UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Hledání nejkratší cesty na mapě

Jan Pospíšil

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan POSPÍŠIL**
Osobní číslo:	**I07760**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Hledání nejkratší cesty na mapě**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je vytvoření aplikace pro vyhledání a zobrazení nejkratší cesty v definované mapě.

Teoretická část:
Vysvětlení základních pojmů z teorie grafů a datových struktur. Vysvětlení např. Dijkstrova a Floyd-Warshallova algoritmu. Popis technologií a nástrojů pro tvorbu GUI a Java ME aplikací.

Implementační část:
V Javě pomocí vybrané datové struktury a vhodných algoritmů navrhnout a realizovat aplikaci pro vyhledání nejkratší cesty.
V desktopové aplikaci bude realizována dopravní síť (orientovaný graf) nad vybranou mapou města. Síť s mapou bude nahrána do mobilního zařízení, ve kterém bude aplikace pro vyhledání nejkratší cesty mezi dvěma zadanými body.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

***Volek, J. Operační výzkum I. Skripta DFJP , Pardubice 2002.**
***Eckel, B. Myslíme v jazyku Java. Knihovna zkušeného programátora. Grada Publishing, 2001.**
***Petzold, C. Programování Microsoft Windows v jazyce C. SoftPress, 2003.**

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

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 9. 8. 2011 Jan Pospíšil

Poděkování

Chci zde poděkovat především svému vedoucímu práce Ing. Zdeňku Šilarovi za rady a čas, který mi věnoval.

Anotace

Práce se věnuje nalezení a následnému zobrazení nejkratší cesty mezi dvěma body na mapě. Teoretická část objasňuje základní pojmy z teorie grafů a datových struktur použitých při tvorbě aplikace.

Klíčová slova

Dijkstrův, Floydův, Bellmanův-Fordův, algoritmus, teorie grafů, graf, cesty v grafu

Title

Finding shortest path on the map

Annotation

This thesis was created to help find and view shortest possible path between two points on the map. The theoretic part explains basic concepts of graph theory and data structures used for application development.

Keywords

Dijkstra, Floyd, Bellman-Ford, algorithm, graph theory, graph, paths in a graph

Obsah

Seznam zkratek

ISO	International Organization for Standardisation
PDA	Personal Digital Assistant
JVM	Java Virtual Machine
GNU GPL	GNU General Public License
GC	Garbage collector
SDK	Software development kit
XML	Extensible Markup Language
DDA	Digital Differential Analyzer
J2EE	Java Platform, Enterprise Edition
J2SE	Java Platform, Standard Edition
J2ME	Java Platform, Micro Edition
UML	Unified Modeling Language

Seznam obrázků

Seznam tabulek

1 Úvod

Hlavní cíl této bakalářské práce je vytvoření takové aplikace pro mobilní zařízení, která dokáže najít nejkratší cestu na uživatelem vybrané mapě. Aby toto bylo možné, je nutné vytvořit hranově ohodnocený graf nad vybranou mapou.

Vytvoření tohoto grafu probíhá v samostatné desktopové aplikaci. Ta umožňuje uložení vytvořeného grafu do souboru ve formátu XML, který lze následně načíst do mobilní aplikace. Dále bylo nutné zvolit takovou datovou strukturu pro uložení grafu, která umožňuje efektivní implementaci Dijkstrova algoritmu.

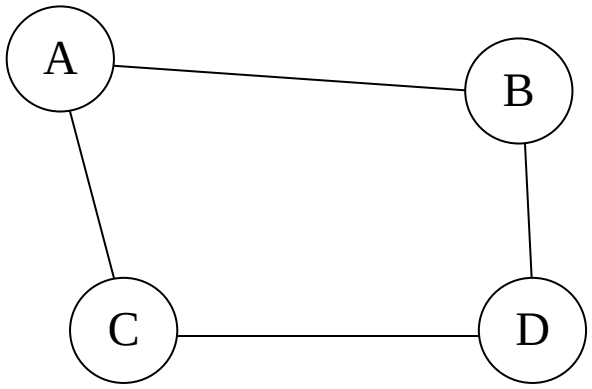
Praktické využití této aplikace zvyšuje to, že o tvorbu grafu se stará přímo uživatel. Díky tomu může graf obsáhnout i cesty a zkratky, které nejsou v různých aplikacích na hledání nejkratší cesty obsažené, ale mohou významně zkrátit délku cesty. Další výhodou je využití platformy Windows Mobile 6.5 a tím i zacílení aplikace na mobilní zařízení, na kterém aplikace pracuje bez nutnosti přístupu na internet.

2 Základní pojmy z teorie grafů

Dvě následující kapitoly jsou věnovány vysvětlení některých základních pojmů a algoritmů z teorie grafů, které budou používány v ostatních kapitolách bakalářské práce. Nejprve je zde popsáno, co to graf je a k čemu se využívá, a následně jsou vysvětleny některé podstatné vlastnosti grafu. Vysvětlení těchto pojmů vychází z použité literatury (1), (2).

V této práci se nejedná o grafické znázornění funkční závislosti ani o graf používaný ve statistice, ale o strukturu v diskrétní matematice. Pomocí této struktury lze vyjádřit model existující sítě, ať už počítačové, dopravní nebo jiné. Díky tomu graf může pomoci při řešení problému z reálného světa jako je hledání nejkratší cesty nebo plánování procesů. Definice obyčejného grafu je následující:

„Graf (rozšířeně obyčejný či jednoduchý neorientovaný graf) je uspořádaná dvojice $G=(V,E)$, kde V je množina vrcholů a E je množina hran – množina vybraných dvouprvkových podmnožin množiny vrcholů.“ (1)



Obrázek 1 – Vyjádření neorientovaného grafu nakreslením

Graf lze vyjádřit různými způsoby. Na obrázku 1 je znázorněna velmi častá reprezentace grafu pomocí jeho nakreslení. Graf lze dále vyjádřit pomocí matice sousednosti anebo výčtovou prezentací grafu. Matice sousednosti

$$A = (a_{i,j})_{i,j=1}^{n} \qquad 1$$

grafu $G = (V, H, p)$ je čtvercová matice n-tého řádu, kde $n = |V|$, ve které platí $a_{i,j} = 0$ když neexistuje hrana (v_i, v_j) a $a_{i,j} = 1$ když hrana (v_i, v_j) existuje.

2.1 Základní pojmy

- *Vrchol*, nebo také uzel, je prvek z množiny *V(G).*
- *Hrana* je prvek z množiny *V(H).*
- *Incidence* přiřazuje každé hraně grafu neuspořádanou dvojici vrcholů. Daný vrchol je krajním vrcholem příslušné hrany.
- *Ohodnocení hrany* je číslo, reprezentující například vzdálenost, přiřazené ke hraně. Značí se zpravidla $o(h)$.
- *Stupeň vrcholu* je v neorientovaném grafu počet hran, které vychází z vrcholu *v* v grafu *G*.
- *Podgraf* grafu *G* je takový graf *H*, který obsahuje libovolnou podmnožinu vrcholů *V(G)* a takovou podmnožinu hran z grafu G, ve které jsou oba vrcholy všech hran v množině *V(H).*
- *Isomorfismus*, 2 grafy jsou izomorfní, když se liší pouze nakreslením vrcholů a hran. Incidence zůstává stejná. Grafy tedy mají stejný počet vrcholů a hran a stejný počet vrcholů stejného stupně.
- *Komplement* (doplněk) obyčejného grafu $G_1 = (V_1, H_1, p_1)$ je takový graf $G_2 = (V_2, H_2, p_2)$, kde $V_2 = V_1 = V_k$ a $H_2 = H_k \setminus H_1$ kde H_k a V_k jsou množiny hran a vrcholů kompletního grafu $G_k = (V_k, H_k, p_k)$. (2)
- *Sled* je posloupnost vrcholů a hran, která začíná a končí v daných vrcholech. Mezi po sobě jdoucími vrcholy musí existovat hrana.
- *Tah* je takový sled, ve kterém není žádná hrana obsažena více než jednou.
- *Cesta* je takový sled (nebo tah), ve kterém se neopakuje žádný vrchol. V orientovaném grafu se cesta může označovat jako *dráha*.
- *Souvislý graf* je takový graf $G = (V, H)$, ve kterém existuje pro každé jeho dva vrcholy $u \in V, v \in V$ cesta z u do v.
- *Řídký graf* je takový graf, ve kterém je mohutnost množiny hran H výrazně menší než $|V|^2$.

2.2 Klasifikace grafů

Grafy lze dále rozlišit například podle orientace hran, existence kružnice v grafu či existence ohodnocení hran. Jelikož se tato práce zabývá vyhledáváním nejkratší cesty na neorientovaném, ohodnoceném obyčejném grafu, tak jsou zde uvedena pouze některá kritéria, podle kterých lze grafy třídit.

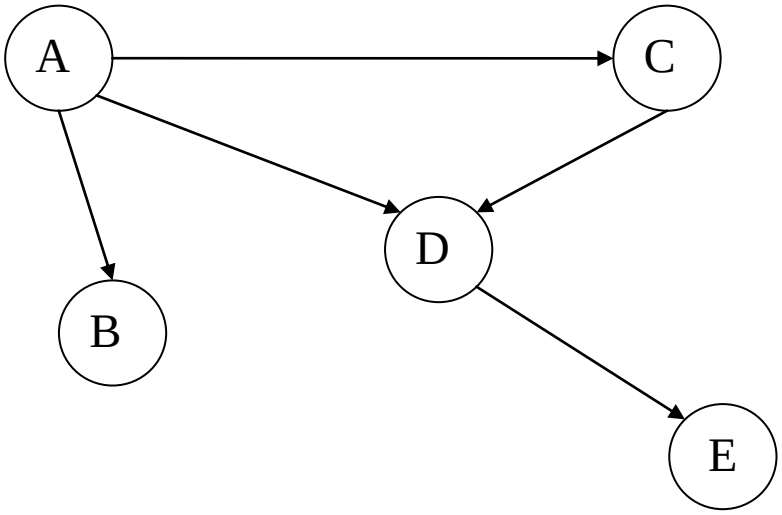
Grafy lze dělit do několika následujících základních skupin:

- *prázdný graf* je graf $G = (V, H)$, kde $V = \emptyset, H = \emptyset$,
- *diskrétní graf* je graf $G = (V, H)$, kde $H = \emptyset$,
- *triviální graf* je graf $G = (V, H)$, kde $H = \emptyset$ a V je jednoprvková množina,
- *kompletní graf* je takový graf, ve kterém je každý vrchol spojený hranou se všemi ostatními vrcholy,
- *multigraf* je takový graf, ve kterém připustíme existenci rovnoběžných (násobných) hran a smyček,
- *prosté grafy* jsou takové grafy, ve kterých připustíme existenci smyček, ale nesmí obsahovat násobné hrany,
- *obyčejné grafy* jsou grafy, které neobsahují smyčky ani násobné hrany.

2.2.1 Neorientované a orientované grafy

Neorientovaný graf je takový graf, kde jsou vrcholy spojeny neorientovanými hranami. U neorientovaných hran jsou oba vrcholy hrany ekvivalentní, není tedy jednoznačně definován výchozí a koncový vrchol a tudíž u grafu $G = (V, H)$, kde $V = \{a, b\}$ a H= $\{\{a, b\}\}$ platí $\{a, b\} \equiv \{b, a\}$. Z toho vyplývá, že při reprezentaci neorientovaného grafu maticí sousednosti platí pro prvky matice sousednosti A $a_{i,j} = a_{j,i}$ a tudíž je matice A symetrická podle hlavní diagonály.

Orientovaný graf (nebo též digraf) $D = (V, H, p)$ je uspořádaná dvojice množin V, H a zobrazení p množiny H→ $V \times V$. Oproti neorientovaným grafům je množina H tvořena uspořádanými dvojicemi $[u, v]$ prvků množiny V, kde u je výchozí vrchol hrany a v je koncový vrchol hrany. Pro dvojici $[u, v]$ musí být splněno, že $u \neq v$ a každá dvojice $[u, v]$ je v množině H obsažena maximálně jednou.



Obrázek 2 – Hranově neohodnocený orientovaný graf

Na obrázku 2 je zobrazeno grafické znázornění orientovaného neohodnoceného grafu. Oproti grafickému zobrazení neorientovaného grafu se liší v tom, že každá hrana je orientovaná, což nám umožní určit výchozí a koncový vrchol hrany. Orientace je na zobrazení nakreslením vyjádřena šipkou. Digraf můžeme považovat za základní strukturou pro reprezentaci sítě $S = (G, z, s, w)$, kde G je digraf, $w: E(G) \rightarrow \mathbb{R}^+$ je ohodnocení hran a $z \in V(G)$, $s \in V(G)$ jsou počátečním (zdroj) a koncovými (strok) vrcholy (1).

2.2.2 Ohodnocené grafy

U ohodnoceného grafu $G = (V, H, p)$ je ke každé hraně z množiny H přiřazeno ohodnocení. Ohodnocení by mělo být číslo, které může znamenat například délku, spolehlivost nebo kapacitu hrany. Ohodnocení může být jak kladné, tak i záporné.

Při reprezentaci ohodnoceného grafu můžeme použít matici vzdáleností. Ta vznikne tak, že v matici sousednosti

$$\mathrm{A} = \left(a_{i,j}\right)_{i,j=1}^{n} \qquad 2$$

grafu $G = (V, H, p)$ při existenci hrany (v_i, v_j) položíme prvek matice A $a_{i,j} = o(v_i, v_j)$.

2.2.3 Cyklické a acyklické grafy

Podle definice je graf cyklický v případě, že jeho součástí je podgraf, který tvoří kružnici. Kružnice se může vyskytovat jak v orientovaných, tak i neorientovaných grafech, je to uzavřená posloupnost propojených vrcholů. Acyklický graf je naopak takový graf, který neobsahuje žádný cyklus. Typickou ukázkou acyklického grafu je strom.

V některých grafech se mohou vyskytovat *záporné cykly*, to jsou orientované cykly, u kterých je záporný součet ohodnocení jejich hran. Při procházení těchto cyklů se můžeme,-například při hledání minimální cesty, zacyklit a cenu donekonečna snižovat. To

je samozřejmě špatně a je třeba se takové situaci vyhnout. Nejlépe tak, že před hledáním nejkratší cesty prohledáme graf a zjistíme, jestli obsahuje záporný cyklus.

3 Algoritmy pro hledání nejkratší cesty

Grafových algoritmů je velká řada, řeší různé problémy a mají tisíce různých využití v praxi. Tato kapitola se věnuje popisu několika rozšířených algoritmů, které slouží k vyhledání nejkratší, nebo též minimální, cesty na souvislém a hranově ohodnoceném[1] grafu. Tento druh algoritmů se v současné době velmi často využívá, nejčastěji v situaci, kdy hrany grafu reprezentují různé dopravní spoje (silniční, železniční), které spojují různé fyzické oblasti, a ohodnocení hrany bývá její fyzická délka. Další možnosti uplatnění jsou například při směrování v počítačové síti nebo při výpočtu dráhy robotů.

Díky tomu, že hraně lze přiřadit libovolné ohodnocení, nemusí být cesta za minimální považována jen z hlediska vzdálenosti. Může být optimální kupříkladu z hlediska ceny (spotřeba paliva, mýto) nebo z hlediska času. Nejkratší cesty se tak mohou významně lišit právě podle zvoleného kritéria.

Vzhledem k množství různých kritérií může být ohodnocení v některých případech záporné, ale ne všechny algoritmy dokážou korektně pracovat se záporně ohodnocenými hranami. Dalším problémem, který může vzniknout u orientovaného grafu, je to, že se v záporně ohodnoceném grafu může vyskytnout záporný cyklus. Za každý oběh záporného cyklu klesá cena za cestu do jakéhokoli vrcholu, který je dosažitelný záporným cyklem. Neexistuje tedy nejkratší cesta, protože cestu lze dalším oběhem učinit ještě levnější.

Následující algoritmy se dají ve zkratce rozdělit do dvou kategorií. Na ty, které hledají nejkratší cestu z vrcholu u do vrcholu v, a na ty, které určí nejkratší cestu mezi vrcholem u a všemi ostatními vrcholy a naopak.

Když bude M množina všech cest $m(u,v)$ z vrcholu u do v v grafu $G=(V,H,p)$ pak nejkratší cesta v grafu $G=(V,H,p)$ je taková cesta $m^*(u,v)\in M$, pro kterou platí:

$$\sum_{h\in m*(u,v)} o(h) = \min_{m(u,v)\in M}\left\{\sum_{h\in m(u,v)} o(h)\right\} \quad (2) \qquad 3$$

Nutným předpokladem pro nalezení nejkratší cesty $m^*(u,v)$ na grafu je existence alespoň jedné cesty $m(u,v)$.

3.1 Dijsktrův algoritmus

Pomocí Dijkstrova algoritmu můžeme najít optimální cestu pouze na grafech, jejichž hrany jsou ohodnoceny nezáporně. Tento algoritmus je možno označit jako speciální případ prohledávání grafu; vrcholy se mohou klasifikovat jako nedosažené, dosažené a probrané[2].

[1] U neohodnoceného grafu lze za délku minimální cesty považovat počet hran na nejkratší cestě.
[2] Jsou to takové vrcholy, do kterých algoritmus již vstoupil a probral všechny možnosti dalšího postupu.

Při procházení grafu je k vrcholům přiřazena proměnná $E(v_i)$, která značí délku dočasně nejkratší cesty z počátečního vrcholu u do vrcholu v_i. Protože při výpočtu je dosažený vrchol v_{i+1} probrán až tehdy, když je $E(v_i)$ rovno délce nejkratší cesty, je hodnota proměnné $E(v_i)$ definitivní a již se v probraném vrcholu dále nemění. Hodnoty $E(v)$ se u vrcholů dosažených samozřejmě měnit mohou. Vrcholy se v Dijkstrově algoritmu řadí do prioritní fronty, kde prioritou je velikost proměnné $E(v)$, přičemž nejvyšší prioritu mají vrcholy s nejmenší hodnotou $E(v)$.

Algoritmus se skládá z následujících kroků:

1. V grafu $G = (V, H)$ vybereme počáteční vrchol $u = v_0$ a koncový vrchol $v = v_n$ cesty, kde $u \in V, v \in V$.

2. Všem vrcholů přiřadíme počáteční ohodnocení, pro vrchol u bude $E(u) = 0$, pro vrcholy $v_i \in V, i = 1, 2, \dots, n$ bude $E(v_i) = \infty$.

3. V grafu hledáme přilehlou dvojici $v_i, v_j \in V$, kde platí

$$E(v_j) > E(v_i) + o(v_i, v_j). \quad (4)$$

Pokud takováto dvojice existuje tak

$$E(v_j) = E(v_i) + o(v_i, v_j). \quad (5)$$

Jestliže takováto dvojice neexistuje tak jsou všechny vrcholy V grafu G probrané a hodnota $E(v)$ udává délku nejkratší cesty.

Avšak pomocí těchto kroků algoritmu pouze zjistíme ohodnocení nejkratší cesty $m = (u, v)$. Abychom zjistili vrcholy a hrany, přes které cesta vede, musíme provést následující kroky:

1. Rekonstrukce cesty probíhá od koncového vrcholu v, tento vrchol zároveň zařadíme do množiny vrcholů U. Množina U obsahuje vrcholy, které byly již zařazeny do cesty. Zároveň položíme $i = 0$.

    a. Určíme množinu sousedů koncového vrcholu v, z množiny sousedů vyloučíme vrcholy, které jsou obsaženy v množině U.

    b. Z množiny sousedů vybereme takový vrchol v_i, pro který platí

$$E(v_i) - E(v_{i+1}) = o(v_{i+1}, v_i). \quad (6)$$

2. Pokud $v_{i+1} \neq u$ vložíme do množiny U vrchol v_{i+1}, proměnnou i inkrementujeme o jedna a vrátíme se zpět ke kroku 1a. Jestliže platí $v_{i+1} = u$ tak U obsahuje všechny vrcholy nejkratší cesty.

Podíváme-li se na algoritmus uvedený výše, je zřejmé, že v případě výskytu záporné hrany by se algoritmus vrátil zpět do již probraného vrcholu. To je důvod, z kterého nelze Dijkstrův algoritmus použít na grafy se záporně ohodnocenými hranami.

Asymptotická složitost Dijkstrova algoritmu je $O(|V|^2 + |H|)$, nejlepších výsledků dosahuje při použití v řídkých grafech. Tento algoritmus je zároveň jednou z variant hladového algoritmu[3].

3.1.1 Modifikace A*

Tato modifikace Dijkstrova algoritmu přidává do algoritmu heuristický prvek. Vrcholy se řadí do prioritní fronty nikoliv podle velikosti proměnné $E(v)$, ale podle hodnoty funkce $f(x)$. Ta v sobě zahrnuje jak heuristickou funkci $h(x)$, která odhaduje cestu k finálnímu vrcholu, tak i funkci $g(x)$, která reprezentuje vzdálenost mezi počátečním vrcholem u a vrcholem v_i.

3.1.2 Johnsonův algoritmus

Tato modifikace dokáže najít optimální cestu i na záporně ohodnocených grafech, ale pouze v případě, že graf neobsahuje záporný cyklus. Detekce záporného cyklu se provádí použitím Bellman-Fordova algoritmu. Jeho asymptotická složitost je pro řídké grafy $O(|V| \log_2 |V| + H|)$.

Při použití Johnsonova algoritmu se nejdříve provede detekce záporného cyklu. Pokud graf takový cyklus neobsahuje, přehodnotí se všechny hrany takovým způsobem, aby žádná z nich nebyla záporně ohodnocena. Na přehodnocený graf se pak použije Dijsktrův algoritmus a následně se na délku nalezené cesty aplikuje zpětná transformace.

3.2 Bellman-Fordův algoritmus

Bellman-Fordův algoritmus se stejně jako Dijsktrův dá označit za zvláštní formu prohledávání grafu. Další podobnost s Dijkstrovým algoritmem je používání fronty pro dosažené vrcholy. Avšak není zde použita prioritní fronta, ale fronta typu FIFO. Tato, na první pohled nepodstatná změna, umožňuje využití algoritmu v grafu, který má záporně ohodnocené hrany. Složitost tohoto algoritmu je $O(|V||H|)$.

Cena za tuto vlastnost může být někdy poměrně vysoká a to díky většímu množství výpočtů. To je způsobeno tím, že hodnota $E(v_i)$ u probraného vrcholu nemusí značit délku nejkratší možné cesty mezi vrcholy u a v_i, ale může se snižovat zároveň s tím, jak se vybírají z fronty další vrcholy. Tím pádem je při každé změně $E(v_i)$ nutné přepočítat všechny hodnoty, které vycházejí z $E(v_i)$.

Při použití tohoto algoritmu v orientovaném grafu, kde se vyskytuje záporný cyklus, existuje několik možností jak se s tímto cyklem vypořádat. Nejjednodušší varianta Bellman-Fordova algoritmu se zacyklí a nikdy se nezastaví. Naopak pokročilejší verze

[3] Hladový algoritmus vybere v každém kroku lokální minimum, protože existuje šance, že toto lokální minimum bude zároveň globálním minimem.

algoritmu dokážou záporný cyklus nejen zjistit, ale i přesně určit. Proto je Bellman-Fordův algoritmus využíván nejen pro vyhledání nejkratší cesty, ale i pro detekci záporného cyklu.

Samotný algoritmus je následovný:

1. V grafu $G = (V, H)$ vybereme počáteční vrchol $u = v_0$ a koncový vrchol $v = v_n$ cesty, kde $u \in V, v \in V$.

2. Inicializujeme proměnou $E(u) = 0$, pro ostatní vrcholy $v_i \in V, i = 1, 2, \dots, n$ nastavíme proměnnou $E(v_i) = \infty$.

3. Položíme $i = 0$ a projdeme všechny hrany h z množiny incidentních hran vrcholu v_i. Pokud u hrany h platí $E(v_j) > E(v_i) + o(h)$ tak se $E(v_j) = E(v_i) + o(h)$. Po projití všech incidentních hran inkrementujeme $i = i + 1$ tak dlouho, dokud neplatí $i = |V|$. V tom případě jsme prošli všechny vrcholy grafu a hodnota $E(v)$ je rovna délce nejkratší cesty z vrcholu u do vrcholu v.

3.3 Floyd-Warshallův algoritmus

Tento algoritmus určí nejkratší cestu mezi vrcholem v a všemi ostatními dostupnými vrcholy; tyto vzdálenosti jsou reprezentovány distanční maticí. Floyd-Warshallův algoritmus má charakter postupného zpřesňování. Vylepšuje odhad nejkratších cest mezi vrcholy grafu tak dlouho, dokud není odhad roven skutečně nejkratším cestám.

Protože Floyd-Warshallův algoritmus využívá reprezentaci grafu pomocí matice, není vzhledem k velké paměťové náročnosti příliš vhodný pro použití v řídkých grafech. Lze jej použít i v případě, že graf má záporně ohodnocené hrany. Časová asymptotická složitost je $O(|V|^3)$.

Před použitím Floyd-Warshallova algoritmu musíme nejdříve vytvořit matici přímých vzdáleností

$$D = \left(d_{ij}\right)_{i,j=1}^{n} \qquad 7$$

s rozměrem $|V| \times |V|$. Prvky této matice nabývají hodnot dle těchto kritérií:

- $d_{ij} = 0$, když $i = j$.
- Nebo $d_{ij} = o(h)$, pokud existuje hrana h, která je incidentní s vrcholy v_i a v_j.
- Nebo $d_{ij} = \infty$, pokud neexistuje hrana h, která je incidentní s vrcholy v_i a v_j.

Dále postupujeme dle vývojového diagramu uvedeného v příloze A.

3.4 Další metody

3.4.1 Algoritmus měnící měřítko

Tento algoritmus lze použít pouze pro grafy, které mají hrany ohodnoceny celým číslem. U tohoto algoritmu je ohodnocení reprezentováno pomocí bitů, kdy při prvním průchodu je ohodnocení hran jednobitové (nejvyšší bit) a pro další iterace algoritmus přidává k ohodnocení další bit tak dlouho, dokud se nedostane ke všem bitům. Díky tomu, že známe výsledky z předchozí iterace je při každé iteraci řešený pouze zjednodušený problém.

3.4.2 Metoda Monte Carlo

Tato metoda, jejímž principem je realizace náhodných procesů, také dokáže nalézt nejkratší cestu v grafu. Ale už jenom z principu této metody je jasné, že pro nalezení nejkratší cesty by muselo proběhnout velké množství náhodných pokusů. To zapříčiní poměrně velkou výpočetní náročnost, a proto je vhodnější použít některý z grafových algoritmů.

4 Datové struktury

Tato kapitola se kromě vysvětlení základních pojmů zabývá popisem možných implementací tří datových struktur – grafu, tabulky a prioritní fronty. Informace vycházejí z použité literatury (3) (4).

4.1 Základní pojmy

Datová struktura je způsob, jakým jsou data, obvykle v paměti, organizována a ukládána. Zároveň umožňuje efektivní přístup a modifikaci uložených dat, rovněž vymezuje objekt zkoumání; je to abstrakce reality. Datová struktura je navržena tak, aby byla efektivní v určité situaci; co může být efektivní v jednom případě, nemusí být optimální v druhém.

Abstraktní datový typ

Abstraktní datový typ (dále jen ADT) je matematická struktura, která je složena z jedné nebo více tříd (domén) a operací nad prvky tříd. Má za úkol zapouzdřit vnitřní strukturu a sjednotit vstupy a výstupy operací. ADT je definován nezávisle na implementaci a platformě, nejsou v něm tedy implementovány jednotlivé operace, jejichž realizace se může lišit.

Abstraktní datová struktura

Abstraktní datová struktura (dále jen ADS) je konkrétní realizace ADT. Implementují se v ní jednotlivé operace tak, aby vstupy i výstupy odpovídaly ADT. Díky tomu ADS poskytuje znovupoužitelnost, protože není závislá na uživatelských datech. Jelikož jsou jednotlivé platformy odlišné a mají určité technické omezení, nelze všechny operace implementovat zcela korektně. Pokud je tedy uvedeno, že daná datová struktura může obsáhnout neomezený počet prvků, narazí se při implementaci na technické limity platformy a počet prvků je omezený.

Klasifikace datových struktur

Datové struktury lze klasifikovat z pohledu logické a fyzické úrovně. Klasifikační kritéria u logické úrovně jsou následující:

- Charakter přístupu:
  - Sekvenční přístup – to znamená, že prvky jsou vybírány na základě svého pořadí.
  - Přímý přístup – přímý přístup k vybranému prvku podle klíče.
- Organizační struktura:
  - Lineární – existuje specifická relace uspořádání prvků, prvek má zpravidla bezprostředního následovníka.
  - Nelineární – specifická relace uspořádání prvků neexistuje a prvek zpravidla nemá bezprostředního následovníka.

U fyzické úrovně se datové struktury klasifikují takto:

- Složení struktury:
  - Homogenní – všechny prvky ve struktuře jsou stejného typu.
  - Nehomogenní – prvky v datové struktuře mohou být různého typu.
- Vymezení pracovního paměťového prostoru
  - Statický – alokace paměti proběhne při spuštění.
  - Dynamický – alokace a dealokace paměti probíhá za běhu.
- Výstavba datových struktur
  - Elementární datové typy – datová struktura se skládá z jednoduchých datových typů.
  - Strukturované datové typy – datová struktura se skládá z datových typů, které v sobě zapouzdřují více datových typů.

Datové struktury lze dělit také dle perzistence (tj. jakým způsobem zachovávají své starší verze).

4.2 Datová struktura – Prioritní fronta

Prioritní fronta je ADT, který umožňuje rychlé zpřístupnění prvku s nejvyšší prioritou. Prioritu reprezentuje klíč, který nabývá různých, většinou číselných, hodnot. Např. zásobník a fronta je speciální případ prioritní fronty, kde prioritu reprezentuje čas vložení prvku do struktury. Tato datová struktura má za úkol poskytovat informace uložené v ní dle hodnoty klíče tak, že nejdříve zpřístupní prvky s maximální prioritou. Tabulka 1 (3) obsahuje základní operace ADT prioritní fronta.

Tabulka 1 – ADT Prioritní fronta

Třída prvků s prioritou
Vytvoř
Zruš
JePrázdná (↑ Boolean)
Mohutnost (↑ Počet prvků)
Vlož (↓ Prvek)
OdeberMax (↑ Prvek)
ZpřístupniMax (↑ Prvek)
Třída konečných prioritních fronta
Sjednocení(↓ PFrontaA, ↓ PFrontaB, ↑ PFrontaC)

Některé implementace umožňují také odebrání libovolného prvku fronty nezávisle na jeho prioritě a zvýšení priority prvku fronty.

Prioritní frontu lze implementovat několika způsoby, na zvoleném způsobu implementace velmi záleží složitost. Proto je vhodné vybrat takovou implementaci prioritní fronty, která je nejlépe uzpůsobena ke zvolenému účelu.

4.2.1 Prioritní fronta nad lineární seznam

Prioritní fronta nad utříděným lineárním seznamem je vhodná v situaci, kdy velmi často prvky odebíráme, ale vkládání provádíme výjimečně. Časová složitost je v tomto případě při vkládání $O(n)$, kde n je počet prvků ve frontě, při odebírání je složitost $O(1)$. Při použití neutříděného lineárního seznamu dostaneme přesný opak, jelikož prvky se v něm neseřazují dle priority při vkládání, ale prvek s nejvyšší prioritou je hledán při operaci Odeber (Zpřístupni). Jelikož je složitost při vkládání $O(1)$ a při odebírání $O(n)$ je vhodné použít tuto frontu v případech, kdy je velmi časté vkládání prvků, ale odebírání je výjimečné.

Také lze implementovat prioritní frontu, která je postavena nad dvěma seznamy, z nichž jeden je neutříděný a druhý utříděný. Utříděný seznam má omezenou velikost na M prvků a jsou do něj vkládány jen prvky, jejichž priorita je vyšší než stanovená mez. V neutříděném seznamu jsou naopak prvky, které nemají takovou prioritu, aby se dostaly do utříděného seznamu. Díky tomu je maximální složitost vkládání i odebírání $O\left(n^{1/2}\right)$, ale zpravidla lze dosáhnout složitosti $O(1)$.

4.2.2 Prioritní fronta a halda

Vlastnost haldy je ta, že klíč v každém vrcholu je větší, než priority jeho následovníků (pokud je má). Maximální složitost operací Odeber a Vlož je $O(\log_2 n)$. Použitím Fibonacciho haldy lze dosáhnout amortizované[4] složitosti $O(1)$ při operacích Vlož a Zpřístupni a $O(\log n)$ při operaci Odeber.

4.3 Datová struktura – Tabulka

Datová struktura tabulka je lineárně uspořádaná množina, kde je uspořádání dáno hodnotami klíčů jednotlivých prvků. Tato množina nemusí obsahovat prvky se všemi klíči.

Tabulka 2 – ADT Tabulka

Třída prvků s klíči
Vytvoř
Zruš
JePrázdná(↑ Boolean)
Mohutnost (↑ Počet prvků)
Prohlídka (↓ TypProhlídky, ↓ Akce)
Vlož (↓ Prvek)
Odeber (↓ Klíč, ↑ Prvek)
Najdi (↓ Klíč, ↑ Prvek)
Třída konečných tabulek
Sjednocení (↓ TabulkaA, ↓ TabulkaB, ↑ TabulkaC)

[4] Teoretická rychlost daných operací. Při složitosti operace $O(f(n))$ nikdy nepřesáhne sekvence n operací přes $O(n \times f(n))$.

ADT tabulka má velké množství možných implementací, lze ji implementovat na utříděném či neutříděném poli nebo seznamu, binárním vyhledávacím stromu nebo kosočtvercové vyhledávací síti. Operace datové struktury tabulka jsou uvedeny v tabulce 2 (3).

4.4 Datová struktura – Graf

Datová struktura graf je reprezentací struktury graf z diskrétní matematiky. Je to heterogenní struktura, která pracuje se dvěma třídami prvků, které představují vrcholy a hrany grafu. Tabulka 3 (3) zobrazuje operace nad datovou strukturou graf.

Tabulka 3 – Super ADT Graf

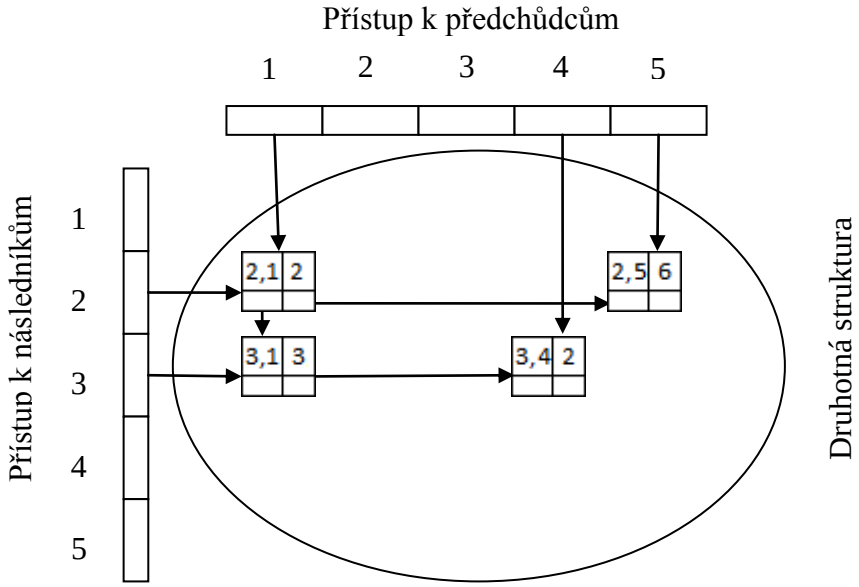
Třídy prvků
Vytvoř
Zruš
JePrázdný (↑ Boolean)
Mohutnost (↑ Počet prvků)
Prohlídka (↓ Typ, ↓ Počátek, ↓ Akce)
VložVrchol (↓ Vrchol)
VložHranu (↓ Hrana)
OdeberVrchol (↓ Klíč, ↑ Vrchol)
OdeberHranu (↓ Klíč, ↑ Hrana)
NajdiVrchol (↓ Klíč, ↑ Vrchol)
NajdiHranu (↓ Klíč, ↑ Hrana)
ZpřístupniNásledníky (↓ Koho, ↑ Prvky)
ZpřístupniPředchůdce (↓ Koho, ↑ Prvky)
ZpřístupniIncidenčníPrvky (↓ Koho, ↑ Prvky)
DefinujBránu (↓ Prvek)
AnulujBránu (↓ Prvek)
ZpřístupniBrány (↑ Prvek)
Třída konečných grafů
Sjednocení (↓ GrafA, ↓ GrafB, ↑ GrafC)

Super ADT graf má velký počet možných implementací. Ty lze kategorizovat podle typu přístupu, tedy zda přistupujeme k vrcholům nebo k hranám. Jak vrcholový, tak i hranový přístup lze dále dělit na statický a dynamický.

Vrcholově orientovaný přístup

U vrcholového přístupu jsou vyhledávácí operace zaměřeny na zpřístupnění vrcholů, s nimiž jsou spojeny informace o incidentních hranách a sousedních vrcholech. Tento přístup se dále dělí na:

- Statický – vhodné pro grafy, u kterých není třeba provádět operace VložVrchol a OdeberVrchol. U tohoto typu přístupu mají totiž operace VložVrchol a OdeberVrchol složitost větší než $O(n)$ nebo je nelze uskutečnit vůbec.
- Dynamický – Složitost operací VložVrchol a OdeberVrchol je menší než $O(n)$.



Obrázek 3 – Křížová reprezentace grafu

Pro vrcholově orientovaný přístup lze datovou strukturu graf realizovat pomocí křížové reprezentace, zobrazené na obrázku 3, anebo hvězdou. Křížové reprezentace umožňují současný přístup k následníkům i k předchůdcům díky použití dvou prvotních struktur, ty obsahují informace o vrcholech, a jedné druhotné struktury, ta obsahuje informace o incidentních hranách. Pomocí jedné prvotní struktury se přistupuje k následníkům a pomocí druhé k předchůdcům, druhotná struktura pak obsahuje informace jak pro přístup k následníkům i k předchůdcům. Hvězda je tvořena jednou prvotní strukturou a jednou druhotnou strukturou, přičemž prvotní i druhotnou strukturu lze realizovat pomocí pole nebo tabulky.

Hvězda může být realizována například jako pole – pole, kde jak prvotní, tak i druhotnou strukturu tvoří pole. To umožňuje velmi rychlé provedení operací NajdiVrchol, NajdiHranu, VložHranu a OdeberHranu (složitost je pouze $O(1)$), ale zpřístupnění následovníků a předchůdců má složitost $O(n)$. U realizace tabulka – tabulka, kde jsou prvotní i druhotná struktura realizována pomocí tabulky, mohou tabulky představovat např. seznam nebo strom.

Hranově orientovaný přístup

Pro hranově orientovaný přístupu je tomu naopak, vyhledávací operace jsou zaměřeny na hrany, z nichž lze získat informace o incidentních vrcholech. Také se dále dělí na statické a dynamické.

- Statické – Operace VložHranu a OdeberHranu mají složitost $O(n)$ a více, popřípadě je nelze realizovat vůbec. Tento přístup není vhodný pro dynamickou práci s grafem z pohledu hran.
- Dynamické – Složitost operací VložHranu a OdeberHranu je menší než $O(n)$.

Realizace hranově orientovaného přístupu pro hranově dynamické operace může být například pomocí tabulky hran, kde se hrana identifikuje pomocí incidentních vrcholů. Při použití pole se hrany identifikují pomocí indexu, tato reprezentace grafu je hranově statická.

5 Jazyky a nástroje pro vývoj mobilní aplikace

Pro výběr programovacího jazyka bylo stanoveno několik základních požadavků, které by měl splňovat. Jazyk by měl být objektově orientovaný, musí mít podporu pro práci s grafikou a podporu pro práci s XML. Protože je tato bakalářská práce složena z aplikace mobilní a z aplikace pro platformu PC, rozhodl jsem se, že použiji dva různé programovací jazyky. Jazyk Java pro vytvoření aplikace pro platformu PC a jazyk C# pro mobilní aplikaci.

5.1 Java

Programovací jazyk Java byl vyvinut v roce 1995 firmou Sun Microsystems. Není závislý na konkrétním hardwaru ani operačním systému, protože program napsaný v Javě je interpretován pomocí virtuálního počítače JVM. Je to objektově orientovaný, interpretovaný a robustní jazyk s podporou více vláken. V současné době je Java open-source pod licencí GNU GPL. Existuje několik verzí Javy, verze J2EE, která je určena pro vývoj serverových aplikací, J2ME pro vývoj aplikací pro mobilní zařízení a nakonec J2SE, která je určena pro vývoj aplikací pro stolní počítače.

Pro jazyk Java jsem se u desktopové aplikace rozhodl především proto, že má všechny potřebné vlastnosti a zároveň v něm mám dostatek zkušeností. Pro samotný vývoj aplikace bylo použito J2SE ve verzi 1.6.0_24.

Výhody Javy

Java je navržena tak, aby minimalizovala počet těžko odhalitelných chyb, které mohou vzniknout např. v jazyku C++. Mezi největší odlišnosti od C++ patří nemožnost používat ukazatele, odstranění vícenásobné dědičnosti (ale může implementovat libovolný počet rozhraní), silná typová kontrola, nemožnost používat příkaz *goto* a neexistence globálních proměnných.

Správu paměti obstarává Java Virtual Machine (JVM), ta automaticky uvolňuje paměť pomocí GC, který běží v samostatném vlákně s nízkou prioritou. Nedochází tedy k situacím, kdy aplikace neuvolňuje paměť korektně.

K programovacímu jazyku Java existuje velké množství volně dostupných komponent a dialogů, další výhodou, která značně urychlí vývoj aplikace je podpora serializace a deserializace objektů. Dále dostupnost velkého množství knihoven a frameworků jako je např. Swing Application Framework nebo knihovny DOM a XSTREAM pro práci s XML.

5.2 C#

Jazyk C#, vyvinutý firmou Microsoft, je součástí platformy .NET, která obsahuje řadu dalších programovacích jazyků jako F#, J#, Visual Basic .NET a další. První verze

C# 1.0 byla uvedena společně s .NET 1.0 v roce 2002, od té doby se jazyk i platforma .NET stále vyvíjí a v současnosti jsou k dispozici ve verzi 4.0.

Podobně jako u jazyku Java existuje několik verzí .NET. Microsoft .NET Framework je platforma pro PC s operačním systémem Windows, aktuální verzi 4.0 lze používat na Windows XP a novějších. Další verzí je .NET Compact Framework, která je určena pro PDA a mobilní telefony, které mají operační systém Windows Mobile, dále je k dispozici verze pro embedded zařízení .NET Micro Framework. Platforma MONO je platformově nezávislá open–source implementace .NET frameworku, je určena především pro použití na OS Mac OS X, Solaris, Linux a UNIX.

Jazyk C# je, stejně jako jazyk Java, objektově orientovaný, nepodporuje vícenásobnou dědičnost (lze implementovat pouze libovolný počet rozhraní), pro uvolňování paměti používá GC, nelze v něm používat globální proměnné a je typově bezpečnější než C++. Na rozdíl od Javy v něm lze používat ukazatele, je ale nutné uvést klíčové slovo *unsafe* při deklaraci třídy nebo metody. Jak naznačuje klíčové slovo, jedná se o netypickou věc, která by se měla používat výjimečně, např. v případech, kdy velmi záleží na rychlosti provedení kódu. Samozřejmostí je velké množství knihoven, například GDI+ pro práci s grafikou nebo knihovna pro práci s XML.

5.3 XML

Extensible Markup Language (XML) je značkovací jazyk, který se velmi často používá pro uchovávání dat, ale lze ho využít například i v různých integračních platformách. Jeho první oficiální specifikace konsorcia W3C XML 1.0 byla publikována v roce 1998, od té doby prošlo XML několika revizemi a aktuální je verze XML 1.1. Velkou výhodou je nezávislost na platformě a uživatelská definice značek. XML je čitelný člověkem a zároveň umožňuje jednoduché strojové zpracování a podporuje velké množství kódování, tím pádem podporuje i řadu jazyků a případná konverze do jiných formátů je velmi snadná.

Nevýhodou tohoto formátu je, že samotné značky mohou zabírat poměrně velký prostor v poměru k uloženým informacím, toho se ale lze vyvarovat použitím různých kompresních algoritmů. Pro definici jiných než textových dat je nutné jiné datové typy definovat pomocí schémat.

Syntaxe

```
1     <?xml version="1.0" encoding="UTF-8"?>
2     <knihovna>
3           <kniha kategorie="vzdělávací">
4                 <název jazyk="cz">JAVA</název>
5                 <autor>HEROUT, Pavel</autor>
6                 <rok>2010</rok>
7                 <cena>500</cena>
8           </kniha>
9     </knihovna>
```

Na prvním řádku je deklarace XML, která obsahuje číslo verze a použité kódování znaků, ostatní řádky obsahují elementy. Elementy jsou složeny z počátečního a koncového tagu, mohou obsahovat atributy a obsah, přičemž hodnota atributu se zapisuje do uvozovek – *jazyk="cz"*, obsah mezi počáteční a koncový tag – *<rok>2010</rok>*.

5.3.1 Knihovna XSTREAM

Koncepce této knihovny pro zpracování XML je velmi odlišná od známých knihoven DOM nebo SAX. Jejím cílem je serializace a deserializace tříd do XML. Její použití je jednoduché, je poměrně rychlá a nenáročná na paměť a především nevyžaduje žádnou úpravu objektu, který chceme uložit. Ukládá i vnořené třídy, neveřejné proměnné a třídy, přičemž u tříd nevyžaduje defaultní konstruktor. Knihovna je open source a je dostupná pod licencí BSD.

Na dalších řádcích je ukázáno použití knihovny XSTREAM. Budeme mít následující třídu *Kniha*:

```
package ukazka;
public class Kniha{
      private String nazev;
      private int cena;
}
```

Pro zápis této třídy do XML postačí několik následujících příkazů:

```
ukazka.Kniha kniha = ukazka.Kniha("Java",500);
XStream xstream = new XStream();
xstream.alias("kniha", ukazka.Kniha.class);
String xml = xstream.toXML();
```

Po zápisu řetězce *xml* do souboru vznikne XML soubor s následujícím obsahem:

```
<kniha>
      <nazev>Java</nazev>
      <cena>500</cena>
</kniha>
```

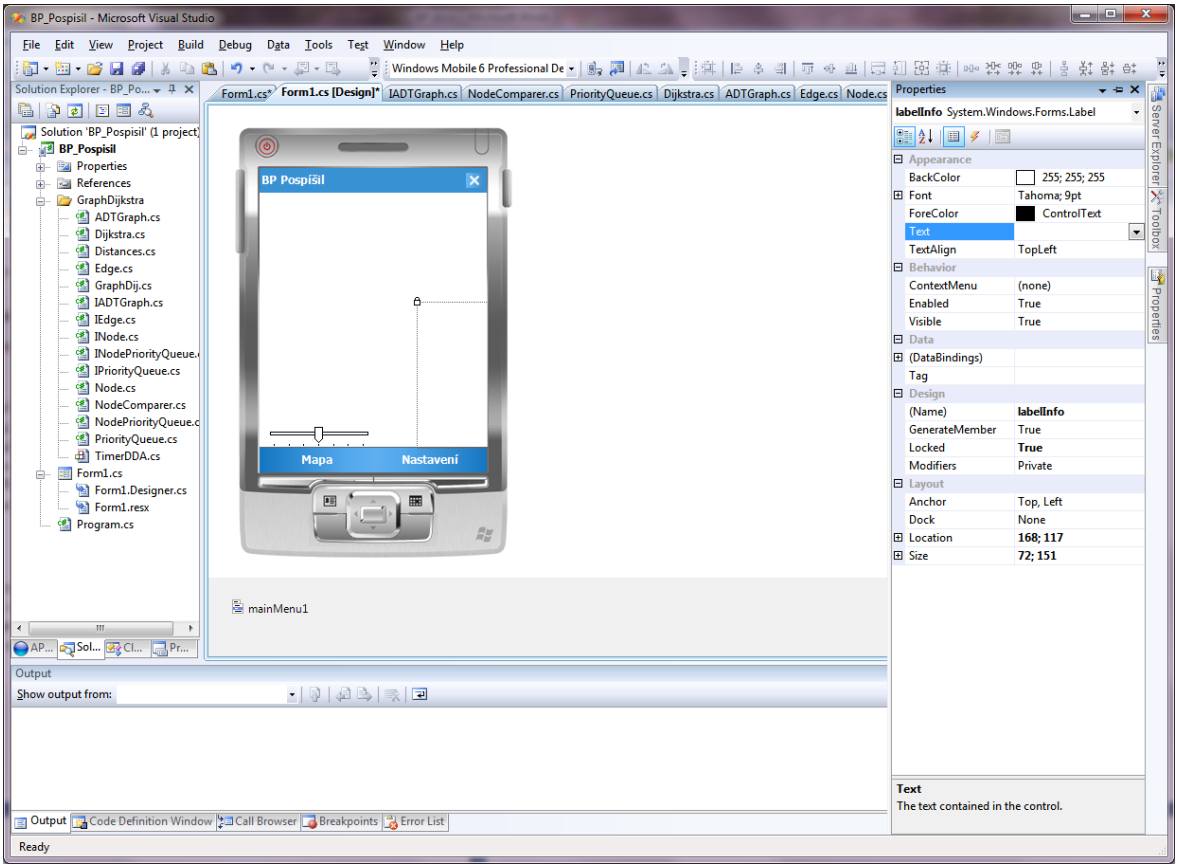
Načtení takového objektu z XML lze provést takto:

```
ukazka.Kniha kniha = (ukazka.Kniha)xstream.fromXML(xml);
```

5.4 Vývojové nástroje

5.4.1 MS Visual Studio a Windows Mobile

Pro vytvoření mobilní aplikace jsem zvolil vývojové prostředí Microsoft Visual Studio 2008 zobrazené na obrázku 4, které je pro účastníky programu MSDN Academic Alliance zdarma. Při vývoji aplikace pro Windows Mobile 6.5 je nutné nainstalovat jak SDK pro Windows Mobile 6 Professional tak i Windows Mobile 6.5 Developer Tool Kit. WM 6.5 Developer Tool Kit přidává dokumentaci, hlavičky a knihovny, ukázky kódu a nástroje do Visual Studia.



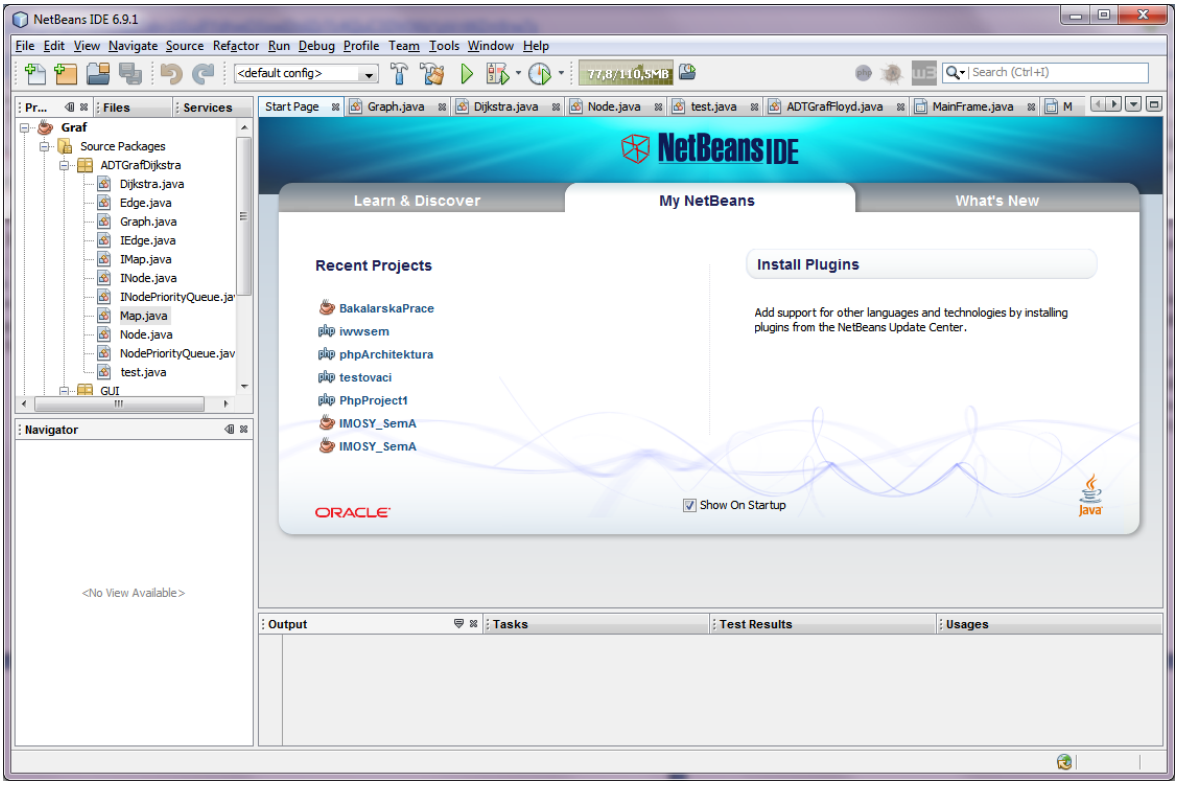
Obrázek 4 – Vývojové prostředí MS Visual Studio 2008

Důležitou částí jsou také emulátory. WM 6.5 Developer Tool Kit jich obsahuje celou řadu. K dispozici jsou emulátory jak pro WM ve verzi Professional (pro zařízení s dotykovým displejem) tak i pro verzi Standard (ovládání pomocí kláves). Emulátory se liší rozlišením emulovaného zařízení, k dispozici jsou rozlišení od QVGA až po WVGA.

Pro testování aplikace ale není nutné používat pouze emulátory. Visual Studio se dokáže propojit přímo s mobilním zařízením a testování, včetně krokování, umožňuje provádět přímo na skutečném zařízení s operačním systémem WM 6.5.

5.4.2 NetBeans

Pro vývoj desktopové aplikace jsem zvolil vývojové prostředí NetBeans IDE, které je zobrazeno na obrázku 5. Mimo prostředí NetBeans je k dispozici velké množství dalších vývojových prostředí pro jazyk Java, mezi nejznámější patří Idea, Eclipse a JDeveloper. NetBeans IDE je nástroj, který umožňuje psaní, překládání a ladění programů v různých jazycích, např. C++ a PHP, ale primárně je určeno pro jazyk Java a podporuje vývoj pro nejrozšířenější dílčí platformy Javy a to J2SE, J2EE a J2ME. Samotný NetBeans IDE je napsán v jazyku Java, díky tomu je nezávislý na operačním systému.



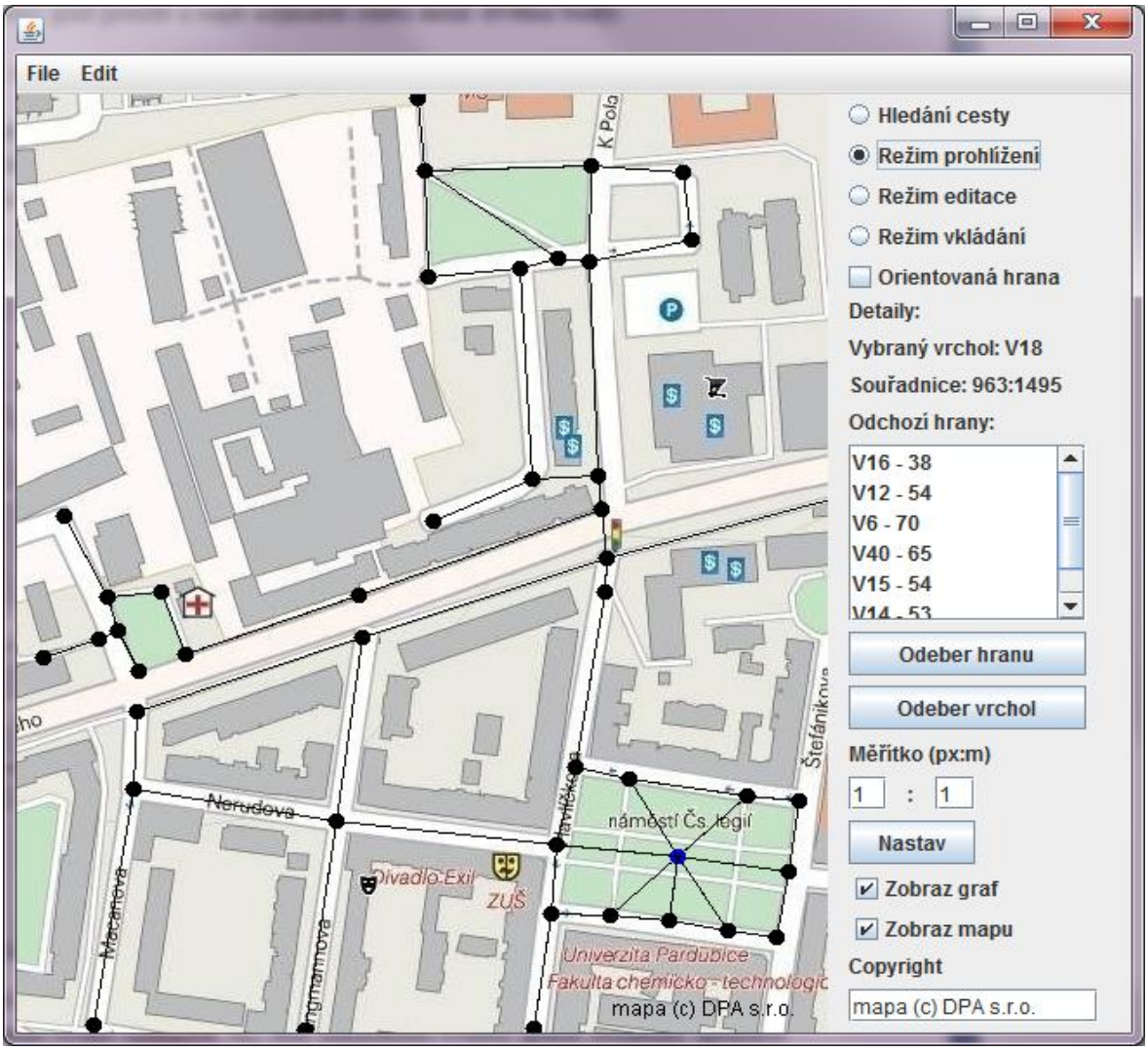
Obrázek 5 – Vývojové prostředí NetBeans 6.9.1

6 Návrh aplikace pro hledání nejkratší cesty na mapě

Cílem této práce je vytvořit aplikaci pro mobilní zařízení, která dokáže najít nejkratší cestu na mapě. Aby to bylo možné tak je nutné nejdříve vytvořit graf, ve kterém bude nejkratší cesta hledána. Mobilní zařízení, ať už jde o PDA nebo smartphone, nemá příliš velký displej a zadávání velkého množství dat je na něm nepohodlné. Proto je tato práce rozdělena do dvou různých aplikací – desktopové, která slouží pro tvorbu grafu a mobilní, jejímž úkolem je vyhledání cesty.

6.1 Desktopová aplikace

Hlavním úkolem desktopové aplikace je především vytvořit graf, který lze následně exportovat do souboru ve formátu XML. Tato aplikace je určena pro platformu PC a je vytvořena v jazyku Java.



Obrázek 6 – Tvorba grafu v desktopové aplikaci

Pro usnadnění tvorby grafu a lepší orientaci je zobrazena na pozadí aplikace mapa, podle které se graf bude tvořit, ukázka je na obrázku 6. Tvorbu grafu usnadňuje několik následujících režimů:

- **Režim vkládání** – slouží k vytvoření grafu, lze v něm přidávat vrcholy a nové hrany. U hran může uživatel rozhodnout, zda bude hrana orientovaná či nikoli. Při základním nastavení se vkládají neorientované hrany. Ohodnocení hran aplikace zjišťuje sama, uživatel ohodnocení nemůže ovlivnit.

- **Režim editace** – v tomto režimu je možné měnit polohu vrcholů a odebírat jednotlivé vrcholy a hrany.

- **Režim prohlížení** – slouží k prohlížení vytvořeného grafu, veškeré editační funkce jsou vypnuty, takže nemůže dojít k nechtěné změně.

- **Režim hledání cesty** – v tomto režimu lze hledat cestu mezi dvěma vybranými vrcholy.

Mimo režimů obsahuje aplikace další důležitá nastavení či funkce. Mezi ně patří například měřítko, které umožňuje korektní přepočet vzdálenosti z pixelů na metry. Ohodnocení hran totiž v rámci aplikace není uvedeno v metrech, ale je uvedeno v pixelech. Po zjištění nejkratší cesty se výsledek vynásobí měřítkem a teprve poté se zobrazí skutečná vzdálenost v metrech. Také je možné nastavit znění copyrightu vlastníka autorských práv mapových podkladů a zobrazit copyright na mapě.

Výsledný graf lze uložit do binárního souboru pomocí serializace. Tento soubor lze využít pouze v desktopové aplikaci, nikoli v mobilní. Pro mobilní aplikaci je určen výstupní soubor ve formátu XML. Mimo jiné lze vypnout zobrazení mapových podkladů a zobrazit pouze vytvořený graf nebo naopak vypnout zobrazení grafu a zobrazit pouze mapové podklady.

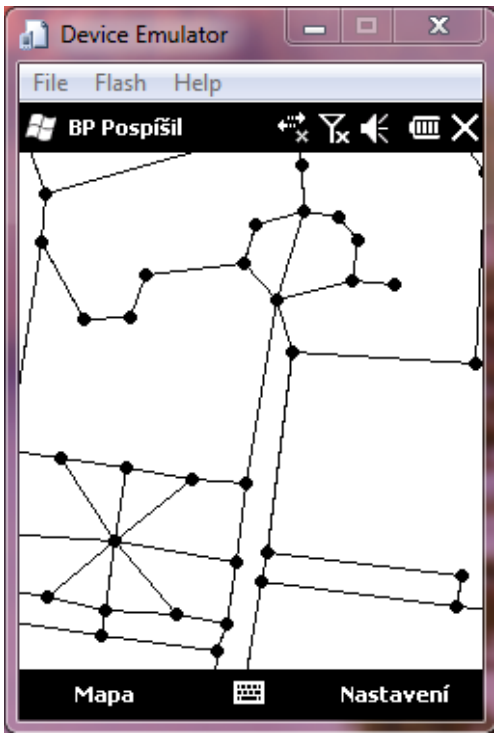
6.2 Mobilní část aplikace

V této etapě bylo nejdůležitější částí vybrat vhodnou implementaci Dijkstrova algoritmu a dále vytvořit uživatelsky příjemné grafické rozhraní. Mobilní verze .NET Compact Framework a C# je oproti standardní verzi pro PC omezena, tudíž bylo nutné se s těmito omezeními vypořádat.

Užitečným pomocníkem byl emulátor mobilního zařízení, který umožnil rychlé a pohodlné testování a odlaďování mobilní aplikace. Emulátor dokáže zastat naprostou většinu věcí totožně jako skutečné zařízení. Bezproblémová je emulace paměťové karty, která je reprezentována uživatelem vybranou složkou. Při testování se objevily pouze menší odlišnosti od skutečného zařízení, ale ty se týkaly pouze nastavení definice pravidel pro zobrazování znaků.

Obrázek 7 – Nejkratší cesta zobrazená v mobilní aplikaci

Mobilní aplikace dokáže mimo nalezení nejkratší cesty na obrázku 7 vykreslit samotný graf bez mapy (obrázek 8). Aplikace obsahuje i funkci zoom pro přiblížení nebo oddálení mapy. Další funkcí je animace nalezené cesty – cesta se vykresluje postupně z počátečního do koncového bodu. Pro postupné vykreslování cesty byl použit algoritmus DDA (Digital Differential Analyzer).

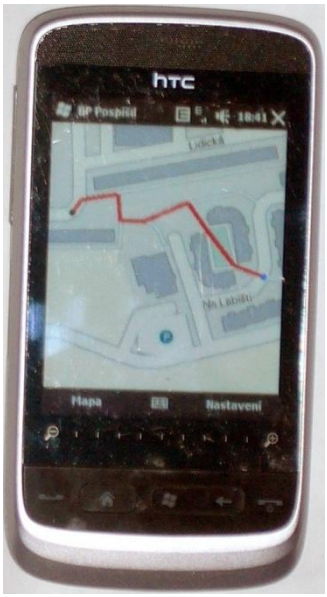


Obrázek 8 – Zobrazení samotného grafu v mobilní aplikaci

6.3 Zvolená datová struktura pro reprezentaci grafu a implementace Dijkstrova algoritmu

Jelikož jsou vyhledávací operace zaměřeny především na zpřístupnění vrcholů, byla zvolena datová struktura s vrcholově orientovaným přístupem. A protože je při vytvoření grafu nutné vložit velké množství vrcholů byl zvolen dynamický přístup. Díky tomu je složitost operace vložení vrcholu menší než $O(n)$. Samotná datová struktura byla realizována jako seznam - seznam.

Dijkstrův algoritmus byl implementován v jazyku C# pod operačním systémem WM 6.5 v mobilním zařízení HTC Touch 2 na obrázku 9, s následující HW konfigurací: Qualcomm® 7225 @ 528 MHz, 256 MB RAM.

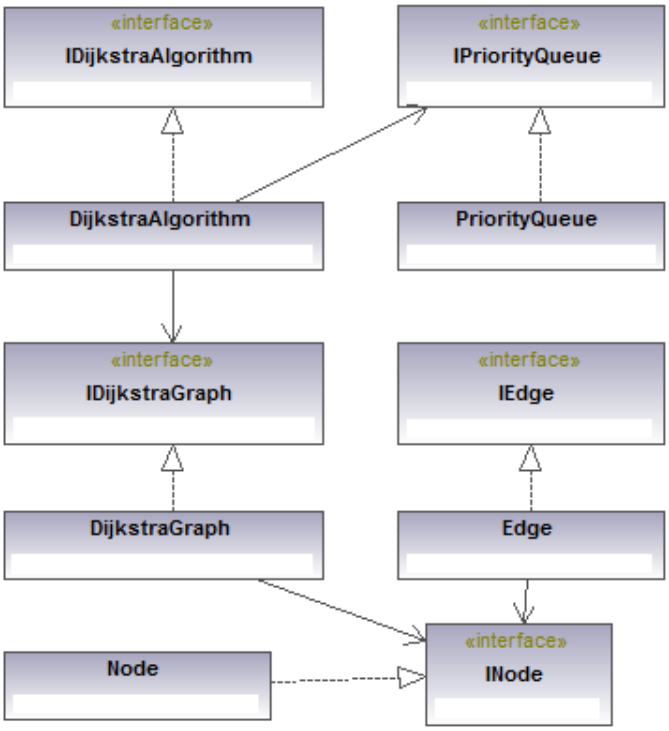


Obrázek 9 – HTC Touch 2

Samotný Dijkstrův algoritmus je implementován následovně:

```
while (!queue.isEmpty()){ // queue – prioritní fronta
      INode node = (INode)queue.remove();
      foreach (IEdge e in node.getOutgoingEdges()){
            INode adjNode = e.getNode();
            UInt16 newPossiblePathCost =
                  Convert.ToUInt16(e.getCost() + node.getDistance());
            if (newPossiblePathCost < adjNode.getDistance()){
                  adjNode.setDistance(newPossiblePathCost);
                  queue.update(adjNode);
                  // predecessors – asociativní pole
                  if (predecessors.ContainsKey(adjNode)){
                        predecessors.Remove(adjNode);
                        predecessors.Add(adjNode, node);
                  }else
                        predecessors.Add(adjNode, node);
            }
      }
}
```

6.4 Popis tříd



Obrázek 10 – Část UML diagramu mobilní aplikace

Na obrázku 10 je zobrazený UML diagram nejdůležitějších tříd, které jsou potřebné pro reprezentaci grafu a výpočet nejkratší cesty. Třída *Node* patří mezi nejdůležitější, obsahuje informace o vrcholu a také o jeho sousedních hranách, které reprezentuje třída *Edge*. Třída *DijkstraGraph* obsahuje dva seznamy – seznam vrcholů a seznam probraných vrcholů, reprezentuje graf pro potřeby Dijkstrova algoritmu.

Vzhledem k tomu, že .NET Compact Framework neobsahuje implementaci prioritní fronty, která je důležitou součástí Dijkstrova algoritmu, bylo nutné prioritní frontu vytvořit (třída *PriorityQueue)*. Fronta je vytvořena nad lineárním seznamem. Tuto prioritní frontu využívá Dijkstrův algoritmus, tedy třída *DijkstraAlgorithm*. Kompletní UML diagram je v příloze B.

6.5 Srovnání rychlosti Dijkstrova a Floyd-Warshallova algoritmu

Tato podkapitola ukazuje rozdíly v rychlosti při použití Dijkstrova algoritmu v řídkých grafech s různým počtem vrcholů a srovnáním Dijkstrova a Floyd-Warshallova algoritmu. Tabulka 4 zobrazuje srovnání těchto algoritmů v desktopové aplikaci na sestavě s konfigurací Intel® Core™ 2 Duo E7200 @ 3,2GHz, 6 GB RAM a operačním systémem Windows 7. Při každém měření bylo provedeno 100 pokusů, ze kterých je uvedeno minimum, maximum a průměr.

Metodika měření byla následující: pomocí generátoru pseudonáhodných čísel se provedl výběr dvou vrcholů, mezi kterými se hledala nejkratší cesta. U Floyd-Warshallova

algoritmu byl počet vrcholů v grafu 100, u Dijkstrova algoritmu byla měřena rychlost výpočtu pro grafy se 100, 200, 300 a 400 vrcholy. Všechny grafy byly řídké. Je uveden celkový čas výpočtu, včetně rekonstrukce cesty a vytvoření matice přímých vzdáleností v případě Floyd-Warshallova algoritmu.

Tabulka 4 – Srovnání časové náročnosti Dijkstrova a Floyd-Warshallova algoritmu (PC)

	Floyd-Warshallův algoritmus*	Dijkstrův algoritmus			
		100 vrcholů	200 vrcholů	300 vrcholů	400 vrcholů
Minimum	4 ms	0 ms	0 ms	0 ms	2 ms
Maximum	9 ms	3 ms	3 ms	7 ms	5 ms
Průměr	5,41 ms	0,38 ms	1.20,ms	2,35 ms	3,73 ms

* Počet vrcholů v grafu je 100.

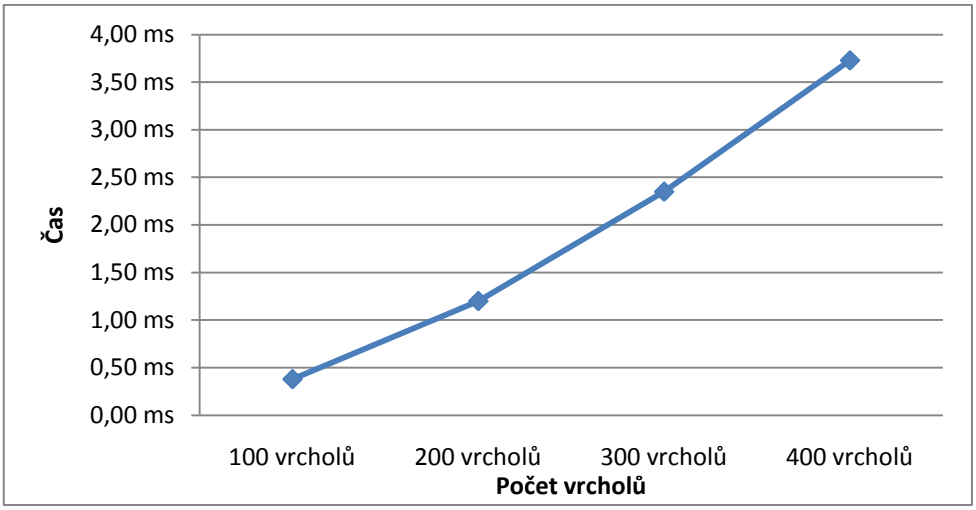
Na výše zobrazené tabulce lze pozorovat poměrně velký rozdíl v rychlosti srovnávaných algoritmů. Zvolená implementace Dijkstrova algoritmu je jednoznačně mnohem rychlejší, u grafu se 400 vrcholy podává lepší výsledky než Floyd-Warshallův algoritmus u grafu se 100 vrcholy.

Tabulka 5 – Srovnání časové náročnosti Dijkstrova algoritmu v závislosti na počtu vrcholů (PDA)

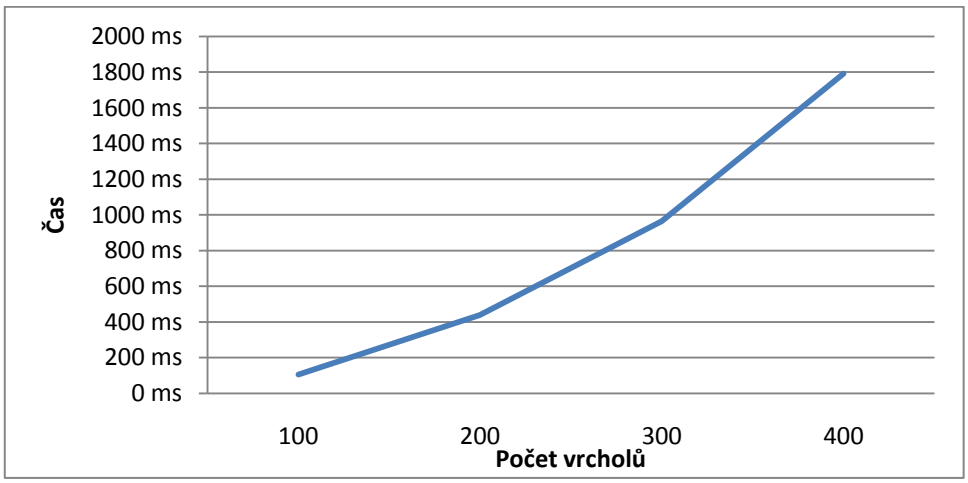
	Počet vrcholů				
	100	200	300	400	445
minimum	87 ms	399 ms	896 ms	1 633 ms	2 091 ms
maximum	199 ms	641 ms	1 099 ms	1 956 ms	2 548 ms
průměr	104,48 ms	493,49 ms	965,00 ms	1 790,94 ms	2 306,18 ms

Rychlost výpočtu nejkratší cesty byla měřena i na mobilním zařízení, výsledky jsou uvedeny v tabulce 5. Mnohem menší výkon mobilního zařízení zapříčinil významně delší čas výpočtu.

Grafy na obrázku 11 a obrázku 12 ukazují závislost délky výpočtu na počtu vrcholů v grafu. Stolní počítač by v tomto případě pravděpodobně bez problémů vyhledal nejkratší cestu i v několikanásobně větších grafech. Mobilní zařízení naopak při větším počtu vrcholů v grafu provádí výpočet přes sekundu a při počtu vrcholů větším než 400 je čas potřebný pro výpočet nejkratší cesty poměrně vysoký.



Obrázek 11 – Délka výpočtu Dijkstrova algoritmu v závislosti na počtu vrcholů (PC)



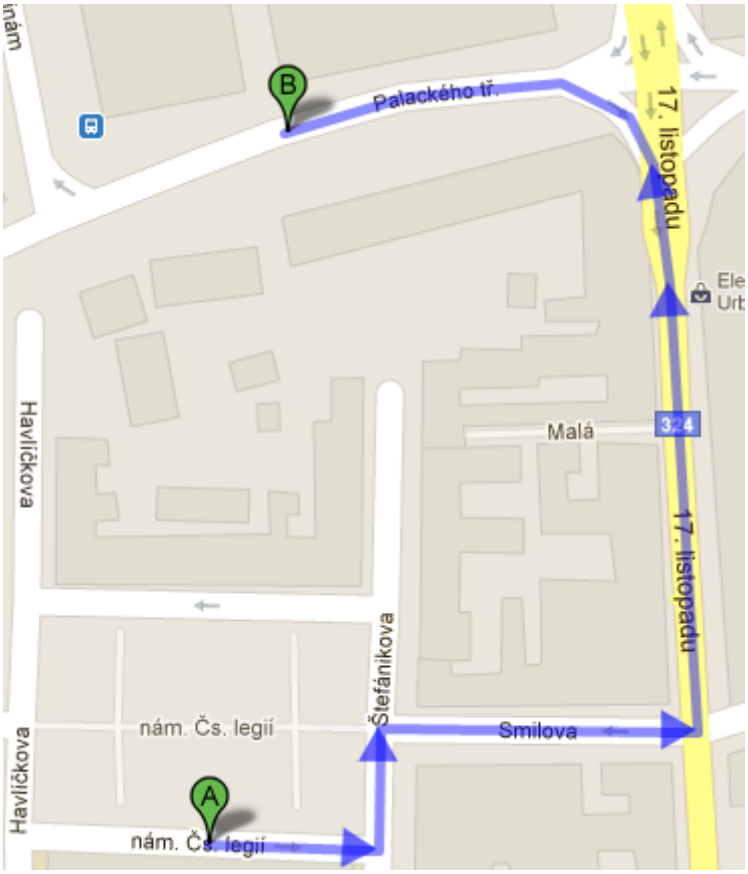
Obrázek 12 – Délka výpočtu Dijkstrova algoritmu v závislosti na počtu vrcholů (PDA)

6.6 Porovnání s existujícími webovými aplikacemi

Tato část se věnuje srovnání výstupu aplikace s podobnými službami, které jsou k dispozici na internetu. Pro srovnání jsem zvolil dvě rozšířené služby – Mapy.cz (www.mapy.cz) a Google Maps (maps.google.com).

Obrázek 13 – Mapy.cz a nejkratší cesta pro automobil a cyklistu

Na obrázku 13 jsou zobrazeny dvě nejkratší cesty mezi dvěma body na mapě. Červeně zvýrazněná cesta představuje nejkratší nalezenou cestu pro automobil, modrá pro cyklistu. Služba Mapy.cz nedokáže nalézt nejkratší cestu s využitím komunikací vyhrazených pro chodce, podchodů nebo nadchodů. Proto je pro pěší vhodnější použít jinou aplikaci.



Obrázek 14 – Nejkratší cesta a Google Maps s optimalizací pro chodce

Google Maps (obrázek 14) dokáže vyhledat cestu nejen pro automobil, ale i pro chodce. Tato funkce je stále v testovací verzi, ale už nyní dokáže při hledání nejkratší cesty využívat některé chodníky a stezky pro chodce. Díky tomu vykazuje z pohledu chodce mnohem lepší výsledky než konkurenční služba (cesta je o třetinu kratší než trasa nalezená službou Mapy.cz). Bohužel nedokáže využít některé, pro chodce průchozí, trasy. Například stále nevyužívá nadchodů nebo pasáží a zjištěná cesta tak není zcela optimální.

Obrázek 15 – Nejkratší cesta s využitím nadchodu a pasáže

Jelikož se v aplikaci tvoří mapa ručně, je možné využít všechny možné cesty, které v mapě třeba ani nejsou zakreslené. Na obrázku 15 modrý ovál zvýrazňuje pasáž a zelený ovál zvýrazňuje nadchod, to jsou dvě možné cesty, které v mapě nejsou zakreslené, ale existují. Díky tomu je v ukázkovém případě cesta o 1/3 kratší oproti cestě, kterou zjistila služba Google Maps.

Tabulka 6 – Porovnání délek nejkratší nalezené cesty (Mapy.cz, Google Maps, vlastní aplikace)

Mapy.cz	Google Maps	Vlastní aplikace
900 m	600 m	391 m

V tabulce 6 je uvedena délka nejkratší nalezené trasy. V tomto ukázkovém případě jsou na tom nejhůře Mapy.cz (s optimalizací trasy pro cyklisty). Google Maps umožní chodci projít jednosměrnou ulicí, díky tomu se cesta zkrátí. Vlastní aplikace využívá všech existujících možností, a proto nalezla skutečně nejkratší cestu.

7 Závěr

Teoretická část práce objasnila pojmy z teorie grafů a dále popisuje nejrozšířenější algoritmy pro vyhledání nejkratší cesty. Jednalo se o Dijkstrův algoritmus a jeho modifikace a dále o Bellman-Fordův a Floyd-Warshallův algoritmus. Další důležitou součástí teoretické části byl popis datových struktur. Hlavní cíl této bakalářské práce - vytvoření aplikace pro mobilní telefon, která je schopná rychle a jednoduše nalézt nejkratší možnou cestu mezi dvěma body a tuto cestu vizualizovat byl splněn.

Díky ukládání dat o grafu do XML nic nebrání použití těchto dat i v aplikaci, která by byla určena pro jinou mobilní platformu (operační systémy Android, iOS). Desktopovou aplikaci lze rozšířit o nové funkce, které by usnadnily tvorbu grafu. Například nástroj pro analýzu mapy a identifikaci možných cest a následnou automatizovanou tvorbu grafu. U mobilní aplikace se nabízí rozšíření v podobě podpory GPS nebo elektronického kompasu.

Literatura

1. **HLINĚNÝ, Petr.** *Základy teorie grafů.* [Dokument] Brno : Masarykova Univerzita, 2010. ISSN 1802-128X.

2. **VOLEK, Josef.** *Operační výzkum 1.* Pardubice : Univerzita Pardubice, 2002. ISBN 80-7194-410-6.

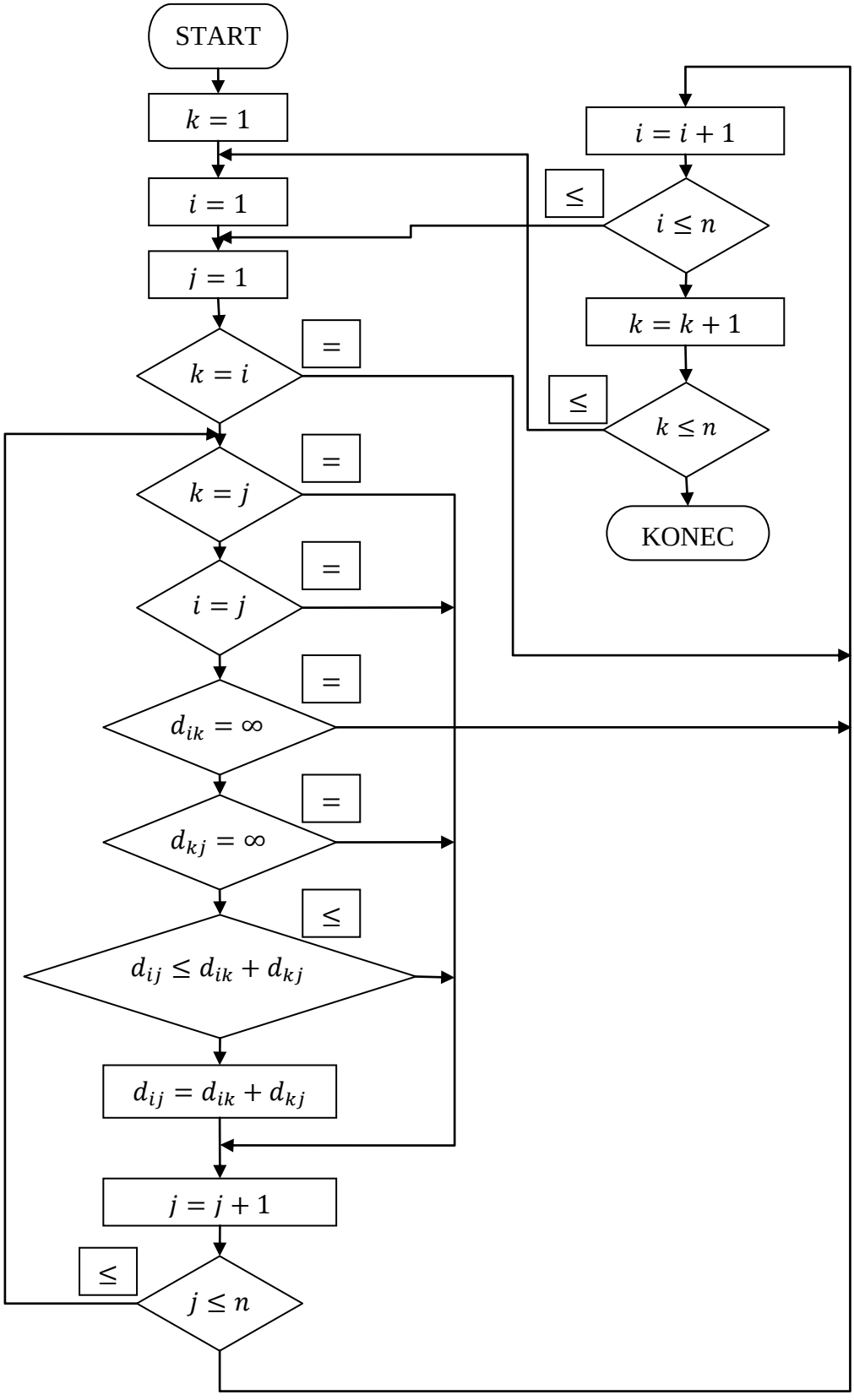
3. **KAVIČKA, Antonín.** *Sylaby z předmětu „Datové struktury" v bakalářském studiu.* [přednášky] Pardubice : Univerzita Pardubice, 2010.

4. Dictionary of Algorithms and Data Structures. *National Institute of Standards and Technology.* [Online] [Citace: 15. Duben 2011.] http://www.nist.gov.

5. **KUČERA, Luďěk.** *Algovize.* Praha : Blatenská tiskárna, 2009. ISBN 978-80-902938-5-4.

6. **KOLÁŘ, Josef.** *Teoretická informatika.* Praha : Česká informatická společnost, 2004. ISBN 80-900853-8-5.

Příloha A – Vývojový diagram Floyd-Warshallova algoritmu (2)



Příloha B – UML diagram