Univerzita Pardubice

Dopravní fakulta Jana Pernera

Aplikace využívající komponentu grafu

Milan Gatyás

Bakalářská práce

2013

Univerzita Pardubice
Dopravní fakulta Jana Pernera
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Milan Gatyás**
Osobní číslo:	**D10842**
Studijní program:	**B3709 Dopravní technologie a spoje**
Studijní obor:	**Aplikovaná informatika v dopravě**
Název tématu:	**Aplikace využívající komponentu grafu**
Zadávající katedra:	**Katedra informatiky v dopravě**

Zásady pro vypracování:

Popis paměťových reprezentací grafu. Návrh a vývoj komponenty umožňující zobrazení a úpravu grafu. Návrh a vývoj ukázkové aplikace používající uvedenou komponentu. Aplikace bude vyvinuta v programovacím jazyku C#.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30 normostran**

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

Seznam odborné literatury:

1. **NAGEL, Christian, Jay GLYNN, Karli WATSON a Morgan SKINNER. C 2008: programujeme profesionálně. Vyd. 1. Brno: Computer Press, 2009, 772 s. Programujeme profesionálně. ISBN 978-80-251-2401-7.**
2. **WIRTH, Niklaus. Algoritmy a štruktúry údajov. 2. vyd. Překlad Pavol Fischer. Bratislava: Alfa, 1989, 481 s. Edícia výpočtovej techniky. ISBN 80-050-0153-3.**
3. **CENEK, Petr, Valent KLIMA a Jaroslav JANÁČEK. Optimalizace dopravních a spojových procesů. 1. vyd. Žilina: Vysoká škola dopravy a spojov, 1994, vi, 343 s. ISBN 80-7100-197-X.**

Vedoucí bakalářské práce: **Ing. Karel Greiner, Ph.D.**
Katedra informatiky v dopravě

Datum zadání bakalářské práce: **6. prosince 2012**

Termín odevzdání bakalářské práce: **31. května 2013**

prof. Ing. Bohumil Culek, CSc.
děkan

L.S.

doc. Ing. Josef Volek, CSc.
vedoucí katedry

V Pardubicích dne 6. prosince 2012

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 Kolíně dne 01. 03. 2013

Milan Gatyás

Poděkování

Rád bych poděkoval panu Ing. Karlovi Greinerovi, Ph. D., za jeho cenné rady a pevné nervy při vypracovávání této práce.

Tato bakalářská práce vznikla v rámci řešení projektu „Podpora stáží a odborných aktivit při inovaci oblasti terciárního vzdělávání na DFJP a FEI Univerzity Pardubice, reg. č.: CZ.1.07/2.4.00/17.0107“, v týmu „Informační systémy v železniční dopravě“.

Anotace

Předmětem bakalářské práce je návrh a následný vývoj aplikační komponenty, která umožňuje zobrazování a úpravu obyčejného grafu. Dále návrh a vývoj ukázkové aplikace, která bude tuto komponentu využívat a obohacovat o praktické využití.

Teoretická část práce se zabývá popisem možných paměťových reprezentací grafu, praktická část potom vlastní implementací komponenty a ukázkové aplikace.

Tato práce je vyvinuta v programovacím jazyku C# na platformě .NET 4.0, komponenta využívá technologii Windows Forms.

Klíčová slova

Informatika, datové struktury, teorie grafů, aplikační komponenty, C#.

Title

Application using graph component.

Annotation

The subject of the bachelor thesis is the suggestion and development of an application component which enables to show and edit simple graph. The further task is development of an example application which will demonstrate functionality of component and enrich the whole project.

Theoretical part of the thesis describes possibilities of memory representation of the graph. Practical part analyzes actual implementation of component and example application.

This project is developed on .NET platform in C# programming language. Application component uses Windows Forms technology.

Keywords

Informatics, data structures, graph theory, application components, C#.

Obsah

Seznam obrázků a tabulek

Seznam zkratek

ADS – Abstraktní datová struktura

ADT – Abstraktní datový typ

BVS – Binární vyhledávací strom

AVL – Samovyvažující se binární vyhledávací strom pojmenovaný po G. M. Andelson - Velskiim a E. M. Landisem

SS – Souřadný systém

UML – Unified Modeling Language, jazyk vytvořen pro efektivní objektové modelování

GoF – Gang of Four. Skupina sestávající z členů Richard Helm, Erich Gamma, Ralph Johnson a John Vlissides. Vydali vysoce ceněnou publikaci pojednávající o návrhových vzorech.

0 Úvod

Cílem bakalářské práce je popsat možné paměťové reprezentace grafu, vytvořit aplikační komponentu v prostředí .NET, vytvořenou v programovacím jazyku C#, která umožní interaktivní zobrazení a úpravu obyčejného grafu, a vytvoření ukázkové aplikace, která bude funkčnost této komponenty demonstrovat.

Práce je rozdělená do dílčích bloků, a to do bloků analýzy problematiky, praktické části a závěrečného shrnutí.

Komponenta s ukázkovou aplikací může také vzhledem ke své povaze sloužit k podpoře výuky metod z oblasti teorie grafů.

1 Analýza problematiky

1.1 Reprezentace grafu

1.1.1 Základní pojmy z oblasti teorie grafů

Před představením jednotlivých paměťových reprezentací grafu v počítači je potřeba osvojit si základní poznatky z oblasti *teorie grafů,* které jsou důležité z hlediska dalšího zkoumání.

Mějme dvě libovolné disjunktní množiny V, X. a $p:X \rightarrow V \times V$. Grafem nazýváme uspořádanou trojici $G = (V, X, p)$, kde prvky množiny V nazýváme vrcholy, prvky množiny X hranami grafu G a zobrazení množiny X na všechny množiny dvojic $V \times V$ incidencí grafu p. Mohutnost množiny V ($|V|$) nazýváme počtem vrcholů grafu, mohutnost množiny X ($|X|$) potom počtem hran grafu.

Jestliže platí $p(h) = (u, v) = (v, u)$, pro $u, v \in V$, pak hranu h nazýváme neorientovanou hranou. O uvedených vrcholech u, v říkáme, že jsou krajní vrcholy hrany h, nebo že jsou s hranou h incidentní (a obráceně hrana h je incidentní s vrcholy u a v). Pro vrcholy u a v můžeme také použít termín sousední vrcholy (pokud $u \neq v$).

Pokud platí $p(h) = [u, v] \neq [v, u]$, pro $u, v \in V$, pak hranu h nazýváme hranou orientovanou. Vrchol u nazýváme počátečním vrcholem a vrchol v koncovým vrcholem. Vrchol u představuje předchůdce vrcholu v a vrchol v představuje následníka vrcholu u.

Graf, který obsahuje pouze neorientované hrany, se nazývá grafem *neorientovaným*, graf, který obsahuje pouze orientované hrany grafem *orientovaným*, a konečně graf, který obsahuje oba typy hran grafem *smíšeným*. V této práci bude využit graf *neorientovaný*.

Jsou-li vrcholy $u, v \in V$ spojeny více hranami, nazýváme tyto hrany *násobné*. Pokud incidenční vrcholy hrany tvoří identické vrcholy, příslušnou hranu nazýváme *smyčkou*. Graf, který obsahuje násobné hrany a smyčky nazýváme *multigrafem*. Graf, který neobsahuje násobné hrany, ale obsahuje smyčky, nazýváme grafem *prostým*. Graf, který neobsahuje smyčky, ani násobné hrany, nazýváme grafem *obyčejným*. V této práci budeme pracovat s grafem *obyčejným*[1].

Má-li každý vrchol grafu z množiny V přiřazené reálné číslo o určité hodnotě, jedná se o *vrcholově ohodnocený graf*. Má-li každá hrana grafu z množiny X přiřazené reálné číslo o určité hodnotě, jedná se o *hranově ohodnocený graf*. Obsahuje-li graf oba typy ohodnocení, jedná se o *ohodnocený graf* [1][2].

[1] Tento rozbor definic grafu není zcela vyčerpávající. Pro další informace viz [1].

1.1.2 Klasifikace abstraktních datových struktur

Graf lze v paměti počítače reprezentovat mnoha způsoby. Jednotlivé způsoby se liší především podle uvažovaného využití grafu. Předpisem pro všechny reprezentace je *abstraktní datový typ Graf* (*ADT Graf)*.

Pojmem *ADT* rozumíme předpis, určující prvky a metody, které má datový typ obsahovat. Tento předpis vlastně určuje, jak se typ reprezentuje svému okolí. Základní metody definované pro *ADT Graf* jsou

- vložení vrcholu/hrany,
- odstranění vrcholu/hrany,
- nalezení vrcholu/hrany,
- zpřístupnění kolekce vrcholů,
- zpřístupnění kolekce hran,
- zpřístupnění sousedů vrcholu.

ADT nepředepisuje konkrétní paměťové reprezentace prvků typu, stejně tak konkrétní implementace metod. K tomuto účelu slouží *abstraktní datová struktura (ADS).*

ADS představuje jednu konkrétní implementaci *ADT*. Je to abstraktní implementace, nezávislá na datech, která bude obsahovat.

Z hlediska datových struktur lze k abstraktnímu datovému typu Graf přistupovat z různých pohledů.

- *Vrcholově statické struktury*
  Operace vložení a odstranění vrcholu nejsou realizovatelné, nebo jejich složitost není menší než $O(n)$. Vhodné pro relativně neměnné grafy vzhledem k vrcholům.
- *Vrcholově dynamické struktury*
  Jsou vhodné pro práci s grafem zejména vzhledem k vrcholům. Výhodné použít při časté modifikaci struktury vrcholů grafu.
- *Hranově statické struktury*
  Operace vložení a odstranění hrany nejsou realizovatelné, nebo jejich složitost není menší než $O(n)$. Vhodné pro málo modifikované grafy vzhledem k hranám.
- *Hranově dynamické struktury*
  Jsou vhodné pro práci s grafem zejména vzhledem k hranám. Umožňuje modifikaci struktury hran s příznivou složitostí.

1.1.3 Implementace abstraktních datových struktur

Implementace grafu lze rozdělit do dvou skupin dle přístupu k prvkům.

- *Vrcholově orientovaný přístup*

  Vstupní bránou do datové struktury je vrchol, s nímž jsou spojeny další informace o sousedních vrcholech a incidenčních hranách. Při vyhledávání ve struktuře jsou přímo k dispozici informace o sousedních vrcholech, a tím i o incidenčních hranách.

- *Hranově orientovaný přístup*

  Vyhledávací operace jsou orientovány na vyhledávání hran. Zpřístupnění vrcholů je druhotné, odvozené z údajů o hraně. Nevýhodou tohoto přístupu je, že většina grafových algoritmů je navržených pro *vrcholově orientovaný přístup*.

Nejčastějším způsobem realizace vrcholového přístupu je docíleno pomocí *hvězd*. Hvězda představuje kompozitní datovou strukturu sestávající z prvotní struktury uchovávající vrcholy grafu a druhotné struktury, která zachovává relace těchto vrcholů s ostatními vrcholy grafu. Obecně jsou hvězdy určeny primárně pro uchovávání orientovaných grafů[2].

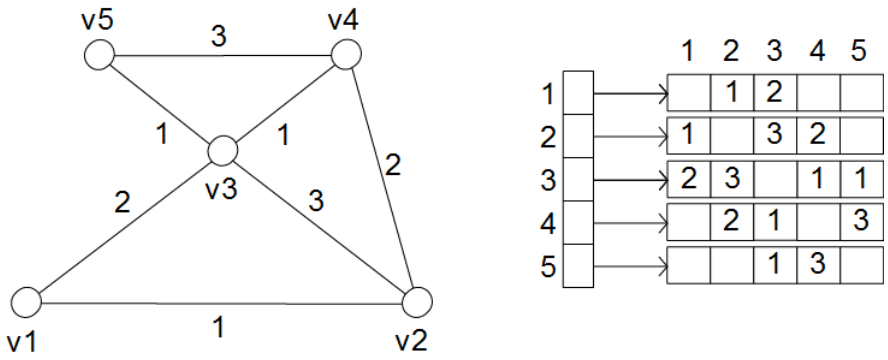
Prvotní, resp. druhotnou strukturu lze reprezentovat buď pomocí *pole* (vrcholy jsou identifikovány pomocí prvních $|V|$ přirozených čísel), nebo tabulkou (vrcholy identifikovány klíčovou položkou). Ohodnocení vrcholů/hran je určitým způsobem přidáno k příslušné implementaci vrcholu/hrany (např. složeným datovým typem).

Hvězdy, jejichž druhotná struktura je reprezentována pomocí ADT Tabulka, podporují implementace *uspořádaného* a *neuspořádaného* grafu. Pro *uspořádaný graf* platí, že sousední vrcholy libovolného vrcholu $v_i \in V$ představují lineárně uspořádanou množinu, tj. každý ze sousedních vrcholů může být jednoznačně určen jako první, druhý, …, *n*-tý sousední vrchol.

První z možných reprezentací hvězdy je použití struktur *pole – pole*. Výhodou této reprezentace je rychlý přístup k vrcholům i hranám se složitostí O(1). Nevýhodou je rezervování míst i pro ještě neexistující hrany ($|V|^2$ míst), což má za následek zhoršení složitosti pro vyhledávací operace na složitost $O(|V|)$. Na druhou stranu tato rezervace má za následek, že složitost vkládání a odebírání hran je $O(1)$. Tato paměťová reprezentace je vhodná jak pro orientované, tak pro neorientované grafy. Rozměr matice tvořené poli je $|V|$ x $|V|$, hlavní diagonála matice je u neobyčejného grafu nevyužita (obyčejný graf nepodporuje smyčky). Matice je souměrná podle hlavní diagonály, jelikož platí, že $p(h) = (u, v) = (v, u)$ (u neorientovaného grafu je možné duplicitní informace buď nerezervovat, nebo je využít pro jiné informace). Tato

[2] Více informací viz [3].

reprezentace je vrcholově statická a hranově dynamická. Ukázka reprezentace na obyčejném grafu viz obrázek 1.

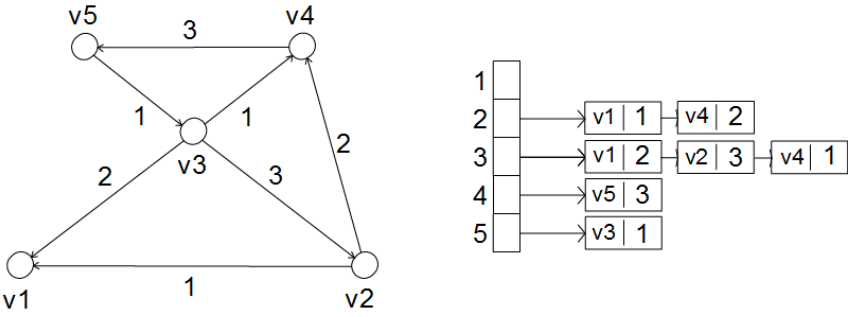


Obrázek 1 Reprezentace hvězdy typu pole – pole

Možné zefektivnění této reprezentace spočívá v uchovávání pouze skutečně existujících sousedů vrcholů. Potom ale již není možné záznam o sousedovi identifikovat indexem, nýbrž je nutné přejít k identifikaci dle klíče. Následující reprezentací je tedy struktura typu *pole – tabulka*.

Reprezentace primární struktury je zde stejná jako u reprezentace typu *pole – pole*, otázkou je tedy jak reprezentovat sekundární strukturu ($|V|$ tabulek sousedů vrcholů). *ADT tabulka* je možné reprezentovat více způsoby (na poli, seznamu, binárním stromě, …). Charakteristickými vlastnostmi sekundární struktury jsou zpřístupňování sousedů vždy ve stejném pořadí, malá velikost množiny sousedů, postupné procházení celé struktury v grafových algoritmech. Těmto požadavkům nejvíce vyhovuje tabulka implementovaná *seznamem*.

Tato paměťová reprezentace je vhodná především pro orientované grafy. Pro grafy neorientované je použitelná, ovšem existují přijatelnější reprezentace z pohledu složitosti operací. Složitost operace nalezení vrcholu je $O(1)$, složitost operací vložení/nalezení hrany, nalezení následníků vrcholu $O(S)$, kde S je počet následníků vrcholu v. Operace nalezení předchůdců není proveditelná s rozumnou složitostí (společný znak všech dopředných hvězd). Tato reprezentace je vrcholově statická a hranově dynamická. Ukázka reprezentace na orientovaném grafu viz obrázek 2.



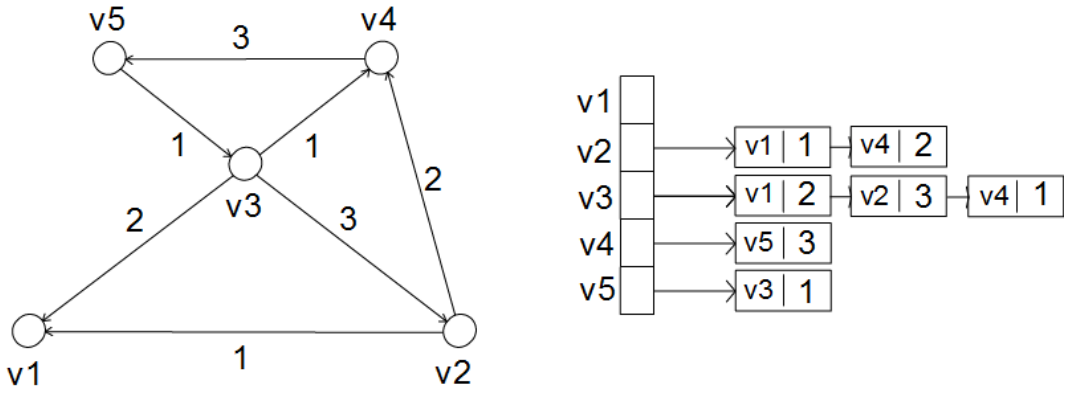
Obrázek 2 Reprezentace hvězdy typu pole – tabulka (seznam)

Někdy se může pevné očíslování vrcholů grafu za účelem uchování v primární struktuře projevovat jako nevýhodné. V takovém případě musíme přistoupit k reprezentaci primární struktury formou tabulky. Tím se dostáváme k reprezentaci hvězdy strukturami *tabulka – pole* a *tabulka – tabulka*.

Implementace grafu dvojicí struktur *tabulka – pole* nemá praktický význam. Indexace vrcholů v sekundární struktuře by bylo proti našemu původnímu záměru vypustit indexaci.

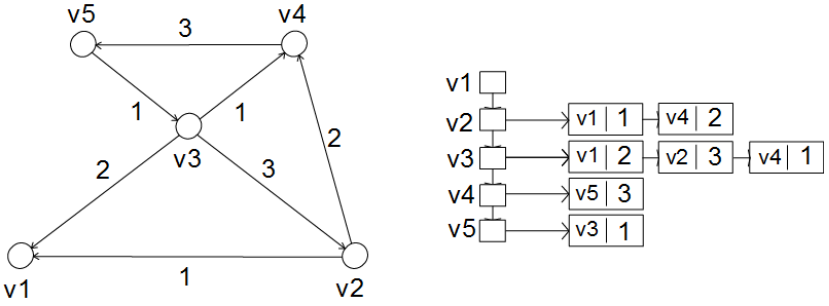
Sekundární struktura v této implementaci zůstává nezměněna. Pozornost tedy soustředíme na strukturu primární.

První implementace primární tabulky je implementace *polem*. Oproti reprezentaci typu *pole – pole* (*tabulka*) zde dochází k zhoršení složitosti při vyhledávaní vrcholů. Za cenu vypuštění indexace se zde při hledání vrcholů dostáváme ze složitosti $O(1)$ k složitostem $O(|V|)$ (neutříděné pole) nebo $O(log_2(|V|))$ (utříděné pole, aplikované binární vyhledávání). To má také dopad pro metody vkládání vrcholu $O(1)$ (neutříděné pole, vrcholově dynamická reprezentace) a $O(|V|)$ (utříděné pole, vrcholově statická reprezentace). Ilustrace reprezentace viz obrázek 3.



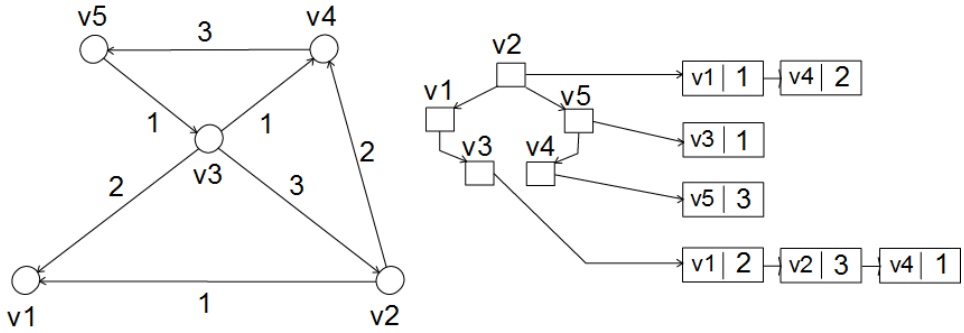
Obrázek 3 Reprezentace hvězdy typu tabulka (pole) – tabulka (seznam)

Prvotní strukturu je dále možné prezentovat pomocí *seznamu*. V tom případě se dostáváme k reprezentaci *seznam – seznam*. Výhodou této volby je dynamičnost prvotní struktury oproti *poli*, nevýhodou je nemožnost implementace operace nalezení vrcholu s menší složitostí než $O(|V|)$. Grafické znázornění reprezentace viz obrázek 4.



Obrázek 4 Reprezentace hvězdy typu tabulka (seznam) - tabulka (seznam)

Pokud je náš graf rozsáhlejší z hlediska počtu vrcholů, je vhodnější reprezentovat primární tabulku pomocí *binárního vyhledávacího stromu*[3]. *BVS* má složitost vkládání i vyhledávání prvků průměrně $O(log_2(|V|))$, což je oproti seznamu výhodnější. Nevýhodou *BVS* je jeho možné zdegenerování, jelikož se strom sám nevyvažuje. Tento problém je možné řešit nahrazením *BVS* např. *AVL stromem*[4]. Ilustrace této implementace viz obrázek 5.

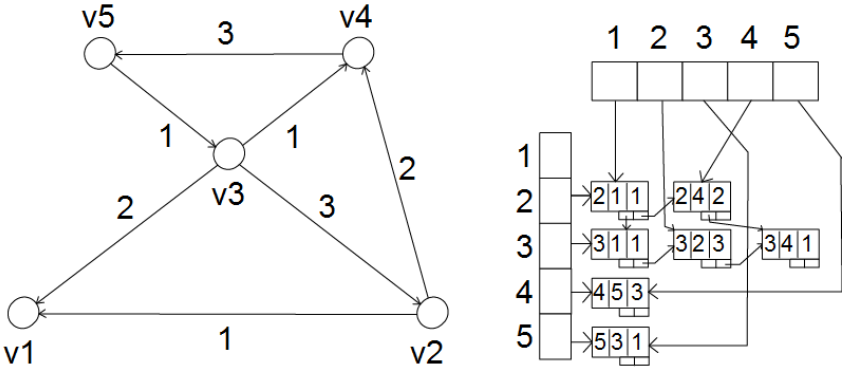


Obrázek 5 Reprezentace hvězdy typu tabulka (strom) – tabulka (seznam)

Pokud bychom pro náš *orientovaný graf* potřebovali realizovat operaci *zpřístupnění předchůdců*, potom z prozatím probraných reprezentací tomuto požadavku vyhovuje pouze reprezentace *pole – pole*.

Další, složitostně přívětivější variantou je uchovávat kromě dopředné hvězdy také hvězdu zpětnou. Nevýhodou této varianty je ovšem určitá duplicita informací v podobě uchovávání informace o ohodnocení hrany apod. S tím je spojena i zvýšená režie u modifikačních operací.

Méně náročnou reprezentací je tzv. *křížová reprezentace*. Tato reprezentace umožňuje současný přístup k předchůdcům i následníkům vrcholu. Realizace spočívá v zavedení dvou primárních datových struktur (jednu pro každý směr) a modifikaci hrany ve smyslu přidání reference na předchozí hranu. Ukázka reprezentace viz obrázek 6.



Obrázek 6 Křížová reprezentace grafu

[3] Podrobnosti o datové struktuře viz [3].
[4] Podrobnosti o datové struktuře viz [4].

Vertikální primární struktura slouží pro procházení následníků, horizontální struktura potom pro procházení předchůdců. Zvolené primární struktury u obrázku jsou typu *pole,* podobně lze ovšem transformovat i ostatní předem zmíněné reprezentace. Sekundární struktura však zůstává stejná u každé implementace.

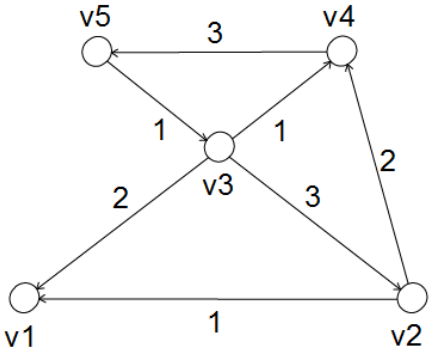
Všechny doposud zmíněné reprezentace grafu využívaly *vrcholově orientovaný přístup* k prvkům grafu. Nyní se podíváme na některé implementace spadající do kategorie *hranově orientovaného přístupu.*

Jednou z možných reprezentací grafu je reprezentace *tabulkou hran*. Tato reprezentace je tvořena pouze jednou strukturou, která obsahuje položky ve formě počáteční vrchol, koncový vrchol, doplňkové informace (ohodnocení, …). Klíčem prvků tabulky je složení identifikátorů vrcholů.

Tabulka může být prezentována staticky (polem), a to buď *polem utříděným*, kde složitost vkládání hran je $O(|X|)$ a složitost hledání hran je $O(log_2(|X|))$, nebo *polem neutříděným*, kde vkládání hran má složitost $O(1)$ a složitost hledání hran $O(|X|)$.

Dále může být tabulka reprezentována dynamicky (např. *seznamem*), který může být rovněž utříděný (vkládání hran $O(|X|)$), nebo neutříděný (vkládání hran $O(1)$). Vyhledávání hran v seznamu má složitost $O(|X|)$.

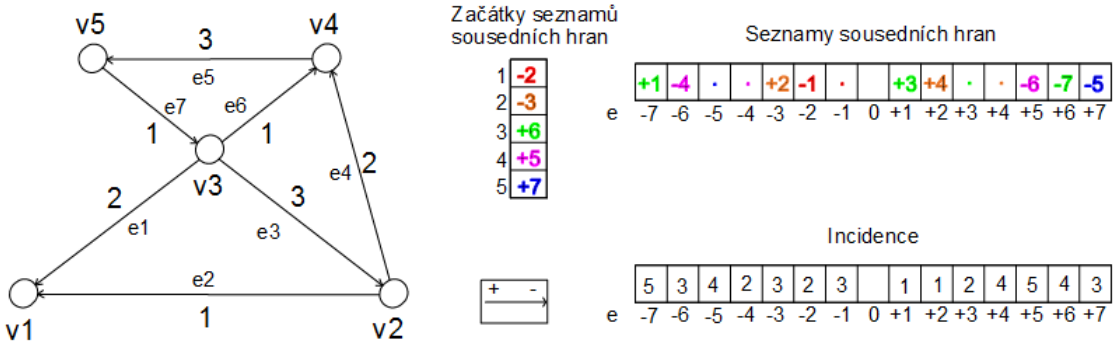
Nevýhodou reprezentace grafu *tabulkou hran* je nemožnost vložení vrcholu se stupněm 0 (vrchol musí vždy incidovat alespoň s jednou hranou). Tento nedostatek by bylo možné vyřešit přidáním sekundární struktury pro ukládání vrcholů, čímž by se zvýšila režie při modifikacích vrcholů grafu. Ukázka této reprezentace viz obrázek 7.



Počáteční vrchol	Koncový vrchol	Ohodnocení
v2	v1	1
v2	v4	2
v3	v1	2
v3	v2	3
v3	v4	1
v4	v5	3
v5	v3	1

Obrázek 7 Reprezentace grafu tabulkou hran

Jinou možnou reprezentací grafu v hranově orientovaném duchu je *pole hran*. Jak název napovídá, jedná se o pole hran se seznamy sousedností. Implementaci nejlépe přiblíží obrázek 8.

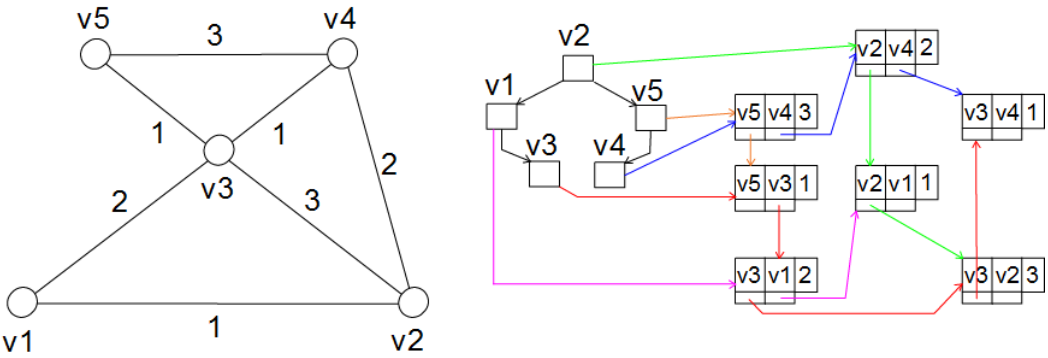


Obrázek 8 Reprezentace grafu polem hran

Jednotlivé hrany jsou očíslovány pevnými indexy, dále je hranám přiřazena polarita, aby bylo možné určit orientaci hrany. V této ukázce je začátek hrany kladný a konec hrany záporný, je ovšem možné polaritu bez problémů otočit.

Tato implementace ADS sestává ze tří polí. První pole, v obrázku nazvané *Začátky seznamů sousedních hran,* slouží k uchování začátků seznamů sousedních hran jednotlivých vrcholů očíslovaných pevnými indexy (každý seznam znázorněn jinou barvou). Tyto seznamy dále pokračují v poli zde nazvaném *Seznamy sousedních hran,* kde je uchovaný index následujícího souseda s ohledem na polaritu, nebo prázdný odkaz (v obrázku znak tečky). Poslední pole s názvem *Incidence* slouží k uchování Incidenčních vrcholů jednotlivých hran vzhledem k jejich zvolené polaritě. Reprezentace je hranově statická.

Poslední zde probíranou reprezentací grafu je *smíšená reprezentace*. Tato reprezentace je vhodná především pro *neorientované grafy*, lze v ní ovšem ukládat i hrany orientované. K pochopení reprezentace opět nejlépe poslouží obrázek (viz obrázek 9).



Obrázek 9 Smíšená reprezentace grafu

Reprezentace se skládá ze dvou struktur. První je struktura totožná s některou z probíraných dopředných hvězd (na obrázku zvolena tabulka na stromě). S volbou struktury souvisí i její vlastnosti z pohledu složitosti.

Specifická je ovšem struktura sekundární. Prvky sekundární struktury jsou hrany, obsahující odkazy na incidenční vrcholy, ohodnocení (nebo jiné doplňující informace o hraně),

a dva odkazy pro pokračování seznamu sousedů jednotlivých incidenčních vrcholů hrany. Tato sekundární struktura může být rovněž implementovaná libovolnou tabulkou.

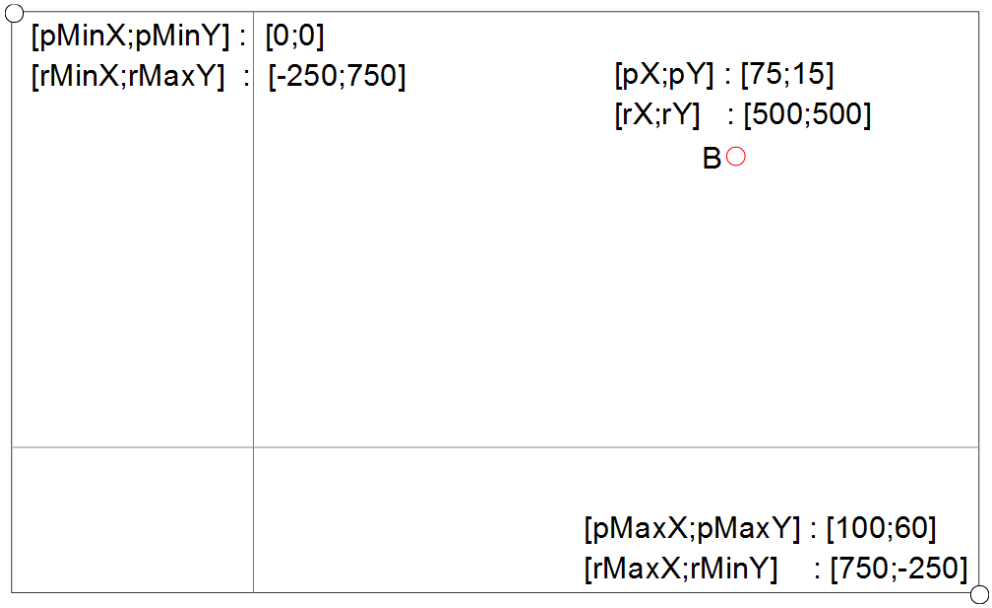
Na obrázku je pro přehlednost každý seznam sousedů jednotlivých vrcholů znázorněn jinou barvou. Aby se předešlo přílišné režii při modifikacích hran, nemají jednotlivé vrcholy vždy aktuální seznam svých sousedů, nýbrž se dynamicky sestavuje, pokud od posledního požadavku na seznam sousedů proběhla změna struktury sousedů tázaného vrcholu. Výsledkem tohoto principu je, že při modifikaci struktury hran se mění jen odkazy v sekundární datové struktuře, a nemodifikují se jednotlivé seznamy sousedů incidenčních vrcholů modifikovaných hran (př. při odstraňování hrany se mění odkazy pouze na jednom místě – sekundární struktura, namísto míst tří – sekundární struktura, seznam sousedů prvního incidenčního vrcholu, seznam sousedů druhého incidenčního vrcholu). Tento princip dává lepší výsledky při hromadné editaci hran (nové silnice, rušení starých,…), a následné jednorázové aktualizace seznamů sousedů vrcholů.

Reprezentace na obrázku je vrcholově dynamická, jelikož vrcholy jsou uchovány v *BVS* (nebo lépe *AVL,* viz reprezentace tabulka (strom) – tabulka (seznam)). [2][3][4]

1.2 Vykreslování grafu

Po výběru vhodné paměťové reprezentace grafu pro komponentu je potřeba analyzovat možnosti jeho vykreslování. Pokud bychom chtěli umožnit uživateli posouvání a změnu měřítka kreslícího plátna, narážíme zde na problém *pixelového a reálného* souřadnicového systému.

Obrazovka monitoru má totiž jen jeden neměnný souřadnicový systém. Naproti tomu my pro naše vykreslování potřebujeme dynamický souřadný systém s pohyblivým měřítkem a posunem. Je proto nutné rozlišovat mezi těmito systémy, a dle potřeby souřadnice mezi systémy převádět. Tento princip ilustruje obrázek 10.



Obrázek 10 Převod souřadnic mezi pixelovým a reálným souřadnicovým systémem

Vykreslovací oblast monitoru má meze pixelové (pMinX, pMinY, pMaxX, pMaxY) a reálné (rMinX, rMinY, rMaxX, rMaxY). U pixelového souřadného systému je otočena polarita y-ové osy oproti klasickému kartézskému souřadnému systému. Pixelový SS má rozměry 100x60 [px], reálný SS má rozměry 1000x1000 [j]. Dále je na obrázku bod B, jenž má pixelové souřadnice [pX, pY] a reálné souřadnice [rX, rY].

Jednou ze složek, které jsou důležité pro převod mezi soustavami je *měřítko*. Vzorce pro výpočet x-ového (dX) a y-ového (dY) měřítka viz následující rovnice:

$$dX = \frac{pMaxX}{rMaxX - rMinX} \tag{1}$$

$$dY = \frac{pMaxY}{rMaxY - rMinY} \tag{2}$$

Pokud by např. reálný i pixelový systém měly rozmezí souřadnic X velikosti 100, bylo by x-ové měřítko mezi systémy 1. Lze to přirovnat k měřítku na mapách. Ve vzorcích se neberou v potaz složky *pMinX* a *pMinY*, protože jsou vždy nulové. Pozice jmenovatele a čitatele ve vzorcích je jen otázkou vkusu, lze je i otočit, musí se to ovšem promítnout v dalších výpočtech.

Další důležitou složkou pro převod mezi souřadnicemi je *posun*, jelikož pixelový systém zůstává stejný, kdežto reálný mění svoje minimální a maximální souřadnice X/Y. Změnou měřítka a posunem rozmezí reálného souřadnicového systému je tedy možné graficky *zoomovat* a *rollovat* kreslící plátno. Převod z pixelového souřadného systému do reálného souřadného systému (P2R) viz následující rovnice:

$$P2RX = rMinX + \frac{pX}{dX} \tag{3}$$

$$P2RY = rMaxY - \frac{pY}{dY} \tag{4}$$

Převod opačný (R2P) viz:

$$R2PX = (rX - rMinX) \cdot dX \tag{5}$$

$$R2PY = (rMaxY - rY) \cdot dY \tag{6}$$

Je dále nutné tuto potřebu znalosti souřadnic prvků grafu implementovat do objektové třídy odvozené z některé abstraktní datové struktury implementující graf.

1.3 Funkcionality komponenty

Naše vyvíjená komponenta by měla být schopna uchovávat a zobrazovat obyčejný graf. Dále by měla umožnit jeho interaktivní editaci, ať už pomocí metod, které lze volat mimo komponentu, tak pomocí sledování událostí na komponentě a následné reagování modifikací grafu. Při modifikaci nebo pokusu o modifikaci uchovávaného grafu "zevnitř" by měla informovat svoje okolí pomocí správné události, a umožnit případné zrušení nebo další ošetření vzniklé modifikace.

Komponenta by dále měla být schopna editovat prvky grafu z vizuálního hlediska (barva vrcholu/hrany, velikost vrcholu, tloušťka hrany, fonty, …), případně poskytovat možnost změny implicitních hodnot nově vzniklých prvků grafu. Rovněž by neměly chybět možnosti resetování vlastností prvků grafu nebo změna barevného schématu komponenty.

Toto uživatelské nastavení by se mělo automaticky ukládat do dalšího spuštění aplikace. Komponenta by také měla být schopna vytvořený graf určitým způsobem ukládat a načítat.

1.4 Funkcionality ukázkové aplikace

Ukázková aplikace by měla demonstrovat plnou funkčnost vyvíjené komponenty po všech směrech. Měla by otestovat zpracování událostí vyvolaných komponentou při vnitřní modifikaci grafu, volat metody pro modifikaci zvenčí, otestovat schopnosti komponenty ukládat a načítat vypracované grafy, případně dále obohatit celkovou aplikaci pro praktické využití.

2 Praktická část

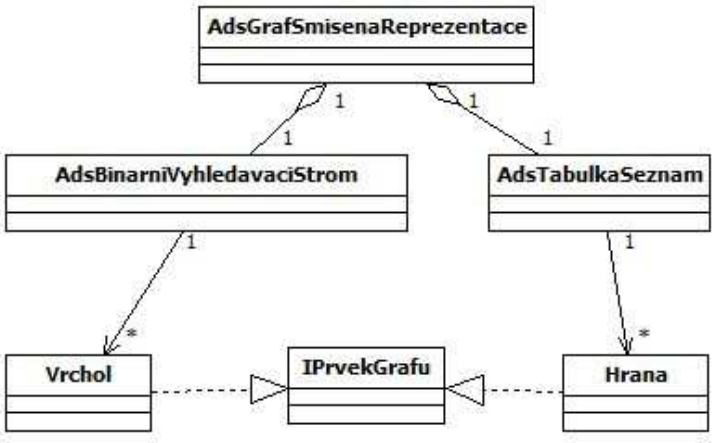
2.1 Výběr datové struktury pro reprezentaci grafu

Rozhodnutí, kterou *ADS* pro implementaci grafu použít, bylo ovlivněno následujícími faktory.

Komponenta uchovává *obyčejný graf*. Jelikož komponenta uchovává *obyčejný graf*, který je *neorientovaný*, odpadávají z možných kandidátů pro volbu všechny v práci zveřejněné reprezentace *hvězd*, kromě reprezentace *pole – pole*. Pro výběr tedy zbývají reprezentace *pole – pole*, *křížová reprezentace*, *tabulka hran, pole hran* a *smíšená reprezentace.*

Většina grafových algoritmů je tvořena pro *vrcholově orientovaný přístup*. Tím ze seznamu kandidátů odpadávají *hranově orientované reprezentace*. Máme tedy na výběr z *ADS pole – pole, křížová reprezentace* a *smíšená reprezentace.*

Reprezentace *pole – pole* je z výše zmiňovaných nejméně vhodná. Je to reprezentace *statická* a alokuje paměťové místo i pro zatím neexistující hrany. *Křížová reprezentace* je poměrně podobná reprezentaci *smíšené*, lze ji použít i pro *neorientované* nebo *smíšené* grafy, její primární určení je ovšem pro grafy *orientované* (dvě primární struktury pro procházení následníků/předchůdců vrcholu). Naší volbou pro tuto komponentu tedy bude reprezentace *smíšená*. Zjednodušený UML[5] diagram tříd této reprezentace viz obrázek 11.

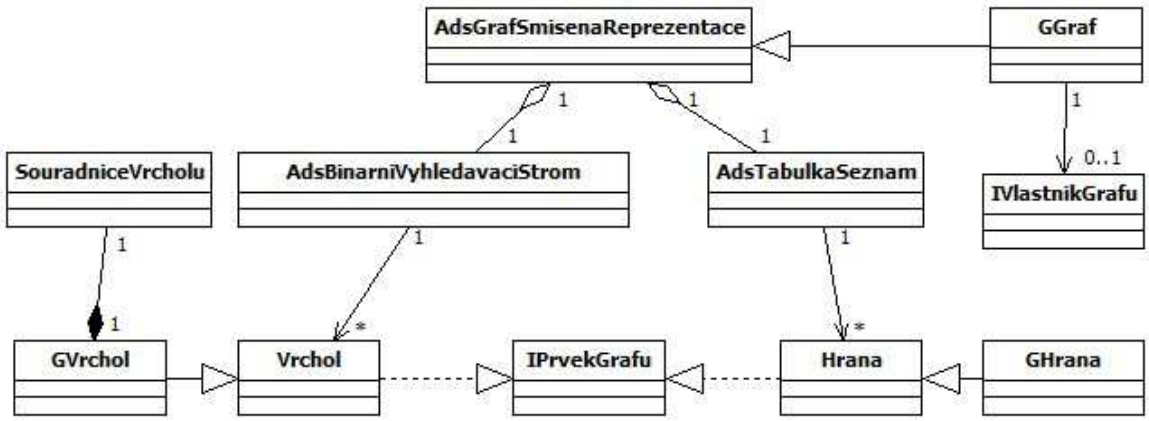


Obrázek 11 UML diagram tříd smíšené reprezentace grafu

[5] Pokud nejste obeznámeni s problematikou UML, lze více informací nalézt v publikaci [6].

Řídící třídou této datové struktury je třída *AdsGrafSmisenaReprezentace,* která poskytuje metody definované *ADT Graf*. Tato struktura je *generická*, při vytváření nového grafu je nutné definovat datový typ vrcholu, hrany, klíčů vrcholů, klíčů hran a ohodnocení hran grafu. Agreguje tabulky *AdsBinarnyVyhledavaciStrom* (primární datová struktura) a *AdsTabulkaSeznam* (sekundární datová struktura). V těchto kolekcích jsou uchované třídy *Vrchol* a *Hrana*, které dědí značkovací rozhraní *IPrvekGrafu*. Toto rozhraní je vhodné např. pro kolekce typu *IPrvekGrafu*.

Graf uchovává *vrcholy* a *neorientované hrany*, nekontroluje ale *násobnost* hran ani možné *smyčky*. Prvky grafu dále nemají vlastnosti (souřadnice, vizuální atributy, …) pro jejich vykreslování. Proto bude nutné od této struktury *dědit* a chybějící funkcionality doplnit. Zjednodušená vizualizace potomka viz obrázek 12.



Obrázek 12 UML diagram tříd smíšeného grafu upraveného pro vykreslování

Třída *GGraf* přidává funkci prevence vkládání *smyček* a *násobných hran* v metodě pro přidání hrany. Dále může uchovávat referenci na vlastníka grafu, kterému bude zasílat požadavek (metoda *Invalidate()*, v případě že reference není prázdná) pro překreslení, pokud se změní jakýkoli vykreslovaný údaj grafu. Dalšími požadovanými schopnostmi vlastníka grafu je schopnost editovat vlastnosti grafu pomocí dialogového okna. Rozhraní *IVlastnikGrafu* proto předpisuje metody *UpravVrchol(GVrchol vrchol)* a *UpravHranu(GHrana hrana*).

Třídě *GVrchol* přibyla důležitá vlastnost implementovaná třídou *SouradniceVrcholu*. Tato třída je *neměnný objekt*, definovaný jako objekt, který je inicializován při svém vytvoření, a dále není měnitelný[6]. K tomu, proč je tato neměnnost důležitá, se vrátíme později.

Pokud má vrchol souřadnice, je možné ho na jejich základě vykreslit. Je možné rovněž vykreslit hrany pomocí incidenčních vrcholů hrany (a jejich souřadnic). Ve třídě *GVrchol* bylo rovněž zvoleno, že klíčem vrcholu budou jeho souřadnice. Toto rozhodnutí zvýší režii při změně

[6] Tento návrhový vzor nepatří do klasických vzorů definovaných GoF, je uveden v publikaci [7].

souřadnic vrcholu (vyjmutí ze stromu dle starého klíče, vložení s klíčem novým), ale zároveň umožní, že na daných souřadnicích může existovat pouze jeden vrchol. Kdyby tento krok nebyl podniknut, muselo by se při každém vkládání vrcholu kontrolovat, zda nějaký vrchol již nemá dané souřadnice. Neměnitelnost třídy *SouradniceVrcholu* je důležitá, jelikož požadavek pro změnu souřadnic vrcholu musí být předán přímo vrcholu (místo třídě *SouradniceVrcholu*) formou nové instance třídy *SouradniceVrcholu*, a vrchol dále může reagovat výše zmíněným vyjmutím vrcholu ze stromu a následným vložením s novými souřadnicemi.

U třídy *GHrana* nebylo nutné volit přesnou formu klíče. Při vytváření nové instance se buď nechá vygenerovat implicitní klíč hrany, nebo se v konstruktoru předá hodnota jiná. Tyto dva přístupy by se ovšem neměly kombinovat.

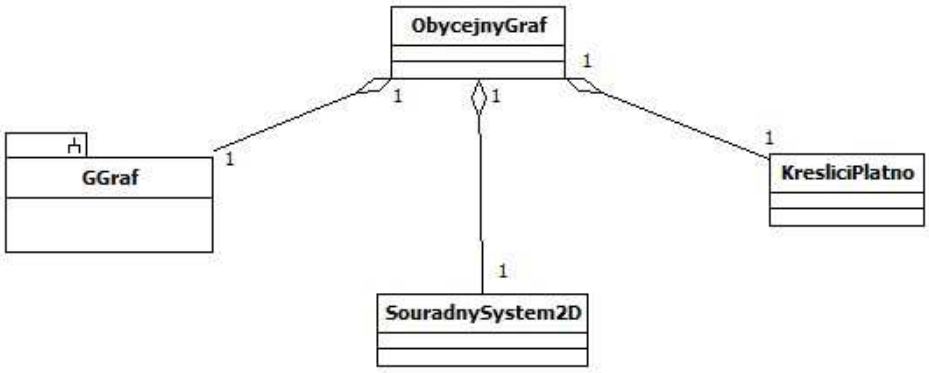
Třídám *GVrchol* a *GHrana* dále přibyly vlastnosti, které jsou potřebné pro uživatelsky přívětivé vykreslení a rozlišení jednotlivých prvků grafu (barva prvků, velikost vrcholu, tloušťka hrany, zobrazení popisku prvku, …).

2.2 Sestavování komponenty

Komponenta bude vytvořena v programovacím jazyku C# na platformě .NET 4. Rovněž bude využívat pro svoji funkci technologii Windows Forms.

2.2.1 Zobrazování grafu

Když máme připravenou datovou strukturu grafu, který je schopen uchovávat informace potřebné k jeho vykreslení, je nutné toto zobrazování implementovat ve vyvíjené komponentě. K tomu budeme potřebovat již zmíněný *graf, kreslící plátno, souřadný systém* a vhodné *posluchače událostí*. UML Vizualizace viz obrázek 13.



Obrázek 13 UML diagram zobrazovací části komponenty

Naše komponenta je implementovaná třídou *ObycejnyGraf*. Ta agreguje mimo jiné objekt typu *KresliciPlatno*, na které kreslí. Na plátno se vykreslují vrcholy a hrany grafu, případně

souřadnicová osa (pokud je v daném rozmezí reálného systému viditelná). Data k vykreslování čerpá ze subsystému *GGraf*. Agregovaná třída *SouradnySystem2D* slouží k přepočítávání reálných souřadnic, uchovaných v třídách *GVrchol*, na souřadnice pixelové, dle kterých se potom zasílají požadavky na kreslící plátno.

Pokud bychom dále zobrazovací část nerozvíjeli, bylo by kreslící plátno fixní. Proto byly přihlášeni posluchači pro události myši a události klávesnice. Posluchači událostí myši naslouchají události *MouseDown*, *MouseMove*, *MouseWheel* a *MouseUp*. Posluchač klávesnice naslouchá události *KeyDown*. Přehled zobrazovacích příkazů viz tabulka 1.

Tabulka 1 Zobrazovací příkazy komponenty

Příkaz	Funkcionalita
Levé tlačítko myši + pohyb myší	Posun kreslícího plátna
Kolečko myši	Přibližování a oddalování kreslícího plátna
Šipka nahoru	Posun plátna nahoru
Šipka dolů	Posun plátna dolů
Šipka doleva	Posun plátna doleva
Šipka doprava	Posun plátna doprava
Page Down	Oddálení plátna
Page Up	Přiblížení plátna

Při výskytu události *MouseDown* se kontrolují počáteční podmínky (levé tlačítko myši a žádné modifikační tlačítka, viz také uživatelská dokumentace komponenty), a při jejich splnění se aktivuje režim posouvání plátna. Událost *MouseUp* potom potenciální režim posouvání ukončuje. Pokud je režim posouvání aktivní, nastává při události *MouseMove* posun kreslícího plátna. Posun se realizuje změnou *reálných* mezí souřadného systému.

Problematická může být volba, o kolik posunout kreslící plátno vzhledem ke změně polohy kurzoru myši. Pokud bychom reálné meze posunuli stejným dílem v *jednotkách*, jakým byl posunut kurzor v *pixelech*, byl by tento princip v pořádku pouze v případě, že měřítko mezi systémy (*dX, dY*) je 1. V případě, že rozpětí reálného systému je ve směru *x* nebo *y* menší než rozpětí pixelové v témže směru, posun by byl příliš velký, naopak pokud by bylo reálné rozpětí větší, posun by byl nedostatečný. Optický výsledek tohoto problému je "zadrhávání" nebo naopak "utíkání" plátna pod kurzorem. Řešení problému spočívá v přenásobení pixelového

posunu kurzoru obrácenou hodnotou současného měřítka. Tím se zajistí, že se plátno opticky posune o stejný díl, jako se posunul kurzor myši.

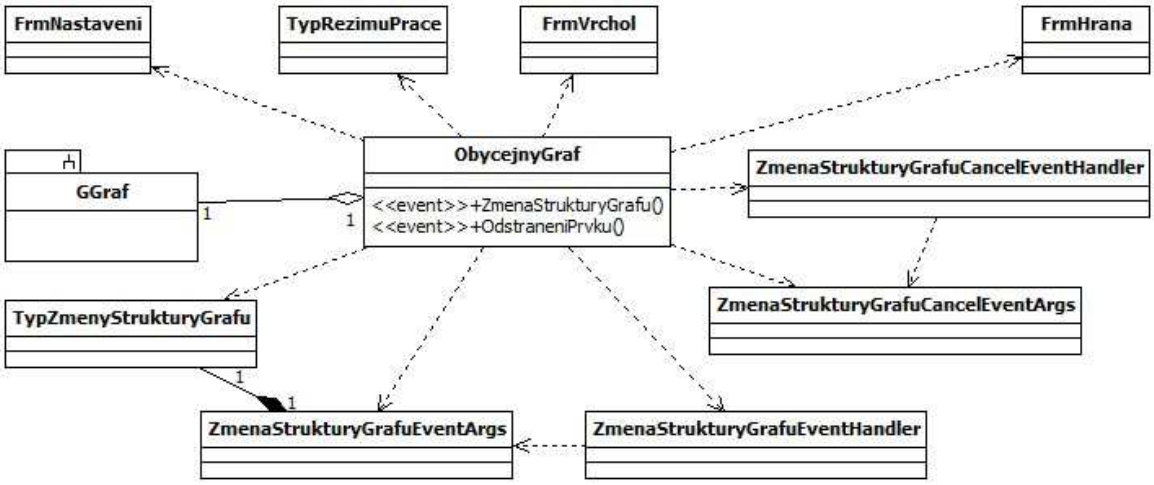
Poslední sledovanou událostí myši je událost *MouseWheel*, při které se provede změna měřítka mezi systémy vzhledem ke směru otočení myši. Tím se zajistí přibližování a oddalování kreslícího plátna.

U událostí klávesnice je situace trochu jednodušší. Jedinou sledovanou událostí je událost *KeyDown*, kde se filtruje, jaká klávesa byla stisknuta. Podle toho se potom provede posun plátna, nebo přiblížení/oddálení.

Poslední věcí, kterou bychom se měli u vykreslování grafu zabývat, je změna velikosti kreslícího plátna. Tento jev nastane při změně velikosti formuláře aplikace. Příslušná událost nese název *Resize*. Na tuto událost je třeba reagovat přepočítáním měřítek *dX* a *dY*.

2.2.2 Editace grafu

Další z požadovaných funkčností komponenty je možnost uchovávaný graf *editovat*. Komponenta bude podporovat dva typy editace. *Interní editace* je editace zprostředkovaná komponentou samotnou. Sleduje Kreslící plátno a při splnění podmínek vykonává editační operace. Druhým typem editace je *externí editace*. Komponenta poskytuje uchovávaný graf pro čtení ve formě *vlastnosti*. Poté můžeme externě volat editační metody grafu. UML diagram viz obrázek 14.



Obrázek 14 UML vizualizace editační části komponenty

Implementačně jednodušší částí je *externí* editace grafu. Komponenta pouze poskytuje graf ze subsystému *GGraf* jako vlastnost, a vnější část aplikace tyto metody volá. *GGraf* je na tyto modifikační metody připraven a po jejich aplikování zasílá svému vlastníkovi (viz podkapitola 2.1) požadavek o překreslení plátna.

Situace je ovšem složitější u interní editace grafu. Můžeme zde problém rozložit na části *vizuální signalizace* potenciálního prvku k editaci, *zachycení požadavku* o editaci prvku, provedení editace a *informování okolí* a překreslení plátna.

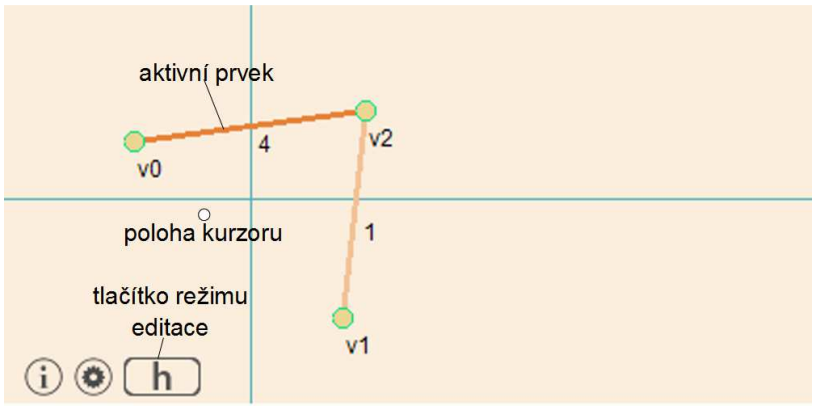
Už u první části narážíme na menší rozhodovací problémy. Jak signalizovat aktuální potenciálně upravovatelný prvek? Signalizovat zároveň vrcholy i hrany, nebo problém rozdělit na dva editační módy?

Pokud mají prvky grafu implementovanou vlastnost barva, nabízí se jako rozumné řešení rozlišit aktivní prvky barevně. Tato aktivní barva se bude dát editovat v menu nastavení komponenty (viz podkapitola 2.2.3).

Ohledně režimu vykreslování se zdá být lepší, pokud se vykreslují najednou aktuální vrcholy i hrany. Uživatel nemusí přepínat mezi režimy a tím se stává ovládání jednodušší. Tato volba ale není zcela vhodná. Aktivní prvek totiž můžeme hledat dvojím způsobem – kurzor myši se nachází na souřadnicích prvku (souřadnice vrcholu, úsečka hrany), nebo je aktivním prvkem prvek, jehož souřadnice jsou nejblíže kurzoru myši.

Pokud bychom chtěli použít druhý způsob hledání aktuálního prvku, narazíme na problém u grafu, který má vrcholy poměrně blízko u sebe. V takovém případě by bylo obtížné nalézt správnou polohu kurzoru pro požadovaný prvek, jelikož by se vrcholy a hrany "kryly". Pokud naopak přejdeme k hledání prvku podle přesné hodnoty souřadnic, narážíme na problém při větším oddálení plátna. Kurzor myši se pohybuje příliš velkými díly a přesné souřadnice požadovaného prvku lze obtížně nalézt.

Z těchto důvodů bude lepší volbou jít cestou režimů editace. Přepínání mezi režimy editace hran nebo vrcholů bude možné buď pomocí *tlačítka na komponentě* (viz obrázek 15), nebo pomocí vlastnosti komponenty s názvem *RezimPrace*. Této vlastnosti lze nastavit hodnoty *PraceSVrcholy* a *PraceSHranami*, které jsou výčtovými konstantami výčtového typu *TypRezimuPrace*. Můžeme rovněž použít vyhledávání aktivního prvku jako nejbližšího prvku vzhledem ke kurzoru, což zvýší pohodlnost ovládání komponenty. Ukázka vykreslování viz obrázek 15.



Obrázek 15 Vykreslování aktivního prvku vzhledem k režimu práce

Další částí procesu interní editace grafu je zachytávání a zpracování požadavků o editaci. K tomu slouží posluchač události *MouseClick*. V tomto posluchači se provede analýza stisknutého tlačítka myši a možných modifikačních tlačítek. Přehled editačních příkazů viz tabulka 2.

Tabulka 2 Editační příkazy komponenty

Příkaz	Funkcionalita
Ctrl + levé tlačítko myši	Vložení vrcholu na souřadnice kurzoru. Režim práce s vrcholy.
Ctrl + pravé tlačítko myši	Vložení hrany. Aplikování příkazu na dva různé (incidenční) vrcholy. Režim práce s vrcholy.
Shift + levé tlačítko myši	Editace vrcholu (režim práce s vrcholy) nebo hrany (režim práce s hranami).
Alt + levé tlačítko myši	Odstranění vrcholu (režim práce s vrcholy) nebo hrany (režim práce s hranami).

Po rozpoznání příkazu se provede daná operace. Okolní aplikace ovšem neví, že uživatel použil předdefinovaný příkaz a editoval graf. Navíc editace formou odstranění prvku grafu může být z pohledu okolní aplikace nežádoucí (odstranění hrany, skrze kterou vede nejkratší cesta, …). Je proto nutné komunikovat s okolní aplikací pomocí zpráv.

K této komunikaci nejlépe poslouží *události*. Až doposud jsme vystačili pouze s událostmi již existujícími, a přihlašovali jsme pouze posluchače a obsluhovací metody. Nyní budeme muset nadefinovat události vlastní. Princip událostí je víceméně aplikací návrhového vzoru *pozorovatel* (*observer*). Pozorovanou třídou je naše komponenta, a pozorující třídou je okolní aplikace. Komponenta musí poskytnout registrační a odhlašující metody pro pozorování daných událostí, přičemž definuje signaturu obslužných metod okolní aplikace pomocí delegátů. Tyto obslužné metody potom volá při výskytu sledované události. Jednu událost může sledovat více

tříd, a jedna třída může sledovat více událostí. Více informací o implementaci tohoto návrhového vzoru lze nalézt v publikaci [8].

Pro naše potřeby si vytvoříme dvě události – *ZmenaStrukturyGrafu* a *OdstraneniPrvku*. Událost *OdstraneniPrvku* slouží k případnému zamezení odstranění prvku grafu. Pomocí vlastnosti *Cancel* třídy *ZmenaStrukturyGrafuCancelEventArgs,* která je jedním ze dvou parametrů delegátu *ZmenaStrukturyGrafuCancelEventHandler,* lze operaci stornovat změnou vlastnosti na hodnotu *true*. Třída *ZmenaStrukturyGrafuCancelEventArgs* rovněž obsahuje referenci na prvek grafu, jehož se požadavek o odstranění týká.

Událost *ZmenaStrukturyGrafu* slouží k informaci o již proběhlé editaci. Součástí třídy *ZmenaStrukturyGrafuEventArgs* je výčtový typ *TypZmenyStrukturyGrafu*. Ten obsahuje výčtové konstanty pro každou z možných editací (vložení vrcholu, vložení hrany, editace vrcholu, editace hrany, odstranění vrcholu, odstranění hrany). Další součástí třídy je prvek grafu, na nemž editační operace proběhla. Třída *ZmenaStrukturyGrafuEventArgs* je opět jedním ze dvou parametrů delegátu *ZmenaStrukturyGrafuEventHandler*, kde se při vyvolávání události *ZmenaStrukturyGrafu* nastaví příslušná hodnota výčtového typu podle provedené editace.

Při editacích typu odstranění nebo přidání prvku grafu se provede daná operace a vyvolá příslušná událost. U editace typu úprava prvku se vyvolá editační dialogový formulář *FrmVrchol* nebo *FrmHrana* dle editovaného prvku (musí být povolena vlastnost komponenty *GrafickaEditaceStruktury*). V těchto formulářích je potom možné upravovat barvu prvku, jeho velikost, název, zobrazení popisku, nebo měnit ohodnocení hrany. Po zavření dialogového okna se provede případné překreslení prvku a vyvolání příslušné události. Vzhled editačních formulářů viz obrázek 16.

Tyto interní editační funkcionality komponenty jsou k dispozici, ale někdy okolní aplikace může chtít jejich využívání zakázat a strukturu grafu řídit samostatně. K tomuto účelu slouží vlastnost komponenty *GrafickaEditaceStruktury*. Pokud se nastaví na hodnotu *false,* nebude komponenta měnit strukturu grafu. Bude jej pouze vykreslovat.

Rovněž je možné zakázat vykreslování aktivního prvku pomocí vlastnosti *VykreslovaniAktivnihoPrvku*. Pokud se vlastnost nastaví na hodnotu *false* nebude komponenta provádět vykreslování aktivního prvku grafu vůči kurzoru. V případě povolení hledání aktivního prvku je prvek přístupný z vlastnosti komponenty s názvem *AktivniPrvek*. Tato vlastnost obsahuje aktivní prvek grafu podle aktuálního režimu práce.

Obrázek 16 Editační dialogová okna komponenty

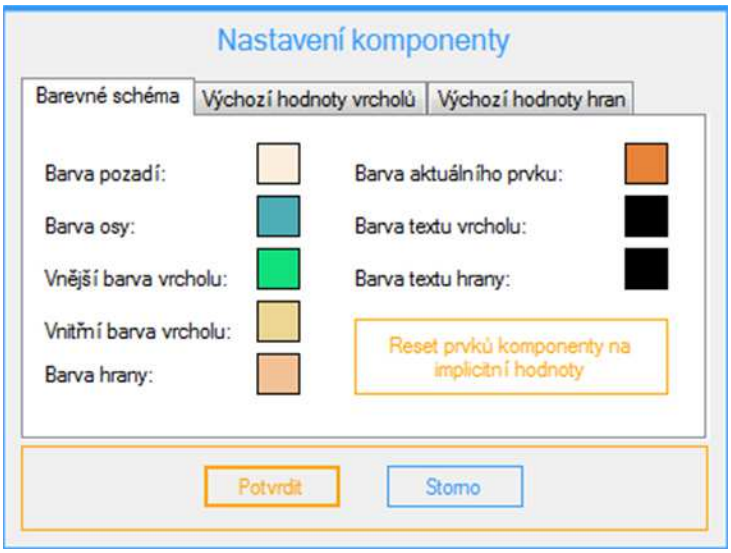
2.2.3 Konfigurace

Při vkládání prvku grafu je potřeba definovat jeho vlastnosti (barva, velikost, ohodnocení, …). Bylo by nepraktické nutit uživatele vyplňovat stále stejné hodnoty, pokud by chtěl udržet jednotný formát prvků. Rovněž barva os kreslícího plátna nebo aktivního prvku může být pro každého uživatele příjemná v jiné podobě. Z těchto důvodů byla zavedena možnost *konfigurace komponenty*.

Konfigurace je uskutečnitelná dvěma způsoby. První způsob je formou dialogového okna *FrmNastaveni* (UML viz obrázek 14, obrázek viz obrázek 17). Okno se zobrazí po kliknutí na tlačítko ozubeného kola na komponentě (obrázek 15). V tomto okně je možné nastavit barevné schéma komponenty a implicitní hodnoty prvků grafu. Při vytváření prvků grafu pomocí interní editace potom komponenta čerpá vlastnosti prvků z této konfigurace.

V konfiguračním dialogovém okně komponenty je rovněž možné resetovat stávající prvky grafu na současné hodnoty konfigurace komponenty (implicitní hodnoty). Po kliknutí na příslušné tlačítko a následném potvrzení dialogového okna se všechny prvky zresetují, vyjma vlastnosti ohodnocení hran grafu.

Druhou možností konfigurace komponenty je nastavení vlastností komponenty. Přehled konfigurovatelných vlastností (včetně nevizuálních) viz příloha A.



Obrázek 17 Konfigurační dialogové okno komponenty

2.2.4 Datový modul

Uživatel již může konfigurovat komponentu. V tomto stádiu by si ovšem svoji oblíbenou konfiguraci musel nastavovat po každém zapnutí aplikace. Je proto nutné vytvořit *datový modul* komponenty, který bude moci konfiguraci komponenty ukládat do souboru dle uživatelské cesty, a při startu aplikace tento soubor načítat. Modul bude rovněž umožňovat ukládat nebo načítat graf komponenty *binární serializací*. Zjednodušený UML diagram tohoto modulu viz obrázek 18.

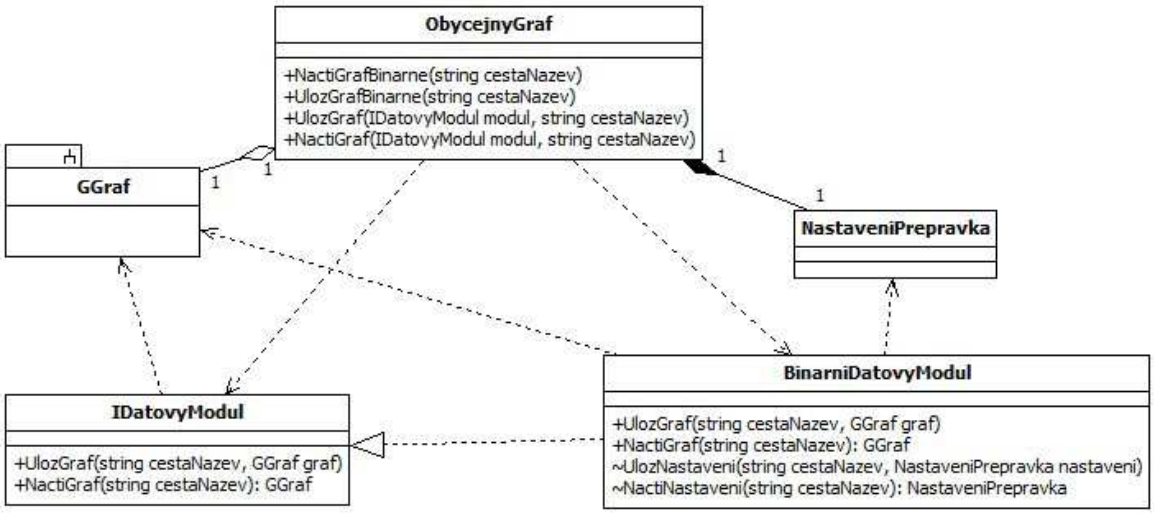
Pro přenášení uživatelského nastavení byla vytvořena speciální třída *NastaveniPrepravka*. Implementace je aplikací návrhového vzoru *Přepravka*[7]. Je to *neměnný objekt*, který nemá žádné metody, pouze veřejné vlastnosti. Slouží k pohodlnému přenášení informací pohromadě. Při vyvolání konfiguračního dialogového okna se předá přepravka se současným nastavením a přednastaví se hodnoty dialogu. Po odsouhlasení dialogu se vytvoří nová přepravka, která se následně rozbalí a komponenta se dle ni rekonfiguruje. Načítání a ukládání konfigurační přepravky provádí třída *BinarniDatovyModul*. Tato třída je implementací návrhového vzoru *Jedináček*, jelikož nepotřebujeme více instancí datového modulu komponenty. Binárnímu modulu se předává přepravka, a ten ji ukládá jako binární soubor do souboru dle vlastnosti *CestaKeKonfiguracnimuSouboru*.

Automatické ukládání se provádí při změně konfigurace komponenty pomocí konfiguračního dialogového okna komponenty. Při změně konfigurace pomocí vlastností komponenty si následné uložení musí obsatrat programátor sám.

[7] Podrobnosti viz [7].

Programátor ovšem může chtít konfiguraci ukládat jiným způsobem, a proto lze toto automatické načítání a ukládání konfigurační přepravky potlačit zakázáním vlastnosti *AutomatickeUkladaniNacitaniKonfigurace*.

Další důležitou schopností komponenty je *ukládat* a *načítat* graf. Proces ukládání a načítání lze realizovat mnoha způsoby. Proto bylo vytvořeno rozhraní *IDatovyModul,* které předepisuje společnou signaturu metodám pro ukládání a načítání grafu. Komponenta implementuje pouze ukládání a načítání pomocí *binární serializace*. K tomu slouží metody komponenty *UlozGrafBinarne* a *NactiGrafBinarne,* které volají metody jedináčka *BinarniDatovyModul*. Je zde ovšem ponechané místo pro jiné datové moduly metodami komponenty *NactiGraf* a *UlozGaf*, které přijímají jako argument implementaci rozhraní *IDatovyModul*, a dále volají jeho metody. Tyto metody jsou implementací návrhového vzoru *Příkaz*.



Obrázek 18 UML diagram datového modulu komponenty

Implementací datového modulu komponenty jsme dovršili její vývoj. Komponenta nyní disponuje všemi požadovanými vlastnostmi.

2.3 Ukázková aplikace

Poslední částí projektu je vytvoření ukázkové aplikace, jež naši komponentu využívá. Úkolem aplikace je otestovat funkčnost komponenty a případně přidat další funkcionality do aplikace jako celku. Přidanou funkcionalitou bude ukázka aplikace některých grafových algoritmů.

2.3.1 Komunikace s komponentou

Zahrnutím komponenty do našeho formuláře jsme získali přístup k jejím veřejným metodám a vlastnostem. V ukázkové aplikaci vykreslujeme aktuální polohu kurzoru na kreslícím plátně.

K tomu nám slouží pozorovatel události *MouseMove* na komponentě, a vlastnost komponenty s názvem *SouradniceKurzoru*. Dále hlavní formulář vykresluje počet vrcholů (vlastnost *PocetVrcholu*) a hran (vlastnost *PocetHran*) grafu. Tyto hodnoty překresluje při změně počtu některého z prvků grafu. Pro bližší představu viz obrázek v příloze B, položka 10.

Ukázková aplikace rovněž sleduje událost *KeyDown* na komponentě, a v případě, že je stisknuta klávesa Caps Lock, změní režim práce komponenty.

Další vykreslovanou informací grafu jsou seznamy vrcholů a hran (položky 5 a 6 v příloze B). Tyto informace se opět aktualizují při změně struktury grafu.

Způsoby změny struktury grafu již byly popsány v předchozích kapitolách. Prvním způsobem je editace *externí*, kdy ukázková aplikace volá metody grafu v komponentě.

K tomu slouží sada tlačítek viz položky 7 a 8 v příloze B. Při vkládání vrcholu se nejdříve vytvoří nová instance třídy *GVrchol*, a poté se zavolá metoda komponenty *UpravVrchol*. Tato metoda je předepsanou metodou rozhraní *IVlastnikGrafu* (podrobnosti viz podkapitola 2.1). Po dodatečné editaci nového vrcholu se vrchol vloží do grafu. Při kliknutí na tlačíko editace vrcholu se vyvolá pouze editační okno komponenty. Rovněž po dvojím kliknutí na řádek seznamu vrcholů se spustí procedura editace vrcholu. Po kliknutí na tlačítko na odstranění vrcholu se nejdříve zkontroluje, zda vrchol je stupně 0 (metoda to nekontroluje), a poté se následně vrchol odstraní.

Při přidávání hrany se vyvolá dialogové okno *FrmVyberDvouVrcholu* pro výběr dvou incidenčních vrcholů hrany. Toto okno obsahuje dva seznamy vrcholů grafu. Dialogové okno kontroluje, zda jsou vybrány oba vrcholy, či zda se nevybrali dva stejné vrcholy (smyčka). Poté se hrana vytvoří a zavolá se metoda komponenty pro grafickou editaci hrany. U editace hrany se pouze vyvolá dialogové okno pro editaci. Poklepání na řádek v seznamu hran na požadovanou hranu opět vyvolá editaci. Tlačítko pro odstranění hrany pouze odstraní hranu z grafu, a překreslí seznam hran a ukazatele počtu hran.

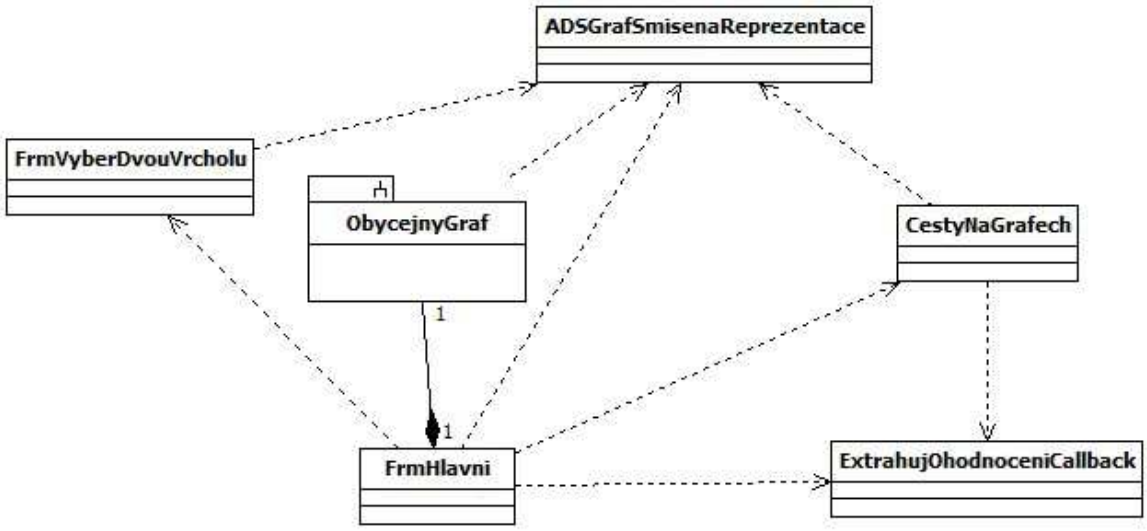
Pokud uživatel uskuteční interní editaci grafu, ukázková aplikace naslouchá události *ZmenaStrukturyGrafu*. U odstraňování prvku nejdříve vypíše potvrzovací dialogové okno, zda chce uživatel opravdu odstranění prvku uskutečnit. Dle rozhodnutí uživatele potom nastaví vlastnost *Cancel* třídy *ZmenaStrukturyGrafuCancelEventArgs*.

Aplikace rovněž využívá datový modul komponenty. Umožňuje uživateli si svůj graf binárně uložit a později načíst. Po potvrzení dialogového okna zasílá ukázková aplikace zprávu komponentě s požadovanou cestou k souboru. Dialogové okno má rovněž aktivní filtr, aby bylo možné ukládat a načítat pouze soubory s koncovkou *.bin. Ilustrace ukládání grafu viz příloha C.

Ukázková aplikace ponechává zapnuté automatické ukládání a načítání konfigurace komponenty. Tuto konfiguraci načítá z kořenu aplikace ze souboru s názvem *MojeNastaveni.bin*.

2.3.2 Grafové algoritmy

Další z funkcionalit ukázkové aplikace je využití komponenty pro výpočet nejkratší a nejspolehlivější cesty na obyčejném grafu. Pro výpočty slouží třída *CestyNaGrafech*. Tato třída ale nepracuje s upraveným grafickým grafem *GGraf*, jelikož metodami a vlastnostmi potřebnými pro výpočet cesty již disponuje třída *AdsGrafSmisenaReprezentace*. Tím se zajistí nezávislost třídy *CestyNaGrafech* na konkrétní implementaci smíšeného grafu. UML diagram vazeb mezi třídami viz Obrázek 19.



Obrázek 19 UML diagram tříd ukázkové aplikace

Závislostní šipky do třídy *AdsGrafSmisenaReprezentace* ze tříd *FrmVyberDvouVrcholu* a *FrmHlavni* jsou přítomny, jelikož tyto třídy využívají enumerátor pro cyklus foreach nad grafem. Dále tyto třídy pracují pouze s třídou *GGraf*.

Výpočet nejkratší cesty se provádí *Dijkstrovým* algoritmem. Výpočet nejspolehlivější cesty obdobně, pouze se v počátku převede orientace úlohy (z maximalizační na minimalizační). Zároveň se operátor $\prod$ převede na operátor Σ. Více informací o těchto grafových algoritmech viz publikace [1].

Datovým typem ohodnocení hran je generický datový typ. Proto třída *CestyNaGrafech* požaduje jako jeden z argumentů svých metod delegát *ExtrahujOhodnoceniCallback*, který z onoho generického datového typu extrahuje pro výpočet podstatné číselné ohodnocení hrany.

Po aktivování položky menu pro výpočet cesty se zobrazí již zmiňovaný formulář *FrmVyberDvouVrcholu*, kde si uživatel vybere počáteční a koncový vrchol cesty. I zde se kontroluje zda jsou oba vrcholy vybrané, a zda se nevybrala smyčka. Po potvrzení dialogového okna ukázková aplikace předá argumenty graf, počáteční vrchol, koncový vrchol a delegát pro extrakci některé z metod třídy *CestyNaGrafech* (*NejkratsiCesta*, *NejspolehlivejsiCesta*). Tyto metody vracejí obousměrný necyklický seznam typu *IPrvekGrafu*, který začíná počátečním vrcholem, a končí koncovým vrcholem. Metody pro svůj výpočet využívají vlastnost *Tag* prvků grafu. Ve vlastnosti *Tag* koncového vrcholu cesty se nachází zjištěná délka cesty nebo pravděpodobnost průchodu cestou. Ukázková aplikace tyto informace zpracuje a vypíše formou modálního okna. Ilustrace viz příloha D.

3 Závěr

Vytvořená aplikace splňuje všechny požadavky a je plně funkční. Komponenta uchovává obyčejný graf, který vykresluje, edituje, ukládá a načítá. Její konfigurace se volitelně automaticky ukládá. Ukázková aplikace plně využívá vyvinutou komponentu, a demonstruje na ni algoritmy pro výpočet nejspolehlivější cesty a nejkratší cesty pomocí Dijkstrova algoritmu.

Možné vylepšení aplikace by bylo dosažitelné nahrazením binárního vyhledávacího stromu jako primární struktury smíšené reprezentace grafu stromem s lepší složitostí operací (AVL strom, červeno-černý strom).

Komponenta může sloužit jako úschovné, vykreslovací a editační jádro pro jakoukoli jinou aplikaci v prostředí .NET 4.0 a vyšším, podporujícím technologii Windows Forms, pracující s obyčejnými grafy. Aplikace jako celek může sloužit k podpoře výuky teorie grafů.

4 Použitá literatura

[1] VOLEK, Josef. *Operační výzkum*. 1. vyd. Pardubice: Univerzita Pardubice, 2002, 111 s. ISBN 80-719-4410-6.

[2] KAVIČKA, Antonín. *Datové struktury* [intranet]. Pardubice, 2010 [cit. 2013-03-11] Dostupné z: Studijní agendy Univerzity Pardubice.

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

[4] WIRTH, Niklaus. *Algoritmy a štruktúry údajov*. 2. vyd. Překlad Pavol Fischer. Bratislava: Alfa, 1989, 481 s. Edícia výpočtovej techniky. ISBN 80-050-0153-3.

[5] VESELÝ, Petr. *Počítačová grafika* [intranet]. Pardubice, 2011 [cit. 2013-04-04]. Dostupné z: Studijní agenda Univerzity Pardubice.

[6] KANISOVÁ, Hana a Miroslav MÜLLER. *UML srozumitelně*. 2. aktualiz. vyd. Brno: Computer Press, 2006, 176 s. ISBN 80-251-1083-4.

[7] PECINOVSKÝ, Rudolf. *Návrhové vzory*. Vyd. 1. Brno: Computer Press, 2007, 527 s. ISBN 978-80-251-1582-4.

[8] NAGEL, Christian, Jay GLYNN, Karli WATSON a Morgan SKINNER. *C# 2008: programujeme profesionálně*. Vyd. 1. Brno: Computer Press, 2009, 772 s. Programujeme profesionálně. ISBN 978-80-251-2401-7.

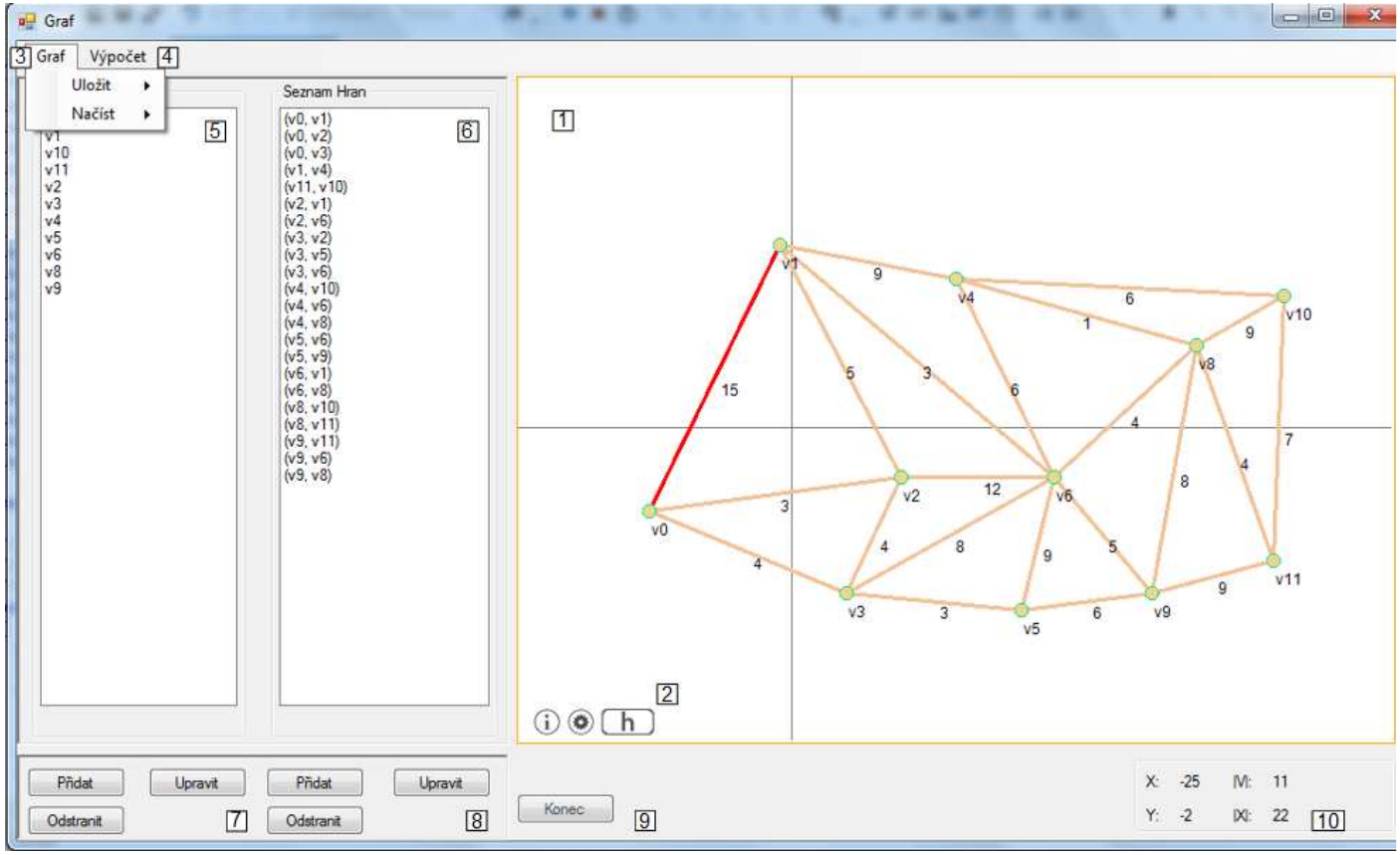
Přílohy

Příloha A

Vlastnost	Komentář
GrafickaEditaceStruktury	Umožní grafickou editaci struktury grafu (Komponenta komunikuje s okolím pomocí událostí).
VykreslovaniAktivnihoPrvku	Umožní grafické vykreslování aktivního prvku grafu dle platného režimu práce.
AutomatickeUkladaniNacitaniKonfigurace	Umožní automatické ukládání konfiguračních údajů při jejich změně pomocí konfiguračního okna komponenty do konfiguračního souboru dle vlastnosti CestaKeKonfiguracnimuSouboru. Dále umožňuje automatické načítání z tohoto souboru při startu aplikace.
CestaKeKonfiguracnimuSouboru	Cesta ke konfiguračnímu souboru komponenty s koncovkou *.bin. V případě, že soubor neexistuje, použije se implicitní konfigurace komponenty.
BarvaOsy	Barva osy komponenty.
VnejsiBarvaVrcholu	Vnější barva vrcholu.
VnitrniBarvaVrcholu	Vnitřní barva vrcholu.
BarvaHrany	Barva hrany.
BarvaTextuVrcholu	Barva textu vrcholu.
BarvaTextuHrany	Barva textu hrany.
BarvaAktivnihoPrvku	Barva aktivního prvku grafu.
VelikostVrcholu	Velikost vrcholu v pixelech.
FontVrcholu	Font vrcholu.
ZobrazitTextVrcholu	Zobrazení (vykreslení) textu vrcholu.
TloustkaHrany	Tloušťka hrany v pixelech.
OhodnoceniHrany	Ohodnocení hrany.
FontHrany	Font hrany.
ZobrazitTextHrany	Zobrazení (vykreslení) textu hrany.
RezimPrace	Režim práce komponenty.

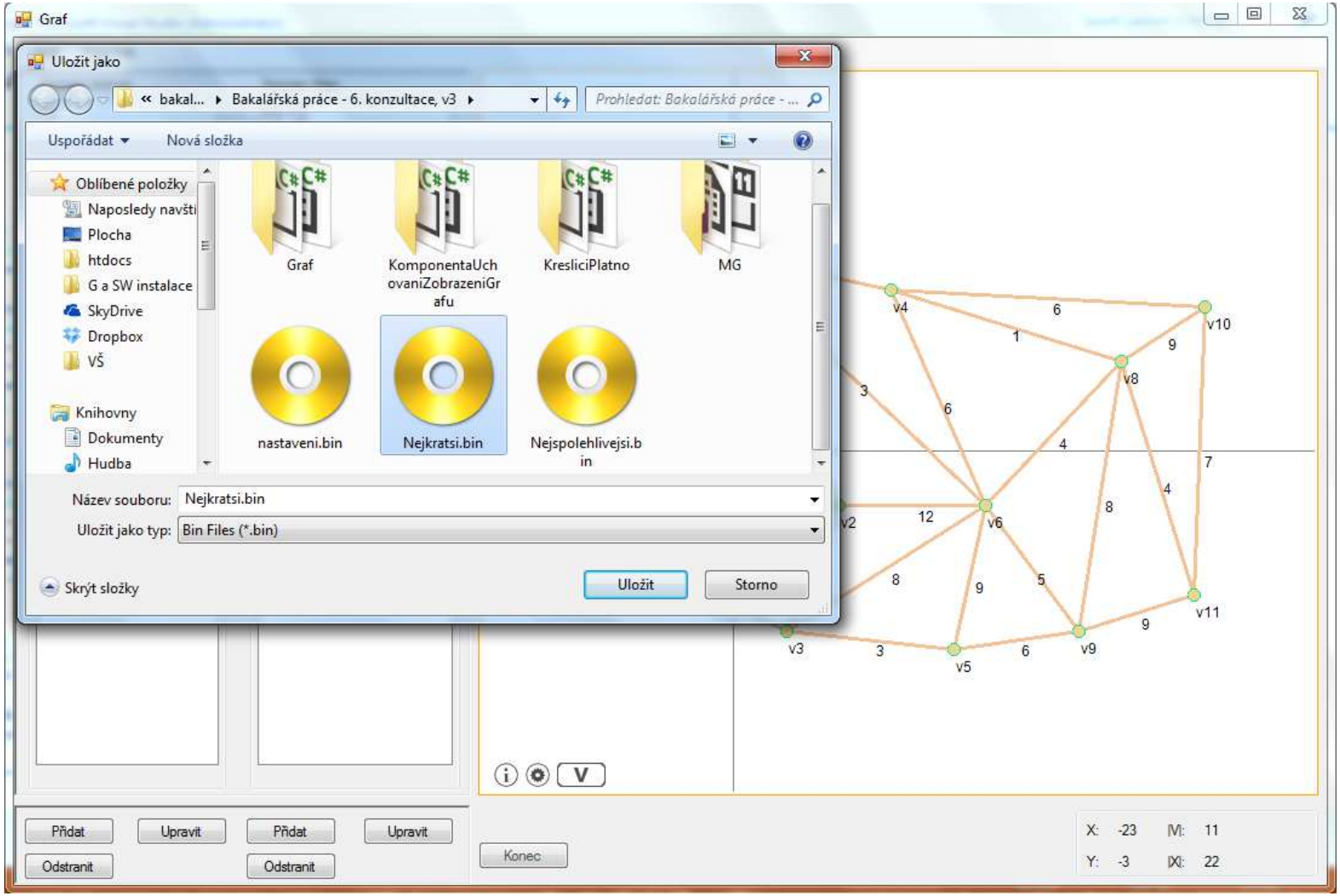
Vlastnost	Komentář
Graf	Graf, který se vykresluje, edituje a uchovává.
SouradnySystem	Systém vykreslování objektů na kreslící plátno. Obsahuje základní prostředky pro přepočty souřadnic mezi reálným SS a pixelovým SS.
SouradniceKurzoru	Aktuální souřadnice kurzoru (reálné souřadnice, zaokrouhlené na celé číslo).
AktivniPrvek	Aktivní prvek grafu dle platného režimu práce, nad nímž se nachází kurzor.

Příloha B

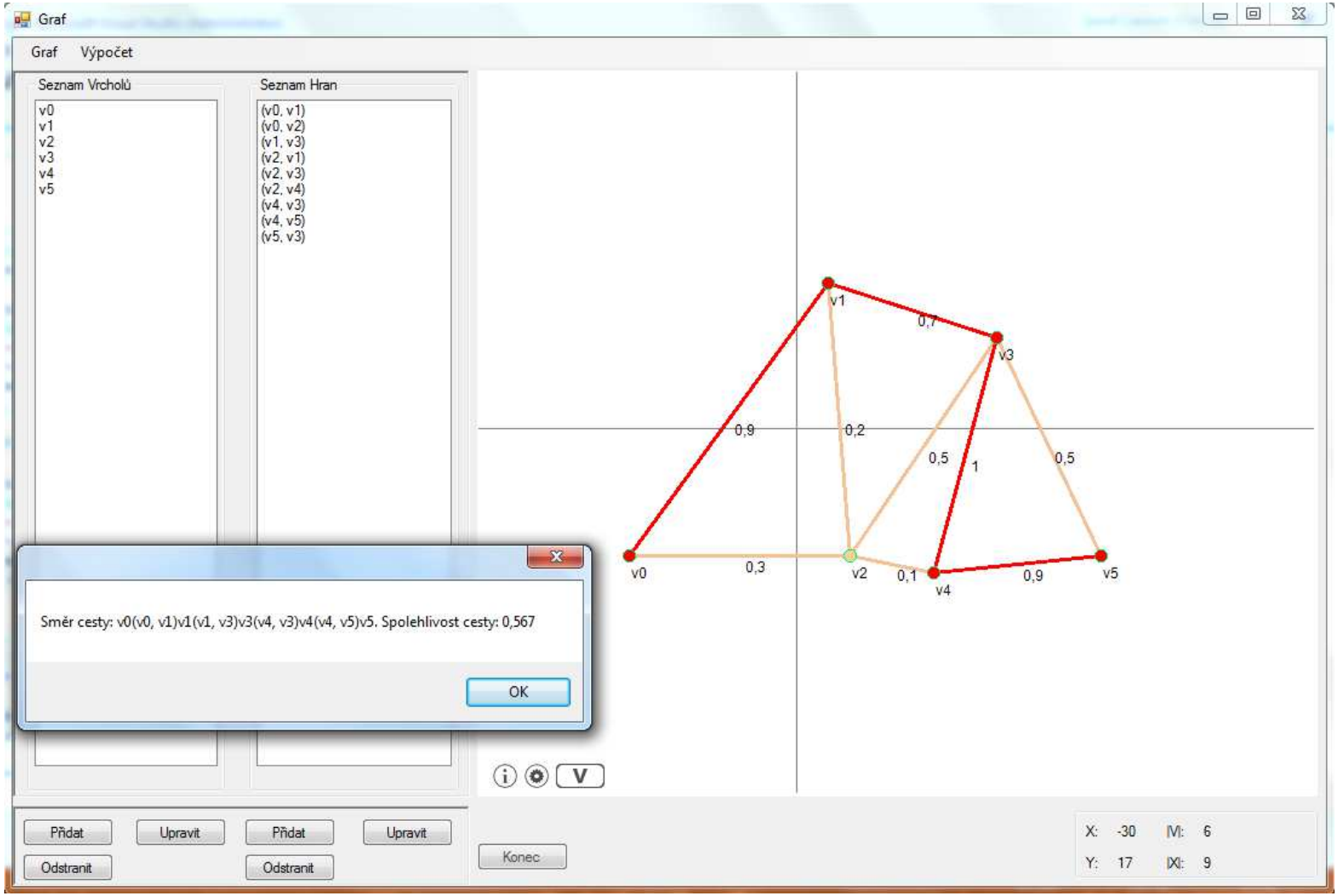


1. Komponenta
2. Tlačítka komponenty (nápověda, konfigurace, režim práce)
3. Uložení a načtení grafu
4. Výpočet nejkratší a nejspolehlivější cesty
5. Seznam vrcholů grafu
6. Seznam hran grafu
7. Editace vybraného vrcholu
8. Editace vybrané hrany
9. Ukončení aplikace
10. Souřadnice kurzoru, mohutnost množin hran a vrcholů grafu

Příloha C


Graf
Uložit jako
« bakal... ▸ Bakalářská práce - 6. konzultace, v3 ▸
Prohledat: Bakalářská práce - ...
Uspořádat
Nová složka
Oblíbené položky
Naposledy navšti
Plocha
htdocs
G a SW instalace
SkyDrive
Dropbox
VŠ
Knihovny
Dokumenty
Hudba
Graf
KomponentaUchovaniZobrazeniGrafu
KresliciPlatno
MG
nastaveni.bin
Nejkratsi.bin
Nejspolehlivejsi.bin
Název souboru: Nejkratsi.bin
Uložit jako typ: Bin Files (*.bin)
Skrýt složky
Uložit
Storno
v1
v2
v3
v4
v5
v6
v8
v9
v10
v11
Přidat
Upravit
Přidat
Upravit
Odstranit
Odstranit
Konec
X: -23
Y: -3
|V|: 11
|X|: 22

Příloha D


Graf
Graf
Výpočet
Seznam Vrcholů
v0
v1
v2
v3
v4
v5
Seznam Hran
(v0, v1)
(v0, v2)
(v1, v3)
(v2, v1)
(v2, v3)
(v2, v4)
(v4, v3)
(v4, v5)
(v5, v3)
Směr cesty: v0(v0, v1)v1(v1, v3)v3(v4, v3)v4(v4, v5)v5. Spolehlivost cesty: 0,567
OK
V
Přidat
Upravit
Přidat
Upravit
Odstranit
Odstranit
Konec
X: -30
Y: 17
|V|: 6
|X|: 9
v0
v1
v2
v3
v4
v5
0,9
0,2
0,7
0,5
1
0,5
0,3
0,1
0,9

Příloha E

CD-ROM se spustitelným souborem aplikace.