# UNIVERZITA KARLOVA

**Přírodovědecká fakulta**

Studijní obor: Geoinformatika, kartografie a dálkový průzkum Země


**Matěj Hrabal**, **Kateřina Spudilová**

**Geoinformatika**

Nejkratší cesta grafem

Praha 2026

## Úloha 3 : Nejkratší cesta grafem

Implementujte Dijkstra algoritmus pro nalezení nejkratší cesty mezi dvěma zadanými uzly grafu.

Vstupní data budou představována silniční sítí doplněnou vybranými sídly (alespoň 100 uzlů). S využitím přiloženého skriptu konvertujte podkladová data do grafové reprezentace.

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

- nejkratší Eukleidovskou vzdálenost,
- nejmenší transportní čas (2 varianty, bez/se zohledněním klikatosti komunikací).

Pro druhou variantu optimální cesty navrhněte také vhodnou metriku, která zohledňuje rozdílnou dobu jízdy na různých typech komunikací dle jejich návrhové rychlosti $v$ a klikatosti $\kappa$, příkladem může být

$$t(u,v) = \kappa \frac{l(u,v)}{v(u,v)}.$$

Pro výpočet $\kappa$ využijte poměr délky polylinie $l$ představující diskrétní křivku a vzdálenosti $s$ koncových bodů polylinie

$$\kappa = \frac{l(u,v)}{s(u,v)}.$$

Každou z variant otestujte pro dvě dvě různé cesty tvořené alespoň 20 uzly. Výsledky umístěte do tabulky, vlastní cesty vizualizujte. Dosažené výsledky porovnejte s vybraným navigačním SW (alespoň 3).

| **Krok** | **Hodnocení** |
|---|---|
| Dijsktra algoritmus. | 20b |
| *Návrh jiného ohodnocení hran zohledňující křivolakost silniční sítě.* | *+5b* |
| *Zohlednění vlivu DMT.* | *+15b* |
| *Nalezení nejkratších cest mezi všemi dvojicemi uzlů.* | *+15b* |
| *Nalezení minimální kostry Borůvka/Kruskal.* | *+15b* |
| *Nalezení minimální kostry Jarník/Prime.* | *+15b* |
| *Využití heuristiky Weighted Union* | *+5b* |
| *Využití heuristiky Path Compression* | *+5b* |
| **Max celkem:** | **95b** |

Čas zpracování: 3-4 týdny.

## Data

### Příprava a zpracování dat

Data pro splnění zadání byla zpracována pomocí skriptu *data_download.py*. Ten využívá knihovny OSMnx pro stažení grafu silniční sítě typu *drive* pro oblast okresu Kutná Hora jako testovacího území. Byla definována mapa rychlostí *speed_map*, která přiřazuje maximální povolenou rychlost v km/h různým typům komunikací (např. *primary* – 90 km/h, *residential* – 50) na základě atributu *highway* z dat OpenStreetMap. Dále skript implementuje výpočet váhy hrany pro tři různé varianty výpočtu: *distance* (vzdálenost) – váha odpovídá reálné délce hrany v metrech (*real_length*), *time* (čas) – váha odpovídá celkovému času průjezdu a *sinuosity* (klikatost). Ta využívá Haversinův vzorec pro výpočet ortodromy (*ortho_dis*). Váha je penalizována koeficientem κ, vypočtenému jako poměr reálné délky ku vzdušné vzdálenosti. Penalizuje tak silniční úseky, které se výrazně odchylují od přímého směru. Výstupem je textový soubor *graph.txt* ve formátu *x1; y1; x2; y2; w* (*weight* - váha*)*, který slouží jako vstup pro grafové algoritmy.

### Reprezentace grafu

Za pomoci skriptu *lines_to_graph2.py* dochází k načtení dat a vytvoření grafové struktury v paměti. Funkce *loadEdges* parsuje textový soubor a ukládá souřadnice a váhy. Unikátní souřadnice bodů jsou převedeny na celočíselná ID (*pointsToIDs*) pomocí hashovací tabulky, což zjednodušuje indexaci v polích. Graf G je reprezentován jako seznam sousednosti (*adjacency list*) pomocí vnořených slovníků (*defaultdict(dict)*). To umožňuje efektivní přístup k sousedům uzlu v čase *O(1)*. Graf je konstruován jako neorientovaný, což odpovídá charakteru většiny silniční sítě nižších tříd, kde je možný obousměrný pohyb.

## Teoretická část

### Dijkstrův algoritmus

Pro vyhledání optimální cesty byl zvolen Dijkstrův algoritmus. Jedná se o grafový algoritmus, který pro graf s nezáporným ohodnocením hran nalezne nejkratší cestu z počátečního uzlu do cílového uzlu. Využívá hladový princip (greedy approach). Jádrem výpočtu je využití prioritní fronty, která v každém kroku vybírá uzel s aktuálně nejmenší vzdáleností od počátku, čímž je zaručeno nalezení optimální trasy. Iterativně dochází k odebírání uzlů z fronty.

### Minimální kostra grafu

Problém minimální kostry spočívá v nalezení podgrafu, který propojuje všechny uzly, neobsahuje cykly a součet vah jeho hran je minimální.

Algoritmus Borůvka/Kruskal pracuje s hranami seřazenými dle váhy. Postupně přidává hrany s nejnižší hodnotou, pokud již nejsou součástí spojených komponent. Pro efektivní správu komponent byla využita struktura Union-Find s heuristikami Path Compression a Weighted Union. Path compression zajišťuje, že při hledání kořene komponenty jsou všechny uzly na procházené cestě přepojeny tak, aby odkazovaly přímo na kořen. Tím dochází k radikálnímu zploštění stromové struktury, což zajišťuje, že následné operace vyhledávání proběhnou v téměř konstantním čase. Weighted union zabraňuje neefektivnímu prodlužování větví stromu. Algoritmus sleduje hodnost (rank) nebo velikost každého stromu a při sjednocování dvou komponent vždy připojuje kořen menšího stromu pod kořen stromu většího. Tím je zaručeno, že hloubka stromu roste pouze logaritmicky.

Algoritmus Jarník/Primův buduje kostru lokálně od počátečního uzlu a postupně ji rozšiřuje. Proces začíná v libovolném uzlu, který tvoří zárodek budoucí kostry. Využívá prioritní frontu k výběru hrany s nejnižší hodnotou, která vede z již objevené části grafu do neobjevené.

### Implementace

Pro vyhledání nejkratší či nejrychlejší cesty mezi dvěma sídly byl implementován Dijkstrův algoritmus ve skriptu *dijkstra.py*.

Pro zvýšení přívětivosti aplikace byla implementována funkce *start_end_point*, která integruje geocoding. Uživatel zadává počáteční a cílový bod názvem obce. Skript pomocí *ox.geocode* získá centroid obce a metodou nejbližšího souseda (*Nearest Neighbor*) identifikuje odpovídající uzel v topologii grafu.

```
def start_end_point(name, point_list_pse):
    try:
        lat, lon = ox.geocode(name)
        print(f"{name} -> {lat:.5f}, {lon:.5f}")

        closest_id = -1
        min_dist = float('inf')

        for i in range(len(point_list_pse)):
            x = point_list_pse[i][0]
            y = point_list_pse[i][1]

            if x > 1000: continue

            dist = (x - lon)**2 + (y - lat)**2

            if dist < min_dist:
                min_dist = dist
                closest_id = i

        return closest_id
```

Po ukončení výpočtu je výsledná trasa rekonstruována zpětným průchodem přes pole předchůdců. Výstupem je počet uzlů na trase a její celková váha. Trasa je vizualizována nad podkladovou silniční sítí pomocí knihovny *matplotlib.*

## Nalezení minimální kostry

Pro řešení problému minimální kostry (Minimum Spanning Tree), tedy nalezení podgrafu, který propojuje všechny uzly s minimálním součtem vah hran, byly implementovány dva přístupy.

### Borůvka/Kruskal

Prvním z nich je algoritmus Borůvka/Kruskal implementovaný ve skriptu *kruskal.py*, který pracuje se seznamem všech hran seřazeným podle váhy. Pro efektivní správu komponent grafu a detekci cyklů byla vytvořena třída *UnionFind*. V rámci této struktury byly implementovány pokročilé heuristiky. Komprese cesty (*Path Compression*) ve funkci *find* přepojuje uzly na cestě přímo ke kořeni, což snižuje hloubku stromu.

```
def find(self, i):
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]
```

Vyvážené sjednocování (*Weighted Union*) ve funkci *union*, při němž je vždy menší strom připojen pod kořen většího stromu (dle hodnosti rank). Tyto optimalizace výrazně snižují časovou složitost operací a zabraňují degeneraci stromové struktury.

```
def union(self, i, j):
        root_i = self.find(i)
        root_j = self.find(j)

        #tree union by ranks
        if root_i != root_j:
            if self.rank[root_i] < self.rank[root_j]:
                self.parent[root_i] = root_j
            elif self.rank[root_i] > self.rank[root_j]:
                self.parent[root_j] = root_i
            else:
                self.parent[root_j] = root_i
                self.rank[root_i] += 1
            return True
        return False
```

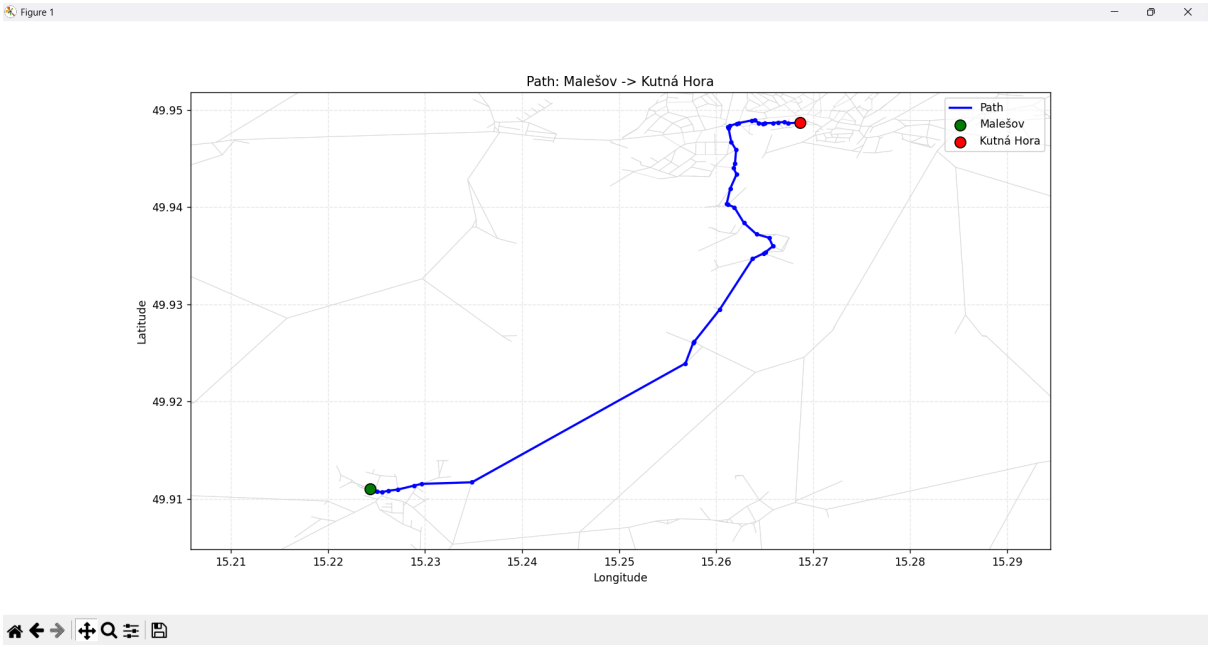
### Jarník/Prim

Jako alternativní metoda pro ověření výsledků byl použit Jarníkův (Primův) algoritmus ve skriptu jarnik.py. Tento přístup konstruuje minimální kostru lokálně, postupným rozrůstáním od zvoleného počátečního uzlu. Algoritmus využívá binární haldu z knihovny *heapq* pro

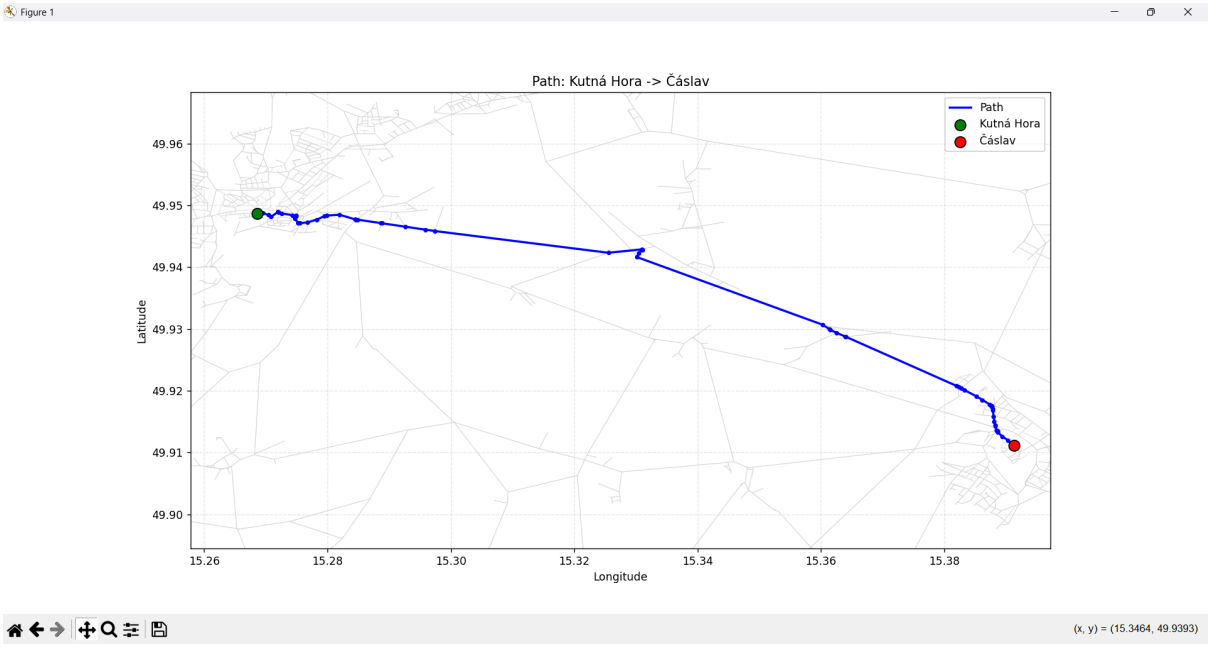
efektivní výběr nejlevnější hrany, která spojuje již objevenou část grafu s dosud nenavštívenými uzly. Oba implementované algoritmy pro výpočet MST byly vizualizovány pomocí knihovny matplotlib, což umožňuje grafické ověření, že výsledná kostra pokrývá všechny uzly grafu s minimálními celkovými náklady.

## Výsledky

V rámci praktické části byly testovány implementované grafové algoritmy nad silniční sítí okresu Kutná Hora. Testování bylo provedeno na dvou vybraných trasách mezi obcemi Kutná Hora → Čáslav a Malešov → Kutná Hora, která představuje reprezentativní vzorek zahrnující komunikace různých tříd. Vizualizované nejrychlejší trasy je možné vidět na obrázku 1 a 2. Porovnání výsledků vlastní implementace a komerčních aplikací Mapy.cz , Google mapy a Waze je možné vidět v tabulkách 1 a 2.




Obrázek 1: Nejrychlejší trasa z obce Malešov do Kutné Hory.




Obrázek 2: Nejrychlejší trasa z Kutné Hory do Čáslavi.

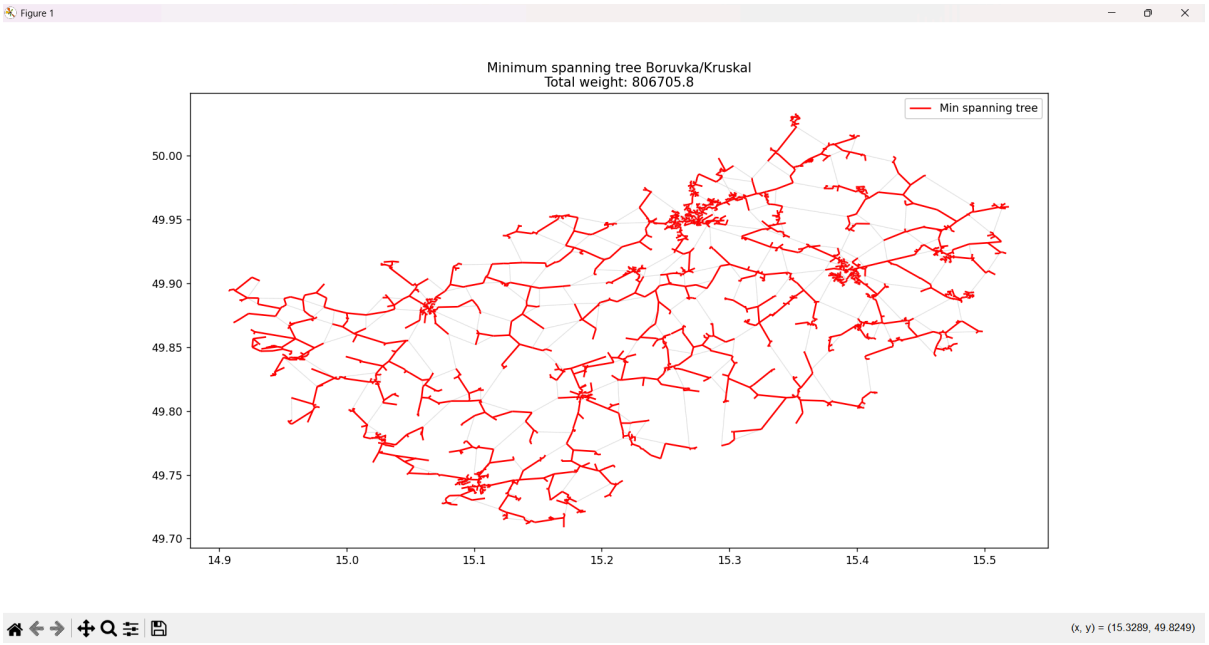
Byla nalezena minimální kostra pro celý okres Kutná Hora, a to dvěma způsoby. Jak pro algoritmus Borůvka/Kruskal, tak i pro algoritmu Jarník/Prim byla spočítána výsledná celková váha jako 806,71 km. Minimální kostry grafu je možné vidět na obrázcích 3 a 4.

| | | Celková váha/Počet uzlů | | | |
|---|---|---|---|---|---|
| | | Kutná Hora --> Čáslav | | Malešov --> Kutná Hora | |
| varianta | distance (m) | 10778.737 | 61 | 6630.842 | 43 |
| | time (s) | 463.030 | 62 | 287.396 | 44 |
| | sinuosity (s) | 471.296 | 62 | 295.550 | 44 |

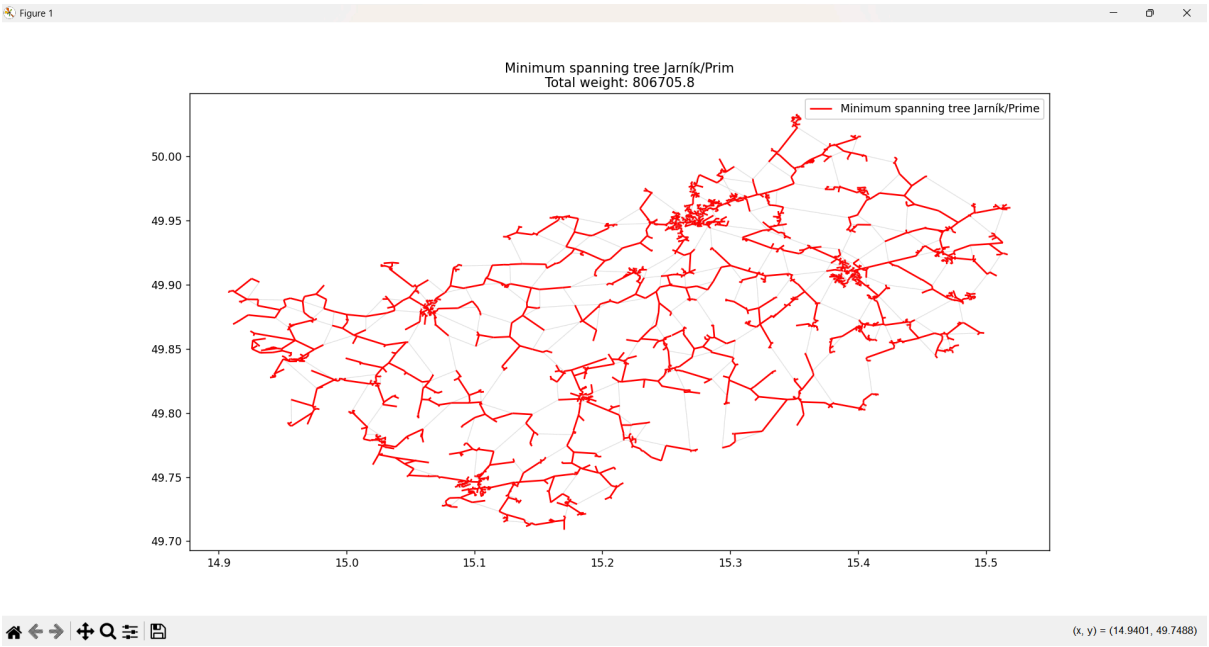
Tabulka 1: Porovnání výsledných výpočtů tras u vlastního algoritmu.

| | Kutná Hora --> Čáslav | | Malešov --> Kutná Hora | |
|---|---|---|---|---|
| | nejkratší trasa | nejrychlejší trasa | nejkratší trasa | nejrychlejší trasa |
| **Mapy.cz** | 11,2 km | 14 min | 6,6 km | 9 min |
| **Google mapy** | 11,1 km | 14 min | 6,6 km | 11 min |
| **Waze** | 10,9 km | 12 min | 6,6 km | 9 min |

Tabulka 2: Porovnání výsledných tras v komerčních aplikacích.




Obrázek 3: Minimální kostra grafu spočítána podle algoritmu Borůvka/Kruskal.




Obrázek 4: Minimální kostra grafu spočítána podle algoritmu Jarník/Prim.

## Diskuze

Při srovnání vypočtené nejrychlejší trasy s komerčními aplikacemi bylo zjištěno, že vlastní implementace uvádí zpravidla optimističtější (kratší) časy dojezdu. Tento rozdíl je způsoben zjednodušením modelu. Vlastní implementace počítá s konstantní maximální povolenou rychlostí v celém úseku hrany. Model nezohledňuje zpomalování na křižovatkách, do zatáček ani aktuální dopravní situaci. Zároveň uvažuje všechny cesty jako obousměrné a nepočítá s uzavírkami komunikací.

V práci nebyl zohledněn vliv DMT, ani nalezeny nejkratší cesty mezi všemi dvojicemi uzlů.

Pro zpracování této práce byly využity nástroje umělé inteligence. Konkrétně se jednalo o pomoc při ladění (debugging) složitějších závislostí knihovny osmnx a o návrhy optimalizace výkonu třídy *UnionFind*. AI byla rovněž využita pro návrh struktury závěrečné zprávy. Výsledný kód i text jsou autorským dílem studentů, přičemž návrhy AI sloužily pouze jako podklad pro vlastní implementaci.

**Zdroje**

BOEING, G. (2025): OSMnx documentation. Dostupné z: https://osmnx.readthedocs.io/en/stable/index.html (cit. 4. 1. 2026).

GEEKSFORGEEKS (2025): Kruskal’s Minimum Spanning Tree Algorithm | Greedy Algo-2. Dostupné z: https://www.geeksforgeeks.org/dsa/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/ (cit. 4. 1. 2026).

GEEKSFORGEEKS (2025): Prim’s Minimum Spanning Tree (MST) | Greedy Algo-5. Dostupné z: https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/ (cit. 4. 1. 2026).

GEEKSFORGEEKS (2025): Union by Rank and Path Compression in Union-Find Algorithm. Dostupné z: https://www.geeksforgeeks.org/dsa/union-by-rank-and-path-compression-in-union-find-algorithm/ (cit. 4. 1. 2026).

W3SCHOOLS (2026): DSA Graphs Dijkstra's Algorithm. Dostupné z: https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php (cit. 4. 1. 2026).

W3SCHOOLS (2026): DSA Minimum Spanning Tree Kruskal's Algorithm. Dostupné z: https://www.w3schools.com/dsa/dsa_algo_mst_kruskal.php (cit. 4. 1. 2026).