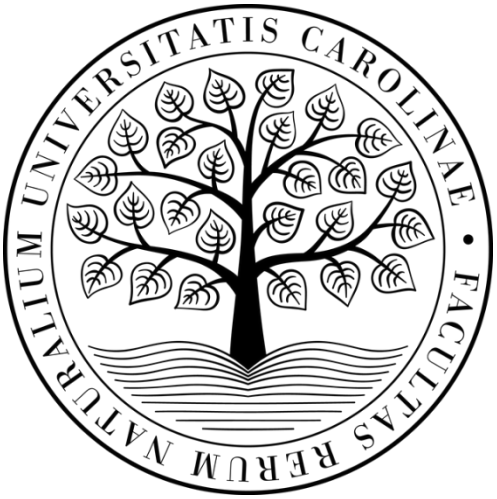
Univerzita Karlova

Přírodovědecká fakulta



GEOINFORMATIKA

Nejkratší cesta grafem

Pavel Horák, Eliška Tomášková

1. ročník, N-GKDPZ

Praha 2023

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**
Dijsktra 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**

Úvod

Cílem této práce je najít nejméně nákladnou cestu mezi dvěma uzly grafu (zvolenými sídly). Konkrétně bude hledána délkově nejkratší cesta a také nejrychlejší cesta. K těmto výpočtům bude využit Dijkstra algoritmus, což je asi nejznámější algoritmus pro hledání nejkratších cest mezi dvěma uzly. Řešena bude i úloha nalezení nejkratších cest mezi všemi dvojicemi uzlů.

Dijkstrův algoritmus lze použít jen tehdy, obsahuje-li graf pouze nezáporně ohodnocené hrany. Proto bude dále využit algoritmus Bellman-Ford, který dokáže najít nejkratší cestu i v případě záporného ohodnocení hran (ale nikoliv záporných cyklů).

Dalším řešeným problémem bude nalezení minimální kostry grafu (*Minimum Spanning Tree*). Tato úloha bývá často řešena v dopravě, energetice a telekomunikacích. Problém lze řešit pomocí několika různých algoritmů, příkladem mohou být Borůvkův (modifikace Kruskal) nebo Jarníkův (modifikace Prim) algoritmus. Zde budou implementovány dvě heuristiky, a to Weighted Union a Path Compression, které vylepšují efektivitu.

Zpracování

Vstupní data

Práce byla zhotovena v prostředí Visual Studio Code s využitím jazyka Python. Pro zadaný úkol byla využita podkladová data *OpenStreetMap* dostupná v knihovně *osmnx*. Díky této knihovně se dají geografická data stáhnout přímo ve formě grafu a mohou být tak snadno využita pro hledání nejkratších cest. Nejprve bylo definováno zájmové území (souřadnice: *north, south, east, west = 50.220, 50.120, 12.330, 12.150*), jenž pokrývá ORP Aš a jeho blízké okolí. Následně byl graf načten pomocí funkce *ox.graph_from_bbox ,* konkrétně byly využity silnice (*network_type='drive'*) a jako vybraná sídla restaurace (*"amenity": 'restaurant*). Zvolené území pokrývá ORP Aš a jeho blízké okolí (definováno souřadnicemi: *north, south, east, west = 50.220, 50.120, 12.330, 12.150*). Restauracím byly přiřazeny souřadnice nové, a to souřadnice jejich nejbližšího uzlu, aby se staly součástí grafu.

Aby byl Dijkstra algoritmus schopný pracovat s daty z *OpenStreetMap* bylo je nejprve nutné převést do požadovaného formátu (vytvořit nový slovník):

G = {V1 : {V1:W1, V2:W2, ..., Vk:Wk},
V2 : {V1:W1, V2:W2, ..., Vk:Wk},
...
Vk : {V1:W1, V2:W2, ..., Vk:Wk}}

V případě úlohy hledání minimální kostry byla data naopak převedena do formátu:

E = [[V1, V2, W1], [V1, V2, W1],... [Vk, Vk, Wk]],

kde V představují identifikátory jednotlivých uzlů a W váhy mezi nimi.

Algoritmus Dijkstra

Dijkstrův algoritmus je považován za nejefektivnější metodu pro hledání nejkratších cest mezi jedním zadaným uzlem a všemi ostatními uzly v grafu, přičemž graf neobsahuje hrany s negativním ohodnocením. U tohoto typu algoritmu není třeba značkovat uzly a realizuje se opakovanou relaxací hrany. Dijkstra tedy zobecňuje strategii BFS (prohledávání do šířky) + relaxace + prioritní fronta.

Princip algoritmu Dijkstra:

1. Inicializační fáze:
   Nastavení nekonečné vzdálenosti pro všechny uzly, nastavení předchůdců na "None".
   Nastavení zdroje na 0 a přidání do Q.
2. Iterativní fáze:
   Dokud Q není prázdná:
      Z Q je vybrán uzel s nejmenší hodnotou.
      Relaxace - aktualizace vzdáleností sousedních uzlů:
         Pokud je nová vzdálenost menší než původní, provede se aktualizace.
         Aktualizace předchůdce.
         Přidání sousedního uzlu do Q.
3. Opakování Iterativní fáze:
   Iterativní fáze se opakuje, dokud Q není prázdná, což znamená, že existuje nějaký otevřený uzel.

Algoritmus Dijkstra byl implementován pro dvě různé varianty:

1. Nejméně nákladná cesta byla hledána pouze na základě délky silnic mezi dvěma zvolenými uzly.
2. Nejméně nákladná cesta byla hledána na základě nejmenšího transportního času. Jako vhodná metrika byla zvolena maximální rychlost, kterou je možné po daném typu silnice jet. Proto byla vytvořena nová funkce, jež vypočetla nové váhy jako:
   délka úseku silnice / 1000 (převod na km) / přiřazená rychlost

Jako zvolená metrika custom weight byl použit vzorec:

$$t = s/v,$$

,kde t představuje čas, s je dráha a v značí rychlost. Dráha musela být nejprve vydělena tisícem, aby byla v základních jednotkách (kilometry). Rychlosti v km/h byly definovány následujícím způsobem: residential 50, primary 90, primary_link 90, secondary 90, secondary_link 90, tertiary 90, tertiary_link 90 a unclassified 30. Výsledný čas t nám vyšel v hodinách a pro výsledek v minutách byl vynásoben 60.

Jako počáteční a koncový bod byly zvoleny uzly s ID: 412695093, 4406578975 (jedná se o uzly nacházející se nejblíže dvěma restauracím), které jsou od sebe vzdálené natolik, abychom mohli dobře pozorovat rozdíly ve výsledcích.

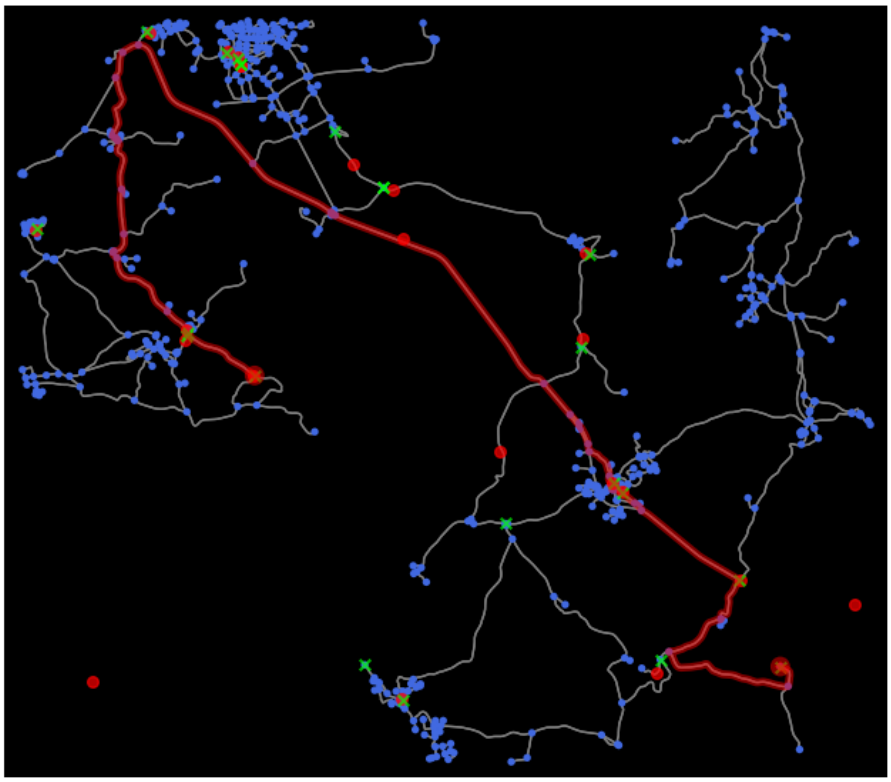
Vizualizace a výsledky

Namísto vizualizace výsledné trasy v GIS software byly cesty vizualizovány pomocí knihovny *osmnx* přímo ve Visual Studio Code. Nalezená nejkratší Euklidovská vzdálenost je vyobrazena na obrázku 1, nalezená nejméně nákladná cesta (nejrychlejší) je vizualizována na obrázku 2. V obrázcích 1 a 2 jsou hrany znázorněny šedou barvou, uzly modrou, restaurace představují červené body a zelenými křížky jsou označeny nejbližší uzly daným restauracím. Výsledná trasa je vykreslena červenou barvou. Stejný úkol byl řešen také na stránkách mapy.cz (viz obrázek 5) - trasa je velmi podobná, ne-li totožná s tou, která vyšla v případě hledání nejrychlejší cesty.

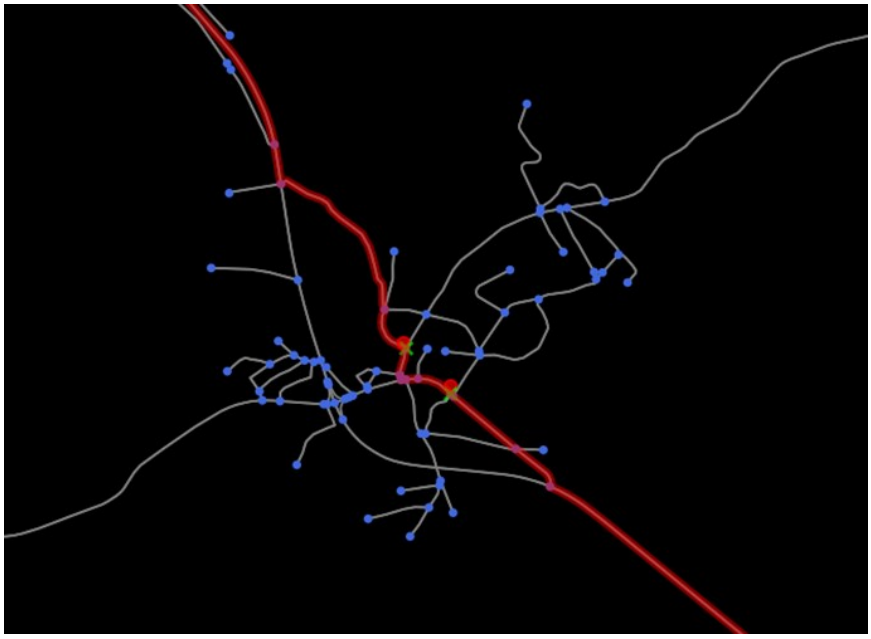
V tabulce 1 jsou porovnány nejrychlejší a nejkratší trasa. Lze si všimnout, že nejkratší trasa je kratší o 400 metrů než trasa nejrychlejší, ale naopak čas trasy je zhruba o jeden a půl minuty delší. Nejkratší cesta mezi zvolenými body prochází celkem 42 uzly, zatímco nejrychlejší trasa 41 uzly. Z obrázku 1 a 2 si lze pak povšimnout, že úseky, jež se mezi dvěma trasami liší, se nachází zejména v obcích. Na obrázcích 2 a 4 je právě detailní výřez jedné obce, ve které jsou nejrychlejší a nejkratší cesta značně rozdílné.

Pro nejrychlejší cestu byl dále vypočítán celkový čas, který vyšel 23,49 minut. Když srovnáme výsledek s mapy.cz, tak se liší přibližně o jeden a půl minuty. Podle mapy.cz by se totiž tato vzdálenost měla ujet za 26 minut. Tento rozdíl může být způsoben například z toho důvodu, že mapy.cz pracují se semafory, nepočítají s tím, že auta jezdí maximální povolenou rychlostí, započítávají zpomalení v zatáčkách, zohledňují klikatost či převýšení apod. Zatímco použitý

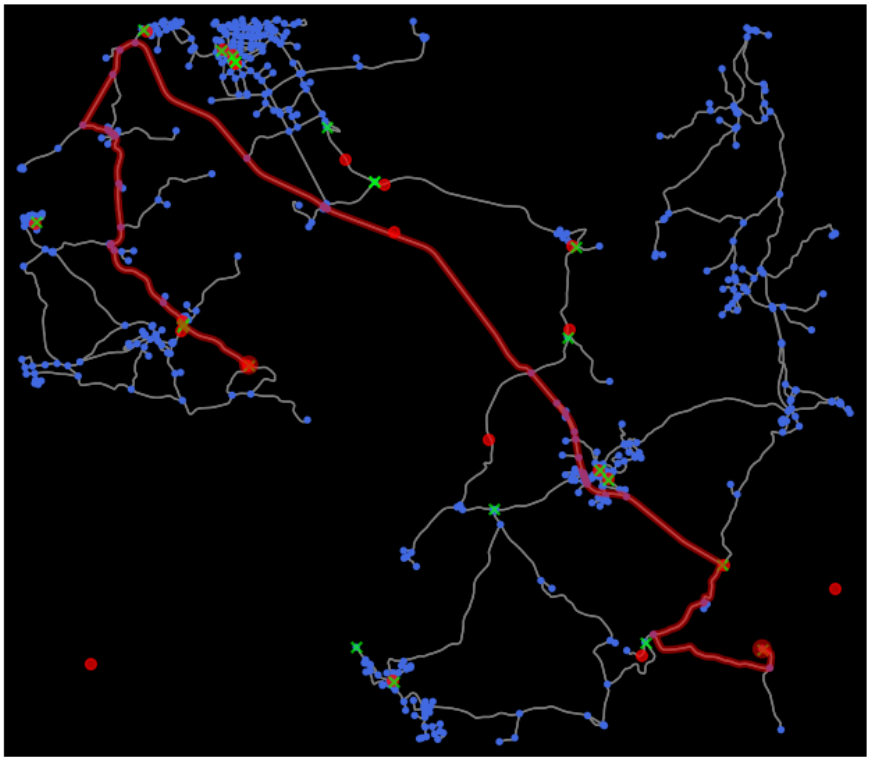
algoritmus používá pouze námi zadané nejvyšší povolené rychlosti. Grafická podoba cesty z mapy.cz odpovídá té, kterou jsme zjistili ve Visual Studio Code (viz obr. 5).



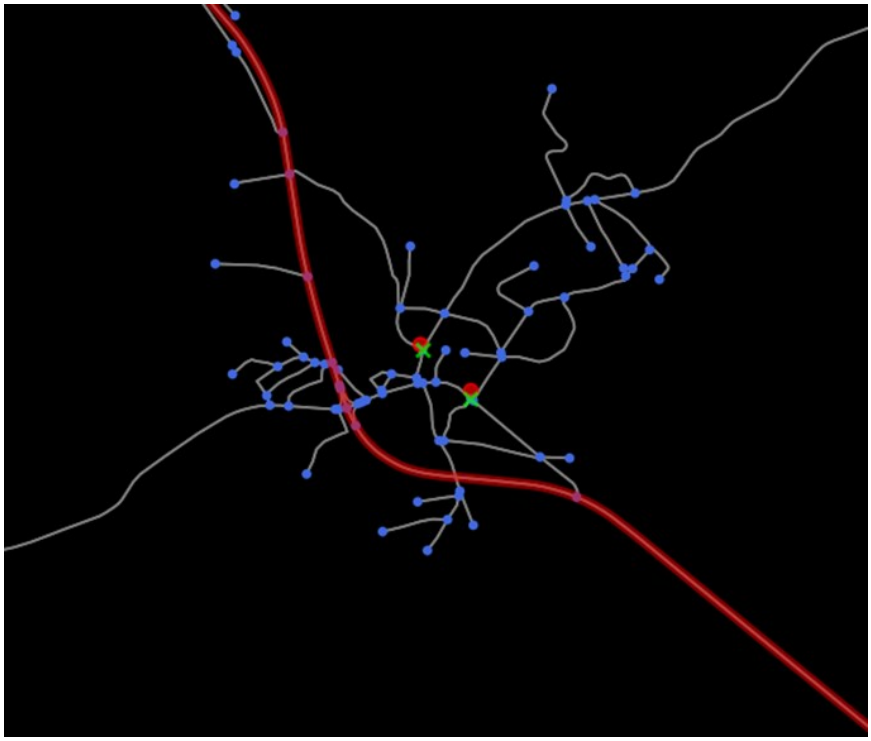
Obr. 1 – Nejkratší cesta



Obr. 2 – Detailní výřez nejkratší cesty



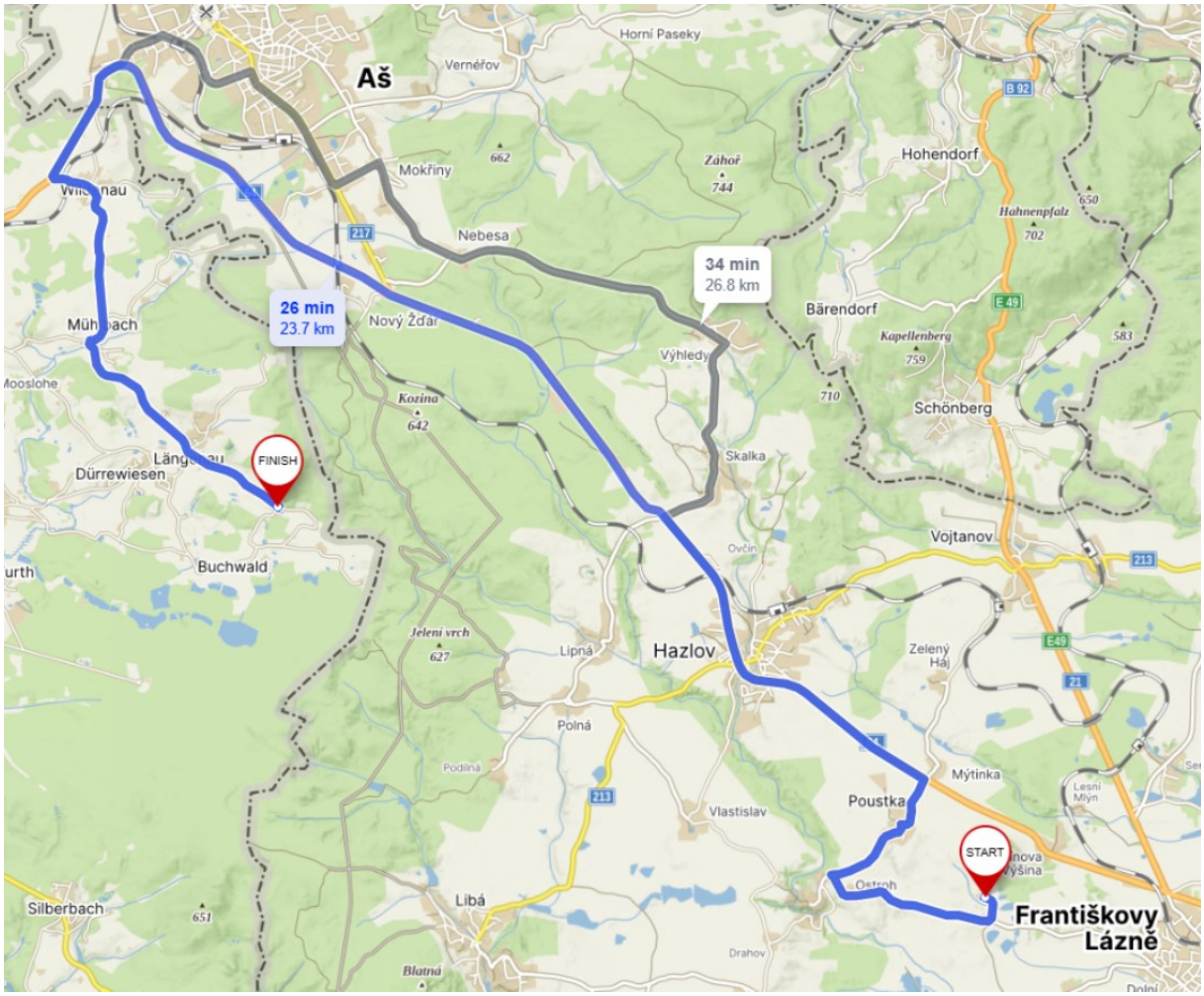
Obr. 3 – Nejrychlejší cesta



Obr. 4 – Detailní výřez nejrychlejší cesty

Tabulka 1 – Porovnání dvou různých tras

Trasa	Vzdálenost	Čas
Nejkratší	23,33 km	25,15 min
Nejrychlejší	23,73 km	23,49 min



Obr. 5 – Nejrychlejší cesta mezi zvolenými restauracemi zjištěná pomocí stránky mapy.cz

Nalezení nejkratších cest mezi všemi dvojicemi uzlů

Pro nalezení nejkratších cest mezi všemi dvojicemi uzlů byla vytvořena nová funkce (*all_pairs*), jež funguje na tomto principu:

Procházení všech uzlů ve slovníku (*start_node*)

Pro každý uzel *start_node* se prochází přes všechny uzly (*end_node*) ve slovníku

Vyloučení párů uzlů se stejným ID

Použití algoritmu Dijkstra pro nalezení nejkratší cesty mezi *start_node* a *end_node*

Vytvoření cesty *path* pomocí informací o rodičích vrácených Dijkstra algoritmem

Výpis nejkratší cesty pro daný pár uzlů

Pro tuto část bylo trochu zmenšeno území, neboť jinak existovalo mnoho různých kombinací a trvalo dlouho, než nám program vygeneroval výsledky. Ve výsledcích je vždy uvedeno, která dvojice uzlů se řeší a také kterými uzly daná nejkratší cesta prochází (viz obr. 6).

```
Start Node: 451868004, End Node: 451867999, Shortest Path: [451868004, 451867999]
Start Node: 451868004, End Node: 451868038, Shortest Path: [451868004, 451868038]
Start Node: 451868004, End Node: 451868732, Shortest Path: [451868004, 451868038, 451868732]
Start Node: 451868004, End Node: 451868932, Shortest Path: [451868004, 451868038, 451868932]
```

Obr. 6 – Ukázka výsledku nejkratších cest mezi všemi dvojicemi uzlů

Řešení úlohy pro grafy se záporným ohodnocením

Byla také řešena úloha pro graf se záporným ohodnocením. Konkrétně šlo o to, že rychlost na určitém typu cestu byla nastavena záporná (secondary_link -90km/h). Byl využit Bellman-Ford algoritmus, u kterého mohou být některé hrany ohodnoceny i záporně.

Algoritmus Bellmann-Ford

Bellman-Fordův algoritmus je algoritmus pro nalezení nejkratší cesty z jednoho zadaného vrcholu v grafu. Tento algoritmus pracuje s grafy, které mohou mít záporné váhy hran, ale zároveň dokáže detekovat negativní cykly. Klíčovým principem algoritmu je iterativní relaxace hran grafu, dokud nejsou nalezeny nejkratší vzdálenosti ke všem vrcholům. Tento algoritmus používá celkem tři kroky – inicializace grafu, opakovaná relaxace hran a kontrola záporných cyklů.

Princip algoritmu Bellmann-Ford:

1. Inicializační fáze:
   Počet vrcholů je označen U (unvisited)
   Nastavení nekonečné vzdálenosti pro všechny uzly, nastavení předchůdců na "None".
2. Relaxační fáze:

Algoritmus prochází cyklem a opakuje následující kroky pro maximálně U-1 iterací:

Pro každý vrchol a jeho souseda v grafu je zkontrolováno, zda by vzdálenost k sousedovi mohla být zlepšena procházením přes aktuální vrchol. Pokud ano, aktualizuje se vzdálenost a nastaví se předchůdce na aktuální vrchol.

3. Kontrola negativních cyklů:

   Dodatečný cyklus pro kontrolu negativních cyklů. Pokud je nalezena kratší cesta, znamená to přítomnost negativního cyklu v grafu.

Nalezení minimální kostry některou z metod

Předpokladem je, že graf musí být neorientovaný, souvislý a také bez záporných hran. Úloha není jednoznačná, existuje více minimálních koster.

Kruskalův algoritmus

Kruskalův algoritmus, který byl poprvé publikován Josephem Kruskalem v roce 1956, slouží k nalezení minimální kostry grafu (takové kostry, u které je součet jejích hran minimální). Tento algoritmus patří mezi greedy algoritmy, což znamená, že v každém kroku vybírá nejlepší možnost na základě aktuální situace. Tím dosahuje optimálního výsledku. Algoritmus je efektivní a obvykle používá strukturu union-find pro rychlé určení, zda vrcholy patří ke stejnému stromu.

Kruskalův algoritmus princip:

1. Seřazení hran podle vah:

   Nejprve se všechny hrany grafu seřadí podle váhy od nejnižší k nejvyšší.

2. Inicializace lesa:

   Vytvoří se prázdný les (množina stromů), kde každý vrchol je samostatným stromem.

3. Procházení hran:

   Hrany se postupně zpracovávají podle seřazení.

   Pro každou hranu se zjistí, zda spojuje dva vrcholy, které jsou v různých stromech. To znamená, že přidání této hrany do lesa nespojí dva stromy a nevytvoří cyklus.

Pokud je toto splněno, hrana je přidána do lesa, a oba spojené stromy jsou sloučeny do jednoho.

4. Opakování kroku 3:

   Tento proces se opakuje, dokud nejsou všechny vrcholy spojeny nebo les obsahuje pouze jeden strom.

5. Výstup:

   Minimální kostra grafu je tvořena hranami, které byly postupně přidávány do lesa.

Využití heuristik Weighted Union a Path Compression

Weighted UNION a Path Compression jsou dvě heuristiky vylepšující efektivitu Union-Find operací. Při Weighted UNION je lepší připojovat kratší strom k delšímu. Typické je, že kořen menšího stromu je připojen na kořen většího stromu podle r (hodnost stromu). Pokud jsou kořeny stejné, tak je vybrán libovolný z nich. Path Compression se provádí při hledání kořene a jde o postupné propojení všech uzlů stromu ke kořenu. Při této operaci se každému prvku na cestě ke kořenu upraví odkaz na kořen/prarodiče. Je tedy snížena výška stromu a zároveň urychlena operace FIND. Existují dvě varianty – s jedním nebo se dvěma průchody. V praxi je preferována varianta se dvěma průchody, kdy výška klesne na jednu polovinu stromu.

Součástí Kruskalova programu jsou tyto funkce:

1. Union

   Spojuje dvě oddělené množiny reprezentované kořeny u a v.

   Využívá funkci find k nalezení kořenů množin s kompresí cesty.

   Pokud jsou kořeny root_u a root_v různé (tj. u a v jsou v různých podstromech), rozhoduje se, který podstrom připojit k druhému na základě hodnosti podstromů.

   Pokud jsou hodnosti různé, menší podstrom se připojí k většímu, a pokud jsou hodnosti stejné, provede se libovolný výběr, ale hodnost výsledného podstromu se zvýší.

2. Find

   Hledá kořen množiny obsahující prvek u.

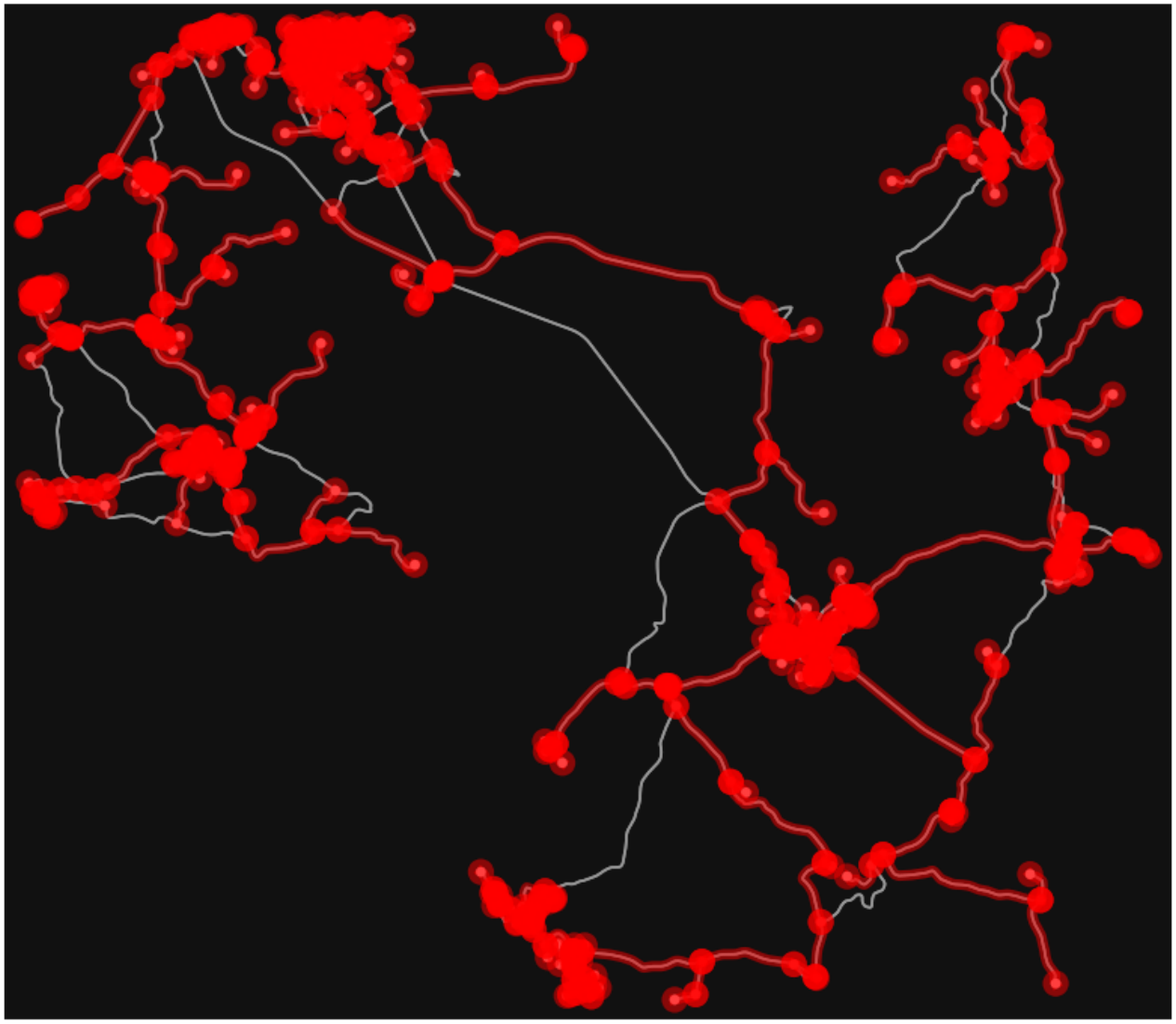
   Využívá kompresi cesty k zploštění struktury stromu aktualizací ukazatelů rodiče podél cesty k kořeni.

   Pomáhá snížit výšku stromu a tím zrychlit operaci find.

3. Make set

Inicializuje jednoprvkovou množinu pro prvek u, nastavením jeho rodiče na něj samotného (p[u] = u) a hodnosti na 0.

Výsledná váha minimální kostry grafu nám vyšla 138,04 min a finální grafickou podobu lze vidět na obrázku 7. Dále obrázek 8 zachycuje detailní výřez určité oblasti.



Obr. 7 – Minimální kostra grafu

Obr. 8 – Detailní výřez minimální kostry grafu

Závěr

V programu Visual Studio Code byla nalezena nejkratší a také nejrychlejší cesta (v závislosti na přidělených maximálních rychlostech, kterými lze po určitých typech silnice jet) mezi dvěma zvolenými restauracemi pomocí Dijkstra algoritmu. Zatímco nejkratší cesta procházela 42 uzly, nejrychlejší 41 uzly, společných uzlů pak měly obě cesty celkem 33. Stejná úloha byla provedena rovněž na webových stránkách mapy.cz, jež vyšla pravděpodobně totožně jako nejrychlejší cesta. Časový rozdíl mezi výsledkem zjištěným pomocí Dijkstra algoritmu a stránkami mapy.cz byl zjištěn minimální. Konkrétně jde o odchylku přibližně jeden a půl minuty. Tento rozdíl může být způsoben například tím, že mapy.cz zahrnují do výpočtů mnohem více parametrů než pouze samotnou maximální rychlost. Nalezeny byly i nejkratší cesty mezi všemi dvojicemi uzlů - v tomto případě však bylo zmenšeno zájmové území, jelikož by program generoval zbytečně velké množství kombinací.

S využitím Bellman-Ford algoritmu byla provedena úloha pro graf se záporným ohodnocením. Byla tedy nastavena záporná rychlost na určitém typu silnice. Následně byla vytvořena minimální kostra grafu s využitím dvou heuristik, které vylepšují efektivitu. Byly využity Weighted UNION a Path Compression. Výsledná váha (součet všech vah hran) minimální kostry grafu v jednom z případů vyšla 138,04 min.

Zdroje

BAYER, T. (2023): Úvod do grafových algoritmů, prezentace k předmětu Geoinformatika, https://web.natur.cuni.cz/~bayertom/images/courses/Geoinf/geoinf_grafy1.pdf (28. 12. 2023).

BAYER, T. (2023): Nejkratší cesty v grafu, prezentace k předmětu Geoinformatika, https://web.natur.cuni.cz/~bayertom/images/courses/Geoinf/geoinf_grafy2.pdf (28. 12 .2023).

AUTOGIS (2020): Retrieving OpenStreetMap data, https://autogis-site.readthedocs.io/en/2019/notebooks/L6/retrieve_osm_data.html?fbclid=IwAR3TSoJ1DEt_HyROjx8LKXUGur1t7M0ItixhRjfrBbLmb9w6mfSaOT0FmbY (30. 12. 2023)

Algoritmy.net (2016): Dijkstrův algoritmus, https://www.algoritmy.net/article/5108/Dijkstruv-algoritmus#google_vignette (30. 12. 2023)

Algoritmy.net (2016): Kruskalův algoritmus, https://www.algoritmy.net/article/1417/Kruskaluv-algoritmus (30. 12. 2023)

Algoritmy.net (2016): Bellman_fordův algoritmus, https://www.algoritmy.net/article/7478/Bellman-Forduv-algoritmus (30. 12. 2023)