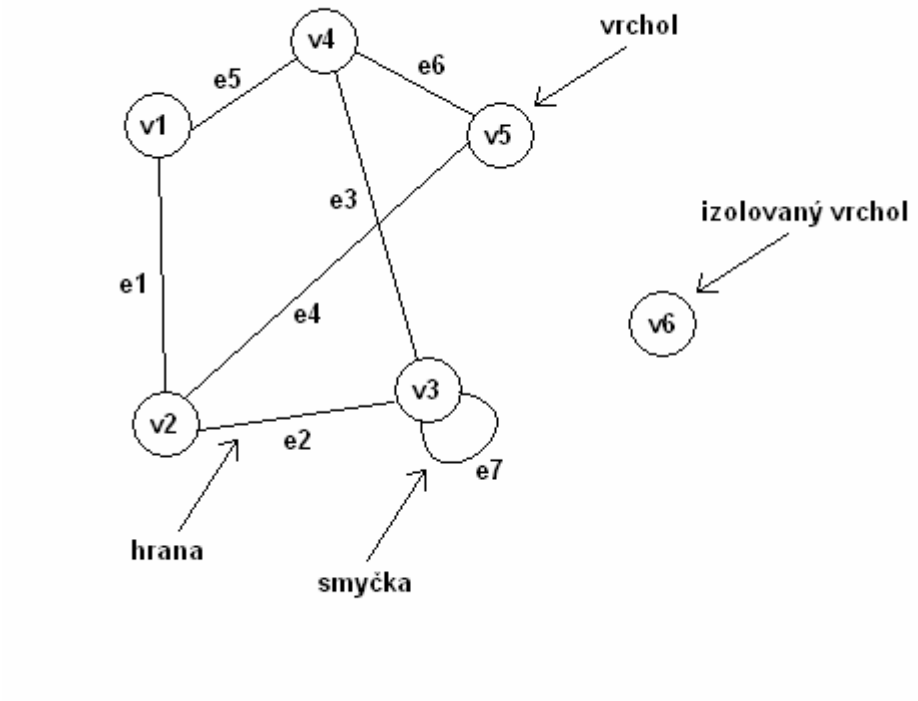
1. Úvod

V praktickém životě se velmi často setkáváme s problémem, jak nalézt nejkratší cestu. Nejkratší cestou nemusí být nutně jen nejmenší kilometrová vzdálenost mezi místem A a místem B, ale může to být například i posloupnost úkolů zvolená tak, aby zabrala co nejméně času nebo posloupnost dopravních spojů zvolená tak, aby nás tyto dopravily na dané místo v nejkratším možném čase. Časté hledání nejkratší cesty v aplikacích z nejrůznějších oborů bylo důvodem, proč se touto problematikou zabývám i ve své bakalářské práci. Práce podává stručný úvod do teorie grafů, definuje pojmy, které jsou důležité pro hledání nejkratších cest a je zde uveden přehled nejznámějších metod (algoritmů) pro hledání nejkratších cest nad různými typy grafů. Praktickou část práce tvoří program, který pomocí základních programovacích technik implementuje Dijkstrův algoritmus na zjednodušenou síť linek MHD v Plzni. Výstupem programu je taková posloupnost linek MHD, po které se v nejkratším čase dá dostat na zastávku u areálu ZČU na Borech.

2. Teorie grafů

Grafem se obecně nazve uspořádaná dvojice *(V, H)*, kde *V* je neprázdná množina prvků zvaných *vrcholy* nebo též *uzly* a *H* je množina dvojic prvků z *V*. Prvky množiny *H* se nazývají *hrany* a říká se, že dva prvky (vrcholy) $x, y \in V$ spolu *sousedí*, pokud existuje hrana $e=\{x,y\}$ taková, že $e \in H$. Vrcholy grafu se znázorňují body a hrany jako spojnice příslušných vrcholů. Jestliže hrana spojuje vrchol se sebou samým, nazývá se taková hrana *smyčkou*. Pokud z nějakého vrcholu ani do něj nevede žádná hrana, nazývá se takový vrchol *izolovaným vrcholem*. O různých hranách, které spojují tutéž dvojici vrcholů, se říká, že jsou *rovnoběžné* nebo také *násobné*. Grafy se mohou dále dělit podle nejrůznějších kritérií, z nichž pro účel této práce jsou nejdůležitějšími *orientace*, *ohodnocení* a *souvislost*.

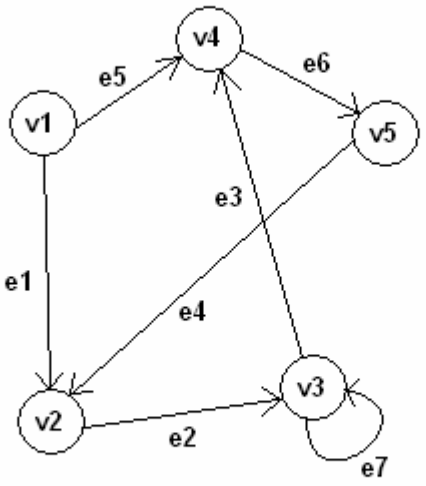


Obrázek 2.1 : Graf

2.1 Pojmy nad orientovaným grafem

Orientovaný graf je trojice $\vec{G}=(V, H, \varepsilon)$ tvořená neprázdnou konečnou množinou *V*, jejíž prvky se nazývají taktéž *vrcholy*, konečnou množinou *H*, jejíž prvky

se nazývají *orientovanými hranami*, a zobrazením $\varepsilon : H \rightarrow V^2$, které se nazývá *vztahem incidence*. Toto zobrazení přiřazuje každé hraně $e \in H$ uspořádanou dvojici vrcholů *(x, y)*. Vrchol *x* se nazývá *počátečním vrcholem hrany e* a značí se $P_v(e)$. Druhý vrchol *y* se nazývá *koncovým vrcholem hrany e* a značí se $K_v(e)$. O hraně se říká, že *vede* z vrcholu *x* do vrcholu *y* a také, že *spojuje* vrcholy *x* a *y*. O vrcholech se pak říká, že *incidují* s hranou *e* a naopak, že hrana *e inciduje* s vrcholy *x* a *y*. Vrcholy se mohou souhrnně nazvat *krajními vrcholy hrany e*. Jestliže je počáteční a koncový vrchol hrany shodný, pak se hrana nazývá *orientovanou smyčkou*.



Obrázek 2.2 : Orientovaný graf

Orientovaným sledem v grafu $\vec{G}$ se nazve posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, \ldots e_K, v_K$, taková, že pro každou hranu e_i této posloupnosti platí $P_v(e_i)=v_{i-1}$ *a* $K_v(e_i)=v_i$. U orientovaného sledu je nutné, aby všechny hrany byly orientovány vpřed ve směru sledu. Vrcholy v_0 a v_K se nazývají *počátečním* a *koncovým vrcholem* sledu. Říká se, že sled vede z vrcholu v_0 do vrcholu v_K nebo, že je sled *spojuje*. Příkladem orientovaného sledu může být posloupnost $v_1, e_1, v_2, e_2, v_3, e_3, v_4$ na obrázku 2.2.

2.2 Pojmy nad neorientovaným grafem

V případech, kdy není třeba rozlišovat mezi počátečním a koncovým vrcholem hrany se zavádí pojem neorientovaného grafu. *Neorientovaný graf* je trojice $G=(V, H, \varepsilon)$ tvořená konečnou neprázdnou množinou V, jejíž prvky se nazývají *vrcholy*, konečnou množinou H, jejíž prvky se nazývají *neorientované hrany*, a zobrazením ε, tzv. *vztahem incidence*, které každé hraně $e \in H$ přiřadí jedno nebo dvou prvkovou množinu vrcholů, tzv. *krajních vrcholů* hrany *e*. Opět se říká, že vrcholy jsou s hranou *e incidentní* (nebo s ní *incidují*) a že hrana *e spojuje* tyto vrcholy. Je-li hrana *e* incidentní pouze s jedním vrcholem, tzn. přiřadí-li zobrazení ε hraně *e* jednoprvkovou množinu, nazývá se hrana *e neorientovanou smyčkou*. Posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, \ldots e_K, v_K$, se nazývá *neorientovaným* sledem, pokud každá hrana e_i spojuje vrcholy v_{i-1} a v_i. Vrcholy v_0 a v_K se nazývají *koncovými vrcholy* sledu. Pojem neorientovaný sled se může zavést i v grafu orientovaném, smysl tohoto je ale čistě teoretický. Každý orientovaný sled je i sledem neorientovaným, protože splňuje podmínky jeho definice.

Triviální sled je sled obsahující pouze jediný vrchol a žádnou hranu, může být proto považován za orientovaný i neorientovaný. Orientovaný nebo neorientovaný sled, v němž se žádna hrana neopakuje, se nazývá *tah* (orientovaný nebo neorientovaný). Dále, pokud se ve sledu neopakuje žádný z vrcholů, nazývá se sled *cestou* (orientovanou nebo neorientovanou). V cestě se neopakují vrcholy, to rovněž znamená, že se nemohou opakovat ani hrany. Každá cesta je tedy zároveň i tahem a sledem. Každý tah je taktéž sledem, ale už nemusí být cestou.

Orientovaný nebo neorientovaný sled mající alespoň jednu hranu a jehož počáteční a koncový vrchol je totožný, se nazývá *uzavřený sled*. Podobně se mluví o *uzavřeném tahu* a „*uzavřené cestě*“ jako speciálním uzavřeném tahu, kde navzdory definici cesty platí $v_0 = v_K$.

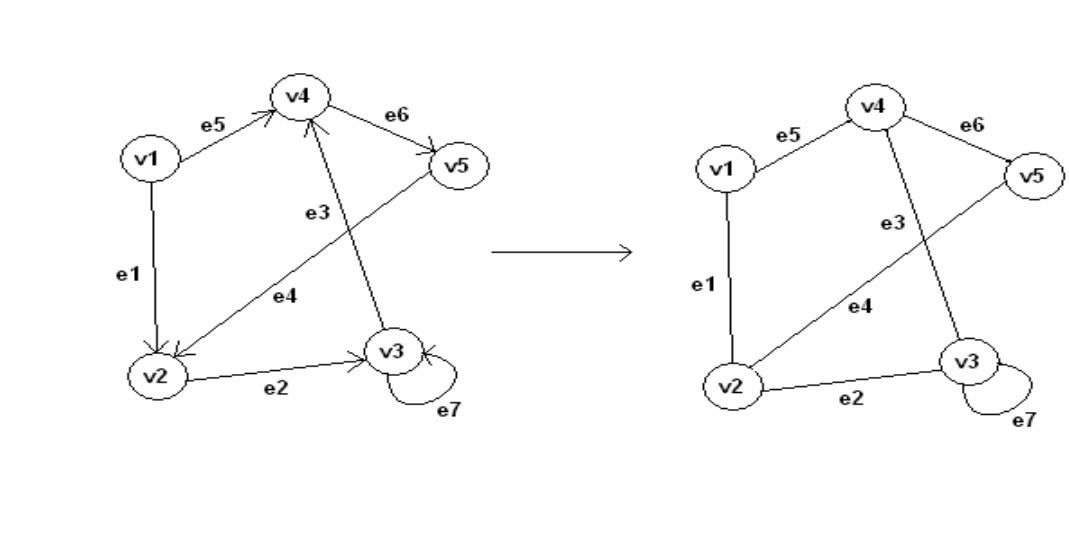
Pro „*uzavřené cesty*“ se používá speciální názvosloví. *Cyklus* je orientovaná „uzavřená cesta“ ($v_2, e_2, v_3, e_3, v_4, e_6, v_5, e_4, v_2$ na obrázku 2.2) a *kružnice* je neorientovaná „uzavřená cesta“ ($v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_5, v_1$ na obrázku 2.1). Speciálním případem kružnice je tzv. Hamiltonovská kružnice. *Hamiltonovská kružnice* v grafu je kružnice procházející všemi vrcholy grafu.

2.3 Pojem ohodnocení

V mnoha situacích, kdy se pomocí grafu modeluje situace z reálného prostředí, je pouhá existence hrany mezi dvěma vrcholy nedostačující informací. Pro lepší vyjádření reality se k hraně přidává její ohodnocení. *Ohodnocení* je zobrazení, které každé hraně přiřazuje hodnotu, obvykle číselnou. Tato hodnota je vyjádřením míry náročnosti přesunu z vrcholu *x* do vrcholu *y* nebo naopak. Ohodnocení může například reprezentovat vzdálenost (délkovou nebo časovou), propustnost potrubí, pravděpodobnost událostí apod.. Je možné, aby v jednom grafu existovalo více ohodnocení podle různých kritérií.

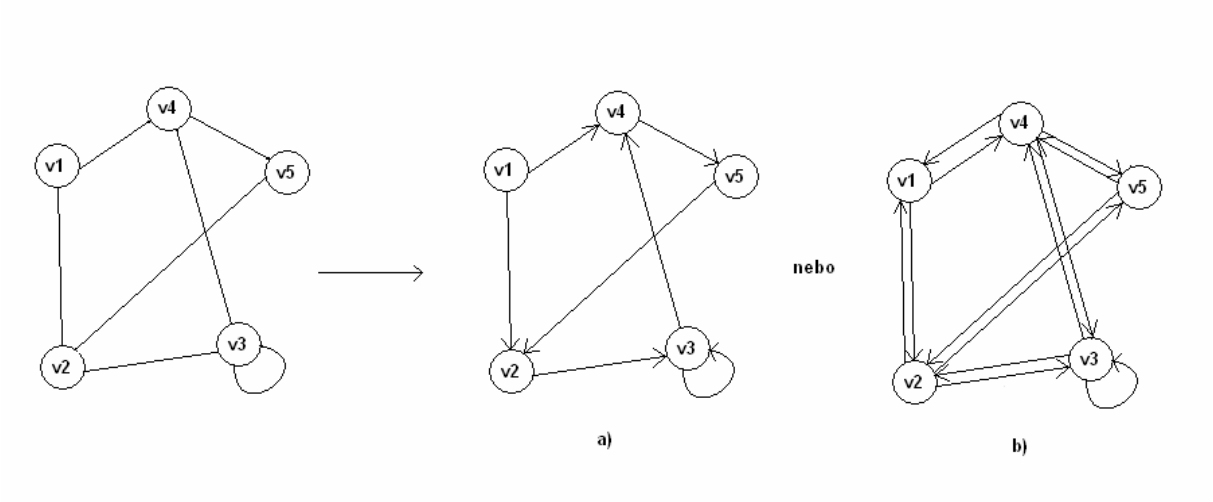
2.4 Pojem souvislosti

Pojem souvislosti grafu musí být rozlišován pro případy, když je graf orientovaný a když orientovaný není. Neorientovaný graf je *souvislý*, pokud pro každé jeho dva vrcholy *x* a *y* existuje alespoň jedna cesta z *x* do *y*. Graf na obrázku 2.1 je tedy nesouvislý. Pro orientovaný graf se zavádí souvislost dvojí a to slabá souvislost a silná souvislost. Říká se, že orientovaný *graf je silně souvislý*, pokud pro každé jeho dva vrcholy *x* a *y* existuje orientovaná cesta z *x* do *y*. Graf z obrázku 2.2 není silně souvislý, protože neexistuje žádná cesta do vrcholu v_1. Orientovaný graf je *slabě souvislý*, když je souvislá jeho symetrizace. *Symetrizací grafu* $\vec{G}$ *(V, H)* se přitom míní graf *G'(V, H')*, u kterého byla „vypuštěna" informace o orientaci hran.



Obrázek 2.3 : Symetrizace grafu

Opakem symetrizace je orientace grafu. *Orientace grafu* se může získat dvěma způsoby. Buď se v neorientovanému grafu *G*, třeba i náhodně, zvolí orientace hran, čímž se získá orientace grafu *G* nebo se každá neorientovaná hrana kromě smyček nahradí dvěma opačně orientovanými hranami a každá neorientovaná smyčka se nahradí smyčkou orientovanou. Tím se získá symetrická orientace grafu *G*.



Obrázek 2.4 : a) orientace grafu, b) symetrická orientace grafu

2.5 Důležité množiny hran a vrcholů

V grafech existují nejrůznější množiny, kombinace a posloupnosti hran a vrcholů, které je nutné definovat, aby mohly být použity při složitějších operacích nad grafem.

Nechť $\vec{G}$ *(V, H, ε)* je orientovaný graf a *x* jeho libovolný vrchol. Symbolem $V^+_{\vec{G}}(x)$ se značí *množina následníků* vrcholu *x*, tj. množina vrcholů, do nichž vede hrana z vrcholu *x*. Obdobně $V^-_{\vec{G}}(x)$ značí *množinu předchůdců* vrcholu *x*, tj. množinu vrcholů, *z* nichž vede hrana do vrcholu *x*. Množina $V_{\vec{G}}(x) = V^+_{\vec{G}}(x) \cup V^-_{\vec{G}}(x)$ je množina všech sousedů vrcholu *x*, tj. množina vrcholů spojených hranou s vrcholem *x*. Symbol $H^+_{\vec{G}}(x)$ značí *výstupní okolí* vrcholu *x*, tj. množinu hran s počátečním vrcholem *x*. Symbol $H^-_{\vec{G}}(x)$ naopak značí *vstupní okolí* vrcholu *x*, tj. množinu hran s koncovým vrcholem *x*. Množina $H_{\vec{G}}(x) = H^+_{\vec{G}}(x) \cup H^-_{\vec{G}}(x)$ je obecně *okolí vrcholu x*, tj. množina všech hran incidujících s vrcholem *x*. Výrazem $m_{\vec{G}}(x, y)$ se označuje *násobnost hrany* (*x*, *y)*, tedy

počet hran, které vedou z x do y. Počet hran vycházejících z vrcholu x se označuje termínem *výstupní stupeň* vrcholu x a značí se $d^+_{\vec{G}}(x)$. Podobně počet hran vcházejících do vrcholu x se nazývá *vstupním stupněm* vrcholu x a značí $d^-_{\vec{G}}(x)$. *Stupněm vrcholu* x v grafu $\vec{G}$ se nazývá počet hran incidujících s vrcholem x a značí se symbolem $d_{\vec{G}}(x)$, přičemž $d_{\vec{G}}(x) = d^+_{\vec{G}}(x) + d^-_{\vec{G}}(x)$. Případné smyčky incidentní s vrcholem x se počítají jako dvě hrany.

Pojem a označení množiny sousedů, okolí vrcholu, násobnosti hrany a stupně vrcholu lze použít i pro graf neorientovaný, protože jak je patrné, na orientaci hrany v těchto případech nezáleží. Pojmy vstupní a výstupní stupeň stejně jako vstupní a výstupní okolí vrcholu ztrácejí v případě neorientovaného grafu smysl.

2.6 Pojem dostupnost

Dalším důležitým pojmem je dostupnost. Říká se, že vrchol y je orientovaně nebo neorientovaně dostupný z vrcholu x, pokud existuje orientovaný nebo neorientovaný sled vedoucí z vrcholu x do vrcholu y.

2.7 Nejkratší cesta

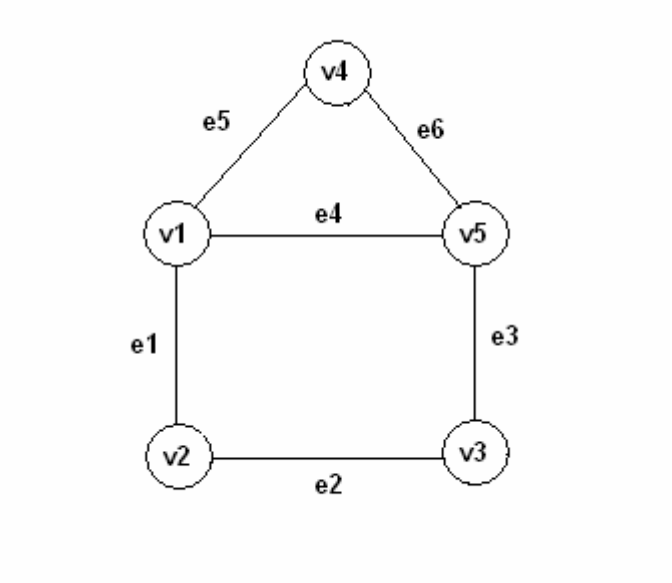
V ohodnocených grafech má dobrý smysl hovořit o součtu ohodnocení hran sledu. Pokud ohodnocení vyjadřuje například náklady nutné na překonání hrany, má součet ohodnocení velký praktický význam. Každá hrana se započítává tolikrát, kolikrát je ve sledu procházena. Hledáním nejkratší cesty (sledu) se rozumí hledání cesty (sledu) s minimálním součtem ohodnocení všech hran této cesty (sledu). Hledání nejkratších, popřípadě nejdelších cest v grafu je jednou ze základních úloh teorie grafů.

3. Způsoby zadávání grafů

Pro zadání a reprezentaci grafu existuje celá řada způsobů. Vzájemně se liší jak vhodností použití pro určitý typ grafu nebo úlohy, tak stupněm univerzálnosti.

3.1 Obrázek

Pokud není graf příliš velký, je nejjednodušším způsobem reprezentace prostý *obrázek*, hlavně v případě, nemá-li graf mnoho hran. S jejich přibývajícím počtem se obrázek stává velmi nepřehledným. Další nevýhodou je zřejmá nevhodnost obrázku pro zadání grafu do počítače.



Obrázek 3.1 : Reprezentace grafu obrázkem

3.2 Seznam vrcholů a hran

Relativně úspornou a univerzální metodou, jak zadat graf, je použití seznamu vrcholů a hran. Vrcholy jsou opět popsány pouhým jejich výčtem, případně názvem, je-li namísto označení v_1 použito například České Budějovice. Hrany jsou reprezentovány seznamem trojic: označením (názvem) hrany a počátečním a koncovým vrcholem hrany. Je to vlastně reprezentace přímo podle definice grafu. Možný způsob zjednodušení je vypuštění názvu hrany a její reprezentace pouze počátečním a koncovým vrcholem. Seznamem vrcholů a hran lze popisovat orientované, neorientované i ohodnocené grafy. Pokud je graf orientovaný, dodrží se v zápisu hrany

posloupnost vrcholů (směr šipky). Pokud orientovaný není, je možné vrcholy zaměnit. Pro ohodnocený graf se k hraně či vrcholu přidává informace o hodnotě. Reprezentace grafu z obrázku 3.1 pak vypadá následovně:

Vrcholy: v_1, v_2, v_3, v_4, v_5
Hrany: $(e_1, v_1, v_2), (e_2, v_2, v_3),$
$(e_3, v_3, v_5), (e_4, v_5, v_1),$
$(e_5, v_1, v_4), (e_6, v_5, v_4).$

3.3 Seznam vrcholů a jejich okolí

Úspornější variantou předchozí metody je použití seznamu vrcholů a seznamu okolí vrcholů. Množina vrcholů je popsána stejně jako u předchozí metody. Hrany jsou popisovány tak, že pro každý vrchol *x* je uveden výčet prvků z množiny hran $H^+_{\vec{G}}(x)$. Každý tento prvek, tedy hrana, je reprezentován svým označením (názvem) a koncovým vrcholem, protože počáteční vrchol je stejný pro celou množinu hran. Graf z obrázku 3.1 bude tedy zapsán:

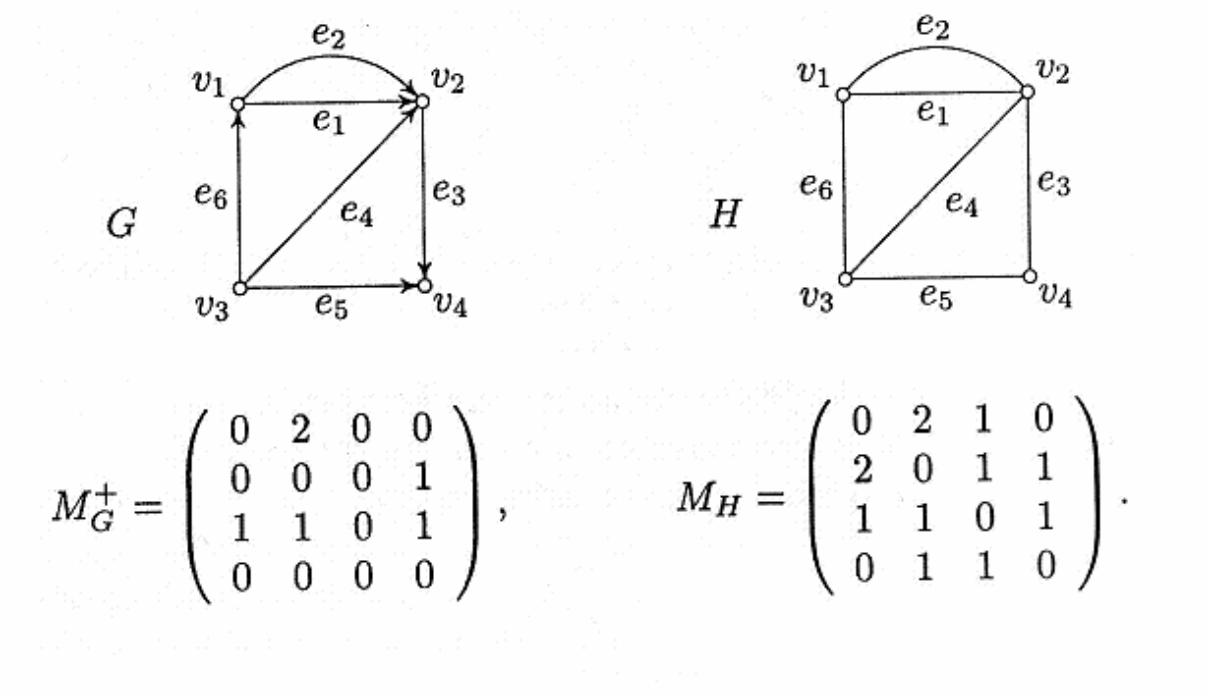
v_1: $(e_1, v_1), (e_5, v_4), (e_4, v_5)$
v_2: $(e_2, v_3), (e_1, v_1)$
v_3: $(e_3, v_5), (e_2, v_2)$
v_4: $(e_6, v_5), (e_5, v_1)$
v_5: $(e_4, v_1), (e_6, v_4), (e_3, v_3)$

Je taktéž možné v závislosti na účelu reprezentace grafu použít množinu $H^-_{\vec{G}}(x)$ nebo pro neorientovaný graf množinu $H_G(x)$, což je ovšem méně úsporné, protože každá hrana je uvedena ve dvou seznamech, jak je z příkladu patrné. Z tohoto pohledu by mohlo být vhodnější použití některé orientace grafu.

3.4 Matice sousednosti

Matematicky velmi elegantní metodou reprezentace grafu je jeho vyjádření maticí sousednosti. Nechť $\vec{G}$ je orientovaný graf. Zvolíme-li na *n* vrcholech libovolně, ale pevně, pořadí vrcholů $v_1,.., v_n$, můžeme grafu $\vec{G}$ přiřadit *matici sousednosti* $M(\vec{G})$

řádu *n* předpisem $m_{ij} = m_{\vec{G}}(v_i, v_j)$. Pro neorientovaný graf G se *matice sousednosti M(G)* definuje předpisem $m_{ij} = m_G(v_i, v_j)$, přičemž pro všechna *i,j* platí, že $m_{ij}=m_{ji}$. Matice sousednosti neorientovaného grafu je tedy *symetrická*. Podle symetričnosti matice sousednosti však nelze rozhodnout o tom, zda-li je graf orientovaný nebo neorientovaný. Pokud pro ohodnocený graf *G* platí, že násobnost všech jeho hran m_G je rovna jedné nebo nule (nula vlastně značí neexistenci hrany), může se prvku m_{ij} přiřadit ohodnocení hrany (nenulovost značí existenci hrany a zároveň vyjadřuje její ohodnocení). Pro praxi je tato metoda ovšem méně vhodná obzvlášť v případě, že graf má velký počet vrcholů a malý počet hran. Taková matice sousednosti by obsahovala mnoho nul.



Obrázek 3.2 : Matice sousednosti

3.5 Matice incidence

Podobně elegantně, leč taktéž neúsporně, se dá graf vyjádřit pomocí matice incidence. Nechť $\vec{G}$ je orientovaný graf bez smyček s *n* vrcholy a *m* hranami. Opět se libovolně, ale pevně zvolí pořadí vrcholů $v_1,.., v_n$, zde však navíc i pořadí hran $e_1,.., e_m$. Pak může být grafu $\vec{G}$ přiřazena matice incidence $B(\vec{G})$ typu *(m, n)* předpisem $b_{ij} = 1$ pokud je v_i počátečním vrcholem hrany e_j, $b_{ij} = -1$ pokud je v_i koncovým vrcholem

hrany e_j a $b_{ij} = 0$ ve všech ostatních případech. Pro neorientované grafy bez smyček platí $b_{ij} = 1$, pokud je vrchol v_i incidentní s hranou e_j, jinak $b_{ij} = 0$. Neúspornost metody je zřejmá.V každém sloupci budou právě dva nenulové prvky. Dvě jedničky u grafu neorientovaného a *1* a *-1* u grafu orientovaného.

$$B_G = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & -1 & 0 & -1 & 0 \end{pmatrix}$$

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

Obrázek 3.3 : Matice incidence

3.6 Zadání pomocí algoritmu

V některých případech lze graf zadávat nepřímo pomocí algoritmů. A to tehdy, když není pro práci s grafem nezbytné znát seznam hran jako celek, ale stačí když algoritmus v případě, kdy narazí na vrchol *v*, dokáže vygenerovat postupně všechny hrany z vrcholu *v* vycházející. Totéž platí pro vrcholy. Není nutná explicitní znalost, ale algoritmus o vrcholu informuje ve chvíli, kdy zpracovává hranu, která ve vrcholu končí nebo začíná. Tento přístup se používá při zpracování rozsáhlých grafů, které jsou definovány nepřímo prostřednictvím popisu nějaké situace nebo soustavy.

4. Zpracování grafů na počítači

4.1 Datové struktury pracující s maticemi

Jak už bylo řečeno, matice incidence a matice sousednosti nejsou příliš vhodné pro reprezentaci grafu s malým počtem hran. Díky velkému počtu nulových prvků v matici je vyhledávání nějaké nebo další hrany vycházející z vrcholu relativně pomalé. Výhodné může být použití matice pro prosté ohodnocené grafy s mnoha hranami. V předchozím textu byla zmíněna možnost použití ohodnocení hrany jako signálu, že hrana mezi vrcholy existuje a nuly, pokud tomu tak není. Místo nuly může být však zvolena jiná konstanta v závislosti na typu úlohy a to tak, aby se s ní v použitém algoritmu dalo bez problémů pracovat. Může nastat případ, kdy ohodnocení hrany bude nula a v programu dojde ke konfliktu z důvodu milné informace o neexistenci hrany. Například pro hledání nejkratší cesty tak bude vhodnější použití ekvivalentu nekonečna nebo maximální integerové hodnoty namísto nuly. Algoritmus tak bude používat nový úplný graf místo původního, ovšem ohodnocení některých hran bude tak vysoké, že výsledek nebudou moci nijak ovlivnit. Neohodnocené grafy lze také ukládat tak, že pro každý vrchol vytvoříme kompaktní seznam následníků. Následníkem vrcholu v_i je každý jeho soused. Pokud je uvažována posloupnost vrcholů $v_1,.., v_n$, kde n je počet vrcholů grafu, je pro vrchol v_i následníkem vrchol v_{i+1} a předchůdcem vrchol v_{i-1}. Při použití kompaktního seznamu následníků by tak každý *i*-tý řádek dvourozměrného pole s proměnou délkou řádku obsahoval čísla následníků vrcholu *i*. Je-li neohodnocený graf navíc prostý, tedy bez smyček a násobných hran, může být matice sousednosti uložena jako matice bitů. To sice zmenší paměťové nároky, práce s bity je však složitější.

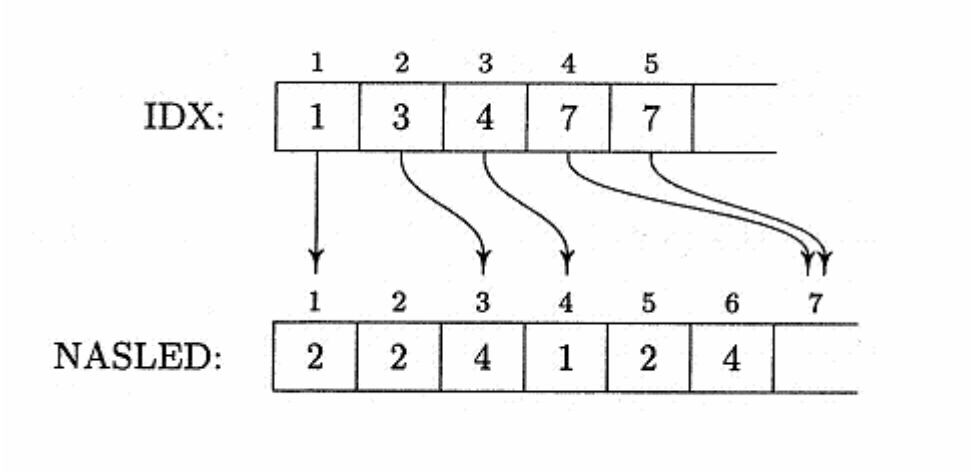
4.2 Seznam hran v jednorozměrných polích

Zde se vychází z předpokladu, že vrcholy i hrany jsou očíslovány od 1 do *n*, respektive od 1 do *m,* v libovolném, ale pevném pořadí. Seznam hran pak je uložen pomocí dvou polí P_v a K_v, kde pro hranu s indexem *i* lze najít počáteční vrchol v poli P_v na pozici *i* a koncový vrchol v poli K_v na pozici *i*. Prvky $P_v(i)$ a $K_v(i)$ tedy určují hranu. V případě, že je graf ohodnocený, bude vytvořeno další pole, ve kterém budou na pozici *i* hodnoty vyjadřující ohodnocení hrany s indexem *i*. Tento způsob se využívá v programovacích jazycích, které neumožňují bohatší strukturování dat. Nevýhodou je nutnost předem určit velikost pole a pomalý přístup k sousedům vrcholu. Řešením může

být použití dalšího pole, jehož prvek na pozici *i* bude obsahovat index hrany, která má stejný počáteční vrchol jako hrana *i*. Pomocí hodnot v tomto poli se dá snadno přistupovat ke všem hranám vycházejícím ze stejného vrcholu. Alternativou je, že třetí pole bude obsahovat indexy hran se stejným koncovým vrcholem.

4.3 Seznamy následníků v jednorozměrném poli

Zde jsou použita dvě pole. Pole IDX a pole NASLED. Hodnota IDX(i) je pořadovým číslem prvku, kterým v poli NASLED začíná seznam následníků vrcholu *i*. V poli NASLED jsou následníci uloženi těsně za sebou. Konec seznamu následníků vrcholu i se pozná jako začátek seznamu pro vrchol *i+1*. Kvůli ukončení posledního seznamu je v poli IDX o jeden prvek víc než vrcholů. Každý prvek pole NASLED se dá pokládat za záznam o hraně, kde je vynechaný počáteční vrchol. Délka tohoto pole je rovna počtu hran. Pro uchovávání případného ohodnocení se použije další pole, jako tomu bylo v předchozím případě. Tento způsob uložení grafu v počítači má velmi nízké paměťové nároky a umožňuje rychlou práci a vyhledání následníků, ale vyhledávání a práce s předchůdci je o to složitější.

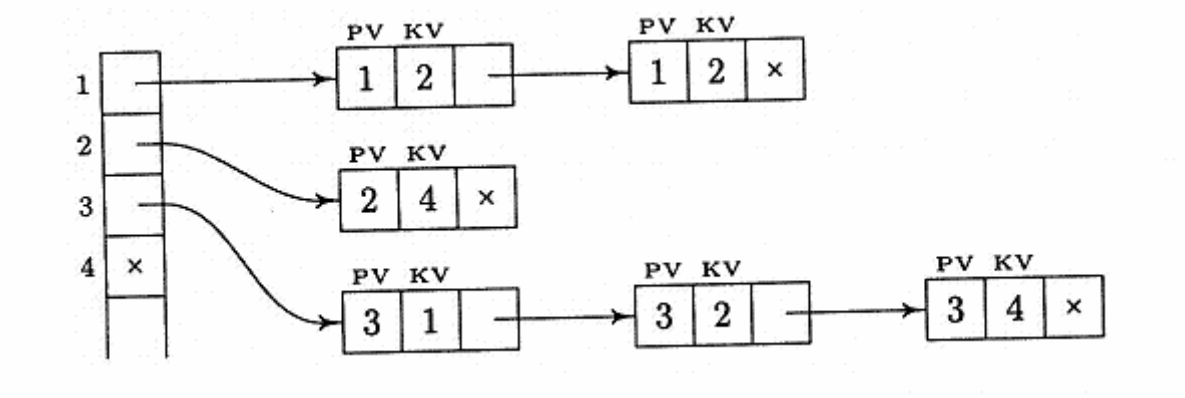


Obrázek 4.1 : Seznam následníků v jednorozměrném poli pro graf G z obrázku 3.2

4.4 Datové struktury založené na ukazatelích

Moderní programovací jazyky umožňují práci s adresami datových objektů a tudíž pohodlnou realizaci spojových seznamů. Data popisující vrchol či hranu se sdružují do záznamů. Seznam obsahuje kromě dat samotných i ukazatel na další záznam

v seznamu. Výhodou spojových seznamů oproti polím je, že není nutné dopředu definovat velikost seznamu. I z hlediska realizace změn ve spojovém seznamu je tato struktura flexibilnější. Malou vadou na kráse jsou vyšší nároky na paměť. Pole a spojový seznam mohou být použity zároveň.



Obrázek 4.2: Reprezentace grafu spojovým seznamem pro graf G z obrázku 3.2

5. Algoritmy pro hledání nejkratších cest v grafu

Algoritmy hledající nejkratší cesty v grafech mohou být rozděleny do dvou základních skupin. Algoritmy řešící problém nejkratší cesty mezi dvěma danými vrcholy nebo z jednoho zadaného (single source problem) a algoritmy, které naleznou nejkratší cestu mezi každými dvěma vrcholy grafu (all pairs shortest path problem).

5.1 Pojem algoritmus

Algoritmus (nebo dřívějším pravopisem algorithmus) je přesný návod či postup, kterým lze vyřešit daný typ úlohy. Pojem algoritmu se nejčastěji objevuje při programování, kdy se jím myslí teoretický princip řešení problému (oproti přesnému zápisu v konkrétním programovacím jazyce). Obecně se ale algoritmus může objevit v jakémkoli jiném vědeckém odvětví. Jako jistý druh algoritmu se může chápat i například kuchyňský recept. V užším smyslu se slovem algoritmus rozumí pouze takové postupy, které splňují některé silnější požadavky jako konečnost a determinovanost.

5.2 Konečnost

Každý algoritmus musí skončit v konečném počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný. Postupy, které tuto podmínku nesplňují, se mohou nazývat výpočetní metody. Speciálním příkladem nekonečné výpočetní metody je reaktivní proces, který průběžně reaguje s okolním prostředím. Mezi algoritmy se však často zahrnují i takovéto postupy.

5.3 Determinovanost

Každý krok algoritmu musí být jednoznačně a přesně definován. V každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat. Protože běžný jazyk obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, byly pro zápis algoritmů navrženy programovací jazyky, ve kterých má každý příkaz jasně definovaný význam. Vyjádření výpočetní metody v programovacím jazyce se nazývá program.

5.4 Dijkstrův algoritmus

Dijkstrův algoritmus slouží k nalezení nejkratší cesty mezi dvěma vrcholy grafu. Algoritmus pracuje nad orientovaným i neorientovaným grafem. Předpokladem však je, že graf musí být vždy ohodnocený a to nezáporně. Dalším předpokladem je souvislost grafu. Pokud je graf nesouvislý, tedy do nějakého vrcholu nevede hrana (v případě neorientovaného grafu ani z něj), nemůže být do tohoto vrcholu nalezena cesta, natož ta nejkratší. Počáteční vrchol, ten, ze kterého hledáme cestu, se značí v_0. Každý vrchol kromě počátečního má na začátku nastavenu hodnotu nejkratší cesty do něj na nekonečno (nebo ekvivalent). U počátečního vrcholu je tato hodnota nulová. Každý vrchol se může v průběhu algoritmu nacházet ve dvou stavech: dočasně ohodnocený, trvale ohodnocený. Pokud je vrchol trvale ohodnocený, je u něj známa délka nejkratší cesty. V každém kroku algoritmu se pak provede následující: je vybrán vrchol *w*, který je dočasně ohodnocený, a mezi všemi takovými vrcholy je délka zatím nalezené cesty do něj nejkratší možná (v prvním kroku to tedy bude vrchol v_0 a položíme $w = v_0$). Vrchol *w* se prohlásí za trvale ohodnocený. Dále je testováno, zda pro nějaký vrchol *v* není cesta z vrcholu v_0 do *w* a po hraně z *w* do *v* kratší, než zatím nalezená cesta z v_0 do *v* a je-li tomu tak, pak se změní hodnota délky nejkratší cesty do *v*. Stejný postup je prováděn, dokud nejsou všechny vrcholy grafu trvale ohodnoceny nebo dokud vrcholy, co nejsou trvale ohodnoceny, mají délku nejkratší cesty do nich rovnou nekonečnu. To, že u nějakého vrcholu zůstala hodnota nejkratší cesty „nekonečno“ značí, že se graf skládá z více nesouvislých částí, což je v rozporu s předpoklady a u většiny implementací vyvolá chybové hlášení. Algoritmus tedy najde nejkratší cestu ze zadaného vrcholu do všech ostatních vrcholů grafu. Pro možnost výpisu nejkratší cesty do zvoleného koncového vrcholu je nutné v algoritmu uchovávat posloupnost vrcholů, po kterých nejkratší cesta vede. Tu je nejvýhodnější uchovávat v poli a jednotlivé prvky zjišťovat pomocí ukazatele *předchozí*, který ukazuje na vrchol, do kterého vedla nejkratší cesta v předchozím kroku. Jinými slovy, pokud je $v_0,.., v_k$ nejkratší cesta do vrcholu *k*, pak *předchozí* $[v_k] = v_{k-1}$ a tak dále až k vrcholu v_0. Konečnost výpočtu je zřejmá. Algoritmus končí maximálně po *n* krocích, kde *n* je počet vrcholů grafu, což vyplývá ze skutečnosti, že při každém opakování cyklu je jeden vrchol označen jako trvale ohodnocený. Pokud k uchování délek do jednotlivých vrcholů grafu v průběhu algoritmu použijeme pole a ne například haldu, má algoritmus časovou složitost $O(n^2)$, kde *n* je počet vrcholů. Více o typech, struktuře a vlastnostech haldy lze nalézt například v [20].

Zápis Dijkstrova algoritmu pomocí pseudokódu:

```
function Dijkstra
    for každý vrchol v z V[G]            // inicializace
        d[v] := nekonečno                // neznámé vzdálenosti do všech vrcholů
        previous[v] := nedef.
    d[s] := 0                            // vzdálenost do počátečního vrcholu
    S := prázdné                         // pevně ohodnocené vrcholy
    Q := V[G]                            // dočasně ohodnocené vrcholy

 while Q není prázdná                 // samotný algoritmus
        u := Vyjmi_Min(Q)                // vyjmutí prvku s nejmenším ohodnocením
        S := S sjednocení s {u}          // označení jako pevně ohodnocený
        for každou hranu (u,v) vycházející z u
            if d[u] + w(u,v) < d[v]      // test jestli je nové ohodnocení lepší než původní
                d[v] := d[u] + w(u,v)
                předchozí[v] := u
            end
        end
    end
```

5.5 Bellman-Fordův algoritmus

Velmi podobný Dijkstrovu algoritmu je algoritmus Bellman-Fordův (dále jen BFA). BFA je vlastně zobecněním Dijkstrova algoritmu. BFA se stejně tak používá k nalezení nejkratší cesty mezi dvěma vrcholy ohodnoceného orientovaného nebo neorientovaného grafu, ale s tím rozdílem, že u BFA může být ohodnocení hran grafu i záporné. V grafu však nesmí existovat cyklus se záporným součtem délek hran dosažitelný z počátečního vrcholu. Jestliže by v grafu G existoval cyklus $w_0,..,w_k = w_0$ takový, že součet délek hran (w_0, w_1) až (w_{k-1}, w_k), (w_k, w_0) je záporný a navíc existuje cesta z počátečního vrcholu v_0 do nějakého vrcholu cyklu w, pak neexistuje minimum z délek cest z v_0 do w, protože je-li $v_0, v_1,.., v_l = w$ libovolná cesta z v_0 do w, pak cesta $v_0, v_1.., v_l = w = w_0, w_1,.., w_k = w_0$ (původní cesta doplněná o oběh cyklu) je kratší a množina délek cest z v_0 do w by tedy neměla minimum. V případě, že se záporný cyklus v grafu vyskytne, algoritmus se zacyklí. Toto nežádoucí chování je možno ošetřit tím, že se do algoritmu přidá čítač cyklů, který po dosažení maxima opakování algoritmus ukončí. Kolik průchodů cyklem je přípustných a kolik už ne je dáno velikostí grafu, respektive počtem vrcholů. Ukončení algoritmu čítačem tedy detekuje přítomnost záporného cyklu v grafu a BFA je tedy možno použít i v tomto smyslu. Časová složitost

BFA je vyšší než u Dijkstrova algoritmu, tudíž k jeho použití se přistupuje pouze tehdy, je-li reálný předpoklad výskytu záporného ohodnocení. Udávaná složitost je *O(n*m)*, kde *n* je počet vrcholů a *m* je počet hran. Ve výpočtu pomocí BFA mohou být vrcholy ve třech stavech: nedosažený, k probrání a probraný. Vrcholy označené k probrání jsou uchovávány ve frontě *Q*. Každý vrchol *v* má svoji proměnou *E(v)*, ve které je po dobu výpočtu uchováván jakýsi odhad nebo prozatímní délka cesty z počátku do vrcholu *v*. Na začátku nebo pokud jsou nedosažené, mají všechny vrcholy kromě počátečního tuto hodnotu nedefinovanou, u počátku je nulová. Dále má každý vrchol v proměnné *P(v)* ukazatel na předchůdce vrcholu *v*. Na konci výpočtu budou hodnoty *E(v)* udávat délku nejkratší cesty, která bude jednoznačně určena ukazateli *P(V)*.

Zápis BFA pomocí pseudokódu:

```
Function BFA
v0 je označen k probrání a zařazen do Q, ostatní vrcholy jsou neprobrané
E(v0): = 0,
E(Vi) := nedef.                    // i = 1,.., k
P(vi): = nedef.                    // i = 0,.., k

While Q je neprázdná
    v :=Vyjmi_První ( Q)
    for všechny hrany (v, w) vycházející z v
        if vrchol w je neprobraný nebo  (l := E(v) + L(v; w))<E(w))
            E(w) := l
            P(w) := v
        end
    end
end
```

5.6 Floyd-Warshallův algoritmus

Předchozí dva algoritmy dokázaly najít nejkratší cestu ze zadaného vrcholu do všech ostatních. Floyd-Warshallův algoritmus hledá nejkratší cestu mezi každými dvěma vrcholy grafu. Realizuje se nad ohodnoceným a orientovaným grafem. Stejně jako u BFA mohou být hrany záporně ohodnocené, ale v grafu se nesmí vyskytovat záporný cyklus. Do algoritmu vstupuje matice sousednosti jako reprezentace grafu. Algoritmus pracuje následujícím způsobem. V každém kroku algoritmu zjišťuje, jestli existuje mezi vrcholy *i,j* kratší cesta přes vrchol *k*, pokud ano, nastaví vzdálenost v *D[i][j]* (viz. níže) na novou velikost a do *Path[i][j]* zaznamená vrchol *k*. Matice *D* je

matice aktuálně spočtených nejkratších vzdáleností a *Path* je matice nejkratších mezicest. Všechny její prvky jsou inicializovány na hodnotu -1. Maticí sousednosti je matice *A*.

Zápis Floyd-Warshallova algoritmu pomocí pseudokódu:

```
Function Floyd_Warshall
int [][] A, int [][] D, int [][] Path     // vstupy

int n = A.length;
for (int i = 0; i < n; i++)           //inicializace
  for (int j = 0;j < n; j++)
       D[i][j] = A[i][j];
       Path[i][j] = -1
  end
end

for (int k = 0; k < n; k++)
  for (int i = 0; i < n; i++)
    for (int j = 0;j < n; j++)
     if (D[i][k] + D[k][j] < D[i][j]) // aktualizace nejkratší cesty
     D[i][j] = D[i][k] + D[k][j];
     Path[i][j] = k;
     end
    end
  end
end
```

Algoritmus má časovou složitost $O(n^3)$ a při použití nad hustými grafy je rychlejší než algoritmus Dijkstrův použitý pro každou dvojici vrcholů zvlášť.

5.7 Johnsonův algoritmus

Základní myšlenkou Johnsonova algoritmu je n-násobné použití Dijkstrova algoritmu, kde *n* je počet vrcholů grafu. Aby bylo možné Dijkstrův algoritmus použít, je nutné graf, který může být obecně záporně ohodnocený, přehodnotit. Získání nového ohodnocení *c'* z původního *c* se nazývá přehodnocením grafu *G* a musí splňovat následující dvě podmínky:

-Pro každou dvojici uzlů *u, v* je nejkratší cesta pro ohodnocení *c* shodná s nejkratší cestou pro ohodnocení *c'*.
-Pro každou hranu *(u, v)* je ohodnocení *c'(u, v)* nezáporné.

Přehodnocení lze provést v čase $O(n*m)$, kde *n* je počet vrcholů a *m* je počet hran. Jako další podprogram se v Johnsonově algoritmu používá BFA. Výsledkem algoritmu je buď matice vzdáleností nebo detekce záporného cyklu a ukončení algoritmu. Efektivnější pro řešení „all-pairs shortest path problem" než Floyd-Warshallův algoritmus je algoritmus Johnsonův při použití nad řídkými grafy. Pro určení složitosti Johnsonova algoritmu je rozhodující *n*-násobné použití Dijkstrova algoritmu. Odhad složitosti Johnsonova algoritmu má hodnotu $O(n^2 log\ n + m* n)$, což je stále pro řídké grafy rychlejší než Floyd-Warshallův algoritmus se složitostí $O(n^3)$.

Zápis Johnsonova algoritmu pomocí pseudokódu:

```
Function Johnson
  přehodnocení grafu G na graf G'              //Přidání uzlu s a hran (s, u).

 if not BFA                                    //Našly se v G' cykly záporné w-délky?
      then výpis „graf obsahuje cyklus záporné délky"
      else for každý uzel u z V'
              h(u) := D(s, u)                  //S výsledky BFA proveď přehodnocení hran.
           for každou hranu (u, v) z H'
              c'(u, v) := c(u, v) + h(u) - h(v)
          for každý uzel u z V                 //Pro každý uzel původního grafu spočti c'-vzdálenosti
d'(u, v)a uprav je na c-vzdálenosti.
              Dijkstra
                 for každý uzel v z V
                     D(u, v) := D'(u, v) + h(v) - h(u)
 end
 vrať D
```

5.8 A* search algoritmus

Algoritmus „A hvězda" hledá nejkratší cestu mezi zadaným počátečním a koncovým vrcholem. Používá heuristickou funkci (heuristický odhad) *h(x)*, který každému vrcholu *x* přiřadí odhad vyjadřující pravděpodobnost, že nejkratší cesta do koncového vrcholu povede právě přes tento vrchol. Postup algoritmu je stejný jako v případě Dijkstrova algoritmu jen s tím rozdílem, že v průběhu prohledávání jsou

vrcholy navštěvovány v závislosti na hodnotě tohoto heuristického odhadu. Jinými slovy, na základě jeho hodnoty je jim přiřazována priorita. Priorita je vyjádřena funkcí $f(x) = g(x) + h(x)$, kde $g(x)$ je ohodnocení hrany a $h(x)$ je výše zmíněný heuristický odhad. Čím nižší má funkce $f(x)$ hodnotu, tím vyšší má vrchol x prioritu a je „probírán“ dřív. Heuristický odhad může být vyjádřením například toho, jestli vrchol x leží ve směru polohy cílového vrcholu nebo jak je vrchol x od cílového vrcholu vzdálen po přímé spojnici (například v mapě). Časová náročnost závisí na optimálnosti heuristického odhadu. V nejhorším případě je počet navštívených vrcholů exponenciální v porovnání s počtem vrcholů nejkratší cesty. Může být však polynomiální pokud odhad h splňuje podmínku $|h(x) - h^*(x)| \in O(\log h^*(x))$, kde $h^*(x)$ je hodnota optimálního odhadu.

Zápis A*search algoritmu pomocí pseudokódu

```
function A*search
   probrané := prázdné pole
   q := nová_fronta
  while q je neprázdná
         p := vyjmi_první(q)
         x := poslední vrchol z p
      if x je v probrané
         continue
      end
      if x = koncový vrchol
         return p
     end
      přidej x do probraných
      for každý vrchol y z následníků(p)
         přidej_do_fronty(q, y)
      end
 end
  return chyba
```

5.9 Problém obchodního cestujícího (traveling salesman problem)

Speciálním případem úlohy nalezení nejkratší cesty je tzv. problém obchodního cestujícího. Problém obchodního cestujícího je obtížný diskrétní optimalizační problém, matematicky vyjadřující a zobecňující úlohu nalezení nejkratší možné cesty procházející všemi zadanými body na mapě. V úloze jde o to, jak v ohodnoceném

(případně úplném) grafu efektivně nalézt tzv. Hamiltonovskou kružnici s minimálním součtem ohodnocení hran. Známá formulace úlohy je také: je dáno *n* měst, mezi nimiž je systém silnic, u nichž je známa vzdálenost. Úkolem obchodního cestujícího je navštívit všechna města (právě jednou) a vrátit se do výchozího města. Přitom klademe důraz na to, aby ujetá vzdálenost (cesta) byla co nejkratší. Problém nespočívá ani tak ve stanovení libovolného postupu nalezení nejkratší cesty – jeden takový postup je totiž skoro samozřejmý: stačí jednoduše prohledat všechny možné kružnice procházející všemi městy a vybrat nejkratší z nich. Obtíž však je, že s rostoucím počtem měst (či vrcholů grafu) počet možných cest velice rychle narůstá, a tím se doba potřebná k propočtu „hrubou silou“ na počítačích stává zcela neúnosnou už při několika málo desítkách vrcholů. Klíčová obtíž je tedy v nalezení časově efektivního algoritmu hledání nejkratších cest.Tato úloha patří mezi tzv. NP-úplné (non-polynomial complete) úlohy, tzn. v obecném případě není známo ani jak nalézt přesné řešení v rozumném čase a dokonce ani zda vůbec může existovat algoritmus, který takové řešení najde v čase úměrném nějaké mocnině počtu vrcholů. Že jde o nedeterministicky polynomiální problém je patrné z toho, že nedeterministický počítač, umožňující v každém kroku rozvětvit výpočet na libovolný počet větví, by mohl začít v některém „městě“, rozdělit propočet délky trasy na tolik větví, kolik z města vede silnic, a v každém z cílových měst postupovat stejně – s výjimkou tras vedoucích do již navštívených měst. Tak by prohledal všechny možné trasy v n výpočetních krocích, pokud počet měst činí *n*, a rozvětvil by se maximálně do $(n-1)!$ větví. Prakticky se podobná úloha obvykle řeší pouze přibližně (heuristické algoritmy, např. genetické algoritmy, tabu prohledávání, neuronové sítě, algoritmus mravenčích kolonií ACO). Tím se za cenu vzdání se nároku na nalezení přesného řešení dosahuje prakticky použitelných výsledků.

6. Užití metod pro hledání optimálních cest v GIS

GIS, neboli geografický informační systém, je systém umožňující uživateli snadnou a efektivní správu, analýzu a zobrazení prostorových dat. Tato data reprezentují objekty a situace z nejrůznějších oborů lidské činnosti, obecně svět kolem nás. Přesné vyjádření skutečnosti je však prakticky nemožné. Pro zjednodušení se používají tzv. datové modely. *Datový model* je abstrakce reálného světa, zobrazující pouze ty jeho části, které jsou pro použití v dané aplikaci relevantní. Datový model definuje specifické skupiny entit, vztahy mezi nimi a jejich atributy. Existují dva základní druhy datových modelů: rastrový a vektorový.

6.1 Rastrový datový model

V rastrovém datovém modelu je obraz světa reprezentován sítí buněk. Každá buňka zná svoji polohu (souřadnice) v této síti, cenu přesunu do všech sousedících buněk a svoji hodnotu vyjádřenou atributem. Buňka má obvykle tvar čtverce, ale může být i trojúhelníková nebo hexagonální. Velikost buněk může být stejná nebo různá. Rastrový datový model se používá především pro ukládání dat reprezentujících souvislé povrchy, mající jinou hodnotu v každém bodě jako jsou letecké snímky, teplotní mapa apod. a aplikacích pro management zdrojů.

6.2 Vektorový datový model

Ve vektorovém datovém modelu je využíváno k popisu reality tří prvků: bodu, linie a polygonu (oblasti). Základním elementem tohoto datového modelu je bod. Spojením dvou bodů vzniká linie. Body mohou být spojeny přímou spojnicí, částí kružnice (obloukem) nebo složitější křivkou, jako je například SP-line. Více linií za sebou tvoří řetězec linií a pokud je tento uzavřený, vzniká polygon. Stejně jako u rastrového modelu, tak i zde každý z těchto elementů nese nějakou informaci o sobě. Polygon o tom, jaké linie ho tvoří, se kterými jinými polygony sousedí, linie taktéž atd.. Na rozdíl od rastrového modelu je pro ukládání těchto informací použito složitějších struktur (modelů). Podrobně se o nich lze dočíst například v [5]. Vektorový datový model je používán pro data mající diskrétní hranici (přesně vymezenou oblast se stejnou hodnotou), proto se tento model využívá zejména v aplikacích zabývajících se dopravou a marketingem.

6.3 Síť a síťová analýza

Velmi významnou skupinou objektů, které je výhodné zobrazovat a spravovat pomocí geoinformačních systémů jsou objekty tvořící na zemském povrchu (případně pod i nad ním) síť.

Síť je množina vzájemně propojených linek, reprezentujících možnou cestu pohybu zdrojů (předmětu obecně) z jednoho umístění do jiného. Konce nebo křížení linek se nazývají síťové uzly. Analogickým výrazem teorie grafů pro síť je graf a pro linku hrana. Příkladem mohou být dopravní nebo rozvodná síť jakékoliv formy a rozsahu od silniční sítě v rámci města po mezinárodní ropovody. Spojení mezi jednotlivými objekty však nemusí být jen fyzické ale i intuitivní jako třeba v případě telekomunikační sítě.

V GIS se sítě dělí do dvou kategorií na dynamické sítě a statické sítě. U statické sítě se ohodnocení nemění (kilometrové ohodnocení silniční sítě), případně se mění ve velkém časovém intervalu, který je v řádu měsíců nebo let. V dynamické síti se ohodnocení mění neustále v závislosti na čase. S dynamickou sítí pracují například tzv. AVL systémy (Automatic Vehicle Location), které v reálném čase informují o stále se měnící situaci v dopravě a na základě toho navigují. Pro usnadnění se dynamická síť převádí na statickou síť. To se dá provést dvěma způsoby. Nově vzniklé síti se buď přiřadí ohodnocení ve startovacím čase a startovacím bodě, kdy se toto během vykonávání nějaké operace nemění nebo se síti přiřadí ohodnocení pouze pro předem daný časový interval a síť tak bude statická alespoň pro tento interval. Po tomto intervalu se síť přehodnotí a operace je provedena s těmito novými hodnotami.

Data popisující síť se pak v prostředí geoinformačního systému stávají předmětem tzv. síťové analýzy. Pod pojmem síťová analýza si můžeme představit soubor funkcí, pomocí kterých je uživatel schopen modelovat a optimalizovat procesy nebo situace probíhající v síti. Podle definice firmy ESRI (jedna z předních firem zabývajících se GIS) je síťovou analýzou jakákoliv metoda řešící problém v síti, jako je dostupnost, kapacita nebo cena průchodu, využívající propojení sítě. Způsob, jakým operace síťové analýzy probíhají, závisí na tom, jestli geoinformační systém pracuje s vektorovými nebo rastrovými daty. Vektorová data jsou díky své reprezentaci (bod, linie, polygon) vhodnější a častěji využívaná pro síťovou analýzu. Rastr tvořený sítí buněk se na první pohled pro síťovou analýzu přímo nabízí. Opak je pravdou. Práce

s rastrem je daleko složitější a má řadu nevýhod a omezení. Rastrových dat se proto častěji využívá pro analýzy povrchů. Jsou ale i GIS, které umí rastrová data zpracovat tak, aby byla pro síťovou analýzu použitelná.

6.4 Realizace sítě nad rastrovými daty

Jedním z GIS, který takovýto převod dokáže provést je systém MFworks. MFworks realizuje síť v několika krocích. Prvním krokem je propojení středů buněk se stejnou hodnotou atributu do linie. V druhém kroku je spojnicím mezi jednotlivými buňkami přiřazen směr, kterým je možný pohyb mezi buňkami, a informace o buňce, do které se daným směrem dá dostat. Třetím krokem je přiřazení cen průchodu buňkou a následné vypočtení délek cest z uzlu do uzlu. Toto je jen velmi hrubý popis postupu, detailněji je realizace sítě v MFworks popsána v [8].

6.5 Realizace sítě nad vektorovými daty

Jak již bylo řečeno, vektorová data jsou častěji využívaná pro modelování sítí. Jednotlivé uzly sítě (vrcholy) jsou reprezentovány body, které jsou zároveň koncovými nebo mezilehlými body nějaké linie, popřípadě místem jejich křížení. Linie pak představují linky (hrany) a tedy možnou cestu pohybu z jednoho uzlu do druhého. Vektorová reprezentace dat tedy tvoří síť sama o sobě. Při použití terminologie firmy ESRI je takto reprezentována tzv. geometrická síť, tedy síť, ve které je konektivita prvků založena na jejich geometrické shodnosti. Informace o propojení a ohodnocení jsou uloženy v jiné, tzv. logické síti.

6.6 Algoritmy pracující nad sítí

V GIS se pro výpočet nejkratších cest nejčastěji používají algoritmy vycházející z Dijkstrova algoritmu případně používající heuristiku jako A*search algoritmus. Pokud je síť rozsáhlejší, používá se složitějších, ale efektivních algoritmů například ACO. Většinu GIS je také možno doplnit vlastními implementacemi algoritmů, jejichž užití je pro danou úlohu vhodnější. Možnosti GIS jsou v tomto ohledu velmi široké.

7. Popis programu

Jedním z cílů této bakalářské práce bylo vytvořit program na vyhledání nejrychlejšího spoje MHD v Plzni. Úkolem tohoto programu je vygenerovat orientovaný graf, ohodnotit jej a najít v něm nejkratší cestu pomocí Dijkstrova algoritmu.

Grafem je v tomto případě zjednodušená síť linek plzeňské městské hromadné dopravy. Síť se skládá z osmi vrcholů, které reprezentují uzlové stanice MHD. Uzlovou stanicí je myšlena taková zastávka MHD, na které se dá přímo přestoupit z jedné linky na druhou (případně více jiných linek). Protože takovýchto uzlových stanic není po Plzni mnoho nebo jsou pro účely této práce nevhodně umístěny, bylo nutné toto zobecnit. Uzlovou stanici zde tedy tvoří i zastávky, mezi kterými je možný přesun v zanedbatelném čase. Takto sjednoceny byly následující zastávky:

Uzel Bory = zastávka tramvaje č.4 Bory a zastávka autobusu č. 30 a 24 Bory

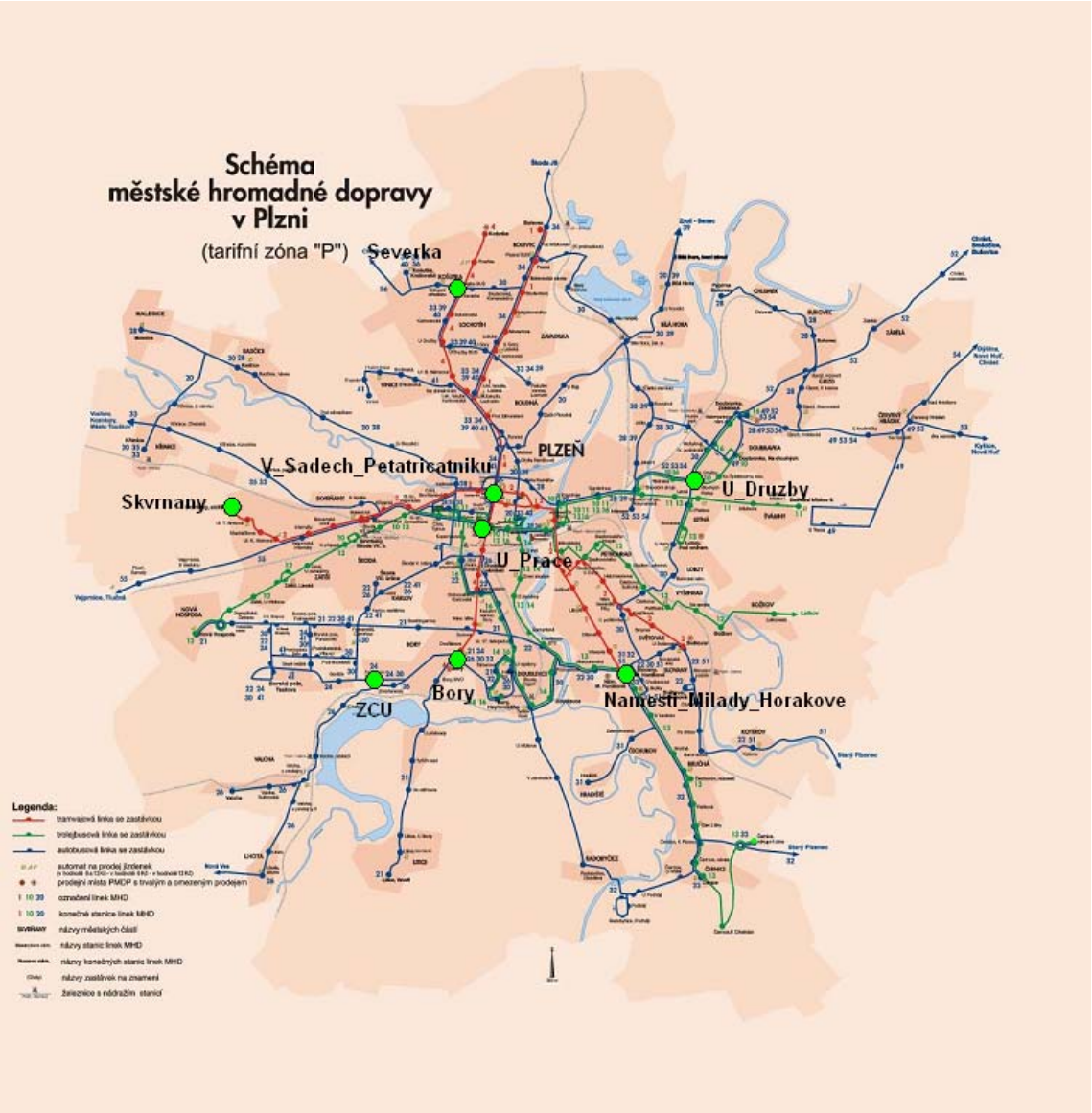
Uzel Náměstí Milady Horákové = zastávka tramvaje č. 1 Náměstí Milady Horákové a zastávka autobusu č. 30 Náměstí Milady Horákové

Uzel U Družby = zastávka trolejbusu č. 16 a autobusu č. 30 Ulice Družby a zastávka trolejbusu č.10 Na Dlouhých

Uzel Severka = zastávka tramvaje č. 4 Severka a autobusu č. 30 Severka, BUS

Uzel U Práce = zastávka trolejbusu č. 16 a 10 U práce, Tylova a zastávka tramvaje č. 4 U Práce, Klatovská

Rozmístění uzlů sítě po Plzni je vidět na následujícím obrázku.



Obrázek 7.1 : Schéma linek MHD v Plzni

Spojení mezi uzly sítě (hrany grafu) reprezentují trasy jízd autobusových linek číslo 24 a 30, trolejbusových linek číslo10 a 16 a tramvajových linek číslo 1, 2 a 4.

Nejkratší cesta je hledána ve smyslu nejkratšího času nutného pro přepravu mezi uživatelem voleným počátečním vrcholem a pevně daným koncovým vrcholem, kterým je autobusová zastávka u areálu ZČU na Borech.

Program je napsaný v objektově orientovaném jazyce Java pomocí editoru JCreator a jeho grafické rozhraní je vytvořeno pomocí editoru NetBeans. Program tvoří

šest tříd (class): *jizdnirady*, *zastavky*, *WeigtedGraph, Dijkstra*, *GUI a Console,* které navzájem používají své metody a proměnné. Třídy *WeigtedGraph* a *Dijkstra* byly převzaty z knihovny na adrese http://www.cs.fit.edu/~ryan/java/programs/graph a upraveny pro potřeby této práce. Třída *Console* je taktéž převzatá a je volně dostupná na adrese http://www.comweb.nl/java/Console/Console.html .

7.1 Třída jizdnirady

Tato třída obsahuje pouze jízdní řády jednotlivých linek pro každou uzlovou stanici. Jízdní řád je reprezentován dvourozměrným polem s proměnnou délkou řádku. Jednotlivé indexy pole jsou typu *string*. Pokud dvě linky MHD jezdí ze stejné uzlové stanice po stejné trase, obsahuje prvek pole kromě časového údaje také číslo linky namísto toho, aby byly použity dva jízdní řády.

7.2 Třída zastávky

Tato třída obsahuje zejména metody vytvářející ohodnocení grafu, tedy metody zjišťující, jak dlouho bude trvat cesta spojem a čekání na další v závislosti na zadaném nástupním čase a nástupní stanici, respektive zjišťující pouze dobu čekání, doba jízdy je pevná. Každé z těchto metod je na začátku předán čas odjezdu z nějaké uzlové stanice a k němu je přičten pevný čas jízdy do další uzlové stanice. Tím se dostane čas příjezdu do této stanice. V této se pak prohledá jízdní řád linky, na kterou se přestupuje a vyhledá se první možný čas odjezdu z této uzlové stanice. Tento čas a čas příjezdu se odečtou a dostane se doba čekání. Metody vracejí dobu jízdy mezi dvěma uzly a čekání na další spoj tedy ohodnocení hrany grafu. Třída dále obsahuje metody pro čtení vstupů *ctiString*, *ctiInt*, metodu *cas,* která sčítá datový typ *string* a datový typ *integer,* metodu *casInt,* která sčítá dvě hodnoty typu *integer*, metodu *zadani*, která v závislosti na vybrané nástupní stanici volá metodu pro vygenerování grafu ze třídy *WeigtedGraph*.

7.3 Třída WeightedGraph

Třída *WeigtedGraph* obsahuje metody generující ohodnocený orientovaný graf v závislosti na zvolené nástupní stanici (počátečním vrcholu). Tento graf je pro každou nástupní stanici jiný, jinak orientovaný a pochopitelně jinak ohodnocený. Pro každý graf je pomocí matice sousednosti *edges* uchováváno ohodnocení grafu. Ve stejně strukturované matici *linky* jsou uchovávána čísla linek, spojujících jednotlivé zastávky.

Dále jsou programem používány metody *setLabel*, která vytváří vrcholy grafu a *addEdge*, která mezi nimi vytváří ohodnocené a orientované hrany. Třída *WeightedGraph* obsahuje ještě několik dalších metod, které jsou jen pomocné.

7.4 Třída Dijkstra

Tato třída obsahuje čtyři metody. Metoda *dijkstra* postupem popsaným v předchozí kapitole realizuje Dijkstrův algoritmus. Pomocí metody *minVertex* je kontrolováno jestli je graf souvislý a jestli je tudíž možné nalézt nejkratší cestu do všech vrcholů. Metoda také provádí kontrolu hodnoty nejkratší cesty a její případné přehodnocení. Metoda *printPath* pak ukládá posloupnost zastávek a vypisuje výsledky hledání nejkratší cesty. Poslední metoda *dvojtecka* je pomocná a pouze upravuje formát času pro výstup na obrazovku.

7.5 Třída GUI

V této třídě je definováno grafické rozhraní, které se zobrazí při spuštění programu a pomocí kterého uživatel zadává parametry pro výpočet nejkratší cesty a kde se zároveň zobrazují výsledky. Třída obsahuje metodu *main,* která celý program spouští. Zbylé metody *jButton1MouseClicked, jButton2MouseClicked, zavriDialog a closeDialog* říkají grafickému rozhraní, co má být provedeno při kliknutí na nějaké tlačítko.

7.6 Třída Console

Protože není možné do okna grafického rozhraní, respektive do jeho prvku *jTextArea*, vypisovat přímo pomocí metody *System.out.print()*, je nutné výstup vytvořený metodou *printPath* ze třídy *Dijkstra* převést do formy, kterou je v *jTextArea* možno zobrazit. To provádí třída *Console.* Ta vezme veškerý text, který by byl v průběhu programu vypsán do příkazové řádky a přesměruje ho do okna grafického rozhraní v nezměněné podobě.

8. Závěr

Pro hledání nejkratších cest v ohodnocené síti se jako nejvýhodnější jeví použití Dijkstrova algoritmu nebo algoritmu založeném na Dijkstrově jako A* search nebo Johnsonova algoritmu. Pro rozsáhlé sítě s velkým množstvím hran a vrcholů je pak lepší použít sofistikovanějších metod, obzvlášť pokud má být nad sítí řešena i složitější úloha, než jen nejkratší cesta mezi dvěma vrcholy.

Během tvorby programu se ukázalo, že vzhledem velikosti zde používané sítě, není volba algoritmu nijak důležitá. Daleko větším problémem je však ohodnocení sítě. Síť linek MHD je sítí dynamickou. Její ohodnocení se mění neustále v průběhu dne tím, jak houstne doprava a zvyšuje se počet přepravovaných lidí. Dokonce i dva spoje na stejné lince jedoucí několik minut po sobě dorazí často do cílové stanice bezprostředně za sebou. Pro ohodnocení sítě bylo tedy opět nutné použít nějaké zobecnění. Doba jízdy mezi dvěma stanicemi byla získána jako průměr doby jízdy v několika částech dne. Obvykle prvního spoje, posledního spoje a spoje jedoucího během dopravní špičky kolem 15. hodiny. Pokud byly rozdíly mezi dobami jízdy velké, bylo zprůměrováno více hodnot.

Program tedy zobrazí nejkratší cestu mezi zadanou nástupní zastávkou a zastávkou u areálu ZČU, u každé uzlové stanice zobrazí čas odjezdu dalšího spoje a jeho číslo. U zastávky ZČU pak jen čas příjezdu. Tyto časy nejsou a nemohou být zcela přesné, ale vzhledem k malým časovým rozestupům mezi jednotlivými spoji MHD je lze považovat alespoň za orientační, stejně jako časy v jízdních řádech PMDP.

Zdroje informací

Použitá literatura

[1] Demel, J. *Grafy a jejich aplikace.* Praha: Academia, 2002. ISBN 80-200-0990-6.

[2] Sedláček, J. *Úvod do teorie grafů.* Praha: Academia, 1981.

[3] Herout, P. *Učebnice jazyka Java.* České Budějovice: Kopp, 2003. ISBN 80-7232-115-3.

[4] Kolář, J. *Teoretická informatika*. Praha: ČIS, 1996.

[5] Tuček, J. *Geografické informační systémy*. Praha: Computer Press, 1998. ISBN 80-7226-091-X.

[6] Longley, P.A., Goodchild, M.F., Maguire, D.J., Rhind, D.W. *Geographic information systems and science*. Chichester, England: John Wiley & sons, Ltd., 2001. ISBN 0-471-89275-0

Ostatní zdroje

[7] *JustinHeyes-Jones web pages-A*Algorithm*[on-line]. c2006, poslední revize 18.8.2006 [cit. 2007-02-05].URL: <http://www.geocities.com/jheyesjones/astar.html>

[8] *Network analysis in raster GIS using MFworks*[on-line]. [cit. 2007-04-18]. URL: <http://www.husdal.com/mfworks/showcase.htm>

[9] *Technology - GIS*[on-line]. [cit. 2007-04-21]. URL: <http://www.gisdevelopment.net/technology/gis>

[10] *Rapidis - Transportation planning and optimization*[on-line]. [cit. 2007-04-15]. URL: <http://www.rapidis.com/trafficanalyst.htm?gclid=CJvSrJ7JxIsCFR8OZwod-H5YAw>

[11] *ArcRevue 2/2004* [on-line]. Praha:ArcData Praha, 2004. [cit. 2007-03-25]. URL: <http://www.arcdata.cz/publikace/arcrevue/2004/2004-2>. ISSN 1211-2135,

[12] *AGI GIS dictionary-free edition*[on-line]. c1996 [cit. 2007-04-13]. URL: <http://www.geo.ed.ac.uk/agidict/welcome.html>

[13] *GIS Dictionary-ESRI Support*[on-line]. Poslední revize 31.9.2006 [cit.2007-04-13]. URL: <http://support.esri.com/index.cfm?fa=knowledgebase.gisDictionary.gateway>

[14] *Table of Contents for the NCGIA Core Curriculum 1990 Version*[on-line]. Poslední revize 30.4.1997 [cit. 2007-04-10]. URL: <http://www.geog.ubc.ca/courses/klink/gis.notes/ncgia/toc.html>

[15] *GISáček 2005-Petr Fuks: Aplikace pro plánování rozvozu* zboží[on-line]. [cit. 2007-03-06]. URL: <http://gis.vsb.cz/GISacek/GISacek_2005/Sbornik/fuks/fuks.html>

[16] *Shortest path problem - Wikipedia, the free encyclopedia*[on-line].[cit.2007-02-21]. URL: <http://en.wikipedia.org/wiki/Shortest_path_problem>

[17] *Index of /PT/ prednasky* [on-line]. Poslední revize 9.1.2006 [cit. 2007-02-21]. URL: <http://fav.q-e-e.net/PT/prednasky>

[18] Kučera, Luděk. *Dijkstrův algoritmus* [on-line]. 28.12.2003 [cit. 2007-02-15]. URL: <http://kam.mff.cuni.cz/~ludek/skrys/Dijkstra.ps>

[19] Kučera, Luděk. *Bellman-Fordův algoritmus* [on-line]. 28.12.2003 [cit.2007-02-15]. URL: <http://kam.mff.cuni.cz/~ludek/skrys/BellmanFord.ps>

[20] *UIN009* [on-line].URL: <http://ksvi.mff.cuni.cz/~dvorak/vyuka/UIN009/Haldy_tisk.pdf>

Seznam obrázků