ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE, KARTOGRAFIE A GEOINFORMATIKA
KATEDRA SPECIÁLNÍ GEODÉZIE

název předmětu
GEOINFORMATIKA
název úlohy
Úloha 4: Nejkratší cesta grafem

akademický rok	semestr	vypracoval(a)	Datum	klasifikace
2023/24	Zimní	Jan Koudelka, Vojtěch Müller	31.1.2024	

Technická zpráva

1 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 Euklidovskou vzdálenost,
- Nejmenší transportní čas.

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

Jako doplňkové úlohy implementujte:

- Nalezení nejkratší cesty mezi všemi dvojicemi uzlů.
- Nalezení minimální kostry.
- Využití heuristiky Weighted Unin.
- Využití heuristiky Path Compression.

2 Popis

Hledání nejkratší cesty grafem vychází z teorie grafů. Jedná se zde o určení nejkratší cesty v jakékoli reálné situaci, kterou lze reprezentovat grafem. Nejběžněji se graf používá pro silniční síť, kde uzly reprezentují významné body a hrany reprezentují samotné silniční úseky. Řešení nejkratší cesty se pak používá při navigaci v kterémkoli softwaru. [1]

3 Postup

3.1 Vytvoření grafové reprezentace

K vytvoření grafu byl použit software ArcGIS Pro, kde byla vytvořena bodová vrstva s významnými sídli, křižovatkami a hranicemi obcí. Následně byla vytvořena liniová vrstva, která byla naplněna jednotlivými silnicemi, které spojují body. Do vrstvy hran byly následně přidány atributy pro ohodnocení jednotlivých hran. Pomocí jazyka Python byly následně naplněny grafy s jednotlivými ohodnoceními. Skript, pomocí kterého byly grafy naplněny se nachází v přílohách. Graf byl reprezentován seznamem souřadnic vrcholů a seznamem vrcholů s informacemi o sousedech a ohodnocení k jednotlivým sousedům.

3.1.1. Euklidovská vzdálenost

Byl přidán atribut délky pro jednotlivé úseky siliční sítě a tento atribut poté plní roly ohodnocení grafu.

3.1.2 Transportní čas

Byly přidány atributy pro výpočet metriky:

- Délka jednotlivých úseků. - *d*
- Přímá délka mezi uzly. - *D*
- Maximální povolená rychlost na úseku. - *v*

Z maximální povolené rychlosti a délky úseku byl nejprve vypočítán čas, který je potřeba na projetí jednotlivých úseků, a to ze vzorce

$$t = \frac{d}{v}$$

Následně byl vypočítán koeficient křivolakosti ze vzorce

$$k = \frac{D}{d}$$

A poté byl vypočítán výsledný čas průjezdu úseku, který byl upraven o koeficient křivolakosti.

$$T = k.t$$

Hodnota T byla poté použita jako ohodnocení grafu.

3.2 Dijkstra algoritmus [2]

Dijkstra algoritmus hledá nejkratší cestu mezi uzly grafu. Využívá princip, při kterém se postupně zpřesňuje odhad nejkratší délky. Od prvního uzlu procházíme graf prodlužováním postupně po jednom uzlu a počítáme ohodnocení, které je potřeba, abychom se k tomuto uzlu dostali. Najdeme tedy vždy předchůdce určitého uzlu, který zajišťuje nejkratší přístup.

K výpočtům se používá princip relaxace, kdy v uzlech procházených prioritní frontou testujeme, zda platí

$$d[v] > d[u] + w[u][v]$$

Hodnoty d představují aktuální odhad ohodnocení pro cestu do těchto uzlů a hodnota w je ohodnocení hrany mezi těmito uzly. Pokud platí psaný předpoklad, tak se aktualizuje odhad na

$$d[v] = d[u] + w[u][v]$$

a pro uzel v vzniká nový předchůdce

$$p[v] = u$$

Samotné hledání nejkratší cesty probíhá tak, že se nejprve nastaví odhad ohodnocení všech uzlů na nekonečno, poté se u prvního uzlu nastaví ohodnocení nula a prochází se nejprve sousední uzly. K nim

se zapíše hodnota odhadu ohodnocení, pokud je splněna výše psaná podmínka. Takto se postupně prochází celý graf, z kterého bude výstupem list předchůdců.

```
def dijkstra(G, u, v): # Function for Dijkstra algoritm

    # Supplementary data structure
    n = len(G)

    # List of predecessors
    P = [-1] * (n + 1)

    # List of distances
    D = [math.inf] * (n + 1)

    # Priority queue
    PQ = PriorityQueue()

    # First note to queue
    D[u] = 0
    PQ.put((D[u], u))

    # Repeat until Q is empty
    while not (PQ.empty()):

        # Get first note
        du, u = PQ.get()

        for v, wuv in G[u].items():

            if D[v] > D[u] + wuv :

                # Update distance
                D[v] = D[u] + wuv

                # Store Predecessor
                P[v] = u

                # Add to queue
                PQ.put((D[v], v))

    return P
```

Do funkce vstupuje graf a body z kterého a do kterého se chceme dostat. Nejprve vytvoříme seznam předchůdců P a seznam vzdáleností D. Oba seznamy budou mít velikost n + 1, kde n je velikost grafu a seznam předchůdců naplníme hodnotami -1 a seznam vzdáleností hodnotami nekonečno. Následně vytvoříme prioritní frontu.

Dále vezmeme první uzel, jehož délku nastavíme na 0 a vložíme ho do prioritní fronty. Nyní procházíme prioritní frontu, dokud se v ní nachází nějaké prvky. U každého uzlu, který procházíme kontrolujeme všechny uzly, ke kterým se lze z daného uzlu dostat a porovnáváme, zda je cesta do tohoto uzlu menší, než cesta již uložená v matici D. Pokud je tato nová cesta kratší, tak k sousedním uzlům ukládáme novou vzdálenost, nového předchůdce a do prioritní fronty vkládáme tyto uzly.

Výstupem z funkce je poté seznam předchůdců s nejkratší cestou.

3.3 Zpětná rekonstrukce cesty [1]

Následně byla vytvořena funkce, která ze seznamu předchůdců určuje seznam uzlů, přes které prochází nejkratší cesta. Z těchto uzlů byly poté pomocí knihovny matplotlib vytvářeny výsledné grafy.

```
def Reconstruction(G,P,u,v): # Function for path

    path = []
    while v != u and v != -1:
        path.append(v)
        v = P[v]
    path.append(v)
    if v == -1:
        path = 'Path does not exist'

    # Sum of path
    sum = 0
    for i in range(len(path) - 1):
        sum = sum + G[path[i]][path[i + 1]]

    print('Path: ' + str(sum))

    return path
```

Do funkce vstupuje graf, seznam předchůdců a body z kterého a do kterého se chceme dostat. Nejprve vytvoříme prázdný seznam, do kterého budeme ukládat cestu. Následně procházíme uzli od konce, a to nejprve poslední uzel cesty a následně vždy předchůdce tohoto uzlu, podle seznamu předchůdců. Tyto uzly postupně vkládáme do seznamu cesty.

Ve funkci je dále přidaná podmínka, pokud hledaná cesta neexistuje, a nakonec počítáme ohodnocení cesty.

3.4 Nalezení nejkratší cesty mezi všemi dvojicemi uzlů

Pomocí dvou cyklů for byla zjištěna nejkratší cesta mezi všemi uzly.

3.5 Nalezení minimální kostry [2]

K určení minimální kostry byl použit Borůvkův/Kruskalův algoritmus. Probíhá na principu, kdy se prochází jednotlivé hrany pomocí prioritní fronty a poté se hrany testuje se, zda hranu přidáme do stromu. Při testování hrany mohou nastat čtyři situace:

- Žádný uzel h neleží v jiném podstromu. Hrana vytvoří nový podstrom.
- Jeden uzel h je součástí některého z podstromů. Hranu připojíme k podstromu.
- Oba uzly h jsou v různých podstromech. Oba podstromy spojíme do jednoho.
- Oba uzly h jsou v jednom podstromu. Hranu zahodíme.

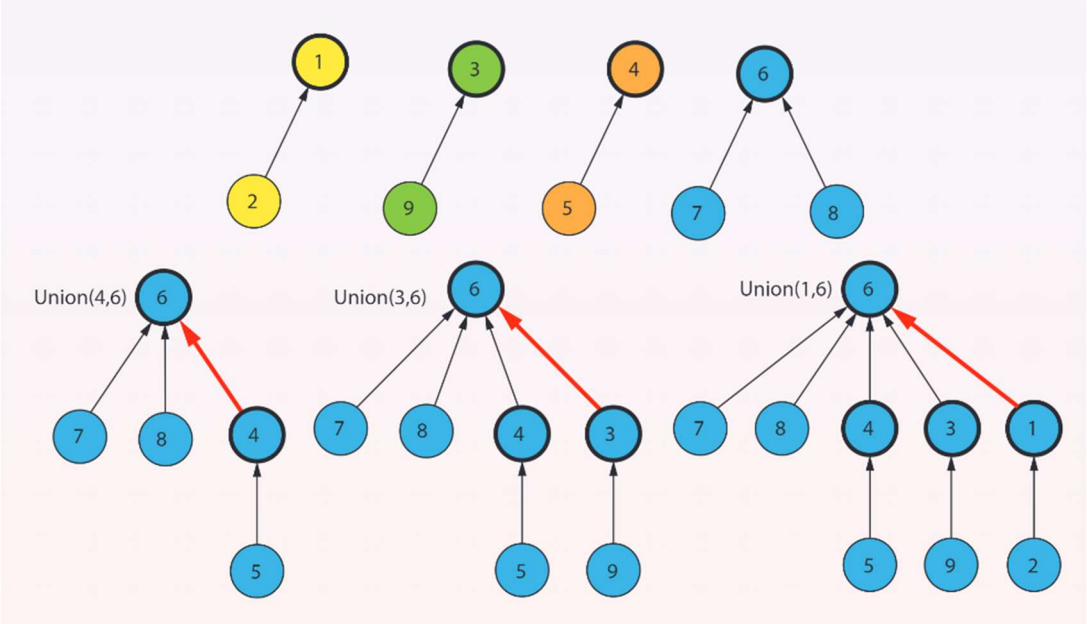
```
def make_set(u, p, r):
    p[u] = u
    r[u] = 0

def mstk(V, E):
    T=[]                                #Empty tree
    wt = 0                              #Sum of weights of T
    p = [math.inf] * (len(V) + 1)       #List of roots
    r = [math.inf] * (len(E) + 1)       #Rank of the node
    for v in V:                         #Make set
        make_set(v, p, r)               #Initilize p and r
    ES = sorted(E, key=lambda it:it[2]) #Sort edges by w
    for e in ES:                        #Process all edges
        u, v, w = e                     #Take an edge
        if (find(u, p) != find(v, p)):  #roots u, v in different trees?
            union(u, v, p, r)           #Create union
            T.append([u, v, w])         #Add edge to tree
            wt = wt + w                 #Compute weight of T
    return wt, T
```

Do funkce vstupuje graf, který je pro tuto funkci vypsán jako seznam uzlů a seznam hran. Nejprve vytvoříme prázdný seznam, který bude reprezentovat strom. A nastavíme hodnotu celkového ohodnocení na 0. Dále vytvoříme seznam kořenů a seznam uzlů. Tyto seznamy naplníme hodnotami nekonečno. Velikost seznamu kořenů bude o jedna větší než velikost seznamu vstupních uzlů a velikost seznamu nových uzlů bude o jedna větší než velikost seznamu hran. Dále procházíme jednotlivé uzly a pomocí funkce make_set naplňujeme hodnoty do dvou seznamů.

Následně seřadíme seznam hran podle jejich ohodnocení. Tyto hrany poté postupně procházíme. Pomocí heuristiky Path Compresion, konkrétně funkcí find, porovnáváme, zda přidáváme hranu do minimální kostry. Funkce find bude popsána níže. Pokud hranu přidáváme, tak pomocí heuristiky Weighted Union, konkrétně funkcí union, hledáme, do které části stromu hranu připojíme. Funkce union bude také popsána níže. Následně hranu ke stromu připojíme a přepočítáme celkové ohodnocení kostry.

3.6 Využití heuristiky Weighted Union [2]

Tato funkce je využívána k efektivnímu spojování stromů, kdy je vždy lepší připojit kratší strom k delšímu. Princip je takový, že nalezneme kořen obou stromů, určíme, který strom je delší a ke kořenu delšího stromu připojíme kořen kratšího stromu.



Obrázek 1: Weigted Union [2]

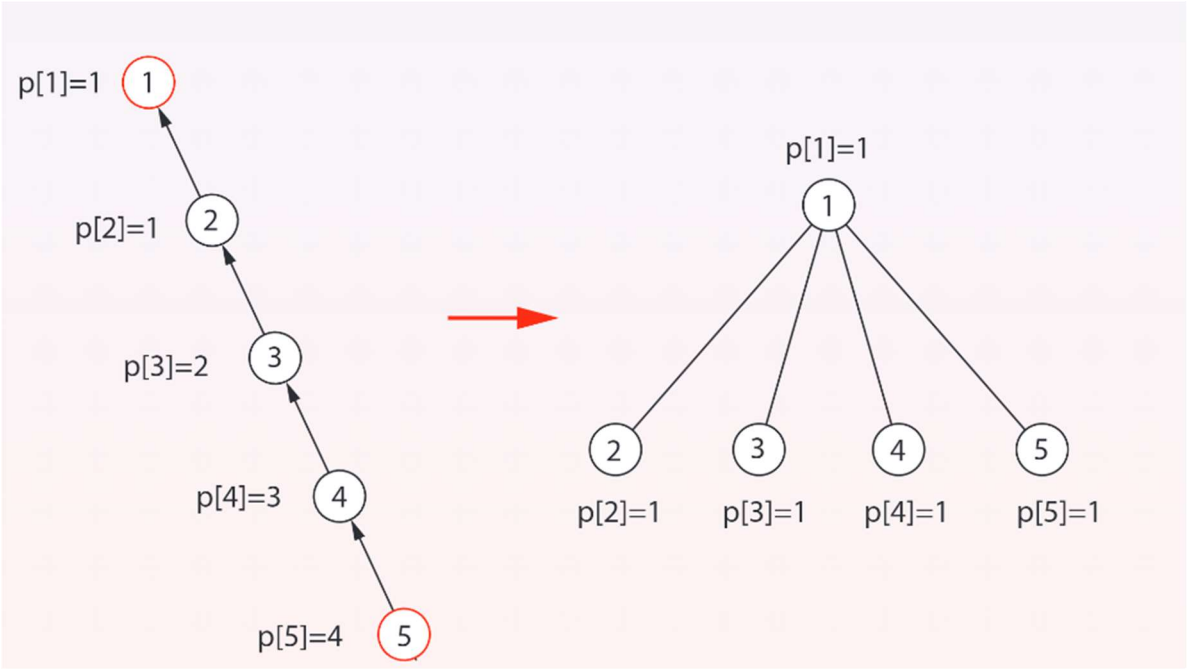
```
def union(u, v, p, r):
    root_u = find(u, p)              #Find root for u + compress
    root_v = find(v, p)              #Find root for v + compress
    if root_u != root_v:             #u, v in different subtrees
        if r[root_u] > r[root_v]:    #u subtree is longer
            p[root_v] = root_u       #Connect v to u
        elif r[root_v] > r[root_u]: #v subtree is longer
            p[root_u] = root_v       #Connect u to v
        else:                        #u, v have equal lengths
            p[root_u] = root_v       #Connect u to v
            r[root_v] = r[root_v]+1 #Increment rank
```

Do funkce vstupují okrajové uzle hrany a kompletní seznam kořenů a uzlů. Pomocí funkce find nalezneme kořeny jednotlivých uzlů a také seskupíme strom. Následně přiřazujeme hranu tak, aby byla velikost stromu co nejnižší.

3.7 Využití heuristiky Path Compression [2]

Tato funkce je využívána ke zkrácení stromu, ab operace s ním probíhaly rychleji. Strom lze zkrátit dvěma metodami. Jedním průchodem lze strom zkrátit o půlku. Dvouprůchodovým postupem lze celý strom připojit na jeden kořen. V této úloze byl použit dvouprůchodový systém.

Princip funkce probíhá v nalezení kořene a následném připojení všech uzlů na tento kořen.



Obrázek 2: Path Compression [2]

```
def find(u, p):
    while (p[u] != u):              #Find root
        u = p[u]
    root = u
    while u != root:
        up = p[u]                   #Store predecessor
        p[u] = root                 #Change predecessor to root
        u = up                      #Go to parent
    return u
```

Do funkce vstupuje uzel a seznam kořenů. Nejprve nalezneme hlavní kořen všech předchůdců a následně přiřadíme všechny uzly, skrze které procházíme k hlavnímu kořeni, přímo k tomuto kořeni.

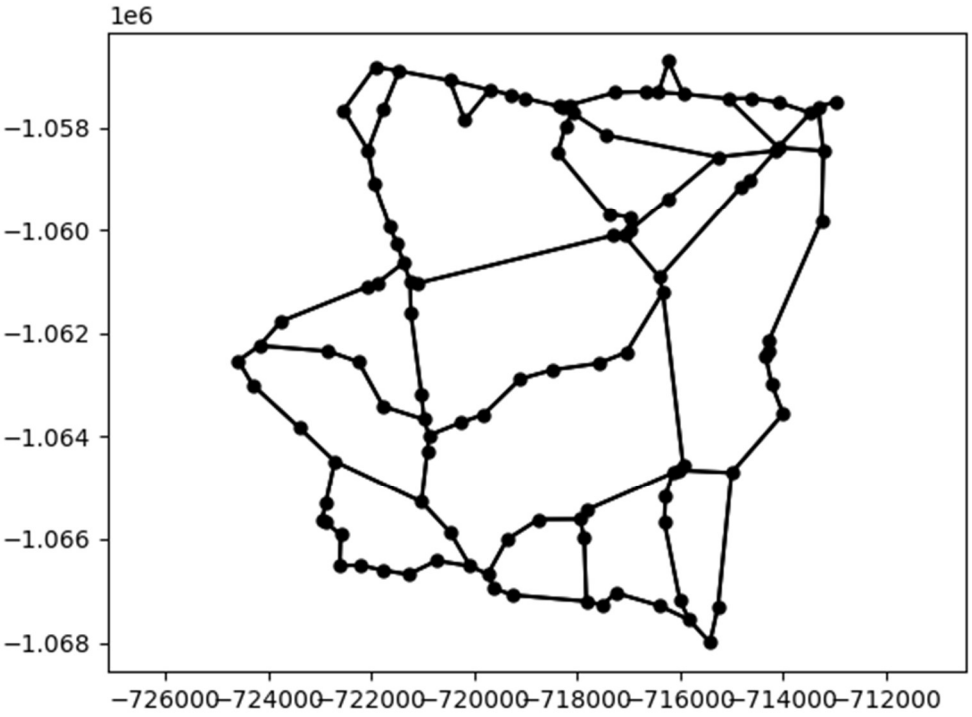
4 Výsledky

Prvním výstupem je grafová reprezentace silniční sítě. Ta se nachází v programu, který je přiložen v přílohách. Proměnná Graf3 obsahuje ohodnocení pomocí Euklidovské vzdálenosti a proměnná Graf2 obsahuje ohodnocení pomocí transportního času.

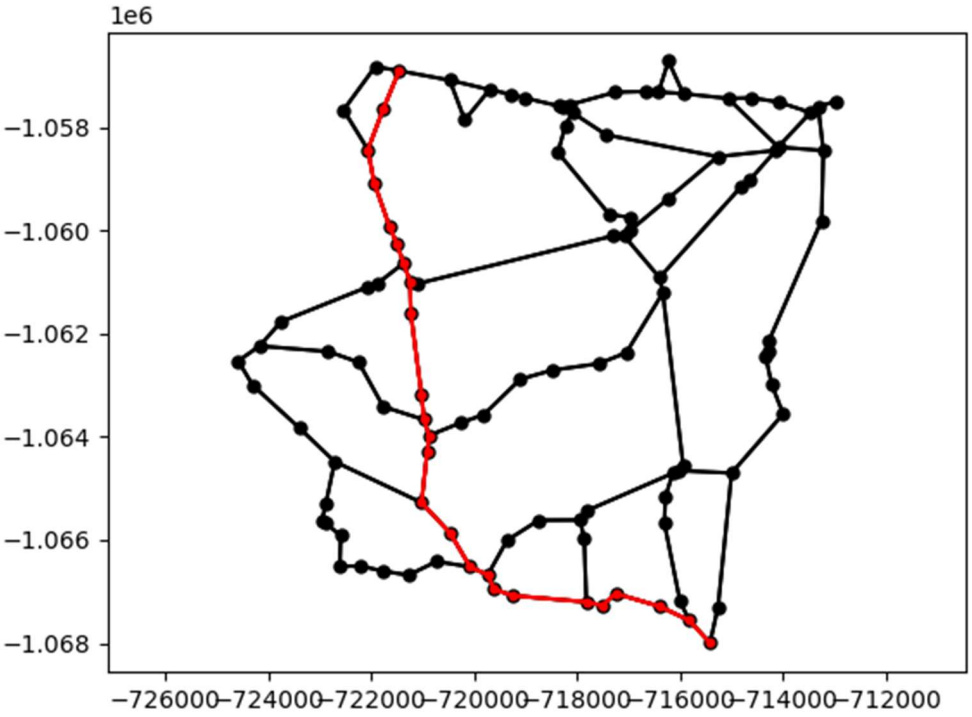
Následně byla určována cesta mezi dvěma uzly. Tyto uzly se nachází v obci Stříbrná Skalice a v obci Mukařov. Pro nejkratší Euklidovskou vzdálenost vychází ohodnocení na 16,46 km a pro nejkratší transportní čas vychází 13,66 minut.

Tabulka 1: Porovnání výsledků úlohy s daty z Mapy.cz

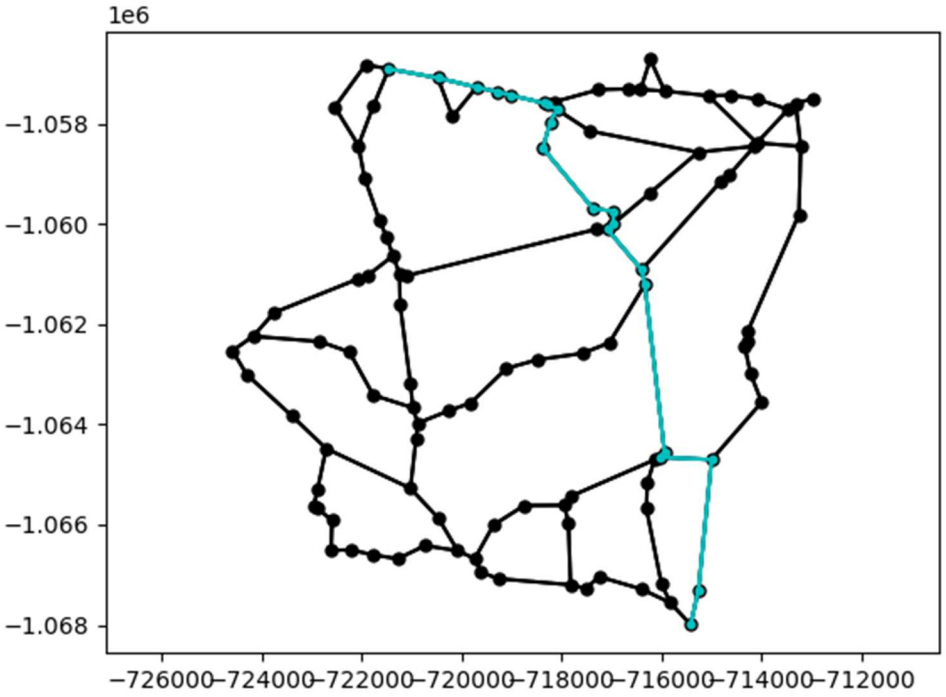
	Řešení v této úloze	Mapy.cz	Odchylka
Nejkratší Euklidovská vzdálenost [km]	16,46	16,4	- 0,06
Nejmenší transportní čas [min]	13,66	19	5,34



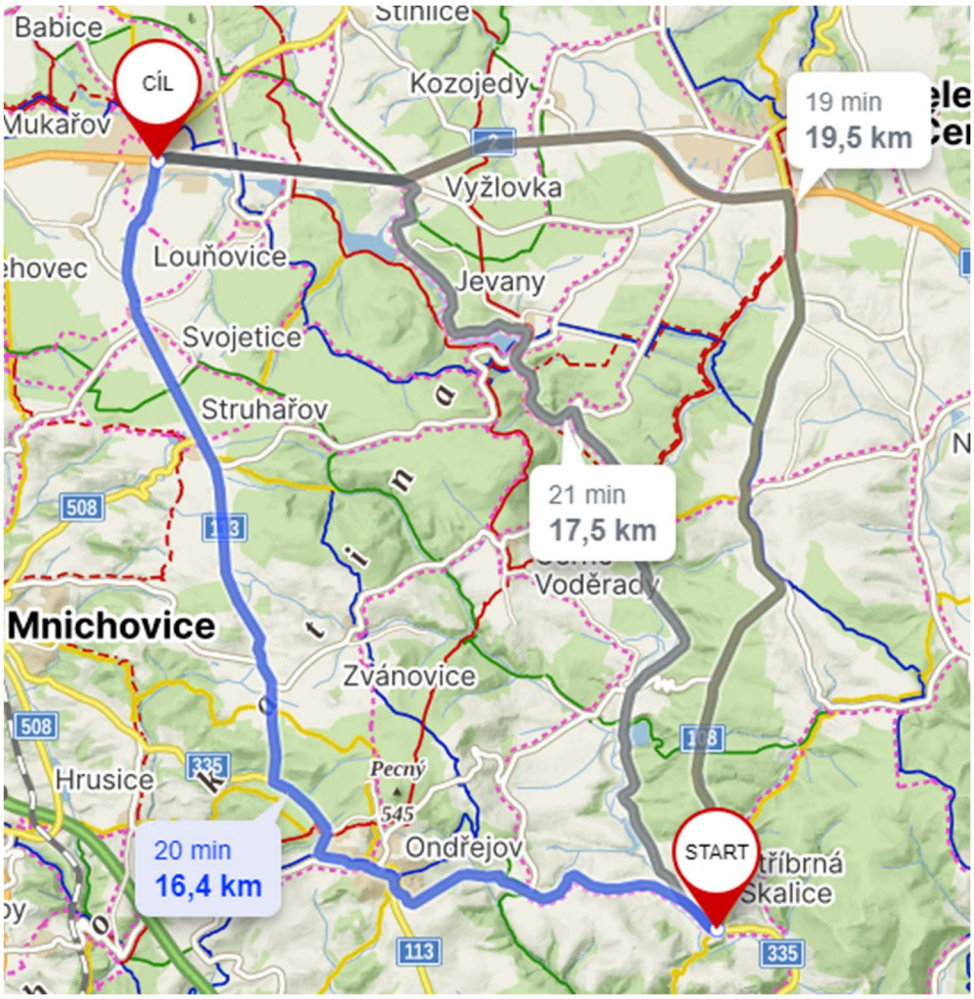
Obrázek 3: Vizualizace grafu.



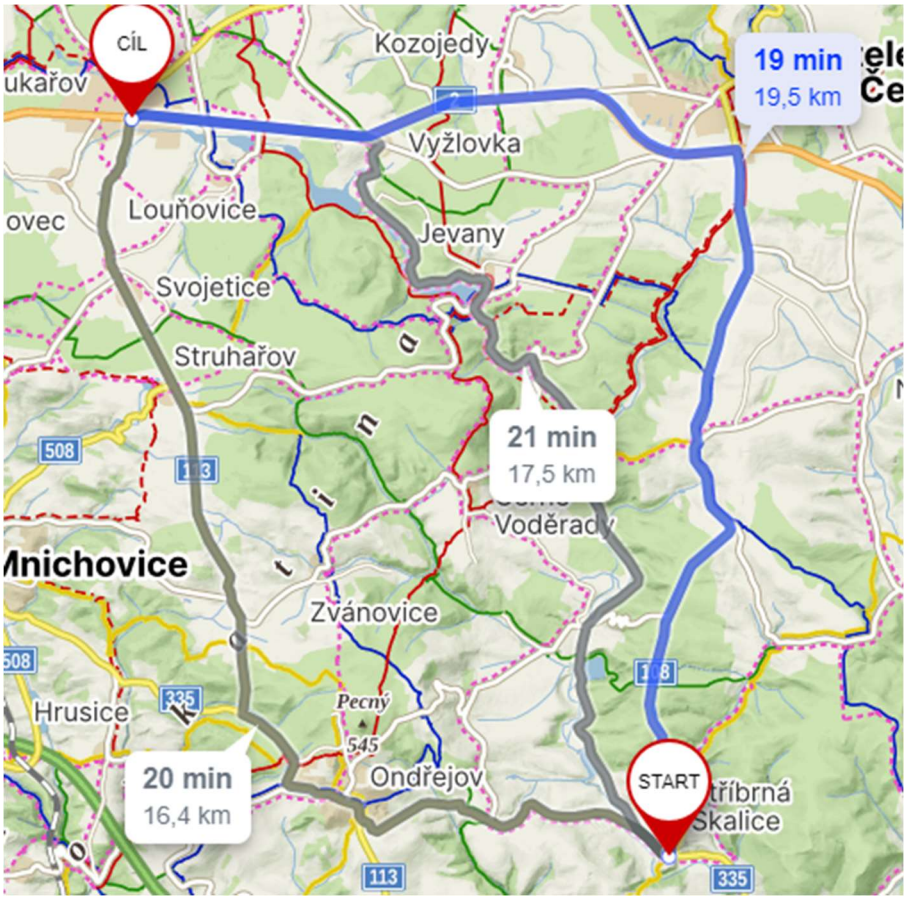
Obrázek 4: Cesta s nejkratší Euklidovskou vzdáleností.



Obrázek 5: Cesta s nejmenším transportním časem.

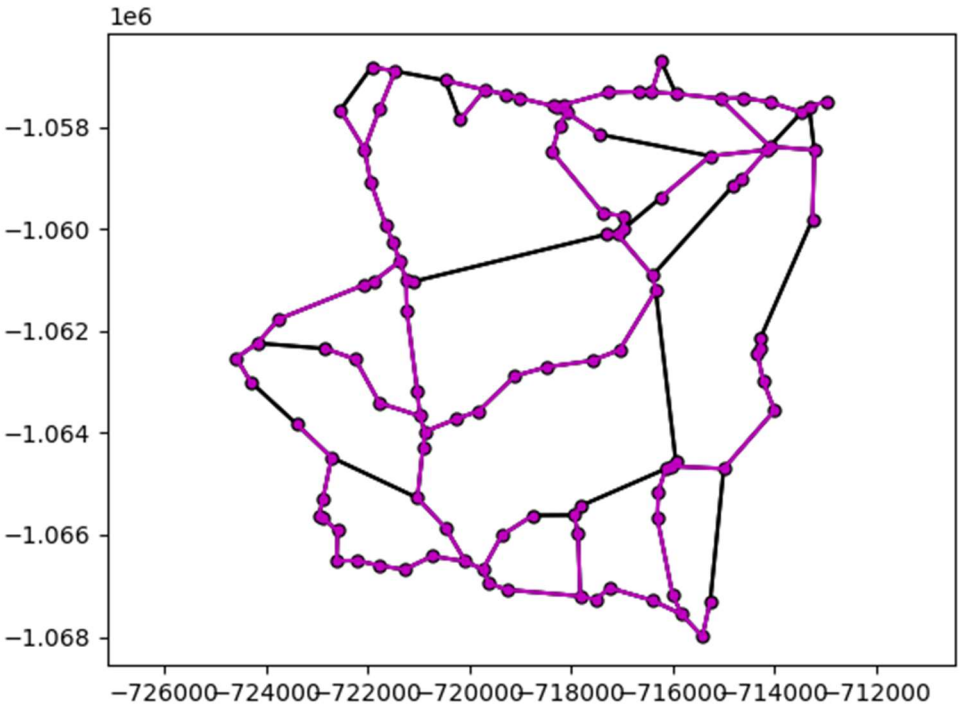


Obrázek 6: Cesta s nejkratší vzdáleností podle mapy.cz. [3]



Obrázek 7: Cesta s nejmenším transportním časem podle mapy.cz. [3]

Textový soubor s nejkratšími cestami mezi všemi dvojicemi uzlů se nachází v textovém souboru v přílohách. Nejkratší cesty byly vytvořeny na základě nejmenšího transportního času.



Obrázek 8: Minimální kostra grafu.

Minimální kostra byla nalezena na grafu s ohodnocením Euklidovskými vzdálenostmi. Ohodnocení vychází na 64,12 km. Počet hran minimální kostry je 109.

5 Závěr

Byl vytvořen program, který určuje nejkratší cestu grafem. První určení cesty proběhlo na grafu, kde bylo ohodnocení reprezentováno Euklidovskou vzdáleností. Výsledná cesta se shoduje se softwarem mapy.cz. ohodnocení cesty bylo určeno na 16,46 a na Mapy.cz na 16,4. Výstup je tedy přibližně shodný. Druhá cesta byla vytvořena z ohodnocení nejmenším transportním časem. Zde se cesta neshoduje s cestou v softwaru mapy.cz a navíc je čas výrazně kratší, konkrétně o 5,34 minut. Toto je způsobeno více faktory, ale všechny tyto faktory jsou nedokonalým ohodnocením a vytvořením grafu. Jedná se převážně o nedokonalé zpracování možností rychlostí. V úseku, kde se cesty rozdělí, se v cestě určené touto úlohou nachází serpentýny. V těchto místech je maximální povolená rychlost 90 km/h a díky koeficientu křivolakosti je tato rychlost výrazně zpomalena, ale stále je výrazně rychlejší, než se dá jet v realitě. To samé platí o úseku mezi vesnicemi Hradec, Jevany a Vyžlovka. Zde je navíc místy i nucené zpomalení, které nebylo do gradu započítáno. Další faktor, který zkracuje čas cesty jsou křižovatky, které v realitě nutí ke zpomalení, ale v grafu s ním nebylo počítáno. Časová reprezentace tedy neodpovídá skutečnosti.

Určení minimální kostry proběhlo na principu Borůvkova algoritmu. Jeho zobrazení lze vidět v obrázku 6 a lze pozorovat, že kostra skutečně obsahuje všechny uzly, ale ne všechny hrany. Ohodnocení minimální kostry vyšlo na 64,12 km. Celkové ohodnocení grafu bylo 89,55 km, tedy minimální kostra obsahuje 71,6 % celkové délky cest, které jsou obsaženy v grafu. Pro minimální kostru pak bylo použito 109 hran ze 127, tedy 85,8 % hran.

6 Přílohy

- Skript pro naplnění grafu (*graf.py*)
- Skript pro spuštění funkce a s popisem funkce (*Koudelka_Muller_U4.py*)
- Textový soubor s nejkratšími cestami mezi všema uzly (*Shortest_path.txt*)

7 Zdroje

[1] https://github.com/k155cvut/ygei/blob/main/prednasky/geoinf8.pdf

[2] https://github.com/k155cvut/ygei/blob/main/prednasky/geoinf9.pdf

[3] https://mapy.cz

V Praze dne 31.1.2024 **Jan Koudelka, Vojtěch Müller**