Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Daniel Remiš

Vizualizace algoritmů pro vyhledávání v textech

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.

Studijní program: Informatika, Programování

2011

Na tomto místě bych chtěl poděkovat všem, kteří mě při psaní práce podporovali, především pak rodině za pochopení a zvládnutí stresových situací, které spolu se mnou zažili a protrpěli. Neméně bych pak chtěl poděkovat vedoucímu bakalářské práce, za pomoc a shovívavost při postupu tvorby.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Daniel Remiš

Obsah

Název práce: Vizualizace algoritmů pro vyhledávání v textech
Autor: Daniel Remiš
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.
e-mail vedoucího: Michal.Kopecky@mff.cuni.cz

Abstrakt: Práce se zabývá návrhem a implementací programu sloužícího k objasnění fungování algoritmů pro vyhledávání v textech pomocí grafického zobrazení jejich struktur. Součástí práce je i analýza některých zajímavých, již existujících, vizualizačních nástrojů, zaměřených na řešeních jiných oblastí.

Klíčová slova: vizualizace algoritmů, e-learning, vyhledávání v textech

Title: Visualization algorithms for searching through text
Author: Daniel Remiš
Department: Department of Software Engineering
Supervisor: RNDr. Michal Kopecký, Ph.D.
Supervisor's e-mail address: Michal.Kopecky@mff.cuni.cz

Abstract: The work deals with design and implementation of the program, which is used to clarify the operation of algorithms for searching through text using a graphical view of their structures. The work also includes some analyzis of already existing, interesting visualization tools designed to address other areas.

Keywords: algorithm visualization, e-learning, text retrieval

1 Úvod

1.1 Motivace

Při výuce řady předmětů je často problémem jak látku řádně objasnit posluchačům. Informatika v tomto směru není žádnou výjimkou, ba naopak. Algoritmy, které jsou určeny k řešení různorodých problémů, bývá při výkladu na základě abstraktní matematiky nezřídka složité pochopit bez jejich bližšího zkoumání. Specifickým příkladem může být vyhledávání v textech. V této oblasti se často využívají mnohorozměrné vektorové prostory a netriviální datové struktury. Grafické znázornění přináší možnost učinit si detailní představu o postupu výpočtů a případně i jejich úskalích. Obrázek mnohdy řekne více než tisíc slov. Tím více může k pochopení přispět interaktivní animace algoritmu krok za krokem, s možností vidět výsledek algoritmu pro různá vstupní data.

Digitalizace a vstup výpočetní techniky do běžného života s sebou přinesl velké množství oblastí pro její využití. Součástí většiny z nich je nutnost uchování dat a potřeba vyhledávání v nich. Algoritmy pro vyhledávání v textu se tak staly nutnou potřenou pro rozumné zpřístupnění většího množství informací.

1.2 Cíle práce

Cílem této práce je proto zhodnocení některých dostupných vizualizačních nástrojů, které se snaží o znázornění určitých algoritmů a dále vlastní návrh a implementace nástroje pro vizualizaci vybraných algoritmů zaměřených na vyhledávání v textu.

1.3 Obsah práce

Ve druhé kapitole je provedeno představení některých existujících vizualizačních implementací a nastíněno principiální fungování vyhledávačů, základní zpracování textu a tvorba indexu. Kapitola následující je věnována návrhu řešení zadané úlohy, včetně informací o použitých technologiích, výběru programovacího jazyka, návrhu aplikace a jejího grafického rozhraní.

Obsahem čtvrté kapitoly je programátorská dokumentace …

Pátá kapitola se týká uživatelské dokumentace…

V šesté kapitole se nachází závěr a zhodnocení výsledků, které se podařilo dosáhnout, dále se zde nachází i úvaha nad možnostmi dalšího rozvoje.

2 Analýza

2.1 Princip vyhledávačů

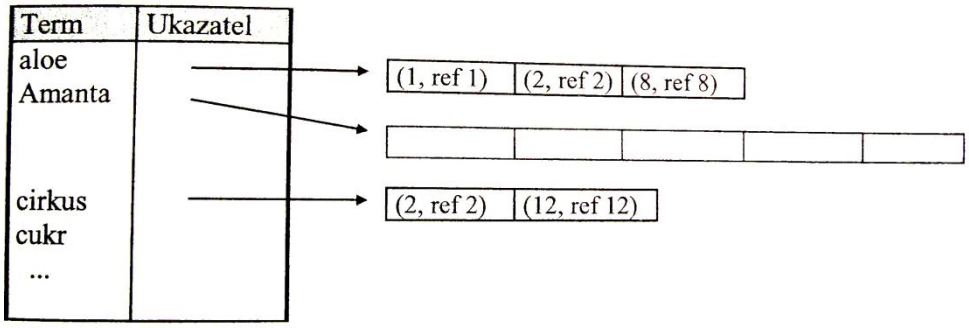
2.1.1Indexace

Pro indexaci textu je třeba nejdříve pomocí lexikální analýzy ze vstupní posloupnosti znaků vytvořit posloupnost slov, termů. Dle požadavků tak převedeme číslice, písmena a další interpunkční znamínka a znaky na slova. V běžně dostupných textech je možné se setkat s dvaceti až třiceti procenty frekventovaných slov [PSK05], která nebudeme chtít indexovat z důvodu jejich neunikátnosti. Lze k tomu využít slovník nevýznamových slov sestavený pro oblast indexovaných dat. Nejefektivnější selekce nevýznamových slov je při provádění lexikální analýzy, kdy je možné využít konečnosti nevýznamového slovníku a sestrojit pro selekci konečný automat. Dalším krokem převodu textu je lemmatizace, zde dochází k tvoření základních tvarů slov pomocí pravidel nebo vyhledávání v databázi. Hlavním nedostatkem lemmatizátorů je neschopnost jednoduše zpracovat mnohoznačnost některých slov. Jednoduchý lemmatizátor lze implementovat jako odstraňování koncovek a přípon. Z původního textu máme nyní soubor termů.

2.1.2Invertované soubory

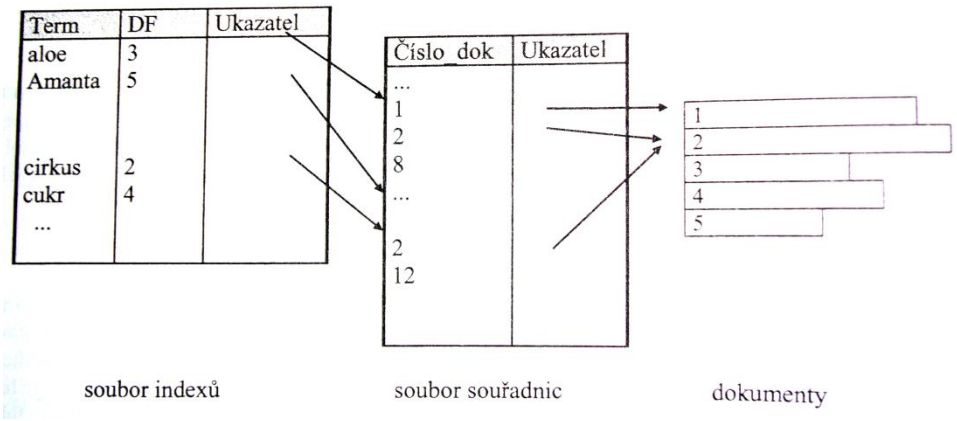
Z termů lze sestavit invertované soubory. Jedná se o setříděný seznam termů, kde každý z nich si udržuje vazbu na dokumenty, v kterých je obsažen. Základní

strukturou invertovaného souboru je seznam dvojic, tedy termu a ukazatele na seznam dokumentů a seznamu souřadnic (číslo dokumentu, ukazatel) [obr. 2.1].



obr. 2.1: struktura základního invertovaného souboru [PSK05]

Tato struktura může být obohacena o další informace. Je možné rozšířit informaci o termu, o jeho četnost v dokumentu (DF), vytvořit váhy termů, definovat synonyma. Další variantou je upřesnění pozice termu v dokumentu a jeho umístění v odstavci nebo větě [obr. 2.2].



obr. 2.2: struktura rozšířeného invertovaného souboru [PSK05]

2.1.3Booleovské DIS

Booleovský model je nejstarší a zároveň nejrozšířeněji používaným modelem. Jeho teoretické základy sahají až do padesátých let minulého století, přesto se i dnes stále používá v implementacích vyhledávačů na Internetu.

Základem modelu je reprezentace dokumentů v indexu pomocí množiny termů, které jej charakterizují. Druhou nedílnou součástí jsou dotazy sestavené z termů pomocí závorek, logických operátorů [tab. 2.1] a primitiv dávajících logický výraz. Logickým výrazem dotazu je např.: term1 AND NOT term2 OR term3

T1 AND T2	Ve výsledku budou dokumenty obsahující jak term T1, tak term T2. (konjunkce, logický součin)
T1 OR T2	Ve výsledku budou dokumenty obsahující buď term T1 nebo term T2 nebo oba současně. (disjunkce, logický součet)
T1 XOR T2	Ve výsledku budou dokumenty obsahující buď term T1 nebo term T2, nikoliv oba současně. (exkluzivní logický součet)
NOT T	Ve výsledku budou dokumenty neobsahující term T. (negace)

tab. 2.1: logické operátory [PSK05]

Standardní logické operátory umožňují pouze základní dotazy, proto se používá další rozšíření. Obohacení o tzv. proximitní operátory [tab. 2.2] umožňuje určení maximální vzdálenosti výskytu termů v textu.

T1 adj T2	Dokumenty, ve kterých je výskyt termu T1 následovaný termem T2.
T1 (n) words T2	Dokumenty, ve kterých je výskyt termu T1 následovaný termem T2 do vzdálenosti n slov.
T1 sentence T2	Dokumenty, ve kterých je výskyt termu T1 a T2 ve stejné větě.
T1 paragraph T2	Dokumenty, ve kterých je výskyt termu T1 a T2 ve stejném odstavci.

tab. 2.2: proximitní operátory [PSK05]

Dalším častým rozšířením *Booleovského modelu* bývá možnost tvorby masek pomocí regulárních výrazů. Znaky ^, ., *, +, a [] nabývají specifického významu [].

.	Znak tečky odpovídá libovolnému znaku.
*	Znak hvězdičky za symbolem odpovídá libovolnému počtu výskytů tohoto symbolu (včetně nulového). Např. ef* odpovídá termům e, ef, eff, efff atd.
+	Znak plus za symbolem odpovídá libovolnému počtu výskytů tohoto symbolu (vyjma nulového). Např. ef+ odpovídá termům ef, eff, efff atd.
[]	Znaky v hranatých závorkách odpovídají jednomu ze znaků v nich uvedených. Např. [efg] odpovídá libovolnému znaku e,f nebo g.
[^]	Znak stříšky na začátku řetězce v hranatých závorkách odpovídá libovolnému znaku kromě znaků v nich uvedených. Např. [^efg] odpovídá libovolnému znaku kromě e,f nebo g.
[-]	Znak pomlčky v hranatých závorkách odpovídá rozsahu znaků. Např. [e-h] odpovídá libovolnému znaku od e do h.

Za hlavní nedostatek Booleovského modelu se dá považovat neschopnost seřadit výsledek v pořadí podle relevance jednotlivých dokumentů vzhledem k položenému dotazu, všechny dotazy jsou chápány jako stejně důležité. V návaznosti na to je zde druhé omezení a to je nemožnost určení rozsáhlosti výstupu, může se stát, že výstup bude prázdný nebo naopak bude obsahovat velké množství dokumentů.

2.1.4Vektorové DIS

Novějším modelem je model vektorový, který pochází z let sedmdesátých. Hlavním cílem je vylepšení chování oproti booleovskému modelu. Ve vektorovém modelu máme *n* dokumentů reprezentovaných za pomoci *m* různých termů. Každý z *n* dokumentů je pak indexován vektorem s *m* složkami, kde každá z nich odpovídá váze významu určitého termu.

$$d_i = (w_{i1}, \dots, w_{im}) \in \langle 0; 1 \rangle^m$$

Pokud je $w_{ij} = 0$, pak term j není v dokumentu důležitý, ale neznamená to, že zde není vůbec obsažen, naopak pokud $w_{ij} = 1$, je term j pro dokument velice důležitý. Indexovým souborem ve vektorovém modelu nazýváme matici o rozměrech $n*m$.

$$D = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix} \in \langle 0; 1 \rangle^{nm}$$

Dotazem ve vektorovém prostoru je pak opět vektor s m složkami.

$$q = [q_1, \dots, q_m] \in \langle 0; 1 \rangle^m$$

Pokud je $q_j = 0$, pak nezáleží dle tazatele na výskytu tohoto termu j v dokumentu, naopak pokud $q_j = 1$, pak je výskyt termu j pro vyhledávání zásadní. Z anglického similarity (podobnost) pochází název ohodnocovací funkce $sim(d_i,q)$, která vrací podobnost dvou dokumentů. Zápis této funkce

$$sim(d_i, q) = \sum_{k=1}^{m} w_{ik} * q_k$$

je vzorec pro skalární součin vektorů d_i a q.

Při použití funkce *sim* se zbavíme hlavního neduhu booleovského modelu, a to nemožnosti řazení. Řazení se provádí sestupně na základě výsledku této funkce, trpí však neduhem jiným. Protože výsledek je závislý na velikosti vektorů, jsou následně zvýhodněny delší dokumenty, s větším množstvím slov. Možnosti jak zvýhodnění zredukovat jsou dvě. První je zavedení komplikovanější definice funkce *sim*, která zohlední parametr délky vzorců. Druhou je pak normalizace vektorů podle vzorce

$$d_i = \frac{d_i}{|d_i|}$$

Dojde tak k převodu vektorů z jednotkové m-dimenzionální krychle do jednotkové m-dimenzionální sféry. Ve vektorovém modelu dále není prostředek jak vynutit obsažení určitého termu či naopak jak jej zakázat. První možnost je částečně kompenzována řazením podle podobnosti. Druhou variantu lze umožnit povolením zadávání záporné váhy v dotazu následovně

$$q = [q_1, \dots, q_m] \in \langle -1; 1 \rangle^m$$

Pokud je term následně zastoupen zápornou váhou, je díky záporné hodnotě znevýhodněn v součtu. Ve vektorovém modelu se pro řízení vstupu využívají dvě pravidla. Prvním je stanovení nejnižší povolené hodnoty koeficientu podobnosti, výsledky pod tuto mez se do výsledků nezapočítávají. Druhým pravidlem je stanovení maximální délky výstupu, následně se na výstupu zobrazí jen omezený počet dokumentů. Díky tomu, že výsledky se řadí sestupně, bude se jednat o nejvíce relevantní zdroje.

Základem vektorového modelu je stejně jako u modelu booleovského indexace. Zde je však nejdůležitějším faktorem frekvence výskytu určitého termu. Zajímavostí je, že pokud nahradíme všechny nenulové váhy ve vektorech jedničkami, dostaneme zpět booleovský model.

2.2 Požadavky

Všechny uvedené problémy výše se týkají tématu vyhledávání informací v textu, vyhledávače jsou dnes již nedílnou součástí potřeb každého uživatele elektronického zařízení, stali se tak i součástí výuky informatiky na vysokých školách.

Každé z řešení se skládá z několika netriviálních částí, které mohou posluchačům při prvotním seznámení činit komplikace s pochopením. Hlavním požadavkem k řešení je tak umožnění vizuálního zobrazení jednotlivých částí a jejich způsobu fungování. Uživatel by měl mít také možnost vkládat vlastní data a pozorovat změny, které jejich vložení přineslo. Při návrhu a vývoji implementaci by mělo být pamatováno na snadnost použití a uživatelský komfort, opomenuta by neměla být ani možnost budoucí rozšiřitelnosti o další části a také na celková modularita aplikace.

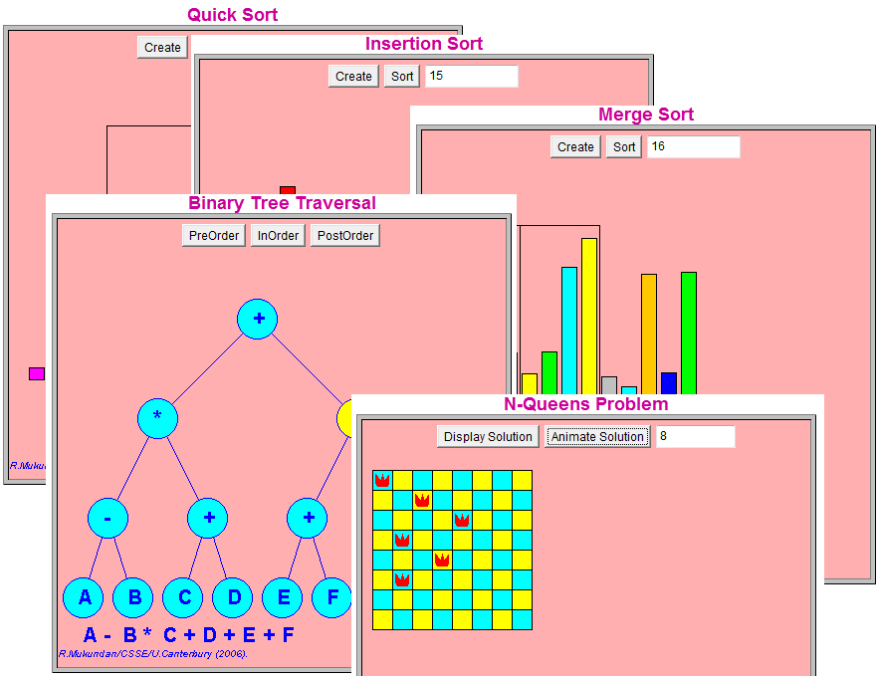
2.3 Existující implementace

Při zkoumání existujících řešení lze narazit na několik variant. Jedná se o řešení pomocí jednotlivých jednoduchých Java appletů, složitějších Java appletů či aplikací sdružujících vizualizaci několika algoritmů a Enginu pro vizualizaci jednoduchých algoritmů zapsaných ve zdrojovém kódu.

2.3.1Java Applet

Pro základní algoritmy a datové struktury lze nalézt množství Java appletů, které se snaží zobrazit jejich fungování. Jedná se o jednoduché aplikace, které mají za cíl objasnit čtenáři průběh algoritmu, či fungování datové struktury. Nedisponují žádnou větší možností konfigurace, některé se po načtení pouze spustí a v cyklech opakují. Hodí se jako názorný doplněk přednášky či prezentace, pro lepší pochopení probírané látky.

2.3.1.1 *Java Applets Centre*[1]



obr. 2.3: několik ukázkových appletů z *Java Applets Centre*

Java Applets Centre [Muk] je soubor několika jednotlivých appletů, každý je zaměřen na jiný informatický problém, datovou strukturu, či algoritmus [obr. 2.3]. Applety jsou doplněny o krátký stručný popis s návodem použití a případně zdrojovým kódem algoritmu, uživatelské rozhraní je jednoduché a přesně popsané.

Applety jsou jednoduché a přehledné, vylepšením by mohla být možnost změny grafického zobrazení, která by umožnila více pohledů na průběh stejného výpočtu a nabídnout tak každému uživateli výběr vyhovující prezentace.

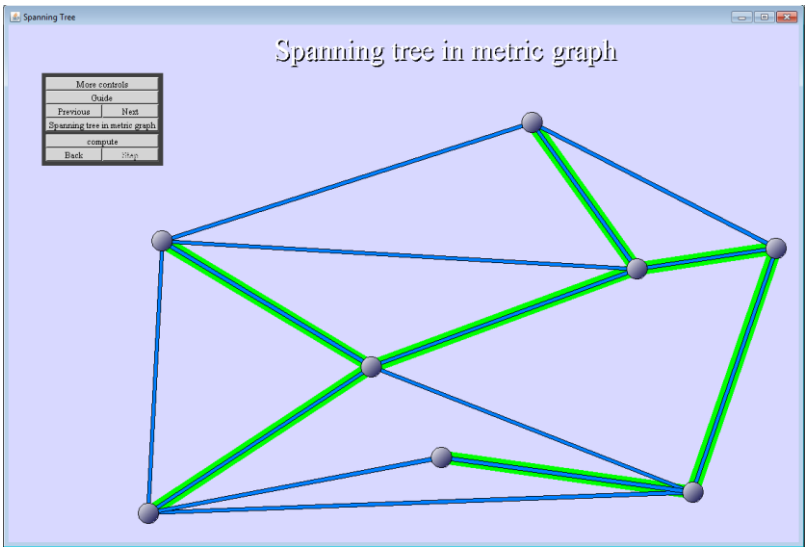
[1] http://www.cosc.canterbury.ac.nz/mukundan/JavaP.html

2.3.1.2 *Algovision*[2]

obr. 2.4: úvodní obrazovka *Algovision*

Algovision [Algo] je obsáhlejší applet, který sjednocuje více algoritmů do tematických okruhů [obr. 2.4]. Oproti předchozímu řešení je tak možné bez dalšího hledání jednotlivá řešení přímo porovnávat. Pro lepší schopnost srovnání by bylo vhodné rozšířit jej tak, aby zde byla možnost spustit podobné algoritmy ze stejných skupin na stejných zdrojových datech. Nevýhodou však je, že jednotlivé applety ze stejné skupiny mají rozdílné grafické znázornění, což v konečném důsledku porovnání znesnadňuje.

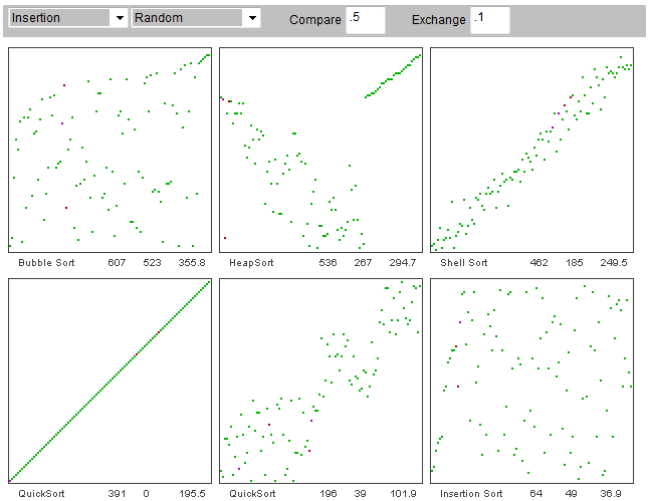
[2] http://kam.mff.cuni.cz/~ludek/Algovision/Algovision.html



obr. 2.5: ukázka průběhu vizualizace v *Algovision*

Ovládání [obr. 2.5] je pokročilé, obsahuje i krokovací funkci, je tedy možné jednotlivé kroky libovolně vracet a posouvat, u vybraných algoritmů je možné měnit nebo přidávat způsoby zobrazení a další nastavení animace.

2.3.1.3 Sort Algorithms Visualizer[3]



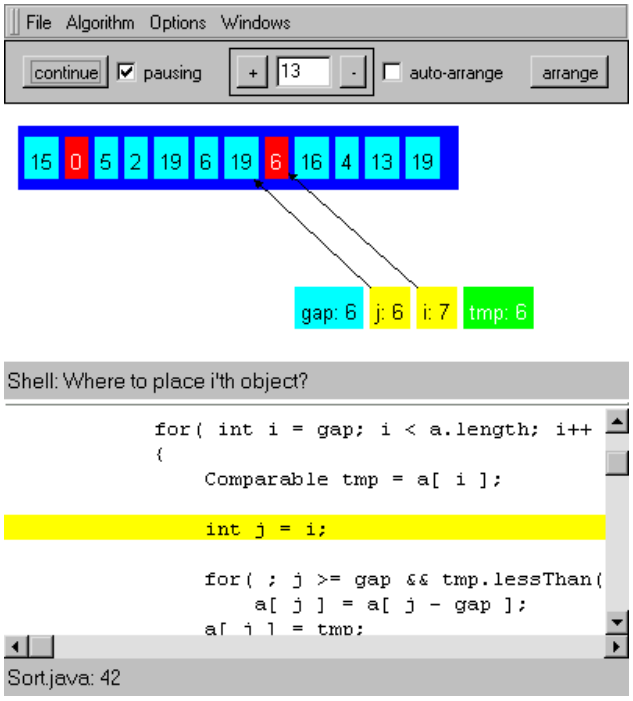
obr. 2.6: průběh vizualizace v *Sort Algorithms Visualizer*

[3] http://thomas.baudel.name/Visualisation/VisuTri/

Sort Algorithms Visualizer [Bau], jak už název napovídá, je applet zabývající se čistě třídícími algoritmy. Výhodou tohoto appletu je možnost přímého porovnání práce jednotlivých algoritmů [obr. 2.6], jejich výhod a nevýhod při zpracování různého kategorie dat, srovnání počtu porovnání, přesunů a celkového času. Jako jediný z představených appletů je doplněn o rozsáhlejší textový podklad s informacemi.

2.3.2Java Framework

2.3.2.1 Algorithm Animation Engine[4]



obr. 2.7: krokování výpočtu v *Algortihm Animation Engine*

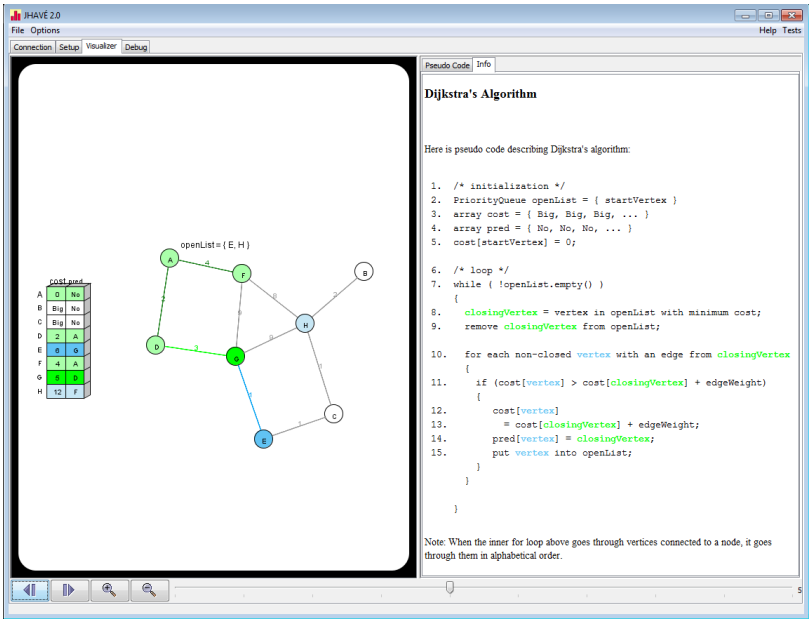
Algorithm Animation Engine [SZ] je framework umožňující tvorbu animací algoritmů zapsaných v jazyce Java nebo C++. Průběh algoritmu je možné spustit najednou nebo krokovat po jednotlivých řádcích kódu, aktuální řádek je při provádění vždy zvýrazněn [obr. 2.7], každá operace může být doplněna o informativní popisek průběhu akce. Do jednotlivých prezentací je možné

[4] http://www.cs.odu.edu/~zeil/algae.html

zakomponovat více různých algoritmů a případně i upravit generování úvodních dat. Nevýhodou tohoto frameworku je nepokračování vývoje, kdy poslední verze vyšla v roce 2005.

2.3.3Java-Hosted Algorithm Visualization Environment

2.3.3.1 JHAVÉ[5]



obr. 2.8: okno *JHAVÉ* při vizualizaci

JHAVÉ [Nap] je Java aplikace, která zobrazuje průběh algoritmů. Po spuštění nabídne připojení k serveru, kde načte seznam algoritmů, které nabídne k vizualizaci. Nabízí příjemné prostředí s možností zápisu animací v třech různých jazycích: GAIGS, Animal a XAAL. Postup výpočtu algoritmu je zvýrazňován v pravé části, v zápisu pomocí pseudokódu [obr. 2.8]. Prezentace algoritmů mohou být doplněny o otázky, které napomohou porozumění textu. Podrobnější výklad k danému tématu je dostupný pod záložkou info. Hlavním negativem této aplikace je fakt, že již od roku 2008 se beze změny jedná o stále o betaverzi.

[5] http://jhave.org/

2.3.4Další implementace

Vyjma výše zmíněných implementací existuje nespočet dalších aplikací velice podobného charakteru. Žádný z nich však nenaplnil představu fungování vizualizace DIS.

2.4 Doplnění požadavků

Z analýzy nalezených existujících řešení vizualizací vyplynulo, že cílem této práce nebude vizualizace chápána jako animování postupu práce jednotlivých algoritmů, ale půjde o simulaci zjednodušeného DIS a pohledů na uspořádání zpracovaného textu uvnitř. Bude se jednat o možnost zobrazení a vzájemného porovnání různých vizuálních pohledů na struktury využívající DIS na stejných datech, pro možnost lepšího porozumění jejich fungování a rozdílů. Uživatel bude mít možnost nastavit a změnit parametry potřebné ke zpracování textu, přidat i odebírat zdrojová data. Bude moci vkládat vlastní data ke zpracování. Bude možné zobrazit zpracovaný text před vložením do databáze a vypsat položky v databázi z posledního vkládání.

3 Návrh

Následující kapitola se bude věnovat návrhu implementace vizualizace, grafického rozhraní a výběru jazyka.

3.1 Jazyk

Základním požadavkem pro jazyk u aplikací tvořených pro potřeby výuky a studia by měla být jeho nezávislost na operačním systému. Student, jako uživatel aplikace pro znázornění a snazší pochopení látky by neměl být omezen svázaností s určitým operačním systémem.

Jazyk by měl také nabízet rozsáhlejší množství knihoven a pokročilých funkcí pro zjednodušení implementace řešení a budoucí možnost snazší rozšiřitelnosti aplikace.

Do omezujících podmínek pro výběr jazyka zde nepatří extrémní výkon, aplikace bude sloužit pro potřeby demonstrace a znázornění požadovaných dat, nebude sloužit jako výkonná část žádného vyhledávače.

Dle zmíněných kritérií se jako vhodný kandidát ukazuje platforma Sun Java 2 Standard Edition, potažmo programovací jazyk Java[6].

[6] http://www.java.com/

3.2 Výkonná část aplikace

Na aplikaci nejsou kladeny žádné nároky vzhledem k výkonu, není tedy třeba primárně řešit krajní optimalizace pomocí složitých algoritmů.

Naopak je požadováno oddělení jednotlivých částí, aby nebyl problematický jejich následný vývoj, nahrazení, rozšiřování a přidávání. Každý z funkčních celků by tak měl být umístěn do vlastního balíčku, dle logického rozdělení příbuznosti funkce.

Základní ideou bude rozdělení na tři základní funkční celky, databázová část, která se bude starat o datový úsek kódu, existuje zde možnost využití některého z externích řešení, tato možnost bude ještě zvážena. Vzhledem k podmínkám napsaným výše považuji za reálnější řešení, využití pomoci běžných nástrojů jazyka, jedná se především o možnosti budoucího vývoje, kdy plně funkční jednoduchá datová sada bude jistě lépe a flexibilněji rozšiřitelná než sofistikované řešení třetích stran.

Úvodní zpracování textu bude prováděno jednoduchou lematizací, odstraněním stop-slov a ořezáním koncovek. Protože bude tato část také samostatná, je možné ji do budoucna nahradit vylepšeným řešením, například s využitím tezaurů, které by spojování stejnovýznamových slov posunulo o značný kus kupředu.

3.3 Grafické rozhraní aplikace

Grafické rozhraní aplikace je jedna z nejdůležitějších částí, mělo by být především jednoduché, srozumitelné, nejlépe intuitivní.

Konkrétní volba rozhraní není lehkou úlohou, pro každý druh aplikace se hodí zcela odlišné řešení. Do úvahy přichází klasické zobrazení jednoho okna s množstvím ovládacích prvků, kdy se veškeré informace musí vměstnat na jednu plochu. Právě ona možnost, mít zde pouze jednu plochu, je poněkud limitující. Je možné prostor rozdělit na několik částí, ale pravidelné rozdělení nemusí být vždy zcela ideální.

První cestou, kterou je možné se vydat je volba záložek. Tato varianta představuje již model, ve kterém lze mít v jeden okamžik načteno více zobrazení, avšak ani zde není bez rozdělení možné sledovat několik náhledů zároveň.

Řešením může být, pojetí hlavního okna aplikace, jako plochy určené pro zobrazení více pohledů. Použití *InnerFrame*[7], které nabízí programovací jazyk Java, se zdá vhodným kandidátem pro podobné účely. Menu v hlavní obrazovce bude zabezpečovat dostatečný přístup k různorodým volbám a v případě potřeby zobrazení dvou, tří, či pěti náhledů, je možné jednotlivá okna uzpůsobit nárokům a porovnávat je okamžitě vedle sebe v jedné rovině.

I grafické rozhraní by mělo zůstat samostatné a oddělené od ostatních modulů, ideálně každé zobrazení, by mělo mít svou vlastní třídu, aby budoucí rozvoj byl přímo dán tvořením samostatných zobrazovacích tříd.

3.4 Rozšiřitelnost

V průběhu implementace a vývoje aplikace musí být kladen důraz na rozdělení jednotlivých funkčních celků do samostatných součástí. Dojde tak ke zjednodušení dalšího vývoje, či rozšiřování aplikace.

Bude možné tak snadno nahradit, či rozšířit modul pro lemmatizaci a zpracování vstupního textu. Při nahrazení bude také nutné dodržení a implementace kompletního rozraní tohoto modulu. Další možností a zároveň hlavní možností pro rozšíření bude doplnění modulu pro vizualizaci o další pohledy do struktury uložených dat. Aplikaci tak bude možné rozšířit o požadované části, či doplnit o další DIS k porovnání. Poslední možnost rozšíření pak nabízí záměna databázového modulu, který bude moci být tak nahrazen případnou pokročilejší nebo efektivnější variantou nebo jej pouze rozšířit o další možnosti získání dat pro vizualizační moduly.

[7] http://download.oracle.com/javase/tutorial/index.html

Pro rozšíření o další zobrazovací metodu je počítáno, že bude třeba pouze vytvořit zobrazovací třídu, přidat modul do nabídky a doplnit metodu pro získávání specifického formátu dat z databáze.

4 Programátorská dokumentace

Program je strukturován do třech hlavních pasáží, jsou to databáze, lemmatiser a engine. Postupně zde budou rozebrány principy jejich fungování.

4.1 Lemmatiser

Lemmatiser je tvořen pěti třídami, které se postupně starají o rozebrání zadaného textu.

Hlavní z nich je právě třída *Lemmatiser*, která má několik úkolů. Především se stará o komunikaci s ostatními moduly. Obsahuje několik privátních položek, v proměnné *text* jsou dostupná výstupní data určená k výpisu po lematizaci, ty dostává od třídy *Text*, která se stará o samostatný průběh lematizace. Dalším úkolem této třídy je základní načtení dat pro slovník koncovek, v proměnné *ending*, který je využíván v dalších třídách a také načtení seznamu stop-slov, proměnná *dictonary*, která jsou neméně důležitá pro korektní odstranění nepotřebných slov z textu.

Druhou důležitou součástí je třída *Text*, ta se stará o samotný průběh, jak již bylo zmíněno. Zabezpečuje přípravu dat pro zápis do databáze, na druhou stranu také sestavuje tematizovaný text pro výpis po lematizaci a poslední hlavní náplní je rozdělit text na odstavce, které jsou odeslány ke zpracování další třídě a to, *Paragraph*.

Tato se podobně jako předchozí stará o další rozdělování textu, tentokrát na věty, naopak z vět zpětně skládá odstavce, pro výpis tematizovaného textu a slouží jako transportní uzel pro data mířící do databáze.

Obdobně řeší vývoj i třída *Sentence* se zaměřením na dělení a slučování slov v rámci vět a souvětí.

Posledním krokem je třída *Word,* kde probíhá hlavní část lematizace, porovnávání s načteným slovníkem stop-slov, ořezávání koncovek, zvýraznění termů ve slovech. Nemenší úkol je také to, že slouží jako transportní třída pro přenos informací a dat do databáze, kde dojde k jejich následnému uložení.

4.2 Databáze

Databáze není tvořená složitým systém, některým ze speciálních datových struktur nebo externích modulů třetích stran, k tomuto účelu určených, ale díky povaze zadání je plně postačující řešení pomocí dostupných základních typů programovacího jazyka Java.

Databáze má podobně jako část lematizeru svoji hlavní třídu pojmenovánu *Database*, pomocí ní je zabezpečen veškerý přístup k ostatním modulům. Její funkcí je přenášet požadavky na své podtřídy, které je díky zjednodušování na postupných úrovních textu vyřeší. Součástí třídy *Database* jsou přístupové metody pro data, ty jsou následně využívány voláním z modulu engine.

Databáze má dvě hlavní části a to část třídy *Terms*, kde dochází k ukládání samostatných zpracovaných dat do proměnných sestavených do stromové struktury. Druhou hlavní částí je pak třída *Documents,* která zabezpečuje uložení původního textu do dočasných souborů na disku a uchování s nimi spárovaných identifikačních kódů dokumentů.

V třídě *Terms* je schováno hashované (asociativní) pole, kde je každému termu přiřazena třída *TermDocuments.* Ta zde symbolizuje přidělený identifikátor dokumentu. Tomu přiřazuje díky třídě *TermDocumentWords* seznam slov v dokumentu, které náleží příslušnému termu. A posledním článkem skládačky je pak třída *TermDocumentWordData*, která je hlavním datovým držitelem.

Veškeré datové požadavky přicházející z enginu se distribuují posloupností volání do patřičných tříd databáze, odkud získají své výsledky.

4.3 Engine

Třetí část, tedy engine obsahuje řídící část programu, kde díky činnosti uživatele, využívá funkce nadefinované v ostatních balíčcích, pro splnění zadaného požadavku.

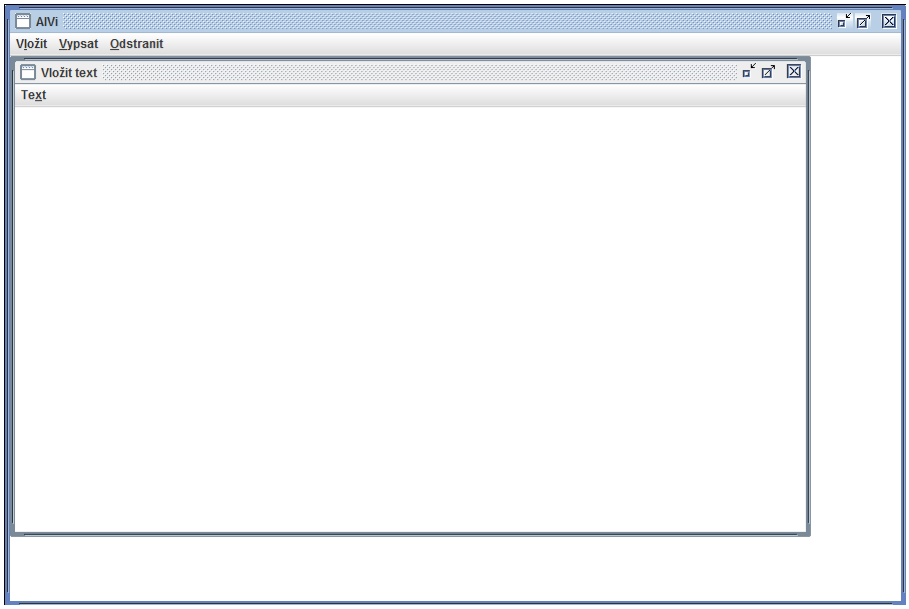
Hlavní třídou enginu je třída *Frame*, která zastupuje hlavní okno programu a jsou v ní nadefinovány veškeré nabídky aplikace, menu, i následné reakce vyvolané uživatelskou činností. Důležitým ovládacím prvkem aplikace je pak třída *ScrollableDesktop*, která u vnitřních okének zabezpečuje rolovací schopnost a ty tak jsou i při vetším množství dat použitelná pro pohodlné zobrazování.

Třída *InternalFrame* je základním kamenem zobrazení vnitřního rozložení plochy, od ní jsou zde pak odvislé třídy *InternalFrameLemmatiser* a *InternalFrameLemmatiserInput,* využité pro obstarávání výpisu a nabídek modulu lemmatiseru. Posledním zastřešujícím prvkem je pak třída *InternalFrameTable*, která umožňuje pohodlné vypisování datových požadavků od uživatele za pomoci příbuzných tříd *InternalFrameTableDocuments, InternalFrameTableDocumentTerms, InternalFrameTableTermDocuments, InternalFrameTableTermsCount.*

Součástí programátorské dokumentace je i její generovaná část ve verzi JavaDocu, která se nachází spolu s programem na přiloženém CD.

5 Uživatelská dokumentace

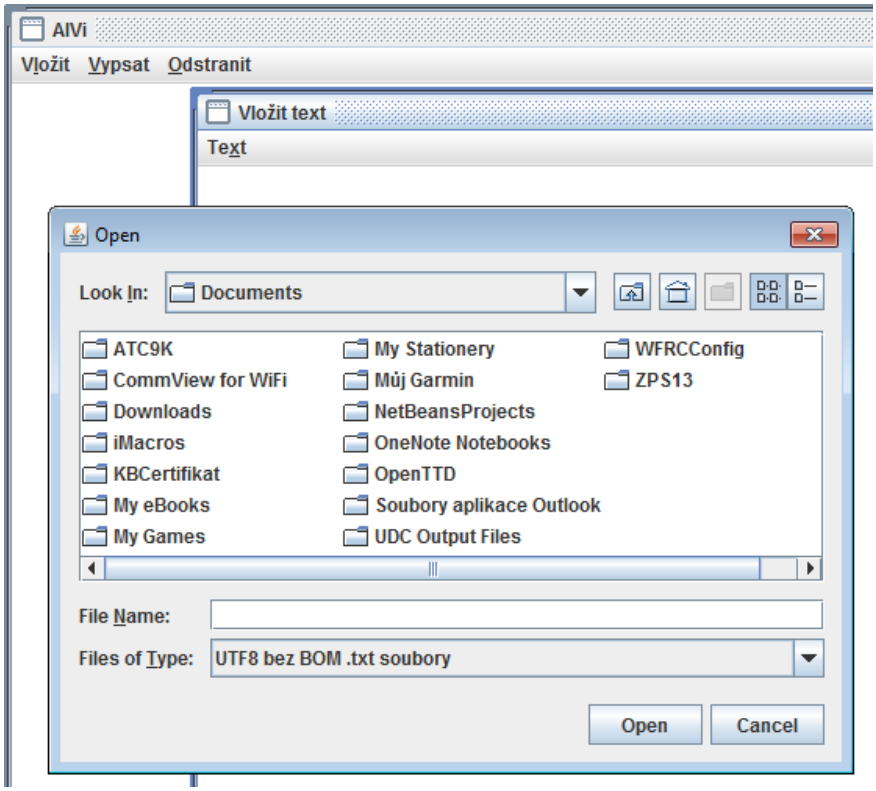
5.1 Úvodní obrazovka



obr. 5.1: AlVi – úvodní obrazovka

Uživatelské rozhraní je již na první pohled velice jednoduché, představuje jej především základní okno, s menu nabídkou v horní liště, ze které jsou dostupné volby základních tří operací s daty, a to vkládání potřebných dat, možnosti jejich zobrazení a pro případné úpravy je zde i možnost odstranění nechtěných, či nešikovných dat. Nyní se podíváme na každou z variant podrobněji.

5.2 Možnosti vkládání



obr. 5.2: AlVi – možnosti vkládání

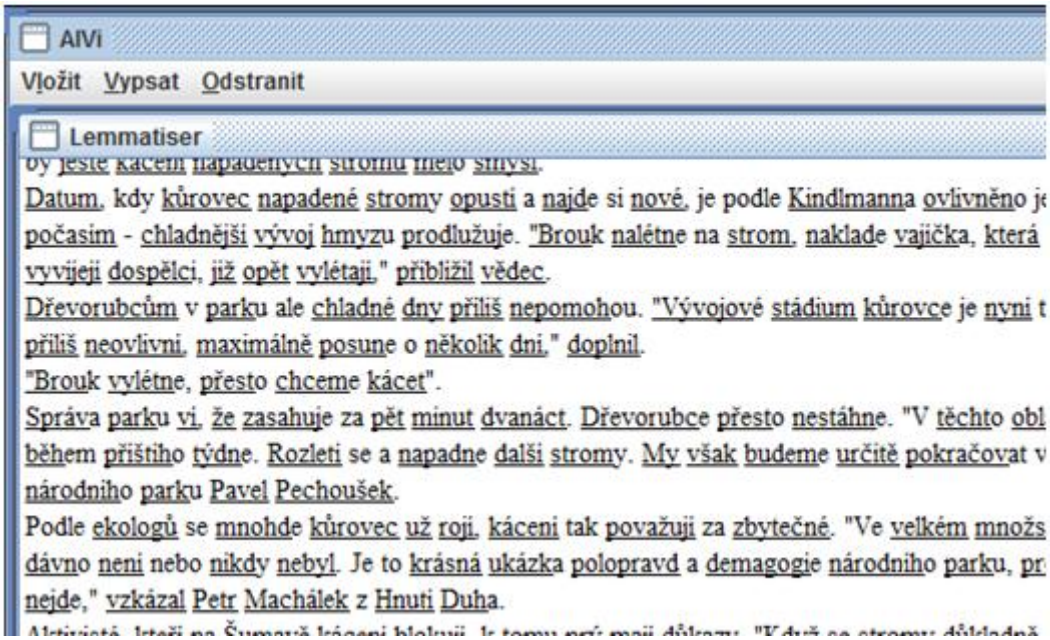
Vkládání dat do aplikace je velice snadné, jsou zde pro tento účel připraveny dvě základní varianty. První z nich je možnost ručního zadávání textu, ať už pro využití vkládání dat z externích zdrojů na internetu, kde je zajisté možno dohledat široké spektrum textů, nebo přímo pro psaní z vlastních nápadů, či testování speciálních případů, které se v běžném prostředí vyskytnou zřídka.

Druhou variantou se pak nabízí vkládání textů obsažených v souborech. Podporován je základní typ čistého textu v kódování UTF8, varianta bez úvodní značky.

Ruční zadání textu je nutné odeslat ke zpracování volbou *Text* v horní části okna a příkazem *zpracovat*. Po vložení textového souboru pokračuje vše automaticky.

Nyní, text projde *Lemmatiserem,* který dodaný text roztrhá na jednotlivá slova a ta následně projdou ořezáním o koncovky a vyřazením nepotřebných a nezajímavých slov. Seznam koncovek a stop-slov je možné doplňovat a to editací přiložených

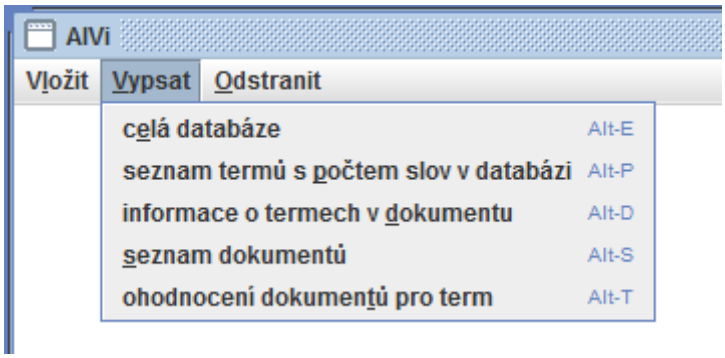
souborů *dictonary.txt*, pro slovníkové úpravy a *ending.txt*, pro úpravy seznamu ořezávaných koncovek.



obr. 5.3: Alvi - Lemmatiser

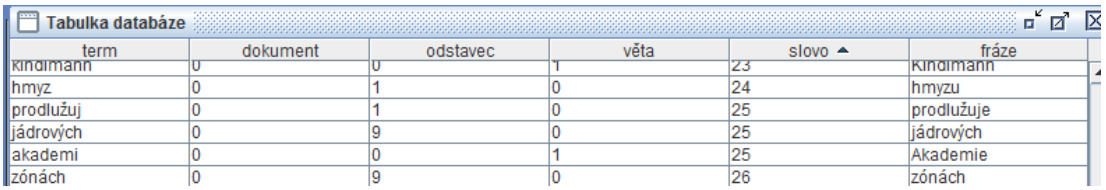
Po průchodu Lemmatiserem se zpracovaný text zobrazí k prohlížení. Podtržením jsou v něm zvýrazněna spojení, která byla označena za termy a slouží, k uspořádání dat v databázi. Data máme tedy uložena a nyní se přesuneme k možnostem jejich zobrazení.

5.3 Volby zobrazení



obr. 5.4: AlVi – volby zobrazení

Zobrazovacích módů nabízí AlVi v současné podobě pět. Jedná se, za prvé, o kompletní výpis slov použitých k uchování v databázi, s uvedením všech základních informací k nim dostupných.

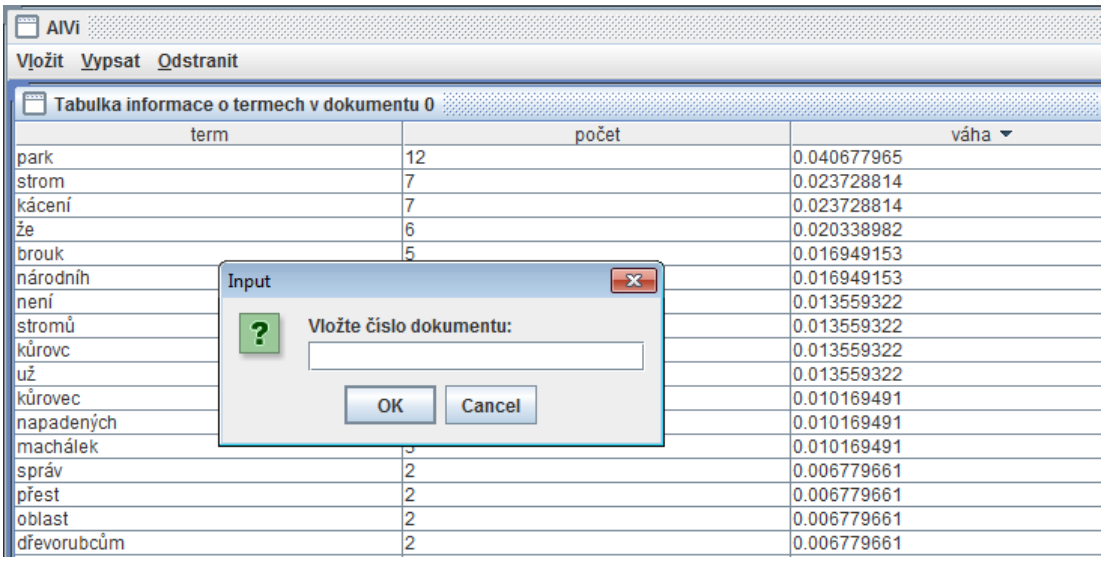


term	dokument	odstavec	věta	slovo	fráze
kindlmann	0	0	1	23	Kindlmann
hmyz	0	1	0	24	hmyzu
prodlužuj	0	1	0	25	prodlužuje
jádrových	0	9	0	25	jádrových
akademi	0	0	1	25	Akademie
zónách	0	9	0	26	zónách

obr. 5.5: AlVi – výpis databáze

Na výše uvedeném obrázku (obr. 5.5) je viditelný částečný náhled jak tento výpis vypadá. Za povšimnutí stojí možnost řazení sloupců, která je zde zobrazena trojúhelníčkem u páté položky, *slovo*. Další možností jak lze výpis přizpůsobit vlastní potřebě je možnost přetažení sloupce do libovolné polohy, kde jej chci mít, pořadí sloupců pak může být úplně jinak, než původně.

Druhou položkou výpisu je *seznam termů s počtem*(četností) *slov v databázi*, tento výpis slouží jako přehledový , především je pak využitelný pro případné odmazávání jednotlivých termů, viz níže.

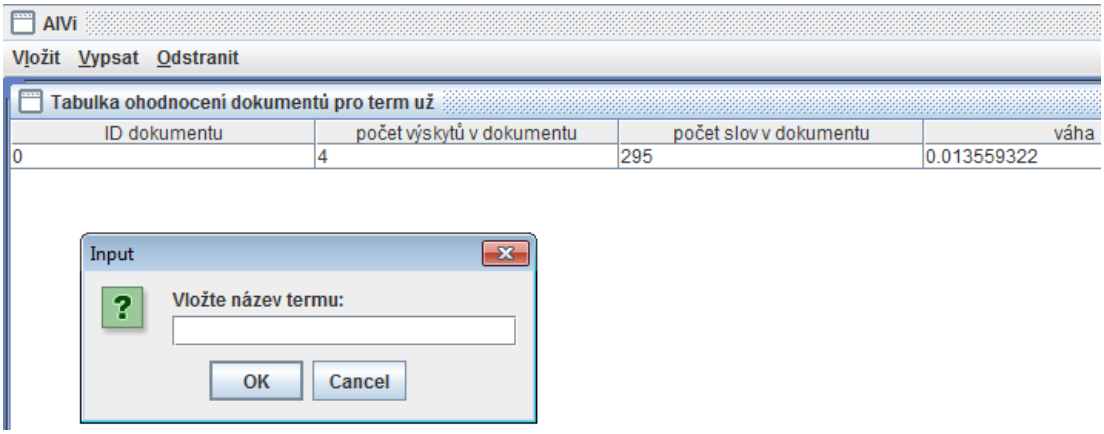


term	počet	váha
park	12	0.040677965
strom	7	0.023728814
kácení	7	0.023728814
že	6	0.020338982
brouk	5	0.016949153
národních		0.016949153
není		0.013559322
stromů		0.013559322
kůrovc		0.013559322
už		0.013559322
kůrovec		0.010169491
napadených		0.010169491
machálek		0.010169491
správ	2	0.006779661
přest	2	0.006779661
oblast	2	0.006779661
dřevorubcům	2	0.006779661

obr. 5.6: AlVi – zobrazení detailů termů dokumentu

Třetí položkou je *informace o termech v dokumentu,* zde se po výběru této položky otevře dialog pro volbu dokumentu, který tímto chceme zobrazit.

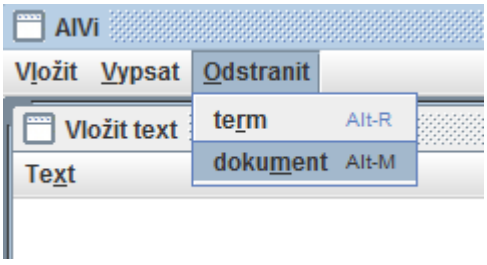
Pomocníkem pro získání správného identifikačního čísla dokumentu pak bude nejspíše položka čtvrtá, která se nazývá, *seznam dokumentů* a po jejím zobrazení je možné se dostat na seznam vložených dokumentů. U každého z nich je součástí výpisu identifikační číslo a také náhled prvního odstavce vloženého textu, který tak zjednoduší orientaci, o který text se jedná.



obr. 5.7: AlVi – ohodnocení dokumentů pro term

Poslední položkou kategorie výpisu je zobrazení *ohodnocení dokumentů pro term,* zde se ke konkrétnímu termu zobrazí seznam všech dokumentů, ve kterých se vykytuje. Společně s informacemi o četnosti výskytu a rozsáhlosti dokumentu je pak možné odvodit váhu slova definující jeho pozici vůči ostatním dokumentům.

5.4 Odstranění dat



obr. 5.8: AlVi – odstranění dat

Poslední položkou v menu je *Odstranit*, tato volba umožňuje uživatelům zásah do databáze obráceným způsobem. Pomocí identifikačního čísla dokumentu nebo nazvu termu lze nežádaná data vyjmout.

5.5 Minimalizace

obr. 5.9: AlVi – minimalizace

Díky využití vnitřních oken aplikace je možné data přeskládávat, dle své volby a porovnávat obsah a výsledky. Problémem díky minimalizaci není ani uchování dřívějších verzí pohledů na data, která lze později vyvolat.

6 Závěr

Aplikace nabízí široké možnosti dalšího vývoje, díky uživatelskému rozhraní ve formě vnitřních oken, je jednoduché jakékoliv další rozšiřování a doplnění nových modulů. V zásadě se jedná o doplnění uživatelského přístupu do nabídky menu, vytvoření potřebného funkčního základu v samostatném balíčku, grafického panelu a potřebného náhledu na data v databázi.

Pozdější rozšíření je zajisté vítané, aplikace by se tak mohla stát komplexnějším pomocníkem při potřebě porozumění databázovým aplikacím pracujícím s texty.

Na projektu je možnost stále co vylepšovat a rozšířit jej o konkurenční modely, s kterými řešený model soupeří. Omezený časový horizont pro tvorbu prací nepřináší možnost trvalého vývoje, ale aplikace zaměřené na studium si zajisté zaslouží pozornost a neměli by zůstat v zapomnění.

Literatura

[Muk] R. Mukundan: Java Applets Centre,
http://www.cosc.canterbury.ac.nz/mukundan/JavaP.html

[Algo] Luděk Kučera: Algovision,
http://kam.mff.cuni.cz/~ludek/Algovision/Algovision.html

[Bau] Thomas Baudel : Sort Algorithms Visualizer,
http://thomas.baudel.name/Visualisation/VisuTri/

[SZ] Dr. Steven J. Zeil: Algorithm Animation Engine,
http://www.cs.odu.edu/~zeil/algae.html

[Nap] Thomas Naps: Java-Hosted Algorithm Visualization Environment,
http://jhave.org/

[PSK05] Pokorný J., Snášel V., Kopecký M. (2005): Dokumentografické informační systémy. Karolinum, Praha.

[SZ07] Sharon Zakhour a kol. (2007): Java 6 – výukový kurz. Computer Press, a.s., Brno

[OJT] Oracle: The Java Tutorials
http://download.oracle.com/javase/tutorial/index.html