UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2009 **Petr NEJMAN**

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Grafická demonstrace evoluce výpočtu Dijkstrova algoritmu

Petr Nejman

Bakalářská práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Petr NEJMAN**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Grafická demonstrace evoluce výpočtu Dijkstrova algoritmu**

Zásady pro vypracování:

Primárním cílem bakalářské práce je realizace softwarové podpory pro grafickou demonstraci evoluce výpočtu Dijkstrova algoritmu nad hranově ohodnoceným grafem odrážejícím aktuálně obsazenou dopravní síť. Reprezentace grafu je postavena nad vhodnou abstraktní datovou strukturou umožňující efektivní implementaci výše zmíněného algoritmu. Pro testování cílové aplikace se použije vybraný segment reálné dopravní sítě.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

CENEK, P. , KLIMA, V., JANÁČEK, J. Optimalizace dopravních a spojových procesů, Žilina, Univerzita Žilina, 1994. VOLEK, J. Operační výzkum I., skripta DFJP UPa, Pardubice 2002 CORMEN a kol.: Introduction to algorithms, MIT Press, Cambridge, 2001

Vedoucí bakalářské práce: **doc. Ing. Antonín Kavička, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Cegan
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlašuji:

Tuto práci jsem 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 2. 8. 2009

Petr Nejman

Poděkování:

Rád bych zde poděkoval především vedoucímu práce doc.Ing. Antonínu Kavičkovi, Ph.D. za všechny rady, připomínky a čas, který mi věnoval.

Anotace

Práce se věnuje nalezení vhodných abstraktních datových struktur pro výpočet Dijkstrova algoritmu v hranově ohodnoceném grafu. Dále se věnuje grafickému zobrazení výše zmíněného algoritmu s možností krokování celého algoritmu, aby bylo možné sledovat jeho vývoj.

Klíčová slova

grafy, Dijkstrův algoritmus, cesty v grafu, datové struktury

Title

Graphic demonstration of the evolution of the calculation Dijkstra’s algorithm

Annotation

This work is dedicated to finding suitable abstract data structures for the calculation Dijkstra’s algorithm in the edge valued graph. Furthermore, the dedicated graphics display the above algorithm with the possibility of stepping all over the algorithm in order to follow its development.

Keywords

graphs, Dijkstra’s algorithm, paths in a graph, data structures

Obsah

1. ÚVOD 8

2. ZÁKLADNÍ POJMY 9

2.1. POJMY Z DATOVÝCH STRUKTUR 9
2.2. POJMY Z TEORIE GRAFŮ 11

3. DIJKSTRŮV ALGORITMUS 13

4. DATOVÉ STRUKTURY GRAF 15

4.1. SUPER ADT GRAF 15
4.2. VRCHOLOVĚ ORIENTOVANÝ PŘÍSTUP 15
4.2.1. Hvězdy 16
4.2.2. Křížové reprezentace 17
4.3. HRANOVĚ ORIENTOVANÝ PŘÍSTUP 18
4.3.1. Tabulka hran 18
4.3.2. Pole Hran 19

5. VÝBĚR VHODNÝCH DATOVÝCH STRUKTUR 20

5.1. NÁROKY NA DATOVOU STRUKTURU GRAF 20
5.2. VÝBĚR VHODNÉ STRUKTURY GRAF 22
5.2.1. Statická verze 22
5.2.2. Dynamická verze 23
5.3. POMOCNÉ STRUKTURY 24

6. APLIKACE 26

6.1. VÝPOČET ALGORITMU VE ZVOLENÝCH STRUKTURÁCH 26
6.2. UML DIAGRAMY 27
6.3. GRAFICKÉ ZOBRAZENÍ 29
6.3.1. Nároky na grafické zobrazení 29
6.3.2. Dosažené výsledky 30
6.4. UKÁZKA DATOVÝCH STRUKTUR V PRŮBĚHU VÝPOČTU 31
6.5. TESTOVÁNÍ 32

7. ZÁVĚR 33

1. Úvod

Hlavním cílem této práce je vytvoření softwarového prostředí pro demonstraci evoluce Dijkstrova algoritmu nad hranově ohodnoceným neorientovaným grafem. Přesné nároky na grafické znázornění budou uvedeny v příslušné kapitole.

Dále se tato práce zabývá výběrem vhodné abstraktní datové struktury pro uchování grafu, která umožňuje efektivní implementaci výše zmíněného algoritmu. Datové struktury se vyberou ze dvou pohledů. První je určena pro grafy, kde se předpokládá minimální nebo žádná manipulace se samotným grafem, jako např. přidávání nebo odebírání vrcholů. Druhá je určena naopak pro grafy, na kterých se podobné operace očekávají často. Obě by ale stále měli umožňovat efektivní implementaci Dijkstrova algoritmu a vzájemnou zaměnitelnost.

Cílová aplikace je určena pro názornou ukázku výpočtu a dobré pochopení tohoto algoritmu. Je možné ji také využít jako učební pomůcku pro předměty Teorie grafů a Datové struktury. Mimo běžných operací s grafem by tato aplikace měla poskytovat možnost nastavení obsazenosti částí grafu, který bude reprezentovat dopravní síť.

2. Základní pojmy

V této kapitole jsou uvedeny a vysvětleny některé základní pojmy z oblastí datový struktur a teorie grafů, které se budou vyskytovat ve zbytku této práce. Vysvětlení těchto pojmů vychází z použité literatury [2, 3]

2.1. Pojmy z datových struktur

Datové struktury představují způsob uchovávání a organizace dat s cílem umožňovat jejich zpřístupňování a modifikace. Pomocí datových struktur se snažíme v počítači kvalitně realizovat zkoumaný model vzhledem k úspornému využívání pamětí a efektivitě často využívaných operací nebo algoritmů.

Abstraktní datový typ

Abstraktní datový typ (ADT) je to matematická struktura, skládající se ze dvou částí. Jedné nebo více domén, tj. tříd matematických objektů (prvků) a jedné nebo více matematických operací na prvcích těchto domén.

Abstraktní datová struktura

Abstraktní datová struktura (ADS) je konkrétní realizací ADT. Pro jeden ADT se může vytvořit mnoho realizací ADS. ADS také umožňují znovupoužitelnost, tedy vytvoření struktur, které jsou nezávislé na uživatelských datech.

Prioritní fronta

Množina s lineárním uspořádáním, přičemž uspořádání je určováno prioritami prvků vzhledem k jejich pořadí odebírání ze struktury. Jinými slovy je to lineárně uspořádaná množina prvků, které mají určitým způsobem vyjádřenu svou vlastní prioritu. Jednotlivé prvky se postupně ze struktury odebírají v pořadí, které je určeno prioritami prvků od nejvyšší priority k nejnižší.

Pro ukázku je zde využita tabulková specifikace ADT, která bude dále používána i pro další pojmy. V První části je název, v tomto případě *ADT Prioritní fronta*. Druhá část obsahuje

seznam využitých domén a v dalších částech jsou vypsány základní operace pro jednotlivé domény.

Tabulka 1 – ADT Prioritní fronta

ADT Prioritní fronta
A. Třída prvků s prioritou B. Třída konečných prioritních front
A. Vytvoř Zruš JePrázdná (↑Boolean) Mohutnost (↑PočetPrvků) Vlož (↓Prvek) OdeberMax (↑Prvek) ZpřístupniMax (↑Prvek)
B. Sjednocení (↓PrioFrontaA, ↓PrioFrontaB, ↑PrioFrontaC)

Legenda: ↑ - výstupní parametr

↓ - vstupní parametr

Tabulka

Množina s lineárním uspořádáním, přičemž uspořádání je určováno jednoznačnými klíčovými hodnotami (klíči) prvků. Tedy jedná se o lineárně upořádanou množinu prvků, které obsahují svůj identifikátor (klíč). Klíč může být jakéhokoliv typu, ale stále musí splňovat funkci identifikátoru.

Tabulka 2 – ADT Tabulka

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

Graf

Abstraktní datový typ Graf (odrážející binární relaci v množině) představuje heterogenní (nehomogenní) bipartitní strukturu pracující se dvěma odlišnými třídami prvků – Vrcholy a Hranami.

Tabulka 3 – Super ADT Graf

Super ADT Graf	
A. Třída prvků B. Třída konečných grafů	
A. Vytvoř Zruš JePrázdný (↑Boolean) Mohutnost (↑PočetPrvků) 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 (↑Prvky)	vrcholová/hranová, do hloubky/šířky
B. Sjednocení (↓GrafA, ↓GrafB, ↑GrafC)	

2.2. Pojmy z teorie grafů

Teorie grafů zkoumá vlastnosti struktur, kterým říkáme grafy. Grafy jsou obvykle znázorňovány pomocí množiny bodů spojených čarami. Pomocí těchto grafů můžeme znázornit struktury nebo úlohy z nejrůznějších oborů. To znamená, že mnoho problémů z praktického života může být převedeno na úlohu z teorie grafů.

Neorientovaný graf

Můžeme ho zapsat jako uspořádanou trojici $G = (V, X, p)$. Prvky množiny V nazýváme vrcholy grafu, prvky množiny X hrany grafu a p (incidence) je zobrazení množiny X na množinu všech neuspořádaných dvojic (u, v), kde $u, v \in V$. Vrcholy u a v můžeme nazvat sousední vrcholy.

Orientovaný graf

$D = (V, Y, p)$ je podobný neorientovanému grafu. Je složen ze dvou množin V a Y a zobrazení p množiny $Y \rightarrow V \times V$. Množina V má stejný význam jako u neorientovaného grafu, ale množina Y je tvořena uspořádanými dvojicemi $[u, v]$ prvků množiny V. Každá tato dvojice se vyskytuje v množině Y nejvýše jednou a platí, že $u \neq v$. Prvky množiny Y nazýváme orietované hrany grafu.

Souvislost grafů

Máme dvojici vrcholů $u, v \in V$ grafu $G = (V, X, p)$, nechť existuje střídavá posloupnost vrcholů a hran:

$S=\{u_0, h_1, u_1, h_2, \ldots, h_n, u_n\}$,

kde $h_i \in X$, $p(h)=(u_{i-1}, u_i)$ pro $i=1, \ldots, n$, $u_i \in V$ pro $i=1, \ldots, n$, $u_0=u, u_n=v$

S potom můžeme nazvat **sledem** grafu G mezi vrcholy u a v. Sled, ve kterém se neopakují žádné hrany, nazýváme **tahem**. Tah, ve kterém se neopakují žádné vrcholy, nazýváme **cestou**.

Graf, ve kterém existuje pro každou dvojici vrcholů u, v alespoň jedna cesta, nazýváme **souvislým grafem**.

Hranově ohodnocený graf

Graf, jehož každá hrana má své ohodnocení. Toto ohodnocení může symbolizovat např. délku, kapacitu nebo spolehlivost objektu, který hrana symbolizuje.

3. Dijkstrův algoritmus

Jedná se o grafový algoritmus, který dokáže vyhledat nejkratší cesty z jednoho vrcholu do všech ostatních v hranově ohodnoceném grafu. Může se použít i na hranově neohodnocený graf, ale v tomto případě uvažujeme ohodnocení každé hrany rovno 1. Nejkratší cesta potom označuje minimální počet hran k dosažení kteréhokoli vrcholu. Jedinou podmínkou správné funkčnosti je nezápornost ohodnocení všech hran. Algoritmus je konečný pro jakýkoliv konečný graf, který samozřejmě splňuje výše zmíněnou podmínku. Počet cyklů v algoritmu je roven maximálně počtu vrcholů zkoumaného grafu. V případě nesouvislého grafu neexistuje cesta mezi všemi vrcholy grafu a tím pádem bude počet cyklů v algoritmu nižší v závislosti na počátečním vrcholu. Informace uvedené v této kapitole jsem čerpal z použité literatury [2,4].

Tento algoritmus můžeme vysvětlit na hranově ohodnoceném grafu $G=(V, X, p)$, kde $V=\{v_0, v_1, \ldots, v_n\}$ a ohodnocení hran $o(h)$ vyjadřuje délky jednotlivých hran. Dále potřebujeme množinu dočasně ohodnocených vrcholů $D=\{\}$ a množinu trvale ohodnocených vrcholů $T=\{\}$. Samotný algoritmus potom můžeme rozdělit do několika kroků:

1. krok: V grafu G vybereme počáteční vrchol v_z (např. $v_z = v_0$).

Všem vrcholům $vi \in V$ přiřadíme ohodnocení:

- $t_0 = 0$ pro v_0 (v_z)
- $t_i = \infty$ pro $v_i \in V$, $i = 1, 2, \ldots, n$

Do množiny D přiřadíme počáteční vrchol u.

2. krok: Vybereme z množiny D vrchol v_p s nejnižším ohodnocením a zařadíme ho do množiny T. V této chvíli už vrchol v_p nenáleží množině D.

3. krok: Pro každý vrchol v_j, který je sousedem vrcholu v_p a zároveň není v množině T provedeme test: $t_p + o(v_p, v_j) < t_j$ pokud toto platí:

- provedeme přehodnocení: $t_j = t_p + o(v_p, v_j)$
- přiřadíme vrcholu v_j předchůdce v_p: $p_j = v_p$
- pokud vrchol v_j dosud nebyl v množině D, tak ho tam zařadíme

4. krok: Pokud množina D není prázdná, vracíme se na **krok 2**. Jinak algoritmus končí.

Vyhledání stromu nejkratších cest

Pokud byl graf G, na kterém jsme aplikovali Dijkstrův algoritmus, souvislý, tak můžeme říci, že pro každý vrchol $v_i \in V$ se t_i rovná délce nejkratší cesty mezi počátečním vrcholem a vrcholem v_i. Dále máme k dispozici označení p_i, které nám udává předchůdce vrcholu v_i.

Všechny potřebné informace k vytvoření stromu nejkratších cest už nám algoritmus vyhledal. Pokud si uvědomíme, že známe pro každý vrchol v_i jeho předchůdce p_i, tak už není problém z jakéhokoli vrcholu nalézt cestu k počátečnímu vrcholu. Pokud bychom si tento graf kreslili, tak nám k vytvoření stromu nejkratších cest postačí spojit každý vrchol se svým předchůdcem.

Vyhledání nejkratší cesty mezi dvěma vrcholy

Jak vyhledat jednotlivé cesty z výsledků algoritmu už bylo zmíněno výše. Ale pokud nepotřebujeme vyhledávat celý strom cest, tak můžeme i samotný algoritmus urychlit.

Tohoto urychlení dosáhneme drobnou úpravou algoritmu. V *kroku 1* si kromě výběru počátečního vrcholu v_z, vybereme i koncový vrchol v_k. Teď už jen postačí přidat novou podmínku ukončení algoritmu. V *kroku 2* nám algoritmus končí, pokud z množiny D vybereme výše zmíněný vrchol v_k. Zde se algoritmus ukončí, protože našel nejkratší cestu do potřebného vrcholu a dále už by jen hledal cesty do ostatních vrcholů.

Nejkratší cesta mezi vrcholy v_z a v_k lze vyhledat i bez použití předchůdců. Rekonstrukci cesty začneme v koncovém vrcholu v_k. Dále hledáme takový sousední vrchol v_s vrcholu v_k, pro který platí $t_s + o(v_s, v_k) = t_k$. Tento vrchol v_s potom zahrneme do cesty. Obdobným způsobem jako jsme hledali souseda vrcholu v_k, hledáme souseda i vrcholu v_s. Tímto způsobem postupně vyhledáme nejkratší cestu až k počátečnímu vrcholu.

Může se stát, že podmínku pro zařazení vrcholu do cesty splňují najednou 2 nebo více sousedních vrcholů. To znamená, že existuje více nejkratších cest k dosažení počátečního vrcholu. Který vrchol si vybereme, je už na nás, ale k počátečnímu vrcholu vedou všechny.

Tento způsob rekonstrukce je sice zdlouhavý oproti použití předchůdců, ale nabízí možnost vyhledání všech možných nejkratších cest mezi dvěma vrcholy.

4. Datové struktury Graf

Abychom byly schopni efektivně uchovat a pracovat s grafem, musíme vybrat vhodnou datovou strukturu. Datových struktur schopných uchovat graf je mnoho. Liší se svou strukturou, vlastnostmi a přístupem ke grafu. V této kapitole jsem vycházel z použité literatury [3, 1]

4.1. Super ADT Graf

Jedná se o abstraktní datový typ, definující obecné základní operace nad grafem. Vzhledem k nárokům a povaze odvozeného ADS, nemusejí být implementovány všechny operace, které Super ADT Graf definuje, ale pouze ty, které se využijí.

ADS vymezené nad Super ADT Grafem můžeme klasifikovat podle složitostí některých operací. Tyto operace jsou typu *Vlož* a *Odeber* pro vrcholy a hrany. Pokud operace *VložVrchol* resp. *OdeberVrchol* není realizována nebo složitost této operace není menší než *O(n)*, jedná se o vrcholově statickou strukturu. Pokud jsou naopak tyto složitosti menší než *O(n)*, tak se jedná o vrcholově dynamickou strukturu. Obdobně můžeme posoudit pro operace *VložHranu* a *OdeberHranu*, jestli je struktura hranově statická či hranově dynamická

4.2. Vrcholově orientovaný přístup

Vstupní branou do struktury je vždy vrchol nebo vrcholy. Zpřístupňování hran je až druhotné. Podstatou vrcholově orientovaného přístupu je rozdělení struktury na dvě části:

Prvotní struktura

Obsahuje informace o vrcholech. V této struktuře je záznam o každém vrcholu pouze jednou a obsahuje určitý identifikátor vrcholu, informace pro vstup do patřičné části druhotné struktury a ostatní potřebné informace o vrcholu.

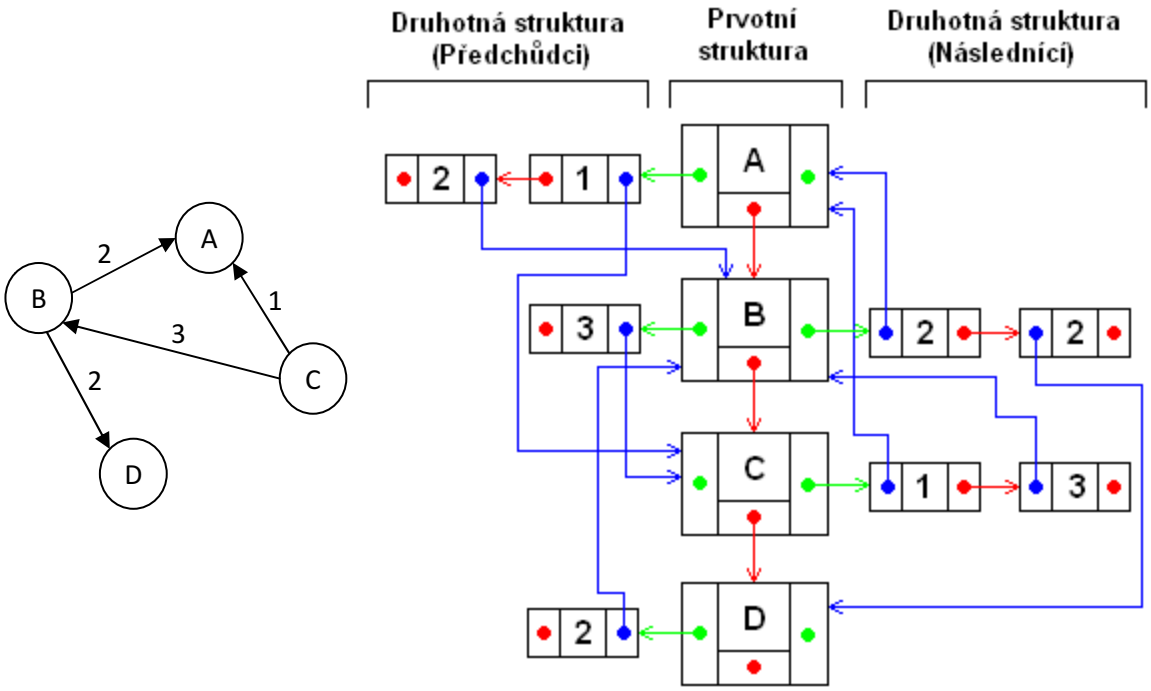
Druhotná struktura

Zde jsou uloženy informace o incidentních hranách, následnících nebo předchůdcích pro každý vrchol prvotní struktury. Druhotná struktura může být celistvá a jednotlivé vrcholy

prvotní struktury se budou odkazovat na určitou část druhotné struktury, nebo se může druhotná struktura skládat z *n* dílčích struktur, kde *n* je počet vrcholů v prvotní struktuře. Takže každý vrchol v prvotní struktuře může obsahovat odkaz na vlastní druhotnou strukturu.

4.2.1. Hvězdy

Hvězda je tvořena prvotní strukturou, uchovávající vrcholy, a druhotnou strukturou, která obsahuje informace o hranách a následnících, nebo předchůdcích.



Obrázek 1 – Dopředně-zpětná hvězda

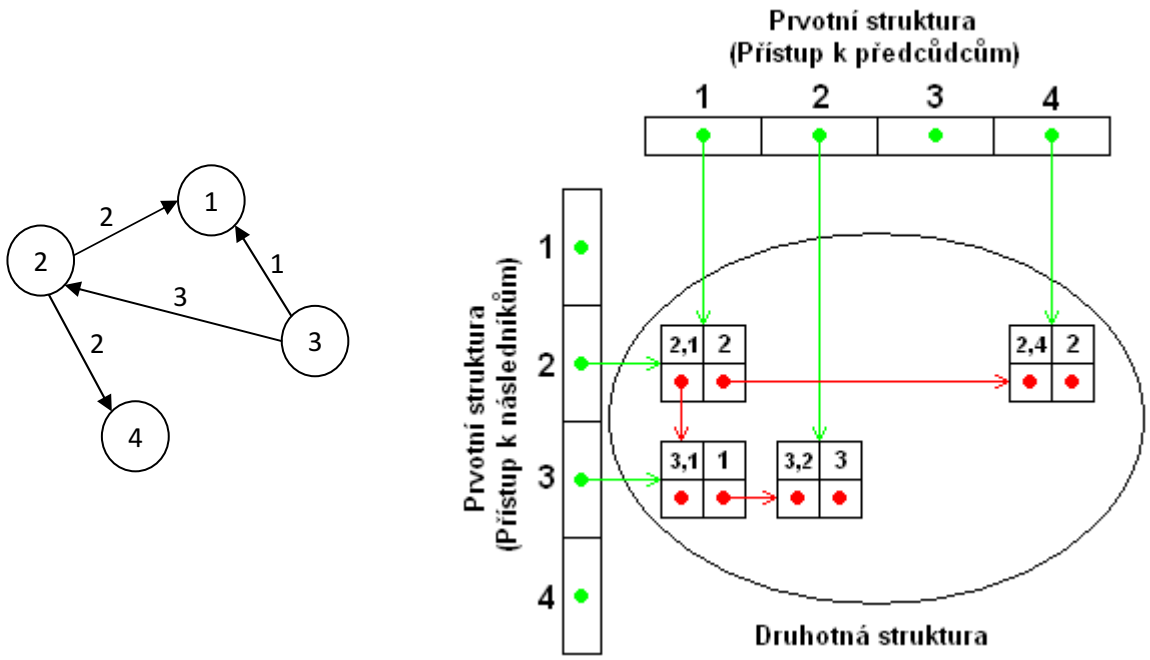
Na ukázku možné konstrukce hvězdy jsem vybral dopředně-zpětnou hvězdu *Tabulka(seznam) - Tabulka(seznam) - Tabulka(seznam)* vytvořenou pro uchování orientovaného grafu. Prvotní struktura obsahuje ukazatele do druhotných struktur (zvlášť pro předchůdce a následníky). Obsahuje také klíčovou položku identifikující vrchol (v tomto případě písmena A, B, …) a ukazatel na další vrchol v prvotní struktuře. Položky seznamů druhotných struktur obsahují ukazatel na následníka/předchůdce (dle druhotné struktury), ohodnocení hrany a ukazatel na dalšího následníka/předchůdce pro konkrétní vrchol.

V případě neorientovaných grafů nám postačí jedna druhotná struktura, protože nemusíme uchovávat zvlášť informace o hranách, vedoucích k následníkovi, nebo k předchůdci. Takovým hvězdám říkáme dopředné hvězdy.

Jednotlivé realizace prvotních a druhotných struktur se odvíjejí od požadavků na danou strukturu. Obecně lze říci, že prvotní i druhotnou strukturu lze realizovat pomocí pole nebo pomocí tabulky, kterou lze samozřejmě vytvořit mnoha způsoby. Využitelné jsou téměř všechny kombinace až na kombinaci *tabulka-pole*, která se prakticky nepoužívá. Pokud v prvotní struktuře identifikuji vrcholy pomocí klíčů, už není důvod k identifikaci pomocí indexů pole v druhotné struktuře. Výhody či nevýhody hvězdy vyplývají až z výběru konkrétních datových struktur pro vytvoření prvotní a druhotné struktury.

4.2.2. Křížové reprezentace

Pokud potřebujeme současný přístup jak k následníkům, tak i k předchůdcům, můžeme také zvolit tzv. křížovou reprezentaci. Není jednoznačně lepší než hvězda, je to jen jistá alternativa, kterou můžeme zvážit při výběru vhodné datové struktury pro uchování grafu. Podstatou této reprezentace je udržování dvou prvotních struktur. Jedné pro přístup k následníkům a druhé pro přístup k předchůdcům. Druhotná struktura je sice jen jedna, ale obsahuje informace k přístupu jak k následníkům, tak i k předchůdcům.



Obrázek 2 – Křížová reprezentace

Pro ukázku křížové reprezentace jsem vybral variantu *pole-tabulka(seznam) x pole-tabulka(seznam)*. Jak už je vidět z obrázku, jednotlivé vrcholy jsou identifikovány pomocí indexů pole. Prvky druhotné struktury obsahují ukazatele pro přístup následníkům i

k předchůdcům. Obsahují také ohodnocení hrany, kterou reprezentují a souřadnice, podle kterých můžeme zjistit totožnost hrany.

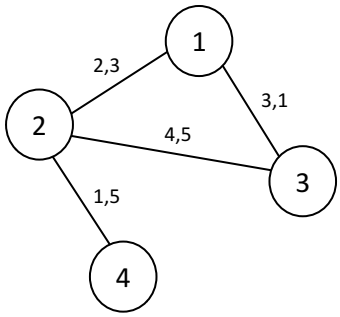
Obecně se tyto struktury vytvářejí dvěma způsoby: *pole-tabulka x pole-tabulka a tabulka-tabulka x tabulka-tabulka*. Při výběru záleží samozřejmě na požadavcích a jednotlivé výhody nebo nevýhody opět vycházejí až z konkrétních realizací.

4.3. Hranově orientovaný přístup

Při hranově orientovaném přístupu ke grafu, jsou vyhledávací operace zaměřeny na vyhledávání hran. Zpřístupnění vrcholů je potom až druhotné. Identifikace hran potom jde realizovat použitím dvojice vrcholů, se kterými je hrana incidentní nebo pomocí pevného očíslování hran.

4.3.1. Tabulka hran

Při identifikaci hran dvěma vrcholy, které jsou s hranou incidentní, můžeme graf považovat za tabulku hran. Každá hrana je tedy identifikována klíčem, který je složen z identifikátorů incidentních vrcholů. Při tomto způsobu je potom možné při implementaci vytvořit tabulku libovolným způsobem. Toto řešení potom umožňuje hranově dynamické operace.



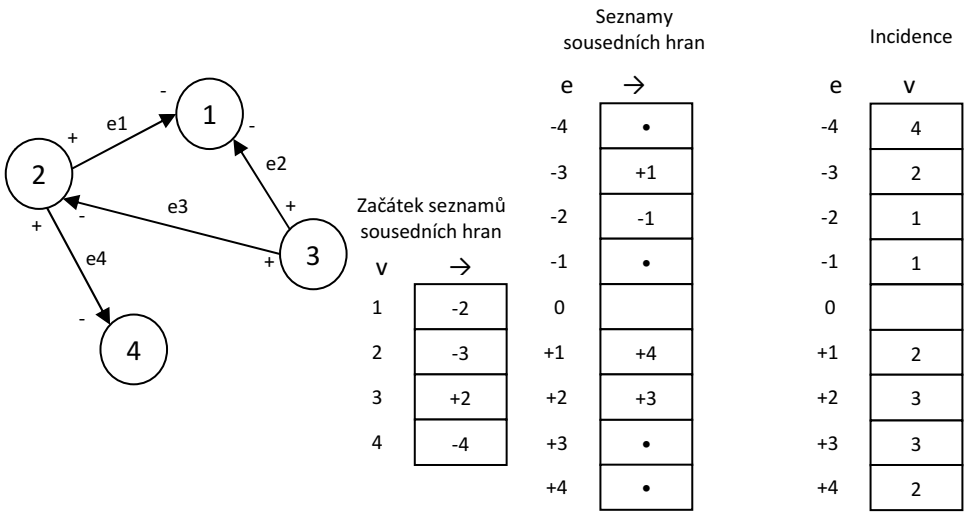
Počáteční vrchol	Koncový vrchol	Ohodnocení
2	1	2,3
2	3	4,5
4	2	1,5
1	3	3,1

Obrázek 3 - Tabulka hran

Na ukázku je zde opět obrázek Tabulky hran. V tomto případě identifikujeme každou hranu složeným klíčem, který se skládá z identifikátoru počátečního a koncového vrcholu, tedy např. v prvním řádku tabulky je klíč *(2, 1)*.

4.3.2. Pole Hran

Pokud hrany identifikujeme prvními *n* přirozenými čísly, můžeme graf považovat za pole hran. Tato reprezentace má několik nedostatků, které se ale dají vyřešit jako např. orientace hran. Pokud máme pouze pole hran, tak z něho nepoznáme orientaci jednotlivých hran. Tento problém můžeme vyřešit, když přidáme seznamy sousedních hran. Vyřešení podobných problémů nám umožní navrhnout poměrně efektivní hranově statickou reprezentaci grafu.



Obrázek 4 – Pole hran se seznamy sousednosti

Legenda: v - identifikátor vrcholu (index pole)

→ - index následující položky pole

• - značí, že už v poli není žádná další sousední hrana

e - identifikátor hrany (pomocí +/- můžeme hlídat orientaci)

Na obrázku můžeme vidět příklad *Pole hran se seznamy sousednosti*. Oproti předchozím strukturám je na pochopení trochu složitější, ale na druhou stranu můžeme pomocí této struktury efektivně uchovávat smíšené grafy (obsahují orientované i neorientované hrany) a grafy obsahující smyčky.

Jak vidíme z obrázku, tato struktura se skládá ze tří polí. První s označením *Začátek seznamu sousedních hran* obsahuje pro každý vrchol, identifikovaný indexem tohoto pole, začátek seznamu svých sousedních hran (např. pro vrchol s označením *1* je začátek seznamu na indexu *-2*). Tento seznam nalezneme v sousedním poli s označením *Seznamy sousedních hran*. Indexy tohoto pole označují jednotlivé hrany i s jejich orientací (orientace od + k -). Položky tohoto pole ukazují další index v seznamu hran (např. pro vrchol s označením *1* je seznam hran: *-2, -1*, to znamená, že do vrcholu *1* vstupují hrany *e2, e1*). Posledním polem je

pole s označením *Incidence*. Jak už název napovídá, v tomto poli je pro každou stranu hrany uveden vrchol, se kterým inciduje (např. první položka *e: -4, v: 4* značí hrana *e4* vchází do vrcholu s označením *4*).

5. Výběr vhodných datových struktur

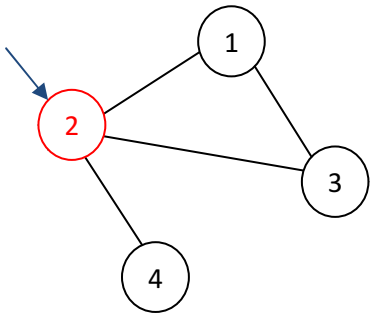
Výběr vhodných datových struktur je velice důležitý z hlediska úspory paměti a efektivního používání častých operací a algoritmů. V této kapitole si rozebereme nároky, které na datovou strukturu klade Dijkstrův algoritmus a vybereme vhodné struktury k uchování grafu a k použití výše zmíněného algoritmu.

5.1. Nároky na datovou strukturu graf

Dijkstrův algoritmus klade na vybíranou datovou strukturu nároky, které vycházejí ze samotného průchodu algoritmem. Abychom byli schopni efektivně využít tento algoritmus, musíme při výběru datové struktury zohlednit i nároky kladené algoritmem.

Rychlý přístup ke vybranému vrcholu

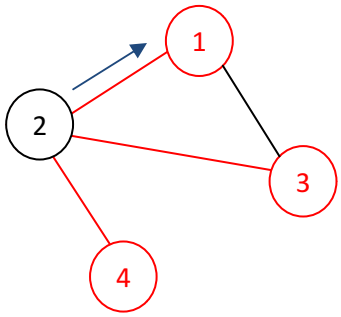
V Dijkstrově algoritmu postupujeme pomocí vyhledávání jednotlivých vrcholů s nejmenším dočasným ohodnocením. Tudíž potřebujeme, aby nám volená datová struktura poskytla co nejrychleji přístup k vybranému vrcholu.



Obrázek 5 – Přístup k vybranému vrcholu

Rychlý přístup k sousedním vrcholům

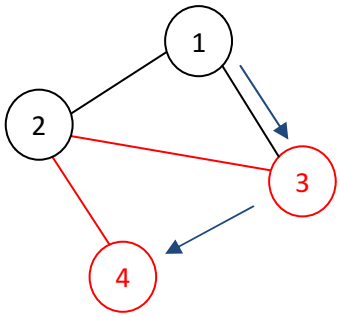
Když v algoritmu prohlásíme dočasně ohodnocený vrchol za trvale ohodnocený, tak potřebujeme přístup k jeho sousedním vrcholům a také incidentním hranám pro případnou úpravu ohodnocení vrcholů. Není nutné mít okamžitě přístup ke všem sousedním vrcholům najednou, stačí přístup k jednomu libovolnému sousednímu vrcholu a jeho incidentní hraně a potom postupně procházet ostatní, jak bude uvedeno dále.



Obrázek 6 – Přístup k sousedním vrcholům

Rychlé procházení sousedních vrcholů

Při přehodnocování dočasných ohodnocení sousedních vrcholů musíme projít celé vrcholové okolí. Pokud už máme přístup k jednomu sousednímu vrcholu, musí nám volená datová struktura umožnit co nejrychlejší přechod na další sousední vrchol a přístup k patřičné incidentní hraně. Pořadí procházení vrcholů je libovolné, důležité je jen to, abychom otestovali všechny sousední vrcholy.



Obrázek 7 – Průchod mezi sousedními vrcholy

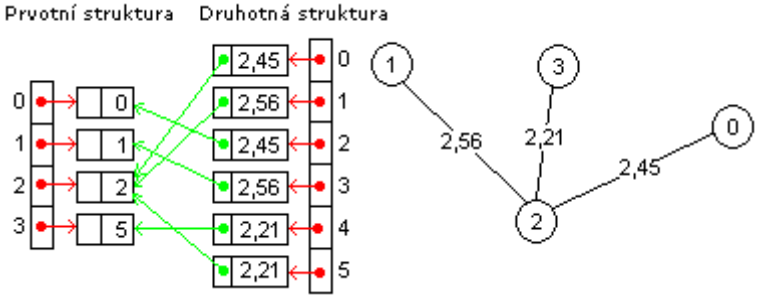
5.2. Výběr vhodné struktury graf

Jak už je zřejmé z nároků na datovou strukturu, jedná se o vrcholově orientovaný přístup ke grafu. Vstupní branou do struktury je vždy vrchol. Protože potřebujeme udržovat pouze neorientovaný graf, odpadá řešení pomocí křížové reprezentace. Z možností, které jsou uvedeny v kapitole 4, zbývá už jen reprezentace grafu pomocí hvězdy. Jelikož je aplikace určena pro zpracování neorientovaných grafů, není nutné udržovat zvlášť následníky a předchůdce jednotlivých vrcholů. Proto postačí realizace pomocí dopředné hvězdy. Prvotní struktura bude obsahovat jednotlivé vrcholy a druhotná struktura incidentní hrany. Prvotní struktura musí podle požadavků rychle zpřístupnit vrchol, přes který se musíme rychle dostat k jeho incidentním hranám. K druhotné struktuře se váže jen požadavek rychlého přístupu k další incidentní hraně. Vyhledávání konkrétní hrany není prioritní účel, postačí procházení jednotlivých hran. Hrany by měly rychle zpřístupňovat incidentní vrcholy.

5.2.1. Statická verze

Hvězda postavená jako pole - pole. Vyhledávání jednotlivých vrcholů probíhá pomoví indexů pole, které jsou zde jako kódy (identifikátory) jednotlivých vrcholů. Přístup z vrcholu k jeho incidentním hranám je realizován pomocí indexu první incidentní hrany v druhotné struktuře tedy opět v poli. Každá hrana obsahuje ukazatel na následující vrchol, takže je možné bez vyhledávání přistoupit až k incidentnímu vrcholu.

Vrcholy se hledají pomocí indexů pole, takže složitost této operace je O(1). Zpřístupnění incidentních hran probíhá také pomocí indexů pole, takže obtížnost je opět O(1). A nakonec zpřístupnění další incidentní hrany probíhá navýšením posledního indexu o 1, takže složitost by měla být opět O(1). Musíme jen hlídat, abychom nebrali incidentní hrany jiného vrcholu.



Obrázek 8 – Příklad navržené statické verze hvězdy

Jedná se o velmi specifický návrh hvězdy, protože každý vrchol prvotní struktury má mimo jiného jen index do pole druhotné struktury. Počet incidentních hran se potom zjistí výpočtem s vrcholem, který je identifikován následovným indexem v poli. Pro lepší pochopení uvedu příklad z obrázku 8:

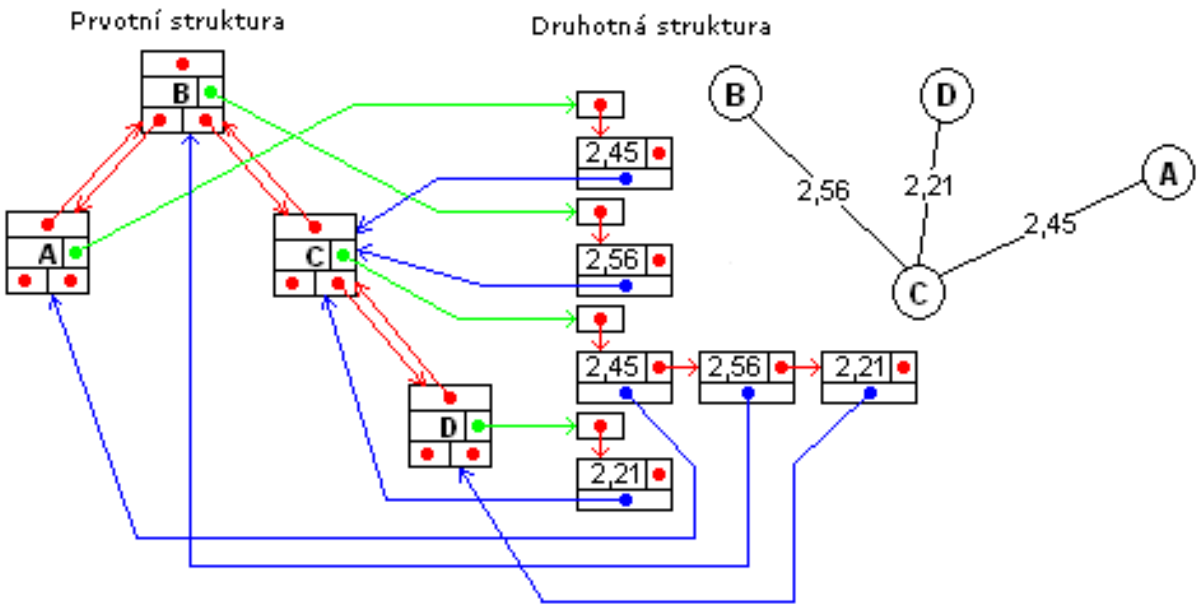
Vrchol s indexem 2 uchovává číslo 2(index do pole druhotné struktury). Vrchol označený indexem 3 uchovává číslo 5. Takže jednoduchým výpočtem zjistíme, že vrchol s indexem 2 má 3 incidentní hrany (5-2=3) na indexech druhotné struktury 2, 3, 4 (pod indexem 5 je už uložena hrana incidentní s vrcholem značeným indexem 3).

Tato verze hvězdy je určena pro grafy, kde se nepředpokládá přidávání či odebírání vrcholů nebo hran. Tyto operace jsou v této implementaci hvězdy poměrně náročné, protože při vkládání nového prvku do zaplněného pole, často dochází k posouvání celých polí, což je při velkém počtu prvků v poli velice zdlouhavé. Na druhou stranu tato struktura plně splňuje nároky, které klade Dijkstrův algoritmus.

5.2.2. Dynamická verze

Tato hvězda je stavěna jako *tabulka(AVL strom) – tabulka(seznam).* Logická struktura AVL stromu je stejná jako struktura binárního stromu, ale pomocí vyvažování se udržuje v nezdegenerovaném stavu. Prvotní struktura je realizována jako dynamický vyvážený binární strom. Druhotná struktura je realizována jako *n* dynamicky zřetězený seznamů, kde *n* je počet vrcholů. Stejně jako u statické verze i tady každá hrana obsahuje ukazatel na incidentní vrchol.

Vyhledávání jednotlivých vrcholů v AVL stromu má složitost O ($\log_2 n$). Podmínka rychlého přístupu k incidentním hranám je splněna tím, že každý vrchol obsahuje ukazatel na seznam svých incidentních hran. Navíc jsou hrany v jednotlivých seznamech dynamicky zřetězeny, takže podmínka rychlého průchodu z jedné incidentní hrany (sousedního vrcholu) na druhou je splněna se složitostí O(1).



Obrázek 9 – Příklad navržené dynamické verze hvězdy

Oproti statické verzi má tato dynamická verze pomalejší vyhledávání vrcholů, ale na druhou stranu se mnohem snáze vkládají vrcholy a hrany. Takže tato struktura je určena pro případy, kde se předpokládají časté dynamické operace.

5.3. Pomocné struktury

V této podkapitole jsem čerpal z použité literatury [3].

Dijkstrův algoritmus pracuje tak, že vybírá vrchol z množiny dočasně ohodnocených vrcholů a prohlásí ho za finálně označený. Poté je nutno upravit některá ohodnocení sousedních vrcholů v závislosti na jejich původním ohodnocení.

Z toho vyplývá, že potřebujeme efektivně vybírat vrcholy s nejmenším ohodnocením. K tomu se nám hodí prioritní fronta.

Jako první a nejjednodušší implementace je obyčejný seznam. Můžeme do seznamu vkládat prvky utříděně a vybírat vždy první prvek. Tato implementace má složitosti operaci *vlož O(n)* a *odeber O(1)*. Nebo můžeme vkládat prvky vždy na první místo, ale pokud chceme vybírat, tak si v seznamu prvek s nejnižší prioritou musíme najít. Složitosti operací potom vypadají takto: *vlož O(1)*, *odeber O(n)*.

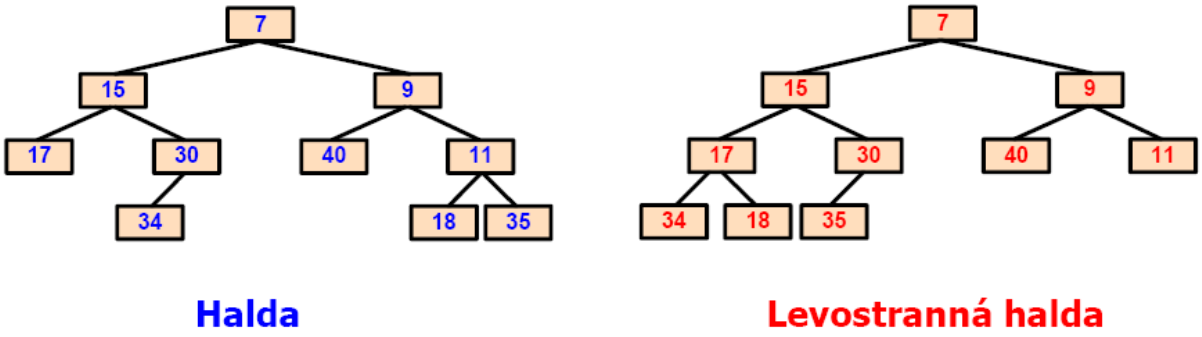
Jak je vidět ze složitostí operací, tak seznam není nejlepší způsob implementace prioritní fronty. Dalšími způsoby implementace jsou dvojúrovňová prioritní fronta a dvojseznamová prioritní fronta. Jejich složitosti, pro operace typu vlož a odeber, jsou shodné a to $O(n^{1/2})$. Sice

nemají žádnou operaci se složitostí *O(1)*, ale při velkém počtu prvků ve frontě jsou efektivnější než utříděný nebo neutříděný seznam.

Je tu také možnost implementace v podobě haldy. Operace typu *vlož* a *odeber* mají složitost $O(log_2 n)$, takže i při vysokém počtu čísel je tato fronta poměrně rychlá a v porovnání s ostatními uvedenými implementacemi efektivnější.

Z výše uvedených možností vychází, pro větší počet prvků ve frontě, nejlépe realizace prioritní fronty pomocí haldy. Pro každou haldu platí, že priorita každého prvku je větší než priorita obou jeho synů, pokud existují.

S výhodou můžeme použít implementaci haldy pomocí pole. K této implementaci se ale nejlépe hodí varianta haldy a to sice levostranná halda. Pro levostrannou haldu platí vše, co již bylo zmíněno obecně k haldě. Liší se jen v uspořádávání prvků, jak můžete vidět na obrázku 10.



Obrázek 10 – Haldy [3]

Implementace levostranné haldy pomocí pole má potom následující pravidla. Pokud máme prvek s indexem „i“, tak jeho synové, pokud existuji, jsou na indexech *2i* a *2i+1*. Otce tohoto prvku bychom mohli najít pomocí celočíselného dělení (*div*) na indexu *i div 2*. Z toho vyplývá, že při použití levostranné haldy máme zaplněno vždy prvních n pozic pole, kde n je počet prvků haldy.

6. Aplikace

V této kapitole můžeme najít výpočet Dijkstrova algoritmu za pomoci datových struktur. Budou zde ukázány vazby mezi jednotlivými strukturami statické i dynamické verze hvězdy a prioritní fronty pomocí UML diagramů. Dále zde budou řešeny nároky na vzhled grafu v jednotlivých fázích výpočtu a ukázka aktuálně obsazených datových struktur v průběhu algoritmu.

6.1. Výpočet algoritmu ve zvolených strukturách

Dijkstrův algoritmus už byl obecně vysvětlen v kapitole 3, ale zde je vysvětlen výpočet za použití datových struktur:

1. Inicializace

Před spuštěním algoritmu je třeba inicializovat některé datové struktury. Nejdříve je potřeba nastavit ohodnocení všech vrcholů, mimo vrchol, kterým začínáme, na ∞. Tuto hodnotu však nemůžeme reálně nastavit, proto místo ní nastavíme hodnotu, o které předpokládáme, že jí nedosáhneme.

Musíme také inicializovat frontu, tak aby byla prázdná. Do této fronty vložíme startovní vrchol s ohodnocením 0 - (ve frontě přidáme dalšímu prvku pole ukazatel na potřebný vrchol v datové struktuře uchovávající graf).

2. Výběr z prioritní fronty

Z prioritní fronty se vybere prvek s nejvyšší prioritou tj. vrchol s nejnižší dočasným ohodnocením. Tento vrchol můžeme prohlásit za trvale ohodnocený, protože neexistuje kratší cesta, kterou bychom už nezpracovali.

3. Aktualizace ohodnocení

Musíme prozkoumat všechny sousední vrcholy (vrcholové okolí) právě vybraného vrcholu. Pokud $t_i + o(v_i, v_j) < t_j$, kde t_i je ohodnocení právě vybraného vrcholu, t_j je ohodnocení

sousedního vrcholu a $o(v_i, v_j)$ je ohodnocení hrany mezi těmito vrcholy, tak musíme udělat několik úprav:

- Provedeme přehodnocení $t_j = t_i + o(v_i, v_j)$.
- Nastavíme vrchol v_i jako předchůdce vrcholu v_j.
- Pokud vrchol v_j dosud není v prioritní frontě, tak ho do ní zařadíme. Pokud už tam je, tak ho jen případně zařadíme na správné místo, odpovídající nové prioritě.

Zjišťování zda je prvek ve frontě či ne, bychom mohli provádět průchodem frontou a zkoumáním jestli tento vrchol samotná fronta obsahuje. Toto řešení by bylo velice zdlouhavé, obzvlášť při vysokém počtu prvků ve frontě. Efektivnější způsob je ten, že každý vrchol, v prvotní struktuře dopředné hvězdy, obsahuje příznak, označující jeho existenci ve frontě. Výhoda je ta, že nemusíme neustále prohledávat prioritní frontu, ale na druhou stranu si musíme udržovat aktuální příznaky vrcholu. Toto udržování příznaků je ale časově mnohem méně náročné.

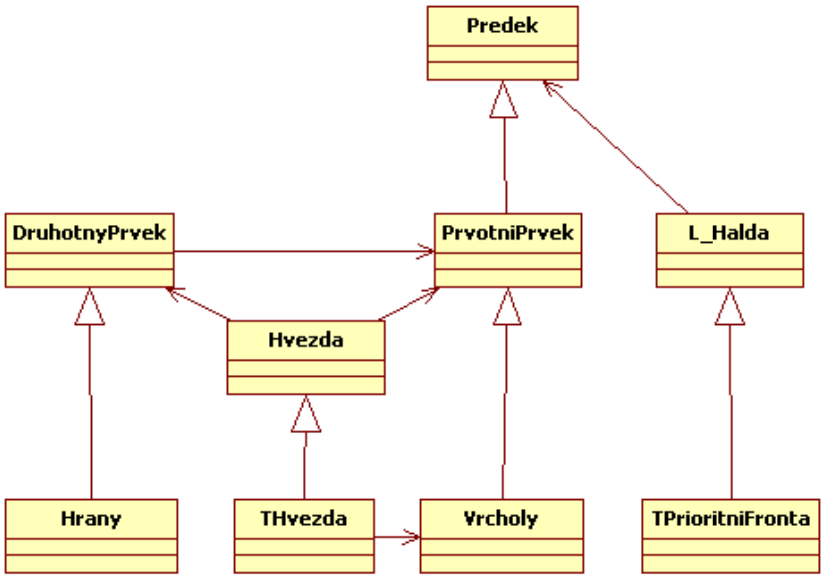
4. Návrat na bod 2

Pokud prioritní fronta stále obsahuje alespoň jeden prvek, tak se vracíme na bod 2. Pokud už je tato fronta prázdná, tak algoritmus končí.

6.2. UML diagramy

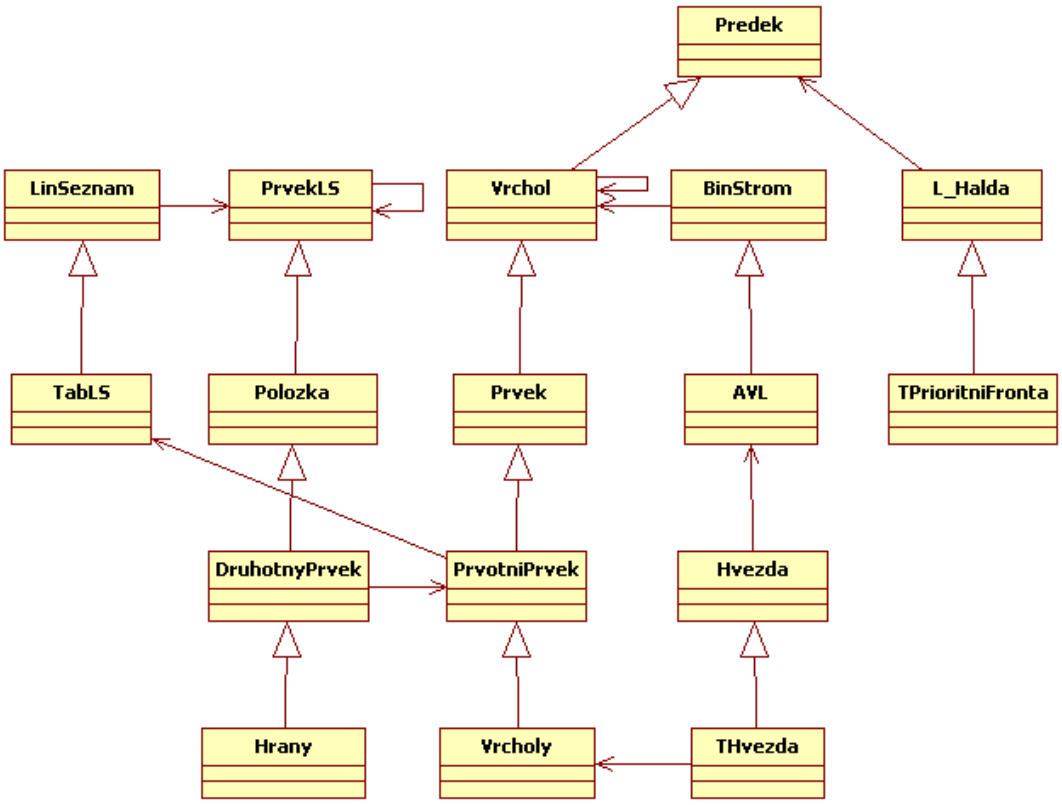
Aby bylo možné názorně ukázat spojitosti (vazby) mezi jednotlivými strukturami, je vhodné využít nějakého diagramu. Pro tento účel jsem zvolil UML digramy. Kvůli rozsáhlosti původních diagramů, které byly exportovány přímo ze zdrojových kódů, jsem vytvořil zjednodušené diagramy, které obsahují pouze názvy tříd. Ze zjednodušených diagramů jsou ale stále patrná jednotlivá provázání datových struktur. Původní UML diagramy jsou uvedeny v příloze a na přiloženém CD.

Statická verze



Obrázek 11 - Zjednodušený UML diagram statické verze

Dynamická verze



Obrázek 12 - Zjednodušený UML diagram dynamické verze

6.3. Grafické zobrazení

Aby bylo možno snáze sledovat a pochopit postupný výpočet algoritmu, je třeba určitý způsobem zobrazovat jednotlivé kroky algoritmu. Je také zapotřebí nějakým způsobem odlišovat (např. barevně) různé situace, do kterých se můžeme s algoritmem dostat.

6.3.1. Nároky na grafické zobrazení

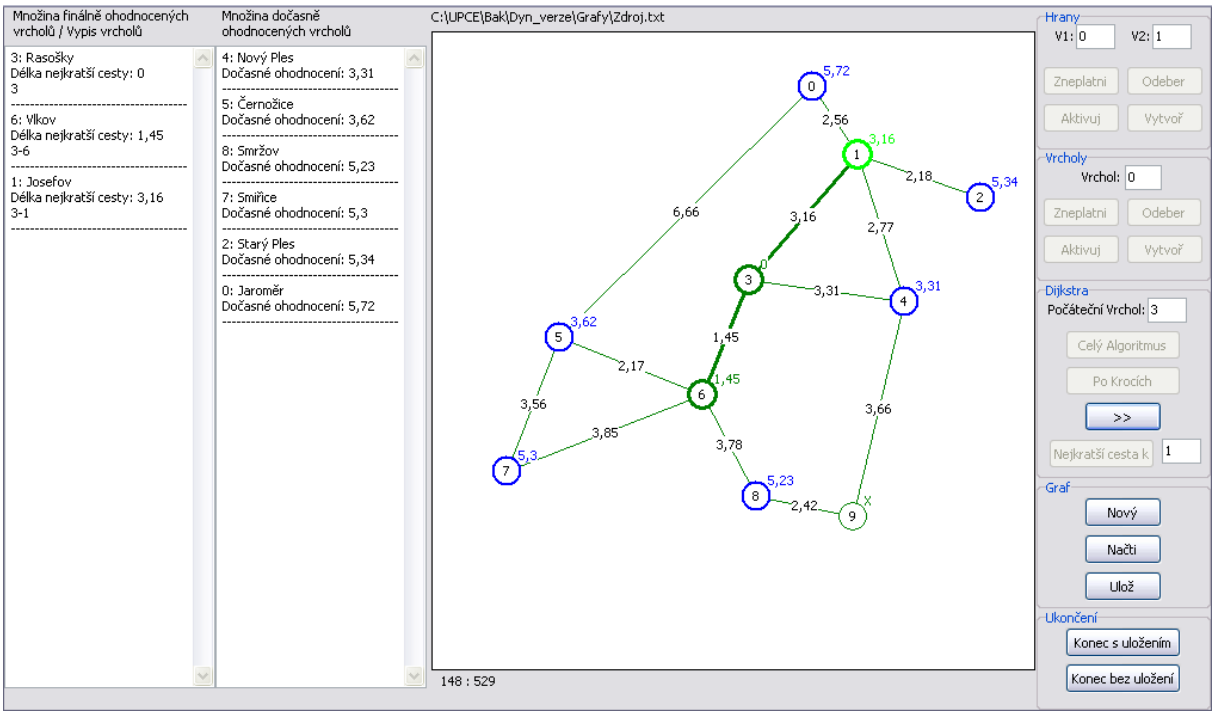
V prvé řadě je samozřejmě třeba zobrazit samotný graf. U tohoto zobrazení potřebujeme od sebe rozeznat jednotlivé vrcholy tj. přidat k jednotlivým vrcholům identifikátory nebo přímo nějaké názvy. Na hranách by mělo být poznat, jaké mají ohodnocení.

Veškeré změny, které provedeme, např. přidání vrcholu nebo hrany by se měli okamžitě zobrazit.

Při krokování algoritmu bychom měli vidět aktuální ohodnocení všech vrcholů. Dál je třeba určitým způsobem odlišit vrcholy, které jsme dosud nenavštívili, které mají zatím jen dočasné ohodnocení, které už jsou trvale ohodnoceny a vrchol který jsme právě prohlásily za trvale ohodnocený.

V průběhu krokování bychom měli vidět aktuální strom nejkratších cest a obsah prioritní fronty. A co se týče výsledků, tak by měly být lehce čitelné.

6.3.2. Dosažené výsledky



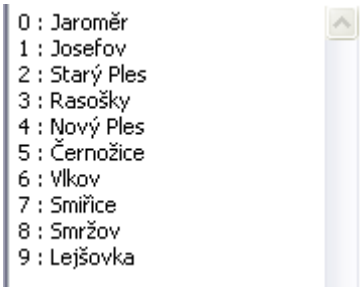
Obrázek 13 - Finální vzhled aplikace

Jak je vidět z obrázku jednotlivé vrcholy jsou označeny čísly, pomocí kterých se dá s grafem manipulovat, jak už bylo řečeno v předchozích článcích. Hrany jsou čitelně označeny jejich ohodnocením. Na obrázku můžeme dále vidět ohodnocení všech vrcholů, kromě vrcholu s označením *9*, který má místo ohodnocení pouze *X*. Je to pouze náhradní znak, který má znázorňovat ∞, tedy skutečnost, že do tohoto vrcholu se zatím nepodařilo najít cestu.

Aktuální strom nejkratších cest, odpovídající jisté fázi výpočtu, je na obrázku znázorněn tučnou tmavě zelenou barvou. Vrchol s označením *1* má světlejší barvu, protože ta značí, že jsme tento vrchol právě prohlásily za trvale ohodnocený. Modrá barva značí skutečnost, že je vrchol ohodnocen jen dočasně a jeho ohodnocení se tedy může změnit. A poslední odlišení vrcholů můžeme vidět u vrcholu *9*, do tohoto vrcholu se zatím algoritmus ještě nedostal. Je vykreslen tenkou zelenou barvou.

Aktuální obsah prioritní fronty, tedy vrcholy s dočasným ohodnocením, můžeme vidět ve výpisu s označením *Množina dočasně ohodnocených vrcholů*. Můžeme zde vidět všechny potřebné údaje o těchto vrcholech, tedy jejich označení, název a aktuální ohodnocení. Výpis je seřazen stejně jako sama prioritní fronta.

Ve výpisu vlevo s označením *Množina finálně ohodnocených vrcholů/Výpis vrcholů* můžeme už vidět výsledky algoritmu. Jak je vidět z obrázku 13, tak už jsme našli cesty ke třem vrcholům včetně počátečního. Ve výpisu můžeme vidět označení a název vrcholu, délku nejkratší cesty a označení vrcholů, přes které vede tato cesta. Při načtení grafu nebo při změnách jeho struktury se nám v tomto poli pro jednodušší orientaci vypisují jednotlivé vrcholy jejich názvy, jak můžeme vidět na obrázku 14.

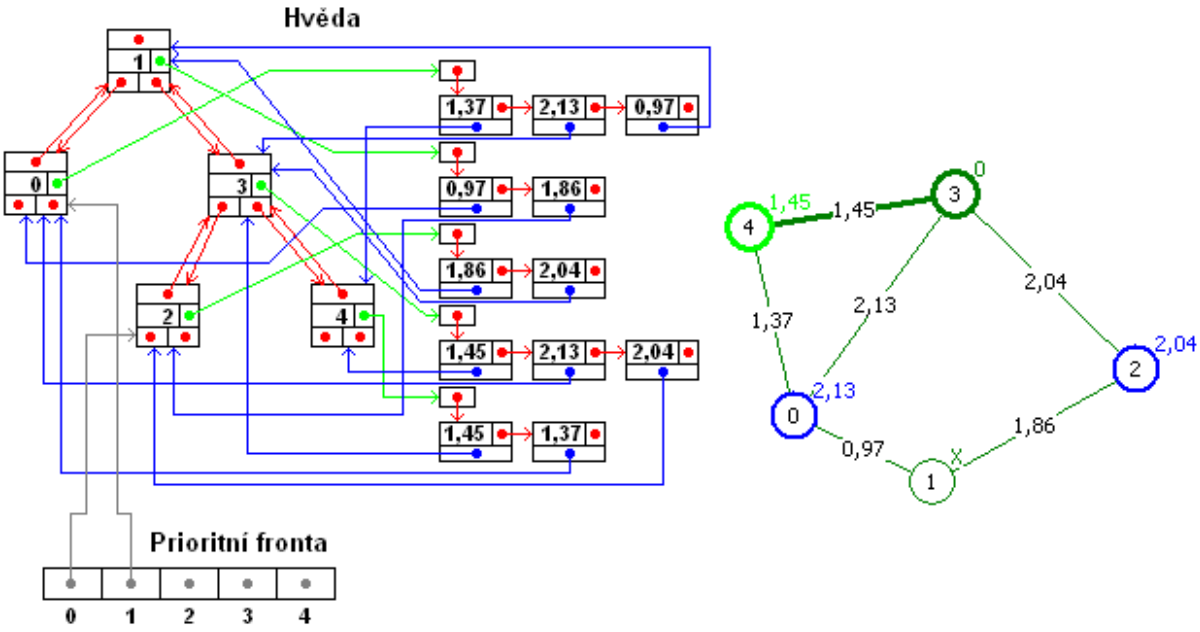


Obrázek 14 - Vypis názvů vrcholů

6.4. Ukázka datových struktur v průběhu výpočtu

Je zde ukázka aktuálně obsazených datových struktur v jisté fázi výpočtu Dijkstrova algoritmu a obrázek grafu po ukončení tohoto algoritmu. Navíc zde budou vypsány množiny dočasně ohodnocených vrcholů a množiny finálně (trvale) ohodnocených vrcholů, jednotlivé cesty a jejich délky. Počáteční vrchol výpočtu má v tomto případě označení (klíč) *3*.

V průběhu algoritmu

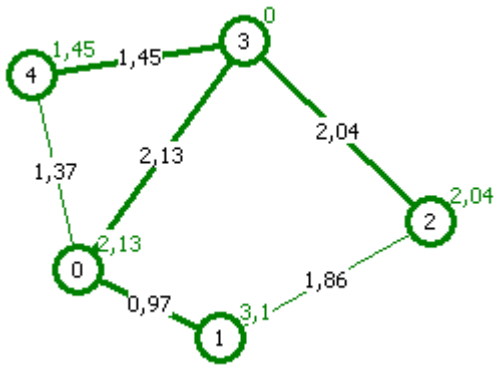


Obrázek 15 – Ukázka datových struktur v jisté fázi výpočtu algoritmu

Množina dočasně ohodnocených vrcholů: {0, 2}

Množina trvale ohodnocených vrcholů: {3, 4}

Po ukončení algoritmu



Obrázek 16 – Graf po dokončení algoritmu

Množina dočasně ohodnocených vrcholů: {}

Množina trvale ohodnocených vrcholů: {0, 1, 2, 3, 4}

Jednotlivé cesty:	0	3→0	**Délka:** 2,13
	1	3→0→1	**Délka:** 3,1
	2	3→2	**Délka:** 2,04
	3	3	**Délka:** 0
	4	3→4	**Délka:** 1,45

6.5. Testování

Aplikace byla testována na dvou různých grafech. Na prvním z nich je v kapitole 6.3.2 ukázán konečný vzhled aplikace. Jedná se o malý úsek v Královehradeckém kraji a je určen pro testování krokování. Pro testování aplikace na netriviálním grafu je určen druhý graf, který můžeme i s podkladovou mapou vidět v příloze B.

7. Závěr

Na závěr bych shrnul dosažené výsledky. Povedlo se vytvořit dvě rozdílné abstraktní datové struktury přesně podle předpokladů. Obě struktury podporují všechny potřebné operace na grafech. Aktuální obsazenost dopravních sítí zajišťují funkce zneplatnění a aktivace vrcholů a hran.

Všechny požadavky na zobrazení grafu a různá odlišení vrcholů a hran při krokování jsou splněna. Je tu jen drobný problém v tom, že se nijak nedá zobrazovat část grafu, která sahá mimo zobrazovací plochu. To je řešeno tím, že se zobrazují jen grafy, které mají 20 a méně vrcholů. Menší grafy, pro které je aplikace primárně určena, se do zobrazovací oblasti pohodlně vejdou. Bez ohledu na vykreslení grafu, můžeme Dijkstrův algoritmus spustit vždy a potřebné výsledky se vypíší do patřičného pole.

Co se týče ukládání, veškeré údaje o grafech se ukládají do textových souborů.

Aplikace je jinak bez problémů spustitelná a připravena na použití.

Seznam obrázků

OBRÁZEK 1 – DOPŘEDNĚ-ZPĚTNÁ HVĚZDA 16
OBRÁZEK 2 – KŘÍŽOVÁ REPREZENTACE 17
OBRÁZEK 3 - TABULKA HRAN 18
OBRÁZEK 4 – POLE HRAN SE SEZNAMY SOUSEDNOSTI 19
OBRÁZEK 5 – PŘÍSTUP K VYBRANÉMU VRCHOLU 20
OBRÁZEK 6 – PŘÍSTUP K SOUSEDNÍM VRCHOLŮM 21
OBRÁZEK 7 – PRŮCHOD MEZI SOUSEDNÍMI VRCHOLY 21
OBRÁZEK 8 – PŘÍKLAD NAVRŽENÉ STATICKÉ VERZE HVĚZDY 22
OBRÁZEK 9 – PŘÍKLAD NAVRŽENÉ DYNAMICKÉ VERZE HVĚZDY 24
OBRÁZEK 10 – HALDY [3] 25
OBRÁZEK 11 - ZJEDNODUŠENÝ UML DIAGRAM STATICKÉ VERZE 28
OBRÁZEK 12 - ZJEDNODUŠENÝ UML DIAGRAM DYNAMICKÉ VERZE 28
OBRÁZEK 13 - FINÁLNÍ VZHLED APLIKACE 30
OBRÁZEK 15 – UKÁZKA DATOVÝCH STRUKTUR V JISTÉ FÁZI VÝPOČTU ALGORITMU 31
OBRÁZEK 14 - VYPIS NÁZVŮ VRCHOLŮ 31
OBRÁZEK 16 – GRAF PO DOKONČENÍ ALGORITMU 32
OBRÁZEK 17 - PANEL GRAF 36
OBRÁZEK 18 - PANEL UKONČENÍ 36
OBRÁZEK 19 - PANEL HRANY 36
OBRÁZEK 20 - PANEL VRCHOLY 37
OBRÁZEK 21 - PANEL DIJKSTRA 37
OBRÁZEK 22 – TESTOVACÍ NETRIVIÁLNÍ GRAF 38
OBRÁZEK 23 – KOMPLETNÍ UML DIAGRAM STATICKÉ VERZE 39
OBRÁZEK 24 – KOMPLETNÍ UML DIAGRAM DYNAMICKÉ VERZE 40

Seznam tabulek

TABULKA 1 – ADT PRIORITNÍ FRONTA 10
TABULKA 2 – ADT TABULKA 10
TABULKA 3 – SUPER ADT GRAF 11

Seznam příloh

A. OVLÁDACÍ PRVKY APLIKACE 36
B. TESTOVACÍ NETRIVIÁLNÍ GRAF 38
C. UML DIAGRAM STATICKÉ VERZE 39
D. UML DIAGRAM DYNAMICKÉ VERZE 40

Zdroje

[1] CENEK, Petr, KLIMA, Valent, JANÁČEK, Jaroslav. *Optimalizace dopravních a spojových procesů*. Žilina : Vysoká škola dopravy a spojov, 1994. 343 s. ISBN 80-7100-197-X.

[2] VOLEK, Josef. *Operační výzkum I*. Pardubice : Univerzita Pardubice, 2002. 111 s. ISBN 80-7194-410-6.

[3] KAVIČKA, Antonín. *Datové struktury*. Elektronické sylaby přednášek předmětu Datové struktury. 2007.

[4] HOKSZA, David. *Nejkratší cesty v ohodnoceném grafu* [online]. 2002 [cit. 2009-06-20]. Dostupný z WWW: <http://reboot.cz/howto/programovani/nejkratsi-cesta-v-ohodnocenem-grafu/articles.html?id=223>.

[5] CANTÚ, Marco. *Myslíme v jazyku Delphi 7*. [s.l.] : [s.n.], 2003. 580 s. ISBN 80-247-0694-6.

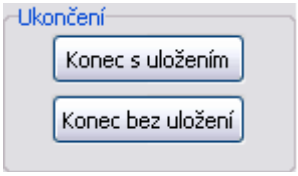
Příloha

A. Ovládací prvky aplikace

Celou práci jsem vytvářel v prostředí Turbo Delphi 2006, takže tlačítka a ostatní části grafiky jsou vytvořeny v tomto prostředí.

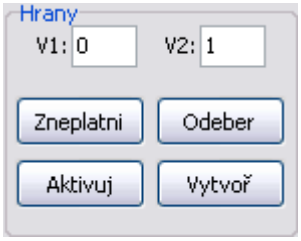
Obrázek 17 - Panel Graf

Tlačítkem *Nový* vytvoříme nový prázdný graf. Pokud jsme měli už nějaký jiný graf otevřený, otevře se dialogové okno s dotazem na uložení předchozího grafu. Tlačítkem *Načti* otevřeme dialog pro výběr souboru s grafem. Opět se nám, stejně jako u předchozího tlačítka, nejprve otevře dialog s nabídkou uložení změn předešlého grafu, pokud jsme měli nějaký otevřený. Pomocí *Ulož* otevřeme opět dialog, kde si vybereme umístění a název souboru, který bude obsahovat aktuální graf.



Obrázek 18 - Panel Ukončení

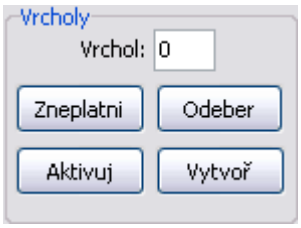
Aplikaci můžeme bezpečně ukončit dvěma tlačítky. Jak je vidět z obrázku vpravo, rozdíl je jen v tom jestli si chceme aktuální graf, na kterém jsme mohli udělat nějaké změny, uložit. *Konec s uložením* je aktivní jen v případě, že je aktuální graf již uložený. V tom případě je známo jméno a umístění souboru s grafem a není problém tento soubor přepsat.



Obrázek 19 - Panel Hrany

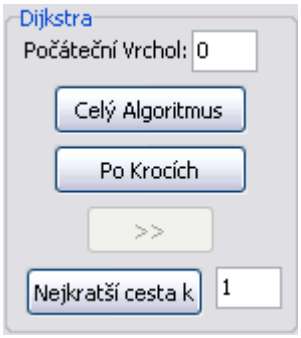
Pro manipulaci s hranami slouží panel *Hrany*. *V1* a *V2* jsou označení dvou vrcholů, mezi kterými se hrana, se kterou chceme pracovat, nachází. Tlačítko *Odeber* hranu kompletně odstraní z grafu. Tlačítkem *Zneplatni* označenou hranu pouze zneplatníme, takže graf tuto hranu stále obsahuje, ale pro výpočet algoritmu se jeví jako neviditelná. Tlačítkem *Aktivuj* vrátíme zneplatněnou hranu opět do původního stavu. Pomocí *Vytvoř* jsme schopni vytvořit a vložit do grafu novou hranu. Je to jediné tlačítko v tomto panelu, které nevyužívá označení hrany pomocí *V1* a *V2*, ale otevře se nám nové okno, kde doplníme ohodnocení hrany a dva vrcholy, mezi kterými potřebujeme novou hranu vytvořit.

Podobně jako pro manipulaci s hranami máme i pro práci s vrcholy samostatný panel. *Vrchol* v tomto panelu označuje vrchol, se kterým pracujeme. Tlačítka *Odeber*, *Zneplatni* a *Aktivuj* mají stejnou funkci jako u hran jen s tím rozdílem, že manipulujeme s vrcholy. Tyto tři tlačítka nemají vliv pouze na vrcholy, ale mají stejný efekt i na incidentní hrany. Například odebráním vrcholu odebereme i incidentní hrany, protože by neměli dva vrcholy, mezi kterými by mohli existovat. Tlačítkem *Vytvoř* můžeme vytvořit nový vrchol. Po jeho stisknutí se nám objeví nové okno, kam zadáme název a souřadnice nového vrcholu. Souřadnice si můžeme zjistit pouhým přejetím myši přes zobrazovací prostor grafu. Aktuální souřadnice myši se nám zobrazují pod oknem s grafem.



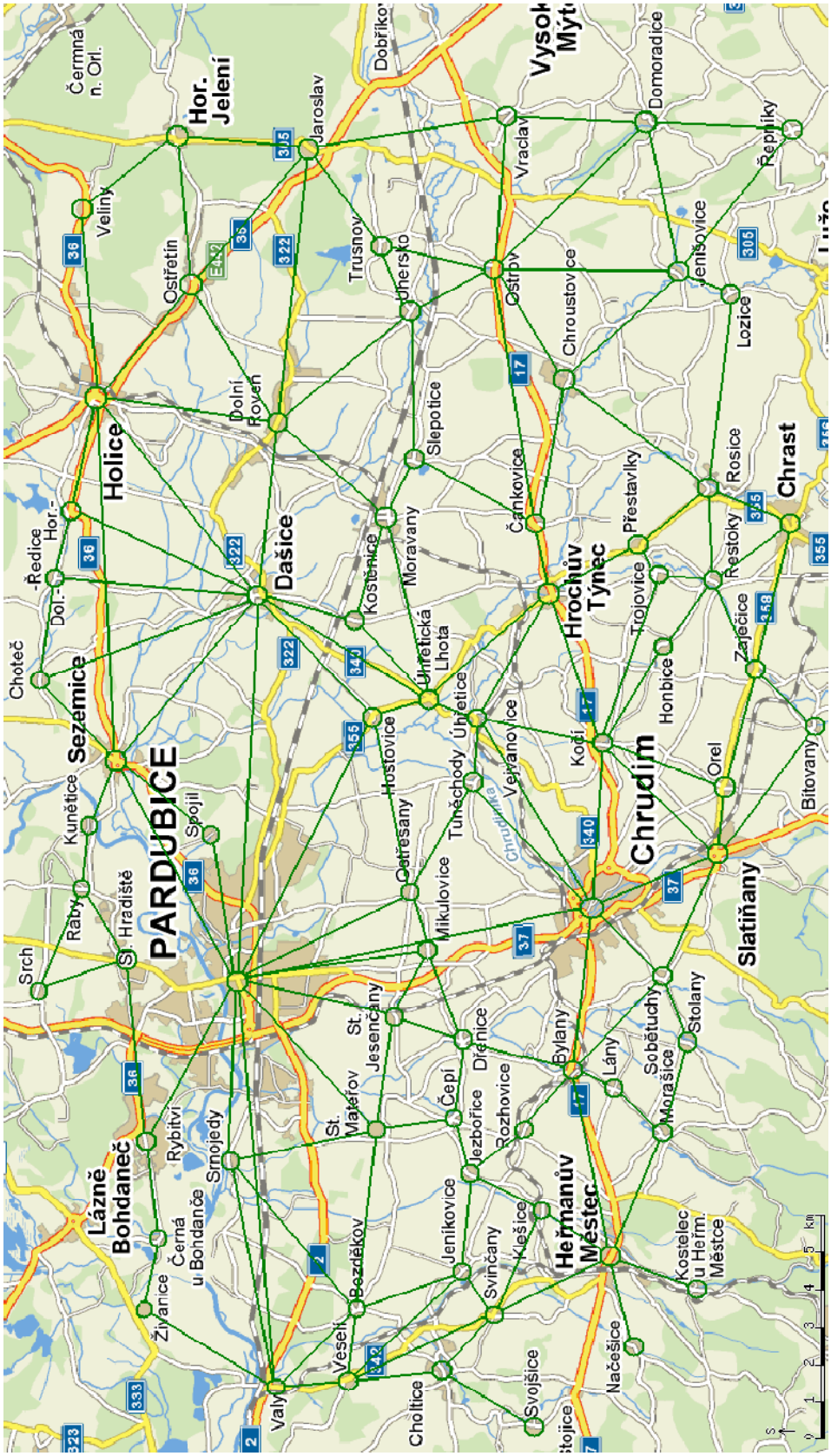
Obrázek 20 - Panel Vrcholy

Spuštění algoritmu se věnuje panel *Dijkstra*. *Počáteční vrchol* označuje vrchol, ze kterého se bude algoritmus rozvíjet. Tlačítkem *Celý Algoritmus* spustíme ze zadaného vrcholu algoritmus bez krokování. Pomocí tlačítka *Po Krocích* můžeme algoritmus spustit tak, že jsme potom schopni tlačítkem >> procházet jednotlivé kroky algoritmu. A nakonec tlačítko *Nejkratší cesta k* nám spustí algoritmus k vyhledání pouze jedné nejkratší cesty k vrcholu, který je zadán vpravo od tohoto tlačítka.



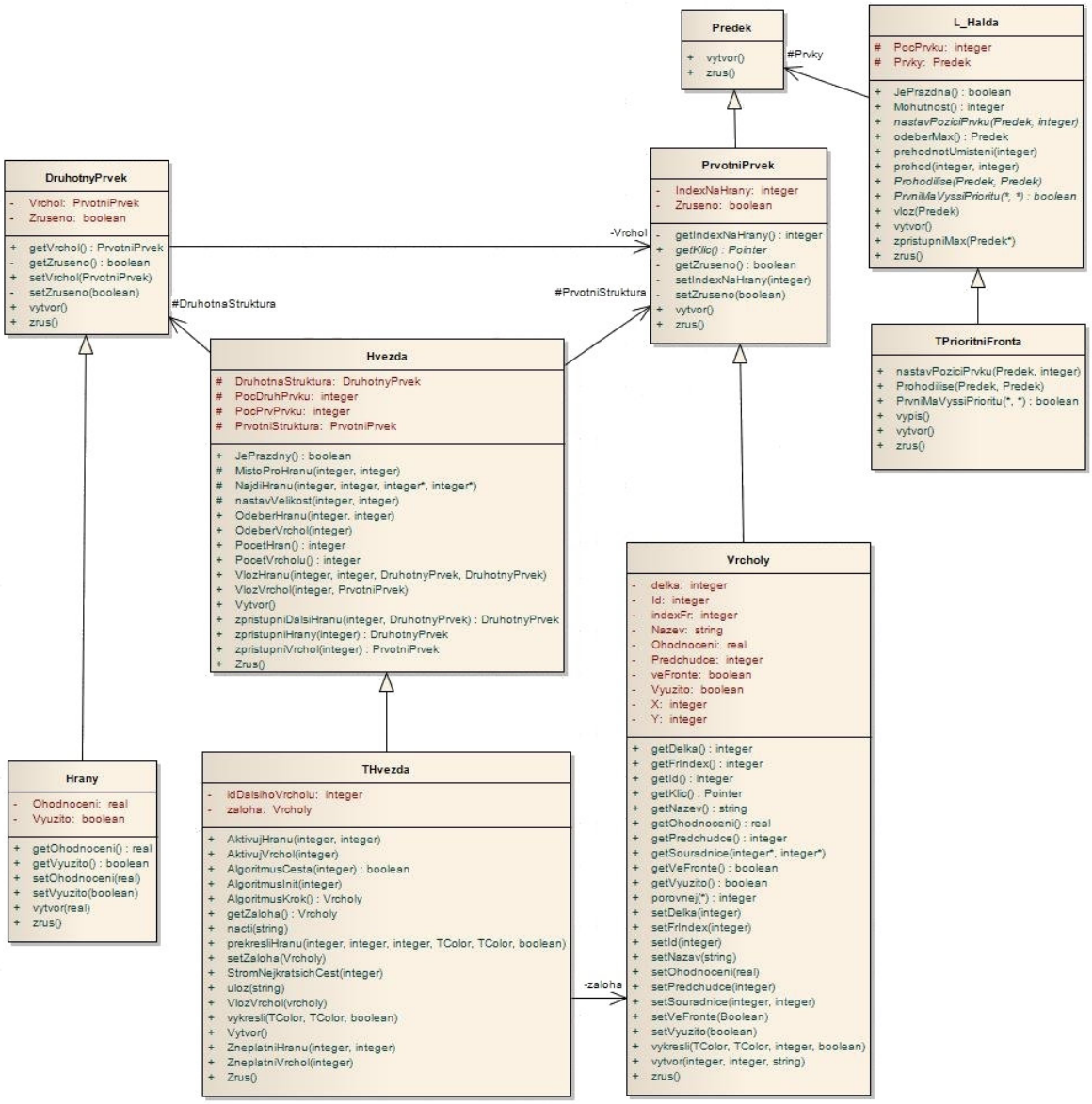
Obrázek 21 - Panel Dijkstra

B. Testovací netriviální graf



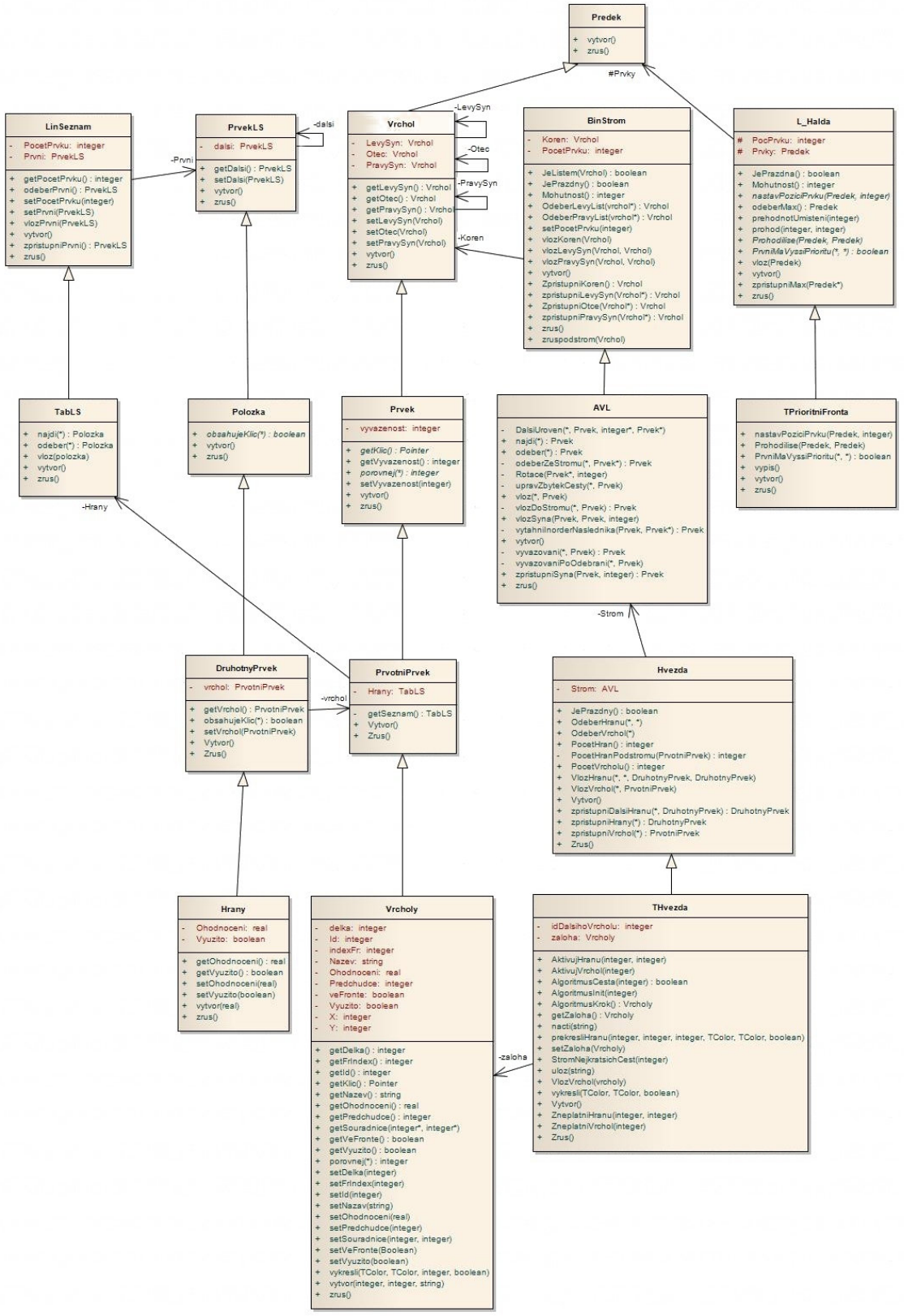
Obrázek 22 – Testovací netriviální graf

C. UML diagram statické verze



Obrázek 23 – kompletní UML diagram statické verze

D. UML diagram dynamické verze



Obrázek 24 – Kompletní UML diagram dynamické verze