České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Tomáš Tušla**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Vizualizace prerekvizit předmětů**

Pokyny pro vypracování:

Proveďte analýzu knihoven a algoritmů zaměřených na vizualizaci grafů. Zaměřte se zejména na automatické rozmisťování uzlů grafů v 2D prostoru. Na základě analýzy navrhněte a implementujte webovou aplikaci, která umožní vizualizovat předměty a jejich prerekvizity ve formě grafu (podobně jako na http://stm.fel.cvut.cz/prerekvizity). Vstupem pro aplikaci bude tabulka obsahující pro každý předmět jeho prerekvizity. Výslednou aplikaci otestujte na datech o předmětech vyučovaných na FEL ČVUT.

Seznam odborné literatury:

Roberto Tamassia (editor), Handbook of Graph Drawing and Visualization, CRC Press, 2013.

Vedoucí: Ing. Ladislav Čmolík, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

[redacted]

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

[redacted]

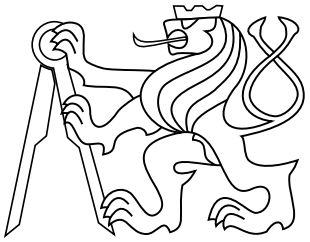
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 3. 2015

bakalářská práce

Vizualizace prerekvizit předmětů

Tomáš Tušla



Květen 2015

Ing. Ladislav Čmolík, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra Počítačů

Poděkování

Rád bych poděkoval Ing. Ladislavu Čmolíkovi Ph.D. za připomínky a rady při tvorbě této bakalářské práce.

Prohlášení

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

V Praze dne 22.5.2015 .....

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací webové aplikace umožňující automatickou tvorbu grafů se studijními závislostmi mezi předměty. Obsahuje analýzu a zhodnocení algoritmů na rozvržení uzlů grafu. Dále obsahuje zhodnocení grafových knihoven, které by bylo možné použít k implementaci. Existující knihovny ale nejsou schopny vykreslovat hierarchické grafy s více uzly na stejné úrovni a často nekontrolují překrytí popisků. K řešení těchto problémů jsem navrhl novou knihovnu, poskytující implementaci upraveného algoritmu hierarchického rozvržení. Také jsem implementoval webovou aplikaci využívající navrženou knihovnu k automatickému generování grafů závislostí mezi předměty. Výsledné grafy jsem vyhodnotil a porovnal proti jiným řešením. V porovnání se zbylými knihovnami poskytuje navržená knihovna znatelně lepší výsledky.

Klíčová slova

Rozvrhovací algoritmy grafů; Webová aplikace; Hierarchické rozvržení; Grafové knihovny

Abstrakt

This bachelor thesis deals with the design and implementation of web application allowing automatic creation of graphs with study dependencies between courses. It contains analysis and evaluation of graph layout algorithms. Furthermore, it comprises evaluation of graph libraries which could be used for implementation. But existing libraries are not able to render hierarchical graphs with multiple nodes on the same level, and often do not check overlaps of labels. To address these issues, I designed a new library, providing the implementation of the modified hierarchical layout algorithm. I also implemented a web application that uses the designed library to automatically generate graphs of dependencies between courses. I evaluated the resulting graphs and compared them with other solutions. In comparison to the remaining libraries, results obtained with designed library are noticeably better.

Keywords

Graph layout algorithms; Web application; Hierarchy Layout; Graph libraries

Obsah

1 Úvod **1**
- 1.1 Motivace . . . 1
- 1.2 Cíle . . . 2
- 1.3 Struktura práce . . . 2

2 Analýza aplikace **3**
- 2.1 Požadavky . . . 3
- 2.2 Uživatelské role . . . 3
  - 2.2.1 Host . . . 4
  - 2.2.2 Administrátor . . . 4
- 2.3 Doménový model . . . 5

3 Analýza algoritmů **7**
- 3.1 Algoritmy na rozmístění uzlů grafu . . . 7
- 3.2 Možné grafové algoritmy . . . 7
  - 3.2.1 Rozložení do stromu . . . 8
  - 3.2.2 Hierarchické rozvržení . . . 9
  - 3.2.3 Dominance drawing layout . . . 10
  - 3.2.4 Silově řízené rozvržení . . . 11
- 3.3 Srovnání algoritmů . . . 12

4 Knihovny na zobrazení grafů **15**
- 4.1 Porovnávané knihovny . . . 15
  - 4.1.1 Gephi . . . 15
  - 4.1.2 JGraph . . . 16
  - 4.1.3 JUNG . . . 16
  - 4.1.4 GraphStream . . . 16
  - 4.1.5 Prefuse . . . 16
  - 4.1.6 Grappa . . . 17
  - 4.1.7 Porovnání knihoven . . . 17
- 4.2 Řešení s pomocí knihovny JGraph . . . 18

5 Návrh knihovny **21**
- 5.1 Změny oproti JGraph . . . 21
- 5.2 Popis rozvrhovacího algoritmu . . . 21
- 5.3 Podrobnější rozbor minimalizace křížení . . . 26

6 Webová aplikace **29**
- 6.1 Použité technologie . . . 29
- 6.2 Využívané systémy třetích stran . . . 30
- 6.3 Architektura . . . 31
  - 6.3.1 Vrstva zdrojů dat . . . 31
  - 6.3.2 Servisní vrstva . . . 32
  - 6.3.3 Zobrazovací vrstva . . . 32
- 6.4 Požadavky a omezení vyplývající ze vstupních dat . . . 33

7 Testování **35**
- 7.1 Automatické testování . . . 35

7.2 Manuální testování 36

8 Vyhodnocení výsledků **37**
8.1 Výsledné grafy 37
8.1.1 Porovnání s JGraph 37
8.1.2 Porovnání s ručně vytvořenými grafy 38
8.1.3 Známé vady 41
8.2 Webová aplikace 42
8.3 Možná vylepšení 43
8.3.1 Grafový algoritmus 43
8.3.2 Webová aplikace 43

9 Závěr **45**

Přílohy

A Manuál k nasazení **47**
A.0.3 Inicializace databáze 47
A.0.4 Konfigurační soubory aplikace 47
A.0.5 Konfigurace administrátorského účtu 47
A.1 Konfigurace testovacího prostředí 48

B Obsah CD **49**

Literatura **51**

Zkratky

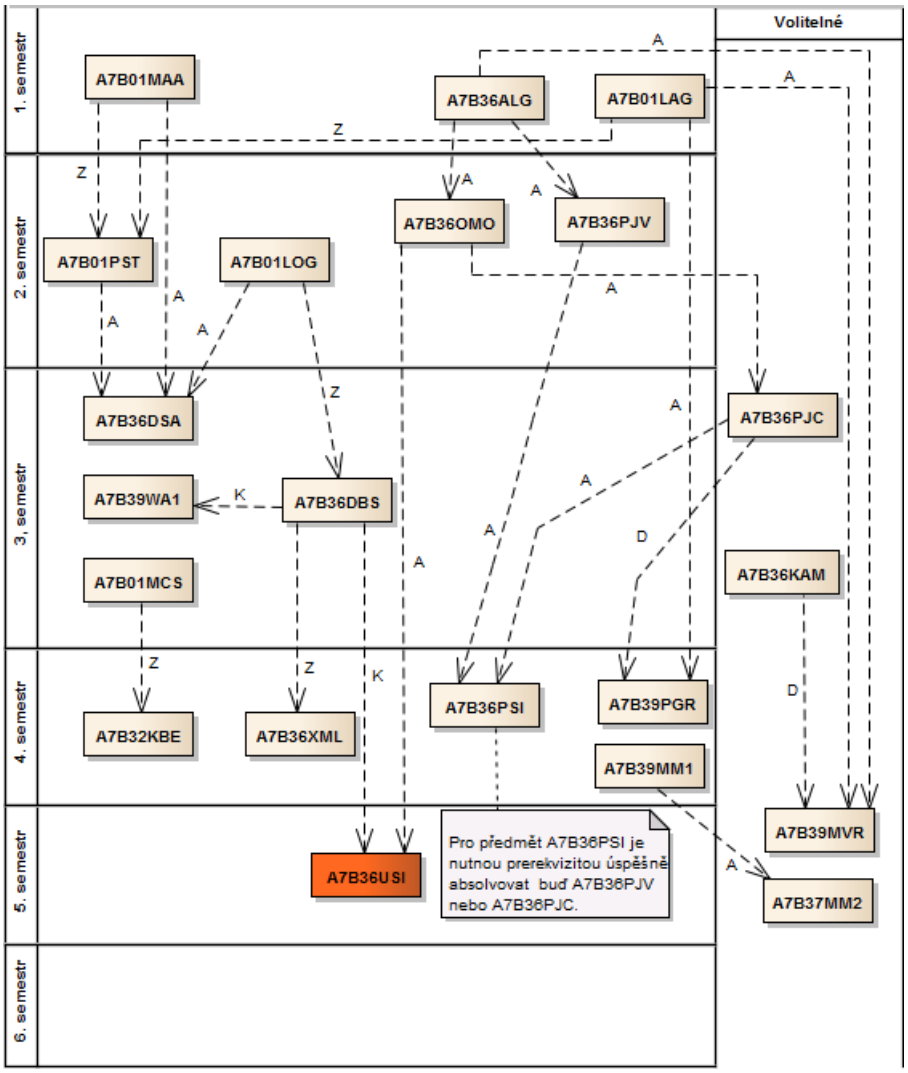
ČVUT	České vysoké učení technické v Praze
KOS	Komponenta studium
FEL	Fakulta elektrotechnická
SVG	Scalable Vector Graphics
GUI	Graphical user interface
API	Application Programming Interface
CSS	Cascading Style Sheets
XML	EXtensible Markup Language
SQL	Structured Query Language
HQL	Hibernate Query Language
ORM	Object-relational mapping
JSF	JavaServer faces
Java EE	Java Platform, Enterprise Edition
REST	Representational state transfer
HTTP	Hypertext Transfer Protocol
xls	Microsoft Excel Spreadsheet
DAO	Data access object
DOM	Document Object Model
HTML	HyperText Markup Language
XHTML	EXtensible HyperText Markup Language
CRON	Command Run On

1 Úvod

Každý student ČVUT musí při studiu splnit řadu předmětů dle svého studijního plánu. Některé předměty rozšiřují znalosti, které měli studenti získat v rámci jiného předmětu. Vznikl proto systém prerekvizit, tedy návazností mezi předměty, který udává pořadí, ve kterém si studenti zapisují jednotlivé předměty. Účelem je zajistit, aby studenti měli všechny nutné vstupní znalosti k absolvování předmětů.

1.1 Motivace

Pokud chce student zjistit, jaké má prerekvizity v rámci svého studijního programu, musí vyhledávat prerekvizity přímo na stránkách jednotlivých předmětů, nebo v systému KOS po jednotlivých předmětech. To však není příliš názorné.



Obrázek 1 Graf prerekvizit [29]

Zobrazení prerekvizit jako grafu (viz Obrázek 1) dá studentům lepší přehled, jak bude jejich studium probíhat. Student také snadno zjistí, v jakých předmětech mu případný neúspěch prodlouží studium a je tedy důležitější zaměřit se na jejich absolvování.

V současné době však neexistuje automatický systém na generování grafů prerekvizit předmětů pro jednotlivé průchody studiem. Pokud je potřeba takový graf vytvořit, musí být vytvořen ručně. Bylo by tedy vhodné vytvořit systém, který dokáže sám zpracovat potřebná data a vytvořit dané vizualizace.

1.2 Cíle

Cílem bakalářské práce je vytvořit systém, který provede tvorbu grafu prerekvizit automaticky. Tedy nalezne, které studijní programy se na fakultě vyučují, jejich jednotlivá zaměření a předměty. Následně z předmětů, které si student daného studijního plánu může zapsat, a návazností předmětů vytvoří výsledný graf. Sadu takto vytvořených grafů zveřejní na webových stránkách, kde si tyto grafy mohou zájemci prohlédnout, případně získat odkaz pro vložení na vlastní webové stránky.

1.3 Struktura práce

Text je rozdělen do devíti kapitol. Druhá kapitola Analýza aplikace je zaměřena na popis požadovaných vlastností aplikace, její uživatele a požadavky. V třetí kapitole Analýza algoritmů jsou porovnány jednotlivé algoritmy, které by bylo možné použít jako základ pro vizualizaci grafů. Čtvrtá kapitola Analýza knihoven obsahuje popis jednotlivých knihoven, které je možné pro vizualizaci použít. Obsahem páté kapitoly Návrh knihovny je návrh vlastní knihovny na vizualizaci grafů. V šesté kapitole Webová aplikace je popsána implementace aplikace. Sedmá kapitola se zabývá testováním aplikace. Osmá kapitola Vyhodnocení výsledků prezentuje vzniklé grafy, porovnává je a navrhuje možné vylepšení aplikace. V poslední kapitole Závěr jsou shrnuty výsledky práce.

2 Analýza aplikace

Obsahem této kapitoly je definice požadavků, které musí vzniklá aplikace splňovat. Dále jsou uvedeni uživatelé, kteří budou aplikaci využívat a jejich uživatelská práva. Posledním bodem je doménový model aplikace.

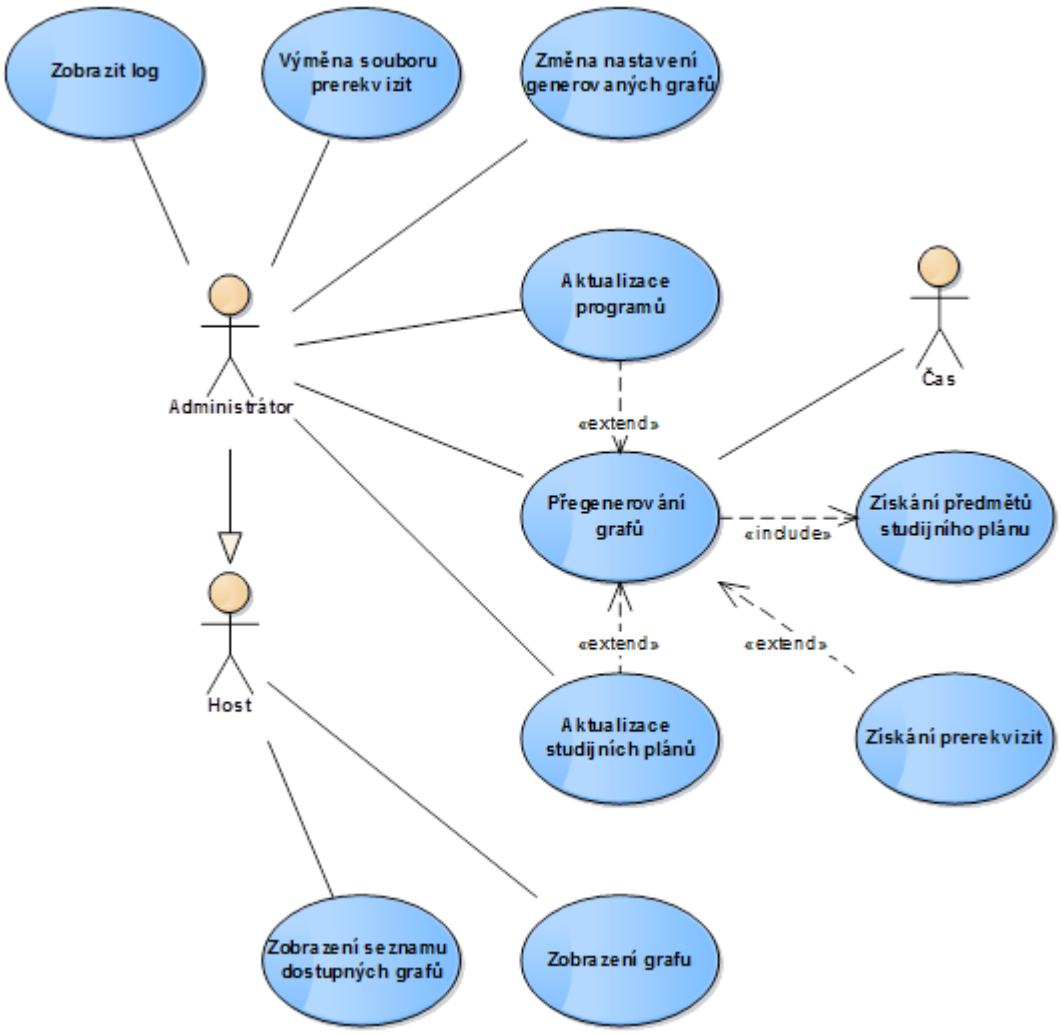
2.1 Požadavky

Aplikace musí splňovat následující požadavky:

- Systém umožní vytvořit graf prerekvizit pro jednotlivé studijní plány.
- Systém bude k načítání existujících prerekvizit využívat vložený xls soubor.
- Systém umožní nahrát xls soubor se seznamem prerekvizit přes webové rozhraní.
- Systém umožní zobrazit vytvořené grafy uživatelům.
- Systém bude automaticky zjišťovat existující studijní programy, jejich studijní plány a předměty ze systému KOSapi[22].
- Systém bude provádět automatickou aktualizaci grafů.
- Systém umožní administrátorovi provést okamžité přegenerování grafů.
- Systém umožní administrátorovi vybrat, pro které programy bude systém generovat grafy.
- Systém umožní administrátorovi vybrat jednotlivé studijní plány, pro které bude systém generovat grafy.

2.2 Uživatelské role

Vzhledem k povaze systému jsou zamýšlenými uživateli jednak současní studenti ČVUT FEL, ale také potenciální studenti, kteří se dle rozložení předmětů mohou rozhodovat o studiu. Běžná funkčnost systému je tedy veřejně přístupná, bez potřeby být přihlášen. Pro potřeby správy aplikace je vytvořena role administrátora.



Obrázek 2 Diagram případů užití

Uživatelské role a práva jednotlivých uživatelů znázorňuje Obrázek 2. Aplikace rozeznává dva druhy uživatelů, hosta a administrátora. Další položkou je automatické volání aplikace v nastavených časech, které provede aktualizace grafů dle nastavení.

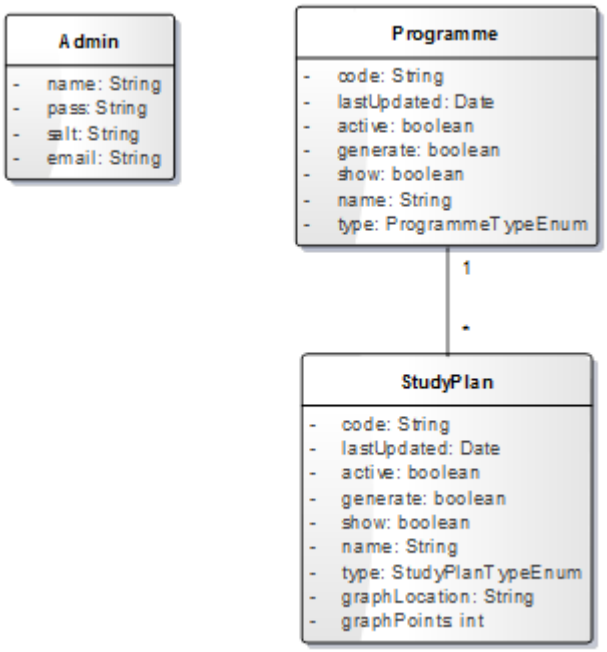
2.2.1 Host

Nepřihlášený uživatel, případně externí systém, jehož cílem je nechat si zobrazit graf prerekvizit. Tento uživatel si má možnost zobrazovat pouze grafy ze seznamu všech zveřejněných grafů.

2.2.2 Administrátor

Přihlášený uživatel spravující aplikaci. Má právo nahrát nový xls soubor s prerekvizitami. Může také vynutit přegenerování grafů, případně aktualizaci informací ze systému KOS. Dále může vybírat, které studijní plány má systém ignorovat při vytváření grafů. V případě potřeby může také prohlížet log aplikace, aby zkontroloval správnou funkčnost aplikace.

2.3 Doménový model



Obrázek 3 Doménový model

Doménový model, jak ho zachycuje Obrázek 3 obsahuje tři entity. První entitou je entita Admin, která uchovává informace o uživatelských účtech, které mají možnost spravovat aplikaci. Další entity Programme a StudyPlan slouží jako úložiště nastavení pro jednotlivé studijní programy a studijní plány. Relace mezi Programme a StudyPlan vyjadřuje propojení mezi programem, například Otevřená informatika, a jeho studijními plány, například Otevřená informatika - Počítačové systémy. Studijní plán je seznamem předmětů, které je nutné splnit k dokončení daného studijního programu. Aplikace si však jednotlivé předměty u daných studijních plánů neukládá.

3 Analýza algoritmů

V této kapitole jsou definovány požadavky na výsledný vzhled grafu a dle těchto požadavků jsou hodnoceny různé existující algoritmy pro automatické rozmístění uzlů grafu.

3.1 Algoritmy na rozmístění uzlů grafu

Jednotlivé algoritmy lze rozlišovat dle typu grafu, buď orientovaného, nebo neorientovaného. Pro potřeby aplikace se hodí algoritmy pracující s orientovanými grafy, které umožní vykreslit posloupnost předmětů. Vybraný algoritmus by měl splňovat následující podmínky:

- Algoritmus umožňuje více vstupních a výstupních hran z jednoho uzlu.
- Algoritmus umožňuje více kořenů.
- Algoritmus umožňuje znázornit v jakém semestru je předmět vyučován. Musí tedy být možné vytvořit vizuálně jasně oddělitelné vrstvy pro jednotlivé semestry. Předměty reprezentované jednotlivými uzly se pak přiřadí do jim odpovídajících vrstev.
- Algoritmus umožňuje vytvořit hranu mezi dvěma uzly grafu na větší vzdálenost, než sousední vrstvy. Umožní se tak například vykreslit závislost mezi předmětem zapisovaném v druhém semestru a ve čtvrtém semestru. Pokud algoritmus s touto variantou nepočítá, může dojít k deformaci grafu nebo špatnému vykreslení, například hrana vedoucí přes uzel.
- Algoritmus minimalizuje množství křížení mezi jednotlivými hranami za účelem zlepšení přehlednosti.

Dále je potřeba brát speciální požadavky vycházející ze vztahu typu korekvizita. Tento vztah znamená, že požadovaný předmět nemusí být zapsán či splněn v předchozím semestru, ale může být zapsán až zároveň s požadujícím předmětem. Tento typ vztahu vytváří další požadavky.

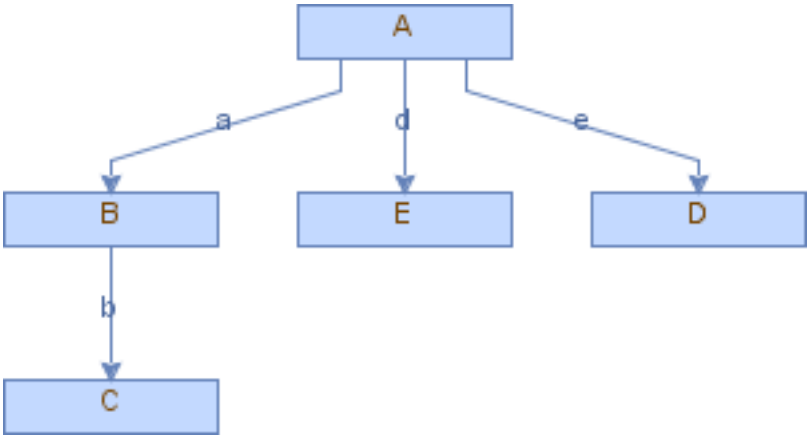
- Algoritmus umožňuje vytvořit hranu mezi dvěma uzly grafu ve stejné vrstvě.
- V rámci jedné vrstvy může vzniknout cyklická návaznost předmětů. Tento stav nastává, pokud jsou dva či více předmětů ve vzájemném vztahu korekvizit.

3.2 Možné grafové algoritmy

Algoritmů na rozvržení grafů je velké množství. Z výběru byly předem vyřazeny takové algoritmy, které rozmístí uzly a hrany grafu do sice zajímavé kompozice, ale za cenu čitelnosti ať už celkové struktury, nebo jednotlivých závislostí mezi uzly grafu. I přes množství druhů algoritmů se dá očekávat, že některé požadavky nebudou jednotlivými algoritmy splněny. Důležitým měřítkem algoritmu je tedy potřebné množství zásahů do implementace, aby bylo možné algoritmus využít v rámci vytvářené aplikace. Ve výsledku tedy byly vybrány čtyři algoritmy, které jsou podrobněji rozebrány z hlediska jejich výhod a nevýhod.

3.2.1 Rozložení do stromu

Rozmístění uzlů pro grafy stromové struktury. Graf tedy musí splňovat podmínku, že má jeden kořen a mezi každou dvojicí uzlů existuje pouze jedna cesta. Klasické rozmístění je dle vzdálenosti od kořene. Úroveň uzlu grafu udává vzdálenost od kořene.



Obrázek 4 Stromové rozvržení

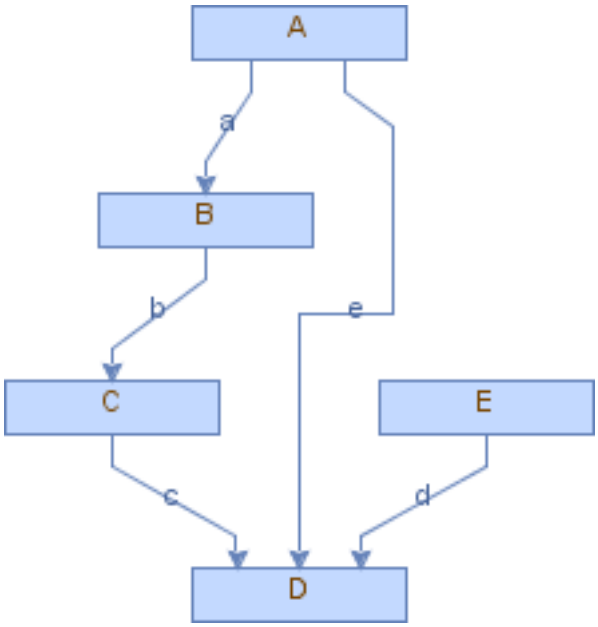
Způsobů na vykreslení stromové struktury je mnoho[5]. Jedním z vhodných ukázkových algoritmů je Reingold-Tilfordův algoritmus[26]. Ten spočívá v rekurzivní stavbě stromu odspoda. Implementace tohoto algoritmu je velmi jednoduchá. V jednotlivých krocích se pravá větev nejdříve přiblíží k levé a následně se nad obě větve vloží rodič daných větví a spojí obě větve do jedné. Takto se postupně budují větve odspoda, dokud nedojde k vytvoření celého stromu. Výsledek algoritmu zachycuje Obrázek 4.

Rozložení do stromu má řadu nedostatků, prvním z nich je, že prerekvizity předmětů netvoří strom. Tato podmínka je porušena v několika ohledech. Prvním porušením stromové struktury je existence více kořenů. Tento problém by byl řešitelný vytvořením více stromů a následným spojením do jednoho grafu. Problém zde však může nastat ve chvíli, kdy se v grafu nachází hrana, která dvojici takových stromů spojuje. Projeví se tak další nedostatek stromového rozvržení a to existence více vstupních hran do některého z uzlů, což stromové rozvržení neumožňuje. Bylo by tedy nutné vytvořit graf bez takové hrany a po dokončení algoritmu takovéto hrany doplňovat. Tento přístup však přináší další problémy, neboť by bylo nutné zajistit, aby byly spojované uzly obou stromů blízko u sebe. Tento nedostatek rozvržení je těžce řešitelným, jelikož algoritmus nemá způsob jak vyjádřit potřebu blízkosti uzlu jinak, než samotnou spojující hranou. Problémem algoritmu se stává i to, že nativně nepodporuje dvojici uzlů na stejné úrovni. Muselo by tedy docházet k následnému přidávání takovýchto uzlů. Stejně tak má algoritmus problém s vynecháním úrovní mezi dvojicí uzlů, tento problém je však možné řešit doplněním speciálních uzlů. Poslední nevýhodou stromového rozložení je existence cyklických závislostí ve vstupních datech. Tento problém je možné řešit vytvořením jednosměrné vazby a po vytvoření rozložení přidat vazbu v opačném směru, případně v rámci zobrazení nahradit původní hranu hranou se speciálním stylem.

Je patrné, že stromové rozložení má velké množství nedostatků, je ale také nutné brát v potaz, že je snadné na implementaci. Část nedostatků vyplývá ze specifických nároků na výsledný algoritmus. Většina algoritmů je postavena na jednoduchých a snadno pochopitelných principech. Proto požadavek na dělení uzlů do vrstev dle návazností jejich hran a následné narušení tohoto pravidla umožněním mít dvojici uzlů propojených hranou na stejné vrstvě zpravidla nebývá běžnými algoritmy podporováno.

3.2.2 Hierarchické rozvržení

Rozvržení tvořící podobné rozložení, jako stromové rozvržení. Na rozdíl od stromového rozvržení však může mít uzel libovolný počet rodičů, tedy vstupních hran. Každý uzel má přiřazenu úroveň, ve které se nachází. Snahou je dodržet řadu pravidel[1]. Hrany by měly mířit stejným směrem, být krátké, rovné a s co nejmenším počtem křížení. Zároveň by měly být uzly rovnoměrně rozprostřené po grafu. Jelikož algoritmus počítá s tím, že hrana může být přes několik úrovní, vytvářejí se na jednotlivých úrovních virtuální uzly, které zajistí, že bude při rozvrhování místo pro spojovací hranu. Tím však rychle narůstá počet uzlů grafu.



Obrázek 5 Hierarchické rozvržení

Přístupů k tvorbě hierarchických grafů je více. Většinou vycházejí z nějakého základního rámce, který nějakým způsobem upravují [2]. Tyto upravené algoritmy pak mají lepší vlastnosti pro určitý druh grafů, případně považují splnění některých estetických vlastností za podstatnější. Příkladem zde může být snaha minimalizovat počty ohybů hran, což je často vykoupeno zvýšenými nároky na celkovou šířku grafu. Příklad snížení šířky i za cenu více ohybů ukazuje Obrázek 5.

Nejčastější implementace hierarchického rozvržení vycházejí z Sugiyamova frameworku[1], který udává základní kroky pro tvorbu hierarchického rozvržení a navrhuje přidání dalších volitelných kroků. Algoritmus dle Sugiyamova frameworku má čtyři až sedm základních kroků:

- Nejprve se ze vstupního grafu odstraní existující orientované cykly. Hrana způsobující cyklus se buď otočí, nebo se odstraní a na konci algoritmu opět vrátí. Tento krok je volitelný, ale je nutné ho provést, pokud se v datech mohou cykly vyskytovat.
- Následně dochází k rozvržení vrstev pro jednotlivé uzly. Uzly jsou přiřazeny na nejnižší možnou vrstvu tak, aby se dodržela podmínka, že hrany směřují jedním směrem, tedy výstupní hrana vede vždy na další vrstvu.
- Pokud vede nějaká hrana přes více vrstev, je pro každou vrstvu vytvořen pomocný uzel a původní hrana je nahrazena hranami vedoucími přes pomocné uzly.

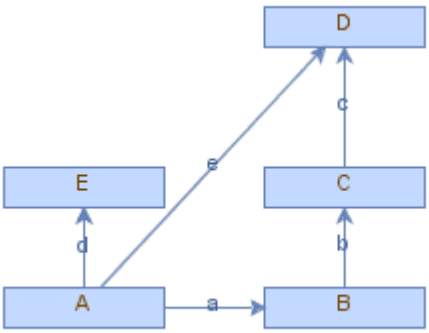
- Provede se minimalizace pomocných uzlů. Pokud lze uzel z nižší vrstvy nahradit za pomocný uzel ve vyšší vrstvě tak, aby nedošlo k poruše nějakého pravidla, je tento uzel nahrazen. Tento krok je volitelný, ale pomáhá snížit celkový počet uzlů a délku hran.
- Další volitelný krok je koncentrace hran. Její princip spočívá v tom, že pokud má skupina uzlů stejné rodiče, lze nad tuto skupinu umístit uzel, který je zastřešuje. Rodičovské uzly, které tak původně měly mnoho hran na uzly dané skupiny, mohou mít jedinou hranu na daný zastřešující uzel. Výhodou je snížení počtu hran a křížení, nevýhodou pak je změna struktury grafu a případná změna výšky grafu.
- V další fázi se provádí minimalizace křížení hran grafu. Tento problém patří mezi NP složité. Využívá se proto heuristik. Původní algoritmus byl navržen s využitím barycenterické heuristiky, která umisťuje uzel na středovou pozici mezi potomky.
- V poslední fázi dochází k přiřazení samotných souřadnic pro jednotlivé uzly. Souřadnice jsou přiřazeny tak, aby bylo zachováno pořadí uzlů v jednotlivých vrstvách, které bylo určeno předchozím krokem algoritmu.

Hlavní nevýhodou algoritmu je výpočetní náročnost a složitost implementace. Z důvodu výpočetní náročnosti algoritmu se využívá řada heuristik, ty mohou mít velký vliv na výsledné grafy. Jednotlivé implementace tak mohou lépe fungovat pro různé typy grafů a naopak selhávat při jiných vstupních datech. Obecně vysoká složitost implementace hraje roli z hlediska nároků na kvalitu návrhu a také vyšší rizika vzniku chyb. Mezi nedostatky samotného algoritmu se pak řadí opět nepřipravenost algoritmu na dvojici uzlů na stejné úrovni. Bylo by tedy nutné zasáhnout do algoritmu ve fázi úvodního rozřazení úrovní mezi uzly a rozřadit uzly do úrovní dle jejich výukových semestrů. To samozřejmě povede k nutnosti upravit i následující kroky algoritmu.

Hierarchické rozvržení řeší velkou část problémů, se kterými se potýkalo stromové rozvržení. Cenou za tento specifičtější algoritmus je ale větší složitost. Algoritmus má také proměnlivou spolehlivost na základě způsobu implementace jednotlivých kroků.

3.2.3 Dominance drawing layout

Rozložení pro orientované grafy bez tranzitivních hran. Jednotlivé uzly jsou rozmisťovány od rodičovského uzlu vždy směrem doprava a nahoru v souřadnicové mřížce.



Obrázek 6 Dominance drawing layout

Algoritmus, tak jak je popsán [7] má na vstupu uzel a z něj vystupující hrany. Algoritmus má v principu následující kroky:

- Nejprve musí být odstraněny tranzitivní hrany.

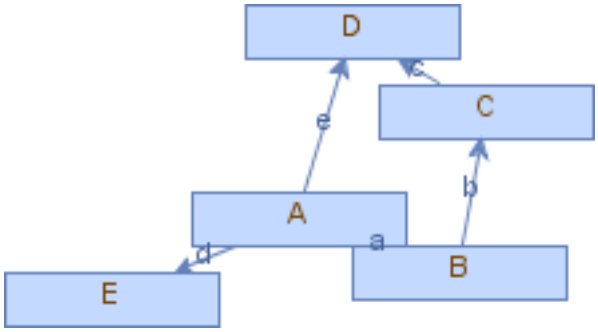
- Jakmile je graf připraven, seřadí se pro každý uzel grafu jeho výstupní hrany ve směru hodinových ručiček.
- Následně dojde k označení uzlů zleva. Označení probíhá zvyšující se číselnou řadou. Vždy se označí daný uzel a následně se pokračuje s rekurzivním označením větve jeho nejlevějšího potomka, jakmile je větev potomka označena, pokračuje se na další větev v pořadí.
- Jakmile jsou označeny všechny uzly zleva, provede značení zprava. Jediným rozdílem je postup od pravých potomků k levým.
- Po dokončení značení lze použít označení jako souřadnice pro vykreslení jednotlivých uzlů, jak zobrazuje Obrázek 6.

Hlavním problémem tohoto algoritmu je seřazení potomků jednotlivých uzlů. Vytvoření tohoto setřídění je z principu vytvoření optimálního rozmístění uzlů z třetí fáze algoritmu pro hierarchické rozložení. Algoritmus také nemá řešení pro přiřazení uzlů do daných úrovní. Podobně jako u stromového rozvržení je úroveň dána čistě vzdáleností od kořene dané části grafu. Oproti hierarchickému rozvržení je zde také problém s delšími hranami, algoritmus sice počítá s tím, že takové hrany existují, neprovádí však na základě dlouhých hran optimalizaci pozice uzlu. Protože algoritmus nepoužívá žádný systém na vytvoření místa pro dlouhé hrany, může být také problém s jejich samotným vykreslením, které nemusí probíhat vždy ideálně. Upravená verze algoritmu však místo odstranění tranzitivních hran tyto hrany převádí pomocí pomocných uzlů, stejně jako v případě hierarchického rozvržení.

Rozvržení je do jisté míry odlehčenou verzí hierarchického rozvržení. Bohužel, vzhledem k požadavkům aplikace patří chybějící části oproti hierarchickému rozvržení mezi ty, které byly žádoucí a minimálně některé části hierarchického rozvržení by tedy bylo nutné zpět doimplementovat.

3.2.4 Silově řízené rozvržení

Algoritmus v principu velmi odlišný od ostatních možných algoritmů. Funguje na fyzikálních principech přitažlivých a odpudivých sil mezi částicemi[4]. Mezi každými dvěma uzly grafu působí odpudivá síla, která vytváří mezery mezi uzly. Zároveň hrany mezi uzly grafu působí jako přitažlivé síly, které podle síly hrany vyvažují odpudivé síly a seskupují uzly blíže k sobě.



Obrázek 7 Silově řízené rozvržení

Algoritmus v jednotlivých krocích iteruje přes všechny uzly grafu a spočítá síly působící na daný uzel. Na základě výsledného vektoru posune daný uzel do nové pozice. Takto algoritmus postupuje, dokud se nedostane celý graf do rovnovážného stavu, kdy v iteraci nedochází ke změnám větším, než je nastavená hranice. Výsledný stav takového běhu algoritmu zachycuje Obrázek 4. Samotný algoritmus je velmi jednoduchý,

nabízí se však řada možností, jak upravit různé vlastnosti algoritmu. Jednotlivým hranám je možné přiřazovat různé váhy. Pohyb uzlů může být ovlivněn různými způsoby. Uzlům mohou být přiřazeny omezující kritéria. Ovlivnit se dají také samotné průběhy funkcí na výpočet působících sil.

Algoritmus netvoří uspořádané rozložení grafu. Bylo by tedy potřeba uměle vytvořit vrstvy tím, že se jednotlivým uzlům přiřadí omezení pohybu, které je udrží na úrovni jejich semestru. Bohužel výsledný vzhled také závisí na úvodní pozici uzlů v grafu, tento nedostatek by byl navíc ještě prohlouben právě zafixováním úrovně daného uzlu. Algoritmus se špatně nastavenými vahami nemusí být schopen dostat uzel začínající na druhé straně grafu na správnou stranu, neboť bude přetlačován moc velkým množstvím ostatních uzlů. Kvalita výsledného grafu by se tedy mohla v různých případech velice lišit. Neexistence vrstev v základním algoritmu také znamená, že v případě dvou uzlů spojených hranou, které jsou od sebe vzdáleny o více než jednu vrstvu, může výsledná hrana vizuálně procházet jiným uzlem. Algoritmus tedy může vrátit dobré rozvržení z hlediska pozic uzlů, ale nevhodné z hlediska vykreslování.

Silově řízený algoritmus je velice zajímavý z hlediska možností jeho úprav. Oproti ostatním algoritmům je velice rozdílný, je však vidět, že i tento algoritmus trápí podobné problémy, jako ostatní algoritmy. Z tohoto hlediska je nutné uznat, že ač jsou ostatní algoritmy mnohem více specializovány právě na tvorbu grafů podobných požadovaným vizualizacím, je silově řízený algoritmus dobrým konkurentem.

3.3 Srovnání algoritmů

Z jednotlivých algoritmů vyplývá, že vždy bude nutné provést nějaké úpravy. Hledaný algoritmus by tedy měl splňovat tyto podmínky. Počet nutných úprav algoritmu bude co nejmenší. Zároveň nutné úpravy lze snadno implementovat, bez nutnosti přepracování velké části daného algoritmu.

Požadavek	Stromové rozvržení	Hierarchické rozvržení	Dominance Drawing Layout	Silově řízené rozvržení
Více vstupních hran do uzlů	NE	ANO	ANO	ANO
Více kořenů	NE	ANO	NE	ANO
Podpora vrstev	NE	ANO	NE	NE
Hrana mezi uzly stejné vrstvy	NE	NE	NE	NE
Hrana přes více než jednu vrstvu	NE	ANO	NE	NE
Umí pracovat s cykly	NE	ANO	ANO	ANO
Minimalizuje křížení	ANO	ANO	ANO	ANO
Složitost implementace	Nízká	Vysoká	Vysoká	Nízká
Složitost úprav	Vysoká	Střední	Vysoká	Vysoká

Tabulka 1 Porovnání rozvrhovacích algoritmů

Porovnání jednotlivých algoritmů ukazuje Tabulka 1. Z té vychází rozložení do stromu jako rozložení s nejvíce nedostatky, ačkoliv vytvářené vizualizace grafů jsou velmi podobné požadovaným vizualizacím. Požadovaných úprav je velké množství a vyžadovaly by velké zásahy do samotného algoritmu. Je tedy lepší upřednostnit jiný z algoritmů.

Dominance drawing layout je velmi podobný hierarchickému rozložení, rozdílem oproti hierarchickému rozvržení je neexistence podpory pro vrstvy jednotlivých uzlů. Navíc je nutné výsledný graf dále transformovat, přitom obtížnost samotné tvorby grafu je zhruba na úrovni hierarchického rozvržení. Lze tedy Dominance drawing layout vyloučit ve prospěch hierarchického rozvržení, které má požadované vlastnosti již v základním algoritmu.

Silově řízené rozvržení je ze všech rozložení nejodlišnější. Výhodou oproti ostatním rozvržením je jednoduchost na implementaci. Nevýhodou by byla nutnost přepracování systému na výpočet změny polohy uzlů. Potíží je také náchylnost na úvodní rozřazení uzlů, která by u některých vizualizací mohla vytvářet nechtěné výsledky.

Hierarchické rozvržení je naopak složité na implementaci. Ani hierarchickému rozvržení se sice nevyhýbá náchylnost na neoptimální výsledky z důvodu heuristik, nicméně rizika jsou nižší, než v případě silově řízeného rozvržení.

Mezi vhodné kandidáty se tedy řadí buď hierarchické rozvržení, nebo silově řízené rozvržení. Zde je velkou výhodou hierarchického rozvržení zaměření na podobné grafy, jakými jsou právě vizualizace prerekvizit předmětů. Algoritmus tedy řeší téměř všechny požadované aspekty. Silově řízené rozvržení je naopak spíše rámcem, do kterého by bylo nutné doplnit jednotlivé síly a výpočetní pravidla, která by vedla k výslednému grafu. Pro použití v této aplikaci je tedy zvoleno hierarchické rozložení.

4 Knihovny na zobrazení grafů

Jednotlivé knihovny poskytují různé množství operací, které lze s grafy provádět. Některé knihovny nejsou primárně určeny na vykreslování grafů, ale na práci s nimi, například vyhledávání cest, statistické analýzy, data mining, či výpočty různých metrik a optimalizací. Pro potřeby této aplikace jsou brány v úvahu pouze vizualizační schopnosti daných knihoven.

Grafové knihovny lze rozdělit do dvou skupin, opensource a placené knihovny. Opensource knihovny většinou poskytují méně rozmisťovacích algoritmů a změn stylu vykreslování. Na druhé straně umožňují upravit si implementaci podle vlastní potřeby. Druhou skupinou jsou placené knihovny. Problémem placených knihoven je naopak vysoká cena v řádech tisíců dolarů. Za tuto cenu získáte zkompilované knihovny. Pokud je tedy potřeba upravit nějakou funkcionalitu knihovny, musí se tato úprava domluvit přímo s tvůrci knihovny, kteří by ji museli doimplementovat navíc.

4.1 Porovnávané knihovny

Vybrané knihovny jsou z nabídky opensource grafových knihoven, které jsou určeny pro jazyk Java a existuje okolo nich alespoň nějaká komunita. Důvodem volby jazyku Java je jednak větší nabídka knihoven, ale také vhodnost pro tvorbu webové aplikace a zkušenost autora. Knihovny jsou porovnávány z hlediska vhodnosti pro vytvoření této aplikace. Nejdůležitějším bodem je tedy podpora hierarchického rozvržení. Dalším bodem je schopnost exportovat nejlépe do formátu SVG či PNG. Hodnocena je také vizuální stránka výsledného grafu. Nejmenší ohled je brán na složitost implementace a upravitelnost knihovny.

Mnoho z knihoven je vytvářeno paralelně s GUI aplikacemi umožňujícími tvorbu a práci s grafy. Tyto aplikace často poskytují větší funkcionalitu než samotná knihovna, případně mají možnost připojení komunitou vytvořených pluginů. Aplikace ale většinou již nejsou opensource a nelze z nich tyto funkcionality získat. V porovnání jsou tedy brány v úvahu schopnosti dané knihovny a ne stejnojmenné aplikace.

4.1.1 Gephi

Knihovna Gephi[12] vzniká společně s GUI aplikací. Bohužel vývoj této aplikace je pro komunitu hlavní na úkor samotné knihovny. Dokumentace k API je proto téměř neexistující. Knihovna má podporu pro model hierarchického grafu, ale nemá nativní rozvržení k jeho vykreslení. Pro GUI aplikaci existuje stažitelný doplněk, který přidává podporu pro zobrazení hierarchického rozvržení. Knihovna má však dobrou podporu pro export do různých typů souborů včetně SVG. Gephi primárně slouží k vykreslení grafů, kde je výsledkem zajímavé vizuální rozložení. Informativní hodnota získaná z jednotlivých uzlů a hran je spíše v pozadí. Výhodou Gephi je vysoká modularita. Jednotlivé prvky grafu lze jednoduše vytvořit a využít s pomocí knihovny v aplikaci. Na druhé straně je celková struktura knihovny velmi rozsáhlá a komplikovaná.

4.1.2 JGraph

Projekt JGraph[10] poskytuje grafovou a vizualizační knihovnu pro jazyky JavaScript a Java. JGraph poskytuje vysoké množství rozložení uzlů v grafu. Jako jediná open-source knihovna má nativní podporu pro hierarchické rozvržení. Hierarchické rozvržení však neumožňuje žádnou specifikaci úrovní. Nelze proto určit, že má uzel grafu začínat na druhém stupni, případně má-li být mezi dvěma uzly vynechaná úroveň. Při tvorbě grafů by tak nebylo možné určit, v jakém semestru se bude daný předmět vyučovat. Další nevýhodou knihovny je špatný návrh a implementace knihovny. Metody vracejí převážně Object a nutí tak programátora k jakékoliv činnosti využívat přetypování. K úpravě vlastností je nutné přepisovat celé metody, což může být problematické. Přepisování je ztíženo využíváním magických hodnot a metod, které mívají řádově stovky řádků. Knihovna nemá téměř žádnou podporu pro modularizaci. JGraph umožňuje nastavit styl pro celý graf i pro jednotlivé hrany a uzly. Nastavení se provádí pomocí formátovacích textových řetězců, nebo parametrových map. Základní nastavení stylu vyžaduje minimum změn. Knihovna podporuje výstup do SVG.

4.1.3 JUNG

JUNG[17] je menší knihovna umožňující snadnou tvorbu grafů a práci s grafy. Verze 2 této knihovny vznikala po přidání generik do Javy ve verzi 1.5. Generika jsou tak v knihovně velice rozsáhle využívána, což působí spíše negativně. Knihovna se také zaměřuje spíše na nevizuální práci s grafy. Vizuální schopnosti jsou na základní úrovni s nutností si jakoukoliv změnu doimplementovávat. Výhodou zde je možnost snadno zapojit vlastní transformační a renderovací prvky, kterými se upraví vzhled grafu. Vizualizace je podporována pouze směrem do zobrazení pomocí javax.swing, nemá tedy žádnou vlastní podporu pro export grafu ve formě nějakého typu obrázku. Knihovna bohužel nepodporuje hierarchické rozvržení.

4.1.4 GraphStream

GraphStream[13] se odlišuje tím, že nemá běžnou strukturu jako ostatní knihovny. Tedy Graph, Model, Layout, Node, Transformator, Renderer, ale pracuje pouze s grafem a jednotlivými uzly grafu. Knihovna bohužel umí pouze silově orientované rozložení. Hierarchické rozvržení bylo přislíbeno, ale prioritou je rozšíření základů knihovny. Knihovna je cílena k použití na webu. Styly jednotlivých prvků grafu se tedy definují ve formátu CSS. Stylování nabízí mnoho možností a je v dokumentaci dobře popsáno. Knihovna má podporu pro SVG, ale pouze v rozsahu uložení záznamů o jednotlivých uzlech. Neprovádí se uložení stylu a pozice získané z aplikace rozvrhovacího algoritmu.

4.1.5 Prefuse

Prefuse[15] je původně profesionálně vyvíjená knihovna, která byla uvolněna jako open-source. Zajímavostí Prefuse je, že zpracovává grafy jako tabulky. Vlastnosti jednotlivých uzlů a hran jsou nadefinovány jako sloupce. Uzly a hrany následně tvoří řádky daného grafu. Ze všech porovnávaných knihoven je podporováno největší množství různých rozložení. Nicméně knihovna nemá žádnou podporu pro hierarchické rozložení. Knihovna umožňuje vybírat styl dle přednastavených konstant, ale pro některé konstanty není vyroben renderer, který by je uměl zpracovat. Práce na této knihovně byly ukončeny a komunita se přesunula na vývoj nového projektu pro ActionScript.

4.1.6 Grappa

Grappa[14] je portem knihovny GraphViz do jazyka Java. Jelikož se jedná o dílo jedné osoby, umí knihovna pouze velmi malou podmnožinu schopností původní knihovny pro jazyk C++. Zajímavostí knihovny je, že každý prvek grafu je brán jako vlastní podgraf. Knihovna nejlépe zpracovává grafy jako soubory v DOT formátu[31], ze kterých se pomocí parseru vytvoří výsledný graf včetně stylů. Grappa umožňuje následně s takovým grafem pracovat. Výstup je však možný opět pouze do DOT formátu, tedy není možné Grappa použít k výpočtu rozložení, ačkoliv umožňuje z grafu získat Panel dědící od javax.swing.JPanel, jehož obsah lze převést do obrázku.

4.1.7 Porovnání knihoven

Vlastnosti jednotlivých knihoven shrnuje Tabulka 2.

Knihovna	Hierarchické rozvržení	Podpora SVG	Vizuální stránka 0-5	Složitost knihovny 0-5
Gephi	NE	ANO	3	3
JGraph	ANO	ANO	4	5
JUNG	NE	NE	2	2
GraphStream	NE	ANO	3	1
Prefuse	NE	NE	4	3
Grappa	NE	NE	3	3

Tabulka 2 Porovnání grafových knihoven

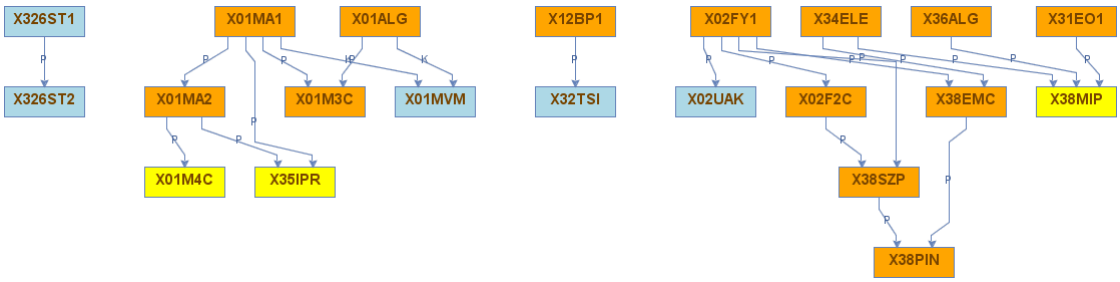
Jak Tabulka 2 ukazuje, podpora pro hierarchické rozložení je velice malá. To je nejspíše způsobeno složitostí implementace. Podpora SVG exportů je již na lepší úrovni, PNG exporty jsou podporovány s výjimkou Grappa. Z hlediska vizuální stránky nabízejí jednotlivé knihovny podobné možnosti, rozdíl je spíše v nutnosti zásahů do nastavení, případně vytvoření implementace pro požadovaný styl uzlů a hran. Ve způsobech implementace jsou mezi knihovnami větší rozdíly, vycházející ze zamýšleného využití a způsobu vzniku dané knihovny. Původní opensource knihovny mají místy velmi nepřehlednou strukturu oproti projektům, které mají stálé týmy, nebo vycházejí z profesionálních knihoven.

Jako nejslabší knihovny vyšly z porovnání knihovny JUNG a Grappa. V obou případech je okolo knihoven příliš malá komunita, která by tyto knihovny dále rozvíjela. Na další úrovni jsou knihovny Gephi, GraphStream a Prefuse. Problémem Gephi je zaměření komunity na rozvoj GUI aplikace, do které přibývají funkcionality, které se však neodrážejí zpět do možností knihovny. GraphStream je ze všech porovnávaných knihoven nejmladší, mimo přislíbeného hierarchického rozvržení tedy chybí mnoho dalších důležitých funkcionalit. Prefuse byla v roce 2009 převedena na opensource a došlo k vypuštění zdrojových kódů, nicméně se kolem knihovny neudržela komunita, která by tuto knihovnu dále rozvíjela. Poslední knihovnou je JGraph, která splňuje oba základní požadavky. JGraph má velikou výhodu v početnější komunitě a také v podpoře stálého týmu, který vyvíjí placenou verzi knihovny. Knihovna tak má větší schopnosti jak v poskytovaných rozvrženích, tak vzhledové stránce výsledného grafu oproti konkurenci. Nicméně knihovně se nevyhnuly některé problémy převážně v rámci implementace, ale i návrhu některých částí knihovny. I přes tyto nedostatky je knihovna nejvhodnějším kandidátem na využití v této aplikaci.

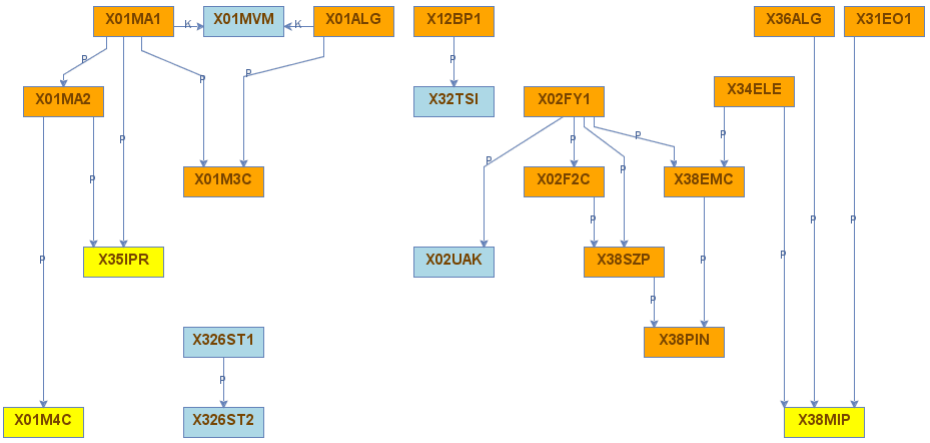
Je ale nutné brát v úvahu, že JGraph nesplňuje všechny požadované vlastnosti. Nabízí se tak dvě možnosti, buď upravit současnou implementaci knihovny JGraph, která by opravila současné nedostatky, nebo vytvořit vlastní implementaci hierarchického rozvržení, která bude upravena tak, aby vyhovovala přesně požadavkům na výsledné grafy. Zde je nutné brát v úvahu, že právě potřeba mít dvojici uzlů na jedné úrovni je velkým zásahem do samotného algoritmu a ovlivňuje prakticky celý průběh jak výpočtu. Stejně tak je nutné změnu zohlednit i při samotném vykreslování grafu do výsledného obrázku. Obecně je lepším způsobem využít existující návrh, nicméně ten je právě v případě JGraph špatně postaven. Aby bylo reálně možné přidat podporu pro určení úrovní jednotlivých uzlů grafu, bylo by potřeba změnit rozhraní části knihovny. K rozhodnutí mezi těmito variantami se tedy vytvoří řešení v JGraph, tak jak ho zvládá knihovna sama a zhodnotí se náročnost úprav.

4.2 Řešení s pomocí knihovny JGraph

Pro zvolenou knihovnu JGraph byla implementována část aplikace na tvorbu grafu. Implementace využívá čistě možností knihovny, které nejsou nijak upraveny.



Obrázek 8 Automaticky vytvořený graf pomocí JGraph



Obrázek 9 Řešení grafu ručně

Při porovnání řešení s pomocí automatického hierarchického rozvržení v JGraph (viz Obrázek 8, s ručně vytvořeným řešení (viz Obrázek 9), jsou na první pohled patrné zásadní rozdíly. První chybou je nemožnost určit pro jednotlivé uzly grafu v jaké vrstvě se mají nacházet. Knihovna toto provádí plně automaticky a pouze na základě toho, aby dodržela pravidlo, že vazby směřují pouze do následujících úrovní. Problémem zde je provedení implementace, které postupuje přesně dle algoritmu hierarchického rozvržení

na úkor použitelnosti. Důsledkem chyby je neschopnost reálně vyjádřit vztah korekvizity, tedy že předměty si student může zapsat zároveň v jednom semestru. Druhým důsledkem je nemožnost na první pohled určit, které prerekvizity jsou kdy relevantní. Tedy zda bude mít student při nesplnění některé z prerekvizit možnost stále dostudovat v řádném termínu, nebo bude muset studium prodlužovat. Další chybou je překřížení dvojic hran, navíc se hrana ohýbá právě v místě křížení. Poslední chyba vyplývající z výsledného grafu je už čistě na straně knihovny a jedná se o překrytí dvojice písmen popisujících hrany grafu. Student tak sice vidí, že mezi předměty je nějaká závislost, nemůže již však určit, zda si bude moct daný předmět zapsat například pokud sice dostal zápočet, ale nebyl schopen úspěšně složit závěrečnou zkoušku. Obě tyto chyby mají velký vliv na užitečnost výsledného grafu. Grafy vzniklé podle JGraph tedy nejsou názorné, naopak mohou být matoucí.

Použitelnosti knihovny nepřispívá ani její návrh. Metody knihovny mají jako parametry vždy buď primitivní typy, nebo Object. To samé platí i v případě návratových hodnot. Jediné místo, kde je možné obhájit Object je parametr value, tím se umožní grafu obsahovat složitější hodnoty, přičemž v zobrazení grafu se využije value.toString() jako identifikátor. Metody také v některých případech vyžadují více parametrů, než je reálně potřeba k provedení dané akce.

Nevhodné využívání parametrů a návratových hodnot s typem Object vede k tomu, že se musí často přetypovávat. Navíc je tím narušena schopnost typové kontroly od vývojového prostředí. Z implementace je patrné, že rozhraní knihovny nesedělo i samotným tvůrcům hierarchického rozvržení. Rozhraní je určeno spíš pro ostatní rozvržení, které knihovna podporuje. V této situaci, kdy je výběr mezi přepisem velké části implementace s nevhodným rozhraním a vytvořením vlastní implementace od začátku, je vhodnější vytvoření vlastní implementace.

5 Návrh knihovny

Hlavním cílem vytvořené knihovny je dodávat kvalitní vizualizace grafů dle potřeb této aplikace. Účelem tedy není vytvořit knihovnu srovnatelnou právě například s JGraph ve smyslu množství možností, ale naopak se zaměřit na řešení specifických požadavků jednoho problému.

5.1 Změny oproti JGraph

Hlavní změny oproti JGraph vychází právě z nedostatků JGraph. Jedná se tedy o možnost zadat uzlu grafu na jaké úrovni se má nacházet. Jelikož je možné přiřazovat uzlům úrovně dle doporučených semestrů pro jejich zápis, jsou také jednotlivé úrovně rozděleny a popsány podle jejich semestru. Dalším rozdílem je schopnost mít vazbu mezi dvojicí uzlů na stejné úrovni, tak jak je potřeba pro vztah korekvizita. Umožněna je také obousměrná vazba pro konektory, tudíž je možné právě v případě vzájemného vztahu korekvizity nahradit dvojici hran jedinou hranou. Oproti JGraph je také kontrolována pozice popisků hran tak, aby nedocházelo k jejich překrývání. Ostatní vlastnosti JGraph, které jsou pozitivní, jsou v co největší míře zachovávány.

5.2 Popis rozvrhovacího algoritmu

Vytvořená implementace vychází z hierarchického rozvržení a využívá tedy Sugiyamův framework[1]. Tento framework slouží jako rámec pro algoritmy na vykreslování orientovaných grafů. Framework udává jednotlivé kroky, které k vytvoření grafu s hierarchickým rozvržením vedou. Oproti základnímu frameworku jsou odstraněny některé kroky a naopak přidány jiné tak, aby byly splněny definované požadavky na vzhled grafu. Jednotlivé kroky algoritmu jsou:

1. Odstranění cyklů
2. Přiřazení vrstev
3. Nahrazení dlouhých hran
4. Minimalizace křížení
5. Odstranění prázdných pozic
6. Optimalizace pozic
7. Přiřazení souřadnic
8. Přizpůsobení souřadnic
9. Odsazení dlouhých hran

V následujících sekcích jsou popsány jednotlivé kroky detailněji.

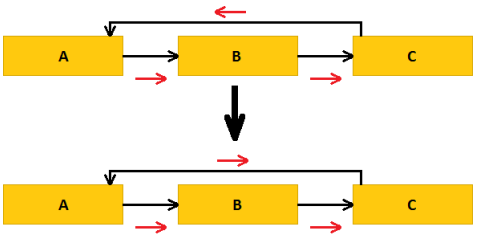
1. krok - Odstranění cyklů

Jelikož je pro rozvrhovací algoritmus potřeba pracovat s orientovaným acyklickým grafem, je prvním krokem odstranění orientovaných cyklů grafu. Obecně je řešení tohoto problému komplikované, jelikož hledání minimální množiny hran, které musí být odstraněny, je NP složitý problém. Z hlediska implementace v této aplikaci mohou cykly

nastat pouze u korekvizit, jinak by byl studijní program nedostudovatelný. Cykly tak vznikají pouze ve dvou případech.

Obrázek 10 Jednoduchý cyklus

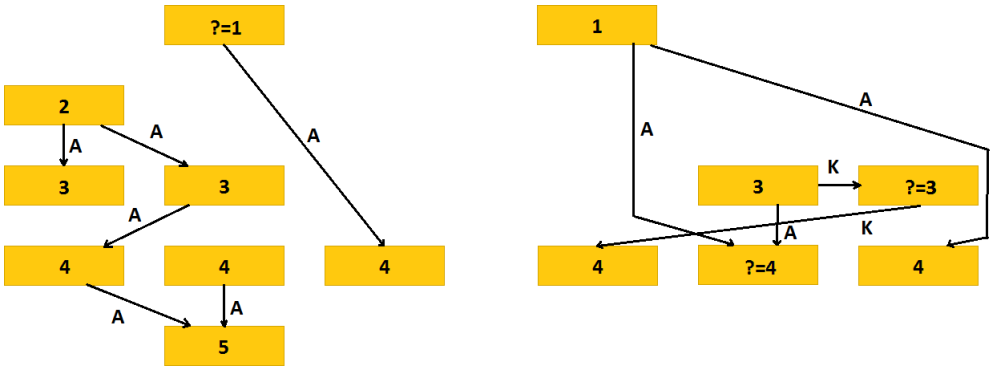
První případ zachycuje Obrázek 10, je zde dvojice předmětů vzájemně ve vztahu korekvizit. Červené šipky vyznačují směr vztahu, černé šipky ukazují jak je konektor vykreslen. V tomto případě se cyklus přeruší smazáním konektoru v jednom směru a změnou vlastností druhého konektoru na obousměrný.



Obrázek 11 Složitý cyklus

Druhý případ vzniku cyklu předvádí Obrázek 11. Situace nastává v případě, že předmět A má korekvizitu s předmětem B, ten je ve vztahu korekvizity s předmětem C atd., až má nějaká z těchto korekvizit opět vztah korekvizity s předmětem A. K vyhledání takovýchto cyklů je využito rekurzivní procházení korekvizitních vztahů. Ke sledování již navštívených uzlů se využije budování jednosměrně zřetězeného seznamu. Ten umožňuje vytvořit celý strom vztahů jedním průchodem. Jakmile daná větev narazí na cyklus, je možné daný konektor otočit.

2. krok - Přiřazení vrstev

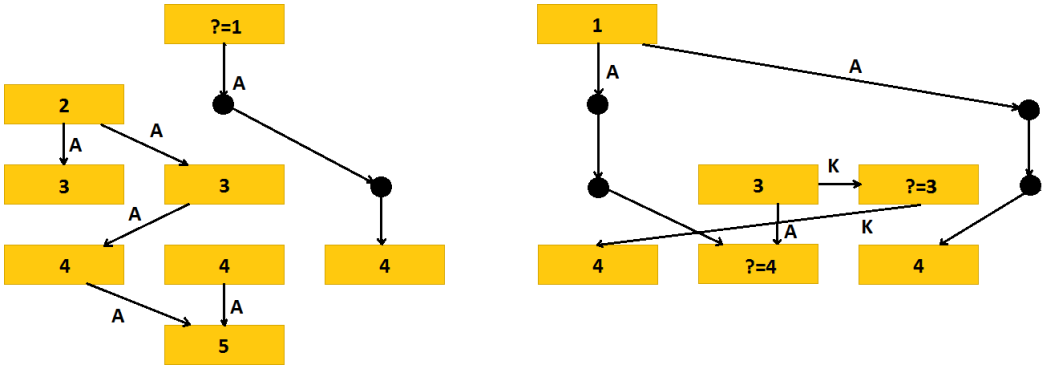


Obrázek 12 Přiřazení vrstev

Druhým krokem frameworku je přiřazení vrstev pro jednotlivé uzly grafu. Stav po přiřazení vrstev zobrazuje Obrázek 12. Řešení tohoto problému je v aplikaci velice usnadněno již ze zadání řešeného problému. Pro většinu předmětů je možné získat jejich umístění do vrstvy ze systému KOS, který obsahuje informace o doporučeném semestru k zápisu daného předmětu. V případě, že tato informace chybí, je stále možné vycházet alespoň

z typu závislosti mezi předměty. Všechny typy závislostí mimo korekvizity umožňují zápis předmětu nejdříve v následujícím semestru, tedy i následující vrstvě. Korekvizita pak umožňuje zápis i ve stejném semestru. Výsledné rozvržení tedy využije informací v systému KOS a následně dopočítá vrstvu pro předměty, kde tato informace chybí. V případě, že by předmět nemohl být kvůli závislostem zapsán v doporučeném semestru, algoritmus automaticky posune předmět na nejbližší takovou vrstvu, kde nedojde k porušení pravidel závislostí.

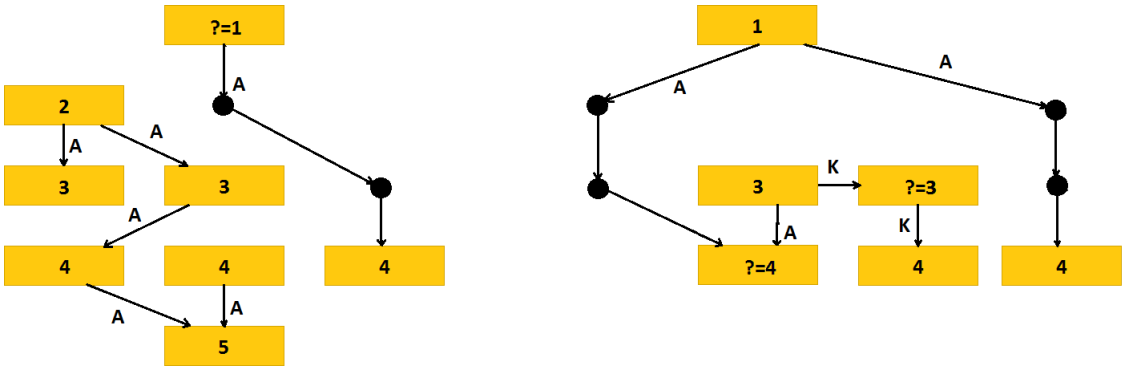
3. krok - Odstranění dlouhých hran



Obrázek 13 Odstranění dlouhých hran

Dalším krokem je nahrazení dlouhých hran (viz Obrázek 13). Dlouhá hrana je taková hrana, jež vede mezi 2 uzly, které mezi sebou mají alespoň 1 vrstvu. Účelem tohoto rozdělení je schopnost reprezentovat potřebu na volné místo, kudy bude moci daná hrana procházet. Hrana přes více úrovní se odstraní a nahradí se řetězem hran a speciálních vrcholů přes jednotlivé úrovně, dokud se nedosáhne druhého vrcholu.

4. krok - Minimalizace křížení

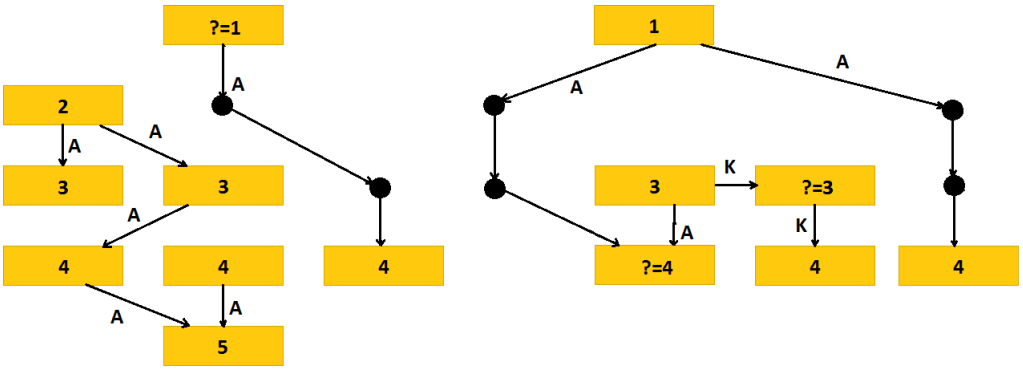


Obrázek 14 Minimalizace křížení

Minimalizace křížení je nejkomplikovanějším krokem při tvorbě hierarchického grafu. Cílem je určit pořadí, v jakém mají být seřazeny jednotlivé uzly tak, aby bylo ve výsledném grafu minimální množství křížících se hran. Optimální řešení tohoto problému je NP složité, tudíž se k řešení tohoto problému využívají heuristické metody. K vytvoření pořadí uzlů v aplikaci se tedy využívá principu, kdy jsou uzly umístěny na průměr

pozic svých předků. Uzly se tedy nejdříve umístí na úvodní pozice s tím, že se mezi nimi zachovají dostatečné mezery. Po inicializaci se iteruje po jednotlivých vrstvách a vypočítávají se nejvhodnější pozice pro jednotlivé uzly. Postupně tak dojde k minimalizaci křížení. Obrázek 14 ukazuje konečný stav.

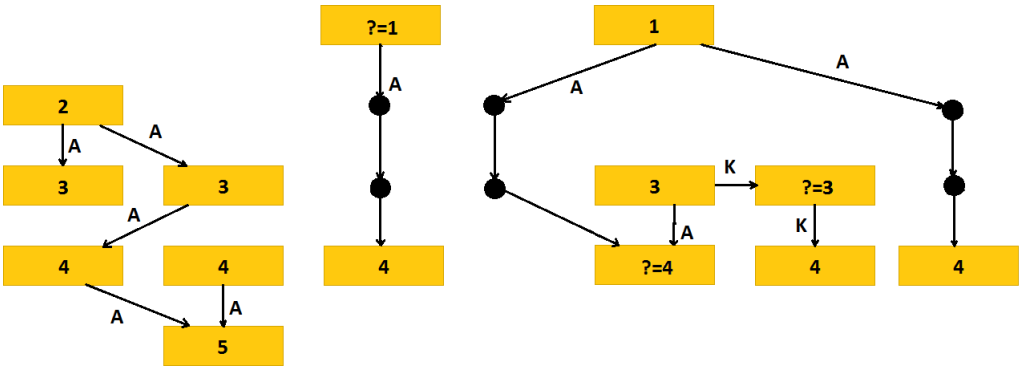
5. krok - Odstranění prázdných pozic



Obrázek 15 Odstranění prázdných pozic

Po dokončení jednotlivých iterací přichází normalizační krok, vycházející ze způsobu této implementace. Jelikož byly nechány mezi uzly při inicializaci mezery, mohly zůstat některé pozice prázdné. Toto obecně není problém, ale jelikož jsou souřadnice uzlů na výsledném obrázku vypočítávány právě na základě pozic, je nutné upravit pozice tak, aby nebyly na výsledném grafu prázdná místa. Tento krok symbolicky ukazuje Obrázek 15. Při reálných výpočtech jsou obvykle mezery mezi více uzly.

6. krok - Optimalizace pozic



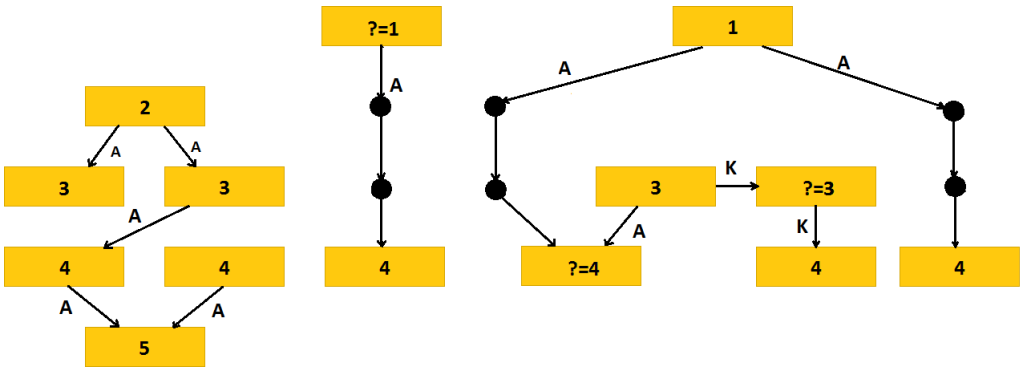
Obrázek 16 Narovnání dlouhé hrany

Má na starosti dvojici úkolů. První je kontrola sousedících dvojic uzlů a přepočtení jejich pozic, zda pro ně není výhodnější si své pozice vyměnit. Pro výpočet vhodnosti pozice se zde využije průměrování pozice jak proti předkům, tak proti potomkům. Druhým úkolem je volitelný krok Sugiyamova frameworku. Jedná se o estetický krok, kdy jsou, pokud je to možné, narovnávány dlouhé hrany (viz Obrázek 16). Účelem tohoto kroku je lepší čitelnost výsledného grafu.

7. krok - Přiřazení souřadnic

Jelikož byl využit průměrovací algoritmus pro přiřazení pozic, tak spolu s dalšími implementačními rozhodnutími lze převést pozice přímo na výsledné x souřadnice, y se přiřadí na základě úrovně.

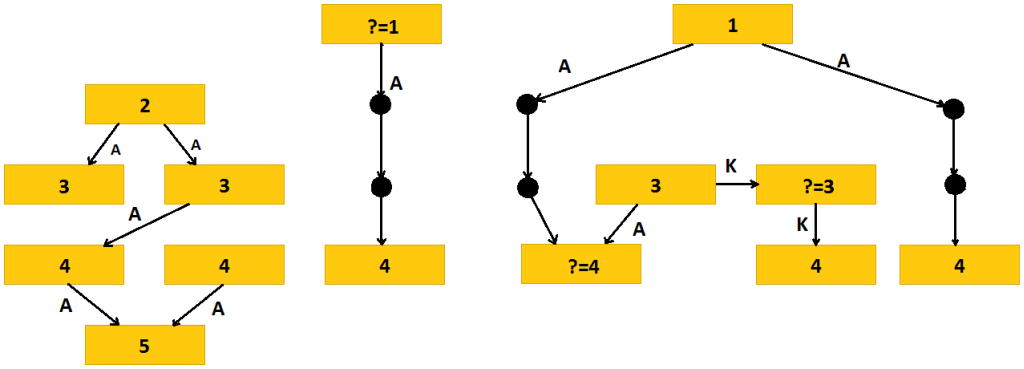
8. krok - Přizpůsobení souřadnic



Obrázek 17 Přizpůsobení souřadnic

V rámci tohoto kroku dochází k vyřešení chyb ze zaokrouhlení pozic. Jednotlivé uzly zde již nemohou měnit své pořadí, ale mohou využít volného místa, které se kolem nich nachází a posunout se například na střed mezi dva své potomky (viz Obrázek 17). Tento krok zlepšuje estetickou stránku výsledného grafu, aby nebyly všechny uzly umístěny do mřížky. Uzly tak mohou být blíže reálných pozic a zkrátí se tak délky hran.

9. krok - Odsazení dlouhých hran



Obrázek 18 Odsazení dlouhých hran

Je krokem, který udává, jak bude vykreslena dlouhá hrana. Tyto hrany byly v 3. kroku nahrazeny speciálními uzly. V tomto kroku se určí, zda bude hrana vykreslena v levé, pravé, nebo centrální části prostoru vymezeného uzlem. Snahou je vizuálně zkrátit délku dlouhých hran, ty se tak mohou více přimknout k té straně, ze které dlouhý konektor vychází a do které míří. Obrázek 18 zachycuje situaci s přimknutím k pravé straně.

5.3 Podrobnější rozbor minimalizace křížení

Metod, jak minimalizovat křížení je velké množství[1-2] a různé metody mohou dávat pro různá data rozdílné výsledky. Jednotlivé metody jsou založeny převážně na dvou principech.

- První se zakládá na nalezení jednotlivých křížení a vytvoření matice křížení. Na základě této znalosti lze jednotlivé uzly rozmisťovat tak, aby se počet křížení minimalizoval.
- Druhým principem je výpočet vhodné pozice každého uzlu jako mediánu, nebo průměru z pozic uzlů, se kterými je tento uzel spojen hranou. Opakováním těchto výpočtů by pak mělo docházet k přiblížení těch částí grafu, které spolu souvisí, a snížit tak počet zbytečných křížení.

Výhodou druhého principu je větší rychlost a často dosažení lepších výsledků. Z vypočtených pozic se navíc zjistí nejen pořadí, v jakém by měly být uzly umístěny, ale také se dají využít k samotnému přiřazení souřadnic jednotlivých uzlů výsledného grafu.

K vytvoření pořadí uzlů v aplikaci se tedy využívá druhého principu, kdy jsou uzly umístěny na průměr pozic. Tato pozice je vyjádřena celým číslem. Tento krok zavádí na první pohled jistou zaokrouhlovací chybu, jelikož průměr nemusí být celé číslo. Je ale potřeba, neboť umožní algoritmu rychleji konvergovat, navíc zachovává možnost převádět pozici na souřadnice bez starosti o překrývání vykreslených uzlů. Hlavním problémem při vytváření pořadí dle průměrů jsou kolize, které nastávají, pokud chce dvojice uzlů na stejnou pozici. Tento problém ale desetinné hodnoty pouze oddalují.

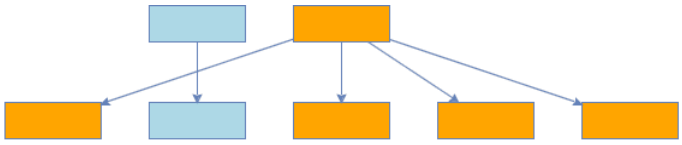
Algoritmus vyžaduje úvodní inicializaci, kdy se uzlům přiřadí startovní pozice. Jednotlivým uzlům se postupně přiřadí pozice bez preferencí. Mezi uzly je však ponechán dostatek prostoru. Účelem je zamezit situaci, kdy se do sebe zaklesnou dva nezávislé větve, které mezi sebou zápasí o pozice. Po inicializaci následuje samotný algoritmus, ten je iterační. V každé iteraci prochází jednotlivé vrstvy grafu a vypočítává průměrnou pozici každého uzlu.

Průchod po vrstvách probíhá od nejnižší po nejvyšší, tedy od prvního semestru po poslední v každé iteraci. Popis algoritmu dle Sugiyamova frameworku[1] využíval průchod od nejnižší po nejvyšší a následně opět iterovat směrem k nejnižší. Tímto však na krajních vrstvách dochází dvakrát po sobě k výpočtu pozice s minimálními změnami zbytku grafu. Průchodu jedním směrem je také lépe uzpůsoben výpočet průměru pozice. Výpočet průměrné pozice je založen na pozicích předků ze dvou předcházejících vrstev. V případě kořenů se jedná o dvě následující vrstvy. Spolu s jednosměrným průchodem vrstev je tedy evidentní, že pouze v případě kořenů jde výpočet proti směru. Tento krok je ale nutný, aby se umožnila mobilita kořenů. Výpočet pozic ze dvou předcházejících vrstev nebyl jediný vyzkoušený způsob. Vyzkoušeny byly také následující varianty:

- Varianta s výpočtem na základě pouze jedné vrstvy. Zde jsou rozdíly zřejmé. Prochází se méně hran, vypočtená pozice je však méně přesná a algoritmus také konverguje pomaleji.
- Výpočet průměru na základě jak následující, tak předchozí vrstvy. Zde vlastně opět přichází do výsledného průměru směs nových pozic a směs starých. V případě průchodu po úrovních jedním směrem je tento způsob spíše kontraproduktivní, jelikož ovlivňuje nový výpočet zastaralou informací. V případě obousměrného průchodu je již tento způsob lepší, nicméně stále bude z jedné strany přicházet starší informace, tedy informace, která je z větší části založená na počáteční pozici, tedy pozici neznámé kvality. Tento způsob je však využit v rámci 6. kroku ke koncovému vyladění pozic, v této chvíli se již nejedná o vytváření pořadí, jen případné optimalizace.

Ať už je využíván libovolný způsob procházení a výpočtu, dochází při výpočtech ke kolizím, tedy případům, kdy chce uzel na pozici, která je již obsazena jiným uzlem. Zde je zavedeno několik pravidel, na základě nichž algoritmus takové kolize vyřeší. Z principu jde o posunutí ostatních uzlů a umístění nového na danou pozici v závislosti na vypočtených ideálních pozicích. Případně, pokud se jedná právě o kolizi na hraně zaokrouhlovací chyby, je zkontrolována nižší pozice.

V rámci implementace byl také pokus řešit kolize nalezením nejbližšího volného místa. Tento způsob však vede k závažným chybám ve výsledných rozvrženích, neboť působí proti seskupování souvisejících uzlů a narušuje následující výpočty pozic.



Obrázek 19 Minimalizace dle hledání pozic

Obrázek 19 zobrazuje situaci, která nastává, pokud jsou uzly umístěny na nejbližší volnou pozici. Při rozmístění oranžových uzlů je nalezena nejbližší volná pozice až za modrými uzly a vytvoří se tak zbytečné křížení. Na rozdíl od používaného algoritmu, který kolize řeší posunem, nemá řešení prostým hledáním nejbližší pozice schopnost modré uzly nijak přesouvat. To samozřejmě vede k většímu množství křížení ve výsledném grafu i k větší nepřehlednosti.

6 Webová aplikace

Druhou částí práce bylo vytvoření webové aplikace, která by umožnila ovládat grafovou knihovnu a také zobrazení vytvořených grafů. Webová aplikace tedy zajišťuje získávání potřebných informací pro grafy, dále využívá grafovou knihovnu pro vytvoření grafů a následně poskytuje vzniklé grafy prostřednictvím webových stránek. Aplikace je schopna pracovat automaticky, s tím, že administrátor může měnit její nastavení.

Implementace je provedena v jazyce Java verze 1.7.

6.1 Použité technologie

Maven 3.0.5 [24] Maven je nástroj na správu a kompilaci aplikací. Hlavním využitím pro Maven je usnadnění kompilace. Maven se při kompilaci stará o nalezení, stažení a přidání externích knihoven do výsledné zkompilované aplikace. Potřebné knihovny stačí zadat do konfiguračního souboru projektu. Mimo tohoto umožňuje přidávat k projektům různá metadata. Maven je velice dobře podporovaným nástrojem ve všech běžných vývojových prostředích. Aplikace je postavena jako webová aplikace kompilovaná pomocí Maven. Má tedy standardní adresářovou strukturu a konfigurační soubor pom.xml.

Spring 3.2.2 [21] Spring je populárním frameworkem postaveným nad JavaEE platformou. Mezi jeho hlavní výhody patří velmi dobrá schopnost integrace a spolupráce s ostatními technologiemi. Zároveň je postaven na konceptu konvence nad konfigurací, to znamená, že konfiguraci provádí programátor pouze v případě, že potřebuje nějakou změnu oproti normálnímu fungování frameworku. Základní nastavení je ale pro běžné webové aplikace naprosto dostačující.

Spring zajišťuje životní cyklus aplikace. Vytváří tak sám instance jednotlivých tříd dle potřeby a zajišťuje vkládání závislostí takto vytvořených objektů. Programátor pouze označí třídy, které mají být Springem spravovány. Označovat je možno dvěma způsoby. První je využívání anotací, které se píší přímo třídám, metodám či proměnným. Výhodou zde je rychlost, programátor má okamžitý přehled. Druhým způsobem je zápis pomocí konfiguračních XML souborů, tento způsob přináší naopak možnost provádět změny v již zkompilované aplikaci bez nutnosti jí znovu kompilovat.

Další důležitou vlastností je schopnost vyvolávat akce v zadaných časech, nebo časových intervalech. Tato funkcionalita je nezbytná pro automatické fungování této aplikace, neboť umožňuje snadno spouštět algoritmus na aktualizaci informací a grafů.

Spring také zajišťuje zabezpečení aplikace a stará se o kontrolu práv přihlášených a nepřihlášených uživatelů. Toho je využito pro zabezpečení administrátorské části aplikace.

Hibernate 4.1.7 [20] Slouží aplikaci jako spojení s databází. Jedná se o framework implementující Java persistence API. Umožňuje provádět ORM, tedy převod mezi objektem v aplikaci a entitou uvnitř databáze standardizovaným způsobem. Programátor může přistupovat k datům v databázi pomocí jazyka HQL, který je odvozený z SQL.

Konfigurace Hibernate ve spolupráci se Spring je velice jednoduchá, stačí pouze dodat knihovnu k použité databázi a přístupové údaje. Samotné entity pro ORM lze pak určit opět pomocí anotací, nebo XML souborů.

JSF 2.2.4 [18] JavaServer Faces je webový framework Java EE standardu sloužící jako zprostředkovatel komunikace mezi webovým prohlížečem a aplikací v jazyce Java. Webové stránky se vytváří jako XHTML soubory s použitím kombinace JSF XML značek s tím, že je možné tyto značky kombinovat s klasickým jazykem XHTML. JSF značky pak umožňují provádět volání jednotlivých metod uvnitř aplikace.

Stejně jako v případě Hibernate je i JSF velice dobře integrovatelný do Spring. Je tak možné nechat celé JSF řídit pomocí Spring. Programátor pak nemusí mít téměř žádnou vlastní znalost samotného JSF mimo samotných značek na webových stránkách.

PostgreSQL 9.3 [19] Je opensource databázový systém pro relační databáze s podporou ze strany Hibernate. Jelikož nemá aplikace žádné zvláštní požadavky vzhledem k používané databázi a nepracuje s velkým množstvím dat, je tato databáze vybrána čistě na osobní preferenci a zkušenosti.

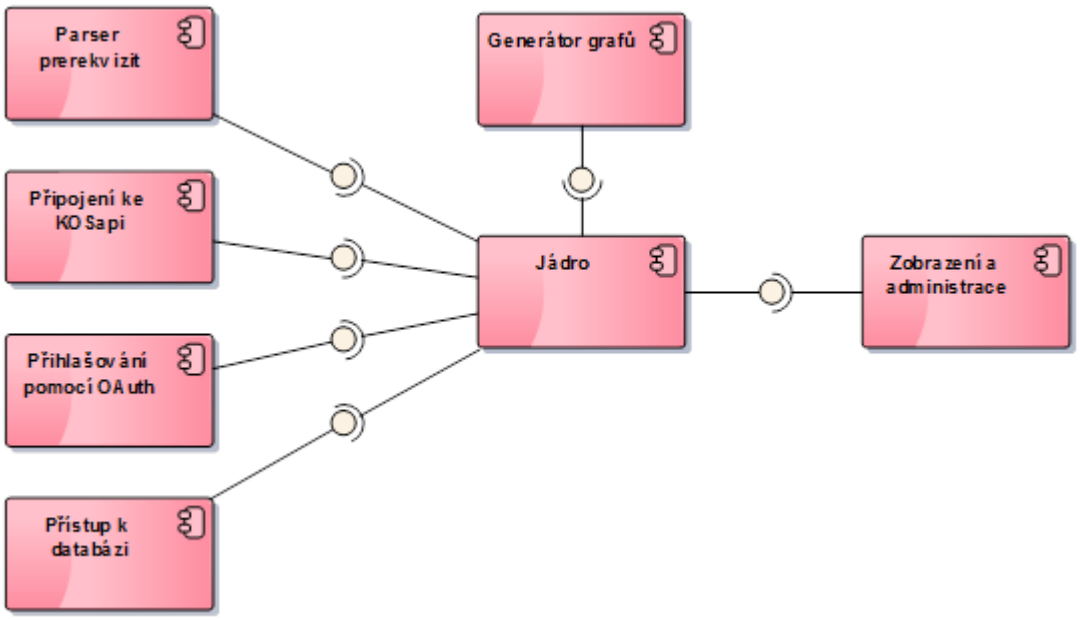
Apache POI 3.11 [23] Projekt Apache, jehož cílem je vytvoření Java API, které umožňuje v rámci Java aplikací pohodlně pracovat se soubory formátů používaných programy ze sady MS Office. V aplikaci se konkrétně využívá knihovna umožňující čtení xls souborů, ve kterých jsou uloženy seznamy prerekvizit. Pomocí knihovny lze přistupovat k jednotlivým buňkám souboru, přičemž knihovna automaticky vyhodnocuje výrazy uložené v jednotlivých buňkách.

6.2 Využívané systémy třetích stran

auth.fit.cz [25] Tento systém poskytuje autentizaci pomocí protokolu OAuth2.0 [27]. Účelem systému je umožnit uživatelům přihlášení do studentských aplikací pomocí jejich univerzitních přístupových údajů bez toho, aby museli zadávat své heslo přímo dané aplikaci. Uživatel, či přistupující aplikace, se tedy přihlásí vůči serveru auth.fit.cz. Server při úspěšném přihlášení vrátí token, jenž slouží jako ověření uživatele dalším službám. Další poskytovanou funkcionalitou je možnost aplikace jednak autentizovat sama sebe vůči systému, ale také autentizovat se za přihlášeného uživatele a tedy například přistupovat k systému KOS jménem přihlášeného uživatele.

KOSapi [22] Systém poskytující REST rozhraní pro přístup k vybraným informacím z databáze KOS. Aplikace může formulovat dotazy na toto rozhraní k získání záznamů ve formátu atom XML. Z těchto záznamů má pro vyvíjenou aplikaci smysl získání informací o současně vyučovaných programech a možnost získat informace o předmětech vyučovaných v rámci daných programů. Tyto informace jsou chráněny proti přístupu neautorizovaných osob. Proto je k ověření identity využit systém auth.fit.cz.

6.3 Architektura



Obrázek 20 Přehled komponent

Aplikace využívá vícevrstvou architekturu, jejíž jednotlivé komponenty zobrazuje Obrázek 20. Každou komponentu je možné nahradit libovolnou jinou implementací. Zároveň je kladen důraz na existenci pouze slabých závislostí, účelem tohoto opatření je jednak snaha omezit počet chyb, které mohou při vytváření aplikace vzniknout a také zjednodušení logiky aplikace. Hlavním smyslem tohoto opatření je umožnit velmi snadnou modifikovatelnost například ve smyslu možnosti lehce nahradit modul na získání prerekvizit předmětů z xls souboru například za modul, který by prerekvizity získával z XML.

6.3.1 Vrstva zdrojů dat

Parser Prerekvizit Tato komponenta využívá knihovnu Apache POI, pomocí ní lze extrahovat informace z jednotlivých listů a buněk xls dokumentu. Z nastaveného xls dokumentu načte kódy požadovaného a požadujícího předmětu a typ vztahu. Z tohoto výběru jsou odfiltrovány ty vztahy, které nemají pro aplikaci význam, neboť neurčují pořadí předmětů. Centrální třídou je třída PrerequisitesParser, která má metody getPrerequisites() k získání všech prerekvizit a metodu getCourses(), která vrací všechny předměty s prerekvizitami.

Připojení ke KOSapi Komponenta obsahuje rozhraní k provádění dotazů na systém KOSapi. Připojení je realizováno třídou KOSapiConnectionImpl a metodou getRequestResult, ta má jako parametry OAuth token a KOSapiRequest, který je objektem definujícím požadavek na KOS. Lze tak snadno změnit získávané informace úpravou právě tohoto dotazového objektu. Výsledky dotazů jsou atom XML, které se dále zpracovává třídou AtomParser. Ta má metody getProgrammes(), getStudyPlans() a getCourses(), které ze vstupních atom XML získávají programy, studijní plány a předměty daných studijních plánů.

Přihlašování pomocí OAuth Třída OAuthLogin zajišťuje získání autentizačního tokenu pro ověření vůči KOSapi. Přihlášení probíhá zavoláním metody login(). Ta provede zaslání požadavku s identifikátorem a heslem aplikace. Token je následně uchováván po omezenou dobu, aby nedocházelo k opakovaným autentizacím při více dotazech v krátké době. Systém automaticky zajišťuje, že je token v případě potřeby obnovován. Aplikace samotná tak k tokenu přistupuje pomocí metody getAccessToken();

Přístup k databázi Tato komponenta má dvě části. První částí jsou business objekty, které reprezentují jednotlivé databázové entity tak, jak byly popsány v doménovém modelu (viz Obrázek 3). Druhou částí je pak třída DaoImpl umožňující provádět dotazy nad danými entitami. ORM je zajištěno pomocí frameworku Hibernate.

6.3.2 Servisní vrstva

Je centrální vrstvou, jež se stará o tvorbu grafů a zpracování požadavků uživatelů. Centrální třídou je VisualisationServiceImpl, ta využívá nižší vrstvu zdrojů dat k získání aktualizací programů, studijních plánů a vyučovaných předmětů, stejně jako známých prerekvizit. Na základě těchto informací a podle nastavení pro jednotlivé studijní plány generuje jednotlivé grafy. Generování grafů je spouštěno metodou generateVisualisations, která má jako parametry studijní plán, seznam prerekvizit a předmětů. Třída dále umožňuje měnit nastavení a získání informací o nabízených vizualizacích i získání samotných grafů pro REST rozhraní, které je dále distribuuje.

Generátor grafů Třída HierarchyGraphGeneratorImpl se stará o transformaci informací o předmětech a prerekvizitách a předává tyto informace grafové knihovně. Při tvorbě grafu prochází aplikace jednotlivé prerekvizity a kontroluje, zda jsou smysluplné pro daný studijní plán, tedy zda studijní plán má požadovaný i požadující předmět. Pokud taková dvojice existuje, vloží se uzly do grafu a spojí se hranou. Aplikace má definovány styly pro skupiny předmětů dle jejich role v daném studijním plánu. Jsou rozlišovány povinné, povinně-volitelné a volitelné předměty. Modul má také k dispozici semestr, ve kterém je doporučen zápis předmětu.

Samotná knihovna u grafových knihoven běžné třídy Graph, Node, Connector pro popis grafu, uzlu a hrany. Vytvoření hierarchického provedení je pak provedeno třídou HierarchyLayout, jejíž metoda applyLayout bere jako parametr daný graf. Po vytvoření rozvržení je možné využít třídu PlotGraph, která daný graf převede do výsledného obrázku metodou plotGraph, parametry zde jsou název výsledného souboru, vizualizovaný graf a typ programu.

6.3.3 Zobrazovací vrstva

REST rozhraní Obsahuje controller ImageController, který se stará o distribuci vytvořených grafů jednotlivým uživatelům, či jiným serverům. Jednotlivé grafy lze získat dotazem na /graph?studyplan=hodnota, kde hodnota je zkratka pro daný studijní plán, například BPSTMSI pro zaměření Softwarové inženýrství programu Softwarové technologie a management. Návratovou hodnotou je čistě obrázek výsledného grafu. Tento systém slouží primárně jako přístupový bod pro ostatní systémy, které potřebují pouze obrázek pro svou vlastní potřebu.

View vrstva Do této vrstvy patří všechny webové stránky, renderované pomocí frameworku JSF a jejich podpůrné třídy z balíku cz.vopbp.viewbb. Webové stránky obsa-

hují přehled zveřejněných grafů, stránky s jednotlivými grafy a také administrátorské rozhraní. V rámci administrátorského rozhraní může administrátor vybírat, pro které studijní plány budou generovány grafy a které grafy se mají generovat. Administrátor zde má také přehled kolik uzlů v daném grafu existuje, nebo zda je daný studijní plán otevřený pro zápis.

6.4 Požadavky a omezení vyplývající ze vstupních dat

Funkčnost aplikace spočívá ve schopnosti získávat data z KOSapi a také ze souboru s prerekvizitami. Výsledné fungování aplikace je velmi závislé na kvalitě a úplnosti potřebných dat. V průběhu implementace byly ale v těchto zdrojích nalezeny případy, které mohou způsobit chyby při tvorbě grafů, nebo rozdíly oproti očekávaným výsledkům.

KOSapi V některých případech chybí v datech získaných z rozhraní KOSapi důležité informace, nebo jsou nevhodně zapsány. Příkladem je zde hodnota recommendedsemester u předmětů studijního plánu. Pole má obsahovat semestr doporučený k zápisu daného předmětu, v některých případech tato hodnota chybí. U některých předmětů je těchto semestrů zapsáno více, ale tyto zápisy nemají jednotnou formu, tudíž jsou obtížně strojově zpracovatelné. Aplikace se v tomto případě pokusí hodnotu získat, pokud to však není možné, nebo hodnota chybí, aplikace přiřazuje semestr čistě na základě návazností tak, aby nedošlo k porušení jejich pravidel.

Další chybějící hodnoty jsou v případě homepage jednotlivých předmětů. Tato hodnota má obsahovat odkaz na webové stránky daného předmětu. Přičemž dodané xls obsahuje 208 předmětů s prerekvizitami, z nichž u pouhých 57 je uvedena homepage.

Soubor prerekvizit Aplikace vyžaduje z prerekvizit získávat kód požadujícího a požadovaného předmětu a také označení s typem prerekvizity. Dle dodaného souboru s prerekvizitami počítá implementace s rozřazením informací do 3 sloupců pro jednotlivé parametry. V dodaném souboru však existují řádky, kde je kód předmětu posunut na jiný sloupec, případně kde je před kódem předmětu znak -. Pokud začíná obsah buňky znakem -, jde z pohledu buněk xls formátu o začátek výrazu. Knihovna se pak při čtení takové buňky pokusí výraz vyhodnotit a vrátí chybový obsah buňky.

7 Testování

Součástí tvorby software je i otestování správné funkčnosti. Testování lze rozdělit na manuální a automatické. Snahou je testovat co nejvíce automaticky s výjimkou míst, kde je tvorba automatického testu nákladnější, než manuální test. Z tohoto pohledu je testována i aplikace na vizualizaci prerekvizit. Jako webový server byl používán Apache Tomcat 8.0.15.

7.1 Automatické testování

Cílem testování bylo ověřit správnou funkčnost jednotlivých prvků aplikace. Automatické testování bylo prováděno pomocí JUnit framework[30]. Tento framework je vzájemně kompatibilní se Spring, lze tedy testovat aplikaci se stejným nastavením jako v reálném prostředí. Spring v tomto případě spouští jednotlivé testy. Aby nedošlo k případnému narušení reálných grafů, nebo dat v databázi, využívá Spring upravený konfigurační soubor pro testování umístěný v src/test/java/cz/vopbp/testConf. Obsahem složky jsou i SQL soubor na inicializaci testovací databáze a testovací soubor s prerekvizitami.

Testovací data pro jednotlivé testy jsou založena převážně na reálných datech. Výjimkou jsou případy, kdy bylo potřeba otestovat potenciálně reálný případ, který se však v současných datech nevyskytuje.

Jelikož aplikace z velké části spoléhá na systémech třetích stran, jsou pro potřeby testování přiloženy stub objekty. Principem tvorby těchto objektů je nahrazení nějaké komponenty, nebo systému třetí strany vlastní implementací, která vrátí vždy přesně definované výsledky a umožní tak lépe zacílit testování a oddělit jednotlivé části testovaného systému.

Testování je zaměřeno na ověření funkčnosti nižších vrstev architektury (viz Obrázek 20). Konkrétně komponent Parser prerekvizit a Připojení ke KOSapi. Parser prerekvizit je otestován, že vrací prerekvizity ve správném formátu a že komponenta reaguje na nastavení v konfiguračním souboru. Komponenta Připojení ke KOSapi je testována na schopnost správně interpretovat příchozí atom XML z KOSapi. Není však již dobře možné testovat samotné připojení na KOSapi, nebo jednotlivé dotazy, neboť se informace v KOSapi v čase mění.

Testováno je i jádro aplikace. Je otestována schopnost aktualizovat informace v databázi jak ve smyslu vyhledání nových studijních programů a plánů, tak ve změnách dat v databázi již existujících. Otestovány jsou také funkcionality směrem k uživatelům. Tedy změny nastavení pro jednotlivé studijní plány a správné předávání informací vyšším vrstvám.

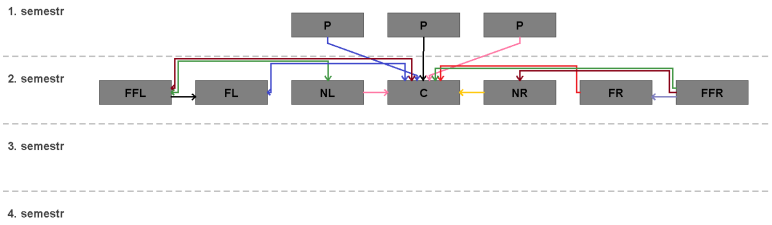
Automatické testování nemohlo být provedeno k ověření správnosti vytvářených grafů. Testy ověřují jen základní informace, tedy zda má daný graf správný počet uzlů, ale k ověření správné funkčnosti je nutné manuální testování. Stejně tak byla manuálnímu testování ponechána vrstva Zobrazení a administrace.

7.2 Manuální testování

Manuálního testování bylo využito k testování funkčnosti uživatelského rozhraní. Testování bylo v prováděno v prohlížečích Internet Explorer 11, Firefox 32 a Chrome verze 39 a 42. Cílem bylo ověřit správnou funkčnost všech ovládacích prvků aplikace a ověřit, že na stránky administrátorského rozhraní se nelze dostat bez přihlášení do systému.

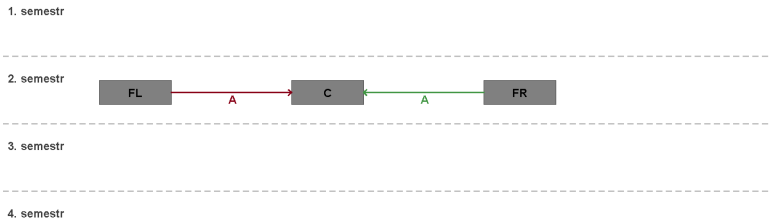
Další částí, která byla testována manuálně, je samotné testování výsledků grafového algoritmu. Důvodem je nemožnost přesně definovat jaký graf je ještě kvalitní a který je už naopak nepřehledný algoritmicky. Jako pomoc jsou v balíčku aplikace připraveny testy, které obsahují testování jak celého grafického algoritmu, tak pouze kreslící části. Tyto připravené grafy obsahují složitější situace a případy, které by mohly v reálných datech nastat, nebo v nich nejsou časté. Lze tak urychlit nalezení nedostatků v těchto případech.

Připravené testy na speciální situace se zabývají hlavně řešením grafů obsahujících korekvizity.



Obrázek 21 Testování korekvizit

Obrázek 21 je příkladem takového testu. Graf obsahuje řadu přímých i nepřímých korekvizit, mezi které je zapojeno několik normálních hran. Lze tak ověřit, zda fungují všechny druhy vykreslení hrany v případě korekvizit a vykreslení jednotlivých druhů šipek u těchto konektorů.



Obrázek 22 Přímé korekvizity s mezerou

Obrázek 22 pak umožní ověřit, že jsou při vykreslování správně detekovány přímé korekvizity.

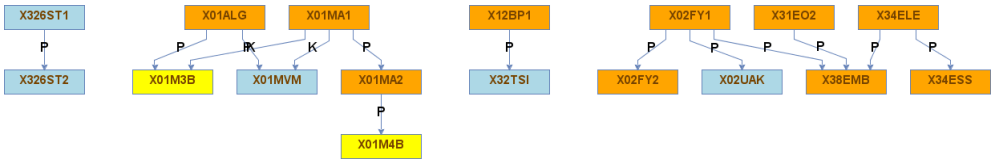
8 Vyhodnocení výsledků

Jak již bylo při testování v části 7.2 zmíněno, testování grafů je možné pouze manuálně. Důležité je zde jednak ukázat, zda došlo ke splnění požadavků na výsledné grafy definovaných v části 3.1. Dále také porovnat, jak kvalitní jsou výsledné grafy vzhledem k referenční implementaci s pomocí knihovny JGraph. Nejdůležitějším je pak také porovnat, jak kvalitní jsou vytvářené grafy vzhledem k poskytnutým ručně vytvořeným grafům.

8.1 Výsledné grafy

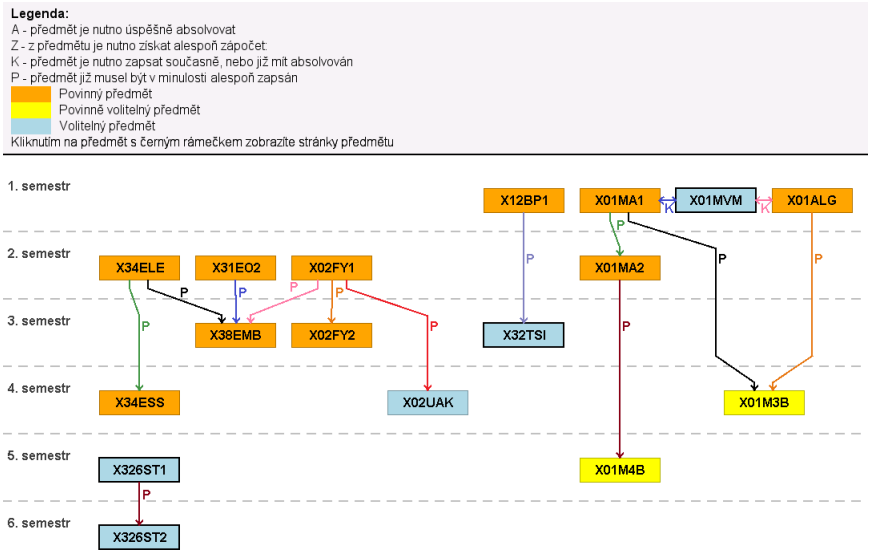
Výsledné grafy jsou porovnány s grafy vytvářené knihovnou JGraph a také se současným ručně vytvořeným grafem.

8.1.1 Porovnání s JGraph



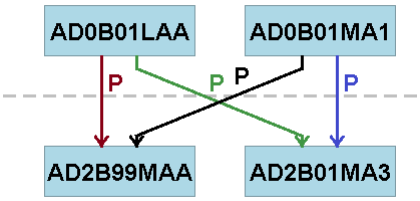
Obrázek 23 Vygenerovaný graf v JGraph

Obrázek 23 obsahuje řešení grafu v knihovně JGraph. Ta neumožní nastavit vrstvu uzlu, což vede k tomu, že naprostá většina uzlů je v nižších úrovních. To navíc vede k větší koncentraci konektorů a jejich popisků. Knihovna ale nekontroluje pozici popisků, tudíž může dojít k jejich překrytí.



Obrázek 24 Vygenerovaný graf pomocí vlastní knihovny

K řešení pomocí JGraph (viz Obrázek 23) je zde ekvivalentní vykreslení grafu pomocí vlastní knihovny (viz Obrázek 24). Na první pohled je zde patrné, že reálně jsou prerekvizity více rozprostřeny po semestrech. Také je zde vidět případ, kdy jsou předměty ve vzájemných vztazích korekvizit v rámci jednoho semestru, což nebylo možné v JGraph vyjádřit. Schopnost správně zařadit předmět do semestru také vede k možnosti vyznačit samotné semestry přímo do obrázku. Do samotného obrázku je také přidána legenda popisující graf.



Obrázek 25 Kontrola překrytí popisků

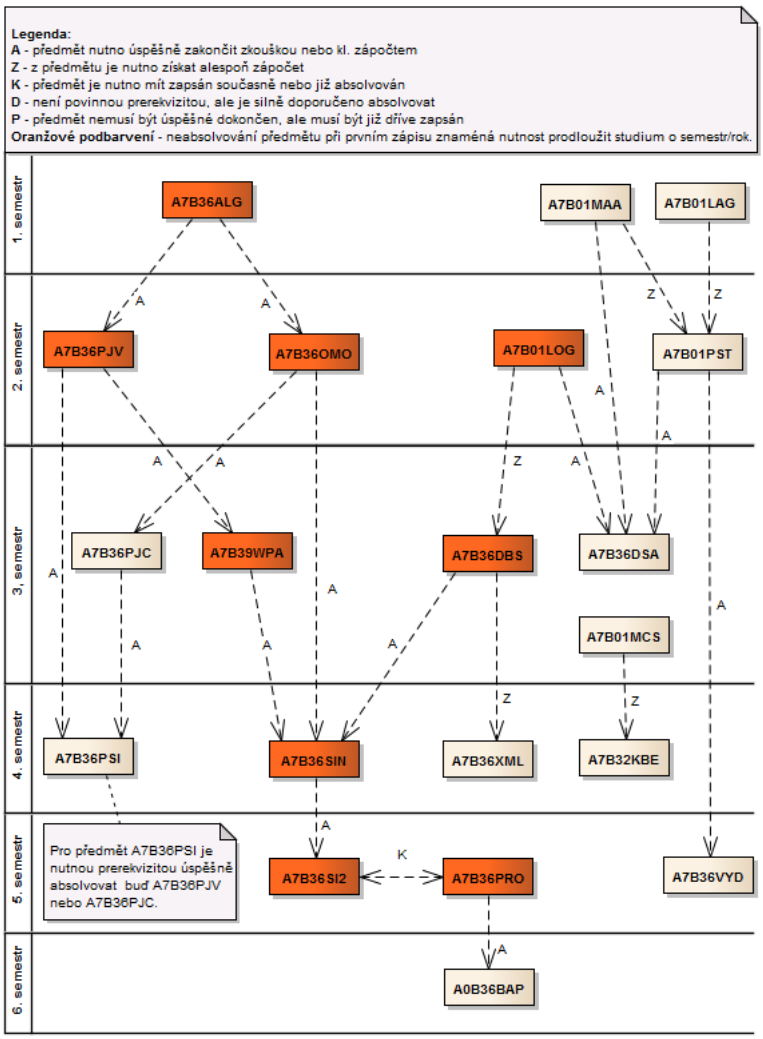
Knihovna dále kontroluje umístění popisků jednotlivých hran. Nedochází tak na rozdíl od JGraph k situacím, kdy jsou popisky přes sebe. Navíc jsou všechny popisky vykreslovány až po vykreslení všech hran, takže také nedochází k překrytí samotnými hranami. Obrázek 25 ukazuje situaci, při které došlo k posunu popisku. Kvůli zvýšení přehlednosti jsou také jednotlivým hranám a popiskům přiřazovány různé barvy. V případě, že dojde ke koncentraci více hran, tak lze přiřadit správný popisek k odpovídající hraně.

Je tedy zřejmé, že vlastní knihovna řeší funkční nedostatky, které byly nalezeny v rámci analýzy knihovny JGraph. Podařilo se také odstranit některé estetické nedostatky. Vlastní knihovna je tedy vhodnější pro využití, než knihovna JGraph.

8.1.2 Porovnání s ručně vytvořenými grafy

V zadání práce byl přiložen odkaz na webovou stránku http://stm.fel.cvut.cz/prerekvizity, kde jsou umístěny čtyři ručně vytvořené grafy pro jednotlivé zaměření programu STM. Tyto grafy mohou sloužit k porovnání grafů aplikace oproti ručně vytvářeným grafům.

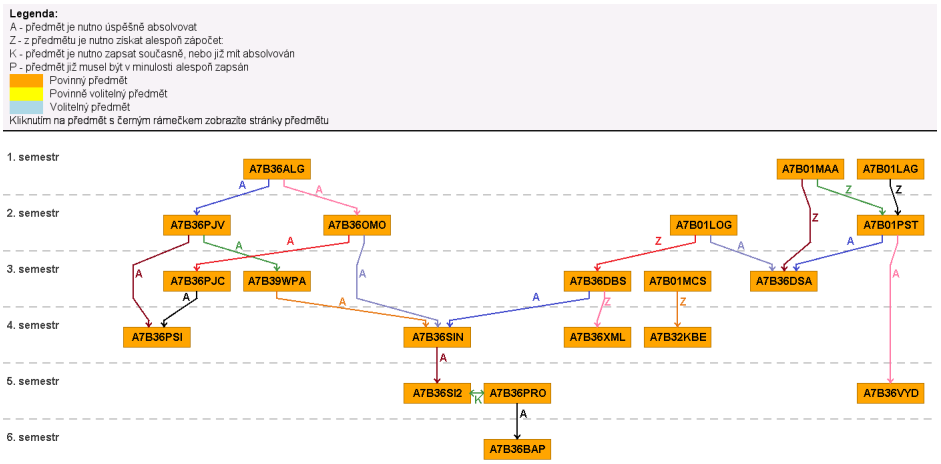
V následující části jsou porovnány tyto grafy se stejnými grafy vytvořenými touto aplikací. Je však nutné upozornit, že byly manuálně dodány informace o předmětech a prerekvizitách. Důvodem je, že dodaný soubor obsahující prerekvizity neobsahuje všechny prerekvizity tak, jak jsou na ručně vytvářeném grafu zobrazeny. U několika konektorů se navíc vyskytuje prerekvizita typu D, která je popsána jako doporučení. Tedy neexistuje omezení pro zápis předmětu, ale je doporučeno dříve studovat vyznačený předmět. Tento typ prerekvizity se však nikde v dodaném souboru nevyskytuje. Pokud by tedy měla aplikace reálně tyto grafy generovat, bylo by nutné doplnit chybějící prerekvizity do souboru prerekvizit.



Obrázek 26 Ručně vytvořený graf pro STM Softwarové inženýrství [28]

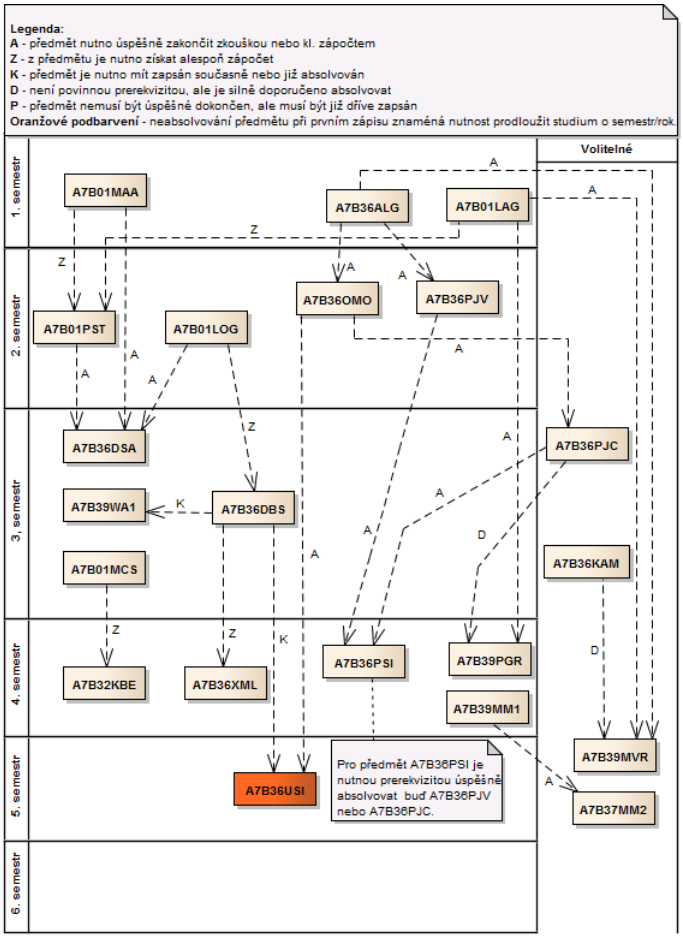
Obrázek 26 reprezentuje ručně vytvářené grafy. Oproti obrázkům z vlastní knihovny si lze povšimnout čtyř patrných rozdílů:

- Rozdíl ve způsobu barvení uzlů. Zatímco u ručně vytvářeného grafu jsou uzly bílé, případně oranžové v případě předmětů, které musí být splněny na první zápis. Grafová knihovna používá různé barvy uzlů dle toho, zda je pro daný studijní plán povinný, povinně volitelný, nebo volitelný.
- Různé obarvení hran v případě grafů z vlastní knihovny. Účelem je čistě umožnit lepší rozlišení popisků a hran v případě, kdy dochází k jejich zvýšené koncentraci.
- Možnost mít v rámci jednoho semestru dva uzly pod sebou a snížit tak šířku grafu. Vlastní knihovna toto neumožňuje. Hlavním důvodem je vysoce zvýšená náročnost implementace vykreslování hran a kontroly situací, kdy je možné tento krok provést.
- Přidání poznámky k předmětu. Jelikož jsou poznámky ručně vytvářeny na základě osobní znalosti tvůrce grafu a nelze je přímo získat, tak není možné je reprezentovat automaticky fungující knihovnou.



Obrázek 27 Automaticky vytvořený graf pro STM Softwarové inženýrství

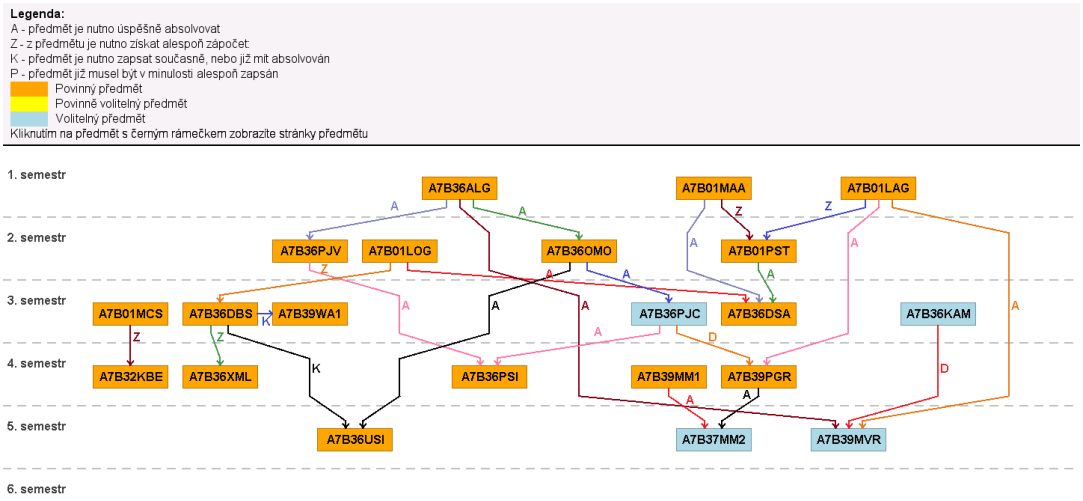
Obrázek 27 ukazuje stejný graf jako ručně vyrobený Obrázek 26. Ručně vytvořený obrázek je užší a esteticky příjemnější. Nicméně grafový algoritmus dokázal zachovat ideální počet křížení.



Obrázek 28 Ručně vytvořený graf pro STM Web a multimedia [29]

Obrázek 28 zachycuje složitější graf. Již se zde objevuje více křížení. Navíc se zde vykytuje oddělený sloupec, pro volitelné předměty. Jak již bylo řečeno, vlastní knihovna

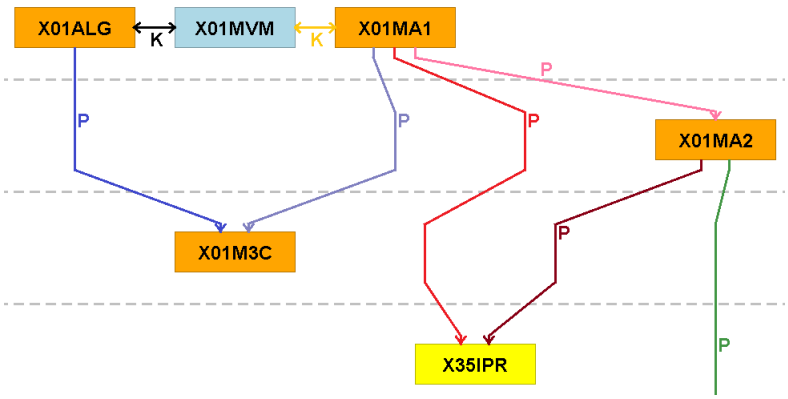
takto předměty neodděluje, ale k rozlišení využívá různé obarvení uzlů.



Obrázek 29 Automaticky vytvořený graf pro STM Web a multimedia

V případě automaticky vygenerovaného grafu pro obor Web a multimedia již došlo k více křížením, než při ideálním řešením. Vzhledu jako na ručně vytvářeném grafu nelze z principu s používaným algoritmem dosáhnout. Oddělení předmětů na povinné a volitelné by silně narušovalo princip použitého algoritmu, který se snaží zkracovat délky hran umisťováním na průměr. Ve větší míře se zde také ukazuje problém s dlouhými hranami, které graf rozšiřují. To je navíc umocněno tím, že ruční graf má v jednom semestru až tři předměty umístěny do jednoho sloupce, což automatická knihovna neumí.

8.1.3 Známé vady



Obrázek 30 Estetické vady

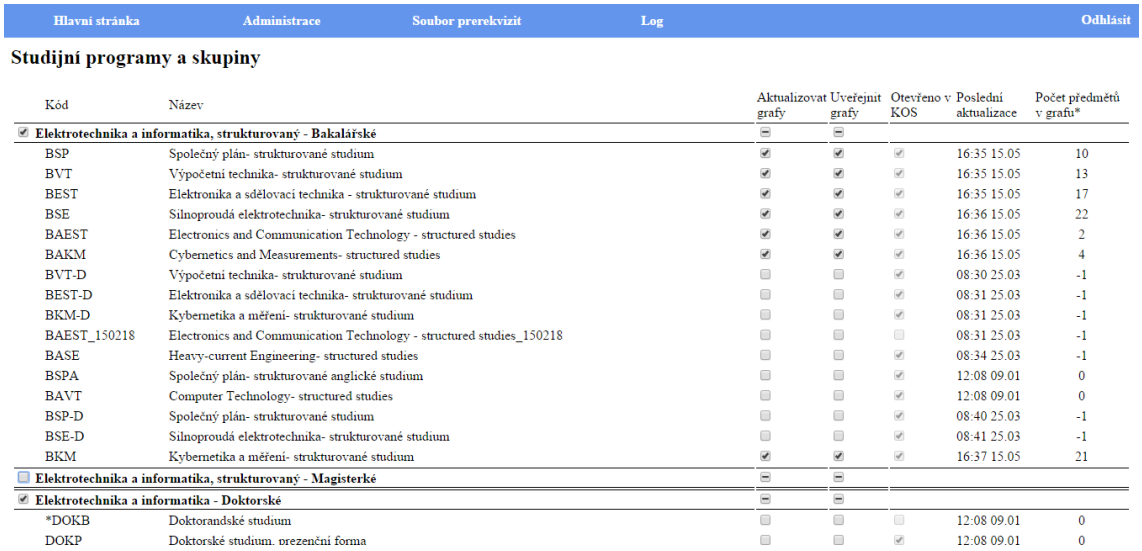
Příklad estetické vady ukazuje Obrázek 30. Jmenovitě se jedná o nedostatky dvojice dlouhých konektorů z uzlu X01MA1, které jsou nevzhledně zalomeny. Problémem je zde kombinace navázanosti pozic uzlů při výpočtu rozvržení na výsledné souřadnice a způsobu vykreslování konektorů, který více upřednostňuje zajištění dostatečné mezery. Problém by šlo také případně řešit rozšiřováním grafu, některé grafy by se ale mohly stát naopak příliš širokými. Druhým řešením by bylo umožnit uzlům mít různou šířku,

pak by mohly být pomocné uzly pro dlouhé hrany užší, větším problémem by pak ale bylo jejich narovnávání a také celková složitost algoritmu.

Další potenciální vady se mohou projevit v případech grafů s mnoha uzly a hranami. Jak již bylo zmíněno v kapitole 5.3, výsledky grafového algoritmu jsou částečně závislé i na pořadí vstupních dat, nedá se tak vždy zaručit stejná kvalita pro výsledné grafy. Toto je však problém většiny známých hierarchických grafových algoritmů. Algoritmus byl tedy nastaven tak, aby byly jeho výsledky kvalitní v případě stejného typu grafů, jako jsou současné grafy.

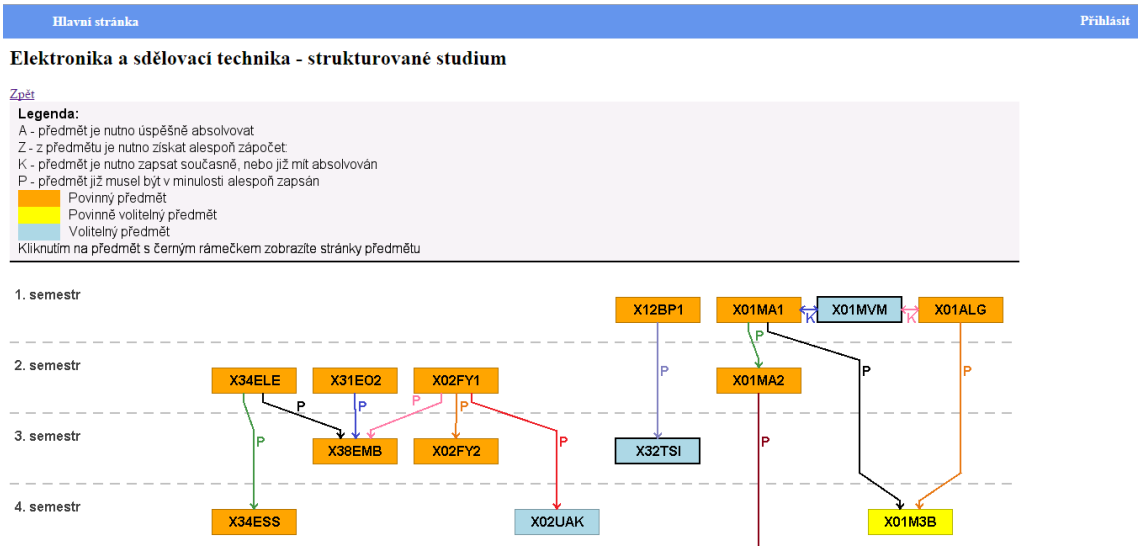
8.2 Webová aplikace

Webová aplikace splňuje všechny vlastnosti uvedené v požadavcích. Tedy umožňuje uživatelům zobrazit si požadované grafy a administrátorům spravovat aplikaci. V následující části jsou ukázky prostředí webových stránek, konkrétně administrátorského rozhraní a zobrazeného grafu.



Obrázek 31 Administrátorské rozhraní

Obrázek 31 představuje administrátorské rozhraní. Na vrchu stránky je menu v případě přihlášeného administrátora. Obsahem stránky je tabulka zobrazující programy a jejich studijní plány. V pravé části tabulky jsou nastavení daných plánů, datum posledního vygenerování grafů a aktuální počet uzlů grafu. Ze stránky administrace je také možné aktualizovat současné informace o programech, studijních plánech a také stáhnout aktuální informace o jednotlivých předmětech. Stránka také umožňuje vyvolat přegenerování grafů.



Obrázek 32 Zobrazení grafu ve webové aplikaci

Obrázek 32 se zobrazeným grafem. Stránka obsahuje legendu grafu a samotný graf.

8.3 Možná vylepšení

Možná vylepšení se dají rozdělit na vylepšení webové aplikace a vylepšení grafového algoritmu.

8.3.1 Grafový algoritmus

V rámci grafového algoritmu by se dala vylepšit práce s pomocnými uzly dlouhých hran. Současné fungování pomocných uzlů vede k tomu, že dlouhé hrany místy nepůsobí příliš esteticky. Některé grafy jsou také poměrně dost široké. Problémem jsou zde opět pomocné uzly, které zabírají stejně širokou pozici, jako normální uzel. Přitom využívají jenom malou část svého vymezeného prostoru.

8.3.2 Webová aplikace

Ve webové aplikaci by bylo potřeba vylepšit vzhled uživatelského prostředí, které je poměrně holé. Také v případě velkého množství grafů může být pro uživatele obtížnější najít jím požadovaný graf. V současnosti je možnost nastavení různých hodnot přes konfigurační soubory, jejichž obsah se načítá při startu. V případě některých hodnot by bylo vhodnější umožnit jejich změnu za běhu v rámci administrátorského rozhraní.

9 Závěr

Cílem práce bylo vytvořit systém, který by umožnil vytvářet vizualizace prerekvizit automaticky. Nejprve došlo ke specifikaci požadavků na aplikaci a také grafový algoritmus. Dále byla v rámci práce byla provedena analýza možných grafových algoritmů na generování prerekvizit předmětů. Z analýzy vyplynulo, že žádné rozvržení nesplňuje všechny požadavky, ale nejvhodnějším algoritmem je hierarchické rozvržení. Ve zkoumaných grafových knihovnách však neexistuje dostatečně kvalitní implementace hierarchického rozhraní. Z možností upravit existující implementaci, nebo vytvořit vlastní, byla zvolena možnost vytvořit vlastní implementaci knihovny, která by poskytovala upravený algoritmus hierarchického rozvržení. Byla tedy navržena a implementována vlastní knihovna, která vyhovuje všem požadavkům kladeným na výsledné grafy. K ovládání vzniklé knihovny byla v rámci práce vytvořena také webová aplikace, která s pomocí knihovny vytváří a prezentuje grafy s vizualizacemi prerekvizit.

Webová aplikace může fungovat automaticky díky spolupráci s univerzitním systémem KOSapi a dodaného souboru s prerekvizitami. Umožňuje v nastavených časech aktualizovat informace právě proti systému KOSapi a dle nastavení aktualizovat vybrané grafy. Aplikace má také administrátorské rozhraní, skrz které může administrátor sám aktualizovat informace, či přegenerovat grafy. K nahlédnutí má také logy se záznamy o průběhu tvoreb grafů.

Vytvářené grafy byly porovnány s dodanými ručně vytvořenými grafy. Slabou stránkou vlastní knihovny je pak práce s dlouhými hranami. I tak se ale daří vytvářet kvalitní grafy s jen minimálním nárůstem křížení.

Příloha A

Manuál k nasazení

Pro správné fungování aplikace je nutné před spuštěním správně inicializovat databázi, vyplnit konfigurační soubory a pro získání přístupu ke správě aplikace je nutné přidat administrátora do databáze. Také je nutné provést konfiguraci testovacího prostředí.

A.0.3 Inicializace databáze

Pro inicializaci databáze je připraven skript ve složce WEB-INF/dbinit.sql. Skript přidá potřebné tabulky do databáze.

A.0.4 Konfigurační soubory aplikace

Konfigurační soubory pro aplikaci se nacházejí ve složce WEB-INF/confproperties.

Do souboru dbconf.properties je nutné vyplnit adresu, na které je databáze a přihlašovací jméno a heslo pro přihlášení do databáze, případně je možné vyměnit ovladač dané databáze. Je však potřeba vzít v úvahu, že je následně nutné dodat knihovnu obsahující daný ovladač.

Druhým konfiguračním souborem pro nastavení samotné aplikace je soubor configuration.properties. Zde je nutné vyplnit cestu, kam se budou ukládat vytvářené grafy a také cestu k xls souboru s prerekvizitami předmětů. Lze také nastavit, které sloupce, číslované od 0, budou použity pro čtení daných informací. Dále je možné provést konfiguraci pro autentizační server OAuth. Další možností je změnit nastavení času, ve kterém jsou grafy automaticky generovány zadáním CRON výrazu. Poslední možnou volbou je zda se má aplikace sama při startu inicializovat. V rámci inicializace aplikace vyhledá programy a studijní plány a pokusí se pro ně vygenerovat grafy. Tato operace může trvat několik minut.

A.0.5 Konfigurace administrátorského účtu

Pro získání přístupu ke správě aplikace je nutné vložit záznam o administrátorovi do databáze. Konkrétně do tabulky adminuser. Pro přihlášení jsou nutné hodnoty email, pass a salt obsahující přihlašovací login, heslo a sůl. Login a sůl se do databáze zadává v textové formě. Ke tvorbě hesla zadaného do databáze lze využít webovou stránku /passhelp.xhtml, nebo lze heslo vytvořit ručně následujícími kroky.

- Vytvořit hash z hesla a a soli pomocí SHA-256.
- Spojit hash z hesla a soli dohromady pomocí operace xor přes jednotlivé byte.
- Spojený hash opět zpracovat hashovací funkcí SHA-256.
- Získaný hash převést do hexadecimální formy. Výsledkem je dané heslo.

A.1 Konfigurace testovacího prostředí

Pro správnou funkčnost testů je nutné nastavit konfigurační soubory pro testy. Soubory jsou uloženy v src:test/java/cz/vopbp/testConf. Vyplnění těchto souborů je stejné jako celé aplikace, nicméně pro potřeby testování by měla být vypnuta automatická inicializace. Testování musí probíhat na oddělené databázi, tato databáze musí být nejprve inicializována dle dodaných skriptů u testovacích konfiguračních souborů.

Příloha B

Obsah CD

Obsah přiloženého CD je rozdělen do následujících adresářů:

- Adresář **text** obsahuje text této práce a dále adresář se zdrojovými kódy textu v Latex.
- Adresář **aplikace** obsahuje zkompilovanou aplikaci ve formátu .war a adresář se zdrojovými kódy aplikace.
- Adresář **spoustecibalik** obsahuje Apache Tomcat s nasazenou aplikací a připravenou databázi pro snadné vyzkoušení aplikace.

Literatura

[1] S. NIKOLOV, Nikola a Patrick HEALY. Hierarchical Drawing Algorithms. In: R. Tamassia, Editor.*Handbook of graph drawing and visualization.* CRC Press, 2013, s. 409-453.ISBN 978-158-4884-125. Dostupné z: https://cs.brown.edu/~rt/gdhandbook/chapters/hierarchical.pdf

[2] REYNOLDS, Jason. *A Hierarchical Layout Algorithm for Drawing Directed Graphs.* Kanada, Ontario, Queen's University Kingston: 1997. Dostupné z: http://www.collectionscanada.gc.ca/obj/s4/f2/dsk2/ftp04/mq20694.pdf. Thesis. Queen's University Kingston, Department of Computer and Intormation Science. Vedoucí práce Dr. D. Rappaport.

[3] M. KORNAROPOULOS, Evgenios. *Dominance Drawing of Non-Planar Graphs.* Řecko, Heraklion, University of Crete: 2012. Dostupné z: http://cs.brown.edu/people/evgenios/files/MS_ Thesis.pdf. Master's thesis. University of Crete, School of Sciences and Engineering, Computer Science Department. Vedoucí práce Prof. Ioannis G. Tollis.

[4] G. KOBOUROV, Stephen. *Spring Embedders and Force Directed Graph Drawing Algorithms* [online]. CoRR, 2012, abs/1201.3011 University of Arizona. [cit. 6.1.2015] Dostupné z: http://arxiv.org/pdf/1201.3011.pdf

[5] RUSU, Adrian. Tree Drawing Algorithms In: R. Tamassia, Editor. *Handbook of graph drawing and visualization.* CRC Press, 2013, s. 155-192.ISBN 978-158-4884-125. Dostupné z: https://cs.brown.edu/~rt/gdhandbook/chapters/trees.pdf

[6] CHIMANI, Markus, Carsten GUTWENGER, Petra MUTZEL a Hoi-Ming WONG. Upward Planarization Layout. In: *LNCS 5849: 17th International Symposium on Graph Drawing 2009, Chicago (GD09)* Springer, 2010. Dostupné z: http://ls11-www.cs.uni-dortmund.de/people/chimani/files/UpwardPlanLayout_ pre.pdf

[7] GERGERITAKI, M. a A. VALSAMAK I. *Dominance Drawing.* [online]. 2002 [cit. 6.1.2015]. Dostupné z: www.csd.uoc.gr/~hy583/papers/ch12.pdf

[8] HANRAHAN, Pat. *Trees and Graphs.* [online]. [cit. 6.1.2015]. Dostupné z: http://graphics.stanford.edu/courses/cs448b-02-winter/lectures/treesgraphs/tree.graph.pdf

[9] F. CRUZ, Isabel a Roberto TAMASSIA. *How to Visualize a Graph: Specification and Algorithms.* [online]. [cit. 6.1.2015]. Dostupné z: ftp://ftp.cs.brown.edu/pub/papers/compgeo/draw-tutorial-1.pdf

[10] *JGRAPH* [online]. JGraph Ltd. [cit. 12.12.2014]. Dostupné z: www.jgraph.com

[11] *JGRAPHX* [online]. JGraph Ltd. [cit. 6.1.2015]. Dostupné z: github.com/jgraph/jgraphx

[12] *Gephi* [online]. Gephi Consortium [cit. 6.1.2015]. Dostupné z: gephi.github.io

[13] *GraphStream* [online]. GraphStream team [cit. 6.1.2015]. Dostupné z: graphstream-project.org

[14] *Grappa* [online]. Graphviz team [cit. 6.1.2015]. Dostupné z: www.graphviz.org

[15] *Prefuse* [online]. Prefuse [cit. 6.1.2015]. Dostupné z: github.com/prefuse/Prefuse

[16] *Prefuse* [online]. Prefuse [cit. 6.1.2015]. Dostupné z: sourceforge.net/p/prefuse/discussion

[17] *JUNG* [online]. JUNG framework development team [cit. 6.1.2015]. Dostupné z: http://jung.sourceforge.net/

[18] *JSF* [online]. Tutorialspoint [cit. 6.1.2015]. Dostupné z: http://www.tutorialspoint.com/jsf/

[19] *Postgres* [online]. PosgreSQL Global Development Group [cit. 6.1.2015]. Dostupné z: http://www.postgresql.org/

[20] *Hibernate* [online]. Red Hat [cit. 6.1.2015]. Dostupné z: http://hibernate.org/

[21] *Spring framework* [online]. Pivotal Software, Inc. [cit. 6.1.2015]. Dostupné z: https://spring.io/

[22] *KOSapi* [online]. ČVUT [cit. 3.5.2015]. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki

[23] *Apache POI* [online]. The Apache Software Foundation [cit. 3.5.2015]. Dostupné z: https://poi.apache.org/

[24] *Apache Maven Project* [online]. The Apache Software Foundation [cit. 4.5.2015] Dostupné z: https://maven.apache.org/

[25] *Apps Manager* [online]. České vysoké učení technické v Praze [cit. 6.5.2015] Dostupné z: https://auth.fit.cvut.cz/manager/index.jsf

[26] M. REINGOLD, Edward a John S. TILFORD *Tidier Drawings of Trees* [online]. IEEE Transactions on software engineering 1981, VOL. SE-7, NO. 2., s. 223-228. doi:10.1109/TSE.1981.234519. [cit. 6.5.2015] Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.3559&rep=rep1&type=pdf

[27] *OAuth 2.0* [online]. The OAuth community [cit. 6.5.2015] Dostupné z: http://oauth.net/

[28] prerekvizity_STM_2012_SI. In: ČVUT FEL STM [online]. ČVUT FEL 2012. [cit. 6.5.2015]. Dostupné z: http://stm.fel.cvut.cz/prerekvizity

[29] prerekvizity_STM_2012_MW. In: ČVUT FEL STM [online]. ČVUT FEL 2012. [cit. 6.5.2015]. Dostupné z: http://stm.fel.cvut.cz/prerekvizity

[30] *JUnit* [online]. JUnit [cit. 6.5.2015]. Dostupné z: http://junit.org/

[31] *The DOT Language* [online]. Graphviz team [cit. 17.5.2015]. Dostupné z: http://www.graphviz.org/doc/info/lang.html