

Algoritmy hledání nejkratší trasy

Shortest Path Problem Solving Algorithms

Michal Kopřiva

Bakalářská práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2011/2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Michal KOPŘIVA**
Osobní číslo: **A09053**
Studijní program: **B 3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**

Téma práce: **Algoritmy hledání nejkratší trasy**

Zásady pro vypracování:

1. Zpracujte literární rešerši, mapující problematiku hledání nejkratší vzdálenosti mezi dvěma body grafu.
2. Teoreticky popište principy nejčastěji používaných algoritmů.
3. Popište příklady jejich praktických implementací.
4. Naprogramujte vybrané algoritmy a otestujte jejich implementace.
5. Vypracujte statistické zhodnocení provedených experimentů.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskrétní matematiky. 3. Praha: Karolinum, 2007. ISBN 978-80-246-1411-3.
2. KOLÁŘ, Josef. Teoretická informatika. Praha: Česká technika, 2009. ISBN 978-80-01-04331-8.
3. CORMEN, Thomas H., Charles E. LEISERSON, Ronald L. RIVEST a Clifford STEIN. Introduction to Algorithms. 2. Cambridge (Massachusetts): The MIT Press, 2001. ISBN 0-262-53196-8.
4. Základní grafové algoritmy. ČERNÝ, Jakub. Katedra aplikované matematiky Online1. 24. 10. 2010 [cit. 2012-01-14]. Dostupné z: <http://kam.mff.cuni.cz/kuba/ka>
5. HEROUT, Pavel. Učebnice jazyka Java. 2. České Budějovice: Kopp, 2005. ISBN 80-7232-115-3.
6. HEROUT, Pavel. Java: grafické uživatelské prostředí a čeština. České Budějovice: Kopp, 2007. ISBN 978-80-7232-328-9.

Vedoucí bakalářské práce:

Ing. Bc. Pavel Vařacha

Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce:

24. února 2012

Termín odevzdání bakalářské práce:

8. června 2012

Ve Zlíně dne 24. února 2012

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato práce pojednává o algoritmech sloužících především k hledání nejkratší cesty grafem. Jsou zde popsány algoritmy Dijkstrův, Floyd-Warshallův, Bellman-Fordův a algoritmus A-star. Je zde také popsán samo-organizující se migrační algoritmus.

Klíčová slova:

Dijkstrův algoritmus, Bellman-Fordův algoritmus, Floyd-Warshallův algoritmus, algoritmus A-star, SOMA.

ABSTRACT

This thesis discusses the algorithms used especially for finding the shortest path graph. There are described Dijkstra's algorithm, Floyd-Warshall algorithm, Bellman-Ford algorithm and the A-star algorithm. Further is described self-organizing migrating algorithm.

Keywords:

Dijkstra's algorithm, Bellman-Ford algorithm, Floyd -Warshall algorithm, A-star algorithm, SOMA.

Rád bych poděkoval Ing. Bc. Pavlu Vařachovi, Ph.D. za odbornou pomoc a připomínky při vypracování této bakalářské práce.

Dále bych chtěl poděkovat své rodině za pomoc a trpělivost při mém studiu.

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 a jeden výtisk bude uložen u vedoucího práce;
- 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 s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- 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

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST.....	10
1 GRAF.....	11
1.1 ZÁKLADNÍ POJMY	12
2 ALGORITMUS	18
2.1 VLASTNOSTI ALGORITMU	18
2.2 ZPŮSOBY POPISU ALGORITMŮ	18
2.3 ČASOVÁ SLOŽITOST.....	19
2.4 ASYMPTOTICKÁ SLOŽITOST	19
2.5 PRŮMĚRNÁ A AMORTIZOVANÁ SLOŽITOST	21
3 GRAFOVÉ ALGORITMY	23
3.1 PROHLEDÁVÁNÍ DO ŠÍŘKY	23
3.2 PROHLEDÁVÁNÍ DO HLOUBKY	23
3.3 DIJKSTRŮV ALGORITMUS	24
3.4 BELLMAN – FORDŮV ALGORITMUS	26
3.5 FLOYD – WARSHALLŮV ALGORITMUS.....	27
3.6 ALGORITMUS A-STAR	27
3.7 SOMA	28
II PRAKTICKÁ ČÁST	31
4 PŘÍKLADY PRAKTICKÝCH IMPLEMENTACÍ.....	32
5 VLASTNÍ IMPLEMENTACE ALGORITMŮ NEJKRATŠÍ TRASY.....	34
5.1 JAZYK JAVA	34
5.2 PROGRAM PRO HLEDÁNÍ NEJKRATŠÍ TRASY	35
5.2.1 WindowRun.java.....	35
5.2.2 PanelGraf.java	36
5.2.3 Node.java	36
5.2.4 Edge.java.....	36
5.2.5 Cesta.java	36
5.2.6 Open.java	36
5.2.7 Reader.java.....	37
5.2.8 Graf.java.....	37
5.2.9 Dijkstra.java	38
5.2.10 Bellman.java.....	38
5.2.11 Floyd.java.....	39
5.2.12 AStar.java.....	39
5.3 OVLÁDÁNÍ PROGRAMU	39
5.4 TESTOVÁNÍ ALGORITMŮ	39
ZÁVĚR	42
CONCLUSION	43
SEZNAM POUŽITÉ LITERATURY.....	44
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	46
SEZNAM OBRÁZKŮ	47

SEZNAM TABULEK.....	48
SEZNAM PŘÍLOH.....	49

ÚVOD

Tématem této bakalářské práce jsou algoritmy, které slouží k hledání nejkratší trasy grafem. Hledání nejkratší cesty grafem nachází uplatnění v mnoha oblastech, především pak v logistice. Metody popisované v této práci tak nacházejí svoje uplatnění například při plánování nejrychlejší trasy, ale i v oblastech zdánlivě s logistikou nesouvisejících jako je provoz počítačových sítí, řízení pohybu robotů nebo pro návrh plošných spojů. Cílem této práce je shrnutí teoretických poznatků dané problematiky, dále bude uveden přehled praktických použití algoritmů hledání nejkratší trasy a vybrané algoritmy budou naprogramovány a bude otestována jejich implementace. Na závěr bude provedeno hodnocení výsledků testování algoritmů.

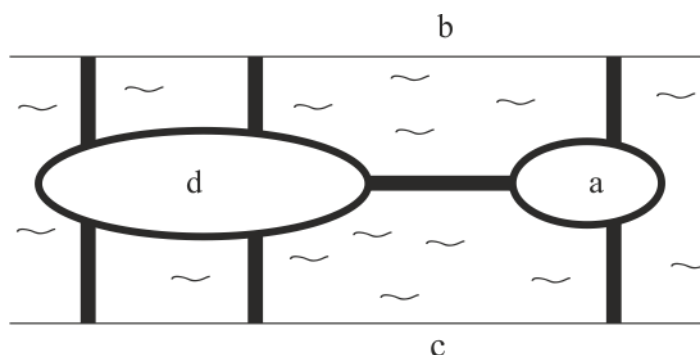
V první kapitole budou vysvětleny pojmy a principy, uplatňující se při práci s grafy. Mimo jiné zde budou uvedeny typy grafů a způsoby jejich prezentací. V další části budou popsány vlastnosti algoritmů a jejich časová náročnost. V poslední kapitole teoretické části budou popsány konkrétní algoritmy sloužící pro práci s grafy. Jsou to Dijkstrův, Bellman-Fordův, Floyd-Warshallův algoritmus, algoritmus A-Star a SOMA.

Praktická část se bude zabývat příklady praktických použití algoritmů, popisem programovacího jazyka Java, stručným popisem jednotlivých tříd programu a výsledky provedených testů.

I. TEORETICKÁ ČÁST

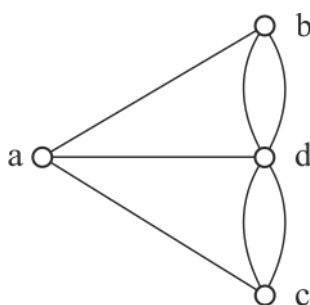
1 GRAF

Velké množství situací v informatice, matematice, elektrotechnice a praktických úlohách (například silniční síť, chemické vazby mezi prvky) lze vyjádřit pomocí schématu, které se skládá z množiny bodů a množiny spojníc těchto bodů. Takové schéma nazýváme grafem a jeho teorií se zabývá obor diskrétní matematika. Za počátek teorie grafů se považuje řešení tzv. problému sedmi mostů, které přinesl Leonhard Euler. Popis problému zní následovně. Ve městě se nachází řeka se dvěma ostrovy, které jsou spojeny s břehem pomocí sedmi mostů. Úkolem je projít cestou všech sedm mostů tak, abychom se vrátili na původní místo tak, že každý most projdeme pouze jednou. Situace je znázorněna na obrázku (Obr. 1).



Obr. 1 Problém sedmy mostů

Euler dokázal, že tato cesta není možná, protože by z každého bodu musel vycházet sudý počet spojníc. Eulerovým přínosem při řešení tohoto problému bylo zjednodušení situace a její znázornění pomocí bodů a spojníc, jak znázorňuje schéma (Obr. 2).



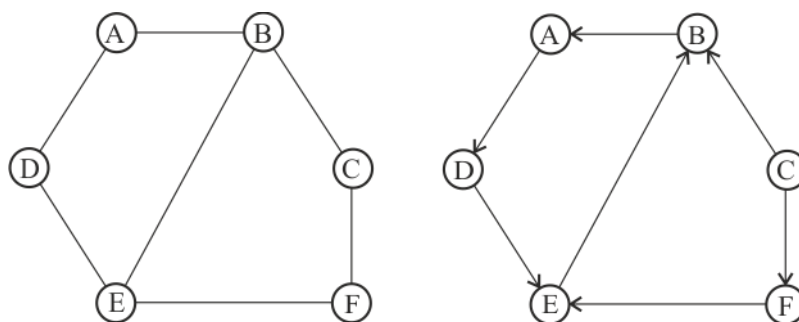
Obr. 2 Graf problému

z obrázku 1 [2]

V této části budou vysvětleny některé pojmy z této oblasti. Definice, které jsou uvedeny v této kapitole, jsou převzaty z [1 - 5].

1.1 Základní pojmy

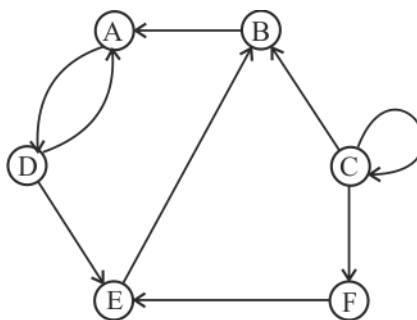
Výrazem graf je v teorii grafů myšleno něco jiného než zobrazení průběhu funkce. Grafem rozumíme dvojici (V, E) , kde V je množina vrcholů a E je množina hran. Každá hrana je v neorientovaném grafu přiřazena neuspořádané dvojici vrcholů (je-li graf orientovaný pak uspořádané) (u, v) , kde u a v náleží E . V grafickém znázornění orientovaného grafu vyjadřujeme orientaci pomocí šipky. Uzel, ze kterého šipka vychází, nazýváme předchůdcem a uzel, do kterého šipka směřuje, označujeme jako koncový. Na obrázku (Obr. 3) vlevo je znázorněn neorientovaný a vpravo orientovaný graf.



Obr. 3 Neorientovaný a orientovaný graf

Další grafové pojmy:

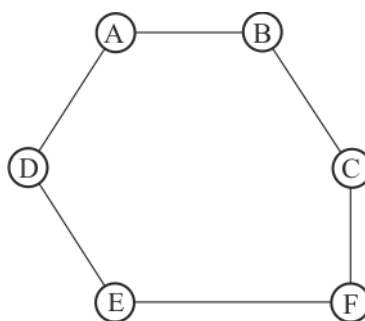
- Triviální graf – graf, který obsahuje jediný vrchol a žádné hrany.
- Násobné hrany – hrany, které jsou přiřazeny stejné dvojici vrcholů.
- Smyčka grafu – hrana, která začíná a končí ve stejném vrcholu.



Obr. 4 Násobné hrany a smyčka
v grafu

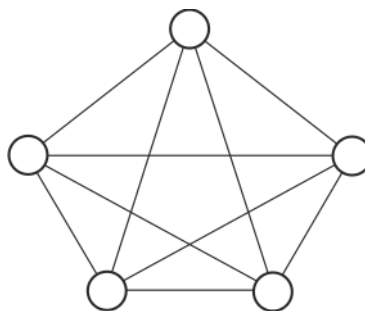
- Sled S je posloupnost uzlů u a hran h taková, že $S = (u_0, h_1, u_1, h_2, \dots, u_{n-1}, h_n, u_n)$, tzn., že každé dvě po sobě následující hrany mají stejný vrchol.
- Tah – taková posloupnost, ve které se neopakují hrany.
- Cesta – taková posloupnost, ve které se neopakují hrany ani vrcholy.

- Cyklus, uzavřený tah nebo také kružnice je uzavřená cesta – viz obrázek (Obr. 5).



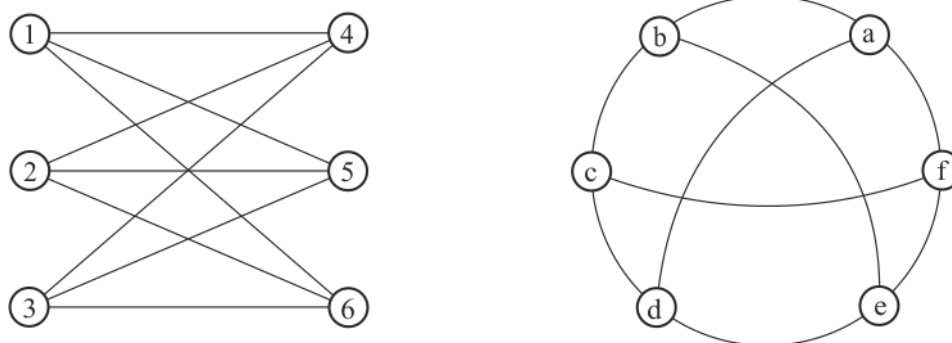
Obr. 5 Cyklus grafu

- Stupeň vrcholu – počet hran, které jsou spojeny s vrcholem.
- Pravidelný nebo regulární graf je takový, pokud jsou všechny jeho vrcholy stejného stupně – viz obrázek (Obr. 6).



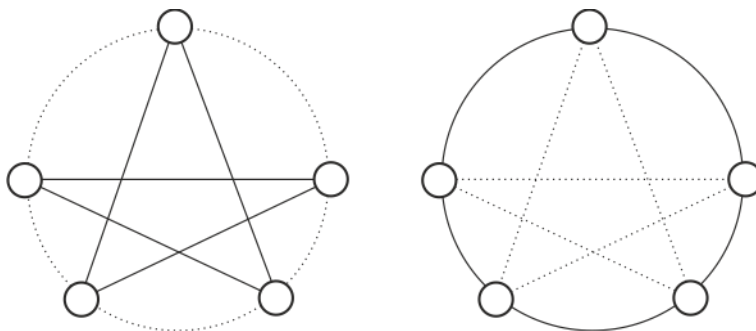
Obr. 6 Regulární graf

- Isomorfismus – grafy jsou isomorfní, pokud mají totožné množiny vrcholů a hran a mohou se lišit pouze označení vrcholů a hran, jako na obrázku (Obr 7).



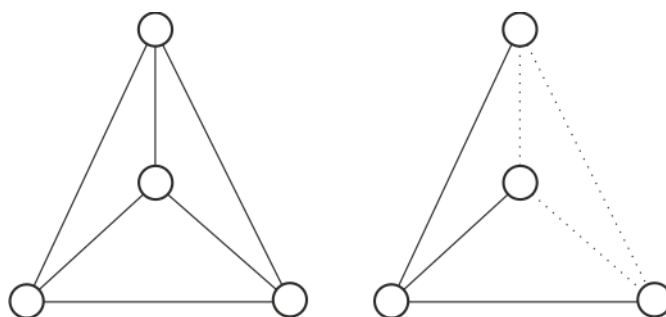
Obr. 7 Isomorfní grafy [4]

- Doplněk grafu je graf, který má stejné vrcholy jako originální graf a ty jsou propojeny hranami, které originál neobsahuje, jak ukazuje obrázek (Obr 8).



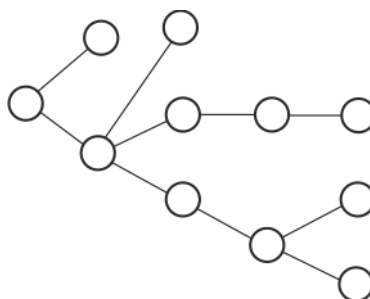
Obr. 8 Doplněk grafu [4]

- Podgraf vznikne odstraněním některých hran a vrcholů (včetně hran z nich vedoucích) z originálního grafu. Na obrázku (Obr. 9) vlevo je znázorněn originální graf a vpravo jeho podgraf.



Obr. 9 Podgraf grafu

- Souvislost grafu – graf je souvislý, pokud existuje cesta pro každé dva vrcholy tohoto grafu.
- Strom je v teorii grafů definován jako souvislý graf bez cyklů. Každý strom s n vrcholy má právě $n - 1$ hran. Příklad na obrázku (Obr. 10).



Obr. 10 Strom

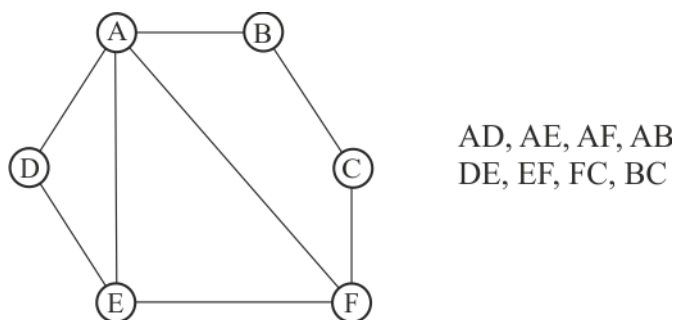
- Kostra grafu je podgraf, který spojuje všechny vrcholy grafu a neobsahuje kružnici.
- Kořen grafu je takový vrchol v orientovaném stromu, pro který platí, že z něj vede orientovaná cesta do ostatních uzlů.
- Kořenový strom je graf, který je stromem a obsahuje kořen.

- Hloubka uzlu je délka cesty z kořene do uzlu.
- Minimální kostra grafu je taková kostra, která spojuje všechny vrcholy grafu a součet délek nezáporně ohodnocených hran je minimální.

Délka cesty v neohodnoceném grafu je rovna počtu hran na cestě. Vzdálenost mezi vrcholy u a v je délka nejkratší cesty mezi u a v . Pokud neexistuje cesta mezi u a v , tak je vzdálenost nekonečná. Takto definovaná vzdálenost odpovídá vzdálenosti v ohodnoceném grafu, kde je každá hrana ohodnocena číslem jedna. Ohodnocený graf G je graf $G = (V, E)$, ke kterému je přiřazena funkce $h : E \rightarrow \mathbb{R}$, která každé hraně e přiřadí hodnotu $h(e)$, kde \mathbb{R} je množinou reálných čísel. Toto číslo reprezentuje nějakou vlastnost hrany, například vzdálenost mezi dvěma vrcholy, kapacitu hrany, cenu (může nabývat záporné hodnoty) nebo určitou náročnost spojenou s průchodem vrcholu.

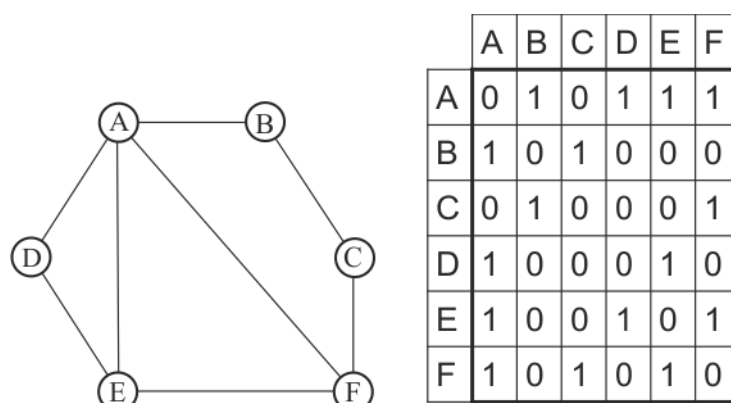
Aby bylo možné grafy zpracovávat výpočetní technikou, je nutné vyjádřit tyto struktury způsobem, který dovoluje použití algoritmů. Grafy je možné reprezentovat následujícími způsoby.

Vyjádření pomocí seznamu hran. Tímto způsobem udržujeme seznam dvojic vrcholů. Jestliže takto popisujeme neorientovaný graf, nezáleží na tom, v jakém pořadí uzlů jsou hrany uloženy. V případě orientovaného grafu na tomto pořadí záleží, protože tím je uložena informace o orientaci hrany. Na obrázku (Obr. 11) je znázorněn neorientovaný graf a jeho vyjádření pomocí seznamu hran.



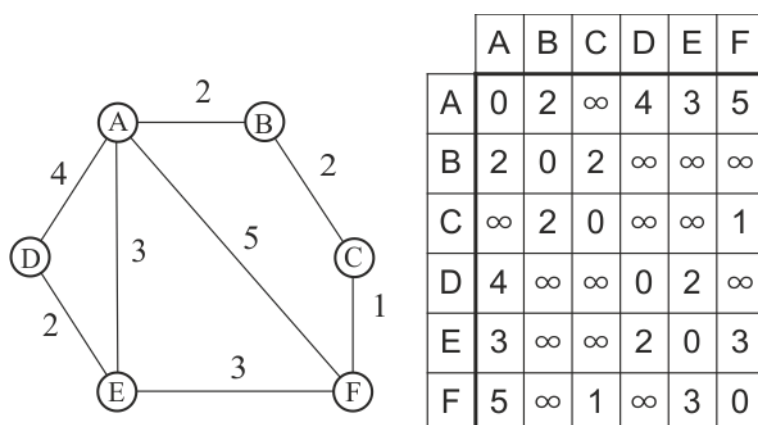
Obr. 11 Reprezentace grafu seznamem hran

Další způsob se nazývá matice sousednosti. Tato matice o rozměrech $n \times n$, kde n je počet vrcholů, zachycuje, mezi kterými vrcholy existuje hrana. V případě, že spolu dva vrcholy sousedí, obsahuje matice na spojnici řádku prvního uzlu a sloupce druhého uzlu jedničku. V opačném případě je na příslušné pozici nula. Na hlavní diagonále matice jsou obsaženy nuly. Takto se zaznamenávají vztahy mezi vrcholy u neohodnoceného grafu – viz obrázek (Obr. 12).



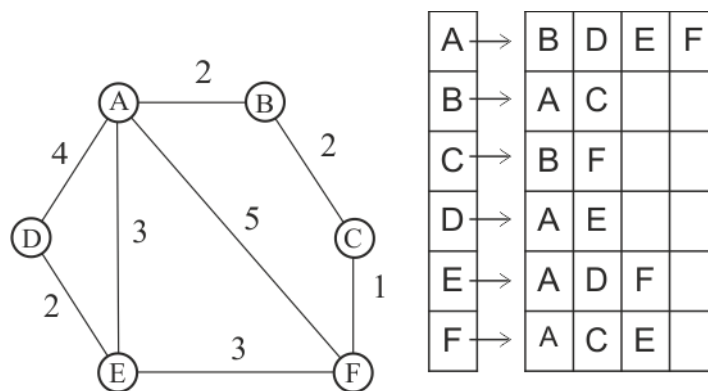
Obr. 12 Matice sousednosti

Jestliže ohodnotíme hrany grafu, nahradíme jednotkovou vzdálenost příslušnou hodnotou a získáme matici vzdáleností. Tato matice obsahuje nuly na hlavní diagonále, příslušné hodnoty, pokud existuje cesta mezi uzly a nekonečno, pokud tato cesta neexistuje – viz obrázek (Obr. 13).



Obr. 13 Matice vzdáleností

Třetí způsob vyjádření grafu je seznam sousedů, který je implementován jako seznam uzlů, kde každá jednotlivá položka ukazuje do dalšího seznamu, který obsahuje příslušné sousedy. I tímto způsobem zápisu lze zaznamenat orientaci hran. Příklad implementace je znázorněn na obrázku (Obr. 14).



Obr. 14 Seznam sousedů

2 ALGORITMUS

Algoritmus je v podstatě návod, pomocí něhož lze provádět určitou činnost. V této části budou vysvětleny základní pojmy a vlastnosti týkající se algoritmů. Definice jsou převzaty z [4 - 8].

2.1 Vlastnosti algoritmu

Aby se o návodu dalo hovořit jako o algoritmu, musí splňovat následující podmínky:

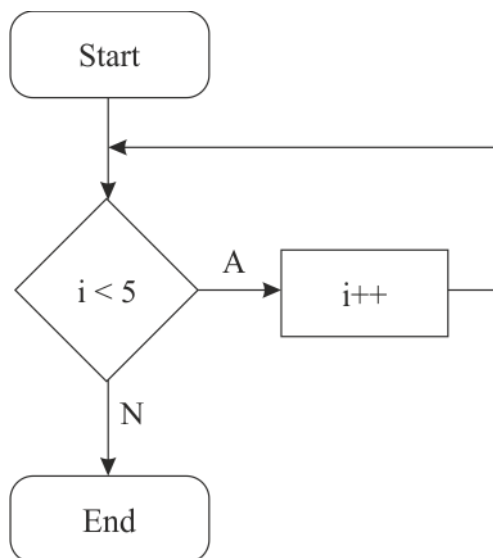
- Je elementární – tzn., že se skládá z konečného, dále nedělitelného počtu kroků, které jsou snadno proveditelné.
- Je determinovaný – tzn., že v každém kroku lze určit, v jakém se nachází stavu.
- Je konečný – skládá se z konečného počtu kroků.
- Má výstup – má alespoň jeden výstup, který umožní vyjádření výsledku.
- Jeho činnost vede ke správnému výsledku.

Každý algoritmus má vstup, který slouží k zadávání různých hodnot, potřebných pro činnost algoritmu. Tyto hodnoty se zadávají před startem algoritmu nebo v průběhu jeho běhu. Například i algoritmus pro vypsání Fibonacciho posloupnosti musí mít na vstupu počet opakování výpočtu, jinak se opakuje do nekonečna a nesplňuje podmínku konečnosti. Iterací rozumíme provedení jednoho kroku algoritmu.

2.2 Způsoby popisu algoritmů

Popis algoritmů lze vyjádřit pomocí mnoha prostředků. Některé způsoby používají slovní popis, některé využívají grafického znázornění. Nejčastěji používané prostředky jsou:

- Slovní popis – popisuje jednotlivé kroky algoritmu metodou shora dolů, velmi často se využívá zápis pomocí pseudokódu.
- Strukturogramy – graficky znázorňují strukturu algoritmu. Pomocí tvarového nebo barevného vyjádření se vyjadřují základní struktury algoritmu (posloupnost, cyklus, podmínka).
- Jacksonovy diagramy – vyjadřují vztahy mezi objekty reálného světa.
- Vývojové diagramy – jednotlivé symboly vyjadřují procesy algoritmu a ty jsou propojeny pomocí šipek, které určují směr běhu programu. Příklad je na obrázku (Obr. 15).



Obr. 15 Vývojový diagram

2.3 Časová složitost

Často je nutné najít algoritmy, které vyhovují určitým požadavkům. Díky porovnání těchto vlastností se lze rozhodnout o vhodnosti či nevhodnosti určitého algoritmu pro daný účel. Tyto požadavky jsou především:

- Rychlost výpočtu – tzn., za jak dlouho program zjistí výsledek.
- Paměťová náročnost – jak velký paměťový prostor algoritmus potřebuje pro svou činnost. Například využívání externích paměťových zdrojů může ovlivnit rychlost programu.
- Rychlost implementace vlastního kódu programu – tzn., za jakou dobu je výpočet převeden pomocí programovacího jazyka do kódu, který je proveditelný na počítači.

Při porovnávání časové složitosti algoritmu se vychází z celkového počtu kroků a velikosti vstupních dat. Časová složitost je funkce $T : \mathbb{N} \rightarrow \mathbb{N}$, kde $T(n)$ je (maximální) počet kroků, které provede algoritmus běžící na datech o velikosti n (pro všechny vstupy D velikosti n) a \mathbb{N} je množina přirozených čísel. Časová složitost $c \cdot n$ se nazývá lineární, $c \cdot n^2$ kvadratická, $c \cdot n^3$ kubická, $c \cdot a^n$ pro $a > 1$ exponenciální, kde n je velikost vstupu a c časová konstanta.

2.4 Asymptotická složitost

Asymptotická složitost je rozdělení algoritmů do tříd složitostí, u kterých platí, že od určité velikosti dat, je algoritmus dané třídy vždy pomalejší než algoritmus třídy

předchozí, bez ohledu na to, jestli je některý z počítačů c -násobně výkonnější. O – omikron, vyjadřuje horní hranici chování, tzn., že vyjadřuje dobu, do které algoritmus určitě skončí. Ω – omega vyjadřuje dolní hranici chování, tzn. že vyjadřuje dobu, do které algoritmus určitě neskončí. Θ – theta vyjadřuje třídu chování, neboli určuje dolní a horní hranici, mezi kterými se pohybuje doba vykonávání výpočtu. Definice: Necht' f a g jsou dvě funkce z přirozených čísel do přirozených čísel. Řekneme, že:

$$f(n) = O(g(n)) \quad (1)$$

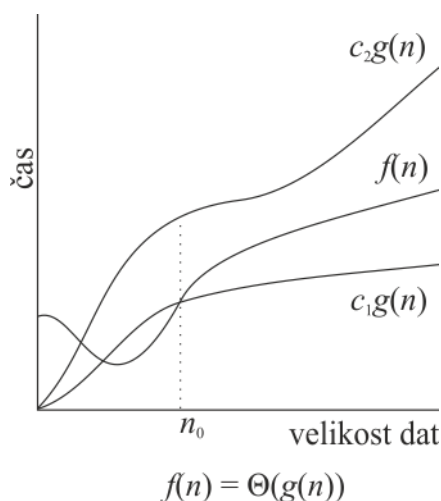
právě tehdy, když existuje konstanta $c > 0$ a n_0 takové, že pro každé $n \geq n_0$ platí $f(n) \leq c \cdot g(n)$, tzn., že funkce f neroste řádově rychleji než funkce g .

$$f(n) = \Omega(g(n)) \quad (2)$$

právě tehdy, když existuje konstanta $c > 0$ a n_0 takové, že pro každé $n \geq n_0$ platí $f(n) \geq c \cdot g(n)$, tzn. že funkce f roste řádově alespoň tak rychle jako funkce g .

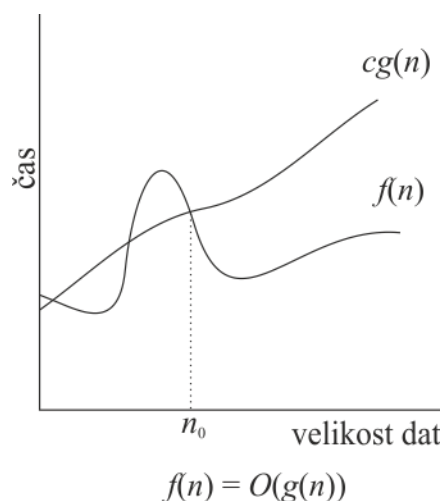
$$f(n) = \Theta(g(n)) \quad (3)$$

právě tehdy, když zároveň $f(n) = O(g(n))$ i $f(n) = \Omega(g(n))$, tzn., že obě funkce rostou řádově stejně rychle. Grafické znázornění asymptotických složitostí je na obrázcích (Obr. 16 – 18).



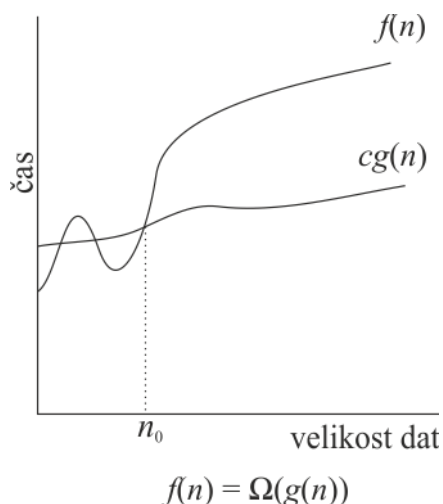
Obr. 16 Třída složitosti theta

[6]



Obr. 17 Třída složitosti

omikron [6]



Obr. 18 Třída složitosti omega

[6]

Níže jsou uvedeny příklady tříd asymptotické složitosti:

- $O(1)$ – algoritmy s konstantní časovou složitostí – např. skok v poli.
- $O(\log(n))$ – algoritmy s logaritmickou časovou složitostí – např. vyhledávání v seřazeném poli.
- $O(n)$ – algoritmy s lineární časovou složitostí, např. vyhledávací algoritmy.
- $O(n \cdot \log(n))$ – lineární složitost – např. řadící algoritmy.
- $O(n^2)$ – kvadratická časová složitost – např. dva vnořené cykly čítající do n .
- $O(n^3)$ – kubická časová složitost. Tyto algoritmy se používají pouze pro jednodušší problémy, protože náročnost výpočtu rychle vzrůstá.
- $O(k^n)$ – označení pro algoritmy s exponenciální časovou složitostí. Algoritmy tohoto typu jsou nepoužitelné pro náročnější problémy. Příkladem algoritmu této složitosti je problém obchodního cestujícího.

2.5 Průměrná a amortizovaná složitost

Pokud se mění rychlost algoritmu v závislosti na vstupních datech, je nutné vypočítat tzv. průměrnou časovou složitost. Průměrnou složitost vypočítáme jako průměr časových složitostí přes všechny možné vstupy. Jako příklad, kdy je průměrná časová složitost lepší než časová složitost v nejhorším případě, může sloužit algoritmus quicksort. Amortizovaná časová složitost označuje časovou složitost algoritmu v sekvenci nejhorších možných vstupních dat. Na rozdíl od průměrné složitosti nevyužívá pravděpodobnosti a je proto

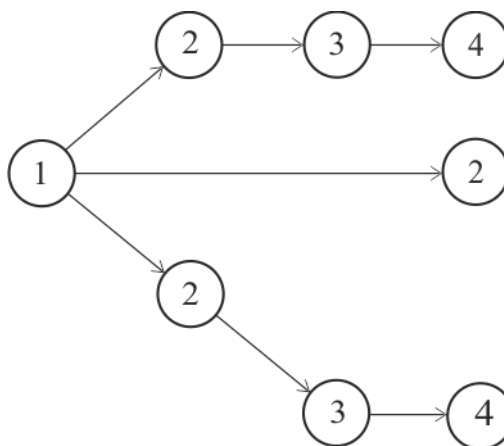
zaručená. Amortizovanou časovou složitost jedné operace vypočítáme jako celkovou časovou složitost posloupnosti operací v nejhorším případě dělenou počtem operací.

3 GRAFOVÉ ALGORITMY

V této části budou popsány základní grafové algoritmy, sloužící k průchodu grafu a algoritmy pro hledání nejkratší cesty. Dále bude popsán evoluční algoritmus SOMA. Definice a principy uvedených algoritmů jsou převzaty z [1, 2, 4, 5, 9, 10, 11].

3.1 Prohledávání do šířky

Prohledávání do šířky (Breadth-first search, BFS) patří mezi základní grafové algoritmy, který slouží k procházení všech uzlů daného grafu. Graf prozkoumáváme po tzv. vlnách, protože každým dalším krokem algoritmus nalezne všechny přímé potomky výchozího uzlu, v pořadí, které je dané vzdáleností od výchozího uzlu. Při prohledávání do šířky lze začínat z libovolného uzlu grafu. Tento uzel označíme jako zpracovaný. V dalším kroku nalezneme všechny jeho potomky, které uložíme do fronty a ty budou postupně zpracovány stejným způsobem. Algoritmus pokračuje, dokud fronta obsahuje nějaké uzly. Obrázek (Obr. 19) naznačuje princip prohledávání do šířky.



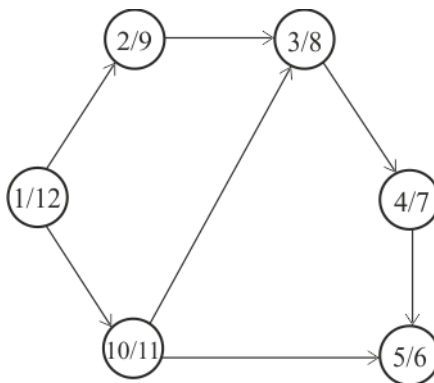
Obr. 19 Prohledávání do šířky – čísla
označují vlnu, ve které je uzel
zpracován

Výstupem algoritmu BFS je tzv. BF strom, který obsahuje všechny uzly, které jsou dostupné z výchozího uzlu.

3.2 Prohledávání do hloubky

Strategie algoritmus prohledávání do hloubky (Depth-first search, DFS) spočívá ve snaze postupovat co „nejhlouběji“ do grafu. Algoritmus postupuje tak, že zvolí libovolný uzel grafu, ten označí jako otevřený. Pokud objeví potomka tohoto uzlu, označí jej také

za otevřený a takto postupuje do největší hloubky grafu. Pokud objeví uzel, který nemá neprozkoumaného potomka, označí jej za uzavřený a vrací se do předchozího uzlu a tam testuje, zda nemá uzel další neprozkoumané potomky. Takto se pokračuje, dokud graf obsahuje nějaký neprozkoumaný uzel. Situaci s otevíráním a zavíráním uzlů znázorňuje obrázek (Obr. 20). První z dvojice čísel značí, ve kterém kroku byl uzel otevřen a druhé číslo označuje krok, ve kterém byl uzel uzavřen.



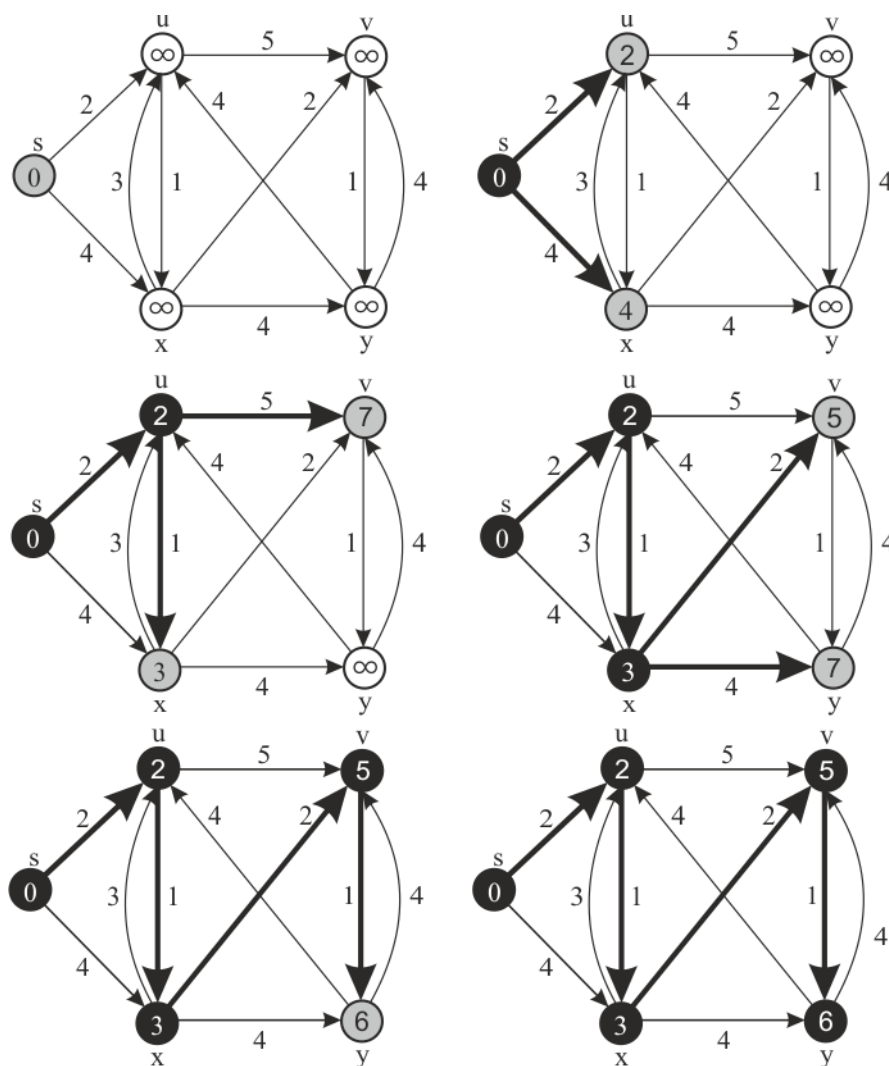
Obr. 20 Procházení grafu
do hloubky

3.3 Dijkstrův algoritmus

První algoritmus, který se řadí mezi deterministické, vymyslel nizozemský informatik Edsger Dijkstra. Slouží k nalezení všech nejkratších cest z počátečního uzlu do ostatních uzlů grafu, který obsahuje nezáporně ohodnocené hrany. Vstupem algoritmu je graf, počáteční a koncový uzel. U každého uzlu je udržován stav uzlu (dočasný, trvalý), vzdálenost uzlu od počátku a odkaz na předchůdce uzlu, z důvodu rekonstrukce cesty. Dijkstrův algoritmus pracuje jako zobecněné prohledávání grafu do šířky, při kterém se vlna nešíří na základě počtu hran od zdroje, ale vzdálenosti od zdroje. Tato vlna proto zpracovává jen ty uzly, k nimž již byla nalezena nejkratší cesta. Dijkstrův algoritmus uchovává všechny uzly v prioritní frontě, ve které jsou řazené dle vzdálenosti od zdroje. Na počátku se nastaví startovnímu uzlu hodnota nula, všem ostatním uzlům nekonečno. Algoritmus v každém svém kroku vybere z fronty uzel s nejnižší vzdáleností od zpracované části (v prvním kroku je to startovní uzel) a zařadí jej mezi zpracované uzly. Poté projde všechny jeho dosud nezpracované potomky, přidá je do fronty, nejsou-li tam již obsaženi, a ověří, zdali nejsou blíže zdroji, než byli před zařazením právě vybraného uzlu mezi zpracované. To znamená, že pro všechny potomky ověřuje:

$$vzdalenost_{zpracovavany} + delka\ Hrany_{zpracovavany,potomek} < vzdalenost_{potomek} \quad (4)$$

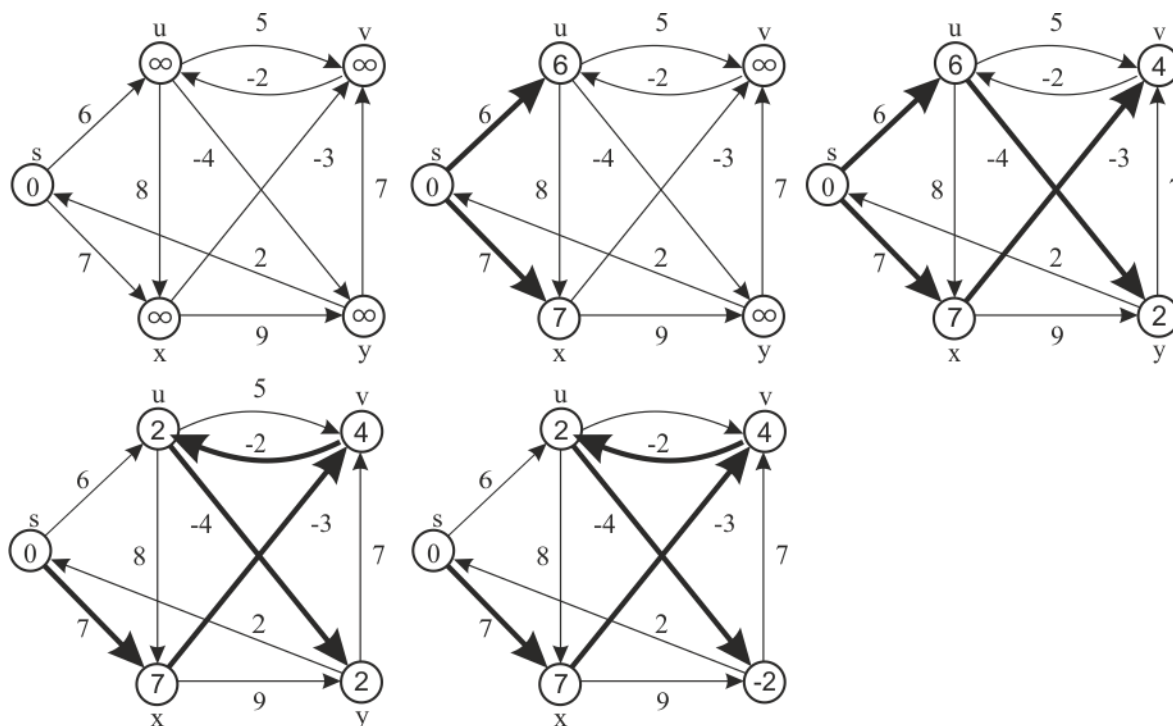
Tato operace je označovaná jako relaxace nebo také update hrany. Pokud nerovnost platí, tak danému potomkovi nastaví novou vzdálenost a označí za jeho předka zpracovávaný uzel. Po průchodu přes všechny potomky algoritmus vybere z fronty uzel s nejvyšší prioritou, ten označí za zpracovaný a celý krok opakuje. Pokud hledáme nejkratší cestu z počátečního bodu do koncového bodu, končí algoritmus, pokud je koncový uzel označen jako trvalý. V případě hledání nejkratší cesty do všech uzlů, končí algoritmus po vyprázdnění prioritní fronty. Složitost algoritmu závisí na provedení prioritní fronty. Pokud je fronta implementována pomocí pole, dostaneme celkový čas $O(V^2)$, pokud je fronta vyjádřena pomocí binární haldy, dostaneme celkový čas $O(E \cdot \log_2 V)$. Princip je znázorněn na obrázku (Obr. 21). Šedá barva značí dočasný a černá trvalý uzel. Tučně zvýrazněné šipky značí nejkratší cestu do uzlu.



Obr. 21 Princip Dijkstrova algoritmu [2]

3.4 Bellman – Fordův algoritmus

Bellman-Fordův algoritmus slouží k nalezení všech nejkratších cest z počátečního uzlu do ostatních uzlů grafu v orientovaném grafu s libovolným ohodnocením hran, tím se liší od Dijkstrova algoritmu. Základem Bellman-Fordova algoritmu je opět operace relaxace. Do této operace vstupují dva uzly a hrana, která mezi nimi vede. Pokud je vzdálenost zdrojového uzlu sečtená s délkou hrany menší než aktuální vzdálenost cílového uzlu, tak se za předchůdce cílového uzlu na nejkratší cestě označí zdrojový uzel. V případě nesplnění nerovnosti tato hrana cestu nezkracuje a neprovádí se proto žádné změny. Na rozdíl od Dijkstrova algoritmu se ale systematicky relaxují všechny hrany. To znamená, že program prochází v cyklu seznam všech vrcholů a upravuje jejich hodnoty. Délka cesty ze zdrojového do každého z cílových uzlů může být dlouhá maximálně $|V| - 1$ hran (protože by jinak musela obsahovat cyklus). Pokud provedeme operaci relaxace se všemi hranami grafu $(|V| - 1)$ krát, tak jsou nalezeny všechny nejkratší cesty. Toto ověříme ještě jedním spuštěním relaxací všech hran. Pokud dojde k nějaké relaxaci, tak graf obsahuje cyklus záporné délky, pokud k relaxaci nedojde, algoritmus může vrátit výsledek. Asymptotická složitost algoritmu je $O(|V| \cdot |E|)$, protože vnější smyčka proběhne právě $(|V|-1)$ krát a vnitřní smyčka probíhá přes všechny hrany grafu. Na obrázku (Obr. 22) je znázorněn princip algoritmu. Tučně zvýrazněné šipky značí nejkratší cestu do uzlu.



Obr. 22 Princip Bellman-Fordova algoritmu [2]

3.5 Floyd – Warshallův algoritmus

Floyd-Warshallův algoritmus nalezne nejkratší cestu mezi všemi dvojicemi vrcholů jediným průchodem grafu. Tento algoritmus pracuje na orientovaném grafu, který neobsahuje záporné cykly a ze všech cest stejné délky vybere tu s nejmenším počtem hran. Floyd-Warshallův algoritmus má na vstupu matici délek D^0 . Tato matice obsahuje vzdálenosti uzlů, které nevedou přes žádného prostředníka. V každé další iteraci algoritmu se tato matice přepočítá tak, aby vyjadřovala vzdálenost všech dvojic uzlů, která vede přes postupně se zvětšující množinu přípustných prostředníků. To znamená, že matice vyjadřuje vzdálenost všech uzlů s možností využití jednoho prostředníka, D^2 vyjadřuje vzdálenost při využití dvou prostředníků a matice D^k při možnosti využití k prostředníků. Algoritmus zpracovává vztah porovnání délek cesty pomocí tří vnořených cyklů.

```
n:=pocet uzlu grafu G (rozmer matice D)
for k:=1 to n do
    for i:=1 to n do
        for j:=1 to n do
            D[i,j]:=min (D[i,j], D[i,k]+D[k,j])
        end for
    end for
end for
```

Výstupem Floyd-Warshallova algoritmu je matice předchůdců. Cesta mezi uzly (i, j) je vyjádřena jako pole s indexem řádek i , sloupec j . Pokud je pole nulové, pak cesta neexistuje, v opačném případě udává předchůdce koncového uzlu j na této cestě. Algoritmus je třeba doplnit o funkci rekonstrukce cesty. Časová složitost Floyd-Warshallova algoritmu je $O(n^3)$.

3.6 Algoritmus A-STAR

Tento algoritmus pro vyhledávání optimálních cest v kladně ohodnocených grafech se řadí mezi algoritmy umělé inteligence. Využívá prioritní frontu podobně jako Dijkstrův algoritmus, ve které jsou ale uzly ohodnoceny podle speciální funkce $f(x) = g(x) + h(x)$, kde funkce $g(x)$ představuje vzdálenost mezi počátečním a konkrétním uzlem a funkce $h(x)$ představuje tzv. heuristickou funkci. Tato funkce vyjadřuje odhad zbylé vzdálenosti z konkrétního uzlu do cíle. Odhad vzdálenosti vzniká na základě alespoň částečné znalosti struktury problému a tento odhad nesmí být větší, než je skutečná vzdálenost do cíle. Jako heuristická funkce může být použita například vzdálenost vzdušnou čarou nebo počet hran mezi daným uzlem a cílem. Během algoritmu je vytvořena prioritní fronta ještě

nenavštívených uzlů. Uzel x má tím vyšší prioritu, čím je jeho funkce $f(x)$ nižší. V každém kroku algoritmu je odebrán z fronty a jsou spočítány hodnoty funkcí $f(x)$ pro jeho sousední uzly a tyto jsou poté přidány do prioritní fronty. Algoritmus končí, pokud je hodnota $f(x)$ koncového uzlu menší než hodnota libovolného uzlu ve frontě nebo pokud je fronta prázdná. Hodnota funkce $f(x)$ koncového uzlu je pak nejkratší cestou grafem. Asymptotická složitost je závislá na použité heuristické funkci, ale obecně není horší než složitost algoritmu prohledávání do šířky, tedy $O(|V|+|E|)$.

3.7 SOMA

SOMA, neboli Samo-Organizující se Migrační Algoritmus, jehož autorem je prof. Ivan Zelinka, se řadí mezi algoritmy evoluční. Tento optimalizační algoritmus slouží k řešení mnoha problémů inženýrské praxe a liší se od předešlých algoritmů, protože neslouží pouze k nalezení nejkratší cesty. Většina těchto problémů (například nalezení optimální dráhy robota, nastavení parametrů regulátoru atd.) může být definována jako optimalizační problém a může být převedena na matematický problém vyjádřený vhodným funkčním předpisem. Optimalizace vede k nalezení argumentů tzv. účelové funkce, tedy k nalezení globálního maxima, respektive minima. Principy SOMA jsou analogií soutěživě - kooperativního chování inteligentních jedinců řešících společný problém. Příkladem takového chování může být hmyzí společenstvo při hledání potravy. Ve fázi spolupráce si jednotlivý jedinci vzájemně vyměňují informace o kvalitě nalezeného zdroje a podle toho se snaží přizpůsobit své chování. Ve fázi soutěžení se snaží každý jednotlivý jedinec zvítězit nad ostatními a nalézt co nejvyšší zdroj potravy. Poté se opět opakuje fáze spolupráce a po výměně informací se určí vůdčí jedinec s nejlepšími výsledky. Ostatní jedinci opustí své zdroje a migrují (opět fáze soutěžení) směrem k vůdci. Tento proces se opakuje, dokud se všichni nesejdou u nejvyššího zdroje potravy. Dále budou vysvětleny některé pojmy.

- Jedinec je složen z množiny argumentů účelové funkce a z hodnoty účelové funkce, která se neúčastní vlastního procesu, ale nese informaci o kvalitě příslušného jedince. Každý jedinec představuje aktuální řešení daného problému.
- Populace je složena z jednotlivých jedinců.
- Speciment je vzor, podle něhož se generuje celá počáteční populace. Speciment slouží také ke korekci jedince při překročení hranic

prohledávaného prostoru. Tvoří jej tři parametry a to typ proměnné a dolní a horní hranice rozsahu, ve kterých se může hodnota pohybovat.

SOMA může používat různé varianty pohybu a ovlivňování jedinců, které se nazývají strategie. Základní verze algoritmu SOMA, strategie AllToOne, se skládá z následujících kroků:

a) Definice parametrů

Nejkritičtější částí optimalizačního procesu je definice účelové funkce, nad kterou bude optimalizace probíhat. Protože rozsah optimalizačních problémů je velký, není možné přesně specifikovat kritéria pro sestavování účelové funkce. Pro správný výběr funkce je obvykle nutná dobrá znalost problému a jisté zkušenosti. Před startem SOMA je nutné definovat řídicí a ukončovací parametry.

- PathLength - určuje, jak daleko se při migraci aktivní jedinec zastaví od vedoucího jedince.
- Step - určuje jednotlivý krok při migraci, tedy rozlišení s jakým bude prostor řešení prohledáván.
- PRT - znamená perturbaci. Podle tohoto parametru se tvoří perturbační vektor, který ovlivňuje dráhu pohybu jedince. Perturbace nahrazuje pojem mutace z klasických evolučních algoritmů, protože nevznikají noví jedinci klasickým „křížením“ dvou jedinců.
- D - počet argumentů účelové funkce, který je dán zadaným problémem.
- NP - počet jedinců tvořících populaci.
- Migrate - udává, kolikrát se populace přeorganizuje.
- MinDiv (Minimal Diversity) - ukončovací parametr, který určuje jaký maximální rozdíl mezi nejlepším a nejhorším jedincem je povolen.

b) Tvorba populace

Populace je vyjádřena jako matice $D \times NP$, kde sloupce představují jednotlivé jedince a řádky jsou argumenty účelové funkce. Každý jedinec také obsahuje hodnotu účelové funkce, která určuje jeho kvalitu. Jednotlivé parametry jedinců prvotní populace se tvoří pomocí specimentu a generátoru náhodných čísel. Speciment určuje typ proměnné a koriguje rozsah hodnoty, kterou může nabývat.

c) Migrační kola

Každý jedinec je ohodnocen účelovou funkcí a je zvolen leader (jedinec s nejlepším výsledkem). Na základě parametru PRT je vytvořen vektor, který určuje směr pohybu jedince. Poté se ostatní jedinci začnou pohybovat směrem k leaderovi pomocí skoků, které jsou určeny parametrem Step. Po každém skoku jedince je přepočítána hodnota účelové funkce jedince, a pokud je lepší než předchozí hodnota, tak je uložena v paměti jedince. Pohyb k leaderovi pokračuje, dokud není dosaženo pozice dané parametrem PathLength. Tento postup nahrazuje křížení, které se používá u klasických evolučních algoritmů. Na konci běhu se vrací jedinec na pozici, kde byla zjištěna nejlepší hodnota účelové funkce a tím v podstatě vzniká nový jedinec. Hodnoty účelové funkce jedinců jsou porovnány a je zvolen nový leader.

d) Testování naplnění ukončovacích parametrů

Zde je testován parametr MinDiv a dále je kontrolováno, zda počet migračních kol dosáhl hodnoty parametru Migrace. V případě nesplnění podmínek se proces vrací do bodu c.

II. PRAKTICKÁ ČÁST

4 PŘÍKLADY PRAKTICKÝCH IMPLEMENTACÍ

Jak vyplývá z níže uvedených zdrojů, u většiny významných komerčních projektů není prakticky možné zjistit použité principy vyhledávání. Tyto informace jsou dostupné především u open-source projektů, výjimečně u komerčních programů a samozřejmě u akademických projektů.

Podle [5] je Bellman-Fordův algoritmus implementován ve směrovacím protokolu RIP (Routing Information Protocol). Ten slouží k řízení komunikace směrovačů zapojených v síti. Při změně topologie sítě je zhruba do 30 sekund algoritmem nalezena nejkratší cesta mezi síťovými prvky a je provedena změna ve směrovací tabulce routeru.

V této aplikaci [12] byl použit Dijkstrův algoritmus pro plánování trasy robota. Pomocí tohoto algoritmu jsou napřed vybrány bezkolizní trasy v prostředí, kterým se robot pohybuje a poté je na tuto množinu možných cest aplikován heuristický algoritmus mravenčí kolonie, který vybere optimální řešení.

Cílem této bakalářské práce [13] bylo vytvořit klientské a serverové programové vybavení pro hledání nejkratší cesty v geografických datech silniční sítě. Projekt rozšiřuje program pgRouting, který mimo jiné využívá Dijkstrův algoritmus a algoritmus A-Star.

V tomto příspěvku [14] je uveden přehled využití algoritmů pro vyhledávání tras v grafech, znázorňujících geografické údaje. Jako online implementace jsou uváděny projekty OpenRouteService, PHPRoute, které využívají algoritmus A-Star, ať už v základní verzi nebo modifikovaný. Dále je opět zmíněn projekt pgRouting, který umožňuje vyhledávání pomocí Dijkstrova algoritmu a algoritmu A-Star. Z off-line implementací jsou uvedeny aplikace pro mobilní telefony Pyroute a Rana, využívající A-Star algoritmus. Další je uvedena aplikace pro přenosné počítače TravellingSalesman používající A-Star a Dijkstrův algoritmus a to klasický nebo upravený pro současné hledání ze startu a cíle. Projekt Navit, což je automobilová navigace, využívá opět Dijkstrův algoritmus. Mezi významnější uvedené aplikace používající A-Star algoritmus patří GpsMid, VGPS a We-Travel pro mobilní telefony a True Maps pro iPhone.

Příspěvek [15] se zabývá použitím Floydova – Warshallova algoritmu pro výpočty propustnosti sítě, vzdáleností na síti a nejspolehlivější trasy sítě.

V této práci [16] se autor zabývá tvorbou programu pro vizualizaci činnosti algoritmů hledání nejkratší cesty grafem. Zpracované jsou Dijkstrův, Bellman-Fordův, Floyd-Warshallův algoritmus.

5 VLASTNÍ IMPLEMENTACE ALGORITMŮ NEJKRATŠÍ TRASY

Některé výše popsané algoritmy pro hledání nejkratší cesty grafem byly naprogramovány v jazyce Java za pomoci vývojového prostředí Eclipse. V této části budou popsány některé vlastnosti programovacího jazyka a vybrané vlastnosti jednotlivých tříd programu. Uvedené pojmy a definice jsou převzaty z [17 a 18].

5.1 Jazyk Java

Firma Sun Microsystems vyvíjela od roku 1991 nový, objektově orientovaný programovací jazyk na principech jazyka C a C++, který byl určen pro použití v embedded systémech. Původně měl projekt název Oak, ale po té, co vývojáři zjistili, že jazyk s takovým názvem už existuje, byl název změněn na Java. Programovací jazyk Java byl oficiálně představen roku 1995 a to především pro využití ve www aplikacích. V roce 2009 firma Oracle koupila firmu Sun Microsystems a s tím získala i technologii Java.

Programy napsané v jazyce Java prochází standardně při vykonávání na počítači pět fází a to editací, kompilací, zavedením, ověřováním a prováděním. Fáze ověřování byla přidána jako nová procedura, která zajišťuje zabezpečení spuštěného programu. Spouštěný program se nepřekládá přímo do strojového jazyka počítače, ale je přeložen do pseudojazyka nazývaného byte-code. Hlavní výhodou tohoto jazyka je to, že program je nezávislý na platformě, na které bude provozován. Přeložený byte-code je uložen v souboru s příponou .class. Při zavádění do paměti počítače je soubor ověřen a poté spuštěn pomocí interpreteru. Pro práci s jazykem Java se používá vývojový nástroj s názvem Java Development Kit (JDK). Podle toho, pro jaké nasazení je určen vyvíjený program, lze získat z www stránek firmy Oracle tři základní typy JDK. Jsou to: Java SE (Standard Edition) - určený pro vývoj běžných aplikací, Java ME (Micro Edition) – určený pro vývoj aplikací pro mobilní zařízení a Java EE (Enterprise Edition) – určený pro nasazení v podnikovém prostředí. Tak jak se vyvíjí jazyk Java, vyvíjí se i jednotlivé verze JDK. Seznam obsahuje jednotlivé verze JDK a jejich hlavní změny.

- JDK 1.0 – původní verze.
- JDK 1.1 – změny jazyka a rozšíření aplikačního programového rozhraní (Application Programming Interface - API), což jsou veřejné, standardní knihovny třídy.

- JDK 1.2 – změny především v API, zejména zařazení některých knihoven, např. knihovna Swing.
- JDK 1.3 – zrychlení, vylepšení a rozšíření části API.
- JDK 1.4 – rozšíření API, urychlení kódu.
- JDK 1.5 – rozšíření jazyka (přidán např. výčtový typ enum) a API.
- JDK 1.6 – rozšíření API.

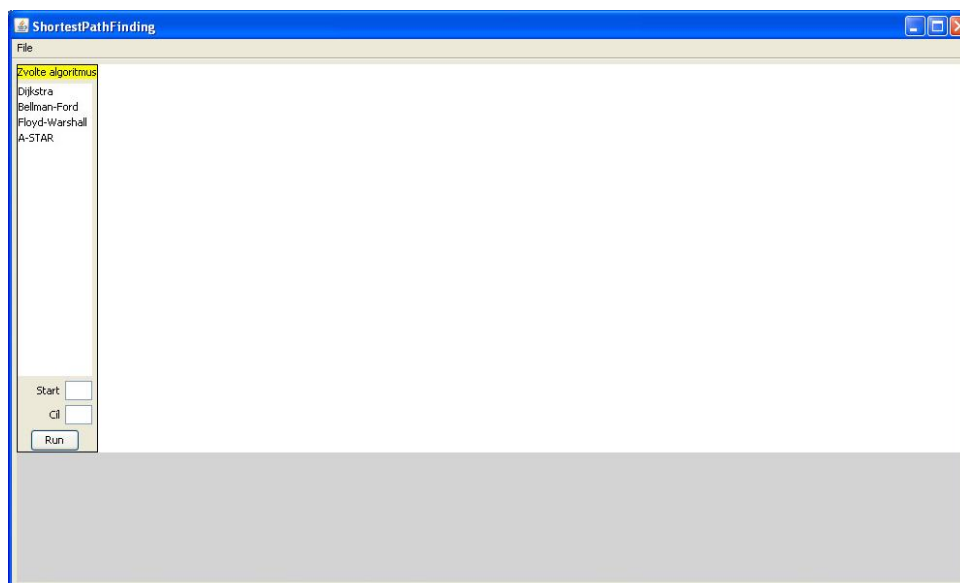
V současné době (květen 2012) je k dispozici vývojové prostředí s označením Java SE Development Kit 7.

5.2 Program pro hledání nejkratší trasy

Zde bude popsán vytvořený program, který implementuje Dijkstrův, Bellman-Fordův, Floyd-Warshallův algoritmus a algoritmus A-star. Program se skládá z následujících tříd, které budou dále stručně popsány.

5.2.1 WindowRun.java

Hlavní třída, která obstarává vzhled a spouštění samotného programu. Jako knihovna grafického rozhraní (GUI) byla použita třída JFC Swing a pro některé komponenty byla použita třída AWT. Hlavní struktura třídy byla vytvořena pomocí nástroje Eclipse WindowBuilder Editoru. Ten umožňuje rozvržení jednotlivých komponent a nastavení jejich vlastností, např. nastavení událostí. Obrázku (obr. 23) ukazuje rozložení komponent grafického rozhraní.



Obr. 23 Grafické rozhraní programu

5.2.2 PanelGraf.java

Tato třída zajišťuje zobrazení grafu do panelu. Ukázka kódu funkce pro uložení hran grafu:

```
public void mojeHrany() {
    hrany.clear();
    if(soused != null) {
        for(int i = 0; i < soused.length; i++) {
            for(int j = 0; j < soused[i].length; j++) {
                if(soused[i][j] != null) {
                    x1=(coordinates[i][0] * pomerX) + (prum / 2);
                    y1=(coordinates[i][1] * pomerY) + (prum / 2);
                    x2=(coordinates[soused[i][j]][0]*pomerX)+(prum / 2);
                    y2=(coordinates[soused[i][j]][1]*pomerY)+(prum / 2);
                    hrany.add(new Edge(x1, y1, x2, y2, Color.LIGHT_GRAY));
                }}}
    }
```

5.2.3 Node.java

Pomocí této třídy se tvoří objekty uzlů, které se ukládají do kolekce. Mimo jiné implementuje reakci na kliknutí myši na uzel, po kterém se zobrazí číslo uzlu.

5.2.4 Edge.java

Jednotlivé objekty této třídy tvoří hrany grafu. Jediná funkce třídy zajišťuje vykreslení hran do grafu:

```
public void draw(Graphics g) {
    g.setColor(this.color);
    g.drawLine(x1, y1, x2, y2);
}
```

5.2.5 Cesta.java

Objekt třídy Cesta.java uloží informace o bodech, kterými prochází nejkratší cesta. K vykreslení cesty je použita funkce drawPolyline().

5.2.6 Open.java

Zajišťuje zobrazení dialogového okna pro otevření souboru. Ukázka funkce load() této třídy:

```
public void load(final int xx, final int yy) {
    JFileChooser fc = new JFileChooser(".\\") {
        protected JDialog createDialog(Component parent) throws HeadlessException
        {
            JDialog dialog = super.createDialog(parent);
            dialog.setLocation(xx + 20, yy + 20);
            return dialog;
        }
    };
    fc.setDialogType(JFileChooser.OPEN_DIALOG);
    fc.setDialogTitle("Otevřít");
    if (fc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
        f = fc.getSelectedFile();
}
```

5.2.7 Reader.java

Objekt třídy Reader.java zajišťuje převedení načteného souboru do kolekce ArrayList, obsahující pole objektů String. Čtení je implementováno pomocí následujícího kódu:

```
public ArrayList<String[]> Reading(File file) {
    BufferedReader br;
    polePoli = new ArrayList<String[]>();
    try {
        FileReader fr = new FileReader(file);
        br = new BufferedReader(fr);
        String s = "";
        while ((s = br.readLine()) != null) {
            polePoli.add(s.split("[;|,]"));
        }
        br.close();
        fr.close();
    } catch (IOException e) {
        System.err.println("chyba");
        e.printStackTrace();
    }
    return polePoli;
}
```

5.2.8 Graf.java

Tato třída zajišťuje zpracování vstupních dat, ze kterých tvoří matici vzdáleností, seznam sousedů a seznam souřadnic jednotlivých bodů. Ukázka funkce pro vytvoření matice vzdáleností:

```

public Float[][] setMatrix() {
    maticeVzdalenosti = new Float[data.size()][data.size()];
    int sloupec;
    int hodnota;
    String zero = "";
    for(int i = 0; i < data.size(); i++)
        for(int k = 0; k < (max - 3)/2; k++) {
            if(!data.get(i)[k + 3].equals(zero)) {
                sloupec=Integer.parseInt(data.get(i)[k + 3]);
                hodnota=Integer.parseInt(data.get(i)[k + 3 + ((max-3)/2)]);
                maticeVzdalenosti[i][sloupec] = (float)hodnota;
            }
        }
    for(int i = 0; i < maticeVzdalenosti.length; i++)
        for(int j = 0; j < maticeVzdalenosti[i].length; j++) {
            if(maticeVzdalenosti[i][j] == null)
                if(i == j)
                    maticeVzdalenosti[i][j] = 0f;
                else
                    maticeVzdalenosti[i][j] = Float.POSITIVE_INFINITY;}
    return maticeVzdalenosti;}

```

5.2.9 Dijkstra.java

Za pomoci této třídy se provádí Dijkstrův algoritmus. Obsahuje funkci pro samotný algoritmus a funkci pro rekonstrukci nejkratší cesty. Ukázka kódu pro relaxaci hran:

```

for(int i = 0; i < matice.length; i++) {
    if(matice[uzel][i] != Float.POSITIVE_INFINITY) {
        if(!trvale[i]) {
            if(hodnotaUzlu[i] > hodnotaUzlu[uzel] + matice[uzel][i]) {
                hodnotaUzlu[i] = hodnotaUzlu [uzel] + matice[uzel][i];
                predchudci[i] = uzel;
                kandidati.add(i);}}}}

```

5.2.10 Bellman.java

Provádí Bellman-Fordův algoritmus. Implementuje funkci pro relaxaci hran ze všech uzlů a funkci pro rekonstrukci cesty.

5.2.11 Floyd.java

Tato třída zajišťuje provádění Floyd-Warshallova algoritmu. Ukázka kódu funkce pro upravování vzdáleností v matici předchůdců:

```
for(int k = 0; k < graMat.length; k++)
    for(int i = 0; i < graMat.length; i++)
        for(int j = 0; j < graMat.length; j++) {
            if(graMat[i][j] >= graMat[i][k] + graMat[k][j]) {
                maticePredchudcu[i][j] = maticePredchudcu[i][j];
            } else { maticePredchudcu[i][j] = maticePredchudcu[k][j]; }
            graMat[i][j] = Math.min(graMat[i][j], graMat[i][k]+graMat[k][j]);}
```

5.2.12 AStar.java

Objekt této třídy vykonává A-star algoritmus.

5.3 Ovládání programu

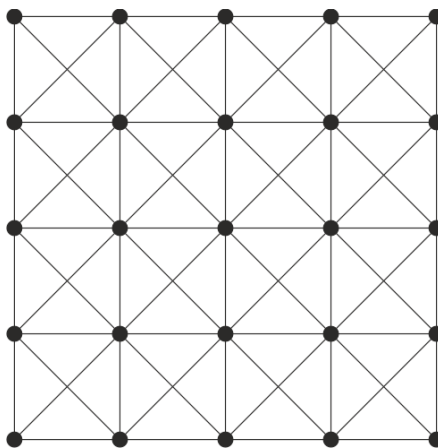
Položka menu File, obsahuje položky Open, kterou se otevírá dialogové okno pro výběr souboru grafu a položka Exit, kterou se program ukončí. Soubory, obsahující informaci o grafu jsou uložena ve formátu csv. Informace o jednotlivých uzlech grafu jsou uloženy po řádcích a jednotlivá data v řádku jsou oddělená středníkem. První sloupec značí číslo uzlu, další dva sloupce obsahují souřadnice uzlu a další sloupce obsahují sousedící uzly a délku cesty, která do nich vede. Po načtení souboru se zvolí typ algoritmu, zadají se požadované vstupní parametry a algoritmus se spustí tlačítkem Run. Po skončení běhu algoritmu se ve spodní části okna vypíše informace o pořadí uzlů, kterými prochází cesta, hodnota délky cesty a doba běhu algoritmu.

5.4 Testování algoritmů

Algoritmy byly testovány pomocí osobního počítače s následující konfigurací:

CPU Intel E6850 3GHz, operační paměť RAM 4 GB DDR3 1066 MHz, operační systém Windows 7 32 bit.

Testování algoritmů bylo provedeno na vzorových grafech, které byly reprezentovány čtvercovou sítí, jak ukazuje obrázek (obr. 23), kde byly všechny sousedící body propojeny mezi sebou, a každá hrana měla velikost 2. Byly postupně testovány grafy o rozměrech 5x5, 10x10, 20x20, 30x30 a 40x40, tzn., že počet vrcholů byl 25, 100, 400, 900 a 1600.



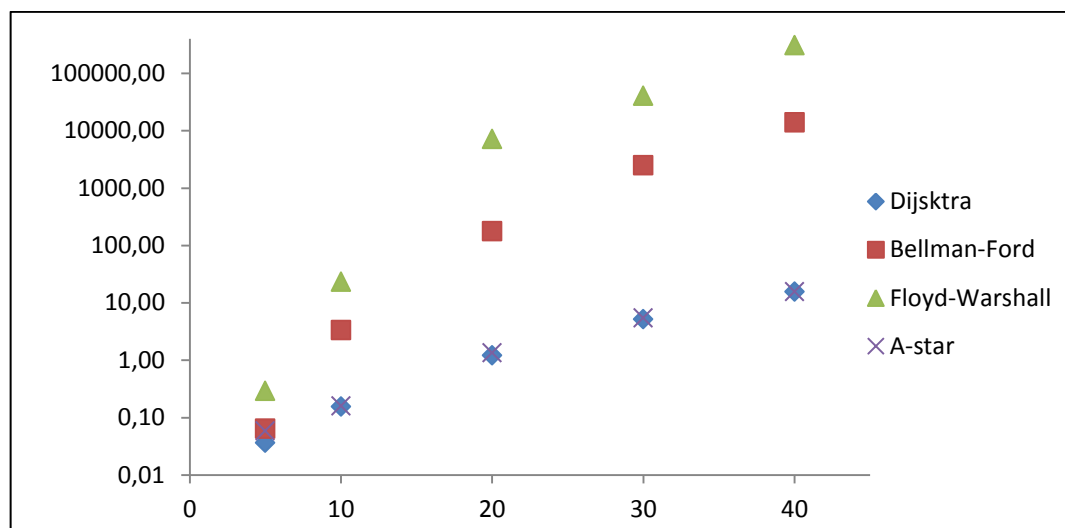
Obr. 24 Graf 5x5

U každého grafu a typu algoritmu bylo provedeno deset po sobě jdoucích měření. Výsledky byly zprůměrovány a hodnoty byly zaneseny to tabulky (tab. 1). V tabulce jsou uvedeny rozměry hrany testovacího grafu a doba běhu každého algoritmu v milisekundách.

Velikost hrany grafu	Doba běhu [ms]			
	Dijkstra	Bellman-Ford	Floyd-Warshall	A-star
5	0,04	0,06	0,29	0,06
10	0,15	3,34	23,2	0,16
20	1,21	176	7176	1,34
30	5,19	2497	40353	5,43
40	15,5	13985	307473	15,5

Tab. 1 Výsledky testování algoritmů

Naměřené hodnoty byly zaneseny do grafu (obr. 25). Na vodorovné ose je vyznačen rozměr grafu a na svislé ose je vynesena čas v milisekundách v logaritmickém měřítku. Na grafu je možné pozorovat rozdílnou časovou náročnost jednotlivých algoritmů.



Obr. 25 Graf závislosti doby běhu algoritmu

ZÁVĚR

Bakalářská práce se zabývá algoritmy, které slouží k hledání nejkratší cesty grafem. Tyto algoritmy se používají například při plánování nejrychlejší trasy, při řízení provozu počítačových sítí nebo při řízení pohybu robotů.

Bylo provedeno shrnutí teoretických poznatků, pojednávajících o problematice teorie grafů a hledání nejkratší cesty v grafu. V této části byly popsány různé typy grafů, způsoby prezentace grafů, vlastnosti algoritmů všeobecně a časová náročnost algoritmů. Větší část teoretické části se zabývá popisem algoritmů, sloužících k hledání nejkratší cesty v grafu. Byl vysvětlen princip Dijkstrova, Bellman-Fordova, Floyd-Warshallova algoritmu, algoritmu A-star a samo-organizujícího se migračního algoritmu.

V praktické části byly popsány příklady praktických použití těchto algoritmů. Vzhledem k tomu, že u komerčních projektů se nezveřejňují použité postupy, byly popsány především akademické projekty a projekty typu open-source. Dále byly popsány jednotlivé třídy programu, který implementuje Dijkstrův, Bellman-Fordův, Floyd-Warshallův algoritmus a algoritmus A-star. V posledním bodu praktické části bylo provedeno měření doby běhu jednotlivých algoritmů a tyto výsledky byly zobrazeny v grafu.

Výsledky této práce byly také publikovány v rámci [19] a [20].

CONCLUSION

This thesis deals with algorithms, which are used to finding the shortest path in a graph. These algorithms are used for example in planning the quickest route, the traffic management of computer networks or the robot motion control. There was performed a summary of theoretical knowledge that dealing with the problem of graph theory and finding the shortest path in the graph. This part of the thesis describes the different types of graphs, different way of presenting a graph, general way of describing the properties of algorithms and time-consuming of algorithms. Most of the theoretical part deals with the description of algorithms used for finding the shortest path in graph. There was explained principle of the Dijkstra's, the Bellman-Ford, the Floyd-Warshall algorithm, the algorithm A-star and the self-organizing migration algorithm.

In practical part were described examples of practical application of these algorithms. Whereas that the commercial projects are not published procedures, there were reported mainly academic projects and projects of open-source type. Furthermore, there were reported individual classes of program, which implements the Dijkstra's, the Bellman-Ford, the Floyd-Warshall algorithm and the algorithm A-star. In the last point of practical part were carried out measurements of running time of individual algorithms and the results were shown in graph.

The results of this thesis were also published in [19] and [20].

SEZNAM POUŽITÉ LITERATURY

- [1] MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. *Kapitoly z diskrétní matematiky*. 3., upr. a dopl. vyd. V Praze: Karolinum, 2007, 423 s. ISBN 978-802-4614-113.
- [2] KOLÁŘ, Josef. *Teoretická informatika*. Vyd. 1. V Praze: České vysoké učení technické, 2009, 206 s. ISBN 978-80-01-04331-8.
- [3] DEMLOVÁ, Marie a Bedřich PONDĚLÍČEK. *Matematická logika*. Vyd. 1. Praha: ČVUT, 1997, 160 s. ISBN 80-010-1683-8.
- [4] Základní grafové algoritmy. ČERNÝ, Jakub. Katedra aplik. matematiky [online]. 24. 10. 2010 [cit. 2012-05-02]. Dostupné z: <http://kam.mff.cuni.cz/~kuba/ka>.
- [5] JÁGR, Petr. Hledání nejkratších cest grafem [online]. Brno, 2007 [cit. 2012-05-02]. Dostupné z: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=4340>. Bakalářská práce. VUT Brno.
- [6] CORMEN, Thomas H., Charles E. LEISERSON, Ronald L. RIVEST a Clifford STEIN. *Introduction to algorithms*. 2nd. ed. Cambridge (Massachusetts): MIT Press, 2007. ISBN 02-625-3196-8.
- [7] TÖPFER, Pavel. *Algoritmy a programovací techniky*. 2. vyd. Praha: Prometheus, 2007c1995, 300 s. ISBN 978-80-7196-350-9.
- [8] VIRIUS, Miroslav. *Základy algoritmizace*. Vyd. 2., přeprac. Praha: Česká technika - nakladatelství ČVUT, 2008, 265 s. ISBN 978-80-01-04003-4.
- [9] Algoritmy [online]. [2012] [cit. 2012-05-02]. Dostupné z: <http://www.algoritmy.net>
- [10] Algoritmus A-Star. WOHO [online]. [2012] [cit. 2012-05-02]. Dostupné z: <http://voho.cz/wiki/informatika/algoritmus/graf/a-star/>
- [11] ZELINKA, Ivan. *Umělá inteligence v problémech globální optimalizace*. 1. vyd. Praha: BEN - technická literatura, 2002, 189 s. ISBN 80-730-0069-5.
- [12] TAN, Guan-zheng, Huan HE a Sloman AARON. Global optimal path planning for mobile robot based on improved Dijkstra algorithm and ant system algorithm. In: JOURNAL OF CENTRAL SOUTH UNIVERSITY OF TECHNOLOGY [online]. 2006 [cit. 2012-05-02]. Dostupné z: <http://www.springerlink.com/content/x2222w55g44m4g03/>

- [13] GRMELA, Jan. Aplikace algoritmu hledání nejkratší cesty v geografických datech silniční sítě České republiky [online]. Brno, 2009 [cit. 2012-05-02]. Dostupné z: <https://akela.mendelu.cz/~xgrmela1/gpl-sw/mroute/dokumentace/bakalarska-prace.dokument.pdf>. Bakalářská práce. Mendelova zemědělská a lesnická univerzita.
- [14] MAREK, Martin. Geografický routing. In: Celostátní konference TVORBA SOFTWARE 2010 [online]. 2010 [cit. 2012-05-02]. ISBN 978-80-248-2225-9. Dostupné z: <http://formular-ekf.vsb.cz/formulare/F01/tsw/?page=prispevek&rok=2010&prispevekid=980>
- [15] MOCKOVÁ, Denisa. VYUŽITÍ FLOYDOVA ALGORITMU NA SITÍCH. In: Perner's Contacts [online]. 2011 [cit. 2012-05-02]. ISSN 1801-674x. Dostupné z: http://pernerscontacts.upce.cz/23_2011/Mockova.pdf
- [16] HAVRÁNEK, Tomáš. Vizualizace hledání nejkratší cesty na grafech [online]. Pardubice, 2010 [cit. 2012-05-02]. Dostupné z: <http://dspace.upce.cz/handle/10195/37529>. Diplomová práce. Univerzita Pardubice.
- [17] HEROUT, Pavel. Učebnice jazyka Java. 5., rozš. vyd. České Budějovice: Kopp, 2010, 386 s. ISBN 978-80-7232-398-2.
- [18] HEROUT, Pavel. Java: grafické uživatelské prostředí a čeština. 2. vyd. České Budějovice: Kopp, 2007, 316 s. ISBN 978-80-7232-328-9.
- [19] VAŘACHA, Pavel a Michal KOPŘIVA. Algoritmy hledání nejkratší cesty. In: Logistika v teorii a praxi III. Uherské Hradiště, 2011, s. 7. ISBN 978-80-7454-126-1.
- [20] VAŘACHA, Pavel a Michal KOPŘIVA. Vybrané grafové algoritmy. In: Internet, Competitiveness and Organizational Security. Zlín, 2012, s. 6. ISBN 978-80-7454-142-1.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

V	Množina vrcholů.
E	Množina hran.
u	Vrchol grafu.
v	Vrchol grafu.
S	Posloupnost uzlů.
n	Počet vrcholů.
c	Časová konstanta.
Ω	Dolní hranice chování.
Θ	Třída chování.
O	Horní hranice chování.
SOMA	Samo-organizující se migrační algoritmus.
BFS	Breadth-first search.
DFS	Depth-first search.
D	Matice délek.
PRT	Perturbace.
D	Počet argumentů účelové funkce.
NP	Počet jedinců.
MinDiv	Minimal Diversity.
RIP	Routing Information Protocol.
JDK	Java Development Kit.
GUI	Graphical User Interface.
AWT	Abstract Windows Toolkit.
CPU	Central Processing Unit.
API	Application Programming Interface.

SEZNAM OBRÁZKŮ

Obr. 1 Problém sedmy mostů	11
Obr. 2 Graf problému z obrázku 1	11
Obr. 3 Neorientovaný a orientovaný graf	12
Obr. 4 Násobné hrany a smyčka v grafu	12
Obr. 5 Cyklus grafu	13
Obr. 6 Regulární graf	13
Obr. 7 Isomorfní grafy	13
Obr. 8 Doplněk grafu	14
Obr. 9 Podgraf grafu	14
Obr. 10 Strom	14
Obr. 11 Reprezentace grafu seznamem hran	15
Obr. 12 Matice sousednosti	16
Obr. 13 Matice vzdáleností	16
Obr. 14 Seznam sousedů	17
Obr. 15 Vývojový diagram	19
Obr. 16 Třída složitosti theta	20
Obr. 17 Třída složitosti omikron	20
Obr. 18 Třída složitosti omega	21
Obr. 19 Prohledávání do šířky – čísla označují vlnu, ve které je uzel zpracován	23
Obr. 20 Procházení grafu do hloubky	24
Obr. 21 Princip Dijkstrova algoritmu	25
Obr. 22 Princip Bellman – Fordova algoritmu	26
Obr. 23 Grafické rozhraní programu	35
Obr. 24 Graf 5x5	40
Obr. 25 Graf závislosti doby běhu algoritmu	41

SEZNAM TABULEK

Tab. 1 Výsledky testování algoritmů	40
---	----

SEZNAM PŘÍLOH

P I DVD médium obsahující zdrojové kódy a text BP.