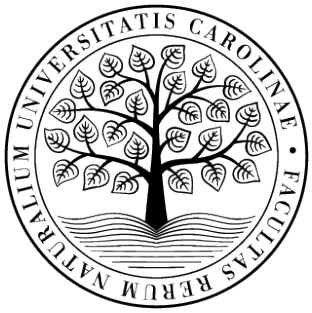
Geoinformatika
Úloha 3: Nejkratší cesta grafem

František Macek, Josef Zátka



Univerzita Karlova
Přírodovědecká fakulta
Katedra aplikované geoinformatiky a kartografie

1 Úvod

V této úloze je implementován program pro nalezení nejkratší cesty v grafu pomocí **Dijkstrova algoritmu**. Vypočítává nejkratší a nejrychlejší trasy mezi dvěma městy v rámci silniční sítě, která je reprezentována jako graf. Implementuje také další algoritmy pro výpočty v grafech: **Bellman-Fordův algoritmus**, který si poradí s grafy se zápornými vahami a vypočítává nejkratší cesty, a **Kruskalův algoritmus**, který nalezne minimální kostru grafu.

2 Zadání

Implementujte Dijkstrův 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**
Dijkstrův 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**

3 Řešení

3.1 Vstupní data a tvorba grafu

Vstupní data v podobě silniční sítě a databáze uzlů představujících města byla získána z OpenStreetMap prostřednictvím webu GeoFabrik.de, který umožněje jejich stažení ve formátu *shapefile*. Takto získané shapefily byly následně upraveny v prostředí ArcGIS Pro tak, aby (za účelem zrychlení výpočtů) zahrnovaly pouze data z Libereckého kraje. Shapefily byly převedeny do podoby grafů, čímž rozumíme datový typ defaultdict se strukturou

{uzel1_id: ({sousední_uzel_id: [VZDÁLENOST, PŘÍMÁ VZDÁLENOST, MAX.RYCHLOST],
...další sousední uzel atd..., [souřadnice_uzel_id])}, uzel2_id ...}.

Při exportu, jak je zřejmé z řádků výše, byly zachováno atributy euklidovské délky komunikace (VZDÁLENOST), nejkratší vzdušné vzdálenosti mezi počátečním a koncovým bodem komunikace (PŘÍMÁ VZDÁLENOST) a také atribut maximální povolené rychlosti (MAX. RYCHLOST).

Dále je vytvořen slovník uzlů, které reprezentují sídla. Takové body byly vytvořeny v místech průniku bodových prvků sídel z databáze OSM s nejbližší linií silnice. Celý výsledek (tj. graf a seznam měst) je uložen jako soubor formátu *json* pro snadnější načtení a použití v dalším běhu programu. Tento proces realizuje skript *shp_to_graph.py*.

3.2 Algoritmus Dijkstra

Hlavním úkolem zadané úlohy je implementace Diskstrova algoritmu pro vyhledávání njekratší cesty v grafu. Dijkstrův algoritmus je jedním z nejefektivnějších způsobů, jak najít nejkratší cestu od jednoho uzlu ke všem ostatním v grafu, pokud neobsahuje hrany se zápornými hodnotami. Funguje tak, že postupně uvolňuje (relaxuje) hrany a aktualizuje nejkratší známé vzdálenosti k jednotlivým uzlům. Na rozdíl od některých jiných algoritmů není potřeba uzly speciálně označovat. V podstatě kombinuje strategii prohledávání do šířky (BFS), relaxaci hran a prioritní frontu pro efektivní výběr nejbližšího uzlu k dalšímu zpracování (Algoritmy.net 2024b).

3.2.1 Princip fungování algoritmu Dijkstra

1. Inicializace:
   - Každému uzlu je přiřazena počáteční vzdálenost nekonečno, kromě startovního uzlu, tomu je nastavena hodnota 0.
   - Je vytvořen seznam rodičovských uzlů (parents), kam bude ukládáno, odkud se do daného uzlu přišlo.
   - Je iniciována prioritní fronta (min-heap) pro vybírání uzlů s nejmenší známou vzdáleností.
2. Zpracování uzlů ve while cyklu:
   - Vždy je vybrán uzel s nejmenší známou vzdáleností z prioritní fronty.
   - Pokud je tato vzdálenost již delší než dříve nalezená, je přeskočena.
   - Pro každý sousední uzel (*neighbour*) je spočítána nová možná vzdálenost přes aktuální uzel.
   - Pokud je tato nová vzdálenost kratší než ta, kterou jsme dosud znali, je aktualizována a jsou nastaveni rodiče uzlu.
   - Sousední uzly jsou vloženy do prioritní fronty, aby byly později zpracovány.
3. Pokračování, dokud njesou zpracovány všechny dosažitelné uzly
4. Výsledek:
   - Pole vzdáleností (dists) obsahuje nejkratší vzdálenosti od startu ke všem uzlům.
   - Pole rodičů (parents) nám umožní rekonstruovat nejkratší cestu zpět sledováním rodičů od cílového uzlu.

3.2.2 Vlastní implementace algoritmu Dijkstra

Samotná implementace Dijkstrova algoritmu je rozdělena do dvou tříd: **Graph** a **ShortestPath**.

Třída `Graph` slouží k uchování struktury grafu, kde hrany odpovídají silnicím a některé z uzlů přísluší městům (některé uzly jsou pouze křížením či napojením úseků silnic). Obsahuje metody:

- `populate_graph(graph_data)` – naplní graf daty ze souboru formátu *json* (viz kapitola 3.1).
- `populate_cities(city_nodes)` – přiřadí uzlům názvy měst dle slovník z téhož souboru.
- `get_neighbours(node)` – vrací sousedy daného uzlu.
- `get_city(city_id)` – vrací název města podle jeho ID.

Každý uzel v grafu má seznam sousedních uzlů, kde jsou uloženy váhy hran (viz atributy v kapitole 3.1).

Třída `ShortestPath` **(implementace Dijkstrova algoritmu)** obsahuje metody pro výpočet nejkratší cesty.

Metoda `calc_weight(weights, mode="basic")`

- Vypočítá váhu hrany podle zvoleného režimu:
  - **basic** – využívá euklidovskou vzdálenost mezi uzly.
  - **advanced** – bere v úvahu nejen vzdálenost, ale také parametr klikatosti a maximální rychlosti a přepočítává vzdálenost (váhu) dle vzorce

$$\text{váha}_{\text{advanced}} = \frac{\text{skutečná délka silnice} \times \frac{\text{skutečná délka silnice}}{\text{přímá vzdálenost mezi poč. a koncovým bodem}}}{\text{maximální rychlost}}$$

Metoda `dijkstra(start, end, mode="basic")`

- Implementuje algoritmus Dijkstra dle principu popsaného v kapitole 3.2.1.
- Vrací nejkratší vzdálenost mezi startovním a cílovým uzlem a pole `parents`, které umožňuje rekonstruovat nalezenou cestu.

3.3 Řešení pro grafy se záporným hodnocením (Bellman-Ford)

Algoritmus Dijkstra neumožňuje pracovat s grafy, které obsahují hrany se zápornými vahami. Pokud graf obsahuje takové hrany, algoritmus Dijkstra může poskytnout nesprávné výsledky nebo nemusí skončit vůbec. Proto je pro takové grafy vhodnější použít **Bellman-Fordův algoritmus**, který dokáže najít nejkratší cesty i v grafu se zápornými hranami a zároveň detekovat existenci **cyklů se zápornou délkou**.

Záporná hrana je hrana grafu, jejíž váha je záporná. Tyto hrany samy o sobě nejsou problém, ale mohou vést ke vzniku **záporného cyklu**.

Záporný cyklus je uzavřená cesta v grafu, jejíž celková váha je záporná. Pokud by existoval v síti silnic, znamenalo by to, že se lze nekonečně zrychlovat pouhým opakováním průchodu tímto cyklem. To vede k tomu, že v grafu není dobře definovaná nejkratší cesta, protože ji lze teoreticky zkracovat donekonečna.

3.3.1 Princip Bellman-Fordova algoritmu

1. Inicializace:
   - Ke všem uzlům je přiřazena počáteční vzdálenost nekonečno, kromě startovního uzlu, tomu je nastavena hodnota 0.
   - Je vytvořen seznam rodičovských uzlů (`parents`), do kterého se ukládá, odkud se do daného uzlu přišlo.

2. Relaxace hran:
    - Algoritmus opakuje $(n-1)$ iterací, kde n je počet uzlů.
    - V každé iteraci se projde každá hrana a pokud lze přes její počáteční uzel zlepšit známou vzdálenost sousedního uzlu, aktualizuje se jeho vzdálenost i rodič.
3. Detekce záporných cyklů:
    - Po dokončení relaxací se algoritmus pokusí provést další průchod hranami.
    - Pokud by bylo možné ještě zlepšit vzdálenost, znamená to, že v grafu existuje cyklus se zápornou délkou a algoritmus tuto skutečnost nahlásí (Algoritmy.net 2024a).

3.3.2 Vlastní implementace Bellman-Fordova algoritmu

Vlastní implementace je součástí třídy `ShortestPath`, kde metoda `bellman_ford(start, mode)` provádí výpočet nejkratší cesty. Algoritmus probíhá tak jak je popsáno v předchozí kapitole:

- Inicializuje se pole vzdáleností a rodičovských uzlů stejně jako u Dijkstrova algoritmu.
- V hlavní smyčce proběhne $(n-1)$ iterací, přičemž v každé se provede relaxace všech hran v grafu.
- Po skončení smyčky proběhne kontrola přítomnosti záporného cyklu. Pokud existuje, metoda ohlásí chybu.
- Metoda vrací seznam nejkratších vzdáleností od startovního uzlu ke všem ostatním uzlům (dists) a seznam rodičovských uzlů (parents), který umožňuje rekonstruovat nejkratší cesty.

3.4 Nalezení minimální kostry grafu (Kruskalův algoritmus)

Kruskalův algoritmus je jednou z metod k nalezení minimální kostry grafu (*Minimum Spanning Tree*), tedy podgrafu s nejmenším součtem vah hran, který spojuje všechny uzly (Algoritmy.net 2024c).

3.4.1 Princip Kruskalova algoritmu

- Inicializace: Seřazení všech hran podle váhy a inicializace disjunktních množin.
- Výběr hran: Prochází se hrany v pořadí od nejnižší váhy a přidávají se do kostry, pokud nespojí uzly, které jsou už propojeny.
- Výstup: Výsledkem je kostra obsahující $V-1$ hran, kde V je počet vrcholů.

3.4.2 Vlastní implementace Kruskalova algoritmu

Vlastní implementace Kruskalova algoritmu je založena na zpracování hran v pořadí podle jejich váhy a využívá datovou strukturu **disjunktních množin** k efektivnímu spojování komponent grafu. Implementace probíhá v několika fázích:

1. Vytvoření seznamu hran – projdou se všechny sousedy každého uzlu a vytvoří se seznam hran včetně jejich vah.
2. Setřídění hran – hrany jsou seřazeny vzestupně podle váhy, což zajišťuje, že první vybírané hrany budou mít nejmenší váhu.
3. Inicializace disjunktních množin – každý uzel je na začátku samostatnou komponentou, což umožňuje snadno sledovat, které uzly jsou propojené.
4. Výběr hran pro minimální kostru – prochází se hrany v pořadí podle váhy a přidávají se do výsledné kostry pouze tehdy, pokud nespojují uzly, které již jsou ve stejné komponentě.
5. Sloučení komponent – pokud přidání hrany propojí dvě různé komponenty, dojde k jejich sloučení pomocí operace sjednocení.

K fungování metody Kruskal je v naší implementaci nutná existence třídy **DisjointSet**, která umožňuje správu množin uzlůu. Metodami této třídy jsou:

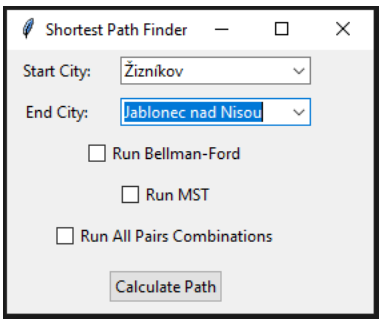
- `find(node)` – vrací kořen stromu, ke kterému uzel patří. Používá optimalizaci **path compression** - pokud uzel není svým vlastním rodičem, voláme find() rekurzivně na jeho rodiče. Při návratu aktualizujeme rodiče všech uzlů na cestě tak, aby ukazovaly přímo na kořen.
- `union(root1, root2)` – spojuje dvě množiny a využívá heuristiku **weighted union**, která udržuje stromové struktury vyvážené.

3.5 Nalezení nejkratších cest mezi všemi dvojicemi uzlů

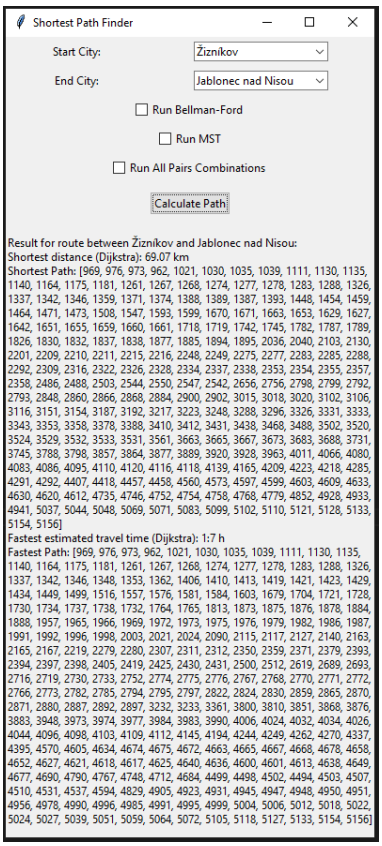
Nalezení nejkratších cest mezi všemi dvojicemi uzlů je realizováno statickou metodou nazvanou `calculate_combinations`, která pomocí funkce *combinations* z knihovny *itertools* vytváří všechny možné kombinace dvojic měst, pro které je následně vypočítána a rekontruována cesta pomocí Dijkstrova algoritmu. Výsledky jsou pro každou dvojici měst uloženy do výstupního souboru formátu *json*. U metody je na vstupu možné nastavit parametr *limit*, který určí pro kolik prvních n měst budou kombinace tvořeny - to je využitelné především při testování funkčnosti metody z důvodu časové náročnosti výpočtu pro všechna města.

4 Architektura programu a uživatelské prostředí

Hlavní běh programu zajišťuje skript *main.py*, který načítá data do grafu a spouští uživatelské rozhraní uložené ve skriptu *interface.py*. V dialogovém okně aplikace uživatel volí počáteční a cílovou destinaci a také vybírá, jaké dodatečné výpočty mají proběhnout (obrázek 1). Po volbě požadovaného spojení uživatelem jsou spuštěny algoritmy uložené ve skriptu *algorithms.py* a výsledky jsou následně vypsány uživateli přímo v dialogovém okně (obrázek 2). Samostatně stojí skript *shp_to_graph.py*, který je nutné ručně spustit, pokud uživatel požaduje zpracovat nová data uložená ve formátu *shapefile* do formátu grafu.



Obrázek 1: Dialogové okno aplikace Shortest Path Finder



Obrázek 2: Výsledky vzorového výpočtu

5 Testování a výsledky

Test proběhl na dvojici tras. První trasou je spojení Žižníkova a Jablonce nad Nisou napříč Libereckým krajem, druhou trasou je spojení Liberec-Semily. V tabulce 2 jsou zaneseny výsledky námi implementovaného algoritmu Dijkstra s nejběžnějšími vyhledávači trasy Google Maps a Mapy.cz.

	Žižníkov - Jablonec n. N.	Liberec - Semily
Implementovaná Dijkstra	69,07 km, 1:07 hod	41,59 km, 0:41 hod
Google Maps	64,3 km, 0:58 hod	43,4 km, 0:40 hod
Mapy.cz	63,7 km, 0:56 hod	38,7 km, 0:44 hod

Tabulka 2: Porovnání výsledků vypočítaných vytvořenou metrikou a běžnými vyhledávači trasy

Z tabulky výsledků je zřejmé, že lepší shody s běžnými vyhledávači trasy je dosaženo pro trasu Liberec – Semily, kdy se výsledky naší implementace Dijkstrova algoritmu liší pouze o 1, respektive 3 minuty. Naopak pro trasu Žižníkov – Jablonec nad Nisou jsou rozdíly vyšší. To může být způsobeno nově otevřeným úsekem rychlostní silnice mezi Novým Borem a Svorem nebo jinou neznámou příčinou.

6 Závěr

V této úloze jsme implementovali Dijkstrův algoritmus pro hledání nejkratší cesty v silniční síti Libereckého kraje a porovnali jeho výsledky s běžně používanými navigačními systémy.Řešení bylo rozšířeno o Bellman-Fordův algoritmus pro práci s grafy obsahujícími záporné hrany a Kruskalův algoritmus pro nalezení minimální kostry grafu. Také byla implementována metoda pro výpočet nejrychlejších tras mezi všemi dvojicemi měst. Testování ukázalo, že vytvořená metrika poskytuje výsledky blížící se komerčním navigačním systémům, přičemž menší rozdíly mohou být způsobeny aktualizací silniční sítě a různými optimalizačními přístupy jednotlivých komerčních systémů.

Zdroje

Algoritmy.net (2024a). *Bellman-Fordův algoritmus*. Dostupné online, navštíveno 2. února 2025. URL: `https://www.algoritmy.net/article/7478/Bellman-Forduv-algoritmus`.

— (2024b). *Dijkstrův algoritmus*. Dostupné online, navštíveno 1. února 2025. URL: `https://www.algoritmy.net/article/5108/Dijkstruv-algoritmus`.

— (2024c). *Kruskalův algoritmus*. Dostupné online, navštíveno 6. února 2025. URL: `https://www.algoritmy.net/article/1417/Kruskaluv-algoritmus`.