UNIVERZITA PALACKÉHO V OLOMOUCI

PEDAGOGICKÁ FAKULTA

Katedra matematiky

Bakalářská práce

Jakub Ševčík

Hledání nejkratší cesty v ohodnoceném grafu s využitím softwaru

Olomouc 2024

Vedoucí práce: doc. RNDr. Jitka Laitochová, CSc.

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a uvedl jsem v ní veškerou literaturu a ostatní informační zdroje, které jsem použil.

V Olomouci dne ..... Podpis .....

Poděkování

Rád bych poděkoval vedoucí své bakalářské práce paní doc. RNDr. Jitce Laitochové, CSc. za inspiraci, vstřícný přístup a cenné rady při vedení práce. Dále bych chtěl poděkovat své rodině, přítelkyni a kamarádům za jejich podporu a motivaci během celého studia a během psaní bakalářské práce.

Anotace

Jméno a příjmení:	Jakub Ševčík
Katedra:	Katedra matematiky
Vedoucí práce:	doc. RNDr. Jitka Laitochová, CSc.
Rok obhajoby:	2024

Název práce:	Hledání nejkratší cesty v ohodnoceném grafu s využitím softwaru
Název v angličtině:	Finding the shortest path in the evaluated graph using software
Zvolený typ práce:	Aplikační
Anotace práce:	Cílem bakalářské práce je implementace aplikace s využitím vyššího programovacího jazyka Python a knihoven Tkinter a NetworkX. Při implementaci aplikace je využit Dijkstrův algoritmus pro hledání nejkratší cesty v ohodnocených grafech.
Klíčová slova:	Teorie grafů, ohodnocený graf, hledání nejkratší cesty, Dijkstrův algoritmus, Python, Tkinter, NetworkX, implementace aplikace
Anotace v angličtině:	The aim of the bachelor thesis is to implement an application using the higher-level programming language Python and the libraries Tkinter and NetworkX. The implementation of the application uses Dijkstra's algorithm to find the shortest path in the evaluated graphs.
Klíčová slova v angličtině:	Graph theory, weighted graph, shortest path search, Dijkstra's algorithm, Python, Tkinter, NetworkX, application implementation
Přílohy vázané v práci:	0
Rozsah práce:	53 stran
Jazyk práce:	Čeština

Obsah

Úvod

Hledání nejkratší cesty v grafu patří mezi klíčové matematické problémy s řadou rozsáhlých aplikací v různých oborech. V této bakalářské práci se zaměříme na jednotlivé pojmy z oblasti teorie grafů, konkrétně na pojmy související s hledáním nejkratší cesty v ohodnocených grafech. Cílem práce bude popis a implementace námi vytvořené aplikace, která bude v neorientovaných i orientovaných ohodnocených grafech hledat nejkratší cestu zvoleným algoritmem. Aplikace bude rovněž sloužit jako edukační prostředek pro výuku na školách s možností rozvíjet mezipředmětové vztahy.

Práce je rozdělena na teoretickou a praktickou část. Teoretická část zahrnuje stručnou historii o teorii grafů následovanou vysvětlením základních pojmů z oblasti teorie grafů, které jsou pro tuto práci podstatné. Dále jsou zde uvedeny nejčastěji se vyskytující typy grafů a podrobněji vysvětleno znázornění, ohodnocení a orientace grafů. Závěr teoretické části práce je vyhrazen pro popis algoritmů hledající nejkratší cestu v ohodnoceném grafu. Podrobněji je popsán Dijkstrův algoritmus, pomocí něhož bude aplikace hledat nejkratší cestu v ohodnocených grafech.

Praktická část se zaměřuje na popis, ovládání a funkčnost aplikace. Následně je zde stručně popsán vyšší programovací jazyk Python a knihovny, které jsou využity při vytváření aplikace. Další částí je implementace aplikace, kde jsou podrobněji vysvětleny jednotlivé funkce a jejich využití. V závěru praktické části práce jsou reprezentovány vzorové příklady, jejichž řešení je ověřeno s pomocí vytvořené aplikace.

TEORETICKÁ ČÁST

1. Historie teorie grafů

Teorie grafů vznikla v 18. a 19. století jakožto nástroj pro řešení různých typů úloh. Za zakladatele teorie grafů je považován matematik L. Euler, který se zabýval a vyřešil známou úlohou o sedmi mostech města Královce. Tato úloha spočívala v tom, zda je možné přejít přes všech sedm mostů právě jednou. L. Euler jako první dokázal, že to možné není. Teorie grafů však zaznamenala nárůst problémových úloh převážně v 19. století v různých odvětvích. Mezi významné osobnosti patří například fyzik G. Kirchhoff zabývající se problematikou konstrukce elektronických obvodů nebo A. Cayley zabývající se organickou chemií. (Sedláček, 1977)

Pro teorii grafů je ovšem velmi významné 20. století, právě kvůli vzniku odborné literatury a rozvoji výpočetní techniky. V roce 1936 významný maďarský matematik D. König vydal první obsáhlejší publikaci o teorii grafů *„Theorie der endlichen und unendlichen Graphen“*. Tato publikace se později v roce 1950 stala klasickou učebnicí, se kterou se vyučovalo ve školách po desetiletí. Dalším významným norským matematikem je Oystein Ore, který v 60. letech 20. století vydal tři publikace zabývající se problematikou teorie grafů. (Sedláček, 1977)

V Československu se objevily první zmínky o teorii grafů v roce 1926 v práci od matematika Otakara Borůvky, která se zabývá nalezením minimální kostry grafu. V 50. letech 20. století vznikají další práce, vycházející především z publikace D. Königa, od známých československých matematiků mezi, než patří například Jiří Sedláček nebo Miroslav Fiedler. První československou publikaci zabývající se teorií grafů vytvořil v roce 1964 Jiří Sedláček pod názvem *„Úvod do teorie grafů“*. Nejpoužívanější současnou publikací je *„Kapitoly z diskrétní matematiky“* od J. Matouška a J. Nešetřila. (Šišma, 1997)

Vzhledem k tomu, že publikace o teorii grafů se začaly vytvářet ve 20. století, tak se v terminologii neobjevují archaická slova a taktéž se nevytváří nové výrazy. Používají se slova především hovorová například uzel, hrana, cesta a další. Teorie grafů není výlučně spjatá pouze s matematikou, jelikož ji odborníci z různých odvětví dále rozvíjejí například ve fyzice, chemii, biologii, elektrotechnice nebo v ekonomice či lingvistice. (Sedláček, 1977)

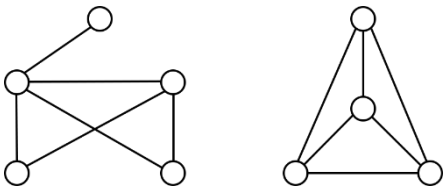
2. Základní pojmy teorie grafů

Tato kapitola se bude zaměřovat na základní pojmy z oblasti teorie grafů. Výčet veškerých pojmů z oblasti teorie grafů nebude v této práci zcela kompletní. Budou zde popsány především nejdůležitější pojmy z oblasti teorie grafů a pojmy, které jsou důležité pro tuto práci a které budou použity v následujících kapitolách.

2.1 Graf

Pod pojmem *graf* si můžeme představit spoustu různých věcí v různých oborech, nicméně v mnoha situacích v matematice se vždy bude jednat, ať už o konečnou, či nekonečnou množinu bodů a spojnice mezi některými dvojicemi bodů. Pro představu mohou body reprezentovat například krajská města v České republice a spojnice dálnice mezi nimi. V takovém případě nazveme bod *vrcholem* (je možné použít i označení *uzel*) a spojnici nazveme *hranou*. (Matoušek, 2009)

Definice: *Graf (rozšířeně obyčejný či jednoduchý neorientovaný graf) je uspořádaná dvojice* $G = (V, E)$*, kde* V *je množina vrcholů a* E *je množina hran – množina vybraných dvouprvkových podmnožin množiny vrcholů.*



Obrázek 1: Příklady grafů
Zdroj: Vlastní zpracování

Grafy v podobě obrázku výše a dále v textu značí znázornění grafů do roviny. Takové znázornění grafů je velmi užitečné, jelikož plní pomocnou úlohu pro orientování se v těchto grafech. V našem případě vrchol znázorníme do roviny pomocí kruhu a hranu pomocí úsečky mezi danou dvojicí vrcholů. Uvnitř jednotlivých kruhů se taktéž mohou vyskytovat číselné hodnoty, které slouží k pojmenování vrcholů. Podrobněji bude znázornění grafů do roviny popsáno ve 4. kapitole této práce.

Dále v textu se budeme zabývat konečnými grafy neboli grafy s konečnou neprázdnou množinou vrcholů. Pro množinu vrcholů grafu G budeme užívat značení $V(G)$ a pro množinu hran $E(G)$.

2.2 Isomorfismus grafů

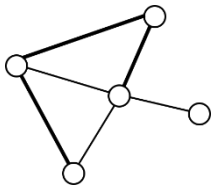
Definice: *Isomorfismus grafů G a H je bijektivní (navzájem jednoznačné) zobrazení $f: V(G) \to V(H)$, pro které platí, že každá dvojice vrcholů $u, v \in V(G)$ je spojena hranou v G právě tehdy, když je dvojice $f(u), f(v)$ spojena hranou v H.*

Jinými slovy by se dalo říci, že grafy G a H jsou isomorfní, pokud grafy mají totožné množiny vrcholů a hran. To znamená, že $G = H$, pokud platí, že $V(G) = V(H)$ a zároveň $E(G) = E(H)$. V takovém případě budeme isomorfní grafy značit $G \cong H$. Pod pojmem isomorfismus si v teorii grafů dokážeme běžně představit různé druhy značení vrcholů, jinak řečeno dva grafy jsou totožné až na názvy daných vrcholů. (Matoušek, 2009)

2.3 Podgraf

Pod pojmem *podgraf* si můžeme zjednodušeně představit, že z grafu G odstraníme některé vrcholy a hrany. V tuto chvíli si představme, že bychom chtěli vytvořit podgraf H z grafu G odstraněním veškerých vrcholů a zachováním všech hran. Téhle situaci je zapotřebí se vyvarovat, jelikož by se nejednalo o podgraf, a proto musíme zavést přesné tvrzení. (Kovář, 2014)

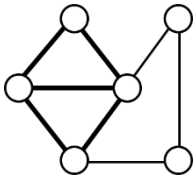
Definice: *Graf H nazveme podgrafem grafu G, jestliže $V(H) \subseteq V(G)$ a $E(H) \subseteq E(G)$. Píšeme $H \subseteq G$.*



Obrázek 2: Příklad podgrafu
Zdroj: Vlastní zpracování

Zavedená definice ukazuje, že při vynechání některých vrcholů z grafu G musíme vynechat hrany, které jsou spjaté s vynechanými vrcholy, ale taktéž mohou být vynechány i hrany, které s vynechanými vrcholy spjaté nejsou. Vynecháním těchto vrcholů a hran vznikne podgraf H. V případě, že vynecháme některé vrcholy a pouze hrany s nimi spjaté z grafu G, nazveme podgraf I *indukovaným podgrafem*. (Kovář, 2014)

Definice: *Podgraf I grafu G nazveme indukovaným podgrafem grafu G, jestliže $E(I)$ obsahuje všechny hrany grafu G, které jsou incidentní s vrcholy z $V(I)$.*

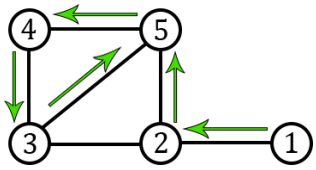


Obrázek 3: Příklad indukovaného podgrafu
Zdroj: Vlastní zpracování

2.4 Sled

V teorii grafů je *sled* určitý způsob procházení grafu G, tudíž budeme sled považovat za podgraf grafu G. Abychom byli schopni se v grafu pohybovat, musíme si stanovit *počáteční* a *koncový* vrchol. Počáteční vrchol označíme v_0 a koncový vrchol označíme v_n, kde $n \in \mathbb{N}_0$. V případě, že $n = 0$, hovoříme o *triviálním* sledu. Pokud $v_0 = v_n$, kde $n \in \mathbb{N}$, hovoříme o *uzavřeném* sledu. Sled má velmi důležitou vlastnost a sice, že můžeme přes vrcholy a hrany procházet opakovaně, dokud nedojdeme do koncového vrcholu. Můžeme si například představit, že se ztratíme v centru města a budeme tak dlouho procházet ulicemi (i opakovaně), dokud nenajdeme místo, kam jsme chtěli dojít. (Kovář, 2014)

Definice: *Sled v grafu G je taková posloupnost vrcholů a hran $(v_0, e_1, v_1, e_2, v_2, \cdots, e_n, v_n)$, že hrana e_i má koncové vrcholy v_{n-1} a v_n pro všechna $i = 1,2, \cdots, n$. Sled se nazývá (v_0, v_n)-sled.*



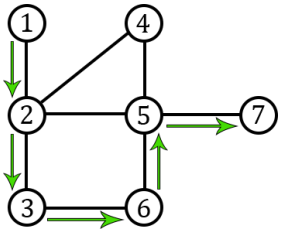
Obrázek 4: Sled v grafu
Zdroj: Vlastní zpracování

2.5 Cesta

V předešlé podkapitole jsme definovali pojem sled, který nyní bude úzce souviset s pojmem *cesta*, který je pro tuto práci zásadní. Cesta je sled, ve kterém se neopakují vrcholy. Stejně jako u sledu je zapotřebí i u cesty definovat počáteční a koncový vrchol. Počáteční vrchol budeme značit u a koncový vrchol v. Dále si popíšeme, co je to *délka*

cesty. Délka cesty je celkový počet hran, které se v cestě nacházejí mezi počátečním a koncovým vrcholem. V případě, že délka cesty bude rovna nule, budeme hovořit o *triviální* cestě. Je zřejmé, že pokud bychom chtěli hovořit o nejkratší netriviální cestě, tak délka cesty bude rovna jedné. (Kovář, 2023)

Definice: *Cesta v grafu je sled, ve kterém se neopakují vrcholy. Cestu s počátečním vrcholem u a koncovým vrcholem v budeme nazývat (u, v)-cesta.*



Obrázek 5: Cesta v grafu
Zdroj: Vlastní zpracování

2.6 Vzdálenost v grafu

Pojem *vzdálenost* zavádíme, jelikož vyžadujeme nalezení nejkratší délky (u, v)-cesty. Vzhledem k tomu, že délka cesty je vždy celé nezáporné číslo, tak i vzdálenost bude mít tuto vlastnost. Nyní se pojďme blíže podívat na definici vzdálenosti.

Definice: *Vzdálenost vrcholu u do vrcholu v (nebo vzdálenost mezi vrcholy u a v) v grafu G je délka nejkratší (u, v)-cesty. Značíme ji $dist(u, v)$, případně $dist_G(u, v)$. Jestliže vrchol v není dosažitelný z u, definujeme $dist(u, v) = \infty$.*

Všimněme si, že pokud neexistuje (u, v)-cesta, tak délka (u, v)-cesty není definována, ale přesto $dist(u, v)$ přiřadíme hodnotu nekonečna, a to především kvůli tomu, abychom jasně rozlišili dosažitelné vrcholy od nedosažitelných. Dále platí, že v neorientovaných grafech je $dist(u, v) = dist(v, u)$. V případě totožného počátečního a koncového vrcholu platí, že $dist(u, u) = 0$. Později v textu budeme vzdálenost modifikovat v souvislosti s ohodnocením hran v grafu.

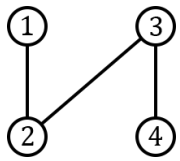
2.7 Matice sousednosti

Matice jsou v matematice velmi důležité, ať už se pohybujeme například v oblasti geometrie či algebry, a v teorii grafů tomu nebude jinak. V teorii grafů budeme matici používat k práci s příslušnými grafy. V matici budou řádky a sloupce znázorňovat

příslušné vrcholy a hodnoty na daných pozicích hrany. Takovou matici v teorii grafů nazveme jako *matice sousednosti* a budeme ji značit A_G. (Matoušek, 2009)

Definice: *Nechť* $G = (V, E)$ *je graf s* n *vrcholy. Označíme vrcholy* $v_1, \cdots, v_n$ *(v nějakém libovolném pořadí). Matice sousednosti grafu* G *je čtvercová matice* $A_G = (A_{ij})_{i,j=1}^{n}$ *definována předpisem* a_{ij} $\begin{cases} 1 & pro \{v_i, v_j\} \in E \\ 0 & jinak. \end{cases}$

Matice sousednosti bude vždy symetrická podle diagonály, pokud se bude jednat o neorientovaný graf. U orientovaných grafů tato vlastnost nemusí vždy platit. Hodnota prvků na diagonále bude vždy rovna nule, jelikož se v této práci zaměřujeme na nalezení $dist(u, v)$, tak nebudeme brát v potaz smyčky neboli hrany se stejným počátečním a koncovým vrcholem. Taktéž se nebudeme zabývat násobnými hranami mezi dvojicemi různých vrcholů. Dále je potřeba říci, že pokud grafy G a H mají totožnou matici sousednosti, tak $G \cong H$. Matice sousednosti bude taktéž jako vzdálenost modifikována později v textu.



$$A_G = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

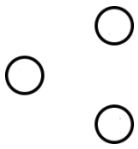
Obrázek 6: Matice sousednosti vycházející z uvedeného grafu
Zdroj: Vlastní zpracování

3. Nejdůležitější typy grafů

V teorii grafů se velmi často objevují různé typy grafů, které postupem času nabyly svých vlastních označení. Nyní se blíže podíváme na několik typů grafů, které jsou pro tuto práci důležité.

Nulový graf

Graf považujeme za *nulový* v případě, kdy $E(G)$ bude prázdná množina hran, tudíž každý vrchol z množiny $V(G)$ bude izolovaný. Nulový graf s počtem n vrcholů budeme značit N_n, kde $n \in \mathbb{N}$. N_n bude podgraf grafu G, pokud množina $V(G)$ bude obsahovat n vrcholů. (Saidur, 2017)

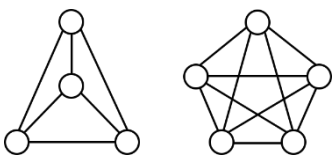


Obrázek 7: Nulový graf
Zdroj: Vlastní zpracování

Úplný graf

Graf nazveme *úplný* (nebo také *kompletní*), pokud mezi každou dvojicí různých vrcholů bude existovat hrana. To znamená, že množina $E(G)$ musí obsahovat všechny hrany vytvořené z kombinace vrcholů z množiny $V(G)$. Úplný graf označíme K_n, kde stejně jako u nulového grafu bude n značit počet vrcholů, tudíž $n \in \mathbb{N}$. Počet všech kombinací (hran) je vždy roven $\binom{n}{2}$. Dále platí, že každý libovolný graf G bude podgrafem grafu K_n, pokud součet prvků množiny $V(G)$ bude roven n. (Saidur, 2017)

Definice: *Graf na n vrcholech, kde $n \in N$, který obsahuje všech $\binom{n}{2}$ hran se nazývá úplný nebo také kompletní graf a značí se K_n.*

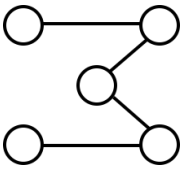


Obrázek 8: Příklady úplných grafů
Zdroj: Vlastní zpracování

Cesta

V podkapitole 2.5 jsme definovali pojem cesta jakožto podgraf grafu G, ale nyní se zaměříme na graf typu *cesta*. Cesta je typ grafu, kde mezi seřazenými vrcholy $v_1, v_2, \cdots, v_n$ bude existovat hrana (v_i, v_{i+1}), pro kterou platí, že $i \in \langle 1, n-1 \rangle \wedge i \in \mathbb{N}$. Tento typ grafu budeme značit P_n (z anglické slova *path*), kde n je počet vrcholů a $n \in \mathbb{N}$. (Saidur, 2017)

Definice: *Graf na n vrcholech, které jsou spojeny po řadě $n-1$ hranami se nazývá cesta a značí se P_n.*

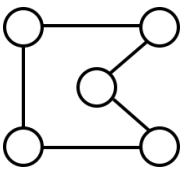


Obrázek 9: Graf typu cesta
Zdroj: Vlastní zpracování

Cyklus

Cyklus vznikne z grafu typu cesta tím, že vytvoříme novou hranu (v_1, v_n). Pro cyklus můžeme užívat i označení *kružnice*. Aby se jednalo o cyklus, musí být součet prvků množiny $V(G)$ větší roven nebo třem. Zajištěním této podmínky docílíme toho, aby nevznikl jiný typ grafu. Cyklus budeme značit C_n, kde $n \geq 3 \wedge n \in \mathbb{N}$ a dále platí, že n označuje počet vrcholů nebo hran (používá se i označení *délka cyklu*). (Saidur, 2017)

Definice: *Graf na n vrcholech (kde $n \geq 3$), které jsou spojeny po řadě n hranami tak, že každý vrchol je spojen s následujícím vrcholem a poslední vrchol je navíc spojen s prvním vrcholem, se nazývá cyklus na n vrcholech a značí se C_n. Číslo n je délka cyklu C_n.*



Obrázek 10: Graf typu cyklus
Zdroj: Vlastní zpracování

4. Znázornění, ohodnocení a orientace grafů

V této kapitole si nejprve popíšeme, jakým způsobem znázornit grafy do roviny a která pravidla je potřeba při znázorňování dodržet. Dále si vysvětlíme, co je to planární graf, který je úzce spojen s nejdůležitějšími typy grafů. V podkapitole 2.6 a 2.7 byly popsány pojmy, ve kterých bylo zmíněno ohodnocení grafů, na něž se v aktuální kapitole podrobněji zaměříme. Na závěr kapitoly se podíváme na rozdíl mezi orientovaným a neorientovaným grafem a užitečností orientovaných grafů.

4.1 Znázornění grafů do roviny

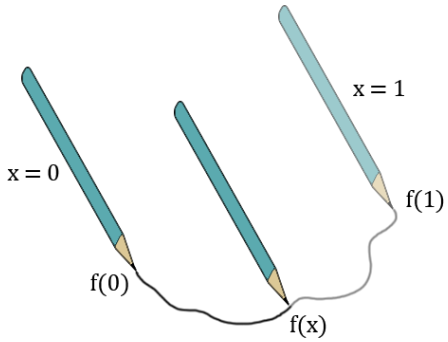
Až do této chvíle jsme považovali graf G za abstraktní strukturu, která obsahovala dvě množiny, a sice množinu $V(G)$ a $E(G)$. Nyní se zaměříme na to, jakým způsobem znázornit do roviny tyto množiny, kvůli lepšímu se orientování v grafech. Z předešlé části práce již máme základní představu o tom, jak grafy znázornit, ale i přesto zde popíšeme pravidla při znázornění grafů do roviny.

V prvém případě bude vyžadováno, aby všechny prvky z množiny $V(G)$ a $E(G)$ byly zakresleny do roviny. Prvky z množiny $V(G)$ budeme znázorňovat kruhem a prvky z množiny $E(G)$ úsečkou mezi danou dvojicí vrcholů. Uvnitř jednotlivých kruhů se taktéž mohou vyskytovat číselné hodnoty, které slouží k pojmenování vrcholů. Kromě úsečky můžeme použít i jakkoliv zakřivenou křivku, pokud si to situace žádá. Zakřivená křivka je ovšem poněkud nejednoznačné označení v teorii grafů, proto je potřeba zavést přesnou definici. Zakřivenou křivku budeme označovat jakožto *oblouk*. (Matoušek, 2009)

Definice: *Oblouk je podmnožina roviny tvaru* $\gamma = f([0,1]) = \{f(x); x \in [0,1]\}$, *kde* $f : [0,1] \to \mathbb{R}^2$ *je nějaké prosté spojité zobrazení uzavřeného intervalu* $[0,1]$ *do roviny. Přitom body* $f(0)$ *a* $f(1)$ *se jmenují koncové body oblouku* γ.

Všimněme si, že oblouk je definován na intervalu $[0,1]$, což budeme považovat za určitý časový úsek při znázornění křivky do roviny z počátečního vrcholu $f(0)$ do koncového vrcholu $f(1)$. To znamená, že $f(x)$ bude znázorňovat aktuální pozici hrotu tužky na křivce v rovině v čase x. (Kovář, 2023)

Dále je vyžadováno, aby se dvě navzájem různé hrany protínaly nejvýše v jednom bodě a v případě nutného zakřivení dané hrany je nezbytné, aby se hrana sama se sebou nekřížila. Dále je důležité, aby se vrcholy nepřekrývaly a aby hrana procházela vždy mezi danou dvojicí vrcholů bez protnutí dalšího vrcholu.

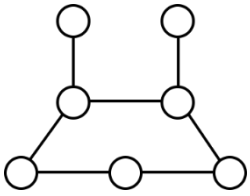


Obrázek 11: Oblouk
Zdroj: Vlastní zpracování

4.1.1 Planární graf

Planární (neboli také *rovinný*) graf, je takový typ grafu, který musí splňovat následující podmínky. V prvém případě musí platit, že daný graf lze znázornit do roviny, jak je uvedeno v předešlé části kapitoly. Dále musí platit, že všechny vrcholy a hrany budou uspořádány tak, aby dvě navzájem různé hrany měly společný bod pouze v počátečním nebo koncovém vrcholu. Jinými slovy se hrany nesmí navzájem křížit. (Kovář, 2023)

Definice: *Graf je rovinný, jestliže máme dáno jeho nakreslení do roviny takové, že žádné dvě křivky, které odpovídají hranám, nemají společné body kromě koncových bodů (vrcholů).*



Obrázek 12: Planární graf
Zdroj: Vlastní zpracování

Planární graf je nejlepším znázorněním grafu do roviny, jelikož analýza takového grafu je pro člověka mnohem intuitivnější, což výrazně usnadňuje práci s těmito grafy. Nicméně mohou nastat případy, kdy graf není možné znázornit planárně. Pro ukázku si představme, že budeme chtít do roviny znázornit graf K_5 (úplný graf) bez křížení hran. Po určité době dojdeme k závěru, že vždy najdeme dvě různé hrany, které se musí

bezprostředně křížit, jinak graf do roviny neznázorníme. V takovém případě graf nelze znázornit do roviny planárně.

4.2 Ohodnocení grafů

Obvykle nám v mnoha situacích v reálném světě nestačí pouze vytvořit graf jako abstraktní strukturu, skládající se ze dvou množin $V(G)$ a $E(G)$. Dokonce i když daný graf znázorníme v rovině, tak často potřebujeme určitým způsobem lépe charakterizovat danou situaci. Z tohoto důvodu se velmi často přiřazuje jednotlivým hranám a vrcholům číselná hodnota. Číselné hodnoty mohou například reprezentovat vzdálenost, čas, náklady a jiné. Graf, který je opatřen v našem případě číselnými hodnotami, budeme označovat za *ohodnocený* graf. (Demel, 2019)

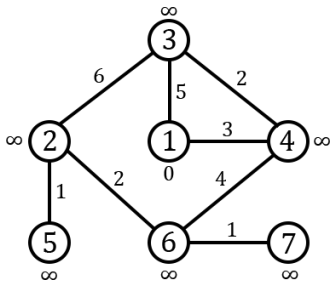
Definice: *Ohodnocený graf G je funkce $w : E(G) \to \mathbb{R}$, která každé hraně $e \in E(G)$ přiřadí reálné číslo $w(e)$, kterému říkáme váha hrany (značení w pochází z anglického „weight“). Ohodnocený graf je graf G spolu s ohodnocením hran reálnými čísly.*

Ve značné části praktických aplikací se setkáváme pouze s kladně ohodnocenými grafy. V určitých odvětvích se ovšem můžeme setkat i se záporně ohodnocenými grafy například ve financích, energetice či dopravě a logistice. Dále se z praktických důvodů vyplatí ohodnotit graf přirozenými čísly včetně nuly například z toho důvodu, že při spuštění naimplementovaných algoritmů může na různých zařízeních dojít k zaokrouhlovací chybě. Dále v textu se budeme zabývat převážně kladně ohodnocenými grafy s přirozenými čísly. (Kovář, 2014)

Definice: *Kladně ohodnocený (říkáme také vážený) graf G má takové ohodnocení w že pro každou hranu $e \in E(G)$ je její váha $w(e)$ kladná.*

Nyní se zaměříme na znázornění číselných hodnot do roviny. Číselné hodnoty jednotlivých hran se vždy znázorňují buď na střed přímky (oblouku) s tím, že se kousek přímky vynechá kvůli lepší čitelnosti hodnoty, nebo se hodnota znázorní ke středu mimo přímku (oblouk). Taktéž se číselné hodnoty vrcholů znázorňují buď vně kruhu, nebo

mimo kruh blízko vrcholu s podmínkou, že číselná hodnota nesmí překrývat hranu. Také je možné číselné hodnoty vrcholů přehledně zapsat do tabulky mimo graf.



Obrázek 13: Ohodnocený graf
Zdroj: Vlastní zpracování

Obrázek výše popisuje ohodnocený graf skládající se ze sedmi vrcholů, jež jsou označeny číselnými hodnotami uvnitř kruhu. Hrany mezi dvojicemi vrcholů jsou opatřeny číselnými hodnotami určující vzdálenost, kterou je potřeba překonat pro přesun z jednoho vrcholu do druhého. Číselné hodnoty se však vyskytují i v bezprostřední blízkosti každého vrcholu. Tyto hodnoty ukazují, jakou nejmenší vzdálenost je potřeba překonat z počátečního do koncového vrcholu. Na obrázku si lze povšimnout, že první vrchol je opatřen hodnotou 0 a zbývající vrcholy hodnotou ∞. Z toho vyplývá, že první vrchol bude zároveň počátečním vrcholem a u ostatních vrcholů momentálně neznáme nejkratší cestu z prvního vrcholu.

4.2.1 Vzdálenost v ohodnocených grafech

Jak již bylo řečeno v podkapitole 2.6, budeme vzdálenost nyní modifikovat. Vzdálenost v ohodnocených grafech již nebude počet všech hran mezi počátečním a koncovým vrcholem, ale bude se jednat o součet vah jednotlivých hran. Abychom pojem vzdálenost mohli modifikovat, tak nejprve musíme zavést délku ohodnoceného sledu. (Kovář, 2014)

Definice: *Mějme ohodnocený graf G s ohodnocením w. Délkou ohodnoceného sledu $S = v_0, e_1, v_1, e_2, v_2, \cdots, e_n, v_n$ v G rozumíme součet ohodnocení všech jeho hran. Každá hrana se počítá tolikrát, kolikrát se ve sledu S vyskytuje, a proto* $d_G^w(S) = w(e_1) + w(e_2) + \cdots + w(e_n)$.

V předchozí části textu jsme uvedli, že se budeme zabývat kladně ohodnocenými grafy, tudíž $d_G^w(S)$ bude nabývat hodnot tak, jak bylo již uvedeno v podkapitole 2.6. Nyní již můžeme modifikovat vzdálenost v ohodnocených grafech. (Kovář, 2014)

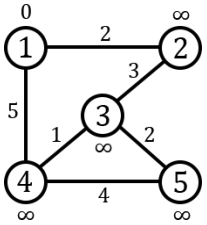
Definice: *(Váženou) vzdáleností mezi dvěma navzájem dosažitelnými vrcholy* u, v *ve váženém nebo ohodnoceném grafu* (G, w) *rozumíme číslo* $dist_G^w(u,v) = min\{d_G^w(S)$*, kde* S *je cesta s koncovými vrcholy* $u, v\}$*. Jestliže vrcholy* u *a* v *jsou nedosažitelné, klademe* $dist_G^w(u,v) = \infty$.

4.2.2 Matice sousednosti v ohodnocených grafech

V podkapitole 2.7 jsme vysvětlili a popsali, jakým způsobem zkonstruovat matici sousednosti grafu G. Nyní se zaměříme na modifikaci této matice v souvislosti s ohodnoceným grafem. Jednotlivé hodnoty prvků a_{ij} v matici sousednosti A_G zjednodušeně řečeno, nahradíme vahou hran všude tam, kde se do této chvíle vyskytovala hodnota rovna jedné. Modifikovanou matici sousednosti budeme nazývat *vážená matice sousednosti*. Je velmi výhodné matici sousednosti takto modifikovat vzhledem k implementaci algoritmů pro hledání nejkratší cesty v ohodnoceném grafu. (Čada, 2004)

Definice: *Vážená matice sousednosti ohodnoceného orientovaného grafu* (G, w) *s vrcholy* $v_1, \cdots, v_n$ *je matice* $W_G = (w_{ij})$*, kde* $w_{ij} = \begin{cases} w(v_i v_j) & pokud\ v_i v_j \in E(G), \\ 0 & jinak, \end{cases}$ *pro* $i, j = 1, \cdots, n$.

Výše uvedená definice poukazuje na orientovaný graf, nicméně daná definice bude platit, i kdybychom pracovali s grafy neorientovanými. Rozdíl spočívá v tom, že u neorientovaných grafů bude W_G symetrická, kdežto u orientovaných grafů tato vlastnost nemusí vždy platit.



$$W_G = \begin{pmatrix} 0 & 2 & 0 & 5 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 0 & 3 & 0 & 1 & 2 \\ 5 & 0 & 1 & 0 & 4 \\ 0 & 0 & 2 & 4 & 0 \end{pmatrix}$$

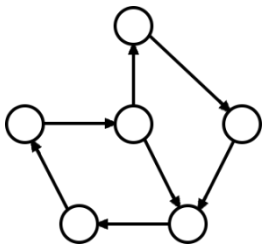
Obrázek 14: Vážená matice sousednosti vycházející z uvedeného grafu
Zdroj: Vlastní zpracování

4.3 Orientovaný graf

Až do této chvíle jsme se vždy zabývali neorientovanými (obyčejnými) grafy. Dříve než začneme popisovat orientovaný graf, se zaměříme na definici.

Definice: *Orientovaný graf je dvojice* (V, E)*, kde* V *je množina vrcholů a* $E \subset V \times V$ *je množina hran.*

Orientovaný graf vychází z definice neorientovaného grafu s tím rozdílem, že $E(G)$ již nebude množina dvouprvkových podmnožin množiny vrcholů, ale množina uspořádaných dvojic vrcholů. U orientovaných grafů již budeme důsledně rozlišovat počáteční a koncový vrchol příslušné hrany. Uspořádanou dvojici $[u, v]$ budeme chápat tak, že u je počáteční vrchol a v je vrchol koncový. Rozdíl oproti neorientovanému grafu spočívá v tom, že ne vždy je možné se dostat zpět z koncového do počátečního vrcholu, jelikož hrana $[u, v]$ není totožná s hranou $[v, u]$ a tím pádem tato hrana nemusí existovat. Taktéž platí, že váhy hran $[u, v]$ a $[v, u]$ mohou být různé. Z toho důvodu se vzdálenost mezi dvojicí vrcholů může lišit v závislosti na orientaci průchodu mezi dvojicí vrcholů. Avšak stejně jako u neorientovaných grafů se v orientovaných grafech nebudeme zabývat smyčkami a násobnými hranami. Orientované hrany budeme v rovině znázorňovat pomocí šipek, abychom přesně určili směr orientace hrany. (Čada, 2004)



Obrázek 15: Orientovaný graf
Zdroj: Vlastní zpracování

5. Hledání nejkratší cesty

Hledání nejkratší cesty v grafu je jednou z nejčastějších úloh v teorii grafů. Požadavek pro nalezení nejkratší cesty především vzniká právě kvůli mnoha praktickým aplikacím, například pokud je zapotřebí nalézt nejkratší cestu v dopravním spojení nebo pro výstavbu elektrické rozvodné sítě. Představme si, že bychom chtěli najít nejkratší dopravní spojení mezi Prahou a Ostravou. V takovém případě by nám jednoznačně nepostačoval jednoduchý orientovaný či neorientovaný graf, jelikož je potřeba v reálném světě znát, jaká je vzdálenost mezi jednotlivými stanicemi či čas potřebný k překonání těchto vzdáleností na trase mezi Prahou a Ostravou. Z tohoto důvodu budeme při hledání nejkratší cesty pracovat pouze s ohodnoceným grafem.

Pro nalezení nejkratší cesty v ohodnoceném grafu G existuje řada různých algoritmů. Za nejznámější a nejčastěji používaný je označován Dijkstrův algoritmus, na který se podrobněji zaměříme, jelikož jej budeme v praktické části této práce implementovat a dále využívat pro hledání nejkratší cesty v ohodnocených grafech. Nicméně existují i další známé a často používané algoritmy, například Floyd-Warshallův algoritmus či Bellman-Fordův algoritmus. Rozdíl mezi těmito algoritmy spočívá v tom, že Dijkstrův algoritmus dokáže nalézt nejkratší cestu pouze v kladně ohodnoceném grafu, kdežto Floyd-Warshallův algoritmus dokáže nalézt nejkratší cestu i v záporně ohodnoceném grafu, stejně tak jako Bellman-Fordův algoritmus. Dále platí, že Bellman-Fordův algoritmus dokáže v záporně ohodnoceném grafu detekovat záporný cyklus. Oproti tomu Floyd-Warshallův algoritmus záporný cyklus detekovat nedokáže. (Erciyes, 2021)

5.1 Dijkstrův algoritmus

Jak již bylo řečeno, nyní se podrobněji zaměříme na Dijkstrův algoritmus. Na úvod je důležité říci, že je potřeba mít kladně ohodnocený graf G a počáteční vrchol $u \in V(G)$. Dijkstrův algoritmus následně vypočítá $dist_G^w(u, v)$ pro každý vrchol $v \in V(G)$. Abychom však mohli konstatovat, že pro každý vrchol $v \in V(G)$ je $dist_G^w(u, v)$ definitivní hodnota po ukončení Dijkstrova algoritmu, musíme pro každý vrchol $v \in V(G)$ zavést pomocnou proměnnou $d(v)$. Tato pomocná proměnná nám bude sloužit k zapamatování si hodnoty $dist_G^w(u, v)$ v jednotlivých krocích Dijkstrova algoritmu. V určitých krocích Dijkstrova algoritmu můžeme konstatovat, že $d(v)$ je

definitivní hodnota pro některé vrcholy $v \in V(G)$, a tudíž $d(v) = dist_G^w(u, v)$. V opačném případě je potřeba $d(v)$ v následujících krocích přepočítat. Dále vytvoříme novou množinu P, která bude obsahovat vrcholy z množiny $V(G)$, u kterých jsme dosud definitivně neurčili $dist_G^w(u, v)$. (Matoušek, 2009)

Na začátku Dijkstrova algoritmu zavedeme $d(v) = 0$ pro $v = u$, $d(v) = \infty$ pro $v \neq u$ a $P = V(G)$. Je zřejmé, že na začátku známe vzdálenost z vrcholu u do vrcholu u, jelikož se jedná o počáteční vrchol, kdežto zbylé vrcholy ohodnotíme nekonečnem, protože jsme je dosud neprozkoumali. V následujících krocích Dijkstrova algoritmu se bude cyklicky provádět tentýž postup. Nejprve je potřeba vytvořit pomocnou množinu $R = min\{d(v) \mid v \in P\}$. Množina R tedy bude obsahovat vrcholy s nejmenší hodnotou $d(v)$ z množiny P. Všeobecně platí, že množina R bude nejčastěji obsahovat pouze jeden vrchol, ale mohou nastat i případy, kdy množina R může obsahovat více vrcholů s totožným ohodnocením $d(v)$. Následně z množiny P vyjmeme veškeré vrcholy obsažené v množině R. V dalším kroku porovnáme hodnotu $d(x)$ s hodnotou $d(v) + w(\{v, x\})$ pro každou hranu $\{v, x\} \in E(G)$, pro které platí, že $v \in R$ a $x \in P$. Pokud z vrcholu v vede do vrcholu x kratší cesta, než jsme doposud znali, tak platí $d(v) + w(\{v, x\}) < d(x)$ a hodnota proměnné $d(x)$ se změní na hodnotu $d(v) + w(\{v, x\})$. Po prozkoumání všech hran mezi vrcholy z množiny R a P se přejde na začátek cyklu a celý postup se znovu opakuje. Algoritmus končí, pokud je množina $P = \emptyset$ nebo pokud množina P obsahuje pouze vrcholy ohodnocené $d(v) = \infty$, což znamená, že zbylé vrcholy v množině P jsou nedosažitelné z počátečního vrcholu u. Na samotném konci algoritmus zajistí, že $d(v) = dist_G^w(u, v)$ pro každý vrchol $v \in V(G)$ z počátečního vrcholu $u \in V(G)$. (Matoušek, 2009)

Dijkstrův algoritmus je schopen nalézt nejkratší cestu v kladně ohodnocených neorientovaných i orientovaných grafech. V případě, že bychom chtěli hledat nejkratší cestu v kladně ohodnocených orientovaných grafech, tak se musí v zápisu Dijkstrova algoritmu změnit jedna věc. Namísto práce s hranami $\{v, x\} \in E(G)$ v neorientovaných grafech budeme pracovat s hranami $[v, x] \in E(G)$, jelikož v orientovaných grafech jsou hrany uspořádané dvojice vrcholů z množiny $V(G)$, tak jak je uvedeno v podkapitole 4.3.

PRAKTICKÁ ČÁST

1. Popis a ovládání aplikace

Cílem praktické části této práce je vytvoření aplikace pro hledání nejkratší cesty v ohodnocených grafech s využitím Dijkstrova algoritmu. Aplikace bude sloužit jako edukační prostředek pro výuku na školách, kontrolu výsledků žáků či studentů a rozvíjení mezipředmětových vztahů. Pro spuštění aplikace je vyžadován operační systém Windows a není vyžadována instalace či jiné podpůrné programy nebo aplikace. Nyní důkladně popíšeme strukturu aplikace a její ovládání.

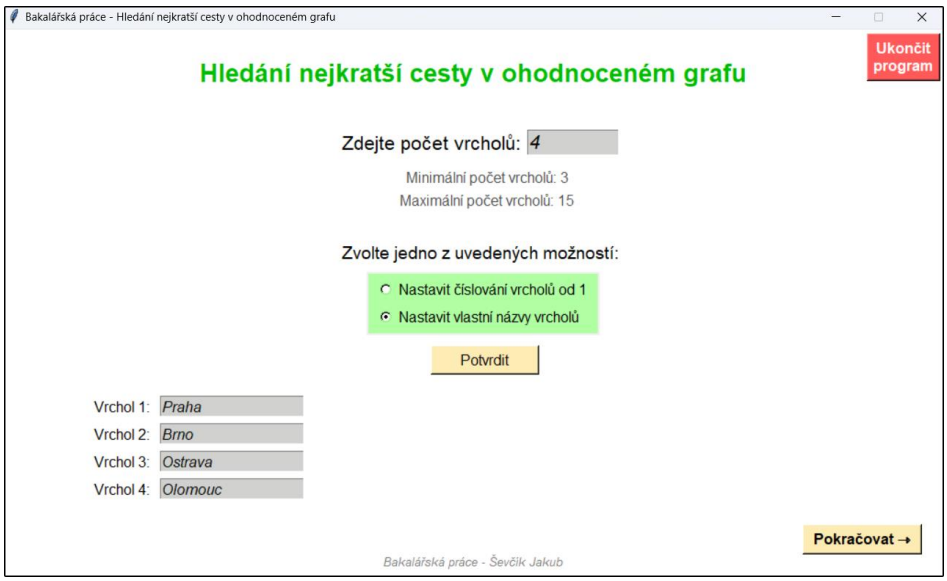
Aplikace nám bude sloužit k vytvoření si vlastního grafu. To znamená, že budeme schopni si zvolit počet vrcholů, definovat jednotlivé hrany mezi vrcholy a ohodnotit je, zvolit si počáteční a koncový vrchol při hledání nejkratší cesty v kladně ohodnoceném grafu a vybrat si, zdali chceme graf orientovaný či neorientovaný. Poslední možnost, kterou si budeme moci zvolit, bude, zdali chceme vytvořit a zobrazit postup při hledání nejkratší cesty v grafu anebo zdali chceme zobrazit pouze definitivní nalezení nejkratší cesty v grafu.

1.1 Zadání počtu vrcholů a jejich pojmenování

Po spuštění aplikace je zapotřebí, abychom nejprve zadali parametry pro tvorbu vrcholů, mezi něž patří počet vrcholů a jejich pojmenování. Počet vrcholů se zadává do vstupního pole a je možné jej zvolit v rozmezí od tří do patnácti. Rozmezí je zavedeno z následujících důvodů. Pokud by graf obsahoval méně než tři vrcholy, tak by hledání nejkratší cesty postrádalo význam. Taktéž platí, že pokud bychom zadali více než patnáct vrcholů, tak by graf mohl být poněkud nepřehledný. Avšak na funkčnosti aplikace by to vliv nemělo. V případě, že bychom zadali počet vrcholů mimo toto rozmezí, tak by aplikace nepokračovala do další části a upozornila by nás, abychom hodnotu změnili.

Po správném zadání počtu vrcholů si můžeme zvolit, zdali bychom chtěli jednotlivé vrcholy pojmenovat přirozenými čísly v rozsahu od jedné do hodnoty počtu vrcholů anebo zdali bychom chtěli jednotlivé vrcholy pojmenovat vlastními názvy. Pokud bychom zvolili první možnost, tak by aplikace pokračovala do části pro tvorbu a ohodnocení hran. Při volbě druhé možnosti se vytvoří nová vstupní pole podle toho, jaký počet vrcholů jsme zvolili. Do jednotlivých vstupních polí nyní můžeme zapsat názvy pro dané vrcholy dle naší libosti. Ovšem vlastní jména vrcholů jsou omezena

v množství znaků. To znamená, že každé jméno se může skládat pouze z deseti znaků, aby byl později graf co nejvíce přehledný. Pokud bychom některý vrchol nepojmenovali, tak aplikace nebude pokračovat do další části a upozorní nás, abychom u nepojmenovaných vrcholů doplnili jejich názvy. Totéž by se stalo, kdybychom dva různé vrcholy pojmenovali totožnými názvy, avšak s tím rozdílem, že by nás aplikace upozornila, abychom u těchto vrcholů změnili jejich názvy. Pokud je vše zadáno správně, tak aplikace bude pokračovat do další části pro tvorbu a ohodnocení hran.



Obrázek 16: Zadání počtu vrcholů a jejich pojmenování
Zdroj: Vlastní zpracování

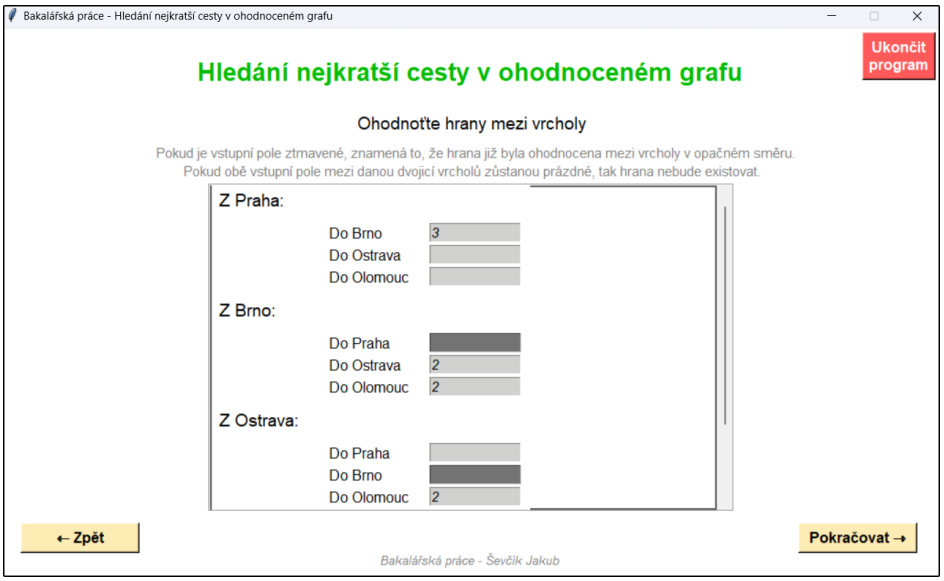
1.2 Tvorba a ohodnocení hran

V této části aplikace se budeme zabývat vytvořením a ohodnocením hran mezi danými dvojicemi vrcholů. Aplikace vytvoří přehlednou posuvnou tabulku, kde se budou vyskytovat veškeré možnosti výběru hran mezi danými dvojicemi vrcholů. Dále se zde bude u každé hrany vyskytovat vstupní pole, do kterého můžeme zadat své vlastní ohodnocení hrany. Ohodnocení hrany je, jako v minulé části aplikace, omezeno v daném rozsahu. Je možné zvolit hodnotu v rozmezí od jedné do řádově jednotek milionů. Ohodnotit hranu hodnotou nula postrádá jakýkoliv význam, jelikož by to znamenalo, že buď jsou dva vrcholy totožné, anebo hrana mezi danou dvojicí vrcholů neexistuje. Dále aplikace neumožňuje zadat zápornou hodnotu, jelikož se v této práci zabýváme pouze kladně ohodnocenými grafy, jak již bylo řečeno v předešlé části práce.

V případě, že určitou hranu ohodnotíme, tak je aplikací zahrnuta do našeho grafu. V opačném případě aplikace pracuje s tím, že daná hrana v grafu neexistuje. Pokud mezi

uspořádanou dvojicí vrcholů ohodnotíme příslušnou hranu, tak aplikace již nenabízí možnost ohodnotit hranu v opačném směru. Vstupní pole se pro ohodnocení hrany v opačném směru zatmaví a znepřístupní se pro zápis hodnoty, aby nedošlo k chybě v dalších částech aplikace. Je zřejmé, že u neorientovaných grafů postrádá význam ohodnotit tutéž hranu různými hodnotami. Aplikace by ovšem mohla pracovat s násobnými hranami, ale jak již bylo dříve řečeno, tak v této práci se násobnými hranami zabývat nebudeme. Z toho důvodu bude aplikace pracovat vždy s jednou hranou mezi danou dvojicí vrcholů u neorientovaných i orientovaných grafů. Pro orientovaný graf je navíc podstatné, zda hranu mezi dvojicí vrcholů ohodnotíme v opačném směru či nikoliv, jelikož podle toho bude hrana orientovaná určitým směrem.

Po úspěšném ohodnocení (vytvoření) hran mezi danými dvojicemi vrcholů, aplikace přejde do další části pro výběr počátečního a koncového vrcholu. Pokud by ovšem žádná z hran nebyla ohodnocena, tak i přesto bude aplikace pokračovat do další části, jelikož v takovém případě budou všechny vrcholy v grafu izolované.

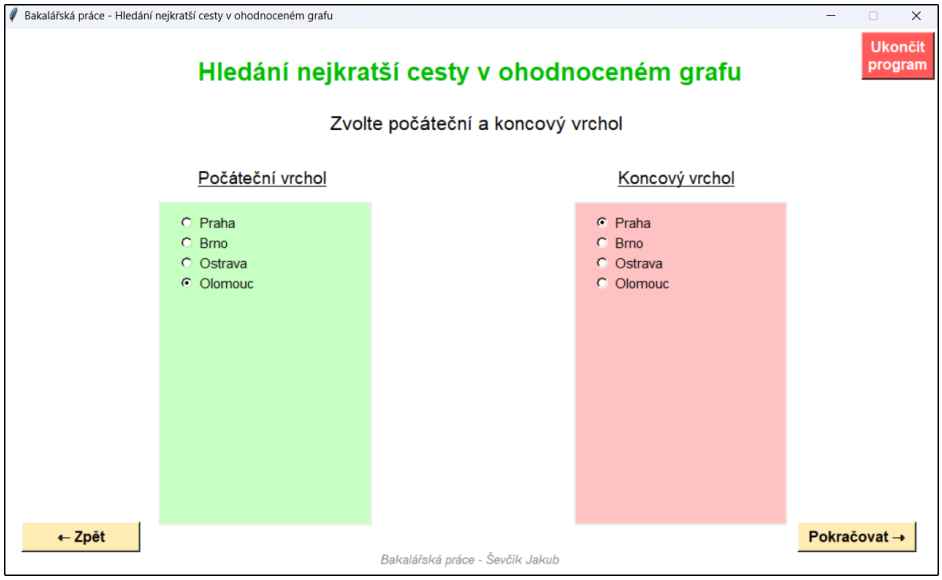


Obrázek 17: Tvorba a ohodnocení hran
Zdroj: Vlastní zpracování

1.3 Výběr počátečního a koncového vrcholu

V další části aplikace se budeme zabývat výběrem počátečního a koncového vrcholu. Aplikace nám bude nabízet na levé straně grafického uživatelského rozhraní výběr počátečního vrcholu ze všech vrcholů, které jsme vytvořili v první části aplikace. Totéž nám bude aplikace nabízet i na pravé straně, akorát zde již budeme vybírat koncový vrchol. Při vstupu do této části aplikace bude za počáteční a koncový vrchol vždy zvolen

první vrchol dle pořadí vytvořených vrcholů. Následně se můžeme rozhodnout dle vlastního uvážení, jaký počáteční a koncový vrchol si vybereme. Ovšem pokud si vybereme počáteční vrchol totožný s koncovým vrcholem, tak již dopředu budeme znát nejkratší cestu i vzdálenost, to jest rovno nule.



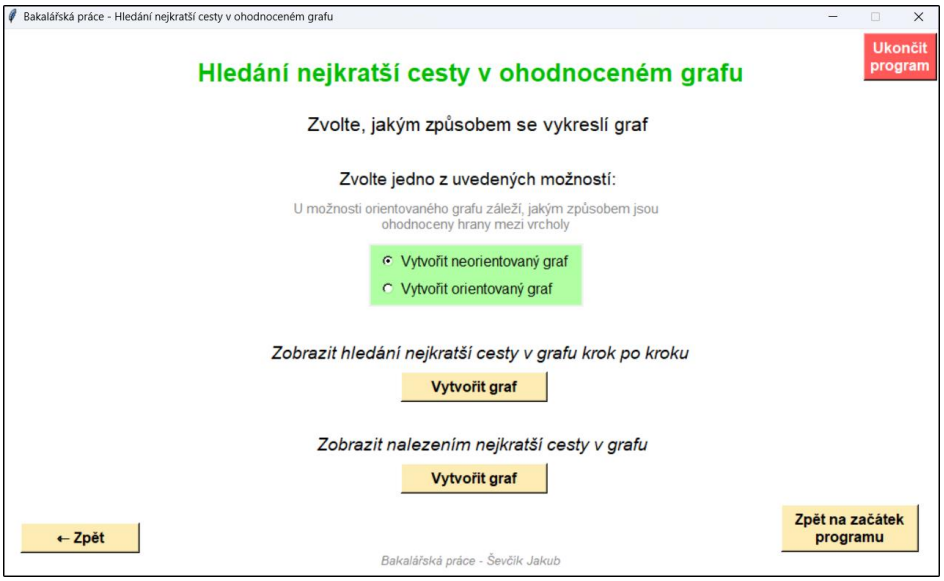
Obrázek 18: Výběr počátečního a koncového vrcholu
Zdroj: Vlastní zpracování

1.4 Orientace grafu a hledání nejkratší cesty

Nacházíme se v poslední části aplikace pro nastavení parametrů grafu a hledání nejkratší cesty. Aplikace umožňuje výběr mezi neorientovaným a orientovaným grafem. V případě, že si vybereme první či druhou možnost, tak aplikace vždy vytvoří váženou matici sousednosti skládající se z hodnot, které jsme zadali v části aplikace pro tvorbu a ohodnocení hran. Pokud zvolíme možnost neorientovaného grafu, tak matice sousednosti bude symetrická. Oproti tomu při volbě orientovaného grafu bude matice sousednosti symetrická pouze tehdy, pokud v grafu nebudou existovat hrany.

Po zvolení orientace grafu jsou veškeré parametry, pro vytvoření grafu a hledání nejkratší cesty z počátečního do koncového vrcholu, stanoveny. Již zbývá vybrat, jakým způsobem chceme zobrazit nalezení nejkratší cesty v grafu. Aplikace umožňuje výběr ze dvou možností, a sice znázornit postup hledání nejkratší cesty v grafu krok po kroku anebo znázornit pouze výsledek nalezení nejkratší cesty v grafu. Při výběru z obou možností se vždy využije Dijkstrův algoritmus pro hledání nejkratší cesty v grafu. Při průchodu Dijkstrovým algoritmem se budou postupně vytvářet obrázky jednotlivých kroků hledání nejkratší cesty, které se budou ukládat do počítače. Po dokončení tvorby

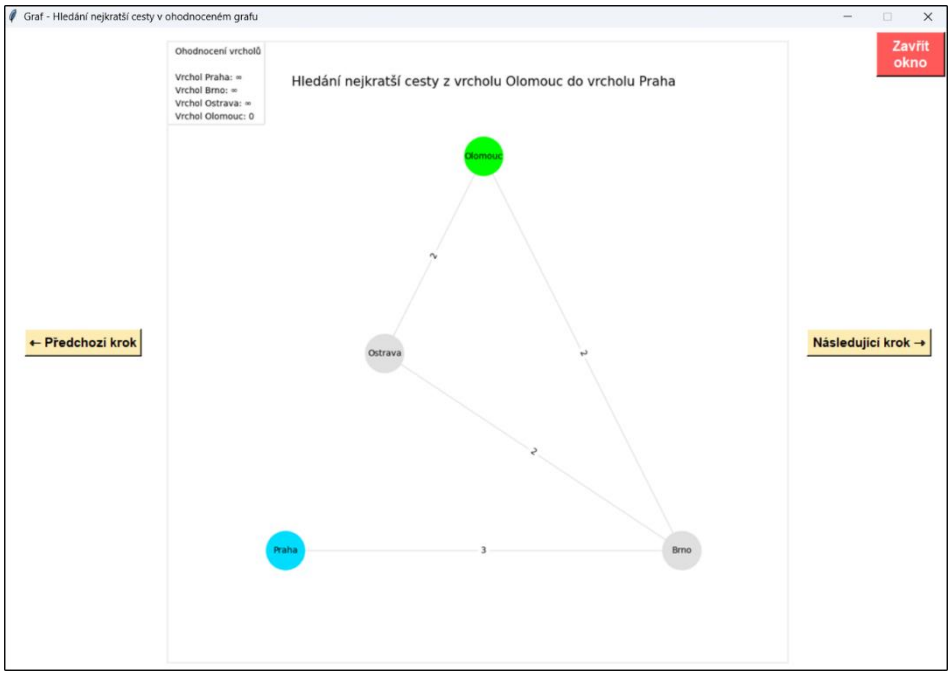
veškerých obrázků se vytvoří nové grafické uživatelské rozhraní, kde se dle našeho výběru zobrazí graf s hledáním či nalezením nejkratší cesty.



Obrázek 19: Výběr orientace a zobrazení grafu
Zdroj: Vlastní zpracování

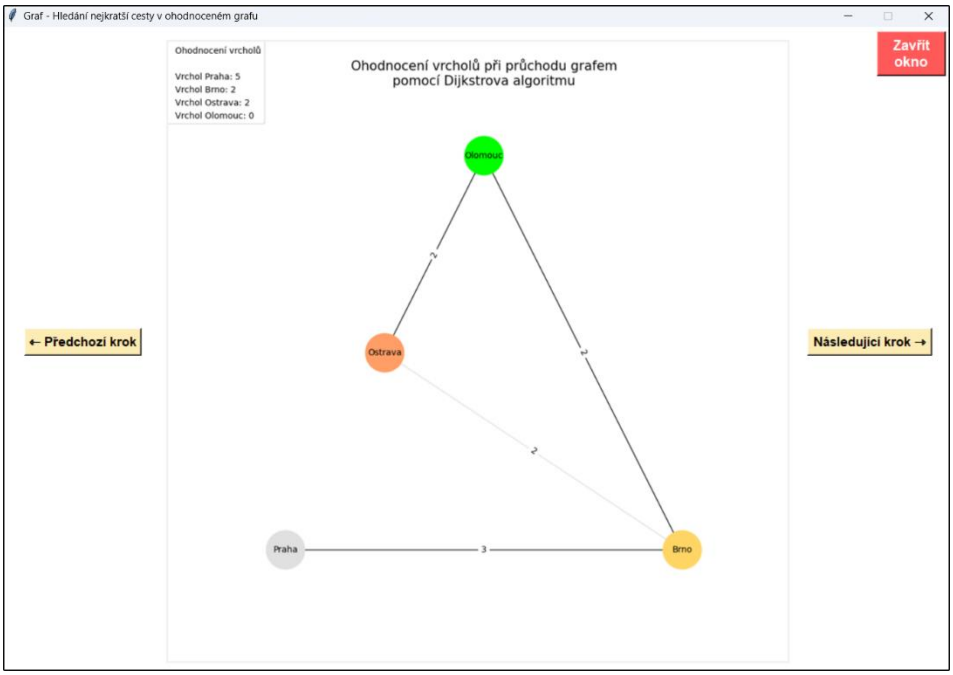
1.5 Grafické uživatelské rozhraní s grafem

Po výběru možnosti zobrazení postupu hledání nejkratší cesty v grafu krok po kroku a vytvoření nového grafického uživatelského rozhraní, si lze na první pohled povšimnout grafu a dvou postranních tlačítek. Postranní tlačítka nám budou složit k přechodu mezi jednotnými kroky (obrázky). V prvním kroku bude znázorněn úvodní obrázek, ve kterém budou zobrazeny veškeré vrcholy a námi definované hrany. Povšimněme si, že vrcholy jsou zobrazeny v určitých barvách a jednotlivé názvy vrcholů jsou vždy znázorněny uprostřed kruhu. Pokud je vrchol zobrazen v zelené barvě, tak se jedná o počáteční vrchol. Oproti tomu bude koncový vrchol znázorněn modrou barvou. Ostatní vrcholy i hrany budou zobrazeny v šedé barvě, jelikož jsme je doposud neprozkoumali. Jednotlivé vrcholy jsou ohodnoceny v přehledné tabulce vždy v levém horním rohu obrázku.



Obrázek 20: Úvodní obrázek
Zdroj: Vlastní zpracování

V dalších krocích budeme hledat nejkratší cestu z počátečního do koncového vrcholu pomocí Dijkstrova algoritmu. Při prvním průchodu cyklem Dijkstrova algoritmu se barva koncového vrcholu změní na šedou, jelikož jsme tento vrchol dosud neprozkoumali. Počátečnímu vrcholu zachováme barvu zelenou, abychom věděli, kde započalo hledání nejkratší cesty. V následujících krocích se hrany z šedé barvy zobrazí do černé. To znamená, že jsme danou hranu prozkoumali a nalezli kratší cestu do daného vrcholu. Po prozkoumání všech hran mezi počátečním vrcholem a ostatními vrcholy se cyklus opakuje. V dalších průchodech cyklu Dijkstrova algoritmu se bude odehrávat totéž. Nejprve se vybere vrchol s nejmenším ohodnocením, jež zobrazíme červenou barvou. Následně prozkoumáme veškeré hrany z vybraného vrcholu do ostatních vrcholů. Pokud jsme nalezli kratší cestu mezi danou dvojicí vrcholů, tak hranu zobrazíme černou barvou. V opačném případě zůstane hrana zobrazena v šedé barvě. Po prozkoumání všech hran mezi vybraným vrcholem a ostatními vrcholy se vybraný vrchol zobrazí v oranžové barvě a cyklus se opakuje.

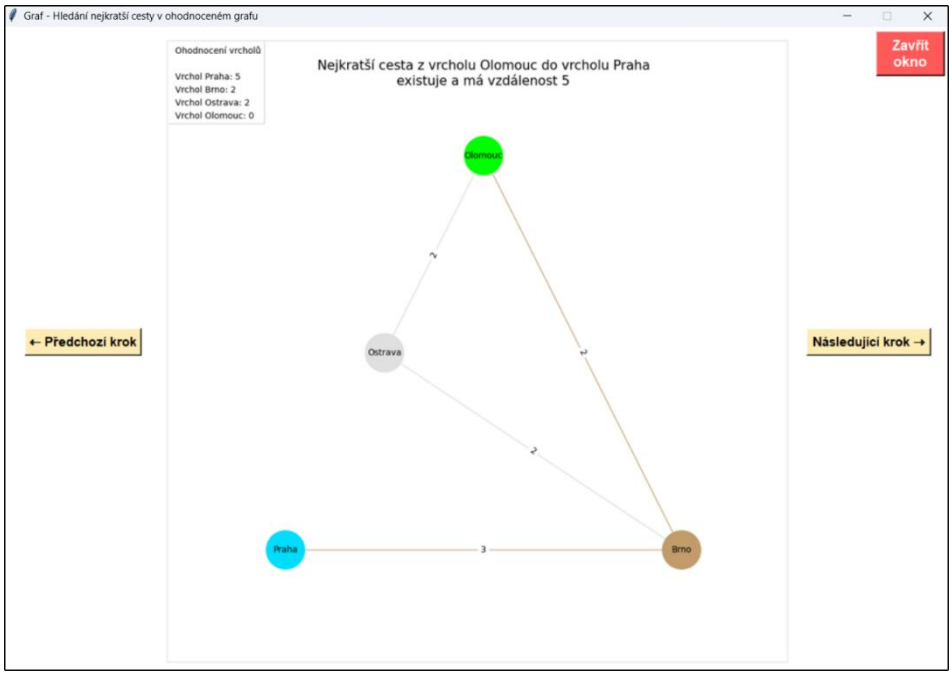


Obrázek 21: Cyklus Dijkstrova algoritmu
Zdroj: Vlastní zpracování

Po prozkoumání všech dosažitelných vrcholů z počátečního vrcholu Dijkstrův algoritmus končí a zobrazí se finální obrázek s nalezením nejkratší cesty mezi počátečním a koncovým vrcholem. Finální obrázek bude totožný s úvodním obrázkem s tím rozdílem, že vrcholy a hrany patřící do nejkratší cesty v grafu se vykreslí pomocí hnědé barvy, vyjímaje počáteční a koncový vrchol. Pro lepší orientaci v jednotlivých barvách na obrázcích uvedeme jejich vlastnosti do přehledné tabulky. Dále platí, že se zobrazí pouze finální obrázek, pokud bychom v poslední části aplikace zvolili možnost zobrazit nalezení nejkratší cesty v grafu.

Barva	**Vlastnosti**
Zelená	Znázorňuje počáteční vrchol
Modrá	Znázorňuje koncový vrchol
Šedá	Znázorňuje doposud neprozkoumané vrcholy a hrany
Černá	Znázorňuje prozkoumanou hranu, pomocí které jsme nalezli kratší cestu do daného vrcholu
Červená	Znázorňuje aktuálně prozkoumávaný vrchol
Oranžová	Znázorňuje již prozkoumaný vrchol
Hnědá	Znázorňuje vrcholy a hrany patřící do nejkratší cesty mezi počátečním a koncovým vrcholem

Tabulka 1: Vlastnosti barev

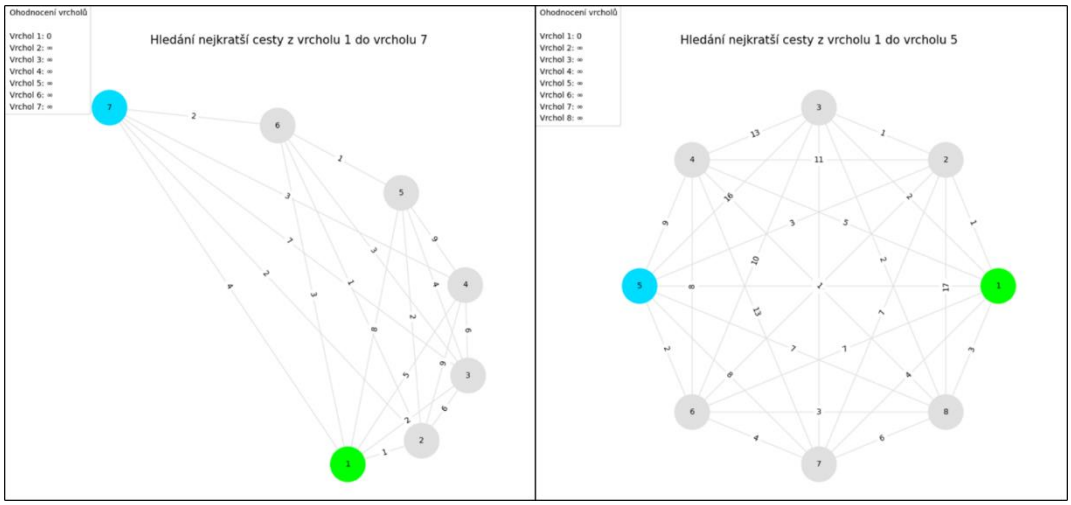


Obrázek 22: Nalezení nejkratší cesty mezi vrcholy Olomouc a Praha
Zdroj: Vlastní zpracování

Povšimněme si ještě textu v záhlaví každého obrázku. V úvodním obrázku se vyskytuje, mezi kterým počátečním a koncovým vrcholem vyžadujeme nalezení nejkratší cesty. V následujících obrázcích vždy v záhlaví uvádíme, že aplikace pro ohodnocení vrcholů a nalezení nejkratší cesty využívá Dijkstrův algoritmus. V záhlaví finálního obrázku se zobrazí, zdali jsme nalezli nejkratší cestu mezi počátečním a koncovým vrcholem. Pokud jsme nejkratší cestu nalezli, tak se zde vypíše i vzdálenost mezi počátečním a koncovým vrcholem.

Na závěr je důležité říci, že při tvorbě obrázků se aplikace rozhoduje, zda vytvoří planární, spirálový či kruhový graf. Ve spirálovém grafu jsou vrcholy uspořádané do spirály a v kruhovém grafu jsou vrcholy uspořádány do kruhu. Vše ovšem závisí na tom, kolik zvolíme vrcholů a jaké hrany v grafu definujeme. Aplikace se vždy bude snažit vytvořit planární graf, avšak některé grafy takto znázornit nelze. V takovém případě se aplikace rozhoduje na základě počet vrcholů, zdali vytvoří spirálový či kruhový graf. Pokud graf nelze znázornit planárně a obsahuje méně než osm vrcholů, tak se vytvoří spirálový graf, právě kvůli větší přehlednosti. V opačném případě se vytvoří kruhový graf. Ten však může být značně nepřehledný, či dokonce v některých situacích nečitelný. Problém nastává v tom, že takový graf nelze lepším způsobem znázornit do roviny.

Ovšem funkčnost aplikace a nelezení nejkratší cesty pomocí Dijkstrova algoritmu se nikterak nezmění, pouze bude pro nás náročné se v daném grafu zorientovat.



Obrázek 23: Příklad spirálového a kruhového grafu
Zdroj: Vlastní zpracování

2. Programovací jazyk a knihovny

V této kapitole si nejprve popíšeme, jaký programovací jazyk jsme si vybrali pro vytvoření aplikace, kterou jsme si podrobněji popsali v minulé kapitole. Dále zde uvedeme knihovny, které jsme použili pro tvorbu aplikace a popíšeme si, jak se s těmito knihovnami pracuje a jaké je jejich využití.

Tato a následující kapitola není důležitá pro uživatele, kteří chtějí aplikaci pouze využívat pro výuku či ověření výsledků hledání nejkratší cesty v ohodnocených grafech. Je však vhodná pro ty uživatele, kteří by chtěli rozvíjet mezipředmětové vztahy například mezi matematikou a informačními technologiemi anebo se zabývají programováním a chtěli by zjistit, jakým způsobem je aplikace vytvořena.

2.1 Vyšší programovací jazyk Python

Python se řadí mezi vysokoúrovňové programovací jazyky, jenž získal značnou oblibu u programátorů po celém světě. Svoji oblibu získal především kvůli jednoduchosti, čitelnosti a přehlednosti, což jak začínajícím, tak pokročilým programátorům výrazně usnadňuje učení se a použití tohoto programovacího jazyka. Python se řadí mezi interpretované jazyky. To znamená, že programátor nemusí vytvářet samostatný spustitelný soubor (například soubor typu EXE spustitelný v operačním systému Windows), ale může program spustit přímo ve vývojovém prostředí. Kód se tím pádem bude vykonávat řádek po řádku ve vývojovém prostředí.

Dále programovací jazyk Python disponuje vysokým množstvím různých knihoven a modulů. Programátoři po celém světě neustále vyvíjí a udržují knihovny, které je možné přímo stáhnout do vývojového prostředí z různých webových stránek. V souvislosti s těmito knihovnami Python umožňuje jednoduše vytvářet složitější programy či aplikace například pro strojové učení, analýzu dat nebo pro tvorbu výukových materiálů. Z těchto důvodů jsme si vybrali vyšší programovací jazyk Python pro tvorbu naší aplikace.

2.2 Knihovna Tkinter

Knihova Tkinter se využívá pro tvorbu grafického uživatelského rozhraní. Jinými slovy lze pomocí této knihovny vytvářet „okenní“ aplikace, ve kterých zobrazujeme

určité komponenty, pomocí kterých mohou uživatelé ovládat aplikaci nebo zapisovat svá data, která jsou následně zpracovávána. Za komponenty považujeme například různá tlačítka, textová a vstupní pole, rámečky nebo plátna, do kterých lze vkládat grafické objekty či obrázky.

Knihovna Tkinter je k dispozici po instalaci vývojového prostředí pro programovací jazyk Python. Abychom byli schopni vytvořit grafické uživatelské rozhraní, tak na začátku programu je důležité provést instanci okna, abychom byli dále schopni vytvářet a vykreslovat jednotlivé komponenty. Na konci programu je nezbytné použít příkaz ke spuštění smyčky, aby se grafické uživatelské rozhraní správně zobrazilo.

2.3 Knihovna NetworkX

Knihovna NetworkX se zaměřuje na vykreslení, analýzu a manipulaci s grafy. Jedná se nejčastěji programátory používanou knihovnu pro tvorbu a správu grafů. Pomocí NetworkX lze vizuálně vytvářet a zpracovávat například dopravní síť, internetovou síť nebo elektrickou rozvodnou síť. Knihovna taktéž umožňuje nalezení nejkratší cesty v ohodnoceném grafu, což je pro tuto práci zásadní.

Abychom však byli schopni využít knihovny NetworkX pro vytvoření naší aplikace, tak ji nejprve musíme nainstalovat do vývojového prostředí. Instalace knihovny je velmi jednoduchá, jelikož je na internetu dostupný návod na instalaci v několika krocích. Při práci s touto knihovnou je potřeba před tvorbou grafu provést instanci typu grafu. Knihovna nabízí volbu typu grafu mezi neorientovaným a orientovaným grafem. Následně knihovna vyžaduje, aby byly zadány vrcholy a hrany mezi těmito vrcholy. Po správném zadání vrcholů a hran je potřeba zvolit, jak se graf znázorní do roviny. V našem případě budeme volit mezi planárním, spirálovým nebo kruhovým grafem. Na závěr je důležité použít funkci pro vykreslení grafu, přičemž knihovna NetworkX disponuje několika funkcemi, které dokáží graf vykreslit v různých provedeních.

2.4 Knihovna Matplotlib

Matplotlib je velmi často používaná knihovna především pro tvorbu grafů či diagramů nebo pro vizualizaci určitých dat. Knihovna Matplotlib umožňuje vytvářet sloupcové grafy, bodové grafy, grafy matematických funkcí či histogramy. Dále nabízí

možnost přidání legendy či popisků, aby data v těchto grafech byla pro uživatele více srozumitelná.

V našem případě knihovnu Matplotlib využijeme k vytvoření obrázků jednotlivých kroků hledání nejkratší cesty v grafu, které se budou ukládat do počítače. Abychom toho docílili, tak potřebujeme nejprve definovat velikost obrázku. Následně je zapotřebí knihovně Matplotlib předat vytvořený graf pomocí knihovny NetworkX. Knihovna Matplotlib z daného grafu vytvoří obrázek a uloží jej na námi určené místo v počítači. Dále je potřeba říci, že za pomoci této knihovny budeme v každém obrázku znázorňovat legendu a popisek, který bude vždy zobrazen v záhlaví obrázku. Legenda bude sloužit pro znázornění všech vrcholů a jejich ohodnocení.

3. Implementace aplikace

V této kapitole si popíšeme kroky při implementaci naší aplikace, kterou jsme si blíže popsali v první kapitole praktické části této práce. Postupně se zaměříme na jednotlivé naimplementované funkce, které zajišťují správný chod aplikace.

Jak již bylo řečeno v úvodu 2. kapitoly praktické části této práce, tak i tato kapitola je vhodná zejména pro uživatele, kteří by chtěli rozvíjet mezipředmětové vztahy anebo se zabývají programováním a chtěli by zjistit, jakým způsobem je aplikace vytvořena.

Abychom však aplikaci dokázali vytvořit, tak je potřeba nejprve naistalovat vývojové prostředí a knihovny, se kterými bude dále pracovat. Nejčastěji budeme pracovat s knihovnami, které jsou uvedené v minulé kapitole. Nicméně budeme využívat také knihovny pro zjištění kořenového adresáře aplikace, abychom zabezpečili správnost ukládání a načítaní obrázků. Dále je potřeba vytvořit globální proměnné, se kterými bude aplikace pracovat v naimplementovaných funkcích. Posledním krokem je instance a zacyklení grafického uživatelského rozhraní se všemi potřebnými parametry, jako je například pojmenování či nastavení rozměrů okna.

3.1 Funkce pro tvorbu a pojmenování vrcholů

Po naimportování knihoven, vytvoření globálních proměnných a instanci grafického uživatelského rozhraní se zavolá funkce *Create_nodes_window*, která vytvoří komponenty v grafickém uživatelském rozhraní pro pojmenování a zadání počtu vrcholů. Správné zapsání počtu vrcholů zabezpečuje funkce *Window_update*. Tato funkce kontroluje, zda uživatel zapisuje do vstupního pole pouze číslice a maximální požadované množství číslic. Funkce se poprvé zavolá po provedení všech příkazů ve funkci *Create_nodes_window* a vždy po dvaceti milisekundách se bude opakovat. Po potvrzení uživatelem zadaných parametrů se přejde na funkci *Check_nodes_option*.

Funkce *Check_nodes_option* zkontroluje parametry, které uživatel zadal. Především se zkontroluje, zda uživatel zapsal počet vrcholů v požadovaném rozmezí. Pokud se tak nestalo, tak je vypsáno chybové hlášení a uživatel je upozorněn na to, aby počet vrcholů zapsal v požadovaném rozmezí. Dále se zkontroluje, jakou možnost uživatel zvolil pro pojmenování vrcholů. Pokud uživatel zvolil možnost pojmenovat vrcholy od jedné do celkového počtu vrcholů, tak se zavolá funkce *Create_edges_window*

a začnou se vykonávat její příkazy. V případě, že si uživatel zvolil možnost pojmenovat vrcholy vlastními názvy, tak se v grafickém uživatelském rozhraní vytvoří sada vstupních polí. Množství vstupních polí je závislé na počtu zadaných vrcholů. Uživatel může pojmenovat vrcholy libovolnými znaky v maximálním požadovaném rozmezí deseti znaků. Funkce *Window_update* průběžně kontroluje znaky a množství znaků zapsaných v těchto vstupních polích. Při potvrzení názvů se zavolá funkce *Check_nodes_name*.

Funkce *Check_nodes_name* zkontroluje, zda bylo každé vstupní pole vyplněno neboli, zda uživatel každému vrcholu přiřadil vlastní název. Dále se kontroluje, zdali se dva názvy u dvou různých vrcholů shodují. Pokud tato situace nastane, tak aplikace upozorní uživatele chybovým hlášením, aby příslušné vrcholy přejmenoval. V opačném případě, kdy je vše v pořádku, se zavolá funkce *Create_edges_window*.

3.2 Funkce pro tvorbu a ohodnocení hran

Po správném pojmenování a zadání počtu vrcholů se vždy zavolá funkce *Create_edges_window*. Tato funkce v prvním kroku odebere stávající komponenty a vytvoří nové komponent v grafickém uživatelském rozhraní pro tvorbu a ohodnocení hran. Nejdůležitější komponentou je posuvná tabulka, kde se nacházejí veškeré kombinace hran mezi dvojicemi různých vrcholů a k nim přiřazená vstupní pole. Funkce *Create_edges_window* zajistí pravidelné zapisování hodnot hran, které jsou obsažené v jednotlivých vstupních polích, do globální proměnné typu pole. Po ohodnocení všech uživatelem požadovaných hran se zavolá funkce *Choose_nodes_window*.

Funkce *Window_update* pravidelně kontroluje, aby uživatel vyplňoval vstupní pole dle zadaných požadavků. Do vstupních polí lze zadat pouze číslice o celkovém počtu sedmi číslic. Funkce dále kontroluje, zda uživatel vyplnil vstupní pole mezi danou dvojicí vrcholů. Pokud tato situace nastane, tak se vstupní pole u stejné dvojice vrcholů, akorát v opačném pořadí, zatmaví a uzamkne. V případě, že je vstupní pole uzamknuto, tak do něj nelze nic zapsat.

Uživatel má taktéž možnost přejít z aktuální části aplikace do části předchozí. Tento přechod zajišťuje funkce *Back_from_edges_window*, která odebere veškeré komponenty z grafického uživatelského rozhraní a globálním proměnným potřebným v aktuální části nastaví výchozí hodnoty. Ovšem zachová hodnoty globálních

proměnných používaných v předchozí části aplikace. Následně je zavolána funkce *Create_nodes_window*.

3.3 Funkce pro výběr počátečního a koncového vrcholu

Po ohodnocení hran se zavolá funkce *Choose_nodes_window*, která jako v minulé části odebere všechny komponenty v grafickém uživatelském rozhraní a vytvoří nové komponenty pro výběr počátečního a koncového vrcholu. V levé části okenní aplikace se budou nacházet tlačítka pro výběr počátečního vrcholu a v pravé části tlačítka pro výběr koncového vrcholu. U každého tlačítka bude zobrazeno jméno daného vrcholu. Po volbě počátečního a koncového vrcholu se zavolá funkce *Create_set_graph_window*.

Taktéž je možné přejít z aktuální části aplikace do části předchozí, což zajišťuje funkce *Back_from_choose_nodes_window*. Funkce odebere veškeré komponenty v grafickém uživatelském rozhraní a nastaví globální proměnné pro počáteční a koncový vrchol na nulovou hodnotu. Následně se zavolá funkce *Create_edges_window*.

3.4 Funkce pro výběr orientace a zobrazení grafu

Po zvolení počátečního a koncového vrcholu se zavolá funkce *Create_set_graph_window*. Tato funkce odebere veškeré komponenty v grafickém uživatelském rozhraní a vytvoří nové komponenty pro volbu orientace grafu a způsob zobrazení nejkratší cesty v grafu. Uživatel má možnost si zvolit, zdali se bude hledat nejkratší cesta v ohodnoceném neorientovaném či orientovaném grafu. Následně funkce umožňuje výběr pro zobrazení nejkratší cesty v grafu ze dvou možností. Lze zobrazit postup hledání nejkratší cesty v grafu krok za krokem anebo zobrazit nalezení nejkratší cesty v grafu. V obou možnostech se zavolá funkce *Create_new_window*.

Uživatel může jako v minulých částech aplikace přejít z aktuální části do části předchozí. Přechod se uskuteční zavoláním funkce *Choose_nodes_window*. Je zde však také možnost přejít zpět do první části aplikace. Pokud by uživatel zvolil tuto možnost, tak se zavolá funkce *Start_again*. Funkce odebere veškeré komponenty z grafického uživatelského rozhraní a nastaví všem globálním proměnným výchozí hodnoty. Následně se zavolá funkce *Create_nodes_window*, která je vždy jako první volána při spuštění aplikace.

3.5 Funkce pro hledání nejkratší cesty a tvorbu obrázků

Po zvolení orientace a vykreslení grafu se zavolá funkce *Create_new_window*. Funkce nejprve odstraní všechny soubory a adresáře, které se nacházejí v adresáři *Images*. V adresáři *Images* se vždy totiž budou vyskytovat obrázky s jednotlivými kroky hledání nejkratší cesty v grafu. Po odstranění všech souborů se zavolá funkce *Create_array_of_edges*, která vytvoří nové dvojrozměrné pole (váženou matici sousednosti) v závislosti na volbě orientace grafu s využitím globální proměnné obsahující hodnoty s ohodnocením hran. Následně je zavolána funkce *Dijkstra_shortest_path*, která bude hledat nejkratší cestu z počátečního do koncového vrcholu v grafu.

Funkce *Create_new_window* provede instanci a zacyklení nového grafického uživatelského rozhraní po dokončení funkce *Dijkstra_shortest_path*. Následně zde funkce vytvoří komponenty pro zobrazení grafu. Pokud uživatel zvolil možnost zobrazit postup hledání nejkratší cesty v grafu krok po kroku, tak se vytvoří i komponenty pro přechod mezi jednotlivými kroky. Přechody mezi jednotlivými kroky zajišťují funkce *Previous_image* a *Next_image*. Funkce dále zajišťuje, že teprve poté, až uživatel ukončí nové grafické uživatelské rozhraní, tak může vytvářet nové grafy.

3.5.1 Funkce Dijkstra_shortest_path

Funkce *Dijkstra_shortest_path* bude sloužit ke hledání nejkratší cesty v ohodnoceném grafu mezi počátečním a koncovým vrcholem. Implementace je provedena dle kapitoly 5.1 v teoretické části této práce. Na začátku funkce se vytvoří lokální proměnné, se kterými bude funkce dále pracovat. Poté se zavolá funkce *Generate_graph*, která vytvoří úvodní graf a uloží jej ve formě obrázku do adresáře *Images*.

V následujících krocích se bude cyklicky provádět tentýž postup. Nejprve se vybere vrchol s nejmenším ohodnocením. Pokud je tato hodnota menší než maximální hodnota, kterou nabízí operační systém, tak se zavolá funkce *Generate_graph*. Funkce vytvoří graf, kde barevně vyznačí, se kterým vrcholem bude Dijkstrův algoritmus aktuálně pracovat, a uloží jej ve formě obrázku. V opačném případě Dijkstrův algoritmus končí, jelikož jsme nenalezli žádný další dosažitelný vrchol z počátečního vrcholu.

V dalším kroku cyklu se budou prozkoumávat hrany mezi zvoleným vrcholem a ostatními vrcholy. V případě, že se nalezne kratší cesta mezi danou dvojicí vrcholů, tak se opětovně zavolá funkce *Generate_graph*. Funkce v tomto případě v grafu zvýrazní hranu mezi danou dvojicí vrcholů. Graf se následně uloží do adresáře *Images* ve formě obrázku.

Po skončení Dijkstrova algoritmu a nalezení nejkratší cesty mezi počátečním a koncovým vrcholem se zavolá funkce *Generate_graph*, která v grafu vyznačí počáteční, koncový vrchol a nejkratší nalezenou cestu mezi nimi. Graf je poté uložen ve formě obrázku.

3.5.2 Funkce Generate_graph

Funkce *Generate_graph* zajišťuje vytvoření grafu v aktuálním kroku funkce *Dijkstra_shortest_path*. Graf je následně převeden do formátu obrázku a uložen do adresáře *Images*. Na začátku funkce se provede tvorba lokálních proměnných, které budou sloužit pro vytvoření grafu a tvorbu obrázku. V následujícím kroku se vytvoří legenda obrázku obsahující vrcholy a jejich ohodnocení.

Při tvorbě grafu se nejprve zvolí neorientovaný či orientovaný graf na základně uživatelem dříve zvolené možnosti. Poté se vytvoří jednotlivé vrcholy, kde každému vrcholu bude přiřazen daný název a barva, která slouží k rozeznání vrcholů v jednotlivých krocích Dijkstrova algoritmu. Po vytvoření všech vrcholů se následně vytvoří hrany mezi danými vrcholy. Jednotlivým hranám se přiřadí ohodnocení a zvýraznění, pokud byla s její pomocí nalezena kratší cesta mezi danou dvojicí vrcholů. Po skončení Dijkstrova algoritmu se vytvoří výsledný graf s nalezením nejkratší cesty mezi počátečním a koncovým vrcholem. V případě, že koncový vrchol je dosažitelný z počátečního vrcholu, tak se jednotlivým vrcholům a hranám patřící do nejkratší cesty přiřadí specifická barva. V opačném případě se tak nestane.

Po vytvoření daného grafu se zvolí jeho znázornění do roviny. Pokud lze graf znázornit tak, že se hrany nebudou překrývat, tak se vytvoří planární graf. V opačném případě se vytvoří spirálový graf, ale pouze za podmínky, že uživatel zadal méně než osm vrcholů. Pokud graf nelze znázornit planárně a počet vrcholů je větší rovno osmi, tak se vytvoří kruhový graf.

Na konci funkce se vytvořený graf převede na formát obrázku. V obrázku se bude nacházet daný krok Dijkstrova algoritmu, legenda s vrcholy a jejich ohodnocením a popisek obrázku. Obrázek se následně uloží do adresáře *Images* a graf nacházející se v lokální proměnné je vymazán.

4. Vzorové příklady

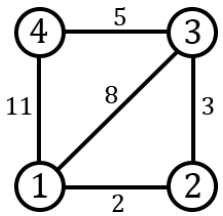
V této kapitole se budeme zabývat řešením vzorových příkladů s využitím aplikace pro kontrolu výsledků. V zadání každého příkladu bude popsána množina vrcholů a hran a sestavená vážená matice sousednosti s jejíž pomocí vytvoříme ohodnocený graf. V grafu zvolíme počáteční a koncový vrchol, mezi nimiž nalezneme všechny možné cesty, které přehledně zapíšeme do tabulky. U všech těchto cest určíme jejich délku, přičemž řešením příkladu bude cesta s nejmenší délkou. S využitím vytvořené aplikace si ověříme, zdali jsme postupovali při hledání nejkratší cesty v ohodnoceném grafu správně.

4.1 Příklad 1

Mějme dán kladně ohodnocený graf $G = (V, E)$, kde $V = \{1, 2, 3, 4\}$ a $E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{3, 4\}\}$. Vážená matice sousednosti bude ve tvaru

$$W_G = \begin{pmatrix} 0 & 2 & 8 & 11 \\ 2 & 0 & 3 & 0 \\ 8 & 3 & 0 & 5 \\ 11 & 0 & 5 & 0 \end{pmatrix}.$$

Nyní si pojďme ukázat, jak daný graf bude vypadat po znázornění do roviny.



Obrázek 24: Příklad 1 - Graf G znázorněný do roviny
Zdroj: Vlastní zpracování

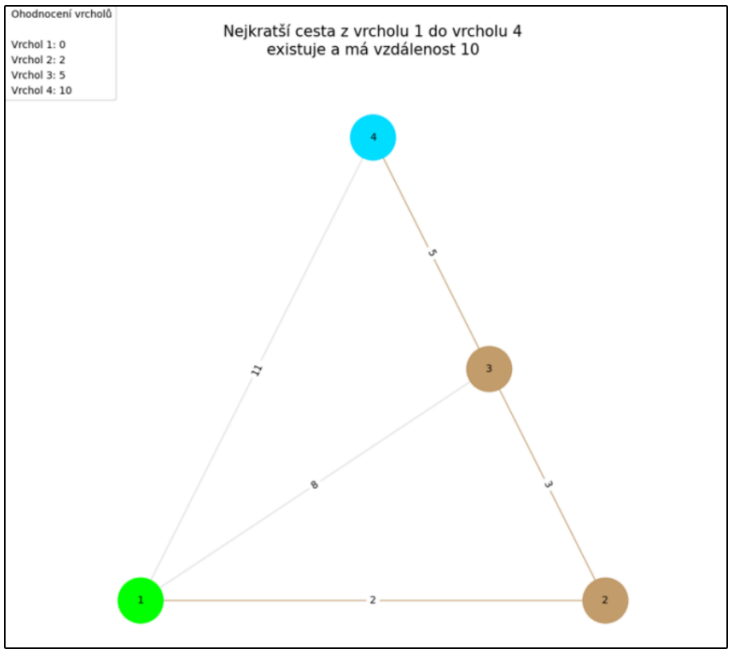
Na první pohled si lze povšimnout, že graf do roviny lze znázornit planárně a všechny vrcholy jsou navzájem dosažitelné. Aby však bylo možné v grafu hledat nejkratší cestu mezi počátečním a koncovým vrcholem, je potřeba tyto vrcholy určit. Počáteční vrchol $u = 1$ a koncový vrchol $v = 4$. S využitím tabulky určíme všechny možné cesty mezi počátečním a koncovým vrchole, z nichž vybereme tu nejkratší.

Cesta mezi vrcholy u a v	**Délka ohodnoceného sledu**
$P_1 = (u, v)$-cesta $= \{[1, 4]\}$	$d_G^w(P_1) = 11$
$P_2 = (u, v)$-cesta $= \{[1, 3, 4]\}$	$d_G^w(P_2) = 13$

$P_3 = (u, v)$-cesta $= \{[1, 2, 3, 4]\}$	$d_G^w(P_3) = 10$

Tabulka 2: Příklad 1 - Sestavení cest mezi počátečním a koncovým vrcholem

Po prozkoumání grafu a sestavení tabulky lze určit, že nejkratší cesta z vrcholu 1 do vrcholu 4 povede přes všechny zbývající vrcholy, tudíž (u, v)-cesta $= \{[1, 2, 3, 4]\}$. Z toho vyplývá, že vážená vzdálenost $dist_G^w(u, v) = 10$. Nyní si pojďme ověřit správnost našeho postupu pomocí vytvořené aplikace.



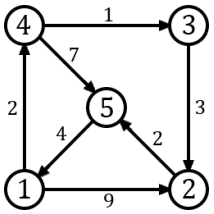
Obrázek 25: Příklad 1 - Ověření správnosti výsledku pomocí aplikace
Zdroj: Vlastní zpracování

4.2 Příklad 2

Mějme dán kladně ohodnocený orientovaný graf $G = (V, E)$, kde množina vrcholů $V = \{1, 2, 3, 4, 5\}$ a množina hran $E = \{[1, 2], [1, 4], [2, 5], [3, 2], [4, 3], [4, 5]\}$. Vážená matice sousednosti bude u orientovaného grafu G ve tvaru

$$W_G = \begin{pmatrix} 0 & 9 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 7 \\ 4 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Nyní si pojďme ukázat, jak daný graf bude vypadat po znázornění do roviny.



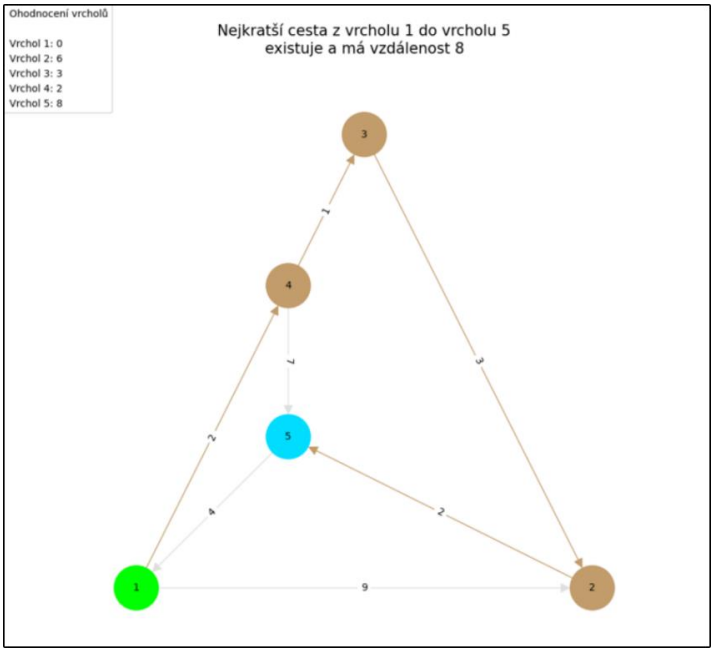
Obrázek 26: Příklad 2 - Graf G znázorněný do roviny
Zdroj: Vlastní zpracování

Jako v minulém příkladě si lze na první pohled povšimnout, že všechny vrcholy jsou navzájem dosažitelné a graf lze znázornit planárně. Stanovme počáteční a koncový vrchol pro hledání nejkratší cesty v grafu. Počáteční vrchol $u = 1$ a koncový vrchol $v = 5$. S využitím tabulky určíme všechny možné cesty mezi počátečním a koncovým vrchole, z nichž vybereme tu nejkratší.

Cesta mezi vrcholy u a v	**Délka ohodnoceného sledu**
$P_1 = (u, v)$-cesta $= \{[1, 2, 5]\}$	$d_G^w(P_1) = 11$
$P_2 = (u, v)$-cesta $= \{[1, 4, 5]\}$	$d_G^w(P_2) = 9$
$P_3 = (u, v)$-cesta $= \{[1, 4, 3, 2, 5]\}$	$d_G^w(P_3) = 8$

Tabulka 3: Příklad 2 - Sestavení cest mezi počátečním a koncovým vrcholem

Po úvaze nad zadaným příkladem a sestavení tabulky zjistíme, že nejkratší cesta mezi počátečním a koncovým vrcholem bude obsahovat všechny vrcholy. Nejkratší nalezená cesta bude ve tvaru (u, v)-cesta $= \{[1, 4, 3, 2, 5]\}$, kde vážená vzdálenost $dist_G^w(u, v) = 8$. Pokusme se o ověření našeho postupu pomocí vytvořené aplikace.



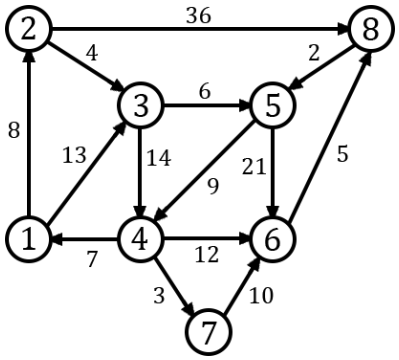
Obrázek 27: Příklad 2 - Ověření správnosti výsledku pomocí aplikace
Zdroj: Vlastní zpracování

4.3 Příklad 3

Mějme dán kladně ohodnocený orientovaný graf $G = (V, E)$, kde množina vrcholů $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ a množina hran $E = \{[1, 2], [1, 3], [2, 3], [2, 8], [3, 4], [3, 5], [4, 1], [4, 6], [4, 7], [5, 4], [5, 6], [6, 8], [7, 6], [8, 5]\}$. Vážená matice sousednosti bude u orientovaného grafu G ve tvaru

$$W_G = \begin{pmatrix} 0 & 8 & 13 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 36 \\ 0 & 0 & 0 & 14 & 6 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 12 & 3 & 0 \\ 0 & 0 & 0 & 9 & 0 & 21 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{pmatrix}.$$

Nyní si pojďme ukázat, jak daný graf bude vypadat po znázornění do roviny.



Obrázek 28: Příklad 3 - Graf G znázorněný do roviny
Zdroj: Vlastní zpracování

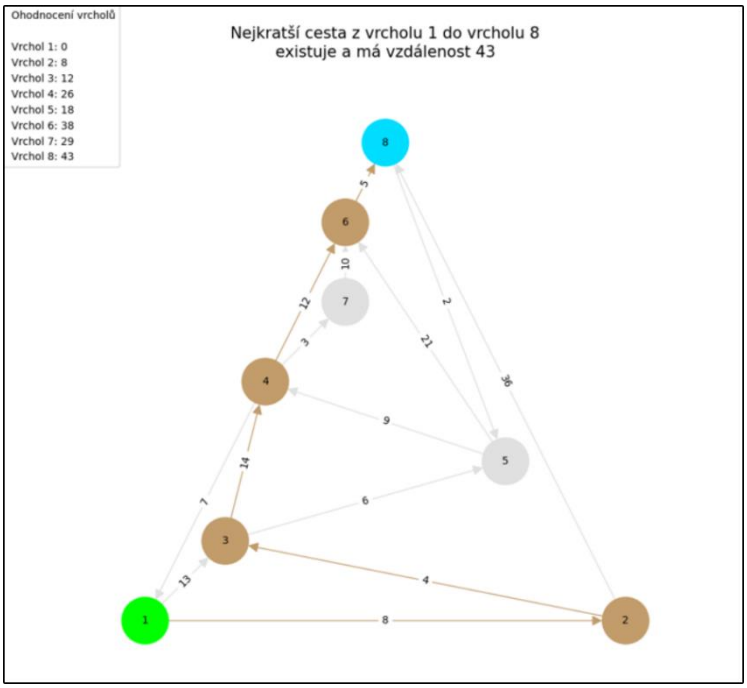
Tento příklad je již poměrně náročnější než předchozí příklady. Pokud však graf znázorníme do roviny, jak můžeme vidět na obrázku výše, tak všechny vrcholy jsou navzájem dosažitelné a graf lze znázornit planárně. Stanovme počáteční a koncový vrchol pro hledání nejkratší cesty v grafu. Počáteční vrchol $u = 1$ a koncový vrchol $v = 8$. S využitím tabulky určíme všechny možné cesty mezi počátečním a koncovým vrcholem, z nichž vybereme tu nejkratší.

Cesta mezi vrcholy u a v	**Délka ohodnoceného sledu**
$P_1 = (u, v)$-cesta $= \{[1, 2, 8]\}$	$d_G^w(P_1) = 44$
$P_2 = (u, v)$-cesta $= \{[1, 2, 3, 4, 6, 8]\}$	$d_G^w(P_2) = 43$
$P_3 = (u, v)$-cesta $= \{[1, 2, 3, 4, 7, 6, 8]\}$	$d_G^w(P_3) = 44$
$P_4 = (u, v)$-cesta $= \{[1, 2, 3, 5, 6, 8]\}$	$d_G^w(P_4) = 44$

$P_5 = (u,v)$-cesta $= \{[1,2,3,5,4,6,8]\}$	$d_G^w(P_5) = 44$
$P_6 = (u,v)$-cesta $= \{[1,2,3,5,4,7,6,8]\}$	$d_G^w(P_6) = 45$
$P_7 = (u,v)$-cesta $= \{[1,3,4,6,8]\}$	$d_G^w(P_7) = 44$
$P_8 = (u,v)$-cesta $= \{[1,3,4,7,6,8]\}$	$d_G^w(P_8) = 45$
$P_9 = (u,v)$-cesta $= \{[1,3,5,6,8]\}$	$d_G^w(P_9) = 45$
$P_{10} = (u,v)$-cesta $= \{[1,3,5,4,6,8]\}$	$d_G^w(P_{10}) = 45$
$P_{11} = (u,v)$-cesta $= \{[1,3,5,4,7,6,8]\}$	$d_G^w(P_{11}) = 46$

Tabulka 4: Příklad 3 - Sestavení cest mezi počátečním a koncovým vrcholem

Po určitém čase při hledání všech možných cest mezi počátečním a koncovým vrcholem dojdeme k závěru, že nejkratší cesta bude ve tvaru (u,v)-cesta $=$ $= \{[1,2,3,4,6,8]\}$, kde vážená vzdálenost $dist_G^w(u,v) = 43$. Pokusme se o ověření našeho postupu pomocí vytvořené aplikace.



Obrázek 29: Příklad 3 - Ověření správnosti výsledku pomocí aplikace
Zdroj: Vlastní zpracování

Závěr

V teoretické části práce byly shrnuty poznatky z oblasti teorie grafů, především pojmy související s hledáním nejkratší cesty v ohodnocených grafech. Dále zde byly vysvětleny rozdíly mezi nejčastěji používanými algoritmy pro hledání nejkratší cesty. Podrobněji byl však popsán Dijkstrův algoritmus, který je využíván ve vytvořené aplikaci. V praktické části práce bylo detailně popsáno ovládání a práce s aplikací, včetně popisu jednotlivých funkcí aplikace implementovaných pomocí vyššího programovacího jazyka Python a knihoven Tkinter, NetworkX a Matplotlib. V závěru praktické části bylo uvedeno několik vzorových příkladů, kde bylo za úkol nalézt nejkratší cestu v kladně ohodnocených grafech a jednotlivá řešení byla ověřena za pomoci vytvořené aplikace.

Cílem práce byla implementace aplikace s využitím vyššího programovacího jazyka Python a knihoven Tkinter a NetworkX. Při implementaci aplikace měly být využity vybrané algoritmy pro hledání nejkratší cesty v ohodnocených grafech, přičemž byl využit Dijkstrův algoritmus. Všech výše uvedených cílů bylo dosaženo.

Aplikace umožňuje vytvářet vlastní grafy, vybrat počáteční vrchol, koncový vrchol a způsob orientace grafu. Při hledání nejkratší cesty v grafu se vždy využije Dijkstrův algoritmus. Ať už po volbě zobrazení postupu hledání nejkratší cesty, nebo zobrazení výsledku nalezení nejkratší cesty, se vždy graf zobrazí v grafickém uživatelském rozhraní v podobě obrázku. Aplikace může sloužit jakožto edukační prostředek pro výuku na školách, kde by byly rozvíjeny mezipředmětové vztahy například mezi matematikou a informačními technologiemi.

Seznam literatury

AVANESYAN, Galina, 2010. *Historie teorie grafů*. Bakalářská práce. Praha: Vysoká škola ekonomická v Praze. Dostupné z: https://theses.cz/id/k07z67/. [citováno 2023-11-13]

ČADA, Roman; KAISER, Tomáš a RYJÁČEK, Zdeněk, 2004. *Diskrétní matematika*. Online. Plzeň: Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Dostupné z: https://hosting.pilsfree.net/tonny/zcu/FAV/KMA/KMA.TGD1/DMA%20%20scripta.pdf. [citováno 2023-11-13]

ČERNÝ, Jakub, 2010. *Základní grafové algoritmy*. Online. Praha: Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra aplikované matematiky. Dostupné z: https://docplayer.cz/4802558-Zakladni-grafove-algoritmy.html. [citováno 2023-11-13]

DEMEL, Jiří, 2002. *Grafy a jejich aplikace*. Praha: Academia. ISBN 978-80-200-0990-6.

DIESTEL, Reinhard, 2017. *Graph Theory*. Fifth Edition. Hamburg: Springer. ISBN 978-3-662-53621-6.

ERCIYES, Kayhan, 2021. *Algebraic Graph Algorithms*. First Edition. Istambul: Springer. ISBN 978-3-030-87885-6.

HAGBERG, Aric; SCHULT, Dan a SWART, Pieter, 2018. *NetworkX Reference, Release 2.2*. Online. Dostupné z: https://networkx.org/documentation/networkx2.2/_downloads/networkx_reference.pdf. [citováno 2023-11-13]

HAVRÁNEK, Tomáš, 2010. *Vizualizace hledání nejkratší cesty na grafech*. Diplomová práce. Pardubice: Univerzita Pardubice, Fakulta elektrotechniky a informatiky. Dostupné z: https://theses.cz/id/tp0rsn/. [citováno 2023-11-13]

HLINĚNÝ, Petr, 2005. *Diskrétní matematika*. Online. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky. Dostupné z: https://is.muni.cz/el/fi/podzim2015/IB000/um/jine/DIM-distext05.pdf. [citováno 2023-11-13]

HLINĚNÝ, Petr, 2010. *Základy teorie grafů*. Online. Brno: Masarykova univerzita, Fakulta informatiky. Dostupné z: https://is.muni.cz/do/1499/el/estud/fi/js10/grafy/Grafy-text10.pdf. [citováno 2023-11-13]

HOLZER, Raphael, 2020. *Tk tutorial Documentation, Release 2020*. Online. Dostupné z: https://tk-tutorial.readthedocs.io/_/downloads/en/latest/pdf/. [citováno 2023-11-13]

HUNTER, John; DALE, Darren; FIRING, Eric a DROETTBOOM, Michael, 2018. *Matplotlib, Release 2.1.2*. Online. Dostupné z: https://matplotlib.org/2.1.2/Matplotlib.pdf. [citováno 2023-11-13]

KOVÁŘ, Petr, 2014. *Úvod do teorie grafů*. Online. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava. Dostupné z: https://homel.vsb.cz/~kov16/files/uvod_do_teorie_grafu.pdf. [citováno 2023-11-13]

KOVÁŘ, Petr, 2023. *Teorie grafů*. Online. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava. Dostupné z: https://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_tisk.pdf. [citováno 2023-11-13]

MAŠTEROVÁ, Ludmila, 2016. *Teorie grafů*. Bakalářská práce. Brno: Masarykova univerzita, Přírodovědecká fakulta. Dostupné z: https://theses.cz/id/tgv4e7/. [citováno 2023-11-13]

MATOUŠEK, Jiří a NEŠETŘIL, Jaroslav, 2009. *Kapitoly z diskrétní matematiky*. 4. vydání. Praha: Karolinum. ISBN 978-80-246-1740-4.

ROSSUM VAN, Guido, 2018. *Python Tutorial, Release 3.7.0*. Online. Dostupné z: https://bugs.python.org/file47781/Tutorial_EDIT.pdf. [citováno 2023-11-13]

SEDLÁČEK, Jiří, 1977. *Úvod do teorie grafů*. 2. vydání. Praha: Academia. ISBN 510-21-826.

SKOPALOVÁ, Lucie, 2023. *Teorie grafů na základních školách*. Bakalářská práce. Ostrava: Ostravská univerzita, Pedagogická fakulta. Dostupné z: https://theses.cz/id/77iwhf/. [citováno 2023-11-13]

STRACHOTA, Pavel, 2006. *Základy Teorie Grafů*. Online. Praha: České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská. Dostupné z: https://saint-paul.fjfi.cvut.cz/base/sites/default/files/ZTG/ztg.pdf. [citováno 2023-11-13]

ŠEDA, Miloš, 2003. *Teorie grafů*. Online. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství. Dostupné z: https://www.euroekonom.sk/download2/materialy-vs-informatika/Teorie-Grafu.pdf. [citováno 2023-11-13]

ŠIŠMA, Pavel, 1997. *Teorie grafů 1736-1963*. 8. svazek. Praha: Prometheus. ISBN 80-7196-065-9.

RAHMAN, Saidur, 2017. *Basic Graph Theory*. Dhaka: Springer. ISBN 978-3-319-49474-6.

Seznam obrázků

Seznam tabulek