ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vizualizace cest v grafu
Student:	Matěj Pokorný
Vedoucí:	Ing. Lukáš Bařinka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat aplikaci, která umožní vizualizovat cesty v grafu.
1. Proveďte rešerši obdobných aplikací a též knihoven či modulů, které práci s grafy usnadňují.
2. Na základě rešerše dopřesněte požadavky na vlastní aplikaci, která bude vizualizovat cestu ve 2D/3D grafu.
3. Navrhněte vlastní řešení, které bude poskytovat GUI i API pro tvorbu a vizualizaci cest v grafu.
4. Diskutujte a zvolte vhodnou implementační platformu, zvažte využití knihoven/modulů z bodu 1.
5. Návrh implementujte, zdokumentujte a řádně otestujte na vhodných datech.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 1. března 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Vizualizace cest v grafu

Matěj Pokorný

Vedoucí práce: Ing. Lukáš Bařinka

16. května 2017

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 16. května 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Pokorný, Matěj. *Vizualizace cest v grafu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

S vizualizací cest v grafu se dnes setkáváme v mnoha oblastech lidského bádání i každodenního života. Ať už pomáhá řidiči v podobě GPS navigace nebo administrátorovi při analýze zatížení sítě, je její správnost a přehlednost k pochopení problematiky často klíčová. V této práci jsou popsány jednotlivé techniky vizualizace cest v grafech a obecné nástroje, které se tímto problémem zabývají. Tyto nástroje jsou však pro začínající uživatele často příliš komplikované. V rámci této práce proto byla vyvinuta webová aplikace, která si klade za cíl co nejsnazší a nejrychlejší vizualizaci cest v grafu při zachování funkcionalit pokročilejších nástrojů v této oblasti. Aplikace je dostupná online.

Klíčová slova vizualizace cest v grafu, implementace webové aplikace, algoritmy prohledávání grafu, force-directed vykreslování grafu, TypeScript, d3.js

Abstract

People come across visualization of paths in graph in many research areas and also in everyday life. Whether it helps a driver as a GPS navigation or an administrator with site load analysis, its accuracy and clarity is often the key of understanding this issue. In this thesis are described individual techniques of

paths visualization in graphs and general tools, which deal with that problem. However, these tools are often too complicated for new users. Therefore a web application was developed. The aim of the application is to visualize paths of a chart following the easiest and the fastest method. Functionality of more advanced tools in this field is preserved. The application is available online.

Keywords graph path visualization, implementation of web application, graph traversal algorithms, force-directed graph drawing, TypeScript, d3.js

Obsah

Seznam obrázků

Seznam ukázek zdrového kódu

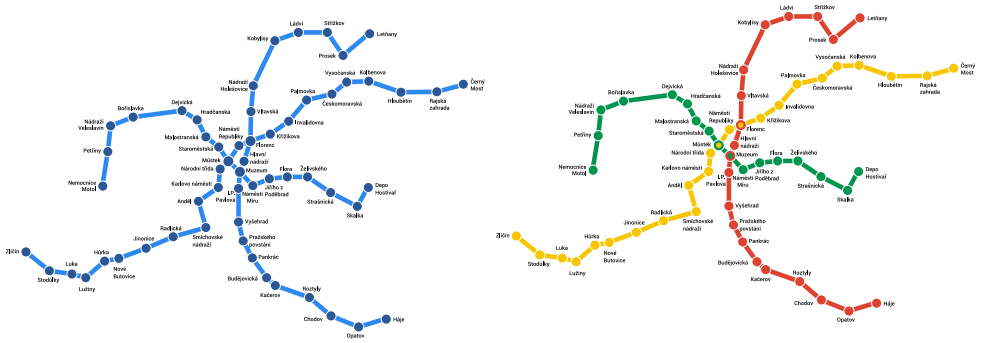
Seznam tabulek

Úvod

Cílem této bakalářské práce je popsat jednotlivé techniky vizualizace cest v grafech, zhodnotit dostupné nástroje, které se touto problematikou zabývají, a navrhnout a vytvořit webovou aplikaci, která do této oblasti přinese něco nového. V následujících odstavcích vysvětlím motivaci, proč jsem si zvolil právě toto téma, a jeho přínos do již značně prozkoumaných vod matematické disciplíny zvané teorie grafů.

Nejprve si ale na příkladu připomeňme pojem graf a cesta v grafu. Představme si mapu pražského metra (0.1b). Síť stanic propojených třemi linkami. V terminologii teorie grafů nazveme stanici **uzlem** a spojení mezi dvěma sousedními stanicemi **hranou**. **Cesta v grafu** pak může vést například ze stanice (uzlu) Anděl do stanice (uzlu) Vyšehrad. Poznal by ovšem cizinec z černobílé varianty tohoto plánku (0.1a), jestli bude muset přestupovat, popřípadě kde to pro něj bude nejvýhodnější?

Teorie grafů, jejíž součástí grafy jsou, se vyskytuje v celé řadě především

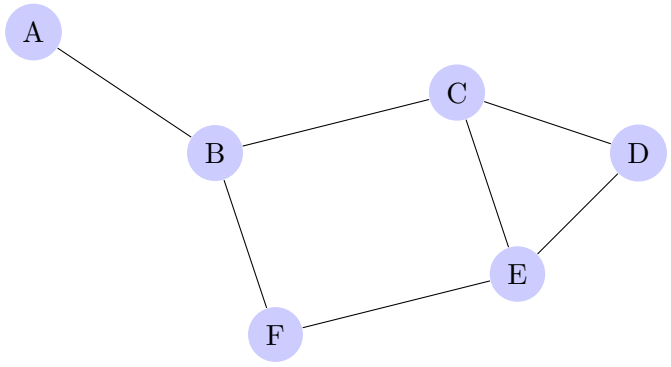


(a) Plán pražského metra.

(b) Plán pražského metra s vyznařenými linkami.

Obrázek 0.1: Porovnání plánu pražského metra s vyznačenými linkami (0.1b) oproti verzi bez vyznačení (0.1a).

technických oborů. V autě jistě oceníme, když nám GPS navigace nalezne a na dopravní mapě přehledně zobrazí nejkratší trasu do cílové stanice (uzly jsou zde města a silnice mezi nimi hrany). Na schématu krevního oběhu člověka nás zase zajímá především směr proudění krve s barevným odlišením tepen od žil a samozřejmě správné vyobrazení jejich vnitřního průměru. Pro elektrikáře je zase často nezbytný nákres elektroinstalace domu s vyznačenými okruhy obvodů. Správná a přehledná vizualizace grafů a cest jimi procházejících je tedy velmi důležitá v mnoha vědních oborech i běžných každodenních situacích.



Obrázek 0.2: Příklad obecného grafu

Když se ale nyní oprostíme od výše uvedených příkladů a představíme si graf jako obecnou strukturu, lze řešit problém vizualizace cest v těchto grafech také, do jisté míry, obecně. Hledání nejkratší cesty např. z uzlu A do uzlu F je obecný problém a pokud naleznu obecný postup, jak tuto cestu vždy a spolehlivě najít a vizualizovat, mohu pak tento princip využít na plánku metra, mapě České republiky, schématu krevního oběhu nebo kdekoliv jinde. V tom tkví krása teorie grafů a má hlavní motivace věnovat se v mé bakalářské práci právě tomuto tématu.

Nástrojů pro vizualizaci cest v grafech je již celá řada. Často jsou však pro začínající uživatele příliš náročné na prvotní pochopení. Pokusím se tedy vytvořit novou aplikaci, která bude více zaměřená na snadné a rychlé použití při zachování funkcionalit pokročilejších nástrojů.

Kapitola 1

Analýza

V této kapitole provedu rešerši aktuálního stavu problematiky vizualizace cest v grafech. Tedy formáty, ve kterých se grafy (a cesty v grafech) definují, postupy, jak se vykreslují, a nakonec uvedu i přehled nejznámějších nástrojů, které se touto problematikou zabývají.

1.1 Reprezentace grafu

Před samotnou vizualizací (cest) grafu si musíme definovat strukturu, která ho reprezentuje. V úvodu této sekce zavedu formální definici grafu spolu s jeho základními vlastnostmi a v jednotlivých podsekcích poté nejčastější formáty, na které lze dnes narazit.

Definice 1. Obyčejný neorientovaný graf je dvojice $G = (V, E)$, kde V je množina uzlů a $E = \{\{u, v\} | u, v \in V \wedge u \neq v\}$ je množina hran.

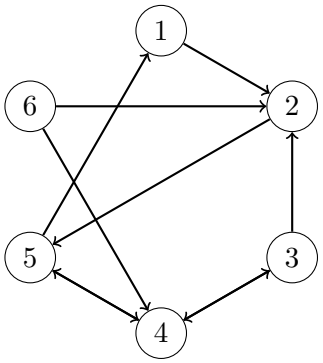
Definice 2. Obyčejný orientovaný graf je dvojice $G = (V, E)$, kde V je množina uzlů a $E = V \times V$ je množina hran.

Definice obyčejného neorientovaného grafu (1) se tedy liší od toho orientovaného (2) v množině hran. Hrany u orientovaného grafu jsou uspořádané dvojice uzlů (u, v), kde má uzel u odchozí hranu a uzel v hranu příchozí. Vypuštěním podmínky $u \neq v$ (tedy, že uzel nesmí incidovat sám se sebou) pak dostáváme graf prostý. Obecný graf ještě navíc povoluje více rovnoběžných hran (existují dvě hrany, které mají stejný počáteční i koncový uzel). Graf s hranami incidujícími s více jak dvěma uzly nazýváme hypergraf. V dalších částech této práce se budu věnovat především obyčejným grafům (orientovaným i neorientovaným).

Definice 3. Cesta v grafu $G = (V, E)$ je posloupnost $P = (v_0, e_1, v_1, \ldots, e_n, v_n)$, pro kterou platí $e_i = (v_{i-1}, v_i)$ a navíc $v_i \neq v_j$ pokud $i \neq j$.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

(a) Matice sousednosti



(b) Výsledný graf

Obrázek 1.1: Matice sousednosti (1.1a) a její graf (1.1b)

Cesta je tedy posloupnost vrcholů, pro kterou platí, že v grafu existuje hrana z daného vrcholu do jeho následníka. Žádné dva vrcholy (a tedy ani hrany) se ale v cestě nesmí opakovat[1]. **Tah** pak nazýváme posloupnost P, ve které se mohou opakovat vrcholy. Ještě větší relaxací je **sled**, kde se mohou opakovat i hrany.

1.1.1 Matice sousednosti

Jedná se o velmi úsporný a přitom přehledný formát pro reprezentaci grafu.

Definice 4. Mějme obyčejný graf $G = (V, E)$ s vrcholy V indexovanými $1, 2, \ldots, n$. Jeho matici sousednosti nazýváme binární matici $\mathbf{A} \in \{0,1\}^{n \times n}$, kde $a_{i,j}$ má hodnotu **1**, pokud $\exists(u_i, v_j) \in E$, jinak **0**.

Pokud u matice sousednosti zrelaxujeme podmínku $\mathbf{A} \in \{0,1\}^{n \times n}$ např. na $\mathbf{A} \in \mathbf{Z}^{n \times n}$, je zde prostor pro zakódování dalších informací o incidujících hranách. Formát už poté ale ztrácí na čitelnosti a především nemůžeme bez dalších informací nic říct o uzlech grafu.

1.1.2 Jazyk DOT

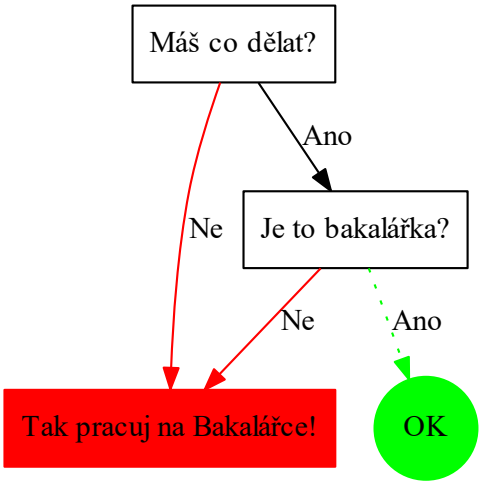
Jazyk DOT vznikl již na přelomu 80. a 90. let v AT&T laboratořích pro potřeby softwaru na vizualizaci grafů - Graphviz. Postupem času jeho syntaxi (či pouze subset) začaly podporovat i další nástroje pro vizualizaci grafů a dnes se tak jedná o poměrně rozšířený formát reprezentace grafu.

[1] Při vizualizacích cest v grafech nám opakující se vrcholy a někdy i hrany příliš nevadí, nebo je to dokonce žádoucí. V mé práci se tak sice nadále budu odkazovat na cesty, často však budu mít na mysli spíše tahy či sledy. Tam kde bude tato terminilogie důležitá, čtenáře upozorním.

```
digraph "Volný čas" {
    node [shape = box]; // Obecné nastavení uzlů
    start [label = "Máš co dělat?"]
    bp [style = filled, color = red, label = "Tak pracuj na Bakalářce!"]
    ok [style = filled, label = "OK", color = green, shape = circle]
    subject [label = "Je to bakalářka?"]
    start -> bp [label = "Ne", color = red]
    start -> subject [label = "Ano"]
    subject -> bp [color = red, label = "Ne"]
    subject -> ok [style = dotted, color = green, label = "Ano"]
}
```

Listing 1: Ukázka syntaxe jazyka DOT (výsledek 1.2).

DOT umožňuje reprezentaci obecných orientovaných i neorientovaných grafů (případně podgrafů) s velkým množstvím atributů pro uzly i hrany. Definice každého grafu sestává z určení jeho typu (`graph` pro neorientovaný graf či `digraph` pro orientovanou verzi grafu), názvu a následném těle se samotnou strukturou grafu. Zde lze pomocí klíčových slov `graph`, `node` či `edge` definovat atribut celého grafu, všech uzlů nebo hran. Atributy se uvádějí v hranatých závorkách oddělené čárkou (vždy dvojice `[atribut] = [hodnota]`). Dále lze uvnitř grafu definovat jednotlivé uzly a hrany (opět na ně lze navěsit atributy). Orientovanou hranu značíme `A -> B`, neorientovanou pak `A -- B`. Kromě uzlů a hran může graf obsahov i podgraf uvozený klíčovým slovem `subgraph`.



Obrázek 1.2: Graf v jazyce DOT vykreslený pomocí programu Graphviz (zdroj 1).

S jazykem DOT lze tedy poměrně snadno a především přehledně popsat i složité grafové struktury. Navíc, jak jsem již zmínil v úvodu, jazyk DOT podporuje velké množství nástrojů pro vykreslování grafů.

Cestu lze v grafu specifikovat posloupností názvů uzlů. Např. `A -- B -- C -- D`. Je však potřeba specifikovat všechny uzly, kterými cesta prochází, což může být v případě dlouhých cest problematické.

1.1.3 JSON/XML

JSON[2] a XML[3] jsou standardní textové formáty pro uchovávání obecných dat. Lze tak s nimi definovat i graf. Výhodou těchto formátů je, že je podporuje praktický každý programovací jazyk/technologie. Pro potřeby mé práce zde popíši definici grafu ve formátu JSON s popisem, jak lze tento formát převést do XML varianty.

```
1  {
2      "title": "Trojúhelník",
3      "nodes": [
4          { "id": 0, "label": "A", "color": "red" },
5          { "id": 1, "label": "B", "size": 2 },
6          { "id": 3, "label": "C", "shape": "rectangle"}
7      ],
8      "edges": [
9          { "id": 0, "source": 0, "target": 1 },
10         { "id": 1, "source": 1, "target": 1, "color": "blue" },
11         { "id": 3, "source": 2, "target": 0, "label": "C - A" }
12     ]
13 }
```

Listing 2: Ukázka formátu JSON.

Jak je z ukázky (2) formátu vidět, jedná se o podobnou strukturu, jakou nám zavádí i obecná definice grafu. Tedy množina uzlů a hran. Pracujeme zde ale s identifikátorem uzlu a jednotlivé uzly a hrany mohou mít další dodatečné vlastnosti. Dodatečných vlastností může být celá řada a různé nástroje podporují různé vlastnosti. Nebudu se zde proto zabývat jednotlivými vlastnostmi. Mnou podporované vyspeficikuji v pozdější části mé práce.

Mezi známé specifikace využívající JSON nebo XML formát jsou např. GEXF (Graph Exchange XML Format), GraphML, GraphSON nebo JGF

[2] JSON (**J**ava**S**cript **O**bject **N**otation) je velmi jednoduchý formát pro uchovávání dat v textové podobě.

[3] XML (e**X**tensible **M**arkup **L**anguage) je textový formát s hierarchickou strukturou pro uchovávání dat v textové podobě.

	Matice sousednosti	DOT	JSON/XML
Obyčejný graf	✓	✓	✓
Prostý graf	✓	✓	✓
Obecný graf	✓[4]	✓	✓
Multigraf			✓
Vlastnosti uzlů		✓	✓
Vlastnosti hran	✓[5]	✓	✓
Podgrafy		✓	✓
Specifikace layotu			✓
Animace			✓
Definice cest		✓	✓

Tabulka 1.1: Tabulka porovnávající grafové formáty.

(JSON Graph Format). Jedná se však zpravilda o formáty, které jsou svázané s konkrétní aplikací a nemají velké rošíření. Proto se o nich podrobněji rozepíši u konkrétních nástrojů pro práci s grafy.

Převod z formátu JSON do XML je velmi snadný. JSON se totiž skládá pouze z párů `key : value`, kde klíč (`key`) je vždy řetězec a hodnota (`value`) je buď primitivní typ (řetezec, číslo, ...), seznam hodnot nebo opět seznam párů. Klíč tak je při převodu vždy nový XML element a hodnota v případě primitivního typu přímo jeho obsah, v případě seznamu párů nový XML element a v případě seznamu hodnot je zde potřeba vytvořit seznam XML elementů. Je zde pouze problém s pojmenováním kořenového XML elementu a XML elementů přetransformovaných ze seznamu hodnot. Ani na jednom pojmenování však v případě využití XML formátu pro reprezentaci grafu nezáleží.

1.1.4 Porovnání formátů

Tabulka (1.1) ukazuje podporu různých vlastností grafů výše zmíněnými formáty. Nástroje pracující s grafy ovšem většinou podporují pouze podmnožinu toho, co podporuje samotný formát. U JSON/XML je navíc kritické, jakou specifikaci si zvolíme.

1.2 Vykreslení grafu

Volba správných pozic uzlů v grafu a stylu hran mezi nimi pro co možná nejpřehlednější vizualizaci je poměrně náročný problém. Abychom vůbec mohli automatizovaně porovnat zdařilost takovéto vizualizace, využívá se hned několika odlišných metrik[1].

- Počet hran v grafu, které se kříží s jinými hranami. U planárních grafů[6]

[6]Planární graf se vyznačuje vlastností, že lze nakreslit bez jediné hrany křížící se s jinou hranou.

je dokonce žádoucí, aby se nekřížila ani jedna hrana.

- Velikost rámečku (obdelníkového tvaru), kterým lze celý graf ohraničit.
- Symetričnost výsledného grafu.
- Jak moc jsou hrany mezi uzly přímočaré (např. hrana s 10 záhyby nepůsobí moc přehledně). Hrany však mohou být často i komplexnější křivky.
- Celková délka hran grafu.
- Nejmenší úhel svírající dvě hrany vycházející ze stejného uzlu[7]. Větší úhly přispívají k lepší čitelnosti grafu.
- Celkový počet směrnic hran[8].

Samotné pozicování (pokud nejsou zadány souřadnice uzlů) je dnes nejčastěji prováděno tzv. force-directed metodou (ve volném překladu silově řízené vykreslování grafu). Při vykreslování touto metodou se typicky nejprve všem uzlům nastaví odpudivá síla, naopak hrany mezi uzly představují sílu přitažlivou. Následně se spustí samotná simulace, kde jsou jednotlivé uzly přitahovány a odpuzovány, dokud nedojde k rovnovážnému stavu v celé simulaci. Tato metoda podává relativně dobré výsledky (ukázka 1.3), je jednoduchá na implementaci a především je nezávislá na typu grafu.

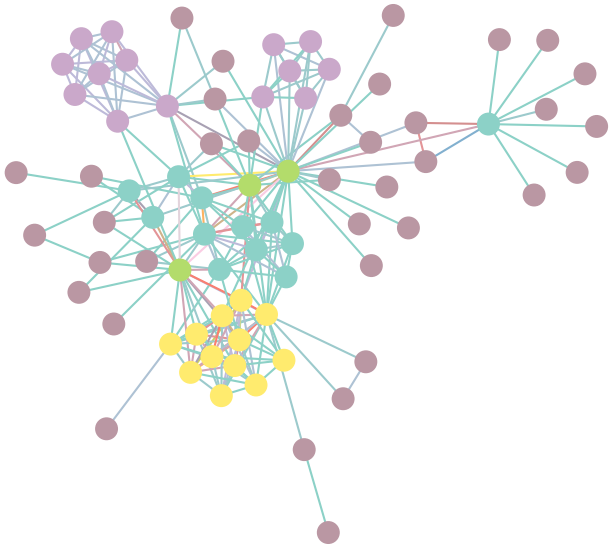
Force-directed metoda ovšem nezohledňuje např. planaritu grafu. Pokud je graf planární, je vhodnější využít některý ze systematičtějších způsobů pozicování uzlů. Nejčastějšími způsoby jsou

- Ortogonální uspořádání (pravoúhle hrany mezi uzly).
- Stromová struktura, popřípadě několik vrstev.
- Uspořádání uzlů do kruhu.
- Obloukový diagram.
- Dominantní kreslení grafu (u orientovaných grafů nevede hrana nikdy doprava).

1.3 Existující nástroje pro vizualizaci

Programů pro vizualizaci a práci s grafy je celá řada. V této sekci popíši pouze ty nejzajímavější, popřípadě přínosné pro mou práci. Pokusím se zároveň zmínit jejich hlavní výhody a omezení. Nebudu zde popisovat placené produkty jako je Mathematica, KeyLines a podobně, protože je často nemám možnost

[7] V angličtině se této vlastnosti říká **angular resolution**.
[8] V angličtině se tato vlastnost nazývá **slope number**.



Obrázek 1.3: Typická ukázka grafu pozicovaného metodou force-directed.

bez koupě otestovat. Vynechám také velmi specifické programy typu GPS navigace, které sice podporují velmi zdařilé vizualizace cest, neumí ale pracovat s obecnými grafy.

1.3.1 Graphviz

Pravděpodobně nejstarší, a přitom stále využívaný nástroj pro vizualizaci grafů je Graphviz. Jedná se o multiplatformní balík sady nástrojů, kterou lze ovládat pouze pomocí příkazové řádky[9]. Graphviz umí vykreslit grafy do opravdu velkého množství formátů. Navíc podporuje některé pokročilé grafové konstrukce, jako jsou podgrafy a podporuje více odlišných uspořádání uzlů. Jako mírnou nevýhodu považuji jediný vstupní formát – jazyk DOT popsaný výše (1.1.2).

1.3.2 Gephi

Gephi je open-source nástroj pro vizualizaci a práci s grafy. Podporuje několik vstupních i výstupních formátů, umožňuje pokročilou analýzu a manipulaci s grafy a v neposlední řadě, oproti např. Graphvizu, se stále aktivně vyvíjí.

Gephi původně vznikl jako studentský projekt na univerzitě v Compiègne v roce 2008. Později okolo něj vzniklo konsorcium složené z vlivných spo-

[9]Graphviz má i grafické nástavby jako GVEdit nebo dotty. Jejich přidaná hodnota je ale prakticky nulová.

lečností podporující další vývoj softwaru. Gephi je vyvíjena jako desktopová multiplatformní aplikace s důrazem na rychlost, uživatelskou přívětivost a co možná největší podporu v této oblasti.

Nástroj využívá primárně vlastní formát GEXF, který nepodporuje přímé definice cest. Lze však využít funkci pro nalezení nejkratší cesty. Pomocí filtrů lze také zvýraznit pouze podmnožinu uzlů a hran, které mají nějakou společnou vlastnost (zbytek grafu lze buď úplně skrýt nebo mu nastavit větší průhlednost). Lze tak vyspecifikovat pomocí nejrůznějších vlastností i cestu. Nenašel jsem ale volbu zobrazení cesty na základě krajních hran.

1.3.3 Cytoscape

Cytoscape je multiplatformní open-source nástroj pro práci s grafy. Podobně jako Gephi cílí především na akademickou a vědeckou činnost. Podporuje širokou škálu vstupních a výstupních formátů, mnoho typů grafových layotů a další pokročilou práci s grafy. Aplikace má také svůj vlastní obchod s pluginy s přehledným filtrováním a statistikou popularity pluginu. Pluginy jsou často tvořeny s cílem rozšířit aplikaci v nějaké specifické doméně grafové problematiky. Pluginy lze psát v programovacím jazyce Java.

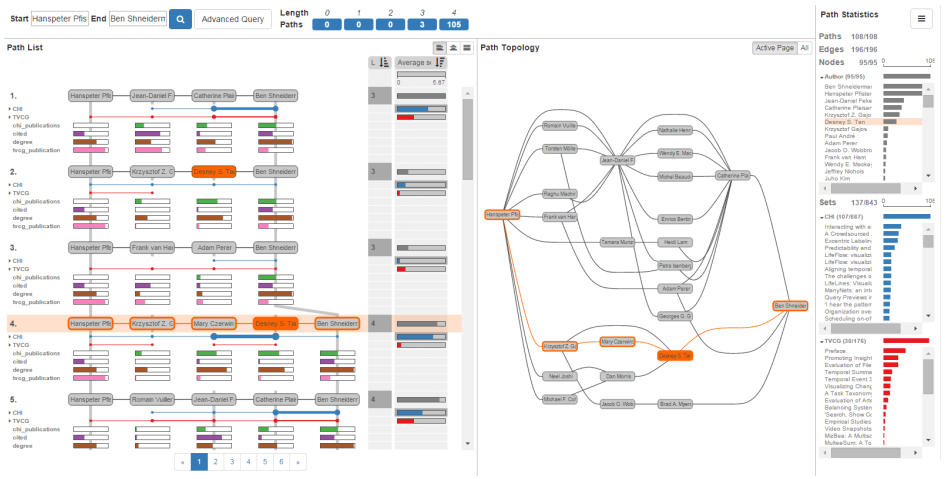
Vizualizace cest v Cytoscape funguje podobně jako v aplikaci Gephi. Nejprve je potřeba vyfiltrovat podle velkého množství kritérii konkrétní uzly a hrany (pro nalezení nejkratší cesty mezi dvěmi uzly mi nejlépe posloužil plugin PathLinker). Poté na celý přefiltrovaný graf nastavit efekty (barva, tloušťka hrany, ...), které ho odliší a následně se přepnout zpět do kontextu celého grafu.

Skupina vývojářů stojící za Cytoscape vyvíjí také JavaScriptovou knihovnu Cytoscape.js, schopnou vykreslovat grafy přímo v prohlížeči. Velká výhoda tohoto spojení pak spočívá v tom, že většinu grafů načtených v programu Cytoscape lze snadno vyexportovat a zobrazit v prohlížeči právě pomocí této knihovny.

1.3.4 Pathfinder

Program Pathfinder je přímo zaměřený na hledání a vizualizaci cest ve velkých grafech[2]. V prvním kroku nalezne všechny nejkratší cesty (nebo i o něco delší, záleží na nastavení) mezi dvěma uzly a zobrazí je v samostatném grafu. Mezi těmito cestami lze pak dále filtrovat a zobrazit si opravdu jen relevantní výsledky. Tento postup lze zvolit i u programů Gephi a Cytoscape, zde je však celý proces nejsnazší. Jako nevýhodu vnímám, že přefiltrované cesty nelze nijak zobrazit v původním grafu.

Pathfinder je postaven nad webovým frameworkem Caleydo a jedná se tak o aplikaci spustitelnou přímo z internetového prohlížeče. Aplikaci se mi ovšem nepodařilo spustit.



Obrázek 1.4: Rozhraní programu Pathfinder.

Kapitola 2

Návrh

V předchozí kapitole jsem provedl rešerši existujících řešení vizualizace (cest) grafů. V této kapitole tyto znalosti zhodnotím a navrhnu řešení, které bude mít co možná největší přínos v této problematice.

2.1 Zaměření

V rámci této práce nemohu vytvořit řešení, které by co do počtu funkcí, rychlosti, počtu podporovaných formátů a podobně mohlo konkurovat velkým systémům typu Gephi, Cytoscape nebo Mathematica. Zaměřím se tedy pouze na vizualizaci menších grafů, a to pouze na 2D ploše. Dále se omezím pouze na jeden vstupní/výstupní formát reprezentace grafu. Neplánuji ani implementovat podporu specifičtějších vlastností grafu, jako jsou animace, toky v síti a podobně.

Smyslem mé práce je přiblížit teorii grafů, popř. problematiku vizualizace cest v grafech spíše novým studentům. Oproti nástrojům Gephi nebo Cytoscape popsaným v kapitole 1.3 si tak kladu za cíl pochopení problematiky v řádu jednotek minut.

2.2 Platforma

Velké nástroje pro práci s grafy vyžadují bez výjimky instalaci. Je to pochopitelné, ale pro nováčky v této oblasti, kterým stačí vizualizovat jeden malý graf, často zbytečná komplikace navíc. Nástroj je pak mnohdy omezen jen na některé platformy, jsou zde „otravné“ aktualizace a vznikají platformě závislé chyby.

Díky specifickému zaměření mého nástroje jsem se rozhodl pro formu webové aplikace. Žádná instalace, vždy aktuální verze, dostupnost z jakéhokoliv zařízení, a to vše jen z krátkého odkazu `https://erol.cz`. Podobnou cestu

zvolil program Pathfinder (1.3.4), který ovšem není nikde dostupný online ve funkční verzi.

V aplikaci chci využít moderní HTML5 API, proto budu cílit pouze na moderní webové prohlížeče. Tedy Google Chrome (58) (včetně mobilní verze), Firefox (53) (včetně mobilní verze), Edge (15), Safari (10) a Opera (45).

2.3 Reprezentace grafu

Graf budu ukládat ve formátu JSON. Jak jsem již popsal v rešerši, je to velmi čitelný, moderní formát, který navíc webové prohlížeče nativně podporují. Reprezentaci grafu v tomto formátu lze rozdělit do šesti kategorií.

- Obecná nastavení grafu (název, typ, ...).
- Obecná nastavení uzlů.
- Obecná nastavení hran.
- Specifikace jednotlivých uzlů.
- Specifikace jednotlivých hran.
- Specifikace jednotlivých cest v grafu.

Kompletní speficikaci formátu lze nalézt v tabulce 2.1. Zde jen stručně popíši některé její výhody.

Zajímavá je především specifikace cest v grafu (klíč `paths`), pro kterou jsem v žádném jiném nástroji podporu nenašel. U každé cesty je nutnost specifikovat počáteční (`source`) a koncový (`target`) uzel (na pořadí záleží pouze u orientovaného grafu) a popřípadě dodefinovat uzly, přes které musí cesta procházet (`through`). Cesta se pak automaticky zvolí jako ta nejkratší mezi těmito uzly. Aby měla takováto definice cesty smysl, je ke každé cestě vhodné dodefinovat styly uzlů a hran. Za zajímavé považuji také obecné nastavení stylů uzlů a hran. Uživatel se tak vyhne repetitivnímu zadávání stejné vlastnosti pro všechny uzly. Tuto funkci však již podporuje více specifikací...

Parametry jednoho uzlu/hrany tak lze definovat na několika místech a je tedy potřeba definovat i prioritu, které nastavení se nakonec projeví na výsledné vizualizaci.

Nejnižší prioritu budou mít obecná nastavení uzlů a hran. Ve většině případů je potřeba nastavit některou vlastnost pro všechny a teprvé poté selektivně vybrat výjimky.

U prioritizace definic vlastností uzlů/hran společných pro cestu na úkor definic jednotlivých uzlů/hran už situace tak jednoznačná není. Prioritizovat vlastnosti uzlů/hran u definice cest jsem se rozhodl z následujícího důvodu.

Klíč	Datový typ	Výchozí hodnota
	Graph	
title?	string	
node?	Node Attributes	
edge?	Edge Attributes	
paths?	Path[]	
nodes	Node[]	
edges	Edge[]	
	Node Attributes	
label?	string	
color?	string	"#2980b9"
radius?	number	5
opacity?	number <0.0 - 1.0>	1.0
	Edge Attributes	
color?	string	"#999"
thickness?	number	2
opacity?	number <0.0 - 1.0>	1.0
	Path	
source	number | string	
target	number | string	
through?	number | string | []	
node?	Node Attributes	
edge?	Edge Attributes	
	Node + Node Attributes	
id	number | string	
x?	number	
y?	number	
	Edge + Edge Attributes	
source	number | string	
target	number | string	

Tabulka 2.1: Definice JSON formátu podporující aplikace Erol.cz. Otazník u klíče značí nepovinnou hodnotu, znak | pak anotuje možnost volby datového typu hodnoty.

Pokud bude na vstupu graf, který bude mít mnoho vlastností navázaných přímo na konkrétní uzly/hrany, vizualizace cest pak přestane mít smysl, protože se na výsledku nemá šanci nijak projevit. Naopak pokud zadefinuji cestu a chtěl bych ještě některý z uzlů v cestě dále specificky zvýraznit, mohu v krajním případě zadefinovat novou cestu, kde bude tento uzel počáteční i koncový. Seznam cest se totiž vyhodnocuje vždy postupně a poslední zadefinovaná cesta má tak nejvyšší prioritu.

2.4 Uživatelské prostředí

Aplikace je určená spíše začínajícím studentům v teorii grafů. Je tedy velmi důležitý pozitivní první dojem, snadné ovládání a obecně spíše výukovější forma aplikace.

Celá aplikace tak bude řešená formou jediné stránky rozdělené na tři části. Boční panel s informacemi o aplikaci, instrukcemi pro nové uživatele a především ukázkovými grafy, ze kterých bude jasně patrný smysl a fungování aplikace. V patičce bočního panelu pak bude odkaz na kompletní dokumentaci k aplikaci a odkaz na tuto práci.

V horních 60% stránky se bude nacházet samotné plátno s vykresleným grafem. Nebude zde chybět podpora pro přiblížení/oddálení grafu a pohyb po plátně.

Pod grafem bude interaktivní editor s možností definovat reprezentaci grafu ve formátu JSON. Graf se automaticky překreslí při každé změně tohoto vstupního formátu.

Tyto dvě části bude oddělovat panel se sadou příkazů pro práci s grafem. Bude zde možnost importovat a exportovat graf, zobrazit názvy uzlů na plátně a získat odkaz na aktuální graf.

Posledním významným prvkem stránky bude notifikační lišta s informacemi o aktuálním stavu grafu (např. chybně zadaný graf, neexistující cesta v grafu a podobně...). Notifikace budou buď informačního charakteru nebo varování nebo chyba.

2.5 Technologie pro vykreslování

Moderní webové prohlížeče dnes podporují několik odlišných technik pro vykreslování grafiky na stránce. Každá z těchto technologií má své uplatnění. Pro některé typy webových projektů byla přímo vyvinutá, na některé ji lze použít s určitými překážkami a u některých projektů je její využití přímo nevhodné.

Obrázek 2.1: Testovací graf.

V následujících kapitolách jednotlivé technologie popíši. Pokusím se ve zkratce shrnout potřebu jejich vzniku, jejich přednosti a jejich slabé stránky. Zároveň s každou technologií vykreslím jednoduchý graf o dvou uzlech propojených hranou a popíši výhody a nedostatky toho řešení. Na konci této kapitoly odůvodním svou volbu pro výběr vykreslovací technologie.

2.5.1 HyperText Markup Language (HTML) a Kaskádové styly (CSS)

HTML (a později CSS) vzniklo s potřebou modifikovat vzhled textové webové stránky. Technologie se stále vyvíjí a dnes tak již nabízí nepřeberné množství možností, jak stylovat nejen text, ale také různé geometrické tvary, ovládací prvky a podobně. HTML tagy slouží jak k rozdělení webové stránky na (sémanticky) logické části, tak i k samotnému stylování textu a tvorbě grafiky. Kaskádové styly poté na množiny těchto HTML tagů aplikují nejrůznější transformace, animace a podobně.

```
<style>
.node { position: absolute; width: 34px;
    height: 34px; line-height: 34px;
    border-radius: 50%; background: #2d89ef;
    color: white; text-align: center;
    border: solid 3px white; z-index: 1; }

.edge { position: absolute; background: #999;
    width: 140px; height: 6px; top: 17px; left: 20px; }

.node:last-child { left: 140px; }
</style>
<div class="node">A</div>
<div class="edge"></div>
<div class="node">B</div>
```

Listing 3: Vykreslení grafu (2.1) pomocí HTML elementů.

Mezi největší přednosti technologie patří opravdu velké možnosti při stylování webové stránky. S několika řádky dobře napsaných kaskádových stylů lze dosáhnout opravdu úchvatných výsledků. Technologie podporuje mimo vzhled i interaktivitu s uživatelem, velkou škálu animací a lze s ní tvořit i trojrozměrný prostor.

HTML (a CSS) je ovšem pořád primárně vyvíjeno s důrazem na stylování textového obsahu stránky a různé neobdélníkové geometrické útvary (například křivky) se s touto technologií vytváří špatně. Při hodně velkém množství

tagů na stránce pak mají webové prohlížeče problém s rychlostí překreslování stránky.

Z ukázky (3) lze již na tomto jednoduchém příkladu zpozorovat, že technologii využíváme poněkud zvláštním způsobem. Např. kulatý uzel definujeme jako `div`, ze kterého uděláme teprve zaoblením rohů kruh. Stejně tak bychom pravděpodobně nepředpokládali, že kvůli vycentrování textu na střed budeme muset nastavit styl `line-height` na výšku elementu.

2.5.2 Canvas

Canvas (v překladu plátno) je HTML tag, na který se kreslí pomocí skriptovacího jazyka – typicky JavaScriptu. Oproti ostatním webovým technologiím si nedrží žádnou vnitřní strukturu již nakreslených objektů. Dokonce ani neumí pracovat s více vrstvami, jako je to běžné u grafických editorů.

Z podstaty fungování canvasu lze s touto technologií vykreslovat libovolné tvary, protože výslednou grafiku můžeme upravovat až na úrovni jednotlivých pixelů. Canvas dále podporuje režim WebGL[10] (který má ovšem jiné JavaScriptové API), kdy je veškeré vykreslování prováděno přímo na grafické kartě a lze tak i ve webovém prohlížeči vytvořit komplexní 3D herní engine.

Pro některé typy projektů tkví hlavní nevýhoda v neexistenci vnitřní struktury vykreslených objektů. O vše se musí vývojář postarat sám v paměti prohlížeče. Z tohoto faktu vyplývá i další handikep. Na vykreslené objekty nelze nijak aplikovat události (najetí myši, kliknutí a podobně) a animace. O vše se opět musí postarat vývojář anebo vhodná knihovna pro vykreslování na canvas.

V ukázce (4) je vidět použití Canvasu pro vykreslení grafu (2.1). Nutno ovšem podotknout, že nad tímto nativním API existuje mnoho knihoven, které tuto činnost zjednodušují. Často pak nabízejí i nějaké mechanismy pro interakci s uživatelem, animace a podobně.

2.5.3 SVG (škálovatelná vektorová grafika)

Mezi nejpopulárnější vektorové formáty dneška patři i SVG. Jedná se o textový formát, který se skládá z XML tagů reprezentující jednotlivé objekty a křivky výsledné grafiky. Tento formát podporují i webové prohlížeče, a navíc k němu přidávají podporu pro uživatelské události.

Celá SVG grafika je tak součástí HTML DOMu[11] webové stránky, což přináší řadu výhod. K jednotlivým elementům lze přistupovat individuálně bez potřeby držet další meta informace někde mimo DOM. Oproti canvasu je zde již z podstaty formátu zařízeno automatické škálování na zařízeních

[10] Web Graphics Library

[11] Document Object Model je objektově orientovaná reprezentace HTML dokumentu, ke které lze přistpovat pomocí JavaScriptu.

```
<canvas id="canvas"></canvas>
<script>
CanvasRenderingContext2D.prototype.drawNode = function(x, y, text) {
    this.fillStyle = 'white';
    this.beginPath();
    this.arc(x, y, 23, 0, 2 * Math.PI);
    this.fill();

    this.fillStyle = '#2d89ef';
    this.beginPath();
    this.arc(x, y, 20, 0, 2 * Math.PI);
    this.fill();

    this.font = "16px Consolas";
    this.fillStyle = 'white';
    this.fillText(text, x - 4, y + 4);
};

var ctx = document.getElementById('canvas').getContext('2d');

ctx.fillStyle = '#999';
ctx.fillRect(20, 17, 140, 6);
ctx.stroke();

ctx.drawNode(20.5, 20.5, 'A');
ctx.drawNode(140.5, 20.5, 'B');
</script>
```

Listing 4: Vykreslení grafu (2.1) na plátno elementu Canvas.

s různou hustotou DPI[12]. Oproti HTML je navíc SVG přímo navrženo jako formát pro grafiku, a tak je s ním tvorba nejrůznějších obrazců velmi snadná. Navíc lze SVG efektivně tvořit pomocí externího vektorového editoru, jako je např. Illustrator nebo Inkscape.

Na SVG elementy platí při stylování jiná pravidla než na HTML tagy. Ne vše lze nastavit přes kaskádové styly. Dále se do budoucna nepočítá s podporou animací (rozšíření SMIL)[13], které je tedy potřeba prozatím programovat pomocí JavaScriptu.

[12] Dots per inch - tedy kolik pixelů se vejde do délky jednoho palce (2,54 cm).

[13] SMIL (Synchronized Multimedia Integration Language) je rozšíření SVG formátu podporující komplexní animace. Ve webových prohlížečích ho má nahradit připravované Web Animations API.

```
<style>
circle { fill: #2d89ef; stroke: white; stroke-width: 3; }
text { fill: white; }
rect { fill: #999; }
</style>
<svg>
    <rect class="edge" x="21.5" y="18.5" width="140" height="6" />
    <g transform="translate(21.5, 21.5)">
        <circle r="20" />
        <text alignment-baseline="middle" text-anchor="middle">A</text>
    </g>
    <g transform="translate(161.5, 21.5)">
        <circle r="20" />
        <text alignment-baseline="middle" text-anchor="middle">B</text>
    </g>
</svg>
```

Listing 5: Vykreslení grafu (2.1) pomocí SVG elementů.

Při přípravě ukázky (5) jsem narazil u pozicování textu uvnitř uzlu. Celý kruh (`circle`) je potřeba obalit elementem `g`, takže nelze využít atributů `x` a `y` pro udání pozice elementu `circle`. Samotné atributy `alignment-baseline` a `text-anchor`, které by měly zarovnat text na střed elementu `g`, pak stejně nefungovaly ve všech prohlížečích.

2.5.4 Volba technologie pro vykreslování

Při výběru technologie pro vykreslování jsem nejdříve zavrhl využití kreslící plochy canvasu. Musel bych si pozice, styl a texty všech vrcholů, hran a dalších prvků grafu držet ve struktuře v kódu JavaScriptu. Při kliknutí či pohybu myši by vznikla nutnost manuálně detekovat, s jakým prvkem myš zrovna koliduje. Stejně tak mě od Canvas API odradila nutnost řešit všechny animace ručně. Navíc se jedná o technologii, která pracuje s bitmapou, a tak je jakékoliv škálování či podpora vyšších DPI jen další komplikace při vývoji. Velká výhoda Canvasu pro mou aplikaci by byla pouze ve vizualizaci opravdu velkých grafů. Jak jsem již ale avizoval v sekci zaměření aplikace, není to ambice tohoto projektu.

Zvažovanou technologií bylo využití HTML tagů a kaskádových stylů. Netrpí problémy, jako má canvas popsané v předchozím odstavci, a oproti SVG disponuje lepší podporou animací a různých další grafických efektů. Navíc umí oproti SVG lépe pracovat s textem.

Zvolil jsem nakonec formát SVG. Nejlépe a nejsnáze vykresluje grafické

obrazce a křivky. Zatím nedokonalou podporu animací lze řešit pomocí JavaScriptu. Silnou motivací pro právě tuto volbu pak pro mě byla rešerše ostatních knihoven pro vykreslování grafů. Velká část z nich využívá právě SVG. Další výhodou je možnost exportu výsledné grafiky do svg obrázku (tedy v podstatě jen uložení aktuálního DOMu) do souboru.

2.6 Skriptovací jazyky

Webová aplikace podobného typu se neobejde bez skriptovacího jazyka. V následujících podkapitolách postupně zhodnotím dnešní možnosti webového vývojáře. Tedy jediný nativní webový skriptovací jazyk – JavaScript a jeho nástavby. Poté odůvodním svou volbu a pokusím se o stručnou úvahu nad aktuální situací.

2.6.1 JavaScript

JavaScript je dynamický, netypový, objektově orientovaný programovací jazyk. Jedná se o jediný skriptovací jazyk, který webové prohlížeče nativně podporují. Všechny webové API, funkcionality a podobně jsou tedy navrhovány s ohledem na možnosti tohoto jazyka.

```
1  function Person(name, birthDate) {
2      this.Name = name;
3      this.BirthDate = birthDate;
4  }
5
6  Person.prototype.SaySomething = function(message) {
7      console.log(this.Name + ' sayd: ' + message);
8  };
9
10 Person.prototype.GetAge = function() {
11     var today = new Date();
12     return today.getFullYear() - this.BirthDate.getFullYear();
13 };
14
15 var jan = new Person('Jan Novák', new Date(1993, 5, 21));
16 jan.SaySomething('What a lovely day!');
17 console.log('Age: ' + jan.GetAge());
```

Listing 6: Ukázka syntaxe jazyka JavaScript.

Počátky jazyka sahají až do roku 1995, kdy jeho základní kostru navrhl Brendan Eich ze společnosti Netscape během pouhých 10 dní. To se negativně

podepsalo na finální specifikaci jazyka, a tak vznikla celá řada nástaveb, jejichž cílem je tyto neduhy minimalizovat.

S rostoucí popularitou internetu se stal JavaScript nejpopulárnějším programovacím jazykem na světě[14]. Jeho vývoj je tak stále velmi aktivní, a především v posledních třech letech doznal velkého množství změn.

Příkládám i ukázku kódu (6). JavaScript je jediný populární programovací jazyk, kde se nové objekty tvoří pomocí prototypování již existujících objektů. Každý objekt nebo vlastnost objektu může kromě speciální hodnoty `null` nabývat ještě hodnoty `undefined`, což pak vede k častému zdroji chyb. Tento problém částečně řeší typová nástavba TypeSript...

2.6.2 TypeScript

TypeScript je navržen jako nástavba JavaScriptu, která jazyk obohacuje o datové typy, generické funkce, třídy, moduly a další drobnosti, které ve starších verzích JavaScriptu citelně chybí.

```
1  class Person {
2      constructor(public Name: string, public BirthDate: Date) { }
3
4      public SaySomething(message: string) {
5          console.log(this.Name + ' sayd: ' + message);
6      }
7
8      public GetAge(): number {
9          let today = new Date();
10         return today.getFullYear() - this.BirthDate.getFullYear();
11     }
12 }
13
14 let jan = new Person('Jan Novák', new Date(1993, 5, 21));
15 jan.SaySomething('What a lovely day!');
16 console.log('Age: ' + jan.GetAge());
```

Listing 7: Ukázka syntaxe jazyka TypeScript.

Syntaxe TypeScriptu je pečlivě volena tak, aby se co možná nejvíce blížila připravovaným specifikacím JavaScriptu. Je tak pravděpodobné, že funkcionality, které se dnes dají využít v TypeScriptu, půjde časem snadno migrovat do nativního JavaScriptu. Další nespornou předností TypeScriptu je, že kód převedený do JavaScriptu je stále velmi dobře čitelný a lze tak hybridně pracovat s oběma jazyky najednou.

[14] Dle anktety na portálu StackOverflow[3].

Jazyk vyvíjí ve spolupráci s komunitou společnost Microsoft a těší se nebývalé popularitě[3]. Podpora pro syntaxi jazyka nechybí v žádném významnějším vývojovém editoru či IDE.

Z ukázky TypeScriptu (7) je patrná především změna v definici třídy a anotace všech proměnných datovým typem. Např. do metody `SaySomethink` už se teď nemá šanci dostat zpráva (`message`) s hodnotou `null` nebo `undefined` a je tak zajištěná její správná funkčnost[15].

2.6.3 CoffeeScript

Velmi zajímavá nástavba JavaScriptu – CoffeeScript, přichází s naprosto odlišnou syntaxí. Autoři jazyka se inspirovali minimalistickým funkcionálním zápisem z Ruby, Pythonu a Haskellu, který naroubovali s ohledem na potřeby webových vývojářů. Skripty napsané v CoffeeScriptu jsou tak velmi úsporné a přitom neztrácejí na přehlednosti. Podobně jako u TypeScriptu pak kompilátor CoffeeScriptu generuje poměrně čitelný kód, se kterým je možné dále pracovat.

```
1  class Person
2      constructor: (@Name, @BirthDate) ->
3      SaySomething: (message) ->
4          console.log("#{ths.name} sayd: #{message}")
5          return
6      GetAge: () ->
7          today = new Date()
8          today.getFullYear() - this.BirthDate.getFullYear()
9
10 jan = new Person('Jan Novák', new Date(1993, 5, 21))
11 jan.SaySomething('What a lovely day!');
12 console.log("Age: #{jan.GetAge()}");
```

Listing 8: Ukázka syntaxe jazyka CoffeeScript.

V době psaní této práce vyšla druhá verze CoffeeScriptu reflektující nové možnosti JavaScriptu. Jazyk se stále zaměřuje pouze na novou syntaxi. Datové typy a některé pokročilé konstrukce zde nenajdeme. S jazykem se tak nicméně stále počítá a mnoho ukázek právě z oblasti zpracování velkého množství dat je napsaných v CoffeeScriptu.

Z ukázky (8) je hned patrný minimalistický zápis všech jazykových konstrukcí. Podobně jako v případě TypeScriptu se zde definují třídy.

[15] Pokud bychom zde chtěli povolit např. i `null` hodnotu, musela by být anotace datového typu ve tvaru `string | null`.

2.6.4 Volba skriptovacího jazyka

Jediný jazyk, který webové prohlížeče nativně podporují, je JavaScript. Nemusí se nijak kompilovat, je v něm napsáno nejvíce projektů a každý webový vývojář ho zná. Takto bych shrnul hlavní přednosti JavaScriptu. Je to ovšem také jazyk, který si s sebou nese velkou sbírku špatných rozhodnutí při jeho návrhu, které kvůli zachování zpětné kompatibility asi jen tak nezmizí. Je to také jediný prototypově orientovaný jazyk, který se ve větší míře stále využívá. Mnoho vývojářů má tak problém zvyknout si na jeho konvence. Nástavby JavaScriptu tyto problémy elegantně řeší a navíc přidávají prvky, na které je majorita vývojářů zvyklá. Proto jsem se rozhodl využít některou z těchto alternativ.

Díky popularitě internetu a faktu, že je JavaScript jediný jazyk, kterému internetové prohlížeče rozumí, vzniklo velké množství překladačů nejrůznějších jazyků právě do JavaScriptu. Například i staré MS DOS hry naprogramované v C tak dnes lze zkompilovat do JavaScriptu. Chtěl jsem se ovšem vydat nativnější cestou, plně využít debuggování kódu v prohlížeči, knihovny pro práci s DOMem a podobně. Proto jsem svou volbu zredukoval na rozhodnutí mezi jazyky TypeScript a CoffeeScript.

Mou volbou pro tuto práci se nakonec stal TypeScript. Podporuje datové typy, díky kterým lze velkou část chyb odhalit již na úrovni kompilace. Navíc mohu díky tomu naplno využít možnosti automatického našeptávání kódu. TypeScript navíc podporuje mnoho prvků, na které jsou zvyklý vývojáři z jazyků jako je C++, C# nebo Java. Jednou z motivací pro tuto volbu je pak i silný hráč na poli IT – Microsoft, který za tímto jazykem stojí.

Na CoffeeScriptu oceňuji jeho povedenou syntaxi, která se na projekt tohoto typu hodí pravděpodobně ještě více. Nepodporuje bohužel anotaci datových typů a mám trochu obavy o jeho možnostech při reflektování budoucích změn v JavaScriptu.

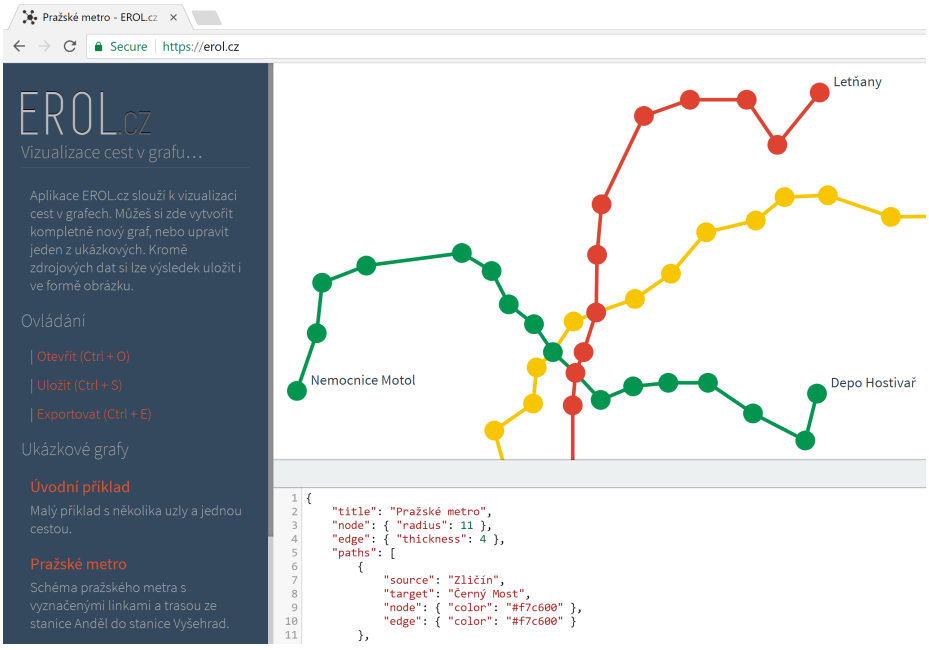
2.6.5 Proč stále JavaScript?

Dovolím si ještě do této práce uvést již avizovanou malou vsuvku o aktuální situaci na poli webových prohlížečů. JavaScript je dnes jediný skriptovací jazyk, který webové prohlížeče nativně podporují, a je to škoda podle mého názoru, názorů vývojářů nejpoužívanějšího prohlížeče Google Chrome i mnoha dalších významných osobností průmyslu. O jeho nedostatcích ve specifikaci a jeho komplikované optimalizaci byla napsána celá řada prací. Přesto se bohužel ani za více jak 15 let nepovedlo prosadit alternativu. Z JavaScriptu se tak spíše pomalu stává jakýsi byte-code webových prohlížečů, do kterého své zdrojové kódy kompilují jiné, lépe navržené jazyky.

Kapitola 3

Realizace

V následujících sekcích popisuji postup, jak jsem celou aplikaci vyvinul. Výsledek lze vidět na obrázku 3.1.



Obrázek 3.1: Ukázka aplikace erol.cz.

3.1 Vývojové prostředí

Z analýzy a návrhu mého projektu vzešlo několik požadavků, které byly klíčové pro volbu vývojového prostředí. Předně jsem se nechtěl vázat na konkrétní operační systém. Sám průběžně využívám Windows i Linux a nějakou dobu jsem nakonec vyvíjel i u přítelkyně pod OS X. Dále jsem chtěl maximálně

využít přednosti zvoleného jazyka TypeScript. Tedy inteligentní našeptávání, podporu komentářů, ladění přímo v editoru a podobně. V neposlední řadě pak rozhodovala obecná podpora webových technologií a popularita nástrojů ve vývojářské komunitě. Osobně mohu doporučit nástroje Sublime Text, Atom a Visual Studio Code.

Sublime Text exceluje především v rychlosti a využíval jsem ho tak pro práci s regulárními výrazy, kdy jsem potřeboval editovat velké JSON soubory s daty pro graf nebo zdrojový kód SVG obrázků. Atom i Visual Studio Code, nástroje implementované v JavaScriptu, měly s těmito soubory problém.

Atom je skvělá volba při stylování stránky. Má velmi povedené GUI (které přispívá k designerské tvůrčí činnosti) a skvělý našeptávač kaskádových stylů i HTML tagů.

Většinu času jsem však vyvíjel v editoru **Visual Studio Code**, které má dle mého úsudku nejlepší podporu jazyka TypeScript. Především ladění (debugging) je zde opravdu povedené. Jedná se o poměrně nový nástroj od společnosti Microsoft s velmi aktivním vyvojem. Podobně jako Atom je navíc open-source.

Celý projekt jsem verzoval pomocí systému **GIT**. Jedná se dnes o nejpopulárnější verzovací systém a neměl jsem žádný důvod volit alternativu typu SVN nebo Mercurial.

Adresářovou strukturu jsem volil co možná nejtypičtější pro projekt mého typu. Tedy jediná stránka index.html a podsložky js pro skripty napsané v javascriptu, css pro kaskádové styly, img pro obrázky, fonts pro vlastní fonty a data s definicemi ukázkových grafů. Složka ts poté obsahuje všechny skripty psané v TypeScriptu a podsložka typings typescriptové definice ostatních knihoven.

Všechny skripty ze složky ts se kompilují do JavaScriptu do jediného souboru app.js umístěného ve složce js. K tomuto souboru se zároveň generuje korespondující mapovací soubor původního TypeScriptu, takže i při ladění v prohlížeči lze procházet originální typový kód. Kompletní strukturu projektu lze najít v příloze B.

Kompilátor TypeScriptu je spouštěn s nejstriktnějšími parametry mající za cíl odhalit potencionální chyby v konverzi datových typů. Celá konfigurace je uložena v souboru tsconfig.json a kompilátor lze spustit buď přímo z editoru (zkratka **Ctrl + Shift + B**) nebo externě pomocí příkazu **tsc**[16] v kořenové složce projektu.

3.2 Použité knihovny

Cílem mé práce byla především vizualizace cest v grafu. Z návrhu ovšem vyplynulo, že bude potřeba zadávat JSON vstup přímo v okně prohlížeče. Samotný prohlížeč nic pokročilejšího než element `textarea` nepodporuje. Pro

[16] tsc je kompilátor TypeScriptu distribuovaný jako npm balíček.

práci s JSON soubory jsem si tak vybral knihovnu **CodeMirror**. Při inicializaci knihovny stačí určit HTML element, který bude sloužit k editování textu a programovací jazyk, jehož syntaxe se má obarvovat a o zbytek se knihovna postará. Žádná alternativa k této knihovně neexistuje a je k ní navíc spousta dodatečných pluginů.

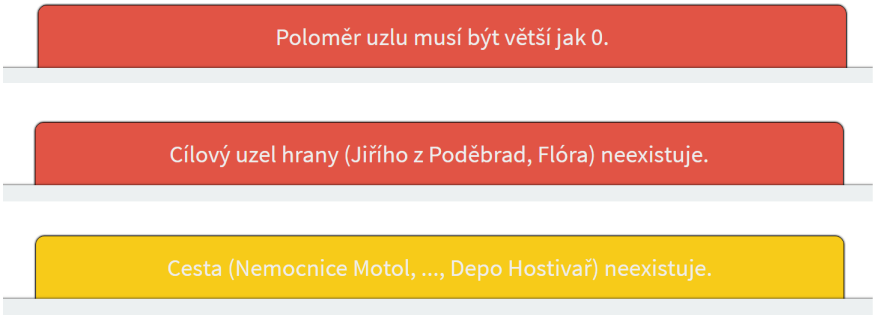
Dále jsem si ulehčil práci s knihovnou **d3.js**, která slouží primárně k manipulaci s DOMem a pohodlnější práci s daty. Tyto činnosti lze sice poměrně snadno řešit i pomocí nativních funkcí HTML specifikace, s knihovnou d3.js je ovšem výsledný kód daleko kratší, přehlednější, konzistentnější a pravděpodobně i rychlejší. d3.js jsem si vybral i z důvodu obrovského množství ukázek okolo vizualizace grafů, které byly napsané právě za pomocí této knihovny. Ne zcela plnohodnotnou alternativou by mohla být také knihovna jQuery, která patří vůbec k těm nejpopulárnějším JavaScriptovým knihovnám.

3.3 Funkční části

V následujících podsekcích popíši nejzajímavější části aplikace a jak probíhal jejich vývoj.

3.3.1 Zpracování vstupu

Vizualizaci grafu aktualizazuji při každé změně ve vstupním JSONu. Pro uživatele je tato funkcionalita přínosná, neboť neustále vidí aktuální verzi grafu. Nese s sebeou však i vyšší nároky na kontrolu vstupu. Je totiž potřeba odlišit, kdy může k překreslení vizualizace dojít a kdy je ještě potřeba počkat, než uživatel upraví vstup do validní podoby. Ukázku varovných hlášení aplikace je možné vidět na obrázku 3.2.



Obrázek 3.2: Různé typy chybových hlášek aplikace erol.cz.

V prvním kroku této validace se pokusím převést vstupní řetezec na data. K tomu mi dopomůže nativní funkce `JSON.parse`, která při špatném formátu vstupu zároveň zahlásí, kde je přesně chyba. V tom případě pouze deleguji chybovou zprávu na uživatele a nepokračuji v pokročilejších validacích.

Nyní je třeba zkontrolovat, zda jsou všechny vlastnosti uzlů a hran zadány správně, aby v pozdější fázi vizualizace nedošlo k chybě. Je tedy třeba kontrola:

- Povinné hodnoty uzlů, hran a cest.
- Zda má každá hrana odpovídající definici zdrojového i cílového uzlu.
- Datové typy vlastností uzlů a hran. Popřípadě správné rozsahy čísel, formát barvy, atd.

V případě správnosti těchto údajů (v opačném případě se uživateli zobrazí chybová zpráva) je třeba doplnit definice uzlů a hran z obecných definic (uzlů/hran) a definic cest. Podle avizované priority v návrhu aplikace skript nejprve projde všechny vlastnosti v obecné definici uzlů a hran a pokud tato vlastnost u odpovídající hrany/uzlu chybí, doplní ji.

V dalším kroku se prochází jednotlivé cesty a jejich vlastnosti se aplikují na uzly a hrany grafu (tentokrát se ovšem přepisují i původní vlastnosti, protože cesta má nejvyšší prioritu). Tuto činnost vykonává metoda `GraphUtils.ApplyPathToGraph`, kterou popisuji ve zvláštní sekci 3.3.3.

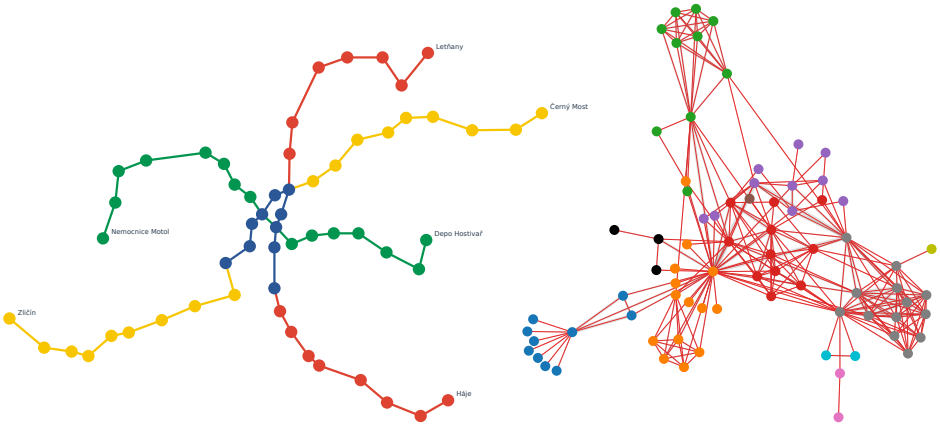
Nyní je jistota, že mají uzly a hrany všechny vlastnosti nastavené a jsou validní. Před předáním dat k vizualizaci je ještě třeba určit layout, ve kterém se graf vizualizuje. Naimplementoval jsem zde pouze dva. Pokud mají všechny uzly souřadnice, vykreslí se graf podle nich. Pokud však nejméně jedna souřadnice u některého z uhlů chybí, zvolí se force-directed layout.

Pokud se všechny tyto kroky úspěšně vykonají, je objekt s grafem a zvolený layout předán třídě **Renderer**.

3.3.2 Vizualizace grafu

Pro samotnou vizualizaci grafu využívám SVG formát a jeho elementy `circle` (kruh) pro vykreslení uzlů a `line` pro vykreslení hran. Celý postup probíhá v následujícím pořadí

1. Vytvoření SVG elementů `circle` a `line` a jejich přidání do plátna `svg` elementu.
2. Nastavení grafických vlastností uzlů a hran SVG elementům (barva, velikost, ...).
3. Zaregistrování událostí pro Drag&Drop uzlů.
4. Nastavení pozice uzlů a hran.
    a) Pokud nejsou pozice předem známy, spustit simulaci force-directed layoutu a přepočítávat pozice uzlů každou iteraci simulace.



(a) Plán pražského metra.

(b) Graf společných pasáží postav v románu Bídnící.

Obrázek 3.3: Ukázky vizualizací vytvořené v aplikaci erol.cz. Oba obrázky byly vyexportovány do formátu SVG. Jednotlivé uzly na obrázku 3.3b jsou pozicovány force-directed simulací.

O celou tuto sekvenci se stará třida **Renderer** ze skriptu **renderer.ts**. Mimo to řeší přibližování/oddalování grafu a pohyb s myší po plátně.

Využívám zde v co možná nejvyšší míře knihovnu d3.js. Jedna z jejích funkcí totiž je, že lze k datům (v mém případě uzlům a hranám) spárovat DOM elementy a pracovat pak s oběma objekty zároveň. Jeden z modulů d3.js je také určený právě pro silově řízené simulace. Této již naimplementované simulace využívám ve svém force-directed layoutu.

Jedná se o poměrně nový algoritmus (2009[4]) od Tima Dwyera, kde jedna iterace simulace trvá pouze $\mathcal{O}(n \log n + m + c)$, kde n je počet uzlů, m je počet hran a c jsou dodatečná omezení simulace (zarovnání, vymezený prostor, atd.). Při implementaci této funkcionality jsem vycházel z práce [5] Mike Bostocka. Ukázku grafu pozicovaného touto technikou přikládám v obrázku 3.3b.

3.3.3 Rekonstrukce cesty

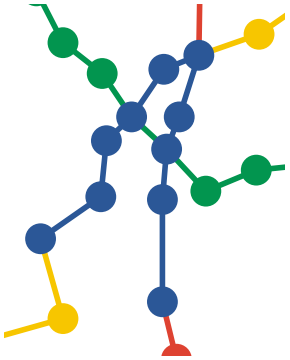
Cestu jsem ve specifikaci JSON formátu definoval jako nejkratší posloupnost uzlů a hran mezi počátečním a koncovým uzlem. Mohou zde však být specifikovány i další uzly, kterými musí cesta (při dodržení pořadí[17]) procházet. Ukázka takovéto vizualizace je na obrázku 3.5.

Nejprve jsem si tak vytvořil posloupnost uzlů $P = (u_1, u_2, \ldots, u_{n-1}, u_n)$, kde u_1 je počáteční uzel (`source`), u_n koncový (`target`) a uzly u_2 až u_{n-1} jsou definovaný v seznamu `through`.

[17] Pokud bychom hledali nejkratší cestu z uzlu u do v s cílem projít všechny uzly z nějaké podmnožiny grafu nehledě na pořadí, jedná se o variantu problému obchodního cestujícího, který má složitost $\mathcal{O}(n^2 2^n)$, kde n je počet uzlů.

```
{
    "source": "Anděl",
    "target": "Vyšehrad",
    "through": "Florenc",
    "edge": { "color": "#2b5797" },
    "node": { "color": "#2b5797" }
}
```

Listing 9: Cesta definovaná v JSON formátu



Obrázek 3.4: Vizualizovaná cesta v grafu.

Obrázek 3.5: Ukázka definice cesty v JSON formátu (9) a její vizualizace v grafu (3.4).

Dále jsem si vytvořil funkci $GetPath(u, v)$, která mi pro zadanou dvojici uzlů vrátila nejkratší cestu mezi nimi ve formě posloupnosti uzlů. Postupným voláním $GetPath(u_1, u_2)$ až $GetPath(u_{n-1}, u_n)$ jsem tak získal kompletní posloupnost uzlů z u_1 do u_n, ke kerým už lze snadno přiřadit incidující hrany.

Funkce $GetPath(u_1, u_2)$ je klasické prohledávání grafu do šířky se složitostí $\mathcal{O}(|V| + |E|)$. To ovšem musím volat $(|P| - 1)$-krát, takže celková složitost rekonstrukce jedné cesty je až $\mathcal{O}(|P|(|V| + |E|))$.

3.3.4 Import/Export grafu

Jeden ze základních pilířů každé aplikace pro práci s grafy je možnost importu a exportu dat. Webové aplikace nabízejí poměrně krkolomné postupy, jak tyto požadavky naplnit. V dalších odstavcích popíši, jak jsem se s tímto faktem vypořádal.

Pro vygenerování a uložení souboru z izolovaného prostředí webové stránky na disk uživatele zatím neexistuje žádné oficiální HTML API, které by podporovali všechny majoritní prohlížeče[6]. Využívá se proto HTML elementu `a` (odkaz), kterému nastavíme atribut `download` na název souboru, který chceme uložit a atributu `href` nastavíme hodnotu `data:text/plain;charset=utf-8,[text]`, kde `[text]` je obsah souboru zakódovaný tzv. URL kódováním. Využíváme zde podpory datových URI, kdy lze obsah cíle odkazu zakódovat přímo do odkazu. Na takto vytvořený odkaz poté stačí vyvolat událost kliknutí. Celou funkci příkládám v ukázce 10.

Při ukládání grafu ve formátu SVG využívám výhody, že celý graf už pomocí SVG vykresluji. Provedu tak pouze nutnou substituci hlavičky SVG obrázku a poté opět využiji funkci pro uložení souboru (10).

Pro načtení JSONu do aplikace také neexistuje zcela triviální řešení. Celý

```
function download(filename: string, text: string) {
    let el = document.createElement('a');
    el.setAttribute('href', 'data:text/plain;charset=utf-8,'
        + encodeURIComponent(text));
    el.setAttribute('download', filename);
    el.style.display = 'none';

    document.body.appendChild(el);
    el.click();
    document.body.removeChild(el);
}
```

Listing 10: Vygenerování a uložení souboru v TypeScriptu.

postup lze nalézt v metodě `Main.Open`. V bodech je však potřeba udělat následující.

1. Vytvoření elementu `input` s atributy `type="file"` a `accept=".json"`. Druhý jmenovaný nám zajistí, že uživatel v dialogovém okně uvidí pouze soubory s příponou `.json`.

2. Zaregistrování události `onchange` na element `input`, která se vyvolá při potvrzení dialogového okna s výberem souboru uživatelem.

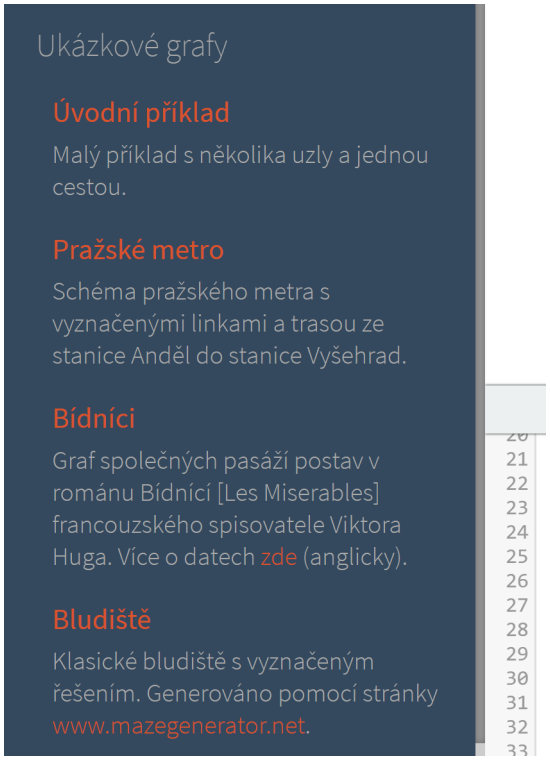
3. Vyvolání události kliknutí na `input` (otevře se modální okno).

4. Po potvrzení modálního okna uživatelem, pomocí HTML API FileReader načíst obsah souboru do paměti a následně do editoru aplikace.

Aplikce podporuje také klávesové zkratky těchto akcí. **Ctrl + O** pro otevření JSON souboru, **Ctrl + S** pro jeho uložení a **Ctrl + E** pro export grafu do SVG formátu.

3.4 Ukázkové grafy

Jednou z předností vyvíjené aplikace má být co nejsnazší použití. Proto obsahuje sekci s ukázkovými grafy, které lze načíst pomocí jednoho kliknutí. Ukázkové grafy jsou k dispozici v bočním panelu aplikace (obrázek 3.6) i s krátkým popiskem, o jaká data se jedná.



Obrázek 3.6: Sekce s ukázkovými grafy.

3.4.1 Úvodní příklad

Jedná se o velmi jednoduchý graf se šesti uzly, který se zobrazí jako první při otevření aplikace. Demonstruje použití jednoduché cesty, popisky uzlů a automatické pozicování force-directed simulací.

3.4.2 Pražské metro

Schéma pražského metra zmiňuji v několika částech této práce a zahrnul jsem ho i do ukázek grafů implementované aplikace. Skvěle podle mého názoru vyjadřuje potřebu vizualizace cest i v malých grafech. Navíc se jedná o schéma, které zná prakticky každý Čech. V tomto grafu definuji čtyři cesty. Linky metra A, B, C a cestu metrem ze stanice Anděl do stanice Vyšehrad. V tomto případě je nutné definovat i přestupní stanici Florenc.

3.4.3 Bídníci

Bídníci [Les Miserables] je román francouzského spisovatele Viktora Huga. Je rozdělen do mnoha, zdánlivě nesouvisejících, kapitol, kde jedinným dějovým pojítkem je hlavní postava Jean Valjeana. Uzly grafu znázorňují jednotlivé postavy příběhu, hrany pak jejich společné pasáže v jednotlivých kapitolách[7].

Tlouška hrany určuje počet kapitol, ve kterých spolu tyto dvě postavy vystupovali. Graf je využíván v mnoha dalších publikacích zabývajících se teorii grafů a poskytuje velmi přehlednou vizualizaci společných pasáží postav jinak 1200 stran dlouhého románu.

3.4.4 Bludiště

Příklad **Bludiště** slouží především k demonstraci algoritmu hledání nejkratší cesty. Jedná se o graf s více jak 3000 uzly a 3000 hranami s jedinou cestou k cíli. Graf byl vygenerovaný pomocí služby `http://www.mazegenerator.net` jako PNG obrázek. Následně jsem napsal skript v jazyce C#, který tento obrázek převedl do podoby grafu. Skript je možné nalézt na přiloženém CD.

3.5 Nasazení

Aplikaci jsem chtěl mít dostupnou online na krátké, zapamatovatelné adrese `https://erol.cz`. Doménu jsem koupil u registrátora ONEbit za 150 Kč, neboť jsem s ním byl v minulosti spokojený.

Samotnou aplikaci hostuji na vlastním serveru, kde je spuštěný Apache HTTP Server a u společnosti ONEbit mám pouze **A** záznamy `erol.cz` a `*.erol.cz` odkazující na IP adresu vlastního serveru.

```
1  <VirtualHost *:80>
2      ServerName erol.cz
3      ServerAlias *.erol.cz
4      DocumentRoot /var/www/erol/
5      RewriteEngine On
6      RewriteCond %{HTTP:CF-Visitor} '"scheme":"http"'
7      RewriteRule ^(.*)$ https://erol.cz$1 [L]
8  </VirtualHost>
9  <VirtualHost *:443>
10     ServerName erol.cz
11     ServerAlias *.erol.cz
12     DocumentRoot /var/www/erol/
13 </VirtualHost>
```

Listing 11: Konfigurace Apache HTTP serveru pro aplikaci erol.cz.

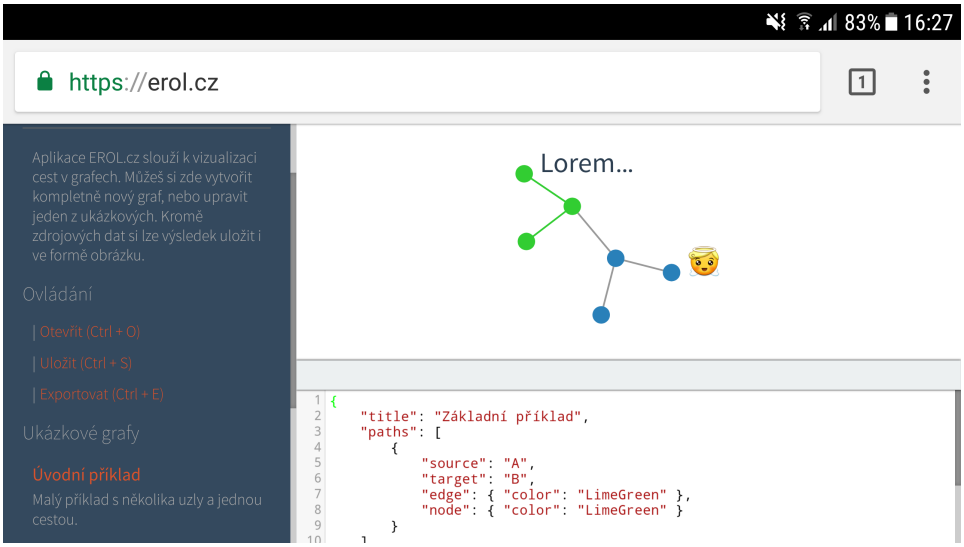
Konfigurace serveru (11) je v mém případě poměrně snadná, protože se celá aplikace skládá pouze ze statického obsahu. Za zmínku stojí akorát automatické přesměrování `http` na `https` verzi stránky pomocí modulu `mod_rewrite`.

O automatické vystavení SSL certifikátu se stará společnost CloudFlare, na jejíž name servery doména `erol.cz` ukazuje. Tento prostředník v komu-

nikaci uživatele se serverem má spostu dalších výhod, jako je ochrana proti DDos útokům, offline verze stránky[18] v případě výpadku serveru, statistiky návštěvnosti a podobně.

3.6 Testování

Jednou z nedílných součástí vývoje aplikace je také testování. V případě této aplikace - menší projekt, vyvíjený pouze jednou osobou, jsem postupoval následovně.



Obrázek 3.7: Aplikace erol.cz testovaná na mobilním zařízení.

V ráné fázi vývoje jsem vytvořil čtyři ukázkové grafy podle specifikace 2.1. Při vývoji jsem poté průběžně kontroloval chování těchto vizualizovaných grafů a opravoval případné chyby. Kompletní testování jsem provedl až při finalizaci aplikace a zahrnovalo následující.

3.6.1 Testování vizualizace cest v grafu

Primárním cílem implementované aplikace byla vizualizace cest v grafu. Proto jsem tento test provedl v mnoha kombinacích na všech ukázkových grafech a všech podporovaných prohlížečích (viz sekce 2.2 Platforma). Nezaznamenal jsem zde žádný nedostatek.

[18] V mém případě se jedná o plnohodnotnou stránku, protože ze serveru žádná data nenačítám.

3.6.2 Testování uživatelského prostředí

Na uživatelské prostřední webových aplikací jsou typicky kladeny poměrně vysoké nároky. Mělo by být přehledné na obrazovkách nejrůznějších velikostí, konzistentní napříč webovými prohlížeči, podporovat dotykové displaye a podobně. S tímto faktem jsem aplikaci vyvíjel a následně testoval.

V prohlížečích Google Chrome, Firefox, Opera a Safari se vše chovalo tak, jak to bylo navrženo. U prohlížeče Edge bohužel nefunguje ukládání souborů metodou popsanou výše (10). Žádné snadné řešení tohoto problému jsem bohužel nenalezl. Uživatel má ale stále možnost uložit JSON zkopírováním obsahu editoru aplikace.

U mobilních prohlížečů Firefox a Google Chrome jsem větší problémy nezaznamenal. Ukázku aplikace na mobilním zařízení je možné vidět na obrázku 3.7.

3.7 Diskuze k realizaci

V rámci této bakalářské práce jsem nemohl konkurovat velkým systémům typu Gephi a Cytoscape. Zároveň jsem ale chtěl přijít s něčím novým, co lze naimplementovat i v jedné osobě. K rozhodnutí, pojmout téma jako implementaci webové aplikace, mě přivedlo hned několik důvodů.

U prací podobného zaměření, jejíž součástí byla i implementace, jsem měl často problém výslednou aplikaci vůbec najít, zkompilovat a spustit. Často jsem se tak musel spokojit s obrázky a textovým popisem. Webové technologie přitom pokročily a na menší, nenáročné aplikace, se stávají čím dál zajímavější volbou. Navíc zde není nutná instalace, uživatel má vždy aktuální verzi a to vše z jednoho krátkého odkazu.

Před implementací jsem zvažoval využít jednu z knihoven pro vizualizaci grafů (vis.js, cytoscape.js, sigmajs). K ruční implementaci za pomocí knihovny d3.js mě přesvědčili až desítky ukázek nejrůznějších grafových vizualizací právě s touto knihovnou. Navíc jsem měl nad výsledkem větší kontrolu.

V možnostech stylizace uzlů a hran grafu se mé řešení s vis.js nebo cytoscape.js nemůže srovnávat. Oproti jmenovaným knihovnám ale tato implementace poskytuje grafické uživatelské prostředí a možnost zadefinovat cestu mezi uzly.

Přímé srovnání tak mohu provést pouze s aplikací Pathfinder. Ta je zaměřená spíše na velké grafy s cílem zobrazit podgraf s nalezenými cestami. Oproti tomuto nástroji zobrazuji cestu vždy i s původním grafem (je zde ovšem pomocí parametru `opacity` možnost původní graf skrýt). Pathfinder má mnohem lepší uživatelské prostředí a dovoluje specifikovat pokročilejší možnosti při hledání cest. Aplikace erol.cz dovoluje navíc specifikovat pouze seznam uzlů, kterými musí cesta procházet.

Za největší chybu při návrhu aplikace považuji podporu pouze nově specifikovaného formátu. Při dalším vývoji aplikace je tuto skutečnost nutno před-

nostně napravit. Nelze totiž očekávat, že si začínající uživatelé budou psát skripty na převod jiných formátů na tento.

Závěr

V úvodu této práce jsem si vytyčil cíl popsat jednotlivé techniky vizualizace cest v grafech a navrhnout a vytvořit webovou aplikaci, která do této oblasti přinese něco nového.

Mnoho předchozích prací se zabývalo spíše vizualizací samotných grafů, popřípadě hledáním nejkratších cest v grafech. Zaměřil jsem se poto na samotné programy pro práci s grafy a popsal konkrétní techniky vizualizace cest, které se zde vyskytují.

Nejčastěji se zde aplikuje postup, kdy se nejprve naleznou nejkratší cesty mezi zvolenými uzly. Ty má uživatel dále možnost odfiltrovat od zbytku zobrazeného grafu a dále s nimi pracovat jako se samostatným grafem. Tento postup nejlépe demonstruje nástroj Pathfinder (1.3.4).

Na základě rešerše těchto nástrojů jsem se rozhodl implementovat aplikaci na vizualizaci cest v grafech s co možná nejsnazším použitím a s vlastním vstupním formátem, který přímo podporuje definici cest.

Implementace navržené aplikace proběhla úspěšně a je dostupná na adrese `https://erol.cz`. Je zde ovšem další prostor ke zlepšení především v podobě podpory více druhů stylování grafu, možnosti zobrazit další parametry vizualizované cesty a podpory více vstupních formátů.

Literatura

[1] Dwyer, T.; Lee, B.; Fisher, D.; aj.: A Comparison of User-Generated and Automatic Graph Layouts. *IEEE Transactions on Visualization and Computer Graphics*, ročník 15, č. 6, Nov 2009: s. 961–968, ISSN 1077-2626, doi:10.1109/TVCG.2009.109.

[2] Partl, C.; Gratzl, S.; Streit, M.; aj.: Pathfinder: Visual Analysis of Paths in Graphs. *Computer Graphics Forum*, ročník 35, č. 3, 2016: s. 71–80, ISSN 1467-8659, doi:10.1111/cgf.12883. Dostupné z: `http://dx.doi.org/10.1111/cgf.12883`

[3] Stack Exchange, Inc: Developer Survey Results 2017. Dostupné z: `https://stackoverflow.com/insights/survey/2017`

[4] Dwyer, T.: Scalable, Versatile and Simple Constrained Graph Layout. *Computer Graphics Forum*, ročník 28, č. 3, 2009: s. 991–998, ISSN 1467-8659, doi:10.1111/j.1467-8659.2009.01449.x. Dostupné z: `http://dx.doi.org/10.1111/j.1467-8659.2009.01449.x`

[5] Bostock, M.: Force-Directed Graph. Dostupné z: `https://bl.ocks.org/mbostock/4062045`

[6] Can I Use: Filesystem & FileWriter API. Dostupné z: `http://caniuse.com/#feat=filesystem`

[7] D. E. Knuth: The Stanford GraphBase: A Platform for Combinatorial Computing.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface (grafické uživatelské rozhraní)

XML E**x**tensible **m**arkup **l**anguage (rozšiřitelný značkovací jazyk)

JSON **J**ava**S**cript **O**bject **N**otation

HTML **H**yper**T**ext **M**arkup **L**anguage

SVG Scalable Vector Graphics (škálovatelná vektorová grafika)

HTTP **H**yper**t**ext **T**ransfer **P**rotocol

HTTPS **H**yper**t**ext **T**ransfer **P**rotocol **S**ecure

GPS Global Positioning System (globální polohový systém)

AT&T American Telephone and Telegraph

GEXF Graph Exchange XML Format

JGF JSON Graph Format

API Application Programming Interface (aplikační programové rozhraní)

CSS Cascading Style Sheets (kaskádové styly)

WebGL Web Graphics Library

DOM Document Object Model (objektový model dokumentu)

DPI Dots per inch

SMIL Synchronized Multimedia Integration Language

IDE Integrated Development Environment (integrované vývojové prostředí)

MS DOS **M**icro**S**oft Disk Operating System

IT Informační technologie

OS Operační systém

SVN Apache **S**ub**v**ersio**n**

IP Internet Protocol *address*

SSL Secure Sockets Layer

DDoS Denial-of-service *attack*

CD Compact disc (kompaktní disk)

PDF Portable Document Format (přenosný formát dokumentů)

PNG Portable Network Graphics

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `text` ..... text práce
  - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF
- `graphs` ..... implementace aplikace erol.cz
  - `css` ..... kaskádové styly
  - `data` ..... ukázkové grafy ve formátu JSON
  - `fonts` ..... fonty aplikace
  - `img` ..... obrázky aplikace
  - `js` ..... JavaScriptové knihovny
  - `ts` ..... zdrojové kódy implementace
    - `typings` ..... definice ostatních knihoven v TypeScriptu
- `misc` ..... pomocné nástoje práce
  - `maze2json` ..... převod obrázků bludiště do JSON formátu