Úlohy dynamického programování

Richard Tomšů

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Richard Tomšů**
Osobní číslo:	**A21281**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Úlohy dynamického programování**

Zásady pro vypracování

1. Nastudujte metody a klasifikaci úloh dynamického programování.
2. Analyzujte metodiku a postup Dijkstrova algoritmu a jeho modifikací.
3. Vypracujte metodiku a programovou realizaci pro separovatelné reálné funkce.
4. Proveďte programovou realizaci metod pro vyhledávání extremální cesty v grafu.
5. Napište studijní oporu a vypracujte ilustrativní příklady.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. NOŽIČKA František. *Dynamické programování I, Diskrétní dynamické programování*, Státní pedagogické nakladatelství, Praha 1977, 1. vydání.
2. SAMEK Jaroslav, NEČASOVÁ Zdeňka, KODERA Jan. *Dynamické programování v ekonomických procesech*, Státní pedagogické nakladatelství, Praha 1983, 1. vydání.
3. TÖPPFER, Pavel. *Algoritmy a programovací techniky*. 1. vyd. Praha: Prometheus, 1995. 299 s. ISBN 80-85849-83-6. S. 250–268.
4. GAJDA Bohumil. *Speciální metody optimalizace*. Diplomová práce, FAI UTB, 2009.
5. ČERNÝ Jakub. *Nejkratší cesta v grafu, in: Základní grafové algoritmy,* https://cs.wikipedia.org/wiki/Portable_Document_Format.
6. MATOUŠEK Jiří, NEŠETŘIL Jaroslav. *Kapitoly z diskrétní matematiky*. Karolinum Praha, 2007.
7. ŠEDA Miloš. Teorie grafů. VUT FSI Brno, listopad 2003.

Vedoucí bakalářské práce: **prof. Ing. Roman Prokop, CSc.**
Ústav matematiky

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 7. května 2024

Richard Tomšů, v. r.
……
podpis studenta

ABSTRAKT

Práce se zabývá analýzou a využitím vyhledávácích algoritmů v třídě úloh dynamického programování. Dynamické programování je disciplínou, která se zabývá řešením problémů, které jsou charakterizovány separovatelnou účelovou funkcí. Tyto může být zadány analyticky nebo formou orientovaného nebo neorientovaného grafu. Práce se soustřeďuje Dijsktrův algoritmus a jeho modifikace. Výsledkem práce je programová realizace v kódu Python a je uvedeno několik ilustrativních příkladů.

Klíčová slova: separovatelná funkce, orientovaný graf, neorientovaný graf, dynamické programování, Dijkstrův algoritmus, Bellman-Fordův algoritmus, Floyd-Warshallův algoritmus

ABSTRACT

The thesis deals with the analysis and use of search algorithms in a class of dynamic programming problems. Dynamic programming is a discipline that deals with solving problems that are characterized by a separable objective function. This can be specified analytically or in the form of an oriented or unoriented graph. The work focuses on the Dijsktr algorithm and its modifications. The work results in a Python code implementation and several illustrative examples are given.

Keywords: separable function, oriented graph, unoriented graph, dynamic programming, Dijkstra's algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm

Chtěl bych poděkovat prof. Ing. Romanu Prokopovi, CSc. za cenné rady, odbornou pomoc, a hlavně za podporu a trpělivost.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Rád bych na úvod zmínil krátký příběh, který je dost spjatý s touto prací. Během mého studia na gymnáziu jsem se účastnil olympiády v informatice. Jedním z úkolů bylo nalezení nejkratší cesty mezi městy. Celková úloha byla znázorněna jako graf cest mezi uzly. Každý uzel znázorňoval město a hrany cesty spojující jednotlivá města. Samozřejmě zadané startovací město a konečné město neměli společnou hranu. Musím ještě dodat, že každá hrana měla přiřazenou hodnotu délky. A naším úkolem bylo najít nejkratší cestu z počátečního uzlu ke konečnému. Bohužel jsem tenkrát neznal algoritmy pro hledání minimálních cest v grafu nebo celkovou metodiku dynamického programování. Tenkrát jsem se rozhodl, že možným nejlepším východiskem jít po nejmenších hranách. Nicméně tohle mé řešení se prokázalo jako chybné. O pár let později jsem konečně narazil na správné řešení během studia na Univerzitě Tomáše Bati v předmětu Optimalizačních metod. Fascinovaly mě zejména dynamické programování a už zmíněné algoritmy pro hledání extrémů v grafech. Nakonec to vyústilo v napsání této práce.

Nicméně tato bakalářská práce se zaměřuje na část dynamického programování, a to v zejména hledání extrému v grafech pomocí algoritmů. Specificky řečeno Dijkstrův algoritmus, Bellman-Fordův algoritmus a Floyd-Warshallův algoritmus. V teoretické části uvádím základy pro pochopení daných problémů spjaté s danými algoritmy. V teoretické části můžete najít základy optimalizačních metod, úvod do dynamického programování, charakteristiku jednotlivých algoritmů, základní znalosti o teorii grafů a programovací jazyk, ve kterém jsou algoritmy realizované.

V praktické části se pak nachází popis jednotlivých algoritmů společně s programovou realizací. Každý algoritmus je pak vysvětlen na vhodném příkladu. Z celkového zpracování jde pak vidět, jak jednotlivé algoritmy fungují. Celkovým smyslem této práce je vytvoření studijních materiálů, které by pak studenti můžou využít pro seznámení možného postupu pro řešení hledání minimálních cest v grafu.

I. TEORETICKÁ ČÁST

1 OPTIMALIZAČNÍ METODY

Optimalizační metody jsou matematickou disciplínou zabývající se nalezením takových hodnot proměnných, pro které daná cílová či účelová funkce nabývá minimálních nebo maximálních hodnoty. Takže se třeba může jednat o nalezení daného bodu, nebo množiny bodů dané funkce, které vychází jako nejlepší výsledek pro dané podmínky námi zadaný problém. Počet konečných řešení čistě závisí na samotné zadané funkci, zároveň na jejím definičním oboru a omezeních, na nichž je prozkoumávána.

Optimalizace je výběr nejlepšího prvku s ohledem na nějaké kritérium z určité množiny dostupných alternativ. Optimalizační problémy se objevují ve všech kvantitativních oborech od informatiky a inženýrství až po operační výzkum a ekonomii a vývoj metod řešení je předmětem zájmu matematiky již po staletí.

V obecnějším pojetí spočívá optimalizační problém v maximalizaci nebo minimalizaci reálné funkce systematickým výběrem vstupních hodnot z přípustné množiny a výpočtem hodnoty funkce. Zobecnění teorie a technik optimalizace na jiné formulace představuje rozsáhlou oblast aplikované matematiky. [1, s. v; 4, s. 12]

1.1 Formulace problému

Pojďme si říct, že máme danou funkci $f: A \rightarrow \mathbb{R}$, kde naše přípustná množina A spadá do množiny reálných čísel a hledáme optimální řešení $x_0; x_0 \in A$. Pokud se jedná o úlohu minimalizace pak hledáme takové $x_0 \in A$, pro které platí $f(x_0) \leq f(x)$ kde $x \in A$. Pokud se ale jedná o úlohu maximalizace pak hledáme takové $x_0 \in A$, pro které platí $f(x_0) \leq f(x)$ kde $x \in A$. [12, s. 8-9; 4, s. 12]

1.2 Základní pojmy

1.2.1 Ostrý a neostrý extrém

V bodě x_0 se funkce f se nachází ostré lokální maximum, jestliže existuje okolí bodu x_0, ve kterém pro všechna x platí $f(x) < f(x_0)$, s výjimkou bodu $x = x_0$.

V bodě x_0 se funkce f se nachází neostré lokální maximum, jestliže existuje okolí bodu x_0, ve kterém pro všechna x platí $f(x) \leq f(x_0)$. [12, s. 9-10; 4, s. 14]

1.2.2 Lokální a globální extrém

Mějme neprázdnou množinu X z euklidovského n-rozměrného prostoru E^n. Dále si definujme funkci f, která má v bodě $x_0 \in X$ lokální maximum vzhledem k X, pokud je f na množině X definované a pokud existuje okolí bodu x_0, ve kterém platí $f(x) \leq f(x_0)$ pro všechna x z tohoto okolí, která jsou součástí množiny X.

Pokud ale na stejné množině platí $f(x) < f(x_0)$ a zároveň $x \neq x_0$, nejedná se o lokální, ale o ostré globální maximu vzhledem k množině X.

Je potřeba dodat, že pokud v předchozích vztazích funkcích obrátíme znaménka nerovnosti, z maxima se nám stává minimum funkce. [12, s. 9-10; 4, s. 14]

1.2.3 Volný a vázaný extrém

Pokud vyšetřujeme extrémy funkce n proměnných na celém prostoru E^n uvádíme extrém jako volný extrém. Ovšem pokud zkoumáme pouze takové extrémy, které splňují nějaké další zadané podmínky, z tohoto hlediska se jedná o vázané extrémy. [12, s. 9-10; 4, s. 14]

1.2.4 Podmínky optimality

Podmínky optimality nám slouží k redukování množiny přípustných řešení a platí pro optimální řešení. [4, s. 14; 9, s. 14]

1.3 Kategorizace

Do optimalizačních metodik spadají:

- Lineární programování
    - Metodika optimalizace pro řešení problému hledání minimálních a maximálních extrémů v lineární funkci. [9, s. 118]
- Nelineární programování
    - Metodika optimalizace pro řešení problému hledání minimálních a maximálních extrémů v nelineární funkci. [9, s. 79]
- Celočíselné programování
    - Disciplína zabývající se optimalizačními případy s podmínkou, že některé nebo všechny proměnné musí nabývat celočíselných hodnot. [7, s. 87]
- Konvexní programování

  - Optimalizační disciplína zkoumající problém minimalizace konvexních funkcích nebo maximalizace konkávních funkcích přes konvexní množinu. [9, s. 168]
- Kvadratické programování
  - Metodika optimalizace pro řešení optimalizačních problémů zahrnující kvadratické funkce. [4, s. 31]
- Dynamické programování
  - Metodika pro řešení optimalizačních problémů, které jsou charakterizované separovatelnou účelovou funkcí. [4, s. 37]
- Stochaistické programování
  - Optimalizační disciplína zabývající se modelováním optimalizačních problémů obsahující neurčitost. [11, s. 27]

2 DYNAMICKÉ PROGRAMOVÁNÍ

Tato kapitole se zabývá definováním dynamického programování a s ním spjaté algoritmy. Zároveň je uvedena lehká ilustrace postupů pro tabulkovou formu algoritmu. Pro síťovou formu je lepší přejít na následující kapitoly, která se už daným problémem zabývá.

2.1 Základní definice

Z hlediska optimalizační matematiky je dynamické programování velmi účinným nástrojem, který se dá využít pro řešení optimalizačních problémů v mnoha oborech. Celkově se uvádí, že za vznikem jejich metodik a principů stojí práce amerického matematika Richarda Bellmana.

Hlavní myšlenkou dynamického programování pro řešení daného problému spočívá v jeho rozkladu na snadnější podproblémy, které jsou řešitelné. Tomuto postupu rozdělení říkáme dekompozice problému. Následné mezivýsledky z jednotlivých vyřešených podproblémů si pak uložíme pro další použití. Tímto nám vzniká princip, který nazýváme princip invariantního vnoření. Tento Bellmanův základní princip optimality díky své jednoduchosti svých základů činí ze sebe velmi zajímavou metodiku pro zpracování pomocí výpočetní techniky.

Následně pak spojením všech mezivýsledků získáváme pak celkové řešení. Výsledkem pak úlohy dynamického programování je vždy nalezení globálního extrému, i když někdy řešení nebývá jednoznačné.

Ovšem dynamické programování se setkává určitou nevýhodou. Metoda má rychle narůstající nároky potřebných materiálů s rostoucím počtem instancí. Hlavním potřebným materiálem je paměť pro algoritmy díky rekurentním výpočtům a uchováváním mezivýsledků. [4, s. 37; 2, s. 283; 3, s. 6-8; 5, s. 1, 7; 9, s. 13]

2.2 Postup

Všechny způsoby řešení pomocí dynamického programování náleží do množiny zvané jako separovatelné programování, fungující na principu

$$f(x_1, x_2 \ldots, x_n) = \sum_{i=1}^{n} f(x_i)$$

Následně jako příklad můžeme uvést: $f(x_1, x_2, x_3) = 0{,}8x_1^3 + 0{,}5x_2^2 + 0{,}2x_3^2$

Jedním ze způsobů řešení těchto problémů je tabulkový zápis. Funguje na principu využívání separovatelné účelové funkce.

Pojďme si vyřešit úlohu dynamického programování za pomocí tabulkové formy. Tento postup má velké využití v ekonomice. Nejvíce je vhodný na rozdělování zdrojů.

$$min\, f(x_1, x_2, x_3) = 0{,}75x_1^2 + 2x_2^2 + 1{,}5x_3^2$$

při omezení: $x_1 + x_2 + x_3 = 1 \wedge x_1, x_2, x_3 \geq 0$; s krokem h = 0,2

Jak můžeme vidět z tabulky, celkovou úlohu můžeme rozdělit na tři části. Ještě je potřeba uvést, že: $F_1 = 0{,}75x_1^2; F_2 = F_1 + 2x_2^2; F_3 = F_2 + 1{,}5x_3^2$

Prvním krokem bude zjištění hodnot pro F_1. Veličiny proměnné x_1 si berou ze sloupce b, které jsou dány limitem $x_1 + x_2 + x_3 = 1$, krokem h a podmínkou nezápornosti $\wedge$ $x_1, x_2, x_3 \geq 0$. Hodnoty pro F_1 se získají čistým dosazením do funkce $F_1 = 0{,}75x_1^2$.

Tabulka 1 – Ukázka prvního kroku úlohy dynamického programování

b	x_1	F_1	x_2	F_2	x_3	F_3
0,0	0,0	0,00				
0,2	0,2	0,03				
0,4	0,4	0,12				
0,6	0,6	0,27				
0,8	0,8	0,48				
1,0	1,0	0,75				

V druhé fázi se zaměřuje na hledání hodnot pro F_2. Ovšem nejdříve bude potřeba určení veličin pro x_2. Ty lze získat pomocí kombinace x_1 a x_2. Kombinací těchto dvou veličin pak lze získat min $f(x_1,x_2)$. $F_2 = F_1 + 2x_2^2$. Jelikož je zadané minimum, vybírá se tedy z kombinacích nejmenší hodnoty.

Tabulka 2 – Mezi výpočet jednotlivých kombinací pro druhý krok

x_1	x_2	F_2	x_1	x_2	F_2	x_1	x_2	F_2	x_1	x_2	F_2
0,0	0,2	0,08	0,0	0,6	0,72	0,0	0,8	1,28	0,0	1,0	2,00
0,2	0,0	0,03	0,2	0,4	0,35	0,2	0,6	0,74	0,2	0,8	1,31
			0,4	0,2	0,20	0,4	0,4	0,44	0,4	0,6	0,84
0,0	0,4	0,32	0,6	0,0	0,27	0,6	0,2	0,35	0,6	0,4	0,59
0,2	0,2	0,11				0,8	0,0	0,48	0,8	0,2	0,56
0,4	0,0	0,12							1,0	0,0	0,75

Tabulka 3 - Ukázka druhého kroku úlohy dynamického programování

b	x_1	F_1	x_2	F_2	x_3	F_3
0,0	0,0	0,00	0,0	0,00		
0,2	0,2	0,03	0,2	0,03		
0,4	0,4	0,12	0,2	0,11		
0,6	0,6	0,27	0,2	0,20		
0,8	0,8	0,48	0,2	0,35		
1,0	1,0	0,75	0,2	0,56		

V třetí fázi stačí určit poslední veličinu v tabulce.

Tabulka 4 - Mezi výpočet jednotlivých kombinací pro třetí krok

x_1,x_2	x_3	F_3
0,0	1,0	1,50
0,2	0,8	0,99
0,4	0,6	0,65
0,6	0,4	0,44
0,8	0,2	0,41
1,0	0,0	0,56

Tabulka 5 - Ukázka třetího kroku úlohy dynamického programování

b	x_1	F_1	x_2	F_2	x_3	F_3
0,0	0,0	0,00	0,0	0,00		
0,2	0,2	0,03	0,2	0,03		
0,4	0,4	0,12	0,2	0,11		
0,6	0,6	0,27	0,2	0,20		
0,8	0,8	0,48	0,2	0,35		
1,0	1,0	0,75	0,2	0,56	0,2	0,41

Konečný výsledek: f_{min}= 0,41

$x_3 = 0{,}2 \rightarrow x_1 + x_2 + 0{,}2 = 1 \rightarrow x_1 + x_2 = 0{,}8$

$b_k = 0{,}8 \rightarrow x_1 + 0{,}2 = 0{,}8 \rightarrow x_1 = 0{,}6$

Určení hodnoty $min\ f(x_1, x_2, x_3) = 0{,}75x_1^2 + 2x_2^2 + 1{,}5x_3^2$:

$f(x_1, x_2, x_3) = 0{,}75x_1^2 + 2x_2^2 + 1{,}5x_3^2 = 0{,}75 * 0{,}6^2 + 2 * 0{,}2^2 + 1{,}5 * 0{,}2^2 = 0{,}41$

Kontrola omezení:

$x_1 + x_2 + x_3 = 0{,}6 + 0{,}2 + 0{,}2 = 1$

Omezení bylo splněno

Dalším způsobem řešení úlohy dynamického programování je síťová forma, často nazývána dopravní úloha. Spočívá v tom, že daný problém je převeden na síťový graf a následně nalezneme optimální trajektorii jeho průchodu. Řešení těchto problémů se zabývá další kapitola této práce. [3, s. 11; 9, s. 15; 4, s. 37-41]

3 DIJKSTRŮV ALGORITMUS A JEHO MODIFIKACE

V následující kapitole lze naleznout rozbor a popis jednotlivých algoritmů dynamického programování určené pro práci s grafy cest. Dále je tu také nastínění řešení hledání optimální trajektorie v síťových grafech za pomocí těchto algoritmů.

3.1 Dijkstrův algoritmus

Dijkstrův algoritmus je algoritmus pro hledání nejkratších cesty mezi uzly v ohodnoceném grafu. Jeho autorem je nizozemský matematik Edsger W. Dijkstra (1930-2002), který ho publikoval roku 1958. Algoritmus existuje v mnoha možných modifikacích. Hlavně se uplatňuje pro řešení nejkratších cest v silniční komunikaci, telefonních rozvodech nebo v routovacích protokolech pro internetovou síť.

Dijkstrův původní algoritmus hledal nejkratší cestu mezi dvěma danými uzly, ale častější varianta určuje jeden uzel jako počáteční uzel a hledá nejkratší cesty od zdroje ke všem ostatním uzlům v grafu, v čemž nám vzniká strom nejkratších možných cest. Pro daný počáteční neboli zdrojový uzel v grafu najde algoritmus nejkratší cestu mezi tímto uzlem a každým dalším. Lze jej také použít k nalezení nejkratších cest z jednoho uzlu do jednoho cílového uzlu, a to tak, že se algoritmus zastaví, jakmile je určena nejkratší cesta do cílového uzlu.

Pokud například uzly grafu představují města a náklady na hranové cesty představují jízdní vzdálenosti mezi dvojicemi měst spojených přímou silnicí, pak lze Dijkstrův algoritmus použít k nalezení nejkratší cesty mezi jedním městem a všemi ostatními městy. Široce používanou aplikací algoritmů nejkratší cesty jsou síťové směrovací protokoly. Dále je na příklad využíván v Johnsonově algoritmu jako podprogram.

Dijkstrův algoritmus používá značky, které jsou kladná celá čísla nebo reálná čísla, která jsou zcela uspořádaná. Lze jej zobecnit na použití libovolných částečně uspořádaných štítků za předpokladu, že následné štítky (následný štítek vzniká při průchodu hranou) jsou monotónně neklesající. Toto zobecnění se nazývá obecný Dijkstrův algoritmus nejkratší cesty. Dijkstrův algoritmus používá datovou strukturu pro ukládání a dotazování dílčích řešení seřazených podle vzdálenosti od počátku. Původní Dijkstrův algoritmus nepoužívá frontu s minimální prioritou a jeho časová složitost se dá určit rovnicí $\Theta(\ |V|^2\ )$ kde |V| nám určuje celkový počet uzlů neboli vrcholů v grafu. Nicméně je možné tento algoritmus modifikovat použitím prioritní fronty implementované haldou k optimalizaci časové složitosti běhu na

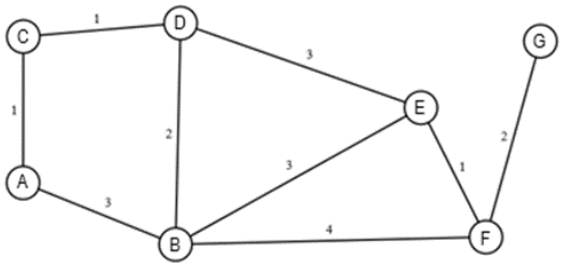
Θ (| E | + | V | log | V |), kde |V| je celkový počet vrcholů a |E| nám vyjadřuje počet hran v grafu. Touto úpravou získáme tak nejrychlejší známý algoritmus nejkratší cesty z jednoho zdroje ke všem ostatním uzlům v grafu s nezápornými hodnotami.

Dále je potřeba uvést, že Dijksktrův algoritmus díky své mechanice se tak řadí mezi hladové (greedy) algoritmy. Zároveň splňuje podmínky, aby se o něm dalo uvažovat jako algoritmus se základy dynamického programování. [8, s. 102-104; 6, s. 36-37; 2, s. 146-149; 7, s. 151-154; 11, s. 31-21]

3.1.1 Postup

Celková metodika spočívá v řešení jednotlivých podproblémů. Podproblém je myšleno vzdálenost od zvoleného uzlu k jeho sousedům. Zvolený uzel u sebe má hodnotu vzdálenosti od souseda, který byl už navštíven (pokud zkoumáme teprve počátek hodnota je nulová). K hodnotě přičte vzdálenost k nenavštívenému sousedovi a pak danou hodnotu mu předá. Jakmile takhle ohodnotí všechny své nenavštívené sousedy, je označen za navštíveného a pokračuje ve zkoumání dalšího nenavštíveného uzlu jenž má k sobě přidanou nejmenší hodnotu.

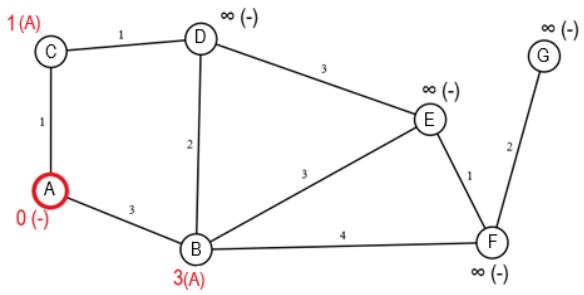
Pro objasnění je uvažován následující graf. Pomocí Dijkstrova algoritmu se zjistí všechny minimální vzdálenosti z uzlu 0 do ostatních. Jako počáteční uzel je zvolen uzel A. K ostatním uzlům je moci si připsat minimální vzdálenost od uzlu A a uvést, který uzel jim danou hodnotu přiřadil. Na začátku se označují zatím neznámé hodnoty vzdálenosti od počátečního uzlu jako ∞ a (-) pro přiřazený uzel. Jak lze vidět na grafu č.1 uzel 0 má společné hrany s uzly B a C. Tímto lze uzly B a C prohlásit za sousedy uzlu A.



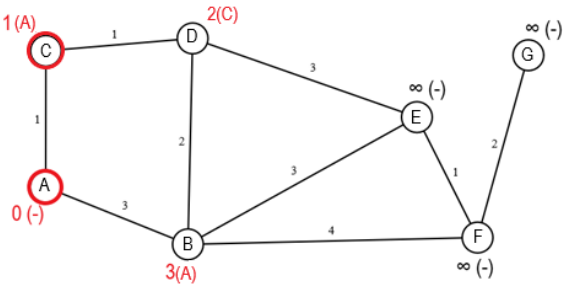
Graf č. 1 pro ukázku postupu Dijkstrova algoritmu

Připíše se tedy k uzlům B a C jejich minimální vzdálenost k uzlu A. Minimální vzdálenost pro uzel B se dá vypočítat jako vzdálenost v uzlu A + hodnota vzdálenosti hrany spojující uzel A s uzlem B. Společně s tím se zaznačí, ze kterého uzlu byla hodnota přidána (daný

uzel je uveden v závorce). Tady se může prohlásit podproblém hledání vzdálenosti zvoleného uzlu k jeho sousedům za vyřešený. Pak do prioritní fronty se přidají označené sousedy uzlu A. Podle pokynů hledání minimálního extrému jako další uzel pro zkoumání je zvolen uzel C kvůli nejkratší vzdálenosti.

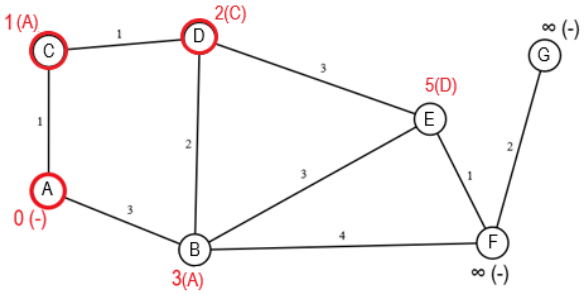


Graf č. 2 – Ukázka zkoumání počátečního uzlu A pomocí Dijkstrova algoritmu

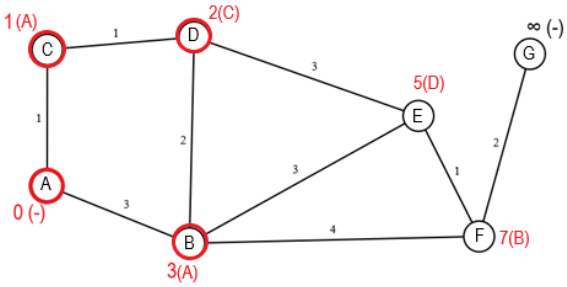


Graf č. 3 - Ukázka zkoumání počátečního uzlu C pomocí Dijkstrova algoritmu

Zvolením uzlu C lze vidět, že má pouze jednoho nenavštíveného souseda (graf č.3). Provede se tedy přepsání hodnoty na uzlu D. Uzel C se označí za navštívený a přejde se na další uzel s nejkratší vzdáleností. Uzel D má dva nenavštívené sousedy. Pouze ale u jednoho dochází zapsání nejkratší možné hodnoty, kterou lze získat sčítáním dané vzdálenosti na zkoumaném uzlu a hodnotou hran. Podproblém je tedy vyřešený a přechází se na další. U zkoumaného uzlu 1 dochází u podobné situace jako předtím. Změny se zapíší a pokračuje se na další.



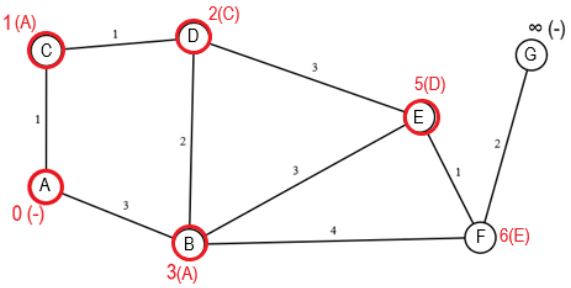
Graf č. 4 - Ukázka zkoumání počátečního uzlu D pomocí Dijkstrova algoritmu



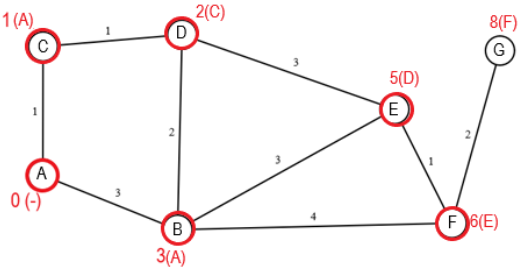
Graf č. 5 - Ukázka zkoumání počátečního uzlu B pomocí Dijkstrova algoritmu

Zkoumaný uzel E má pouze jednoho nenavštíveného souseda, u kterého lze přepsat přidanou hodnotu. U uzlu F je zapsaná vzdálenost z uzlu B, ale z uzlu E vede kratší vzdálenost, proto se hodnoty přepisují. Podproblém se vyhodnocen za vyřešený a pokračuje se na další. Při

zkoumání uzlu F se dochází ke konci, neboť jeho nenavštívený soused nám nenabízí žádnou další možnou cestu. Zapíšou se tedy hodnoty a algoritmus došel do konce.

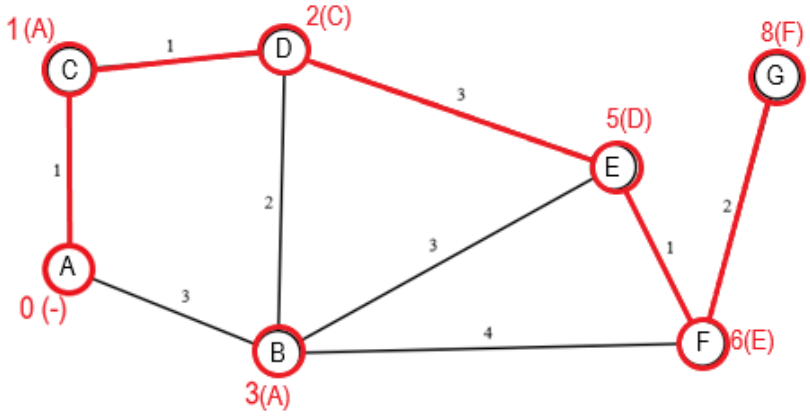


Graf č. 6 - Ukázka zkoumání počátečního uzlu E pomocí Dijkstrova algoritmu



Graf č. 7 - Ukázka zkoumání počátečního uzlu F pomocí Dijkstrova algoritmu

Výstupem je graf, ve kterém jsou zobrazené všechny minimální vzdálenosti od uzlu 0 k ostatním. Může být dáno, že je potřeba vědět cestu z uzlu A do uzlu G. Z pohledu dynamického programování pak stačí učinit zpětnou konečnou kontrolu. Začíná se z uzlu G a podle uvedených hodnot v závorkách se označují jednotlivé hrany k uvedeným uzlům. Tímto postupem je možné se dostat až k počátečnímu uzlu A. Problém vyhledání minimální cesty z uzlu A do uzlu G je tak vyřešen, díky rozložení na jeho podproblémy. [7, s. 154-155; 2, s. 146-149; 8, s. 104]



Graf č. 8 - Ukázka minimální cesty z uzlu A do uzlu G pomocí Dijkstrova algoritmu

3.2 Bellman-Fordův algoritmus

Bellman-Fordův algoritmus je algoritmus, který počítá nejkratší cesty z jednoho počátečního vrcholu do všech ostatních vrcholů váženého grafu. Je pomalejší než Dijkstrův algoritmus

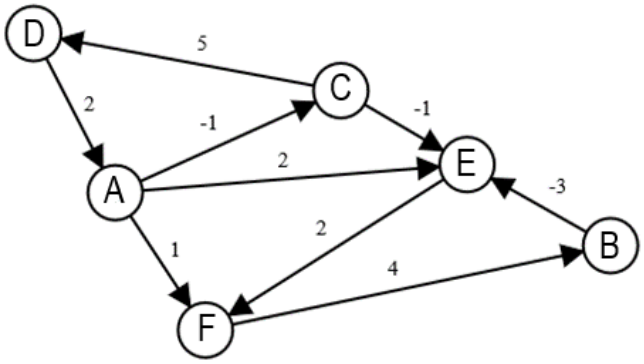
pro stejný problém, ale je univerzálnější, protože je schopen zpracovat grafy, v nichž jsou některé váhy hran záporná čísla. Algoritmus je pojmenován po Richardu Bellmanovi a Lesteru Fordovi mladším, kteří jej publikovali v roce 1958 a 1956. Variantu algoritmu publikoval v roce 1959 také Edward F. Moore, a proto se někdy nazývá také Bellman-Ford-Mooreův algoritmus.

Záporné váhy hran se vyskytují v různých aplikacích grafů. Proto je tento algoritmus užitečný. Pokud graf obsahuje "záporný cyklus" (cyklus, jehož součet hran je záporný), který je dosažitelný ze zdroje, pak neexistuje žádná nejlevnější cesta: každá cesta, která má bod na záporném cyklu, může být zmenšena ještě jednou procházkou kolem záporného cyklu. V takovém případě může Bellman-Fordův algoritmus záporný cyklus odhalit a ohlásit.

Bellman-Fordův algoritmus funguje na podobném principu jako Dijkstrův algoritmus. Až na tu výjimku, že místo prioritní fronty s haldou využívá čistá fronta implementována pouhou tabulkou obsahující všechny vrcholy a k něm přiřazené hodnoty vzdálenosti od počátečního uzlu. Celkově toto pole prochází cyklem $|V|$-1krát, $|V|$ tu vyjadřuje počet uzlů v grafu. Každým cyklem kontroluje jednotlivé uzly, kontroluje jejich jednotlivé vzdálenosti ke svým sousedům a při nalezení menší hodnot, stanovené hodnoty se změní. Díky tomuto postupu jeho časová složitost dá uvést jako $\Theta(\,|V| * |E|\,)$, kde |V| se označuje jako počet uzlů a |E| znázorňuje počet hran v grafu. [8, s. 106-107; 6, s. 41-43]

3.2.1 Postup

Pro lepší vysvětlení je uvažováno o následujícím orientovaném grafu (graf č.9) obsahující některé hrany se zápornými hodnotami. Pomocí Bellman-Fordova algoritmu lze najít všechny nejkratší cesty z počátečního uzlu 0 do ostatních uzlů. V začátku je zapotřebí si vytvořit tabulku, do které budou zaneseny všechny uzly společně se známými vzdálenostmi k stanovenému počátku v uzlu A. Vzdálenost pro uzel A zadaná jako nulová, zatímco ostatním stanovena vzdálenost ∞, neboť jsou zatím neznámé.

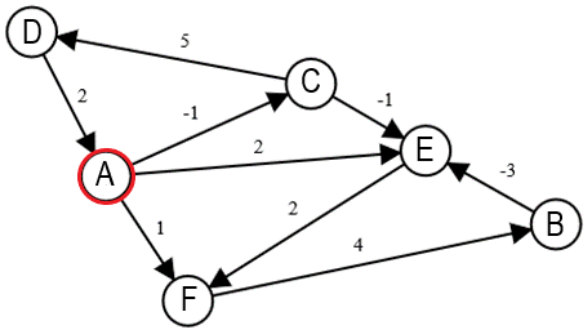


Graf č. 9 – Graf pro ukázku postupu Bellman-Fordova algoritmu

Tabulka 6 – Tabulková realizace vzdáleností od počátečního uzlu

uzel	A	B	C	D	E	F
vzdálenost	0	∞	∞	∞	∞	∞

Celkový počet potřebných iterací pro algoritmus je dán jako |N|-1 z čehož pro tento graf vyplývá 5 iterací (neboť graf je tvořen ze 6 uzlů) pro nalezení všech minimálních cest v grafu bez negativního zacyklení. Každá iterace prozkoumává jednotlivé uzly zapsané v tabulce. Pokud zkoumaný uzel nemá vzdálenost jdoucí k nekonečnu, pokračuje se na další uzel v tabulce. V prvním kroku první iterace je zkoumán uzel A. Jelikož nemá vzdálenost jdoucí k nekonečnu, ověřují se jaké hodnoty lze připsat k jeho sousedům. Nová hodnota připíše k sousednímu uzlu do tabulky, pokud sečtením vzdálenosti zkoumaného uzlu a hodnoty hrany jdoucí k danému sousedovi je menší než hodnota vzdálenosti daného souseda. V tomto případě lze vidět, že zkoumáním uzlu A přepisujeme hodnoty u uzlů C, E a F.



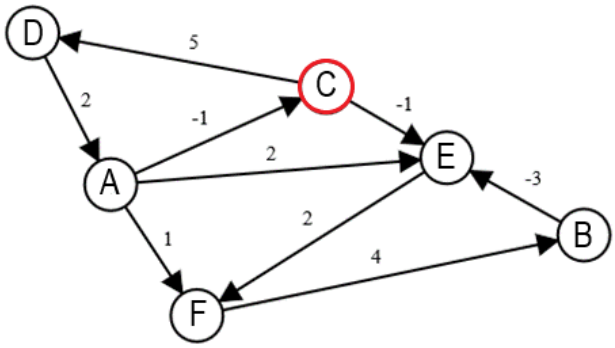
Graf č. 10 - Ukázka zkoumání uzlu A pomocí Bellman-Fordova algoritmu

Tabulka 7 – Ukázka změn po zkoumání uzlu A

uzel	A	B	C	D	E	F
vzdálenost	0	∞	-1	∞	2	1

Pokračuje se tabulkou na uzel B. Ten ovšem nemá přesně danou hodnotu, proto se pokračuje dál. U uzlu C prozkoumáním je zjištěna hodnota pro uzel D. Dále se zjišťuje, že součtem

hodnoty uzlu C a hodnoty hrany jdoucí k uzlu E je získána menší hodnota než hodnota uzlu E. Toto zjištění je uvedeno do tabulky.

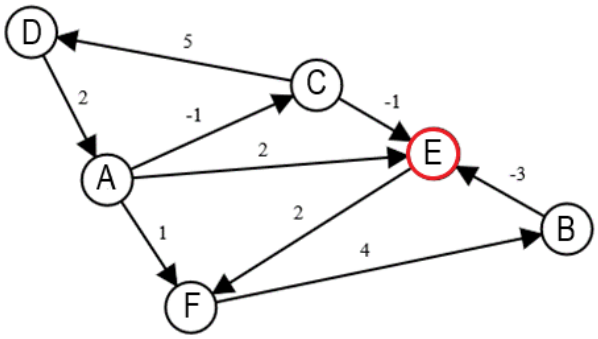


Graf č. 11 - Ukázka zkoumání uzlu C pomocí Bellman-Fordova algoritmu

Tabulka 8 - Ukázka změn po zkoumání uzlu C

uzel	A	B	C	D	E	F
vzdálenost	0	∞	-1	4	-2	1

Pokračuje se na uzel D, ve kterém nedochází k žádné změně. Přejde se na uzel E. Zde nacházíme menší hodnotu pro uzel F. Změnu se zapíše do tabulky.

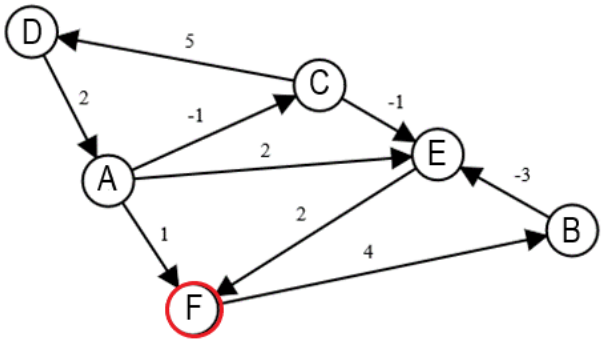


Graf č. 12 - Ukázka zkoumání uzlu E pomocí Bellman-Fordova algoritmu

Tabulka 9 - Ukázka změn po zkoumání uzlu E

uzel	A	B	C	D	E	F
vzdálenost	0	∞	-1	4	-2	0

Na uzlu F lze vidět nalezení hodnoty pro uzel B. Nalezená hodnota se připíše uzlu B do tabulky. Došlo se na konec první iterace.

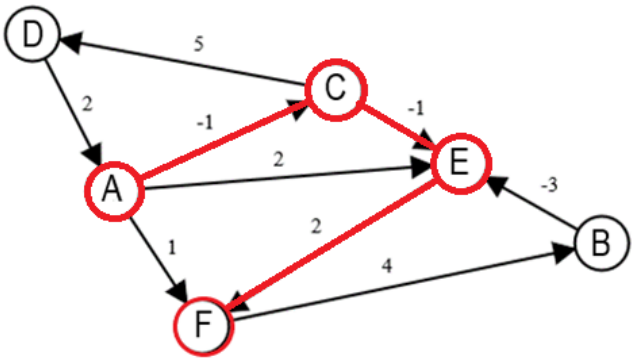


Graf č. 13 - Ukázka zkoumání uzlu F pomocí Bellman-Fordova algoritmu

Tabulka 10 Ukázka změn po zkoumání uzlu F

uzel	A	B	C	D	E	F
vzdálenost	0	4	-1	4	-2	0

Druhá iterace probíhá zase od začátku tabulky. Zkoumáním uzlu A není nalezena žádná změna. Stejný případ nastává i při kontrole uzlu B. Stejné výsledky nastávají i při průzkumu ostatních uzlů až do konce druhé iterace. Po skončení iterace lze ukončit celý algoritmus, neboť pokud jsou vykonány iterace při, které nedojde k žádnému přepisu vzdáleností, pak lze konstatovat, že byly nalezeny všechny nejkratší cesty z uzlu 0 do ostatních uzlů. Zároveň algoritmus byl splněn do stanoveného počtu 5 iterací, z čehož vychází, že graf neobsahuje negativní zacyklení. [8, s. 107; 6, s. 42]



Graf č. 14 – Ukázka cesty z uzlu A do uzlu F

3.3 Floyd-Warshallův algoritmus

Roku 1962 byl publikován a formulován americkým počítačovým vědcem Robertem Floydem. Nicméně celkový postup algoritmu se shodoval s algoritmem od amerického počítačového vědce Stephen Warshall, publikován stejného roku. Floyd-Warshallův algoritmus je klíčovým algoritmem v oblasti teorie grafů, který slouží k nalezení nejkratších cest mezi

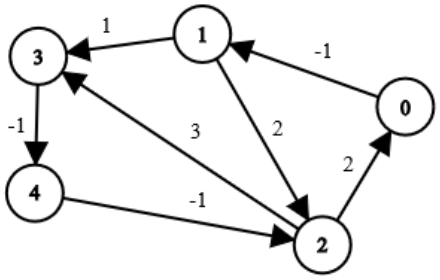
všemi páry uzlů v ohodnoceném orientovaném grafu. Tento algoritmus je efektivní zejména pro grafy jak s kladnými, tak i negativními hranami.

Floyd-Warshallův algoritmus je založen na principech dynamického programování a řeší daný problém iterativně. Základní myšlenkou je postupně aktualizovat tabulku vzdáleností mezi všemi páry uzlů tak, aby nakonec obsahovala nejkratší vzdálenosti mezi všemi dvojicemi uzlů. Algoritmus tak má časovou složitost $\Theta(|V|^3)$, kde |V| vyjadřuje počet uzlů v grafu. Pro výpočet nejkratších cest mezi všemi dvojicemi uzlů je tedy algoritmus poměrně efektivní, zejména pro malé a středně velké grafy.

Jeho celkové uplatnění můžeme nalézt v různých oblastech, jako je například routování v síťových systémech, hledání nejkratších cest v dopravních sítích, nebo optimalizace cest ve logistice. Především je algoritmus důležitým nástrojem pro práci s ohodnocenými orientovanými grafy. Jeho schopnost nalézt nejkratší cesty mezi všemi páry uzlů ho činí užitečným nástrojem pro řešení různých problémů v informatice a dalších disciplínách. [8, s. 104-106; 7, s. 156-158; 11, s. 37-38; 6, s. 45-47; 2, s. 154]

3.3.1 Postup

Je předpokládá orientovaný graf č.15 s N uzly a ohodnocenými hranami. Každá hrana má přiřazenou váhu, která může být jak kladná, tak záporná. Pokud mezi uzly neexistuje hrana, váha je dána jako nekonečno. Jako nejdříve si převedeme náš graf do tabulkového tvaru.



Graf č. 15 - Graf pro ukázku postupu Floyd-Warhallova algoritmu

Tabulka 11 - Tabulková forma matice sousednosti pro graf č.14

	0	1	2	3	4
0	0	-1	∞	∞	∞
1	∞	0	2	1	∞
2	2	∞	0	3	∞
3	∞	∞	∞	0	-1
4	∞	∞	-1	∞	0

V tabulkovém zápisu řádky představují počáteční uzly a konečné uzle jsou pak představeny jednotlivými sloupci. Příkladem může být cesta z uzlu 0 do uzlu 1 jak lze vidět v grafu č.14. Přesněji když se půjde ze řádku nula do sloupce jedna, v dané buňce se nachází stejná hodnota, která je váhou dané hrany v grafu č.14. Tuto tabulku je možné si označit jako zdroj ze kterého se budou brát hodnoty pro první iteraci. Pro tento příklad je pojmenována DIST. Dále bude potřeba veličin *i*, *j* a *k*, která každá má vlastní množinu hodnot {0, 1, …|V|-1}. Pro zjišťování hodnot bude potřeba rovnice DIST[i][k] + DIST[k][j], hodnoty z této rovnice budou pak dávány do speciální tabulky (pomDIST). Algoritmus začíná vytvořením první tabulky pro k = 0, veličiny *i* budou zastupovat jednotlivé řádky tabulky a veličiny *j* zastupují sloupce.

Tabulka 12 – pomDist pro k = 0

k = 0		j				
		0	1	2	3	4
i	0	0	-1	∞	∞	∞
	1	∞	∞ - 1	∞	∞	∞
	2	2	1	∞ + 2	∞ + 2	∞ + 2
	3	∞	∞ - 1	∞	∞	∞
	4	∞	∞ - 1	∞	∞	∞

A nyní zjištěné hodnoty naší tabulky pro k=0 (pomDIST) se bude porovnávat s tabulkou DIST. Pokud podmínka DIST[x][y]> pomDIST[x][y] bude splněna, daná hodnota v DIST[x][y] se přepíše hodnotou z pomDIST[x][y]. Příkladem může být:

DIST [2][1]> pomDIST [2][1] DIST [1][2]> pomDIST [1][2]

∞> 1 2> ∞

Podmínka je splněna Podmínka není splněna

Změny v tabulce se budou vyznačovat pro lepší orientaci. [7, s. 158; 6, s. 46-48; 2, s. 154]

Tabulka 13 – DIST po první iteraci

	0	1	2	3	4
0	0	-1	∞	∞	∞
1	∞	0	2	1	∞
2	2	1	0	3	∞
3	∞	∞	∞	0	-1
4	∞	∞	-1	∞	0

Teď upravenou tabulku DIST lze použít pro další iteraci. Nyní se vytvoří tabulka pro k=1:

Tabulka 14 - pomDist pro k = 1

k = 1		j				
		0	1	2	3	4
i	0	∞ - 1	-1	1	0	∞ - 1
	1	∞	0	2	1	∞
	2	∞ + 1	1	3	2	∞ + 1
	3	∞	∞	∞+2	∞ + 1	∞
	4	∞	∞	∞+2	∞ + 1	∞

Výsledná tabulka DIST po srovnání s pomDIST:

Tabulka 15 - DIST po druhé iteraci

	0	1	2	3	4
0	0	-1	1	0	∞
1	∞	0	2	1	∞
2	2	1	0	2	∞
3	∞	∞	∞	0	-1
4	∞	∞	-1	∞	0

Nyní se vytvoří tabulka pro k=2:

Tabulka 16 - pomDist pro k = 2

k = 2		j				
		0	1	2	3	4
i	0	3	2	1	3	∞ + 1
	1	4	3	2	4	∞ + 2
	2	2	1	0	2	∞
	3	∞ + 2	∞ + 1	∞	∞ + 2	∞
	4	1	0	-1	1	∞-1

Výsledná tabulka DIST po srovnání s pomDIST.

Tabulka 17 - DIST po třetí iteraci

	0	1	2	3	4
0	0	-1	1	0	∞
1	4	0	2	1	∞
2	2	1	0	2	∞
3	∞	∞	∞	0	-1
4	1	0	-1	1	0

Nyní se vytvoří tabulka pro k=3.

Tabulka 18 - pomDist pro k = 3

k = 3	j				
	0	1	2	3	4

i	0	∞	∞	∞	0	-1
	1	∞ + 1	∞ + 1	∞ + 1	1	0
	2	∞ + 2	∞ + 2	∞ + 2	2	1
	3	∞	∞	∞	0	-1
	4	∞ + 1	∞ + 1	∞ + 1	1	0

Výsledná tabulka DIST po srovnání s pomDIST:

Tabulka 19 - DIST po čtvrté iteraci

	0	1	2	3	4
0	0	-1	1	0	-1
1	4	0	2	1	0
2	2	1	0	2	1
3	∞	∞	∞	0	-1
4	1	0	-1	1	0

Nyní se vytvoří tabulka pro k=4.

Tabulka 20 - pomDist pro k = 4

k = 4		j				
		0	1	2	3	4
i	0	0	-1	-2	0	-1
	1	1	0	-1	1	0
	2	2	1	0	2	1
	3	0	-1	-2	0	-1
	4	1	0	-1	1	0

Výsledná tabulka DIST po srovnání s pomDIST a zároveň ukončení všech iterací Floyd-Warhallova algoritmu. Nyní tabulka DIST obsahuje minimální vzdálenosti mezi jednotlivými dvojicemi uzlů v grafu.

Tabulka 21 – Tabulka DIST po ukončení Floyd-Warhallova algoritmu

	0	1	2	3	4
0	0	-1	-2	0	-1
1	1	0	-1	1	0
2	2	1	0	2	1
3	0	-1	-2	0	-1
4	1	0	-1	1	0

4 TEORIE GRAFŮ

Následující kapitole se zabývá definicí a celkovým popisem teorie grafů a pokouší se nastínit prostředí na které jsou algoritmy z dynamického programování spjaté. Celkovou myšlenkou je vysvětlit základní pojmy pro lepší chápání ostatních částí této práce.

4.1 Základní pojmy

Je dobré začít tím, že graf v matematice je chápán jako nejčastější grafické znázornění funkčních závislostí. Zatím co grafem v teorii grafů jsou chápány objekty popsané množinou vrcholů a množinou hran. Celkově se prostý graf (nazveme si ho grafem G) dá popsat jako dvojice (V, H), kde V je dáno jako množina vrcholů grafu G a $H \subseteq V^2 \cup P_2(V) \cup V$ je množinou hran grafu G.

$$V^2 = V \times V = \{(v_1, v_2) | v_1 = v_2, v_1 \in V, v_2 \in V\}$$

$$P_2(V) = \{(v_1, v_2) | v_1 = v_2, v_1 \in V, v_2 \in V\}$$

$$\{(v_1, v_2) | v_1 = v_2, v_1 \in V, v_2 \in V\} = \{v | \, v \in \, V\}$$

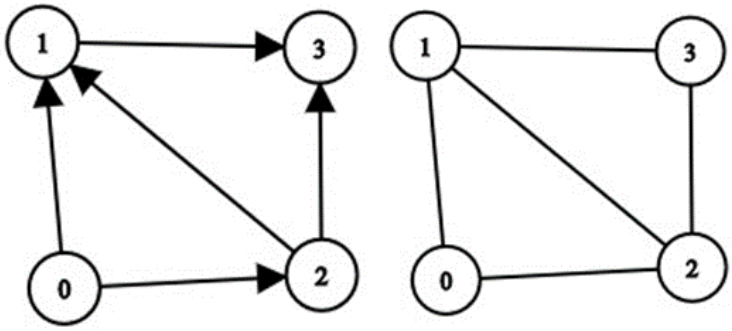
Nutno podotknout, že mezi dvěma vrcholy může existovat více hran téhož typu (např. paralelní vedení). V definici prostého grafu tato varianta není zachycena, neboť v matematice je množina definována jako výčet navzájem různých prvků. Z tohoto hlediska je potom potřeba uvést obecnější definici grafu.

Dále tu máme obecný graf, který se dá definovat jako trojice (V, H, ε), kde V je uvedené jako množina vrcholů grafu G a ε je definováno jako zobrazení incidence, [10, s. 95-100; 6, s. 9; 11, s. 5-7]

$$\varepsilon: H \longrightarrow V^2 \cup P_2(V) \cup V.$$

4.1.1 Grafické znázornění orientace:

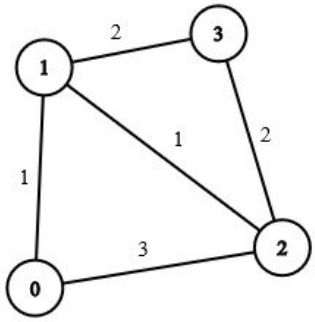
Vrcholy grafu graficky znázorňujeme jako body nebo kroužky a hrany jsou znázorňovány jako čáry spojující dané vrcholy. V orientovaných grafech jsou k hranám přidány šipky od počátečního ke koncovému vrcholu pro znázornění směru. Celkový rozdíl mezi orientovaným a neorientovaným grafem je ten, že u orientovaného grafu máme přesně stanoveno, jakým směrem se uchyluje daná hrana. Zatímco u neorientovaného grafu můžeme stanovit, že směr hrany spojující dva dané uzly je oboustranný. [7, s. 123-127; 11, s. 5-7]



Graf č. 16 – Orientovaná graf (vlevo) a neorientovaný graf (vpravo)

4.1.2 Ohodnocený graf:

Dále to máme možnost přiřazení číselných (nebo jiných) hodnot k hranám. Graf, ve kterém se nachází hrany s přiraženými hodnotami, se nazývá ohodnocený graf. [11, s. 5-7]



Graf č. 17 – Ohodnocený graf

4.2 Časová složitost

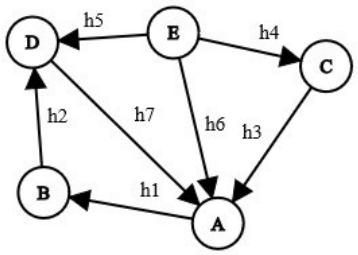
Anglicky *time complexity* z pohledu teorie grafů je dáno jako nejdelší časová doba výpočtu, který algoritmus zabere při vstupu velikosti n. Z čehož se dá vyvodit, že čas výpočtu při nejméně příznivém případu, se stává určitou funkcí *f* pro čísla *n*. Z těchto poznatků jde pak určit přibližný počet operací, který algoritmus bude potřebovat. Z pohledu matematické informatiky můžeme časovou složitost označit jako počet strojového času potřebného pro běh algoritmu. Dále si můžeme k tomuto i uvést paměťovou složitost, které vyjadřují celkové paměťové nároky nebo potřeby programu.

Nicméně tu musíme uvažovat s možností, že celkový výpočet f(n) se vyjeví dost složitě. Proto je doporučené při odhadování časového výpočtu se hlavně zaměřovat na dominantní

složku ve funkci f(n), ve kterém jsou stanoveny meze, v nichž se provedení výpočtu pohybuje. To nám celkově implikuje myšlenku o asymptotickém odhadu. Při asymptotickém odhadu se vyšetřovaná funkce *f(n)* porovnává s jednodušší funkcí g(n), která určuje mez pro provedení výpočtu. [11, s. 8-9; 2, s. 39-52; 8, s. 5-19]

4.3 Reprezentace grafu

Jednotlivé formy zápisu budou reprezentovány na tomto grafu:



Graf č. 18 - Graf pro zobrazení jednotlivých reprezentacích

4.3.1 Seznam uzlů a hran

Množiny uzlů a hran grafu jsou uvedeny ve formě seznamu. Množina uzlů je popsána prostým výpisem prvků. Množina hran je zase popsána jako seznam uspořádaných trojic, přesněji každá trojice je tvořena názvem hrany, počátečním a koncovým uzle. U orientovaného grafu je důležité dodržet správné pořadí. [6, s. 19]

Uzly: A, B, C, D, E

Hrany: (h1, A, B), (h2, B, D), (h3, C, A), (h4, E, C), (h5, E, D), (h6, E, A), (h7, D, A)

4.3.2 Seznam okolí vrcholů

Množiny uzlů jsou vypisovány jako u předchozího způsobu, jenom mají k sobě připsané hrany, pro které jsou počátečním uzlem. Každá hrana je připsaná ke svému počátečnímu uzlu a má u sebe uvedený koncový uzel. [6, s. 20]

A: (h1, B)

B: (h2, D)

C: (h3, A)

D: (h7, A)

E: (h4, C), (h5, D), (h6, A)

4.3.3 Matice incidentů

Matice incidentů je matice popisující vztahy mezi hranou a jejím koncovým uzlem. Tabulkovým zápisem se množina uzlů udává pro řádkové hodnoty a hrany zase pro sloupcové hodnoty. Hodnoty v matici pak udávají vztah mezi uzlem a hranou. [6, s. 22; 11, s. 10]

- $I_{jk} = 1$, v neorientovaný grafu je uzel incidentní s hranou, v orientovaný grafu je uzel počáteční pro hranu
- $I_{jk} = 0$, uzel není incidentní s hranou
- $I_{jk} = -1$, v orientovaný grafu je uzel koncový pro hranu

Tabulka 22 – Matice incidentů grafu č.18

I	h1	h2	h3	h4	h5	h6	h7
A	1	0	-1	0	0	-1	-1
B	-1	1	0	0	0	0	0
C	0	0	1	-1	0	0	0
D	0	-1	0	0	-1	0	1
E	0	0	0	1	1	1	0

4.3.4 Matice sousednosti

Matice sousednosti je matice zachycující sousedící uzly. Přesněji množina uzlů vyjadřuje hodnoty pro řádky i sloupce a hodnota v tabulce udává vztah dvojce uzlů. Zápis je vždy formou čtvercové matice. [6, s. 20; 11, s. 11]

- $S_{jk} = 1$, v neorientovaném grafu jsou uzly u_i a u_j sousedící, v orientovaném grafu z uzlu u_i vede cesta do uzlu u_j
- $S_{jk} = 0$, v neorientovaném grafu nejsou uzly u_i a u_j sousedící, v orientovaném grafu z uzlu u_i nevede cesta do uzlu u_j

Tabulka 23 - Matice sousednosti grafu č.18

S	A	B	C	D	E
A	0	1	0	0	0
B	0	0	0	1	0
C	1	0	0	0	0
D	1	0	0	0	0
E	1	0	1	1	0

5 PYTHON

5.1 Historie Pythonu

Python je vysokoúrovňový, interpretovaný programovací jazyk, který vytvořil Guido van Rossum a poprvé byl vydán v roce 1991. Jazyk je známý svou jednoduchostí, čitelností kódu a silnou komunitou. Od svého vzniku prošel řadou vývojových fází a stal se jedním z nejoblíbenějších programovacích jazyků na světě. [13; 14]

5.2 Filozofie Pythonu

Filozofie Pythonu, často označována jako "Zen Pythonu", obsahuje soubor zásad a hodnot, které vedou vývojáře k psaní kódu, který je čitelný, elegantní a efektivní. Mezi tyto zásady patří jednoduchost, explicitnost, krásný je lepší než ošklivý, a praktičnost před dokonalostí. [13; 14]

5.3 Charakteristiky a vlastnosti Pythonu

5.3.1 Dynamické typování

Python je dynamicky typovaný jazyk, což znamená, že datové typy jsou přiřazeny proměnným za běhu programu a nemusí být explicitně deklarovány. To zjednodušuje psaní kódu a umožňuje rychlejší vývoj. [13; 14]

5.3.2 Interpretovaný jazyk

Python je interpretovaný jazyk, což znamená, že kód není předem kompilován do strojového kódu, ale spouští se přímo interpretací zdrojového kódu. To usnadňuje vývoj a ladění kódu, ale může snížit rychlost výkonu. [13; 14]

5.3.3 Multi-paradigmatický jazyk

Python podporuje různá programovací paradigmaty, včetně imperativního, objektově orientovaného a funkcionálního programování. To umožňuje vývojářům psát kód ve stylu, který nejlépe vyhovuje danému problému. [13; 14]

5.4 Syntaxe a základní koncepty

5.4.1 Datové typy a struktury

Python nabízí širokou škálu vestavěných datových typů, včetně seznamů, slovníků, množin a n-tic. Tyto datové struktury umožňují efektivní manipulaci s daty a usnadňují implementaci různých algoritmů. [13; 14]

5.4.2 Kontrolní struktury

Python podporuje běžné kontrolní struktury, jako jsou podmínky (if-else), smyčky (for, while) a řídicí konstrukce (break, continue). Tyto struktury umožňují programátorům řídit tok programu a provádět různé operace v závislosti na podmínkách. [13; 14]

5.4.3 Funkce a moduly

Funkce jsou základní stavební bloky v Pythonu, které umožňují znovupoužitelnost kódu a organizaci programu do logických bloků. Moduly jsou soubory obsahující funkce, třídy a proměnné, které mohou být importovány do jiných částí programu. [13; 14]

5.5 Python ve vývoji softwaru

5.5.1 Využití Pythonu

Python je široce využíván ve vývoji softwaru, včetně webových aplikací, analýzy dat, strojového učení, automatizace, vědeckého výzkumu a dalších oblastí. Jeho jednoduchá syntaxe a bohaté knihovny usnadňují vývoj v mnoha různých odvětvích. [13; 14]

5.5.2 Knihovny a frameworky

Python má bohatý ekosystém knihoven a frameworků, které umožňují vývojářům rychle vytvářet sofistikované aplikace. Mezi populární frameworky patří Django a Flask pro webové aplikace, TensorFlow a PyTorch pro strojové učení a NumPy a Pandas pro analýzu dat. [13; 14]

5.5.3 Testování a ladění

Python má silnou podporu pro testování kódu a ladění aplikací. Existují různé nástroje a frameworky, jako jsou `unittest` a `pytest`, které umožňují psát a spouštět testy jednotek, integrační testy a testy přijetí. [13; 14]

5.6 Výhody a omezení Pythonu

5.6.1 Výhody Pythonu

Mezi hlavní výhody Pythonu patří čitelnost kódu, jednoduchá syntaxe, bohatý ekosystém knihoven, multi-paradigmatický charakter a silná komunita. Tyto vlastnosti činí Python oblíbeným volbou pro vývoj softwaru. [13; 14]

5.6.2 Omezení Pythonu

Mezi omezení Pythonu patří relativně pomalý výkon ve srovnání s některými jinými jazyky, omezená podpora pro některé oblasti vývoje a závislost na externích knihovnách pro některé pokročilé funkce. [13; 14]

II. PRAKTICKÁ ČÁST

V následující praktické části budou provedeny programová realizace jednotlivých algoritmů v programovacím jazyce Python. Následně ke každému jednotlivému algoritmu budou vypracovány studijní příklady s postupem.

6 DIJKSTRŮV ALGORITMUS

Podle definice se dá algoritmus realizovat jako:

T := ∅

d[s] = 0 a ∀ v ∈ V \ {s} : d[v] = ∞

∀ v ∈ V : odkud[v] = nil

vytvoř prioritní frontu H z vrcholů V s prioritami d[v]

while fronta H neprázdná do

v = DELETE MIN(H) {vyber netrvalý vrchol s nejmenším d[v]}

T := T ∪ {v}

for each vw ∈ E do

if d[w] > d[v] + c(vw) then

d[w] = d[v] + c(vw)

odkud[w] = v

DECREASE KEY(H, w) {aktualizuj haldu}

6.1 Programová realizace

Programová realizace v jazyce Python:

import heapq

Implementací algoritmu v jazyce Python importujeme knihovnu heapq pro usnadnění práce s haldou.

```
def dijkstra(graph, start):
    distances = {vertex: float('infinity') for vertex in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
```

V první části funkce si připravíme pole pro výsledné vzdálenosti od našeho počátečního bodu. Obsahuje názvy všech uzlů a jejich vzdálenosti od začátku. Definujeme vzdálenost v počátku jako nulovou a vytvoříme si frontu priority.

```
while priority_queue:
    current_distance, current_vertex = heapq.heappop(priority_queue)
```

Následuje cyklus, jenž bude procházet frontu priorit. Cyklus začíná vyjmutím prvního členu z haldy fronty priorit skládající se z vzdálenosti od začátku (current_distance) a momentální uzel (vertex).

```
    if current_distance > distances[current_vertex]:
        continue
    for neighbor, weight in graph[current_vertex].items():
        distance = current_distance + weight
        if distance < distances[neighbor]:
            distances[neighbor] = distance
            heapq.heappush(priority_queue, (distance, neighbor))
```

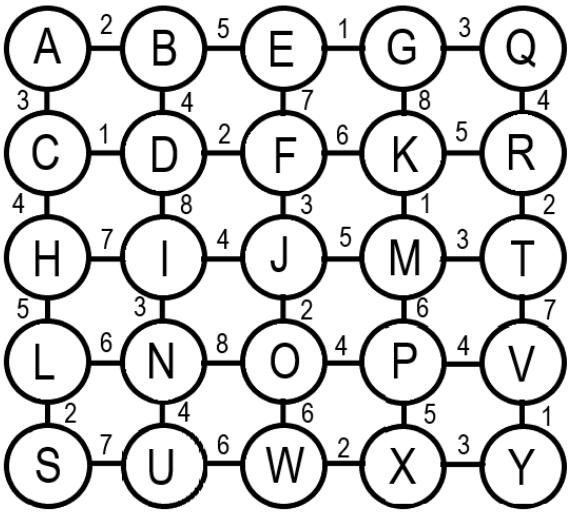
Následující for cyklus prochází sousedy zvoleného uzlu (current_vertex). Pokud je vzdálenost od current_vertex menší, než vzdálenost, kterou má soused, sousední uzel dostane novou hodnotu pro vzdálenost a je přidán do fronty priorit.

```
return distances
```

6.2 Příklady

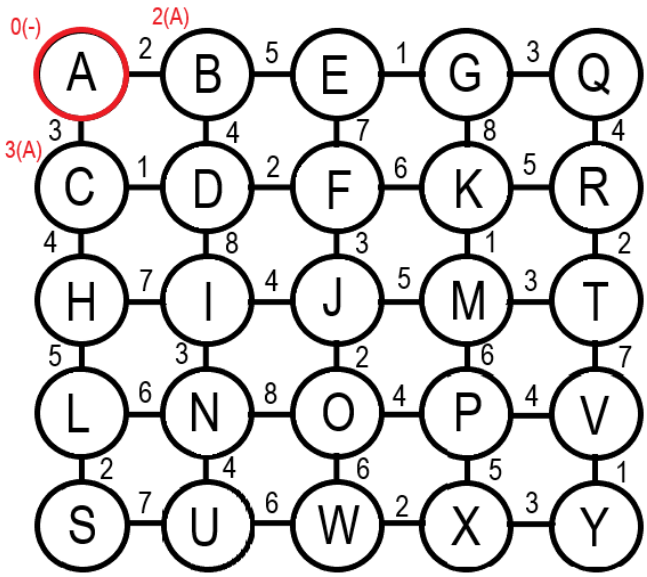
6.2.1 Příklad 1

Je dán neorientovaný graf č.19 s 25 vrcholy. Úkolem bude najít nejkratší cestu z uzlu A do uzlu Y za pomocí Dijkstrova algoritmu.



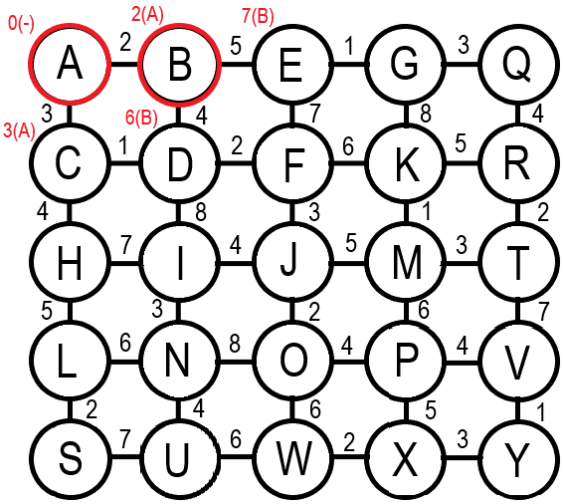
Graf č. 19 pro příklad 1 aplikace Dijkstrova algoritmu

Uzel A je počátkem a k jeho sousedícím uzlům se zapíše vzdálenost hrany, kterou mají uzly společnou. Zároveň se doplní, ze kterého uzlu byla vzdálenost brána. Cyklus pokračuje tak, že uzel A se označí za prozkoumaný a půjde se na nejbližší uzel.



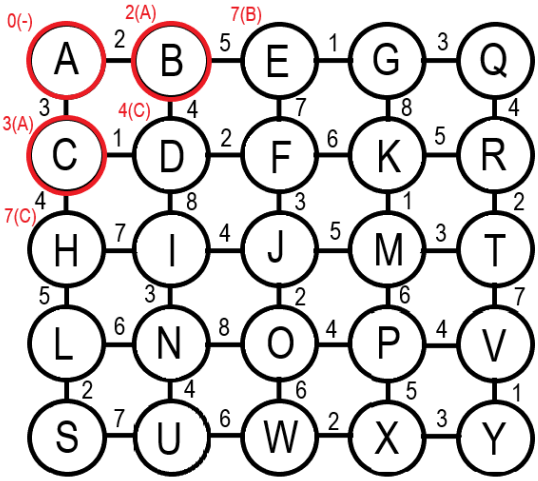
Graf č. 20 – krok 1 Dijkstrova algoritmu

Dalším zvoleným uzlem bude uzel B, neboť má menší vzdálenost od uzlu A než uzel C od uzlu A. Provede se zde ta samá operace jako byla provedena u uzlu A, jenom je potřeba si uvědomit, že ke vzdálenosti od uzlu B k jeho sousedům se přičítá vzdálenost od uzlu A k uzlu B. (Graf č.21)



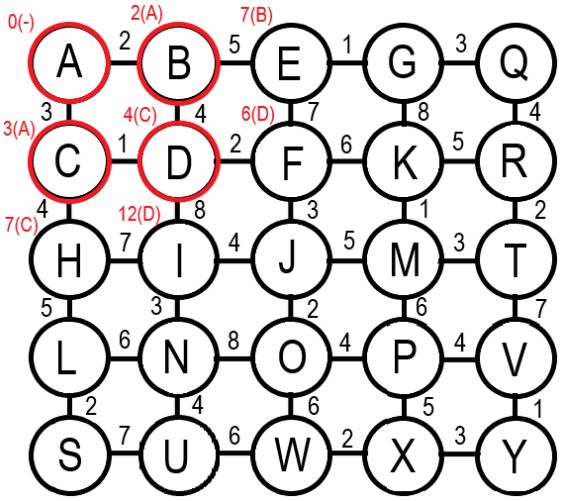
Graf č. 21 - krok 2 Dijkstrova algoritmu

V následující fázi se zjistilo, že vzdálenost z uzlu C do uzlu D je kratší než vzdálenost z uzlu B do uzlu D. Dojde tedy k přepisu informace a pokračuje se dál.



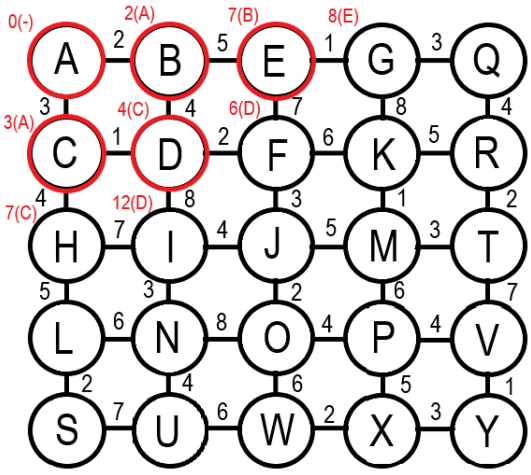
Graf č. 22 – krok 3 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel D. Označí se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



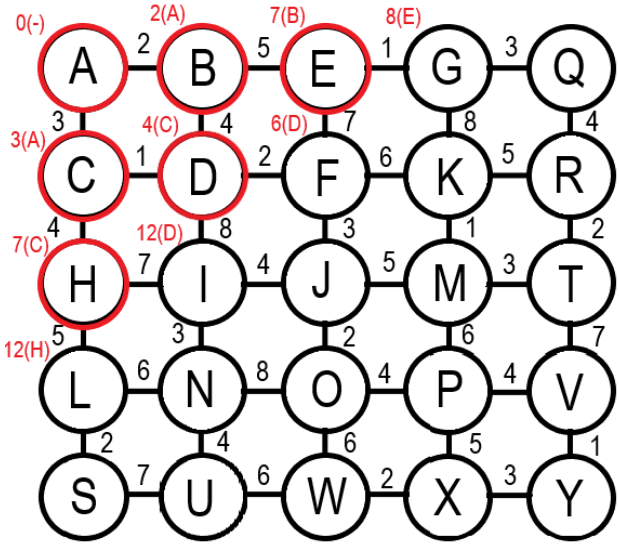
Graf č. 23 – krok 4 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel E a uzel H. Pro tento příklad se zvolí uzel E. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



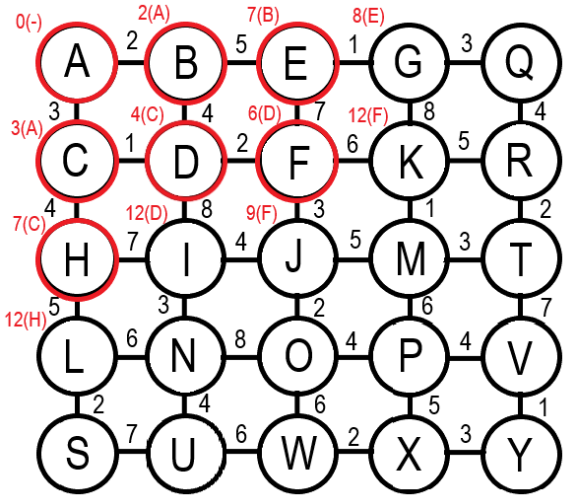
Graf č. 24 – krok 5 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel H. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



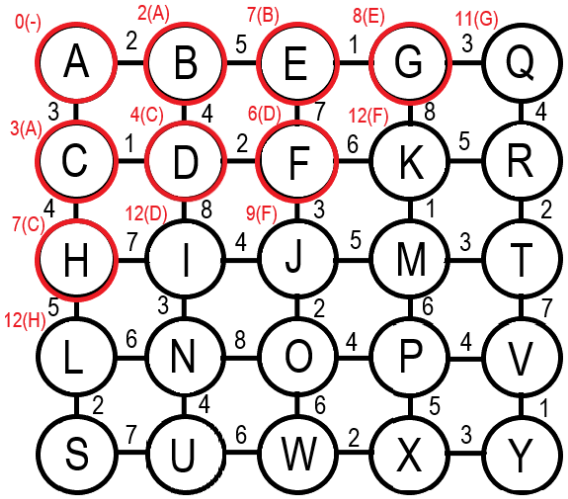
Graf č. 25 – krok 6 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel F. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



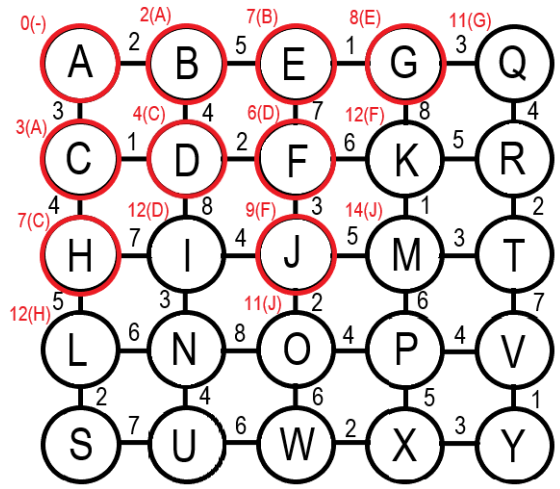
Graf č. 26 – krok 7 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel G. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



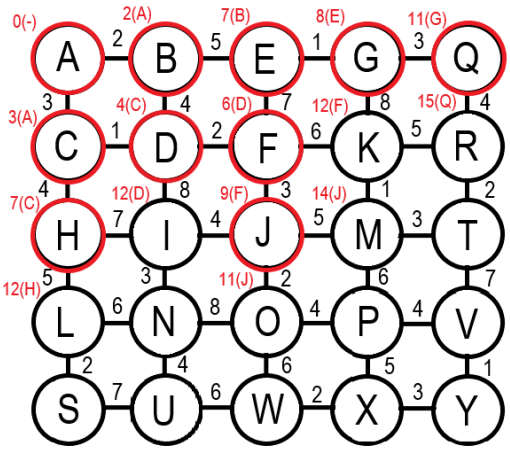
Graf č. 27 – krok 8 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel J. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



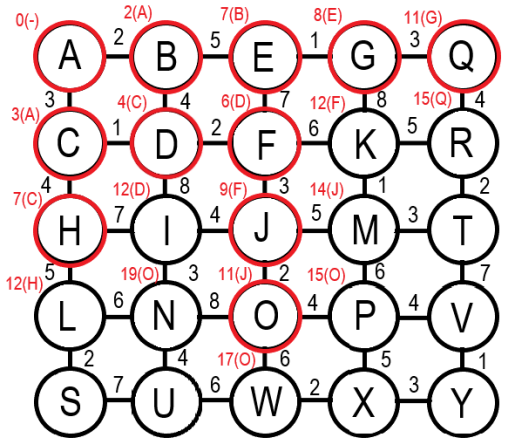
Graf č. 28 – krok 9 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel Q. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



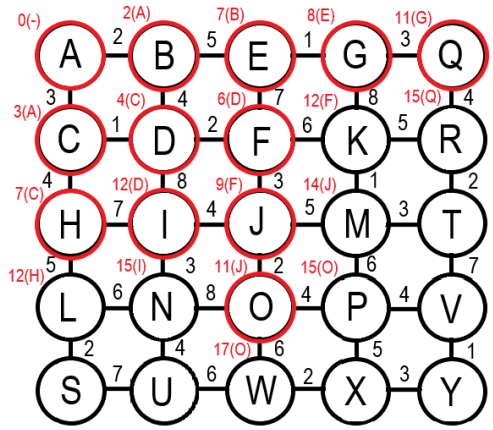
Graf č. 29 - krok 10 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel O. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



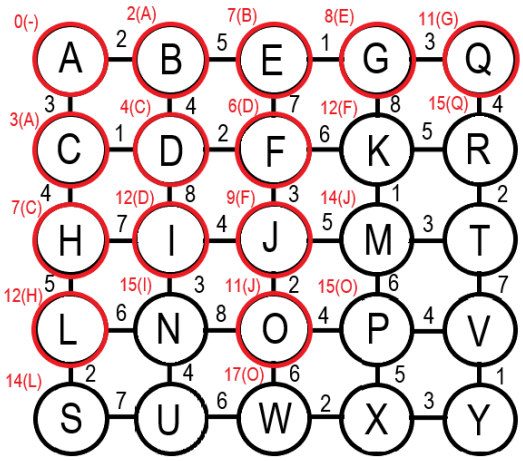
Graf č. 30 - krok 11 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel I. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace. Dochází tady k přepisu informací na uzlu N.



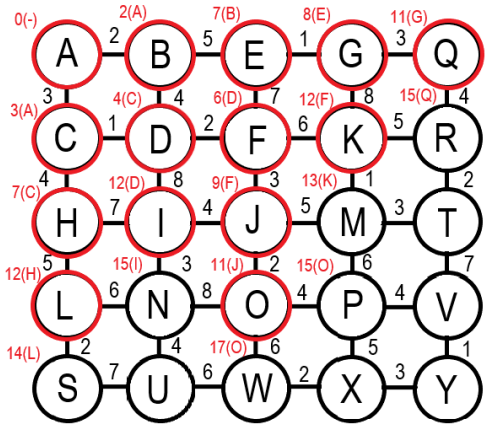
Graf č. 31 - krok 12 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel L. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



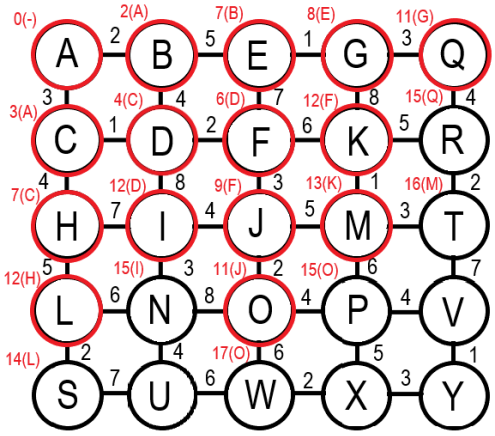
Graf č. 32 - krok 13 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel K. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace. Dochází tady k přepisu informací na uzlu M.



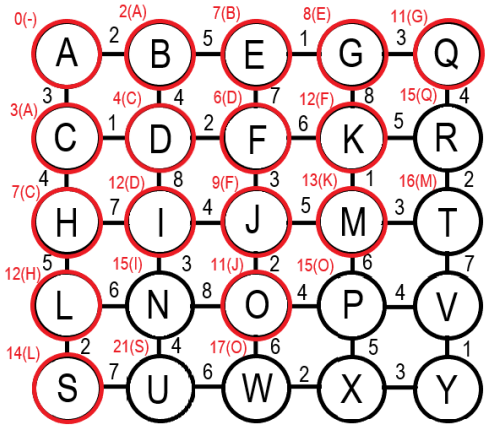
Graf č. 33 - krok 14 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel M. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



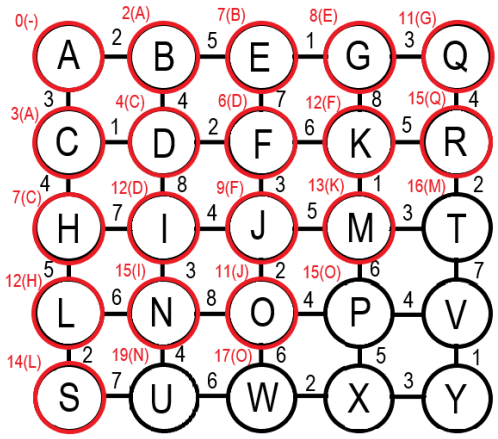
Graf č. 34 - krok 15 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel S. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace.



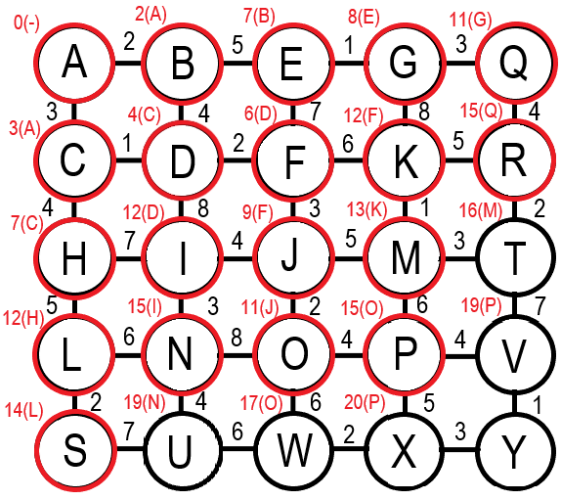
Graf č. 35 - krok 16 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel N. Zaznamená se jako prozkoumány a k jeho neprozkoumaným sousedům se přidá informace. Dochází tady k přepisu informací na uzlu U. Dále při zkoumání uzlu R nedochází k žádné změně u jeho sousedů. Zaznamená se jako prozkoumaný



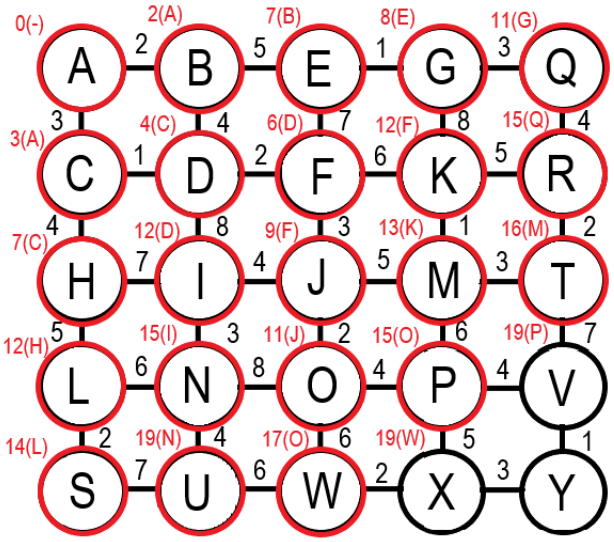
Graf č. 36 - kroky 17 a 18 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel P. Označí se jako prozkoumány a k jeho neprozkoumaným sousedům přidáme informace.



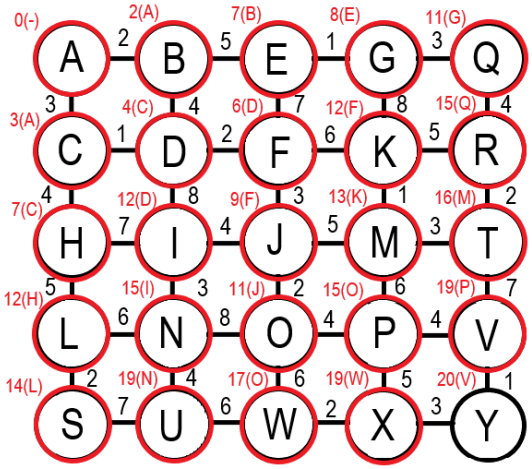
Graf č. 37 - krok 19 Dijkstrova algoritmu

V následující fázi má nejmenší vzdálenost od předchozího prozkoumaného uzlu uzel T. Označí se jako prozkoumány a k jeho neprozkoumaným sousedům neměníme nic. Volí se pak uzel W, který se označí za prozkoumaný a změníme informace u uzlu X. Uzel U se označí za prozkoumaný, protože už se nedá, kam pokračovat. (Graf č.38)



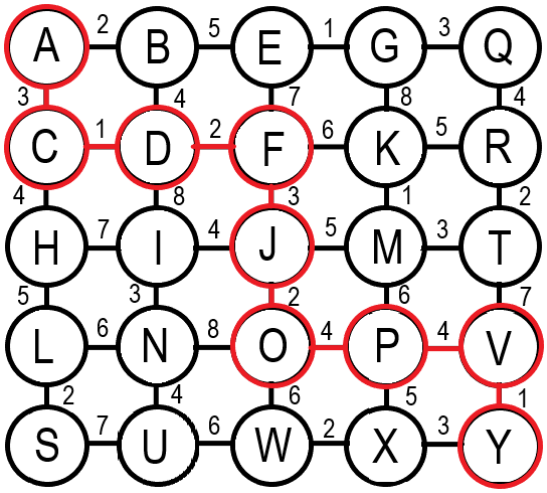
Graf č. 38 - kroky 20, 21 a 22 Dijkstrova algoritmu

V následující fázi se zkoumá uzel V. Označí se jako prozkoumány a k jeho neprozkoumaným sousedům změní se informace. Volí se pak uzel X, který se označí za prozkoumaný a nemění se nic. V konečné fázi se došlo na uzel Y, ze kterého se nelze se nijak posunout dál. Označí se to tedy za prozkoumaný.



Graf č. 39 - kroky 23 a 24 Dijkstrova algoritmu

Ve zpětném cyklu se jde z uzlu Y podle informacích, které mají jednotlivé uzly napsané u sebe. Příkladem tedy je, že z uzlu Y se přejde na uzel V. Jednotlivé navštívené uzly se označí a postupuje se, dokud se nedojde do počátku v uzlu A. Po ukončení cyklu jako výsledek vzejde nalezená cesta grafem sítě z uzlu A do uzlu Y. (Graf č.40)



Graf č. 40 – Výsledná nejkratší cesta mezi uzly A a Y

Vstup programové realizace:

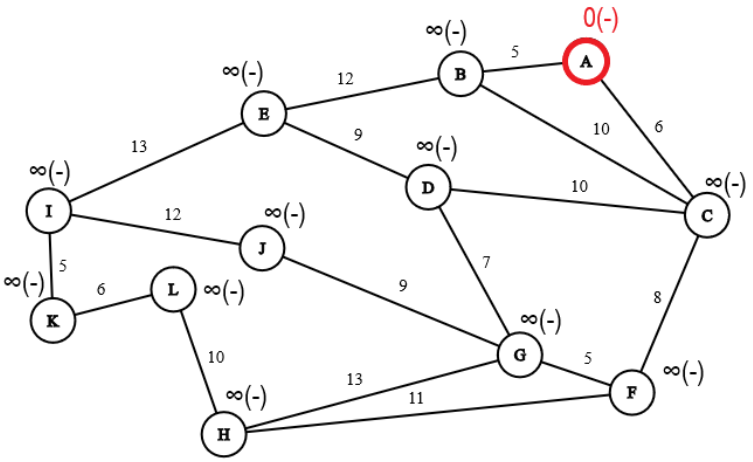
graph = {'A': {'B': 2, 'C': 3}, 'B': {'A': 2, 'D': 4, 'E': 5},'C': {'A': 3, 'D': 1, 'H': 4}, 'D': {'C': 1, 'B': 4, 'F': 2, 'I': 8}, 'E': {'B': 5, 'F': 7, 'G': 1}, 'F': {'E': 7, 'K': 6, 'J': 3, 'D': 2}, 'G': {'Q': 3, 'K': 8, 'E': 1}, 'H': {'C': 4, 'I': 7, 'L': 5}, 'I': {'D': 8, 'J': 4, 'N': 3, 'H': 7}, 'J': {'F': 3, 'M': 5, 'O': 2, 'I': 4}, 'K': {'M': 1, 'G': 8, 'F': 6, 'R': 5}, 'L': {'H': 5, 'N': 6, 'S': 2}, 'M': {'K': 1, 'T': 3, 'P': 6, 'J': 5}, 'N': {'I': 3, 'O': 8, 'U': 4, 'L': 6}, 'O': {'J': 2, 'P': 4, 'W': 6, 'N': 8}, 'P': {'M': 6, 'V': 4, 'X': 5, 'O': 4}, 'Q': {'G': 3, 'R': 4}, 'R': {'T': 2, 'Q': 4, 'K': 5}, 'S': {'L': 2, 'U': 7}, 'T': {'M': 3, 'V': 7, 'R': 2}, 'U': {'S': 7, 'N': 4, 'W': 6}, 'V': {'P': 4, 'T': 7, 'Y': 1}, 'W': {'U': 6, 'O': 6, 'X': 2}, 'X': {'W': 2, 'P': 5, 'Y': 3}, 'Y': {'X': 3, 'V': 1}}

Výstup programové realizace:

{'A': 0, 'B': 2, 'C': 3, 'D': 4, 'E': 7, 'F': 6, 'G': 8, 'H': 7, 'I': 12, 'J': 9, 'K': 12, 'L': 12, 'M': 13, 'N': 15, 'O': 11, 'P': 15, 'Q': 11, 'R': 15, 'S': 14, 'T': 16, 'U': 19, 'V': 19, 'W': 17, 'X': 19, 'Y': 20}

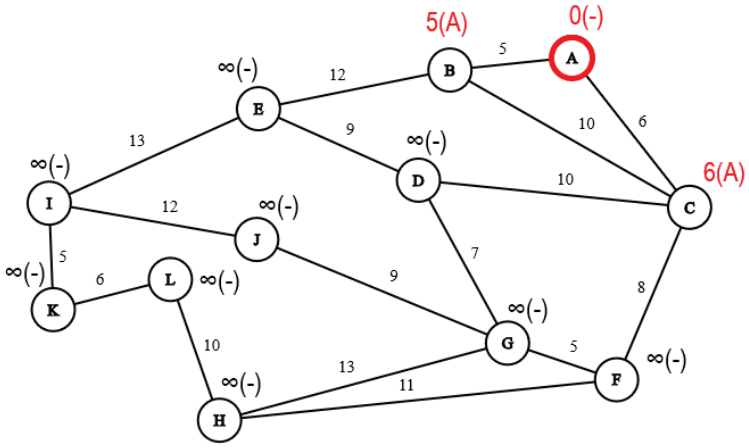
6.2.2 Příklad 2

Je dán neorientovaný graf č.41 s 12 uzly. Úkolem bude najít nejkratší cestu z uzlu A do uzlu H pomocí Dijkstrova algoritmu.



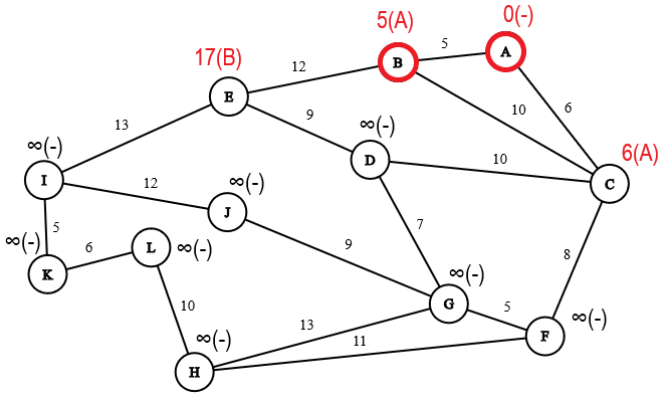
Graf č. 41 pro příklad 2 aplikace Dijkstrova algoritmu

Začne se průzkumem počátečního uzlu A. K jeho sousedům se zapíše vzdálenosti od počátku (hodnota vzdálenosti + hodnota hrany k danému sousedu). Zkoumaný uzel se odebere z fronty a přidá se do ní jeho sousedy uzel B a uzel C. Přechází se na uzel B, neboť má nejkratší hodnotu ve frontě.



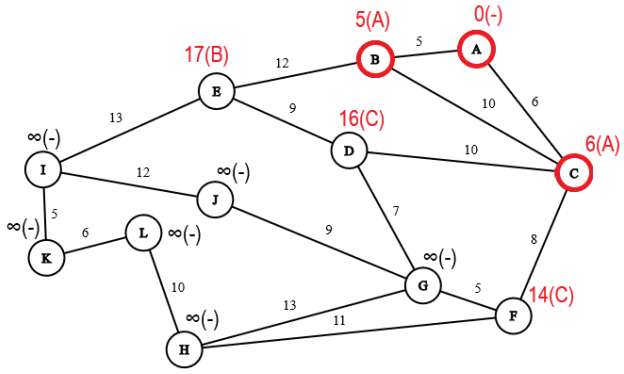
Graf č. 42 – krok 1 Dijkstrova algoritmu

Zkoumáním uzlu B se zjistí, že hodnoty se mění pouze v sousedním nenavštíveném uzlu E. Uzel B se odebere z fronty a označí se za navštívený. Pak se přidá uzel E do fronty. Přejde se na uzel C.



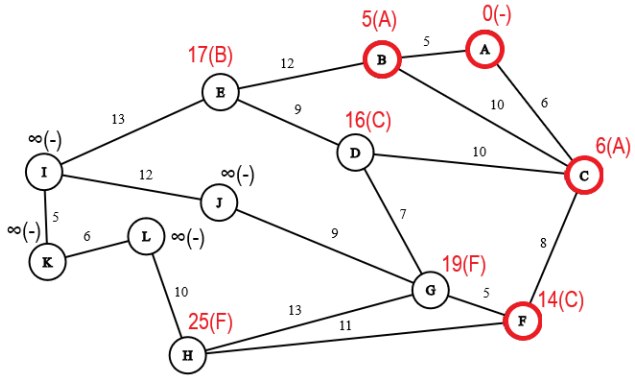
Graf č. 43 - krok 2 Dijkstrova algoritmu

Zkoumáním uzlu C se nalézají hodnoty pro sousední nenavštívené uzly D a F. Uzel C se odebere z fronty a označí se za navštívený. Dále se přidají uzly D a F do fronty. Přejde se na uzel F.



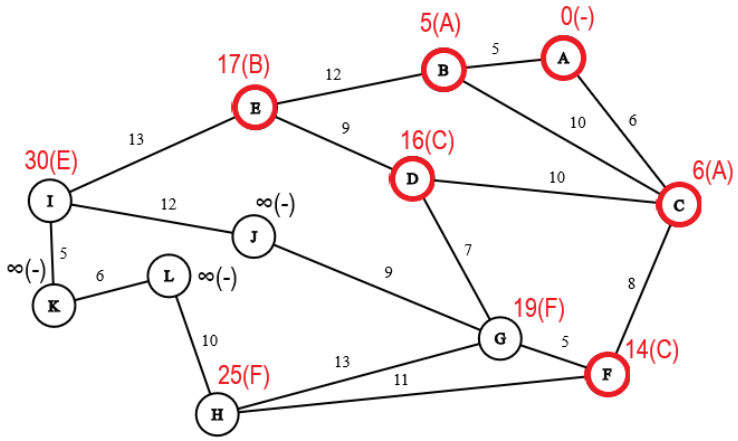
Graf č. 44 - krok 3 Dijkstrova algoritmu

Zkoumáním uzlu F se nalézají hodnoty pro sousední nenavštívené uzly G a H. Uzel F se odebere z fronty a zaznamená se jako navštívený. Dále se přidají uzly G a H do fronty. Přejde se na uzel D.



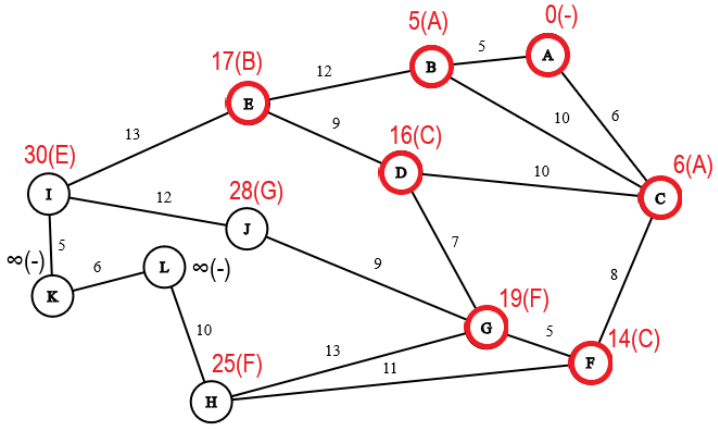
Graf č. 45 - krok 4 Dijkstrova algoritmu

Zkoumáním uzlu D se nezíská žádné nejmenší hodnoty pro jeho sousední nenavštívené uzly. Uzel se odebere z fronty a označí za navštívený. Přejde se na uzel E. Zkoumáním tohoto uzlu se získává hodnota pro uzel I. Uzel E se odebere z fronty a označí za navštívený. Dále se přidá uzel I do fronty. Přejde se na uzel G.



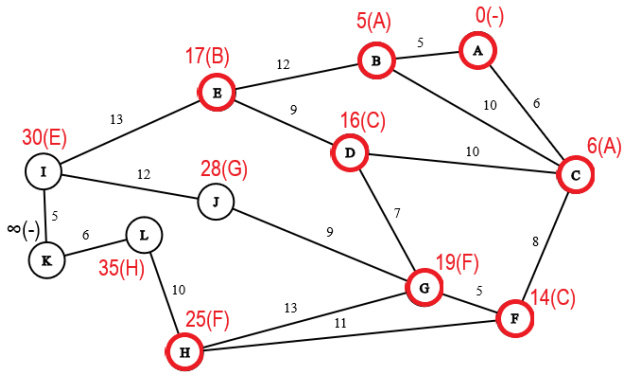
Graf č. 46 - kroky 5 a 6 Dijkstrova algoritmu

Zkoumáním uzlu G se získá hodnota pro uzel J. Uzel G se odebere z fronty a označí za navštívený. Dále se přidá uzel J do fronty. Přejde se na uzel H.



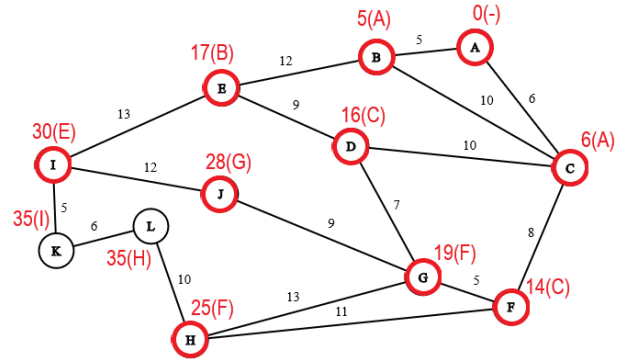
Graf č. 47 - krok 7 Dijkstrova algoritmu

Zkoumáním uzlu H se nalézá hodnota pro uzel L. Uzel H se odebere z fronty a označí za navštívený. Dále se přidá uzel L do fronty. Přejde se na uzel J.



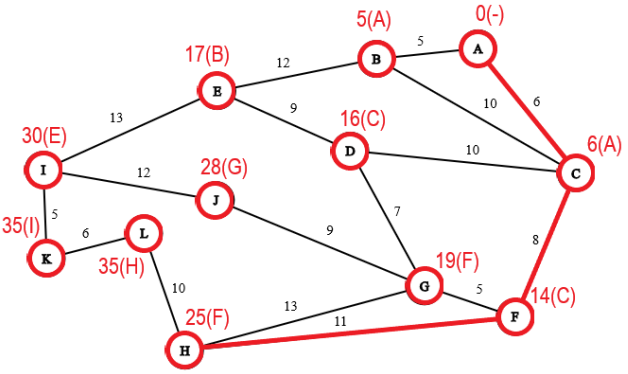
Graf č. 48 - krok 8 Dijkstrova algoritmu

Zkoumáním uzlu J nedojde k nalezení žádné nejmenší hodnoty pro jeho sousední nenavštívený uzel. Uzel se odebere z fronty a označí za navštívený. Přejde se na uzel I. Zkoumáním tohoto uzlu se nalezne hodnota pro uzel K. Uzel I se odebere z fronty a označí za navštívený. Dále se přidá uzel K do fronty. Přejde se na zbývající uzly.



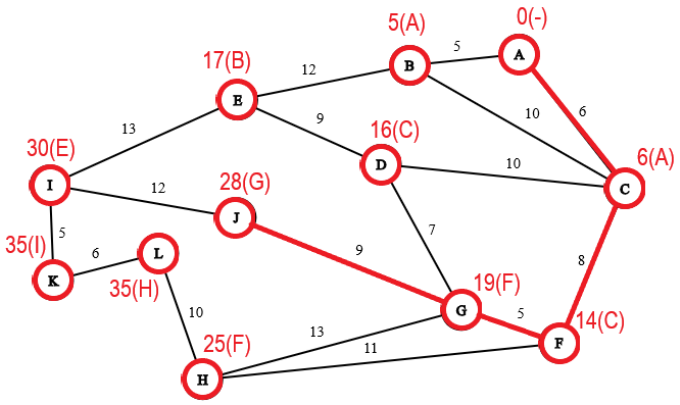
Graf č. 49 – poslední kroky Dijkstrova algoritmu

Zkoumáním zbylých uzlů se přichází k závěru, že nedochází ke změně uvedených hodnot. Proto se zaznačí jako navštívené a algoritmus dochází svého konce. Teď už se jenom provede zpětný krok nalezení cesty. Z uzlu H se půjde na uzel uvedený v závorkách. Tímto postupem u každého uzlu se dojde až do uzlu A.



Graf č. 50 – Výsledná nejkratší cesta mezi uzly A a H

Stejným postupem lze najít cestu z uzlu A do uzlu J například:



Graf č. 51 - Výsledná nejkratší cesta mezi uzly A a J

Vstup programové realizace:

graph = {

'A': {'B': 5, 'C': 6}, 'B': {'A': 5, 'C': 10, 'E': 12}, 'C': {'A': 5, 'D': 10, 'B': 10, 'F': 8}, 'D': {'C': 10, 'G': 7, 'E': 9}, 'E': {'B': 12, 'D': 9, 'I': 13}, 'F': {'G': 5, 'C': 8, 'H': 11}, 'G': {'D': 7, 'F': 5, 'J': 9, 'H': 13}, 'H': {'L': 10, 'G': 13, 'F': 11}, 'I': {'E': 13, 'J': 12, 'K': 5}, 'J': {'I': 12, 'G': 9}, 'K': {'I': 5, 'L': 6}, 'L': {'H': 10, 'K': 6}

}

Výstup programové realizace:

{'A': 0, 'B': 5, 'C': 6, 'D': 16, 'E': 17, 'F': 14, 'G': 19, 'H': 25, 'I': 30, 'J': 28, 'K': 35, 'L': 35}

7 BELLMAN-FORDŮV ALGORITMUS

Pseudokód:

```
procedure BellmanFord(graph, source):
    distance[source] := 0
    for each vertex v in graph:
        if v ≠ source:
            distance[v] := infinity
    for i from 1 to size(graph)-1:
        for each edge (u, v) in graph:
            if distance[u] + weight(u, v) < distance[v]:
                distance[v] := distance[u] + weight(u, v)
                predecessor[v] := u
    for each edge (u, v) in graph:
        if distance[u] + weight(u, v) < distance[v]:
            return "Graph contains a negative-weight cycle"
    return distance, predecessor
```

Kde:

- `graph` je graf reprezentovaný maticí sousednosti nebo seznamem hran,
- `source` je počáteční vrchol, pro který hledáme nejkratší cesty,
- `distance` je pole, které uchovává vzdálenosti od počátečního vrcholu ke každému jinému vrcholu,
- `predecessor` je pole, které uchovává informace o předchůdcích na nejkratších cestách,
- `weight(u, v)` je váha hrany mezi vrcholy `u` a `v`.

Tento pseudokód provádí relaxaci všech hran ve všech iteracích až do dosažení optimálních hodnot. Poté se kontroluje, zda graf neobsahuje záporný cyklus.

7.1 Programová realizace

Programová realizace v jazyce Python:

```
def bellman_ford(graph, src):

    vertices = len(graph)

    dist = {vertex: float('infinity') for vertex in graph}

    dist[src] = 0
```

Program začíná vytvořením konstanty určující počet uzlů v grafu a datové struktury dist. Dist slouží jako dictionary, kde uzel slouží jako key faktor. Každému uzlu v dist je přiřazena hodnota vzdálenosti ∞. Počátečnímu uzlu přiřadíme hodnotu nula.

```
    for _ in range(vertices - 1):

        for u in dist.keys():
```

```
        for v, w in graph[u].items():
            if dist[u] != float('infinity') and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
```

Algoritmus probíhá ve třech for cyklech. První for cyklus běží podle dané podmínky, kolik iterací algoritmus potřebuje pro nalezení cesty. Druhý for cyklus bere názvy uzlů z dictionary dist. Třetí cyklus si z dictionary graph bere názvy a hodnoty hran sousedních uzlů pomocí názvu zkoumaného uzlu. Pak se kontroluje, jestli zkoumaný uzel má hodnotu a jestli součet hodnoty zkoumaného uzlu s délkou hrany je menší než hodnota v sousedním uzlu. Pokud je podmínka splněna dojde k přepisu hodnoty v sousedícím uzlu.

```
    for u in dist.keys():
        for v, w in graph[u].items():
            if dist[u] != float('infinity') and dist[u] + w < dist[v]:
                print("Negativní zacyklení")
                return
```

Následující dva for cykly slouží pouze kontrolu, jestli graf neobsahuje negativní zacyklení.

```
    print_solution(dist)
```

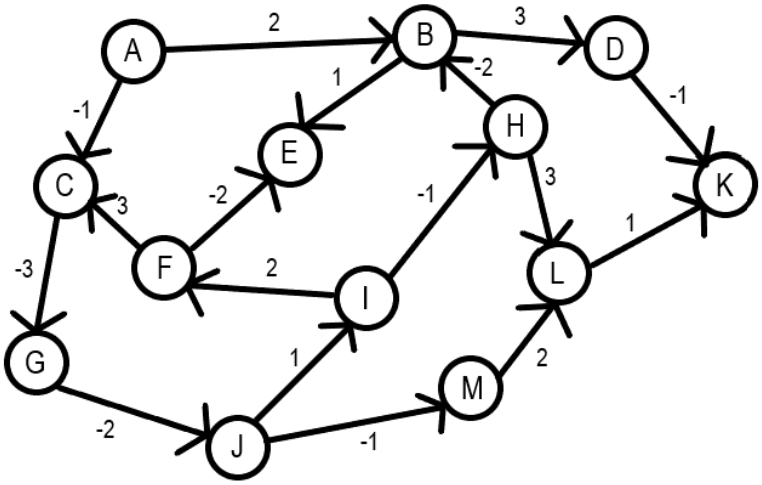
Funkce pro výpis výstup.

```
def print_solution(dist):
    for k, i in dist.items():
        print(k + ': ' + str(i))
```

7.2 Příklady

7.2.1 Příklad 1

Je zadán orientovaný graf sítě s 13 vrcholy (graf č.52). Úkolem bude najít nejkratší cesty z uzlu A do ostatních uzlů za pomocí Bellman-Fordova algoritmu. Ke grafu se vytvoří tabulka se svými vzdálenostmi od počátečního uzlu. Při startu algoritmu se vzdálenosti všech ostatních uzlů předpokládá jako nekonečno. (Tabulka 24)

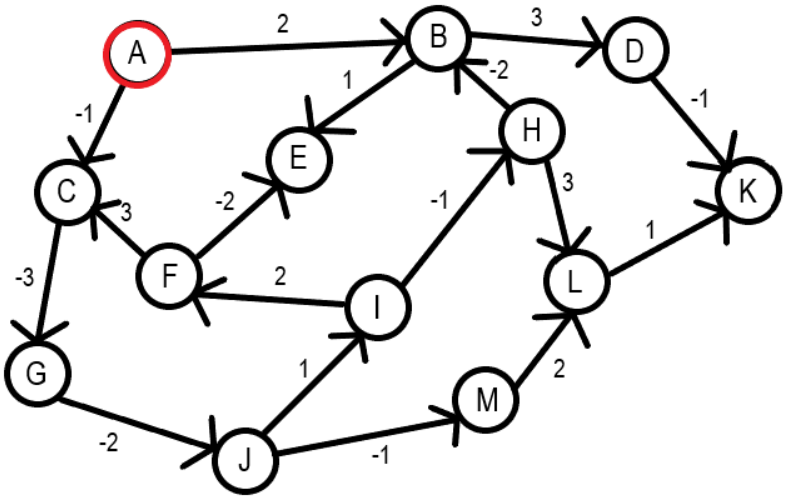


Graf č. 52 pro příklad 1 aplikace Bellman-Fordova algoritmu

Tabulka 24 - Tabulka vzdáleností od počátečního uzlu

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Jednotlivá iterace v cyklu probíhá v tom, že se postupuje podle zadané tabulky a zapisuje se nejmenší hodnota vzdálenosti. Jako první je označen uzel A. Zapíšou se vzdálenosti k jeho sousedům do tabulky 25.

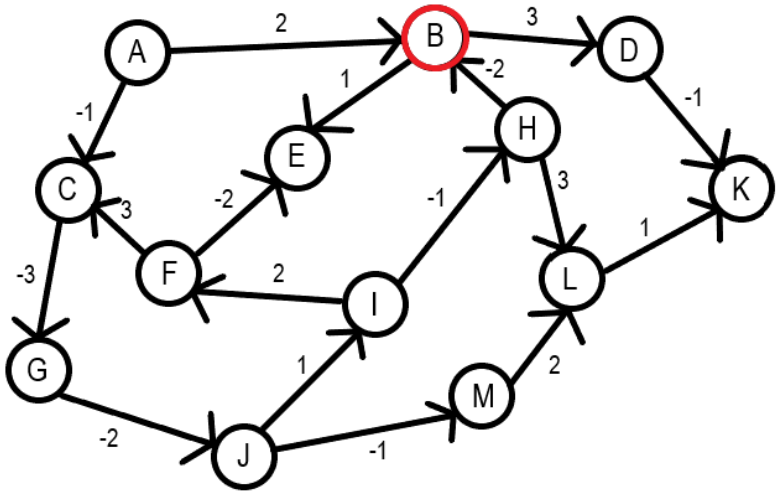


Graf č. 53 - Zkoumání uzlu A pomocí Bellman-Fordova algoritmu

Tabulka 25 – Změny vzdáleností při zkoumání uzlu A

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Dále následuje v tabulce uzel B. Do tabulky se zapíšou nejmenší vzdálenosti k sousedům, kam směřuje. (Tabulka 26)

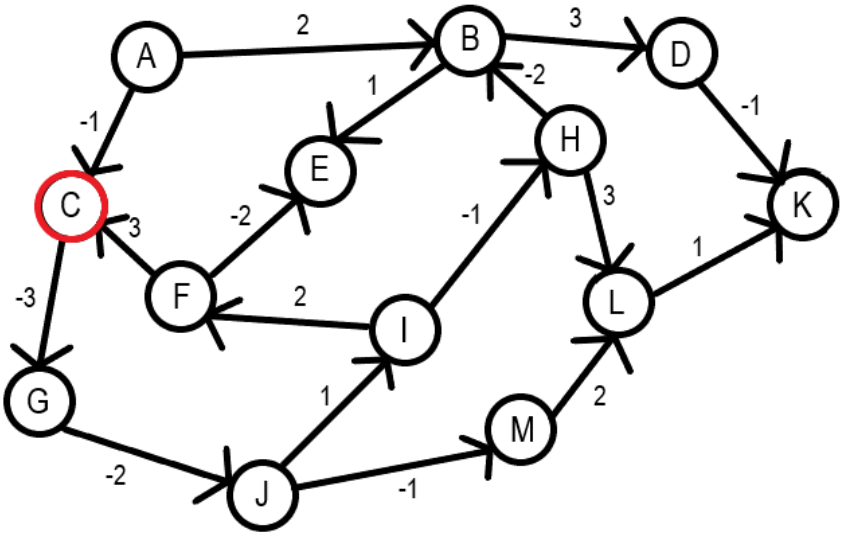


Graf č. 54 - Zkoumání uzlu B pomocí Bellman-Fordova algoritmu

Tabulka 26 - Změny vzdáleností při zkoumání uzlu B

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	∞	∞	∞	∞	∞	∞	∞	∞

Dále následuje v tabulce uzel C. Do tabulky se zapíše nejmenší vzdálenost k sousedovi, kam směřuje. (Tabulka 27)

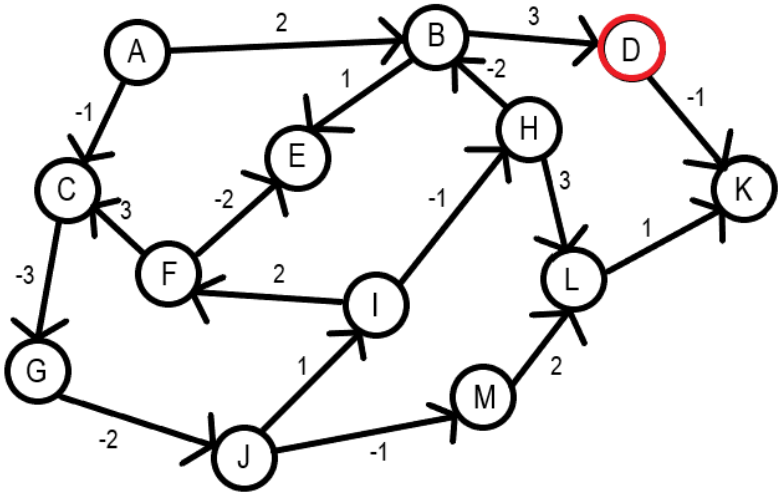


Graf č. 55 - Zkoumání uzlu C pomocí Bellman-Fordova algoritmu

Tabulka 27 - Změny vzdáleností při zkoumání uzlu C

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	∞	-4	∞	∞	∞	∞	∞	∞

Dále následuje v tabulce uzel D. Do tabulky se zapíše nejmenší vzdálenost k sousedovi, kam směřuje. (Tabulka 28)

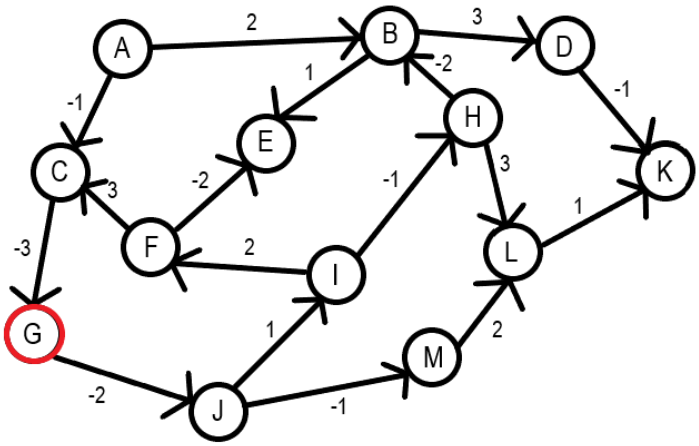


Graf č. 56 - Zkoumání uzlu D pomocí Bellman-Fordova algoritmu

Tabulka 28 - Změny vzdáleností při zkoumání uzlu D

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	∞	-4	∞	∞	∞	4	∞	∞

Dále následuje v tabulce uzel E. Nemění se nic. Přejde se na uzel F, který nemá danou vzdálenost. Pokračuje se na uzel G. Do tabulky se zapíše nejmenší vzdálenost k sousedovi, kam směřuje. (Tabulka 29)

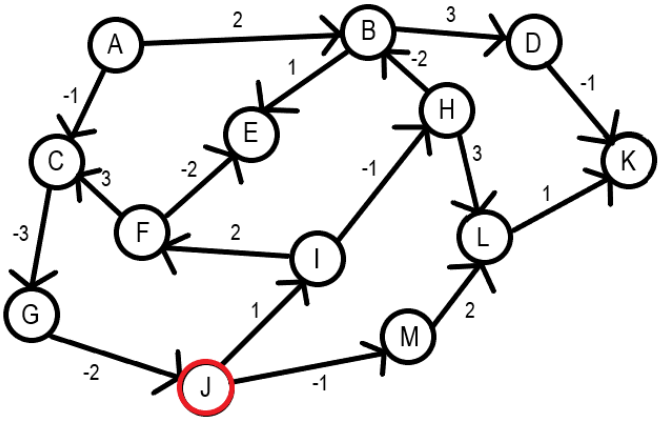


Graf č. 57 - Zkoumání uzlu G pomocí Bellman-Fordova algoritmu

Tabulka 29 - Změny vzdáleností při zkoumání uzlu G

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	∞	-4	∞	∞	-6	4	∞	∞

Dále následuje v tabulce uzel H. Nemění se nic, neboť nemá danou vzdálenost. Přejde se na uzel I, který ale také nemá danou vzdálenost. Pokračuje se na uzel J. Do tabulky se zapíšou nejmenší vzdálenosti k sousedům, kam směřuje. (Tabulka 30)

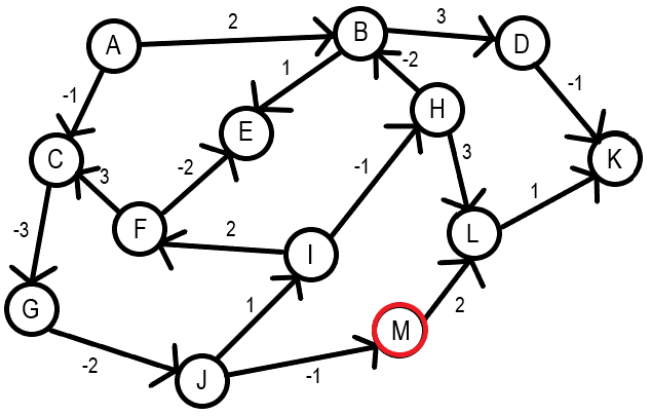


Graf č. 58 - Zkoumání uzlu J pomocí Bellman-Fordova algoritmu

Tabulka 30 - Změny vzdáleností při zkoumání uzlu J

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	∞	-4	∞	-5	-6	4	∞	-7

Dále následuje v tabulce uzel K. Ten, ale nesměřuje k žádnému sousedovi, jak jde vidět v grafu č.59. Přejde se na uzel L, který ale nemá danou vzdálenost. Pokračuje se na uzel M. Do tabulky se zapíše nejmenší vzdálenosti k sousedům, kam směřuje. (Tabulka 31)

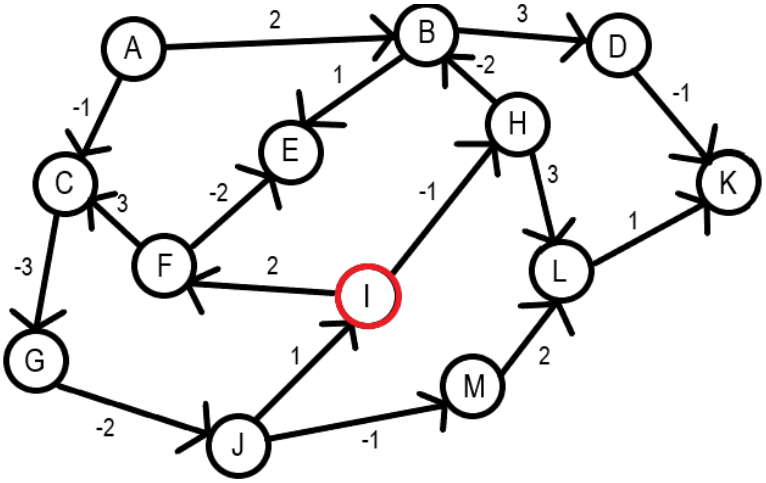


Graf č. 59 - Zkoumání uzlu M pomocí Bellman-Fordova algoritmu

Tabulka 31 - Změny vzdáleností při zkoumání uzlu M

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	∞	-4	∞	-5	-6	4	-5	-7

Právě se ukončila první iteraci. Druhá iterace probíhá ve stejném stylu jako první iterace. Zvolením uzlu A se vzdálenosti nemění. Stejné výsledky nastávají i u uzlů B, C, D, E, F, G a H. Zvolením uzlu I, ale dochází ke změně vzdáleností u uzlů F a H. Změny se zapíšou do tabulky a pokračuje se dál. (Tabulka 32)

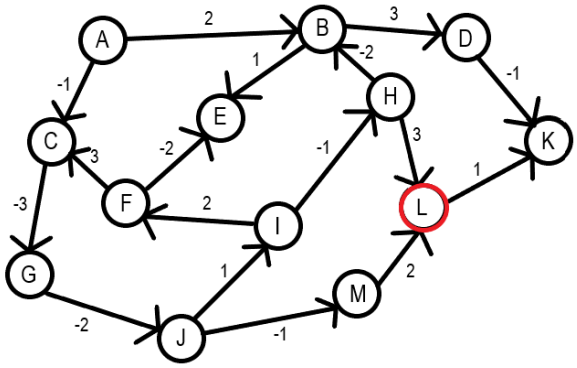


Graf č. 60 - Zkoumání uzlu I pomocí Bellman-Fordova algoritmu

Tabulka 32 - Změny vzdáleností při zkoumání uzlu I

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	-3	-4	-6	-5	-6	4	-5	-7

Při zvolení uzlu J nenastává žádné změny v tabulce vzdáleností. Stejný případ se odehrává i při zvolení uzlu K. Při zvolení uzlu L se přepíše vzdálenost uvedenou u uzlu K. Pokračuje se zvolením uzlu M, při kterém se zjistí, že vzdálenosti jeho sousedů se nijak nemění. Ukončuje se tedy druhá iterace a přechází se na třetí. (Tabulka 33)

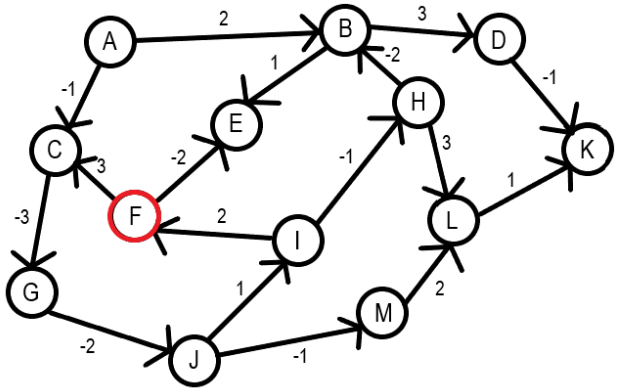


Graf č. 61 - Zkoumání uzlu L pomocí Bellman-Fordova algoritmu

Tabulka 33 - Změny vzdáleností při zkoumání uzlu L

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	3	-3	-4	-6	-5	-6	-4	-5	-7

Třetí iterace zase probíhá od začátku tabulky vzdáleností. Při jednotlivém zkoumání uzlů A až po uzel E dojde k závěru, že se vzdálenosti nijak nemění. Ale při zvolení uzlu F dochází k objevení kratší vzdálenosti do uzlu E. Tato změna se zapíše do tabulky. (Tabulka 34)

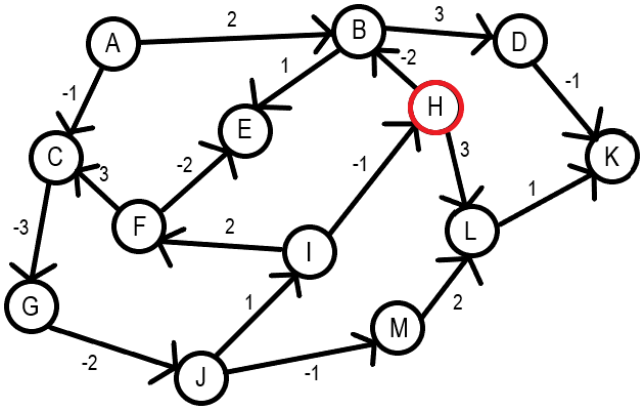


Graf č. 62 - Zkoumání uzlu F pomocí Bellman-Fordova algoritmu

Tabulka 34 - Změny vzdáleností při zkoumání uzlu F

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	2	-1	5	-5	-3	-4	-6	-5	-6	-4	-5	-7

Pokračuje se dál v tabulce. Zvolením uzlu G nedojde k žádné změně. Zvolením uzlu H dojde ke změně vzdálenosti v uzlu B. Změny se zapíší do tabulky 35.

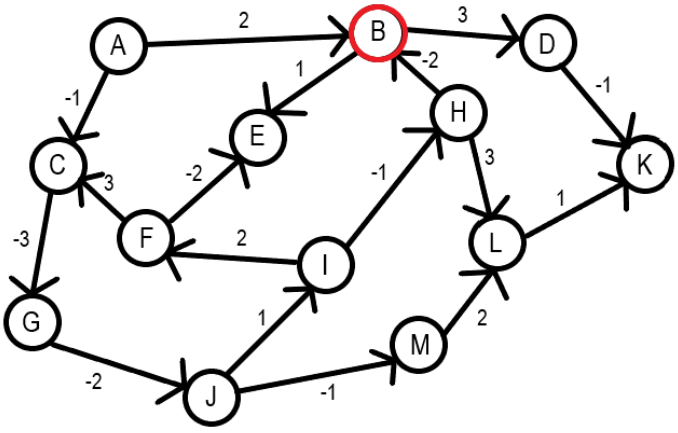


Graf č. 63 - Zkoumání uzlu H pomocí Bellman-Fordova algoritmu

Tabulka 35 - Změny vzdáleností při zkoumání uzlu H

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	-8	-1	5	-5	-3	-4	-6	-5	-6	-4	-5	-7

Pokračuje se dál v tabulce. Zvolením uzlu I nedojde k žádné změně. Stejné případy se opakují i při procházení uzlů J, K, L a M. Ukončuje se tedy třetí iterace a přechází se od začátku tabulky na čtvrtou iteraci. Prvotní zvolení uzlu A nemění nic. Změna přichází ve chvíli, kdy se přejde na kontrolu uzlu B. Mění se hodnoty vzdálenosti u uzlů D a E. Změny se zavedou do tabulky 36.

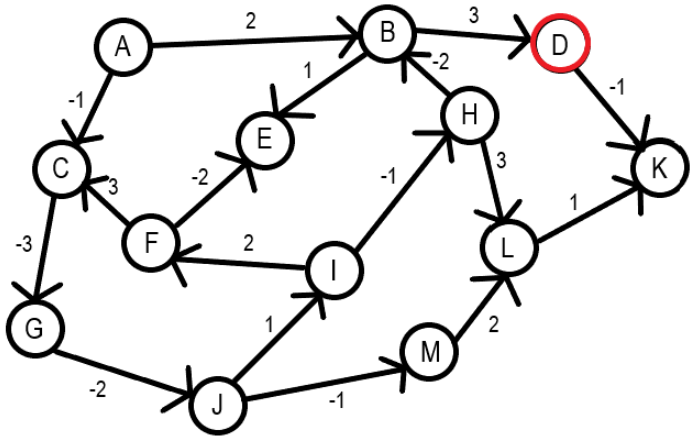


Graf č. 64 - Zkoumání uzlu B pomocí Bellman-Fordova algoritmu

Tabulka 36 - Změny vzdáleností při zkoumání uzlu B

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	-8	-1	-5	-7	-3	-4	-6	-5	-6	-4	-5	-7

Pokračuje se dál v tabulce. Zvolením uzlu C nedojde k žádné změně. Při zvolení uzlu D je nalezen velikostní rozdíl mezi původní a novou vzdáleností v uzlu K. Změna je zapsána do tabulky 37.



Graf č. 65 - Zkoumání uzlu D pomocí Bellman-Fordova algoritmu

Tabulka 37 - Změny vzdáleností při zkoumání uzlu D

Uzel	A	B	C	D	E	F	G	H	I	J	K	L	M
Vzdálenost	0	-8	-1	-5	-7	-3	-4	-6	-5	-6	-6	-5	-7

Dochází k ukončení čtvrté iterace a přechází se na začátek tabulky a začíná se pátá iterace. Během páté iterace se zjišťuje, že jednotlivým zkoumáním uzlů z tabulky nedochází k žádnému přepisu vzdáleností. Pokud po konci iterace nebyla zapsána žádná nová hodnota, lze prohlásit algoritmus za ukončený. Jak bylo zmíněno v teoretické části o Bellman-Fordově algoritmu, algoritmus musí skončit do n-té fáze iteracích. Pro tento příklad tedy platí, že musí skončit do 13. fáze iterace. Algoritmus na tomto grafu skončil v 5. fázi, z čehož vyplývá, že podmínka byla splněna.

Vstup do programové realizace:

graph = {'A': {'B': 2, 'C': -1}, 'B': {'D': 3, 'E': 1}, 'C': {'G': -3}, 'D': {'K': -1}, 'E': {}, 'F': {'C': 3, 'E': -2}, 'G': {'J': -2}, 'H': {'B': -2, 'L': 3}, 'I': {'F': 2, 'H': -1}, 'J': {'M': -1, 'I': 1}, 'K': {}, 'L': {'K': 1}, 'M': {'L': 2}}

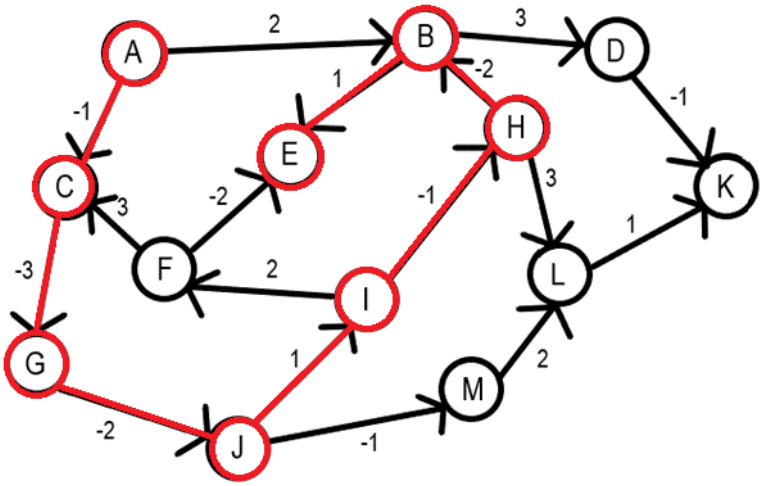
bellman_ford(graph, 'A')

Výstup z programové realizace:

A: 0 B: -8 C: -1

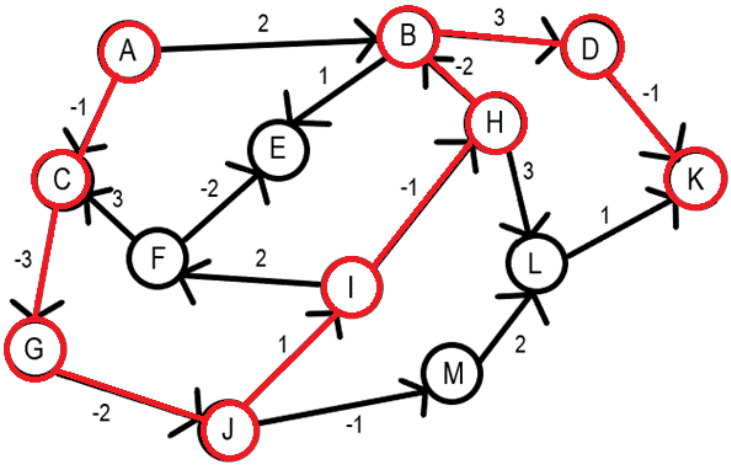
D: -5	G: -4	J: -6
E: -7	H: -6	K: -6
F: -3	I: -5	L: -5
M: -7		

Tabulka 37 uvádí minimální destinace z bodu A do všech ostatních bodů zadaného grafu. Příkladem je cesta z A do E přes uzly A, C, G, J, I, H, B, E:



Graf č. 66 – Cesta z uzlu A do uzlu E

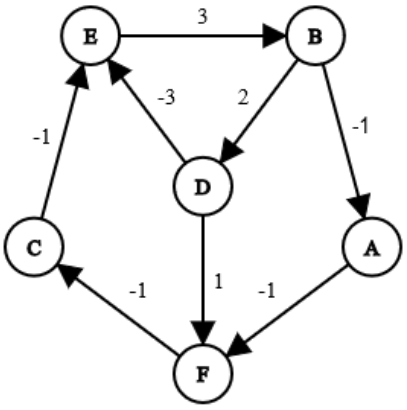
Překvapivé ohodnocení ukazuje trajektorie z uzlu A do uzlu K přes uzly A, C, G, J, I, H, B, D a K:



Graf č. 67 - Cesta z uzlu A do uzlu K

7.2.2 Příklad 2

Je uvažován následující orientované graf č.68 s obecně ohodnocenými hranami. Úkolem bude ověření, jestli následující graf obsahuje negativní zacyklení. K nalezení negativního zacyklení se využije Bellman-Fordův algoritmus. Jako počáteční uzel se zvolí uzel A a přidá se mu hodnotu 0 do tabulky. Ostatním uzlům se přidá hodnota ∞. (Tabulka 38). V grafu se nachází 6 uzlů z čehož lze vyvodit, že pokud algoritmus bude pokračovat po páté iteraci, tak se v grafu nachází negativní zacyklení.

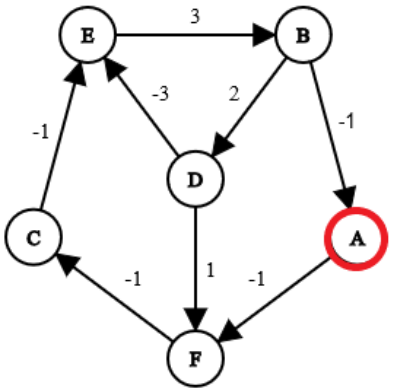


Graf č. 68 pro příklad 2 aplikace Bellman-Fordova algoritmu

Tabulka 38 – Tabulka vzdáleností od počátečního uzlu

uzel	A	B	C	D	E	F
vzdálenost	0	∞	∞	∞	∞	∞

V první iterace se začíná od začátku tabulky. Zkoumáním uzlu A se nalezne vzdálenost pro uzel F. (Tabulka 39)

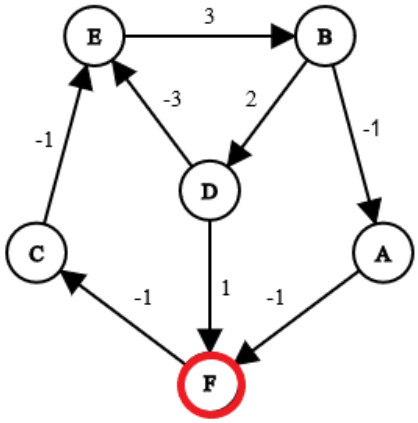


Graf č. 69 - Zkoumání uzlu A pomocí Bellman-Fordova algoritmu

Tabulka 39 - Změny vzdáleností při zkoumání uzlu A

uzel	A	B	C	D	E	F
vzdálenost	0	∞	∞	∞	∞	-1

Zkoumáním uzlu B se nenalezne žádná menší hodnota, neboť uzel B má hodnotu ∞. Přejde se v tabulce na další uzel. Stejným případem jsou uzly C, D a E. Až během zkoumání uzlu F se nalezne hodnota pro uzel C. Ukončuje se první iteraci. (Tabulka 40)

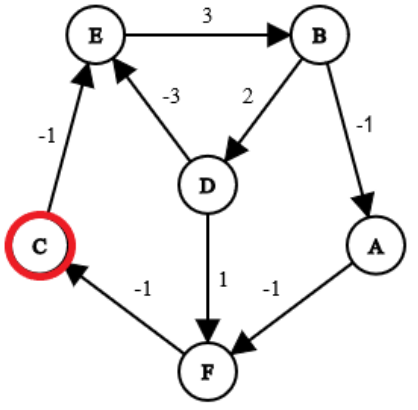


Graf č. 70 - Zkoumání uzlu F pomocí Bellman-Fordova algoritmu

Tabulka 40 - Změny vzdáleností při zkoumání uzlu F

uzel	A	B	C	D	E	F
vzdálenost	0	∞	-2	∞	∞	-1

V druhé iteraci se začíná znovu od začátku tabulky. Zkoumáním uzlu A se nenalezne žádná menší hodnota. Ani zkoumáním uzlu B se nenalézají žádné hodnoty. Ale zkoumáním uzlu C se nalezne hodnota pro uzel E. (Tabulka 41)

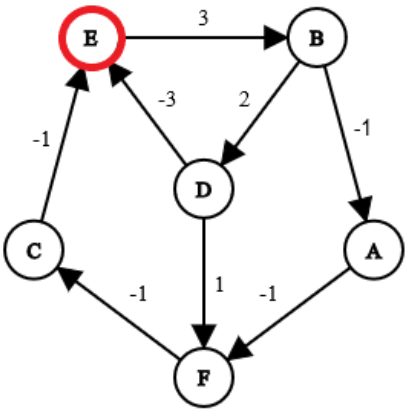


Graf č. 71 - Zkoumání uzlu C pomocí Bellman-Fordova algoritmu

Tabulka 41 - Změny vzdáleností při zkoumání uzlu C

uzel	A	B	C	D	E	F
vzdálenost	0	∞	-2	∞	-3	-1

Zkoumáním uzlu D se nenalézá žádná nejmenší hodnota. Při kontrole uzlu E se dojde k hodnotě pro uzel B. Následně zkoumáním uzlu F nedochází k žádným změnám. Ukončuje se tedy druhá iterace. (Tabulka 42)

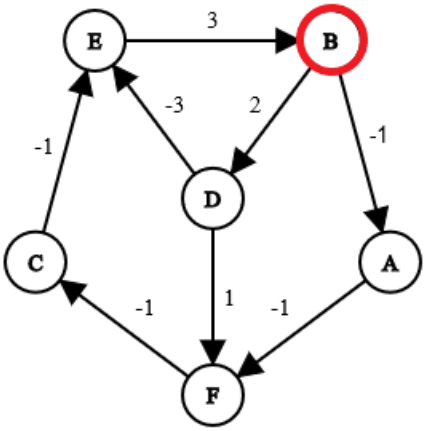


Graf č. 72 - Zkoumání uzlu E pomocí Bellman-Fordova algoritmu

Tabulka 42 - Změny vzdáleností při zkoumání uzlu E

uzel	A	B	C	D	E	F
vzdálenost	0	0	-2	∞	-3	-1

Začíná třetí iterace. Zkoumáním uzlu A se nezjistí žádná nová hodnota. Zkoumáním uzlu B se najde nejnižší hodnota pro uzly A a D. Zkoumáním uzlu C se zase nezjistí žádná nová hodnota. Stejného výsledku dochází i při zkoumání uzlů D, E a F. Může se ukončit třetí iteraci. (Tabulka 43)

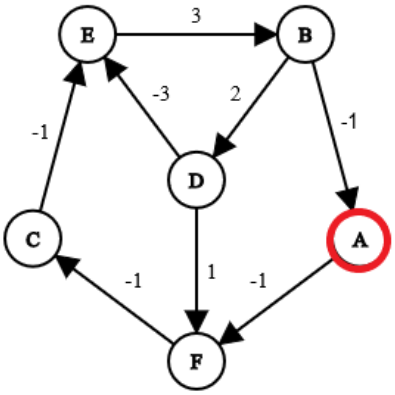


Graf č. 73 - Zkoumání uzlu B pomocí Bellman-Fordova algoritmu

Tabulka 43 - Změny vzdáleností při zkoumání uzlu B

uzel	A	B	C	D	E	F
vzdálenost	-1	0	-2	2	-3	-1

V čtvrté iteraci zkoumáním uzlu A se zjistí nejmenší hodnota pro uzel F. Pak zkoumáním uzlu B nedojde k žádné změně pro jeho sousedy. Stejným případem je i zkoumání uzlů C, D a E. (Tabulka 44)

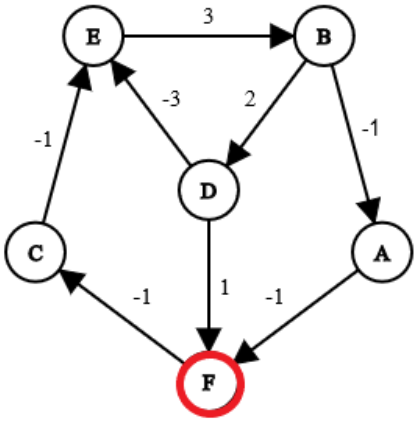


Graf č. 74 - Zkoumání uzlu A pomocí Bellman-Fordova algoritmu

Tabulka 44 - - Změny vzdáleností při zkoumání uzlu A

uzel	A	B	C	D	E	F
vzdálenost	-1	0	-2	2	-3	-2

Při zkoumání uzlu F se najde nejnižší hodnota pro uzel C. Může se ukončit čtvrtá iterace. (Tabulka 45)

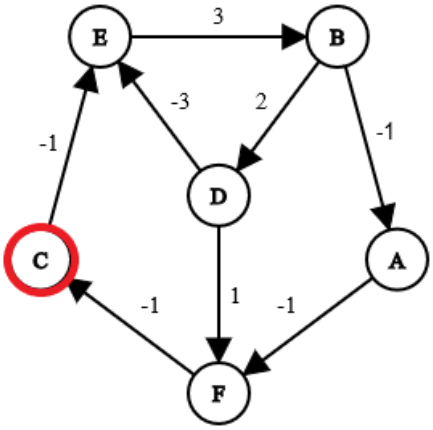


Graf č. 75 - Zkoumání uzlu F pomocí Bellman-Fordova algoritmu

Tabulka 45 - Změny vzdáleností při zkoumání uzlu F

uzel	A	B	C	D	E	F
vzdálenost	-1	0	-3	2	-3	-2

Začíná se zase od začátku tabulky. Zkoumáním uzlu A se nezískává žádná nová hodnota pro jeho sousední uzel. Zkoumáním uzlu B se dochází ke stejnému závěru. Při zkoumání uzlu C se nalézá nová hodnota pro uzel E. (Tabulka 46)

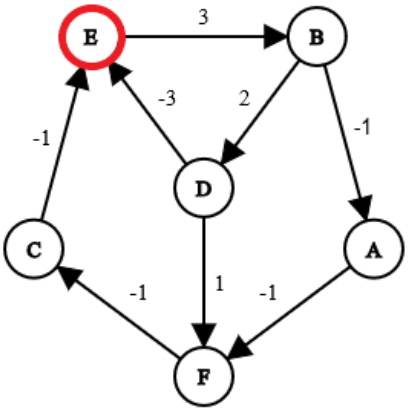


Graf č. 76 - Zkoumání uzlu C pomocí Bellman-Fordova algoritmu

Tabulka 46 - Změny vzdáleností při zkoumání uzlu C

uzel	A	B	C	D	E	F
vzdálenost	-1	0	-3	2	-4	-2

Zkoumáním uzlu D se nezískává žádná nová hodnota. Ale zkoumáním uzlu E se najde nová hodnota pro uzel B. Dále zkoumání uzlu F se nezískává nic. Ukončuje se tak poslední iterace. (Tabulka 47)



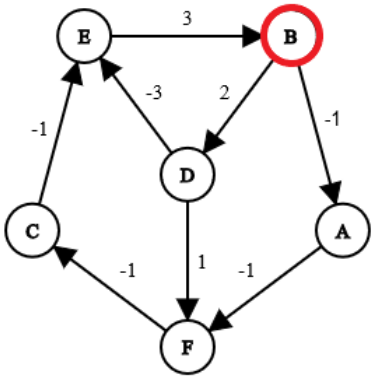
Graf č. 77 - Zkoumání uzlu E pomocí Bellman-Fordova algoritmu

Tabulka 47 - Změny vzdáleností při zkoumání uzlu E

uzel	A	B	C	D	E	F
vzdálenost	-1	-1	-3	2	-4	-2

Jelikož algoritmus v tomto grafu došel maximálního počtu iterací, jenž je dána podmínkou o Bellman-Fordově algoritmu, je potřeba provést ještě jednu iteraci pro kontrolu. Jestli během této iterace dojde k přepisu vzdáleností v tabulce lze konstatovat, že v grafu se nachází negativní zacyklení.

Kontrolní iterace začne od začátku tabulky 48. Zkoumáním uzlu A se nedojde k žádné nejmenší hodnotě. Zkoumáním uzlu B se zjišťuje, že dochází k nálezu nové nejmenší vzdálenosti pro uzly A a D, než jsou původní. V kontrolní iteraci došlo k nalezení nových hodnot, z čehož lze vyvodit, že graf obsahuje negativní zacyklení. Z toho také vyplývá, že nelze najít nejkratší cesty z počátečního uzlu A k ostatním za pomocí Bellman-Fordova algoritmu.



Graf č. 78 – Kontrolní iterace Bellman-Fordova algoritmu

Tabulka 48 - Změny vzdáleností při kontrolní iteraci

uzel	A	B	C	D	E	F
vzdálenost	-2	-1	-3	1	-4	-2

Vstup do programové realizace:

graph = {'A': {'F': -1}, 'B': {'D': 2, 'A': -1}, 'C': {'E': -1}, 'D': {'E': -3, 'F': 1}, 'E': {'B': 3}, 'F': {'C': -1}}

Výstup z programové realizace:

Negativní zacyklení

Poskytuje negativní zacyklení, což znamená, že úloha nemá konečné řešení, ale je zacyklena.

8 FLOYD-WARSHALLŮV ALGORITMUS

Pseudokód:

Vstup: Matice délek hran D^0

Pro k = 0, . . . , n − 1:

Pro i = 1, . . . , n:

Pro j = 1, . . . , n:

$$D_{ij}^{k+1} \leftarrow \min\left(D_{ij}^{k}, D_{i,k+1}^{k} + D_{k+1,j}^{k}\right)$$

Výstup: Matice vzdáleností D

8.1 Programová realizace

Programová realizace v jazyce Python:

```
INF = float('infinity')
```

Vytvoření konstanty pro nekonečno.

```
def floydWarshall(graph):
    V = len(graph)
    dist = graph.copy()
    for k in range(V):
        for i in range(V):
            for j in range(V):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Algoritmus je realizován ve třech for cyklech. Počet iterací pro cyklus je dán počtem uzlů v grafu. Zkoumanému uzlu se zapisuje pouze nejmenší hodnota.

```
    printSolution(dist)


def printsolution(dist):
    V = len(graph)
    print("")
```

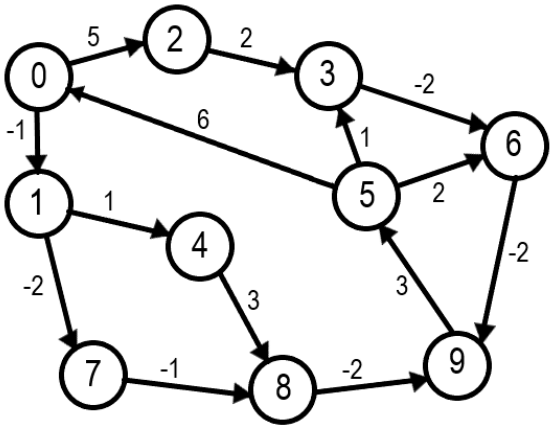
```
for i in range(V):
    for j in range(V):
        print((dist[i][j]), end=" ")
        if j == V - 1:
            print("")
```

Funkce pro výpis výsledku.

8.2 Příklady

8.2.1 Příklad 1

Je uvažován následující orientovaný graf sítí o 10 vrcholech (graf č.79). Orientovaný graf se převede do maticového tvaru sousednosti (Tabulka 49). Pak na matici se použije Floyd-Warshallův algoritmus pro nalezení všech nejmenších cest mezi jednotlivými dvojicemi uzlů v grafu.



Graf č. 79 pro příklad 1 aplikace Floyd-Warshallova algoritmu

Tabulka 49 - Matice sousednosti tabulkovou formou s názvem DIST

	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	∞	∞	∞	∞	∞	∞	∞
1	∞	0	∞	∞	1	∞	∞	-2	∞	∞
2	∞	∞	0	2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	∞	∞	1	∞	0	2	∞	∞	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞

8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞	∞	∞	∞	∞	3	∞	∞	∞	0

Matici, do které se zanesly vzdálenosti mezi danými uzly s pojmenuje jako DIST pro tento příklad. Zároveň se použije následující rovnice, která se aplikuje na matici:

$$DIST[i][j] > DIST[i][k] + DIST[k][j]$$

Veličiny *k*, *i* a *j* patří do intervalu <0;9> $\in \mathbf{Z}$ pro uvedený příklad. Postupným procházením iterací cykly se budou tyto veličiny dosazovat do výše uvedené rovnice jako souřadnice do matice. Dosazením souřadnic pak lze získat odpovídající hodnoty vzdálenosti mezi jednotlivými uzly.

Pro lepší pochopení lze uvést, že při $i = 0$ a $j = 1$ pro DIST[i][j] se získá DIST[0][1] = -1 jenž je zároveň hodnota vzdálenosti hrany z uzlu 0 do uzlu 1.

Pro lepší přehled se vytvoří pomocná tabulka do, které se vypisují všechny výsledky po provedení $DIST[i][k] + DIST[k][j]$.

Pro následující tabulku platí, že k = 0, i je rovno hodnotě čísla řádku a j rovno hodnotě čísla sloupce. Hodnoty do tabulky budou dosazeny z rovnice $DIST[i][k] + DIST[k][j]$.

Tabulka 50 – pomDIST pro k = 0

k = 0	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	0	-1	5	∞	∞	∞	∞	∞	∞	∞
1	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
2	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
3	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
4	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
5	6	5	11	∞+6	∞+6	∞+6	∞+6	∞+6	∞+6	∞+6
6	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
7	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
8	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞
9	∞	∞-1	∞+5	∞	∞	∞	∞	∞	∞	∞

Př.

$$DIST[0][0] + DIST[0][1] = -1$$

Nyní se porovnají hodnoty matice *DIST* s pomocnou maticí pro *k=0*. Pokud dojde ke splnění podmínky, že *DIST[x][y]> pom_DIST[x][y]* pak je hodnota v *DIST[x][y]* přepsána za hodnotu v *pom_DIST[x][y].* Změny byli zapsány do matice *DIST:*

Tabulka 51 – DIST po srovnání s pomDIST pro $k = 0$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	∞	∞	∞	∞	∞	∞	∞
1	∞	0	∞	∞	1	∞	∞	-2	∞	∞
2	∞	∞	0	2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	∞	0	2	∞	∞	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞	∞	∞	∞	∞	3	∞	∞	∞	0

Pokračuje se na iteraci k=1. Hodnoty se berou z upravené matice *DIST*.

Tabulka 52 - pomDIST pro $k = 1$

k = 1	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞-1	-1	∞-1	∞-1	0	∞-1	∞-1	-3	∞-1	∞-1
1	∞	0	∞	∞	∞+1	∞	∞	∞-2	∞	∞
2	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞
3	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞
4	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞
5	∞+5	5	∞+5	∞+5	6	∞+5	∞+5	3	∞+5	∞+5
6	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞
7	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞
8	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞
9	∞	∞	∞	∞	∞+1	∞	∞	∞-2	∞	∞

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 53 - DIST po porovnání s pomDIST pro $k = 1$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	∞	0	∞	∞	-3	∞	∞
1	∞	0	∞	∞	1	∞	∞	-2	∞	∞
2	∞	∞	0	2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	2	3	∞	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞	∞	∞	∞	∞	3	∞	∞	∞	0

Pokračuje se na iteraci k=2. Hodnoty se berou z upravené matice *DIST*.

Tabulka 54 - pomDIST pro $k = 2$

k = 2	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞+5	∞+5	5	7	∞+5	∞+5	∞+5	∞+5	∞+5	∞+5
1	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞
2	∞	∞	0	2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞
5	∞+11	∞+11	11	13	∞+11	∞+11	∞+11	∞+11	∞+11	∞+11
6	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞
8	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞
9	∞	∞	∞	∞+2	∞	∞	∞	∞	∞	∞

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 55 - DIST po porovnání s pomDIST pro $k = 2$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	∞	-3	∞	∞
1	∞	0	∞	∞	1	∞	∞	-2	∞	∞
2	∞	∞	0	2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	2	3	∞	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞	∞	∞	∞	∞	3	∞	∞	∞	0

Pokračuje se na iteraci k=3. Hodnoty se berou z upravené matice *DIST*.

Tabulka 56 - pomDIST pro $k = 3$

k = 3	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞+7	∞+7	∞+7	7	∞+7	∞+7	5	∞+7	∞+7	∞+7
1	∞	∞	∞	∞	∞	∞	∞-2	∞	∞	∞
2	∞+2	∞+2	∞+2	2	∞+2	∞+2	0	∞+2	∞+2	∞+2
3	∞	∞	∞	0	∞	∞	∞-2	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞-2	∞	∞	∞
5	∞+1	∞+1	∞+1	1	∞+1	∞+1	-1	∞+1	∞+1	∞+1
6	∞	∞	∞	∞	∞	∞	∞-2	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	∞-2	∞	∞	∞
8	∞	∞	∞	∞	∞	∞	∞-2	∞	∞	∞
9	∞	∞	∞	∞	∞	∞	∞-2	∞	∞	∞

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 57 - DIST po porovnání s pomDIST pro $k = 3$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	5	-3	∞	∞
1	∞	0	∞	∞	1	∞	∞	-2	∞	∞
2	∞	∞	0	2	∞	∞	0	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	-1	3	∞	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞	∞	∞	∞	∞	3	∞	∞	∞	0

Pokračuje se na iteraci k=4. Hodnoty se berou z upravené matice *DIST*.

Tabulka 58 - pomDIST pro $k = 4$

k = 4	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞	∞	∞	∞	0	∞	∞	∞	3	∞
1	∞+1	∞+1	∞+1	∞+1	1	∞+1	∞+1	∞+1	4	∞+1
2	∞	∞	∞	∞	∞	∞	∞	∞	∞+3	∞
3	∞	∞	∞	∞	∞	∞	∞	∞	∞+3	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	∞+6	∞+6	∞+6	∞+6	6	∞+6	∞+6	∞+6	9	∞+6
6	∞	∞	∞	∞	∞	∞	∞	∞	∞+3	∞
7	∞	∞	∞	∞	∞	∞	∞	∞	∞+3	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	∞+3	∞
9	∞	∞	∞	∞	∞	∞	∞	∞	∞+3	∞

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 59 - DIST po porovnání s pomDIST pro $k = 4$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	5	-3	3	∞
1	∞	0	∞	∞	1	∞	∞	-2	4	∞
2	∞	∞	0	2	∞	∞	0	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	-1	3	9	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞	∞	∞	∞	∞	3	∞	∞	∞	0

Pokračuje se na iteraci k=5. Hodnoty se berou z upravené matice *DIST*.

Tabulka 60 - pomDIST pro $k = 5$

k = 5	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
1	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
2	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
3	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
4	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
5	6	5	11	1	6	0	-1	3	9	∞
6	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
7	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
8	∞+6	∞+5	∞+11	∞+1	∞+6	∞	∞-1	∞+7	∞+9	∞
9	9	8	14	4	9	3	2	6	12	∞+3

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 61 - DIST po porovnání s pomDIST pro $k = 5$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	5	-3	3	∞
1	∞	0	∞	∞	1	∞	∞	-2	4	∞
2	∞	∞	0	2	∞	∞	0	∞	∞	∞
3	∞	∞	∞	0	∞	∞	-2	∞	∞	∞
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	-1	3	9	∞
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	9	8	14	4	9	3	2	6	12	0

Pokračuje se na iteraci k=6. Hodnoty se berou z upravené matice *DIST*.

Tabulka 62 - pomDIST pro $k = 6$

k = 6	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞+5	∞+5	∞+5	∞+5	∞+5	∞+5	5	∞+5	∞+5	3
1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞-2
2	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
3	∞+2	∞+2	∞+2	∞+2	∞+2	∞+2	-2	∞+2	∞+2	-4
4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞-2
5	∞-1	∞-1	∞-1	∞-1	∞-1	∞-1	-1	∞-1	∞-1	-3
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞-2
8	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞-2
9	∞+2	∞+2	∞+2	∞+2	∞+2	∞+2	2	∞+2	∞+2	0

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 63 - DIST po porovnání s pomDIST pro $k = 6$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	5	-3	3	3
1	∞	0	∞	∞	1	∞	∞	-2	4	∞
2	∞	∞	0	2	∞	∞	0	∞	∞	-2
3	∞	∞	∞	0	∞	∞	-2	∞	∞	-4
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	-1	3	9	-3
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	9	8	14	4	9	3	2	6	12	0

Pokračuje se na iteraci k=7. Hodnoty se berou z upravené matice *DIST*.

Tabulka 64 - pomDIST pro $k = 7$

k = 7	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞-3	∞-3	∞-3	∞-3	∞-3	∞-3	∞-3	-3	-4	∞-3
1	∞-2	∞-2	∞-2	∞-2	∞-2	∞-2	∞-2	-2	-3	∞-2
2	∞	∞	∞	∞	∞	∞	∞	∞	∞-1	∞
3	∞	∞	∞	∞	∞	∞	∞	∞	∞-1	∞
4	∞	∞	∞	∞	∞	∞	∞	∞	∞-1	∞
5	∞+3	∞+3	∞+3	∞+3	∞+3	∞+3	∞+3	3	2	∞+3
6	∞	∞	∞	∞	∞	∞	∞	∞	∞-1	∞
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	∞-1	∞
9	∞+6	∞+6	∞+6	∞+6	∞+6	∞+6	∞+6	6	5	∞+6

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 65 - DIST po porovnání s pomDIST pro $k = 7$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	5	-3	-4	3
1	∞	0	∞	∞	1	∞	∞	-2	-3	∞
2	∞	∞	0	2	∞	∞	0	∞	∞	-2
3	∞	∞	∞	0	∞	∞	-2	∞	∞	-4
4	∞	∞	∞	∞	0	∞	∞	∞	3	∞
5	6	5	11	1	6	0	-1	3	2	-3
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	9	8	14	4	9	3	2	6	5	0

Pokračuje se na iteraci *k*=8. Hodnoty se berou z upravené matice *DIST*.

Tabulka 66 - pomDIST pro $k = 8$

k = 8	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	∞-4	∞-4	∞-4	∞-4	∞-4	∞-4	∞-4	∞-4	-4	-6
1	∞-3	∞-3	∞-3	∞-3	∞-3	∞-3	∞-3	∞-3	-3	-5
2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	∞+3	∞+3	∞+3	∞+3	∞+3	∞+3	∞+3	∞+3	3	1
5	∞+2	∞+2	∞+2	∞+2	∞+2	∞+2	∞+2	∞+2	2	0
6	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
7	∞-1	∞-1	∞-1	∞-1	∞-1	∞-1	∞-1	∞-1	-1	-3
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	∞+5	∞+5	∞+5	∞+5	∞+5	∞+5	∞+5	∞+5	5	3

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 67 - DIST po porovnání s pomDIST pro $k = 8$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	7	0	∞	5	-3	-4	-6
1	∞	0	∞	∞	1	∞	∞	-2	-3	-5
2	∞	∞	0	2	∞	∞	0	∞	∞	-2
3	∞	∞	∞	0	∞	∞	-2	∞	∞	-4
4	∞	∞	∞	∞	0	∞	∞	∞	3	1
5	6	5	11	1	6	0	-1	3	2	-3
6	∞	∞	∞	∞	∞	∞	0	∞	∞	-2
7	∞	∞	∞	∞	∞	∞	∞	0	-1	-3
8	∞	∞	∞	∞	∞	∞	∞	∞	0	-2
9	9	8	14	4	9	3	2	6	5	0

Pokračuje se na poslední iteraci k=9. Hodnoty se berou z upravené matice *DIST*.

Tabulka 68 - pomDIST pro $k = 9$

k = 9	0 (j)	1	2	3	4	5	6	7	8	9
0 (i)	3	2	8	-2	3	-3	-4	0	-1	-6
1	4	3	9	-1	4	-2	-3	1	0	-5
2	7	6	12	2	7	1	0	4	3	-2
3	5	4	10	0	5	-1	-2	2	1	-4
4	10	9	15	5	10	4	3	7	6	1
5	6	5	11	1	6	0	-1	3	2	-3
6	7	6	12	2	7	1	0	4	3	-2
7	6	5	11	1	6	0	-1	3	2	-3
8	7	6	12	2	7	1	0	4	3	-2
9	9	8	14	4	9	3	2	6	5	0

Matice se potom porovnají a hodnoty přepíšou, pokud je splněna podmínka.

Tabulka 69 - DIST po porovnání s pomDIST pro $k = 9$

DIST	0	1	2	3	4	5	6	7	8	9
0	0	-1	5	-2	0	-3	-4	-3	-4	-6
1	4	0	9	-1	1	-2	-3	-2	-3	-5
2	7	6	0	2	7	1	0	4	3	-2
3	5	4	10	0	5	-1	-2	2	1	-4
4	10	9	15	5	0	4	3	7	3	1
5	6	5	11	1	6	0	-1	3	2	-3
6	7	6	12	2	7	1	0	4	3	-2
7	6	5	11	1	6	0	-1	0	-1	-3
8	7	6	12	2	7	1	0	4	0	-2
9	9	8	14	4	9	3	2	6	5	0

Ukončuje se devátá iteraci. Podle definice Floyd-Warshallova algoritmu výsledná matice sousednosti DIST obsahuje nejmenší cesty mezi jednotlivými dvojicemi uzlů v grafu.

Vstup do programové realizace:

```
graph = [[0, -1, 5, INF, INF, INF, INF, INF, INF, INF],
         [INF, 0, INF, INF, 1, INF, INF, -2, INF, INF],
         [INF, INF, 0, 2, INF, INF, INF, INF, INF, INF],
         [INF, INF, INF, 0, INF, INF, -2, INF, INF, INF],
         [INF, INF, INF, INF, 0, INF, INF, INF, 3, INF],
         [6, INF, INF, 1, INF, 0, 2, INF, INF, INF],
         [INF, INF, INF, INF, INF, INF, 0, INF, INF, -2],
         [INF, INF, INF, INF, INF, INF, INF, 0, -1, INF],
         [INF, INF, INF, INF, INF, INF, INF, INF, 0, -2],
         [INF, INF, INF, INF, INF, 3, INF, INF, INF, 0]]
```

Výstup z programové realizace:

```
0  -1 5 -2 0 -3 -4 -3 -4 -6
4   0  9 -1 1 -2 -3 -2 -3 -5
7   6  0 2 7 1 0 4 3 -2
5   4  10  0 5 -1 -2 2 1 -4
10 9 15 5 0 4 3 7 3 1
```

6 5 11 1 6 0 -1 3 2 -3

7 6 12 2 7 1 0 4 3 -2

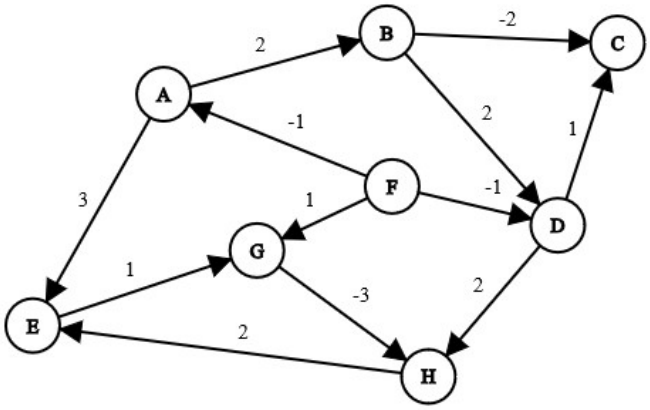
6 5 11 1 6 0 -1 0 -1 -3

7 6 12 2 7 1 0 4 0 -2

9 8 14 4 9 3 2 6 5 0

8.2.2 Příklad 2

Nyní je uvažován následující orientovaný graf o 8 uzlech (graf č.80). Pomocí Floyd-Warshallova algoritmu se najdou všechny možné nejkratší cesty mezi jeho jednotlivými dvojicemi uzlů. Daný síťový graf se převede do tvaru tabulky 70 vyjadřující sousednost jednotlivých uzlů. Tabulka 70 se bude jmenovat DIST pro tento případ.



Graf č. 80 pro příklad 2 aplikace Floyd-Warshallova algoritmu

Tabulka 70 – Matice sousednosti tabulkovou formou s názvem DIST

uzly	A	B	C	D	E	F	G	H
A	0	2	∞	∞	3	∞	∞	∞
B	∞	0	-2	2	∞	∞	∞	∞
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	∞	∞	-1	∞	0	∞	∞
G	∞	∞	∞	∞	∞	1	0	-3
H	∞	∞	∞	∞	2	∞	∞	0

Nyní se vytvoří pomocná tabulka podle $DIST[i][k] + DIST[k][j]$ kde proměnné veličiny *k*, *i* a *j* jsou z intervalu <0;7> $\in \mathbb{Z}$ pro tento uvedený příklad. Začne se tabulkou pro $k = 0$.

Tabulka 71 – pomDIST pro $k = 0$

k = 0		j							
		0	1	2	3	4	5	6	7
i	0	0	2	∞	∞	3	∞	∞	∞
	1	∞	∞+2	∞	∞	∞+3	∞	∞	∞
	2	∞	∞+2	∞	∞	∞+3	∞	∞	∞
	3	∞	∞+2	∞	∞	∞+3	∞	∞	∞
	4	∞	∞+2	∞	∞	∞+3	∞	∞	∞
	5	-1	1	∞-1	∞-1	2	∞-1	∞-1	∞-1
	6	∞	∞+2	∞	∞	∞+3	∞	∞	∞
	7	∞	∞+2	∞	∞	∞+3	∞	∞	∞

Nyní tabulka pro $k = 0$ se porovná s DIST tabulkou. Pokud hodnota v pomocné tabulce je menší než v DIST, přepíše se hodnota v tabulce DIST za hodnotu v tabulce pro $k = 0$. Přepsané hodnoty jsou zvýrazněny. (Tabulka 72)

Tabulka 72 – DIST po porovnání s pomDIST pro $k = 0$

uzly	A	B	C	D	E	F	G	H
A	0	2	∞	∞	3	∞	∞	∞
B	∞	0	-2	2	∞	∞	∞	∞
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	1	∞	-1	2	0	∞	∞
G	∞	∞	∞	∞	∞	1	0	-3
H	∞	∞	∞	∞	2	∞	∞	0

Pokračuje se na tabulku $k = 1$:

Tabulka 73 - pomDIST pro $k = 1$

k = 1		j							
		0	1	2	3	4	5	6	7
i	0	∞+2	2	0	4	∞+2	∞+2	∞+2	∞+2
	1	∞	0	-2	2	∞	∞	∞	∞
	2	∞	∞	∞-2	∞+2	∞	∞	∞	∞
	3	∞	∞	∞-2	∞+2	∞	∞	∞	∞
	4	∞	∞	∞-2	∞+2	∞	∞	∞	∞
	5	∞+1	1	-1	3	∞+1	∞+1	∞+1	∞+1
	6	∞	∞	∞-2	∞+2	∞	∞	∞	∞
	7	∞	∞	∞-2	∞+2	∞	∞	∞	∞

Porovnání s DIST:

Tabulka 74 - DIST po porovnání s pomDIST pro $k = 1$

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	∞	∞	∞
B	∞	0	-2	2	∞	∞	∞	∞
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	1	-1	-1	2	0	∞	∞
G	∞	∞	∞	∞	∞	1	0	-3
H	∞	∞	∞	∞	2	∞	∞	0

Tabulka k = 2:

Tabulka 75 - pomDIST pro $k = 2$

k = 2		j							
		0	1	2	3	4	5	6	7
i	0	∞	∞	0	∞	∞	∞	∞	∞
	1	∞-2	∞-2	-2	∞-2	∞-2	∞-2	∞-2	∞-2
	2	∞	∞	0	∞	∞	∞	∞	∞
	3	∞+1	∞+1	1	∞+1	∞+1	∞+1	∞+1	∞+1
	4	∞	∞	∞	∞	∞	∞	∞	∞
	5	∞-1	∞-1	-1	∞-1	∞-1	∞-1	∞-1	∞-1
	6	∞	∞	∞	∞	∞	∞	∞	∞
	7	∞	∞	∞	∞	∞	∞	∞	∞

Porovnání s DIST:

Tabulka 76 - DIST po porovnání s pomDIST pro $k = 2$

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	∞	∞	∞
B	∞	0	-2	2	∞	∞	∞	∞
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	1	-1	-1	2	0	∞	∞
G	∞	∞	∞	∞	∞	1	0	-3
H	∞	∞	∞	∞	2	∞	∞	0

Tabulka k = 3:

Tabulka 77 - pomDIST pro $k = 3$

k = 3		j							
		0	1	2	3	4	5	6	7
i	0	∞+4	∞+4	5	4	∞+4	∞+4	∞+4	6
	1	∞+2	∞+2	3	2	∞+2	∞+2	∞+2	4

	2	∞	∞	∞+1	∞	∞	∞	∞	∞+2
	3	∞	∞	1	0	∞	∞	∞	2
	4	∞	∞	∞+1	∞	∞	∞	∞	∞+2
	5	∞-1	∞-1	0	-1	∞-1	∞-1	∞-1	1
	6	∞	∞	∞+1	∞	∞	∞	∞	∞+2
	7	∞	∞	∞+1	∞	∞	∞	∞	∞+2

Porovnání s DIST:

Tabulka 78 - DIST po porovnání s pomDIST pro $k = 3$

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	∞	∞	6
B	∞	0	-2	2	∞	∞	∞	4
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	1	-1	-1	2	0	∞	1
G	∞	∞	∞	∞	∞	1	0	-3
H	∞	∞	∞	∞	2	∞	∞	0

Tabulka pro k = 4:

Tabulka 79 - pomDIST pro $k = 4$

k = 4		j							
		0	1	2	3	4	5	6	7
i	0	∞+3	∞+3	∞+3	∞+3	3	∞+3	4	∞+3
	1	∞	∞	∞	∞	∞	∞	∞+1	∞
	2	∞	∞	∞	∞	∞	∞	∞+1	∞
	3	∞	∞	∞	∞	∞	∞	∞+1	∞
	4	∞	∞	∞	∞	0	∞	1	∞
	5	∞+2	∞+2	∞+2	∞+2	2	∞+2	3	∞+2
	6	∞	∞	∞	∞	∞	∞	∞+1	∞
	7	∞+2	∞+2	∞+2	∞+2	2	∞+2	3	∞+2

Porovnání s DIST:

Tabulka 80 - DIST po porovnání s pomDIST pro $k = 4$

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	∞	4	6
B	∞	0	-2	2	∞	∞	∞	4
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	1	-1	-1	2	0	3	1
G	∞	∞	∞	∞	∞	1	0	-3

H	∞	∞	∞	∞	2	∞	3	0

Tabulka pro k = 5:

Tabulka 81 - pomDIST pro *k* = 5

k = 5		j							
		0	1	2	3	4	5	6	7
i	0	∞-1	∞+1	∞-1	∞-1	∞+2	∞	∞+3	∞+1
	1	∞-1	∞+1	∞-1	∞-1	∞+2	∞	∞+3	∞+1
	2	∞-1	∞+1	∞-1	∞-1	∞+2	∞	∞+3	∞+1
	3	∞-1	∞+1	∞-1	∞-1	∞+2	∞	∞+3	∞+1
	4	∞-1	∞+1	∞-1	∞-1	∞+2	∞	∞+3	∞+1
	5	-1	1	-1	-1	2	0	3	1
	6	0	2	0	0	3	1	4	2
	7	∞-1	∞+1	∞-1	∞-1	∞+2	∞	∞+3	∞+1

Porovnaní s DIST:

Tabulka 82 - DIST po porovnání s pomDIST pro *k* = 5

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	∞	4	6
B	∞	0	-2	2	∞	∞	∞	4
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	∞	∞	∞	∞	0	∞	1	∞
F	-1	1	-1	-1	2	0	3	1
G	0	2	0	0	3	1	0	-3
H	∞	∞	∞	∞	2	∞	3	0

Tabulka pro k = 6:

Tabulka 83 - pomDIST pro *k* = 6

k = 6		j							
		0	1	2	3	4	5	6	7
i	0	4	6	4	4	7	5	4	1
	1	∞	∞+2	∞	∞	∞+3	∞+1	∞	∞-3
	2	∞	∞+2	∞	∞	∞+3	∞+1	∞	∞-3
	3	∞	∞+2	∞	∞	∞+3	∞+1	∞	∞-3
	4	1	3	1	1	4	2	1	-2
	5	3	5	3	3	6	4	3	0
	6	0	2	0	0	3	1	0	-3
	7	3	5	3	3	6	4	3	0

Porovnání s DIST:

Tabulka 84 - DIST po porovnání s pomDIST pro $k = 6$

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	5	4	1
B	∞	0	-2	2	∞	∞	∞	4
C	∞	∞	0	∞	∞	∞	∞	∞
D	∞	∞	1	0	∞	∞	∞	2
E	1	3	1	1	0	2	1	-2
F	-1	1	-1	-1	2	0	3	0
G	0	2	0	0	3	1	0	-3
H	3	5	3	3	2	4	3	0

Tabulka pro k = 7:

Tabulka 85 - pomDIST pro $k = 7$

k = 7		j							
		0	1	2	3	4	5	6	7
i	0	4	6	4	4	3	5	4	1
	1	7	9	7	7	6	8	7	4
	2	∞+3	∞+5	∞+3	∞+3	∞+2	∞+4	∞+3	∞
	3	5	7	5	5	4	6	5	2
	4	1	3	1	1	0	2	1	-2
	5	3	5	3	3	2	4	3	0
	6	0	2	0	0	-1	1	0	-3
	7	3	5	3	3	2	4	3	0

Porovnání s DIST:

Tabulka 86 - DIST po porovnání s pomDIST pro $k = 7$

uzly	A	B	C	D	E	F	G	H
A	0	2	0	4	3	5	4	1
B	7	0	-2	2	6	8	7	4
C	∞	∞	0	∞	∞	∞	∞	∞
D	5	7	1	0	4	6	5	2
E	1	3	1	1	0	2	1	-2
F	-1	1	-1	-1	2	0	3	0
G	0	2	0	0	-1	1	0	-3
H	3	5	3	3	2	4	3	0

Došlo se až na konec algoritmus s výslednou tabulkou DIST. Je dobré si všimnout řádku C, který zůstal v nekonečných hodnotách, neboť z uzlu C nevede žádný cesta do ostatních uzlů. Nyní tabulka obsahuje nejkratší cesty mezi jednotlivými dvojicemi uzlů v grafu. (Tabulka 86)

Vstup do programové realizace:

```
graph = [[0, 2, INF, INF, 3, INF, INF, INF],
        [INF, 0, -2, 2, INF, INF, INF, INF],
        [INF, INF, 0, INF, INF, INF, INF, INF],
        [INF, INF, 1, 0, INF, INF, INF, 2],
        [INF, INF, INF, INF, 0, INF, 1, INF],
        [-1, INF, INF, -1, INF, 0, INF, INF],
        [INF, INF, INF, INF, INF, 1, 0, -3],
        [INF, INF, INF, INF, 2, INF, INF,0]]
floydwarshall(graph)
```

Výstup z programové realizace:

```
0 2 0 4 3 5 4 1
7 0 -2 2 6 8 7 4
inf inf 0 inf inf inf inf inf
5 7 1 0 4 6 5 2
1 3 1 1 0 2 1 -2
-1 1 -1 -1 2 0 3 0
0 2 0 0 -1 1 0 -3
3 5 3 3 2 4 3 0
```

ZÁVĚR

Cílem v dané práci bylo vytvoření studijní opory, které může čtenář použít k seznámení s metodikami dynamického programování. Tato práce se hlavně zaměřuje na algoritmy pro vyhledávání extremálních cest v grafech z hlediska dynamického programování. Specificky řečeno se zaměřuje na Dijkstrův algoritmus a jeho modifikace. Algoritmy jsou také programově realizovány a uplatněny na ukázkových příkladech.

V teoretické části se začíná úvodem do optimalizačních metod. Čtenáři jsou uvedeny základní znalosti o optimalizačních metodách, které jsou potřebné pro dynamické programování. Dynamické programování je pak vysvětleno v následující kapitole, kde jsou uvedeny jeho definice, formy a postupy, které jsou pak zpracovány na ukázkovém příkladu. Následná kapitola se pak zabývá specifikací části metodik dynamického programování, a to o algoritmech pro hledání nejkratší cesty v grafu. Jedná se o Dijkstrův algoritmus, Bellman-Fordův algoritmus a Floyd-Warshallův algoritmus, které jsou jednotlivě analyzovány a vysvětleny jejich postupy na ilustračních příkladech. Další kapitola je o teorii grafů. Kapitola udává základní informace o teorii grafů pro čtenáře, aby mohl lépe pochopit prostředí na které se dané algoritmy aplikují. Poslední kapitola je věnována programovacímu jazyku Python. Čtenáři se tak mohou seznámit s programovacím jazykem, ve kterém byly algoritmy realizovány. Přínosem této práce jsou programové kódy uvedených algoritmů, které mohou sloužit v pedagogickém procesu.

V praktické části se nachází kódy programové realizace jednotlivých algoritmů. Každý jednotlivý algoritmus je ilustrován na dalších příkladech na procvičení jejich postupů. Celkově lze konstatovat, že daná práce seznamuje čtenáře se základy dynamického programování a úvodem pro studium algoritmů pro hledání nejkratší cesty v grafu. Tato práce však neřeší všechny problémy, se kterými je možno se v praxi setkat. Dijkstrův algoritmus například nelze aplikovat na grafy se záporně ohodnoceními hranami. Floyd-Warshallův algoritmus neposkytuje optimální trajektorii mezi libovolnými dvěma uzly. A Bellman-Fordův algoritmus je značně časově náročný při použití na složitých grafech s velkým množstvím hran. Tyto modifikace nebo použití jiných vyhledávacích algoritmů jsou inspirací pro navazující práce.

SEZNAM POUŽITÉ LITERATURY

[1] BELLMAN, Richard. *Mathematical Optimization Techniques*. University of California Press; First Edition (May 27, 2022). 364 stran. ISBN 978-0520319868

[2] MAREŠ, Martin a VALLA, Tomáš. *Průvodce labyrintem algoritmů.* 1. vydání. Praha: CZ.NIC, z.s.p.o., 2017. 486 stran. CZ.NIC, 15. publikace. ISBN 978-80-88168-63-8

[3] MOLNÁROVÁ, Marika. *Metody dynamického programování v logistice a plánování*. Online. Diplomová práce. Praha: Vysoká škola ekonomická v Praze. 2009. Dostupné z: https://theses.cz/id/va5mto/. [cit. 2024-03-28].

[4] GAJDA, Bohumil. *Optimalizace - metody síťové analýzy*. Online. Bakalářská práce. Zlín: Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky. 2007. Dostupné z: https://theses.cz/id/mqy5q9/. [cit. 2024-03-28].

[5] MAGEROVÁ, Veronika. *Dynamické programování: teorie a příklady*. Online. Diplomová práce. Brno: Masarykova univerzita, Přírodovědecká fakulta. 2018. Dostupné z: https://theses.cz/id/gi5eco/. [cit. 2024-03-28].

[6] PAVLÁSEK, Ondřej. *Grafy, grafové algoritmy a jejich využití*. Online, Bakalářská práce. Brno: Vysoké učení technické v Brně. Fakulta podnikatelská. Ústav informatiky, 2010. Dostupné z: http://hdl.handle.net/11012/1094. [cit. 2024-03-28].

[7] FISCHETTI, Matteo. *Introduction to Mathematical Optimization*. Padova. M. Fischetti, 2019. ISBN 978-1692792022

[8] ČERNÝ, Jakub. *Základní grafové algoritmy*. - 1. vyd. - V Praze: České vysoké učení technické v Praze, 2013. ISBN 978-80-01-05258-7

[9] KYBIC, Jan. *Dynamické programování*. Online, prezentace. 2016. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/b3b33alp/prednasky/11_dynamicke_programovani.pdf . [cit. 2024-04-20].

[10] MATOUŠEK, Jiří a NEŠETŘIL, Jaroslav. *Kapitoly z diskrétní matematiky*. Praha, Karlova Univerzita. 2002. ISBN: 978-80-246-1740-4

[11] ŠEDA, Miloš. Teorie grafů. Online. Brno: Vysoké Učení Technické, Fakulta Strojního Inženýrství. 2003. Dostupné z: https://www.euroekonom.sk/download2/materialy-vs-informatika/Teorie-Grafu.pdf. [cit. 2024-03-19].

[12] WERNER, Tomáš. *Optimalizace*. Online. Prezentace. Praha: České vysoké učení technické v Praze. 2018. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/a4b33opt/opt.pdf. [cit. 2024-04-13]

[13] PYTHON SOFTWARE FOUNDATION. Python 3.12.3 Documentation. Online. 2024. Dostupné z: https://docs.python.org/3/. [cit. 2024-04-15]

[14] W3SCHOOLS. Python Introduction. 2024. Dostupné z: https://www.w3schools.com/python/python_intro.asp. [cit. 2024-04-15]

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

$\mathbb{R}$	Množina reálných čísel
$f: A \rightarrow \mathbb{R}$	Zobrazení množiny A v množině $\mathbb{R}$
{0, 1, …|V|-1}	Množina posloupnosti od 0 do počtu uzlů v grafu -1
$\|N\|$ nebo $\|V\|$	Počet uzlů v grafu

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: DIJSKTRA.PY

Soubor obsahuje programovou realizaci Dijkstrova algoritmu, Bellman-Fordova algoritmu a Floyd-Warshallova algoritmu v jazyce Python. Dále jsou tam přidány vstupy grafů použitých v dané práci.