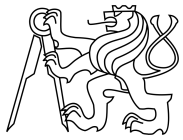
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Prohledávání dopravních síti v GTFS formátu
Student:	Bc. Roman Jelínek
Vedoucí:	Ing. Jan Baier
Studijní program:	Informatika
Studijní obor:	Systémové programování
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Seznamte se s formátem GTFS [1] pro ukládání dat o dopravní síti a prozkoumejte existující nástroje pro práci s tímto formátem. Navrhněte algoritmy pro efektivní zpracování těchto dat za účelem tvorby grafu dopravní sítě. Umožněte následnou editaci sítě bez nutnosti manuálního zásahu do vstupních dat. Editací se rozumí např. spojování či přesun zastávek, přidávání mimořádných spojů, zadávání výluk a další. Implementujte prototyp ukázkové aplikace, která bude s grafem dopravní sítě pracovat a umožňovat vyhledávání tras. Prototyp aplikace porovnejte s již existujícím řešením a navrhněte další vylepšení.

Seznam odborné literatury

[1] https://developers.google.com/transit/gtfs

L.S.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 6. března 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretické Informatiky

Diplomová práce

Prohledávání dopravních sítí v GTFS formátu

Bc. Roman Jelínek

Vedoucí práce: Ing. Jan Baier

9. ledna 2017

Poděkování

Děkuji vedoucímu práce Ing. Janu Baierovi za cenné rady při tvorbě diplomové práce a také rodině a přátelům za dlouhodobou podporu při studiu. Především pak kamarádce, Šárce, která dělala všechno možné i nemožné, aby mě motivovala na této práci pracovat. A v neposlední řadě musím velké díky vyjádřit i svému fanklubu, nebyl to lehký úkol, udržet mě při smyslech během tvorby této práce, ale zvládli to na jedničku.

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 9. ledna 2017 .....

Odkaz na tuto práci

Jelínek, Roman. *Prohledávání dopravních sítí v GTFS formátu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato diplomová práce se zabývá problematikou GTFS dat, tj. dat o hromadné dopravě. V rámci práce byl zpracován přehled některých softwarových produktů pracujících s GTFS daty, dále byla vytvořena ukázková aplikace, která umožňuje úpravu grafu dopravní sítě a vyhledávání v ní. A také práce navrhuje možná vylepšení pro podobné aplikace.

Klíčová slova GTFS, GTFS Feed, transitfeed, Plánování cest

Abstract

This master thesis deals with GTFS data, i.e. data about public transport. The thesis covers an overview of several software products that are being used to do various tasks with GTFS data. The thesis also includes development of a sample application which allows you to edit GTFS data and use it for trip planning. And finally the thesis suggests several improvements that could be used in applications which deal with GTFS data.

Keywords GTFS, GTFS Feed, transitfeed, Trip planning

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato práce se zabývá problematikou zpracování informací o dopravních sítích v GTFS formátu. Potažmo pak dalšími možnostmi využití těchto dat. Typicky jsou tato data využívána k plánování cest za využití spojů městské hromadné dopravy. Aplikace spolupracující s daty v GTFS formátu bývají využívány převážně provozovateli dopravních spojení, tedy dopravci, jelikož existují aplikace, které jim umožňují efektivně spravovat jednotlivé linky, vizualizovat si uložená data a v neposlední řadě z těchto dat například i exportovat jízdní řády pro jednotlivé zastávky apod.

Jelikož je GTFS formát relativně novou záležitostí, není divu, že se stále vyvíjí, dokonce by se dalo říct, že jde o komunitní formát. Na vývoji formátu spolupracují jak dopravci využívající GTFS formát, tak i nezávislí vývojáři a kdokoliv další, kdo má zájem, se může také zapojit a podat žádost, aby byl do feedu zařazen například nový atribut. Například poslední úprava formátu byla schválena v únoru 2016, což bylo jen relativně nedávno před napsáním této práce.

GTFS formát se za poslední léta stal standardem pro správu dat o jízdních řádech a jeho masovému rozšíření přispěly, v neposlední řadě, právě aplikace umožňující efektivnější spravování linek v GTFS formátu. Na internetu je k dohledání několik takových aplikací, tato práce se bude zabývat především těmi nejzákladnějšími využitími takových dat, a to vyhledáváním dopravních spojení při cestování mezi dvěma různými místy a správou dat v GTFS formátu. Na toto téma už bylo napsáno mnoho různých článků a existuje spousta nejrůznějších aplikací, ať už webových nebo mobilních a celkově se jedná o relativně dobře zmapované téma.

Různá existující řešení mívají oproti konkurenčním řešení drobné výhody, například jedna aplikace může umět napovědět nejbližší zastávku na základě polohy, kterou aplikace získá od GPS modulu mobilního telefonu, jiná zase umí dobře přeplánovat další spoje v případě zpoždění některého ze spojů. Aplikace se různí, ale faktem je, že se zmíněná funkcionalita dosti opakuje a v posledních letech i silně stagnuje a nedochází k dalšímu vylepšování existujících řešení.

Motivace

Ačkoliv v dnešní době existuje celá řada aplikací na vyhledávání spojení z jedné zastávky městské hromadné dopravy na jinou zastávku, tak stále nejsou ani zdaleka dokonalé. Toho jsem si všiml především v posledních měsících, kdy jsem byl nucen začít hojně využívat večerních a nočních spojů městské hromadné dopravy. Tehdy jsem teprve poznal krátkozrakost některých řešení, například v situaci kdy je v malém prostoru rozmístěno několik zastávek, mezi kterými by byl uživatel bez problému schopen dojít během pouhých pár minut pěšky, ale aplikace předpokládají pohyb čistě pomocí dopravních prostředků. Tehdy se, v závislosti na tom, kterou zastávku aplikaci uživatel zadá, může čas příjezdu znatelně lišit, třeba i o půl hodiny, zatímco přechod mezi zastávkami je záležitostí maximálně pěti minut.

Toto byl vlastně úplně první nápad, který by mohl být využit při vylepšování stávajících řešení. Jelikož využívaný GTFS formát obsahuje mimo jiné i GPS souřadnice jednotlivých zastávek, tak by mohla aplikace automaticky ve výsledném grafu vytvořit hrany mezi zastávkami, které jsou mezi sebou vzdálené, řekněme maximálně sto padesát metrů, nebo že by rovnou měl sám uživatel možnost si výsledný graf tímto způsobem modifikovat.

Základní pojmy [1]

- **Dopravní cesta (komunikace)** - pás terénu spojující dva koncové body, na němž se uskutečňuje doprava.
- **Dopravní bod** - místa ležící na dopravních cestách, na nichž se uskutečňuje vykládka či nakládka či překládka nákladu, resp. výstup či nástup či přestup cestujících.
- **Dopravní uzel** - dopravní bod, v němž se sbíhají nejméně tři dopravní cesty.
- **Dopravní síť** - sjednocení dopravních bodů a dopravních cest, tedy graf.
- **Dopravní linka** - pravidelné dopravní spojení uskutečňované konkrétním dopravním prostředkem, mezi konkrétními body a v konkrétním čase.
- **Dopravní tah** - svazek dopravních linek ve stejném směru, jinak řečeno soubor různých cest mezi dopravními uzly.

Kapitola 1

Cíl práce

V rámci této práce bych se měl jednak seznámit s formátem GTFS, který se používá pro ukládání informací o dopravních sítích, prozkoumat jeho různá existující využití a pokud možno nahlédnout i do dalších existujících rozšíření tohoto formátu.

Následně bych se měl věnovat i algoritmům, které se používají pro práci s tímto formátem dat, respektive pro transformaci GTFS dat do výsledného grafu dopravní sítě.

Dále by měl být v rámci této práce vypracován i přehled různých existujících programů, které s tímto formátem dokáží dále pracovat. Mimo jiné bych tedy měl poskytnout i určité srovnání existujícího softwaru, což je nutné brát s rezervou, protože GTFS formát má poměrně široké využití a aplikace s ním pracující budou tedy vzájemně dost obtížně porovnatelné, jelikož takové aplikace mohou mít nejrůznější využití.

V neposlední řadě bych měl implementovat i ukázkovou aplikaci, která zvládne s daty v GTFS formátu také pracovat a umožní uživateli data upravovat, a to i bez toho, aby se musel uživatel sám pouštět do editace jednotlivých CSV souborů. Mimo jiné by zmiňovaná aplikace měla umožňovat úpravu grafu dopravní sítě v tom rozsahu, že bude uživatel mít možnost si do grafu přidávat další hrany, přesouvat umístění jednotlivých zastávek a tak podobně. A také by aplikace měla umožňovat vyhledávání spojení v grafu dopravní sítě.

Na závěr bych měl tuto aplikaci, respektive její prototyp, porovnat s existujícími softwarovými řešení a navrhnout další vylepšení, kterými by se dala ukázková aplikace vylepšit oproti konkurenčním aplikacím. A mohl tak přispět k vývoji podobných aplikací.

Kapitola 2

Formát GTFS

GTFS je zkratkou pro General Transit Feed Specification. Jde o poměrně nedávný vynález, formát vešel v existenci v roce 2006 a dalo by se říci, že staví na formátu CSV, jelikož veškerá data jsou uložená v CSV souborech a formát samotný je vlastně jen sada CSV souborů s předepsaným obsahem.

Takzvaný GTFS feed, což je textová reprezentace grafu dopravních bodů, dopravních cest a dalších metadat, se skládá z několika, přesněji šesti až třinácti, textových souborů, respektive CSV souborů, zabalených v jednom ZIP archivu. Jednotlivé CSV soubory obsahují data o zastávkách, trasách a tak podobně, podrobnější rozbor obsahu jednotlivých souborů, viz tabulka 2.1. Je šest základních souborů, které jsou vyžadovány v každém GTFS feedu, plus může feed obsahovat další upřesňující informace.

2.1 Obsah jednotlivých souborů

Jak bylo zmíněno, GTFS formát je sestavou několika různých CSV souborů. Rozhodnutí, že GTFS formát bude stavět na CSV je logické z důvodu, že parsování CSV souborů je algoritmicky jednoduché. Dalším příjemným důsledkem je to, že se nechá pro účely přečtení a úpravě GTFS feedu využít i některá z běžných kancelářských aplikací, jako Microsoft Excel či LibreOffice, které umí s formátem CSV (*comma separated values*) pracovat.

Šest z možných třinácti souborů obsahuje kritické informace o dopravní síti, jako například GPS souřadnice jednotlivých zastávek, časy, kdy se dopravní spoje na jednotlivých zastávkách vyskytují a tak dále [2].

Zbývající soubory mohou obsahovat další dodatečné informace, které mohou být při plánování zohledněny. Mimo jiné informace o ceně jízdného na jednotlivých trasách, nebo pomocná data pro zakreslování linek do mapy či data o samotném GTFS feedu, například datum, kdy přestává platit GTFS feed, tedy datum, kdy dojde ke změně jízdního řádu.

Stručný přehled obsahu jednotlivých souborů je zmíněn v tabulce 2.1 a přehled toho jak jsou spolu jednotlivé soubory provázány je na obrázku 2.1. Další

Tabulka 2.1: Soubory v GTFS feedu

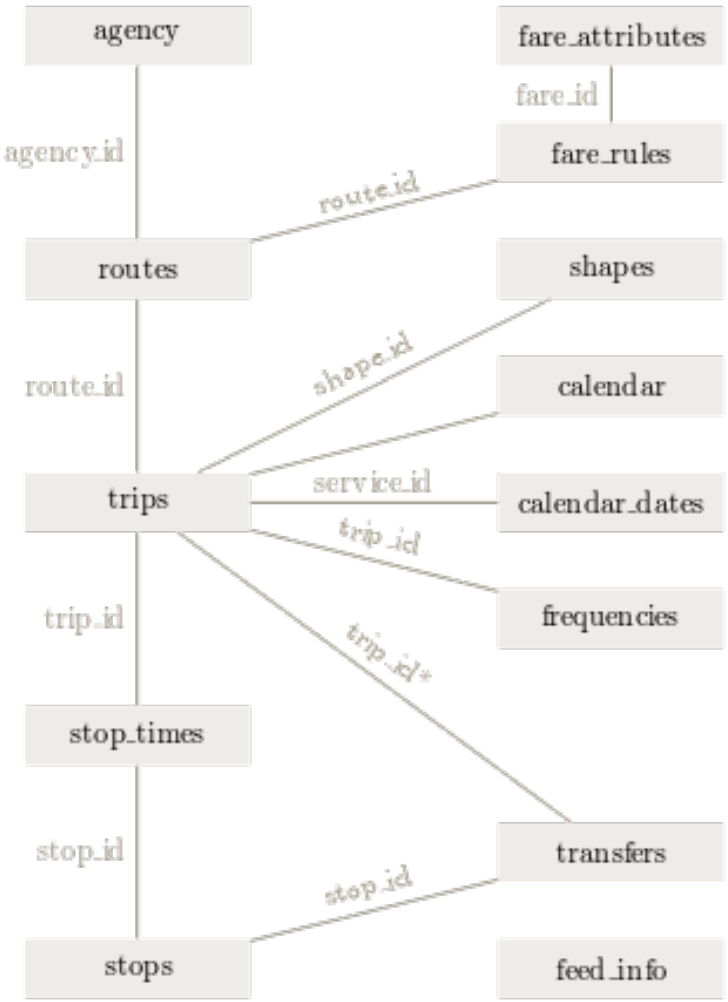
Název	Povinný	Obsah
agency.txt	Ano	Jedna nebo více společností poskytujících data o dopravních linkách.
stops.txt	Ano	Jednotlivé zastávky, kde mohou pasažéři nastoupit či vystoupit.
routes.txt	Ano	Soubor obsahuje popis jednotlivých dopravních linek. Route je skupina tripů z trips.txt.
trips.txt	Ano	Atributy jednotlivých spojů např. zda je autobus bezbariérový atp.
stop_times.txt	Ano	Časy kdy dopravní prostředky přijíždí a odjíždí z jednotlivých zastávek.
calendar.txt	Ano	Specifikace, kdy jednotlivé služby fungují.
calendar_dates.txt	Ne	Výjimky v kalendářním rozložení jednotlivých linek, jako státní svátky a podobné výjimečné události.
fare_attributes.txt	Ne	Informace o ceně jízdného pro dopravní linky dopravní společnosti.
fare_rules.txt	Ne	Pravidla pro aplikaci informacích o jízdném.
shapes.txt	Ne	Pravidla pro zakreslování čar reprezentujících linky do mapy.
frequencies.txt	Ne	Časy mezi jednotlivými spoji na linkách s pravidelnou frekvencí.
transfers.txt	Ne	Další pomocná pravidla pro bližší specifikaci možností přestupu.
feed_info.txt	Ne	Další informace o samotném GTFS feedu jako například vydavatel, verze nebo informace o platnosti feedu.

hezký popis GTFS formátu byl zveřejněn dopravcem Trillium na jeho stránkách, respektive jde spíše o pomocné příklady, které mohou pomoci pochopit komplikovanější využití GTFS formátu, viz `http://trilliumtransit.com/2016/07/05/gtfs-examples/`.

2.2 Další rozšíření GTFS formátu

V průběhu let se samozřejmě postupně přicházelo na to, že formát GTFS není všemocný a nedá se dobře využít pro některé specifické situace. Z toho důvodu se postupně vyvíjely i další odnože GTFS formátu. Jde pouze o modifikace GTFS formátu v tom smyslu, že se pro některá využití GTFS formátu hodilo

Obrázek 2.1: Provázání jednotlivých souborů z GTFS feedu



Zdroj: Wikimedia.org

přenášet více informací, takže bylo pouze nutné přidat další soubory s daty nebo bylo do některých souborů přidáno více atributů.

Mohla by vzniknout otázka, když se někdy hodí více atributů, tak proč se jednoduše nepřidají do specifikace GTFS formátu? Je to z toho důvodu, že základním principem GTFS formátu je, aby byl jednoduchý a pokud možno nenáročný. Jinak by šel formát proti přáním a pohodlí dopravců a nikdo by ve výsledku tento formát nechtěl využívat. Jednak přidání atributu znamená, že najednou dopravce musí dodávat více dat anebo v případě, kdy jsou data využitelná pouze v nějakých okrajových případech a dopravce by je nevyplňoval, tak bude pouze zbytečně docházet ke zvětšování paměťové náročnosti GTFS souborů, protože budou muset obsahovat více oddělovačů hodnot. Dalším dů-

Tabulka 2.2: Rozšíření GTFS formátu

Název	Rozšíření
Google Extensions to GTFS	Rozšíření spočívá v tom, že je například možné zadat data o jízdním do GTFS feedu, který postihuje více dopravců, dalším z přídavků je maximální počet přestupů na jedno zaplacené jízdné, byly přidány další typy dopravních linek a další[5].
GTFS-Plus	Model přepravní dopravní sítě, založený na GTFS formátu, který je vhodný pro dynamické modelování dopravy.
Extra files for GTFS-to-HTML	timetables.txt, timetable_stop_order.txt, route_pages.txt, route_page_assoc.txt - Tyto soubory byly přidány, aby bylo možné generovat časové rozvrhy ve formátu HTML z GTFS feedů pomocí open-source programu GTFS-to-HTML.
GTFS+	Toto rozšíření využívá SF Bay Area's Metropolitan Transportation Commission a jde o sadu dalších souborů, které udávají informace o zlevněném jízdném, dalších atributech zastávek a tak dále.
TriMet Extensions to GTFS	Toto rozšíření přidává další atributy, které informují například o stránce, kde si uživatel může přečíst o tom, jakou politiku vyznává dopravce při přepravě jízdních kol, dále o stránkách, kde uživatel může získat informace o spojích v reálném čase a tak podobně.

vodem je, že někteří menší dopravci stále k udržování informací o dopravní síti využívají běžné kancelářské aplikace a těm mnohdy mohou větší soubory působit potíže. V zájmu využitelnosti formátu je proto dbáno na to, aby nebyl zbytečně složitější než musí být a jeho využívání bylo žádoucí nejen pro dopravce[3].

Existuje několik rozšíření GTFS formátu, některá z nich jsou i dobře zdokumentovaná [4]. Malý přehled několika existujících a využívaných rozšíření GTFS formátu je obsažen v tabulce 2.2.

2.3 Dostupnost GTFS feedů

V nedávné době došlo k masovému zveřejňování GTFS feedů, v mnoha zemích tento akt vedl i k velkému rozmachu služeb využívajících právě GTFS feedy. Tyto feedy jsou dostupné z různých stránek, kam je obvykle nahrávají zástupci společností provozující dopravní linky. Každá dopravní společnost si tedy typicky může spravovat data o svých dopravních linkách.

Hlavním zdrojem GTFS feedů je web `http://gtfs-data-exchange.com/`. Stránka byla založena v roce 2008 a mimo jiné měla sloužit jako podpora používání formátu GTFS a celkově sdílení dat o jízdních řádech. Na stránce jsou k nalezení, mimo spousty dalších, i jízdní řády pražské hromadné dopravy, což jsou data, která jsem se rozhodl využít v implementační části této práce, jelikož je s nimi možné provádět v uvozovkách i ostré testování. Hlavně je to také něco, co sám využiji a nemusím se pouze spoléhat na to, že jiný web najde podobnou cestu, takže je můj výsledek pravděpodobně správný.

Data použitá v implementační části jsou data o jízdních řádech pražského dopravního podniku a jsou jednak uložena na přiloženém CD nebo jsou dostupná aktuální data na stránce `http://www.gtfs-data-exchange.com/`.

2.4 Využití GTFS dat

Samozřejmě GTFS data jsou reprezentací jízdních řádů, ale k čemu se dále dají využívat? Samozřejmě by je dopravci nechtěli využívat, kdyby jim jejich využívání přinášelo pouze nepříjemnosti. Proto pokud by pro dopravce znamenalo využití GTFS dat pouze to, že budou pracovat s daty v nějakém jiném formátu a až pro potřeby zveřejnění musí data převádět do GTFS formátu, určitě by nedošlo k tak masivnímu rozšíření využívání tohoto formátu. Existuje několik různých aplikací, které zvládnou s GTFS formátem pracovat. A mimo jiné usnadňují následující činnosti:

- **Plánování tras** - Této problematice se věnuje mnoho aplikací, plánování přesunu z jednoho místa na druhé za pomoci městské hromadné dopravy patří mezi každodenní problémy většiny lidí, výstupem takových aplikací je posloupnost dopravních linek, jež má uživatel využít, aby se co nejrychleji dostal kam potřebuje.

- **Vytváření časových rozvrhů** - Aplikace usnadňující vytváření časových rozvrhů pomáhají dopravcům, ti totiž musí každou zastávku vybavit časovým rozpisem jednotlivých dopravních spojení.

- **Vizualizace dat** - Aplikace z této skupiny umožňují si GTFS data vhodně vyobrazit a dále tyto vizualizace využívat například ke zlepšení aplikací vyhledávajících v jízdních řádech. Z vizualizovaných dat dokáže člověk vyvodit spojitosti jako kde je rozumné využít k přestupu mezi zastávkami pěší chůze a tak podobně.

- **Asistence postiženým cestujícím** - Tyto aplikace mají za úkol zlehčit život postiženým cestujícím, například jim mohou pomáhat tak, že jim budou doporučovat pouze dopravní prostředky, které jsou pro tělesně postižené cestující přizpůsobené. Nebo v případě slepců jim mohou například nahlas předčítat jízdní řády.
- **Poskytování informací o dopravě v reálném čase** - Další kategorie aplikací umožňuje využívat GTFS data v kombinaci s dalšími daty poskytovanými v reálném čase například předpovídat čas příjezdu dopravního prostředku a tak podobně. Pro toto využití se používá rozšíření základního GTFS formátu GTFS-realtime, který byl navržen tak, aby dopravcům umožňoval sdílet informace o spojích v reálném čase.
- **Sdílení dopravních prostředků** - V neposlední řadě se dají GTFS data využít v aplikacích, kde chce nějaký řidič nabídnout volné místo dalším potencionálním pasažérům.

Formát GTFS je nástroj jako každý jiný a samozřejmě záleží jen na naší fantazii, k čemu všemu ho ještě v budoucnu dokážeme využít.

Kapitola 3

Existující softwarové produkty

V této kapitole bych rád udělal stručný přehled existujících aplikací, které spolupracují s GTFS formátem, ať se věnují řešení jakékoliv problematiky. Většina aplikací se samozřejmě orientuje na vyhledávání tras pomocí dat, nebo je to alespoň můj subjektivní pocit, že zdaleka nejvíce aplikací se orientuje na plánování cesty za pomoci vyhledávání dopravních spojů v jízdních řádech. Proto se v této kapitole budu držet spíše členění, kde kategorizuji aplikace podle jejich účelu a ne například podle toho, zda jsou volně přístupné se zdrojovými kódy anebo komerční. Bohužel není v možnostech této práce poskytnout informace ani o všech existujících typech aplikací pracujících s GTFS daty.

3.1 Plánování cesty

První probíraná, a také asi nejzajímavější, kategorie aplikací se orientuje na vyhledávání dopravních spojů v jízdních řádech.

3.1.1 Google Maps

Jednou z nejznámějších webových aplikací, která zvládá plánovat trasy s využitím GTFS dat o městské hromadné dopravě jsou Google Mapy. Samozřejmě, že v případě aplikace od takového technologického giganta, jakým společnost Google bez pochyby je, existuje i mobilní varianta. Aplikace je pro nekomerční použití zcela zdarma, takže pro osobní potřebu je vlastně ideální. Nad touto aplikací staví i další aplikace od technologického giganta, společnosti Google, jako Google Ride Finder nebo Google Transit. Webová stránka dokonce umožňuje plánovat cesty i za použití automobilu, nebo pěší chůze. Pro vývojáře získávají mapy další rozměr, jelikož umožňují využít rozsáhlé API pro použití map pro vlastní účely. Mezi podporované funkce API Google Map patří například možnost zanést si do mapy vlastní značky, je možné si do mapy něco zakreslit nebo i mapy úplně překreslit, vývojář si tedy může vyznačit třeba

celou ulici, pokud to uzná za vhodné. Možnosti jsou omezené pouze vlastní představivostí.

3.1.2 IDOS

IDOS je zkratka pro informační dopravní systém a jedná se možná o vůbec nejrozšířenější aplikaci pro vyhledávání v jízdních řádech v České republice. Aplikace má tu výhodu, že je dostupná jak její webová, tak i mobilní varianta a obě jsou k použití zdarma. Aplikace umožňuje vyhledávat dopravní spoje nejenom v jízdních řádech městské hromadné dopravy, ale obsahuje i jízdní řády vlakových souprav a autobusů v rámci celé české republiky. Další pozitivum aplikace tkví v tom, že aplikace má ve své databázi i ceníky některých spojů, při používání jsem si toho všiml asi pouze v případě vlakové dopravy, ale zrovna u vlakové dopravy se dá předpokládat, že nemá cestující předplacenou dopravu například na měsíc dopředu, jak tomu bývá v případě městské hromadné dopravy, takže zde má tato funkcionalita svoje využití.

3.1.3 Pubtran

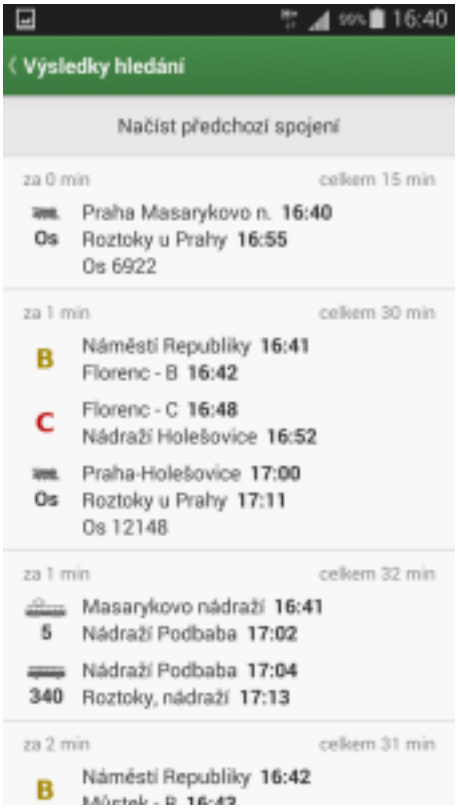
Jde o mobilní aplikaci, která je velmi podobná mobilní aplikaci IDOS, i její výsledky vyhledávání se alespoň na první pohled nijak neliší od aplikace IDOS. Obě aplikace pravděpodobně čerpají ze stejného zdroje dat. Mezi aplikacemi je však rozdíl v grafickém uživatelském prostředí (ukázka aplikace Pubtran je na obrázku 3.1), takže i pokud jsou skutečně jinak obě aplikace stejné (v tom smyslu, že poskytují stejné výsledky při vyhledávání), uživatel má možnost si vybrat, která z aplikací lahodí jeho oku více. Aplikace má i pár dalších zajímavých funkcí, například je možné si nalezené dopravní spojení uložit do kalendáře nebo sdílet informace o spojení pomocí SMS zprávy.

3.1.4 CG Transit

Mobilní aplikace CG Transit je také jednou z oblíbenějších aplikací mezi uživateli, a to i navzdory tomu, že jako jedna z mála není zcela zdarma. Respektive samotná aplikace je dostupná zdarma a první měsíc používání aplikace je zcela zdarma, ale po měsíci musí uživatel začít platit licenční poplatky za využívané jízdní řády. Ovšem samotná aplikace bez jízdních řádů je samozřejmě celkem k ničemu, takže pokud ji člověk chce využívat, nezbývá mu, než si připlatit. V aktuální době je podle stránky výrobce cena jízdních řádů pro českou republiku stanovena na devadesáti korunách za rok, což není až tak mnoho[6].

Aplikace má spoustu doplňkových funkcí, například umožňuje vyhledávání v řádech i bez připojení k internetu anebo si uživatel může po vyhledání spojů pouhým přejetím prstem zobrazit další spoj mezi stejnou dvojicí zastávek, pokud například jedu z Chrudimi do Prahy a nestíhám navazující vlak z Pardubic, tak přejedu prstem a rázem vidím v kolik hodin jede další vlak

Obrázek 3.1: Ukázka aplikace Pubtran



Zdroj: Android.chaputo.cz

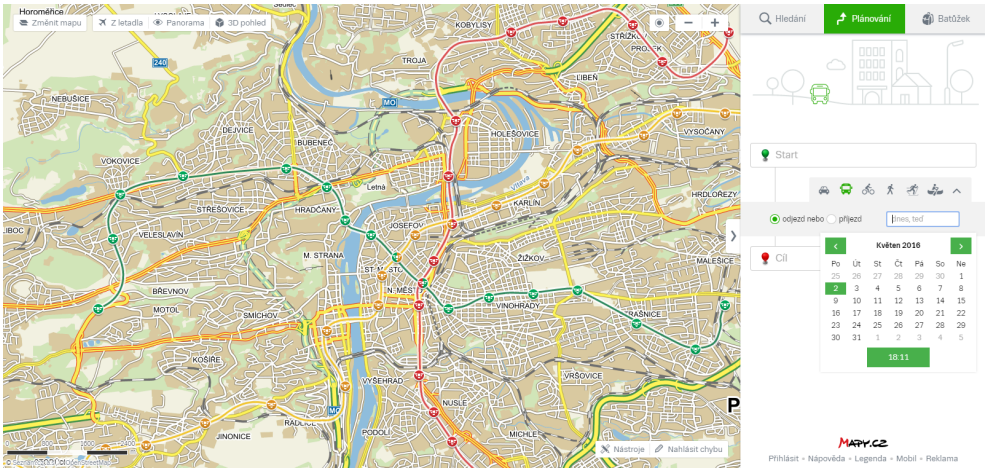
z Pardubic do Prahy. Podobnou funkcionalitu jistě uživatel ocení v případě, že sedí ve zpožděném dopravním prostředku a už bezpečně ví, že navazující spoj nestíhá, a tak si chce ihned zjistit, jak dlouho bude čekat na přestup. Tato funkcionalita může uživateli hodně usnadnit plánování.

3.1.5 Mapy.cz

Konceptuálně velmi podobná webová aplikace jako v případě Google map. Podobně jako Google Maps, podporuje plánování cest za využití více způsobů dopravy, jmenovitě autem, městskou hromadnou dopravou, na kole, pěšky a dalšími. Nicméně je vyhledávání v dopravních spojeních městské hromadné dopravy stále v beta verzi.

Oproti Google Maps za portálem Mapy.cz nestojí takový technologický gigant, takže je tento projekt o něco méně ambiciózní a nesnaží se pokrýt celý svět, soustředí se primárně na Českou republiku, ale dělá to dobře. Dokonce také umí nakombinovat využívání pěší chůze a dopravních prostředků městské hromadné dopravy a staví se tím do pozice jedné ze dvou aplikací u kterých jsem zjistil, že oplývají takovouto funkcionalitou.

Obrázek 3.2: Plánování trasy na Mapy.cz



Zdroj: Mapy.cz

3.1.6 Open Trip Planner

Tato aplikace se ostatním vymyká především tím, že jde o open-source řešení. Podle dokumentace, která je na internetu veřejně přístupná, je aplikace stále v aktivním vývoji, i když hned vedle je psáno, že je plánován release ve verzi 1.0 do konce roku 2015 [7]. Prakticky jde o víceúčelovou platformu, která umožňuje plánování tras za použití dat od více dopravců, umožňuje plánování více typů cestování (například pěší, na kole, kombinaci městské hromadné dopravy a pěší chůze...) a také usnadňuje analýzu dopravní sítě. Aplikace zvládá pracovat nejen s klasickou verzí GTFS formátu, ale i s jeho rozšířením GTFS-realtime. Aplikace také poskytuje, jak webové rozhraní, tak i API, které jistě mnoho vývojářů ocení.

3.1.7 Shrnutí

Aplikací určených k tomuto účelu, ať webových, desktopových anebo mobilních existuje celá řada.

Obzvláště v případě mobilních aplikací by rozhodně nebylo možné dělat závěry, který z jmenovaných aplikací je nejlepší, jelikož se zkoumané aplikace ve své funkcionalitě dosti liší. Slovy klasika „jedna aplikace umí to a ta zase tohle“, takže v oblasti mobilních aplikací skutečně závisí na tom, které funkcionality si uživatel váží nejvíce. Proti jedné z aplikací by se dalo namítnout, že není k použití zcela zdarma, ale nejde o nijak závratnou částku, kterou by si uživatel využívající denně hromadnou dopravu nemohl dovolit.

Celkově si myslím, že ale nejlepší možnosti vyhledávání dopravních spojů poskytují Google Mapy, jelikož podporují možnost kombinovat využívání dopravních prostředků městské hromadné dopravy s pěší chůzí. V rámci České

republiky by pak mohly být konkurence schopné i Mapy.cz, ale občas je zatím bohužel znát, že je tato služba v betaverzi, protože Google Mapy jsou občas schopné nalézt časově výrazně lepší trasu.

3.2 Tvorba časových rozvrhů

Tato skupina aplikací bude zajímavá víceméně pouze pro dopravce, nebo alespoň mě nenapadá, k čemu by běžnému uživateli bylo, že si například vygeneruje časový rozvrh jedné zastávky a někde si ho vylepí. Nicméně užší cílová skupina neznamená, že jsou tyto aplikace neužitečné.

3.2.1 TimeTable Publisher

TimeTable Publisher je aplikace, která je dostupná zcela zdarma, dokonce je vydaná jako open-source. Tento program vytváří časové rozvrhy v HTML a PDF a uživateli už je vlastně stačí jenom vytisknout na tiskárně. TimeTable Publisher získává data buď z databáze, nebo z GTFS souborů a převede je do tisknutelné podoby. Tento software je populární především ve spojených státech, kde je využíván velkými dopravci, jako je například TriMet.

Program obsahuje i zajímavou funkci, kdy dokáže ukázat změny provedené v jízdních řádech, sice kvůli tomu musí být připojen k databázi, ale i tak je to zajímavá funkcionalita.

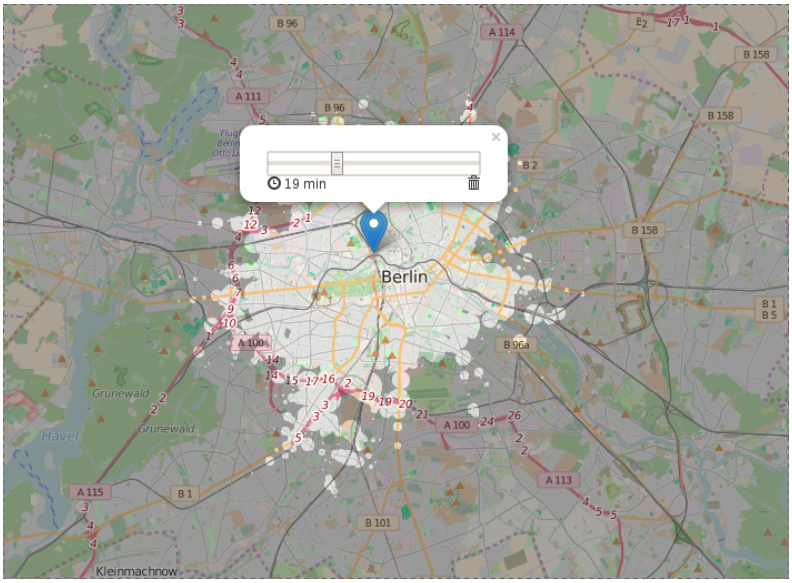
3.2.2 ITO GO

Další aplikace, která je určená k usnadnění tvorby časových rozvrhů, které následně musí dopravce vylepit u jednotlivých zastávek městské hromadné dopravy. I tato aplikace by měla docílit toho, že rozparsuje GTFS data a převede je do tisknutelné formy, respektive automatizovat značnou část práce, aby dopravci stačilo už jenom vytisknout rozvrhy a rozdistribuovat je na zastávky.

3.3 Vizualizace

Další typ aplikací je opět zajímavý pro širší spektrum uživatelů, některé aplikace totiž dokáží využít veřejné dostupnosti GTFS feedů vskutku kreativně. Existují aplikace, které dokáží jen tak do roviny vykreslit trasy různých dopravních prostředků a tím de facto i efektivně vykreslit téměř až mapu města a také jsou aplikace, které se sice zabývají úplně jinou problematikou, jmenovitě prodejem nemovitostí a GTFS využívají jen k tomu, aby demonstrovaly, v jaké dobré lokalitě se nemovitost nachází.

Obrázek 3.3: Ukázka aplikace Mapnificent



Zdroj: Mapnificent.net

3.3.1 Mapnificent

Tato webová aplikace je vskutku originální, nestará se o jakékoliv plánování tras, správu GTFS dat, vlastně by se dalo říct, že snad ani nedělá nic užitečného. Uživatel dostane na výběr jedno z měst v Evropě nebo spojených státech, dále si vybere časový interval a aplikace mu vyznačí na mapě, do kterých míst na mapě by se uživatel z daného počátečního místa dostat s pomocí městské hromadné dopravy, pokud by k tomu měl zadaný časový interval. Možná aplikace není nijak průlomová, co se týče její komerční využitelnosti, ale alespoň podle mě jde o vskutku kreativní nápad. Ukázka, jak aplikace vypadá je na obrázku 3.3.

3.3.2 Walk Score

Tato webová aplikace se primárně orientuje na nabízení nemovitostí k pronájmu a k prodeji a přistupuje k tomuto záměru dosti zodpovědně. Na jejich webu si tak člověk může zobrazit, co se nachází v okolí dané nemovitosti za restaurace, obchody, kavárny, školy, parky a tak podobně. A k tomu nabízí stránka i přehled jak to vypadá v okolí s dopravou, respektive jaké jsou po okolí možnosti cestování autem, na kole, autobusem a samozřejmě i pěšky. Na základě těchto údajů aplikace přidělí nemovitosti tzv. „Walk Score“, odtud vlastně pochází název webové stránky, že počítá co se dá u nemovitosti obejít pěšky.

3.4 Spravování GTFS feedů

Tato problematika už se týká ryze dopravců, možná v nějakých extrémních případech i nadšených jednotlivců, ale to je čistě okrajová záležitost. Dopravci musí řešit problémy se správou svých GTFS feedů, aby je samozřejmě mohli později již bezpracně zveřejnit.

3.4.1 Google Transit Data Tool for Small Transit Agencies

Tato aplikace byla takovým pomocným krokem od Googlu menším dopravcům, aby neměli problém zdarma poskytnout data o svých linkách službě Google Transit. Pro menší dopravce bylo problematické převádět svoje aplikace do GTFS formátu, a tak jim Google vytvořil open-source aplikaci, ve které mohou svoje linky přímo spravovat.

Principem aplikace nebylo vůbec, aby poskytovala nějaké úžasné funkce, kterými by vyšachovala podobné aplikace, ale pouze poskytnou menším dopravcům nějakou alternativu pro správu dat, která by je přiměla přejít na GTFS data.

3.4.2 Trillium GTFS Manager

Údajně první a samozvaně nejpoužívanější software pro správu GTFS dat je Trillium GTFS Manager [8]. Program umožňuje v GTFS feedech provádět jednoduché úpravy, jako přesun zastávky jejím posunutím na mapě, podobným způsobem se nechá změnit i trasa linky a další data, která jsou v GTFS feedu uložena.

Jako další svoji velkou výhodu Trillium GTFS Manager uvádí, že jakožto webová aplikace, je dostupný všude a uživatel má vždy k dispozici aktuální verzi veškerých dat. Ačkoliv je tato vlastnost velmi užitečná, tak nevím zda je důvod se s tím v dnešní době chlubit, když takovým způsobem funguje už spousta různých aplikací.

Kapitola 4

Algoritmy pro aplikace využívající GTFS

Při vývoji aplikací pracujících s GPS souřadnicemi a GTFS formátem se hodí znát a umět používat všemožné algoritmy. V aplikacích, které pracují s problematikou GPS souřadnic se určitě může hodit vědět, jak například spočítat vzdálenost mezi dvěma body na povrchu zemském pomocí znalosti zeměpisné šířky a délky obou bodů. Případně by se mohla hodit znalost převodů mezi UTM (Univerzální transverzální Mercatorův systém souřadnic je obdoba GPS, jde pouze o jiný způsob určování polohy na povrchu Země založený na mřížkách) a GPS souřadnicemi.

Při práci přímo s GTFS se může hodit i základní znalost technik využívaných při parsování textu, protože pokud chce člověk s daty dále pracovat, je nejlepší si je v paměti stroje reprezentovat lépe, než jako dlouhý text přečtený z CSV souboru, v němž nejsou reprezentovány žádné souvislosti mezi zastávkami, spoji a tak podobně. Základní varianta CSV se skládá pouze z hodnot oddělených čárkou a pro lepší variabilitu je možné vložit text do uvozovek, aby nedošlo k jeho nechtěnému oddělení čárkou (například, když potřebujeme mít v souboru nějaký text, který může obsahovat čárky). Pro lepší pochopení problematiky by bylo dobré zmínit úvod do gramatik, kterými se dá popsat CSV formát.

Gramatika je čtveřice $G = (N, T, P, S)$, kde[9]:

- N je konečná množina neterminálních symbolů
- T je konečná množina terminálních symbolů ($N \cap T = \emptyset$)
- P je množina pravidel, je to konečná podmnožina množiny $(N \cup T)^*.N.(N \cup T)^* \times (N \cup T)^*$
- $S \subset N$ je počáteční symbol gramatiky

Gramatiky lze dle Noama Chomského klasifikovat do čtyřech typů[9]. Pakliže $G = (N, T, P, S)$, pak říkáme, že G je:

- Neomezená (typu 0), jestliže odpovídá obecné definici gramatiky.
- Kontextová (typu 1), jestliže každé pravidlo z P má tvar $\gamma A\delta \to \gamma\alpha\delta$, kde $\gamma, \delta \in (N \cup T)*, \alpha \in (N \cup T)+$, $A \in N$, nebo tvar $S \to \epsilon$ v případě, že S se nevyskytuje na pravé straně žádného pravidla.
- Bezkontextová (typu 2), jestliže každé pravidlo má tvar $A \to \alpha$, kde $A \in N$, $\alpha \in (N \cup T)*$.
- Regulární (typu 3), jestliže každé pravidlo má tvar $A \to aB$ nebo $A \to a$, kde $A, B \in N$, $a \in T$, nebo tvar $S \to \epsilon$ v případě, že S se nevyskytuje na pravé straně žádného pravidla.

Podle toho, jakým typem gramatiky lze jazyk popsat následně klasifikujeme i jazyky na rekurzivně spočetné (typ 0), kontextové (typ 1), bezkontextové (typ 2) a regulární (typ 3). Celý formát (odborněji řečeno mluvíme o jazyce) je velice jednoduchý, je popsatelný regulární gramatikou[10] a lze ho proto strojově číst za pomoci pouhého konečného automatu[9].

Přesnou specifikaci CSV formátu je možno dohledat i jako RFC4180. Ve zmíněném RFC lze dohledat i gramatiku v rozšířené Backus–Naurově formě[11]:

$$
\begin{aligned}
file &= [header\ CRLF]\ record*(CRLF\ record)\ [CR\ LF] \\
header &= name*(COMMA\ name) \\
record &= field*(COMMA\ field) \\
name &= field \\
field &= (escaped\ /\ nonescaped) \\
escaped &= DQUOTE \\
&\quad *(TEXTDATA\ /\ COMMA\ /\ CR\ /\ LF\ /\ 2DQUOTE) \\
&\quad DQUOTE \\
nonescaped &= *TEXTDATA \\
COMMA &= \%x2C \\
CR &= \%x0D\ ;\ as\ per\ section\ 6.1\ of\ RFC\ 2234\ [2] \\
DQUOTE &= \%x22\ ;\ '\backslash n'\ as\ per\ section\ 6.1\ of\ RFC\ 2234\ [2] \\
LF &= \%x0A;\ '\backslash r'\ as\ per\ section\ 6.1\ of\ RFC\ 2234\ [2] \\
CRLF &= CR\ LF;\ as\ per\ section\ 6.1\ of\ RFC\ 2234\ [2] \\
TEXTDATA &= \%x20-21\ /\ \%x23-2B\ /\ \%x2D-7E
\end{aligned}
$$

Jelikož budeme dále pracovat s grafem dopravní sítě, bylo by dobré si zadefinovat graf a zmínit i další vlastnosti, které graf dopravní sítě bude splňovat. Jednotlivé zastávky si budeme reprezentovat

Neorientovaný graf G je definován jako uspořádaná dvojice množiny uzlů V a množiny hran E. Přičemž množina uzlů musí být neprázdná a zároveň konečná a množina hran je podmnožina všech dvojic vrcholů. Množinu všech dvouprvkových podmnožin množiny V (tedy všech možných hran) budeme označovat $\binom{V}{2}$.

$$G = (V, E) \tag{4.1}$$

$$E \subseteq \binom{V}{2} \tag{4.2}$$

Pro potřeby této aplikace budeme potřebovat orientovaný graf s ohodnocenými hranami, kde ohodnocení hran bude reprezentovat kolik času budeme potřebovat. Ale v rámci implementace se problém ještě trochu zkomplikuje, takže to vezmeme postupně. Orientovaný graf se od neorientovaného liší tím, že hrana mezi dvěma vrcholy má jednoznačně určený směr, kterým směřuje, z to plyne následující vztah4.3.

$$E = V \ \times V \tag{4.3}$$

Dále budeme vyžadovat nezáporné ohodnocení všech hran v grafu a pro běžné využití nám postačí i obor celých čísel, kdy nám ohodnocení hran bude vyjadřovat počet sekund potřebných na přesun mezi zastávkami, respektive mezi dvěma vrcholy. V rámci implementace přibude i drobné zesložitění v tom, že hrany reprezentující dopravní spoj budou muset reflektovat i to, že je nutné vyčkat času příjezdu dopravního spoje, ale to je čistě implementační záležitost. Ohodnocení jednotlivých hran je matematicky definováno jako zobrazení z množiny hran do (v tomto konkrétním případě) množiny nezáporných celých čísel, viz zápis 4.4 (i nulové ohodnocení hrany může mít svůj význam, například když autobus staví na témže místě jako tramvaj, ale z hlediska feedu se jedná o dvě různé zastávky).

$$w : E \to N_0 \tag{4.4}$$

Jelikož předpokládám u čtenářů tohoto dokumentu alespoň elementární znalost grafů a grafových algoritmů, tak v této kapitole nechci dopodrobna rozebírat různé možnosti reprezentace grafu v paměti počítače. Chci se spíše zaměřit na to, jak s již uloženými daty pracovat, respektive jak je dále využít pro potřeby vznikající aplikace. Elementárním bodem aplikace tedy sice bude i převod dat mezi CSV soubory a pamětí počítače, ale protože nejde o nijak algoritmicky náročnou činnost, tak bude tento problém diskutován až v kapitole Analýza.

4.1 Výpočet vzdálenosti dvou bodů

Prvním základním kamenem, který se dá využít ve snad každé aplikaci pracující s GPS souřadnicemi, je výpočet vzdálenosti mezi dvěma místy zadanými pomocí GPS souřadnic. Například v aplikacích na vyhledávání dopravních spojení se dá tato funkcionalita využít pro detekci dvojice zastávek, které jsou u sebe blízko a tím pádem je pravděpodobně možné mezi nimi dojít pěší chůzí.

Existuje více matematických vzorců, které více či méně přesně dokáží vypočítat vzdálenost mezi místy. Dá se setkat s přístupy, které výpočet vzdálenosti mezi dvěma místy zjednodušují až na úroveň Pythagorovy věty [12]. Tu známe všichni ze střední školy a víme, že se dá dobře využít na zjištění vzdálenosti dvou bodů v rovině, nicméně i navzdory tomu, že v některých případech tento názor přetrvává i do dnešní doby, tak Zeměkoule zkrátka není rovina a při měření vzdálenosti mezi dvěma místy, která jsou od sebe vzdálená, řekněme, tisíce kilometrů už bude vznikat značná odchylka.

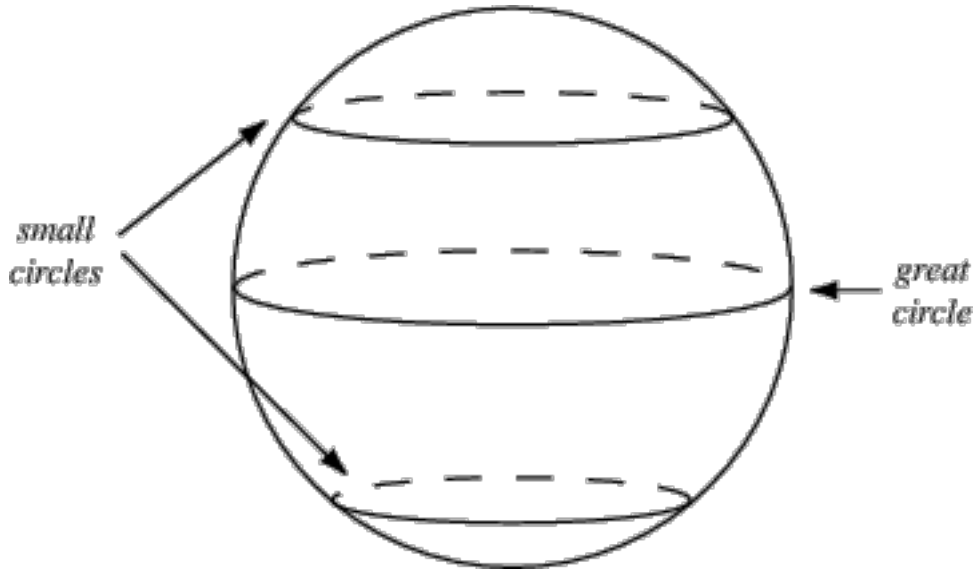
I když by se dalo říct, že na úrovni například jednoho dopravního spoje nebudeme řešit žádné enormní vzdálenosti, a že by nám tato odchylka nemusela až tolik vadit, tak nás bude zajímat poněkud přesnější metrika, už jen proto, aby nedocházelo k velkým odchylkám, když se uživatel pokusí nějaký program využít na práci s GTFS feedem z nějakého většího města. V případě zájmu lze nahlédnout i jaké přibližně odchylky může tento způsob výpočtu vzdálenosti způsobit a i jaké jsou problémy s používáním dalších způsobů výpočtů vzdálenosti dvou míst na Zeměkouli [13].

O něco přesnější matematické vzorce, které mají pro většinu aplikací více než dostatečnou přesnost výpočtu, vycházejí z předpokladu, že je Zeměkoule kulatá. Je daleko jednodušší akceptovat drobné nepřesnosti, než se snažit dokonale nasimulovat výpočet vzdálenosti dvou bodů na geoidu. Díky tomu, že se Zeměkoule zjednodušuje, pro potřeby výpočtu na kulovou plochu, tak je výpočetní vzorec jednak neskonale jednodušší, což usnadňuje i jeho následnou implementaci a také zkracuje výpočetní čas potřebný k získání výsledku, což přináší i malé plus v případě mobilních zařízení, kdy při opakovaném výpočtu vzdálenosti trochu šetříme energií.

Jelikož si výpočet vzdálenosti dvou míst upravíme na výpočet vzdálenosti dvou bodu na kruhové ploše, tak můžeme tento problém pojmenovat také jinak, a to jako výpočet délky ortodromy. Ortodroma je nejkratší spojnice dvou bodů na povrchu kulové plochy (v tomto případě na povrchu Zeměkoule), viz obrázek 4.2. Když si představíme hlavní kružnici (tedy kružnici jejíž střed je ve středu koule a jež se rozprostírá po povrchu koule, viz obrázek 4.1), která prochází oběma body, mezi nimiž chceme změřit vzdálenost, tak ortodroma je kratší z oblouků mezi dvojicí bodů [14]. V zahraničních zdrojích lze tuto problematiku dohledat pod hesly jako *great circle distance* nebo *haversine formula*.

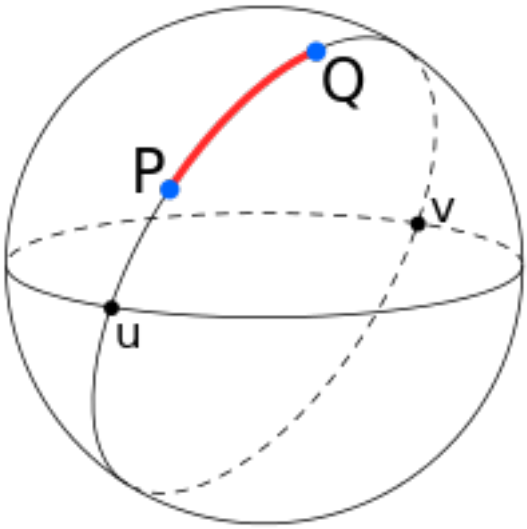
Nejkratší vzdálenost bodů P a Q na povrchu kulové plochy (respektive Zeměkoule, v následujícím textu budu předpokládat, že je Zeměkoule dokonale

Obrázek 4.1: Ilustrace hlavní kružnice



Zdroj: Mathworld.wolfram.com

Obrázek 4.2: Ilustrace ortodomy



Zdroj: Wikipedia.org

kulatá a pojmy jsou tedy zaměnitelné), které jsou umístěné na povrchu kulové plochy na zeměpisné šířce δ_1 a zeměpisné délce λ_1, respektive zeměpisné šířce δ_2 a zeměpisné délce λ_2 v případě bodu Q. Abychom tedy zjistili vzdálenost bodů P a Q na kulové ploše o poloměru a, převedeme si sférické souřadnice [15] na polární souřadnice za pomoci vztahu 4.5.

$$\mathbf{r}_i = a \cdot \begin{pmatrix} \cos\lambda_i \cos\sigma_i \\ \sin\lambda_i \cos\sigma_i \\ \sin\sigma_i \end{pmatrix} \tag{4.5}$$

Nyní si dopočítáme úhel α mezi $\mathbf{r}_1$ a $\mathbf{r}_2$ s pomocí skalárního součinu a vztahu 4.6.

$$\begin{aligned} cos\alpha &= \widehat{\mathbf{r}_1} \cdot \widehat{\mathbf{r}_2} \\ &= \cos\sigma_1 \cos\sigma_2 \left(\sin\lambda_1 \sin\lambda_2 + \cos\lambda_1 \cos\lambda_2\right) + \sin\sigma_1 \sin\sigma_2 \\ &= \cos\sigma_1 \cos\sigma_2 \cos\left(\lambda_1 - \lambda_2\right) + \sin\sigma_1 \sin\sigma_2 \end{aligned} \tag{4.6}$$

Ještě bych rád doplnil poznámku, přeci jen ne každý se s tímto v matematice běžně setká, že stříška nad vektorem $\mathbf{r}_i$ v předchozím matematickém vzorci má význam normalizovaného vektoru, viz vztah 4.7.

$$\widehat{\mathbf{r}_i} = \mathbf{r}_i / \left\|\mathbf{r}_i\right\| \tag{4.7}$$

Poté už stačí do vzorců jen dosadit požadované hodnoty, tedy zeměpisné šířky a délky obou bodů a v případě Zeměkoule poloměr $a \approx 6378$ km a máme hotovo.

Z mého pohledu poskytuje tento vztah už téměř pro každou aplikaci dostatečnou přesnost při výpočtu vzdálenosti dvou bodů na mapě, ale tímto možnosti výpočtu ani zdaleka nekončí, existují ještě přesnější matematické vzorce. Případní zájemci se mohou více dočíst na internetové stránce MathWorld[16]. Na zmíněné stránce je k nalezení i další vzorec, který je ještě přesnější, neboť zkouší lépe aproximovat tvar Zeměkoule, tedy nepředpokládá, že je Země dokonale kulatá, ale uvažuje o ní jakožto o elipsoidu. Zmíněný vzorec je sice přesnější, ale i výrazně komplikovanější, což je vidět i na zmíněné webové stránce, kdy popis komplikovanějšího vzorce sestává z daleko většího počtu vzorců,

4.2 Transformace GTFS dat do grafu

Jak jsem zmínil už v úvodu, na samotném převodu GTFS dat do grafové reprezentace není vcelku nic příliš náročného, proto zde nepůjdu až tolik do detailu. Zajímavější situace nastává, když se programátor rozhoduje jakou reprezentaci grafu použije, ale tento problém bude diskutován až v pozdější části této práce.

Obecně řečeno, problém sestává z rozparsování několika CSV souborů a využití jejich logických závislostí, abychom z nich dostali potřebné informace. Formát souborů CSV byl navržen tak, aby byl jednoduše strojově čitelný, což situaci dost usnadňuje. Jak už z názvu formátu vyplývá, celý soubor je vlastně jen posloupnoust jednotlivých hodnot oddělených čárkami (případně i jinými oddělovači, oddělovač je většinou nastavitelný v konkrétním programu). Situace je složitější, pokud je například součástí nějaké hodnoty samotný oddělovač, pak je nutné celou hodnotu obalit uvozovkami. Toto samozřejmě trochu komplikuje implementaci parseru, ale nijak zvlášť, stále lze parser implementovat za pomoci konečného automatu, kdy si parser pouze drží stav, zda je aktuálně mezi uvozovkami nebo ne,

K implementaci takového parseru dobře poslouží gramatika zmíněná na začátku kapitoly, dále už jde spíše o mechanickou práci, minimálně pro někoho, kdo někdy už nějaký LL(1) parser napsal. Případně je možnost sáhnout po některém z již existujících parserů.

Další částí problému je převod přečtených textových dat do více abstraktní úrovně, tedy nějakých datových struktur, se kterými se bude lépe pracovat. Zde postačí využít jednoduchého mapování stejnojmenných atributů, které jsou ve více souborech, nebo ještě lépe se orientovat podle nákresu 2.1. Tedy v praxi si vytvořím například strukturu reprezentující zastávku a zajímá mě její spojitost s časy, kdy by měly jednotlivé dopravní prostředky na této zastávce zastavovat, kouknu na nákres a hned vím, že bych měl na sebe namapovat záznamy ze souborů stop_times.txt a stops.txt, které sdílejí stejné stop_id. Základním stavebním kamenem je tedy úspěšně postavit graf dopravní sítě, kde uzly grafu reprezentují jednotlivé zastávky a hrany grafu vyjadřují jízdní řády samy o sobě, tedy vlastně odjezdy dopravních prostředků. Přesná implementace už pak závisí na použitém programovacím jazyce.

4.3 Vyhledávání spojů v grafu

Nejzajímavější algoritmy, které jsou zde probírané se týkají vyhledávání cest v již vytvořeném grafu. Grafových algoritmů, které by se daly využít existuje více. Ale základ mají stejný, vycházejí z tak zvaného Dijkstrova algoritmu [17], potažmo pak z procházení do šířky (anglicky breadth first search) [18].

Vyhledávací algoritmy sice staví na základech Dijkstrova algoritmu, ale asi jen málo kdy bychom se v podobné aplikaci setkali s čistou implementací Dijkstrova algoritmu (i když v aplikaci, které postačuje pokrýt relativně malou dopravní sítí by se to sneslo), většinou se totiž snažíme co nejvíce vyhledávání urychlit, a to tak, že se snažíme za pomocí různých heuristik minimalizovat počet uzlů, které musíme v běhu algoritmu zpracovat před tím, než dorazíme do cíle. Pokud bychom například hledali nejrychlejší dopravní spojení mezi Prahou a Brnem v železniční síti, tak pravděpodobně bude lepší začít při prohledávání grafu začít zpracovávat uzel Pardubice dříve, než uzel Plzeň. Tedy

jako taková základní heuristika by mohla dobře posloužit vzdálenost dvou uzlů mezi sebou na mapě. Někdy není jednoduché přijít s dobrou heuristikou, obzvlášť protože od heuristiky chceme, aby nám nerozbila vyhledávání a proto na ni máme určitě požadavky. Například u problému hledání nejkratší cesty budeme chtít, aby nikdy nepřecenila cenu za dopravu z aktuálního uzlu do cíle, pokud by tomu tak nebylo, tak by se mohlo stát, že nenalezneme skutečně nejkratší cestu, nalezli bychom ji třeba rychleji, ale nebylo by to naše požadované řešení. Tomuto algoritmu (tedy vlastně Dijkstrově algoritmu s použitím heuristiky) se říká A* (anglický název je A star) vyhledávání [19].

4.3.1 Dijkstrův algoritmus

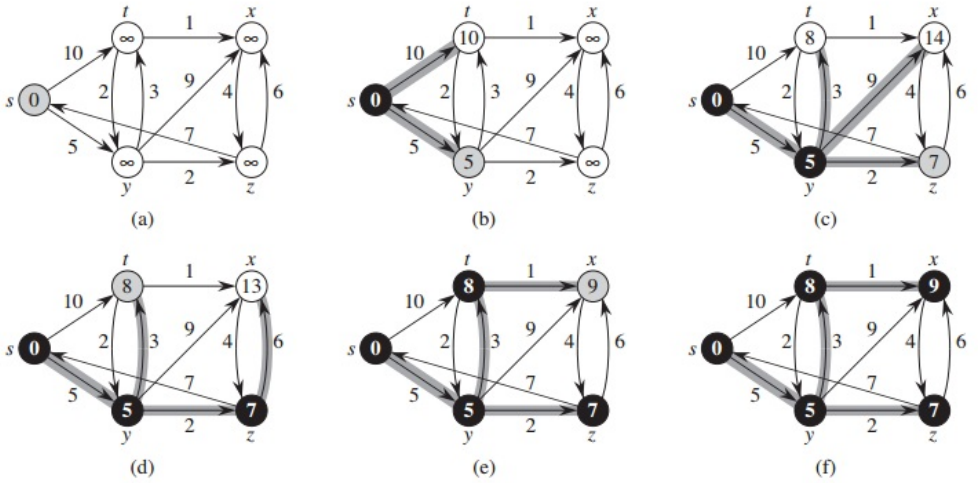
Dijkstrův algoritmus je určený na hledání nejkratší cesty v grafu s nezáporně ohodnocenými hranami. Respektive lze ho použít i v grafu, kde se vyskytují záporně ohodnocené hrany, nicméně by se v grafu neměl vyskytovat negativní cyklus (to by algoritmus nemusel nikdy skončit) a také si negativním ohodnocením hran zavíráme dveře k dalším optimalizacím, jako například ukončení hledání nejkratší cesty ve chvíli, kdy poprvé vybereme z prioritní fronty cílový uzel. Otázka nezáporných hran nás v grafu dopravní sítě netrápí, ať už budeme počítat s jakoukoliv metrikou hran (čas potřebný na přejetí ze zastávky na zastávku, cena jízdného mezi zastávkami, vzdálenost mezi zastávkami...), tak velice pravděpodobně nebudeme potřebovat počítat se zápornou cenou hran. Pokud bychom náhodou potřebovali počítat se záporným ohodnocením hran, museli bychom použít jiný grafový algoritmus.

Dijkstrův algoritmus je velmi podobný Primově algoritmu na hledání minimální kostry v grafu, i v tomto grafu si de facto vytváříme strom nejkratších cest, které mají kořen v zadaném vrcholu grafu. Vlastně máme dvě existující množiny, jedna obsahuje uzly grafu, k u nichž už jsme nalezli nejmenší cenu za přesun do daného uzlu z počátečního uzlu a druhá množina obsahuje ostatní vrcholy.

Algoritmus pak funguje následovně, pro každý vrchol v si pamatujeme délku nejkratší cesty, kterou jsme se do vrcholu dostali ($dist[v]$) a uzel, který byl na nejkratší cestě před uzlem v, tedy předchůdce uzlu v ($prev[v]$). Dále máme množinu Q (typicky se v implementaci pracuje s prioritní frontou, která bývá implementována jako halda), ze které budeme vybírat vždy hladově vrchol, který potencionálně vede k nejkratší cestě do cíle, do této množiny vložíme na počátku všechny vrcholy a postupně z ní budeme odebírat jeden vrchol po druhém. Na začátku všem vrcholům nastavíme vzdálenost na nekonečno a předchůdce na cestě na nedefinovanou hodnotu, s výjimkou počátečního vrcholu, kterému nastavíme vzdálenost na nulu. Blíže je algoritmus popsán pseudokódem 1 a je připojena i ilustrace průběhu algoritmu 4.3.

S pomocí Dijkstrova algoritmu si můžeme vytvořit jakékoliv hledání bude jenom zapotřebí, ať už budeme chtít optimalizovat cestu na čas, počet pře-

Obrázek 4.3: Ilustrace průběhu Dijkstrova algoritmu



Zdroj: Cyberlingo.blogspot.cz

stupů nebo třeba počet ujetých kilometrů, stačí nám jen upravit metriku, podle níž se vybírají uzly z množiny Q.

Algorithm 1 Pseudokód dijkstrova algoritmu

```
1:  procedure DIJKSTRA(GRAPH, SOURCE)
2:      for all vertex v in Graph do
3:          dist[v] ← infinity
4:          prev[v] ← undefined
5:      dist[Source] ← 0
6:      Q ← all vertices in Graph
7:      while Q is not empty do
8:          u ← vertex from Q with smallest dist remove u from Q
9:          for all neighbor v of u do
10:             if dist[u] + dist_between(u,v) < dist[v] then
11:                 dist[v] ← dist[u] + dist_between(u,v)
12:                 prev[v] ← u
```

Zde ukázaný pseudokód je sice funkčním řešením, ale obvykle se implementuje trochu jinak. Problém s tímto způsobem implementace je ten, že se na začátku běhu algoritmu vloží do množiny Q všechny uzly v grafu, což není vůbec nutné. V praxi se setkáme spíše s implementací, kde Q bude prioritní fronta, která bude obsahovat na začátku pouze jeden uzel s minimalizovanou vzdáleností, a to ten počáteční. Dalo by se říct, že v tomto případě vlastně Q reprezentuje doplňkovou množinu vrcholů, tedy ty, jejichž vzdálenost od uzlu, ze kterého jsme spustili vyhledávání nejkratší cesty, už je minimalizo-

vaná, i když toto tvrzení je poněkud liché, protože jsou uzly z prioritní fronty průběžně odebírány. V nejhorším případě sice paměťovou náročnost programu nezlepšíme, ale v praxi většinou nebudeme mít v Q všechny uzly, pokud tedy zapojíme další optimalizaci. V případě kdy hledáme nejkratší cestu z uzlu u do uzlu v a minimální vzdálenost do žádného jiného uzlu nás nezajímá, tak můžeme algoritmus ukončit ve chvíli, kdy z prioritní fronty Q vybereme uzel v, protože díky nezápornému ohodnocení hran víme, že už se rozhodně do uzlu v s menší vzdáleností nedostaneme.

Kapitola 5

Analýza a návrh

V této kapitole pojednávám o různých návrzích a nápadech, které je zapotřebí zvážit ještě před vytvořením finální aplikace.

5.1 Použité technologie

Původní záměr byl, že pojmout a implementovat celou aplikace jakožto webovou aplikaci, s využitím primárně PHP a MySQL na backendovou část a CSS ještě v kombinaci s javascriptovým frameworkem jQuery a knihovnou obsahující spoustu předpřipravených designových i funkčních vychytávek pro frontendovou část zvanou Bootstrap. Všechny zmiňované technologie mají poměrně volné licence, což z nich dělá velice rozšířené záležitosti ve světe vývoje webových aplikací.

Přičemž jsem vycházel z toho, že v rámci této aplikace potřebuji parsovat soubory do velikosti přibližně šedesáti megabajtů, což je cca velikost největšího z použitých souborů. Ovšem očekával jsem, že PHP bude s tímto trochu mít problém, PHP je přímo vyhlášené svojí neefektivitou a nemuselo by proto zvládat dobře práci při parsování CSV souborů z GTFS feedy. Proto jsem zkusil malý test a zjistil, že PHP při parsování cca šedesáti megabajtového souboru spotřebuje více než půl gigabajtu paměti (což byl nastavený paměťový limit skriptu a v důsledku jeho překročení skript spadl). Při dalším testu jsem zkusil zefektivnit parsování, aby PHP stačilo udržet si víceméně pouze výsledný graf, ale i zde PHP selhalo na celé čáře. Proto jsem se rozhodl od něj upustit a neimplementovat prototyp jako webovou aplikaci.

Po prozkoumání této slepé uličky padlo rozhodnutí, že prototyp vytvořím jako mobilní aplikaci. Vzhledem k tomu, že bylo potřeba vytvořit nějaké uživatelské rozhraní, tak jsem pátral a zjistil, že v úvahu připadají vlastně 2 možnosti, jednak použít asi nejčastější způsob vývoje aplikací pro android, a to využít Android studio a psát celou aplikaci v Javě, anebo vyzkoušet o něco méně probádanou cestu a použít Xamarin od Microsoftu a celou aplikaci psát v C#. První možnost by byla asi o něco jistější, jelikož Android studio používá

pro vývoj určitě řádově více vývojářů, a tak bych asi minimalizoval riziko, že narazím na problém, na který nikde na internetu nenaleznu odpověď. Nicméně na druhé variantě mi imponovalo jednak to, že je mi C# o něco bližší, než Java, a pak také to, že po napsání aplikace bude fungovat nejenom na Androidu, ale i na dalších operačních systémech, a to navíc bez jakýchkoli nutných úprav.

5.1.1 Xamarin

Xamarin je softwarová firma založená v roce 2011 vývojáři, kteří stáli za zrodem projektu Mono, což byl open source projekt, který se snažil umožnit spouštění kódu využívajícího .NET framework i na jiných platformách, než na Windows. Mono dalo vzniknout mnoha nástrojům užitečným pro vývoj v C# a dalších jazycích, které úzce spolupracují s .NET frameworkem, který odjakživa spadá pod Microsoft, ať už šlo o C# kompilátor, různé profilery anebo celý runtime jazyka. Ale co je důležité, Mono pokrývalo více platforem, nejenom Linux, jak si někteří myslí a jednou z těchto platforem byl právě operační systém Android.

Xamarin samotný už pak cílí právě na vývoj aplikací pro mobilní zařízení, kdy umožňuje sdílet zdrojový aplikace pro všechny tří hlavní mobilní platformy, tj. iOS, Android a Windows phone.

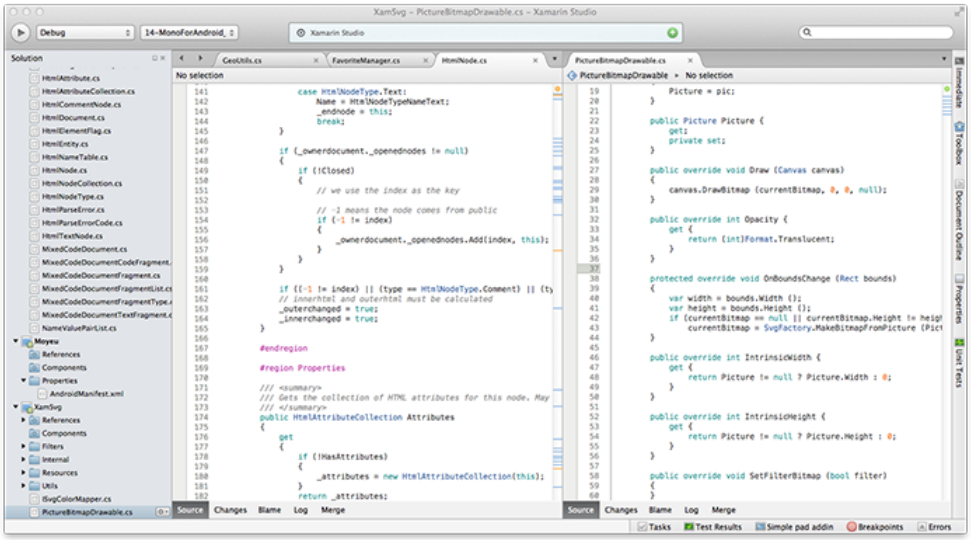
Xamarin nedodává pouze možnost psát mobilní aplikace v C#, ale poskytuje například i takzvané Xamarin Forms, což je další rozšíření, které vývojářům dává možnost sdílet napříč platformami nejenom výpočetní část kódu, ale i části GUI. Samozřejmě Xamarin Forms nepokrývá veškeré možnosti, kterých lze na jednotlivých platformách využít, ale i tak je užitečné, že může vývojář většinu kódu sdílet napříč platformami. A i pokud vývojář chce použít něco specifického pro jednotlivé platformy, tak mu to Xamarin usnadňuje. Xamarin má jednak vlastní IDE, *Xamarin Studio* (ukázka na obrázku 5.1), ale také existuje v podobě doplňku pro *Visual Studio* od Microsoftu. A v obou variantách se pro vytvoření nové aplikace vytvoří vlastně hned čtyři různé projekty, jeden je pro obecný kód, který bude sdílený pro všechny platformy a pak existuje separátní projekt pro jednotlivé hlavní mobilní platformy, a to Android, Windows a iOS.

5.1.2 XAML

Úvodem je nutné zmínit, proč zde zmiňuji další věc od Microsoftu, je to protože když chce někdo v Xamarinu vytvářet jednotné formuláře pro všechny hlavní mobilní platformy, tak na ně musí použít XAML, Xamarin se už následně postará o přeložení GUI prvků do nativní podoby jednotlivých platforem.

XAML (Extensible Application Markup Language) je deklarativní jazyk založený na XML, je silně využíván v .NETových technologiích, jmenovitě ve Windows Presentation Foundation (WPF), Silverlightu, Windows Workflow

Obrázek 5.1: Ukázka programu Xamarin Studio



Zdroj: Xamarin.com

Foundation (WF), Windows Runtime XAML Framework a Windows Store aplikacích.

XAML umožňuje vytvářet uživatelské rozhraní podobným stylem jako například v HTML, kdy si člověk pomocí textové notace nadefinuje rozložení prvků na monitoru a následně jim přiřkne různé akce, které lze s těmito prvky dělat. Tomuto provázání akcí a prvků se v XAMLové terminologii říká binding.

5.2 Navrhovaná vylepšení

Součástí této práce je i návrh několika vylepšení, především pro aplikace, které využívají GTFS data pro účely plánování přesunu z místa na místo za pomoci městské hromadné dopravy. Jakožto uživatel aplikací na vyhledání spojení jsem u nich v průběhu času začal shledávat různé nedostatky. Proto jsem se rozhodl, že v rámci své diplomové práce zkusím současnou situaci trochu vylepšit a přispět svou trochou do mlýna. V praxi byla představa taková, že se zkusím zamyslet nad tím, o co by se stávající aplikace daly ještě obohatit. A následně se pustím i do praktické implementace prototypu testovací aplikace, kde některé z návrhů zkusím využít a případně i doladit, pokud by původní návrh nebyl z nějakého důvodu v pořádku.

Snažil jsem se volit taková vylepšení, která by mohla zlepšit kvalitu služeb zákazníkům a která by byla dále aplikovatelná i v již existujících aplikacích. Některá z vylepšení už jak jsem zjistil v některých aplikacích existují, ale většinou by potřebovaly ještě trochu dotáhnout k dokonalosti. Samozřejmě bych rád, aby i testovací aplikace, která vznikne v rámci této práce mohla

posloužit jako odrazový můstek pro další vývojáře podobných aplikací a aby se z ní mohli inspirovat při tvorbě vlastních aplikací a dále je díky ní rozvíjet. V této práci nebudu implementovat všechna navrhovaná vylepšení, ale budu probírat tyto možnosti:

- Přidání pěší trasy do vyhledávání spojení.
- Vyhledání nejbližší zastávky podle GPS souřadnic.
- Vyhledání zastávky na základě názvu ulice.
- Doplňování zastávek do vyhledávání na základě předchozích hledání.
- Automatické doplnění možnosti pěšího přesunu mezi více zastávkami na základě jejich vzájemné vzdálenosti.
- Možnost efektivního přeplánování trasy v případě zpoždění.
- Sledování zpoždění na základě sledování GPS polohy.

5.2.1 Přidání pěší trasy

Většinou návaznost spojů nezpůsobuje velký problém, ale ve chvíli, kdy jsem začal hojně využívat nočních dopravních spojů, narazil jsem na drobný problém. Vezměme v úvahu například průměrného studenta fakulty informačních technologií, který bydlí na strahovských kolejích a vrací se skoro až k ránu domů.

Pokud si fiktivní student nechá vyhledat spojení na zastávku *Koleje Strahov*, může dostat časově velmi rozdílné výsledky, oproti situaci, kdy si do vyhledávání spojů zadá zastávky *Stadion Strahov* nebo *Malovanka*. Zpravidla dojde k situaci, kdy se člověk může rychle dostat například na stanici *Malovanka*, ale těsně před nosem mu ujede autobus směřující na stanici *Koleje Strahov*. Proto pak kvůli nižší frekvenci jízdy nočních autobusů musí student čekat, i když z hlediska studenta už není problém mezi stanicemi *Koleje Strahov* a *Malovanka* bez problému dojít pěšky.

Z hlediska plánovacího softwaru pak někdy dochází k určitým optimalizacím kvůli návaznosti jednotlivých spojů, což zapříčiní, že kýženou posloupnost dopravních spojů, které by studenta dostaly relativně rychle na stanici *Malovanka* program ani nezobrazí, jelikož ji vyhodnotí jako nevhodnou. A přitom by stačilo málo, buď podat studentovi kompletní informace, aby se mohl rozhodnout sám anebo v lepším případě mu poskytnout rozhraní, aby si sám mohl určit, v kterých částech města se natolik orientuje, aby byl schopný přesunu mezi zastávkami, aniž by musel využít dopravní prostředek.

Ukázka toho, jak rozdílně může takové plánování nočního návratu domů vypadat je vidět na obrázcích 5.2, 5.3 a 5.4. Zkusil jsem využít webové aplikace IDOS a zajímalo mě, jakou posloupnost dopravních spojení aplikace navrhne

Obrázek 5.2: Plánování trasy Ortenovo náměstí – Malovanka

Zdroj: IDOS.cz

pro shodné zadání, kdy jsem hledal dopravní spojení ve dvě hodiny v noci ze zastávky *Ortenovo náměstí* vždy na jednu z trojice zastávek *Malovanka*, *Koleje Strahov* a *Stadion Strahov*. Jak je z obrázků vidět, tak zdaleka nejlépe dopadla cesta ze zastávky *Ortenovo náměstí* na zastávku *Malovanka*, kdy uživatel musel na začátku 6 minut počkat a následně mu na překonání cesty stačilo dalších 35 minut (samozřejmě v případě, kdy vyhledané dopravní spoje neměly žádné zpoždění, ani poruchu a ani žádný jiný problém). Druhá nejlepší nalezená cesta byla na zastávku *Stadion Strahov*, zde opět uživatel čekal na první dopravní spoj v pořadí 6 minut, ovšem následně mu cesta do cíle zabrala 51 minut, tedy skoro celou hodinu a znatelně více, než tomu bylo v předchozím případě se zastávkou *Malovanka*. A v posledním případě, kdy jsem nechal vyhledat posloupnost dopravních spojení, která by mě dostala až do úplného cíle, tedy na zastávku *Koleje Strahov* dopadlo plánování z mého pohledu až katastrofálně. Nalezená dopravní spojení se fakticky shodovala s těmi, která byla nalezena při vyhledání spojení na zastávku *Stadion Strahov*, ovšem s tím rozdílem, že poslední kus cesty, tj. cca 200 metrů mezi zastávkami *Stadion Strahov* a *Koleje Strahov* obstará autobus, který jede až skoro o půl páté ráno. Z toho důvodu dojde k pozdržení už na počátku cesty a uživatel dostane doporučení, aby si počkal na první spoj téměř hodinu a půl.

Jako nejjednodušší řešení tohoto problému se jeví poskytnout uživateli možnost si přidat pomyslnou hranu do grafu, vyznačující možnost pěšího přesunu mezi zastávkami. Dále by se dalo s myšlenou pracovat a vylepšit ji například o automatické přidávání pomyslných hran mezi zastávky, které jsou od sebe například maximálně dvě stě metrů vzdušnou čarou. GTFS feedy běžně obsahují GPS souřadnice jednotlivých zastávek, takže není problém pomocí známých vzorců dovodit vzdálenost mezi více zastávkami.

5.2.2 Vyhledání nejbližší zastávky podle GPS souřadnic

Toto vylepšení už některé aplikace více či méně dobře implementují. Vzhledem k tomu, že ve veřejně dostupných GTFS feedech jsou dostupné GPS souřadnice jednotlivých zastávek, tak se problém dá převést na hledání nejbližšího bodu v rovině k bodu X, kde X je bod vyjadřující aktuální polohu uživatele. Jelikož je zastávek relativně málo, tak pak není problém spočítat vzdálenost

Obrázek 5.3: Plánování trasy Ortenovo náměstí – Koleje Strahov

Zdroj: IDOS.cz

Obrázek 5.4: Plánování trasy Ortenovo náměstí – Stadion Strahov

Zdroj: IDOS.cz

ke každé jednotlivé zastávce a vybrat tu zastávku, k níž je vzdálenost nejnižší. Vlastně největší implementační problém při tomto přístupu může být při zjišťování polohy uživatele, ale v dnešní době je už prakticky každý chytrý telefon vybaven GPS modulem, takže ani to nebývá problém. Pokud nelze polohu zjistit, ať už je to z důvodu toho, že je tato funkcionalita v daném zařízení deaktivovaná, není přítomná vůbec, nebo třeba jen nelze z jiného důvodu chytnout GPS signál, tak ji zkrátka nelze využít. Pak už je jediná možnost využít nějakých předchozích zjištěných poloh nebo hledání, což už je ale pouze heuristika, která nám nic negarantuje. U tohoto vylepšení tedy vlastně není nic moc zajímavého, ať už implementačně nebo myšlenkově.

5.2.3 Vyhledání zastávky na základě názvu ulice

Občas se stane, že uživatel ví, kam chce jet, ale nemá ani ponětí, jaké jsou v okolí zastávky, respektive netuší, jak se jmenuje zastávka kam chce dojet, zato ale ví kam chce následně dojít. Proto by se uživateli mohla hodit možnost, že by zadal namísto cílové zastávky pouze název ulice (rozšířením by klidně mohlo být, že by uživatel zadal i číslo popisné) a aplikace by mu sama vybrala zastávku, kam by měl dojet, tedy nějakou z těch nejbližších, samozřejmě i s ohledem na to, na kterou ze zastávek se dá dojet nejrychleji a s předpokládaným časem potřebným na překonání mezi zastávkou a požadovanou ulicí.

Osobně si myslím, že takováto funkcionalita může být velmi užitečná, obzvláště, když se uživatel musí pohybovat v nějaké části města, kde se nevyzná. Samo dohledání ulice na základě GPS souřadnic by se dalo realizovat díky volně dostupným datům. Takováto data se dají získat například z Open-

StreetMap projektu (`http://openstreetmap.org`), což je komunitní projekt, který se snaží sbírat různé informace o mapových podkladech a už se za posledních pár let výrazně rozrostl. OpenStreetMap je krásnou ukázkou toho, jak někdy komunita dokáže společným úsilím konkurovat i komerčním řešením, jako jsou Google Maps anebo třeba česká alternativa Mapy.cz. OpenStreetMap obsahuje nejrůznější informace od názvů ulic přes typ povrchu, který se v dané lokalitě nachází až po informace, které by se skutečně daly využít při plánování trasy, například vytíženost komunikace. Bohužel však má tento přístup tu nevýhodu, že by následně bylo nutné pro rozumné plánování spojů, mít v aplikaci možnost navigovat uživatele i po zemi, což by znamenalo výrazně vyšší, jak paměťové, tak i výpočetní nároky mobilní aplikace.

5.2.4 Doplňování zastávek do vyhledávání na základě předchozích hledání

Toto vylepšení obsahuje hodně různých aplikací, navíc není ani nijak náročné na implementaci. Ale i přesto si myslím, že by tato funkcionalita rozhodně neměla v žádné aplikaci chybět, leda by silně narušovala rozvržení uživatelského grafického rozhraní.

5.2.5 Automatické doplnění možnosti pěšího přesunu mezi zastávkami na základě jejich vzájemné vzdálenosti

Zde jde víceméně jen o rozšíření prvního vylepšení. Zatímco v prvním vylepšení uživatel dostane možnost přidávat si pěší hrany dle libosti, v tomto případě by byly hrany přidávány bez zásahu uživatele. V souvislosti s tím by mohly vznikat problémy, protože pouze na základě znalosti vzdálenosti mezi zastávkami je těžké posoudit, jak dlouho potrvá přechod mezi zastávkami, ty sice mohou být blízko, ale to, že napřed uživatel musí dojít ke kilometr vzdálenému nadchodu, protože mezi zastávkami je rušná čtyřproudová silnice, to už samotný stroj podchytí jen těžko. Proto bývá tato funkcionalita vázána na lidský faktor.

5.2.6 Možnost efektivního přeplánování trasy

Všichni jsme to zažili, čekal nás přestup, ale dopravní prostředek, ve kterém jsme se aktuálně nacházeli měl zpoždění, což způsobilo, že nám ujel navazující spoj.

Už jsem viděl v jedné aplikaci podobnou funkcionalitu, která ale byla trochu odlišná. Ta spočívala v tom, že si uživatel nechal vyhledat posloupnost dopravních spojení a pak (pozn. šlo o aplikaci pro Android) uživatel mohl přejet prstem po vybraném spoji a aplikace mu ukázala další spoj mezi stejnými zastávkami. Vylepšení, které bych já rád viděl v podobných aplikacích má jinou podobu.

Moje vize je taková, že by uživatel měl možnost zadat zpoždění konkrétního spoje, ve kterém se nachází. Na základě této informace by si aplikace dopočítala, kde by měl zrovna dopravní prostředek být a dopočítala by i další návazné spoje za předpokladu, že zpoždění zůstane konstantní. De facto by v uživatelském rozhraní pravděpodobně postačil nějaký posuvník, kterým by uživatel nastavil aktuální zpoždění a spustil přepočet.

5.2.7 Sledování zpoždění na základě sledování GPS polohy

V tomto případě jde vlastně o úpravu předchozího vylepšení, kdy by neměl být nutný zásah uživatele v uživatelském rozhraní, ale už na základě toho, že by si uživatel vyhledal spoj, tak by aplikace začala v periodických intervalech sledovat polohu uživatele a porovnávala by ji s předem známými trasami dopravních spojů.

Do jisté míry by se nechalo vycházet ze základních údajů, které nutně musí být obsažené v GTFS feedu, a to ze souřadnic jednotlivých zastávek. Bohužel, když nejsou dostupné informace o konkrétní trase, kterou dopravní prostředek projíždí, tak by bylo velice komplikované odhadovat, jak dlouho by měl dopravní prostředek ještě pokračovat, než dorazí k další zastávce, někdy totiž může například kvůli jednosměrné ulici, dělat větší objížďky.

Dalším problémem je, že někdy mohou některé spoje sdílet větší část trasy, takže by nejspíše byl stejně zapotřebí další zásah uživatele, aby alespoň aplikaci napověděl, pro kterou z navrhovaných variant cesty se rozhodl. Pak by se už dalo něco odhadovat i v případě, kdy feed obsahuje pouze GPS souřadnice zastávek a ne vyznačené celé trasy jednotlivých spojů

5.3 Reprezentace grafu v programu

Pro úspěšnou implementaci testovací aplikace je nutné promyslet i další implementační detaily, jako například co jak a proč reprezentovat. V tomto případě může být kritickým problémem způsob reprezentace grafu dopravní sítě, pokud nebudu data vhodně reprezentovat, může dojít k problému, že budou některé operace s grafem trvat příliš dlouhou dobu, nebo že budu u některé z funkcionalit muset použít zbytečně složitou implementaci. Proto to chce napřed dobře promyslet, které funkcionality, respektive jejich časová či paměťová náročnost, jsou pro mě nejdůležitější a reprezentaci grafu tomu přizpůsobit.

Také je dobré si uvědomit, že v tomto případě se jedná o graf orientovaný, tedy může existovat hrana z vrcholu u do vrcholu v, ale už nemusí existovat hrana z vrcholu v do vrcholu u. V reálném světě by se to dalo vysvětlit na příkladu, že normálně autobusová linka projíždí postupně zastávky u, v a w, v opačném směru je ta samá linka projíždí také, pouze v opačném pořadí. Ale jednoho dne přijdou kopáči, půlku ulice rozkopou a z toho důvodu musí autobusová linka v opačném změnit trasu a jet jinudy, což se vyřeší tak, že zkrátka

autobusová linka v opačném směru už bude stavět pouze na zastávkách w a u, anebo namísto zastávky v zastaví na úplně jiné zastávce, která je poblíž.

Z hlediska reprezentace grafu v paměti existují tři základní přístupy jak graf reprezentovat, a to:

- Seznam hran
- Matice sousednosti
- Seznam sousedů

Předně bych si měl určit, jaké nároky mám na reprezentaci grafu a podle čeho se tím pádem budu rozhodovat která z možných reprezentací je nejvhodnější a kterou tedy nakonec zvolím. Počítám s tím, že budu v grafu často vyhledávat, tedy často bude zapotřebí operace proiterování sousedů konkrétního uzlu, zatímco úpravy v grafu se nebudou provádět příliš často a ještě k tomu se budou provádět stylem, že si načtu GTFS feed do paměti, udělám potřebné úpravy, zase ho uložím a před hledáním si načtu GTFS feed, přičemž v průběhu celé doby, po kterou budu úpravy provádět bude k dispozici předchozí verze GTFS feedu, takže i kdyby úprava grafu trvala například 10 minut, tak by to nebyl vůbec problém. Předně tedy budu cílit na rychlost operace proiterování sousedů, dále je pro mě samozřejmě důležitá paměťová náročnost reprezentace, protože vysoká paměťová náročnost také nevyhnutelně způsobí nárůst časové náročnosti výsledného programu.

5.3.1 Seznam hran

U této reprezentace si program uloží do pole seznam všech hran v grafu, tedy de facto pro každou hranu si budu pamatovat dvojici vrcholů u a v, což mi bude říkat, že v grafu existuje hrana z vrcholu u, která vede do vrcholu v. Je žádoucí si pro každou hranu pamatovat i další informace, například jak dlouho trvá přesun z vrcholu u do vrcholu v po dané hraně. Ideálně si mohu ukládat hrany za sebe seřazené v pořadí podle uzlu, ze kterého hrana vychází. Ve své aplikaci totiž počítám s tím, že budu využívat algoritmus založený na Dijkstrově algoritmu a proto vím, že se bude hodit, když budu moct efektivně procházet všechny sousedy konkrétního uzlu, které při ukládání seřazených hran budu mít v řadě za sebou.

Výhodou této implementace je její malá paměťová náročnost v řídkých grafech, tedy v grafech, které obsahují poměrně málo hran vůči počtu uzlů.

Na druhou stranu, alespoň v základní verzi, kdy hrany v seznamu nejsou seřazené je tato reprezentace dosti nevhodná, jelikož operace zjištění přítomnosti hrany v grafu mezi uzly u a v a projití všech sousedů konkrétního uzlu potřebují asymptoticky $\mathcal{O}(|E|)$ operací. Pokud využijeme možnosti držet si hrany seřazené, tak můžeme nejprve za pomoci binárního vyhledávání [20] vymezit blok hran, které vycházejí z uzlu v a následně ho celý proiterovat, čímž snížíme

časovou složitost z počtu všech hran v grafu na $\mathcal{O}(\#soused\text{ů} + \log|E|)$, ale samozřejmě opět záleží na konkrétním použití, pokud bychom například v grafu vyhledávali jen občas, ale naopak ho často měnili, mohla by být původní verze rychlejší.

5.3.2 Matice sousednosti

Matice sousednosti je matice A o velikosti $n \times n$, tj. $A \in \mathbb{R}^{n,n}$, která si pamatuje pro každé dva vrcholy zda spolu sousedí. V nejjednodušších případech obsahuje matice pouze nuly a jedničky, přičemž jednička signalizuje existenci hrany a nula naopak její absenci v grafu. Ale v komplexnějších problémech si můžeme přímo v matici pamatovat cenu hrany, což může v praxi představovat jakoukoliv požadovanou metriku.

Prakticky si tedy musíme všechny uzly unikátně očíslovat čísly od 0 do $n-1$ včetně, matematicky řečeno, potřebujeme získat bijektivní zobrazení, které nám zobrazí uzel na číslo.

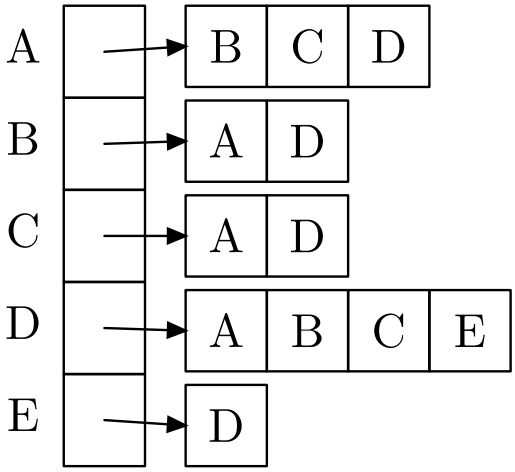
Pokud chceme zjistit, zda existuje v grafu hrana vedoucí z uzlu u do uzlu v, tak si musíme uzly u i v zobrazit na čísla, pro jednoduchost budou v následujícím textu u a v značit nikoliv uzly samotné, ale jejich přidělená čísla (pakliže nebude specifikováno jinak). Jinak řečeno musíme si uzly u a v namapovat na čísla a následně se podívat do matice A na prvek $A_{u,v}$ (programově zapsáno jako $A[u][v]$), je-li tento prvek nulový, nebo obsahuje-li nějakou hodnotu, která byla předem stanovena jako příznak značící neexistenci hrany, pak se v grafu nenachází hrana vedoucí z uzlu u do uzlu v, v opačném případě v grafu hrana vedoucí z uzlu u do uzlu v je.

V případě neorientovaného grafu se dá spolehnout na to, že $A_{u,v} = A_{v,u}$, protože se obousměrná hrana mezi uzly u a v reprezentuje jako dvě orientované hrany, jedna z uzlu u do uzlu v a druhá z uzlu v do uzlu u. Z toho plyne i to, že v případě neorientovaného grafu budeme mít symetrickou matici.

Výhodou tohoto způsobu reprezentace hran grafu je, že zvládneme efektivně zjistit, zda je nějaká hrana přítomna v grafu, protože nám stačí ověřit jediný prvek matice a časová složitost je tedy $\mathcal{O}(1)$.

Nevýhody této reprezentace hran tkví především v její paměťové náročnosti, která je kvadratická vůči počtu vrcholů v grafu, a to nezávisle na tom, zda je graf hustý anebo řídký. Obzvlášť v případě řídkého grafu, kde není příliš mnoho hran nám může tato paměťová složitost vadit, je to totiž důvod, proč už u relativně malých grafů, řekněme s řádově 10^5 uzly, nám nestačí operační paměť, program musí začít odkládat data na pevný disk, což vede k enormnímu zpomalení výpočtu. Nemluvě o tom, že se někdy může hodit mít více hran mezi stejnou dvojicí uzlů, na což tato reprezentace není dobře připravená.

Obrázek 5.5: Seznam sousedů



Zdroj: Algoritmy.eu

5.3.3 Seznam sousedů

Reprezentace hran v grafu seznamem sousedů je trochu podobná reprezentaci seznamem hran, nebo lépe řečeno navržené optimalizované verzi. Zde se sice také vyskytuje seznam hran, ale tento seznam je separátní pro každý jednotlivý uzel. Pokud tedy chci zjistit, zda vede nějaká hrana z uzlu u do uzlu v, tak musím prohledat seznam sousedů odpovídající uzlu u a zjistit, zda je v něm přítomen uzel v. Reprezentace by se dala znázornit nějak takto 5.5

De facto je tento přístup skoro stejný jako kdybychom vytvořili seznam hran, v němž jsou hrany seřazeny podle uzlu, z něhož hrana vychází a pak si jen přidali pole ukazatelů, kde bude ukazatel pro každý z uzlů a budeme tyto ukazatele využívat k označení počátku seznamu hran vycházejících z korespondujícího uzlu, lépe tuto myšlenku vyjádří názorný obrázek 5.6.

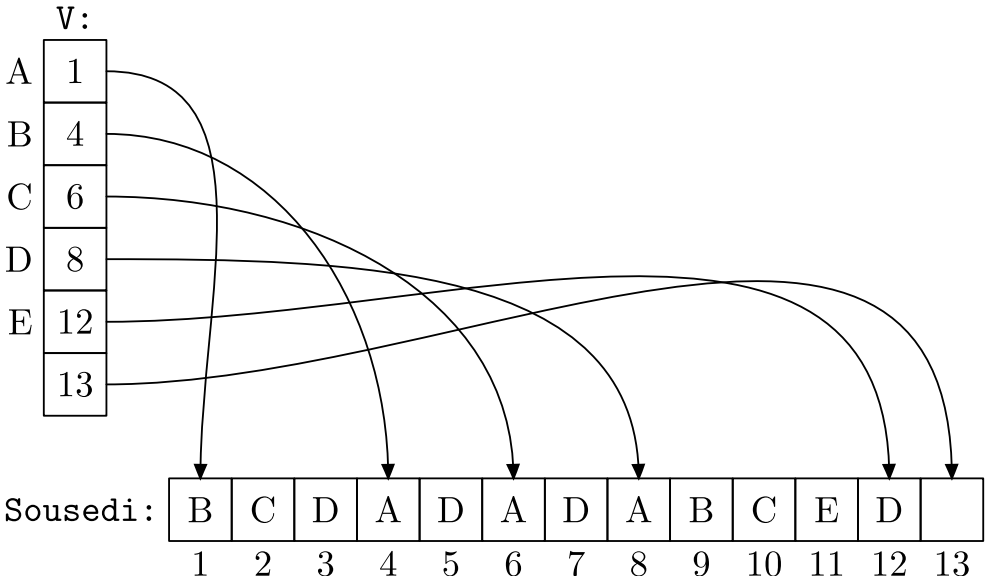
5.3.4 Porovnání implementací

Pro využití v testovací aplikaci je důležité především rychlé vyhledávání v grafu dopravní sítě, proto je důležité, aby bylo rychlé proiterování hran, které propojují jeden konkrétní uzel a všechny jeho sousedy. Tato podmínka, společně s faktem, že není dobré plýtvat pamětí, protože by se to výrazně podepsalo na výkonu ve chvíli, kdy bychom pracovali s většími grafy, vylučuje použití reprezentace grafu pomocí matice sousednosti.

A ze zbylých dvou možností se dá vyloučit reprezentace pomocí seznamu hran, i kdybych držel hrany seřazené, tak bych při expandování (tj. zpracování) konkrétního uzlu musel navíc vždy ještě pomocí binárního vyhledávání najít, kde začínají sousedé konkrétního uzlu, což je něco, co si můžu ušetřit.

Posledním faktorem už je pouze paměťová složitost, pokud si budu držet

Obrázek 5.6: Seřazený seznam hran



Zdroj: Algoritmy.eu

Tabulka 5.1: Srovnání různých reprezentací grafu

Reprezentace	Existuje hrana uv	Projití sousedů	Paměťová složitost
Seznam hran	$\mathcal{O}(\|E\|)$	$\mathcal{O}(\|E\|)$	$\mathcal{O}(\|E\|)$
Matice sousedů	$\mathcal{O}(1)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|V\|^2)$
Seznam sousedů	$\mathcal{O}(\#sousedů)$	$\mathcal{O}(\#sousedů)$	$\mathcal{O}(\|V\| + \|E\|)$

Pozn. E je množina hran a V je množina uzlů a časová náročnost některých operací se nechá trochu vylepšit, pokud si držíme seznamy seřazené.

dynamicky alokované pole jako seznam sousedů pro každý uzel, tak typicky se podobná dynamicky alokovaná pole realokují na dvojnásobek předchozí velikosti, takže typicky budu mít o něco větší paměťovou složitost, než bych potřeboval. V průměrném případě ale bude plýtvání pamětí o něco menší, pokud budu držet všechny hrany v jednom dynamicky alokovaném poli a pouze si někde bokem přidám pole ukazatelů. V této implementaci je důležité, aby byly hrany seřazené, takže jelikož se nemůžu spoléhat, že na vstupu dostanu hrany již seřazené, tak bych je musel pokaždé seřadit, anebo použít více oddělených seznamů, kde je mi jedno v jakém pořadí budu hrany držet. V těchto dvou případech už je volba silně závislá na tom, zda je limit pro paměť hodně nízký nebo zda-li je malé plýtvání snesitelné za cenu, že si ušetřím řazení hran. Dalo by se ještě silně diskutovat, že výkon implementace může vylepšit to, že je vhodnější pro využití cache paměti, ale to už záleží na tolika dalších faktorech, že tuto problematiku nebudu v této práci diskutovat.

Kapitola 6

Realizace

V následující kapitole se pokusím vylíčit, jak vznikala implementační část této práce a jaké provázely její vznik porodní bolesti. Kapitolu jsem rozdělil na různé části, podle toho k jakému implementačnímu celku patří, abych nemísil problémy, které nastaly při vzniku grafického uživatelského rozhraní s problémy, které mi komplikovaly práci na backendové části, kde jsem se zaobíral s prací s GTFS formátem, různými jeho úpravami a tak podobně.

6.1 Backend

Podle mě ta nejzajímavější část implementace byla backendová část, kdy jsem řešil zpracování dat v GTFS formátu, jejich další použití a samozřejmě i spoustu problémů, které v průběhu implementační fáze vyvstaly.

6.1.1 Zpracování GTFS dat

První velká komplikace, na kterou jsem při tvorbě implementační narazil byla ta, že jsem zatoužil po tom, že si usnadním práci. Toto rozhodnutí se záhy ukázalo jako ne příliš šťastné. Měl jsem za to, že není dobrý nápad, abych dělal celou implementaci ad-hoc, tedy abych psal všechno, včetně parsování GTFS souborů úplně od začátku a naivně jsem si myslel, že bude moudré zkusit rozšířit nebo alespoň využít nějakou existující open-source knihovnu. Říkal jsem si, že je zbytečné vymýšlet znovu kolo, a tak jsem začal s průzkumem existujících knihoven pracujících s GTFS feedy, které existují a jsou volně dostupné.

Jak se ukázalo, takových knihoven existuje celá řada, ale je s nimi spousta problémů. V drtivé většině případů totiž existuje velice tristní nebo ještě lépe, vůbec žádná dokumentace. Pod slovem tristní si lze představit dokumentaci jako poznámku, že v adresáři examples je ukázkový kód, ve kterém jsou některé funkce knihovny použity, a tak by ho měl případný uživatel prozkoumat. Až mě zarazilo, kolik projektů řeší dokumentaci takto a možná ještě více ji

pro jistotu neřeší vůbec. To je pro případného programátora, který by chtěl knihovnu využít dosti zatěžující, protože sám musí zjistit co knihovna vlastně vůbec umí a případně pak i jak to dělá, aby se tím programátorův kód nestal nepoužitelným z důvodu přílišné neefektivity kódu.

Většina knihoven, se kterými jsem se setkal nebyla co se týče funkcí kdovíjak obsáhlá a proto jsem se tak jako tak musel smířit s tím, že stejně i pokud použiji nějakou externí knihovnu, tak prakticky jediný problém, který mi odpadne, je parsování GTFS feedu. Alespoň tuto část implementace mi mohou externí knihovny ušetřit tím, že zvládnou GTFS feed dostat do paměti stroje a já už nemusím řešit alespoň tuto banalitu.

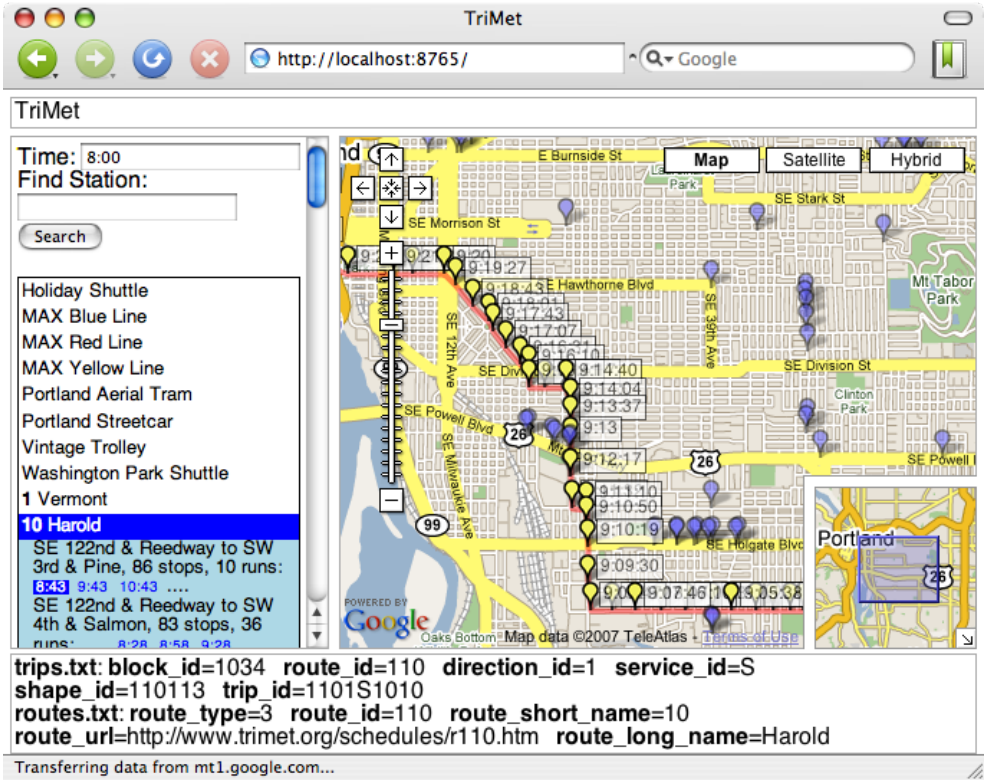
Tato představa se ukázala jako velice naivní, ve výsledku jsem vyzkoušel mnoho knihoven, pro několik programovacích jazyků, jmenovitě pro Python, Javu, C++ a C#. V jednom případě byla knihovna dokonce v takovém stavu, že ještě ani zdaleka nebyla schopna přečíst feed. V dalších případech byl občas skutečný oříšek rozchodit, ale nakonec jsem přiměl knihovnu k tomu, aby začala fungovat.

Co mě zarazilo, tak většina vyzkoušených knihoven měla naprosto tristní problémy s výkonem. Někde byla na vině nemožnost zakázání různých validací GTFS feedu, které někdy byly implementovány neefektivním způsobem a celkové tak celou práci dost brzdily. Vlastně jsem narazil na jedinou knihovnu, která měla rozumné rychlostní výsledky. Už jsem měl opravdu docela strach, že si stejně ve výsledku budu muset napsat vlastní knihovnu od píky. Shodou okolností se to týká knihoven, které byly napsané v Pythonu a v Javě, které nepatří k nejrychlejším jazykům, nicméně tentokrát nebyly jazyky na vině, protože i pouhé načtení cca osmdesáti megabajtového GTFS feedu v různých knihovnách trvalo řádově minuty.

Vůbec nejhorší časové výsledky poskytovala knihovna *transitfeed* pro Python 2, kterou vyvinul Google. Na, řekl bych, ne úplně pomalém stroji, se GTFS feed načítal cca 6 minut (a to s vypnutými extra validacemi), což mi přišlo velice pomalé. Na druhou stranu, knihovna *transitfeed* je zamýšlená primárně jako validátor GTFS feedů a při jejím využití je to dost znát, obzvlášť pokud si programátor zvolí při načítání GTFS feedu pomocí příznaku, že chce použít extra validace. V základu knihovna hlásí například nesrovnalosti v názvech zastávek, že tu a tam není feed dokonalý a obsahuje v názvu zastávky nějaký extra netisknutelný znak, nebo podobné problémy. Po zapnutí validace knihovna už i hlídá, že aby řidič dodržel jízdní řád, tak by mezi dvojicí zastávek musel mít příliš vysokou průměrnou rychlost (v použitém GTFS feedu mi knihovna zahlásila, že nutná průměrná rychlost je mezi některými zastávkami i přes 120 km/h, a to ještě musíme vzít v potaz, že knihovna musí vycházet pouze z GPS souřadnic, tj. vzdáleností mezi dvojicí zastávek). Už jen z toho by se dalo usoudit, že knihovna je míněná jako validační pomůcka a že to co má dělat dělá dobře.

Knihovna *transitfeed* by za jiných okolností byla celkem jednoznačně nejlepší volbou, protože patrně jako jediná měla nějakou dokumentaci, sice byl

Obrázek 6.1: Schedule Viewer



Zdroj: Github.com

problém se k ní proklikat, ale alespoň nějaká dokumentace existovala, což byla oproti ostatním knihovnám obrovská výhoda. Navíc Google disponoval i dalšími pomocnými utilitami pro práci s GTFS feedy, například je dostupná aplikace, zvaná Schedule Viewer (ukázka je na obrázku 6.1), která dokáže GTFS feed zobrazit na mapě, to je sice asi tak jediné, co aplikace umí, ale i tak to potěší.

Nakonec jsem však skončil s knihovnou jednoduše nazvanou *GTFS* napsanou v C#, což se mi hodilo. Zmíněná knihovna je dostupná na adrese `https://github.com/itinero/GTFS`. V průběhu práce s knihovnou jsem přicházel na některé její nedostatky, v době psaní práce například knihovna měla i problém načíst GTFS feed, který sama uložila na disk. Na vině bylo například chybné obalování údajů v CSV souboru uvozovkami. První práce tedy spočívala v opravě podobných chyb a přidání několika optimalizací, které práci s knihovnou zrychlily.

Knihovna je velice minimalistická, ale prakticky jsem od ní ani nechtěl více, než aby zvládla načíst a uložit GTFS feed na disk. Navíc díky tomu, že je knihovna tak minimalistická, tak skoro ani nevadilo, ze knihovna nemá žádnou propracovanou dokumentaci, a pak když chce vývojář mít bleskurychlou

aplikaci, tak mu zpravidla nezbývá, než některé věci udělat ručně.

6.1.2 Úprava GTFS dat

Další věc, se kterou jsem si musel v implementační části poradit byla, jak budu realizovat úpravy vstupních dat, které byly vyžadovány zadáním práce. Jelikož jde spíše o pouhý prototyp aplikace, tak jsem se především snažil zjednodušit implementaci, a to i za cenu, že přinese jisté zpomalení při práci s grafem. Jediná implementační část práce, kde jsem dbal na rychlost bylo vyhledávání spojení v dopravní síti, jednak je to akce, kterou uživatel bude dělat nejčastěji, a pak u něj mu bude nejvíce vadit, pokud bude program pomalý, i kdyby přidání pěší hrany trvalo několik sekund, tak se to dá strávit, protože to nejspíše uživatel udělá jednorázově.

Jmenovitě to byly:

- Přesun zastávek
- Slučování zastávek
- Přidávání pěších hran
- Vyhledávání spojení
- Vyhledávání spojení z nejbližší zastávky podle jména ulice
- Přidávání výluk
- Přidávání mimořádných spojů

6.1.2.1 Přesun zastávek

V tomto případě nebylo zapotřebí žádné vymýšlení komplexního algoritmu, jak danou věc naimplementovat, vlastně jde jen o to, že musím identifikovat konkrétní zastávku, vzít její GPS souřadnice a nahradit je novými. Prakticky tedy jen postup, že za pomoci knihovny načtu údaje o zastávkách do paměti, dohledám zastávku, kterou potřebuji, nechám uživatele zadat nové GPS souřadnice, přepíšu souřadnice zastávky v paměti a nakonec data zase s pomocí knihovny zapíšu na disk, abych o změněná data nepřišel.

Největší problém tedy v tomto případě bylo vytvoření pomocného grafického uživatelského rozhraní, které by uživateli umožnilo dohledat konkrétní zastávku a pozměnit u ní potřebná data, což v tomto případě znamená pouze GPS souřadnice.

Rozhodl jsem se tedy vytvořit jakýsi výběr ze seznamu zastávek, kterým by si uživatel mohl určit, která zastávka ho přesně zajímá. A aby uživatel nemusel vyhledávat zastávku třeba v dropdown menu, rozhodl jsem se zapojit do akce jakousi formu našeptávače. To byl trochu větší problém, než by nutně musel být, pakliže by bylo unikátní jméno zastávky, pak by nebylo nutné

jakkoliv komplikovaně zasahovat do klasického konceptu našeptávače, stačilo by napsat začátek názvu zastávky a bylo by hotovo. Takto jsem musel jít o krok dále a promyslet jak vhodně odlišit jednotlivé zastávky při zobrazování výsledků vyhledávání. Z pohledu programátora by bylo nejlepší zobrazovat pouze unikátní identifikátor zastávky, tj. stop_id z GTFS feedu, jenže to není ani zdaleka vhodné pro uživatele, protože ten podle stop_id nebude absolutně tušit o jakou zastávku se jedná. Nakonec jsem se rozhodl pro kombinaci GPS souřadnic a stop_id, které jsem k GPS souřadnic přidal tak pro efekt, kdyby někdo omylem nastavil více stejnojmenných zastávek na stejné souřadnice. To je ale extrémní případ, který bez asistence uživatele nenastane.

Poté už se jen uživateli zobrazí možnost nastavení údajů o zastávce, kde může GPS souřadnice změnit a případně provedené změny i uložit kliknutím na tlačítko. A následně může uživatel nechat změny trvale zapsat na disk.

6.1.2.2 Slučování zastávek

Slučování zastávek je takový poměrně netriviální problém, ke kterému by mohlo dojít například kvůli stavebním pracím, kdy před začátkem prací existovaly dvě zastávky se stejným názvem, které stály blízko sebe, ale dočasně byly sloučeny, protože na jedné z nich probíhají stavební práce. V pražské hromadné dopravě by mě jako takový případ napadala zastávka *Malovanka*, kam v době psaní práce jezdí ze zastávky *Koleje Strahov* dva různé autobusy, a to 143 a 149. Pokud by jedna ze zastávek byla dočasně zrušena, tak jak by bylo nejlepší to vyřešit?

Patrně nejjednodušší řešení, jak sloučit dvě různé zastávky, nebo alespoň implementačně nejjednodušší, na které jsem přišel bylo, že oběma zastávkám nastavím stejné GPS souřadnice (což de facto ani není nutné) a následně pouze přidám v grafu hrany pro pěší chůzi, viz další bod. Takto zkrátka při vyhledání dopravního spojení uvažuji tak, že jsem schopen z jedné zastávky na druhou dojít hodně rychle, třeba za minutu, nebo možná i za 0 minut, což je vlastně také zbytečné v případě, kdy se zastávky jmenují stejně, protože je pro to uzpůsoben vyhledávací algoritmus. Implementačně nejjednodušší je toto řešení díky tomu, že jsem v něm využil pouze další funkcionality, které jsem stejně tak jako tak musel implementovat. Aneb když už mám něco hotové, vím o tom, že to funguje dobře a není nějaký dobrý důvod proč bych to nemohl použít znovu, tak proč už implementovanou funkcionalitu nevyužít.

První řešení počítalo s variantou, že se zastávky spojí pouze dočasně, kdyby bylo předem jasné, že se zastávky spojí trvale, tak by bylo asi čistší řešení proiterovat všechny výskyty identifikátoru jedné ze zastávek v souboru stop_times.txt (soubor udává pro každý dopravní spoj pořadí a čas, kdy zastavuje u jednotlivých zastávek) a přepsat je identifikátorem druhé zastávky, respektive k tomu využít výše zmíněnou knihovnu pro práci s GTFS feedem.

Tabulka 6.1: Přehled typů dopravy

route_type	Typ dopravy	Poznámka
0	Tramvaj	Nepostihuje jenom tramvaje, někde může implikovat i podobné způsoby dopravy, například jednokolejku.
1	Metro	Značí podzemní dopravu.
2	Vlak	Značí nejenom dopravu na dlouhé vzdálenosti, může i jít i o dopravu v rámci města.
3	Autobus	Značí nejenom dopravu na dlouhé vzdálenosti, může i jít i o dopravu v rámci města.
4	Trajekt	Značí nejenom dopravu na dlouhé vzdálenosti, může i jít i o dopravu v rámci města.
5	Lanovka	Pozemní lanovka, u níž kabely vedou pod kabinou lanovky.
6	Lanovka	Typická lanovka, kdy je kabina lanovky ve vzduchu.
7	Lanovka	Jakýkoli jiný lanový systém navržený pro prudké stoupání.

Pozn. některé názvy nebylo možné vhodně přeložit z angličtiny, proto přidávám odkaz na původní znění `https://developers.google.com/transit/gtfs/reference#routestxt`.

6.1.2.3 Přidávání pěších hran

Jelikož jde v této práci pouze o vývoj prototypu aplikace, tak jak už jsem zmínil dříve, snažil jsem se upřednostňovat jednoduchost a obecnost implementace na úkor efektivity a paměťové náročnosti aplikace. Proto je v některých případech implementace řešena ne úplně nejefektivnějším způsobem, jak by bylo daný problém možné řešit, de facto jsem chtěl jen otestovat jak bude výsledná aplikace fungovat a proto jsem se řídil přístupem „Neoptimalizuj, dokud to nefunguje“.

Pěší chůze je poměrně specifická a hodně se liší od ostatních způsobů dopravy, především v tom ohledu, že člověk nemusí čekat na příjezd nějakého dopravního prostředku, ale může vyrazit kdykoliv po tom, co dorazí na místo startu svojí pěší trasy. Tyto rozdíly jsou znát i ve specifikaci GTFS formátu, soubor routes.txt totiž i atribut route_type, který se používá k odlišení více způsobů dopravy, ale v základní verzi formátu povoluje pouze hodnoty obsažené v tabulce.

Odlišnost pěší chůze od ostatních způsobů dopravy mě tedy stavěla před rozhodnutí, jak ji implementovat. Určitě jsem věděl, že se chci co nejvíce držet formátu GTFS, což mě přivedlo na myšlenku, že zavedu do feedu nového dopravce, který bude reprezentovat pěší chůzi a budu v rámci něj moci spra-

vovat veškeré hrany grafu pro pěší chůzi. Podpora pro více dopravců už byla implementovaná v použité knihovně, takže toto rozhodnutí mi v ničem neuškodilo. Zrovna tak jsem po lehkém zamyšlení rozhodl, že jednotlivé hrany pro pěší chůzi by se daly pro změnu namodelovat jako dopravní linka, potřebuji totiž nějak určit mezi kterými zastávkami hrana vede, to dopravní linka zvládne a opět jde o funkcionalitu, která byla již implementována v použité knihovně. Další zvláštností by bylo, že hrana reprezentující pěší chůzi by najednou mohla být obousměrná, ale to už je pouze implementační detail, že se při přidání hrany rovnou vytvoří dvě. Respektive, obousměrná hrana má sloužit jako berlička pro uživatele, aby nemusel vždy zadávat každou hranu 2x, i když by někdy dávalo smysl mít pro různé směry různé časy cesty, ku příkladu cesta z kopce oproti cestě do kopce.

Poslední část implementace, tedy už samotné hrany, kde obvykle mají dopravní prostředky nějaký čas příjezdu a odjezdu už ale bohužel skýtá nějaké implementační problémy, které už za mě nemůže použitá knihovna vyřešit. Zde už jsem musel něco implementovat sám a nemohl jsem jenom chytře delegovat práci na externí knihovnu.

Celkově mě napadly dva různé přístupy, jak se k implementaci postavit, první způsob a ten, řekněme, méně zajímavý a silně neefektivní obnášel to, že hranu pro pěší chůzi budu reprezentovat jako 144 různých hran, kdy bude jako čas odjezdu použitá každá minuta dne od *00.00* až do *23.59* a samozřejmě čas příjezdu na další zastávku bude čas odjezdu plus předpokládaný čas potřebný na pěší přesun mezi dvěma zastávkami. Výhody tohoto přístupu leží v tom, že pokud bych měl už nějaký hotový kód pro vyhledávání spojení, tak bych ho nemusel nijak upravovat a pokud bych tímto způsobem přidal nějakou pěší hranu, tak by kód stále korektně fungoval a zvládl by vyhledávat i za použití nově přidané hrany. Oproti tomu jsou nevýhody tohoto přístupu jednak vyšší paměťová náročnost a sekundárně i zpomalení vyhledávání, protože se v uzlu grafu snažím expandovat všechny hrany, což by při takto vysoké duplikaci a vyšším počtu pěších hran mohl být velice znatelný rozdíl.

Dala by se ještě zapojit variace na první přístup implementace, která je vlastně spíše jen paměťovou optimalizací. Jednak se v GTFS feedu vyskytuje soubor obsahující jednotlivé časy příjezdů a odjezdů, jak je zmíněno v předchozím odstavci, ale nepovinnou součástí GTFS feedu je i soubor *frequencies.txt*, kde je možné vyhnout se explicitnímu vyjmenovávání každého jednotlivého času příjezdu a odjezdu a namísto toho určit časový interval, jak často spoj jezdí a určit kdy tento interval probíhá, tj. mohl bych například říci, že mezi druhou a třetí hodinou odpoledne jezdí dopravní spoj každé dvě minuty a ušetřil bych si tak specifikování šedesáti různých časů příjezdů, potažmo odjezdů.

Druhým přístupem je zavedení dalšího typu hrany, kdy je pro efektivní vyhledávání asi nejlepší přístup držet si v každém uzlu dva seznamy hran, jeden seznam pro klasické hrany, reprezentující dopravní prostředky, vázané na konkrétní čas odjezdu a druhý seznam pro uskladnění pěších hran, u kte-

rých znám de facto namísto času odjezdu a příjezdu pouze délku trvání cesty. Tento přístup znamená jednak malou úpravu vyhledávání, kdy se musí při zpracování každého uzlu expandovat dva seznamy hran, dále je nutné upravit parser, aby začal rozpoznávat nový typ hran a nestěžoval si na nepovolenou hodnotu route_type a v neposlední řadě je nutné upravit tak zvaný loader, aby začal plnit při načítání GTFS feedu i druhý seznam hran. Nicméně testovací GTFS feed je natolik malý, že nevadí ani když budou existovat všechny typy hran v jednom seznamu a rychlost prohledávání grafu dopravní sítě tím nijak neutrpí.

V zásadě oba přístupy vyžadují komplexní implementační zásahy, první přístup bude možná o něco málo horší co se týče úpravy grafu, druhý přístup zase co se týče načítání uzlů do paměti a následného vyhledávání v přečteném grafu. V případě, že by knihovna podporovala nějaké vyhledávání v dopravní síti, tak bych neváhal a zvolil první možnost, ta by totiž už ihned fungovala prakticky bez nutnosti dělat do implementace jakékoliv zásahy. Možná, že bych byl nucen řešit různé speciální případy při rozpoznávání zda jde o pěší hranu nebo ne, ale to už se nechá zařídit jednoduše tím, že budu mít rezervovaný název dopravce nebo linky, který bude zaštiťovat pěší trasy. Řekněme, že není takový problém, aby název každé linky, která reprezentuje pěší hranu začínal například „PeskobusXXX“. Není to úplně ideální způsob, kdy se vývojář spoléhá na to, že dopravce nikdy nebude potřebovat pojmenovat zastávku nějakým nesmyslným názvem, ale docela často se s takovými přístupy je možno setkat i v komerčních softwarových produktech, a to nejenom ve vývojových verzích. Spousta vývojářů si totiž takto snaží ulehčit práci.

Takže tedy kolem a kolem, první výstup je vhodnější z pohledu programátora, protože může být o něco málo snazší na implementaci, tedy alespoň v některých případech, a oproti tomu druhý způsob implementace je vhodnější zase pro uživatele, protože vyhledávání je tak efektivnější. Při implementaci jsem se rozhodl pro druhou možnost, kdy budu mít speciální typ hran vyhrazený pro pěší hrany. Kvůli tomuto přístupu jsem tedy musel GTFS knihovně rozšířit některé moduly, aby zvládla s novým rozšířením GTFS formátu pracovat. Jinak byl druhý přístup tou více žádoucí volbou z důvodů, že oba přístupy už samy o sobě vyžadovaly razantní zásah do existující implementace knihovny, a tak bylo nakonec nutné rozhodnout podle sekundárního kritéria, efektivity.

6.1.3 Vyhledávání spojení

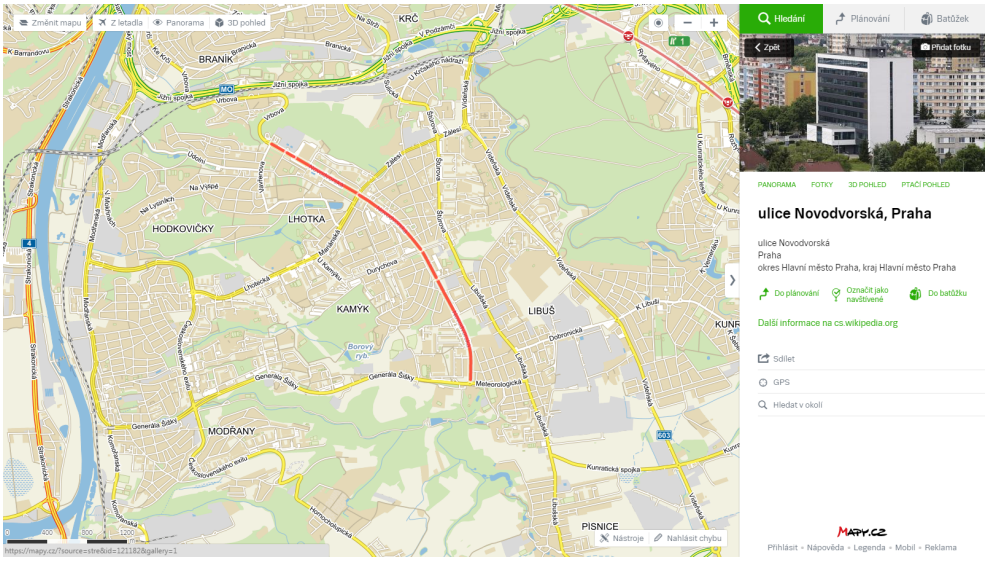
Asi nejzajímavější část implementace bylo samotné naimplementování vyhledávání v jízdních řádech. Musel jsem samozřejmě prozkoumat jakým způsobem si knihovna *GTFS* ukládá data o spojích, zastávkách atp. a dále jsem se z toho musel snažit nějak vycházet, abych si nepsal vlastní reprezentaci grafu, kde bych veškerá data zkopíroval, pak by využití knihovny nemělo takový význam.

Bohužel jak jsem zjistil, tak toto byl jeden z kamenů úrazu použité knihovny, kde zjevně autor při jejím návrhu nepočítal s tím, že by v ní chtěl někdo někdy efektivně vyhledávat data. Proto jsem musel zařídit pro efektivní vyhledávání spojů možnost přidání vyhledávání v datech za pomocí indexů. K tomu mi v C# posloužil vestavěný datový typ *Dictionary*, což je v C# ekvivalent hashovací tabulky, proto vyhledání prvku trvá průměrně $\Theta(1)$ [21]. Ve výsledku se mi podařilo úspěšně vyhnout tomu, abych si někam bokem zkopíroval celý graf do úplně jiné podoby, která by neměla s původním GTFS formátem nic společného. Ve výsledku jen místy vypadá vyhledávací algoritmus trochu odlišně od klasické reprezentace původní verze algoritmu, protože procházení grafu je komplikovanější v tom ohledu, že nestačí jenom projet seznam sousedů uzlu, ale musím si seznam sousedů vydedukovat z GTFS dat.

Samotné vyhledávání pak vychází z Dijkstrova algoritmu s heuristikou (tzn. A* vyhledávání), která zohledňuje vzdálenost mezi cílem a expandovaným uzlem. Vyhledávání funguje tím stylem, že se z dat ze souboru stop_times.txt dohledá, kdy jaký spoj projíždí touto zastávkou a u dalších zastávek, které ten samý spoj projíždí, si poznamenám, že se do nich dostanu za čas rovný době čekání na příjezd spoje navýšený o čas potřebný k přejezdu spoje z počáteční zastávky. Zde by stálo i za zmínku, že pro potřeby vyhledávání jsou všechny zastávky se stejným názvem chápány jako jedna a ta samá zastávka, není to sice úplně korektní přístup, ale je to chování, které uživatel intuitivně očekává. Implementaci algoritmu jsem se snažil nechat co nejobecnější, přeci jenom jde pouze o testovací aplikaci, a tak nemá úplně smysl ji přehnaně optimalizovat, obzvlášť ne pro konkrétní platformu, absolutně nemá smysl ji optimalizovat jinak než algoritmicky. Proto obsahuje pouze základní optimalizace, například kdybych vyrážel ze stanice *Dejvická* a mohl vyrazit metrem v 10.05 směrem ke stanici *Nemocnice Motol*, tak už nemá smysl uvažovat, že vyrazím stejnou linkou ve stejném směru o 5 minut později, v takovém případě můžu uživateli doporučit, aby oněch 5 minut počkal jinde a není důvod takto znatelně zpomalovat vyhledávání spoje.

Heuristika by měla v ideálním případě zařídit, že se budou jako první expandovat uzly, které jsou po cestě k cíli, což by v praxi mělo přinést znatelné zrychlení. Samozřejmě se dá dosáhnout ještě většího zrychlení, a to tak, že si předpočítám výsledky, ideálně pro každou minutu dne z každé počáteční zastávky do každé jiné zastávky. To by s sebou sice přinášelo kvadratickou paměťovou složitost, ale v reálné aplikaci by to přineslo kýžené zrychlení, nakolik je rychlejší pouze vytáhnout z paměti výsledek, než ho znovu pracně počítat. V rámci testování aplikace jsem nic takového nedělal, to by naopak spíše znehodnotilo ostatní vylepšení, kdy jsem si pro vyšší výkon upravil reprezentaci grafu v paměti a samozřejmě jsem využil i další implementační triky, kdy například ukončuji vyhledávání v momentě, kdy se vyhledávání dostanu do cílového uzlu. Jinak bych totiž zbytečně počítal vzdálenost všech uzlů od počátečního uzlu, i přestože mě zajímá pouze vzdálenost jediného konkrétního uzlu.

Obrázek 6.2: Ukázka vyhledání podle názvu ulice



Zdroj: Mapy.cz

6.1.4 Vyhledávání spojení z nejbližší zastávky podle jména ulice

Toto byla zamýšlená funkcionalita, která by měla svoje využití, protože ne vždy je možné buď využít GPS nebo vydedukovat z názvu zastávky, kde ve městě se zastávka vyskytuje. Nicméně jsem došel k závěru, že byť by pro danou funkcionalitu byla dostupná data (viz například ukázka ze serveru Mapy.cz na obrázku 6.2), tak bez možnosti nechat se aplikací navigovat i při pěší chůzi, ztrácí funkcionalita z větší části svůj smysl, protože by uživatel musel dále zjišťovat, jak se dostane ze zastávky do zvolené ulice.

6.1.5 Přidávání výluk

K implementaci tohoto problému jsem se, ostatně jako vždycky, snažil postupovat analyticky, přeci jen je dobré si problém napřed v klidu zanalyzovat a zhodnotit, co z už existující knihovny bych mohl využít. Konec konců už samotný GTFS formát má pro podobnou funkcionalitu zabudovaný mechanismus, kdy je možné nadefinovat nějakou dopravní službu, říct o ní kdy a jak funguje a pak například na státní svátky nebo podobné výjimečné dny nastavit v GTFS feedu do souboru calendar_dates.txt výjimku, že v den kdy by měla fungovat nebude fungovat a naopak. Jinými slovy říct, že služba funguje takto, ale v konkrétní dny je to jinak.

Pak už záleží na konkrétním zpracování feedu, v tomto případě nás budou zajímat data v souborech trips.txt, calendar.txt a calendar_dates.txt. Když se na problém podíváme hierarchicky, tak v souboru routes.txt jsou defino-

vány jednotlivé dopravní linky a jejich konečné stanice, tj. například, že linka metra *A* jezdí mezi stanicemi *Depo Hostivař* a *Nemocnice Motol*. K těmto údajům nám následně soubor trips.txt, respektive v kombinaci se souborem stop_times.txt, přidává informace o odjezdech jenotlivých dopravních stojů, tedy, že metro *A* odjíždí ze stanice *Depo Hostivař* v 10.05, 10.09 a tak podobně. Co nás bude zajímat více je, že je v souboru trips.txt na trip navázáno i service_id, přes které je trip vázán na soubor calendar.txt, potažmo calendar_dates.txt. GTFS formát umožňuje nastavit výjimky pouze na úrovni celého dne, proto by se mohlo stát, že když budeme chtít zakázat za den pouze jeden spoj, tak intuitivně by šel problém vyřešit tak, že koukneme na service_id a přidáme pro něj záznam do calendar_dates.txt, že ten a ten den služba nebude v provozu. Problém je, že v mnoha feedech bude jedno service_id využíváno více tripy a mohli bychom si tak přidat i další nechtěnou výluku.

Pokud by nastal případ, že bude více tripů sdílet jedno service_id, tak máme vlastně dvě různé varianty, jak tento problém vyřešit, jedna je, že si vytvoříme novou dopravní linku, která bude mít jako denní plán podmnožinu denního plánu původní linky, respektive bychom smazali spoje, které by pro nás nebyly žádoucí a následně bychom řekli, že tato linka bude fungovat v námi požadovaný den, zatímco původní linka by v tento den nefungovala. Tento způsob řešení je však relativně implementačně náročný, a proto jsem se rozhodl jít cestou, kdy vždy při editaci tripu vygeneruji unikátní service_id a následně mu přidám záznam do calendar.txt, který vznikne jako kopie původního záznamu v calendar.txt, který se k tripu vázal a přidám záznam calendar_dates.txt, že tehdy a tehdy nefunguje.

6.1.6 Přidávání mimořádných spojů

Ač se to nemusí na první pohled zdát, tak i přidání mimořádného spoje je velice podobný problém jako přidávání výluk. Vlastně jediný rozdíl je, že tentokrát k problému přistupujeme tak nějak z druhé strany. V tomto případě vytvoříme novou dopravní linku (tentokrát můžeme chtít i, aby nový spoj jezdil po nějaké trase, kde nejezdí žádný již existující spoj), která bude obsahovat jediný dopravní spoj a nastavíme ji tak, že bude fungovat pouze v ty dny, kdy chceme mimořádný spoj přidat, a tak jsem získal možnost jak přidat mimořádný spoj, aniž bych musel zbytečně přehnaně zasahovat do již naimplementované knihovny. Je pouze škoda, že tato minimalistická knihovna nepodporuje podobné akce, ale slouží opravdu jen jako mezivrstva, která umožňuje převádět GTFS feed mezi persistentní a operační pamětí stroje.

6.2 Frontend

Jak bylo zmíněno už v části s analýzou tak na tvorbu grafického uživatelského rozhraní jsem použil prostředky Xamarinu, potažmo XAMLu, ve kterém se

vytváří GUI pro mobilní aplikace postavené na Xamarinu. Nakonec GUI není až tak obsáhlé a náročné, jak by se na podobnou aplikaci slušelo, například by bylo hezké kdyby bylo možné zobrazit si jednotlivé zastávky na mapě a přesouvat je pouhým přetažením po displeji mobilního telefonu (nebo myší v případě počítače), to však aplikace v současné verzi nepodporuje, je znát, že se jedná o pouhý prototyp. Také je důležité si nechat nějaké možnosti rozšíření pro budoucí práci.

Při tvorbě frontendové části aplikace jsem možná využíval pomoci vyhledávače a serverů jako například stack overflow daleko více, než jinak v rámci tvorby celé práce, a to i přesto, že jsem s XAMLem už dříve něco dělal. Nicméně má XAML spoustu různých omezeni, a tak jsem vždy musel zkoumat jak daný problém obejít, abych zbytečně neplýtval časem na vymýšlení kola.

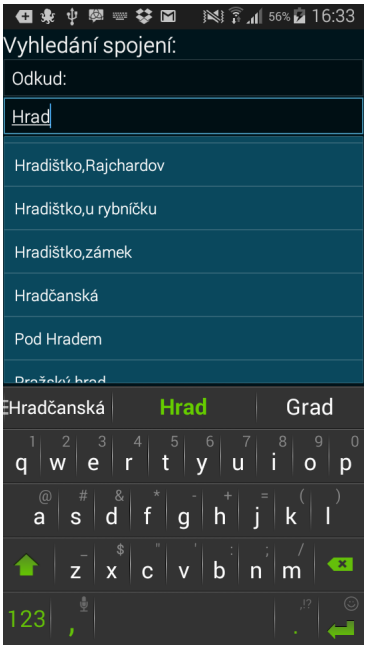
Jako neocenitelné se tedy ukázaly různé návody, jak Xamarin s XAMLem používat a podobně. Dál to samozřejmě chtělo i dávku vlastní iniciativy, abych si mohl rozmyslet, jak grafické uživatelské rozhraní navrhnout tak, aby se s ním dalo rozumně pracovat a uživatel si nemusel rvát vlasy. Nechtěl jsem aplikaci pro testovací účely zbytečně komplikovat a proto veškeré úpravy dat probíhají na textové úrovni, tedy uživatel si vybere zastávku, kterou chce upravovat a následně její atributy už pouze přepisuje v textovém políčku.

Prvním z problémů, se kterými jsem se potýkal, byl výběr konkrétní zastávky při pokusu o editaci zastávky. K tomu jsem využil třídu *Auto complete* z frameworku Xamarin. *Auto complete* je prvek, který umožňuje filtrovat prvky seznamu na základě napsání části názvu a následně zobrazí seznam několika položek, které vyhovují zadanému filtru (viz obrázek 6.3). Poté může uživatel kliknout na položku seznamu, ta se vybere, seznam se opět sbalí a cíl je splněn, byla určena jedna položka ze seznamu. Podobný prvek se vyskytuje snad na každém webu. Hlavně mi to přišlo jako nejlepší možnost, jak zvládnout výběr z velkého počtu různých záznamů, obzvláště na mobilním zařízení.

Další věc, kterou jsem musel v rámci programu řešit byla možnost zadat nové GPS souřadnice zastávky, pro tyto účely má Xamarin připravenou třídu *Entry*, jde o podobné políčko, jaké vyplňujeme například pokud chceme v prohlížeči zadat svoje přihlašovací údaje do emailu, umožňuje i standardně skrýt zadávaný text anebo zobrazit pomocný text, když políčko není vyplněno.

Vůbec asi nejvíce úsilí jsem věnoval tomu, jak bych nejlépe ztvárnil v rámci uživatelského grafického prostředí přidávání pěších hran, nějakou dobu jsem vyhledával, zda by se pro tyto účely nenašla v Xamarinu nějaká vhodná specializovaná komponenta, ale nic zajímavého, natož pak speciálně uzpůsobeného pro tyto účely jsem nenašel. Proto jsem upustil od toho, abych umožnil uživateli přidávání hrany pomocí metody *Drag and Drop* na mapě a nakonec jsem celou věc vyřešil tak, že jsem zapojil dva různé *Auto complete* prvky, kterými uživatel určí dvojici zastávek mezi nimiž hranu povede a přidal jsem *Entry*, aby uživatel mohl doplnit i informaci jak dlouho podle něj cesta mezi zastávkami bude trvat.

Obrázek 6.3: Ukázka výběru cílové zastávky v prototypu aplikace



Dále jsou v implementaci zakomponovány další věci, jako výběr adresáře, ze kterého se bude GTFS feed načítat (bohužel použitá knihovna nepodporuje standardně využít načítání ze zabaleného archivu, což je škoda), ale to už jsou pouze drobnosti, které jsou přímo vysvětlené ve spoustě různých návodů na internetu a proto si dovolím se jimi blíže nezaobírat.

Závěr

V rámci této diplomové práce jsem se blíže seznámil s formátem GTFS, který je už dnes prakticky standardem, co se týče práce s informacemi o dopravních sítích, ať už jde o jejich spravování, sdílení anebo cokoliv jiného. Nyní už vím, co obnáší takový GTFS feed vytvořit, z čeho se vlastně skládá, jak s ním pracovat a i k čemu se dá použít. V tom mi výrazně pomohlo i to, že než jsem mohl použitou knihovnu začít využívat, tak jsem v ní napřed musel opravit několik chyb, díky tomu se dá říct, že jsem svou prací na diplomové práci podpořil i open-sourcový projekt, knihovnu GTFS pro C#, kterou můžou v budoucnu používat i další vývojáři.

V rešeršní části jsem zmapoval i různé existující aplikace, které pracují s daty v GTFS formátu. Zjistil jsem, že aplikace mají velice široké využití a ani zdaleka se nelimitují na vyhledání dopravních spojů v dopravní síti. Pro některé z těchto nástrojů jsou důležitá i různá existující rozšíření GTFS formátu.

Dále jsem nastínil principy kolem algoritmů, které jsou používány pro převod textové reprezentace GTFS formátu do grafu dopravní sítě, aby se s daty dalo dále pracovat. V práci jsem se věnoval i algoritmům, které jsou potřebné pro práci s grafem dopravní sítě.

V rámci práce vznikla aplikace, která umožňuje několik různých úprav grafu dopravní sítě, jmenovitě to jsou přesun zastávky, spojení dvou zastávek, přidání mimořádného spoje a zadávání výluky. Mimo možných úprav umožňuje aplikace i vyhledávání tras v grafu dopravní sítě, což byla z implementovaných funkcionalit největší výzva, protože by vyhledávání mělo být rychlé a bylo tedy nutné promyslet, jak hledání co nejvíce zefektivnit.

A také během se mi podařilo přijít s několika možnostmi, jak by se daly vylepšit již existující aplikace. Například to je přidávání pěších hran, efektivní přeplánování trasy v případě zpoždění, vyhledávání cesty na základě znalosti názvu ulice, sledování zpoždění spojení na základě GPS souřadnic uživatele a další.

Prototyp si oproti existujícím aplikacím stojí celkem obstojně, sice podle

očekávání z hlediska přívětivosti k uživateli trochu zaostává, ale to je dáno tím, že tento prototyp byl prací jednotlivce a nebyl vyvíjen celým týmem, kde by byl k dispozici někdo, kdo by se věnoval čistě uživatelské přívětivosti a dotáhnul ji k dokonalosti. V dalších ohledech naopak existující aplikace převyšuje, například prototyp aplikace umožňuje na rozdíl od jiných aplikací dělat úpravy v dopravní síti, takovou funkcionalitu jsem u žádné jiné aplikace nezaznamenal.

Budoucí práce

Do budoucna se nabízí spousta způsobů, jak aplikaci dále vylepšit. Jak už jsem zmínil, tak by určitě stálo za to, aby bylo dále vylepšování uživatelské rozhraní aplikace, aby byla vzhledově přívětivější. V budoucí verzi aplikace by se mohl například přidat do vyhledávání jakýsi index nepohodlí, aby aplikace preferovala cesty, kde sice může být čas dojezdu třeba o minutu později, ale za cenu, že uživatel nemusí využít pěší chůze, nebo nabídnout uživateli hned několik možností, jakou z variant cestování má aplikace preferovat, zda variantu s nejméně přestupy nebo nejrychlejší přesun ze startu do cíle a tak podobně. Zrovna v systému vyhledávání dopravních spojení je určitě vždy co vylepšovat.

Literatura

[1] Univerzita Palackého v Olomouci: *Druhy dopravy [online]*. [cit. 2016-04-02]. Dostupné z: `http://geography.upol.cz/soubory/lide/hercik/GEDP/Prednasky/dopravni_sit.pdf`

[2] Google: *General Transit Feed Specification Reference [online]*. [cit. 2016-04-02, poslední změna 2016-07-12]. Dostupné z: `https://developers.google.com/transit/gtfs/reference`

[3] Google: *Changes to GTFS format [online]*. [cit. 2016-04-05]. Dostupné z: `https://developers.google.com/transit/gtfs/changes#guiding-principles`

[4] Transitwiki.org: *General Transit Feed Specification [online]*. [cit. 2016-04-02, poslední změna 2016-10-27]. Dostupné z: `http://www.transitwiki.org/TransitWiki/index.php?title=General_Transit_Feed_Specification`

[5] *Google Transit Extensions to GTFS [online]*. [cit. 2016-04-05, poslední změna 2016-07-26]. Dostupné z: `https://developers.google.com/transit/gtfs/reference/gtfs-extensions`

[6] Circlegate: *CG Transit [online]*. [cit. 2016-04-18, poslední změna 2016]. Dostupné z: `http://www.circlegate.com/cs/cgt`

[7] *Open Trip Planner Docs [online]*. [cit. 2016-04-18, poslední změna 2016-09-09]. Dostupné z: `http://docs.opentripplanner.org/en/latest/`

[8] *Trillium: GTFS Manager [online]*. [cit. 2016-04-18, poslední změna 2016]. Dostupné z: `http://trilliumtransit.com/gtfs/gtfs-manager/`

[9] Holub, J.: *Automaty a gramatiky: Přednášky [online]*. České vysoké učení technické v Praze, Fakulta informačních technologií, 2016, [cit. 2016-12-12, poslední změna 2016]. Dostupné z: `https://edux.fit.cvut.cz/courses/BI-AAG/`

[10] *Common Format and MIME Type for Comma-Separated Values (CSV) Files [online].* [cit. 2016-12-12, poslední změna 2005]. Dostupné z: https://tools.ietf.org/html/rfc4180

[11] *Augmented BNF for Syntax Specifications: ABNF [online].* [cit. 2016-12-12, poslední změna 2008]. Dostupné z: https://tools.ietf.org/html/rfc5234

[12] Matematika.cz: *Pythagorova věta [online].* [cit. 2016-04-07, poslední změna 2014]. Dostupné z: http://www.matematika.cz/pythagorova-veta

[13] *GIS FAQ Q5.1: Great circle distance between 2 points [online].* [cit. 2016-04-10, poslední změna 2005]. Dostupné z: http://www.movable-type.co.uk/scripts/gis-faq-5.1.html

[14] Univerzita Karlova: *Matematické metody v kartografii [online].* [cit. 2016-04-07]. Dostupné z: https://web.natur.cuni.cz/~bayertom/Mmk/mk2.pdf

[15] Univerzita Karlova: *Soustavy souřadnic [online].* [cit. 2016-04-10]. Dostupné z: http://matematika.cuni.cz/dl/analyza/animace/k0033/souradnice/home.htm

[16] Wolfram Research, Inc.: *Great Circle [online].* [cit. 2016-04-19, poslední změna 2017-01-04]. Dostupné z: http://mathworld.wolfram.com/GreatCircle.html

[17] GeeksforGeeks: *Dijkstra's algorithm [online].* [cit. 2016-04-19, poslední změna 2016-11-16]. Dostupné z: http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

[18] GeeksforGeeks: *Breadth First Traversal for a Graph [online].* [cit. 2016-04-19, poslední změna 2016-12-11]. Dostupné z: http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/

[19] MIT: *A* search [online].* [cit. 2016-04-25, poslední změna 2002]. Dostupné z: http://web.mit.edu/eranki/www/tutorials/search/

[20] Algoritmy.net: *Binární vyhledávání [online].* [cit. 2016-04-24]. Dostupné z: https://www.algoritmy.net/article/21/Binarni-vyhledavani

[21] *Dictionary(TKey, TValue) Class [online].* [cit. 2017-01-05]. Dostupné z: https://msdn.microsoft.com/en-us/library/xfhwa508.aspx#Remarks

[22] Google: *What is GTFS [online].* [cit. 2016-03-30, poslední změna 2016-07-26]. Dostupné z: https://developers.google.com/transit/gtfs

[23] Transitwiki.org: *Deriving the Haversine Formula [online]*. [cit. 2016-04-07, poslední změna 2016-10-27]. Dostupné z: http://www.transitwiki.org/TransitWiki/index.php?title=General_Transit_Feed_Specification

[24] *GTFS Data Exchange [online]*. [cit. 2016-04-10, poslední změna 2016-01-12]. Dostupné z: http://www.gtfs-data-exchange.com/agency/dopravni-podnik-hl-m-prahy-akciova-spolecnost/

[25] *Performance Comparison - C++ / Java / Python / Ruby/ Jython / JRuby / Groovy [online]*. [cit. 2016-04-12, poslední změna 2008-07-08]. Dostupné z: http://blog.dhananjaynene.com/2008/07/performance-comparison-c-java-python-ruby-jython-jruby-groovy/

[26] *Xamarin: Mobile App Development & App Creation Software [online]*. [cit. 2016-11-15, poslední změna 2017]. Dostupné z: http://xamarin.com

[27] *Qt - Developers [online]*. [cit. 2016-04-15, poslední změna 2017]. Dostupné z: http://www.qt.io/developers/

[28] *Qt in use [online]*. [cit. 2016-04-15, poslední změna 2017]. Dostupné z: https://www.qt.io/qt-in-use/

[29] *TimeComplexity - Python Wiki [online]*. [cit. 2016-04-30, poslední změna 2015-06-15]. Dostupné z: https://wiki.python.org/moin/TimeComplexity

[30] *First programs in PyQt4 [online]*. [cit. 2016-04-12, poslední změna 2015-01-25]. Dostupné z: http://zetcode.com/gui/pyqt4/firstprograms/

[31] Jungnickel, D.: *Graphs, Networks and Algorithms (Algorithms and Computation in Mathematics)*. Springer, 2012, ISBN 3642322778. Dostupné z: http://www.amazon.com/Graphs-Networks-Algorithms-Computation-Mathematics/dp/3642322778%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creativeq%3D165953%26creativeASIN%3D3642322778

[32] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to Algorithms, 3rd Edition (MIT Press)*. The MIT Press, 2009, ISBN 0262033844. Dostupné z: https://www.amazon.com/Introduction-Algorithms-3rd-MIT-Press/dp/0262033844%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0262033844

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

GPS Global Positioning System

CSV Comma Separated Values

GTFS General Transit Feed Specification

GTFS Feed Soustava několika CSV souborů, které dohromady obsahují informace o dopravní síti

GUI Graphical User Interface

XML Extensible markup language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `data` ..... adresář s daty použitými při tvorbě práce
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF