FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vizualizace cest v grafu
Student:	Vojtěch Polcar
Vedoucí:	Ing. Lukáš Bařinka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Analyzujte problematiku 2D/3D vizualizace grafů a její možná řešení. Navrhněte, naimplementujte a otestujte program pro vizualizaci cest v zadaném grafu. Výsledný program bude disponovat zároveň GUI i API pro tvorbu a vizualizaci cest. V rámci programu řešte problémy prohledávání, orientace a navigace v rozsáhlých grafech. Řešení přizpůsobte problematice informační architektury webu FIT a otestujte je na datech z prototypu pro hledání souvislostí obsahu webu FIT.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 1. března 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Vizualizace cest v grafech

Vojtěch Polcar

Katedra softwarového inženýrství
Vedoucí práce: Ing. Lukáš Bařinka

14. května 2018

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, Ing. Lukáši Bařinkovi za možnost zpracovat toto téma a jeho rady a nápady v počátcích její tvorby.

Dále bych chtěl poděkovat své rodině, přátelům a přítelkyni za podporu ve chvílích, kdy jsem sám nevěřil ve své schopnosti a motivaci k úspěšnému dokončení této práce. Největší poděkování patří zejména Anežce Polcarové a Anně Prchalové za pomoc s hledáním chyb, testováním aplikace, za jejich poznatky a za usměrňování mých nápadů.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2018 .....

Odkaz na tuto práci

Polcar, Vojtěch. *Vizualizace cest v grafech.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: ⟨`https://gitlab.fit.cvut.cz/polcavoj/PathViz`⟩.

Abstrakt

Cílem této bakalářské práce je vytvořit aplikaci pro menší a rozsáhlé grafy, která se zabývá vizualizací grafů a cest v nich. Aplikace umožní samotné grafy vytvářet. V rámci této bakalářské práce jsou nejdříve analyzovány metody vizualizace grafů a jejich existující řešení. Dále jsou zanalyzována a popsána testovací data, dle kterých je výsledná aplikace otestovaná.

Na základě této analýzy a testovacích dat jsou specifikovány požadavky na vytvoření nové aplikace. Následně je vytvořen návrh aplikace, který je přizpůsoben vytvořeným požadavkům. Navržená aplikace je následně implementována a otestovaná.

Vytvořená aplikace umožňuje tvorbu grafů, jejich editaci, prohledávání grafu za účelem vizualizace cest a importování a exportování dat.

Klíčová slova webová aplikace, vizualizace cest v grafech, prohledávání grafu, rozsáhlé grafy, JavaScript, vis.js

Abstract

The objective of this bachelor thesis is to create an application for small and extensively graphs, which will be focused on the visualization of graphs and paths in them. The application allows a creation of the graphs. In this bachelor thesis are analyzed methods of visualization of graphs and their existing solutions. In addition, test data are analyzed and described, which are used in the testing of final application.

Based on analysis and test data are specified functional requirements on creating a new application. Then an application design is created, which is adapted to the specified requirements. The proposed application is implemented and tested.

The final created application allows creation of graphs, it's editation, searching in graph for visualize paths and import and export data.

Keywords web application, graph path vizualization, searching in graph, extensive graphs, JavaScript, vis.js

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době existuje mnoho programů a aplikací, které se zabývají vyhledáváním cest v grafech. Pod tímto obecným problémem si můžeme představit například mapu a následné vyhledání trasy mezi dvěma městy. Zde nás pak může zajímat nejrychlejší, nejkratší nebo neplacená trasa. Když se zaměříme na grafy s velkým množstvím uzlů a cest, jejich procházením a vizualizací se zabývají převážně komerční a placené programy. Je to z toho důvodu, že práce s tak rozsáhlým a složitým množstvím dat je náročná na realizaci.

Jedním z problémů jsou rozsáhlé webové stránky se spoustou provázaností a příbuzným obsahem. Mezi dvěma stránkami může existovat více cest, jak se od jedné k druhé dostaneme, ale jednotlivé uživatele budou zajímat rozdílné vztahy, které mezi nimi existují.

Výsledek práce má za cíl pomoci při tvorbě nové webové stránky FIT ČVUT v Praze, která obsahuje obrovské množství stránek, které mezi sebou mají různé druhy vazeb. Při vytváření je potřeba zjišťovat spojitost mezi danými stránkami, která se obtížně domýšlí bez její vizualizace. Kromě toho byla mojí motivací při výběru tématu práce, možnost pracovat se zajímavou oblastí vizualizace grafů.

Cílem mé práce je zanalyzovat různé možnosti vizualizace cest v grafech, porovnat již existující řešení a na jejich základě navrhnout a implementovat vhodnou aplikaci pro vizualizaci cest v grafech nad zadanými daty.

V první části této práce se věnuji analýze informační architektury webu FIT, existujícím řešením a nástrojům pro vizualizaci grafů a možnosti pro 2D/3D vizualizaci cest v grafech. Následně na základě analýzy navrhnu a implementuji aplikaci, která se bude zabývat zadanými daty.

Kapitola 1

Analýza

Tato kapitola se zabývá analýzou problematiky vizualizace grafů, rozborem testovacích dat pro které bude aplikace vytvořena a prozkoumáním existujících řešení v oblasti vizualizace grafů.

Nejprve se zaměříme na určení toho, co je to graf a cesty v něm, společně s jeho reprezentací a různými druhy vykreslování. Následně rozebereme existující knihovny a aplikace, které se touto problematikou zabývají. Na základě teorie analyzujeme strukturu webové stránky FIT a vymezíme požadavky a případy užití na výslednou aplikaci.

1.1 Grafy a jejich reprezentace

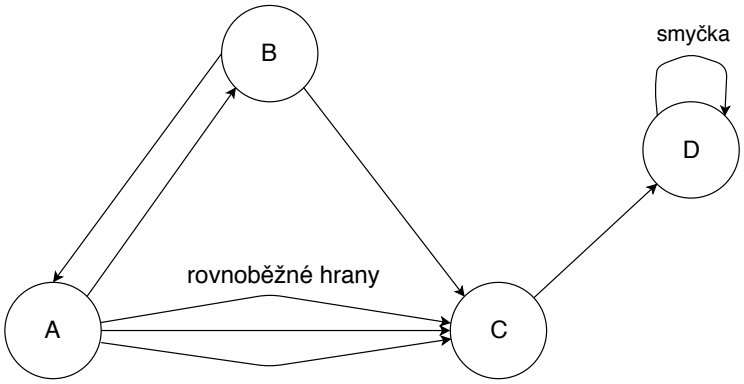
Nejprve se podíváme na samotné grafy, cesty v grafech a to jakým způsobem se graf reprezentuje. Je více druhů grafů a rozlišují se svými vlastnostmi a definicemi. Ať už se budeme bavit o grafech orientovaných, neorientovaných, planárních, multigrafech nebo o různých druzích stromů, vždy je potřeba je správně zadefinovat.

Definice 1. *Neorientovaný graf G je uspořádaná dvojice (V, E), kde V je neprázdná konečná množina uzlů a E je množina hran. [1]*

Definice 2. *Orientovaný graf G je uspořádaná dvojice (V, E), kde V je neprázdná konečná množina uzlů a $E \subseteq V \times V$ je množina orientovaných hran.*

Rozdíl mezi orientovaným a neorientovaným grafem je pouze v množině hran. Tato množina se u orientovaných grafů skládá z uspořádaných dvojic různých uzlů (u, v), kde *u* je předchůdce *v* a *v* je následník *u*. Grafu u kterého povolíme smyčky (u = v) a rovnoběžné hrany (dvě hrany mající stejný počáteční a koncový uzel) říkáme Multigraf(1.1).

U vizualizace grafů nás dále může zajímat zda je daný graf planární. Pro jednoduchost jsou to takové grafy, které se dají v rovině nakreslit bez křížení



Obrázek 1.1: Multigraf

jakékoli hrany grafu s jeho dalšími hranami. Tuto vlastnost splňují převážně menší grafy, které neobsahují velké množství hran.

Definice 3. *Posloupnost $(v_0, e_1, v_1, e_2, \ldots, e_n, v_n)$ se nazývá sled v orientovaném grafu G, pokud $v_i \in V$ pro všechna $i \in \{0, \ldots, n\}$ a $e_i = (v_{i-1}, v_i) \in E(G)$ pro všechna $i \in \{1, \ldots, n\}$.*

Pro neorientované grafy je definice analogická. Z definice je vidět, že ve sledu povolujeme opakování vrcholů. Pokud bychom opakování nedovolili, bude se jednat o **cestu**.

1.1.1 Reprezentace grafu

Možností jak reprezentovat grafy v paměti nebo v souboru, který se bude dále zpracovávat, je hned několik. Nejprve se podíváme na dva klasické způsoby uložení v paměti.

Definice 4. *Nechť G = (V,E) je graf s $V = \{v_1, v_2, \ldots, v_n\}$. Matice sousednosti grafu G je čtvercová matice $A_G = (a_{ij})_{i,j=1}^{n}$ definovaná předpisem:*

$$a_{ij} = \begin{cases} 1, & \textit{když } \{v_i, v_j\} \in E \\ 0, & \textit{jinak} \end{cases}$$

Tento jednoduchý způsob ukládání je při použití 2D pole rychle přístupný, nicméně při větším počtu uzlů velmi nepřehledný. Dalším způsobem jak uložit graf do paměti může být například seznam sousedů. Pro každý vrchol v grafu G = (V,E) uchováváme seznam jeho sousedů - to se dá realizovat například spojovým seznamem.

1.1.1.1 JSON a XML

V rámci webové aplikace bude potřeba uchovávat data v takových formátech, aby se dala lehce importovat/exportovat. Dva podobné formáty pro uchová-

vání obecných dat jsou JSON (JavaScript Object Notation) a XML (eXtensible Markup Language) [2]. Oba formáty jsou dobře čitelné a mohou být syntakticky analyzovány a použity mnoha programovacími jazyky.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph id="G" edgedefault="directed">
    <node id="A"/>
    <node id="B"/>
    <node id="C"/>
    <node id="D"/>
    <edge source="A" target="B"/>
    <edge source="A" target="C"/>
    <edge source="A" target="C"/>
    <edge source="A" target="C"/>
    <edge source="B" target="A"/>
    <edge source="B" target="C"/>
    <edge source="C" target="D"/>
    <edge source="D" target="D"/>
  </graph>
</graphml>
```

Listing 1.1: Multigraf z obrázku 1.1 ve formátu GraphML.

XML je značkovací jazyk, kde se všechna data uzavírají do otevíracích a uzavíracích značek. Udržuje stromovou strukturu, což je velmi užitečné při ladění. Nespornou výhodou oproti JSON formátu je to, že díky stromové struktuře a způsobu zapisování dat, lze ukládat metadata jako atributy. U JSON formátu je nutné pro tyto data vytvořit nové objekty a vložit je do nich.

JSON formát je určený primárně pro přenos dat, která jsou organizovaná v polích nebo objektech. Jednou z výhod JSON formátu je to, že je jeho zápis kratší, protože zde nejsou potřeba žádné uzavírací značky. Všechna data jsou ukládána pomocí dvojice `klíč : hodnota`. Soubor v JSON formátu, který má stejný datový obsah jako soubor v XML formátu, zabírá na disku méně místa a je rychleji zpracováván. Bezesporu největší výhodou tohoto formátu je to, že JSON je podmnožinou jazyka JavaScript, takže se při programování v jazyku JavaScript velmi jednoduše používá.

V ukázkách 1.1 a 1.2 můžeme vidět rozdíl zápisu dat ve formátu XML a JSON. Délky zápisu jsou zde sice podobné, nicméně JSON formát máme připravený i na popis a zápis informací o jednotlivých uzlech, popřípadě hranách. Ukázka JSON formátu je zároveň vytvořena tak, jak ji bude možné importovat a exportovat do budoucí aplikace.

```
[{
    "id": "A",
    "label": "A",
    "connections": [
      { "from": "A", "to": "B", "arrows": "to" },
      { "from": "A", "to": "C", "arrows": "to" },
      { "from": "A", "to": "C", "arrows": "to" },
      { "from": "A", "to": "C", "arrows": "to" }
    ]
  },
  {
    "id": "B",
    "label": "B",
    "connections": [
      { "from": "B", "to": "A", "arrows": "to" },
      { "from": "B", "to": "C", "arrows": "to" }
    ]
  },
  {
    "id": "C",
    "label": "C",
    "connections": [
      { "from": "C", "to": "D", "arrows": "to" }
    ]
  },
  {
    "id": "D",
    "label": "D",
    "connections": [
      { "from": "D", "to": "D", "arrows": "to" }
    ]
}]
```

Listing 1.2: Multigraf z obrázku 1.1 ve formátu JSON.

1.2 Druhy vizualizace grafů

Grafy se dají vizualizovat různými způsoby. Tím nejjednodušším způsobem je použití klasického rozvržení, které je složené z uzlů a hran mezi nimi. Toto rozvržení zobrazuje vztahy mezi daty ve formě čar. Další možností jak přistupovat k vizualizaci, je využití technik prostorového vyplňování nebo uspořádání pomocí prostorového vnoření, které implicitně reprezentují vztahy mezi jednotlivými daty [3].

1.2.1 Klasické uspořádání pomocí uzlů a hran

Základním požadavkem pro tento typ zobrazení je výpočet souřadnic uzlů a reprezentace hran. Pro zlepšení čitelnosti by měl zobrazený graf splňovat co nejvíce následujících kritérií:

- hrany a uzly jsou rovnoměrně rozděleny,
- minimální křížení hran,
- symetrické podgrafy by se měly zobrazovat stejně,
- minimalizovat ohýbání hran,
- minimalizovat délky hran.

Bohužel je těžké kombinovat většinu těchto kritérií najednou, protože během vizualizace se některá z nich navzájem vylučují. Většina algoritmů, které se vizualizací zabývají, se snaží tato kritéria kombinovat a zobrazovat graf co nejčitelněji. Při výběru rozvržení je velmi důležité brát ohled na vstupní data, která se mají zobrazovat.

1.2.1.1 Force-directed metoda

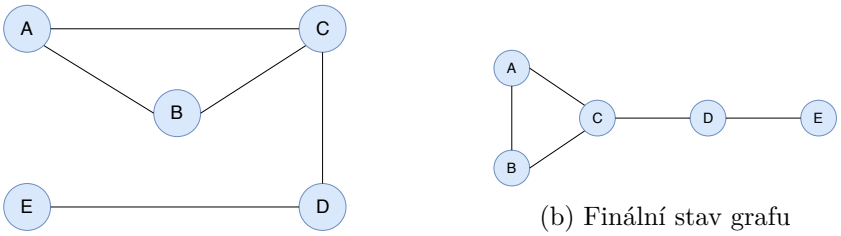
Jedna z nejvíce využívaných metod pro vizualizování grafů je Force-directed metoda [4]. Metoda k určení pozic jednotlivých uzlů a komponent využívá síly, které mezi uzly působí. Prvním typem jsou síly přitažlivé, které jsou založené na Hookovu zákonu a používají se k posouvání uzlů na opačných koncích hran k sobě. Dále jsou zde síly odpudivé, které jsou na základě nabitých částic (uzlů) použity k odpuzování všech komponent od sebe.

Na počátku algoritmu musí mít všechny uzly přiřazenou nějakou pozici (1.2a). Z těchto pozic se poté v iteracích počítají síly, které mezi nimi působí. Odpudivá síla (1.3a) je počítána pro každý uzel v závislosti na všech ostatních. Je to z důvodu, aby uzly, které nejsou propojené, neskončily na stejném nebo podobném místě.

Přitažlivá síla (1.3b) je naopak počítána pro každý uzel v závislosti na tom, se kterými uzly je propojen. Díky tomuto způsobu jsou uzly, které mají málo hran (nebo žádnou), odstrčeny mimo hlavní shluk, protože jsou odpuzovány všemi uzly a přitahovány pouze malým množstvím uzlů (nebo žádným). Tento jev funguje samozřejmě i opačně. Při velkém počtu sousedů je uzel blízko středu hluku uzlů.

Po spočtení všech těchto sil jsou všechny uzly současně posunuty do nově spočtené pozice. Tento postup se v iteracích opakuje a jejich počet závisí zejména na velikosti grafu a počátečnímu rozmístění uzlů. Výsledkem by měl být přehlednější graf (1.2b).

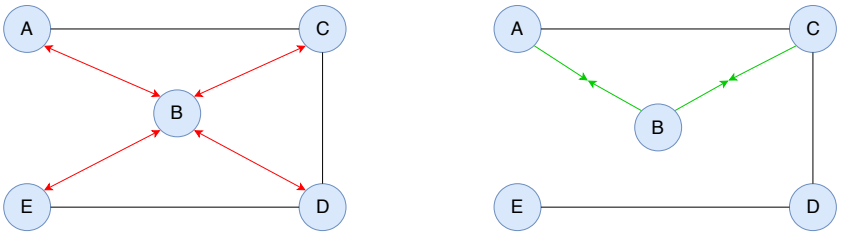
Existuje mnoho algoritmů a metod založených na Force-directed metodě, které jsou různě upravovány a vylepšovány.



(a) Počáteční stav grafu

(b) Finální stav grafu

Obrázek 1.2: Graf před (a) a po (b) aplikování Force-directed metody



(a) Odpudivé síly

(b) Přitažlivé síly

Obrázek 1.3: Odpudivé (a) a přitažlivé (b) síly působící na uzel B.

1.2.1.2 Topologické uspořádání

Tento přístup k rozvržení uzlů se skládá ze čtyř fází. První fáze se nazývá `dekompoziční` a graf je během ní rozdělen na mnoho podgrafů, dle topologických vlastností jednotlivých podgrafů. Pokud jsou například v jednom podgrafu uzly propojené ve formě stromu, tak se všechny tyto uzly spojí dohromady a vytvoří tzv. „meta-uzel“. Tento druh uspořádání dokáže rozpoznat několik druhů grafů – stromy, bipartitní grafy, úplné grafy a další.

Další fáze, která se nazývá `uspořádání dle vlastností`, rozvrhne meta-uzly z první fáze a vytvoří z nich jedno velké uspořádání, typicky takové, které bylo nejčastěji nalezeno jako meta-uzel v předchozí fázi.

Zbylé dvě fáze slouží převážně pro zlepšení přehlednosti a čitelnosti grafu. `Redukce křížení` eliminuje křížení hran v daném uspořádání a `eliminace překrývání` mění velikosti uzlů ve finálním rozvržení tak, aby k žádnému překrývání nedošlo.

Výstupem algoritmu je graf, ve kterém každý uzel reprezentuje podgraf v originálním grafu a každá hrana představuje spojení mezi těmito podgrafy. Tato technika může být užitečná při vizualizaci relativně rozsáhlých grafů.

1.2.2 Stromové uspořádání

Pro tento druh uspořádání již bylo navrženo mnoho algoritmů. Důvodem může být jednoduchost a popularita stromové struktury. U většiny algoritmů z této

sekce je obtížné najít co nejideálnější kořen stromu, podle kterého se graf zobrazí.

Většina algoritmů pro zobrazení grafu ve stromové struktuře vychází z klasického algoritmu pro `stromové uspořádání`, jehož implementace je přímočará. Tento základní algoritmus vykreslí graf jako stromovou 2D reprezentaci, algoritmus bohužel velmi špatně využívá místo a stane se nepřehledným ve chvíli, kdy je graf složitější. Tento problém se snaží řešit další algoritmy zobrazující stromové uspořádání, které vycházejí z tohoto klasického algoritmu.

Prvním takovým je algoritmus pro `paprskovité uspořádání`. V něm je nejdříve umístěn kořen stromu doprostřed kruhu a následně jsou kolem něj, dle jejich hloubky, rovnoměrně umísťovány další uzly. Nejvzdálenější uzly od kořene stromu jsou umístěny na okraji kruhu. Algoritmus dokáže lépe využívat prostor, ale je hůře čitelný.

Velmi podobné předchozímu algoritmu je i `balonové uspořádání`, které opět umístí kořen stromu doprostřed a jeho potomky rozmístí do kruhu kolem něj. Další hladiny stromu jsou vždy umísťovány do kruhu kolem jejich rodiče. Tento způsob zobrazování efektivně zobrazuje stromovou strukturu.

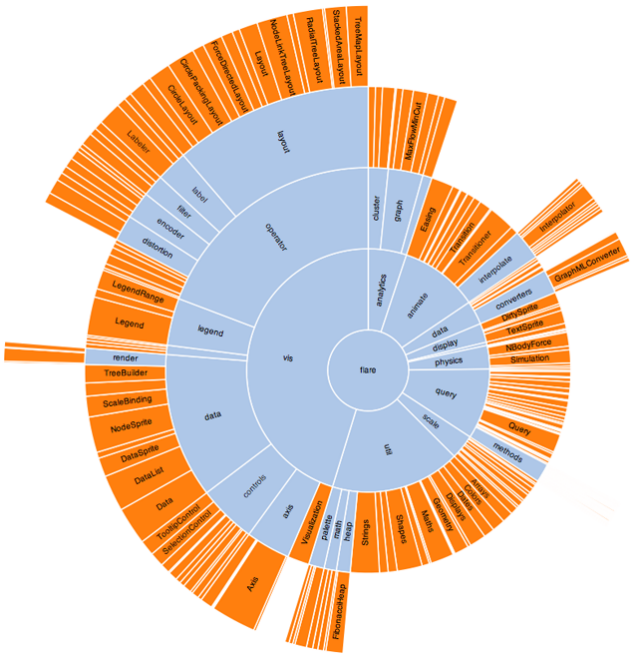
Všechna zmiňovaná stromová uspořádání fungují výborně s vyváženými stromy a produkují výsledek v lineárním čase. Algoritmus pro klasické zobrazení umožňuje lepší pochopení struktury grafu než další algoritmy, které lépe využívají vykreslovací plochu. U nich je problém identifikovat správný kořen stromu pro následnou vizualizaci.

1.2.3 Metody prostorového vyplňování

Dalším způsobem, jak můžeme přistupovat k vizualizaci dat jsou metody prostorového vyplňování, které se snaží maximálně využít prostor pro zobrazení. Tyto metody můžeme rozdělit na dva typy.

Prvním typem jsou `metody prostorového dělení`, kde jsou vztahy mezi uzly reprezentovány tak, že uzly přímo připojíme k sobě. Tyto metody často používají kruhové nebo paprskové zobrazení. Největším problémem tohoto uspořádání je to, že je těžké rozpoznat od sebe vztahy mezi uzly. Vazby rodič-potomek a potomek-potomek se v obou případech zobrazují jako sousedící. Dalším problémem je situace, kdy mají uzly mnoho potomků. V závěrečném zobrazení jsou pak potomci reprezentováni pouze úzkými obdélníky, které nestačí pro zobrazení názvu nebo barvy uzlu (1.4).

Druhým typem jsou uspořádání pomocí `prostorového vnoření`. Uzly jsou reprezentovány jako obdélníky a každý z nich je rozdělen na tolik částí, kolik má daný uzel potomků. Tento rozdělovací proces je zpracováván rekurzivně. Technika je populární, protože efektivně využívá zobrazovací plochu. Nevýhodou je špatná čitelnost hierarchie a provázanosti stromu a problém s interaktivitou – při velkém počtu potomků se budou objevovat malé obdélníky, s kterými půjde špatně manipulovat.



Obrázek 1.4: Graf pomocí metody prostorového dělení.[5]

1.2.4 3D uspořádání

K doplnění 2D reprezentace bylo mnoho druhů uspořádání rozšířeno i do 3D. Jako lidé si dokážeme lépe představit 3D vizualizaci některých modelů, díky podobnosti s reálným světem. 3D vizualizace ovšem čelí několika problémům, s kterými se musí vývojář vypořádat. Prozkoumávání grafu může být složité, protože se objekty ve 3D zobrazení můžou navzájem překrývat, s čímž souvisí, že jednou z výzev při využívání 3D zobrazení je vypořádání se s překrýváním uzlů a křížením hran. Velmi důležité je také zvolit si správné rozložení pro čitelnost informací na základě dat, ze kterých je graf sestaven.

Jedním ze zajímavých 3D uspořádání je technika zvaná `Conetree` (kuželovitý strom). Původně byla tato metoda vytvořena pro zobrazení stromů ve 3D prostoru. Kořen stromu je umístěn na vrchol kuželu a pod ním jsou ve vrstvách dle hloubky jeho potomci. Toto rozvržení má mnoho vrstev, kde každá reprezentuje hladinu stromu.

1.3 Vizualizace na webových stránkách

V internetových prohlížečích máme několik možností jak zobrazovat, vykreslovat a pracovat s grafickými prvky. V této části si projdeme ty nejvíce používané, které může programátor použít a jsou využívané mnoha knihovnami pro práci s grafikou, přímo na webové stránce.

1.3.1 HTML a CSS

HTML je základním jazykem pro tvorbu a stylizaci webových stránek. Kaskádové styly (CSS) byly vytvořeny za účelem oddělení vzhledu a stylizování stránky od jeho struktury a obsahu. Tyto základní technologie pro tvorbu webových stránek dokážou vytvářet prakticky jakékoliv grafické obrazce [6], umožňují interakci s uživatelem a tvorbu jednoduchých animací.

HTML a CSS jsou primárně určené k tvorbě obsahu, struktury a vzhledu webových stránek. Práce s grafickými objekty je v nich zbytečně složitá a nepřehledná. V HTML5 ale existují elementy a prvky, které jsou na tvorbu grafiky zaměřené.

1.3.2 SVG - Škalovatelná vektorová grafika

SVG [7] je základním otevřeným formátem pro vektorovou grafiku na webových stránkách. Je to značkovací jazyk a formát souboru, který popisuje 2D vektorovou grafiku pomocí XML. HTML5 umožňuje vložit kód SVG obrázku přímo do kódu webové stránky.

SVG je ideálním formátem pro jednoduchou grafiku – grafy, binární stromy, diagramy apod. U SVG lze měnit rozměry bez ztráty kvality a změny datového objemu. Protože se dá kód SVG obrázku vkládat přímo do kódu webové stránky, lze ho stylizovat pomocí CSS. Tímto způsobem s ním lze interaktivně pracovat. S SVG obrázkem můžeme pracovat i pomocí jazyka JavaScript – několik knihoven popsaných v části 1.4.1 používá k vykreslování grafů právě SVG formát.

Tento formát nelze používat na vše. Grafika, která nevzešla z grafického editoru, ale například z fotoaparátu, se pro ukládání v tomto formátu nehodí. Pro vlastní tvorbu SVG jsou potřeba grafické editory – například Inkscape, Illustrator nebo SVGOMG běžící přímo v prohlížeči. Výsledné obrázky z daných programů je dobré optimalizovat odstraněním zbytečných metadat, uložených grafickým editorem nebo pročištěním od nadbytečných značek.

1.3.3 Canvas

Canvas je HTML prvek zahrnutý ve specifikaci HTML5. Slouží pro kreslení na webové stránce pomocí jazyka JavaScript. Tento element sám o sobě žádné kreslící schopnosti nemá, je to pouze grafický kontejner na který pomocí skriptu kreslíme. Na rozdíl od SVG je canvas reprezentován jako bitmapa a při jeho používání můžeme pracovat s každým pixelem daného plátna.

Jistou nevýhodou při práci s tímto elementem je, že neexistuje žádná vnitřní struktura vykreslených objektů a proto se na dané objekty hůře aplikují události – najetí myší, kliknutí, přetahování apod.

S pomocí canvasu pracuje další JavaScript API WebGL. Toto API slouží pro zobrazování 3D grafiky a vykreslování je prováděno přímo na grafické

kartě. Díky tomuto API lze vytvářet složitější aplikace, vizualizace a 3D herní enginy.

1.4 Existující nástroje

Nástrojů pro vizualizaci grafů je nespočet. V následující části popisuji především knihovny v jazyku JavaScript, které jsem zvažoval pro použití v mé práci a poté desktopové aplikace ze kterých jsem čerpal inspiraci pro vytváření výsledné aplikace. U každé knihovny a aplikace vypichuji jejich klady a zápory.

1.4.1 Knihovny

V rámci této práce je většina analyzovaných knihoven v jazyku JavaScript. Je to z toho důvodu, že JavaScript je prakticky jediný webový skriptovací jazyk, který dokáže pracovat s pokročilou vizualizací dat.

1.4.1.1 Sigma.js

Sigma.js [8] je knihovna, která je určena pro kreslení grafů. Je velmi lehce použitelná a poskytuje mnoho zabudovaných funkcí. Jako vizualizační nástroje používá canvas a WebGL a podporuje i pohyb a klik myší pro lepší manipulaci s grafy.

Protože sigma.js využívá canvas a WebGL, je vykreslování rychlejší než u další velmi populární knihovny d3.js. Zároveň tato knihovna velmi dobře vizualizuje rozsáhlé grafy.

Pro tuto knihovnu jsou vytvořeny zásuvné moduly, které pomáhají jednotlivým uživatelům lépe pracovat se specifickými daty a upravit si vizualizaci podle potřeby.

Aktuálně bohužel žádné zásuvné moduly vytvářeny nejsou a dlouho žádný nový vytvořen nebyl. Sigma.js vykresluje grafy pomocí force-directed uspořádání a má dokonce implementován algoritmus Force atlas 2, který používá aplikace Gephi.

Nejdůležitější vlastnosti:

+ velmi rychlá vizualizace,
+ využívá canvas a WebGL,
+ algoritmus Force atlas 2,
+ možnost rozšíření pomocí zásuvných modulů,
- aktuálně již není tolik využívaná.

1.4.1.2 D3.js

Dalším nástrojem pro vytváření vizualizací dat přímo ve webovém prohlížeči je open-source knihovna d3.js [9] (Data-Driven Documents). Knihovna slouží pro manipulaci s objektově orientovanými dokumenty a dá se použít na velké množství druhů dat, které umožňuje stylizovat podle uživatelské potřeby.

Knihovna vizualizuje data pomocí HTML5, CSS a hlavně SVG. Hlavní předností d3.js je, že umožňuje velkou kontrolu nad výsledným výstupem. To je umožněno tím, že d3.js umožňuje libovolně navázat data na DOM a poté na tento objekt aplikovat datově řízenou transformaci.

Přímo na stránkách knihovny lze nalézt nezměrné množství praktických příkladů jejího použití i se zdrojovými kódy. D3.js je extrémně rychlá knihovna, která navíc podporuje velké množství dat. Data lze vizualizovat ve 2D/3D a knihovna zároveň poskytuje možnost dynamických změn pro iterování nebo animaci při práci s daty.

Nejdůležitější vlastnosti:

+ dokáže zpracovat mnoho druhů vstupních dat,
+ velké množství příkladů použití,
+ rychlá knihovna, která si poradí i s velkými daty,
+ dokáže vykreslovat ve 2D i 3D,
- využívá pouze SVG,
- složitější na implementaci.

1.4.1.3 Vis.js

Další knihovnou je vis.js [10]. Knihovna je taktéž napsaná v jazyce JavaScript a slouží pro vizualizaci skrz webový prohlížeč. Knihovna je vytvořena tak, aby šla jednoduše použít, dokázala se vypořádat s velkým množství dynamických dat a umožňovala s nimi manipulovat.

Knihovna zvládá mnoho druhů 2D i 3D vizualizací a umožňuje s nimi různorodě manipulovat. K vykreslování využívá canvas a umožňuje skvělou práci s uloženými daty. Na stránkách vis.js lze nalézt spoustu příkladů s použitím této knihovny. Velkou výhodou vis.js je také to, že je velmi dobře zdokumentovaná.

Nejdůležitější vlastnosti:

+ 2D a 3D vizualizace,
+ využívá canvas,

+ výborná dokumentace a příklady použití,
+ dynamická práce s daty.

1.4.1.4 Cytoscape a cytoscape.js

Cytoscape [11] je multiplatformní open-source program, který se zaměřuje na vědeckou a akademickou činnost, zejména na molekulární a biologické sítě a interakce v nich. I když byl původně vyvinut pro biologický výzkum, je velmi rozšířen a používá se pro komplexní grafovou (resp. síťovou) analýzu a vizualizaci.

V základu poskytuje několik vstupních a výstupních formátů a funkcí pro integraci, analýzu a vizualizaci dat. Funkce mohou být rozšířené pomocí obrovského množství zásuvných modulů, které jsou dostupné v Cytoscape App store. Zásuvné moduly rozšiřují možnosti práce s daty, poskytují další druhy uspořádání, případně druhy vstupních a výstupních formátů. Většina je jich dostupná zdarma. Ve formě zásuvných modulů lze stáhnout rozšíření pro vizualizaci 3D grafů.

Vývojáři aplikace dále pracují na knihovně napsané v čistém jazyku JavaScript – cytoscape.js, která je vytvořena pro práci s grafy ve webovém prohlížeči. Knihovna reprezentuje grafy jako JSON objekty, podporuje dotazování nad grafy a filtrování dat a dále také teorii grafů a možné operace s nimi. Má zabudováno mnoho algoritmů jako například BFS, DFS, Dijkstrův algoritmus, Kruskalův algoritmus a mnoho dalších. Knihovna také dokáže animovat grafy.

Nejdůležitější vlastnosti:

+ mnoho vstupních a výstupních formátů,
+ možnost rozšíření pomocí zásuvných modulů,
+ podporuje teorii grafů,
+ pro vykreslování využívá canvas,
- cytospace.js závisí na dalších knihovnách.

1.4.1.5 Alchemy.js

Alchemy.js [12] je něco mezi knihovnou a aplikací a je vytvořena firmou graphAlchemist. Je z velké části vystavěna nad výše zmíněnou knihovnou d3.js. Díky tomuto faktu, může být aplikace používající tuto knihovnu jednoduše rozšířena jakoukoliv funkcí, kterou poskytuje d3.js

Knihovna byla vytvořena proto, aby vývojáři ve svých aplikacích mohli s minimem znalostí rychle a jednoduše vizualizovat grafy. Při použití knihovny probíhá většina úprav ve vizualizování spíše přizpůsobením výchozího nastavení, než samotným programováním.

Používání alchemy.js je závislé dalších knihovnách – jQuery, d3, lodash a bootstrap. Jediným vstupním formátem, který alchemy.js podporuje je formát GraphJSON. Vykreslování grafu probíhá na HTML5 element SVG.

Knihovna je sama o sobě zajímavá, bohužel je špatně zdokumentovaná a samotná stránka vývojářů této aplikace již nefunguje.

Nejdůležitější vlastnosti:

+ postavena na d3.js – velká možnost rozšíření,
+ využívá SVG,
- špatně zdokumentovaná,
- jediný vstupní formát – GraphJSON.

1.4.2 Aplikace

V této části jsou vypsané nejdůležitější desktopové aplikace, které jsem vyzkoušel v rámci mé bakalářské práce. Všechny jsou dostupné zdarma.

1.4.2.1 Graph-tool

Graph-tool [13] je výkonný modul v jazyce Python používaný pro manipulaci s grafy a jejich analýzu. Jádro datové struktury a algoritmy modulu jsou implementovány v jazyce C++ a poté propojeny pomocí Boost Graph Library. Díky tomu je modul velmi výkonný – algoritmy jsou stejně rychlé, jako kdyby byla knihovna napsaná v čistém C++.

Graph-tool pracuje s třemi vstupními a výstupními formáty – GraphML (formát založený na XML), GML a DOT. Mnoho algoritmů je implementováno paralelně pomocí OpenMP, což zajišťuje větší výkon na vícejádrových architekturách. Podporuje ukládání libovolných informací o uzlech, hranách, grafech a dovoluje filtrovat a odstraňovat hrany/uzly za běhu.

Graph-tool disponuje vlastními vizualizačními algoritmy a může být použit jako balíček pro Graphviz. Také podporuje teorii grafů a používá se pro práci s rozsáhlými grafy – simulace buněčných tkání, vytěžování dat, analýza sociálních sítí a dalších.

Nejdůležitější vlastnosti:

+ rychlé výpočty a vykreslování,
+ možnost změny grafu za běhu,
+ podporuje teorii grafů,
+ vlastní vizualizační algoritmy,
- malé množství podporovaných formátů.

1.4.2.2 Graphviz

Graphviz [14] je open-source aplikace a jeden z nejstarších nástrojů pro zobrazování a vizualizaci diagramů/grafů. Bohužel akceptuje pouze jediný vstupní formát a to kód v jazyce DOT. Samotný program podporuje mnoho výstupních formátů a ovládá se hlavně z příkazové řádky. Nabízí mnoho funkcí vykreslování grafů, lze v něm použít i Dijkstrův algoritmus nebo ho můžeme využít jako knihovnu v jiných programech. Jakožto jedna z nejstarších aplikací je zaměřena hlavně na vykreslování 2D grafů a diagramů a pro vykreslení 3D grafů Graphviz používá soubory s formátem VRML.

Nejdůležitější vlastnosti:

+ 2D diagramy a grafy,
- ovládání z příkazové řádky,
- jediný vstupní formát – jazyk DOT.

1.4.2.3 Gephi

Gephi [15] je open-source program pro práci s grafy vytvořený v Javě na platformě NetBeans. Byl vytvořen studenty z technologické univerzity v Compiègne a následně podpořen několika vlivnými společnostmi, které kolem tohoto programu vytvořily konsorcium pro podporu jeho vývoje. Nástroj se stále aktivně vyvíjí a jeho poslední verze byla vydána 17. září 2017.

Jedná se o desktopovou multiplatformní aplikaci, která se zaměřuje na manipulaci a analýzu složitějších grafových struktur. Aplikace disponuje jednou z nejrychlejších grafových vizualizací a hledáním cest v rozsáhlých grafech a obsahuje mnoho funkcí pro dynamickou manipulaci s daty a filtrováním informací, které uživatel potřebuje.

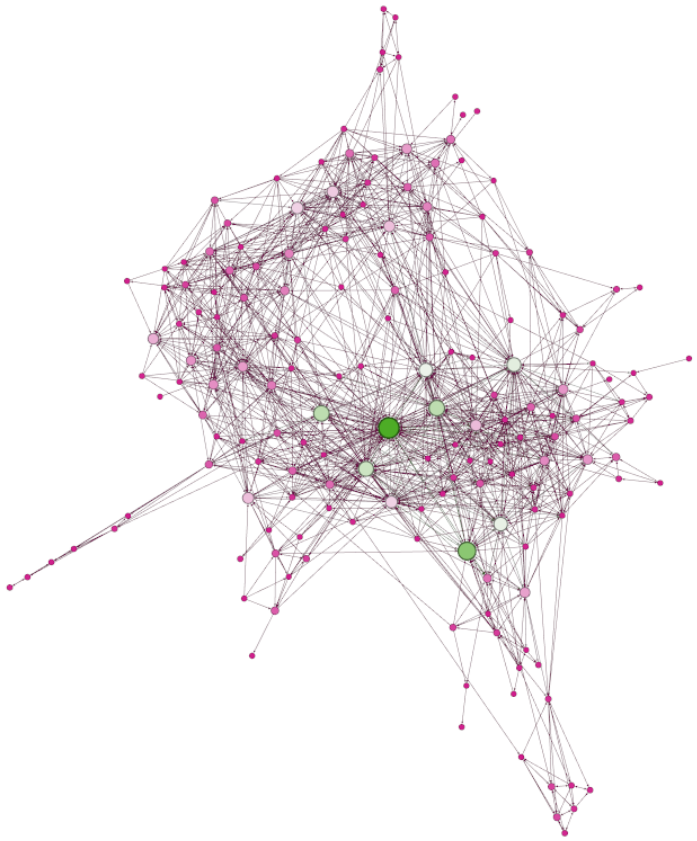
Grafy jsou vizualizovány pomocí algoritmů, které jsou založeny na metodě force-directed uspořádání (1.5). Toto vykreslování uživatel na začátku spustí a lze ho dle potřeby pozastavit a pracovat s tím, co program do té doby vykreslil.

Gephi pro definování vstupních a výstupních dat využívá zejména svůj vlastní formát GEXF, kromě klasických grafových formátů podporuje také vložení dat z databází a CSV souborů. Navíc je zde možnost rozšířit aplikaci pomocí množství zásuvných modulů.

Aplikace podporovala vizualizaci ve 3D, ale v nejnovější verzi se ji vývojáři rozhodli zrušit, dokud nevytvoří lepší a rychlejší implementaci.

Nejdůležitější vlastnosti:

+ multiplatformní aplikace,
+ jedna z nejrychlejších vizualizací,



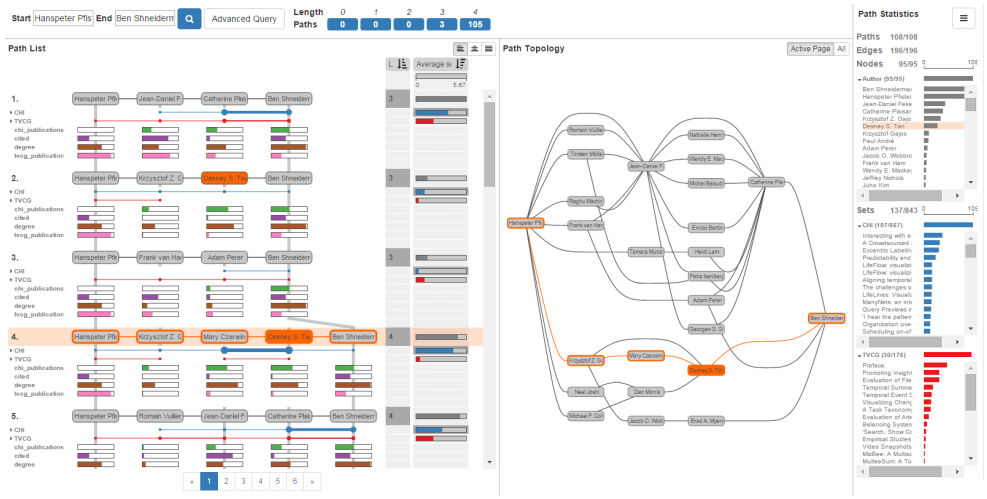
Obrázek 1.5: Force-directed uspořádání v programu Gephi převzaté z [15]

+ funkce pro dynamickou manipulaci a filtrování,
+ algoritmy založeny na force-directed uspořádání,
+ mnoho vstupních i výstupních formátů,
- složitější ovládání,
- aktuálně nepodporuje 3D.

1.4.2.4 Pathfinder

Pathfinder [16] je program zaměřený přímo na vyhledávání cest v rozsáhlých grafech s mnoha cestami. Aplikace je vystavěna nad webovým aplikačním rámcem Caleydo a je proto dostupná online. Pathfinder je velmi silný nástroj - v první fázi nalezne všechny cesty zadané délky mezi vybranými uzly. Ty poté uživateli promítne dvěma způsoby.

Jak můžeme vidět na obrázku 1.6, program na levé straně zobrazí cesty v seznamu a seřadí je podle jejich relevance pro uživatele (váha hran, délka



Obrázek 1.6: Program Pathfinder (převzato z [16]).

cesty apod.). Na pravé straně zobrazí diagram cest v samostatném grafu. Mezi cestami lze dále filtrovat a uživatel si tak může zobrazit informace, které ho opravdu zajímají. Diagram je vždy ve 2D a aplikace 3D vykreslení nepodporuje.

Pathfinder má mnoho dalších funkcí - porovnává cesty mezi sebou a zobrazuje jejich překrytí, dokáže spočítat kolikrát se jaké uzly vyskytují ve všech cestách a obsahuje mnoho funkcí pro filtrování vyhledaných cest.

Nejdůležitější vlastnosti:

+ dva způsoby zobrazení – seznam cest a graf,
+ velké množství funkcí,
+ porovnávání cest mezi sebou,
- pouze 2D grafy,
- nelze zobrazit podgraf v celém grafu.

1.5 Struktura webové stránky FIT ČVUT

Jak bylo zmíněno v úvodu, jedním z vizualizačních problémů může být zobrazení souvislostí jednotlivých stránek, na nějakém rozsáhlém webu. V rámci této práce se budeme zaobírat datovou strukturou webové stránky FIT ČVUT.

Jedná se o velmi rozsáhlou síť ve které se nachází velké množství uzlů. Uzly mezi sebou mají mnoho cest. Cesty vedou buď přímo mezi uzly, nebo jsou vedeny přes jiné části sítě (grafu).

Jednotlivé uzly mají několik vlastností. V budoucí aplikaci je omezíme pouze na vlastnosti, které jsou pro ni důležité. První důležitou vlastností je `URI`. Jedná se o jedinečný identifikátor napříč všemi stránkami. Další užitečnou informací je krátký a plný název. Názvy se budeme zaobírat později, při návrhu uložení dat.

Vazby jsou vždy oboustranné, tudíž by výsledný graf mohl být reprezentován jako neorientovaný. Nicméně je důležité rozlišovat váhy jednotlivých typů vztahů. Ty se u inverzních vazeb občas shodují, není to však pravidlem. Kompletní přehled všech druhů vazeb, vazeb k nim opačných a jejich hodnot, nalezneme v tabulce 1.1.

Důležitou vlastností při hledání cest mezi uzly v této datové struktuře jsou hodnoty jednotlivých vztahů a jejich přepočítávání na základě délky cesty. Pokud máme cestu mezi uzly `A` a `C` přes uzel `B`, tak vezmeme hodnotu hrany mezi uzly `A` a `B` a vynásobíme ji s hodnotou hrany mezi uzly `B` a `C`.

Pokud by obě hrany byly typu `hasChild` s váhou 0.95, pak by hodnota vztahu mezi uzly `A` a `C` byla

$$0.95 * 0.95 = 0.9025.$$

V následující sekci rozebereme a popíšeme jednotlivé vztahy, které mezi sebou mají stránky na prototypu webové stránky FIT:

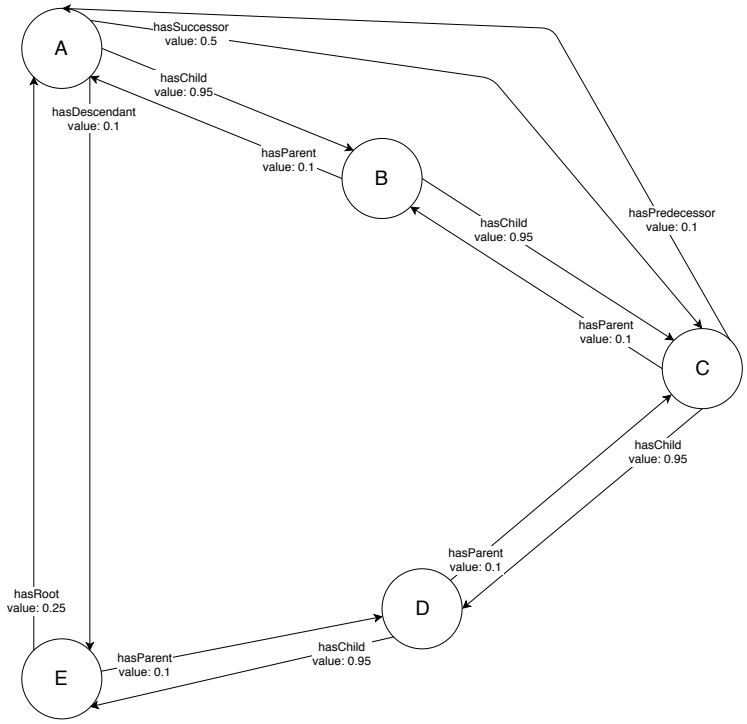
`creates/hasAuthor` Tento jednoduchý typ vazby značí vztah mezi dvěma uzly `A` a `B`, kdy `A` vytvořil `B` a naopak `B` má autora `A`.

`categorizes/hasCategory` Vztah mezi uzly, který značí, že uzel `A` kategorizuje `B` a `B` spadá do kategorie `A`. Tyto dvě vazby mají oproti ostatním typům velmi malou váhu.

`hasAlias/isAlias` Pravděpodobně nejjednodušší vazba, která pouze symbolizuje jiný název (zkrácený) pro daný uzel. `Katedra softwarového inženýrství` má alias `KSI` a `KSI` je aliasem `Katedra softwarového inženýrství`.

`hasChild/hasParent` Vztah dvou uzlů, který reprezentuje situaci, kdy je jeden uzel přímým potomkem (případně rodičem) uzlu druhého. Vztah `hasChild` je podobný vztahům `hasSuccesor` a `hasDescendant`, ale jeho hodnota je větší. Cesta přes 13 uzlů, které spojují vazby typu `hasChild`, by po vynásobení byla stále hodnotnější, než jedna vazba `hasSuccessor`. Totéž platí i pro vazbu typu `hasParent`. Vazby jsou znázorněné na obrázku 1.7.

`hasDescendant/hasRoot` Podobný typ vztahu mezi uzly, jako vazby zmiňované v minulém bodu. Tyto druhy vazeb mají malé ohodnocení a proto pro nás nemají příliš velký význam.



Obrázek 1.7: Druhy vztahů následník/předchůdce

`hasFlag/isFlagOf` Vztah, který reprezentuje příznak daného uzlu. Uzel `Petr Pulc` má příznak `Doktorand` a naopak `Doktorand` je příznakem pro uzel `Petr Pulc`. Uzel, který v těchto vztazích reprezentuje příznak je typicky nějaká vlastnost, kterou má více uzlů společných, ale není jich tolik, aby byla tato vlastnost zahrnuta přímo v samotných uzlech.

`hasPredecessor/hasSuccessor` Tato vazba představuje stav, kdy jsou spolu uzly v posloupnosti cesty o délce nejméně 2 (tudíž je mezi nimi alespoň jeden uzel). Nejedná se o přímého potomka, který je reprezentován vztahem `hasChild`. V posloupnosti tří uzlů `A`, `B`, `C` má `A` následníka `C` a `C` má předchůdce `A`.

`hasResponsiblePerson/isResponsibleFor` Tento vztah představuje situaci, kdy je osoba zodpovědná za určitou věc. Například `Michal Valenta` je zodpovědný za `Systém závěrečné práce` a `Systém závěrečné práce` má zodpovědnou osobu `Michal Valenta`.

`hasUriPart/isUriPart` Na začátku této sekce jsme zjistili, že každý uzel je reprezentován jedinečným identifikátorem – URI. Uzel `Bakalářský studijní program` má URI `bsp` a uzel `Přijímací řízení do BSP` má URI `prijimaci_rizeni_do_bsp`. Tyto dvě vazby pouze symbolizují, že URI jednoho uzlu je z části složená z URI druhého uzlu.

Tabulka 1.1: Tabulka vztahů prototypu webové stránky FIT

Typ vztahu	**Inverzní vztah**	**Hodnota vztahu**
categorizes	hasCategory	0.25
creates	hasAuthor	0.5
hasAlias	isAlias	1
hasAuthor	creates	0.5
hasCategory	categorizes	0
hasChild	hasParent	0.95
hasDescendant	hasRoot	0.1
hasFlag	isFlagOf	0.5
hasParent	hasChild	0.01
hasPredecessor	hasSuccessor	0.1
hasResponsiblePerson	isResponsibleFor	0.5
hasRoot	hasDescendant	0.25
hasSuccessor	hasPredecessor	0.5
hasUriPart	isUriPart	0
isAlias	hasAlias	1
isFlagOf	hasFlag	0.5
isIntersection	isPartOf	1
isPartOf	isIntersection	1
isRelatedFrom	isRelatedTo	0.5
isRelatedTo	isRelatedFrom	0.5
isResponsibleFor	hasResponsiblePerson	0.1
isUriPart	hasUriPart	0

`isIntersection/isPartOf` Vztah, který reprezentuje průnik částí různých uzlů. Tento vztah nemá pevnou hodnotu a odvíjí se od počtu částí na které je uzel rozdělen a z počtu částí, které má s dalším uzlem společné.

`isRelatedFrom/isRelatedTo` Tento typ vztahu se vyskytuje v případě, že spolu dva uzly nějakým způsobem souvisí. Může jít například o `KTI` (Katedru teoretické informatiky) a `STIGMA` (školní vědecká akce). `KTI` je souvislostí `STIGMA` a `STIGMA` souvisí s `KTI`.

1.5.1 Shrnutí

V rámci analyzovaného problému nás nejvíce zajímají váhy jednotlivých vztahů a způsob výpočtu váhy cest mezi uzly. Dále také informace, které si budeme o testovacích datech uchovávat. Popsané typy vztahů a uzlů důležité zejména pro koncového uživatele.

1.6 Požadavky na aplikaci

Nyní, když známe strukturu testovacích dat a seznámili jsme se s metodami, které se nejčastěji používají pro vizualizaci grafů, můžeme přistoupit k analýze a tvorbě požadavků na budoucí aplikaci. Tyto požadavky se skládají z dvou typů – funkční a nefunkční.

1.6.1 Funkční požadavky

Funkční požadavky popisují chování produktu nebo služby, který chceme realizovat.

Vyhledávání v grafu Aplikace bude na základě příkazů od uživatele generovat grafy a vizualizovat cesty ze zadaných dat.

Interaktivita S grafem a zobrazovací plochou bude možné interagovat. Kreslící plochu bude možné přiblížit a posouvat. Dalšími interaktivními prvky budou samotné uzly a hrany. Při najetí na ně budou zobrazeny informace o daných objektech a při kliknutí a podržení tlačítka je bude možné posouvat.

Zobrazení informací v grafu Graf jako takový bude zobrazovat různé informace. Při najetí na uzel se bude zobrazovat jeho název. Při najetí na hranu bude zobrazen její typ.

Přidání uzlu/hrany Na prázdné plátno nebo do již vykresleného grafu bude možné přidávat uzly a hrany.

Odstranění uzlu/hrany Hrany a uzly bude možné odstraňovat ze zobrazeného grafu.

Editace uzlu/hrany Editace uzlů bude spočívat v jejich popisu - id, názvu a URI. Při editaci hrany bude možné změnit uzly, mezi kterými vede a její ID, popis a váhu.

Import a export grafu Aplikace bude umožňovat importování a exportování dat, kterými bude graf reprezentován. Tyto operace budou probíhat pomocí souborů s příponou *.json*.

Uložení vykresleného grafu Vytvořená aplikace bude umožňovat uložení stavu vykreslovací plochy do souboru.

1.6.2 Nefunkční požadavky

Tento druh požadavků popisuje vlastnosti, které přímo nesouvisí s funkčností aplikace - jedná se o například o výkonnost, přístupnost nebo kvalitu softwaru.

Intuitivní uživatelské rozhraní Většina aplikací, které se zabývají vizualizací grafů, postrádají jednoduché a intuitivní ovládání, které by uživateli poskytovalo okamžitou možnost využívání nástroje. Tato aplikace bude jednoduchá a uživatelsky přívětivá.

Nezávislost na platformě Výsledná aplikace bude nezávislá na platformě, ze které bude spuštěna. Minimálně bude podporovat operační systém Windows a další operační systémy založené na linuxovém jádru.

Budoucí rozšířitelnost Aplikace bude vyvíjena tak, aby se i po jejím zhotovení v rámci této bakalářské práce dala dále vylepšovat.

1.7 Případy užití

Tato sekce popisuje chování aplikace při interakci s uživatelem. V zásadě můžeme rozdělit případy užití na dva druhy. První z nich jsou situace, při kterých dochází k interakci s grafem. Dalším druhem jsou operace, při kterých dochází k manipulaci se samotnými daty a soubory. Případy užití jsou znázorněny v diagramech 1.8 a 1.9. Pokrytí funkčních požadavků případy užití je znázorněno v tabulce 1.10.

1.7.1 Vizualizace grafu

Základním prvkem, který bude uživatel v aplikaci nejčastěji využívat je vizualizace samotného grafu. Tato operace je základním prvkem pro jakoukoliv interakci se samotným grafem.

Vizualizace bude probíhat na základě příkazů, které uživatel zadá do příkazového pole. Uživatel nejprve příkaz zadá a pomocí tlačítka „Potvrdit“ ho odešle aplikaci ke zpracování. Aplikace příkaz zpracuje a vizualizuje data dle požadavku na zobrazovací plochu.

Pokud při zpracování příkazu dojde k nějaké syntaktické chybě, uživatel bude na tuto chybu upozorněn, příkaz zůstane v poli vyplněn, aby ho mohl uživatel opravit a na plátně zůstanou zobrazena původní data.

1.7.2 Manipulace s grafem

Tento případ užití popisuje situaci, kdy bude uživatel manipulovat s grafem nebo jeho pozadím. Jedná se o základní operace posouvání a přibližování samotného grafu.

Pro manipulaci s plochou stačí, aby na ni uživatel najel kurzorem. Při posouvání kolečkem myši bude plátno přiblíženo nebo oddáleno. Další vlastností bude posouvání plochy. Uživatel klikne levým tlačítkem myši na plochu, podrží ho stisknuté a při pohybu myši se bude posouvat i plocha s grafem.

Uživatel bude moci manipulovat i se samotnými uzly a hranami. Při kliknutí a podržení tlačítka myši na uzlu se bude uzel posouvat. Zda změní trvale

svoji pozici nehledě na zbytku grafu, bude záležet na obecném nastavení grafu jako celku. Stejná akce provedená s hranou bude posouvat celou plochu, na které je graf zobrazen.

1.7.3 Přidání uzlu/hrany

Další případy manipulace s grafem se již týkají samotných uzlů a hran. První z nich je přidání hrany nebo uzlu.

Pro manipulaci s objekty na kreslícím plátně bude k dispozici lišta v horní části této plochy. Uživatel nejdříve klikne na tlačítko „Editace“, které rozbalí lištu pro manipulaci s objekty. V liště nyní budou zobrazena dvě tlačítka. Prvním z nich je „Přidat uzel“. Uživatel na toto tlačítko klikne a ocitne se v režimu přidávání nového uzlu. V liště se nyní objeví tlačítko „Zpět“, které uživatele vrátí do režimu editace a vpravo od tlačítka bude zobrazen text s krátkou informací, jak přidat nový uzel.

Uzel se přidá tak, že uživatel klikne na volné místo v kreslící ploše. Po kliknutí aplikace zobrazí vyskakovací okno, do kterého uživatel může zadat údaje o uzlu. Těmito údaji jsou `ID`, `Název` a `Popis`. `ID` musí být unikátní, protože je to jednoznačný identifikátor mezi uzly. Údaj `Název` reprezentuje jméno, které se zobrazuje v grafu na uzlu. Poslední údaj je `Popis`, který se zobrazí potom, co uživatel s kurzorem najede na daný uzel.

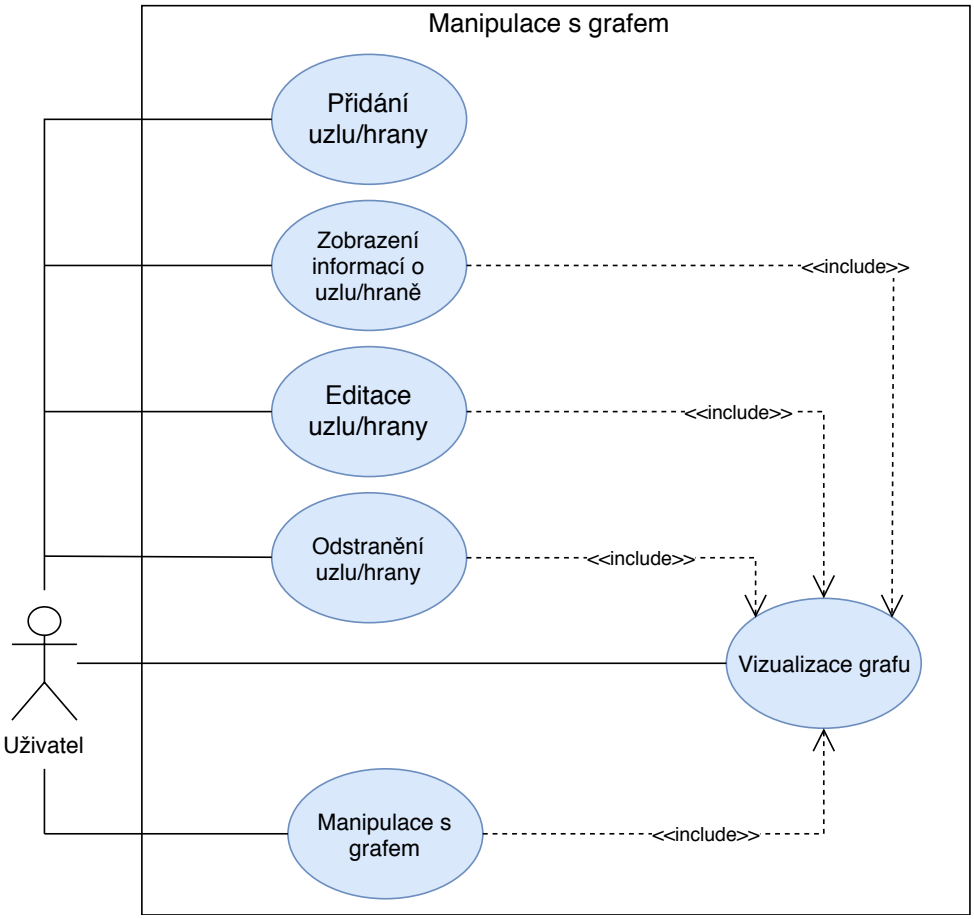
Druhým tlačítkem je „Přidat hranu“. Když uživatel klikne na tuto možnost, opět mu bude zobrazeno tlačítko „Zpět“ a krátký popis, jak přidat hranu. Přidání hrany probíhá tak, že uživatel klikne na uzel ze kterého má hrana vycházet a za stálého držení tlačítka myši přetáhne kurzor na cílový uzel, čímž je spojí novou hranou. Po přetáhnutí se uživateli zobrazí vyskakovací okno, do kterého musí zadat `ID`, které musí být unikátní, `typ hrany` a `hodnotu hrany`, která musí být číselná a nesmí být menší než 0.

V případě, že uživatel zadá již existující ID nebo špatně vyplní pole s hodnotou hrany, aplikace ho na nesprávně zadaný údaj upozorní.

1.7.4 Odstranění uzlu/hrany

Objekty můžeme z kreslícího plátna také odstraňovat. Hodit se to může v případě, když máme zobrazený moc velký graf a chceme ho zpřehlednit a odstranit uzly nebo hrany, které nás nezajímají.

Uživatel musí nejprve kliknout na tlačítko „Editace“, aby mohl s objekty začít manipulovat. Nyní stačí pouze kliknout na uzel nebo hranu, čímž je označíme a v modifikační liště se zobrazí nové tlačítko „Smaž označené“. Pokud uživatel odstraní hranu, z výsledného grafu se smaže pouze tato hrana. Naopak pokud uživatel odstraní uzel, kromě samotného uzlu se smažou i všechny hrany s tímto uzlem spojené.



Obrázek 1.8: Případy užití spojené s manipulací s grafem

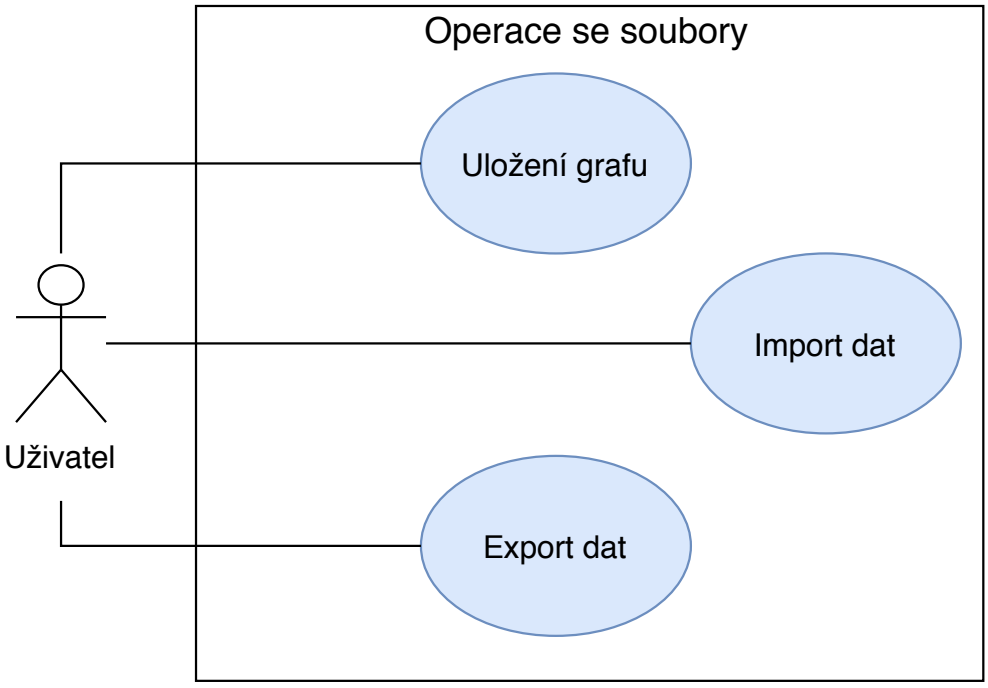
1.7.5 Editace uzlu/hrany

Při používání aplikace budeme mít většinou připravená data ze souboru nebo databáze. Nicméně pokud přidáme nějaký uzel nebo hranu přímo na kreslící plochu, budeme muset informace týkající se těchto uzlů nebo hran přidat ručně. K tomu se nám hodí možnost editace objektů.

Pokud chce uživatel editovat objekt, první co musí udělat je kliknout na tlačítko „Editace“. Poté si vybere uzel nebo hranu, kterou chce editovat, klikne na ni, čímž ji označí a v modifikační liště se objeví nové tlačítko.

Pokud bude označený objekt uzel, objeví se tlačítko „Uprav uzel“. Po kliknutí na tlačítko aplikace zobrazí nové modifikační okno, ve kterém lze upravit údaje o uzlu.

Pokud bude označeným objektem hrana, objeví se tlačítko „Uprav hranu“. Při kliknutí na tlačítko se u hrany zvýrazní oba koncové body, které jsou napojeny na nějaké uzly. Uživateli bude v tu chvíli umožněno přetáhnout body na libovolný nový koncový uzel. Po umístění nového konce dostane uživatel



Obrázek 1.9: Případy užití při práci se soubory

možnost upravit údaje o hraně. Aplikace opět reaguje na nesprávně zadané údaje, jimiž jsou ID a hodnota hrany.

1.7.6 Zobrazení informací o uzlu/hraně

Pouze zobrazený graf nám sám o sobě mnoho neřekne. Důležité jsou i informace o samotných uzlech a hranách.

Uzly budou v základním nastavení přímo u každého uzlu zobrazovat zkrácené jméno. Po najetí kurzorem na daný uzel se uživateli zobrazí celý název. Hrany oproti uzlům nebudou na první pohled zobrazovat nic, ovšem poté co na ně uživatel najede kurzorem, zobrazí informace o typu hrany.

1.7.7 Uložení grafu

První operací, která pracuje s externím souborem je vyexportování plochy s grafem do souboru. Jedná se o uložení souboru ve formátu *.png*. Tento formát je zvolen z důvodu, které jsou vysvětleny v sekci 2.2.

Uživatel pravým kliknutím myši na kreslící plátno zobrazí kontextové menu, ve kterém bude mít možnost „Uložit jako“. Po kliknutí na tuto položku se uživateli zobrazí prohlížeč adresářů, ve kterém má možnost zvolit jméno souboru a vybrat umístění, kam bude soubor uložen.

1.7.8 Import dat

Importování dat do aplikace nám může usnadnit spoustu práce. Pokud máme soubor, který chceme importovat ve správném formátu, stačí nám ho pouze importovat a ihned se zobrazí na kreslící ploše. Soubory s grafy, které budeme do aplikace importovat, budou ve formátu *.json*.

Uživatel v levém panelu stiskne tlačítko „Import“ a rozbalí se mu prohlížeč adresářů. Po výběru souboru se graf vykreslí na kreslící ploše. V případě chyby v importovaném souboru se žádný graf nezobrazí a uživatel bude upozorněn na chybu.

1.7.9 Export dat

Poslední operací s daty je exportování grafu. To se může hodit ve chvíli, kdy si chceme uložit vytvořený graf pro další použití nebo kdybychom ho chtěli použít v jiné aplikaci. Export dat bude probíhat ve formátu *.json*.

Uživatel pouze stiskne tlačítko „Export“ v levém panelu a aplikace automaticky exportuje daný stav grafu do souboru ve formátu *.json*. Při jeho opětovném importu se graf zobrazí ve stejném rozložení, v jakém byl exportován.

Funkční požadavky vs. případy užití	Vizualizace grafu	Manipulace s grafem	Přidání uzlu/hrany	Odstranění uzlu/hrany	Editace uzlu/hrany	Zobrazení informací o uzlu/hraně	Uložení grafu	Export dat	Import Dat
Vyhledávání v grafu	X								
Interaktivita		X				X			
Zobrazení informací v grafu		X				X			
Přidání uzlu/hrany	X		X						
Odstranění uzlu/hrany	X			X					
Editace uzlu/hrany	X				X				
Import grafu	X								X
Export grafu								X	
Uložení vykresleného grafu							X		

Obrázek 1.10: Pokrytí funkčích požadavků případy užití

Kapitola 2

Návrh

Tato kapitola je zaměřena na návrh aplikace dle analýzy vykonané v předchozí kapitole. Porovnáme analyzované knihovny a možnosti vykreslování a vybereme vhodný způsob, jak vytvořit aplikaci pro zadaná data. Poté vytvoříme samotný návrh aplikace od uložení dat, přes výběr platformy a rozvržení GUI.

2.1 Platforma

Již od počátku této bakalářské práce směřujeme k tomu, že jejím výstupem bude webová aplikace. Pro tuto volbu jsem se rozhodl z několika důvodů.

Při analýze existujících aplikací a knihoven, které se zabývají vizualizací grafů a zkoumáním vztahů v rozsáhlých sítích jsme zjistili, že drtivá většina, ne-li všechny, vyžadují instalaci. Důvod k tomu je prostý. Při analýze a vizualizaci rozsáhlých sítí musí počítač provádět tak velké výpočty, že jsou v nich webové aplikace jednoduše pomalejší.

V rámci této bakalářské práce by vytvoření další desktopové aplikace nepřinášelo mnoho nového. Aplikace analyzované v 1.4.2 disponují velkým počtem funkcionalit a nastavení a uživatel si tak může vybírat, kterou z aplikací na základě jeho potřeb použije. Žádná z těchto aplikací bohužel nedisponuje funkční online verzí.

Výhodou webové aplikace je její přístupnost. Stačí na jakémkoliv zařízení spustit webový prohlížeč s adresou dané aplikace a tu je možné hned využívat. Aplikace je zároveň stále dostupná v aktuální verzi. Menší nevýhodou může být fakt, že zařízení na kterém budeme chtít aplikaci používat, musí být připojeno k internetu.

Aplikace bude zaměřena na vizualizaci rozsáhlejších grafů, ale bude obsahovat funkce, které umožní její použití pro vizualizaci a zobrazování grafů menších.

2.2 Porovnání a výběr technologií

Cílem této bakalářské práce je vytvořit webovou aplikaci. Nyní si musíme rozmyslet, jaké technologie použijeme pro vytvoření budoucí aplikace. Jak bylo řečeno v analýze, prakticky jediným způsobem, kterým lze vytvořit webovou aplikaci, která bude schopna reálně vizualizovat data, je použití skriptovacího jazyka JavaScript.

2.2.1 JavaScript

JavaScript [17] je multiplatformní, dynamický, objektově orientovaný skriptovací jazyk. V dnešní době se používá zpravidla pro WWW stránky, na kterých ovládá různé interaktivní prvky GUI, jsou v něm vytvářeny animace apod.

Protože byl JavaScript v roce 1995 vyvinut za pouhé 3 týdny, obsahuje mnoho chyb a nedokonalostí, které jsou eliminovány a řešeny pomocí řady nástaveb. Pro JavaScript je vytvořeno také mnoho aplikačních rámců, které usnadňují vývoj určitých typů aplikací.

Důležitým faktem je, že JavaScript běží na straně klienta a všechny aplikace jsou spouštěny v prohlížeči u uživatele. Můžeme tedy dynamicky měnit obsah webové stránky přímo u uživatele. Oproti serverovým jazykům (např. PHP) je to obrovský rozdíl.

Menší nevýhodou je to, že uživatel si může ve svém prohlížeči JavaScript vypnout, což může vést k tomu, že všechny dynamické prvky přestanou fungovat. Tento problém řeší různé knihovny a aplikační rámce, které poskytují předpřipravené miniaplikace, ve kterých vývojař pouze nastaví co v nich má být a knihovna se postará o správné zobrazení i v případě, že bude mít uživatel v prohlížeči vypnutý JavaScript.

Společně s HTML5 a elementem canvas, lze pomocí jazyku JavaScript vytvářet různé hry nebo vizualizace. V dnešní době je v jazyku JavaScript vytvořeno mnoho aplikací tak, jak je známe z jejich desktopových verzí – například MS Office.

2.2.2 Vizualizační knihovna

Při použití jazyku JavaScript můžeme využít některé z vizualizačních knihoven, které jsme analyzovali v části 1.4.1. Použití knihovny nám velmi usnadní implementaci vizualizace, která by jinak byla zdlouhavá a pro tuto práci nadbytečná.

Protože se v budoucí aplikaci chceme zaměřit na rozsáhlejší grafy, které se budou na kreslící ploše dynamicky měnit, pro vizualizování dat si zvolíme element canvas. Tím se nám redukují možné knihovny, které budeme moci použít. První z nich je alchemy.js. Tu jsem zavrhl z několika důvodů. K vizualizaci využívá SVG, je velmi špatně zdokumentovaná, podporuje pouze jeden

vstupní formát a její použití sestává zejména z konfigurace již naprogramované aplikace, kterou pouze vložíme do té naší.

Další knihovnou, kterou na základě výběru elementu pro vizualizování vyškrtneme, je d3.js. Její výhodou je, že je velmi rozšířená, dokáže pracovat i s rozsáhlými grafy a na stránkách knihovny můžeme nalézt mnoho příkladu různých použití. Nevýhodou je kromě využívání SVG také fakt, že oproti ostatním knihovnám je d3.js složitější zakomponovat do projektu.

V rámci analyzovaných knihoven nám zbývají další tři, které využívají pro vizualizaci element canvas. První z nich je sigma.js. Mezi její silné stránky patří vizualizace pomocí WebGL. Sigma.js v základní verzi neobsahuje mnoho funkcí, ty jsou realizovány formou zásuvných modulů. V poslední době knihovna není příliš vyvíjena a její uživatelská základna je slabá.

Vis.js je knihovna, mezi jejíž silné stránky patří zejména jednoduché použití, mnoho užitečných zabudovaných funkcí pro práci s daty a výborná dokumentace. Na stránkách knihovny je mnoho příkladů s použitím a zároveň vis.js patří mezi jednu z nejrozšířenějších knihoven pro vizualizaci dat.

Poslední knihovnou je cytospace.js. Její největší výhodou je podpora teorie grafů, dotazování nad grafy a filtrování dat a možnost animování dat. Cytospace.js má mnoho funkcí, s čímž souvisí i fakt, že pro její funkci je potřeba dalších JavaScriptových knihoven. Knihovna také postrádá základní příklady použití.

2.2.2.1 Výběr knihovny

Na základě porovnání knihoven mezi sebou a vybráním způsobu, kterým budeme vizualizovat data, je potřeba rozhodnout, kterou z knihoven použijeme v budoucí aplikaci.

Konečné rozhodování zůstalo mezi použitím vis.js a cytospace.js. Sigma.js nebude použita, protože pro její lepší funkce je potřeba použít zásuvné moduly a není tolik využívána. To by mohlo komplikovat budoucí vývoj.

Knihovnou, kterou použijeme pro implementaci naší webové aplikace bude vis.js. Jediné co budeme potřebovat je právě tato knihovna, oproti cytospace.js lépe pracuje s daty a obsahuje spoustu funkcí, které nám ulehčí realizaci funkčních požadavků. Nespornou výhodou je také to, že na webových stránkách vis.js můžeme nalézt spoustu příkladů s použitím v různých situacích.

Vis.js využívá pro vizualizaci dat několik algoritmů, z nichž je jeden založen na metodě prostorového vyplňování (1.2.3) a ostatní na Force-directed metodě (1.2.1.1). Samotné vizualizování je možné nastavit a upravit podle potřeby vývojáře nebo uživatele.

2.2.3 Vizualizační metoda

Při vybírání vizualizační knihovny jsme přihlíželi i na způsoby uspořádání dat, které využívá. Většina knihoven využívá pro uspořádávání uzlů Force-directed

metodu, která si umí velmi dobře poradit s jakoukoliv velikostí grafu.

Pokud bychom chtěli použít jinou metodu, museli bychom ji sami implementovat, což by oproti využití metod, které používá knihovna vis.js mělo pouze malé výhody. Navíc jediná opravdu použitelná metoda pro malá i velká data je právě Force-directed metoda, případně stromové uspořádání.

3D vizualizace byla zavržena. Její výhoda tkví v lepší představě některých sítí a hierarchií a při použití s rozsáhlejšími daty se stává nepřehledná.

2.2.4 Způsob uložení dat

Důležitou částí návrhu aplikace je výběr uložení dat. Na výběr je několik možností od relačních nebo grafových databází, po proměnné a různé datové struktury.

Testovací data obsahují enormně větší množství hran než uzlů, proto bylo první zvažovanou možností uložení dat do grafové databáze. Ta je zaměřena na data, ve kterých vztahy mezi uzly převyšují počet uzlů. Vztahům mezi uzly lze přiřazovat různé vlastnosti. Data se pak získávají pomocí dotazů nad celou na databází.

Nejznámější grafová databáze Neo4j [18] poskytuje vlastní knihovnu Neovis.js, která je vytvořena spojením námi zvolené vizualizační knihovny a grafové databáze. Knihovna je velmi rozsáhlá, ale postrádá většinu funkcí, které poskytuje původní knihovna vis.js. Tím se ukázala jako nepoužitelná pro naši aplikaci. Pokud bychom chtěli použít grafovou databázi, museli bychom všechny operace a dotazování nad databází implementovat sami.

Z tohoto důvodu jsem se rozhodl maximálně využít knihovnu vis.js a pro uchovávání dat použít datové kontejnery (DataSety [19]) dané knihovny. DataSety jsou objekty, které v sobě uchovávají data. Je nad nimi implementováno mnoho užitečných metod od přidávání, mazání a dotazování nad daty, po pokročilé funkce filtrování. S DataSety pracuje vizualizační část knihovny, která si z nich bere informace pro samotnou vizualizaci. Protože si můžeme data definovat libovolně, kromě informací pro vizualizaci v ní budeme uchovávat i dodatečné informace, použité v implementační části.

```
1  var nodeData = new vis.DataSet();
2  var edgeData = new vis.DataSet();
3
4  nodeData.add({id: "katedra", label: "Katedra", title: "Katedra"});
5  nodeData.add({id: "ksi", label: "KSI", title: "Katedra SI"});
6
7  edgeData.add({id: 0, from: "katedra", to: "ksi", arrows: "to",
8                title: "hasChild", relValue: 0.95});
9
10 var uzel = nodeData.get("katedra"));
11 nodeData.update({id: "katedra", title: "Katedra FIT" });
12
13 edgeData.remove(0);
```

Listing 2.1: Vytvoření nového DataSetu a uložení dat.

V aplikaci budeme udržovat vždy dva DataSety – jeden pro uzly a jeden pro hrany. U uzlů si budeme ukládat ID, krátký název a popis. U hran to bude ID, zdrojový uzel, cílový uzel, směr hrany, popis a její hodnotu. Ukázka práce s datovými kontejnery je v 2.1.

Nejdřív na řádcích 1 a 2 inicializujeme prázdné objekty. Na řádcích 4,5 a 7 do nich přidáváme data. Každý záznam musí mít unikátní ID. Na řádku 10 pomocí metody `.get(ID)` uložíme záznam s daným ID do proměnné. Pokud záznam neexistuje, návratovou hodnotou je `null`. Zbylé dvě operace slouží pro aktualizaci dat podle zadaného ID a odstranění záznamu z objektu.

2.3 Uživatelské rozhraní

Při tvorbě webových stránek, potažmo aplikací, je důležité správně rozvrhnout prvky a elementy na stránce, aby první dojem neodradil uživatele, od jejího budoucího používání.

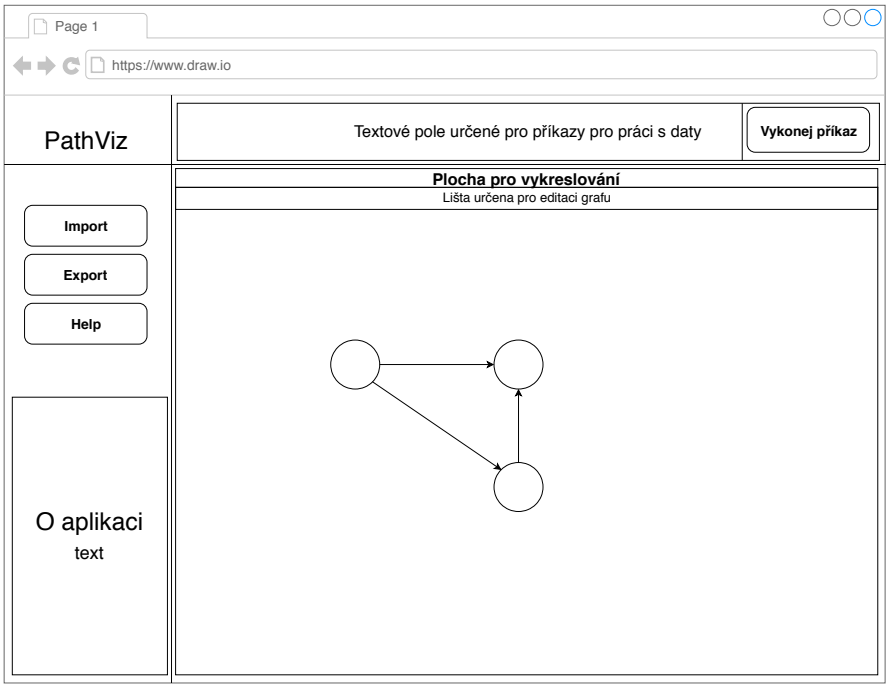
V analýze jsme si jako jeden z nefunkčních požadavků vytyčili intuitivní uživatelské rozhraní. Ve všech aplikacích se kterými jsme se v rámci analýzy k této práce setkali, bylo složité se ze začátku zorientovat a začít využívat jejich funkce. Aplikace se bude skládat z jedné jediné webové stránky, na které budou umístěny všechny informace a funkce. Návrh aplikace lze vidět na obrázku 2.1.

Z jedné stránky se bude aplikace skládat z toho důvodu, aby měl uživatel neustále přehled o celém rozhraní a dění v aplikaci. Celá obrazovka bude rozdělena do dvou částí. Na levých 26% obrazovky bude umístěn informační panel se třemi tlačítky. Na zbytku obrazovky bude umístěno textové pole pro zadávání příkazů, tlačítko pro spouštění příkazů zadaných do textového pole a samotné kreslící plátno. Plátno bude v horní části obsahovat lištu pro modifikaci uzlů a hran.

V panelu na levé straně budou umístěny tři tlačítka – „Import“, „Export“ a „Manuál“. Funkce tlačítek pro importování a exportování grafu jsou popsány v sekci 1.7. Jediné tlačítko, které nebylo popsané je „Manuál“. Po kliknutí na něj se uživateli zobrazí manuál k vytvořené aplikaci ve formátu *.pdf*. Pod tlačítky bude v krátkosti popsán smysl aplikace a informace o ní. V dolní části se bude nacházet patička s odkazem na vypracovanou práci.

Hlavní část obrazovky bude patřit samotnému účelu aplikace a to vizualizaci grafů. Prvním prvkem v hlavní části obrazovky bude textové pole. Přes něj bude uživatel zadávat příkazy pro specifikaci a vizualizaci dat. Příkazy se budou spouštět tlačítkem „Potvrdit“, vpravo od textového pole.

Nejdůležitější částí aplikace bude kreslící plocha. Plocha zabírá celý zbytek stránky a bude využívána k samotné vizualizaci grafů. Plocha bude realizována jako HTML element canvas (1.3.3). Nedílnou součástí plochy bude modifikační lišta, která dovolí uživateli přidávat, editovat a odstraňovat uzly a hrany.



Obrázek 2.1: Drátový model webové aplikace

Dalším prvkem, který je součástí aplikace jsou vyskakovací okna. První z nich je okno, které se zobrazí při vytváření nebo editaci objektů. Okno bude vyskakovat přibližně uprostřed kreslící plochy.

Další okna upozorňují uživatele na chyby. První z nich se objevuje při chybně zadaném údaji do formuláře pro úpravu nebo vytváření objektů a tento formulář překrývá. Druhé je realizováno jako lišta, která upozorňuje na chyby, zadané do příkazové řádky. Lišta je umístěna přímo pod příkazovou řádkou.

2.4 Shrnutí

Výsledek implementace bude webová aplikace s jednou stránkou, na které budou probíhat všechny akce. Aplikace bude vytvořená pomocí HTML5 a skriptovací akce na pozadí budou implementovány v jazyku JavaScript.

Veškeré vizualizace budou probíhat skrz vizualizační knihovnu vis.js, která je také implementovaná v jazyku JavaScript. Navíc využijeme datové objekty vytvořené pro danou knihovnu a budeme v nich ukládat data.

Kapitola 3

Implementace a testování

Tato kapitola je zaměřena na popis implementace a její samotné testování. V první části si projdeme části realizace, které souvisí s vytyčenými funkčními požadavky na vytvořenou aplikaci. Druhou částí této kapitoly je testování na základě případů užití a testování uživatelského rozhraní. Nakonec zhodnotíme implementaci, cíle které jsme si vytyčili a prodiskutujeme rozhodnutí, která vedla k realizaci.

3.1 Použité nástroje

V této sekci jsou vypsány nástroje, které byly použity při tvorbě této bakalářské práce.

Visual Studio Code Všechny zdrojové kódy aplikace byly napsány v aplikaci Visual Studio Code. Jedná se o open-source program od firmy Microsoft. Editor podporuje několik programovacích jazyků, chytře našeptává a zvýrazňuje, podporuje ladění a kontrolu nad správou verzí pomocí systému GIT. Při tvorbě práce byla použita aplikace ve verzi 1.23.0.

GIT Pro zálohování zdrojových kódů, diagramů a dat bakalářské práce byl použit systém pro správu verzí GIT ve verzi 2.7.4.

draw.io Všechny diagramy a grafy, které nebyly převzaté z externích zdrojů, byly vytvořeny v aplikaci draw.io. Aplikace slouží pro vytváření diagramů. Může být použita online, jako desktopová aplikace nebo v mobilním zařízení. Pro tuto práci byla použita verze 8.5.15. Aplikace je dostupná online na adrese `https://www.draw.io/`.

TeXstudio Pro sazbu této bakalářské práce bylo použito prostředí TeXstudio ve verzi 2.10.8.

3.1.1 Adresářová struktura

Zdrojové kódy aplikace jsou pro snadnou orientaci rozdělené do samostatných souborů podle jejich funkcí. Jsou rozděleny na několik částí:

- main.js, která obsahuje hlavní část aplikace, upravené funkce vizualizační knihovny a validace vstupů,
- importExport.js, obsahující funkce pro importování a exportování grafů,
- alertMessages.js, která obsahuje všechny chybové hlášky,
- commands.js, která obsahuje realizace samotných příkazů,
- basic.js, která obsahuje základní nastavení a inicializaci tlačítek a dat,
- vis.js a vis-network.min.js, obsahující zdrojové kódy vizualizační knihovny.

3.2 Realizace funkčních požadavků

V této sekci popíšu způsob, jakým byly realizovány funkční části vytvořené aplikace. Výraznou pomocí bylo použití vizualizační knihovny vis.js, kterou jsme v předchozích kapitolách popsali a odůvodnili její výběr.

3.2.1 Zobrazování informací

První věcí, kterou nám usnadňuje vizualizační knihovna, je zobrazování informací o hranách a uzlech. Jak jsme si popsali v 2.2.4, pomocí vis.js můžeme každému uzlu a hraně přiřadit různé vlastnosti. Jediné co musíme pro zobrazování informací udělat, je vyplnit každému objektu dané atributy. V grafu se pak atribut `label` zobrazuje přímo na daném objektu. Atribut `title` se zobzrazuje v malém okně, po najetí kurzorem na objekt.

3.2.2 Manipulace s objekty a grafem

Další operace, které nám velmi ulehčuje samotné použití knihovny vis.js je manipulace s objekty na plátně. Bez použití knihovny bychom museli vytvářet metody pro vybrání objektů, jejich posouvání na nové pozice, přibližování plátna a podobně. Všechny operace jsou zajištěny pouhým použitím knihovny a vykreslením prázdného plátna.

3.2.3 Modifikace objektů

Modifikace objektů na plátně zahrnuje přidávání, upravování a mazání uzlů a hran. Základní funkčnost uvedených akcí opět poskytuje vizualizační knihovna. Každou z nich bylo potřeba upravit, kvůli potřebám uložených dat a exportování grafu. Všechny operace probíhají pomocí editační lišty na obrázku 3.1.

Obrázek 3.1: Editační lišta pro manipulaci s objekty

3.2.3.1 Přidávání uzlů a hran

Přidávání bylo pro implementaci nejjednodušším příkazem. V základní verzi přidání uzlu nebo hrany pouze vytvořilo daný objekt na plátně. Kvůli uložení dat, bylo potřeba upravit dané funkce pro přidávání uzlů a hran.

V obou případech bylo nutné implementovat vyskakovací okno, které uživateli umožní zadat další informace o objektech. Mimo to jsou implementovány ošetření pro zadání neunikátního ID. Názvy upravených funkcí jsou `addNode()`, `addEdge()`.

V nich bylo potřeba nastavit HTML elementy pro zadání nových dat a následně provést volání funkcí pro jejich uložení. V daných funkcích nejprve zkontrolujeme, zda je uživatelem zadané ID unikátní, poté uložíme nový objekt do datových kontejnerů a nakonec zobrazíme vytvořené objekty na plátně. Stejné funkce používají i metody pro editaci objektů.

3.2.3.2 Úprava uzlů a hran

Metody, které prošly nejvíce úpravami, byly metody pro editování objektů. Původně fungovaly tak, že se při úpravě ID neupravil daný objekt, ale vytvořil se nový a ten původní zůstal na svém místě. V implementaci jsem tento fakt řešil u uzlů a hran rozdílně.

Úprava uzlu volá stejnou metodu, jako přidání uzlu nového. Metoda se jmenuje `saveDataNode()` a můžou v ní nastat dvě situace. Prvním případem je změna údajů o uzlu, ale ne změna jeho ID. V takovém případě neprobíhá žádná validace jestli je ID unikátní a pouze uložíme nová data.

Pokud změníme i ID uzlu, metoda nejprve zkontroluje, jestli je nové ID unikátní. Dále aplikace kontroluje, zda je k uzlu se starým ID připojena nějaká hrana. Pokud ano, aplikace nedovolí uživateli upravit uzel. Důvodem je to, že v databázi hran používáme ID uzlu jako identifikátor pro určení zda je hrana s uzlem spojená. Při změně ID by pak bylo nutné projít všechny hrany, které s uzlem souvisí a změnit jim tento atribut. Pokud úprava projde validacemi, je uzel v grafu upraven a proběhne aktualizace dat.

Také úprava hrany volá stejnou metodu, jako přidání hrany nové a tou je metoda `saveDataEdge()`. Je trochu odlišná od předchozí metody pro uzly. Protože se hrana týká pouze těch uzlů, které spojuje, validace u ní probíhá pouze ve formě kontroly unikátnosti ID. Pokud hranu pouze přepojujeme mezi jiné uzly nebo měníme její popis, tak se pouze uloží aktualizovaná data a do grafu se zanese změna.

Pokud hraně změníme ID, vytvoří se hrana nová. Původní zůstane na svém místě a právě vytvořená hrana se zapíše do databáze a přidá do grafu.

3.2.3.3 Odstranění uzlů a hran

Funkce pro odstranění objektů z plátna zůstaly skoro nedotčené. V implementaci bylo potřeba zajistit pouze aktualizaci dat v datových kontejnerech.

3.2.4 Práce se soubory

Operace se soubory, které jsme si vytyčili ve funkčních požadavcích jsou tři. Každý z nich přinášel jiné problémy při implementaci. Formát importovaných i exportovaných souborů má jednoznačně určenou strukturu, kvůli zachování kompletnosti testovacích dat. Pokud ovšem nebudou vyplněny všechny údaje, aplikace si s tím poradí. Pokud navíc exportujeme graf a následně ho importujeme zpět, zobrazí se ve stejném rozložení, jako před exportem.

3.2.4.1 Uložení obrázku

Uložení obrázku se v implementaci aplikaci ukázalo jako nejvíc problémové. Vis.js vizualizuje data tak, že mu vývojář poskytne prázdnou vrstvu (značka div) a knihovna ji následně využije jako element canvas, na který vizualizuje data. Pokud bychom v HTML kódu měly tento element přímo, jednoduše by se na něj navázala ukládací operace přes tlačítko v aplikaci.

Kvůli potřebám knihovny, zůstalo uložení obrázku realizováno pouze jako funkce prohlížeče a HTML5. Uživatel klikne pravým tlačítkem myši na plátno a v rozbaleném kontextovém menu vybere možnost „Uložit jako..“.

3.2.4.2 Import grafu

Aplikace akceptuje import grafů, ve formátu *.json*. Při importu je důležité mít soubor ve správném formátu, který jsme popsali v sekci 2.2.4 a ukázce kódu 1.2 v analytické části. Samotnou operaci importování souboru lze rozdělit do několika částí. Nejprve je potřeba nastavit HTML prvky, přes které otevřeme dialogové okno s výběrem souborů. Po výběru souboru, načteme jeho obsah do proměnné a odešleme ho dalším funkcím ke zpracování.

První věc, kterou funkce vykonají, je syntaktická analýza načteného obsahu a její uložení do proměnné. Z proměnné jsou pak postupně vyextrahovány jednotlivé uzly a hrany. Následně je zavolána funkce pro vizualizování grafu s novými daty. Celá funkce `importNetwork()` je k nalezení ve zdrojových kódech aplikace.

Zajímavostí je, že vis.js má zabudované funkce pro konverzi dat, které jsou exportované z aplikace Gephi a dat v jazyku DOT, který používá aplikace Graphviz. V případném rozšíření aplikace by se daly tyto metody použít pro podporu více formátů.

3.2.4.3 Export grafu

Export grafu probíhá také ve formátu *.json*. Je realizován skrz tlačítko v levém panelu, které zavolá funkci `exportNetwork()`.

Funkce je rozdělena na tři části. Nejdříve pomocí metody `getPositions()`, kterou poskytuje knihovna vis.js, uložíme všechny uzly do proměnné. Metoda se volá pomocí proměnné, ve které máme uloženou vykreslenou síť a vrátí nám ID uzlů s jejich pozicemi na plátně. Díky tomu se takto exportovaný graf po importu zobrazí ve stejném uspořádání.

Pomocí metody `getConnectedEdges(ID)`, poté získáme všechny hrany k již připraveným uzlům. U každé hrany proběhne kontrola a k uzlu se uloží pouze hrany, které z něj vycházejí. Poslední částí je vytvoření elementu `a`, kterému nastavíme atributy pro uložení proměnné do souboru a jeho stažení.

3.2.5 Prohledávání grafu

Pro prohledávání grafu byla vytvořena příkazová řádka, do které uživatel zadává příkazy a aplikace je pomocí tlačítka „Potvrdit“ vykonává. Příkazy jsou zadávány společně s jejich parametry, které určují jak se mají příkazy chovat a z jakých dat mají čerpat. V rámci prohledávání grafu je v aplikaci implementováno několik příkazů. Parametry v příkazech jsou oddělovány čárkami.

3.2.5.1 Příkazy pro práci s daty

V aplikaci jsou implementované tři jednoduché příkazy, které změní obsah zobrazovacího plátna. Prvním je příkaz „clear“, který provede vyčištění plátna od všech uzlů a hran. Bylo v něm nutné pouze vytvořit prázdné instance datových kontejnerů. Tento příkaz jako jediný nemá žádné parametry.

Dalšími dvěma příkazy jsou „smallData“ a „bigData“, které vizualizují připravené testovací grafy. Graf, který je výsledkem prvního příkazu je zároveň základním grafem, který je zobrazen po spuštění aplikace.

3.2.5.2 Příkaz „neighbour“

Syntaxe:. *neighbour, zdroj dat, ID zdrojového uzlu, vzdálenost, druh zobrazení*

Příkaz, který prohledá datové kontejnery a k zadanému zdrojovému uzlu nalezne všechny jeho sousedy v zadané vzdálenosti. Je prováděn metodou `findNeighbours()`. Nejprve je provedena kontrola zda zdrojový uzel existuje. Následně je pomocí fronty a algoritmu pro prohledávání do šířky (BFS) procházeno okolí zdrojového uzlu, dokud nedosáhneme zadané vzdálenosti.

Výstupem příkazu je podle zadaného posledního parametru buď graf, ve kterém jsou všechny uzly a hrany v dané vzdálenosti nebo graf, který obsahuje pouze hrany, které uzly spojují s jeho rodičem.

BFS [20] je algoritmus, který systematickým způsobem projde všechny uzly grafu. Nejprve do fronty vložíme startovní uzel. Poté v cyklu procházíme jednotlivé uzly grafu. Opakující se postup je následující. Vyjmeme první uzel ve frontě a procházíme všechny jeho následníky. Pokud jsme objevily uzel, který ještě nebyl navštíven, uložíme si ho do již navštívených uzlů a zároveň si uložíme informaci o rodiči uzlu. Poté uzel vložíme na konec fronty. Tímto způsobem projdeme všechny uzly v grafu a díky ukládaným informacím najdeme i nejkratší cesty (která nebere ohled na váhy hran) ze startovního uzlu, ke všem ostatním uzlům.

zdroj dat Tento parametr určuje, který zdroj dat má příkaz použít. Možnosti jsou `actual`, která použije aktuálně zobrazený graf, `smallData`, která využije menší testovací data a `bigData`, který využije velká testovací data.

ID zdrojového uzlu ID uzlu, ze kterého funkce vyhledává sousedy.

vzdálenost Číslo určující do jaké vzdálenosti se sousedící uzly vyhledají.

druh zobrazení Parametr, který lze zadat jako `all` nebo `tree`. Určuje zda se při vizualizaci grafu zobrazí všechny hrany nebo pouze ty, které vedou na nové uzly.

3.2.5.3 Příkaz „path“

Syntaxe:. *path, zdroj dat, ID zdrojového uzlu, ID cílového uzlu, zobrazení, váha*

Příkaz vyhledávající nejkratší cestu mezi dvěma uzly v grafu. Je implementován v metodě `findPath()` a najde nejkratší cestu mezi uzly, která je větší než zadaná váha (pokud byla zadaná). Pokud uživatel váhu nevyplnil, nebude se na ni brát ohled. Pokud ovšem vyplněna byla, je při každém prodloužení cesty zkontrolováno, zda cesta prodloužená o další hranu splňuje minimální váhu celé cesty.

Metoda nejprve kontroluje zda zdrojový a cílový uzel existují a zda je zadaná váha. Poté je spuštěn algoritmus pro prohledávání do šířky, popsaný v 3.2.5.2. Ten je upraven o přepočítávání celkové váhy cesty. Pokud algoritmus úspěšně najde cestu, uloží graf se všemi uzly a hranami, které doposud našel. Hledanou cestu projde pomocí uložených informací a podle parametru zadaného uživatele buď změní její vzhled, aby byla zvýrazněna v celém grafu nebo ji zobrazí jako samostatný graf.

zdroj dat Parametr určuje, který zdroj dat má příkaz použít.

ID zdrojového uzlu ID uzlu, ze kterého funkce vyhledává cestu.

ID cílového uzlu ID uzlu, který je cílem cesty.

zobrazení Parametr určuje jestli se má nalezená cesta vykreslit jako samotný graf nebo má být zobrazena v původním grafu. Možnosti jsou `inGraph`, která zobrazí cestu v celém grafu a `alone`, která ji zobrazí jako samostatný graf.

váha Parametr, který je volitelný a určuje minimální hodnotu, pod kterou nesmí klesnou hodnota celé cesty mezi uzly.

3.2.5.4 Příkaz „bestPath“

Syntaxe:. *bestPath, zdroj dat, ID zdrojového uzlu, ID cílového uzlu, zobrazení*

Tento příkaz na rozdíl od předchozího nalezne nejlepší cestu mezi danými uzly. Příkaz je realizován v metodě `findBestPath()`. Metoda využívá „Floydův-Warshallův“ [21] algoritmus pro hledání nejkratších cest v grafu. V něm stačí přepsat podmínku na základě které se přepočítává nová cesta. Hlavní výhoda algoritmu je jeho implementační jednoduchost.

Algoritmus má na vstupu matici délek. Pokud mezi uzly `(i, j)` vede hrana délky l, tak matice obsahuje na indexu (i, j) právě tuto hodnotu. Na diagonále má tato matice nuly a na ostatních indexech, které nereprezentují hranu má nekonečno.

V každé iteraci algoritmu se matice přepočítává tak, aby vyjadřovala vzdálenost všech dvojic uzlů skrze iterativně se zvětšující množinu prostředníků. Po poslední iteraci na pozicích `(i, j)` nalezneme hodnotu nejkratší cesty z uzlu `i` do uzlu `j`.

V naší metodě pro hledání nejlepší cesty, jsme museli přepsat podmínku tak, aby splňovala vlastnost hran definovanou v 1.5. Při průběhu algoritmu si také udržujeme matici předchůdců. Pomocí matice předchůdců po skončení algoritmu zrekonstruujeme cestu. Na základě čtvrtého parametru bude cesta buď zobrazena jako samostatný graf nebo bude zvýrazněna v celém grafu.

zdroj dat Parametr určuje, který zdroj dat má příkaz použít.

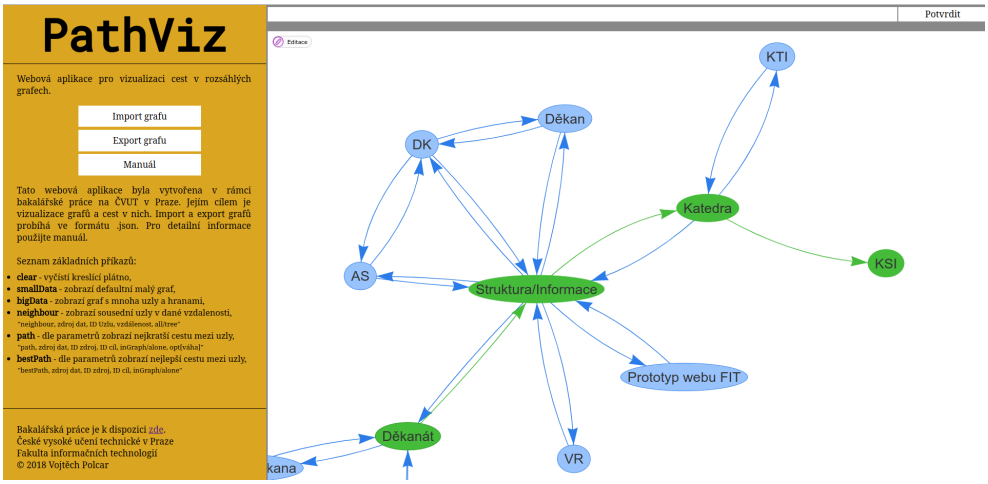
ID zdrojového uzlu ID uzlu, ze kterého funkce vyhledává cestu.

ID cílového uzlu ID uzlu, který je cílem cesty.

zobrazení Parametr určuje jestli se má nalezená cesta vykreslit jako samotný graf nebo má být zobrazena v původním grafu. Možnosti jsou stejné jako v příkazu „path“.

3.3 Testování

Testování aplikace probíhalo ve třech fázích. První probíhala přímo během vývoje. Jednalo se o klasické testování právě implementovaných funkčností. Další



Obrázek 3.2: Výsledná aplikace PathViz

fáze testovala připravené případy užití, které byly vytvořeny na základě funkčních požadavků. Poslední fází bylo testování na uživatelích, kteří zjišťovali, jak je aplikace použitelná a intuitivní.

3.3.1 Testování při vývoji

Při vývoji aplikace byly prováděny testy právě implementovaných částí. Na základě těchto testů bylo potřeba upravit několik funkcí, které byly specifikovány v analýze a návrhu (zde byly následně upraveny). Jednalo se o upravení funkcí pro manipulaci s objekty, vyladění datových struktur a přizpůsobování výsledné vizualizace.

3.3.2 Testování na základě případů užití

Na konci analytické části jsme si definovali případy užití, jakými budou uživatelé interagovat s aplikací. Při jejich postupném procházení byly nalezeny menší chyby, na které se nepřišlo během testování při implementaci.

Nejvíce chyb bylo nalezeno v odchytávání nevalidních vstupů. Všechny chyby na které se přišlo při tomto testování byly opraveny a ošetřeny. Na základě případů užití byly vyladěny funkce pro manipulaci s objekty. Zde se jednalo o úpravu editování uzlů a hran, které původně místo editace objektu stávajícího, vytvářely objekty nové.

Další částí která byla vylepšena na základě špatných výsledků testování, byla funkce pro importování grafu. Při zadávání dat jsme u hran nepovolovali nevyplněnou hodnotu `relValue`, která reprezentuje váhu dané hrany. Import původně povoloval tuto hodnotu nezadávat. Chyba byla objevena při testování příkazů pro vyhledávání cest. Následně byly implementovány kontrolní funkce, pro validaci importního souboru.

Testování probíhalo na dvou typech dat. Nejprve byly testovány případy užití pro menší testovací data v řádech desítek objektů. Výsledky byly velmi příznivé, aplikace reagovala velmi rychle na všechny typy příkazů a správně reagovala i na negativní případy.

Při testování s většími daty (v řádek stovek až tisíců objektů) se objevily neduhy aplikace. V první fázi aplikace velmi pomalu vizualizovala rozsáhlejší grafy. Tuto chybu se podařilo vyladit laděním obecného nastavení vizualizace grafů. Po úpravě aplikace rychle a stále přehledně vizualizuje menší i rozsáhlejší grafy.

Největší limitací při používání rozsáhlejších dat, byla implementace algoritmu v příkazu „bestPath“. Ten v případě, že objekty přesáhnou řádově tisíce, nedokáže podávat uspokojivé výsledky. Při testování dat z datového zdroje „bigData“ se stále ještě použít dal.

3.3.3 Testování uživatelského prostředí

Jedním z funkčních požadavků na aplikaci bylo intuitivní ovládání. Aplikace byla otestována několika uživateli a na základě jejich zpětné vazby byla zaznamenána místa ke zlepšení.

Pomocí manuálu dostupného přímo v aplikaci, byla aplikace celkově zhodnocena jako jednoduchá na ovládání. Při tomto testování byly nalezeny nekonzistentní chybové hlášky a mezery v manuálu, který je pro nezkušené uživatele stěžejní. Textové chyby, které byly nalezeny v průběhu testování byly opraveny a byl zrevidován manuál.

3.4 Zhodnocení implementace

Na začátku jsme si vytyčili cíl, vytvořit webovou aplikaci pro vizualizaci grafů a cest v nich. Daný cíl byl splněn, aplikace je funkční a přívětivá jak pro tvorbu a manipulaci s menšími grafy, tak vizualizaci a prohledávání rozsáhlejších grafů.

Aplikace je implementována tak, aby ji bylo možné rozšířit o další funkce a zároveň se dala používat nezávisle na datech, která bude vizualizovat. Silnou stránkou aplikace je možnost tvorby grafů, bez nutnosti používání souborů. V aplikaci je dostupný manuál s detailním popisem funkcí a příkazů.

Největším nedostatkem aplikace je, že není dostupná přes internet, protože není nasazena na žádném serveru. Další nevýhodou, kterou eliminuje možnost tvorby grafu v aplikaci, je nutnost dodržovat správnou strukturu při importu grafu.

Jednou z dalších nevýhod může být uzpůsobení algoritmů vyhledávání cest pro potřeby zadaných testovacích dat. Použité algoritmy jsou většinou používané na hledání nejkratších cest a násobící podmínka na základě které hledáme nejlepší/nejkratší cesty, může být v první chvíli matoucí.

Využití příkazu „bestPath“ je bohužel limitováno velikostí testovacích dat. Ten při použití opravdu rozsáhlých dat, nezvládá podávat výsledky v uspokojivém čase. V případné budoucí reimplementaci by bylo nutné příkaz přepracovat pro fungování nad daty, ve kterých by objekty překročily řádově tisíce.

Celkově hodnotím vyvinutou aplikaci jako povedenou a použitelnou pro práci s menšími a středně rozsáhlými grafy. Aplikace je intuitivní a dokáže vizualizovat menší i velmi rozsáhlá data, ale výpočetně se nemůže rovnat desktopovým aplikacím, které se zabývají vizualizací rozsáhlých grafů.

Závěr

Cílem této bakalářské práce bylo analyzovat způsoby vizualizace grafů a na základě analýzy navrhnout, implementovat aplikaci pro vizualizace grafů a cest v nich.

Během analytické části jsme se seznámili s různými technikami pro vizualizaci grafů, možnostmi jak vizualizovat grafy ve webové aplikaci a popsali jsme existující řešení této problematiky. Na základě analýzy a struktury testovacích dat jsme vytyčili požadavky na výslednou aplikaci a definovali případy užití.

S podklady z analýzy a vytyčenými požadavky byl vytvořen návrh webové aplikace pro vizualizaci grafů. Aplikace byla navržena tak, aby byla jednoduše použitelná a zároveň disponovala funkcemi pro tvorbu a vizualizaci grafů a cest v nich. Podle analýzy a návrhu byla aplikace implementována a otestována na základě případů užití a dat z informační architektury webu FIT.

Implementace proběhla úspěšně a aplikace je funkční nezávisle na testovaných datech. Poradí si s malými i rozsáhlejšími grafy a lze ji použít pro vizualizaci jakýkoliv dat. Aplikace je zároveň připravena pro rozšíření a implementování dalších funkcí.

Literatura

[1] Prof. Ing. Pavel TVRDÍK, C. *Základy grafů*. [Přednáška], 2017, [cit. 2018-05-12].

[2] STRASSNER, T. *XML vs JSON*. [online], 2018, [cit. 2018-05-12]. Available from: `https://tinyurl.com/yctjatww`

[3] TARAWANEH, R. M.; KELLER, P.; et al. A General Introduction To Graph Visualization Techniques. In *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, *OpenAccess Series in Informatics (OASIcs)*, volume 27, edited by C. Garth; A. Middel; H. Hagen, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, ISBN 978-3-939897-46-0, ISSN 2190-6807, pp. 151–164, doi:10.4230/OASIcs.VLUDS.2011.151, [cit. 2018-05-12]. Available from: `http://drops.dagstuhl.de/opus/volltexte/2012/3748`

[4] WARD, M.; GRINSTEIN, G.; et al. *Visualization Techniques for Trees, Graphs, and Networks*, chapter 8. A.K. Peters, 2010, ISBN 9781482257373, pp. 8–9, [cit. 2018-05-12]. Available from: `https://tinyurl.com/y8tbskle`

[5] *NeoViz*. [online], [cit. 2018-05-12]. Available from: `http://www.neoviz.fr/index.php/fr/un-petit-tour-au-zoo-de-la-dataviz`

[6] *The Shapes of CSS*. [online], [cit. 2018-05-12]. Available from: `https://css-tricks.com/examples/ShapesOfCSS/`

[7] JAHODA, B. *SVG*. [online], 2013, [cit. 2018-05-12]. Available from: `http://jecas.cz/svg`

[8] JACOMY, A.; PLIQUE, G. *Sigma Js*. [online], 2015, [cit. 2018-05-12]. Available from: `http://sigmajs.org/`

[9] BOSTOCK, M. *D3.js*. [online], 2017, [cit. 2018-05-12]. Available from: https://d3js.org/

[10] ALMENDE, B. *Vis.js*. [online], 2015, [cit. 2018-05-12]. Available from: http://visjs.org/

[11] SHANNON, P.; MARKIEL, A.; et al. *Cytospace*. [online], 2003, [cit. 2018-05-12]. Available from: http://www.cytoscape.org/

[12] graphAlchemist. *Alchemy.js*. [online], [cit. 2018-05-12]. Available from: https://tinyurl.com/y9lqrtc7

[13] PEIXOTO, T. P. *Graph-tool*. [online], [cit. 2018-05-12]. Available from: https://graph-tool.skewed.de

[14] *Graphviz*. [cit. 2018-05-12]. Available from: http://www.graphviz.org/documentation/

[15] BASTIAN, M. *Gephi*. [online], 2008, [cit. 2018-05-12]. Available from: https://gephi.org/

[16] Partl, S. S. M. a., C.; Gratzl. *Pathfinder*. ISSN 1467-8659, doi:doi:10.1111/cgf.12883, [cit. 2018-05-12]. Available from: http://dx.doi.org/10.1111/cgf.12883

[17] ČÁPKA, D. *1. díl - Úvod do JavaScriptu*. [online], [cit. 2018-05-12]. Available from: https://tinyurl.com/y7rkh43l

[18] Neo4j, I. *The Neo4j Graph Platform*. [online], 2018, [cit. 2018-05-12]. Available from: https://neo4j.com/

[19] *DataSet*. [cit. 2018-05-12]. Available from: http://visjs.org/docs/data/dataset.html

[20] JAIN, S. *Breadth First Search or BFS for a Graph*. [online], [cit. 2018-05-12]. Available from: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[21] NECKÁŘ, J. *Floydův-Warshallův algoritmus*. [online], 2016, [cit. 2018-05-12]. Available from: https://www.algoritmy.net/article/5207/Floyd-Warshalluv-algoritmus

Příloha A

Seznam použitých zkratek

HTML HyperText Markup Language

CSS Cascading Style Sheets

API Application Programming Interface

WebGL Web Graphics Library

VRML Virtual Reality Modeling Language

GEXF Graph Exchange XML Format

GML Graph Modelling Language

SVG Scalable Vector Graphics

DOM Document Object Model

OpenMP Open Multi-Processing

URI Uniform Resource Identifier

BFS Breadth-first search

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu média
- `text` ..... text práce
  - `BP_Polcar_Vojtěch_2018.pdf` ..... text práce ve formátu PDF
  - `BP_Polcar_Vojtěch_2018.tex` ..... zdrojová forma práce
  - `img` ..... složka s obrázky použitými v práci
  - `BP_parts` ..... jednotlivé části BP ve formátu LaTeX
- `src` ..... implementace aplikace
  - `css` ..... složka s kaskádovými styly
  - `img` ..... obrázky použité v aplikaci
  - `data` ..... obecné nastavení a zdrojová data
  - `js` ..... zdrojové kódy v jazyce Javascript