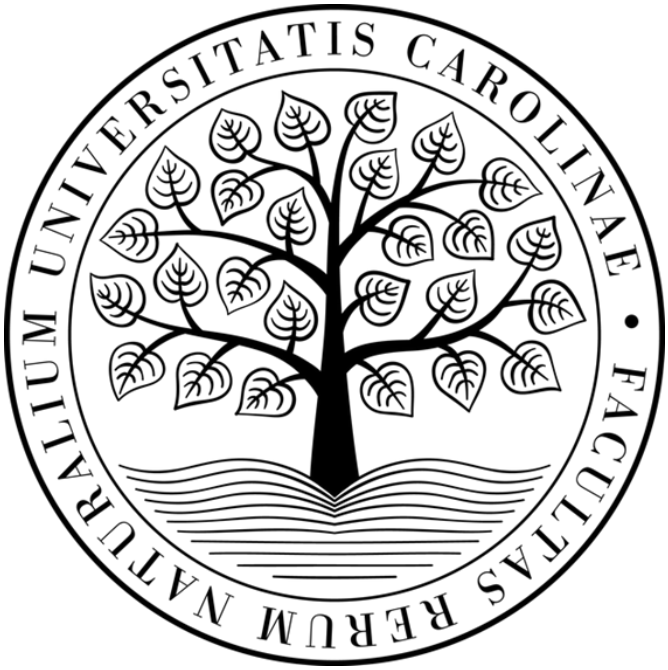
Univerzita Karlova

Přírodovědecká fakulta



GEOINFORMATIKA

Nejkratší cesta grafem

Martina Pavlová, Ludmila Vítková

1 N-GKDPZ

Praha, 2024

Zadání

Implementujte Dijkstra algoritmus pro nalezení nejkratší cesty mezi dvěma uzly grafu. Vstupní data budou představována silniční sítí doplněnou vybranými sídly.

Otestujte různé varianty volby ohodnocení hran grafu tak, aby nalezená cesta měla:

- Nejkratší Eukleidovskou vzdálenost
- Nejmenší transportní čas

Ve vybraném GIS konvertujte podkladová data do grafové reprezentace představované neorientovaným grafem. Pro druhou variantu optimální cesty navrhněte vhodnou metriku, která zohledňuje rozdílnou dobu jízdy na různých typech komunikací. Výsledky (dvě různé cesty pro každou variantu) umístěte do tabulky, vlastní cesty vizualizujte. Dosažené výsledky porovnejte s vybraným navigačním SW.

Krok	**Hodnocení**
Dijkstra algoritmus	20b
Řešení úlohy pro grafy se záporným ohodnocením	*+10b*
Nalezení nejkratších cest mezi všemi dvojicemi uzlů	*+10b*
Nalezení minimální kostry některou z metod	*+15b*
Využití heuristiky Weighted Union	*+5b*
Využití heuristiky Path Compression	*+5b*
Max celkem	**65b**

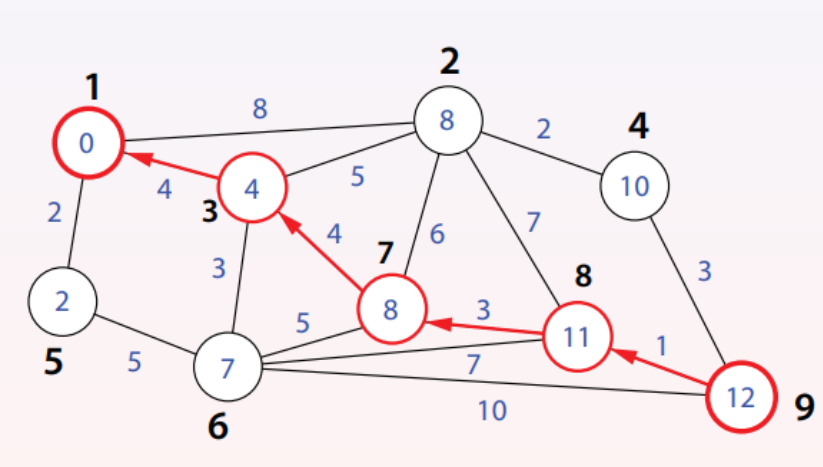
Čas zpracování: do konce zápočtového týdne

1. Popis a rozbor problému

Nalezení nejkratší trasy mezi dvěma body v grafu je častým problémem v oblasti teorie grafů a využívá se například pro optimalizaci tras ve veřejné dopravě, distribuci zboží v logistice či hledání nejkratších cest v grafických modelech. Tento problém může být řešen různými algoritmy, přičemž mezi nejznámější patří Dijkstrův a Bellman-Fordův algoritmus. Nemusí sloužit pouze k hledání nejkratší cesty, ale také například k nalezení nejrychlejší či nejekologičtější trasy. V praxi se často využívá kombinace různých algoritmů k dosažení co nejlepšího výsledku.

1. 1 Dijkstrův algoritmus

Dijkstrův algoritmus se využívá pro nalezení nejkratší cesty z jednoho uzlu v ohodnoceném grafu do všech ostatních uzlů. Začíná se zadaným výchozím uzlem a postupně prochází sousední uzly, aktualizuje jejich vzdálenosti od výchozího uzlu a pokračuje ve výběru uzlu s nejmenší dosavadní vzdáleností. Takto prochází celý graf až do posledního uzlu. Dijkstrův algoritmus využívá prioritní frontu nebo haldovou strukturu k efektivnímu výběru uzlů s nejmenší dosavadní vzdáleností, což přispívá k rychlému nalezení nejkratších cest v grafech s velkým počtem uzlů. Tento algoritmus funguje efektivně v grafech bez záporných hodnot. Jeho výsledkem je seznam nejkratších vzdáleností od výchozího uzlu ke všem ostatním uzlům a seznam předchůdců, který umožňuje rekonstrukci nejkratší trasy. Na obrázku 1 lze vidět, jak může vypadat výsledek tohoto algoritmu.



Obrázek 1: Příklad výsledku algoritmu Dijkstra (Bayer 2023)

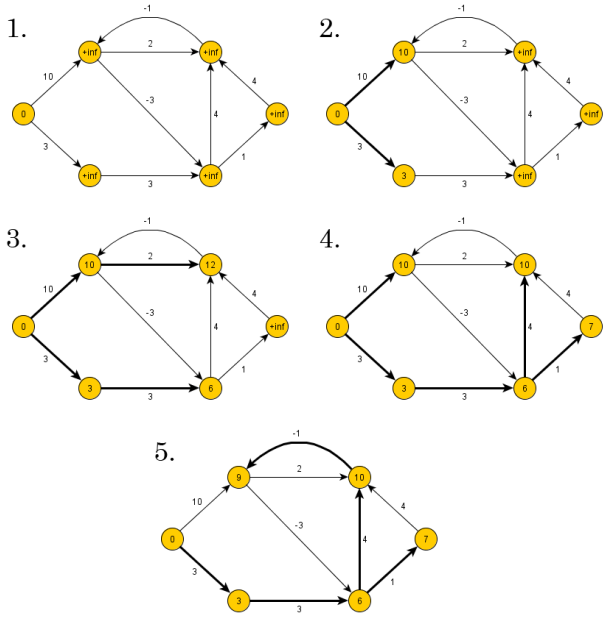
1. 1. 1 Implementace Dijkstrova algoritmu – pseudokód:

Přiřaď počet uzlů v grafu do proměnné *n*
Inicializuj seznam *P* o velikost *n + 1,* kde jsou všechny prvky nastaveny na -1
Inicializuj seznam *D* o velikost *n + 1*, kde jsou všechny prvky nastaveny na nekonečno
Nastav vzdálenosti počátečního uzlu na 0
Vytvoř prázdnou prioritní frontu
Přiřaď počáteční uzel do prioritní fronty se vzdáleností 0
Dokud není prioritní fronty prázdná

Vezmi uzel s nejmenší vzdáleností z prioritní fronty a přiřaď ho do proměnné *u*
Iteruj přes sousedy uzlu *u* v grafu
Načti váhu hrany mezi uzly *u* a *v*
Pokud existuje menší vzdálenost
Aktualizuj vzdálenost uzlu *v*
Aktualizuj předchůdce
Přidej uzel zpět do prioritní fronty s jeho aktualizovanou vzdáleností
Vracej list předchůdců a vzdáleností

1. 2 Bellman-Fordův algoritmus

Bellman-Fordův algoritmus se také využívá k nalezení nejkratší trasy z jednoho uzlu do všech ostatních uzlů v ohodnoceném grafu, avšak funguje i pro grafy obsahující záporně ohodnocené hrany. Algoritmus pracuje postupně, iterativně relaxuje hrany v grafu, aktualizuje vzdálenosti uzlů od výchozího uzlu a opakuje tento proces pro každý uzel ve všech iteracích. Celkový počet iterací je omezen na počet uzlů minus jedna, aby se zajistilo správné nalezení nejkratších cest v grafu. Pokud v průběhu relačních operací dochází k aktualizaci vzdáleností i ve (n-1)-té iteraci, kde *n* je počet uzlů, pak graf obsahuje záporný cyklus. Výsledkem je stejně jako u algoritmu Dijkstra seznam nejkratších vzdáleností od výchozího uzlu ke všem ostatním uzlům a seznam předchůdců. Jak již bylo zmíněno tento algoritmus funguje i se záporným ohodnocením hran, a tak se v těchto případech využívá místo algoritmu Dijkstra. Na obrázku 2 je vidět příklad výsledku tohoto algoritmu.



Obrázek 2: Příklad výsledku Bellman-Fordova algoritmu (Neckář 2016)

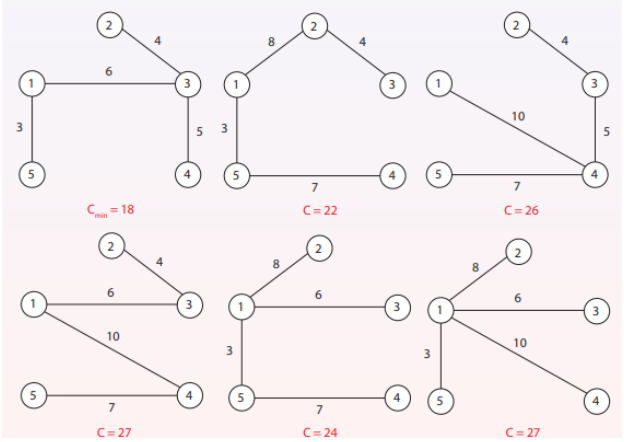
1. 2. 1 Implementace Bellman-Fordova algoritmu – pseudokód:

```
Přiřaď počet uzlů v grafu do proměnné n
Inicializuj seznam P o velikost n + 1, kde jsou všechny prvky nastaveny na -1
Inicializuj seznam D o velikost n + 1, kde jsou všechny prvky nastaveny na nekonečno
Nastav vzdálenosti počátečního uzlu na 0
        Iteruj n – 1-krát
                Iteruj přes všechny uzly v grafu
                        Iteruj přes všechny sousedy uzlu
                                Pokud je váha = 0 (použije délku jako váhu)
                                        Pokud existuje menší vzdálenost
                                                Aktualizuj vzdálenost uzlu v
                                                Aktualizuj předchůdce
                                Jinak pokud je váha = 1 (použije čas jako váhu)
                                        Pokud existuje rychlejší cesta
                                                Aktualizuj čas k uzlu v
                                                Aktualizuj předchůdce
        Iteruj přes všechny uzly v grafu
                Iteruj přes všechny sousedy uzlu
                        Pokud je váha = 0
                                Pokud je přítomna záporně ohodnocená hrana
                                        Vypiš „Graph contains negative weight cycle“
                                        Vrať hodnoty None
                        Jinak pokud je váha = 1
                                Pokud je přítomna záporně ohodnocená hrana
                                        Vypiš „Graph contains negative weight cycle“
                                        Vrať hodnoty None

Vracej list předchůdců a vzdáleností
```

1. 3 Minimální kostra grafu

Minimální kostra grafu je podgraf, ve kterém se nachází všechny uzly původního grafu a zároveň tvoří strom bez cyklů. Hrany této kostry jsou vybírány tak, aby součet jejich vah byl co nejmenší mezi všemi možnými podgrafy tvořící stromy bez cyklů. Pro nalezení minimální kostry grafu existuje několik různých algoritmů, přičemž mezi nejznámější patří Borůvkův (Kruskalův) a Primův algoritmus. Borůvkův algoritmus začíná s každým uzlem jako samostatnou komponentou. Následně v každé iteraci přidá nejlehčí hrany spojující dvě komponenty. Tento proces se opakuje až do té doby, dokud není celý graf spojen do jedné komponenty. Borůvkův algoritmus je vhodný pro grafy s mnoha malými izolovanými částmi. Primův algoritmus začíná s libovolným uzlem jako počátečním stromem a v každé iteraci přidává nejlehčí hranu spojující existující strom s novým uzlem. Tento algoritmus je vhodnější pro husté grafy s velkým počtem hran. Bývá také pomalejší než Borůvkův algoritmus. Na obrázku 3 jsou znázorněné některé minimální kostry grafu.



Obrázek 3: Příklad minimálních koster grafu (Bayer 2023)

1. 3. 1 Implementace Borůvkova algoritmu – pseudokód:

Inicializuj prázdný seznam *T*, pro reprezentaci výsledného stromu
Inicializuj proměnnou *wt*, pro ukládání součtu vah
Inicializuj seznam *p*, pro reprezentaci kořenů
Inicializuj seznam *r*, pro reprezentaci hodnoty každého uzlu
Pro vrchol *v* v množině *V:*
    Vytvoř množinu a inicializuj ji pomocí funkce *make_set*
Seřaď hrany v *E* vzestupně podle jejich vah a ulož je do proměnné *ES*
Pro hranu *e* v *ES:*
    Rozbal aktuální hranu *e* do složek *u, v* a *w*
    Pokud jsou kořeny *u* a *v* v jiných množinách:
        Sjednoť je pomocí funkce *weighted_union*
        Přidej aktuální hranu do stromu
        Aktualizuj celkovou váhu stromu

Vracej celkovou váhu stromu a strom ve formě seznamu hran

1. 4 Floyd-Washallovův algoritmus

Floyd-Washallovův algoritmus se využívá pro nalezení nejkratších cest mezi všemi dvojicemi uzlů v ohodnoceném grafu. Tento algoritmus pracuje iterativně a udržuje matici vzdáleností mezi všemi dvojicemi uzlů. V každé iteraci zkouší aktualizovat vzdálenosti mezi všemi dvojicemi uzlů pomocí všech ostatních uzlů. Algoritmus pokračuje, dokud není celá matice vzdáleností aktualizována. Jeho výsledkem je tedy matice nejkratších vzdáleností mezi všemi dvojicemi uzlů, přičemž současně jsou v něm zahrnuty i informace o předchůdcích. Tento algoritmus je efektivní pro malé a středně velké grafy.

1. 4. 1 Implemenatce Floyd-Washallova algoritmu – pseudokód:

```
Přiřaď počet uzlů v grafu do proměnné num_nodes
Inicializuj
seznam distance_matrix se všemi hodnotami nastavenými na nekonečno
Pro uzel v grafu:
        Pro souseda a jeho atributy:
                Aktualizuj distance_matrix váhou hrany od aktuálního uzlu k jeho sousedovi
Pro mezilehlý uzel
        Pro počáteční uzel
                Pro cílový uzel
                        Pokud existuje kratší cesta
                                Aktualizuj distance_matrix
Vracej distance_matrix
```

1. 5 Výpočet času

Vztah mezi vzdáleností, rychlostí a časem lze vyjádřit pomocí následujícího vzorce, kde *s* značí vzdálenost, *v* rychlost a *t* čas.

$$s = v*t$$

Převedením na tvar $t = s/v$, můžeme tedy vypočítat čas, za který vozidlo urazí daný úsek. Vzdálenost může být reprezentována vzdáleností mezi dvěma uzly a rychlost může být reprezentována maximální povolenou rychlostí mezi těmito dvěma uzly. Maximální povolená rychlost na dálnicích je 130 km/h, na rychlostních silnicích 110 km/h, na ostatních silnicích mimo obce 90 km/h, v obcích 50 km/h a v obytných zónách a na neklasifikovaných komunikacích 30 km/h.

1. 6 Klikatost silniční sítě

Doba projetí určitého úseku silniční sítě záleží také na její klikatosti, která se dá do výpočtu zavést pomocí korekčního koeficientu *c*, který se počítá pomocí následujícího vzorce.

$$c = \frac{Eukleidovská\ vzdálenost}{vzdálenost\ po\ křivce}$$

Tento koeficient nabývá hodnot nula až jedna. Rovná silnice má hodnotu jedna, tudíž čím klikatější silnice tím je hodnota koeficientu menší. Tímto koeficientem byla následně vynásobena nejvyšší povolená rychlost, jelikož po rovné silnici může jet vozidlo rychleji než po klikaté silnici.

2. Struktura programu

Program se skládá z 265 řádek včetně komentářů a odsazení a obsahuje dvanáct funkcí.

První funkce *create_graph* bere jako vstup název místa a pomocí knihovny *osmnx* získá graf silniční sítě pro toto konkrétní místo, který je následně vizualizován pomocí *Matplotlib*. Funkce vrací vytvořený graf.

Druhá funkce *calculate_travel_time* pracuje s dvěma parametry, kterými jsou *distance* (vzdálenost) a *road_type* (typ silnice). Pomocí *speed_limit* získává maximální povolenou rychlost na daném typu silnice. Pokud pro daný typ silnice není maximální povolená rychlost definována nebo je nula, vrátí funkce hodnotu nekonečno, aby se vyhnula dělení nulou. Pokud je ale maximální povolená rychlost definována a je různá od nuly, pak funkce vypočítá doby cesty mezi dvěma uzly a tu následně vrátí.

Funkce *shortest_path_algorithm* implementuje buď Dijkstrův nebo Bellman-Fordův algoritmus pro nalezení nejkratších cest ve váženém grafu. Tato funkce přijímá strukturu grafu, počáteční uzel, atribut váhy specifikující, zda bude počítáno se vzdáleností nebo časem a volitelný parametr pro typ algoritmu. Funkce vypočítá nejkratší cesty a vzdálenosti od počátečního uzlu ke všem ostatním uzlům, přičemž používá buď přístup Dijkstrova algoritmu s prioritní frontou nebo přístup Bellman-Fordova algoritmu k práci s negativními váhami a detekci cyklů s negativní váhou. Funkce vrací seznam předchůdců a vzdáleností pro každý uzel v grafu. Pokud je v algoritmu Bellman-Ford detekován cyklus s negativní váhou, vypíše se zpráva s funkce vrátí *None*.

Funkce *reconstruct_path* přijímá seznam předchůdců, graf, počáteční uzel a koncový uzel. Tato funkce rekonstruuje nejkratší cestu od počátečního uzlu k tomu koncovému pomocí seznamu předchůdců. Funkce iteruje pře předchůdce, přidává každý uzel do cesty a vypočítává celkovou vzdálenost a čas na základě vah hran v grafu. Výsledná cesta je vrácena v obráceném pořadí spolu s akumulovanou vzdáleností a časem.

Funkce *visualize_graph* přijímá graf, pozice uzlů a cestu v grafu. Pomocí knihovny *networkx* vytváří graf, ve kterém jsou všechny hrany šedé a nejkratší či nejrychlejší trasa je zvýrazněna zelenou barvou. Výsledný graf je vizualizován pomocí knihovny *Matplotlib*.

Minimální kostra grafu je hledána pomocí spojení čtyř funkcí. Funkce *make_set* slouží k vytvoření nové množiny s prvkem *u*, přičemž inicializuje obě pole *p* (parrent) a *r* (rank). Funkce *find* slouží k nalezení kořene množiny, ke které patří prvek *u* zároveň implementuje kompresi cesty pro optimalizaci budoucích operací. Funkce *weighted_union* nejprve najde kořeny množin ke kterým patří *u* a *v* a poté provede sjednocení těchto dvou množin. I tato funkce využívá kompresi k optimalizaci hledání kořenů. Poslední funkcí sloužící k nalezení minimální kostry grafu je funkce *boruvka*, která implementuje Borůvkův algoritmus. Tento algoritmus řadí hrany podle váhy a následně iterativně přidává hranu s nejmenší váhou, která spojuje dvě různé množiny, do minimální kostry grafu. Tato funkce skončí po propojení všech uzlů.

Další funkcí je *visualize_minimum_spanning_tree*, která přijímá graf, pozice uzlů a seznam hran reprezentující minimální kostru grafu. Výsledný graf je vytvořen pomocí knihovny *networkx*,

kde jsou všechny hrany šedé a minimální kostra grafu je zvýrazněna červeně. Výsledná vizualizace probíhá za pomoci knihovny *Matplotlib*.

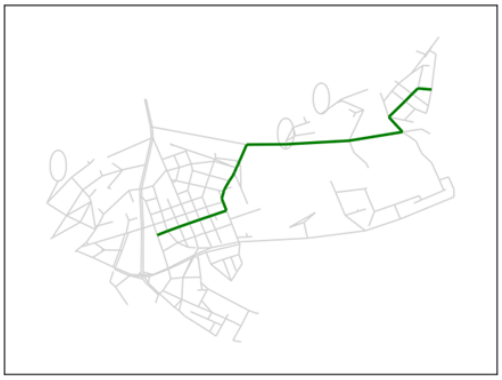
Pro výpočet nejkratší vzdálenosti mezi všemi dvojicemi uzlů byla definována funkce *shortest_distance*, která využívá Floyd-Warshallův algoritmus. Tato funkce inicializuje matici vzdáleností s počátečními hodnotami nastavenými na nekonečno. Tato matice je poté při nalezení kratší cesty aktualizována skutečnými vzdálenostmi mezi uzly. Výsledná matice tedy obsahuje nejkratší vzdálenosti mezi všemi dvojicemi uzlů v grafu.

Poslední funkcí je *main*, kde je prováděno několik úkolů souvisejících s analýzou a vizualizací grafu. Pomocí této funkce je vytvořen graf silniční sítě na základě specifikovaného názvu místa. Tato funkce dále konstruuje graf s uzly, hranami, vzdálenostmi a časy cestování, přečíslovává uzly, provádí algoritmy nejkratších cest (Dijkstra, Bellman-Ford) a rekonstruuje nejkratší/nejrychlejší cestu a vizualizuje ji. Dále vypočítává nejkratší vzdálenost mezi všemi dvojicemi uzlů a vypisuje informace o nejkratší a nejrychlejší trase mezi dvěma body. Nakonec pomocí Borůvkova algoritmu nachází minimální kostru grafu, vizualizuje ji a vypisuje její celkovou váhu.

3. Výsledky

Jako vstupní data byla vybrána data z OpenStreetMap (OSM), která obsahují informace například o ulicích, budovách a využití půdy. Tato data byla načtena pomocí modulu OSMnx, který může sloužit nejen k načítání, ale i analýze a vizualizaci silničních sítí. Data jsou ve formátu grafu, a tak je tedy není třeba nijak konvertovat. Stahování a načítání dat probíhá pomocí zadání adresy v jejímž okolí chceme data analyzovat. Lze vyhledávat celé státy, město a nebo i jen městské části. V tomto případě došlo k načtení dat z městské čtvrti Kobylisy. Ihned po spuštění programu se objeví graf vybrané oblasti, ve kterém se nachází pouze silniční síť a uzly na ní.

Výsledný graf nejkratší trasy lze vidět na obrázku 4 a na obrázku 5 se nachází nejrychlejší nalezená trasa. V tabulce 1 lze vidět vzdálenost a čas těchto dvou tras i s porovnáním s existujícím řešením, které je podrobněji probráno v následující kapitole. Nejkratší trasa měří necelých 2,5 km a trvá necelé 2,5 minuty. Nejrychlejší trasa měří skoro 3 km a je tak o více než 500 metrů delší. Tato trasa trvá přibližně dvě a třetinu minuty a je tedy pouze o cca čtvrt minuty rychlejší. Nejkratší i nejrychlejší trasa byla hledána jak pomocí algoritmu Dijkstra, tak i pomocí Bellman-Fordova algoritmu, přičemž oba tyto algoritmy poskytly stejné výsledky.



Obrázek 4: Nejkratší nalezená trasa

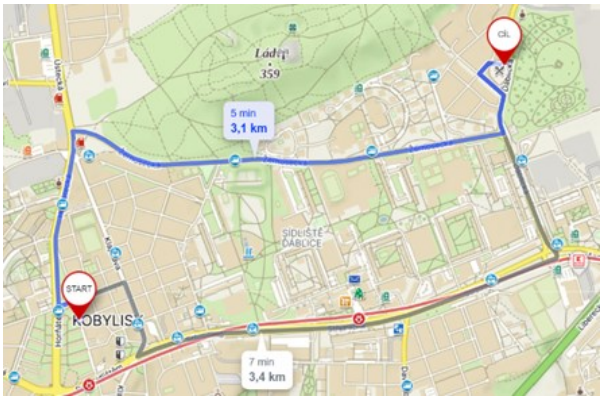
Obrázek 5: Nejrychlejší nalezená trasa

Trasa	Vzdálenost	Čas
Nejkratší	2,46 km	2,42 minut
Nejrychlejší	2,99 km	2,34 minut
Nejkratší podle Mapy.cz	2,50 km	5 minut
Nejrychlejší podle Mapy.cz	3,00 km	5 minut

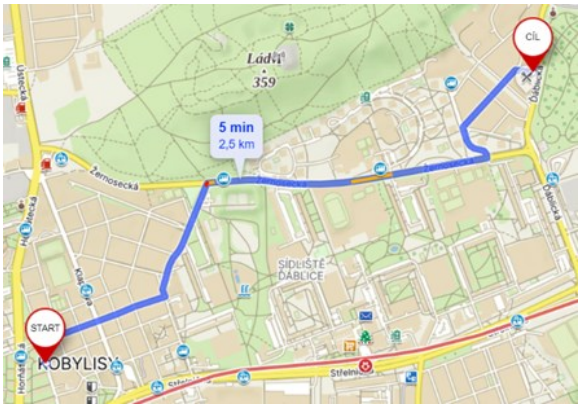
Tabulka 1: Charakteristiky výsledných cest

3. 1 Porovnání s Mapy.cz:

Po zadání počátečního a koncového bodu byla nejprve zvolena možnost nejkratší trasy. Výsledek lze vidět na obrázku 6. Již na první pohled je ale vidět, že se nejedná o nejkratší možnou trasu. Na zkoušku byla zvolena možnost nejrychlejší trasy s aktuální dopravou, jejíž výsledek lze vidět na obrázku 7. Tato trasa je 2,5 km dlouhá, měla by trvat přibližně 5 minut a shoduje s námi nalezenou nejkratší trasou.



Obrázek 6: Nejkratší trasa podle Mapy.cz



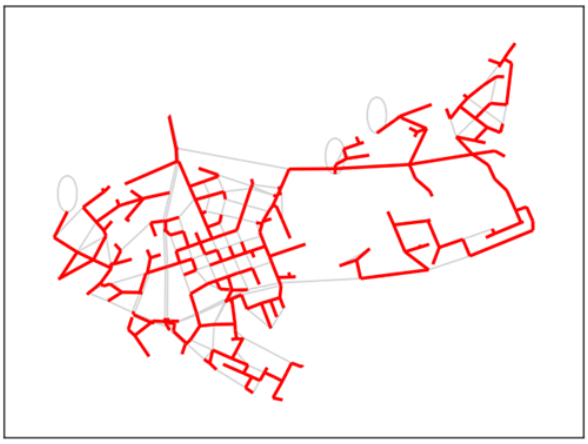
Obrázek 7: Nejrychlejší trasa s provozem podle Mapy.cz

Nejrychlejší trasa podle Mapy.cz je 3 km dlouhá, měla by trvat také 5 minut a lze ji vidět na obrázku 8. Tato trasa se nejdříve napojuje na hlavní silnice, dále odbočuje na další hlavní silnici a na svém konci prochází mezi rodinnými domy. V celé své délce se shoduje s námi nalezenou nejkratší cestou.

Obrázek 8: Nejrychlejší trasa podle Mapy.cz

3. 2 Minimální kostra grafu:

Na obrázku 9 lze vidět minimální kostru grafu vytvořenou s využitím nejkratší trasy. Hned na první pohled je vidět, že spojuje všechny uzly grafu. Hrany v této minimální kostře by měly mít minimální možnou celkovou váhu. V tomto případě je celková váha této kostry 23 509,53 m.



Obrázek 9: Minimální kostra grafu

3. 3 Nalezení nejkratších cest mezi všemi dvojicemi uzlů:

Program je schopný najít nejkratší cesty mezi všemi dvojicemi uzlů. Jelikož je ale ve vstupních datech přes 300 uzlů, vypisuje program pouze vzdálenost mezi dvěma určenými uzly. Jako tyto dva uzly byl zvoleny počáteční a koncový bod, u kterých byla hledána nejkratší/nejrychlejší trasa. Nejkratší nalezená trasa má přibližně 2,5 km a shoduje se tedy s trasou nalezenou pomocí algoritmu Dijkstra.

4. Závěr

V rámci této úlohy byly implementovány algoritmy Dijkstra a Bellman-Ford pro hledání nejkratších cest v ohodnoceném grafu, které byly následně použity k nalezení nejkratší a nejrychlejší trasy mezi dvěma určenými body v silniční síti. Výsledné grafy byly vizualizovány z dat OpenStreetMap pomocí nástroje OSMnx. Nalezené trasy byly nakonec porovnány s navigačním softwarem Mapy.cz, přičemž implementované algoritmy poskytly srovnatelné výsledky.

Dále byl implementován Borůvkův algoritmus pro hledání minimální kostry grafu, který spojil všechny uzly v grafu, tak aby její celková váha byla minimální. Tento výsledek byl následně také vizualizován. Jako poslední byl implementován Floyd-Warshallův algoritmus pro nalezení nejkratších cest mezi všemi dvojicemi uzlů v grafu.

Nejrychlejší nalezené trasy se liší o pouhých 10 metrů, což je s největší pravděpodobností způsobeno zaokrouhlováním tras na stránkách Mapy.cz. Nejkratší nalezené trasy se liší o 40 metrů, což může být způsobeno nepřesným zvolením bodu při manuálním zadávání trasy do navigace na stránkách Mapy.cz a nejspíše také opět zaokrouhlením délky trasy. Vzdálenosti obou nalezených tras se tedy shodují s trasami nalezenými pomocí navigace.

Čas u obou námi nalezených tras je o více než polovinu kratší než u tras na stránkách Mapy.cz. Důvodem by mohlo být, že stránka Mapy.cz možná počítá se zastavením na křižovatkách. S delším časem by tedy mohla počítat v místech, kde jsou semafory, v místech, kde vozidlo na křižovatce odbočuje vlevo anebo například když vozidlo vjíždí na hlavní silnici a musí tedy dávat přednost ostatním vozidlům. Dalším důvodem by mohla být možnost, že typ silnice v našich datech se liší od skutečnosti a tím pádem je námi definovaná nejvyšší povolená rychlost jiná než ve skutečnosti.

5. Zdroje

BAYER, T. (2023): Nejkratší cesta grafem, prezentace k předmětu Geoinformatika [online]. [cit. 1. 10. 2024]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Geoinf/geoinf_grafy2.pdf

GeeksforGeeks (2023): Bellman-Ford Algorithm [online]. [cit. 10. 1. 2024]. Dostupné z: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

GeeksforGeeks (2023): Floyd Warshall Algorithm [online]. [cit. 10. 1. 2024]. Dostupné z: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

GeeksforGeeks (2024): What is Dijkstra's Algorithm? [online]. [cit. 10. 1. 2024]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/

NECKÁŘ, J. (2016): Bellman-Fordův algoritmus [online]. [cit. 10. 1. 2024]. Dostupné z: https://www.algoritmy.net/article/7478/Bellman-Forduv-algoritmus

OpenStreetMap Wiki (2023): Key:highway [online]. [cit. 20. 12. 2023]. Dostupné z: https://wiki.openstreetmap.org/wiki/Key:highway

TENKANEN, H., HEIKINHEIMO, V. (2020): Retrieving OpenStreetMap data [online]. [cit. 20. 12. 2023]. Dostupné z: https://autogis-site.readthedocs.io/en/2019/notebooks/L6/retrieve_osm_data.html?fbclid=IwAR3mYaA6V4Y23-samZ9tXyALsRXx3BkAk1ll9UlLQu9NeknIzdLFMvsJZacv