Úloha 3: Nejkratší cesta grafem

Předmět: Geoinformatika
Vypracoval: David Martínek
Datum: 4.1.2025 (15.2.2025)

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.

1) Popis použitých algoritmů a teorie

V programovacím jazyce *Python* byly datové vstupy pro potřeby dalšího zpracování nahrány do seznamů symbolizující uzly a hrany prostřednictvím nadefinovaných funkcí *loadNodes* a *loadEdges*. Zatímco uzly byly nahrány v podobě seznamu souřadnic (x, y), vstupní hrany byly zpracovávány se čtyřmi údaji o souřadnicích (počáteční a koncový bod) a atributy délka a třída silnice. Atributy byly zpracovány pro každou hranu a byla jim přiřazena váha (viz kapitola 2). Následně byl z hran a uzlů vytvořen slovník – graf.

a) Dijkstra v roce 1956 navrhl algoritmus pro nalezení nejkratší cesty v grafu s nezápornými vahami hran. Algoritmus má široké využití zejména v oblasti síťových analýz, kde umožňuje efektivní plánování tras v navigačních systémech, optimalizaci komunikačních sítí nebo například výpočet nejkratšího spojení v dopravních aplikacích. Při vyhledávání nejkratší cesty využívá tzv. „greedy“ strategii, která postupně volí lokálně nejlepší možnosti s cílem dosáhnout optimálního celkového výsledku. V každé iteraci vybírá uzel s nejmenší známou vzdáleností, aktualizuje vzdálenosti jeho sousedů (tzv. relaxace hran) a pokračuje, dokud nejsou všechny dosažitelné uzly zpracovány. Pro efektivní vyhledávání nejkratších vzdáleností využívá prioritní frontu. Pro následnou rekonstrukci cesty je využíván seznam předchůdců (*list of predecessors*).

V případě grafu se zápornými hranami je nutné Dijkstra algoritmus nahradit Bellman-Fordovým algoritmem, který dokáže pracovat i s negativními vahami, avšak za cenu vyšší časové složitosti. Pokud je cílem zrychlit nalezení cesty, lze použít A* algoritmus, který k Dijkstrovu přístupu přidává heuristickou funkci odhadující vzdálenost k cíli. Díky tomu je často rychlejší, ale jeho efektivita závisí na kvalitě zvolené heuristiky. Při nevhodném návrhu heuristické funkce může dojít ke snížení přesnosti výsledků nebo dokonce k nalezení suboptimální cesty.

Pseudokód:

Dijkstra (vstup: uzly (N), hrany (E), start (u), konec (v))

    Inicializuj seznam předchůdců P hodnotou -1
    Inicializuj seznam minimálních vzdáleností D hodnotou ∞
    Vytvoř prioritní frontu PQ

    Start má vzdálenost D rovnou 0
    Přidej start do PQ

    Dokud není PQ prázdná:
        Odeber z PQ uzel s nejnižší hodnotou D
        Pokud uzel nemá další hrany:

Pokračuj na další iteraci
Procházej sousední hrany:
Pokud D[v] > D[u] + w_uv:
Relaxuj hranu (D[v] = D[u] + wuv)
Ulož předchůdce uzlu v (P[v] = u)
Aktualizuj PQ (Vlož (D[v], v) do PQ)

Navrať seznam předchůzců P a seznam min. vzdáleností D[v]

Navazující kód pro rekonstrukci cesty (*reconstruct_path*) umožňuje prostřednictvím seznamu předchůzců (*list of predecessors*) projít od cíle ke startu navštívené uzly nejkratší cesty. Vytvořením seznamu *path* je následně možné tuto cestu vizualizovat. Navrácená minimální vzdálenost D[v] slouží ke kvantifikaci výsledků.

Pseudokód:

reconstruct_path(vstup: seznam předchůdců (P), start (u), cíl (v))
Inicializuj seznam *path*

Dokud jsme se nedobrali startovního uzlu a *v* není -1:
Přidej *v* do *path*
Předchůdce *v* je nové *v* (v = P[v])

Přidej *u* do *path*
Pokud *v* není *u*, upozorni zprávou: „Chybná cesta“

Navrať zrekonstruovanou cestu (*path*)

b) Jarníkův (Primův) algoritmus je určen k nalezení minimální kostry ohodnoceného grafu. Poprvé jej formuloval Vojtěch Jarník na počátku 30. let 20. století, přičemž v 50. letech jej nezávisle na sobě znovuobjevili Robert C. Prim a Edsger W. Dijkstra. Algoritmus nachází využití zejména v síťové optimalizaci, například při návrhu telekomunikačních sítí, rozvodu elektrické energie či návrhu silniční infrastruktury, kde je cílem minimalizovat celkové náklady na propojení všech uzlů. Primův algoritmus buduje minimální kostru postupně jako rostoucí strom – v každém kroku přidává nový vrchol propojený s již vybranými uzly hranou s nejnižším ohodnocením (greedy strategie). Pro efektivní vyhledávání nejkratších hran vedoucích k dosud nezařazeným vrcholům využívá prioritní frontu a pokračuje, dokud nejsou všechny uzly součástí kostry.

V porovnání s Kruskalovým algoritmem, který pracuje spíše metodou postupného spojování komponent, je Primův algoritmus výhodnější pro husté grafy. Naopak Kruskalův algoritmus se lépe hodí pro řídké grafy díky své efektivitě při třídění hran. Další alternativou je Borůvkův algoritmus, který pracuje na principu paralelního spojování komponent a je vhodný pro distribuované výpočty.

Pseudokód:

```
Jarník/Prim (vstup: uzly (N), hrany (E))
    Inicializuj seznam hran minimální kostry
    Inicializuj celkovou váhu (total_weight = 0)
    Inicializuj seznam navštívených uzlů P jako nenavštívené

    Vytvoř prioritní frontu PQ

    Zvol startovní uzel (u = 0) a označ ho jako navštívený
    Vlož hrany startovního uzle do prioritní fronty

    Dokud PQ není prázdná a seznam hran minimální kostry není plný:
        Odeber hranu s nejnižší váhou
        Pokud je uzel již navštívený
            Pokračuj další iterací

        Přidej hranu do seznamu hran minimální kostry
        Přičti váhu (w) k celkové váze (total_weight)
        Označ uzel jako navštívený
        Přidej všechny hrany z uzle do PQ, pokud jejich cílový uzel nebyl navštíven

    Navrať minimální kostru a celkovou váhu
```

c) Použitým algoritmům předcházely tzv. ***support* funkce**, jejichž úkolem bylo řešit specifické scénáře, které mohly v rámci zpracování vstupních dat nastat. V rámci tvorby grafu a párování uzlů a hran byla vytvořena funkce *findClosestNode*, která předcházela nepřesnostem v zaokrouhlování souřadnic a limitovala množství uzlů bez přiřazených hran prostřednictvím hledání eukleidovsky nejbližších uzlů ke konci a počátku každé hrany. Funkce *cleanGraph* prochází všechny uzle a odstraňuje ty, které nemají přiřazenou žádnou hranu. Dalším předzpracovávacím nástrojem se stala funkce *fixNegativeValue*, která prostřednictvím vytvořeného offsetu umožňuje algoritmům zpracovat i negativně ohodnocené hrany v grafu. Na to navazuje funkce *cancelOffset*, která následně zvrací efekt funkce předchozí a umožňuje správnou interpretaci výsledků. Opomenout nelze ani vizualizační funkce, které prostřednictvím knihovny *matplotlib* vykreslují graf včetně rekonstrukce cesty (*graphVisualization*) a minimální kostru (*minimalSpanningTree*).

2) Použitá metrika

Zatímco v prvním případě hledání cesty grafem byla váha hran přímo rovna jejich eukleidovským vzdálenostem, v případě druhém bylo potřebné přijít s matematickým přepisem pro spravedlivé zohlednění reálných parametrů a jejich dopadu na hledání skutečné časově nejkratší vzdálenosti. Prvním takto zvoleným parametrem byla **maximální povolená rychlost**, kterou bylo možné uzel překonat:

Dálnice (třída 1): 130 km/h
Rychlostní silnice (třída 2): 110 km/h
Silnice I. třídy (třída 3): 90 km/h
Silnice II. třídy (třída 4): 70 km/h
Silnice III. třídy (třída 5): 50 km/h

S předpokladem, že se vozidla v rámci testování Dijskrova algoritmu pohybují právě rychlostí odpovídající jejich třídě, byl čas potřebný k překonání hrany stanovený dle následujícího vzorce:

$$travel\ time\ [s] = \frac{edge\ length\ [m]}{\max speed\ [\frac{m}{s}]}$$

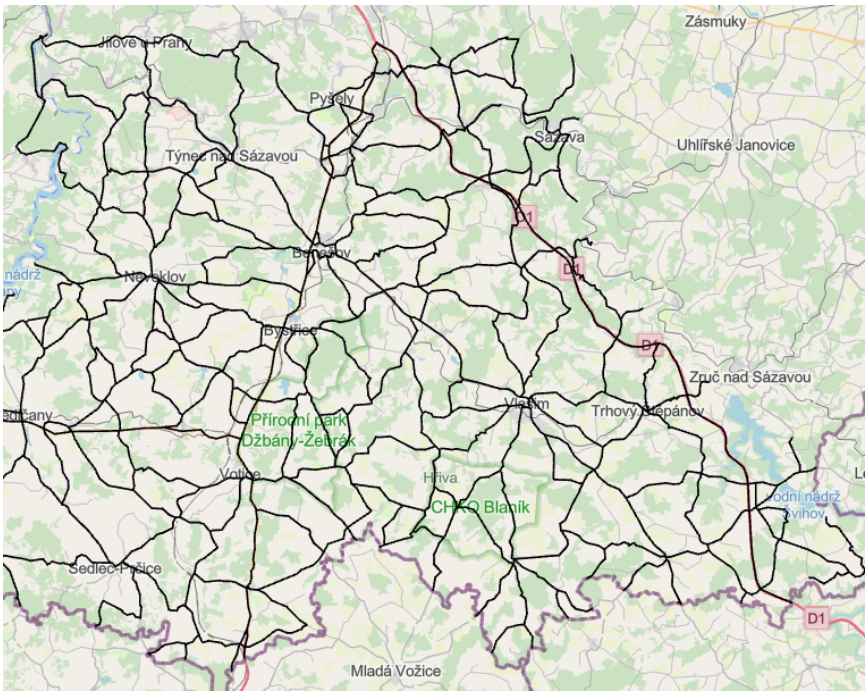
Druhým parametrem byla zvolena křivost (curvature), který zohledňoval časovou náročnost spojenou s objížděním překážek, převýšením atp. Na úseku mezi dvěma spojenými uzly jej šlo definovat takto:

$$curvature\ = \frac{eucleidian\ distance}{total\ distance\ by\ road}$$

Tímto parametrem byl předchozí *travel time* vynásoben. V případě přímé cesty se cestovní čas nezměnil, zatímco v případě křivosti silnice byl cestovní čas spojený s překonáním hrany navýšen.

3) Vypracování

Úloha byla zpracována za využití softwaru ArcGIS Pro (*3.2.0*) a programovacího jazyka Python. Za testovací množinu silnic byla zvolena oříznutá data z jihozápadní části středočeského kraje z databáze ArcČR 500 (*3.3*). Pro potřeby algoritmu byl vygenerován textový soubor se čtyřmi atributy souřadnic bodů (počátku a konce silnice), třídou silnice a její délkou. Na základě silničních dat byly generovány v místech intersekcí uzly, za něž byly do textového souboru předány hodnoty souřadnic (obr. 1)

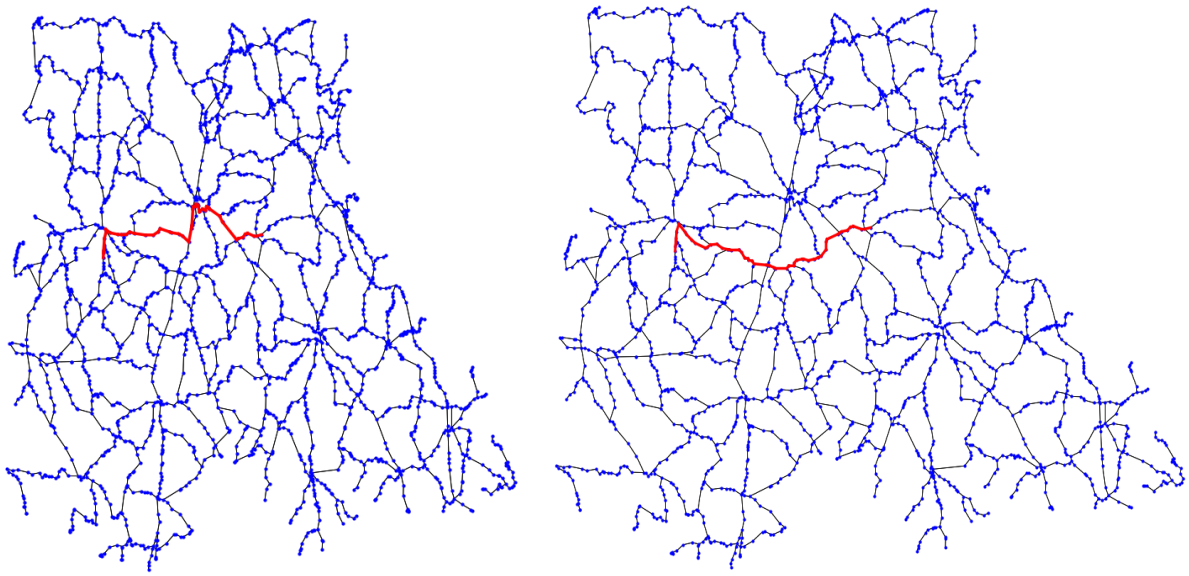


Obrázek 1 - Silniční síť

Připravená data byla konvertována pro potřeby práce s algoritmem do podoby seznamu (souřadnice uzlů) a slovníků hran. První slovník uzlům přiřazoval počáteční a koncové souřadnice hran a jejich délku. Druhý slovník namísto délky obsahoval cestovní čas, který byl spočten na základě délky silnice a maximální povolené rychlosti na dané třídě komunikace (která byla považována za cestovní rychlost). Tento čas byl opraven vynásobením koeficientem křivosti (*curvature*), který představuje podíl mezi délkou silnice a eukleidovskou vzdáleností mezi jejími koncovými body. Výsledek byl zbaven prázdných množin způsobených nekorektními vstupy pomocí nadefinované funkce *cleanGraph*. Pro případ záporně ohodnocených hran skript obsahuje funkci *fixNegativeValues*, která hodnotám všech hran přičte absolutní hodnotu minimální hodnoty + 1. Toto ošetření je následně negováno funkcí cancelOffset po proběhnutí algoritmů. Jakožto algoritmus pro generování minimální kostry byl zvolen Jarníkův algoritmus (*Prim*).

4) Výsledek

Dijsktra algoritmus byl testován pro čtyři různé vstupy, přičemž výsledky byly porovnány s navigačním softwarem Mapy.cz. Prvně důkaz, že algoritmus při hledání nejkratší vzdálenosti a nejkratšího transportního času skutečně došel k jiným výsledkům (obrázek 2).



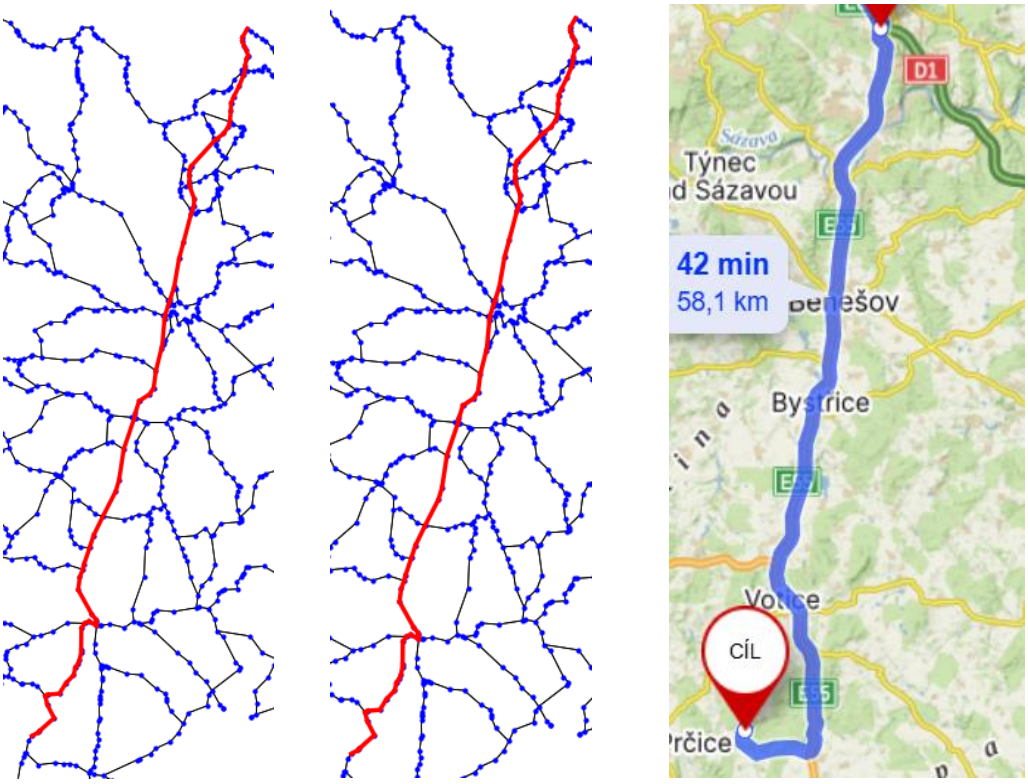
Obrázek 2 - Srovnání rozdílné volby trasy (vlevo: čas, vpravo: nejkratší vzdálenost)

Při porovnání byly dvojice bodů voleny náhodně. Výsledky poukazují na drobné nepřesnosti, kterou mohou mít na svědomí rozdíly v použitém datasetu, nepřesně nastavené váhy, chyby a generalizační procesy v algoritmu nebo pokročilost současných navigačních systémů a aktuální dopravní situace. Rozdíly mezi oběma systémy shrnuje tabulka 1.

Uzel		Dijsktra		Mapy.cz		Rozdíl	
Start	Cíl	délka [km]	čas [min]	délka [km]	čas [min]	délka [km]	čas [min]
Mirošovice	Ješetice	60	59.9	58.2	42	1.8	17.9
Jenkov	Chocerady	46.2	49.1	34.6	41	11.6	8.1
Rabyně	Čestín	50.2	47.9	37.1	44	13.1	3.9
K. Střimelnice	Kosova Hora	65	56.5	50.6	47	14.4	9.5

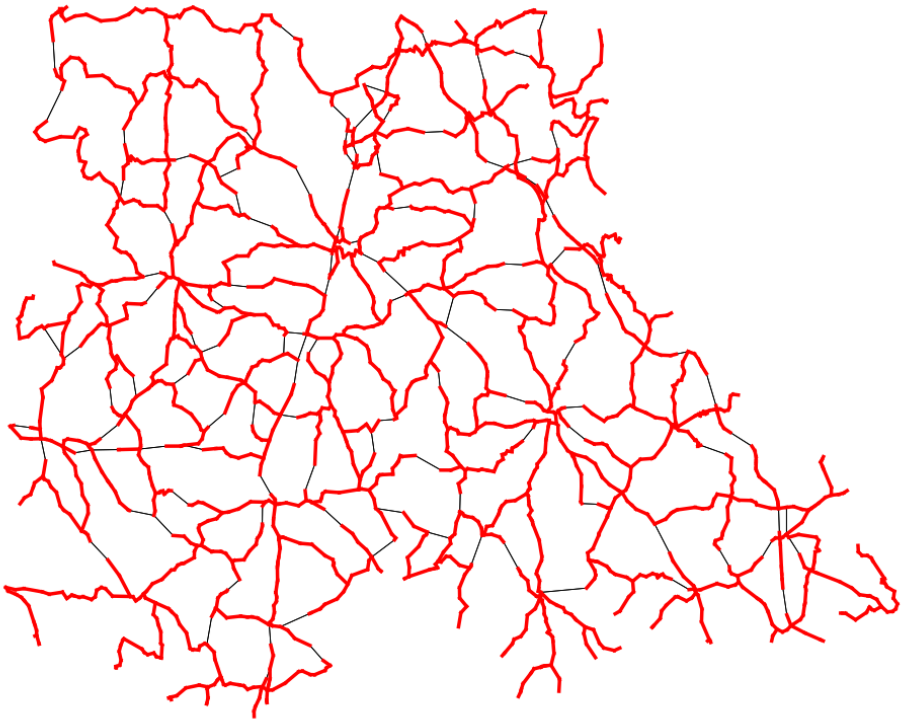
Tabulka 1 - Srovnání Dijkstra a Mapy.cz

Názornost srovnání vizualizuje obrázek 3.



Obrázek 3 - Ješetice - Mirošovice (vlevo: nejkratší, střed: nejrychlejší, vpravo: Mapy.cz

V rámci úlohy byla také vypracována funkce pro nalezení nejmenší kostry Jarníkovou metodou (obrázek 4). Délka kostry odpovídá přibližně 950 km.



Obrázek 4 - Minimální kostra

5) Závěr

Byl implementován Dijkstra algoritmus pro nalezení nejkratší cesty grafem a Jarníkův algoritmus pro nalezení minimální kostry. Zároveň byly testovány dvě varianty metrik – nejkratší eukleidovská vzdálenost a nejmenší transportní/cestovní čas, přičemž výsledky byly porovnány s navigačním softwarem Mapy.cz. Analýza ukázala, že Mapy.cz prezentují kratší trasy s menším transportním časem. Odchylky oproti navigačnímu softwaru lze přičíst odlišným datasetům, nastavení vah a aktuální dopravní situaci. Implementací Jarníkova algoritmu byla nalezena minimální kostra grafu, jež odpovídá přibližně 950 km.

6) Zdroje

ARCDATA PRAHA (2016): ArcČR 500, digitální geografická databáze 1: 500 000, verze 3.3.

BAYER, T. (2024): Nejkratší cesta grafem. Minimální kostra. Výuková prezentace. Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta Univerzity Karlovy

GEEKSFORGEEKS (2025): How to find Shortest Paths from Source to all Vertices using Dijkstra's Algorithm, Geeksforgeeks. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

KOLÁŘ, J. (2004): Teoretická informatika, Česká informatická společnost, 2. vyd.

MAPY.CZ (2025): Turistická mapa, Seznam.cz. https://mapy.cz/turisticka