06. [cit. 2015-07-30]. Dostupné z: http://www.jlatvis.com/cs/.

[6] OUTRATA, Jan. Drawing lattices with a geometric heuristic. In: Yager R. R., Sgurev V. S., Jotsov V. S. (Eds.): Proceedings of IEEE CIS 2008: The Fourth International IEEE Conference on Intelligent Systems, 2008, pp. 15-35–15-41.

[7] OUTRATA, Jan. Algoritmy pro kreslení uspořádaných množin a svazů. Diplomová práce, Univerzita Palackého v Olomouci, 2003.

[8] SKALSKÁ, Dagmar. Algebra. Skriptum, Univerzita Palackého v Olomouci, 2006.

[9] VEČERKA, Arnošt. Grafy a grafové algoritmy. Skriptum, Univerzita Palackého v Olomouci, 2004.

[10] Java SE8 Documentation : dokumentace programovacího jazyka Java SE8 [online]. [cit. 2015-07-29]. Dostupné z: http://docs.oracle.com/javase/8/javase-clienttechnologies.htm.

[11] Welcome to the ToscanaJ Suite : oficiální web k Toscana J [Online]. [cit. 2015-07-29]. Dostupné z: http://toscanaj.sourceforge.net/ .