

Technická zpráva: Nejkratší cesta grafem

Mykhailo Radchenko, Philip-Július Otto

2. prosince 2025

1 Úvod

Cílem této úlohy bylo implementovat algoritmy pro hledání nejkratších cest v grafu nad reálnými daty silniční sítě. Práce zahrnuje přípravu dat z OpenStreetMap, implementaci vlastních datových struktur a řešení optimalizačních úloh.

Nad rámec základního zadání byly ve výsledném skriptu `U3_final.py` úspěšně vyřešeny následující bonusové úlohy:

- **Bonus 1:** Řešení úlohy pro grafy se záporným ohodnocením (Bellman-Fordův algoritmus).
- **Bonus 2:** Nalezení nejkratších cest mezi všemi dvojicemi uzlů (Floyd-Warshallův algoritmus).
- **Bonus 4, 5, 6:** Nalezení minimální kostry grafu Kruskalovým algoritmem (včetně heuristik *Weighted Union* a *Path Compression*).
- **Bonus 3:** Nalezení minimální kostry grafu Jarníkovým-Primovým algoritmem.

2 Technický popis implementace

Skript je napsán v jazyce Python a je rozdělen do logických bloků: definice vlastních tříd, implementace algoritmů a hlavní výkonná část.

2.1 Vlastní datové struktury

Pro účely výuky a optimalizace nebyly použity vestavěné knihovny pro prioritní fronty, ale byly implementovány vlastní třídy:

- **Třída MinHeap:** Implementuje prioritní frontu pomocí pole. Obsahuje metody `push` (vložení a probublání nahoru `_bubble_up`) a `pop` (odebrání kořene a probublání dolů `_bubble_down`). Tato třída je využita v Dijkstrově a Primově algoritmu.
- **Třída UnionFind:** Implementuje strukturu Disjoint Set Union (DSU) pro Kruskalův algoritmus. Pro maximální efektivitu obsahuje obě požadované heuristiky:
 - *Path Compression* (komprese cesty): Při volání metody `find` se rekursivně přepojují uzly přímo ke kořeni.
 - *Weighted Union*: Při spojování (`union`) se menší strom vždy připojuje pod větší strom na základě *ranku*.

2.2 Příprava a zpracování dat

Vstupní data jsou stahována pomocí knihovny `osmnx`.

1. **Stažení:** Je stažena silniční síť typu `drive` pro oblast „Jihomoravský kraj“.
2. **Projekce:** Graf je transformován ze souřadnic WGS84 (stupně) do UTM (metry) pomocí `ox.project_graph`, což umožňuje přesné výpočty vzdáleností.
3. **Topologie:** Graf je převeden na neorientovaný (`to_undirected`).

3 Hlavní úloha: Dijkstrův algoritmus

Algoritmus `dijkstra_algorithm` využívá naši třídu `MinHeap`. Na vstupu přijímá graf, startovní a cílový uzel a atribut váhy.

3.1 Metriky ohodnocení

Pro každou hranu jsou vypočteny tři různé váhy:

- **Eukleidovská vzdálenost** (`weight_euclid`): Fyzická délka silnice v metrech.
- **Teoretický čas** (`weight_time1`): Délka dělená maximální povolenou rychlostí (dle typu silnice, např. dálnice 130 km/h, obec 50 km/h).
- **Čas s vlivem klikatosti** (`weight_time2`): Rychlosť je penalizována faktorem klikatosti $f = \frac{l(P)}{dist(u,v)}$. Skutečná rychlosť je $v_{real} = v_{max}/f$.

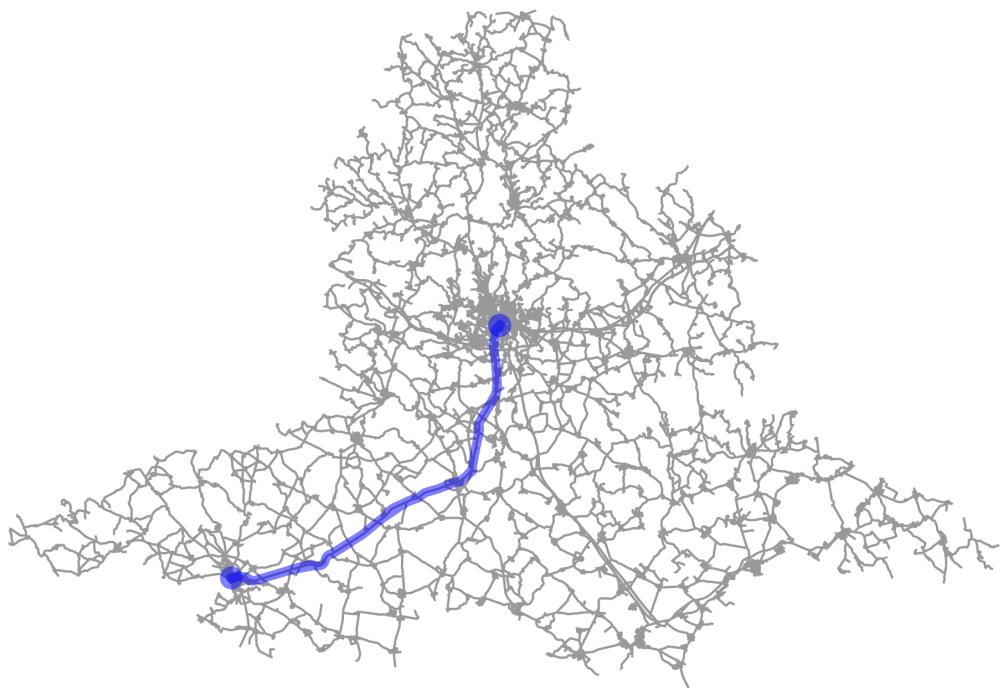
3.2 Výsledky testování

Testování proběhlo na dvou trasách. Výsledky porovnání našich algoritmů s reálnými daty (Google Maps) jsou uvedeny v tabulce níže.

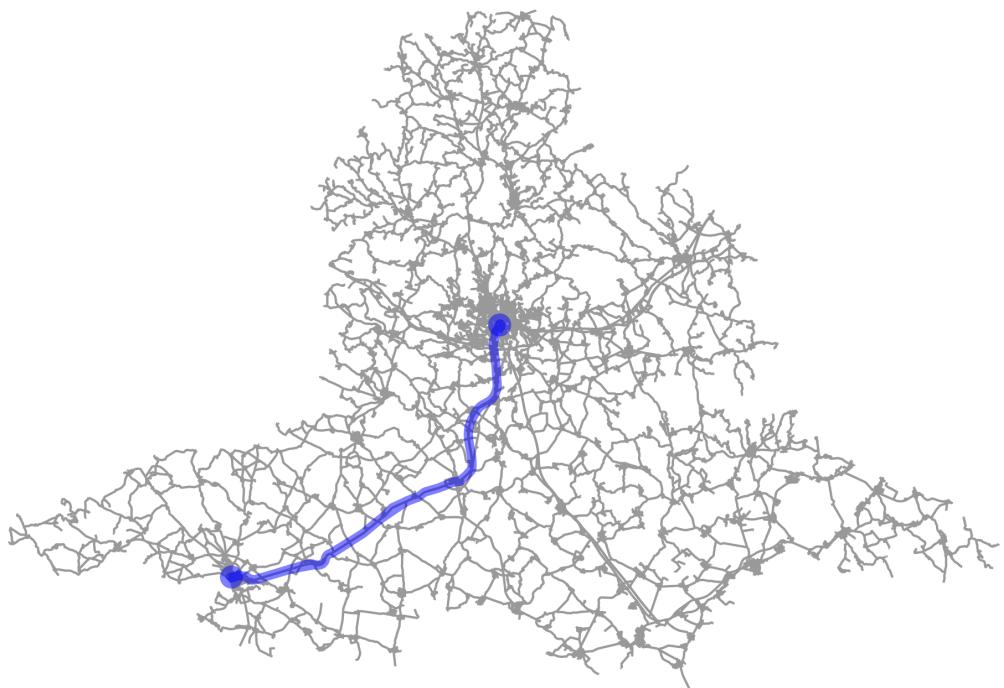
Tabulka 1: Výsledky Dijkstrova algoritmu a porovnání s realitou

Trasa	Metrika	Vypočtená hodnota	Google Maps
Brno → Znojmo	Vzdálenost	65,71 km	66,9 km
	Čas (Teoretický)	41,3 min	-
	Čas (Klikatost)	42,0 min	47 min
Vyškov → Blansko	Vzdálenost	34,58 km	34,6 km
	Čas (Teoretický)	23,3 min	-
	Čas (Klikatost)	24,5 min	38 min

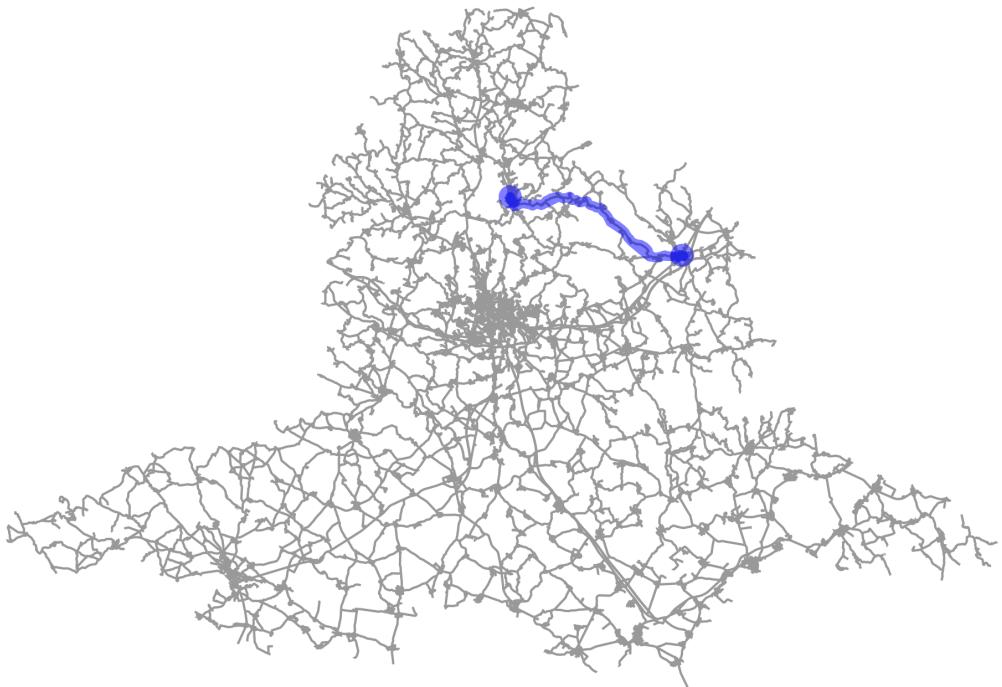
Jak je vidět, zavedení penalizace za klikatost mírně prodloužilo dojezdový čas, což přibližuje model realitě, ačkoliv absence dopravních dat (křižovatky, provoz) stále činí výsledek optimis-tičtějším.



Obrázek 1: Vizualizace trasy Brno - Znojmo (nejkratší Eukleidovská vzdálenost)



Obrázek 2: Vizualizace trasy Brno - Znojmo (nejrychlejší cesta s vlivem klikatosti)



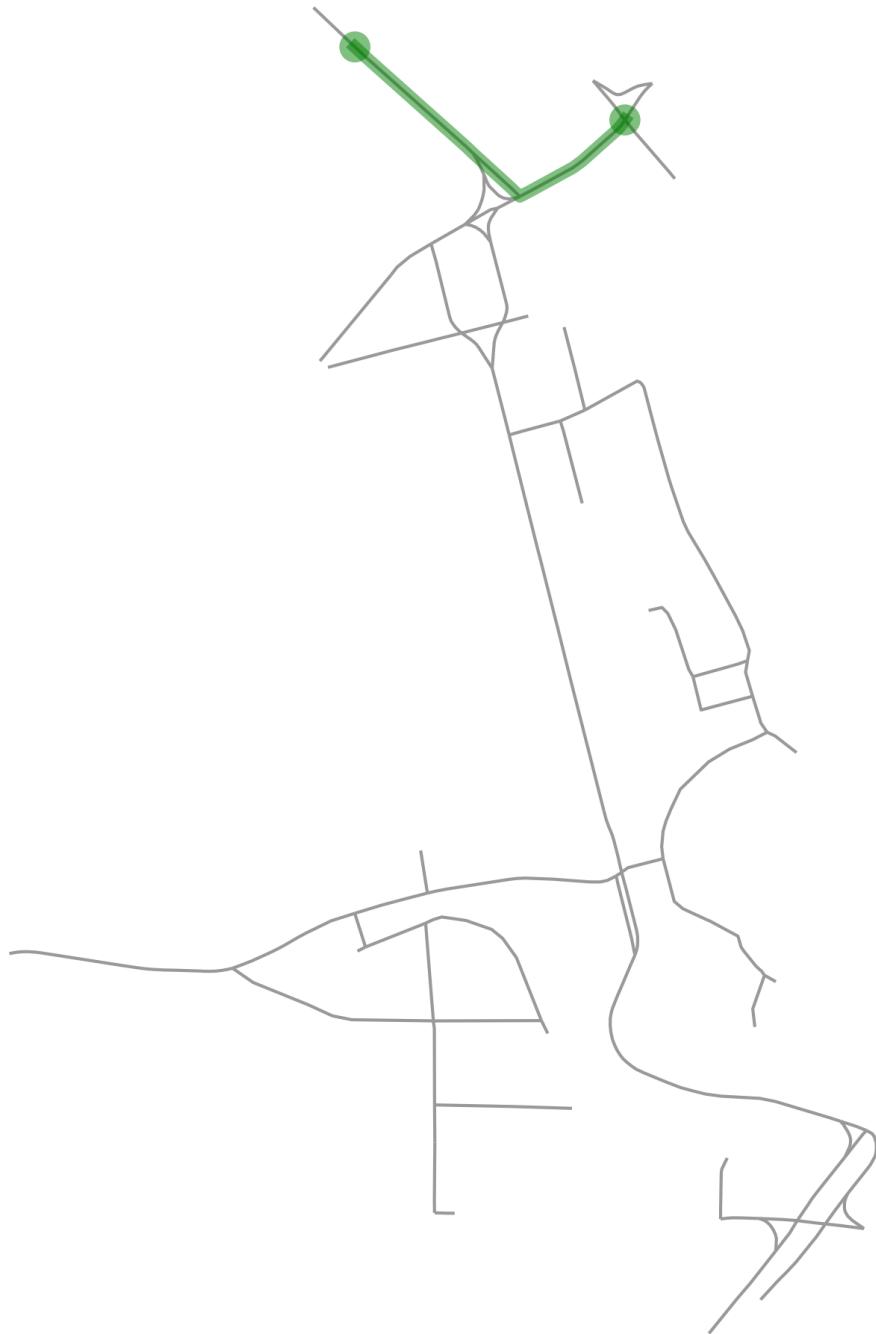
Obrázek 3: Vizualizace trasy Vyškov - Blansko

4 Implementace bonusových úloh

4.1 Bonus 1: Bellman-Fordův algoritmus

Tato část demonstруje hledání cesty v grafu se záporným ohodnocením.

- **Implementace:** Funkce `bellman_ford_algorithm` provádí $|V| - 1$ relaxací hran a následně detekuje záporné cykly.
- **Testovací scénář:** Byl vytvořen malý podgraf (okolí centra Brna, 1 km), ve kterém byla uměle vytvořena hrana se zápornou váhou (-60 sekund). Algoritmus úspěšně nalezl cestu s časem 0,24 min.



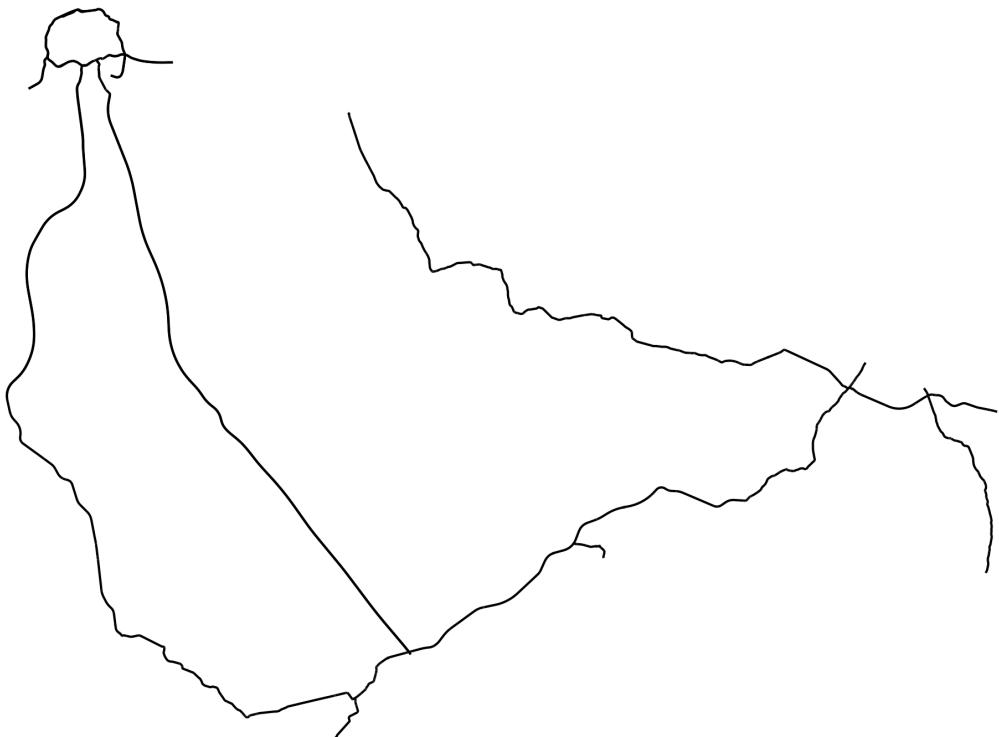
Obrázek 4: Výsledek Bellman-Fordova algoritmu na modifikovaném podgrafu

4.2 Bonus 2: Floyd-Warshallův algoritmus

Cílem bylo nalézt matici nejkratších cest mezi všemi dvojicemi uzlů. Algoritmus využívá dynamické programování na principu: $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$.

- **Optimalizace:** Vzhledem ke kubické složitosti $O(V^3)$ nebylo možné spustit výpočet na celém grafu (37 tisíc uzlů). Byla provedena extrakce **páteřní sítě** (pouze dálnice a silnice I. třídy), čímž se počet uzlů zredukoval na 722.
- **Výsledek:** Na redukovaném grafu proběhl výpočet za přibližně 5 sekund. Byla vygenerována kompletní matice vzdáleností.

- **Verifikace:** Z výsledné matice byla pro kontrolu vybrána cesta mezi dvěma vzdálenými uzly v síti. Hodnota **80,66 min** odpovídá reálnému času přejezdu napříč Jihomoravským krajem po silnicích I. třídy, což potvrzuje korektnost výpočtu.



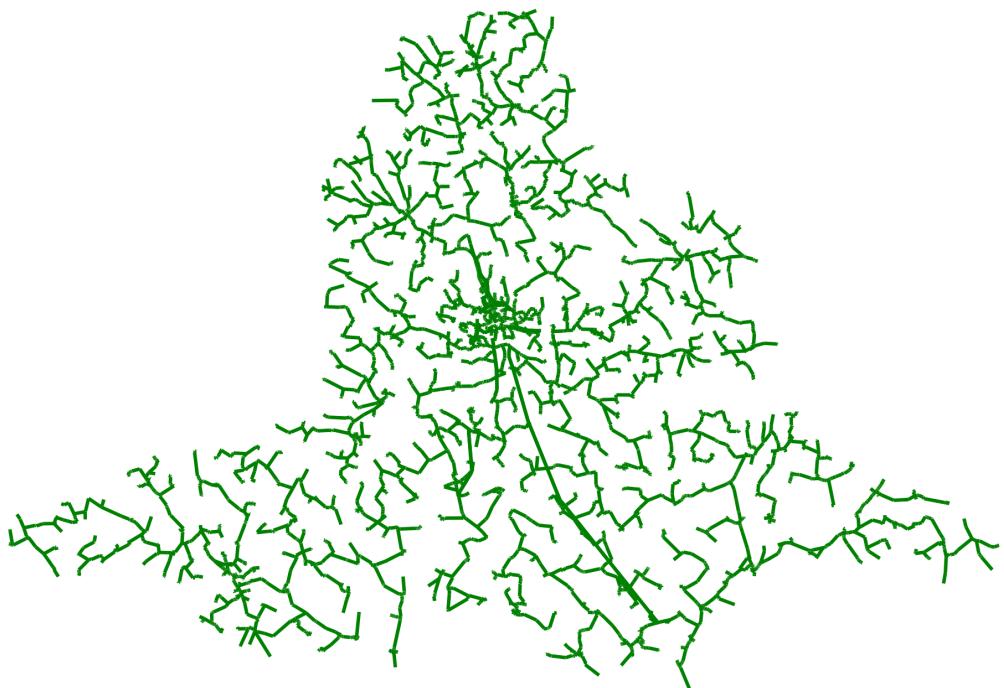
Obrázek 5: Vizualizace extrahované páteřní sítě (G_{fw})

4.3 Bonus 3 a 4: Minimální kostra grafu (MST)

Pro demonstraci algoritmů hledání minimální kostry byla vytvořena hustší síť než v předchozím kroku. K páteřní síti byly přidány i silnice II. a III. třídy (**secondary, tertiary**). Tím vznikl graf s větším množstvím cyklů, na kterém lze lépe demonstrovat schopnost algoritmů eliminovat nadbytečné hrany a ponechat pouze nezbytné spojení.

1. **Kruskalův algoritmus:** Využívá třídění hran podle váhy a vlastní strukturu UnionFind. Postupně spojuje komponenty, dokud nevznikne jeden strom.
2. **Jarníkův-Primův algoritmus:** Využívá vlastní MinHeap. Začíná v náhodném uzlu a postupně připojuje nejbližší sousedy.

Ověření: Oba algoritmy dospěly ke zcela identické celkové délce kostry, což potvrzuje správnost obou implementací. Vizuálně je kostra řidší než vstupní síť, protože neobsahuje žádné uzavřené okruhy.



Obrázek 6: Vizualizace minimální kostry grafu (Kruskalův algoritmus)

5 Závěr

Všechny části zadání včetně čtyř bonusových úloh byly splněny. Skript je plně funkční, využívá vlastní efektivní datové struktury (Heap, Union-Find) a automaticky generuje požadované výstupy a vizualizace.