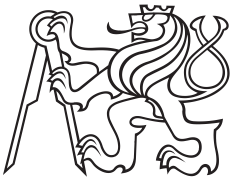
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Míra podobnosti kořenových stromů

Petr Vondrus

Studijní program Softwarové inženýrství a technologie

Květen 2018
Vedoucí práce: Ing. Jiří Šebek

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vondrus** Jméno: **Petr** Osobní číslo: **425619**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Míra podobnosti kořenových stromů

Název bakalářské práce anglicky:

Measure of similarity of rooted trees

Pokyny pro vypracování:

V současnosti je uživatelské rozhraní (UI) velmi důležité a jeho primárním cílem je nabídnout nejlepší komfort uživateli. Vzniká tedy problém s rostoucím počtem různých uživatelů se zvláštními požadavky a schopností.
[4] Framework řeší problém generování různých menu aplikací pro různé uživatele. Každé menu můžeme popsat datovou strukturou stromu.
Cíle pro tuto práci jsou:
1. Prostudujte možnosti algoritmického porovnání podobnosti dvou grafů, zejména ze třídy kořenových stromů.
2. Navrhněte a implementujte metodu pro klasifikaci množiny kořenových stromů.
3. Navrhněte a implementujte metody pro výpočet míry podobnosti dvou kořenových stromů.
4. Porovnejte vlastnosti a výsledky navržených metod.

Seznam doporučené literatury:

[1] J. Demel - Grafy a jejich aplikace, Academia, 2002
[2] G. Chartrand, L. Lesniak, P. Zhang - Graphs and Digraphs, CRC Press, 2016
[3] W. Kocay, D. L. Kreher - Graphs, Algorithms, and Optimization, CRC Press, 2005
[4] Šebek, J. - Richta, K.: Usage of Aspect-Oriented programming in Adaptive Application Structure. New Trends in Databases and Information Systems: ADBIS 2016 Short Papers and Workshops, BigDap, DCSA, DC, Prague, Czech Republic, August 28-31, 2016, Proceedings

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, laboratoř inteligentního testování softwaru FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.01.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

Poděkování / Prohlášení

Rád bych poděkoval vedoucímu práce Ing. Jiřímu Šebkovi za jeho rady, trpělivost a čas, který mi věnoval při řešení dané problematiky.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. května 2018

.....

Abstrakt / Abstract

Tato práce se zabývá návrhem metod pro nalezení míry podobnosti dvou grafů, speciálně kořenových stromů. Uvádí známé postupy, které mohou být pro výpočet této míry použity. Blíže se věnuje algoritmům využívajícím techniky Edit Distance a SimHash a popisuje jejich implementaci v experimentální srovnávací aplikaci SimCom. Její výsledky ukazují, že navržené metody lze pro porovnávání kořenových stromů použít.

Klíčová slova: graf, orientovaný graf, kořenový strom, podobnost, algoritmus, editační vzdálenost, hash, SimHash

The thesis deals with design of methods for finding measure of similarity of graphs, especially rooted trees. It states known approaches which could be used for calculation of this measure. It focuses on algorithms using Edit Distance and SimHash techniques and describes theirs implementation in the SimCom experimental comparative application. Its results show that designed methods can be used for comparing the rooted trees.

Keywords: graph, digraph, rooted tree, similarity, algorithm, edit distance, hash, SimHash

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Při řešení praktických problémů současného světa často potřebujeme zachytit vztahy mezi entitami jistého celku, popsat strukturu zkoumaného objektu nebo analyzovat posloupnost určitých stavů či procesů. Tyto situace modelujeme stále častěji pomocí grafů. Moderní teorie grafů nám totiž nabízí soubor metod a nástrojů pro jejich efektivní zvládnutí.

Stromy, resp. kořenové stromy jsou kategorie grafů se zajímavými vlastnostmi, které se mimo jiné dobře hodí k popisu hierarchických datových struktur. V praxi jsou základem např. XML dokumentů, vyhledávacích a rozhodovacích stromů nebo souborových systémů.

Práce se zabývá možnostmi algoritmického porovnávání dvou grafů kategorie kořenových stromů s cílem navrhnout a implementovat různé metody pro výpočet míry jejich podobnosti. Tyto metody by mohly být obecně využity pro porovnávání datových objektů transformovaných do této kategorie grafů např. v pojišťovnictví, dataminigu, při detekci plagiátů nebo hledání velmi podobných objektů. Původní motivací práce je však plánovaná integrace jejích výsledků do budoucího Adaptive Application Structure (AAS) Frameworku[1].

Teorie grafů jako součást moderní diskrétní matematiky nachází široké uplatnění v současné počítačové vědě a informatice. Úvod do této problematiky je podrobně vysvětlen v bezpočtu publikací, např. [2], [3], [4] nebo [5]. Pro potřeby tohoto dokumentu se předpokládá, že čtenář je již s těmito základy dobře seznámen a stejně tak zná běžné pojmy a postupy z informatiky.

Kapitola 2
Podobnost a její výpočet

Při algoritmickém porovnávání dvou datových objektů nebývá problém detekovat shodu či rozdíl, ale relativně rychle určit míru této shody, resp. rozdílu. Metody klasického porovnávání jsou v případě rozsáhlých objektů velmi neefektivní a pomalé. Navíc v mnoha případech postačuje s dostatečnou přesností míru podobnosti odhadnout.

Většina moderních algoritmů (např. [6], [7]) provádí při výpočtu míry podobnosti tento obecný postup:

1. Transformace objektů do vhodného metrického prostoru[8]
2. Výpočet jejich vzdálenosti v tomto prostoru pomocí zvolené metriky

Cílem je vyjádřit podobnost dvou datových objektů A, B takovou funkcí

$$sim(A, B) \in [0 - 1], \tag{1}$$

aby její funkční hodnota rostla s mírou shody datových objektů.

2.1 Používané metriky

Zde jsou některé často používané metriky[1]) pro stanovení míry podobnosti:

- V prostoru $\mathbb{R}^n$
  - Euklidovská vzdálenost
  - Manhattanská vzdálenost
- V diskrétním metrickém prostoru
  - Hammingova vzdálenost
  - Levenštejnova vzdálenost

2.2 Stromové algoritmy

Pro porovnání kořenových stromů lze použít metody obsahující algoritmy, které jsou založeny na *editační vzdálenosti* (angl. edit distance), např.:

- Selkow's algorithm[9]
- Chawathe's algorithm[10]
- Tekli's algorithm[11]

[1]) Dále je uváděn též formálně nepřesný pojem *vzdálenost*, protože je obvyklejší.

Kapitola 3
Metoda Edit Distance

3.1 Stručný popis

Navržená metoda vychází z postupu, který zveřejnili Matthias Dehmer a kol. [6][12]. Je založena na předpokladu, že strukturální podobnost dvou kořenových stromů je úměrná jejich vzájemné podobnosti v jednotlivých úrovních.

Metoda provádí průchod oběma porovnávanými stromy po úrovních od kořenů níže. Typově shodné atributy vrcholů ležících ve stejné hloubce jsou převedeny na posloupnost.

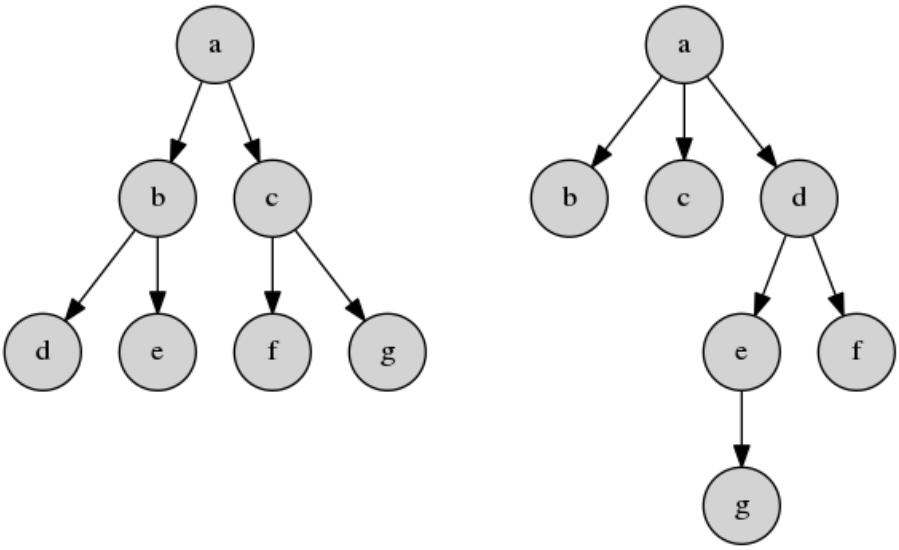
Podobnost dvou posloupností ze stejných úrovní je stanovena výpočtem jejich tzv. *editační vzdálenosti* (*Levenštejnovy vzdálenosti* [13]). Tím jsou vytvořeny množiny hodnot vyjadřujících atributovou podobnost stromů v příslušných úrovních.

Atributové podobnosti na stejné úrovni jsou dále váhově sloučeny do jediné hodnoty. Pro každou úroveň tak zůstane jedna složená hodnota podobnosti.

Výsledná hodnota celkové podobnosti je pak vypočítána jako upravený aritmetický průměr složených hodnot podobností.

3.2 Podrobné vysvětlení

Je dána dvojice grafů G_1 a G_2 z Obrázku 3.1. a pro nalezení míry jejich podobnosti bude použita metoda Edit Distance.



Obrázek 3.1. Graf G_1 (vlevo) a graf G_2 (vpravo).

Oba grafy mají stejný počet vrcholů ($|V(G_1)| = |V(G_2)|$), ale různou hloubku ($depth(G_1) = 3, depth(G_2) = 4$).

Každý vrchol má tři atributy:

- vstupní stupeň *(indegree)*
- výstupní stupeň *(outdegree)*
- označení *(label)*

Existence atributů *indegree* a *outdegree* vyplývá z definice orientovaného grafu a tyto atributy jsou mandatorní. Podle potřeby konkrétního datového modelu je možné přidání dalších atributů (zde *label*).

3.2.1 Sekvence atributů

Budiž množina atributů $A = \{a_1, a_2, \ldots, a_k\}$ $k \in \mathbb{N}$, společných grafům G_1, G_2. Typově shodné atributy vrcholů stejné úrovně l jsou spojeny do jediného datového objektu – sekvence $s_{G,a,l}$. Z typově shodných sekvencí jsou vytvořeny množiny $S_{G,a,l}$

$$\begin{aligned} s_{G_1,a,l} &= (a_{V(G_1,l,1)}, a_{V(G_1,l,2)}, \ldots, a_{V(G_1,l,w)}), \\ S_{G_1,a,l} &= \{s_{G_1,a,1}, s_{G_1,a,2}, \ldots, s_{G_1,a,l}\}, a \in A, l < depth(G_1), w \leq |V(G_1,l)| \\ s_{G_2,a,l} &= (a_{V(G_2,l,1)}, a_{V(G_2,l,2)}, \ldots, a_{V(G_2,l,w)}), \\ S_{G_2,a,l} &= \{s_{G_2,a,1}, s_{G_2,a,2}, \ldots, s_{G_2,a,l}\}, a \in A, l < depth(G_2), w \leq |V(G_2,l)| \end{aligned} \tag{1}$$

Pokud některý graf neobsahuje vrcholy v úrovni l, bude příslušnou sekvenci představovat prázdná množina. Pro grafy G_1, G_2 budou vytvořeny tyto sekvence

$$\begin{aligned} &s_{G_1,indegree,0} = (0), s_{G_1,outdegree,0} = (2), s_{G_1,label,0} = (a) \\ &s_{G_1,indegree,1} = (1,1), s_{G_1,outdegree,1} = (2,2), s_{G_1,label,1} = (b,c) \\ &s_{G_1,indegree,2} = (1,1,1,1), s_{G_1,outdegree,2} = (0,0,0,0), s_{G_1,label,2} = (d,e,f,g) \\ &s_{G_1,indegree,3} = (\emptyset), s_{G_1,outdegree,3} = (\emptyset), s_{G_1,label,3} = (\emptyset) \end{aligned}$$

$$\begin{aligned} &s_{G_2,indegree,0} = (0), s_{G_2,outdegree,0} = (3), s_{G_2,label,0} = (a) \\ &s_{G_2,indegree,1} = (1,1,1), s_{G_2,outdegree,1} = (0,0,2), s_{G_2,label,1} = (b,c,d) \\ &s_{G_2,indegree,2} = (1,1), s_{G_2,outdegree,2} = (1,0), s_{G_2,label,2} = (e,f) \\ &s_{G_2,indegree,3} = (1), s_{G_2,outdegree,3} = (0), s_{G_2,label,3} = (g) \end{aligned}$$

Z uvedeného je zřejmé, že s výjimkou extrémně rozsáhlých stromů je vhodným algoritmem pro průchod *prohledávání do šířky* (angl. Breadth-first search, zkr. BFS) s asymptotickou složitostí $O(|V| + |E|)$.

3.2.2 Editační vzdálenost

Nechť jsou podle (1) dány dvě sekvence s_1, s_2 a M je množina cenově ohodnocených editačních operací. Editační vzdálenost $distance(s_1, s_2)$ je nejmenší celková cena takové posloupnosti editačních vzdáleností, která transformuje s_1 na s_2.

Množina M obsahuje tyto editační operace:

- nahrazení prvku *(substitution)*
- odstranění prvku *(deletion)*
- vložení prvku *(insertion)*

Pro ocenění je definována funkce

$$\beta(x, y, \sigma) = 1 - e^{-\frac{1}{2}\frac{(x-y)^2}{\sigma^2}}, \qquad x, y, \sigma \in \mathbb{R}, \tag{2}$$

ze které je pro ocenění jednotlivých editačních operací odvozena funkce $\alpha(x, y)$ $x, y \in \mathbb{R}$, kde x, y jsou prvky sekvencí s_1, s_2, nebo symbol „–“ znamenající prázdné místo (*gap*)

$$\begin{aligned} substitution : \alpha_{sub}(s_1, s_2) &= \beta(s_1, s_2, 5.0) \\ deletion : \alpha_{del}(s_1, -) &= \beta(s_1, 0, 1.0) \\ insertion : \alpha_{ins}(-, s_2) &= \beta(0, s_2, 1.0) \end{aligned} \tag{3}$$

Budiž $\mathcal{M}(n, m)$ matice, kde $n = length(s_1)$, $m = length(s_2)$. Potom je výpočet editační vzdálenosti sekvencí s_1, s_2 možné provést algoritmem, který matici $\mathcal{M}$ rekurentně vyplní podle předpisu

$$\begin{aligned} &\mathcal{M}(0,0) = 0, \\ &\mathcal{M}(i,0) = \mathcal{M}(i-1,0) + \alpha_{del}(s_1[i], -), && 1 \le i \le n, \\ &\mathcal{M}(0,j) = \mathcal{M}(0,j-1) + \alpha_{ins}(-, s_2[j]), && 1 \le j \le m, \\ &\mathcal{M}(i,j) = min \begin{cases} \mathcal{M}(i-1,j) + \alpha_{del}(s_1[i], -), \\ \mathcal{M}(i,j-1) + \alpha_{ins}(-, s_2[j]), \\ \mathcal{M}(i-1,j-1) + \alpha_{sub}(s_1[i], s_2[j]) \end{cases} && i \in [1,n], j \in [1,m], \end{aligned} \tag{4}$$

Protože invariantem algoritmu je nejnižší cena na pozici $\mathcal{M}(i, j)$, platí

$$distance(s_1, s_2) = \mathcal{M}(n, m). \tag{5}$$

Při implementaci algoritmu lze s výhodou použít dynamické programování a tím snížit asymptotickou složitost klasického řešení. Jedním z prvních takových algoritmů je Needleman–Wunsch[14] zveřejněný v roce 1970, který se používá v bioinformatice ke globálnímu zarovnání sekvencí nukleotidů či aminokyselin.

Pro grafy G_1, G_2 ukazují Tabulky 3.1. – 3.4. obsahy matice $\mathcal{M}$ při výpočtu editační vzdálenosti pro všechny atributy na jednotlivých úrovních. Znaky atributu *label* jsou pro účely výpočtu převáděny na číselné hodnoty.

$\mathcal{M}$	-	0,0000
-	0,0000	0,0000
0,0000	0,0000	0,0000

indegree

$\mathcal{M}$	-	3,0000
-	0,0000	0,9889
2,0000	0,8647	0,0198

outdegree

$\mathcal{M}$	-	97,0000
-	0,0000	1,0000
97,0000	1,0000	0,0000

label

Tabulka 3.1. Obsahy matice $\mathcal{M}$ v úrovni 0.

$\mathcal{M}$	-	1,0000	1,0000	1,0000
-	0,0000	0,3935	0,7869	1,1804
1,0000	0,3935	0,0000	0,3935	0,7869
1,0000	0,7869	0,3935	0,0000	0,3935

indegree

$\mathcal{M}$	-	0,0000	0,0000	2,0000
-	0,0000	0,0000	0,0000	0,8647
2,0000	0,8647	0,0769	0,0769	0,0000
2,0000	1,7293	0,9415	0,1538	0,0769

outdegree

$\mathcal{M}$	-	98,0000	99,0000	100,0000
-	0,0000	1,0000	2,0000	3,0000
98,0000	1,0000	0,0000	1,0000	2,0000
99,0000	2,0000	1,0000	0,0000	1,0000

label

Tabulka 3.2. Obsahy matice $\mathcal{M}$ v úrovni 1.

$\mathcal{M}$	-	1,0000	1,0000
-	0,0000	0,3935	0,7869
1,0000	0,3935	0,0000	0,3935
1,0000	0,7869	0,3935	0,0000
1,0000	1,1804	0,7869	0,3935
1,0000	1,5739	1,1804	0,7869

indegree

$\mathcal{M}$	-	1,0000	0,0000
-	0,0000	0,3935	0,3935
0,0000	0,0000	0,0198	0,0198
0,0000	0,0000	0,0198	0,0198
0,0000	0,0000	0,0198	0,0198
0,0000	0,0000	0,0198	0,0198

outdegree

$\mathcal{M}$	-	101,0000	102,0000
-	0,0000	1,0000	2,0000
100,0000	1,0000	1,0000	2,0000
101,0000	2,0000	1,0000	2,0000
102,0000	3,0000	2,0000	1,0000
103,0000	4,0000	3,0000	2,0000

label

Tabulka 3.3. Obsahy matice $\mathcal{M}$ v úrovni 2.

$\mathcal{M}$	-	1,0000
-	0,0000	0,3935

indegree

$\mathcal{M}$	-	0,0000
-	0,0000	0,0000

outdegree

$\mathcal{M}$	-	103,0000
-	0,0000	1,0000

label

Tabulka 3.4. Obsahy matice $\mathcal{M}$ v úrovni 3.

3.2.3 Výpočet míry podobnosti

Na každé úrovni je vypočítána podobnost pro daný atribut takto

$$q_{a,l} = 1 - \frac{distance(s_{G_1,a,l}, s_{G_2,a,l}) + distance(s_{G_2,a,l}, s_{G_1,a,l})}{m+n}, \tag{6}$$
$$a \in A, l < max(depth(G_1), depth(G_2)), n = length(s_{G_1,a,l}), m = length(s_{G_2,a,l}).$$

Dále se všechny atributové podobnosti na stejné úrovni sloučí do jediné kompozitní hodnoty

$$q_{fin,l} = \frac{1}{k}\sum_{i=1}^{k} q_{a_i,l}, \quad a \in A, k \in \mathbb{N}, l < max(depth(G_1), depth(G_2)). \tag{7}$$

Výsledná míra podobnosti je potom vyjádřena jako

$$sim(G_1, G_2) = \rho \cdot \frac{\prod_{i=0}^{\rho-1} q_{fin,i}}{\sum_{i=0}^{\rho-1} q_{fin,i}}, \quad \rho = max(depth(G_1), depth(G_2)). \tag{8}$$

Pro grafy G_1, G_2 tedy podle (8) platí

$$sim(G_1, G_2) = 4 \cdot \frac{0,3268}{3,0988} \approx 0,4219.$$

Metodou dosažená míra podobnosti je z intervalu $[0-1]$, neleží v blízkosti jeho mezí a odpovídá subjektivně očekávané hodnotě.

Kapitola 4
Implementace metody Edit Distance

4.1 Sestavení sekvence atributů

Přestože je v Kapitole 3.2.1 uvedeno, že s výjimkou extrémně rozsáhlých stromů je vhodným algoritmem pro průchod stromem prohledávání do šířky (BFS), používá implementace této metody z důvodu vícenásobného použití kódu algoritmus DFS (Kapitola 6.3) se stejnou asymptotickou složitostí.

Na výpisu 4.1. je metoda `levelToArray` ze třídy `EditDistanceSimilarity`, jejímž výstupem jsou sekvence atributů vrcholů na stejné úrovni, které jsou transformovány do typu `double[]`.

```
private double[] levelToArray(CustomGraph graph,
                              int levelNumber,
                              Attribute attribute) {
    CustomGraphLevel level = graph.getLevel(levelNumber);

    if (level != null) {
        double[] rv = new double[level.size()];
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < rv.length; i++) {
            switch (attribute) {
                case INDEGREE:
                    rv[i] = graph.inDegreeOf(level.get(i));
                    break;

                case OUTDEGREE:
                    rv[i] = graph.outDegreeOf(level.get(i));
                    break;

                case LABEL:
                    sb.append(level.get(i).getLabel());
                    break;
            }
        }

        // Labels of vertices
        if (attribute == Attribute.LABEL) {
            String labelsString = sb.toString();
            double[] labelsArray = new double[labelsString.length()];

            for (int i = 0; i < labelsArray.length; i++) {
                labelsArray[i] = (int) labelsString.charAt(i);
            }
```

```
            rv = labelsArray;
        }
        return rv;

    } else {
        return new double[0];
    }
}
```

Výpis 4.1. Metoda `levelToArray` třídy `EditDistanceSimilarity`.

4.2 Editační vzdálenost

Výpočet editační vzdálenosti zajišťuje třída `SequenceAlignment`. Její metoda `fillMatrix` (Výpis 4.2) naplní instanční proměnnou `matrix`, kterou vrací metoda `getMinimalEditDistance` (Výpis 4.3).

```
private void fillMatrix() {
    // First column
    for (int i = 1; i < matrix.length; i++) {
        matrix[i][0] = matrix[i - 1][0] + similarityFunction
                    .getElementGapValue(sequence1[i - 1]);
    }

    // First row
    for (int j = 1; j < matrix[0].length; j++) {
        matrix[0][j] = matrix[0][j - 1] + similarityFunction
                    .getGapElementValue(sequence2[j - 1]);
    }

    // Inner of the matrix
    for (int i = 1; i < matrix.length; i++) {
        for (int j = 1; j < matrix[0].length; j++) {
            double substituteCost = matrix[i - 1][j - 1]
                   + similarityFunction
                    .getElementElementValue(sequence1[i - 1],
                                            sequence2[j - 1]);

            double insertCost = matrix[i][j - 1]
                   + similarityFunction
                    .getGapElementValue(sequence2[j - 1]);

            double deleteCost = matrix[i - 1][j]
                   + similarityFunction
                    .getElementGapValue(sequence1[i - 1]);

            matrix[i][j] = Math.min(Math.min(insertCost, deleteCost),
                                    substituteCost);
        }
    }
}
```

Výpis 4.2. Metoda `fillMatrix` třídy `SequenceAlignment`.

```
double getMinimalEditDistance() {
    return matrix[matrix.length - 1][matrix[0].length - 1];
}
```

Výpis 4.3. Metoda `getMinimalEditDistance` třídy `SequenceAlignment`.

4.3 Výpočet míry podobnosti

Finální výpočet míry podobnosti provádí metoda `evaluateSimilarity` ze třídy `EditDistanceSimilarity` (Výpis 4.4.).

```
@Override
public void evaluateSimilarity() {
    final double PARAMETER_ZETA = 1.0 / 3.0;
    int depth1 = graph1.getDepth();
    int depth2 = graph2.getDepth();
    int depth = Math.max(depth1, depth2);
    double sumOfGammaFinals = 0;
    double productOfGammaFinals = 1;

    // Iterate over graph levels
    for (int i = 0; i < depth; i++) {

        double[] sequenceIndegree1
                    = levelToArray(graph1, i, Attribute.INDEGREE);
        double[] sequenceIndegree2
                    = levelToArray(graph2, i, Attribute.INDEGREE);
        double[] sequenceOutdegree1
                    = levelToArray(graph1, i, Attribute.OUTDEGREE);
        double[] sequenceOutdegree2
                    = levelToArray(graph2, i, Attribute.OUTDEGREE);
        double[] sequenceLabel1
                    = levelToArray(graph1, i, Attribute.LABEL);
        double[] sequenceLabel2
                    = levelToArray(graph2, i, Attribute.LABEL);

        SequenceAlignment alignmentIndegree12 =
                new SequenceAlignment(sequenceIndegree1,
                                      sequenceIndegree2,
                                      new ExponentialFunction());

        SequenceAlignment alignmentOutdegree12 =
                new SequenceAlignment(sequenceOutdegree1,
                                      sequenceOutdegree2,
                                      new ExponentialFunction());

        SequenceAlignment alignmentLabel =
                new SequenceAlignment(sequenceLabel1,
                                      sequenceLabel2,
                                      new SimpleFunction());
```

Výpis 4.4. Fragment metody `evaluateSimilarity` třídy `EditDistanceSimilarity`.

Kapitola 5
Metoda SimHash

Navržená metoda používá funkci SimHash, jejímž autorem je Moses Charikar[15] a byla zveřejněna v roce 2002. Patří do kategorie hash funkcí LSH (angl. Locality-sensitive hashing), které snižují dimensionalitu vstupních dat a na rozdíl od kryptografických hash funkcí se snaží maximalizovat počet kolizí pro podobné předlohy.

Běžné, zejména kryptografické, hash funkce často rozbíjejí korelaci vstupních dat a otisku pomocí tzv. *lavinového efektu* (angl. avalanche effect), kdy malá změna vstupu (několik bitů) způsobí velkou změnu otisku (nadpoloviční počet bitů). Pro množinu vzájemně velmi podobných předloh tedy vytváří množinu zcela rozdílných otisků.

Oproti tomu SimHash náleží do skupiny *perceptuálních hashů*[16], která produkuje pro velmi podobné předlohy velmi podobné otisky.

Pro netriviální předlohy tak snižuje asymptotickou složitost, protože při jejím použití není nutné dvě předlohy porovnávat iterativním průchodem jejich vnitřní strukturou. Stačí rychle (tedy s nižší asymptotickou složitostí) spočítat jejich SimHash otisky a ty porovnat.

5.1 Stručný popis

Nejprve je zvolena hash funkce, která bude metodou používána. Tím je zároveň určena délka všech budoucích otisků. Dále je nulami inicializován celočíselný tzv. *distribuční vektor* v délce otisku.

Potom je provedena dekompozice datového objektu na atomické entity (tokeny). V případě kořenového stromu jsou těmito tokeny jeho vrcholy. Token je včetně svých vlastností předán jako vstup hash funkci, která vrátí jeho otisk. Každý takto získaný otisk je po bitech vzestupně iterován. Pokud je aktuální bit nastaven (je roven 1), je hodnota distribučního vektoru na příslušném indexu zvýšena o jednu, jinak je o jednu snížena.

Po zpracování všech tokenů je nulami inicializován výsledný SimHash a vzestupně iterován distribuční vektor. Pokud je hodnota distribučního vektoru na příslušném indexu nezáporná, je příslušný bit SimHashe nastaven na 1, jinak na 0.

Z výsledných SimHashů je za pomocí *Hammingovy vzdálenosti* a *Jaccardova indexu*[17] vypočítána míra jejich podobnosti, která je v posledním kroku metody normalizována.

5.2 Podrobné vysvětlení

Je dána dvojice grafů G_1 a G_2 z Obrázku 3.1. (Kapitola 3.2) a pro nalezení míry jejich podobnosti bude použita metoda SimHash.

5.2.1 Volba hash funkce

Délka konečného otisku (SimHashe) je rovna délce otisku použité hash funkce. Obvykle to bývá 32 – 256 bitů v násobcích 8. Tato volba výrazně ovlivňuje výsledek algoritmu. Při použití kryptografické hash funkce MD5[18] [1]) bude délka otisku 128 bitů.

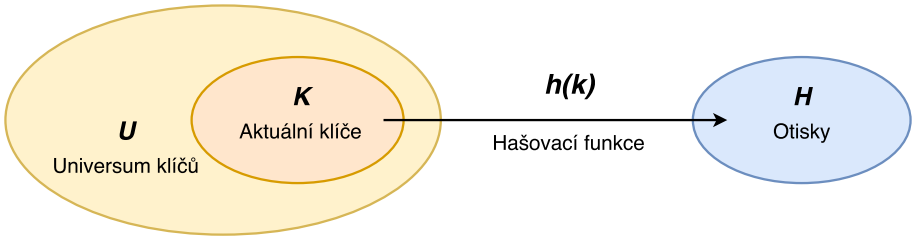
```
n = 128
```

Výpis 5.1. Zvolená délka otisku.

Protože pojem hash funkce je klíčový pro celou metodu, bude nyní krátce vysvětlen.

Rozptylovací funkce[19] (angl. hash function, dále jen *hash funkce*) je takové neprosté zobrazení, které klíči k z relativně velké množiny U přiřadí *otisk* (angl. fingerprint) z relativně malé množiny H. Pro hash funkci platí

$$\begin{aligned} K &\in U \\ |U| &\gg |K| \\ |K| &\cong |H| \end{aligned} \tag{1}$$



Obrázek 5.1. Hash funkce.

Vlastnosti hash funkce:

- Minimální změna klíče způsobí velkou změnu otisku.
- Je ireversibilní, z otisku není možné vytvořit předlohu klíč.
- Pro klíče různých délek vytváří otisky konstantní délky.
- Zobrazuje klíče do množiny otisků „rovnoměrně“ (nevytváří shluky blízkých hodnot).

V praxi nacházejí hash funkce široké uplatnění. Používají se pro rychlé vyhledávání pomocí hash tabulek, detekci datové integrity nebo v kryptografii.

5.2.2 Rozdělení předlohy

Objekt předlohy je rozdělen na elementární datové entity (tzv. *tokeny*), které se stanou prvky množiny tokenů T. Pro grafy (předlohy) G_1 a G_2 tak vzniknou množiny vrcholů T_1 a T_2. Pro dekompozici stromu lze použít např. algoritmus *prohledávání do hloubky* (angl. Depth-first search, zkr. DFS) s asymptotickou složitostí $O(|V| + |E|)$.

```
T1 = { G1(V1), G1(V2), ..., G1(V7) }
T2 = { G2(V1), G2(V2), ..., G2(V7) }
```

Výpis 5.2. Rozdělení předloh na tokeny.

[1]) Skutečnost, že MD5 byla kompromitována, pro účely této metody nevadí.

5.2.3 Použití hash funkce

Na každý prvek t z množiny tokenů T je uplatněna zvolená hash funkce. Do funkce vstupuje token jako datový objekt včetně všech svých atributů (Kapitola 3.2). Tím se vytvoří množina otisků H. Pro množiny tokenů T_1 a T_2 tak vzniknou množiny otisků H_1 a H_2. Pořadí bytů je Little-Endian.

token	MD5 [hex]	MD5 [bin]
G1(V1)	0x951F858C648890F26821736EBA119918	0b10010101...00011000
G1(V2)	0x25E162383CB134DE7DDBDCE7968D010F	0b00100101...00001111
G1(V3)	0xF641497F8913A15C0BE11C3F7661B024	0b11110110...00100100
G1(V4)	0x6E3FF0236E9AD7169D4FF4A2EF415CF6	0b01101110...11110110
G1(V5)	0xFBD40A8D44F994A0B333493959C22920	0b11111011...00100000
G1(V6)	0x2337962C5A1838962921EFAEC5D3D4C0	0b00100011...11000000
G1(V7)	0xC57B46DBB7FDC26D7E2140492D911751	0b11000101...01010001

Tabulka 5.1. Tabulka MD5 otisků tokenů z množiny T_1.

token	MD5 [hex]	MD5 [bin]
G2(V1)	0xFFA060F2AD8B526D9B9C13608BA6D285	0b11111111...10000101
G2(V2)	0x0C27794DF0C7B6B887D2C5D7E1BE3CBC	0b00001100...10111100
G2(V3)	0xFD94918892734BC67BDC2737606E57D2	0b11111101...11010010
G2(V4)	0x6C30B3215C44C7FD8963ED09A18019F3	0b01101100...11110011
G2(V5)	0xB3F9DFA1968DD9B44BDAB297B15336E9	0b10110011...11101001
G2(V6)	0x2337962C5A1838962921EFAEC5D3D4C0	0b00100011...11000000
G2(V7)	0xFB6D0B3731397FA20E030B3D9771EE06	0b11111011...00000110

Tabulka 5.2. Tabulka MD5 otisků tokenů z množiny T_2.

```
H1 = { 0x951F858C.., 0x25E16238.., ..., 0xC57B46DB }
H2 = { 0xFFA060F2.., 0x0C27794D.., ..., 0xFB6D0B37 }
```

Výpis 5.3. Množiny otisků tokenů.

5.2.4 Distribuční vektor

Je deklarováno pole celočíselných proměnných A s délkou n, které je inicializováno nulami. Poté je obsah pole vyplněn takto

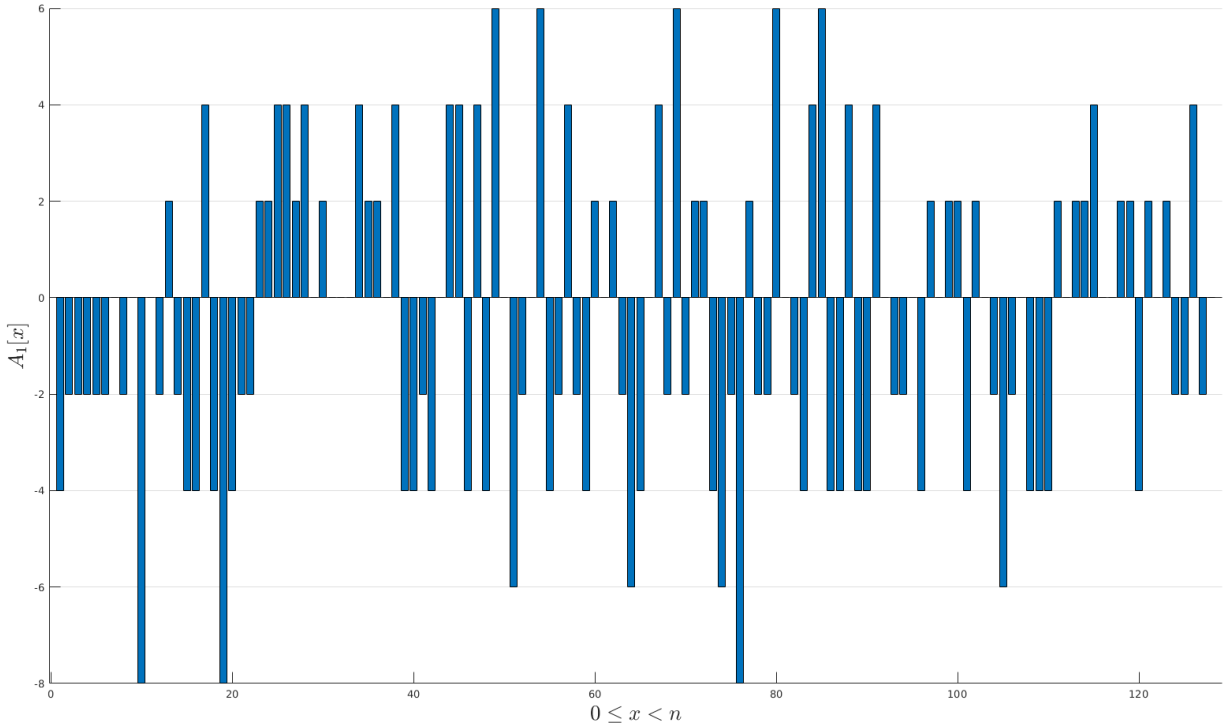
$$h \in H, t \in T, i \in \mathbb{N}, 1 \leq i \leq n$$
$$A[i] = \begin{cases} A[i] - weight(t) & \text{pro } bit(h,i)=0, \\ A[i] + weight(t) & \text{pro } bit(h,i)=1. \end{cases} \tag{2}$$

Každý token může mít celočíselnou váhu odpovídající nějaké jeho vlastnosti (např. četnosti výskytu), což zajišťuje váhová funkce *weight(t)*. Tím je možné zvýšit přesnost algoritmu pro určité typy předloh (zde $weight(t) = 1$).

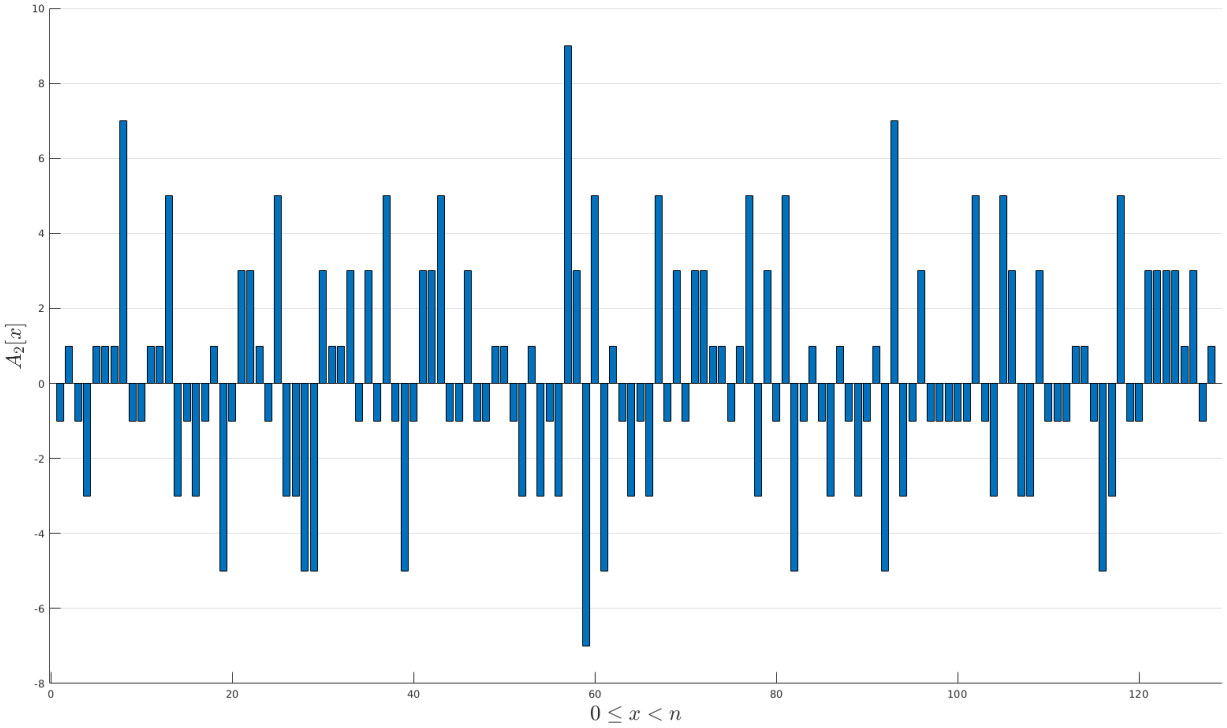
Pro množiny otisků H_1 a H_2 tak budou vytvořena dvě celočíselná pole A_1 a A_2.

```
A1 = [ -4, -2, -2, -2, -2, -2, 0, -2, 0, -8, 0, -2, 2, -2, -4, ..., 0 ]
A2 = [ -1, 1, -1, -3, 1, 1, 1, 7, -1, -1, 1, 1, 5, -3, -1, -3, ..., 1 ]
```

Výpis 5.4. Distribuční vektory množin otisků tokenů.



Obrázek 5.2. Graf distribučního vektoru A_1 pro množinu otisků H_1.



Obrázek 5.3. Graf distribučního vektoru A_2 pro množinu otisků H_2.

5.2.5 Vytvoření SimHashe

Je deklarováno bitové pole Q s délkou n, které je vyplněno takto

$$i \in \mathbb{N}, 1 \leq i \leq n$$
$$Q[i] = \begin{cases} 1 & \text{pro } A[i] > 0, \\ 0 & \text{jinak.} \end{cases} \tag{3}$$

Q je výsledným otiskem, resp. výstupem hash funkce SimHash.

Pro distribuční vektory A_1 a A_2 tak budou vytvořena bitová pole Q_1 a Q_2.

```
Q1 = [ 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, ..., 0 ]
Q2 = [ 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, ..., 0 ]

Q1 = 0xA77FC46F4C9990D639335C3FFFC11540
Q2 = 0xBF23132094495BD42B132715E1721CF2
```

Výpis 5.5. Výsledné otisky Q_1, Q_2 pro grafy G_1, G_2 (SimHashe).

5.2.6 Výpočet míry podobnosti

Hammingova vzdálenost[20] (angl. Hamming distance) je metrikou v prostoru textových řetězců a představuje počet míst, ve kterých se dva objekty stejné datové struktury liší, nebo také počet korekcí, které je potřeba provést pro změnu jednoho objektu na druhý. Pro dva objekty x a y s délkou n ji můžeme vyjádřit vztahem

$$D_H = \sum_{i=1}^{n} |x_i - y_i| \tag{4}$$

```
x = "Jarka a Hana nosily trávu."
y = "Jirka a Jana kosili trávu."
```

$D_H = 4$

Obrázek 5.4. Příklad Hammingovy vzdálenosti dvou řetězců s délkou 26 znaků.

```
x = 0b11010100101011100001011000101100
y = 0b10010110101011000101011000101101
```

$D_H = 5$

Obrázek 5.5. Příklad Hammingovy vzdálenosti binárních sekvencí s délkou 32 bitů.

Předposledním krokem metody je výpočet míry podobnosti předloh na základě jejich SimHash otisků. To je provedeno pomocí Hammingovy vzdálenosti, která byla krátce vysvětlena a *Jaccardova indexu*[17]

$$sim(G_1, G_2) = \frac{|G_1 \cap G_2|}{|G_1 \cup G_2|} \tag{5}$$

(pro $G_1 = \emptyset, G_2 = \emptyset$ definujeme $sim(G_1, G_2) = 1$), kde jsou místo předloh G_1, G_2 použity jejich charakterizující SimHash otisky Q_1, Q_2

$$sim(G_1, G_2) = \frac{|Q_1 \cap Q_2|}{|Q_1 \cup Q_2|} \tag{6}$$

S použitím (4) potom platí

$$\begin{aligned} |Q_1 \cap Q_2| &= \overline{D}_H \\ |Q_1 \cup Q_2| &= n \end{aligned} \tag{7}$$

a míra podobnosti předloh G_1, G_2 je tedy dána vztahem

$$sim(G_1, G_2) = \frac{\overline{D}_H}{n} \tag{8}$$

Dosadíme-li do (8), lze pro uvedený příklad s použitím logické funkce XOR vypočítat

$$sim(G_1, G_2) = \frac{n - \sum_{i=1}^{n} (Q_{1i} \quad XOR \quad Q_{2i})}{n} = \frac{128 - 57}{128} \approx 0,5547.$$

5.2.7 Normalizace výsledku

Vypočítaná míra podobnosti grafů G_1, G_2 z Kapitoly 5.2.6 sice leží v intervalu $[0 - 1]$ a intuitivně se blíží očekávání, přesto není zcela objektivní. Jde o logickou vlastnost uvedeného postupu, který vnáší do výsledků zkreslení v intervalu $[0 - 0,6)$ způsobené pravděpodobností. Dále bude popsán důvod tohoto zkreslení a navržena vhodná normalizace.

Nechť s_1 a s_2 jsou dvě náhodné binární sekvence s délkou n bitů a X diskrétní náhodná veličina představující stejné hodnoty bitů na stejných místech (indexech) v těchto sekvencích.

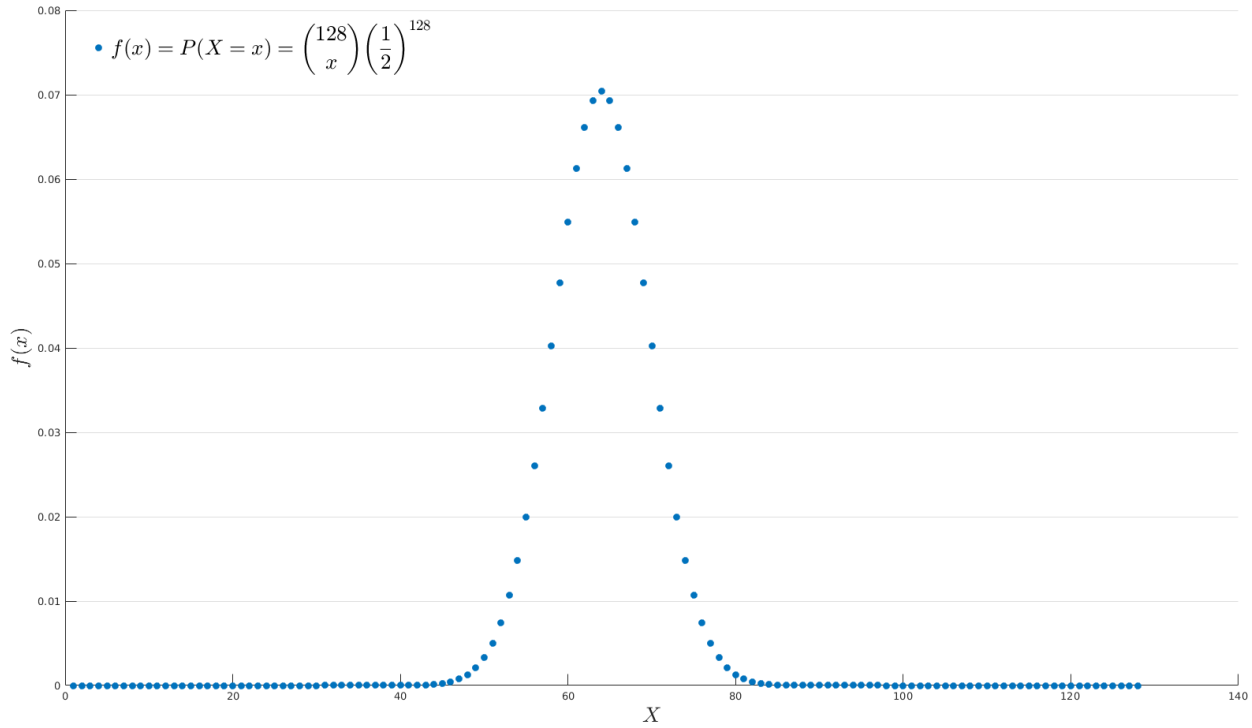
Je zřejmé, že X má *binomické rozdělení* a pro její pravděpodobnostní funkci *f(x)* a distribuční funkci *F(x)* platí

$$\begin{aligned} f(x) = P(X = x) &= \binom{n}{x} p^x \ (1-p)^{n-x}, \\ F(x) = P(X \leq x) &= \sum_{x=1}^{n} \binom{n}{x} p^x \ (1-p)^{n-x}, \end{aligned} \tag{9}$$

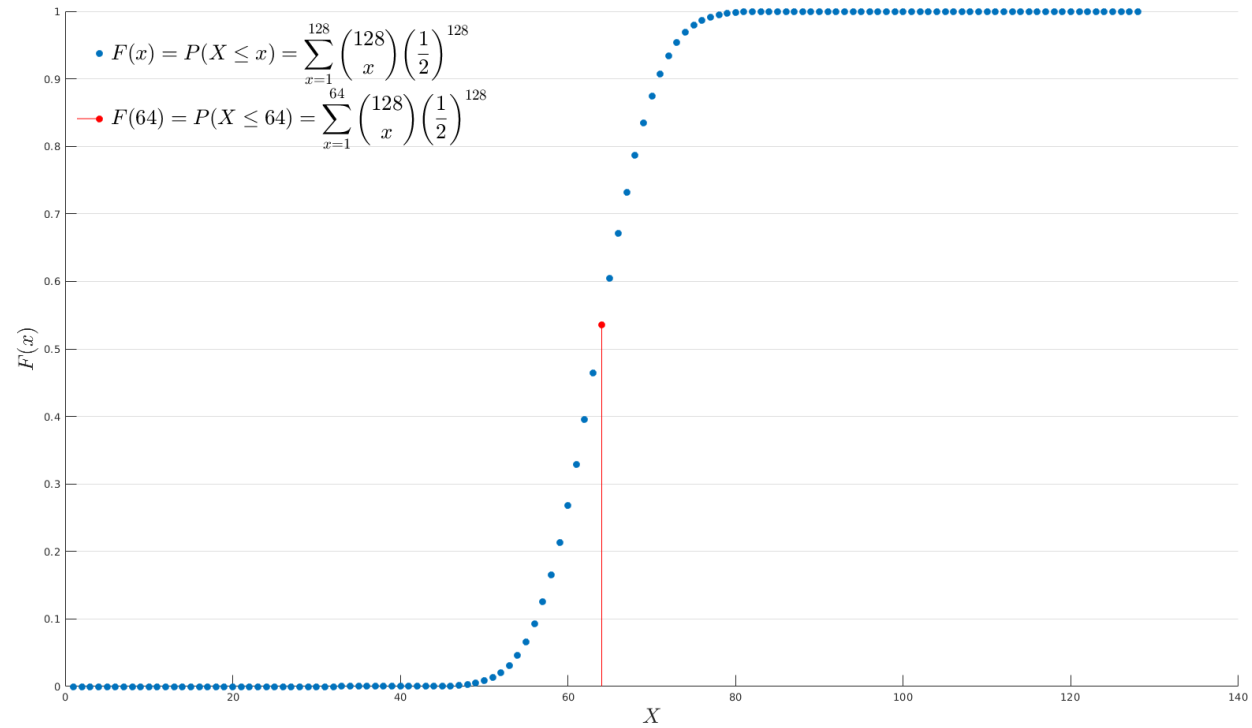
kde n je délka sekvencí (v bitech), x je počet stejných hodnot bitů na stejných místech (indexech) obou sekvencí a p je pravděpodobnost, že jeden bit bude mít hodnotu 0, resp. 1.

Pro $p = \frac{1}{2}$ tedy platí

$$\begin{aligned} f(x) = P(X = x) &= \binom{n}{x} \left(\frac{1}{2}\right)^n, \\ F(x) = P(X \leq x) &= \sum_{x=1}^{n} \binom{n}{x} \left(\frac{1}{2}\right)^n, \end{aligned} \tag{10}$$



Obrázek 5.6. Graf hustoty pravděpodobnosti náhodné veličiny X pro $n = 128$ a $p = \frac{1}{2}$.



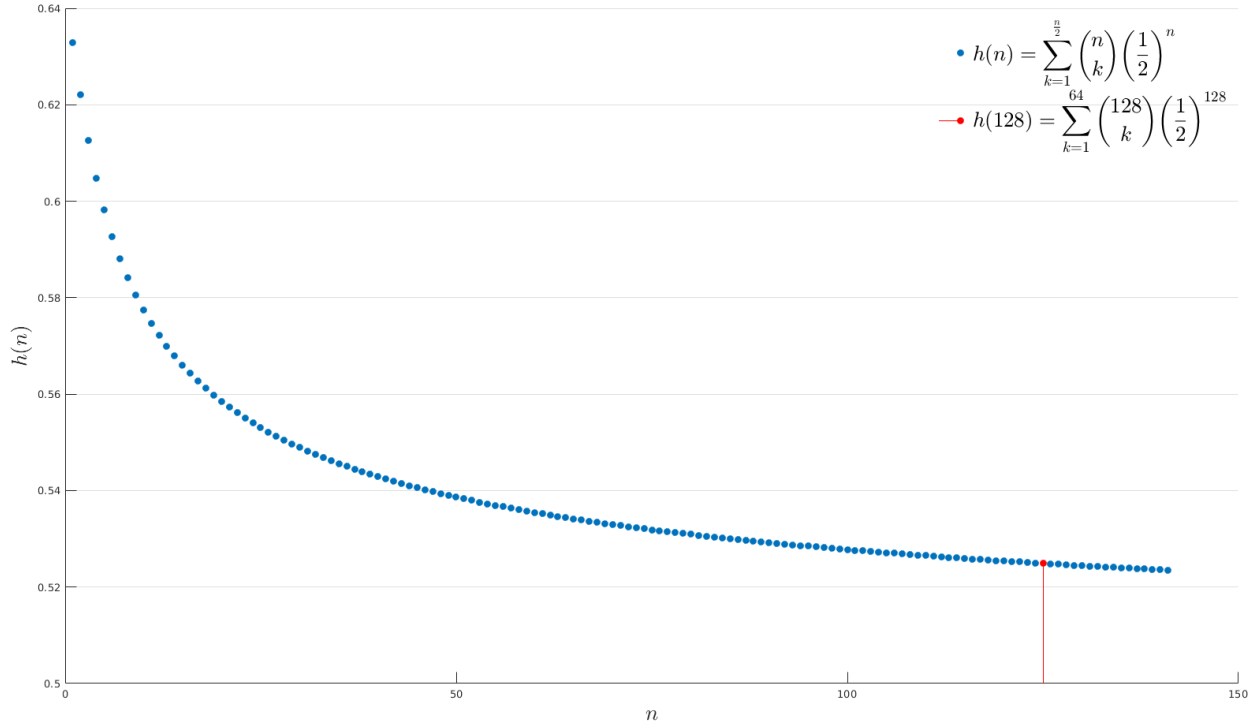
Obrázek 5.7. Graf distribuční funkce náhodné veličiny X pro $n = 128$ a $p = \frac{1}{2}$.

Jak je patrné z grafu na Obrázku 4.7., pravděpodobnost, že dvě náhodné binární sekvence s délkou 128 bitů budou mít shodnou nejvýše polovinu bitů na stejných pozicích, je

$$F(64) = P(X \leq 64) = \sum_{x=1}^{64} \binom{128}{x} \left(\frac{1}{2}\right)^{128} \approx 0,5352,$$

což odpovídá horní mezi intervalu zkreslení, kterým je zatížen výsledek z Kapitoly 5.2.6.

Obrázek 4.8. ukazuje závislost horní meze intervalu zkreslení na délce otisku (SimHashe) a naznačuje konvergenci řady $h(n)$ k očekávané hodnotě $\frac{1}{2}$.



Obrázek 5.8. Graf závislosti zkreslení výsledku na délce otisku (SimHashe).

S použitím (4), (8) a (10) je možné provést normalizaci výsledku takto

$$sim(G_1, G_2) = \begin{cases} \frac{\overline{D}_H}{n} - F(D_H) & \text{pro } D_H \leq \frac{n}{2}, \\ \frac{\overline{D}_H}{n} - F(n - D_H) & \text{jinak.} \end{cases} \tag{11}$$

Dosadíme-li do (11), bude výsledná normalizovaná míra podobnosti grafů G_1, G_2

$$sim(G_1, G_2) = \frac{128 - 57}{128} - 0,1252 \approx 0,4295.$$

Metodou dosažená míra podobnosti je z intervalu $[0 - 1]$, neleží v blízkosti jeho mezí a odpovídá subjektivně očekávané hodnotě. Navíc se jen velmi málo liší (rozdíl 0,0076) od hodnoty dosažené metodou Edit Distance (Kapitola 3.2.3).

Kapitola 6
Implementace metody SimHash

Implementace metody SimHash v zásadě odpovídá popisu v Kapitole 5. Operace rozdělení předlohy (Kapitola 5.2.2) a použití zvolené hash funkce (Kapitola. 5.2.3) lze sloučit do jednoho algoritmického celku.

6.1 Dekompozice stromu

V případě stromu se možnosti rozložení na jednotlivé tokeny nabízí již ze samotné definice grafu. Lze uvažovat o rozdělení na základě vrcholů, hran, případně jejich vhodné kombinace. Za jednotlivé tokeny byly zvoleny vrcholy stromu, protože na rozdíl od hrany i jediný vrchol (v tomto případě kořen) tvoří graf.

```
public class CustomGraphVertex implements Serializable {
    private static final long serialVersionUID = 6690978526162487994L;
    public enum Status { FRESH, OPEN, CLOSED }

    private String label;
    private Status status;

    CustomGraphVertex(String label) {
        this.label = label;
        this.status = Status.FRESH;
    }
```

Výpis 6.1. Fragment třídy `CustomGraphVertex` popisující vrchol grafu.

6.2 Hash funkce

Volba hash funkce je v této metodě důležitá, protože významně ovlivňuje výstupní SimHash otisk. Návrh implementace byl veden snahou o dostatečně otevřené uspořádání příslušných tříd, aby bylo možné postupně skladbu hash funkcí měnit a porovnávat jejich výsledky.

K tomuto účelu byla použita open-source knihovna Google Guava[1]) (Apache Licence 2.0), která ve třídě `com.google.common.hash.Hashing` nabízí různé hash funkce. Z nich bylo implementováno těchto pět:

- Murmur3A
- SipHash-2-4
- FarmHash's Fingerprint64
- MD5
- SHA-256

[1]) `https://github.com/google/guava`

K integraci hash funkcí slouží třída `HashAlgorithm` (enumerátor), Sdružuje popis hash funkcí, které má algoritmus užít pro vytvoření otisků tokenů. Tato třída je zároveň jediným místem, kde je nutné hash funkci specifikovat.

```
package simcom.SimhashSimilarity;

import com.google.common.hash.Hashing;
import com.google.common.hash.HashFunction;

public enum HashAlgorithm {

    // Here is the only place to insert additional hash algorithm.
    MURMUR("Murmur3A",
           Hashing.murmur3_32(),
           "MediumBlue",
           // The length is one half of hash length (32/2, 64/2, ...)
           new double[] { 0, ..., 0.569975 }
    ),

    SIPHASH("SipHash-2-4",
            Hashing.sipHash24(),
            "ForestGreen",
            new double[] { 0, ..., 0.549673 }
    ),

    FARMHASH("FarmHash's Fingerprint64",
             Hashing.farmHashFingerprint64(),
             "Chocolate",
             new double[] { 0, ..., 0.549673 }
    ),
```

Výpis 6.2. Fragment třídy `HashAlgorithm`.

Na Výpisu 6.3. lze vidět část kódu metody `putVertex` třídy `Simhash` zodpovědnou za aplikaci zvolené hash funkce na token (vrchol). Následná aktualizace distribučního vektoru výstupem z této funkce je na Výpisu 6.4.

```
void putVertex(int indegree, int outdegree, String label, int level) {
    // Make hash
    byte[] hash = hashFunction.newHasher()
            .putInt(indegree)
            .putInt(outdegree)
            .putUnencodedChars(label)
            .putInt(level)
            .hash().asBytes();
    // Update vector
    updateVector(hash);
}
```

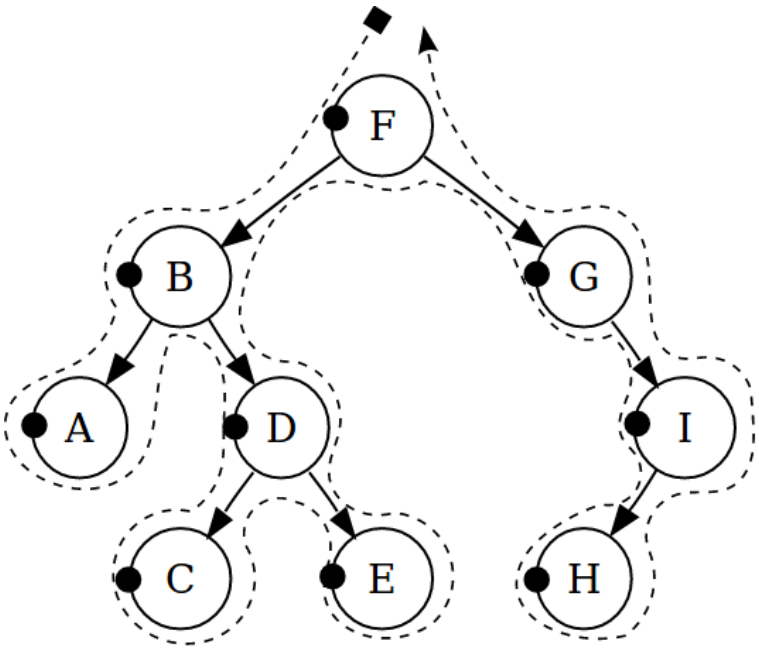
Výpis 6.3. Fragment metody `putVertex` třídy `Simhash`.

```
private void updateVector(byte[] hash) {
    for (int i = 0; i < hash.length; i++) {
        byte mask = 1;
        for (int j = 0; j < 8; j++) {
            if ((hash[i] & mask) == 0) {
                vector[i * 8 + j]--;
            } else {
                vector[i * 8 + j]++;
            }
            mask <<= 1;
        }
    }
}
```

Výpis 6.4. Fragment metody `updateVector` třídy `Simhash`.

6.3 Průchod stromem

Rozložení stromu na tokeny a následné volání hash funkcí je prováděno průchodem stromem a jeho prohledáním do hloubky algoritmem DFS (Obrázek 6.1.). Přestože knihovna JGraphT k tomuto účelu nabízí třídu `DepthFirstIterator<V,E>`, byla nakonec použita vlastní rekurzivní metoda, které lépe odpovídá potřebám implementace (Výpis 6.5.).



Obrázek 6.1. Průchod stromem pomocí algoritmu DFS.

```
private void depthFirstSearch(CustomGraphVertex vertex, int level) {
    // Mark the vertex as open
    vertex.setStatus(CustomGraphVertex.Status.OPEN);

    // If necessary add new level to the hierarchy
    if (levels.size() < level + 1) {
```

```
            levels.add(new CustomGraphLevel());
        }

        // Add the vertex to the appropriate level
        levels.get(level).add(vertex);

        level++;
        for (CustomGraphVertex successor : getVertexSuccessors(vertex))
            if (successor.getStatus() == CustomGraphVertex.Status.FRESH)
                depthFirstSearch(successor, level);
        vertex.setStatus(CustomGraphVertex.Status.CLOSED);
    }
```

Výpis 6.5. Metoda `depthFirstSearch` třídy `CustomGraph`.

6.4 Výpočet míry podobnosti

Metoda `evaluateSimilarity` třídy `SimhashSimilarity` zajišťuje finální výpočet míry podobnosti dvou vybraných stromů. Vytvoří množiny SimHashů obou stromů, zjistí jejich Hammingovu vzdálenost, vypočítá míru jejich podobnosti a provede její normalizaci.

```
@Override
public void evaluateSimilarity() {
    makeSimHashTable(graph1, simhashTable1);
    makeSimHashTable(graph2, simhashTable2);

    // Evaluate similarity
    for (HashAlgorithm hashAlgorithm : HashAlgorithm.values()) {

        int hammingDistance = calculateHammingDistance(
                simhashTable1.get(hashAlgorithm).getSimhashAsBytes(),
                simhashTable2.get(hashAlgorithm).getSimhashAsBytes()
        );

        double similarity = 1
            - (hammingDistance /
                  (double) simhashTable1.get(hashAlgorithm)
                                        .getSimhashLength())
                - hashAlgorithm.getProbNormalization(hammingDistance);

        resultArrayList.add(similarity);

        evaluationResults.get(hashAlgorithm)
                         .setHammingDistance(hammingDistance);
        evaluationResults.get(hashAlgorithm)
                         .setSimilarity(similarity);
    }
}
```

Výpis 6.6. Metoda `evaluateSimilarity` třídy `SimhashSimilarity`.

Kapitola 7
Klasifikace množiny stromů

Vzhledem k tomu, že množinu vybraných kořenových stromů tvoří objekty stejného typu, lze v kontextu zadání práce úlohu *klasifikace* chápat jako *shlukovou analýzu*.

Shlukovou analýzou je myšlen proces, při kterém jsou objekty z konečné množiny řazeny podle jejich společných vlastností do tříd tak, že vzájemně podobné objekty jsou členy stejné třídy. Počet těchto tříd není před začátkem procesu znám a vzniká dynamicky v jeho průběhu.

Metoda Edit Distance (Kapitola 3), se pro vytváření shlukové analýzy nehodí, protože by bylo nutné provádět vzájemné porovnání všech dvojic předloh, což je výkonově velmi náročné. Na rozdíl od ni má pro účely tohoto procesu vhodné vlastnosti metoda SimHash (Kapitola 5), zejména její výsledný otisk, se kterým lze dobře manipulovat.

Níže uvedený návrh používá hierarchické shlukování s metrikou Hammingovi vzdálenosti (Kapitola 5.2.6) a aplikuje myšlenku Dana Lecocqa[21] zveřejněnou v roce 2015.

7.1 Algoritmus shlukové analýzy

Budiž $P = \{G_1, G_2, ..., G_i\}, i \in \mathbb{N}$, množina předloh (kořenových stromů),
$M = \{Q_1, Q_2, ..., Q_i\}$ množina jejich otisků (výstupů funkce Simhash – Kapitola 5.2.5) a $n \in \mathbb{N}$ délka těchto otisků v bitech. Dále zvolme $D_{H_{max}} = 2$ jako nejvyšší vzájemnou Hammingovu vzdálenost všech členů stejné budoucí třídy. A konečně, nechť jsou všechny otisky $Q \in M$ rozděleny na 4 stejně dlouhé segmenty označené A, B, C, D s délkou $l = \frac{n}{4}$ bitů.

Mají-li se dva otisky $Q_i, Qj \in M, i \neq j$, lišit nejvýše ve dvou bitech, projeví se to nejvýše ve dvou jejich segmentech. Zbylé zůstanou stejné (Obrázek 7.1.). V tomto případě může takových kombinací být 6 – AB, AC, AD, BC, BD, CD.

	A	B	C	D
Q_i	0101100101011100	1110111010001011	0010101111001011	1000100011001011
Q_j	0101100101010100	1110111010001011	0010101111001011	1100100011001011

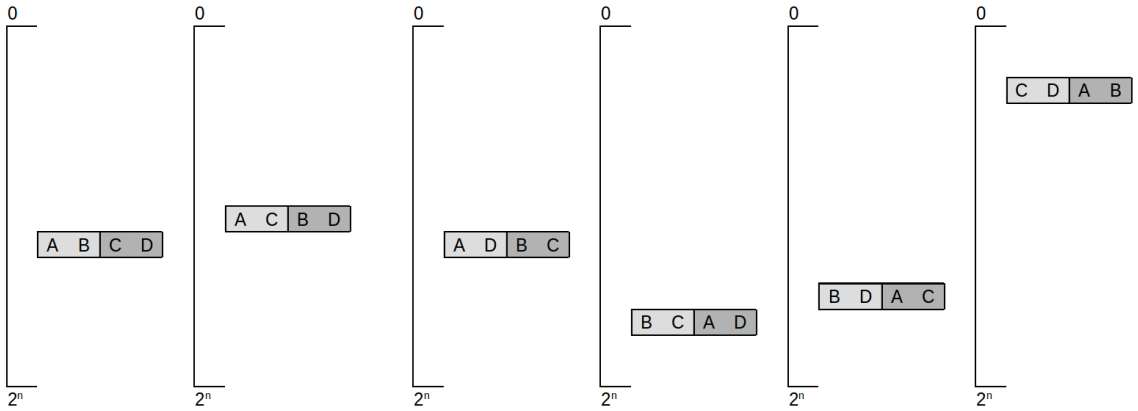
Obrázek 7.1. Segmenty otisků Q_i, Q_j s délkou $n = 64$ bitů [21].

Algoritmus vytvoří 6 řazených seznamů všech otisků, a jejich segmenty v nich přerovná do jedné z permutací $ABCD, ACBD, ADBC, BCAD, BDAC, CDAB$ (Obrázek 7.2.).

Vzájemně podobné otisky budou ležet ve stejných „hladinách“, na kterých lze algoritmicky relativně rychle detekovat jednotlivé třídy.

Protože tento postup je založen na myšlence *Near-Duplicate Detection*, je „šířka“ jednotlivých tříd dosti malá. Závislost maximálního rozdílu míry podobnosti dvou objektů (otisků) v jedné třídě ukazuje Tabulka 7.1.

V některých případech může postačit použití otisků s kratší délkou (32 bitů). Rozšíření tříd je ale možné jen rozdělením otisků do více segmentů, což ovšem prudce zvyšuje počet jejich permutací a tím i složitost celého algoritmu.



Obrázek 7.2. Řazené seznamy všech otisků [21].

n	$D_{H_{max}} = 2$	$D_{H_{max}} = 4$
32	0,0625	0,1250
64	0,0312	0,0625
128	0,0156	0,0312
256	0,0078	0,0156

Tabulka 7.1. Maximální rozdíl míry podobnosti dvou objektů v jedné třídě.

Kapitola 8
Provádění experimentů

Zamyšlení a úvahy v oblasti teorie grafů lze jistě dělat bez použití pomůcek, pouze „v hlavě“. Jednoduché případy je možné modelovat pomocí náčrtků nebo různých diagramů na papíře. Ovšem provádění komplexních experimentů či obecných ověřování dosažených závěrů je tímto způsobem vyloučené.

Uvedená situace vedla k vytvoření *experimentálního srovnávacího nástroje*. Tedy softwarové aplikace, která by usnadnila práci s grafy, obsahovala vlastní implementaci vybraných metod a umožnila jejich testování.

8.1 Experimentální nástroj

Po prvotní analýze a několika ověřovacích pokusech byly identifikovány hlavní problémy, jimiž se bylo nutné vážně zabývat ještě před samotnou implementací navržených metod:

- Popsat grafy pomocí vhodného datového formátu
- Zajistit persistenci takto popsaných grafů
- Najít způsob jejich vizualizace
- Získat nebo napsat knihovnu usnadňující práci s grafy
- Umožnit základní manipulaci s uloženými grafy

Teprve po jejich uspokojivém vyřešení bylo možné pokračovat v praktické realizaci stanovených požadavků:

- Implementovat vybrané metody
- Automatizovat jejich testování
- Prezentovat naměřené hodnoty
- Porovnat a vyhodnotit naměřené hodnoty

8.2 Aplikace SimCom

SimCom je desktopová aplikace napsaná pro snadnou přenesitelnost v jazyku JavaFX 8. Ke správnému fungování potřebuje běhové prostředí JRE 8 a program `dot`, což je nástroj pro vizualizaci grafů (Kapitola 8.2.2).

8.2.1 Popis grafů

Matematickou reprezentaci grafu je možné vytvořit např. výčtem množiny hran $E(G)$ nebo pomocí *matice sousednosti* (angl. adjacency matrix). Tyto způsoby zápisu by jistě bylo možné použít, ovšem snahou nebylo vytvářet proprietární formát, nýbrž najít jiný

způsob, který by se dal s výhodou použít ve vztahu k programu `dot` a zároveň činil aplikaci SimCom otevřenější.

Výsledkem krátké praktické zkoušky několika datových formátů byla volba formátu DOT[1]). Je to plain-textový jazyk pro popis grafů, který má jednoduchou *human-readable* syntaxi a dokáže srozumitelně zachytit i složité grafy (Výpis 8.1.).

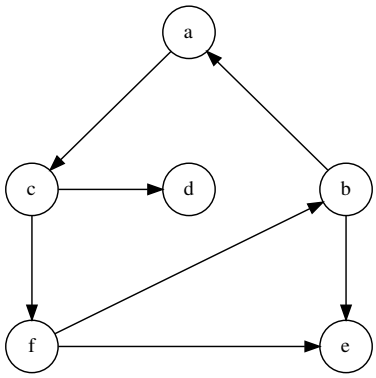
```
digraph G {
    rankdir = TB;
    nodesep = 1.0;
    ranksep = "1.0 equally";

    node [shape = circle];

    a [group = g2]
    {rank = same; b[group = g1]; c[group = g3]; d[group = g4];}
    {rank = same; e[group = g1]; f[group = g3];}

        a -> c;
        c -> d;
        c -> f;
        f -> b;
        f -> e;
        b -> a;
        b -> e;
}
```

Výpis 8.1. Ukázka popisu orientovaného grafu G ve formátu DOT.



Obrázek 8.1. Orientovaný graf G.

8.2.2 Vizualizace grafů

Řešení tohoto problému předcházely jisté obavy, protože nebylo možné věnovat příliš času práci s grafickými objekty, ale přesto bylo zřejmé, že správná vizualizace grafu je důležitá.

Velmi dobře jej ovšem pomohl vyřešit program s názvem `dot`, který je součástí open-source multiplatformního balíku Graphviz[2]) (licence CPL) a slouží k vizualizaci orientovaných i neorientovaných grafů.

[1]) `https://graphviz.gitlab.io/_pages/doc/info/lang.html`
[2]) `https://www.graphviz.org/`

Očekává vstup v jazyce DOT (s příponami `.gv` nebo `.dot`) a dokáže vytvářet hierarchické i vrstvené nákresy grafů. Jeho algoritmus pro plánování směruje hrany jednotně buď vertikálně, nebo horizontálně a snaží se minimalizovat jejich délky i počet křížení. Výstupem mohou být jak rastrové, tak vektorové obrázky v běžných grafických formátech (PNG, GIF, PS, SVG, atd.).

Jak lze vidět na Obrázku 8.1., kvalita výstupu je velmi vysoká a pro účely vizualizace a prezentace grafů v aplikaci SimCom plně dostačuje.

8.2.3 Knihovna JGraphT

Pro správné objektové modelování grafů je nutné mít k dispozici odpovídající třídy všech základních komponent (vrchol, hrana, graf). Ty musí kompletně popisovat jejich vlastnosti a samozřejmě obsahovat metody umožňující zachytit chování a vzájemné vztahy uvedených komponent.

Vytvoření takových tříd bylo časově mimo rozsah této práce a nebylo ani jejím cílem. Proto byla pro softwarové modelování grafů a práci s nimi použita knihovna JGraphT[1]). Je to oblíbená, volně dostupná objektová knihovna (licence LGPL + EPL), která nabízí několik desítek tříd a stovky metod pokrývajících základní pojmy teorie grafů.

8.2.4 Import grafů

Popisy grafů, se kterými má aplikace dále pracovat, jsou uloženy v textových souborech typu DOT (Kapitola 8.2.1). Aplikace umožňuje importovat jeden vybraný soubor, nebo celý obsah zvoleného adresáře. To usnadňuje tvorbu specifických skupin testovacích stromů.

V průběhu importu dochází nejprve ke kontrole, zda je importovaný graf stromem (existence cyklu či více komponent souvislosti), příp. zda již nebyl importován, a poté k vytvoření vnitřní objektové podoby grafu.

8.2.5 Katalog a persistence

Importované grafy jsou ukládány do tzv. *katalogu* (Obrázek 8.2.), což je dynamické datové pole představující množinu grafů, se kterou aplikace pracuje. Persistence katalogu je zajištěna uložením jeho serializovaného obsahu do souboru `catalog.bin` umístěného v domovském adresáři uživatele. Kliknutím na obrázek grafu je možné příslušný graf označit jako vybraný k porovnání.

8.2.6 Práce s aplikací

SimCom se chová jako standardní desktopová aplikace. Má obvyklé ovládací prvky a práce s ním by měla být pro uživatele intuitivní. Jednotlivé funkcionality se spouští výběrem z hlavní nabídky. Největší část pracovní plochy hlavního okna zaujímají tři panely:

- *Graphs* – detailní zobrazení skupiny grafů vybraných k porovnání
- *Console* – textové zobrazení průběhu a výsledků jednotlivých metod
- *Summary* – přehled výsledků skupiny porovnávaných grafů (matice podobnosti)

Práce s aplikací může zahrnovat různé scénáře, ovšem nejčastější pracovní postup uživatele by měl být:

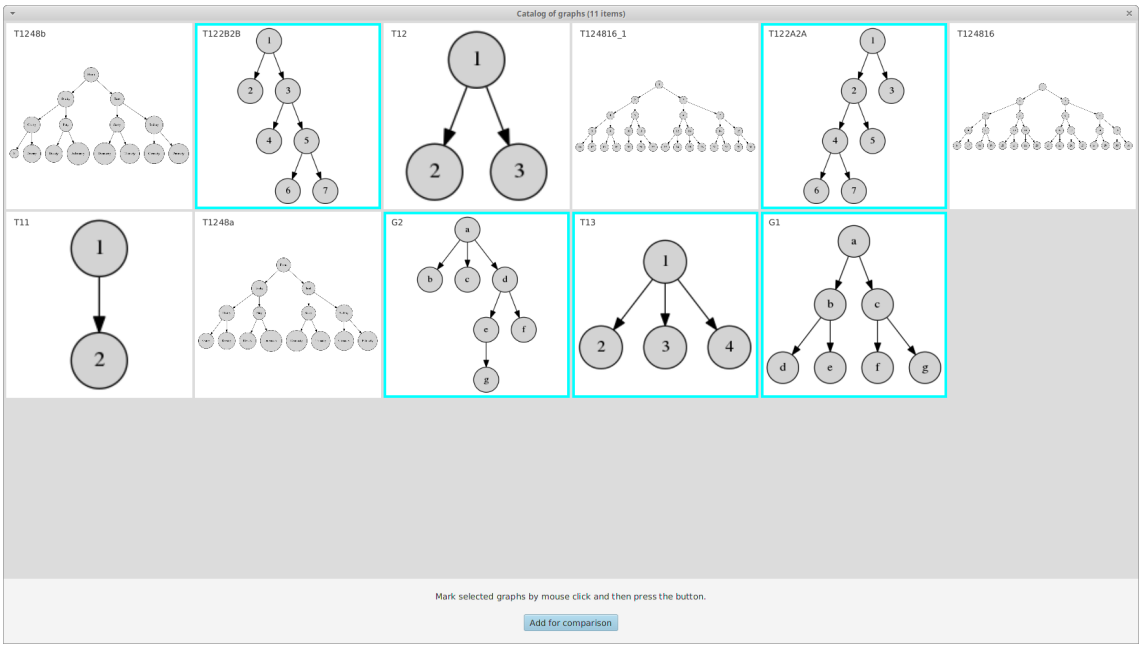
1) Zobrazení katalogu a výběr grafů k porovnání
2) Spuštění porovnání
3) Studium výsledků v panelech *Console* a *Summary* (Obrázky 8.4. a 8.5.)

[1]) `http://jgrapht.org/`

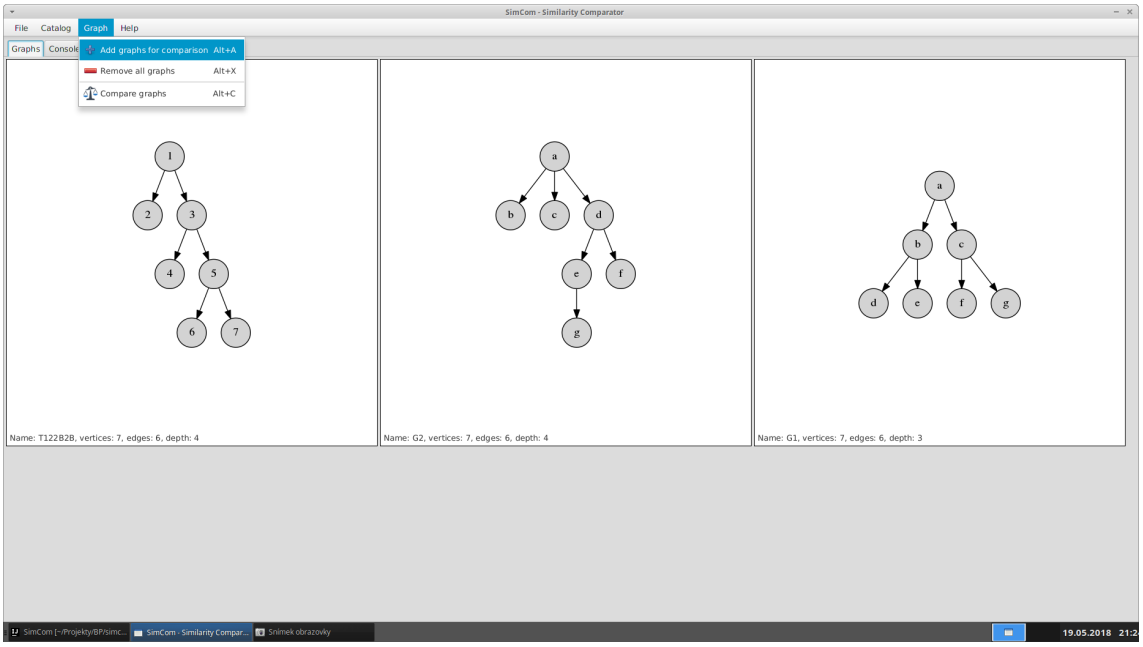
Aplikace rozeznává dva parametry příkazového řádku:

- `resizable` – umožní změnu oken tažením myší
- `debug` – bude vypisovat velmi detailní informace z průběhu metod

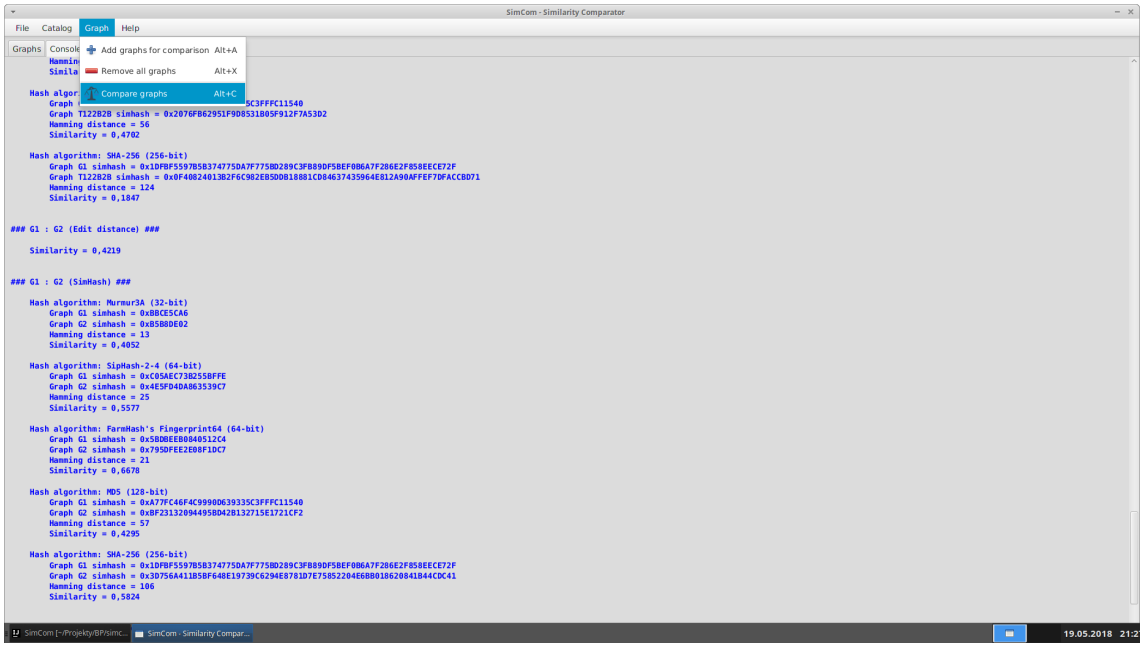
Uživatel tak má vedle obsahu panelů *Console* a *Summary* ještě třetí informační zdroj, který může poskytnout cenné informace při ladění a úvahách nad srovnáváním obou metod.



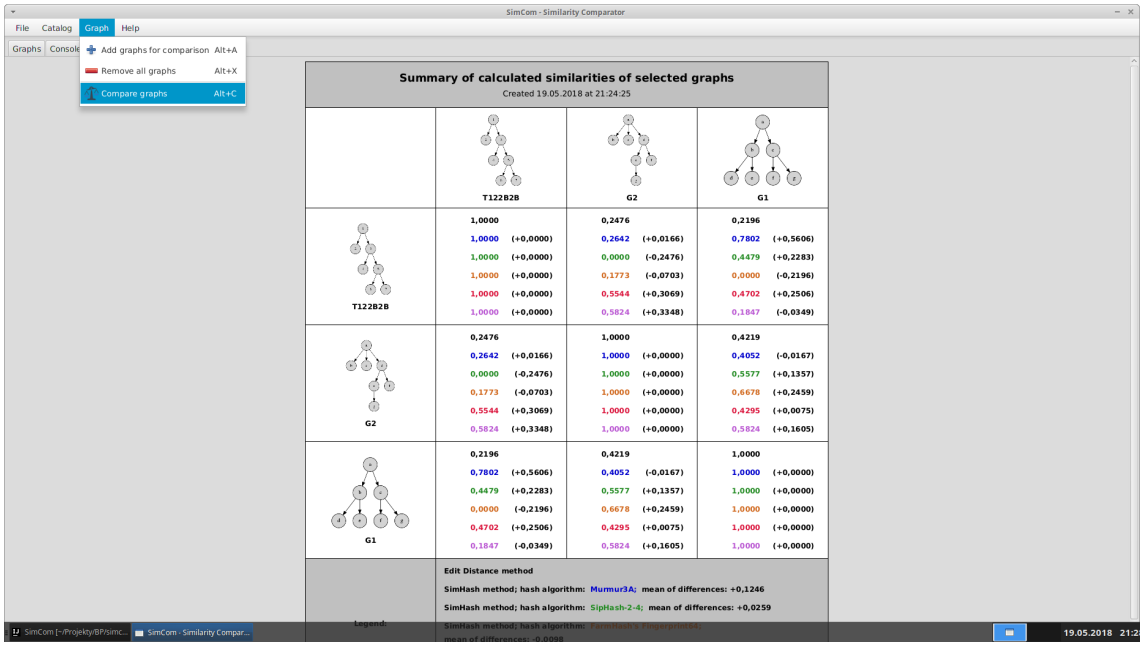
Obrázek 8.2. Okno s obsahem katalogu grafů.



Obrázek 8.3. Hlavní okno – panel *Graphs*.



Obrázek 8.4. Hlavní okno – panel *Console*.



Obrázek 8.5. Hlavní okno – panel *Summary*.

8.2.7 Testovaná prostředí

Korektní fungování aplikace bylo ověřeno na těchto operačních systémech:

- Xubuntu 16.04.4 LTS, Java(TM) SE Runtime Environment (build 1.8.0-171-b11)
- Windows 10, Java(TM) SE Runtime Environment (build 1.8.0-171-b11)
- Windows 7 Enterprise

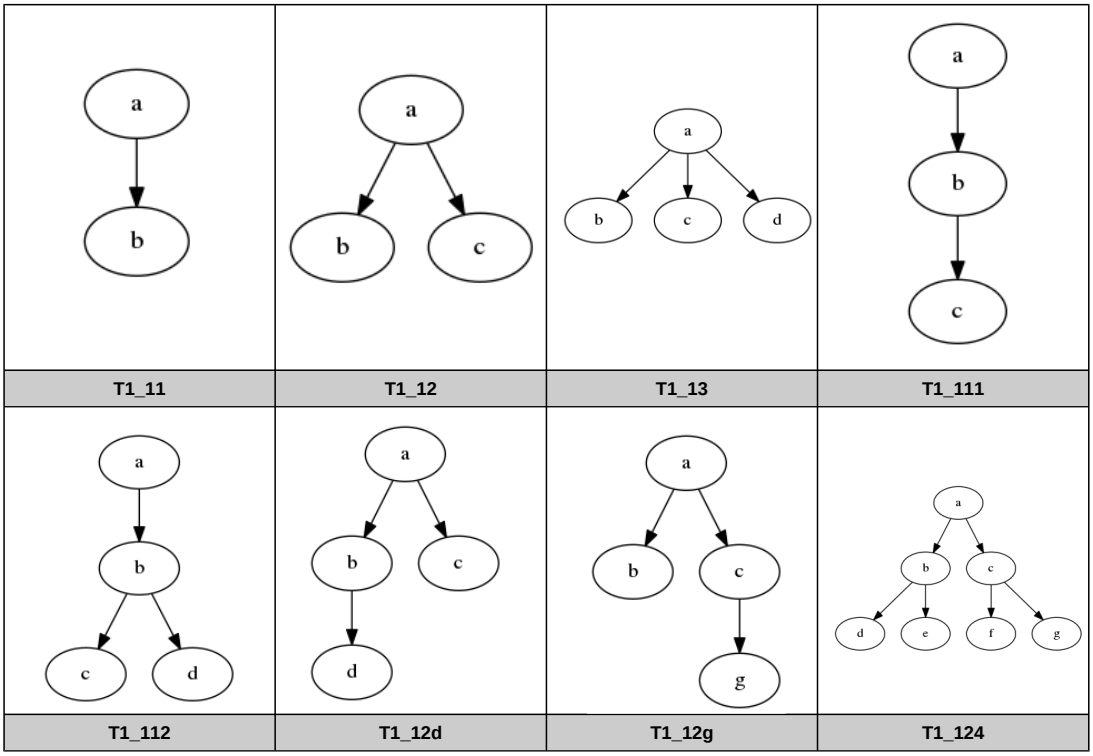
Kapitola 9
Výsledky a hodnocení

9.1 Výsledky

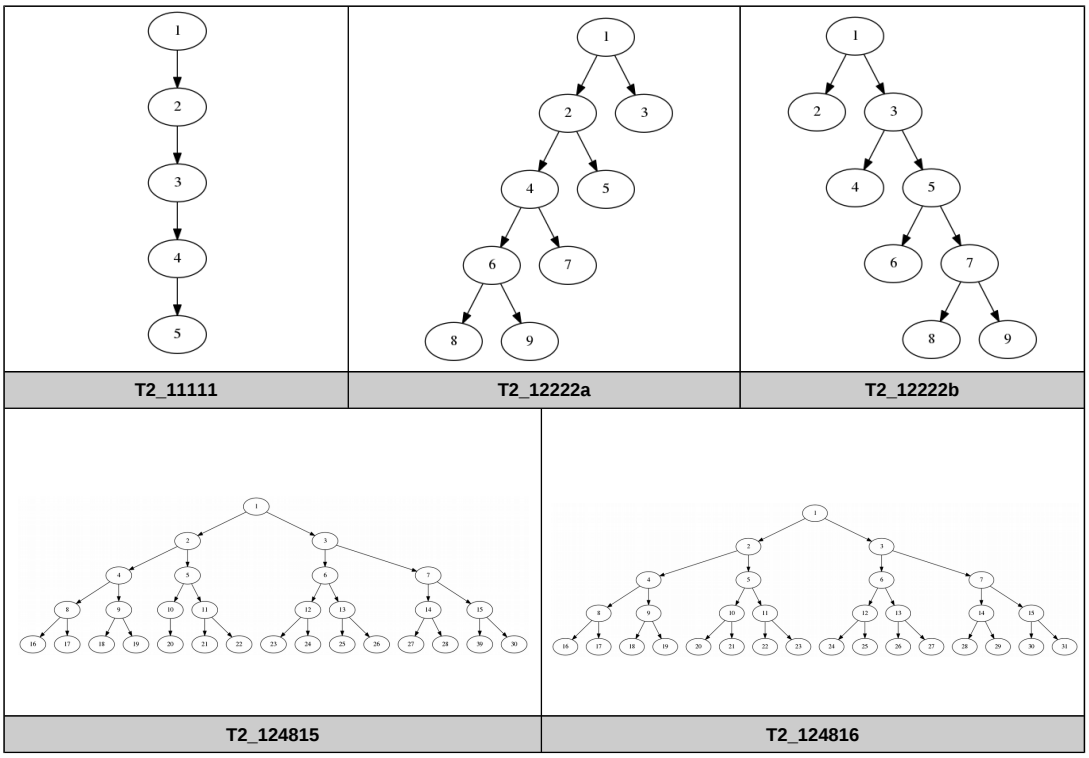
Cílem této části práce byla snaha o komplexní ověření chování obou navržených metod na různých druzích vstupních dat (grafů) a jejich vzájemné srovnání. Pro tento účel byly definovány tři testovací skupiny vybraných grafů s různými vlastnostmi, které měly zajistit dostatečné množství porovnatelných hodnot:

- **I. skupina grafů** – 8 jednoduchých kořenových stromů s hloubkami 2 a 3.
  Obrázky v Tabulce 9.1., vzájemné míry podobností v Příloze A.
- **II. skupina grafů** – 5 větších kořenových stromů s hloubkou 5.
  Obrázky v Tabulce 9.2., vzájemné míry podobností v Příloze B.
- **III. skupina grafů** – kombinace 4 kořenových stromů a 2 orientovaných grafů.
  Obrázky v Tabulce 9.3., vzájemné míry podobností v Příloze C.

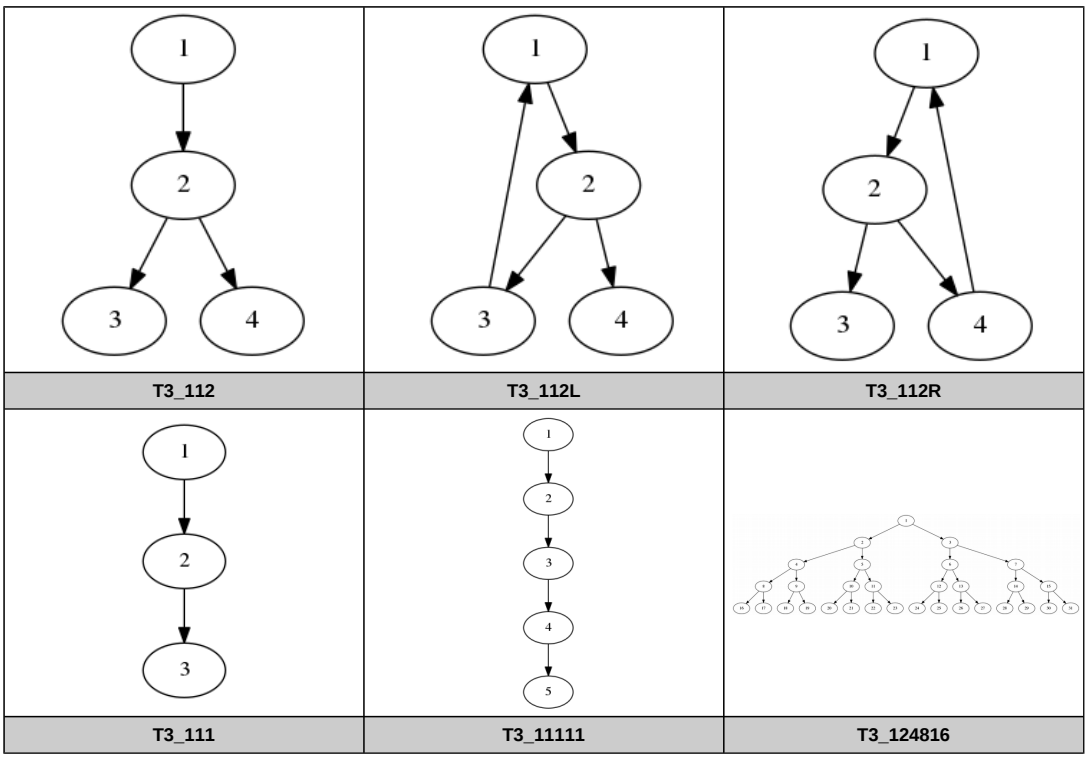
Získané výsledky jsou uvedeny v přílohách této práce a podrobněji jsou interpretovány a zhodnoceny v následující kapitole.



Obrázek 9.1. I. skupina grafů.



Obrázek 9.2. II. skupina grafů.



Obrázek 9.3. III. skupina grafů.

9.2 Hodnocení

Všechny naměřené výsledky obou metod jsou z intervalu $[0 - 1]$. Obě metody dokázaly ve všech příslušných porovnáních detekovat totožné předlohy. Obě byly také schopny zachytit trend podobnosti – s rostoucí objektivní podobností předloh rostla naměřená míra jejich podobnosti. U metody SimHash však toto chování neplatilo pro všechny použité hash funkce.

9.2.1 Metoda Edit Distance

Její výsledky se nacházely v okolí předpokládaných bodů oboru hodnot a nepřibližovaly se bezdůvodně k extrémům. Určila nesprávně jako totožné předlohy T2_12222a a T2_12222b ve II. skupině grafů a T3_112L a T3_112R ve III. skupině grafů, což vyplývá z její aktuální implementace.

9.2.2 Metoda SimHash

Potvrdila očekávání (Kapitola 5.2.1) velké citlivosti na použitou hash funkci. S žádnou z pěti implementovaných hash funkcí nebyla schopna poskytovat vyrovnané hodnoty ve všech testovacích skupinách grafů.

Průměrnou odchylkou se výsledkům metody Edit Distance nejvíce přiblížila s hash funkcí FarmHash's Fingerprint64 v I. a II. skupině grafů a s hash funkcí MD5 ve III. skupině grafů. V obou skupinách však vracela v některých případech bezdůvodně extrémy (minima), např. FarmHash's Fingerprint64 u T1_112 a T1_13 v I. skupině a MD5 u T2_12222b a T2_124815 ve II. skupině.

Velmi dobrých výsledků ovšem dosáhla s hash funkcí SHA-256 v I. a II. skupině grafů.

9.2.3 Další poznatky

III. testovací skupina záměrně obsahuje vedle kořenových stromů také dva orientované grafy s cyklem. Smyslem jejich přítomnosti bylo ověřit chování navržených metod také na nich. Metoda Edit Distance je vyhodnotila jako totožné, ale metoda SimHash dokázala s vysokou přesností rozlišit jejich podobnost, a to všemi použitými hash funkcemi.

9.2.4 Závěr hodnocení

Obě navržené metody prokázaly, že mohou být prakticky použitelné pro algoritmické porovnání dvou kořenových stromů. Metoda Edit Distance je obecně stabilní, zatímco metoda SimHash je úspěšná při účelovém zacílení na určitou skupinu předloh.

Kapitola 10
Náměty pro další rozvoj

Problematika výpočtu míry podobnosti datových objektů je rozsáhlá a možností, jak ladit a optimalizovat postupy je mnoho. Níže jsou uvedeny konkrétní náměty, které by mohly zkvalitnit navržené metody nebo přinést nové informace o jejich chování.

10.1 Změna oceňovací funkce

V metodě Edit Distance je při zarovnávání sekvencí atributů použita exponenciální oceňovací funkce. Tu je možné dále upravit nebo nahradit jiným typem funkce (např. lineární), která by v určitých případech mohla být výhodnější.

10.2 Objem testovacích dat

Při testování a ověření funkčnosti navržených postupů byly použity relativně malé množiny grafů. Ty sice k naplnění zadání této práce postačovaly, ovšem kvalitu testování by zvýšily velké datové korpusy, které by pomohly získat lepší obraz rychlosti a přesnosti implementovaných metod.

10.3 Generátor grafů

K předchozímu námětu se váže možnost integrovat do aplikace SimCom generátor grafů, který by na základě vstupních parametrů (počet vrcholů, jejich atributy, hloubka grafu, atd.) dokázal automaticky vytvořit dostatečně velké množství testovacích objektů.

10.4 Integrace s AASD

Případem vhodného uplatnění výsledků této práce může být integrace navržených metod do konceptu Adaptive Application Structure Design[1], zejména v oblasti generování vysoce personalizovaných uživatelských nabídek.

Kapitola 11
Závěr

Zpočátku jsem měl ze zvoleného tématu trochu strach pro jeho odbornou, zejména matematickou, náročnost. Mé obavy dále sílily v průběhu úvodní orientace v rozsahu celé problematiky. Určité uklidnění mi přinesly až první praktické výsledky a také podpora mého vedoucího. Postupně se mi dařilo pronikat do oblasti tématu a práce mi začala přinášet radost.

Problém, se kterým jsem se však nedokázal zcela vypořádat, byl čas. Velkou část vymezené doby spotřebovala úvodní rešerše odborných textů, jejich pochopení a zasazení do správných souvislostí. Dalším úskalím byla příliš velkorysá specifikace a následná implementace funkcionalit aplikace SimCom.

V této práci jsem se naučil zacházet s vědeckými publikacemi, zopakoval a prohloubil si vědomosti z teorie grafů a znalosti výpočtu složitosti algoritmů. Zároveň jsem využil mnoho jiných informací ze svého dosavadního studia.

Velkou satisfakcí mi byla také skutečnost, že některé myšlenky této práce byly součástí odborného článku *Šebek, J., Vondrus, P. and Černý, T. Degree of similarity of root trees*, (financovaného grantem Studentské grantové soutěže ČVUT SGS15/210/OHK3/3T/13), který byl přijat na konferenci ICISA2018, Hong Kong.

Literatura

[1] Sebek, Jiri, Tomas Cerny a Karel Richta. Adaptive Application Structure Design for Java EE Applications. In: *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. New York, NY, USA: ACM, 2016. s. 159–164. RACS '16. ISBN 978-1-4503-4455-5. Dostupné na DOI 10.1145/2987386.2987417.
http://doi.acm.org/10.1145/2987386.2987417.

[2] SEDLÁČEK, Jiří. *Úvod do teorie grafů*. 3. vyd. Praha: Academia, 1981.

[3] DEMEL, Jiří. *Grafy a jejich aplikace*. 1. vyd. Praha: Academia, 2002. ISBN 80-200-0990-6.

[4] Gary CHARTRAND, Linda LESNIAK, Ping ZHANG. *Graphs & digraphs*. 6th ed. Boca Raton: Chapman & Hall/CRC Press, 2016. ISBN 978-1-4987-3580-3.

[5] Černý, Jakub. *Základní grafové algoritmy*. KAM, MFF UK.
http://kam.mff.cuni.cz/~kuba/ka/ka.pdf.

[6] Dehmer, Matthias a Alexander Mehler. A new method of measuring similarity for a special class of directed graphs. *Tatra Mt. Math. Pub.* Jan, 2007, ročník 36, s. 1–22.

[7] Yang, Rui, Panos Kalnis a Anthony K. H. Tung. Similarity Evaluation on Tree-structured Data. In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2005. s. 754–765. SIGMOD '05. ISBN 1-59593-060-4. Dostupné na DOI 10.1145/1066157.1066243.
http://doi.acm.org/10.1145/1066157.1066243.

[8] Fréchet, Maurice René. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*. Springer, 1906, ročník 22, č. 1, s. 1–72.

[9] Selkow, Stanley M. The tree-to-tree editing problem. *Information Processing Letters*. Dec, 1977, ročník 6, s. 184–186. ISSN 0020-0190.
https://doi.org/10.1016/0020-0190(77)90064-3.

[10] Chawathe, Sudarshan S. Comparing Hierarchical Data in External Memory. In: *Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. s. 90–101. VLDB '99. ISBN 1-55860-615-7.
http://dl.acm.org/citation.cfm?id=645925.671669.

[11] Tekli, Joe, Richard Chbeir a Kokou Yétongnon. Efficient XML Structural Similarity Detection using Sub-tree Commonalities. In: *SBBD*. Brazil, 2007.

[12] Dehmer, Matthias, Frank Emmert-Streib a Jürgen Kilian. A similarity measure for graphs with low computational complexity. *Applied Mathematics and Computation*. 2006, ročník 182, s. 447–459. ISSN 0096-3003.
https://doi.org/10.1016/j.amc.2006.04.006.

[13] Levenshtein, Vladimir Iosifovich. Binary codes capable of correcting deletions, insertions and reversals.. *Soviet Physics - Doklady*. Feb, 1966, ročník 10, č. 8, s. 707–710. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[14] Needleman, Saul B. a Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*. 1970, ročník 48, č. 3, s. 443–453. ISSN 0022-2836. Dostupné na DOI https://doi.org/10.1016/0022-2836(70)90057-4.
http://www.sciencedirect.com/science/article/pii/0022283670900574.

[15] Charikar, Moses S. Similarity Estimation Techniques from Rounding Algorithms. In: *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 2002. s. 380–388. STOC '02. ISBN 1-58113-495-9. Dostupné na DOI 10.1145/509907.509965.
http://doi.acm.org/10.1145/509907.509965.

[16] Zhen-kun, W., Z. Wei-zong, Ouyang-Jie, L. Peng-fei, D. Yi-hua, Z. Meng a G. Jin-hua. A Robust and Discriminative Image Perceptual Hash Algorithm. In: *2010 Fourth International Conference on Genetic and Evolutionary Computing*. 2010. s. 709-712. Dostupné na DOI 10.1109/ICGEC.2010.180.

[17] Jaccard, Paul. Etude de la distribution florale dans une portion des Alpes et du Jura. Lausanne, 1901, ročník 37, s. 547–579. Dostupné na DOI 10.5169/seals-266450.
http://dx.doi.org/10.5169/seals-266450.

[18] Rivest, R. The MD5 Message-Digest Algorithm. United States: RFC Editor, 1992.
https://www.ietf.org/rfc/rfc1321.txt.

[19] Katz, Jonathan a Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd vyd. Chapman & Hall/CRC, 2014. ISBN 1466570261, 9781466570269.

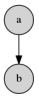
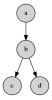
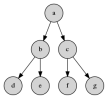
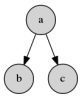
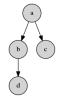
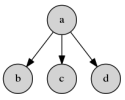
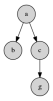
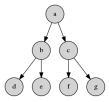
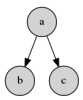
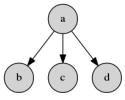
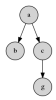
[20] Bookstein, Abraham, Vladimir A. Kulyukin a Timo Raita. Generalized Hamming Distance. *Inf. Retr.* Hingham, MA, USA: Kluwer Academic Publishers, oct, 2002, ročník 5, č. 4, s. 353–375. ISSN 1386-4564. Dostupné na DOI 10.1023/A:1020499411651.
https://doi.org/10.1023/A:1020499411651.

[21] Lecocq, Dan. *Near-duplicate Detection*. Moz Developer Blog.
https://moz.com/devblog/near-duplicate-detection/.

Příloha A
Porovnání I. skupiny grafů

Summary of calculated similarities of selected graphs

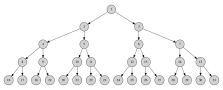
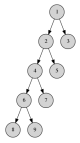
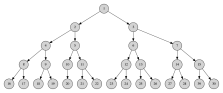
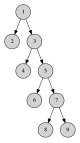
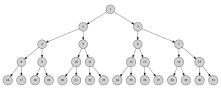
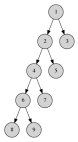
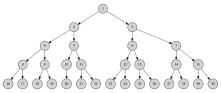
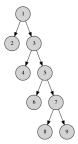
Created 21.05.2018 at 17:45:47

	 T1_11	 T1_112	 T1_24	 T1_12	 T1_111	 T1_12d	 T1_13	 T1_12g
 T1_11	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,6237 0,7087 (+0,0850) 0,6678 (+0,0441) 0,7023 (+0,0787) 0,7188 (+0,0951) 0,7305 (+0,1068)	0,4146 0,6012 (+0,1865) 0,3716 (-0,0430) 0,7023 (+0,2877) 0,4702 (+0,0555) 0,5921 (+0,1775)	0,8520 0,6624 (-0,1896) 0,5577 (-0,2944) 0,7343 (-0,1178) 0,6121 (-0,2399) 0,6719 (-0,1802)	0,6311 0,7087 (+0,0776) 0,7023 (+0,0713) 0,7500 (+0,1189) 0,7891 (+0,1580) 0,7461 (+0,1150)	0,5212 0,1012 (-0,4200) 0,3716 (-0,1496) 0,5577 (+0,0364) 0,3814 (-0,1398) 0,6445 (+0,1233)	0,7774 0,7087 (-0,0686) 0,6678 (-0,1096) 0,7500 (-0,0274) 0,7188 (-0,0586) 0,6484 (-0,1289)	0,5212 0,4052 (-0,1160) 0,3716 (-0,1496) 0,6478 (+0,1266) 0,5318 (+0,0106) 0,6050 (+0,0838)
 T1_112	0,6237 0,7087 (+0,0850) 0,6678 (+0,0441) 0,7023 (+0,0787) 0,7188 (+0,0951) 0,7305 (+0,1068)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,4495 0,5173 (+0,0678) 0,4479 (-0,0016) 0,5950 (+0,1455) 0,6397 (+0,1902) 0,6289 (+0,1793)	0,5152 0,6012 (+0,0860) 0,0000 (-0,5152) 0,5094 (-0,0057) 0,5544 (+0,0393) 0,5874 (+0,0723)	0,8115 0,6624 (-0,1490) 0,6478 (-0,1637) 0,7656 (-0,0459) 0,8203 (+0,0088) 0,7109 (-0,1006)	0,6644 0,5173 (-0,1471) 0,5577 (-0,1068) 0,5950 (-0,0694) 0,6875 (+0,0231) 0,6328 (-0,0316)	0,4628 0,5173 (+0,0545) 0,3716 (-0,0912) 0,0000 (-0,4628) 0,5318 (+0,0690) 0,3912 (-0,0717)	0,5531 0,2642 (-0,2889) 0,0653 (-0,4878) 0,2807 (-0,2724) 0,1201 (-0,4330) 0,5824 (+0,0293)
 T1_24	0,4146 0,6012 (+0,1865) 0,3716 (-0,0430) 0,7023 (+0,2877) 0,4702 (+0,0555) 0,5921 (+0,1775)	0,4495 0,5173 (+0,0678) 0,4479 (-0,0016) 0,5950 (+0,1455) 0,6397 (+0,1902) 0,6289 (+0,1793)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,6237 0,7087 (+0,0850) 0,6678 (+0,0441) 0,6478 (+0,0241) 0,6121 (-0,0115) 0,6641 (+0,0404)	0,4063 0,0000 (-0,4063) 0,5577 (+0,1513) 0,6242 (+0,2179) 0,5040 (+0,0977) 0,5874 (+0,1811)	0,6789 0,5173 (-0,1616) 0,6478 (-0,0311) 0,7184 (+0,0395) 0,6875 (+0,0086) 0,6758 (-0,0031)	0,5553 0,5173 (-0,0380) 0,6242 (+0,0689) 0,5577 (+0,0023) 0,0000 (-0,5553) 0,5921 (+0,0368)	0,6789 0,6624 (-0,0165) 0,5950 (-0,0839) 0,7023 (+0,0234) 0,6560 (-0,0229) 0,6914 (+0,0125)
 T1_12	0,8520 0,6624 (-0,1896) 0,5577 (-0,2944) 0,7343 (-0,1178) 0,6121 (-0,2399) 0,6719 (-0,1802)	0,5152 0,6012 (+0,0860) 0,0000 (-0,5152) 0,5094 (-0,0057) 0,5544 (+0,0393) 0,5874 (+0,0723)	0,6237 0,7087 (+0,0850) 0,6678 (+0,0441) 0,6478 (+0,0241) 0,6121 (-0,0115) 0,6641 (+0,0404)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,5212 0,1012 (-0,4200) 0,3716 (-0,1496) 0,4479 (-0,0733) 0,4702 (-0,0511) 0,4918 (-0,0294)	0,6323 0,1012 (-0,5311) 0,7656 (+0,1333) 0,7184 (+0,0861) 0,7109 (+0,0786) 0,8008 (+0,1684)	0,9082 0,7802 (-0,1280) 0,6478 (-0,2604) 0,6678 (-0,2404) 0,6479 (-0,2602) 0,6953 (-0,2129)	0,6323 0,7087 (+0,0764) 0,7656 (+0,1333) 0,7343 (+0,1019) 0,7578 (+0,1255) 0,7930 (+0,1606)
 T1_111	0,6311 0,7087 (+0,0776) 0,7023 (+0,0713) 0,7500 (+0,1189) 0,7891 (+0,1580) 0,7461 (+0,1150)	0,8115 0,6624 (-0,1490) 0,6478 (-0,1637) 0,7656 (-0,0459) 0,8203 (+0,0088) 0,7109 (-0,1006)	0,4063 0,0000 (-0,4063) 0,5577 (+0,1513) 0,6242 (+0,2179) 0,5040 (+0,0977) 0,5874 (+0,1811)	0,5212 0,1012 (-0,4200) 0,3716 (-0,1496) 0,4479 (-0,0733) 0,4702 (-0,0511) 0,4918 (-0,0294)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,6159 0,0000 (-0,6159) 0,7023 (+0,0864) 0,6678 (+0,0519) 0,6311 (+0,0152) 0,6797 (+0,0637)	0,4677 0,2642 (-0,2035) 0,1773 (-0,2904) 0,6478 (+0,1801) 0,5544 (+0,0867) 0,4483 (-0,0193)	0,6159 0,5173 (-0,0986) 0,5577 (-0,0583) 0,5950 (-0,0209) 0,3258 (-0,2901) 0,5824 (-0,0336)
 T1_12d	0,5212 0,1012 (-0,4200) 0,3716 (-0,1496) 0,5577 (+0,0364) 0,3814 (-0,1398) 0,6445 (+0,1233)	0,6644 0,5173 (-0,1471) 0,5577 (-0,1068) 0,5950 (-0,0694) 0,6875 (+0,0231) 0,6328 (-0,0316)	0,6789 0,5173 (-0,1616) 0,6478 (-0,0311) 0,7184 (+0,0395) 0,6875 (+0,0086) 0,6758 (-0,0031)	0,6323 0,1012 (-0,5311) 0,7656 (+0,1333) 0,7184 (+0,0861) 0,7109 (+0,0786) 0,8008 (+0,1684)	0,6159 0,0000 (-0,6159) 0,7023 (+0,0864) 0,6678 (+0,0519) 0,6311 (+0,0152) 0,6797 (+0,0637)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,5632 0,0000 (-0,5632) 0,4479 (-0,1153) 0,5577 (-0,0055) 0,0000 (-0,5632) 0,6367 (+0,0735)	0,7500 0,2642 (-0,4858) 0,6478 (-0,1022) 0,7343 (-0,0157) 0,6718 (-0,0782) 0,7344 (-0,0156)
 T1_13	0,7774 0,7087 (-0,0686) 0,6678 (-0,1096) 0,7500 (-0,0274) 0,7188 (-0,0586) 0,6484 (-0,1289)	0,4628 0,5173 (+0,0545) 0,3716 (-0,0912) 0,0000 (-0,4628) 0,5318 (+0,0690) 0,3912 (-0,0717)	0,5553 0,5173 (-0,0380) 0,6242 (+0,0689) 0,5577 (+0,0023) 0,0000 (-0,5553) 0,5921 (+0,0368)	0,9082 0,7802 (-0,1280) 0,6478 (-0,2604) 0,6678 (-0,2404) 0,6479 (-0,2602) 0,6953 (-0,2129)	0,4677 0,2642 (-0,2035) 0,1773 (-0,2904) 0,6478 (+0,1801) 0,5544 (+0,0867) 0,4483 (-0,0193)	0,5632 0,0000 (-0,5632) 0,4479 (-0,1153) 0,5577 (-0,0055) 0,0000 (-0,5632) 0,6367 (+0,0735)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,5632 0,2642 (-0,2990) 0,4479 (-0,1153) 0,5950 (+0,0318) 0,0000 (-0,5632) 0,6131 (+0,0499)
 T1_12g	0,5212 0,4052 (-0,1160) 0,3716 (-0,1496) 0,6478 (+0,1266) 0,5318 (+0,0106) 0,6050 (+0,0838)	0,5531 0,2642 (-0,2889) 0,0653 (-0,4878) 0,2807 (-0,2724) 0,1201 (-0,4330) 0,5824 (+0,0293)	0,6789 0,6624 (-0,0165) 0,5950 (-0,0839) 0,7023 (+0,0234) 0,6560 (-0,0229) 0,6914 (+0,0125)	0,6323 0,7087 (+0,0764) 0,7656 (+0,1333) 0,7343 (+0,1019) 0,7578 (+0,1255) 0,7930 (+0,1606)	0,6159 0,5173 (-0,0986) 0,5577 (-0,0583) 0,5950 (-0,0209) 0,3258 (-0,2901) 0,5824 (-0,0336)	0,7500 0,2642 (-0,4858) 0,6478 (-0,1022) 0,7343 (-0,0157) 0,6718 (-0,0782) 0,7344 (-0,0156)	0,5632 0,2642 (-0,2990) 0,4479 (-0,1153) 0,5950 (+0,0318) 0,0000 (-0,5632) 0,6131 (+0,0499)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)
Legend:	**Edit Distance method** **SimHash method; hash algorithm: Murmur3A; mean of differences: -0,1446** **SimHash method; hash algorithm: SipHash-2-4; mean of differences: -0,0808** **SimHash method; hash algorithm: FarmHash's Fingerprint64; mean of differences: +0,0061** **SimHash method; hash algorithm: MD5; mean of differences: -0,0689** **SimHash method; hash algorithm: SHA-256; mean of differences: +0,0265**							

Příloha B
Porovnání II. skupiny grafů

Summary of calculated similarities of selected graphs

Created 21.05.2018 at 22:04:56

	 T2_124816	 T2_12222a	 T2_124815	 T2_11111	 T2_12222b
 T2_124816	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,0649 0,2642 (+0,1993) 0,0000 (-0,0649) 0,0000 (-0,0649) 0,5882 (+0,5233) 0,4483 (+0,3835)	0,9673 0,8750 (-0,0923) 0,7812 (-0,1861) 0,7500 (-0,2174) 0,9063 (-0,0611) 0,8867 (-0,0806)	0,0033 0,7087 (+0,7054) 0,5094 (+0,5062) 0,0000 (-0,0033) 0,2631 (+0,2598) 0,0822 (+0,0789)	0,0649 0,4052 (+0,3403) 0,5577 (+0,4928) 0,5094 (+0,4445) 0,1201 (+0,0552) 0,4918 (+0,4269)
 T2_12222a	0,0649 0,2642 (+0,1993) 0,0000 (-0,0649) 0,0000 (-0,0649) 0,5882 (+0,5233) 0,4483 (+0,3835)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,0776 0,2642 (+0,1865) 0,0000 (-0,0776) 0,1773 (+0,0996) 0,5544 (+0,4768) 0,5090 (+0,4314)	0,2035 0,1012 (-0,1023) 0,2807 (+0,0772) 0,2807 (+0,0772) 0,4295 (+0,2260) 0,5966 (+0,3931)	1,0000 0,7802 (-0,2198) 0,5950 (-0,4050) 0,6478 (-0,3522) 0,7109 (-0,2891) 0,7031 (-0,2969)
 T2_124815	0,9673 0,8750 (-0,0923) 0,7812 (-0,1861) 0,7500 (-0,2174) 0,9063 (-0,0611) 0,8867 (-0,0806)	0,0776 0,2642 (+0,1865) 0,0000 (-0,0776) 0,1773 (+0,0996) 0,5544 (+0,4768) 0,5090 (+0,4314)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,0063 0,6012 (+0,5948) 0,3716 (+0,3653) 0,0653 (+0,0590) 0,0000 (-0,0063) 0,3191 (+0,3127)	0,0776 0,1012 (+0,0236) 0,4479 (+0,3703) 0,5094 (+0,4318) 0,0000 (-0,0776) 0,4717 (+0,3940)
 T2_11111	0,0033 0,7087 (+0,7054) 0,5094 (+0,5062) 0,0000 (-0,0033) 0,2631 (+0,2598) 0,0822 (+0,0789)	0,2035 0,1012 (-0,1023) 0,2807 (+0,0772) 0,2807 (+0,0772) 0,4295 (+0,2260) 0,5966 (+0,3931)	0,0063 0,6012 (+0,5948) 0,3716 (+0,3653) 0,0653 (+0,0590) 0,0000 (-0,0063) 0,3191 (+0,3127)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,2035 0,0000 (-0,2035) 0,2807 (+0,0772) 0,5577 (+0,3542) 0,4702 (+0,2667) 0,6210 (+0,4175)
 T2_12222b	0,0649 0,4052 (+0,3403) 0,5577 (+0,4928) 0,5094 (+0,4445) 0,1201 (+0,0552) 0,4918 (+0,4269)	1,0000 0,7802 (-0,2198) 0,5950 (-0,4050) 0,6478 (-0,3522) 0,7109 (-0,2891) 0,7031 (-0,2969)	0,0776 0,1012 (+0,0236) 0,4479 (+0,3703) 0,5094 (+0,4318) 0,0000 (-0,0776) 0,4717 (+0,3940)	0,2035 0,0000 (-0,2035) 0,2807 (+0,0772) 0,5577 (+0,3542) 0,4702 (+0,2667) 0,6210 (+0,4175)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)

Legend:

Edit Distance method

SimHash method; hash algorithm: Murmur3A; mean of differences: +0,1146

SimHash method; hash algorithm: SipHash-2-4; mean of differences: +0,0924

SimHash method; hash algorithm: FarmHash's Fingerprint64; mean of differences: +0,0663

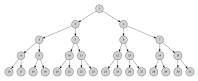
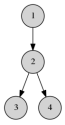
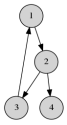
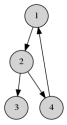
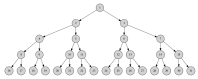
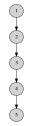
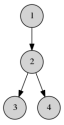
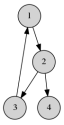
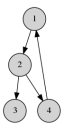
SimHash method; hash algorithm: MD5; mean of differences: +0,1099

SimHash method; hash algorithm: SHA-256; mean of differences: +0,1968

Příloha C
Porovnání III. skupiny grafů

Summary of calculated similarities of selected graphs

Created 21.05.2018 at 22:36:34

	 T3_111	 T3_124816	T3_11111	 T3_112	 T3_112L	 T3_112R
 T3_111	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,0223 0,1012 (+0,0789) 0,6242 (+0,6019) 0,4479 (+0,4256) 0,3814 (+0,3591) 0,3570 (+0,3347)	0,2734 0,7465 (+0,4731) 0,6678 (+0,3944) 0,6857 (+0,4122) 0,7031 (+0,4297) 0,7734 (+0,5000)	0,8115 0,7465 (-0,0650) 0,7184 (-0,0931) 0,7812 (-0,0302) 0,7109 (-0,1006) 0,7578 (-0,0537)	0,8046 0,4052 (-0,3994) 0,2807 (-0,5239) 0,6478 (-0,1568) 0,6397 (-0,1649) 0,6009 (-0,2037)	0,8046 0,4052 (-0,3994) 0,6478 (-0,1568) 0,7812 (-0,0234) 0,6639 (-0,1407) 0,6563 (-0,1484)
 T3_124816	0,0223 0,1012 (+0,0789) 0,6242 (+0,6019) 0,4479 (+0,4256) 0,3814 (+0,3591) 0,3570 (+0,3347)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,0033 0,7087 (+0,7054) 0,5094 (+0,5062) 0,0000 (-0,0033) 0,2631 (+0,2598) 0,0822 (+0,0789)	0,0499 0,4052 (+0,3553) 0,0653 (+0,0154) 0,4479 (+0,3980) 0,0430 (-0,0069) 0,0000 (-0,0499)	0,0504 0,6624 (+0,6120) 0,0000 (-0,0504) 0,2807 (+0,2303) 0,0000 (-0,0504) 0,0000 (-0,0504)	0,0504 0,2642 (+0,2138) 0,0000 (-0,0504) 0,4479 (+0,3975) 0,0000 (-0,0504) 0,0822 (+0,0317)
 T3_11111	0,2734 0,7465 (+0,4731) 0,6678 (+0,3944) 0,6857 (+0,4122) 0,7031 (+0,4297) 0,7734 (+0,5000)	0,0033 0,7087 (+0,7054) 0,5094 (+0,5062) 0,0000 (-0,0033) 0,2631 (+0,2598) 0,0822 (+0,0789)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,2170 0,5173 (+0,3003) 0,5577 (+0,3406) 0,5950 (+0,3780) 0,5882 (+0,3712) 0,6719 (+0,4549)	0,2170 0,7087 (+0,4917) 0,6478 (+0,4308) 0,6857 (+0,4687) 0,6220 (+0,4050) 0,6171 (+0,4001)	0,2170 0,1012 (-0,1158) 0,2807 (+0,0637) 0,3716 (+0,1546) 0,0430 (-0,1740) 0,5921 (+0,3751)
 T3_112	0,8115 0,7465 (-0,0650) 0,7184 (-0,0931) 0,7812 (-0,0302) 0,7109 (-0,1006) 0,7578 (-0,0537)	0,0499 0,4052 (+0,3553) 0,0653 (+0,0154) 0,4479 (+0,3980) 0,0430 (-0,0069) 0,0000 (-0,0499)	0,2170 0,5173 (+0,3003) 0,5577 (+0,3406) 0,5950 (+0,3780) 0,5882 (+0,3712) 0,6719 (+0,4549)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	0,9934 0,8437 (-0,1497) 0,8281 (-0,1653) 0,8125 (-0,1809) 0,7734 (-0,2200) 0,7344 (-0,2590)	0,9934 0,8437 (-0,1497) 0,7184 (-0,2750) 0,8438 (-0,1496) 0,7813 (-0,2121) 0,7266 (-0,2668)
 T3_112L	0,8046 0,4052 (-0,3994) 0,2807 (-0,5239) 0,6478 (-0,1568) 0,6397 (-0,1649) 0,6009 (-0,2037)	0,0504 0,6624 (+0,6120) 0,0000 (-0,0504) 0,2807 (+0,2303) 0,0000 (-0,0504) 0,0000 (-0,0504)	0,2170 0,7087 (+0,4917) 0,6478 (+0,4308) 0,6857 (+0,4687) 0,6220 (+0,4050) 0,6171 (+0,4001)	0,9934 0,8437 (-0,1497) 0,8281 (-0,1653) 0,8125 (-0,1809) 0,7734 (-0,2200) 0,7344 (-0,2590)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)	1,0000 0,8122 (-0,1878) 0,7656 (-0,2344) 0,7500 (-0,2500) 0,8047 (-0,1953) 0,7500 (-0,2500)
 T3_112R	0,8046 0,4052 (-0,3994) 0,6478 (-0,1568) 0,7812 (-0,0234) 0,6639 (-0,1407) 0,6563 (-0,1484)	0,0504 0,2642 (+0,2138) 0,0000 (-0,0504) 0,4479 (+0,3975) 0,0000 (-0,0504) 0,0822 (+0,0317)	0,2170 0,1012 (-0,1158) 0,2807 (+0,0637) 0,3716 (+0,1546) 0,0430 (-0,1740) 0,5921 (+0,3751)	0,9934 0,8437 (-0,1497) 0,7184 (-0,2750) 0,8438 (-0,1496) 0,7813 (-0,2121) 0,7266 (-0,2668)	1,0000 0,8122 (-0,1878) 0,7656 (-0,2344) 0,7500 (-0,2500) 0,8047 (-0,1953) 0,7500 (-0,2500)	1,0000 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000) 1,0000 (+0,0000)

Legend:

Edit Distance method

SimHash method; hash algorithm: Murmur3A; mean of differences: +0,0980

SimHash method; hash algorithm: SipHash-2-4; mean of differences: +0,0447

SimHash method; hash algorithm: FarmHash's Fingerprint64; mean of differences: +0,1150

SimHash method; hash algorithm: MD5; mean of differences: +0,0283

SimHash method; hash algorithm: SHA-256; mean of differences: +0,0496

Příloha D
Zkratky a značení

D.1 Zkratky

BFS	Breadth-first search
CPL	Common Public License
DFS	Depth-first search
EPL	Eclipse Public License
Graphviz	Graph Visualization Software
JRE	Java SE Runtime Environment
LGPL	GNU Lesser General Public License
LSH	Locality-sensitive hashing
MD5	Message-Digest (verze 5)
SimCom	Similarity Comparator
XML	Extensible Markup Language

D.2 Značení

$V(G)$	množina vrcholů grafu G
$E(G)$	množina hran grafu G