UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Martin Hroch

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Algoritmy pro hledání nejkratších cest v grafu

Martin Hroch

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Hroch**
Osobní číslo:	**I13132**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Algoritmy pro hledání nejkratších cest v grafu**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Grafem může být znázorněna například silniční, nebo železniční síť. Celkovým cílem práce rešerše a implementace grafových algoritmů na nalezení nejkratší cesty v grafu.
Cílem teoretické části bude rešerše algoritmů pro nalezení nejkratší cesty v grafu (orientovaném, či neorientovaném). Pro hledání nejkratší cesty bylo popsáno několik algoritmů (např. prohledávání grafu do šířky, prohledávání do hloubky, Dijsktrův algoritmus či heuristický algoritmus A star) Algoritmy budou srovnány z hlediska použitelnosti pro jednotlivé typy grafů, paměťové a časové náročnosti.
Cílem praktické části bude vytvoření aplikací pro jednotlivé algoritmy včetně návrhu vhodných datových struktur pro práci s grafy. Algoritmy budou otestovány na konkrétních datech. Aplikační část lze vytvořit např. ve vývojové prostředí Javy, popř. jiné podobné aplikaci.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

BALAKRISHNAN, V. Schaum's outline of theory and problems of graph theory. New York: McGraw-Hill, c1997, viii, 293 p. ISBN 00-700-5489-4.
TÖPFER, Pavel. Algoritmy a programovací techniky. 1. vyd. Praha: Prometheus, 1995, 299 s. ISBN 80-858-4983-6.
MATOUŠEK, Jiří. Kapitoly z diskrétní matematiky. Vyd. 1. Praha: Karolinum, 2002, 381 s. ISBN 80-246-0084-6.

Vedoucí bakalářské práce: **RNDr. Josef Rak, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **31. října 2016**
Termín odevzdání bakalářské práce: **12. května 2017**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 07. 12. 2017

Martin Hroch

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu této bakalářské práce RNDr. Josefu Rakovy Ph.D. za cenné rady, připomínky a vstřícný přístup při zpracování této práce. Dále bych rád poděkoval mým blízkým za jejich čas, pochopení a podporu při psaní bakalářské práce.

ANOTACE

Cílem bakalářské práce je vytvoření softwaru na hledání nejkratších cest v grafu pomocí základních algoritmů. V práci jsou popsány základní pojmy týkající se grafů a popsán Dijsktrův algoritmus, A-star algoritmus a Floyd-Warshallův algoritmus.

KLÍČOVÁ SLOVA

grafy, algoritmy, vrcholy, hrany, Dijsktra, nejkratší

TITLE

Algorithms for finding the shortest paths in a graph.

ANNOTATION

The aim of the bachelor thesis is to create software for finding the shortest paths in the graph using basic algorithms. The thesis describes basic concepts related to graphs and describes Dijskt's algorithm, A-star algorithm and Floyd-Warshall algorithm.

KEYWORDS

graphs, algorithms, nodes, edges, Dijkstra, shortest

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

0 ÚVOD

Hledání nejkratší vzdálenosti je využívané v mnoha odvětvích informatiky. Nalezneme jej v aplikacích, pro cestování využívajících mapové podklady, určených pro navigování. Využití je i v drážních systémech pro řízení vlakové dopravy, pro sestavování jízdních řádů a v neposlední řadě pro nalezení vhodného vlakového spoje. Souvisejícím a mnohdy propojeným odvětím je poté doprava silniční. Dalším odvětvím jsou hry, kvalitní hra má živé a interaktivní prostředí a takové prostředí potřebuje nějaký algoritmus pro pohyb po mapě, po terénu a podobě.

V případě drážní dopravy vrcholy grafu představují jednotlivý výhybky a stanice a hrany koleje mezi nimi. V případě silniční dopravy vrcholy představuji vrcholy křižovatky a hrany silnice mezi nimi. U her je situace složitější a liší se od použité technologie. Nejjednodušším příkladem jsou pixely u 2D her, složitějším příkladem může být mapa rozdělená na hexy (hex je pravidelný šestiúhelník).

Tato práce se dělí do teoretické a praktické části. V teoretické části jsou popsány základní pojmy pojící se s grafy a fungování vybraných algoritmů. V praktické části je vytvořen program, sloužící pro hledání nejkratší cesty pomocí vybraných algoritmů. Program umožňuje načítání dat z textového souboru v zadaném formátu a práci nad předpřipravenými daty. Výstupem programu je obvykle nejkratší cesta mezi vybranými vrcholy.

V úvodní části práce dochází k popsání základních pojmů pojících se s grafy. Mezi které patří popsání, co představuje graf a jaké máme druhy grafů. Poté jsou popsány jednotlivé dílčí části grafu a na konec je popsáno jakým způsobem je možné reprezentovat graf.

V následující části je krátce popsána historie teorie grafů a poté jsou popsány vybrané operace, jenž můžeme nad grafem provádět. Následuje popis datových struktur, jenž používám nebo jsem vytvořil pro potřeby programu.

Další kapitola popisuje samotný vývoj aplikace, jsou zmíněny programy, které jsem využíval a také formát v jakém načítám a ukládám grafy. Po této kapitole následuje popis částí grafického rozhraní, jenž jsem vytvořil pro usnadnění práce s programem.

V poslední části jsou poté popsány výsledky testování jednotlivých algoritmů. Jejich srovnání z hlediska časové náročnosti a paměťové náročnost a z hlediska náročnosti na vývoj.

1 ASYMPTOTICKÁ SLOŽITOST

Pro účely porovnání jednotlivých algoritmů, byl vymyšlen způsob, který nám ukazuje, jak je daný algoritmus rychlý. Tento pojem se nazývá asymptotická složitost. [1]

Definice 1.1 Asymptotická složitost vyjadřuje růst náročnosti algoritmu (počet provedených elementárních operací) s rostoucím množstvím vstupních dat. [2]

Definice 1.2 Formální definice asymptotické složitosti

Nechť g je funkce. Pak definujeme množinu funkcí O(g(n)) následovně: [2]

$$\boldsymbol{O(g(n)) = \{f \mid \exists c > 0, n_0 \in \mathbb{N} : \forall n > n_0 : |f(n)| \leq |c \cdot g(n)|\}}$$

Rozlišujeme několik tříd složitostí. Tyto složitosti začínají na $O(1)$ a končí na $O(n!)$. Celý výčet je zobrazen v následující tabulce. Kde jsou seřazeny od nejrychlejších k nejpomalejším.

Tabulka 1 - Nejčastější třídy složitosti [2]

Asymptotická složitost	**Vyjádření**	**Popis a nejčastější případ**
Konstantní	$O(1)$	Počet operací pro libovolně velká data je stejný. Typicky třeba přístup k paměti nebo zjištění sudosti čísla.
Logaritmická	$O(\log n)$	Typickým příkladem je binární vyhledávání
Lineární	$O(n)$	Složitost se zvyšuje podobně jako velikost dat. Příkladem může být vyhledání maxima v neseřazeném poli.
Lineárně logaritmická	$O(n \cdot \log(n))$	Řazení polí reálných čísel algoritmem merge sort.
Kvadratická	$O(n^2)$	Řazení polí algoritmem bubble sort
Polynomiální	$O(n^k), k \in R$	Floydův algoritmus, násobení matic
Exponenciální	$O(k^n), k \in R$	Problém obchodního cestujícího
Faktoriálová	$O(n!)$	Obvykle složitost brute-force algoritmů. Kdy se hledají veškeré permutace *n* prvků.

2 GRAF A JEHO ČÁSTI

V této části popíšu jednotlivé pojmy, jenž se pojí s grafy a jeho částmi a druhy. Vyjmenuji vybrané druhy grafů a vybrané druhy částí grafu. Dále popíšu, jakými způsoby lze graf reprezentovat. Tedy jak jej lze zobrazit nebo uložit.

Definice 2.1 Graf definujeme jako uspořádanou trojici $G = (V, H, f)$ kde V představuje neprázdnou konečnou množinu vrchol (uzlů), H představuje konečnou množinu hran a f představuje incidence. Incidencí rozumíme přiřazení každé hraně z množiny H uspořádanou dvojici vrcholů z množiny V. [3]

Definice 2.2 Hranou rozumíme spojnici mezi dvěma vrcholy.

Definice 2.3 Vrcholem rozumíme množinu začátků a konců hran.

2.1 Hrana

Hrana může být orientovaná nebo neorientovaná. Vzhledem k ostatním vrcholům dále rozlišujeme smyčky, rovnoběžné hrany a násobné hrany. Kdy každá hrana může být označena více pojmy.

Definice 2.1.1 Smyčka je taková hrana, jenž vede z jednoho vrcholu do toho stejného vrcholu. [4]

Definice 2.1.2 Rovnoběžné hrany jsou takové hrany, jenž všechny spojují stejnou dvojici vrcholů. [4]

Definice 2.1.3 Násobné hrany jsou takové hrany, jež jsou rovnoběžné a jsou buďto neorientované nebo souhlasně orientované (vedou ze stejného výchozího uzlu do stejného cílového uzlu). [5]

2.2 Vrchol

Vrchol nebo také uzel nám představuje objekt, jenž může být hranou spojen s jiným vrcholem. Dva vrcholy spojené hranou se nazývají sousední vrcholy (uzly). Přiřazení hrany uzlu se označuje pojmem *incidence*. [6]

Definice 2.2.1 Stupeň vrcholu je číslo vyjadřující počet hran, s kterým uzel inciduje. Značí se |x|. Pokud se |x|=0 tak se jedná o izolovaný uzel. [5]

Definice 2.2.2 Izolovaný vrchol je takový vrchol, z kterého nevede žádná cesta a zároveň do něj nevede žádná cesta. Tedy neinciduje s žádnou hranou. Jeho stupeň je roven nule. [6]

2.3 Graf

Grafem rozumíme množinu vrcholů a hran. Kde hrany spojují příslušné vrcholy. Jednotlivé hrany mohou být orientované nebo neorientované. V případě, že je hrana orientovaná tak má začátek a konec. V případě, že je neorientovaná tak se začátek a konec nerozlišuje. [7]

Definice 2.3.1 Graf je dvojice: [4]

$$G = (V, E); \forall e \in E : e = (a, b); a, b \in V$$

- V je neprázdná množina uzlů (disjunktní s množinou E)
- E je množina hran (disjunktní s množinou V)

Definice 2.3.2 Graf je trojice: [4]

$$G = (V, E, R); R : E \to V \times V$$

- V je neprázdná množina uzlů (disjunktní s množinou E)
- E je množina hran (disjunktní s množinou V)
- R je incidenční relace (zobrazení z množiny hran do množiny uzlů)

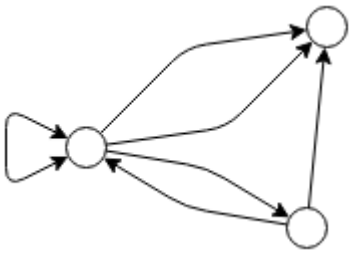
Dle orientace hran se poté odvíjí i to, zdali bude graf orientovaný nebo neorientovaný. Orientovaný graf obsahuje orientované hrany a neorientovaný graf obsahuje pouze neorientované hrany. [7]

Dalším pojem, jenž rozlišujeme je ohodnocení grafu. Každá hrana v grafu může být ohodnocena. Pokud je každá z hran v grafu ohodnocena pak mluvíme o ohodnoceném grafu. Ohodnocené mohou být i vrcholy. V mnoha algoritmech v průběhu jeho provádění postupně jednotlivé vrcholy ohodnocujeme. [5]

Mimo těchto hodnot můžeme sledovat množství hran v grafu množství hran vedoucích z nebo do vrcholu. Popřípadě zdali graf obsahuje rovnoběžné hrany nebo smyčky. Popřípadě celkovou strukturu rozložení hran a vrcholů. Kombinací těchto pojmů nám poté vznikají různé druhy grafů.

Definice 2.3.3 Multigraf je takový graf, ve kterém jsou dva různé uzly spojeny více různými hranami a může obsahovat i smyčky. [4]

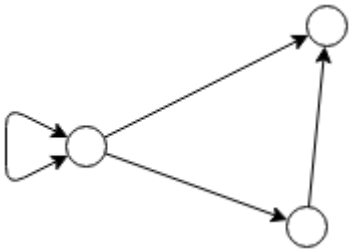
Jednu z mnoha možností zobrazuji i na následujícím obrázku multigrafu (Obrázek 1).



Obrázek 1 – Multigraf [zdroj: autor]

Definice 2.3.4 Prostý graf je takový graf, který neobsahuje žádné rovnoběžné hrany, ale může obsahovat smyčky. [4]

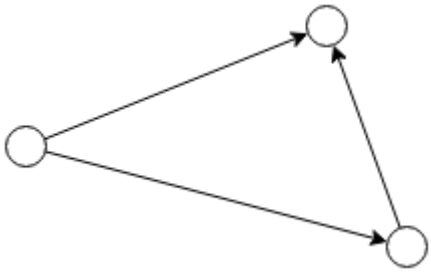
Což je názorně zobrazeno na obrázku prostého grafu (Obrázek 2).



Obrázek 2 - Prostý graf [zdroj: autor]

Definice 2.3.5 Obyčejný graf je takový graf, kde mezi dvěma různými uzly existuje nejvýše jedna hrana a neobsahuje žádnou smyčku. [5]

Pro lepší představu je takový graf zobrazen na obrázku jednoduchého grafu (Obrázek 3).



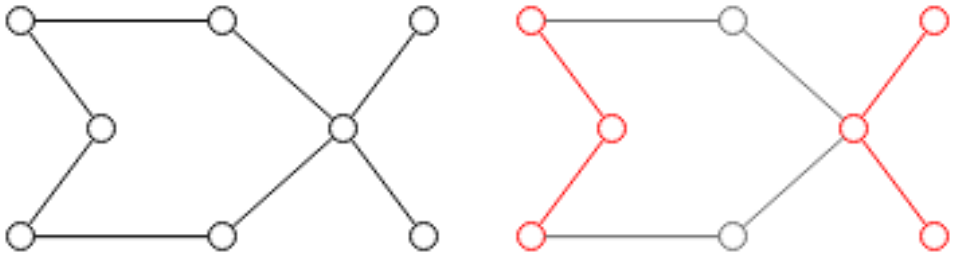
Obrázek 3 – Obyčejný graf [zdroj: autor]

Definice 2.3.6 Pseudograf je takový graf, který obsahuje smyčky a dva různé uzly můžou být spojeny více jak jednou hranou. Pseudograf je neobecnějším typem neorientovaného grafu. [8]

Definice 2.3.7 Úplný graf je graf, jehož každé dva různé vrcholy jsou spojeny hranou. Tedy každý vrchol je spojen se všemi ostatními vrcholy. [5]

Definice 2.3.8 Podgraf takový graf, jenž vznikl odebráním aspoň jednoho vrcholu nebo hrany z původního grafu. Pokud je odebrán uzel je také potřeba odebrat veškeré uzly vedoucí do nebo z tohoto vrcholu. [9]

Následující obrázek (Obrázek 4) zobrazuje výchozí graf a červeně jsou vyznačené možné podgrafy:



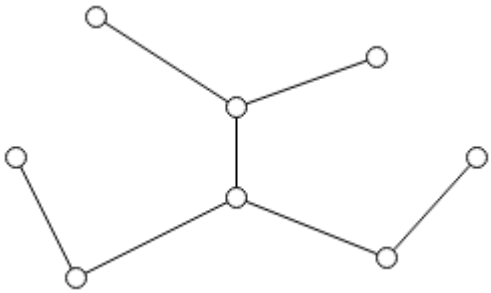
Obrázek 4 - Podgraf a) výchozí graf, b) podgrafy [zdroj: autor]

Definice 2.3.9 Orientovaný graf je takový, jehož hrany mají svůj pevně daný začátek a konec. Tedy každá hrana je orientovaná. [10]

Definice 2.3.10 Neorientovaný graf je takový graf, v kterém hrany nemají určený svůj začátek a konec, pouze spojují dva uzly. [7]

Definice 2.3.11 Strom je souvislý graf neobsahující kružnici. V kterém mezi dvěma vrcholy existuje právě jedna a maximálně jedna cesta. V takovémto grafu nemůže nastat situace kdy, lze do z jednoho vrcholu do druhého nalézt více cest. [11]

Jak takový strom může vypadat zobrazuje následující obrázek grafového stromu (Obrázek 5).



Obrázek 5 – Strom [zdroj: autor]

Stromy se často v různých modifikací používají jako datové struktury. Nejznámějším příkladem může být binární vyhledávací strom. Tento strom je typický tím, že z jednoho

vrcholu vedou nejvýše tři cesty. Jedna nahoru z tzv. předkovy (otci) a dvě dolů k tzv. potomkům (synům). Ve stromě platí, že levý potomek je menší nebo roven předkovy a pravý je větší než předek. Důvodem pro zavedení tohoto stromu je rychlost vyhledávání. [11]

Definice 2.3.12 „**Sled** *je uspořádaná posloupnost uzlů a hran (uzly a hrany se mohou opakovat).“* [5]

Definice 2.3.13 Délka sledu představuje počet hran obsažených ve sledu. [7]

Definice 2.3.14 Tah je takový sled, v kterém je každý hrana obsažena pouze jednou. [5]

Definice 2.3.15 Eulerův tah je takový tah v kterém jsou všechny hrany obsaženy právě a pouze jednou. [4]

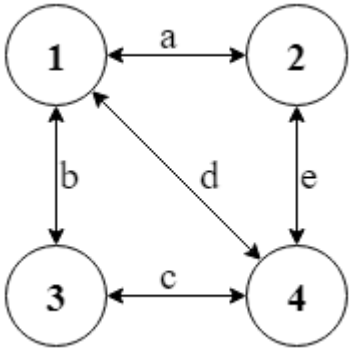
Definice 2.3.16 Cesta je takový tah, v kterém je každý vrchol obsaženou pouze jednou. [5]

Definice 2.3.17 Kružnice je cesta, jenž začíná a končí ve stejném uzlu. [4]

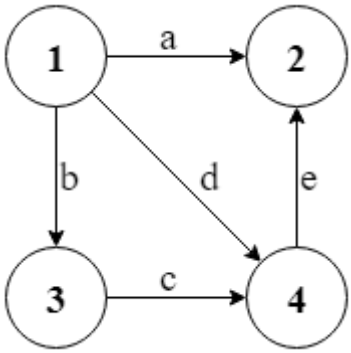
V případě orientovaného grafu mluvíme o ***cyklu*.** [4]

2.4 Reprezentace grafů

Grafy je možné reprezentovat několika způsoby. Základní reprezentací je 2D grafická podoba grafu, kdy vrchol je obvykle značen kruhem a hrana je značena šipkou. Tento typ reprezentace je ovšem špatně čitelný pro počítače. Graf se tedy dá vyjádřit i jinými způsoby nejčastějším způsobem je zapsání do matice. Rozlišujeme několik druhů, mezi nejvýznamnější patří matice sousednosti a matice incidence. Dalším možným vyjádřením grafu je spojový seznam. V následující kapitole budu rozebírat ostatní možnosti reprezentace grafů. A k tomuto účelu budu používat Obrázek 2 pro neorientovaný graf a Obrázek 3 pro orientovaný graf.



Obrázek 6 - Neorientovaný graf [zdroj: autor]



Obrázek 7 - Orientovaný graf [zdroj: autor]

2.4.1 Matice sousednosti

Matice sousednosti reprezentuje graf pomocí N x N matice. Kde N vyjadřuje počet vrcholů. Matice sousednosti je tedy čtvercovou maticí. V řádcích i sloupcích jsou zapsány jednotlivé vrcholy. Pokud mezi danou dvojicí vrcholů vede cesta tak zapíšu 1. Pokud mezi danými uzly cesta nevede zapíši 1. Matice je pro neorientované grafy symetrická podle diagonály. [7]

Pro lepší představu matice sousednosti výše uvedeného neorientovaného grafu (Obrázek 6):

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

A pro porovnání i matice sousednosti výše uvedeného orientovaného grafu (Obrázek 7):

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

2.4.2 Matice incidence

Matice incidence reprezentuje graf pomocí N x M matice. Kde řádky matice představují uzly grafu a sloupce matice představují hrany grafu. Pokud z uzlu m vede cesta n tak v matici se zapíše 1 pokud cesta n vede do uzlu m tak se zapíše -1 a pokud z daného uzlu m cesta n nevede se zapíše 0. V případě neorientovaného grafu se zapisuje 1 tam kde hrana inciduje s vrcholem a 0 tam kde neinciduje. Záporné hodnoty se tedy v neorientovaném grafu neuvádí, jelikož není rozlišen začátek a konec. [7]

Pro lepší představu matice incidence výše uvedeného neorientovaného grafu (Obrázek 6):

$$\begin{pmatrix} & a & b & c & d & e \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 0 & 1 \\ 3 & 0 & 1 & 1 & 0 & 0 \\ 4 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

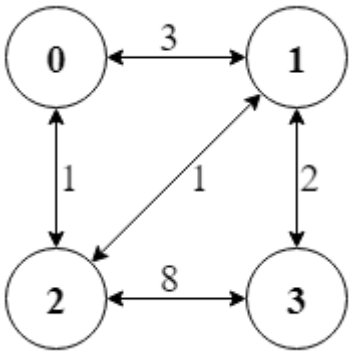
A pro porovnání i matice sousednosti výše uvedeného orientovaného grafu (Obrázek 7):

$$\begin{pmatrix} & a & b & c & d & e \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 2 & -1 & 0 & 0 & 0 & -1 \\ 3 & 0 & -1 & 1 & 0 & 0 \\ 4 & 0 & 0 & -1 & -1 & 1 \end{pmatrix}$$

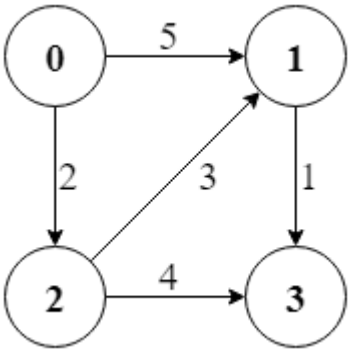
2.4.3 Matice přímých vzdáleností

Další z možností, jak uložit data v matici je pomocí matice přímých vzdáleností. Tato matice je velikosti N x N. Kde N je počet uzlů. Na diagonále jsou 0. Řádky i sloupce představují jednotlivé vrcholy. Pokud mezi danými uzly vede přímá cesta tak je zapsáno číslo, které odpovídá ohodnocení hrany mezi nimi. Pokud mezi uzly nevede přímá cesta je zapsána jako nekonečno (∞). Pro neorientované grafy je tato matice symetrická podle diagonály stejně jako v případě matice sousednosti. [12]

Pro lepší představu uvedu příklad na následujících obrázcích ohodnocených orientovaných a neorientovaných grafů.



Obrázek 8 - Ohodnocený neorientovaný graf [zdroj: autor]



Obrázek 9 - Ohodnocený orientovaný graf [zdroj: autor]

Pro lepší představu matice výše uvedeného neorientovaného grafu (Obrázek 8):

$$\begin{pmatrix} & 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 1 & \infty \\ 1 & 3 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 8 \\ 3 & \infty & 2 & 8 & 0 \end{pmatrix}$$

A pro porovnání i matice sousednosti výše uvedeného orientovaného grafu (Obrázek 9):

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

2.4.4 Spojová reprezentace

Spojová reprezentace grafu je pravděpodobně nejpoužívanější formou reprezentace grafu. Spojová reprezentace obsahuje tolik řádků, kolik má graf vrcholů. Na každý řádek za zapíše číslo (nebo jiný unikátní ukazatel) vrcholu, vybraný znak, jenž bude sloužit jako oddělení a na konec se vyjmenují jednotlivé vrcholy (číslem nebo jiným identifikátorem) do kterých vede cesta z daného vrcholu. Alternativně se může ještě u každého z vrcholu použít ohodnocení hrany. Jeden řádek záznamu by tedy mohl poté vypadat nějak takto: [7]

```
1-2/2,3/1
```

Pro lepší představu uvádím celý záznam spojové reprezentace výše uvedeného neorientovaného grafu (Obrázek 8):

```
0-1/3,2/1
1-0/3,2/1,3/2
2-0/1,1/1,3/8
3-1/2,2/8
```

A pro porovnání i celý záznam spojové reprezentace výše uvedeného orientovaného grafu (Obrázek 9Obrázek 9):

```
0-1/5,2/2
1-3/1
2-1/3,3/4
3-
```

2.4.5 Další reprezentace

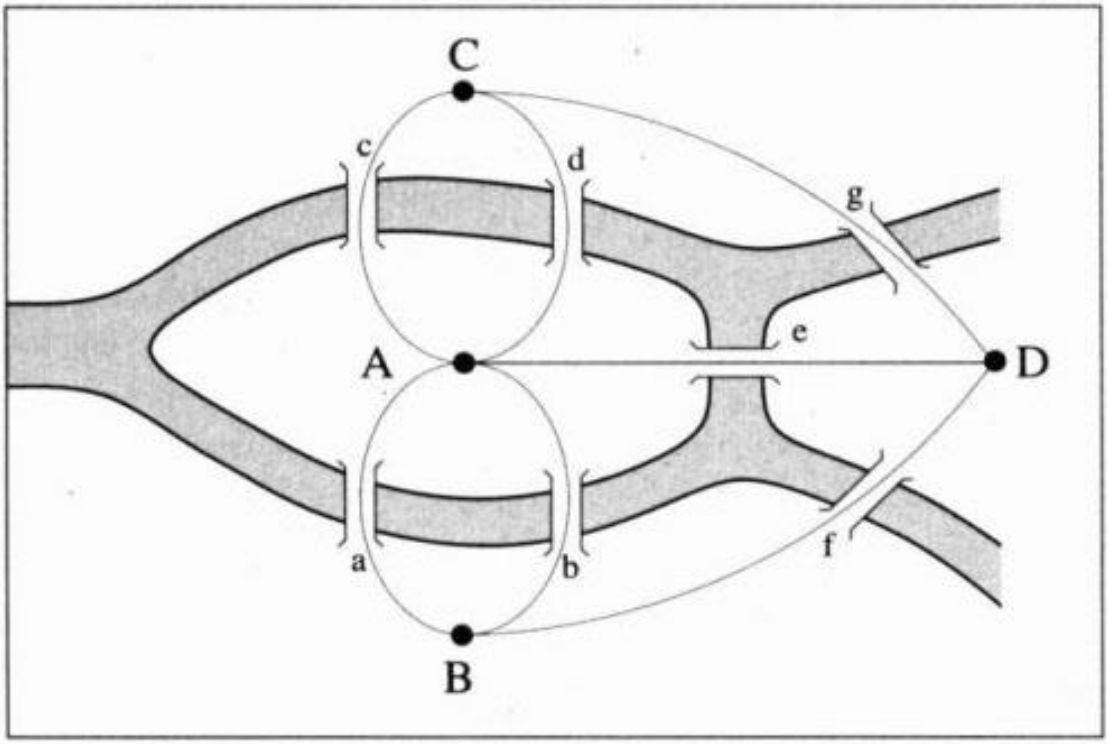
V softwaru není obvykle graf příliš reprezentován některou z těchto metod. Ale používá se pro ukládání dat. Pro samotnou reprezentaci se používají vybrané datové struktury, které do jisté míry reflektují grafickou podobu grafu.

Nejčastěji se tedy používá datová struktura hrana a vrchol (uzel) které jsou poté obaleny datovou strukturou graf. Jednotlivé vztahy mezi nimi poté již záleží na návrhu aplikace, druhu programovacího jazyka. Implementace grafu se může výrazně lišit v Javě a v C++.

3 HISTORIE TEORIE GRAFŮ

3.1 18. století – Sedm mostů města Královce (Königsberg)

Tato úloha je jednou z prvních úloh teorie grafů. V této úloze se řešilo, zdali je možné projít každým mostem ve městě Královec (Königsberg) právě jednou a zároveň skočit tam kde jsme začali. Tato úloha byla vyřešena až v roce 1763 Leonhardem Eulerem, jenž matematicky dokázal, že to není možné. [4]



Obrázek 10 - Sedm mostů města Královce [13]

Hlavní myšlenkou při tomto důkazu je že vrchol, z něhož vychází lichý počet hran může být buď vrcholem počátečním nebo vrcholem koncovým. A jak je možné vidět na obrázku, v případě sedmi mostů města Královce vychází lichý počet hran z každého vrcholu. [14]

Tento graf tedy neobsahuje eulerův tah, přesněji neobsahuje eulerův okruh.

3.2 19. století – Fyzika a chemie

Na teorii grafů navázal v roce 1847 Gustav Kirchhoff, který se zabýval výpočtem proudů v elektrických sítích pomocí *počtu koster grafu*. A v roce 1857 vymyslel sir William Hamilton hru jejímž úkolem bylo pospojovat všechny vrcholy pravidelného dvanáctistěnu tak, aby byl každý vrchol použit právě jednou. Podle této hry vznikl pojem *hamiltonovská kružnice*. Ovšem nejznámější úlohou z toho století je problém *čtyř barev*. Hlavní otázkou této

úlohy bylo, zdali je možné pomocí 4 barev obarvit jakoukoliv mapu tak, aby dvě sousední země nebyly obarveny stejnou barvou. [15]

4 OPERACE NAD GRAFY

Nad grafy v různých podobách lze provádět různé operace. Mezi nejčastější pravděpodobně patří hledání nejkratší cesty, ale můžeme také hledat cestu nejdelší, popřípadě v případě stavebního inženýrství nebo obecně tam kde je potřeba organizovat práci v čase se využívá hledání kritické cesty. Popřípadě hledání minimální kostry, čehož se využívá třeba v sítích pro nalezení nejmenšího počtu skoků do cílového uzlu nebo v případě inženýrských sítí (elektřina, voda, plyn) pro minimalizaci nákladů na výstavbu sítí a propojení jednotlivých domů.

4.1 Grafové algoritmy

Algoritmy přinášejí pravidla nebo také recept pro práci s grafy, na vstupu dostáváme graf v nějaké podobě a na výstupu dostáváme řešení. Řešením může být sled uzlů nebo délka cesty. Popřípadě matice řešení nebo také prázdná množina v případě nenalezení řešení. Každý algoritmus má svá pozitiva a negativa. A každý je něčím jiný. Algoritmy se dělí podle časové náročnosti, paměťové náročnosti a podle složitosti implementace, tedy podle toho, jak dlouho nám bude trvat takový algoritmus napsat. Mezi základní algoritmy patří prohledávání do šířky nebo prohledávání do hloubky. Dalšími algoritmy jsou Dijkstrův algoritmus, A-star algoritmus (A* algoritmus) a Floyd-Warshallův algoritmus.

4.1.1 Hledání nejkratší cesty v grafu

Hledání nejkratší cesty v grafu lze provádět několik způsoby, asi nejzákladnějším způsobem je přímý pohled na graf, ovšem pro tyto účely musíme mít graf nějaký způsobem zobrazený. Tento způsob tedy není ideální a nemusí vždy poskytovat správné výsledky. Z tohoto důvodů matematici vymysleli několik algoritmů, jimiž se dá dosáhnout nalezení nejkratší cesty. Nalezená cesta nemusí být vždy ta nejkratší, ale jedna z nejkratších, popřípadě v při přidání dalšího parametru můžeme hledat cestu s nejvyšší propustností nebo v případě silničních sítí nejrychlejší cestu tedy pro nás nejoptimálnější cestu.

4.1.2 Prohledávání do šířky a do hloubky

Tyto dva algoritmy jsou velice podobné. Liší se v datové struktuře, do které se ukládají jednotlivé vrcholy při průchodu grafem. Pro prohledávání do hloubky se používá zásobník. [15] Pro prohledávání do šířky se používá fronta. [16]

Kroky algoritmu prohledávání do šířky:

```
1) Všechny vrcholy označit jako nenavštívené.
2) Počáteční vrchol nastavit jako navštívený.
3) Vložit počáteční vrchol do fronty.
4) Odebrat první vrchol z fronty a uložit jej do proměnné.
5) Na konec fronty zařadit všechny vrcholy do kterých vede hrana
z uloženého vrcholu a nebyly ještě navštívené a zároveň je nastavit jako
navštívené.
6) Návrat ke korku 4 dokud není fronta prázdná.
```

Asymptotická složitost algoritmu je rovna *O(V+E)*, kde *V* představuje počet vrcholů grafu a *E* počet hran grafu. [16]

Kroky algoritmu prohledávání do hloubky:

```
1) Všechny vrcholy označit jako nenavštívené.
2) Počáteční vrchol nastavit jako navštívený.
3) Vložit počáteční uzel na zásobník.
4) Odebrat vrchol z vrcholu zásobníku a uložit jej do proměnné.
5) Na zásobník vložit všechny vrcholy do kterých vede hrana z uloženého
vrcholu a nebyly ještě navštívené a zároveň je nastavit jako navštívené.
6) Návrat ke kroku 4 dokud není zásobník prázdný.
```

Asymptotická složitost algoritmu je stejná jako u prohledávání do šířky tedy *O(V+E)*, kde *V* představuje počet vrcholů grafu a *E* počet hran grafu. [17]

4.1.3 Dijkstrův algoritmus

Dijkstrův algoritmus je nejrychlejší algoritmus sloužící k nalezení nejkratší cesty v grafu. Používá se tam, kde se v grafu nenalézají žádné záporné hrany. Algoritmus vynalezl v roce 1959 nizozemský informatik Edsger Dijkstra. Algoritmus je konečný. Vstupní graf nesmí obsahovat hrany se záporným ohodnocením. [18]

Během provádění algoritmu postupujeme od výchozího uzlu, jenž si zvolíme, až k cílovému uzlu, jenž si také zvolíme. Výstupem z tohoto algoritmu je poté seznam uzlů, jimiž cesta prochází a dílčí vzdálenosti mezi nimi. V průběhu algoritmu jsou obvykle ohodnoceny všechny vrcholy.

U Dijkstrova algoritmu lze dosáhnout asymptotické složitosti $O(E + V \log V)$, kde *V* představuje počet uzlů a *E* počet hran. [18]

Postup provádění algoritmu:

```
1) Zvolíme výchozí a cílový vrchol.
2) Výchozí vrchol ohodnotíme 0 a ostatní vrcholy ohodnotíme hodnotou
nekonečno.
3) Zároveň výchozí vrchol označíme jako aktuální a ostatní vrcholy jako
nenavštívené.
4) Vybereme nenavštívený vrchol s nejnižším ohodnocením.
5) Projdeme veškeré hrany vedoucí z daného vrcholu. Pokud je hodnota
vrcholu na konci hrany vyšší jak hodnota součtu hrany a aktuálního vrcholu,
```

```
tak hodnotu vrcholu na konci hrany změníme na hodnotu součtu hrany a
aktuálního vrcholu a zároveň nastavíme aktuální vrchol jako předchůdce uzlu
na konci hrany.
6) Nastavíme aktuální vrchol jako navštívený.
7) Pokud není koncový vrchol ohodnocený (hodnota je stále nekonečno)
pokračujeme na bod 4. Pokud je ohodnocený pokračujeme na bod 8.
8) Pokud existuje vrchol s nižším ohodnocením, jak koncového vrcholu
pokračujeme na bod 4. Pokud neexistuje vrchol s nižším ohodnocením
pokračujeme na bod 9.
9) Výslednou cestu zjistím tak, že budeme do seznamu vrcholů postupně
přidávat předchůdce jednotlivých vrcholů. Začneme koncovým vrcholem a
postupujeme tak dlouho dokud nedojdeme k začátečnímu.
```

Tento postup je rozšířením algoritmu procházení do šířky. Kdy se místo běžné fronty využívá prioritní fronty.

4.1.4 A-star algoritmus

Je algoritmus sloužící k nalezení nejoptimálnější cesty. Touto cestou může být i cesta nejkratší. Algoritmus poprvé popsali v roce 1968 P. Hart, N. Nilsson a B. Raphael. [19]

Vstupem algoritmu je ohodnocený graf, vstupní vrchol a konečný vrchol a výstupem je nejoptimálnější cesta ze vstupního vrcholu do konečného vrcholu nebo v případě, že cesta neexistuje tak prázdná množina, popřípadě zpráva o nenalezení cesty. [19]

Algoritmus je rozšířením prohledávání do šířky. Ovšem místo obyčejné fronty se používá fronta prioritní. Jednotlivé položky ve frontě jsou poté seřazeny podle funkce *f*. Tato funkce je součtem heuristické funkce (h), délky hrany a hodnotou předešlého uzlu (g). [19]

Zápis funkce na řazení v prioritní frontě vypadá následovně:

$$f(x) = h(x) + g(x)$$

Kde *f(x)* je předpokládaná hodnota cesty, *h(x)* je heuristická funkce pro konečný vrchol a *g(x)* je samotná délka cesty.

Heuristická funkce musí být větší jak nula. Heuristická funkce může představovat třeba hustotu dopravy na dané silnici nebo maximální rychlost, kterou je možné po silnici jet. Heuristické funkci a hodnotě hrany poté lze přiřadit rozdílné hodnoty důležitosti. Poté by se dala funkce upravit různými způsoby. Jedním ze způsobu je násobení obou hodnot čísly jejichž součet je 1. Vzorec by pak by pak vypadal takto:

$$f(x) = a * h(x) + b * g(x)$$

Kde *f(x)* je předpokládaná hodnota cesty, *h(x)* je heuristická funkce pro konečný uzel, *g(x)* je samotná délka cesty, *a* spolu s *b* jsou koeficienty jejichž součet musí být 1.

Časová složitost algoritmu je silně závislá na použité heuristické funkci. Obecně však jeho asymptotická složitost nebude horší jak prohledávání do šířky což je *O(V + H)*, kde *V* je počet uzlů a *H* počet hran. [19]

Postup provádění algoritmu:

```
1) Zvolíme výchozí a cílový vrchol.
2) Výchozí vrchol ohodnotíme 0 a ostatní vrchol ohodnotíme hodnotou
nekonečno.
3) Zároveň výchozí vrchol označíme jako aktuální a ostatní vrcholy jako
nenavštívené.
3a) Pokud ještě graf není heuristicky ohodnocen tak jej heuristicky
ohodnotíme podle námi zvolené funkce.
4) Vybereme nenavštívený vrchol s nejnižší funkcí f.
5) Projdeme veškeré hrany vedoucí z daného vrcholu. Pokud je hodnota funkce
f na konci hrany vyšší jak vypočtená hodnota funkce f, tak hodnotu funkce f
vrcholu na konci hrany změníme na vypočtenou hodnotu funkce f a zároveň
nastavíme aktuální vrchol jako předchůdce uzlu na konci hrany.
6) Nastavíme aktuální vrchol jako navštívený.
7) Pokud není koncový vrchol ohodnocený (hodnota funkce f je stále
nekonečno) pokračujeme na bod 4. Pokud je ohodnocený pokračujeme na bod 8.
8) Pokud existuje vrchol s nižší funkcí f, jak koncového vrcholu
pokračujeme na bod 4. Pokud neexistuje vrchol s nižším ohodnocením
pokračujeme na bod 9.
9) Výslednou cestu zjistím tak, že budeme do seznamu vrcholů postupně
přidávat předchůdce jednotlivých vrcholů. Začneme koncovým vrcholem a
postupujeme tak dlouho dokud nedojdeme k začátečnímu.
```

4.1.5 Floyd-Warshallův algoritmus

Floyd-Warshallův algoritmus slouží k nalezení všech nejkratších cest v grafu mezi všemi dvojicemi vrcholů. Algoritmus nelze použít, pokud graf obsahuje cykly záporné délky. [12]

Algoritmus se hodí zvláště tam, kde je potřeba zjištění nejkratší vzdálenosti mezi mnoha dvojicemi vrcholů. Je rychlejší použít Floyd-Warshallův algoritmus nežli n-krát použít Dijkstrův algoritmus. [20]

Algoritmus nezávisle na sobě objevili v roce 1962 Robert Floyd a Stephen Warshall [21]

Vstupem algoritmu je čtvercová matice délek. Výstupem je čtvercová matice předchůdců.

Složitost algoritmu je $O(|V|^3)$, kde V je počet vrcholů grafu. Složitost algoritmu je takto vysoká, jelikož řeší nejkratší cestu mezi všemi dvojicemi vrcholů v grafu. [20]

Postup provádění algoritmu:

```
1) Sestavení matice přímých vzdáleností
2) Pomocí třech vnořených cyklů hledáme možnost zkrácení cesty. V každé
iteraci počítáme nejkratší vzdálenosti v příslušeném podgrafu.
3) Pokud je počet iterací hlavního cyklu menší, jak počet vrcholů grafu
vracíme se na krok 2. Pokud ne je algoritmus ukončen.
```

Pro lepší představu pseudokód algoritmu:

```
maticeVzdalenosti D;
maticePredchudcu P;
for (i = 0; i < pocetVrcholu; i++)
      for (j = 0; j < pocetVrcholu; j++)
            for (k = 0; k < pocetVrcholu; k++)
                  if (D[j][k] > D[j][i] + D[i][k])
                        D[j][k] = D[j][i] + D[i][k];
                        P[j][k] = P[i][k];
```

5 DATOVÉ STRUKTURY

V této kapitole popíšu datové struktury, jenž jsem použil v mé aplikaci. Popíšu datové struktury, jenž jsem vytvořil i datové struktury, které jsou již vytvořené v jazyce Java a já je používám.

Jako první popíšu používané datové struktury z jazyka Java, které používám pro ukládání dat konkrétně tedy kolekce. Jako druhé popíšu mnou vytvořené datové struktury, které používám pro ukládání dat.

5.1 List

První datovou strukturou, jenž používám je list. List je rozhraní.

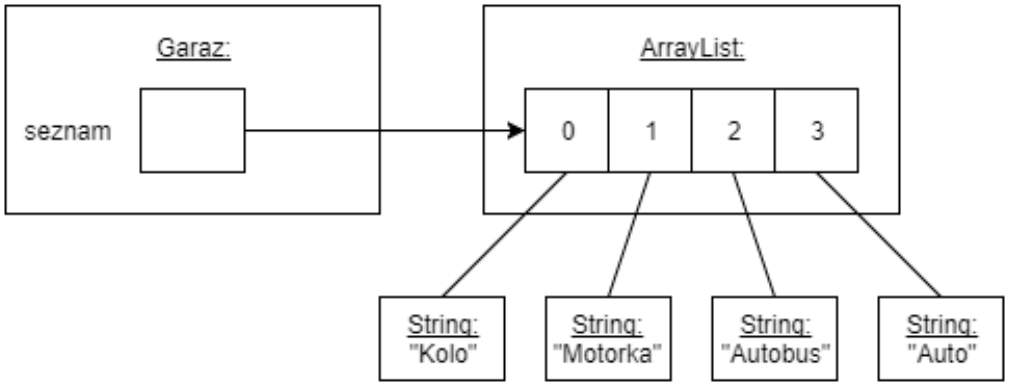
Definice 5.1 Rozhraní je předpis, jenž specifikuje veřejné metody, které musí třída splňovat. [22]

Tedy rozhraní nám pouze popíše, jaké metody musí naše třída obsahovat, ale jejich implementace je již na nás. [22]

5.1.1 ArrayList

První podoba, v které list používám je ArrayList což je pole s proměnnou velikostí. Poskytuje přístup k libovolnému prvku s asymptotickou složitostí $O(1)$. Což je hlavní důvod proč ArrayList používám. ArrayList je vhodné použít všude tam kde je potřeba rychlý přístup k jednotlivým položkám a s položkami v poli se nemanipuluje, tedy nepřesouvají se, nepřidávají se. Jak již bylo řečeno velikost pole uvnitř kolekce se dynamicky mění. Tedy, když přidáte prvek tak velikost naroste a když jej odeberete tak se velikost zmenší. [23]

Vnitřní implementace této kolekce může vypadat podobně ač o hodně složitěji jako na následujícím obrázku (Obrázek 11).

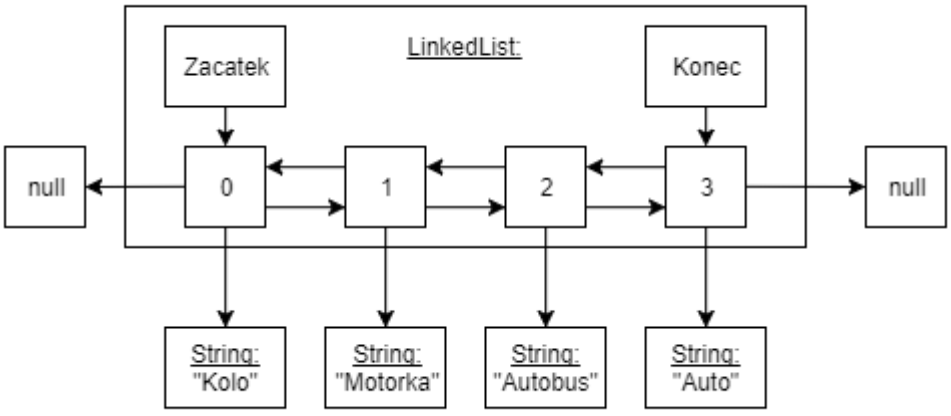


Obrázek 11 - ArrayList schéma [zdroj: autor]

5.1.2 LinkedList

Druhá podoba, v které používám list je LinkedList, což je obousměrný spojový seznam. Výhodou této kolekce je jeho rychlost v přidávání a odebírání prvků na první nebo poslední pozici. Další výhodou je jeho rychlost přístupu k prvnímu nebo poslednímu prvku kolekce která je $O(1)$. A také se více hodí při potřebě postupně iterovat skrz kolekci od začátku na konec nebo naopak. Každý prvek v kolekci si udržuje informaci o jeho pořadí, předchůdci, následníkovy a obsažená data. Kolekce samotná si poté, mimo jiné, udržuje informaci o prvním a posledním prvku kolekce [23]

Vnitřní implementace této kolekce může vypadat nějak podobně, ač složitěji jak na následujícím obrázku (Obrázek 12).



Obrázek 12 - LinkedList schéma [zdroj: autor]

5.1.3 Porovnání rychlostí operací

V textu jsem již zmínil rychlost některých operací, ale je vhodné uvést komplexní srovnání vzhledem k časové náročnosti jednotlivých operací. Porovnány jsou ArrayList a LinkedList v rychlosti přístupu k prvku, rychlosti přidávání prvku, rychlosti procházení kolekcí a rychlosti vyhledávání v kolekci. Toto porovnání je dobré znát, abychom pro potřeby daného algoritmu zvolili správný druh kolekce. Způsob uchování dat a přístupu k nim výrazným způsobem ovlivňuje rychlost (výkon) výsledného algoritmu. Při vývoji algoritmu využívající některou z těchto kolekcí je tedy dobré si uvědomit jaké operace bude náš program nejčastěji provádět nad kolekcí a podle toho zvolit vhodnou kolekci. Porovnání je zobrazeno v následující tabulce.

První sloupec v tabulce představuje operaci prováděnou s kolekcí, druhý a třetí sloupec představuje časovou náročnost v daných operací vyjádřenou pomocí asymptotické složitosti.

Tabulka 2 - Porovnání rychlosti operací nad kolekcemi ArrayList a LinkedList [24]

Operace	ArrayList	LinkedList
Přístup k náhodnému prvku	$O(1)$	$O(n)$
Přidání nebo odebrání z konce	$O(1)$	$O(1)$
Přidání nebo odebrání ze začátku	$O(n)$	$O(1)$
Jeden krok literátoru v před nebo v zad	$O(1)$	$O(1)$
Hledání nebo odebírání náhodného prvku	$O(n)$	$O(n)$

5.2 Queue

Druhou datovou strukturou, jenž používám je fronta. Tato kolekce se vyznačuje tím, že lze přistupovat k prvnímu prvku takzvané hlavě (head). Pro ukládání dat se používá strategie *LILO – last in, last out.* Neboli prvek, který je jako poslední přidán bude také poslední odebrán. [25]

5.2.1 PriorityQueue

Frontu konkrétně používám v podobě prioritní fronty. Tato fronta je seřazená a pokud se neřekne jinak tak vždy na první pozici je prvek s nejnižší hodnotou. V této frontě tedy již neplatí pravidlo LILO jelikož jednotlivé prvky se mohou takzvaně předbíhat. Typickou implementací prioritní fronty je binární halda (heap). [25]

5.3 Map

Dalším rozhraním z jazyka Java, které používám je Map. Tato kolekce obsahuje objekt svázaný s jeho klíčem. V kolekci se nemohou nacházet dva stejné klíče, ale mohou se zde nacházet dva stejné objekty ovšem s rozdílným klíčem. Mapu používám hlavně díky možnosti rychlého vyhledávání podle klíče. [25]

5.3.1 HashMap

Mapou používám v podobě HashMap tato mapa poskytuje přístup k libovolnému prvku s konstantní složitostí. Tato kolekce je nesynchronní, tedy v případě víceválcových aplikací s ní může v daném okamžiku manipulovat pouze jedno vlákno. Prvky v této kolekci nejsou seřazené v takovém pořadí, v jakém jaké byly do kolekce vkládány. Tedy iterování strukturou nám nezajistí vždy stejné pořadí jednotlivých prvků.

5.4 Vrchol

Tato datová struktura představuje vrchol v grafu. Vrchol si v sobě uchovává informaci o jeho pořadí, své hodnoty a také list hran v kterých si uchovává veškeré hrany jež vedou z nebo do vrcholu. Pro účely jednotlivých algoritmů si vrchol uchovává informaci o jeho předchůdci o tom zadli již byl ohodnocen a zpracován a také tři rozdílné hodnoty.

5.5 Hrana

Tato datová struktura představuje granu v grafu. Je spojnicí mezi dvěma vrcholy. Hrana si uchovává informaci o jejím začátku a konci, její hodnotu a pro účely algoritmů informaci o tom, zdali již byla použita.

5.6 OrientovanyGraf

Datová struktura OrientovanyGraf představuje graf, v kterém mají všechny hrany svojí orientaci, tedy každá hrana má svůj začátek a konec. Datová struktura si uchovává informaci o počtu vrcholů, jenž obsahuje a v datové struktuře Javy HashMap si uchovává veškeré obsažené vrcholy. Klíčem je v tomto případě pořadí vrcholu a hodnotou je samotný vrchol. Pro účely algoritmu si uchovává i začátek grafu.

5.7 NeorientovanyGraf

Datová struktura NeorientovanyGraf představuje graf, ve kterém není pevně rozlišen začátek a konec hran. Datová struktura si uchovává informaci o počtu vrcholů, jenž obsahuje a v datové struktuře Javy HashMap si uchovává veškeré obsažené vrcholy. Klíčem je v tomto

případě pořadí vrcholu a hodnotou je samotný vrchol. Pro účely algoritmu si uchovává i začátek grafu.

6 VÝVOJ APLIKACE

V této části popíšu jednotlivé požadavky na aplikaci, jaké jsem měl možnosti při vývoji, proč jsem si zvolil vývoj v jazyce Java a ostatní věci týkající se samotného vývoje aplikace.

6.1 Požadavky na aplikaci

Hlavním požadavkem aplikace bylo nalezení nejkratší cesty nebo cest pomocí vybraných algoritmů a informování uživatele o výsledcích těchto algoritmů. Původně byla aplikace zamýšlena jako konzolová, ale s přibývajícími funkcemi jsem rozhodl o vytvoření grafického rozhraní (dále jen GUI), které bude pro uživatele přívětivější a přehlednější.

6.2 Volba programovacího jazyka

Programovací jazyk Java jsem si zvolil z několik důvodů. Hlavním důvod je ten, že je primárně vyučován ve škole. Druhým důvodem je rozmanitost datových struktur. V době, kdy jsem začal vyvíjet aplikaci jsem jednotlivé datové struktury a kolekce jazyka Java znal mnohem lépe než datové struktury jazyka C#, který by byl také vhodným kandidátem. Jazyk Java jsem zvolil i protože, že se na rozdíl od C nebo C++ nemusím vůbec starat o správu paměti.

6.3 Volba grafického rozhraní

V jazyce Java je momentálně na výběr tvořit grafické rozhraní pomocí knihoven Swing nebo JavaFX já jsem zvolil Swing, jelikož s JavaFX jsem se v průběhu studia vůbec nesetkal, nebyla vyučována v žádném předmětu. Dalším důvodem byly mé zkušenosti s tímto rozhraním.

6.4 Volba vývojového prostředí

Pro vývoj aplikace jsem zvolil vývojové prostředí NetBeans IDE ve verzi 8.2 od firmy Oracle. Které je dostupné zdarma. Vývojové prostředí mi v mnohém usnadnilo práci a pomohlo s přehledností kódu. Mezi velké výhody vývojových prostředí patří jejich schopnost napovídání, generování konstruktorů a metod na základě poskytnutých dat. Další schopností je kontrola syntaxe, formátování textu a v neposlední řadě schopnost ladění, tedy schopnost za běhu programu sledovat a zastavovat a pouštět jeho chod spolu s viditelností dat.

6.5 Ostatní programy

Pro kreslení obrázků grafů jsem používal online aplikaci, ke které je možné přistoupit z webové adresy www.draw.io.

Pro psaní práce a vyhodnocování dat jsem používal aplikaci Microsoft Office 2016, konkrétně její části Microsoft Excel a Microsoft Word.

6.6 Načítání grafu ze souboru

Pro potřeby aplikace bylo rozhodnuto, že bude potřeba načítat data ze souboru. Tento soubor bude textový a data v něm mohou být uložena buď jako matice přímých vzdáleností nebo pomocí spojového seznamu. Soubor je textový. Přípona souboru je txt. Obsah souboru musí přesně splňovat zadaný formát. Tento formát bude spolu s příkladem popsán v následujících dvou kapitolách.

6.6.1 Matice přímých vzdáleností

Tento soubor má strukturu pro neorientované grafy:

```
4
N
0 3 1 ∞
3 0 1 2
1 1 0 8
∞ 2 8 0
```

Nebo pro orientované grafy:

```
4
O
0 5 2 ∞
∞ 0 ∞ 1
∞ 3 0 4
∞ ∞ ∞ 0
```

První řádek představuje počet vrcholů. Druhý řádek představuje druh grafu. Velké písmeno **N** značí neorientovaný graf. Velké písmeno **O** značí orientovaný graf. Na dalších řádcích je poté uvedená samotná matice přímých vzdáleností, která již byla popsána výše v kapitole 2.4.3 Reprezentace grafů.

6.6.2 Spojový seznam

Tento soubor má v případě neorientovaného grafu strukturu:

```
4
N
0-1/3,2/1
1-0/3,2/1,3/2
2-0/1,1/1,3/8
3-1/2,2/8
```

Nebo v případě orientovaného grafu:

```
4
O
0-1/5,2/2
1-3/1
2-1/3,3/4
3-
```

Kdy první řádek představuje počet vrcholů. Druhý řádek představuje druh grafu. Velké písmeno **N** značí neorientovaný graf. Velké písmeno **O** značí orientovaný graf. Na dalších řádcích jsou poté vyjmenované jednotlivé vrcholy a hrany z něj vedoucí. První číslo v řádku představuje pořadí vrcholu. Vrcholy jsou řazeny vzestupně a začínají číslem 0. Následuje pomlčka a poté čárkou oddělené informace o jednotlivých hranách. Pokud se za pomlčkou nenachází žádné další znaky ta z toho vrcholu nevede žádná cesta. První číslo značí, do jakého vrcholu vede hrana a číslo za lomítkem značí ohodnocení této hrany.

Podrobně je spojový seznam popsán v kapitole 2.4.4, kde je uveden příklad spolu s obrázkem daného grafu.

6.7 Ukládání grafu do souboru

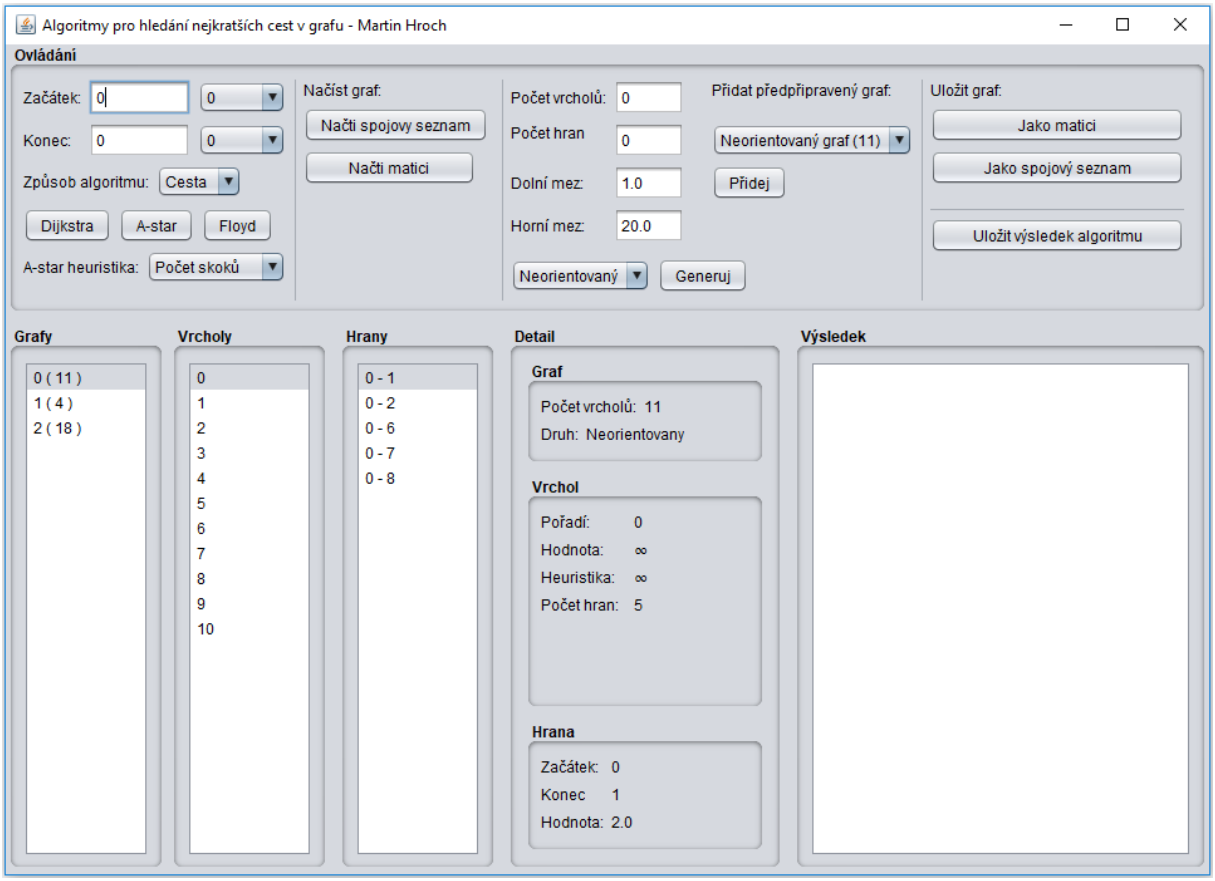
Graf je možné uložit do souboru v jednou z dvou výše zmíněných formátů. Tedy buď jako matici přímých vzdáleností nebo pomocí spojového seznamu. Ukládání respektuje výše popsaný formát dat.

7 POPIS GRAFICKÉHO UŽIVATELSKÉHO ROZHRANÍ

V této části bude popsáno, jak používat aplikaci. Popíšu jednotlivé grafické části aplikace. Grafické uživatelské rozhraní (GUI) je intuitivní na ovládání. Pro spuštění programu je nutné mít nainstalované JRE (Java Runtime Enviroment). JRE obsahuje knihovny a virtuální stroj nutný pro běh aplikací. Aplikaci není potřeba nainstalovat. Stačí spustit příslušný soubor s koncovkou .jar.

7.1 Rozložení

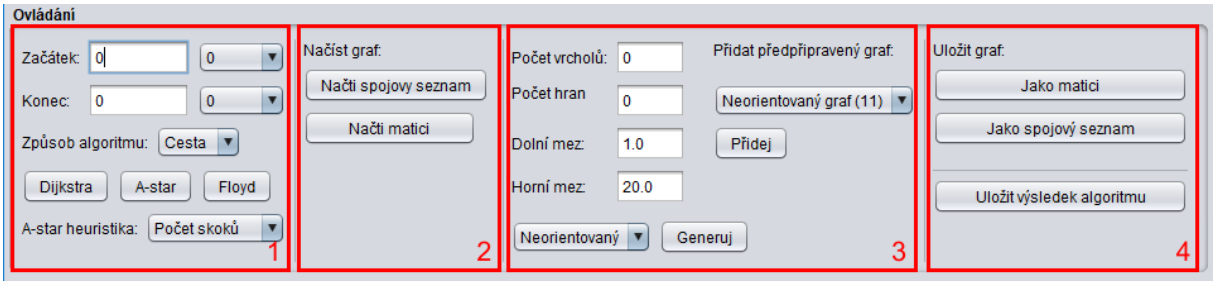
Po spuštění aplikace se nám zobrazí okno rozdělené na několik částí. První částí je Ovládání, v kterém volíme akce, jenž chceme provádět. Druhou částí je část s názvem Grafy, tato část zobrazuje všechny grafy, které jsme načetly, vygenerovali nebo již byly předpřipravené. Hned vedle této části se nachází část s názvem Vrcholy, v této části jsou zobrazeny všechny vrcholy zvoleného grafu. Následuje část s názvem Hrany, kde jsou zobrazené všechny hrany, jež vedou z daného vrcholu. Předposlední velkou částí je část Detail, kde jsou zobrazené detailní informace o grafu samotném a jeho jednotlivých částech. Poslední částí je část s názvem Výsledek, ve které se nám zobrazí výpis výsledku jednotlivých algoritmů. Jednotlivé částí a ostatní části v nich obsažené postupně popíši v následujících kapitolách. Po spuštění vypadá aplikace jako na obrázku 11.



Obrázek 13 - GUI Aplikace [zdroj: autor]

7.2 Ovládání

První částí, jenž popíšu je ovládání. Tato část je rozdělena na několik dalších částí. Jednotlivé části jsem na obrázku 12 označil čísly a budu je dále popisovat.



Část označená číslem 1 slouží k zvolení začátku a konce hledání, vybrání způsobu provedení algoritmu, zvolení druhu algoritmu a v případě A-star algoritmu zvolení druhu heuristiky. Začátek i konec je možné vybrat pomocí comboBoxu nebo zapsáním číslo do textového pole. Číslování jednotlivých vrcholů je od nuly. Způsoby provedení algoritmu jsou dva, prvním je cesta a druhým je matice. V případě vybrání cesty je algoritmus proveden od začátku hledání do konce hledání a výstupem je posloupnost vrcholů, kterými vede nejkratší cesta z počátečního vrcholu do koncového vrcholu spolu časem nutným pro výpočet. Vybrání způsobu provedení matice je výstupem algoritmu matice, kdy jsou vypočítány vzdálenosti z každého vrcholu do každého spolu s časem nutným pro výpočet. Tento výběr je aplikován pouze na Dijkstrův a A-star algoritmus. Výstupem Floydova algoritmu je vždy matice, jelikož Floydův algoritmus je přímo určený pro výpočet cest z každého vrcholu do každého. Jako poslední volíme druh algoritmu kliknutím na příslušné tlačítko. A v případě A-star algoritmu můžeme zvolit způsob výpočtu heuristiky na výběr je počet skoků a euclidean, jednotlivé rozdíly byly popsány v kapitole 4.1.4 A-star algoritmus.

Část označená číslem 2 slouží k načtení grafu z textových souborů. Textové soubory musí být v přesně zadaném formátu. Aplikace umožňuje načtení z textového souboru, v kterém jsou data zapsána jako spojový seznam nebo matice. Po kliknutí na jedno z dvou tlačítek je zobrazen dialog, v kterém vybereme příslušný soubor. Po vybrání souboru klikneme na tlačítko Open a soubor je poté zpracován. V případě, že nechceme soubor načíst klikneme na tlačítko Cancel a akce se neprovede.

Část označená číslem 3 slouží ke generování náhodných grafů nebo přidání již předdefinovaných grafů. Generátor pro svou správnou funkčnost potřebuje údaje o požadovaném počtu vrcholů a hran, o dolním a horním rozmezí ohodnocení hrany. A o druhu grafu, kterým je buď orientovaný graf nebo neorientovaný graf. Pokud chceme, aby nám generátor vygeneroval nějaký graf je nutné zadat větší počet hran, než počet vrcholů a zároveň zadat počet vrcholů větší nebo roven dvěma. Tato část umožňuje i přidání některého

z předdefinovaných grafů. Tyto grafy obvykle obsahují malé množství hran a vrcholů a jsou určené pro otestování funkčnosti programu a algoritmů.

Část označená číslem 4 je poslední částí ovládání a slouží k uložení daných grafů nebo výsledku po provedení algoritmu. Uložení vždy probíhá do textového souboru. Graf můžeme uložit v podobě spojového seznamu nebo matice, výsledek je uložen v takovém stavu, v kterém ho vidíme v části Výsledek. Po kliknutí na některý z těchto tlačítek je zobrazen dialog, v kterém jsme vyzváni k vybrání umístění a názvu souboru. Potvrzení provedeme kliknutím na tlačítko Save a naopak akci zrušíme kliknutím na tlačítko Cancel.

7.3 Grafy

Tato část obsahuje seznam, v kterém jsou zobrazené jednotlivé grafy, s kterými můžeme pracovat. Grafy jsou zobrazeny ve formátu *číslo grafu (počet vrcholů)*. Číslování začíná od 0 a je automatické. Graf můžeme vybrat kliknutím na něj. Po kliknutí jsou načteny příslušné vrcholy, které graf obsahuje a příslušné hrany, pokud prvního vrcholu nějaké vedou. A také jsou načteny příslušné detaily jednotlivých částí grafu. Po zvolení grafu je vždy vybrán první vrchol.

7.4 Vrcholy

Jak již bylo zmíněno tato část obsahuje seznam vrcholů, jenž se nachází v grafu vybraném v části Grafy. Stejně jako v části Grafy je možné zvolit vrchol kliknutím na něj. Vrcholy jsou zobrazeny a seřazeny podle pořadí v grafu. Po zvolení vrcholu jsou v části Hrany načteny hrany, jež vedou z daného vrcholu, pokud nějaké existují. A jsou aktualizovány příslušné detaily vrcholu, popřípadě hrany.

7.5 Hrany

Posledním seznamem, který tato aplikace obsahuje je seznam hran. V tomto seznamu jsou zobrazeny jednotlivé hrany, jež vedou z vrcholu vybraném v části Vrcholy. V případě, že z vybraného vrcholu žádné hrany nevedou je seznam prázdný. Formát zobrazení vrcholu je *číslo začátku vrcholu – číslo konce vrcholu*. V případě neorientovaného vrcholu nemusí být vždy prvním číslem číslo vybraného vrcholu, jelikož v neorientovaném grafu hrany nemají začátek a konec.

7.6 Detail

Detail zobrazuje informace o vybraném grafu a jeho částech. Tato část obsahuje tři dílčí části. Jednotlivé části popíšu od shora dolů.

První částí je část s názvem Graf. Obsahuje informace o celkovém počtu vrcholů v grafu a informaci o tom, zdali je graf orientovaný nebo neorientovaný.

Druhou částí je část s názvem Vrchol obsahující informace o vybraném vrcholu. Jsou zobrazeny informace o jeho pořadí, hodnotě, heuristice a o celkovém počtu hran, jenž obsahuje. V případě, že ještě nebyly příslušné hodnoty vypočteny, tedy graf nebyl ohodnocen. Je zobrazen znak nekonečna.

Poslední částí detailu je část s názvem Hrana, tato část obsahuje detailní informace o zvolené hraně. Informacemi jsou začátek a konec hrany a její ohodnocení. V případě, že není zvolena žádná hrana (z vybraného vrcholu nevedou žádné hrany) nejsou zobrazeny žádné hodnoty.

7.7 Výsledek

Poslední části grafického rozhraní je část s názvem Výsledek. Výsledek obsahuje textový výstup jednotlivých algoritmů. Kterým může být matice nebo seznam vrcholů. Výsledek je možné v části Ovládání uložit do textového souboru. Výsledek není možné označením zkopírovat.

8 VÝSLEDKY TESTOVÁNÍ

V této části textu budu prezentovat jednotlivé výsledky svého testování. Testování bylo prováděno výhradně za použití mnou vytvořeného programu a pro ukládání výsledků z jednotlivých testů jsem zvolil program Microsoft Excel, který mi umožnil data z testování příslušným způsobem zpracovat.

8.1 Z hlediska časové náročnosti

Pro účely testování jsem si pro každou kategorii vygeneroval 10 grafů pomocí generátoru v programu, ohodnocení hran v reálných číslech se pohybovalo náhodně mezi 0 a 100 s přesností na 4 desetinná místa. Poté jsem si pomocí generátoru náhodných čísel v jazyce Java nechal vygenerovat 20 různých dvojic čísel dolní hranice těchto čísel byla vždy 0 a horní hranicí byl celkový počet vrcholů. První číslo představovalo počáteční vrchol a druhé číslo konečný vrchol. Na těchto grafech jsem poté pomocí Dijkstrova, A-star a Floydova algoritmu prováděl testování. Časy získané testováním jsem poté zprůměroval a zanesl do tabulky.

Pro účely porovnání Floydova algoritmu s Dijkstrovým a A-star algoritmem jsem nechal tyto algoritmy vypočítat cesty z každého vrcholu do každého a výsledné časy poté zanesl do tabulky. Metodika testování byla v tomto případě rozdílná. Jelikož algoritmy hledají cesty

z každého vrcholu do každého je zbytečné považovat konkrétní cestu. Z tohoto důvodu jsem změnil metodiku testování. Zvýšil jsem počet vygenerovaných grafů na 20, zachoval jsem rozmezí ohodnocení hran mezi 0 a 100 s přesností na 4 desetinná místa. Každý z grafů jsem poté podrobil testování pomocí výše zmíněný algoritmů. Výsledný časy jsem poté zprůměroval a zanesl do tabulky.

Tabulka 3 - Výsledky testování algoritmů A-star a Dijkstra pro dvojici vrcholů [zdroj: autor]

Počet vrcholů / počet hran	Průměrný čas výpočtu daného algoritmu v ms	
	A-star	Dijksta
1 000/1 005	0,585	0,235
1 000/1 200	0,365	0,37
1 000/2 500	0,55	0,41
5 000/5 200	1,56	0,855
10 000/11 000	1,725	0,905
1 000 000/1 100 000	673,955	437,21

Tabulka 4 - Výsledky testování algoritmů při výpočtu všech cest v grafu [zdroj: autor]

Počet vrcholů / počet hran	Průměrný čas výpočtu daného algoritmu v ms		
	A-star	Dijksta	Floyd
400/405	21,05	14,3	115,2
500/520	28,5	20,75	176,3
800/900	99,5	74,25	785,95
1 000/1 005	97	66,7	1475,5
1 000/1 200	179,9	131,85	1608,15
1 000/2 500	281,55	208	2081,9

Z výsledných dat je patrné že nejrychlejším algoritmem je Dijkstrův algoritmus. Jako druhý je A-star algoritmus. Důvodem, proč je A-star na druhém místě je nutnost výpočtu heuristické funkce pro každý vrchol. Jako poslední je Floydův algoritmus, který má asymptotickou složitost $O(|U|^3)$ což je nejhorší složitost oproti ostatním dvou algoritmům, které v této úloze dosáhnou nejhůře kvadratické složitosti.

8.2 Z hlediska paměťové náročnosti

Využití paměti jsem sledoval pomocí nástroje Oracle Java Mission Control, v kterém jsem na grafické ose sledoval využití paměti programem. Java ukládá veškerá data na haldu (Heap) a tedy jsem sledoval graf vývoje alokované paměti na hladě mým programem.

Testoval jsem systémem, kdy jsem spustil program, nechal vygenerovat náhodný graf o 3000 vrcholech a 3200 hranách. A poté spustil nad tímto grafem jednotlivé algoritmy. Jelikož Floydův algoritmus vyhledává cesty z každého vrcholu do každého použil jsem v případě Dijkstrova a A-star algoritmu jejich verze, které také počítají cesty z každého vrcholu do každého. Testování neprobíhalo v grafickém rozhraní, které může výsledky zkreslovat. Ale probíhalo pomocí účelově vytvořené testovací třídy.

Ze získaných dat jsem poté vybral nejvyšší hodnotu, jaké v průběhu program dosáhl a zanesl jí do tabulky.

Tabulka 5 - Paměťová náročnost algoritmů

Algoritmus	Dijkstrův	A-star	Floyd
Maximální hodnota v MiB	90	138	129

8.3 Z hlediska náročnosti vývoje

Tato část se může lišit dle možností jednotlivých programovacích jazyků. Mým programovací jazykem byla Java a pracoval jsem ve vývojovém prostředí NetBeans IDE 8.2. vývojové prostředí mi v mnohém usnadnilo práci, a tak byl vývoj o trochu rychlejší. Z hlediska času stráveném při vývoji hodnotím jako nejrychlejší vývoj Floydova algoritmu poté následuje Dijkstrův a nakonec A-star. Toto porovnání jsem sestavil z hlediska nutnosti nastudování příslušných materiálů a z hlediska délky a náročnosti napsaného zdrojového kódu.

9 ZÁVĚR

Podařilo se mi splnit všechny cíle bakalářské práce. V úvodu textové části práce jsem popsal metodu porovnávání složitosti algoritmů. Poté jsem popsal jednotlivé pojmy, které se týkají grafů a jeho částí potažmo pojmy pojící se s teorií grafů. Spolu s těmito pojmy jsem popsal, jakými způsoby lze graf reprezentovat.

Následující kapitola se věnuje historii teorie grafů, a to konkrétně pro 18 a 19 století. Následně jsem v další kapitole popsal jednotlivé algoritmy, které lze na grafy aplikovat. Nebyli zmíněné všechny, ale pouze ty vybrané a nejdůležitější

V praktické části bakalářské práce jsem úspěšně implementoval zadané algoritmy tedy Dijkstrův, A-star a Floydův algoritmus. Spolu s těmito algoritmy jsem pro účely testování vytvořil jednoduchý generátor grafů. Grafy je také uložit. Pro uložení byl zvolen textový formát ve formě spojového seznamu nebo matice přímých vzdáleností.

Vybrané datové struktury, které používám a které jsem vytvořil jsem následně popsal v kapitole datové struktury. Pomocí programu jsem poté provedl testování a získané výsledky jsem zpracoval a zanesl do práce v poslední kapitole.

Do budoucna vidím možné rozšíření aplikace v podobě načítání grafů z map nebo jiných zdrojů. Další rozšíření je možné v implementaci dalších druhů výpočtu heuristické funkce u A-star algoritmu. Popřípadě implementování některého z dalších algoritmů.

10 POUŽITÁ LITERATURA

[1] NECKÁŘ, Jan. Asymptotická složitost. *ALGORITMY.NET* [online]. [cit. 2017-12-04]. Dostupné z: https://www.algoritmy.net/article/102/Asymptoticka-slozitost

[2] HORDĚJČUK, Hordějčuk. Asymptotická složitost. *Digitální Wiki - Vojtěch Hordějčuk* [online]. 2017 [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/asymptoticka-slozitost/

[3] DANNHOFEROVÁ, Jana a Tomáš FOLTÝNEK. Vysvětlení pojmu graf. Univerzitní informační systém MENDELU [online]. 2009 [cit. 2017-12-04]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/popupokno.pl?pojem=2748

[4] HORDĚJČUK, Vojtěch. Teorie grafů. Digitální Wiki - Vojtěch Hordějčuk [online]. 2017 [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/graf/

[5] VÍTEČKOVÁ, Miluše, Petr PŘIDAL a Tomáš KOUDELA. Teorie grafů. Fakulta strojní - VŠB-TU Ostrava: sylaby a elektronické učebnice [online]. 2006 [cit. 2017-12-04]. Dostupné z: http://books.fs.vsb.cz/SystAnal/texty/21.htm

[6] DANNHOFEROVÁ, Jana a Tomáš FOLTÝNEK. Základní pojmy teorie grafů. Univerzitní informační systém MENDELU [online]. 2009 [cit. 2017-12-04]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=9295

[7] NECKÁŘ, Jan. Graf. ALGORITMY.NET [online]. [cit. 2017-12-04]. Dostupné z: https://www.algoritmy.net/article/1369/Graf

[8] Pseudograf. Leporelo.info [online]. [cit. 2017-12-04]. Dostupné z: https://leporelo.info/pseudograf

[9] JIROVSKÝ, Lukáš. Podgraf. Teorie grafů [online]. [2010] [cit. 2017-12-04]. Dostupné z: http://teorie-grafu.cz/zakladni-pojmy/podgraf.php

[10] DANNHOFEROVÁ, Jana a Tomáš FOLTÝNEK. Vysvětlení pojmu orientovaný graf. Univerzitní informační systém MENDELU [online]. 2009 [cit. 2017-12-04]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/popupokno.pl?pojem=2749

[11] JIROVSKÝ, Lukáš. Stromy. Teorie grafů [online]. [cit. 2017-12-04]. Dostupné z: http://teorie-grafu.cz/zakladni-pojmy/stromy.php

[12] HORDĚJČUK, Vojtěch. Algoritmus Floyd-Warshall. Digitální Wiki - Vojtěch Hordějčuk [online]. 2017 [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/algoritmus-floyd-warshall/

[13] Sedm mostů města Královce. In: Neviditelný čert: Nepravidelný obšťastník [online]. 2012 [cit. 2017-12-04]. Dostupné z: http://www.neviditelnycert.cz/data/files/img/Seven_Bridges_of_Konigsberg.jpg

[14] LUCIFER. V pavučině sítí - v říši náhody I. Neviditelný čert: Nepravidelný obšťastník [online]. 2012 [cit. 2017-12-04]. Dostupné z: http://www.neviditelnycert.cz/blog/popularne-naucny-koutek/1075-v-pavucine-siti-v-risi-nahody-i.html

[15] JIROVSKÝ, Lukáš. Historie teorie grafů. Teorie grafů [online]. [2010] [cit. 2017-12-04]. Dostupné z: http://teorie-grafu.cz/uvod/historie.php

[16] HORDĚJČUK, Vojtěch. Prohledávání do šířky (Breadth-first Search). Digitální Wiki - Vojtěch Hordějčuk [online]. 2017 [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/algoritmus-bfs/

[17] HORDĚJČUK, Vojtěch. Prohledávání do hloubky (Depth-first Search). Digitální Wiki - Vojtěch Hordějčuk [online]. 2017 [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/algoritmus-dfs/

[18] HORDĚJČUK, Vojtěch. Algoritmus Dijkstra - Vojtěch Hordějčuk. Digitální Wiki - Vojtěch Hordějčuk [online]. 2017 [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/algoritmus-dijkstra/

[19] HORDĚJČUK, Vojtěch. Algoritmus A-Star. Digitální Wiki - Vojtěch Hordějčuk [online]. [cit. 2017-12-04]. Dostupné z: http://voho.eu/wiki/algoritmus-a-star/

[20] ČERNÝ, Jakub. Floyd-Warshallův algoritmus. Algoritmy [online]. [cit. 2017-07-05]. Dostupné z: http://algoritmy.eu/zga/nejkratsi-cesta/floyd-warshalluv-algoritmus/

[21] Robert Floyd. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-04]. Dostupné z: https://cs.wikipedia.org/wiki/Robert_Floyd

[22] NECKÁŘ, Jan. Abstraktní třídy a rozhraní. ALGORITMY.NET [online]. [cit. 2017-12-04]. Dostupné z: https://www.algoritmy.net/article/23831/Rozhrani-13

[23] JELÍNEK, Lukáš. Java (11) - Kontejnery II. Linux Software [online]. 2005 [cit. 2017-12-04]. ISSN 1801-3805. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=750

[24] Java Collections Overview. Wikiversity [online]. Naposledy upraveno 6 March 2017 [cit. 2017-12-04]. Dostupné z: https://en.wikiversity.org/wiki/Java_Collections_Overview

[25] NECKÁŘ, Jan. Kolekce. ALGORITMY.NET [online]. [cit. 2017-12-04]. Dostupné z: https://www.algoritmy.net/article/34009/Kolekce-21