Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Podobnost řetězců

Bakalářská práce

Autor: Ondřej Němec
Studijní obor: Aplikovaná informatika – AI-3

Vedoucí práce: Mgr. Jiří Haviger, Ph.D.

Katedra informatiky a kvantitativních metod

Odborný konzultant: Mgr. Jan Vaněk, Ph.D.

Katedra informatiky a kvantitativních metod

Hradec Králové Duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.4.2018 Ondřej Němec

Poděkování:

Děkuji vedoucímu bakalářské práce Mgr. Jiřímu Havigerovi, Ph.D. za metodické vedení práce, trpělivost a čas, také odbornému konzultantovi Mgr. Janu Vaňkovi, Ph.D. za praktické rady k vytváření nástroje pro porovnávání řetězců. Chci také poděkovat Martinu Novému a paní Petře Nové za vysvětlení notového zápisu a pomoc při počítačové reprezentaci not. Také děkuji své sestře Haně Němcové, která mi pomohla správně reprezentovat genetické řetězce, a Sáře Lebedové za kontrolu anglických textů.

Anotace

Tato práce se zabývá porovnávání řetězců pomocí vzdálenosti řetězců. V první části je definována vzdálenost řetězců a jsou popsány některé existující způsoby výpočtu vzdálenosti řetězců spolu s názornými příklady. Druhá část se věnuje současným článkům, které rozebírají podobnost řetězců, její využití v praxi a některým problémům, se kterým se lze při porovnávání řetězců setkat. Třetí část popisuje nástroje, které lze použít pro porovnávání řetězců. Tyto nástroje jsou rozděleny do dvou skupin. V první jsou programy a webové aplikace a druhá zahrnuje knihovny a funkce v různých programovacích jazycích. V poslední části je představen nový nástroj pro porovnávání řetězců a jeho výhody. Nakonec jsou výsledky výpočtů nového nástroje srovnány s existujícím nástrojem.

Klíčová slova: podobnost řetězců, vzdálenost řetězců, metrika, Levenštejnova vzdálenost, Hammingova vzdálenost, Jaroova vzdálenost, Jaro-Winklerova vzdálenost, Kalkulátor vzdáleností sekvencí objektů

Annotation

Title: Similarity of Sequences

The subject of this Bachelor thesis is a similarity of sequences using edit distance. The first part defines edit distance. Using illustrative examples, it also describes some of many existing methods of calculation of the edit distance. The second part explores contemporary articles dealing with the edit distance, its practical application and some problems that could arise. In the third part tools for calculating of the edit distance are introduced and divided into two groups. The first group consists of programs and web applications. In the second group is possible to find libraries and functions in different programming language. In the last part of the thesis a new tool for calculation of the edit distance and its benefits are introduced. Lastly, the results of the new tool are compared with the results of already existing tool.

Keywords: similarity of sequences, edit distace, metric, Levenshtein distance, Hamming distance, Jaro distance, Jaro-Winkler distance, Sequence of object distance calculator

Obsah

1 Úvod

S nástupem internetu se rapidně zvýšilo množství a dostupnost informací. Je velmi jednoduché nějakou myšlenku najít, ale také použít ve vlastní práci a vydávat za svou. Moderní přístroje jsou schopny přečíst genetický kód buněk. Díky tomu lze zjišťovat například příbuznost nebo odhalovat pachatele trestných činů. Mnohé textové editory, vývojová prostředí a vyhledávače kontrolují napsaný text, nebo k rozepsanému navrhují možné varianty slov, která měl pisatel nejspíš na mysli. Vznikají nové a nové firmy. Některé se snaží získat publicitu nebo popularitu tím, že nesou název podobný jiné, známější a důvěryhodnější firmy. Nebo pojmenují nějaký produkt obdobně jako konkurenční firma.

Všechny popsané příklady spojuje nutnost porovnávat řetězce. Je potřeba počítačově zpracovávat data z řetězců (DNA), objektivně porovnávat, jak jsou dvě slova nebo texty podobné (plagiáty, jména firem a produktů). Aby mohl textový editor zkontrolovat, zda je slovo správně napsáno, musí nejprve určit, kterému slovu z jeho databáze je daný výraz nejvíce podobný.

Stejně, jako je využití porovnávání velmi různorodé, jsou různé i požadavky. Například, pokud textový editor chybně označí slovo za špatně napsané, nebude to mít takové následky jako chyba při testu DNA. Na druhou stranu, je kontrola textu v editoru vyžadována pokud možno ihned po dopsání slova, kdežto testy DNA mohou trvat (a trvají) déle.

Z příkladů vyplývá, nestačí pouze říci, zda jsou informace stejné nebo ne, ale je potřeba stanovit, jak hodně jsou stejné nebo rozdílné. Navíc toto porovnání musí být objektivní (tedy ne: „zdá se mi to celkem stejné“, nebo „vypadají víceméně odlišně“) a počítačově zpracovatelné.

Aby mohl počítač informace zpracovat, musejí mít nějakou strukturu. Proto jsou informace předávány počítači v řetězcích. Obvykle se jedná o řetězce znaků abecedy, ale mohou to být i čísla, fonetické znaky, případně báze DNA nebo noty. Dále potřebuje způsob, jakým určit míru podobnosti. Ta se počítá pomocí vzdálenosti řetězců. Definice vzdálenosti (metriky) řetězců a její druhy jsou první částí této práce. Jednotlivé druhy vzdálenosti řetězců jsou popsány a ukázány na příkladech.

Další část se věnuje praktickému využití podobnosti řetězců a dalším otázkám s tím spojených. Co si o porovnávání řetězců myslí současná vědecká veřejnost? V jakých odvětvích lidského zkoumání se využívá?

Jsou nástroje pro porovnávání řetězců snadno dostupné? A co podpora porovnávání řetězců v programovacích jazycích? Těmto otázkám je věnována třetí část, kde jsou ukázky programů pro porovnávání řetězců a funkcí ve vybraných programátorských

jazycích. Nakonec je představen nový nástroj pro porovnávání řetězců. Tento nástroj je součástí práce. Nástroj je zde otestován a jeho výsledky jsou porovnány s výsledky existujícího nástroje.

2 Cíl práce

Cílem této práce je zaprvé provést rešerši aktuálních článků na téma podobnost řetězců. Druhým cílem je rešerše aktuálních dostupných nástrojů pro porovnávání řetězců. Za třetí je cílem této práce příprava vlastního nástroje.

3 Metodika zpracování

Jak lze porovnávat řetězce pomocí vzdálenosti řetězců? To je hlavní otázka této práce. Aby bylo možné na tuto otázku najít odpověď, co to je vzdálenost řetězců, jaké jsou její druhy, čím se vyznačují a jak se počítají. Pomocí literární rešerše současných vědeckých článků bude hledáno využití v praxi.

Vyhledáváním v dokumentacích programovacích jazyků budou hledány implementace vzdáleností řetězců a na internetu budou hledány webové aplikace a programy, které vzdálenost řetězců počítají. Nový nástroj pro porovnávání řetězců, který bude součástí této práce, bude vycházet z provedeného zkoumání. Jeho cílem je doplnit stávající nástroje v oblastech, ve kterých budou nalezeny mezery.

4 Vzdálenosti

4.1 Vzdálenost

Vzdálenost je matematická funkce, která říká, jak hodně jsou si nějaké objekty vzdáleny. Je nejčastěji spojována s geometrií, nicméně se nepoužívá jen pro objekty fyzického světa. *Vzdálenost* a *metrika* jsou sice synonyma, ale *vzdálenost* se například v geometrii používá v konkrétním významu tohoto slova, proto se pro zdůraznění obecnosti používá *metrika* nebo *distance*. [1]

4.1.1 Metrika

Funkce $d: X \times X \rightarrow \boldsymbol{R}$ a nazývá se metrika právě tehdy, když jsou splněny následující 3 axiomy: 1) Metrika je pozitivní mezi různými vstupy, nulová pokud jsou stejné: $x \neq y: d(x, y)>0$, $x=y: d(x, y)=0$. 2) Metrika je symetrická: $d(x, y)=d(y, x)$. 3) Metrika splňuje trojúhelníkovou nerovnost: $x, y, z: d(x, z) \leq d(x, y) + d(y, z)$

4.1.2 Metrický prostor

Metrický prostor je dvojice $(\boldsymbol{M}, d)$, přičemž $\boldsymbol{M}$ je libovolná neprázdná množina a d je metrika.

Příklady

Je definovaný euklidovský prostor $\boldsymbol{R}^n$ (přímka, rovina, prostor, vícerozměrný prostor). Právě v tomto prostoru se používají geometrické operace.

Euklidovská metrika: jsou dány body p a q z euklidovského n-rozměrného prostoru, vzdálenost těchto bodů je velikost úsečky, která je spojuje. V kartézské soustavě souřadnic platí: $\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$.

Manhattanská metrika: jsou dány body p a q z euklidovského prostoru s pevně daným kartézským systémem souřadnic. Jejich vzdálenost je součtem délek projekcí úsečky mezi body na osách. Lze vyjádřit vztahem $d(p, q) = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n|$. Též lze říci, že vyjadřuje délku mezi dvěma body při pohybu vodorovně a svisle.

Je definovaný diskrétní prostor M, ve kterém pro metriky platí: $p(x, x) = 0$ a $p(x, y) = 1$ pro $x \neq y$. Takové metriky se nazývají diskrétní.

Levenštejnova vzdálenost: viz kapitola 4.4.

Délka nejkratší cesty v grafu: na vrcholech souvislého neorientovaného grafu.

[1] V této práci se budou pojmy vzdálenost a metrika používat souběžně jako synonyma v závislosti na čerpaných zdrojích. Další synonymum, distance, se zde používat nebude.

4.2 Vzdálenost řetězců

Vzdálenost řetězců je definována na diskrétním prostoru, platí: vzdálenost *d(x, y)* mezi dvěma řetězci *x* a *y* je minimální cena řady operací, které transformují *x* do *y* (pokud to nelze provést, je cena nekonečno). Cena řady operací je pak součet cen jednotlivých operací. Tyto operace jsou konečná množina pravidel ve tvaru $\delta(z, w) = t$, kde *z* a *w* jsou rozdílné řetězce a *t* je nezáporné reálné číslo.

Jinými slovy: vzdálenost dvou řetězců je minimální počet kroků, které je potřeba udělat, aby se jeden řetězec změnil na druhý, přičemž jeden krok (jedna operace) má obvykle hodnotu (cenu) jedna.

4.2.1 Operace:

Vložení (ang. Insertion) – vložení znaku, např.: *lesk* → ***b****lesk* ± *(„ “, b)*

Smazání (ang. Deletion) – smazání znaku, opak vložení, např.: ***b****lesk* → *lesk* ± *(b, „ “)*

Výměna nebo Náhrada (ang. Substitution) – vyměnění jednoho znaku za jiný, např.: ***č****očka* → ***k****očka* ± *(č, k)*

Prohození (ang. Transposition) – výměna dvou sousedních znaků uvnitř jednoho řetězce, např.: ***lž****íce* → ***žl****íce* ± *(lž, žl)*

4.2.2 Druhy vzdálenosti řetězců

Levenštejnova[2] (někdy též Editační) vzdálenost

Hammingova vzdálenost

Částečná vzdálenost

Vzdálenost nejdelší společné posloupnosti

Založeno na [1].

[2] Ruské jméno, v českých zdrojích se lze setkat jak s českým přepisem „Levenštejnova“, tak počeštělým anglickým přepisem „Levenshteinova“. Zde se bude – s ohledem na slovanský původ slova – používat první verze

4.3 Hammingova vzdálenost

Autorem je Richard W. Hamming. Tato metrika povoluje pouze náhradu. Je definována jen pro stejně dlouhé řetězce.

Založeno na [2].

4.3.1 Postup výpočtu

Jsou dány dva řetězce znaků – řetězec *p* (**p**rvní) a řetězec *d* (**d**ruhý).

1. Zkontrolovat, zda jsou oba řetězce stejně dlouhé.
   1.1. Nejsou-li, nelze spočítat Hammingovu vzdálenost.
2. Pokud jsou stejně dlouhé:
   2.1. Porovnat první znak z *p* s první znakem z *d*. Pokud jsou stejné, zapsat do pomocného řetězce na první pozici 0, jinak 1.[3]
   2.2. Totéž provést pro každý znak *p*, tj. porovnat *n*-tý znak *p* s *n*-tým znakem *d* a výsledek zapsat *n*-tou pozici pomocného řetězce.
3. Projít pomocný řetězec a sečíst všechna čísla (zjistit počet jedniček). Výsledné číslo je Hammingova vzdálenost.

Založeno na [3].

4.3.2 Příklad

Řetězec *p*: HOUSKA, m = 6, řetězec *d*: HOUSLE, n = 6; *n* = *m* – Lze pokračovat

H	O	U	S	K	A
H	O	U	S	L	E
0	0	0	0	1	1

Obrázek 1 - Hammingova vzdálenost

Zdroj: vlastní zpracování

Jak je vidět na obrázku 1, součet pomocného řetězce je 2. To je tedy Hammingova vzdálenost pro slova „houska“ a „housle“ - „k“ se mění na „l“ a „a“ na „e“.

[3] Vhodnou strukturou je zde tabulka o třech řádcích. Počet sloupců je dán počtem znaků v *p*. V každé buňce tabulky je jeden znak: první řádek - první řetězec, druhý řádek – druhý řetězec, třetí řádek – pomocný řetězec.

4.4 Levenštejnova vzdálenost

Tento typ editační vzdálenosti je pojmenován po ruském matematikovi Vladimirovi Iosifoviči Levenštejnovi. Tato metrika používá tři operace: smazání, vložení a výměnu (dosl. obrácení, angl. „reversals“).

Založeno na [4].

4.4.1 Postup výpočtu

Jsou dány dva řetězce znaků – řetězec *p* (**p**rvní) a řetězec *d* (**d**ruhý).

1. Počet znaků v *p* je *n*, počet znaku v *d* je *m*, tj. řetězec *p* délky *n*, řetězen *d* délky *m*.
   1.1. Pokud je *n=0*, výsledek je *m*
   1.2. Pokud je *m=0*, výsledek je *n*
2. Pokud se ani *n* ani *m* nerovnají nule, sestrojí se matice o *n+1* sloupcích a *m+1* řádcích, přičemž první řádek i sloupec je neoznačen a ostatní řádky i sloupce jsou označeny odpovídajícím znakem z příslušného řetězce (označení sloupců: nic, první znak *p*, druhý znak *p*,…; označení řádků: nic, první znak *d*, druhý znak *d*,….).
   2.1. První řádek se naplní čísly od 0 do *n*
   2.2. První sloupec se naplní čísly od 0 do *m*
3. Porovnání každého znaku z *p* s každým znakem z *d* a určení *cost* (ceny). Pokud jsou znaky shodné, je *cost* = 0, jinak se rovná 1. Je vhodné si toto číslo poznamenat, například jako horní nebo dolní index do příslušné buňky.
4. Doplnění zbylých buněk pomocí vzorce buňka b[i, j] = min (A, B, C)
   4.1. *A* označuje hodnotu buňky těsně nad počítanou buňkou zvětšenou o 1, tj. b[i-1, j] + 1
   4.2. *B* je hodnota buňky vlevo od hledané opět zvětšena o jednu, tj. b[i, j-1]+1
   4.3. *C* značí hodnotu v uhlopříčce vlevo nad hledanou, ke které se přičte *cost* hledané, tj. b[i-1,j-1] + *cost*
5. Po provedení všech těchto kroků je výsledek Levenštejnovi metriky v pravém dolním rohu, tj. v buňce b[i, j]

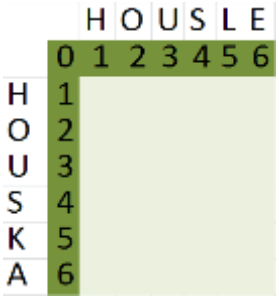
Založeno na [5].

4.4.2 Příklad

Řetězec *p*: HOUSKA, m = 6, řetězec *d*: HOUSLE, n = 6

Obrázek 2 – bod 2.0 postupu - matice n x m, označení sloupců

Zdroj: vlastní zpracování



Obrázek 3 – bod 2.1 a 2.2 postupu - naplnění prvního řádku (sloupce) čísly od 0 do *n* (*m*)

Zdroj: vlastní zpracování

Obrázek 4 - bod 3 postupu - určení *cost*, zde zapsána vpravo od své buňky ve zvýrazněném sloupci

Zdroj: vlastní zpracování

Obrázek 5 - bod 4 postupu - dopočítání zbylých hodnot, konečný výsledek v červeném poli

Zdroj: vlastní zpracování

		H				A	1+1
	0	1				B	1+1
H	1	?	0			C	0+0

Obrázek 6 - detail vzorce začátek

Zdroj: vlastní zpracování

		H	O		A	1+1
	0	1	3		B	3+1
H	1	0 0	1 1		C	2+1
O	2	1 1	0 0			
U	3	2 1	1 1			
S	4	3 1	? 1			

Obrázek 7 - detail vzorce

Zdroj: vlastní zpracování

Jak je vidět z obrázku 5, Levenštejnova vzdálenost pro slova HOUSKA a HOUSLE je 2. První čtyři písmena (HOUS) jsou shodná, vzdálenost je tedy 0, u posledních dvou dojde k nahrazení (k, l) a (a, e).

		O	K	O
	0	1	3	4
K	1	1 1	1 0	2 1
O	2	1 0	2 1	1 0
L	3	2 1	2 1	2 1
O	4	3 0	3 1	2 0

Obrázek 8 - Nestejně dlouhé řetězce

Zdroj: vlastní zpracování

Na obrázku 8 je zachycen případ, kdy řetězce nejsou stejně dlouhé. V tomto případě dojde ve slově KOLO ke smazání „K“ a změně „L“ na „K“.

4.5 Jaroova vzdálenost

Tuto metriku popsal Matthew A. Jaro. Povoluje vložení, smazání a prohození.

Založeno na [6].

4.5.1 Postup výpočtu

Jsou dány dva řetězce znaků – řetězec *p* (**p**rvní) a řetězec *d* (**d**ruhý). Pak Jaroova vzdálenost se spočítá:

$$d_J(p, d)=W_1\times c/|p| + W_2\times c/|d| + W_t\times (c-t)/c$$

$|p|$ - velikost *p* (počet znaků), pokud $|p|=0 \Rightarrow d(a, b)=0$.

$|d|$ - velikost *d* (počet znaků), pokud $|d|=0 \Rightarrow d(a, b)=0$.

c – počet znaků společných pro oba řetězce, pokud $c=0 \Rightarrow d(a, b)=0$.

t – počet prohození

W_1, W_2, W_t – váhy jednotlivých složek, obvykle ⅓

Výsledem je číslo od 0 (rozdílné) do 1 (stejné).

Založeno na [7].

Spočítání *c* a *t*

Krok 1 – spočítání matice:

POZNÁMKA: V tomto kroku se pracuje s rozsahem porovnání. Během výzkumu byly nalezeny dva vzorce. [7] uvádí $r = max(|p|, |d|)/2–1$, ale v [8] se píše $r = min(|p|, |d|)/2$. Během testování se ukázalo, že první vzorec poskytuje přesnější výsledky, proto bude v této práci používán.

1. Spočítání rozsahu porovnávání $r = max(|p|, |d|)/2–1$. Pro znak z *p* na pozici *i* a znak z *d* na pozici *j* platí: $|i-j| < r$.
2. Počet znaků *p* je *n* a počet znaků *d* je *m*. Vytvořit matici $n \times m$.
3. Počítání jednotlivých hodnot - prvek $a_{i,j}$ se vypočítá:
   3.1. Pokud $|i-j| < r$, pak se porovná znak z *p* na pozici *i* a znak z *d* na pozici *j*. Pokud jsou shodné, $a = 1$, jinak $a = 0$.
   3.2. Jinak není hodnota zahrnuta do výpočtu. Pro ruční počítání může zůstat prázdná, pro počítání počítačem je lepší vyplnit ji nulou.

Krok 2 – získání *c* a *t* z matice:

Prochází se vytvořená matice ($n \times m$), výchozím prvkem je $a_{-1,-1}$. Hodnoty c a t jsou nula;

Pokud $a_{x+1,y+1}$ má hodnotu 1, pak je následujícím prvkem $a_{x+1,y+1}$, přičemž c se zvyšuje o jedna.

Pokud není splněna předchozí podmínka a $x+2 < n$, $y+2 < m$, $a_{x+2,y+1}$ má hodnotu 1 a zároveň $a_{x+1,y+2}$ taktéž, je dalším prvkem $a_{x+2,y+2}$. Hodnota t se zvyšuje o jedna a hodnota c se zvyšuje o dvě.

Pokud není splněna předchozí podmínka, jsou definovány tři pomocné proměnné: A, B, C. Pokud existuje prvek $a_{x+k,y+k}$, $k>1$, který má hodnotu 1, pak $A = k$, jinak není hodnota stanovena. Pokud existuje prvek $a_{x+k,y+1}$, $k>1$, který má hodnotu 1, pak $B = k$, jinak není hodnota stanovena. Pokud existuje prvek $a_{x+1,y+k}$, $k>1$, který má hodnotu 1, pak $C = k$, jinak není hodnota stanovena.

Nyní se vybere nejnižší z *A*, *B* a *C*. Nestanovené hodnoty se do výběru nezařazují, v případě rovnosti hodnot se vybírá v abecedním pořadí.

Pokud je nejnižší *A*, pak následující prvek je $a_{x+1,y+1}$.

Pokud je nejnižší *B*, pak následující prvek je $a_{x+1,y}$. V případě, že tento prvek má hodnotu 1 a zároveň předchozí prvek má hodnotu 0, pak se c zvyšuje o jedna.

Pokud je nejnižší *C*, pak následující prvek je $a_{x,y+1}$. V případě, že tento prvek má hodnotu 1 a zároveň předchozí prvek má hodnotu 0, pak se c zvyšuje o jedna.

Pokud není splněna žádná z předchozích podmínek, je následující prvek $a_{x+1,y+1}$.

Založeno na [8].

4.5.2 Příklad

Řetězec *p*: HOUSKA, m = 6, řetězec *d*: HOUSLE, n = 6

Tyto hodnoty jsou dosazeny do vzorce pro výpočet rozsahu:

$r = max(6, 6)/2 - 1 = 2$

	H	O	U	S	L	E
H	1	0	X	X	X	X
O	0	1	0	X	X	X
U	X	0	1	0	X	X
S	X	X	0	1	0	X
K	X	X	X	0	0	0
A	X	X	X	X	0	0

Obrázek 9 - Matice Jaroovi vzdálenosti

Zdroj: vlastní zpracování

Na obrázku 9 je matice Jaroovi vzdálenosti. Růžová pole (vyplněná „X“) označují místa mimo rozsah porovnání. Světle zelená pole jsou zahrnuta do výpočtů. $C = 4$ (tmavě zelená pole), $t = 0$. Pro standartní hodnoty $W_1 = W_2 = W_t = ⅓$: $d_J(housle, houska) = ⅓ \times 4/6 + ⅓ \times 4/6 + ⅓ \times (4-0)/4 = 0{,}777$.

4.6 Jaro-Winklerova vzdálenost

William E. Winkler dosadil Jaroovu vzdálenost do vlastního vzorce. Stejně jako Jaroova vzdálenost i Jaro-Winklerova dovoluje smazání, vložení a prohození.

Založeno na [7].

4.6.1 Postup výpočtu

Jsou dány dva řetězce znaků – řetězec *p* (**p**rvní) a řetězec *d* (**d**ruhý). Pak Jaro-Winklerova vzdálenost se spočítá:

$$d(p, d) = d_j + i \times 0{,}1 \times (1 - d_j)$$

d_j – Jaroova vzdálenost

i – počet počátečních shodných znaků, maximálně však 4

Založeno na [7].

4.6.2 Příklad

Řetězec *p*: HOUSKA, m = 6, řetězec *d*: HOUSLE, n = 6

V předchozí podkapitole vyšla Jaroova vzdálenost 0,777. Počet počátečních shodných znaků je 4. Po dosazení: $d_{JW}(housle, houska) = 0{,}777 + 4 \times 0{,}1 \times (1 - 0{,}777) = 0{,}8662$.

5 Rešerše článků

Během rešerše byly články vyhledávány v elektronické databázi článků ScienceDirect [9]. Jako klíčová slova byly zvoleny „similarity of sequences“, „edit distance“ a „string similarity“. Při zadání první fráze bylo k 15. 12. 2017 nalezeno 511 005 výsledků. V případě druhé fráze bylo nalezeno k témuž dni 123 736 výsledků a v případě třetí 37 421 (k 16. 12. 2017). Díky této rešerši se podařilo nalézt různá uplatnění porovnávání řetězců v praxi. Z nich byly vybrány následující.

5.1 Porovnávání kódu

V [10] se hovoří o porovnávání kódu. „Code similarity studies if two programs are similar or if one program is similar to a portion of another program (code containment). Code similarity is an important component of program analysis that finds application in many fields of computer science, such as reverse engineering of big collections of code fragments [13,16], clone detection [2,8], identification of violations of the intellectual property of programs [1,17,12], malware detection [9,10,15], software maintenance [11,19], software forensics [2,14]. In these applications, when comparing two fragments of code it is important to take into account changes due to code evolution, compiler optimization and post-compile obfuscation. These code changes give rise to fragments of code that are syntactically different while having the same intended behavior. This means that it is important to recognize modifications of the same program that are obtained though compiler optimization or code obfuscation as similar. To this end we need to abstract from syntactic changes and implementation details that do not alter the intended behavior of programs, namely that preserve to some extent the semantics of programs.“ [10]

Podobnost kódů zjišťuje, zda jsou dva programy podobné, nebo zda je nějaký program podobný části jiného programu. Podobnost kódů je důležitou částí programové analýzy, která nachází uplatnění v mnoha směrech počítačových věd, jako například detekce malwaru. V takové aplikaci je důležité vzít v úvahu změny způsobené vývojem kódu, optimalizací kompilátoru a další. Takovéto změny kódu vytvoří části kódy syntakticky odlišné, ale se stejným chováním. Je tedy důležité rozpoznat změny stejného programu získané např. kompilací a označit jako stejné.

5.2 Příbuznost jazyků

Jako určitý protiklad porovnávání počítačového kódu stojí využití vzdálenosti řetězců k porovnávání jazyků a jejich příbuznosti popsané [11]. „Back in 1967[14] the Croat linguist Ž. Muljačić introduced what appears to us as a natural *fuzzy* generalization of crisp Hamming distances between binary strings of fixed length *n*, called henceforth *Muljačić distance*; he wanted to show that Dalmatic, now an extinct language, is a bridge between the Western group of Romance languages and the Eastern group, mainly Romanian and its variants.“ [11] V roce 1967 představil chorvatský lingvista Ž. Muljačić tzv. Muljačićovu vzdálenost vycházející z Hammingovy vzdálenosti. Chtěl dokázat, že dalmátština, nyní vymřelý jazyk, je přechod mezi skupinou západních románských jazyků a skupinou východních románských jazyků.

„Let us ask the (politically incorrect and linguistically untenable) question: is Dalmatic a dialect of Italian or of Romanian? A minimum-distance decoder points to Romanian, $d(R, D) = 11.5 < d(I, D) = 13$, an unreliable verdict since the distinguishability between Italian and Romanian is only $\delta(R, I) = 9.5$. With even less political correctness, let us ask whether Provencal is a dialect of French or of Italian, taking for granted that it *must* be a dialect of the two: the verdict is French, $d(F, Pr) = 10.5 < d(I, Pr) = 16.5$, and now the distinguishability is high enough, $\delta(F, I) = 11$. Notice that we are ignoring the effect of possible homoplasies, unavoidably so in lack of a well-established evolutionary model comparable to those of bioinformatics. We are confident that the notion of distinguishability as opposed to distance may prove useful to build such models also in a linguistic context.“ [11]

Je tedy dalmátština dialektem italštiny nebo rumunštiny? Minimální vzdálenost ukazuje na rumunštinu (*d(rumunština, dalmátština) = 11.5 < d(italština, dalmátština) = 13*), nicméně je toto rozhodnutí nespolehlivé vzhledem k vzdálenosti mezi italštinou a rumunštinou (jen δ(*rumunština*, *italština*) = 9.5).

„We did not present new material of direct linguistic interest, but rather put to work old and new tools for linguistic classification and linguistic evolution on the basis of historical data, in the hope that Muljačić' ideas might be successfully revived, extended and applied to up-to-date material.“ [11] Tato práce nepředstavuje nový nápad, spíše spojuje staré a nové postupy a používá je na stará data.

5.3 Porovnávání matematických stromů

Zcela odlišné od porovnávání textu – ať jazyků nebo kódu – je využití vzdálenosti řetězců na grafy, přesněji stromy. „The first and most widely-used method for comparing trees is the *tree edit distance*, introduced by Tai [24] as an extension of the well-known string edit distance. Tai allows insertion, deletion and substitution of vertices in order to convert a source tree T_s into a target tree T_t . A cost function is then applied to these operations (most commonly setting the cost of each transformation to unity), and the minimum number of these operations is defined to be the distance between T_s and T_t .“ [12]

Nejpoužívanější metodou pro porovnávání stromů je stromová editační vzdálenost, což je rozšížení klasické editační vzdálenosti pro řetězce. Tato metoda povoluje vložení, smazání a nahrazení vrcholů za účelem změnění výchozího stromu T_s do cílového stromu T_t. Pak se k operacím použije cenová funkce (obvykle je cena každé transformace nastavena na jednotku) a minimální počet operací představuje vzdálenost mezi T_s a T_t.

„… a new tree distance measure with several appealing properties. First, it is an edit distance, defined intuitively as the minimum number of atomic local moves of vertices up and down required to turn one tree into the other, weighted by the size of the moved subtree. Second, it is not only intuitive but is also a metric distance, meaning it is easy to use in a wide range of information retrieval and machine learning algorithms. For example, distance-based methods for clustering often require the distance measure to be metric, and metric properties are also used for efficient document retrieval in databases. Third, it can be computed in a time that is quadratic in the total number of vertices in the trees. Finally, our method produces a consensus tree as part of the procedure, allowing us to compute the agreement between a set of trees at no additional cost.“ [12]

Nový způsob výpočtu stromové editační vzdálenosti má několik vlastností. Zaprvé, je to editační vzdálenost, intuitivně definována jako minimální počet základních přesunů vrcholů nahoru a dolů, které jsou vyžadovány pro změnu jednoho stromu na druhý, vážené velikostí přesunutého podstromu. Zadruhé, je to nejen intuitivní, ale také metrická vzdálenost, což znamená, že je snadno použitelná v široké škále algoritmů získávání informací a strojového učení. Například na vzdálenosti založené metody pro sdružování často vyžadují, aby měření vzdálenosti bylo metrické, a vlastnosti metrik se také používají pro efektivní vyhledávání dokumentů v databázích. Zatřetí, může být spočítána v čase odpovídající druhé mocnině celkového počtu vrcholů stromů. Konečně, tato metoda vytváří shodné stromy jako důsledek procesu, což umožňuje spočítat shodu mezi sadou stromů bez dodatečných nákladů.

5.4 Porovnávání DNA

Další využití porovnávání řetězců je již v úvodu zmíněné porovnávání DNA. Na rozdíl od předešlých příkladu použití, má porovnávání DNA a zkoumání DNA jistá omezení. „DNA sequences include health and other information about patients and their families. The disclosure of such genomic sequences could harm patients from different perspectives such as affecting the employment and the education opportunities.“ [13] Řetězec DNA zahrnuje zdravotní a další informace o pacientech a jejich rodinách. Zveřejnění takových řetězců by mohlo v různých směrech poškodit pacienty, jako například ovlivnění pracovních nebo vzdělávacích příležitostí.

Stejný článek zmiňuje ještě jeden problém při porovnávání řetězců DNA. „For human genomic data, edit distance seems to capture the requirement as we can find similar patients [1] based on genomic information. However, this superiority comes with a cost as edit distance is a quadratic time algorithm. That is, given two strings with n lengths, it requires $O(n^2)$ operations to compute the edit distance; this is not acceptable for long string sequences. For this reason, edit distance problem has been studied over the years by the theoretical computer science community in order to find a better alternative, a faster algorithm [5, 6], or an approximation algorithm. Particularly, in human genomic data where we have billions of base pairs and genomic sequences are constructed with nucleotides (*A, T, G, C*), this algorithm falls short as most datasets contain millions of records.“ [13]

Editační vzdálenost se může jevit jako dobrý způsob porovnávání lidského DNA. Nicméně, výpočet editační vzdálenosti je kvadraticky operačně náročný, tj. na porovnání dvou řetězců o délce n, vyžaduje $O(n^2)$ operací. To není vhodné pro dlouhé řetězce. Proto je problém editační vzdálenosti po léta studován za účelem nalezení lepší alternativy – rychlejšího nebo přibližného algoritmu. Pro lidské DNA je algoritmus editační vzdálenosti zcela nevhodný, protože lidská DNA obsahuje řetězce obsahující miliony nukleotidů (A, T, G, C).

5.5 Kruhová editační vzdálenost

Jak bylo zmíněno, výpočet editační vzdálenosti je časově náročný. Proto je v [14] popsán nový nástroj pro výpočet podobnosti řetězců pomocí kruhové editační vzdálenosti. „The cyclic edit distance (CED) problem can be defined as follows. Given a sequence x of length m and a sequence y of length n, find the minimal edit distance between any conjugate (cyclic rotation) of x and any conjugate of y.“ [14] Problém kruhové editační vzdálenosti může být definován následovně: Je dán řetězec x o délce m a řetězec y o délce n, najděte nejmenší editační vzdálenost mezi sdruženým (kruhová rotace) x a sdruženým y.

„In many applications it is common to consider sequences with circular structure: for instance, the orientation of two images or the leftmost position of two linearised circular DNA sequences may be *irrelevant*.

In [21], the authors show that computing the edit distance can be used to classify handwritten digits, where the contours of the digits are represented with an 8-direction chain-code [11]; a se- quence over an eight-letter alphabet, representing the eight cardinal directions that the contour faces when following the outline of an image in a clockwise motion.“ [14]

V mnoha aplikacích je obvyklé posoudit řetězec s kruhovou strukturou: například orientace dvou obrázků, nebo levostranná poloha dvou lineárních kruhových řetězců DNA mohou být irelevantní.

Autoři se dále odkazují na práci, ve které bylo ukázáno, že výpočet editační vzdálenosti může být použit na klasifikaci ručně psaných číslic, kde jsou obrysy číslic reprezentovány osmi-směrovým řetězovým kódem. Řetězec osmipísmenné abecedy představuje osm základních směrů, kam obrys míří při sledování hranice obrazu ve směru hodinových ručiček.

Výpočet probíhá následovně:

„1. The rotation of x that minimises a generalisation of the q -gram distance between x and y is computed using the algorithm in [13];

2. A refinement on this rotation of x is carried out by examining only some short prefixes and suffixes of the rotation and sequence y;

3. Finally, the edit distance between the refined rotation of x and sequence y is computed.“ [14]

1. Pomocí daného algoritmu se vypočítá rotace řetězce *x*, tak aby se minimalizovalo zobecnění vzdálenosti q-gram mezi řetězcem *x* a *y*. 2. Upřesnění této rotace *x* se provádí

zkoumáním pouze některých krátkých předpon a přípon rotace řetězce *y*. 3 Editační vzdálenost mezi zlepšenou rotací *x* a řetězcem *y*.

Velmi zajímavým využitím tohoto nástroje je již zmíněné třídění ručně psaných číslic. „Handwritten digits from the MNIST database [16] were also used and sorted into ten sets. Each image was placed in one of ten datasets, depending on the value of the drawn digit. Each hand- written digit was in the form of a 28 × 28 matrix consisting of pixel values. 5000 of the 60,000 images were extracted and con- verted into binary matrices. A normalised 8-direction chain-code was produced for the handwritten digits, where a subset can be found in Fig. 3. Normalising the chain-code allows the image to be treated as a circular sequence of minimum magnitude. This pro- duces a sequence independent of the rotation of the image. This was calculated by identifying the number of direction changes between two adjacent elements of the chain-code in an anticlockwise direction (see [12], for details).“[4] [14]

Pro test byly použity ručně psané číslice z MNIST databáze. Před testem byly roztříděny do deseti skupin. Každý obrázek byl umístěn do jedné z deseti skupin dat, v závislosti na hodnotě číslice. Každá ručně psaná číslice měla formát matice 28 × 28, která se skládala z hodnot pixelů. 5000 z 60 000 obrázků bylo extrahováno a přeměněno na binární matice. Pro ručně psané číslice byl vytvořen normalizovaný osmi-směrový kód. Normalizování kódu umožňuje, aby byl obrázek zpracován jako kruhový řetězec minimální velikosti. To vytváří řetězec nezávislý na rotaci obrázku. Toto bylo spočítáno pomocí identifikace počtu směrových změn mezi dvěma sousedními prvky řetězového kódu proti směru hodinových ručiček.

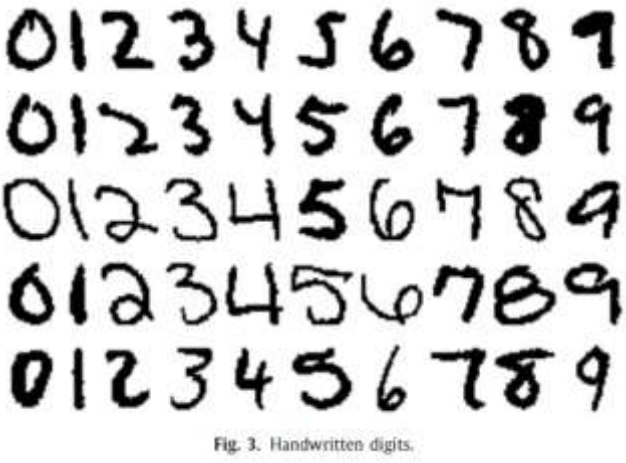


Fig. 3. Handwritten digits.

Obrázek 10 - Ručně psané číslice

Zdroj: [14]

[4] Fig. 3: viz Obrázek 10

5.6 Blokování vulgarismů

Kromě časové náročnosti na výpočet může porovnávání řetězců narazit i na další obtíže. S jednou z nich se setkávají nástroje pro blokování nevhodných slov na internetových diskuzích. „The most obvious moderation tool is blocking comments containing exact copies of instances extracted from a blacklist of plain obscenities. Such a simple filter usually operates on a verbatim-comparison basis, failing to detect terms with involuntary typos or misspellings, or, more worryingly, deliberate obfuscation due to transliteration or segmentation of the letters (symbols) comprising the obscenity. As a result, such filters are insensitive to what we shall call the *homoglyph* anomaly. Homoglyphs are a pair of symbols whose graphical depiction is almost identical (e.g., a ↔ @, i ↔ ¡, S ↔ $), although their computer encodings are completely different.[1] Thus, we refer to a pair of homoglyphs as "twin" characters. An illustration of homoglyphs, including bogus segmentation symbols, is shown in Figure 1."[5] [15]

Nejčastějším způsobem je blokovat komentáře obsahující přesné kopie slov uložených na černé listině běžných vulgarismů. Takový jednoduchý filtr obvykle pracuje na doslovném srovnávání, takže nedokáže rozpoznat slova s překlepy, pravopisnými chybami, nebo – což je znepokojující – záměrně chybná slova v důsledku transliterace nebo oddělení písmen, které obsahují vulgarismus. Ve výsledku jsou takové filtry necitlivé na to, co autoři článku nazývají *homoglifová* anomálie. Homoglifní jsou dvojice symbolů, které graficky vypadají téměř identicky (např. a ↔ @), přesto je jejich počítačové kódování zcela odlišné. Ukázka dalších takových dvojic je na obrázku 11. Jako příklad slova psaného pomocí homoglifů autor uvádí: *„writing the foul word shit obfuscated as s.h.i.t or $h¡t, or even worse, $.h....¡.t"* [15].

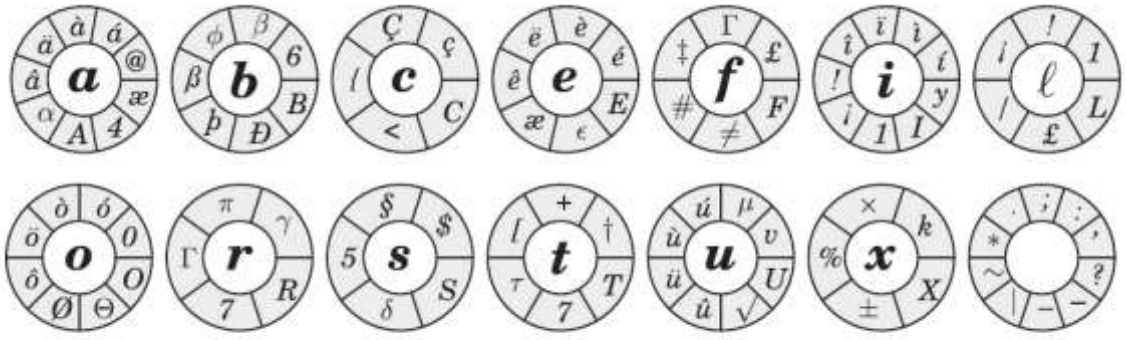


Fig. 1. ASCII code homoglyphs or twin characters for a subset of representative letters ($\{a, b, c, e, f, i, \ell, o, r, s, t, u, x\}$) in the Latin alphabet. At the bottom-right corner, the family of twin bogus segmentators is shown (associated with the blank space).

Obrázek 11 - Ukázka vizuálně podobných znaků

Zdroj: [15]

[5] Figure 1: viz Obrázek 11

„Once a character set and twins lists are defined, the method only requires setting up a seed vocabulary of canonical obscenities; no additional training is needed. However, we reckon that maintaining the vocabulary is not a straightforward task since users keep on conceiving novel obfuscating tricks or simply because of Internet slang and netspeak evolution, particularly in youth communities (the use of abbreviations, acronyms, spellingmodifications, or abandonment of orthographic rules, e.g., see Shaari and Bataineh [2015], Crystal [2008], and Hodson [2016]). We speculate that it would be feasible to maintain linguistic models on each obscenity in the vocabulary while continuously screening user content to identify topics or novelties requiring maintenance in the seed vocabulary; parametric (e.g., Xiang et al. [2012]), non-parametric (e.g., Pimentel et al. [2014]), or ranking approaches (e.g., D'Haro and Banchs [2015]) could be considered for this purpose.“ [15]

Jakmile je sada znaků a seznam dvojic definován, popsaná metoda potřebuje už jen slovník základních vulgarismů, nic dalšího není potřeba. Nicméně, nastavení slovníku není jednoduchým úkolem, protože uživatelé nacházejí nové triky nebo prostě proto, že internetový slang se vyvíjí, zejména pak v komunikaci mladých (používání zkratek, akronymů, hláskování, a další). Zdá se, že by bylo možné udržovat jazykové modely pro vulgarismy a současně prohledávat uživatelský obsah pro údržbu seznamu vulgarismů.

5.7 Detekce pravopisných chyb

Posledním využitím porovnávání řetězců, které zde bude zmíněno, je detekce pravopisných chyb. „…a method specially designed for automatically correcting spelling errors in learner English[1]. These spelling errors, which are often affected by the writer's mother tongue, frequently appear in learner English." Poznámka pod čarou: „In this paper, *learner English* refers to English as a foreign language." [16] V citovaném článku je popsána nová metoda speciálně vytvořená pro automatickou opravu pravopisných chyb v učené angličtině (zde se pod pojmem „učená angličtina" myslí angličtina jako cizí jazyk). Tyto pravopisné chyby, které jsou obvykle způsobeny pisatelovým mateřským jazykem, se často objevují při učení angličtiny.

„Conventional spelling error correction methods are often based on the edit distance[6] between a spelling error and its correction candidates; they choose the correction that minimizes the edit distance from the original misspelled word. Flor and Futagi[7] show that this method can be augmented by phonetic information." [16] Konvenční metody korekce pravopisné chyby jsou obvykle založeny na editační vzdálenosti mezi slova s pravopisnou chybou a možnostmi opravy. Zvolí se oprava s nejmenší editační vzdáleností od chybného slova. Taková metoda může být rozšířena o fonetické informace.

„This paper proposes a method to disambiguate these multiple possible correction candidates, based on the following idea: A misspelled word has the same meaning as its correct form despite its misspelling, and thus the two appear in similar semantic environments." [16] Nová metoda popsaná v článku vychází z následující myšlenky: Špatně napsané slovo má stejný význam jako jeho správná forma, přestože je chybně napsané, a tak se nacházejí ve stejném sémantickém prostředí.

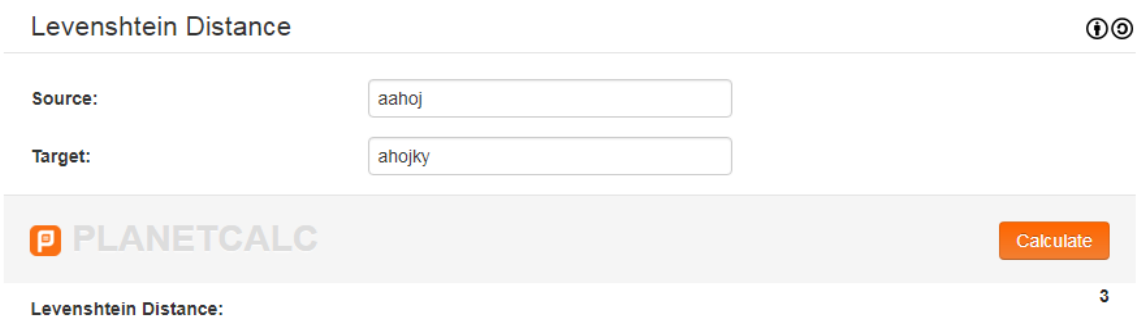
6 Rešerše dostupných nástrojů

Nástroje pro počítání vzdálenosti mezi řetězci lze rozdělit do dvou hlavních skupin: programy nebo webové aplikace pro širší veřejnost a implementace v programovacích jazycích připravené pro programátory. Mezi nástroje se zde nebudou počítat postupy metrik nebo algoritmy.

Tato rešerše probíhala v měsíci červnu a na přelomu června a července 2017. Byla ukončena k 6. 7. 2017. Nástroje byly vyhledávány na internetu pomocí vyhledávače Google. Klíčová slova pro programy a webové aplikace byly: *edit distance calculator, edit distance calculator program, levenshtein distance calculator, hamming distance calculator, jaccard distance calculator, jaro distance calculator, jaro-winkler distance calculator, damerau-levenshtein distance calculator*.

6.1 Programy a webové aplikace

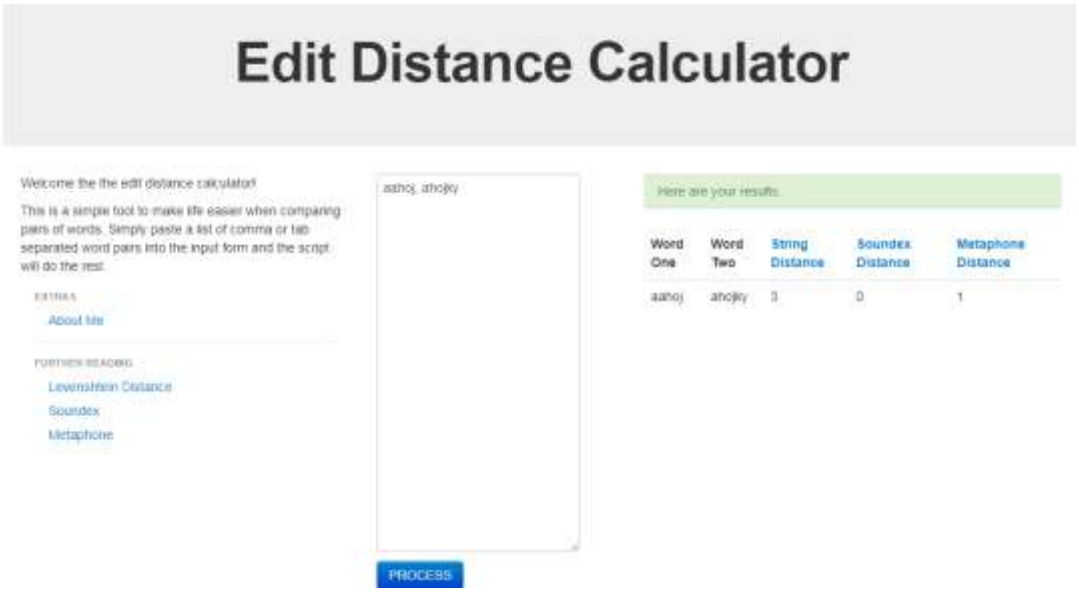
V této části byly vyhledávány počítačové programy a webové aplikace, které počítají vzdálenost řetězců. Bohužel se nepodařilo najít žádný počítačový program. Na druhou stranu, podařilo se najít různé webové aplikace. Na obrázku 14 až 18 jsou webové aplikace počítající Levenštejnovu vzdálenost. Každá z nich byla otestována dvojicí slov „aahoj“ a „ahojky“.



Obrázek 12 - Pouze výsledná vzdálenost

Zdroj: vlastní zpracování

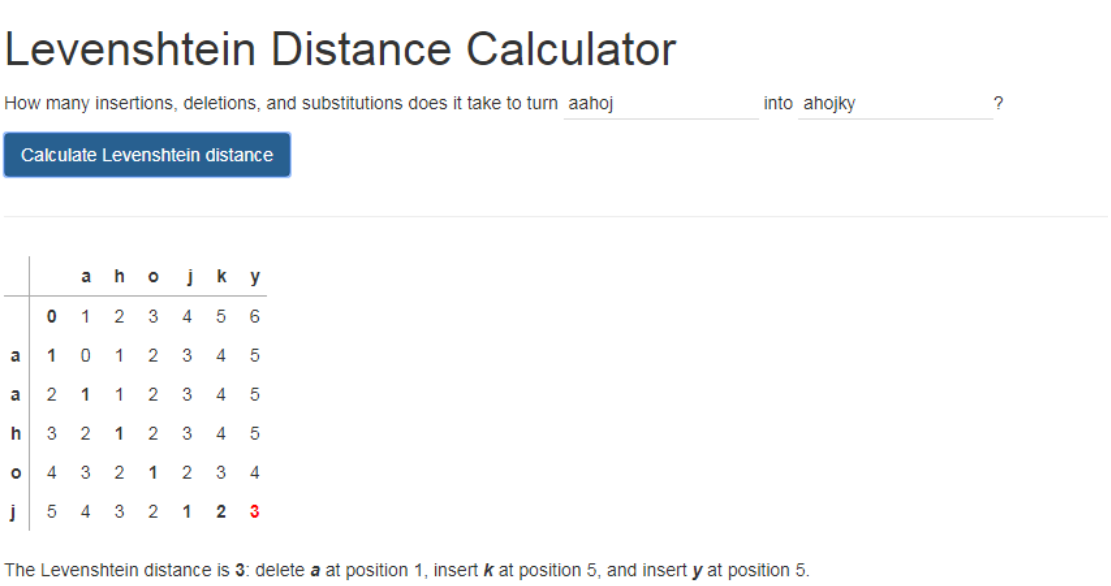
Jako první je zde [17] na obrázku 12. Tato aplikace umožňuje pouze spočítat vzdálenost vždy pouze pro jedinou dvojici slov.



Obrázek 13 – Fonetická vzdálenost – soundex a metaphone

Zdroj: vlastní zpracování

Na obrázku 13 je zachycena [18]. Zde je již možné zadat více dvojic slov. Kromě toho počítá také *soundex* a *metaphone*, které porovnají fonetickou podobu řetězců.[6]

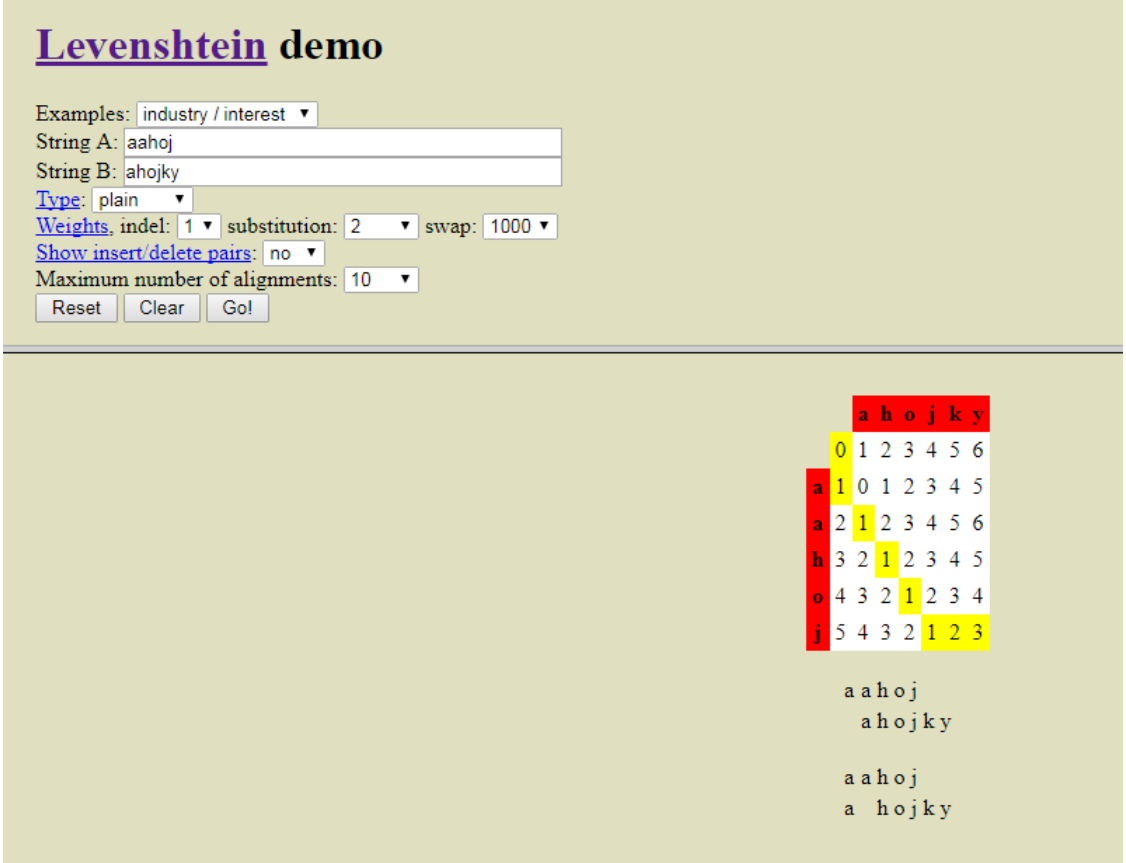


Obrázek 14 - Výpočtová matice a postup úpravy

Zdroj: vlastní zpracování

[6] Soundex a metaphone pracují s anglickou fonetikou, proto se jimi tato práce nebude zabývat detailněji.

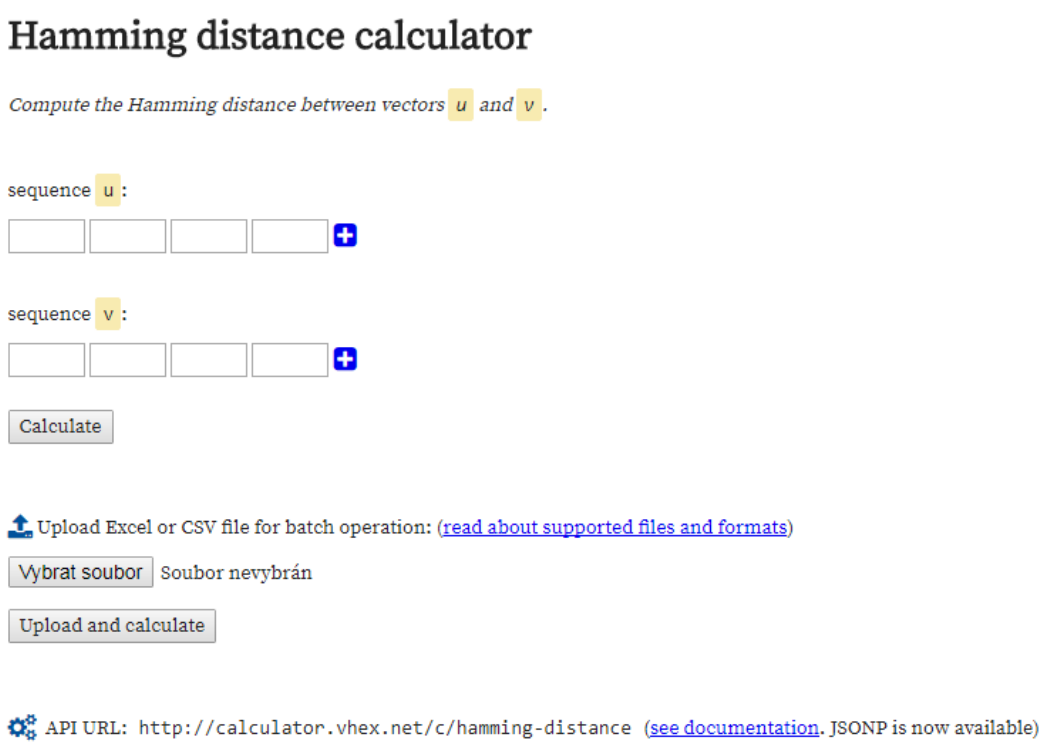
[19] (obrázek 14) sice umožňuje zadat pouze jednu dvojici, na druhou stranu zobrazuje výpočtovou matici a slovní popis provedených operací.



Obrázek 15 - Výpočtová matice a způsob aplikování

Zdroj: vlastní zpracování

Aplikace [20] na obrázku 15 porovnává pouze jednu dvojici najednou, ale zobrazuje výpočtovou matici a ukazuje, jak se z prvního slova stane druhé.



Obrázek 16 - Jediná nalezená Hammingova vzdálenost

Zdroj: vlastní zpracování

Obrázek 16 ukazuje [21], jedinou aplikaci pro výpočet Hammingovi vzdálenosti, kterou se podařilo najít. Tato aplikace počítá pouze vektory s nastavitelnou velikostí. Taktéž umožňuje použít načtení souboru.

Nepodařilo se najít kalkulátory pro výpočet jiných vzdáleností řetězců.

6.2 Předpřipravené implementace

6.2.1 Java

Od verze 1.0, balíček *org.apache.commons.text.similarity*, nutno stáhnout.

Tento balíček obsahuje nástroje pro porovnávání řetězců. Jsou v něm implementovány vzdálenosti: Cosinova, Hammingova, Jaccardova, Jaro-Winklerova, Levenštejnova a vzdálenost nejdelší společné posloupnosti.

Každá implementace umí spočítat danou vzdálenost. Ceny jednotlivých operací jsou vždy 1 (viz kapitola 4.2). Levenštejnova vzdálenost zde má dvě implementace: první jen počítá, druhá podává detailnější přehled – počty jednotlivých operací (vložení, smazání, náhrada).

Kromě výše zmíněných vzdáleností poskytuje tento balíček další možnosti – viz obrázek 17. Založeno na [22].

Interface Summary

Interface	Description
EditDistance<R>	Interface for Edit Distances.
SimilarityScore<R>	Interface for the concept of a string similarity score.

Class Summary

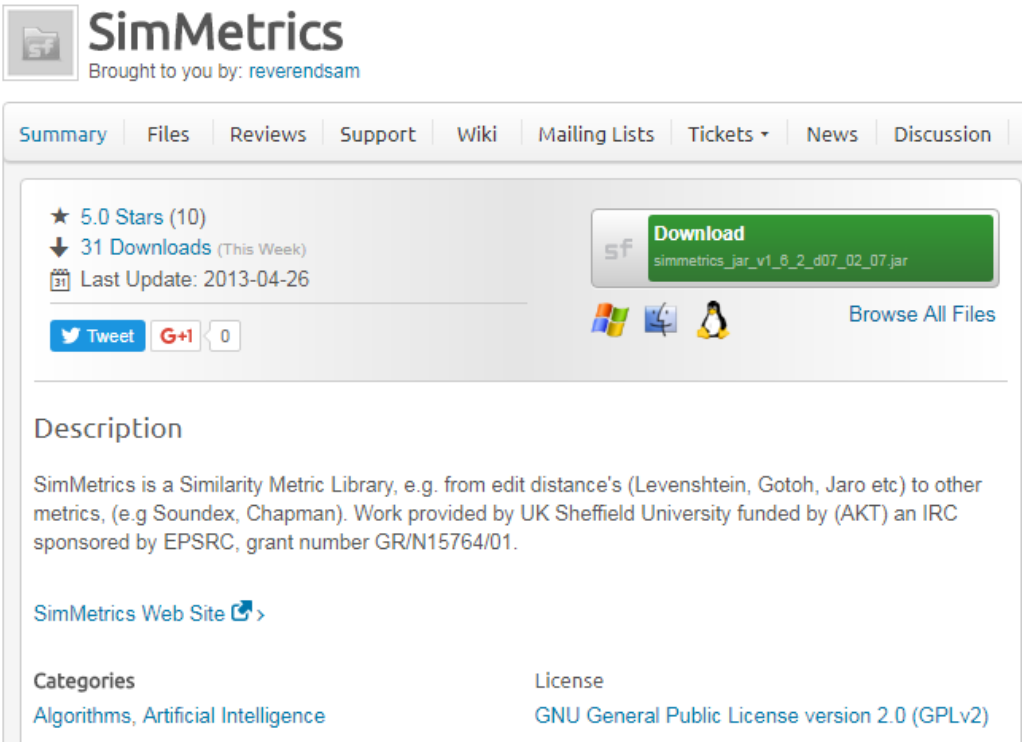
Class	Description
CosineDistance	Measures the cosine distance between two character sequences.
CosineSimilarity	Measures the Cosine similarity of two vectors of an inner product space and compares the angle between them.
EditDistanceFrom<R>	This stores a `EditDistance` implementation and a `CharSequence` "left" string.
FuzzyScore	A matching algorithm that is similar to the searching algorithms implemented in editors such as Sublime Text, TextMate, Atom and others.
HammingDistance	The hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.
JaccardDistance	Measures the Jaccard distance of two sets of character sequence.
JaccardSimilarity	Measures the Jaccard similarity (aka Jaccard index) of two sets of character sequence.
JaroWinklerDistance	A similarity algorithm indicating the percentage of matched characters between two character sequences.
LevenshteinDetailedDistance	An algorithm for measuring the difference between two character sequences.
LevenshteinDistance	An algorithm for measuring the difference between two character sequences.
LevenshteinResults	Container class to store Levenshtein distance between two character sequences.
LongestCommonSubsequence	A similarity algorithm indicating the length of the longest common subsequence between two strings.
LongestCommonSubsequenceDistance	An edit distance algorithm based on the length of the longest common subsequence between two strings.
SimilarityScoreFrom<R>	This stores a `SimilarityScore` implementation and a `CharSequence` "left" string.

Obrázek 17 - Obsah balíčku *similarity*

Zdroj: vlastní zpracování

6.2.2 C/C++/C#

Pro jazyky C, C++ a C# nebyly nalezeny žádné oficiální implementace. Založeno na [23, 24, 25]. Existují ale i neoficiální implementace. Na obrázku 18 je ukázka jednoho takového případu. Tato implementace je pro C# i Javu.



Obrázek 18 - SimMetrics, implementace vzdáleností řetězců

Zdroj: vlastní zpracování

Založeno na [26].

6.2.3 PHP

PHP (verze PHP 4, PHP 5, PHP 7) poskytuje funkci, která spočítá Levenštejnovu vzdálenost. Tato funkce má dvě možnosti výpočtu – viz obrázek 19.

```
/**
* funkce vraci cislo - Levenstejnovu vzdalenost
*/
//implicitni ceny operaci 1
$lev1 = levenshtein($string1, $string2);

//explicitne zadane ceny v poradi      vlozeni, nahrada,   smazani
$lev2 = levenshtein($string1, $string2,$insert,$substitute,$delete);
```

Obrázek 19 - Levenštejnova vzdálenost v PHP

Zdroj: vlastní zpracování

Kromě toho také poskytuje tři další funkce, které nepočítají vzdálenost, ale vzdáleně s tímto tématem souvisí. Viz obrázek 20.

```php
/**
* funkce vraci cislo udavajici pocet shodnych znaku
*/
//pocet stejnych znaku
$sim1 = similar_text($string1, $string2);
//pocet stejnych znaku v procentech
$sim2 = similar_text($string1, $string2, $percent);

/**
* obe funkce prevedou retezec na specialni kod: pismeno a 3 cisla
* tento kod znaci fonetickou podobu (v anglictine)
* dva ruzne retezce mohou mit stejny kod
* vyuziva se pri vyhledavani v databazich
*/
//metaphone
$met = metaphone($string);
//soundex
$sou = soundex($string);
```

Obrázek 20 - Další metody pro práci s řetězci v PHP

Zdroj: vlastní zpracování

Založeno na [27].

6.2.4 Python

Pro jazyk Python lze stáhnout balíčky s různými implementacemi vzdálenosti – viz obrázek 21, červeně vyznačené pracují se vzdáleností řetězců.

Package	Weight	Description
Edit_Distance 1.0.0	16	Computing edit distance on arbitrary Python sequences.
Brew-Distance 1.0.0	14	A Python module that implements the Brew edit distance algorithm
editdistance 0.3.1	14	Fast implementation of the edit distance(Levenshtein distance)
editdist 0.1	10	Calculate Levenshtein's edit distance
edlib 1.1.2.post2	10	Lightweight, super fast library for sequence alignment using edit (Levenshtein) distance.
python-Levenshtein 0.12.0	9	Python extension for computing string edit distances and similarities.
django-earthdistance 1	8	Add support for PostgreSQL earthdistance extension to Django
edittag 1.1	8	Design and check sets of edit metric sequence tags.
StringDist 1.0.9	7	This package provides the stringdist module, which includes several functions for calculating string distances. Under the hood, a C extension module is preferentially used for optimal performance, with an automatic fallback to a Python implementation.
subdist 0.2.1	7	Substring edit distance
pyhacrf-datamade 0.2.1	6	Hidden alignment conditional random field, a discriminative string edit distance
leven 1.0.4	5	Levenshtein edit distance library
pyhacrf 0.1.2	5	Hidden alignment conditional random field, a discriminative string edit distance
pysegbase 1.2.9	5	Graph Cut based 3D segmentation with editor
pyxDamerauLevenshtein 1.4.1	5	pyxDamerauLevenshtein implements the Damerau-Levenshtein (DL) edit distance algorithm for Python in Cython for high performance.
sequtils 0.0.9	5	Python utilities for sequence comparison, quantification, and feature extraction.
zss 1.1.4	5	Tree edit distance using the Zhang Shasha algorithm
highered 0.2.1	4	Learnable Edit Distance Using PyHacrf
gabbar 0.5.1	3	Guarding OpenStreetMap from invalid or suspicious edits!
intervaltree 2.1.0	3	Editable interval tree data structure for Python 2 and 3
lobster 0.1.0	3	Split audio files into tracks with a single command
opengeode 1.5.2	3	A tiny, free SDL editor for TASTE
abstraction 2017.1.16.1534	2	machine learning framework
ACO-Pants 0.5.2	2	A Python3 implementation of the ACO Meta-Heuristic
aioautomatic 0.4.0	2	Asyncio library for the Automatic API
angles 2.0	2	Classes for representing angles, and positions on a unit sphere.
antispoofing.lbptop 2.0.3	2	LBP-TOP based countermeasure against facial spoofing attacks
cadquery 1.0.0	2	CadQuery is a parametric scripting language for creating and traversing CAD models
cbmpy 0.7.15	2	PySCeS-CBMPy
challenge.uccs 1.0.2	2	Running baseline experiments and evaluations for the IJCB 2017 UCCS challenge
ckanext-tagmanager 0.0.2	2	Tag management for CKAN portals
CodeReview 0.3.0	2	CodeReview is a Python 3 / Qt5 GUI to perform code review on files and Git repositories.
collective.anonymousbrowser 0.11	2	A zope testbrowser extension with useragent faking and proxy abilities by Makina Corpus
Corrfunc 2.0.0	2	Blazing fast correlation functions on the CPU
cosmoabc 1.0.5	2	Python ABC sampler
daftlistings 1.1.2	2	A library that enables programmatic interaction with daft.ie
delaunator 1.0.4	2	2D Delaunay Triangulation in C++ with Python wrapper
disco-dop 0.5.2	2	Discontinuous Data-Oriented Parsing
django-netjsongraph 0.3.1	2	Reusable django app for collecting and visualizing network topology
django-oscar-stores 0.8	2	An extension for Oscar to include stores
django-units 0.2.1	2	A Django app to convert units between different systems
drive-ami 1.0.7	2	An interface layer for scripting the AMI-Reduce pipeline.
dummy_package 0.2.0	2	Python library for string matching.
dummy_package1 0.1.0	2	Python library for performing string similarity joins.
ebdata 1.2	2	Data scraper infrastructure for OpenBlock (hyperlocal news for Django)
ebpub 1.2	2	Core models and views for OpenBlock (hyperlocal news for Django)
exodata 2.1.7	2	Exoplanet catalogue interface
FAdo 1.3.3	2	Formal Languages manipulation module
fastss 0.095	2	String similarity searching using the FastSS algorithm.
FiPy 3.1.3	2	A finite volume PDE solver in Python
latticegraph-designer 1.0a1	2	PyQt based GUI tool which allows to visualize, design and export the lattice graph models.
odoo8-addon-base-name-search-improved 8.0.1.0.1.99.dev12	2	Friendlier search when typing in relation fields
odoo8-addon-web-menu-autohide 8.0.1.0.0	2	Hide top and left menu bar
odoo9-addon-base-name-search-improved 9.0.1.0.0.99.dev9	2	Friendlier search when typing in relation fields

Obrázek 21 - Python - balíčky s implementací vzdálenosti řetězců

Zdroj: vlastní zpracování

Založeno na [28].

6.2.5 R

Pro jazyk R byl vytvořen balíček *stringdist* (viz obrázek 22). Mimo jiné počítá Hammingovu, Levenštejnovu a Damerau-Levenštejnovu vzdálenost.

Obrázek 22 - Popis balíčku stringdist

Zdroj: vlastní zpracování

Založeno na [29].

6.3 Shrnutí

Z této kapitoly vyplývá (pro souhrn viz tabulka 1), že webové aplikace pro výpočet Levenštejnovi vzdálenosti jsou snadno dostupné, to ale neplatí o dalších metrikách. Ty spíše chybí. Počítačová aplikace nebyla nalezena žádná. Lze tedy předpokládat, že nejsou volně dostupné. Implementace v programovacích jazycích jsou většinou snadno dostupné a opět hlavně Levenštejnova vzdálenost.

Tabulka 1 - Souhrn rešerše nástrojů

Nástroj	**Podporované druhy vzdáleností**
Počítačové programy	----
Webové aplikace	Levenětejnova vzd., Hammingova vzd.
Java	Levenštejnova vzd., Hammingova vzd., Jaro-Winklerova vzd., a další
C, C++	----
C#	Levenštejnova vzd., Jaroova vzd., Soundex, a další
PHP	Levenštejnova vzd., Soundex, MetaPhone
Python	Damerau-Levenštejnova vzd., Levenštejnova vzd., Brew vzd., a další
R	Damerau-Levenštejnova vzd., Levenštejnova vzd., Hammingova vzd., a další

Zdroj: vlastní zpracování

7 Nový nástroj pro porovnávání řetězců

Jak vyplynulo z rešerše aktuálních nástrojů pro porovnávání řetězců, dostupné kalkulátory počítají téměř výhradně Levenštejnovu vzdálenost, nejsou v češtině a nezpracovávají větší množství dat naráz nebo zobrazují pouze výsledek. Žádný z nalezených nástrojů také neumí zpracovávat jiný typ dat než řetězce znaků (slova). Proto je součástí této práce nový nástroj pro porovnávání řetězců.

Pro vývoj byl zvolen programovací jazyk Java a to hlavně proto, že je nezávislý na operačním systému. V souvislosti s tím se nabízí otázka, proč vyvíjet nový nástroj v Javě, když je v tomto jazyce dostupný balíček *similarity* (viz kapitola 6.2.1). Nicméně jak bylo zmíněno v úvodu, je možné srovnávat i jiné řetězce než jen řetězce znaků, což *similarity* neumožňuje. Další důvody a výhody nového řešení budou podrobně vysvětleny později.

Nové řešení výpočtu vzdálenosti řetězců se skládá z několika částí. Zaprvé balíček *metrics*, který počítá jednotlivé vzdálenosti řetězců. Zadruhé pak počítačový program využívající tento balíček. Byl nazván *Kalkulátor vzdálenosti sekvencí objektů* (KVSO) a umožňuje provádět výpočty z grafického uživatelského rozhraní. Zatřetí bylo vytvořeno několik počítačových reprezentací objektů.

7.1 Balíček metrics

Balíček *metrics* má oproti *similarity* několik výhod. Hlavní výhodou je, že je schopen počítat vzdálenost nejen mezi řetězci znaků (slovy), ale i mezi řetězci jiných objektů, například not.

Dalším rozdílem jsou dva druhy výpočtu. První způsob pouze spočítá vzdálenost, podobně jako *similarity*. Druhý způsob poskytuje detailnější informace o výpočtu, jako je výpočetní struktura (např. matice) a slovní popis kroků, které je potřeba provést na změnu prvního řetězce na druhý. U tohoto způsobu je také možné nastavit cenu jednotlivých operací.

Tento nový balíček také zjednodušuje práci s větším množstvím dat. Není nutné zadávat každou dvojici řetězců zvlášť jako u *similarity*, ale stačí zadat řetězec dvojic řetězců. Kromě toho je možné využít paralelní zpracování.

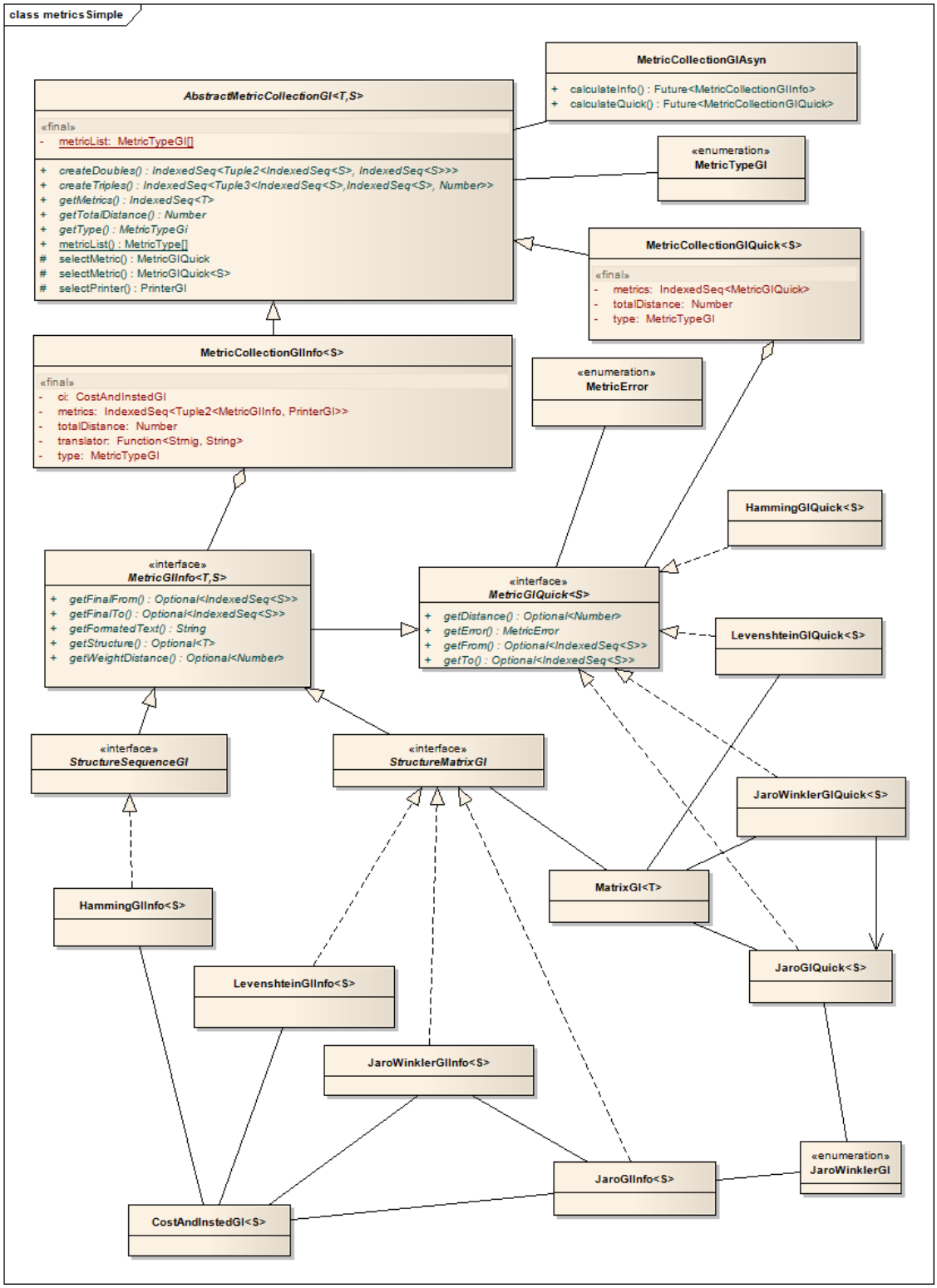
Na druhou stranu je *metrics* limitován počtem druhů výpočtu, pracuje se vzdálenostmi popsanými v této práci (viz kapitola 4). Pro detailnější porovnání viz tabulka 2.

Tabulka 2 - Porovnání similarity a metrics

	similarity	metrics
Výhody	Mnoho druhů výpočtu Snadná použitelnost Snadno rozšiřitelné	Snadná použitelnost Snadno rozšiřitelné Možnost zadat řetězec dvojic řetězců Nehází výjimky Dva módy výpočtu – bez dalších informací – s detailními informacemi o výpočtu Paralelní zpracování
Nevýhody	Zpracovává pouze slova Možnost zadat vždy pouze jednu dvojici slov Při špatném vstupu hází výjimky Neumožňuje získat informace o výpočtu	Méně druhů výpočtu

Zdroj: vlastní zpracování

Zjednodušený diagram tříd balíčku *metrics* je vidět na obrázku 23, pro celý diagram viz Příloha 2. Při tvorbě balíčku byla použita knihovna *io.vavr*, která poskytuje kolekce a další podpůrné třídy.



Obrázek 23 - Diagram tříd balíčku metrics

Zdroj: vlastní zpracování

Výchozí třídou je *AbstractMetricCollectionGI*, která se, spolu se svými potomky *MetricCollectionGIInfo* a *MetricCollectionGIQuick*, stará o zpracování vstupního souboru,

tj. seznamu dvojic řetězců. Generický typ *T* je specifikován v potomcích a *S* je typ objektu, který se porovnává (znak, nota,…). Aby bylo možné provádět výpočet paralelně s jinou částí programu, je do *metrics* umístěna třída *MetricCollectionGIAsyn*. Ta zajistí vytvoření a zpracování výše zmíněných tříd v samostatném vláknu.

Každá třída, počítající vzdálenost řetězců, dědí od rozhraní *MetricGIQuick*. To poskytuje nejzákladnější informace o proběhlém výpočtu: zda vše proběhlo v pořádku, nebo se vyskytl nějaký problém a jaký, vypočítaná vzdálenost a původní řetězce. Pro třídy, které počítají vzdálenost řetězců s dalšími informacemi, je zde rozhraní *MetricGIInfo*, které dědí od *MetricGIQuick*. Tyto třídy získávají v konstruktoru funkci *translator*, která slouží k překladu popisu výpočtu.

Kromě výše zmíněného a tříd implementující zmíněné, jsou v diagramu ještě třídy podpůrné (např. třída pro matice), z nichž je asi nejdůležitější třída *CostAndInsted*. Díky ní je možné do výpočtu s dalšími informacemi přidat rozšiřující možnosti – cenu jednotlivých operací a způsob označení operací ve výsledných řetězcích.

7.1.1 Jak poznat použité operace

Balíček *metrics* umí zjistit, jaké operace byly použity. K tomu využívá následující algoritmy.

7.1.1.1 Postup pro Hammingovu vzdálenost

Poznat použité operace je možné například z pomocného řetězce. Pokud je na *n-té* pozici 0, nebyla použita žádná operace (**shodné**), pokud je na této pozici 1, byla provedena **substituce**.

7.1.1.2 Postup pro Levenštejnovu vzdálenost

Při zjišťování použitých operací jsou klíčové *cost* (ceny) uvedené u každého prvku.

Nechť je definována matice $n \times m$ a výchozím prvkem je prvek $a_{x,y} = a_{0,0}$:

Pokud prvek $a_{x+1,y+1}$ má uvedenu $cost=0$, je následující prvek $a_{x+1,y+1}$, přičemž se neprovede žádná operace (znaky jsou shodné).

Pokud není splněna předchozí podmínka, jsou definovány tři pomocné proměnné: A, B, C. Pokud existuje prvek $a_{x+k,y+k}$, $k>1$, který má $cost=0$, pak $A = k$, jinak není hodnota stanovena. Pokud existuje prvek $a_{x+k,y+1}$, $k>1$, který má $cost=0$, pak $B = k$, jinak není hodnota stanovena. Pokud existuje prvek $a_{x+1,y+k}$, $k>1$, který má $cost=0$, pak $C = k$, jinak není hodnota stanovena.

Nyní se vybere nejnižší z *A*, *B* a *C*. Nestanovené hodnoty se do výběru nezařazují, v případě rovnosti hodnot se vybírá v abecedním pořadí.

Pokud je nejnižší *A*, pak následující prvek je $a_{x+1,y+1}$. Provedena **náhrada**.

Pokud je nejnižší *B*, pak následující prvek je $a_{x+1,y}$. Provedeno **smazání**.

Pokud je nejnižší *C*, pak následující prvek je $a_{x,y+1}$. Provedeno **vložení**.

Pokud není žádná z pomocných proměnných stanovena, je následující prvek $a_{x+1,y+1}$. Opět provedena **náhrada**.

Pokud není možné přejít na další prvek kvůli velikosti matice, pak:

Jestliže $a_{x,y}$, kde $x = n$, je následující prvek $a_{x,y+1}$. Provedeno **vložení**.

Jestliže $a_{x,y}$, kde $y = m$, je následující prvek $a_{x+1,y}$. Provedeno **smazání**.

		K	O	P	L	L	I	S	A
	0	1	2	3	4	5	6	7	8
K	1	0	1	2	3	4	5	6	7
O	2	1	0	1	2	3	4	5	6
O	3	2	1	1	2	3	4	5	6
L	4	3	2	1	1	2	3	4	5
I	5	4	3	2	2	2	2	3	4
P	6	5	4	3	3	3	3	3	4
A	7	6	5	4	4	4	4	4	3
N	8	7	6	5	5	5	5	5	4

Obrázek 24 – Ukázka operací – Levenštejnova vzdálenost

Zdroj: vlastní zpracování

Pro lepší demonstraci byla vytvořena matice pro řetězce „kooplipan“ a „kopllisa“, viz obrázek 24. Výchozí prvek je $a_{0,0}$, prvek $a_{1,1}$ má *cost = 0*, tudíž je následujícím prvkem a nebyla provedena žádná operace. Stejným způsobem se přejde z prvku $a_{1,1}$ na prvek $a_{2,2}$. Prvek $a_{3,3}$ nemá nulovou *cost*, tudíž se prohledává diagonála, sloupec a řádek od prvku a3,3. Protože prvek $a_{4,4}$ má *cost = 0* (žluté pole), *A = 2*, prvek $a_{6,6}$ má *cost = 0*, takže *B = 4*, *C* není stanoveno. Hodnota *A* je nejnižší, proto je následujícím prvkem $a_{3,3}$ a byla provedena **náhrada**.

Jak již bylo uvedeno, prvek $a_{4,4}$ má *cost = 0*, takže se přejde z prvku $a_{3,3}$ na prvek $a_{4,4}$ bez provedení akce. Prvek $a_{5,5}$ nemá *cost = 0*, takže se prohledává diagonála, sloupec a řádek od $a_{5,5}$. Protože se na diagonále nenachází prvek s *cost = 0*, je *A* nestanoveno. Nyní se prohledávají prvky pod prvkem $a_{5,5}$, nicméně ani zde není prvek s nulovou *cost*, tudíž je *B* také nespecifikováno. Proto se prohledávají prvky vpravo od $a_{5,5}$. Zde je prvek $a_{5,6}$ s *cost = 0* (žluté pole), *C = 2*. Jelikož je C jedinou specifikovanou proměnnou, je dalším prvkem $a_{4,5}$ a bylo provedeno **vložení**.

Odtud se přejde na prvek $a_{5,6}$ bez provedení nějaké operace. Prvek $a_{6,7}$ nemá *cost = 0*, takže se opět prohledává diagonála od $a_{6,7}$. Díky tomu je nalezen prvek $a_{7,8}$ (žluté pole). Následujícím prvkem se tedy stává $a_{6,7}$ a je provedena **náhrada**. Poté se přejde bez operace na prvek $a_{7,8}$. Protože nelze zkoumat diagonálně další prvek kvůli velikosti matice, je následujícím a posledním prvkem prvek $a_{8,8}$ a bylo provedeno **smazání.**

V obrázku jsou jednotlivé použité prvky vyznačeny červeně. Byly použity čtyři operace, což odpovídá vzdálenosti obou řetězců (pravý dolní roh). Jak se tedy „koolipan“ změní na „kopllisa“? Druhé „o“ se změní na „p“ („ko**p**lipan“), za „l“ se vloží druhé „l“ („kopl**l**ipan“), původní „p“ se změní na „s“ („koplli**s**an“) a nakonec se smaže „n“ („kopllisa“).

7.1.1.3 Postup pro Jaroovu vzdálenost

U Jaroovi vzdálenosti se při nalezení použitých operací zkoumá, zda je hodnota buňka 1 či nikoliv. Tento algoritmus je velmi podobný postupu nalezení *c* a *t*.

Nechť je definována matice $n \times m$ a výchozím prvkem je prvek $a_{-1,-1}$:

Pokud $a_{x+1,y+1}$ má hodnotu 1, pak je následujícím prvkem $a_{x+1,y+1}$, přičemž nebyla provedena žádná operace (**shodné**).

Pokud není splněna předchozí podmínka a $x+2< n$, $y+2< m$, $a_{x+2,y+1}$ má hodnotu 1 a zároveň $a_{x+1,y+2}$ taktéž, je dalším prvkem $a_{x+2,y+2}$, provedeno **prohození**.

Pokud není splněna předchozí podmínka, jsou definovány tři pomocné proměnné: A, B, C. Pokud existuje prvek $a_{x+k,y+k}$, $k>1$, který má hodnotu 1, pak $A = k$, jinak není hodnota stanovena. Pokud existuje prvek $a_{x+k,y+1}$, $k>1$, který má hodnotu 1, pak $B = k$, jinak není hodnota stanovena. Pokud existuje prvek $a_{x+1,y+k}$, $k>1$, který má hodnotu, pak $C = k$, jinak není hodnota stanovena.

Nyní se vybere nejnižší z *A*, *B* a *C*. Nestanovené hodnoty se do výběru nezařazují, v případě rovnosti hodnot se vybírá v abecedním pořadí.

Pokud je nejnižší *A*, pak následující prvek je $a_{x+1,y+1}$. Provedou se dva kroky. V prvním je následující prvek $a_{x,y+1}$ a ve druhém pak $a_{x+1,y}$, přičemž se nejprve provede **smazání** a poté **vložení** (obdoba náhrady).

Pokud je nejnižší *B*, pak následující prvek je $a_{x+1,y}$. Provedeno **smazání**.

Pokud je nejnižší *C*, pak následující prvek je $a_{x,y+1}$. Provedeno **vložení**.

Pokud není žádná z hodnot stanovena, opět se provede obdoba náhrady (**smazání** a **vložení**).

Pokud není možné přejít na další prvek kvůli velikosti matice, pak:

Jestliže $a_{x,y}$, kde $x = n$, je následující prvek $a_{x,y+1}$. Provedeno **vložení**.

Jestliže $a_{x,y}$, kde $y = m$, je následující prvek $a_{x+1,y}$. Provedeno **smazání**.

	K	P	O	L	L	I	M
K	1	0	0	X	X	X	X
O	0	0	1	0	X	X	X
P	0	1	0	0	0	X	X
L	X	0	0	1	1	0	X
I	X	X	0	0	0	1	0
N	X	X	X	0	0	0	0
N	X	X	X	X	0	0	0

Obrázek 25 - Ukázka operací - Jaroova vzdálenost

Zdroj: vlastní zpracování

Pro snazší pochopení jsou operace demonstrovány na dvojici řetězců „koplinn“ a „kpollim“, viz obrázek 25. Začíná se na pomyslném prvku $a_{-1,-1}$. Protože prvek $a_{0,0}$ má hodnotu 1, je následujícím prvek a nebyla provedena žádná operace. Prvek $a_{1,1}$ nemá hodnotu 1, nicméně prvky $a_{2,1}$ a $a_{1,2}$ ano, takže další prvkem je $a_{2,2}$ a bylo provedeno **prohození**. Díky tomu, že prvek $a_{3,3}$ má hodnotu 1, se stává dalším prvek bez provedení nějaké operace.

Protože prvek $a_{4,4}$ nemá hodnotu 1, testují se prvky v diagonále, sloupci a řadě od něj. V diagonále ani sloupci žádná hodnota 1 není, proto A a B nejsou stanoveny. V řadě však prvek $a_{4,5}$ s hodnotou 1. Následujícím prvkem je tedy $a_{3,4}$ a bylo provedeno **vložení**.

Dalším prvkem je $a_{4,5}$ bez provedení operací. Nyní se zkoumá prvek $a_{5,6}$, ten nemá hodnotu 1 a tato hodnota není nikde pod ním ani vpravo od něj. Tudíž se nejprve přejde na prvek $a_{4,6}$ a ihned poté na $a_{5,6}$. Bylo provedeno **vložení** a **smazání**. Protože nelze zkoumat prvek na diagonále, je dalším prvek $a_{6,6}$ a bylo provedeno **smazání**.

Bylo provedeno pět operací, projité prvky jsou vyznačeny červeně. Jak lze z „koplinn“ vytvořit „kpollim“? Nejprve se prohoní „o“ a „p“ („k**po**linn“), pak se za „l“ vloží další „l“ („kpol**l**inn“). Následně se provede vložení „m“ za „i“ („kpolli**m**nn“) a nakonec se obě „n“ smažou („kpollimn“ a „kpollin“).

7.1.1.4 Postup pro Jaro-Winklerovu vzdálenost

Postup pro poznání použitých operací je stejný jako u Jaroovi vzdálenosti (viz 7.1.1.3 Jaroova vzdálenost).

7.1.2 Testování balíčku metrics

Funkčnost balíčku *metrics* byla otestována na souboru šestnácti dvojic slov, přesněji šestnácti dvojic řetězců znaků. Do tohoto souboru byly použity řetězce, které byly ukázkově porovnávány v této práci, dále různé extrémní případy (oba řetězce stejné, jeden z řetězců prázdný, zcela rozdílné řetězce,…) a další zajímavé dvojice. Zde budou ukázány pouze zestručněné a zaokrouhlené výsledky pro řetězce již použité v této práci. Pro kompletní výsledky viz Příloha 3.

Kromě samotné výsledné vzdálenosti byl zkoumán čas, za který výpočet proběhne. Proto byl každý výpočet zopakován desetkrát a čas zaznamenán.

7.1.2.1 Výsledky Levenštejnovi vzdálenosti

Tabulka 3 – Vybrané výsledky výpočtu Levenštejnovi vzdálenosti s detailem

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	2	590 676	2 659 067	249 822
aahoj	ahojky	3	606 222	2 488 445	206 311
kolo	oko	2	444 840	1 117 111	176 489
koolipan	kopllisa	4	950 107	2 402 889	283 066
koplinn	kpollim	4	921 360	4 108 623	277 200

Zdroj: vlastní zpracování

Tabulka 4 - Vybrané výsledky výpočtu Levenštejnovi vzdálenosti bez dalších informací

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	2	269 182	1 231 022	86 044
aahoj	ahojky	3	255 934	884 401	65 022
kolo	oko	2	341 244	2 159 422	53 289
koolipan	kopllisa	4	384 560	1 101 956	87 022
koplinn	kpollim	4	322 227	989 511	77 245

Zdroj: vlastní zpracování

7.1.2.2 Výsledky Hammingovi vzdálenosti

Hammingova vzdálenost je definována pouze pro stejně dlouhé řetězce. Proto v případě, kdy byly zadány nestejně dlouhé řetězce, nevrátila žádnou vzdálenost (tato pole jsou označena „--“).

Tabulka 5 - Vybrané výsledky výpočtu Hammingovi vzdálenosti s detailem

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	2	270 307	435 600	131 022
aahoj	ahojky	--	147 253	492 312	86 533
kolo	oko	--	128 480	539 734	36 178
koolipan	kopllisa	5	286 392	751 912	129 067
koplinn	kpollim	5	308 244	779 777	97 288

Zdroj: vlastní zpracování

Tabulka 6 - Vybrané výsledky výpočtu Hammingovi vzdálenosti bez dalších informací

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	2	136 351	235 645	50 356
aahoj	ahojky	--	101 591	255 200	31 778
kolo	oko	--	116 796	575 911	31 289
koolipan	kopllisa	5	176 929	438 044	41 066
koplinn	kpollim	5	127 942	261 556	40 577

Zdroj: vlastní zpracování

7.1.2.3 Výsledky Jaroovi vzdálenosti

Tabulka 7 - Vybrané výsledky výpočtu Jaroovi vzdálenosti s detailem

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	0,78	565 254	2 371 600	263 022
aahoj	ahojky	0,82	481 800	1 231 512	248 356
kolo	oko	0,81	340 902	747 023	175 511
koolipan	kopllisa	0,75	622 649	2 005 912	311 422
koplinn	kpollim	0,74	501 063	1 307 289	243 956

Zdroj: vlastní zpracování

Tabulka 8 - Vybrané výsledky výpočtu Jaroovi vzdálenosti bez dalších informací

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	0,78	268 400	374 978	162 800
aahoj	ahojky	0,82	387 200	949 911	165 244
kolo	oko	0,81	179 276	229 778	140 311
koolipan	kopllisa	0,75	393 116	920 578	224 889
koplinn	kpollim	0,74	385 929	990 489	167 200

Zdroj: vlastní zpracování

7.1.2.4 Výsledky Jaro-Winklerovi vzdálenosti

Tabulka 9 - Vybrané výsledky výpočtu Jaro-Winklerovi vzdálenosti s detailem

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	0,87	780 854	3 597 734	275 733
aahoj	ahojky	0,84	405 875	1 297 511	238 578
kolo	oko	0,81	320 858	745 067	194 578
koolipan	kopllisa	0,80	873 938	3 304 889	292 844
koplinn	kpollim	0,77	454 715	1 416 311	241 022

Zdroj: vlastní zpracování

Tabulka 10 - Vybrané výsledky výpočtu Jaro-Winklerovi vzdálenosti bez dalších informací

První řetězec	Druhý řetězec	Vypočítaná vzdálenost	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
houska	housle	0,87	426 116	2 307 556	125 156
aahoj	ahojky	0,84	359 040	1 020 311	112 445
kolo	oko	0,81	284 387	818 400	88 000
koolipan	kopllisa	0,80	553 325	1 587 911	167 200
koplinn	kpollim	0,77	395 560	1 013 956	114 400

Zdroj: vlastní zpracování

Jak vyplývá z obrázku 23, je každá třída, počítající vzdálenost řetězců s detailem, nezávislá na třídě, která počítá vzdálenost řetězců bez dalších informací. Je proto důležité, že pro stejný vstup vracejí stejný výstup.

7.2 Kalkulátor Vzdálenosti Sekvencí Objektů

KVSO je počítačový program, který je určen k určování podobnosti řetězců. Zatímco balíček *metric* je použitelný pouze pro programátory, kteří jej začlení do vlastního programu, KVSO mohou využívat i ostatní uživatelé bez znalosti Javy. Tento program pracuje s těmi druhy vzdálenosti řetězců, které počítá balíček *metrics*.

Program získává data buď ze souborů, nebo přímo z grafického uživatelského rozhraní, kde mohou být vstupní data upravována. Zobrazený výsledek je opět možné uložit do souboru. Kromě samotného výpočtu a zobrazení vstupních dat a výsledku, lze v tomto programu zobrazit detail výpočtu, pokud je k dispozici. Také je možné na základě výsledných dat spočítat podmíněnou entropii (viz 7.2.1).

Tento program vhodné možné použít na porovnávání celé genetické informace. Na druhou stranu by byl vhodný pro porovnávání dvou jazyků. Lze jej také využít ve školství, například jako nástroj pro snazší pochopení a naučení výpočtu vzdálenosti řetězců (matematika), porovnávání slov (lingvistika), nebo zkoumání krátkých řetězců DNA/RNA při hledání mutací (biologie).

7.2.1 Podmíněná entropie

Jednou z možných aplikací podmíněné entropie je fonetické porovnávání jazyků. Na jejím základě lze říci, jak bude posluchač mluvící jazykem A rozumět řečníkovi, který mluví jazykem B. Podmíněná entropie obecně není symetrická – $H(A|B) \neq H(B|A)$ – takže to není druh vzdálenosti, proto také nebyla popsána v této práci. Nicméně aby bylo možné provést výpočet podmíněné entropie, je potřeba vstupní dvojice slov (často zapsané fonetickými znaky) upravit. Právě to umožňují algoritmy popsané v kapitole 7.1.1, proto je výpočet podmíněné entropie součástí programu KSVO (způsob výpočtu podmíněné entropie – viz Příloha 4). Založeno na [30].

7.3 Typy objektů

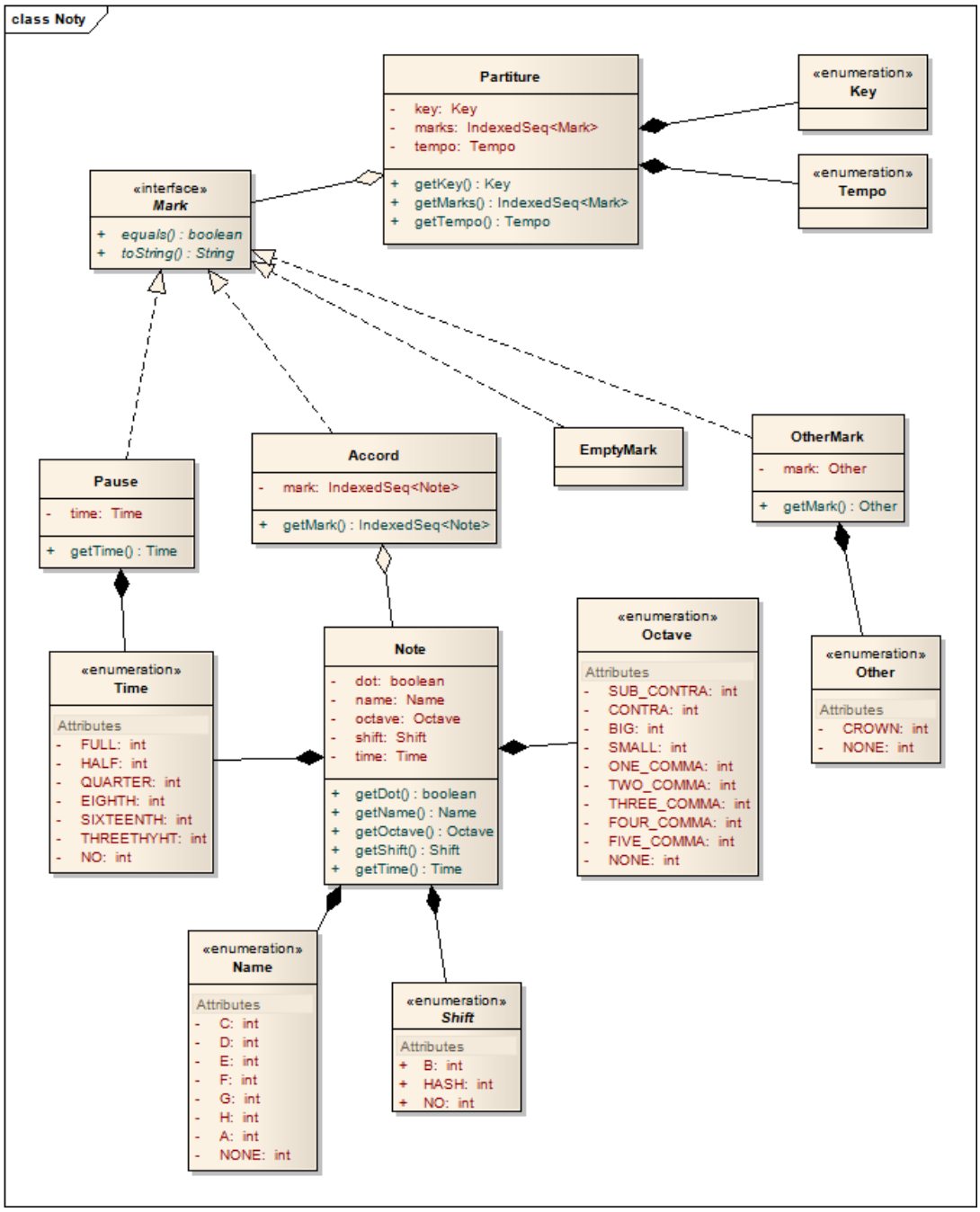
KVSO umožňuje porovnávat několik druhů objektů – znaky (znaky abecedy, speciální symboly, čísla)[7], noty[8] a nukleové báze. Zatímco porovnávání znaků vychází z nativních javovských tříd, pro noty a nukleové báze byla vytvořena počítačová reprezentace.

7.3.1 Noty

Počítačově reprezentované noty jsou umístěny v balíčku *musicnotes*. Při tvorbě byla použita knihovna *io.vavr*. Na základě konzultací a [31] vznikl koncept pro reprezentaci not. V notovém záznamu jsou tři druhy značek – noty či skupiny not (dvojzvuk, popř. akord), pomlky a ostatní značky (např. koruna). Notu lze jednoznačně identifikovat podle pěti atributů: Jméno (c, d, e, f, g, h, a), doby (nota celá, půlová,…), oktávy (malá oktáva, velká oktáva,…), posunu (tj. zda má křížek, béčko nebo nic) a nakonec, zda je u noty tečka. Pomlku určuje její doba a ostatní značky lze identifikovat podle jména nebo vlastnosti.

[7] Vzhledem k použití Javy je možné použít libovolný znak z *Unicode specification*, která zahrnuje i české znaky

[8] Program KVSO je schopen porovnat noty, ale ve verzi, která je součástí této práce, plně nepodporuje zobrazení a zadávání v grafickém uživatelském rozhraní.



Obrázek 26 - Diagram balíčku musicnotes

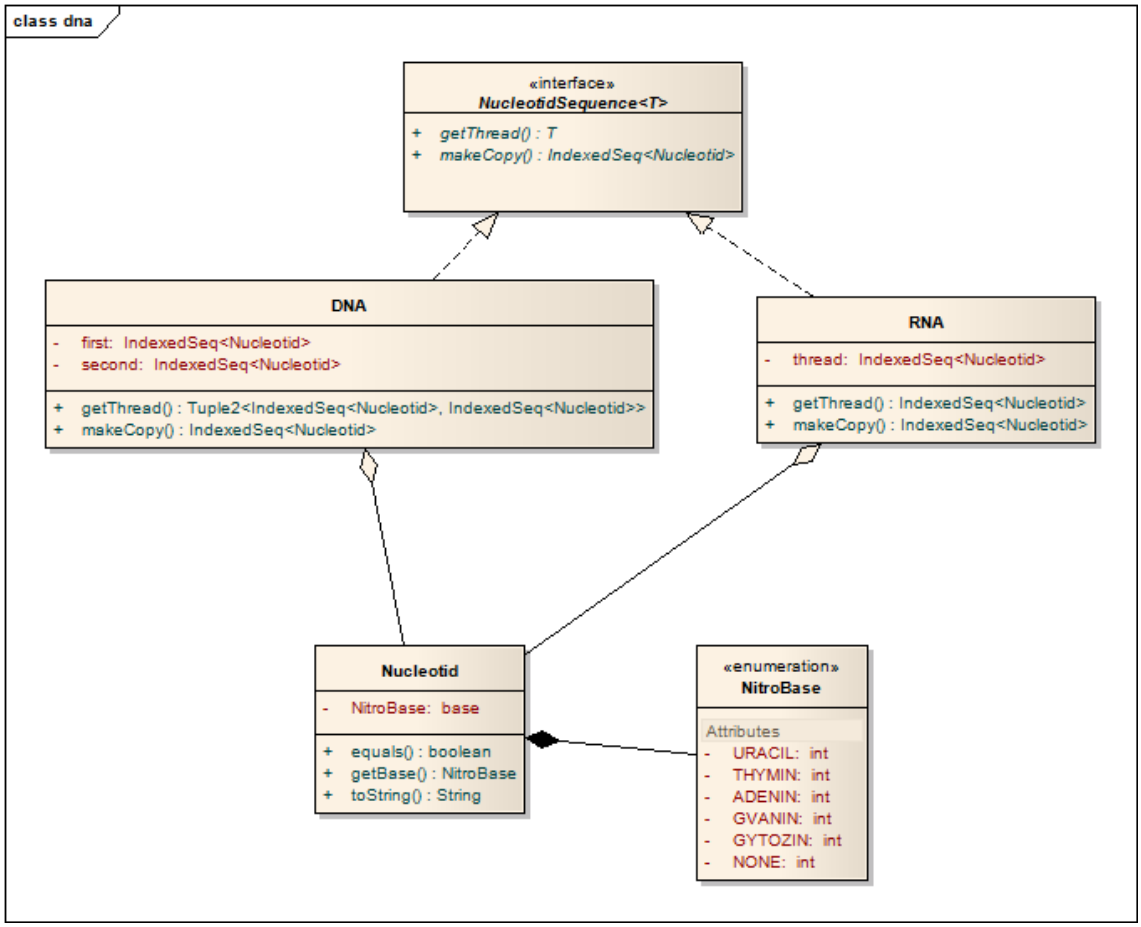
Zdroj: vlastní zpracování

Na obrázku 26 UML diagram tříd balíčku *musicnotes*. Balíček je koncipován tak, aby byl co nejvíce znovupoužitelný, proto obsahuje třídu *Partiture*, která je však pro samotné porovnávání řetězců not nedůležitá. V tomto směru je velmi důležité rozhraní *Mark*, které reprezentuje notové značky na notové osnově. Při porovnávání se používá řetězec implementací rozhraní *Mark*. Pro správnou funkci balíčku *metrics* bylo nutné přepsat

zděděnou metodu *equals()*, která zjišťuje, zda jsou dva objekty stejné. Aby bylo možné kvalitně vypisovat a ukládat byla též předefinována metoda *toString()*.

7.3.2 Nukleové báze

Reprezentace nukleových bází je v balíčku *dnarna*. Stejně jako u předešlých je zde použit *io.vavr*. Balíček *dnarna* vychází z následující myšlenky: Jsou dva druhy genetických řetězců, DNA a RNA. Obě obsahují jeden nebo více řetězců nukleotidů, které se liší dusíkatou bází (A, U, T, C, G).



Obrázek 27 - Diagram balíčku dnarna

Zdroj: vlastní zpracování

Stejně jako balíček *musicnotes* i balíček *dnarna* byl programován co nejobecněji, jak je vidět na obrázku 27. Rozhraní *NucleotidSequence* a jeho implementace umožňují práci s genetickým řetězcem jako celkem, ale pro samotné porovnávání se používá řetězec instancí třídy *Nucleotid*. Obdobně jako u *Mark* zde bylo nutné předefinovat metodu *equals()* a *toString()*.

8 Shrnutí výsledků

8.1 Srovnání výsledků *metrics* a *similarity*

V předchozí kapitole byla ukázána část výsledků testů balíčku *metrics*. Stejná data byla předána i balíčku *similarity* pro otestování. U Levenštejnovi i Hammingovi vzdálenosti byly získány stejné výsledky. Jaroovu vzdálenost balíček *similarity* neobsahuje, tudíž není co srovnávat. V případě Jaro-Winklerovi vzdálenosti se objevily tři rozdíly, viz tabulka 11. Pro ověření bylo provedeno detailní ruční přepočítání.

Tabulka 11 – Rozdíly ve výsledcích metrics a similarity

Vstup		Výsledek z metrics	Výsledek z similarity
koolipan	kopllisa	0,80	0,82
není	nie je	0,65	0,61
nejnezpravděpodo bňovávatelnějšího	nenespravdjepodo bnovavátelnějšího	0,90	0,84

Zdroj: vlastní zpracování

Nejprve byl proveden výpočet pro dvojici „koolipan“ a „kopllisa“. Délka obou řetězců je osm, rozsah výpočtu je tedy $r = max(8, 8)/2 - 1 = 3$. Matice pro tyto řetězce je na obrázku 28. Červeně jsou označena čísla, které jsou zkoumány při hledání použitých operací. Mohlo by se zdát, že počet všech shod $c = 7$ a $t = 0$, ale dvě nelze započítat, protože jsou ve sloupci, respektive v řádku, s jinou. Tudíž $c = 5$. Při použití standartních hodnot $W_1 = W_2 = W_t = ⅓$ je Jaroova vzdálenost $d_J(koolipan, kopllisa) = ⅓ \times 5/8 + ⅓ \times 5/8 + ⅓ \times (5-0)/5 = 0,75$. Kromě Jaroovi vzdálenosti je pro výpočet Jaro-Winklerovi vzdálenosti potřeba počet počátečních shodných znaků, což jsou dva („ko“). Jaro-Winklerova vzdálenost je $d_{JW}(koolipan, kopllisa) = 0,75 + 2 \times 0,1 \times (1 - 0,75) = 0,8$.

Pro úplnost byl proveden výpočet i pro hodnoty $c = 7$, $t = 0$. Jaroova vzdálenost je $d_J(koolipan, kopllisa) = ⅓ \times 7/8 + ⅓ \times 7/8 + ⅓ \times (7-0)/7 = 0,92$. Následně pak Jaro-Winklerova $d_{JW}(koolipan, kopllisa) = 0,92 + 2 \times 0,1 \times (1 - 0,92) = 0,94$. Není tedy zřejmé, odkud pochází rozdíl dvou setin mezi výsledkem *metrics* a *similarity*.

X	K	O	P	L	L	I	S	A
K	1	0	0	-	-	-	-	-
O	0	1	0	0	-	-	-	-
O	0	1	0	0	0	-	-	-
L	-	0	0	1	1	0	-	-
I	-	-	0	0	0	1	0	-
P	-	-	-	0	0	0	0	0
A	-	-	-	-	0	0	0	1
N	-	-	-	-	-	0	0	0

Obrázek 28 – Matice Jaro-Winklerovi vzdálenosti pro „koolipan“ a „kopllisa“

Zdroj: vlastní zpracování

Následuje dvojice „není“ a „nie je“. Rozsah výpočtu $r = max(|není|, |nie je|)/2 – 1 = max(4, 6)/2 – 1 = 2$. Vypočítaná matice je na obrázku 29, červeně jsou vyznačena čísla, kde se hledají provedené operace. Z matice je zřejmé, že $c = 2$ a $t = 0$. Proto Jaroova vzdálenost je pro standartní hodnoty $W_1 = W_2 = W_t = ⅓$: $d_J(není, nie je)=⅓ \times 2/4 + ⅓ \times 2/6 + ⅓ \times (2-0)/2 = 0,61$. Počet počátečních shodných znaků je jedna („n“). Nyní lze vypočítat Jaro-Winklerovu vzdálenost $d_{JW}(není, nie je) = 0,61 + 1 \times 0,1 \times (1 – 0,61) = 0,648$.

Podle ručního přepočítání je výsledek balíčku *metrics* správný. Zdá se, že výsledek balíčku *similarity* je samotná Jaroova vzdálenost. Pravděpodobně zde nebyl započítán první shodný znak.

X	N	I	E		J	E
N	1	0	-	-	-	-
E	0	0	1	-	-	-
N	-	0	0	0	-	-
í	-	-	0	0	0	-

Obrázek 29 - Matice Jaro-Winklerovi vzdálenosti pro „není“ a „nie je“

Zdroj: vlastní zpracování

Třetí problematickou dvojicí je „nejnezpravděpodobňovávatelnějšího“ a „nenesprav-djepodobnovavátelnějšího“. Rozsah výpočtu je $r = max(33, 33)/2 – 1 = 15,5$. Na obrázku 30 je matice pro tato dvě slova. Červeně jsou opět vyznačena čísla, kde se hledají shody a transpozice. Odtud tedy $c = 27$ a $t = 0$. Při standartních hodnotě W_1, W_2 a W_t, je Jaroova vzdálenost $⅓ \times 27/33 + ⅓ \times 27/33 + ⅓ \times (27-0)/27 = 0,879$. Na začátku obou

zkoumaných řetězců jsou dva stejné znaky („ne“). Dosazením se získá Jaro-Winklerova vzdálenost $d_{JW} = 0,879 + 2 \times 0,1 \times (1 - 0,879) = 0,9$.

I v tomto případě se po ručním spočítání zdá výsledek *metrics* správný. Rozdíl mezi *metrics* a *similarity* zřejmě pramení z různých hodnot *c* a *t*. Vzhledem k velikosti vstupních řetězců, absenci názoru třetí strany (např. nějaký kalkulátor) je obtížné rozhodnout, který výsledek je správný.

X	N	E	N	E	S	P	R	A	V	D	J	E	P	O	D	O	B	N	O	V	A	V	Á	T	E	L	N	Ě	J	Š	Í	H	O
N	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
E	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
J	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
N	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
E	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-
P	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-
R	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-
A	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-	-	-	-	-	-	-	-	-
V	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	-	-	-	-	-	-	-	-
D	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-
Ě	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-
P	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-
O	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	-	-	-	-
D	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-
O	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-	-
B	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-
Ň	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O	-	-	-	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V	-	-	-	-	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
Á	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
V	-	-	-	-	-	-	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
A	-	-	-	-	-	-	-	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
T	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
E	-	-	-	-	-	-	-	-	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
L	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
N	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Ě	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
J	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Š	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Í	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
H	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
O	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Obrázek 30 - Matice Jaro-Winklerovi vzdálenosti pro třetí spornou dvojici

Zdroj: vlastní zpracování

8.2 Srovnání *metrics* a *similarity* se vzorovými daty

V [7], kde je kromě jiného popsána Jaro-Winklerova vzdálenost, je též tabulka ukázkových dat právě pro Jaro-Winklerovu vzdálenost. I tato data byla použita jako vstup pro *metrics* i pro *similarity*. Všechny výsledky jsou opět uvedeny v Příloze 3. Po zaokrouhlení na dvě desetinná místa a při toleranci jedné setiny, jsou výsledky *metrics* a *similarity* shodné se vzorovými daty, kromě tří případů, viz tabulka 12.

Tabulka 12 - Rozdílné výsledky mezi vzorovými daty, metrics a similarity

	První řetězec	Druhý řetězec	Vzorový výsledek	Výsledek *metrics*	Výsledek *similarity*
#1	dixon	dickson	0,85	0,83	0,83
#2	yvette	yevett	0,91	0,95	0,90
#3	dwayne	duane	0,86	0,84	0,84

Zdroj: vlastní zpracování

U první a třetí dvojice řetězců je výsledek *metrics* a *similarity* stejný a od vzorových dat se liší o dvě setiny. V případě druhé dvojice se vzorová data a výsledek *similarity* liší o jednu setinu, ale vzorová data a *metrics* o čtyři setiny. Protože je rozdíl tak velký, bylo provedeno ruční přepočítání.

Matice pro tuto zkoumanou dvojici („yvette“ a „yevett“) je na obrázku 31. Rozsah výpočtu je $r = max(6, 6)/2 - 1 = 2$. Z matice plyne $c = 6$ a $t = 1$. Při použití standartních hodnot pro koeficienty vychází Jaroova vzdálenost $d_J(yvette, yevett) = ⅓ \times 6/6 + ⅓ \times 6/6 + ⅓ \times (6\text{-}1)/6 = 0{,}94$. Na začátku řetězců je jeden shodný znak, tudíž Jaro-Winklerova vzdálenost vychází $d_{JW}(yvette, yevett) = 0{,}94 + 1 \times 0{,}1 \times (1 - 0{,}94) = 0{,}95$.

Vzhledem k tomu, že v [7] není ukázán výpočet jednotlivých vzdáleností řetězců, je obtížné určit, co zapříčinilo takový rozdíl. Nezdá se, že by se jednalo o rozdíl způsobený zaokrouhlením.

X	Y	E	V	E	T	T
Y	1	0	-	-	-	-
V	0	0	1	-	-	-
E	-	1	0	1	-	-
T	-	-	0	0	1	-
T	-	-	-	0	1	1
E	-	-	-	-	0	0

Obrázek 31 - Matice Jaro-Winklerovi vzdálenosti pro „yvette“ a „yevett“

Zdroj: vlastní zpracování

8.3 Srovnání výpočetního času

V balíčku *metrics* byl každý druh vzdálenosti řetězců vytvořen dvakrát – jednou počítá pouze vzdálenost a ve druhé verzi poskytuje další informace. Proto by měl teoreticky trvat výpočet vzdálenosti bez dalších informací kratší dobu než výpočet s dalšími informacemi. Jak je vidět z tabulek 12 až 15, tak skutečně je výpočet bez detailu rychlejší. U Levenštejnovi a Hammingovi vzdálenosti (tabulka 13 a 14) zhruba dvojnásobně, u Jaroovi a Jaro-Winklerovi vzdálenosti (tabulka 15 a 16) je výpočet bez detailu rychlejší v průměru přibližně o jednu třetinu.

Tabulka 13 - Výpočetní čas Levenštejnovi vzdálenosti

	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
S informacemi	885 036	13 060 180	146 178
Bez informací	443 187	9 181 824	40 089

Zdroj: vlastní zpracování

Tabulka 14 - Výpočetní čas Hammingovi vzdálenosti

	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
S informacemi	221 531	4 040 179	33 245
Bez informací	116 728	575 911	31 288

Zdroj: vlastní zpracování

Tabulka 15 - Výpočetní čas Jaroovi vzdálenosti

	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
S informacemi	750 160	13 596 491	153 022
Bez informací	533 353	7 569 468	91 422

Zdroj: vlastní zpracování

Tabulka 16 - Výpočetní čas Jaro-Winklerovi vzdálenosti

	Průměrný čas (ns)	Maximální čas (ns)	Minimální čas (ns)
S informacemi	672 192	13 492 358	153 511
Bez informací	498 801	13 394 579	41 067

Zdroj: vlastní zpracování

9 Závěry a doporučení

Tato práce se snažila nalézt odpověď na otázku, jak lze porovnávat řetězce pomocí vzdálenosti řetězců. Nejprve tedy byla definována vzdálenost řetězců jako počet operací potřebných k převedení jednoho řetězce na druhý. Byly představeny čtyři způsoby výpočtu vzdálenosti řetězců – Levenštejnova, Hammingova, Jaroova a Jaro-Winklerova vzdálenost. Díky rešerši vědeckých článků byly nalezeny různé způsoby využití z různých odvětví lidského vědění, např. porovnávání jazyků a zjišťování jejich příbuznosti, porovnávání genetické informace nebo práce s matematickými stromy.

Průzkum aktuálních webových aplikací a počítačových programů pro porovnávání řetězců ukázal jejich nedostatek. Nalezené nástroje pak počítaly téměř výhradně jen Levenštejnovu vzdálenost. Na druhou stranu, knihovny a balíčky v různých programátorských jazycích umožňují počítat více druhů vzdáleností řetězců. Všechny nalezené nástroje počítaly vzdálenost řetězců jen pro slova, případně pro vektory (řetězec číslic). Následně byl popsán nový nástroj, který umožňuje počítat více různých vzdáleností řetězců, a to nejen pro řetězce znaků.

Popsané vzdálenosti řetězců nejsou všechny, které existují. V průběhu výzkumu bylo nalezeno více druhů vzdáleností řetězců, z nichž některé jsou v této práci zmíněné, leč nepopsané. Bohužel nebyl prostor je dostatečně prostudovat. Vzhledem k množství druhů vzdáleností řetězců se očekávalo, že bude k dispozici více webových služeb počítající vzdálenost řetězců. Jak ale bylo ukázáno, většina webových stránek počítá pouze Levenštejnovu vzdálenost.

Problematiku podobnosti řetězců by bylo možné zkoumat z hlediska porovnávání jednotlivých vzdáleností řetězců. Porovnávání jejich rychlosti, přesnosti a užitkové hodnoty. Též by bylo možné zaměřit se na jiný způsob, jak porovnávat řetězce než je vzdálenost řetězců.

Toto téma otvírá nové příležitosti k prozkoumání. Vzhledem k tomu, že se vzdálenost řetězců nejčastěji používá pro slova, je zřejmé napojení na zkoumání jazyků. To se nevztahuje pouze na jejich porovnávání a zjišťování, zda jsou příbuzné. Pomocí již zmíněné podmíněné entropie lze zkoumat, jak si budou mluvčí rozumět. Též je tu využití ke zkoumání jen jednoho jazyka, např. zkoumání fonetické podobnosti, viz kapitola 6.2.3, obrázek 20. Zde jsou PHP funkce, které porovnají anglickou zvukovou shodu. V rámci dalšího výzkumu by bylo možné vytvořit podobné funkce pro češtinu.

10 Seznam obrázků

11 Seznam tabulek

12 Seznam použité literatury

[1] NAVARRO, Gonzalo. A guided tour to approximate string matching. *ACM Computing Surveys* [online]. **33**(1), 31-88 [cit. 2017-05-01]. DOI: 10.1145/375360.375365. ISSN 03600300. Dostupné z: https://www.dcc.uchile.cl/TR/1999/TR_DCC-1999-005.pdf

[2] HAMMING, R. W. Error Detecting and Error Correcting Codes. Bell System Technical Journal [online]. 1950, 29(2), 147-160 [cit. 2017-07-22]. DOI: 10.1002/j.1538-7305.1950.tb00463.x. ISSN 00058580. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6772729

[3] MCKENZIE, Grant D. How to Calculate Hamming Distance. In: *Classroom* [online]. Leaf Group Ltd. / Leaf Group Education [cit. 2017-06-16]. Dostupné z: http://classroom.synonym.com/calculate-hamming-distance-2656.html

[4] LEVENSHTEIN, Vladimir I. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*. 1966, **10**(8), 707-710

[5] GILLELAND, Michael. Levenshtein Distance, in Three Flavors. *University of Pittsburgh* [online]. Pittsburgh [cit. 2017-05-01]. Dostupné z: http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm

[6] WINKLER, William E. Overview of Record Linkage and Current Research Directions. *U.S. Bureau of the Census: Statistical Research Division Report* [online]. 2016 [cit. 2017-11-14]. Dostupné z: http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf

[7] WINKLER, William E. a Yves THIBAUDEAU. An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 U.S. Census. *U.S. Bureau of the Census: Statistical Research Division Report* [online]. [cit. 2018-04-14]. Dostupné z: http://www.census.gov/srd/papers/pdf/rr91-9.pdf

[8] COHEN, William W., Pradeep RAVIKUMAR a Stephen E. FIENBERG. A Comparison of String Metrics for Matching Names and Records. *Carnegie Mellon University: School of Computer Science* [online]. 2003 [cit. 2017-11-14]. Dostupné z: https://www.cs.cmu.edu/~wcohen/postscript/kdd-2003-match-ws.pdf

[9] ScienceDirect [online]. [cit. 2017-12-27]. Dostupné z: https://www.sciencedirect.com/

[10] DALLA PREDA, Mila a Vanessa VIDALI. Abstract Similarity Analysis. Electronic Notes in Theoretical Computer Science [online]. 2017, 331, 87-99 [cit. 2017-12-17]. DOI: 10.1016/j.entcs.2017.02.006. ISSN 15710661. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1571066117300087

[11] FRANZOI, Laura a Andrea SGARRO. Linguistic classification: T-norms, fuzzy distances and fuzzy distinguishabilities. Procedia Computer Science [online]. 2017, 112, 1168-1177 [cit. 2017-12-17]. DOI: 10.1016/j.procs.2017.08.163. ISSN 18770509. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S187705091731520X

[12] MCVICAR, Matt, Benjamin SACH, Cédric MESNAGE, Jefrey LIJFFIJT, Eirini SPYROPOULOU a Tijl DE BIE. SuMoTED: An intuitive edit distance between rooted unordered uniquely-labelled trees. Pattern Recognition Letters [online]. 2016, 79, 52-59 [cit. 2017-12-17]. DOI: 10.1016/j.patrec.2016.04.012. ISSN 01678655. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0167865516300642

[13] AZIZ, Md Momin Al, Dima ALHADIDI a Noman MOHAMMED. Secure approximation of edit distance on genomic data. BMC Medical Genomics [online]. 2017, 10(S2), - [cit. 2017-12-17]. DOI: 10.1186/s12920-017-0279-9. ISSN 1755-8794. Dostupné z: http://bmcmedgenomics.biomedcentral.com/articles/10.1186/s12920-017-0279-9

[14] AYAD, Lorraine A. K., Carl BARTON a Solon P. PISSIS. A faster and more accurate heuristic for cyclic edit distance computation. Pattern Recognition Letters [online]. 2017, 88, 81-87 [cit. 2017-12-17]. DOI: 10.1016/j.patrec.2017.01.018. ISSN 01678655. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0167865517300259

[15] ROJAS-GALEANO, Sergio. On Obstructing Obscenity Obfuscation. ACM Transactions on the Web [online]. 2017, 11(2), 1-24 [cit. 2017-12-17]. DOI: 10.1145/3032963. ISSN 15591131. Dostupné z: http://dl.acm.org/citation.cfm?doid=3079924.3032963

[16] NAGATA, Ryo, Hiroya TAKAMURA a Graham NEUBIG. Adaptive Spelling Error Correction Models for Learner English. Procedia Computer Science [online]. 2017, 112, 474-483 [cit. 2017-12-17]. DOI: 10.1016/j.procs.2017.08.065. ISSN 18770509. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1877050917314096

[17] Levenshtein Distance. *Online calculator catalog* [online]. Russia: Planetcalc [cit. 2017-06-16]. Dostupné z: https://planetcalc.com/1721/

[18] Edit Distance Calculator. *Edward Holsinger* [online]. [cit. 2017-06-16]. Dostupné z: http://www.ripelacunae.net/projects/levenshtein/

[19] Levenshtein Distance Calculator. *Leo Jiang* [online]. [cit. 2017-06-16]. Dostupné z: http://leojiang.com/experiments/levenshtein/

[20] Levenshtein demo [online]. [cit. 2017-06-16]. Dostupné z: http://www.let.rug.nl/kleiweg/lev/

[21] Hamming distance calculator. *Calculator.vhex.net* [online]. [cit. 2017-06-16]. Dostupné z: http://calculator.vhex.net/calculator/distance/hamming-distance

[22] Package org.apache.commons.text.similarity. *Apache Commons* [online]. The Apache Software Foundation, 2017 [cit. 2017-06-15]. Dostupné z: https://commons.apache.org/sandbox/commons-text/apidocs/org/apache/commons/text/similarity/package-summary.html

[23] DevDocs [online]. Thibaut Courouble and other contributors [cit. 2017-06-15]. Dostupné z: http://devdocs.io/c/

[24] DevDocs [online]. Thibaut Courouble and other contributors [cit. 2017-06-15]. Dostupné z: http://devdocs.io/cpp/

[25] Microsoft Docs - C# [online]. Microsoft 2017, 2017 [cit. 2017-06-15]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/csharp

[26] SimMetrics. SourceForge [online]. Slashdot Media, 2017 [cit. 2017-07-06]. Dostupné z: https://sourceforge.net/projects/simmetrics/

[27] *PHP - Filesystem Functions* [online]. The PHP Group [cit. 2017-06-15]. Dostupné z: http://php.net/manual/en/ref.filesystem.php

[28] PyPI - the Python Package Index [online]. Python Software Foundation [cit. 2017-06-15]. Dostupné z: https://pypi.python.org/pypi

[29] Stringdist. RDocumentation [online]. [cit. 2017-07-06]. Dostupné z: https://www.rdocumentation.org/packages/stringdist/versions/0.9.4.4

[30] MOBERG, Jens, GOOSKENS, Charlotte, NERBONNE, John, VAILLETTE, Nathan, 2007. Conditional Entropy Measures Intelligibility among Related Languages. In: Proceedings of the 17th Meeting of Computational Linguistics in the Netherlands. Ultrecht: LOT, s. 51-67. ISBN: 978-90-78328-41-4

[31] BÖHMOVÁ, Z, A GRÜNFELDOVÁ a A SARAUER. Klavírní škola pro začátečníky. 34. vyd., 2. vyd. v Editio Bärenreiter Praha. Praha: Editio Bärenreiter Praha, 2002. ISBN M-2601-0158-6.

13 Přílohy

1) Seznam elektronických příloh
2) UML diagram tříd balíčku *metrics*
3) Kompletní výsledky testů balíčku *metrics*
4) Výpočet podmíněné entropie

13.1 Seznam elektronických příloh

1. Složka *kvso*

Složka se zdrojovým kódem balíčku *metrics* a programem KVSO.

2. Složka *KVSO-program*

Složka obsahující program KVSO. Program je spustitelný pomocí souboru *Kalkulátor vzdáleností sekvencí objeků*. Součástí této složky je i složka *ukazkove_soubory*, ve které jsou vzorové soubory.

a) *character.csv*

Ukázkový soubor pro znaky

b) *musicnote.xml*

Ukázkový soubor pro noty

c) *dna-rna.txt*

Ukázkový soubor pro dna/rna

3. Soubor *KVSO-1.1.1.jar*

Spustitelný soubor, obsahuje program KVSO.

4. Soubor *metrics-1.1.jar*

Javovská knihovna obsahující balíček metrics.

5. Složka *realobject*

Složka se zdrojovým kódem balíčků s počítačovou reprezentací not a DNA

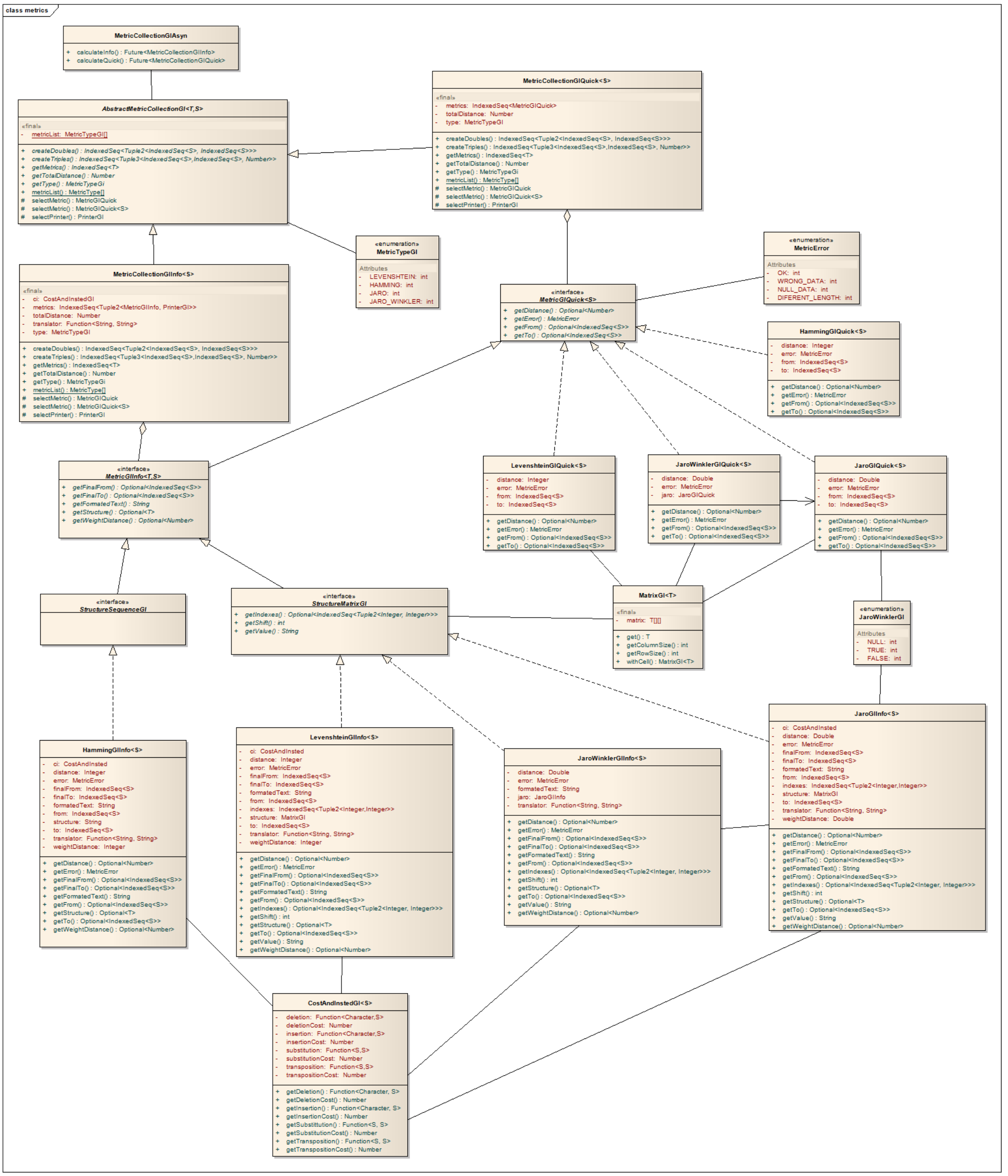
6. Soubor *realobject-1.1.jar*

Javovská knihovna s počítačovou reprezentací not a DNA.

7. Soubor *metrics.png*

UML model tříd balíčku *metrics*.

13.2 UML diagram tříd balíčku *metrics*



Obrázek 32 - UML diagram balíčku metrics

Zdroj: vlastní zpracování

13.3 Kompletní výsledky testů balíčku *metrics*

V této příloze jsou uvedeny všechny výsledky, které byly získány během testování balíčku *metrics*. Výsledky nejsou zaokrouhlovány.

Výpočet vzdálenosti řetězců s dalšími informacemi je schopen na zkoumaných řetězcích zobrazit, jak lze převést jeden na druhý. V případě, že byla provedena náhrada, je písmeno psáno velkým písmem. Stejně je označeno prohození. Vložení je označeno „-“. Smazání zde není značeno, jedná se o vložení do druhého řetězce.

Tabulka 17 - Testovací data

	První řetězec	**Druhý řetězec**
#1	houska	housle
#2	aahoj	ahojky
#3	kolo	oko
#4	koolipan	kopllisa
#5	koplinn	kpollim
#6	slovo	slovo
#7	slovo	
#8		slovo
#9	slovo	l
#10	slovo	sl
#11	slova	a
#12	symboly	*$|€đđ#&@
#13	okno	window
#14	abbcb	abcab
#15	není	nie je
#16	nejnezpravděpodobňovávatelnějšího	nenespravdjepodobnovavátelnějšího

Zdroj: vlastní zpracování

13.3.1 Levenštejnova vzdálenost

Pouze u Levenštejnovi vzdálenosti poskytuje balíček *similarity* informace o počtu operací.

Tabulka 18 - Řetězce s vyznačenými operacemi - Levenštejnova vzdálenost

	Upravený první řetězec	Upravený druhý řetězec
#1	housKA	housLE
#2	aahoj--	a-hojky
#3	koLo	-oKo
#4	koOl-iPan	koPlliSa-
#5	kop-l-iNn	k-polliM-
#6	slovo	slovo
#7	slovo	-----
#8	-----	slovo
#9	slovo	-l---
#10	slovo	sl---
#11	slova	----a
#12	SYMBOLY---	*$|€ĐĐ#&@
#13	OKn-o-	WIndow
#14	abbc-b	ab-cab
#15	n-eNÍ-	nie Je
#16	nejneZpravdě-pODo-bňo-VÁva--telnějšího	ne-neSpravd-j-EPod--obNOvavátelnějšího

Zdroj: vlastní zpracování

Tabulka 19 – Výsledky Levenštejnovi vzdálenosti

	Výpočet s detailem	Výpočet bez detailu	Detailní výpočet balíčku similarity
#1	2	2	Vzdálenost: 2, Vložení: 0, Smazání: 0, Náhrada: 2
#2	3	3	Vzdálenost: 3, Vložení: 2, Smazání: 1, Náhrada: 0
#3	2	2	Vzdálenost: 2, Vložení: 0, Smazání: 1, Náhrada: 1
#4	4	4	Vzdálenost: 4, Vložení: 1, Smazání: 1, Náhrada: 2
#5	4	4	Vzdálenost: 4, Vložení: 1, Smazání: 1, Náhrada: 2
#6	0	0	Vzdálenost: 0, Vložení: 0, Smazání: 0, Náhrada: 0
#7	5	5	Vzdálenost: 5, Vložení: 0, Smazání: 5, Náhrada: 0
#8	5	5	Vzdálenost: 5, Vložení: 5, Smazání: 0, Náhrada: 0
#9	4	4	Vzdálenost: 4, Vložení: 0, Smazání: 4, Náhrada: 0
#10	3	3	Vzdálenost: 3, Vložení: 0, Smazání: 3, Náhrada: 0
#11	4	4	Vzdálenost: 4, Vložení: 0, Smazání: 4, Náhrada: 0
#12	10	10	Vzdálenost: 10, Vložení: 3, Smazání: 0, Náhrada: 7
#13	4	4	Vzdálenost: 4, Vložení: 2, Smazání: 0, Náhrada: 2
#14	2	2	Vzdálenost: 2, Vložení: 0, Smazání: 0, Náhrada: 2
#15	4	4	Vzdálenost: 4, Vložení: 2, Smazání: 0, Náhrada: 2
#16	7	7	Vzdálenost: 7, Vložení: 1, Smazání: 1, Náhrada: 5

Zdroj: vlastní zpracování

Tabulka 20 - Doba výpočtu Levenštejnovi vzdálenosti s detailem

	Průměr	Max	Min
#1	590 675,6	2 659 067	249 822
#2	606 222,4	2 488 445	206 311
#3	444 840,0	1 117 111	176 489
#4	950 106,8	2 402 889	283 066
#5	921 360,1	4 108 623	277 200
#6	318 706,8	820 356	166 222
#7	1 127 427,0	6 626 890	172 578
#8	200 884,7	415 556	146 178
#9	282 675,5	556 356	180 889
#10	585 933,3	3 331 289	167 689
#11	485 515,7	1 359 600	207 289
#12	1 532 129,0	5 918 001	382 800
#13	777 088,9	3 713 112	218 534
#14	392 675,7	1 210 489	199 956
#15	399 813,5	1 041 823	221 956
#16	4 544 516,0	13 060 180	1 860 223

Zdroj: vlastní zpracování

Tabulka 21 - Doba výpočtu Levenštejnovi vzdálenosti bez detailu

	Průměr	Max	Min
#1	269 182,2	1 231 022	86 044
#2	255 933,5	884 401	65 022
#3	341 244,4	2 159 422	53 289
#4	384 560,1	1 101 956	87 022
#5	322 226,8	989 511	77 245
#6	199 368,8	568 089	62 088
#7	621 671,2	2 720 667	51 333
#8	111 857,7	269 377	40 089
#9	149 551,0	373 511	51 822
#10	166 466,6	447 822	54 266
#11	149 355,4	377 911	51 822
#12	339 044,7	1 049 156	88 000
#13	215 844,5	747 023	59 644
#14	184 262,3	501 112	61 600
#15	203 377,8	490 845	59 644
#16	3 177 045,0	9 181 824	587 645

Zdroj: vlastní zpracování

13.3.2 Hammingova vzdálenost

Buňky označené „xxx“ nešlo vypočítat kvůli rozdílné délce řetězců.

Tabulka 22 - Řetězce s vyznačenými operacemi - Hammingova vzdálenost

	Upravený první řetězec	**Upravený druhý řetězec**
#1	housKA	housLE
#2	xxx	xxx
#3	xxx	xxx
#4	koOlIPAN	koPlLISA
#5	kOPlINN	kPOlLIM
#6	slovo	slovo
#7	xxx	xxx
#8	xxx	xxx
#9	xxx	xxx
#10	xxx	xxx
#11	xxx	xxx
#12	xxx	xxx
#13	xxx	xxx
#14	abBCb	abCAb
#15	xxx	xxx
#16	neJNEZPRAVDĚpodobŇovÁvAtelněj-šího	neNESPRAVDJEpodobNovAvÁtelněj-šího

Zdroj: vlastní zpracování

Tabulka 23 – Výsledky Hammingovi vzdálenosti

	Výpočet s detailem	Výpočet bez detailu	Výpočet balíčku similarity
#1	2	2	2
#2	xxx	xxx	xxx
#3	xxx	xxx	xxx
#4	5	5	5
#5	5	5	5
#6	0	0	0
#7	xxx	xxx	xxx
#8	xxx	xxx	xxx
#9	xxx	xxx	xxx
#10	xxx	xxx	xxx
#11	xxx	xxx	xxx
#12	xxx	xxx	xxx
#13	xxx	xxx	xxx
#14	2	2	2
#15	xxx	xxx	xxx
#16	13	13	13

Zdroj: vlastní zpracování

Tabulka 24 - Doba výpočtu Hammingovi vzdálenosti s detailem

	Průměr	Max	Min
#1	270 306,5	435 600	131 022
#2	147 253,3	492 312	86 533
#3	128 480,0	539 734	36 178
#4	286 391,6	751 912	129 067
#5	308 244,3	779 777	97 288
#6	184 262,3	443 422	69 911
#7	162 066,9	653 156	33 734
#8	99 635,6	209 245	33 245
#9	110 342,2	224 400	86 533
#10	97 924,3	215 600	38 622
#11	94 404,3	231 244	38 133
#12	312 400,4	2 356 445	33 245
#13	83 893,4	141 778	33 245
#14	218 826,8	665 867	87 022
#15	124 568,9	486 444	33 733
#16	915 493,4	4 040 179	286 977

Zdroj: vlastní zpracování

Tabulka 25 - Doba výpočtu Hammingovi vzdálenosti bez detailu

	Průměr	Max	Min
#1	136 351,4	235 645	50 356
#2	101 591,1	255 200	31 778
#3	116 795,5	575 911	31 289
#4	176 928,7	438 044	41 066
#5	127 942,2	261 556	40 577
#6	118 066,7	246 400	37 645
#7	108 777,8	261 066	31 289
#8	93 377,9	213 645	31 289
#9	101 200,2	207 289	31 289
#10	78 760,0	144 711	31 288
#11	68 298,0	95 334	31 289
#12	107 213,2	286 978	31 288
#13	68 395,7	97 778	31 289
#14	126 475,6	250 311	40 578
#15	101 249,0	215 112	31 289
#16	236 231,1	377 422	63 066

Zdroj: vlastní zpracování

13.3.3 Jaroova vzdálenost

Balíček *similarity* nepočítá Jaroovu vzdálenost, tudíž není s čím srovnávat. Výsledné vzdálenosti byly zaokrouhleny na tři desetinná místa.

Tabulka 26 - Řetězce s vyznačenými operacemi - Jaroova vzdálenost

	Upravený první řetězec	Upravený druhý řetězec
#1	hous-k-a	housl-e-
#2	aahoj--	a-hojky
#3	KOlo	OK-o
#4	ko-ol-i-pan	kop-llis-a-
#5	kOPl-i-nn	kPOllim--
#6	slovo	slovo
#7	slovo	-----
#8	-----	slovo
#9	slovo	-l---
#10	slovo	sl---
#11	-slova	a-----
#12	-s-y-m-b-o-l-y---	*-$-\-|-€-đ-đ-#&@
#13	-o-kn-o-	w-i-ndow
#14	abbc-b	ab-cbb
#15	n-e-n-í-	nie -j-e
#16	nejne-zpravd--ěpodob-ňov-áv-atelnějšího	ne-nes-pravdee-podobn-ova-vá-telnějšího

Zdroj: vlastní zpracování

Tabulka 27 - Výsledky Jaroovi vzdálenosti

	Výpočet s detailem	Výpočet bez detailu
#1	0,778	0,778
#2	0,823	0,822
#3	0,810	0,810
#4	0,750	0,750
#5	0,743	0,743
#6	1,000	1,000
#7	0,000	0,000
#8	0,000	0,000
#9	0,733	0,733
#10	0,800	0,800
#11	0,000	0,000
#12	0,000	0,000
#13	0,611	0,611
#14	0,867	0,867
#15	0,611	0,611
#16	0,879	0,879

Zdroj: vlastní zpracování

Tabulka 28 - Doba výpočtu Jaroovi vzdálenosti s detailem

	Průměr	Max	Min
#1	565 253,5	2 371 600	263 022
#2	481 800,3	1 231 512	248 356
#3	340 902,4	747 023	175 511
#4	622 648,9	2 005 912	311 422
#5	501 062,5	1 307 289	243 956
#6	249 039,9	693 245	167 689
#7	443 813,2	2 614 089	166 222
#8	200 493,2	422 400	153 022
#9	428 364,6	1 564 933	184 311
#10	528 831,0	3 070 223	162 800
#11	259 844,6	525 067	180 400
#12	765 306,9	2 132 534	347 600
#13	639 271,1	2 932 845	238 578
#14	395 657,9	1 251 067	198 000
#15	300 324,5	631 156	215 600
#16	5 279 951,7	13 596 491	2 278 711

Zdroj: vlastní zpracování

Tabulka 29 - Doba výpočtu Jaroovi vzdálenosti bez detailu

	Průměr	Max	Min
#1	268 400,1	374 978	162 800
#2	387 200,0	949 911	165 244
#3	179 275,5	229 778	140 311
#4	393 115,6	920 578	224 889
#5	385 929,2	990 489	167 200
#6	218 826,6	479 111	119 778
#7	369 795,6	2 525 600	105 112
#8	122 858,0	220 000	91 422
#9	1 933 55,8	390 623	121 733
#10	1 699 37,8	340 756	108 533
#11	1 565 42,5	205 334	122 223
#12	658 875,8	1 730 667	231 245
#13	320 613,4	716 222	160 356
#14	344 422,2	696 667	150 578
#15	260 871,1	473 245	146 667
#16	4 103 636,3	7 569 468	1 496 489

Zdroj: vlastní zpracování

13.3.4 Jaro-Winklerova vzdálenost

Výsledné vzdálenosti byly zaokrouhleny na tři desetinná místa. Pro Jaro-Winklerovu vzdálenost byla v [7] nalezena ukázková data. Proto byla použita jako vstup pro balíček *metrics* i *similarity*.

Tabulka 30 - Řetězce s vyznačenými operacemi - Jaro-Winklerova vzdálenost

	Upravený první řetězec	**Upravený druhý řetězec**
#1	hous-k-a	housl-e-
#2	aahoj--	a-hojky
#3	KOlo	OK-o
#4	ko-ol-i-pan	kop-llis-a-
#5	kOPl-i-nn	kPOllim--
#6	slovo	slovo
#7	slovo	-----
#8	-----	slovo
#9	slovo	-l---
#10	slovo	sl---
#11	-slova	a-----
#12	-s-y-m-b-o-l-y---	*-$-\-|-€-đ-đ-#&@
#13	-o-kn-o-	w-i-ndow
#14	abbc-b	ab-cbb
#15	n-e-n-í-	nie -j-e
#16	nejne-zpravd--ěpodob-ňov-áv-atelnějšího	ne-nes-pravdee-podobn-ova-vá-telnějšího

Zdroj: vlastní zpracování

Tabulka 31 - Výsledky Jaro-Winklerovi vzdálenosti

	Výpočet s detailem	Výpočet bez detailu	Výpočet balíčku similarity
#1	0,867	0,867	0,867
#2	0,840	0,840	0,840
#3	0,810	0,810	0,810
#4	0,800	0,800	0,822
#5	0,769	0,769	0,769
#6	1,000	1,000	1,000
#7	0,000	0,000	0,000
#8	0,000	0,000	0,000
#9	0,733	0,733	0,733
#10	0,840	0,840	0,840
#11	0,000	0,000	0,000
#12	0,000	0,000	0,000
#13	0,611	0,611	0,611
#14	0,893	0,893	0,893
#15	0,650	0,650	0,611
#16	0,903	0,903	0,839

Zdroj: vlastní zpracování

Tabulka 32 - Doba výpočtu Jaro-Winklerovi vzdálenosti s detailem

	Průměr	Max	Min
#1	780 853,5	3 597 734	275 733
#2	405 875,4	1 297 511	238 578
#3	320 857,8	745 067	194 578
#4	873 937,7	3 304 889	292 844
#5	454 715,4	1 416 311	241 022
#6	270 844,3	685 423	183 333
#7	447 137,7	2 656 622	165 733
#8	238 822,0	525 555	153 511
#9	256 128,7	578 844	177 467
#10	429 537,9	987 067	191 155
#11	293 773,4	559 289	178 934
#12	571 315,7	2 047 466	303 600
#13	394 924,4	780 755	220 000
#14	296 315,7	636 533	210 712
#15	309 955,7	723 555	218 533
#16	4 410 071,6	13 492 358	1 792 267

Zdroj: vlastní zpracování

Tabulka 33 - Doba výpočtu Jaro-Winklerovi vzdálenosti bez detailu

	Průměr	Max	Min
#1	426 115,6	2 307 556	125 156
#2	359 040,1	1 020 311	112 445
#3	284 386,6	818 400	88 000
#4	553 324,6	1 587 911	167 200
#5	395 560,1	1 013 956	114 400
#6	207 973,1	602 800	68 444
#7	101 591,0	174 534	52 311
#8	112 835,5	327 066	41 067
#9	172 724,5	414 578	67 466
#10	160 404,8	365 689	58 667
#11	177 711,2	440 000	66 000
#12	371 799,9	631 645	179 911
#13	345 057,8	838 444	110 978
#14	275 635,7	667 334	99 244
#15	222 102,3	502 578	98 756
#16	3 814 555,9	13 394 579	1 297 022

Zdroj: vlastní zpracování

V tabulce 34 jsou vzorová data pro Jaro-Winklerovu vzdálenost. Výsledky výpočtů jsou v tabulce 35. Protože vzorové výsledky jsou zaokrouhleny na čtyři desetinná místa, jsou stejně zaokrouhleny i výsledky balíčku *metrics* a *similarity*.

Tabulka 34 - Vzorová data pro Jaro-Winklerovu vzdálenost

	První slovo	**Druhé slovo**
§1	shackleford	shackelford
§2	cunningham	cunnigham
§3	campell	campbell
§4	nichleson	nichulson
§5	massey	massie
§6	abroms	abrams
§7	galloway	calloway
§8	lampley	campley
§9	dixon	dickson
§10	frederick	fredrick
§11	michele	michelle
§12	jesse	jessie
§13	marhta	martha
§14	jonathon	jonathan
§15	julies	juluis
§**16**	jeraldine	geraldine
§17	yvette	yevett
§18	tanya	tonya
§19	dwayne	duane

Zdroj: vlastní zpracování

Tabulka 35 - Porovnání výsledných vzdáleností se vzorovými daty

	Původní data	**Výsledek metrics**	**Výpočet similarity**
§1	0,9848	0,9818	0,9835
§2	0,9833	0,9800	0,9833
§3	0,9792	0,9750	0,9750
§4	0,9630	0,9551	0,9556
§5	0,9444	0,9333	0,9333
§6	0,9333	0,9222	0,9222
§7	0,9167	0,9167	0,9167
§8	0,9048	0,9048	0,9048
§9	0,8533	0,8324	0,8324
§10	0,9815	0,9778	0,9778
§11	0,9792	0,9750	0,9833
§12	0,9722	0,9667	0,9667
§13	0,9667	0,9611	0,9611
§14	0,9583	0,9500	0,9667
§15	0,9333	0,9222	0,9222
§**16**	0,9246	0,9259	0,9259
§17	0,9111	0,9500	0,9000
§18	0,8933	0,8800	0,8800
§19	0,8578	0,8400	0,8400

Zdroj: vlastní zpracování

13.4 Spočítání podmíněné entropie

Podmíněná entropie se počítá pomocí seznamu dvojic slov, přičemž první slovo je z prvního jazyka a druhé z druhého. Je nutné, aby obě slova byla stejně dlouhá. Právě zde se uplatní algoritmy z balíčku *metrics*, které jsou schopné upravit slova na stejnou délku. Založeno na [30].

Spočítat podmíněnou entropii lze pomocí následujícího vzorce:

$$H(X|Y) = -\sum_{x \in X, y \in Y} p(x,y) \log_2 p(x|y)$$

Obrázek 33 - Vzorec podmíněné entropie

Zdroj: [30]

13.4.1 Postup použitý v přiloženém programu

Vytvoří se seznam dvojic znaků a počet výskytů kažné z nich. Dvojice znaků se tvoří tak, že se k prvnímu znaku prvního slova první dvojice přiřadí první znak druhého slova první dvojice. Vytvoří se seznam znaků a jejich počet obsažených v prvních slovech. Totéž se provede pro druhá slova.

Pro každou dvojici znaků se vypočítá počet *výskytů_dvojice/celkový_počet_dvojic* × log_2*(počet_výskytů_dvojice/počet_výskytů_prvního_znaku_dvojice_v_prvním_jazyce)*. Tato čísla se sečtou a výsledek vynásobí *-1*. To je podmíněná entropie prvního jazyka.

Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Němec Ondřej	Dolní 429, Chotěboř	I1500396

TÉMA ČESKY:

Podobnost řetězců

TÉMA ANGLICKY:

Similarity of sequences

VEDOUCÍ PRÁCE:

Mgr. Jiří Haviger, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce
1. rešerše aktuálních článků na toto téma (WOS, SCOPUS, ...)
2. rešerše aktuálních nástrojů dostupných pro porovnání dvou řetězců
3. příprava vlastního nástroje

Osnova:
úvod
popis metrik užívaných pro porovnání dvou řetězců
popis dostupných nástrojů a knihoven
současné využití metrik pro porovnání řetězců
popis vlastního nástroje
diskuze
závěr

SEZNAM DOPORUČENÉ LITERATURY:

https://en.wikipedia.org/wiki/Levenshtein_distance
https://en.wikipedia.org/wiki/Edit_distance + bod 1

Podpis studenta: Ondřej Němec **Datum:** 10.10.2017

Podpis vedoucího práce: **Datum:** 10.10.2017

(c) IS/STAG , Portál - Podklad kvalifikační práce , I1500396 , 06.10.2017 00:06