České vysoké učení technické v Praze

Fakulta elektrotechnická

Diplomová práce

Kontextové prohledávání textu

Praha, 2007 Autor: Štěpán Tesař

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze 19. ledna 2007 .....

podpis

Poděkování

Děkuji především vedoucímu diplomové práce Richardu Šustovi za jeho vstřícný přístup, dále pak mým blízkým za podporu.

Abstrakt

Cílem práce je program, který prohledá zadané zdroje a uloží informace potřebné k určení kontextu jednotlivých slov. Následující text pojednává o jednotlivých podproblémech a zvolených řešeních. Program byl implementován v jazyce C# a je plně funkční jako desktopová aplikace. Vstupní data jsou omezena na textové soubory v anglickém jazyce.

Abstract

The goal of this diploma thesis was to implement a program that would search the chosen input and save the information needed to specify the context of every single word. This paper describes the particular tasks and their respective solutions. The program was written in the C# programming language and is a fully functional desktop application. The input must consist of text files in the English language.

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Štěpán Tesař

Obor: Technická kybernetika

Název tématu: Kontextové prohledávání textu

Zásady pro vypracování:

1. Prostudujte si možnosti prohledávání textu s vazbou na okolí použitého slova.
2. Navrhněte vhodné uložení dat pro informaci.
3. Naprogramujte řešení v C#.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Richard Šusta, Ph.D.

Termín zadání diplomové práce: zimni semestr 2005/2006

Termín odevzdání diplomové práce: leden 2007

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická

prof. Ing. Vladimír Kučera, DrSc.
děkan

V Praze dne 20.01.2006

Obsah

Přílohy

Seznam obrázků

Seznam tabulek

Kapitola 1 – Úvod

Kontextové prohledávání textu je poměrně široký pojem, který sahá od data-miningu za účelem sběru obchodních dat, až po lingvistickou analýzu struktury jazyka. Cílem této diplomové práce je však funkční program, který uživateli umožní nejzákladnější prohledávání textových souborů.

S pomocí tohoto programu bude uživatel moci zjistit, v jakém kontextu se dané slovo, či sousloví, používá, což ocení zejména v případě, kdy píše například vědecké pojednání v cizím jazyce.

Na tomto místě vymezíme jakousi „ekologickou niku“, kterou program zaplní. Začněme diskuzí momentálně dostupných alternativ:

Prosté prohledávání:

- **Standardní hledání poskytované operačním systémem**
  Umožní nám nalézt soubory, obsahující dané slovo. Chceme-li vidět kontext slova, musíme ručně prohledávat každý soubor takovýto soubor zvlášť. Jednoduché, ale ne příliš praktické.

- **Program grep [1]**
  K dispozici uživatelům operačního systému Linux. Velice mocný nástroj, který je ale orientován na hledání, nikoliv práci s nalezenými výsledky. Navíc po každém dotazu prohledává vše znovu od začátku, což při větším objemu dat znamená dlouhou dobu, než obdržíme výsledek.

- **Internetové vyhledávače [2]**
  Velice rychlé a umožňují i sofistikovanější dotazy, ale máme malou kontrolu nad vstupními daty a práce s výsledky nenabízí příliš možností.

- **Google Desktop [3]**
  Pracuje na principu internetových vyhledávačů, je asi nejbližší alternativou tohoto programu, nicméně jeho primárním zaměřením je hledání, a proto nenabízí tolik možností jak naložit s nalezenými výsledky.

Kontextové prohledávání:

- **Specializované databáze (korpusy) [4, 5]**
  Korpusy umožňují skutečné kontextové prohledávání, ale neumožňují volbu vstupních datových souborů.

Uvážíme-li nyní výše uvedené alternativy, vidíme, že nejvíce prostoru pro nový program skýtá práce s nalezenými výsledky.

Rozdíl mezi klasickým a kontextovým prohledáváním

Představme si vědeckého pracovníka, píšícího odborný článek v angličtině. Nebude-li si jistý, jak správně zasadit do větné struktury (kontextu) slovo „matrix“, může ho zkusit vyhledat internetovým vyhledávačem. Záhy se dozví, že existuje stejnojmenný film, to mu ale moc nepomůže. Zkusí proto využít další možnosti – vyhledat sousloví „linear matrix“, nebo vyhledávání doplnit podmínkou na sousloví „control theory“. Tak nalezne výsledky ze správného oboru, ale chce-li si utvořit přehled, jaká slova mohou slovu „matrix“ předcházet, či po něm následovat (případně tento přehled dále třídit), musí se poohlédnout jinde.

Kapitola 2 – Nástin problematiky

2.1 Předzpracování vstupních dat

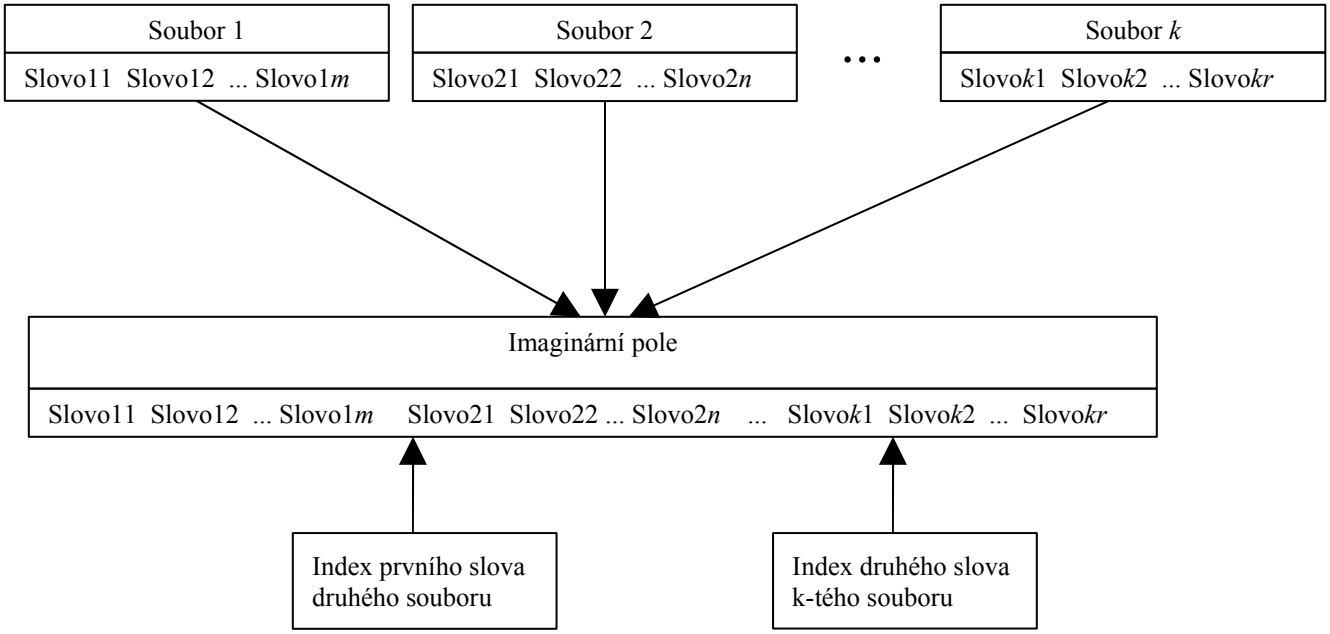
Ze všeho nejdříve je třeba připravit vstupní data na indexaci. V daném adresáři (případně i jeho podadresářích), nalezneme soubory končící na přípony indikující typ souboru, který program umí zpracovat.

U „čistých“ textových souborů, které mají obvykle příponu .txt, je jedinou potíží zvolit správné kódování.

Dále tu jsou soubory, které obsahují jak text, tak i další prvky dokumentu, například údaje o použitém fontu, číslování stránek, vložené obrázky, vzorce, atd. Tyto soubory můžeme rozdělit na soubory čitelné „pouhým okem“, jako například dokumenty napsané v programu LaTeX (přípona .tex, [6]), či soubory ve formátu HTML a soubory, jejichž obsah je čitelný pouze po zpracování nějakým programem (dokumenty PDF). Jelikož u těchto typů souborů nechceme, aby byly indexovány i formátovací (a jiné) příkazy, tak tyto příkazy nejprve odfiltrujeme a potom indexujeme zbylý text.

2.2 Indexace dat

Indexace je algoritmus, jehož vstupem jsou data (v našem případě soubory obsahující text) a výstupem jsou pozice jednotlivých slov v těchto datech. Můžeme si představit, že vstupní soubory seřazeny za sebou tvoří veliké imaginární pole znaků, viz. obr.1. Potom index slova bude indexem prvního písmene tohoto slova v imaginárním poli.



Obrázek 1: Imaginární pole

Samotný algoritmus se dá zjednodušeně popsat tak, že čteme vstupní znaky a v případě, že se jedná o slovo, si příslušný index zapamatujeme.

2.3 Uložení indexů

Po skončení indexace je třeba indexy uložit pro opětovné využití. Informace které ukládáme můžeme rozdělit do třech kategorií. První jsou informace o indexovaných souborech, především se jedná o pořadí, ve kterém byly indexovány, a jejich velikost. Dále uložíme seznam všech slov, jejichž indexy jsme během indexace zjistili. A nakonec uložíme ke každému slovu příslušné indexy, tedy pozice, kde se daný výskyt slova nalézá.

2.4 Práce s indexy

Chceme-li pracovat s uloženými daty, je třeba načíst do paměti alespoň informace o souborech a seznam slov. Dále je možné načíst buď rovnou všechny výskyty slov, nebo načítat výskyty daného slova až v okamžiku, kdy jsou potřeba. V této fázi může uživatel začít klást dotazy. Po zpracování dotazu a zobrazení výsledků je možné výsledky řadit.

2.5 Modifikace programu na vztah server-klient

Výhledově je rovněž možné uvažovat o indexaci velkého množství dat a takto získané informace dát k dispozici na internetový server. Potom by uživatelé mohli využít znalostí o kontextu daného slova aniž by nejprve museli shánět data a potom je indexovat.

Kapitola 3 – Tvorba indexů

Tato kapitola pojednává o obecném postupu vytvoření indexu a konkrétní implementaci v jazyku C#.

3.1 Nalezení vstupních dat

Soubory vhodné k indexaci hledáme v adresáři, který určí uživatel. Dále víme, zda si uživatel přeje prohledávat i případné podadresáře. Zaměřme se v dalším textu na textové (.txt) soubory a soubory vytvořené programem LaTeX (.tex, [6]). Pro informace o souboru, které si chceme zapamatovat, si vytvoříme zvláštní datovou strukturu. Ta obsahuje následující položky: polohu prvního znaku souboru v imaginárním poli (viz. obrázek 1), délku (velikost) souboru v bytech a cestu k souboru, která je relativní k cestě zadané uživatelem. Pokud během prohledávání narazíme na soubor s odpovídající příponou, uložíme informace o něm nejprve do této struktury a tuto potom do dynamické datové struktury, která tak uchovává informace o všech souborech. Nalezneme-li alespoň jeden soubor, je možné pokračovat v indexaci.

```
Vstupy:   Adresář ve kterém hledáme soubory
          Informace, zda chceme prohledávat podadresáře

Výstupy:  Nalezené soubory
          Informace, zda je na vstupu nějaký soubor typu .tex

Implementace v C#:
```

```
public struct Soubor
{
       public uint zacatek;          //na jake pozici je prvni znak souboru v imaginarnim poli
       public uint delka;            //delka souboru v bytech
       public string relcesta;       //cesta k souboru, relativni k zadane ceste
}


public ArrayList soubory = new ArrayList(); //informace o souborech, obsahuje struktury Soubor
public uint velikost=0;                     //pocet indexovanych bytu
```

```
/// <summary>
/// Najde cesty souboru urcenych pro indexaci
/// </summary>
/// <param name="cesta"> adresar ve kterem se budou indexovat soubory</param>
/// <param name="soubory"> arraylist ve kterem budou ulozeny nalezene soubory</param>
/// <param name="podadresare"> udava jestli se maji indexovat i soubory v podadresarich</param>/// <param
name="delkarootu"> puvodni pocet znaku stringu cesta - kvuli rekurzi je potreba extra parametr,
            protoze se cesta meni</param>
/// <param name="texsoubory"> udava zda byl nalezen nejaky .tex soubor (pokud ano, je pred indexaci potreba
             nacist texova stopslova)</param>
private void NajdiZdroje(string cesta, ArrayList soubory, bool podadresare, int delkarootu, ref bool texsoub)
{
      DirectoryInfo di = new DirectoryInfo(cesta);          //ziskame informaci o adresari
       FileInfo[] txtfi = di.GetFiles("*.txt");             //najdeme .txt soubory
      FileInfo[] texfi = di.GetFiles("*.tex");              //najdeme .tex soubory
       Soubor s;
       if (texfi.Length > 0) texsoub = true;
       foreach (FileInfo fiTemp in txtfi)
       {
              s.zacatek=velikost;
              s.delka=(uint)fiTemp.Length;
              s.relcesta=fiTemp.FullName.Remove(0,delkarootu);
              soubory.Add(s);
              velikost+=(uint)fiTemp.Length;
       }
      foreach (FileInfo fiTemp in texfi)
      {
             s.zacatek = velikost;
             s.delka = (uint)fiTemp.Length;
             s.relcesta = fiTemp.FullName.Remove(0, delkarootu);
             soubory.Add(s);
              velikost += (uint)fiTemp.Length;
      }
       if (podadresare)                                     //rekurzivni prohledavani podadresaru
       {
              FileSystemInfo[] dirs = di.GetDirectories();
              foreach (DirectoryInfo diNext in dirs)
              {
                     NajdiZdroje(diNext.FullName, soubory, podadresare, delkarootu);
              }
       }
}
```

Komentář k implementaci:

Začátek souboru je vlastně roven sumě velikostí všech předešlých souborů. V .NET 2.0 byla metoda DirectoryInfo.GetFiles rozšířena o možnost prohledat všechny podadresáře zadáním parametru SearchOption.AllDirectories, ale tento způsob by mohl vést k zacyklení v případě, že by adresář obsahoval zástupce tvořící smyčku.

3.2 Úprava vstupních dat

Soubory, které neobsahují jen „čistý“ text, je třeba před samotnou indexací upravit – jednoduše řečeno, upravit je tak, aby jen „čistý“ text obsahovaly. Zaměřme se v dalším pro jednoduchost na soubory vytvořené programem LaTeX [6]. V těchto typech souborů se vyskytují dva typy

klíčových slov: příkazy (commands) a prostředí (environments). Tato klíčová slova nechceme indexovat, neboť slouží k úpravě vzhledu textu, vkládání obrázků apod.

Podívejme se nejprve na syntaxi příkazů. Každý příkaz je uvozen zpětným lomítkem (\ ) a potom následuje buď jeden znak, jenž není písmenem, nebo řetězec písmen. U některých příkazů pak mohou následovat argumenty v závorkách: {} – povinné, [] – nepovinné. Příkazy o jednom či dvou znacích se nemusíme zabývat, protože budeme indexovat pouze slova s délkou alespoň tři písmena. Z hlediska úpravy vstupních dat nám stačí rozdělit zbývající příkazy do dvou skupin. V první skupině budou příkazy, které prostě vypustíme, například příkaz **\par**. Příkazy v druhé skupině vypustíme také, navíc ale vypustíme i slovo v následujících složených závorkách, například **\end{equation}**. Nechceme indexovat ani slovo ‚end', ani slovo ‚equation', protože nejsou součástí „čistého" textu dokumentu.

Prostředí jsou oblasti dokumentu vymezené příkazy **\begin{prostředí}** a **\end{prostředí}**. Prostředí rozdělíme podobně jako příkazy do dvou skupin. V první skupině budou prostředí, jež budou úplně vypuštěna, například prostředí **picture**. V druhé skupině budou prostředí, jejichž obsahem se nebudeme zabývat a odstraníme pouze příkazy značící začátek a konec prostředí, například prostředí **center**. Protože znaky $, kterým nepředchází zpětné lomítko, značí počátek a konec prostředí **math** a **displaymath**, naložíme s textem mezi nimi jako kdyby šlo o tato prostředí, tj. vypustíme ho.

Zmiňme ještě prostředí **verbatim**, jehož obsah necháme tak jak je, i kdyby obsahoval jiná prostředí nebo příkazy.

Teď již víme, které části dokumentu vypustíme, zbývá si jen ujasnit, jaký způsob je nejvhodnější. Vypuštění ve smyslu vynechání znaků (zkrácení délky souboru) není moc vhodné, neboť takto upravený soubor nebude obsahovat indexovaná slova na stejných pozicích jako soubor původní, čehož využijeme později při práci s výsledky hledání. Proto nahradíme vypouštěné znaky mezerami (v Unicode U+0020, neboli 32 v desítkové soustavě). Takto upravený soubor potom indexujeme.

`Vstupy:  Soubory vytvořené programem LaTeX`

`Výstupy:  Upravené soubory`

`Implementace v C#:`

```
/// <summary>
/// Nalezne pozice substringu ve stringu
/// </summary>
/// <param name="kde"> kde se ma hledat</param>
/// <param name="co"> co se ma hledat</param>
/// <param name="indexy"> pozice vyskyu</param>
/// <returns> true pokud byl nalezen alespon jeden vyskyt</returns>
public bool IndexVsech(string kde, string co, out int[] indexy)
{
	List<int> pomlist = new List<int>();
	int pozice = kde.IndexOf(co, 0);
	while (pozice != -1)
	{
		pomlist.Add(pozice);
		pozice=kde.IndexOf(co, pozice+1);
	}
	indexy = pomlist.ToArray();
	return (pomlist.Count > 0);
}
```

```
/// <summary>
/// Nalezne pozice substringu kterym nepredchazi znak '\' ve stringu
/// </summary>
/// <param name="kde"> kde se ma hledat</param>
/// <param name="co"> co se ma hledat</param>
/// <param name="indexy"> pozice vyskyu</param>
/// <returns> true pokud byl nalezen alespon jeden vyskyt</returns>
public bool IndexVsechNePoLomitku(string kde, string co, out int[] indexy)
{
       List<int> pomlist = new List<int>();
       int pozice = kde.IndexOf(co, 0);
       while (pozice != -1)
       {
              if (kde[pozice-1]!='\\') pomlist.Add(pozice);
              pozice = kde.IndexOf(co, pozice + 1);
       }
       indexy = pomlist.ToArray();
       return (pomlist.Count > 0);
}


/// <summary>
/// Prevede vstup na cisty text
/// </summary>
/// <param name="tr"> vstupni text</param>
/// <returns> upraveny text</returns>
public string UpravTex(TextReader tr)
{
       int[] indexy;
       int[] indexykoncu;
       char[] pompole;
       string texsouborstring = tr.ReadToEnd();
       pompole = texsouborstring.ToCharArray();
       if (IndexVsechNePoLomitku(texsouborstring, "$", out indexy))
       {
           if (indexy.Length%2==0)
              for(int i=0;i<indexy.Length;i++)
              {
                   for (int j = indexy[i]; j < indexy[i + 1] +1; j++)
                       if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                   i++;
               }
            else
               MessageBox.Show("Odd count of $ signs in the file: " + textBoxSoubor.Text);
       }
       foreach (string s in fHO.texenvironmezi)
       {
           if (IndexVsech(texsouborstring, "\\begin{" + s + "}", out indexy))
               if (IndexVsech(texsouborstring, "\\end{" + s + "}", out indexykoncu))
               {
                    if (indexy.Length == indexykoncu.Length)
                        for (int i = 0; i < indexy.Length; i++)
                        {
                           for (int j = indexy[i]; j < indexykoncu[i] + s.Length + 6 + 1; j++)
                              if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                        }
                    else
                        MessageBox.Show("The beginnings and ends of the " +s +
                                " environment do not match in the file: " + textBoxSoubor.Text);
               }
       }
       foreach (string s in fHO.texenvirontag)
       {
            if (IndexVsech(texsouborstring, "\\begin{" + s + "}", out indexy))
              if (IndexVsech(texsouborstring, "\\end{" + s + "}", out indexykoncu))
              {
                   if (indexy.Length == indexykoncu.Length)
                        for (int i = 0; i < indexy.Length; i++)
                        {
                           for (int j = indexy[i]; j <indexy[i] + s.Length + 8 + 1; j++)
```

```
                            if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                              for (int j = indexykoncu[i]; j <indexykoncu[i] + s.Length + 6 + 1; j++)
                                  if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                        }
                    else
                         MessageBox.Show("The beginnings and ends of the " + s +
                                 " environment do not match in the file: " + textBoxSoubor.Text);
            }
        }
        foreach (string s in fHO.texdeklaracezavorky)
        {
            if (IndexVsech(texsouborstring, s, out indexy))
                for (int i = 0; i < indexy.Length; i++)
                {
                    int j = indexy[i];
                    for (; j < s.Length + 1; j++)
                         if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                    while (texsouborstring[j] != '}')
                    {
                         if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                         j++;
                    }
                    if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                }
        }
        foreach (string s in fHO.texdeklarace)
        {
            if (IndexVsech(texsouborstring, s, out indexy))
                for (int i = 0; i < indexy.Length; i++)
                {
                    for (int j = indexy[i]; j < indexy[i] + s.Length /*+ 1*/; j++)
                         if ((pompole[j] != '\n') && (pompole[j] != '\r')) pompole[j] = ' ';
                }
            if (IndexVsech(texsouborstring, "\\begin{verbatim}", out indexy))
                if (IndexVsech(texsouborstring, "\\end{verbatim}", out indexykoncu))
                {
                    if (indexy.Length == indexykoncu.Length)
                         for (int i = 0; i < indexy.Length; i++)
                         {
                            for (int j = 6; j < 16; j++)
                               pompole[indexy[i] + j] = ' ';
                            for (int j = indexy[i] + 16; j < indexykoncu[i]; j++)
                               pompole[j] = texsouborstring[j];
                         }
                    else
                         MessageBox.Show("The beginnings and ends of the verbatim environment do
                                  not match in the file: " + textBoxSoubor.Text);
        }

        texsouborstring = new string(pompole);
        return texsouborstring;
}
```

Komentář k implementaci:

Výše uvedený algoritmus není příliš efektivní, protože nesleduje, které části vstupního souboru již byly nahrazeny mezerami a pokud například prostředí **math** obsahuje příkaz **equation**, tak znaky uvnitř příkazu budou nahrazeny dvakrát, což je zbytečné.

Algoritmus využívá funkce, které určí pozice klíčových slov (a znaku $, kterému nepředchází lomítko).

3.3 Použití stopwords

S rostoucím objemem vstupních dat roste čas potřebný k vytvoření indexu i velikost souboru obsahujícího indexy slov a proto je někdy výhodné vytvořit seznam slov, která nebudou indexována, tato slova nazývejme stopwords. Jedná se zejména o nejčastěji používaná slova, u kterých je splněn předpoklad, že je uživatel nebude chtít vyhledat. Nejlepším příkladem je anglické slovo ‚the'. Padesát nejběžnějších anglických slov, delších než dva znaky, je v příloze 1. Použití stopwords zrychlí vytvoření indexu a sníží místo potřebné pro jeho uložení, ovšem na práci s již vytvořeným indexem už nemá vliv. Jinými slovy, máme-li dost času a místa, je zbytečné stopwords používat. V tabulce 1 jsou výsledky testů nad textovými soubory anglicky psané prózy z Projektu Gutenberg [7].

Velikost vstupních dat [MB]	Stopwords	Doba indexace [s]	Velikost dat indexu [MB]
112	Ne	159	62,6
	Ano	144	44,5
61,8	Ne	72	34,6
	Ano	76	25,1
32,9	Ne	38	18,9
	Ano	41	13,3
16,5	Ne	19	9,8
	Ano	19	6,9

Tabulka 1 – vliv stopwords na dobu vytvoření indexu a jeho velikost

Z tabulky je vidět, že stopwords snižují dobu indexace až při větším objemu dat a úspora místa je zhruba čtvrtinová.

```
Vstupy:   Informace, zda chceme použít stopwords

Výstupy:  Stopwords, neboli slova, která nebudou indexována
```

3.4 Odhad počtu unikátních slov

Během indexace budeme sice pracovat s dynamickými datovými strukturami, nicméně pokud bychom dopředu znali počet unikátních slov ve vstupních datech, mohli bychom snížit dobu indexace. Například pro 112 MB dat došlo ke snížení o 10%. Proto se pokusíme odhadnout počet unikátních slov dopředu na základě velikosti vstupních dat a empirických hodnot v tabulce 2.

Velikost vstupních dat [MB]	16,5	32,9	61,8	97,3
Počet unikátních slov	46359	60217	149456	166198
Počet unikátních slov na 1 MB [MB^{-1}]	2810	1830	2418	1708

Tabulka 2 – počet unikátních slov v závislosti na velikosti vstupních dat

Je zřejmé, že počet unikátních slov na 1 MB vstupních dat by se měl s rostoucím objemem vstupních dat snižovat. V praxi ale může nastat situace, kdy je nějaká část dat mnohem „unikátnější“ než zbytek a způsobí ne pokles, ale nárůst unikátních slov na 1 MB dat. To se stalo i ve třetím sloupci tabulky. Naopak duplicitní výskyt nějakých souborů ve vstupních datech může způsobit prudký pokles této veličiny. Proto nezbývá než konstatovat, že odhad počtu unikátních slov bude velmi nedokonalý. Nicméně dle [8] je přidání prvku do hashovací tabulky s volným místem operace se složitostí O(1), zatímco pokud je hashovací tabulka plná, má přidání dalšího prvku složitost O(n), proto je i nepřesný odhad lepší než žádný odhad.

```
Vstupy:   Velikost vstupních dat

Výstupy:  Předpokládaný počet unikátních slov
```

3.5 Načtení slova ze vstupu

Načtení slova je problém, jehož náročnost stoupá úměrně s tím, jak moc rozmanité znaky a kódování chceme zpracovávat. Pro vysvětlení algoritmu definujme slovo jako řetězec písmen anglické abecedy o délce alespoň tři znaky.

Slovo načítáme tak, že čteme vstupní znaky tak dlouho, dokud nedojdeme na konec vstupních dat, nebo nenačteme písmeno. Pokud jsme načetli písmeno, načítáme další znaky, dokud nedojdeme na konec souboru, nebo dokud nenačteme znak, jenž není písmenem. V tomto okamžiku zkontrolujeme, zda má načtený řetězec písmen alespoň tři znaky a pokud ano, upravíme slovo, aby neobsahovalo žádná velká písmena a prověříme, zda slovo není v seznamu stopwords. Pokud je i tato podmínka splněna, bylo načteno slovo, které chceme indexovat. Všechna načtená slova jsou v malých písmenech (lower-case).

```
Vstupy:   Textový soubor
          Absolutní pozice ve vstupních datech

Výstupy:  Načtené slovo
          Absolutní pozice ve vstupních datech

Implementace v C#:
```

```
/// <summary>
/// Urci zda je char pismenem abecedy
/// </summary>
/// <param name="i"> cislo odpovidajici znaku</param>
/// <returns> true pokud je vstup pismeno</returns>
private bool JePismeno(int i)
{
       return (((i > 64) && (i < 91)) || ((i > 96) && (i < 123)));
}


/// <summary>
/// Urci zda je slovo korektni
/// </summary>
/// <param name="s"> slovo ktere ma byt overeno</param>
```

```
/// <returns> true pokud je slovo korektni</returns>
private bool JeKorektniSlovo(string s)
{
        if (s.Length > 2)                           //slovo neni stopword a ma alespon tri pismena
                return (stopslova.BinarySearch(s) < 0);
        else return false;
}


/// <summary>
/// Nacte slovo ze vstupu
/// </summary>
/// <param name="tr"> vstupni data</param>
/// <param name="pozice"> pozice - kolik znaku jsme jiz precetli</param>
/// <returns> nactene slovo nebo null</returns>
public string NactiSlovo(ref TextReader tr, ref int pozice)
{
        int i, nacteno = 0;
        string s=null;
        char c=' ';
        bool korektni=false;
        do
        {
                i = 0;
                do                              //preskocime nepismena
                {
                        nacteno = tr.Read();
                        pozice++;
                }
                while ((!JePismeno((char)nacteno)) && (nacteno != -1));

                if (nacteno == -1)          //jsme na konci souboru
                {
                        break;
                }
                c = (char)nacteno;
                do          //nacitame a ukladame pismena az do nepismene nebo konce souboru
                {
                        pc[i++] = c;
                        nacteno = tr.Read();
                        pozice++;
                        c = (char)nacteno;
                }
                while (JePismeno(c));

                s = new string(pc, 0, i);
                s = s.ToLower();
                korektni = JeKorektniSlovo(s);
        }
        while ((!korektni) && (nacteno != -1));     //je nalezene slovo korektni?
        if (!korektni) s=null;
        return s;
}
```

Komentář k implementaci:

Funkce pro zjištění, zda je načtený znak písmeno, je v tomto případě nejjednodušší možná.

3.6 Indexace souboru

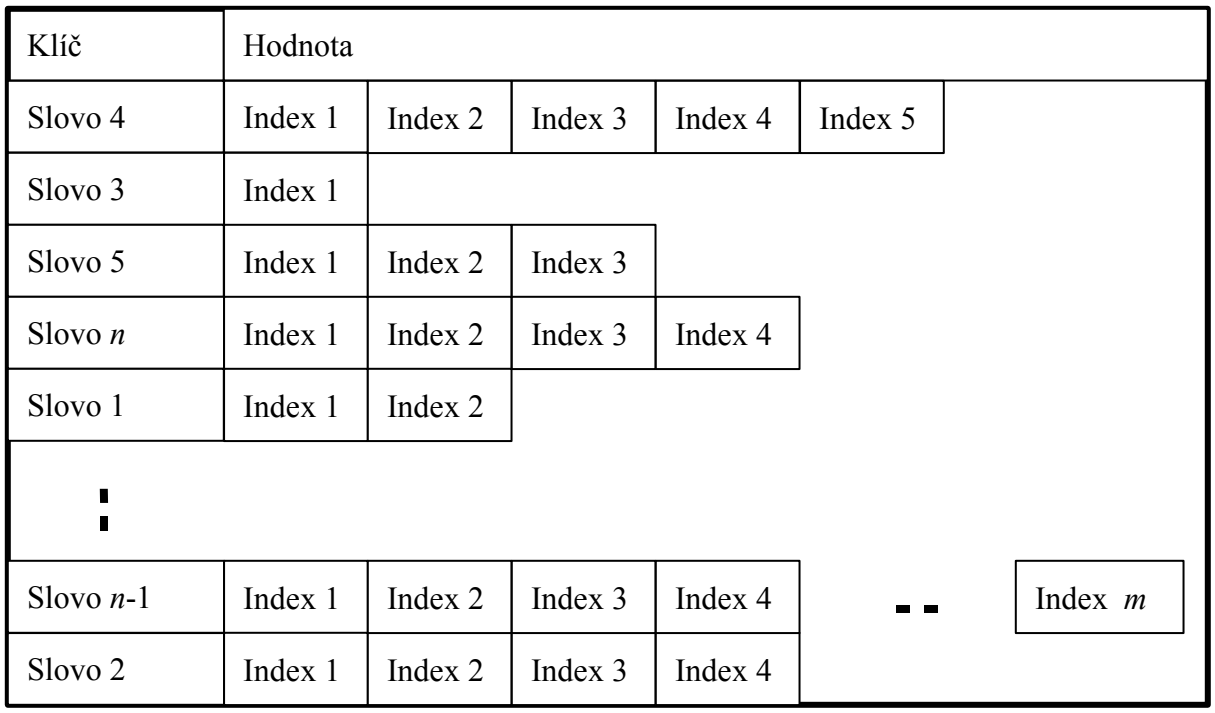
Během indexace budeme pro uložení výskytů slova ve vstupních datech používat hashovací tabulku, což je dynamická datová struktura skládající se ze dvojice klíč – hodnota. K hodnotám lze přistupovat pomocí příslušného klíče, kterým bude dané slovo. Protože dopředu nevíme, kolik výskytů jednoho slova ve vstupních datech nalezneme, bude hodnota hashovací tabulky muset být rovněž dynamická struktura, například ArrayList.

Na obrázku 2 je příklad, jak může hashovací tabulka vypadat. Všimněme si, že klíče nejsou nikterak uspořádané, protože jsou ukládány postupně tak, jak se vyskytnou ve vstupních datech. Protože konečným cílem programu je vyhledávání, bude uspořádání všech nalezených slov podle abecedy dříve či později nutné. Bylo by sice možné uložit výstupní data na disk nesetříděná, s tím, že by se třídila až před vyhledáváním, ale musela by se třídit znovu po každém načtení dat z disku, což by při větším objemu dat bylo uživateli na obtíž, proto data setřídíme ještě před uložením na disk.

Podívejme se nyní na obrázek 1. Je na něm znázorněno, že na vstupní data o více různých souborech lze pohlížet jako kdyby tyto soubory seřazeny za sebou tvořily jedno velké pole, případně jeden velký soubor. Toto realizujeme zavedením for-cyklu přes všechny vstupní soubory s tím, že pro dané slovo nás nezajímá jeho pozice v „jeho“ souboru, ale jeho absolutní pozice, neboli jeho index v imaginárním poli z obrázku 1.

Algoritmus indexace tedy spočívá v tom, že pro každé načtené slovo se podíváme do hashovací tabulky, jestli se v ní už toto slovo jako klíč nalézá. Pokud ne, vytvoříme ArrayList o jednom prvku, kterým bude absolutní pozice slova (jeho index) a tento ArrayList vložíme do hashovací tabulky pod klíčem slova. Pokud ano, tak ArrayList odpovídající tomuto slovu jako klíči vyjmeme, index slova do něj přidáme a takto upravený ArrayList vrátíme na jeho původní místo v hashovací tabulce.

Po skončení algoritmu jsou slova a jejich indexy uloženy v paměti v hashovací tabulce a zbývá je již jen setřídit a uložit na disk.



Obrázek 2: Struktura hashovací tabulky obsahující jako hodnoty ArrayListy

Vstupy: Textové soubory

Výstupy: Hashovací tabulka

Implementace v C#:

```
velikost=0;                     //inicializace
bool texsoubory = false;        //udava jsou-li nektere ze vstupnich souboru ve formatu.tex
soubory = new ArrayList();
cesta=PomFunkce.NormujCestu(textBoxAdresar.Text);
NajdiZdroje(cesta, soubory, checkBoxPodadresare.Checked, cesta.Length, ref texsoubory); //nalezeni souboru
if (soubory.Count==0) throw(new AniJedenSouborException());
NactiStopWords(texsoubory);
velikostMB = (int)velikost / 1048576;
textBoxVelikost.Text = string.Format("{0:F}",(float)velikost / 1048576);
vyskyty = new Hashtable(velikostMB * OdhadPoctuSlovNaMB(velikostMB),((float)fHO.trackBarLoadFactor.Value)/10);
string s;
Slovo pomslovo;
int pozice;
uint abspozice=0;
ArrayList al;
int poslednipocet = 0;              //zabranuje prilis castemu vypisu stavu a zpracovani zprav
bool txtsoubor;                             //udava jestli se prave zpracovava .txt soubor
inicializacniCas = DateTime.Now;
textBoxPrubeh.Text = "Reading Files";
foreach(Soubor pomsoub in soubory)
{
       progressBarCelkem.Maximum=(int)velikost/1024;
      using (StreamReader sr = new StreamReader(cesta + pomsoub.relcesta, System.Text.Encoding.ASCII, true))
       {
```

```
            textBoxSoubor.Text=pomsoub.relcesta;
             progressBarSoubor.Maximum=(int)pomsoub.delka/1024;
            txtsoubor = pomsoub.relcesta.ToLower().EndsWith(".txt");
             pozice=-1;
            bool byltexupraven=false;
            TextReader tr = (TextReader)sr;
            while ((s = NactiSlovo(ref tr, ref pozice, txtsoubor, ref byltexupraven)) != null)
             {
                    if (!vyskyty.ContainsKey(s))  //binarni prohledavani - O(log n)
                    {
                            al = new ArrayList();
                            al.Add(abspozice + (uint)pozice - (uint)s.Length);
                           vyskyty.Add(s, al);
                    }
                    else
                    {
                           al = (ArrayList)vyskyty[s];
                           al.Add(abspozice + (uint)pozice - (uint)s.Length);
                           vyskyty[s] = al;
                   }
                  if ((vyskyty.Count % 100 == 0) && (posledniPocet != vyskyty.Count))
                  {
                           textBoxSlov.Text = vyskyty.Count.ToString();
                           progressBarSoubor.Value = pozice / 1024;
                           progressBarCelkem.Value = (int)(abspozice + pozice) / 1024;
                           textBoxUplynulyCas.Text = (DateTime.Now - pocatecniCas).ToString();
                           posledniPocet = vyskyty.Count;
                           Application.DoEvents();
                   }
              }
              abspozice+=(uint)pozice;
     }
}
textBoxSlov.Text = vyskyty.Count.ToString();
nacitaciCas = DateTime.Now;
textBoxPrubeh.Text = "Sorting words";
Application.DoEvents();
serazenaslova = new ArrayList(vyskyty.Keys);
serazenaslova.Sort();                          //serazeni slov
tridiciCas = DateTime.Now;
```

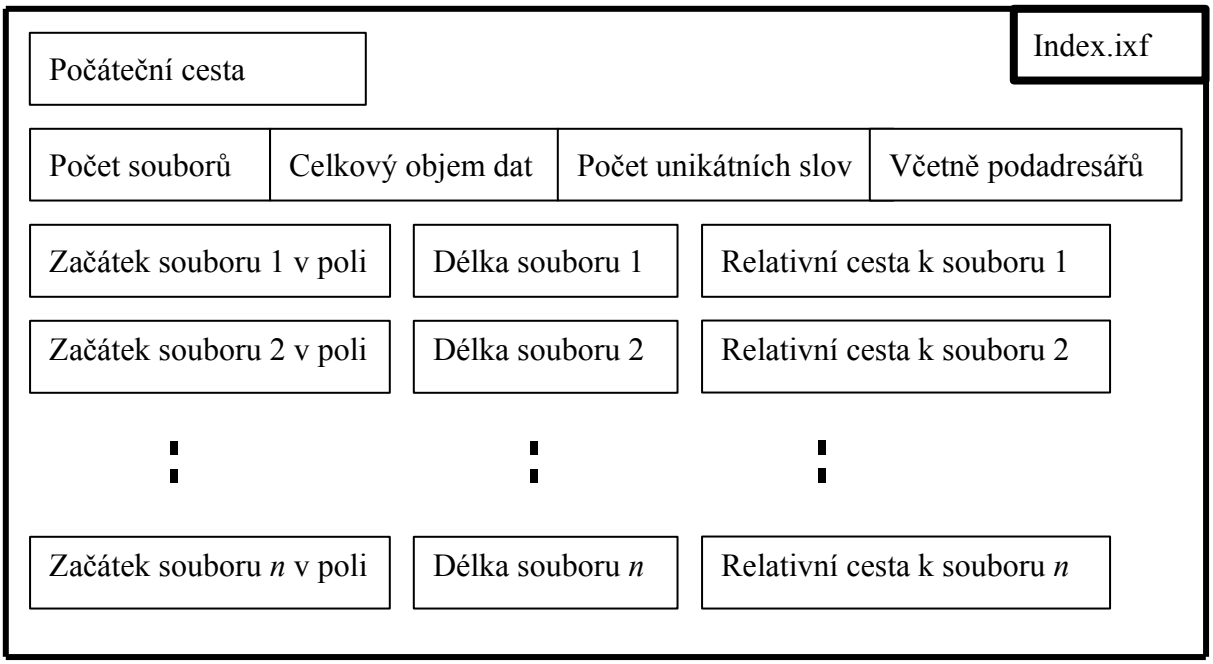
Komentář k implementaci:

Celá aplikace je tvořena jedním vláknem, zprávy zasílané operačním systémem jsou zpracovány vždy, když je načteno sté unikátní slovo. Tento způsob časování by mohl být příliš „řídký“ u vstupních dat se stále se opakujícími slovy, například u logů nějakých programů.

3.7 Uložení dat na disk

Po skončení indexace vstupních dat je třeba výstupní data uložit na disk pro další využití. Data uložíme do tří souborů, v prvním z nich (s příponou .ixf) budou informace o souborech, které byly indexovány. Obsah tohoto souboru je schématicky znázorněn na obrázku 3. **Počáteční cesta** je cesta, kterou zadal uživatel, tedy adresář, ve kterém jsme hledali soubory k indexaci. **Počet souborů** je počet vstupních souborů, **celkový objem dat** pak jejich velikost. **Počet unikátních slov** je počet slov, která byla indexována a položka **včetně podadresářů** udává, zda jsme vstupní soubory hledali i v podadresářích zadané cesty. Dále následují informace o jednotlivých vstupních souborech. **Začátek souboru v poli** udává polohu prvního znaku souboru

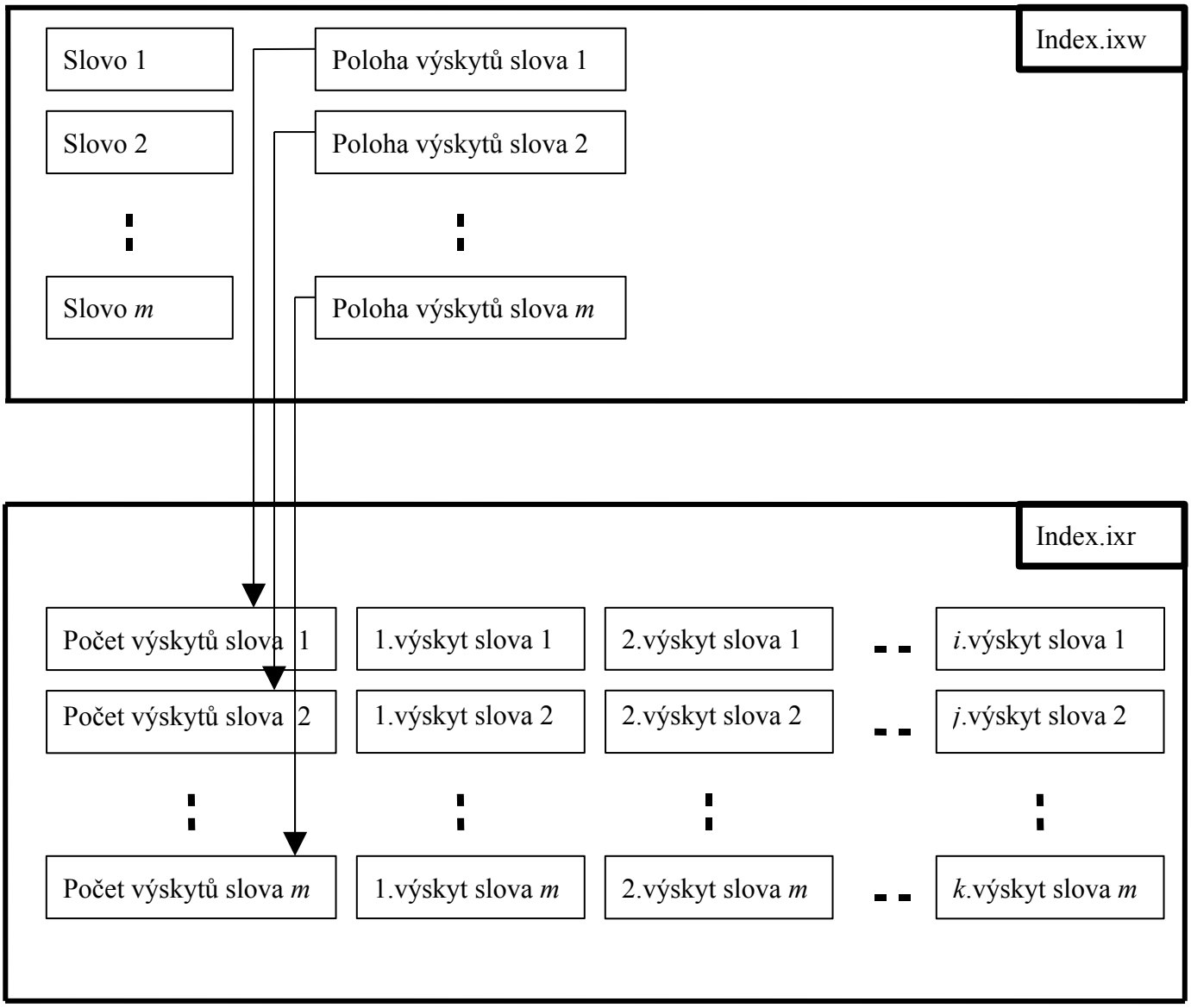
v imaginárním poli na obrázku 1. **Délka souboru** je velikost souboru v bytech. **Relativní cesta k souboru** je cesta k souboru, která spolu s **počáteční cestou** tvoří celkovou cestu k souboru; je v ní uloženo jméno souboru a případně podadresáře nacházející se „mezi“ souborem a počáteční cestou.



Obrázek 3: Způsob uložení informací o souborech

Zbývá uložit informace o nalezených slovech a jejich indexech. K tomu použijeme zbylé dva soubory. Jak již bylo uvedeno v oddílu 3.6, nejprve setřídíme klíče hashovací tabulky (nalezená slova) a na disk je pak ukládáme v abecedním pořadí.

Do druhého souboru (s příponou .ixw) uložíme ke každému slovu dvě hodnoty – slovo samotné a číslo vyjadřující na kolikáté pozici je ve třetím souboru (s příponou .ixr) uloženo číslo udávající počet výskytů tohoto slova. Třetí soubor je tvořen pouze čísly (v binárním tvaru). Na pozici, která je pro každé slovo uložena ve druhém souboru, je číslo, které udává, kolikrát se slovo ve vstupních datech vyskytlo. Následující čísla představují indexy slova, neboli pozice výskytů slova v imaginárním poli na obrázku 1. Těchto následujících čísel je tolik, kolik je hodnota čísla, které udává počet výskytů slova. Po těchto číslech pak následuje počet výskytů slova, které je jako další v abecedě, následují jeho výskyty, potom počet výskytů dalšího slova, atd. Způsob uložení slov a jejich výskytů je znázorněn na obrázku 3.



Obrázek 4: Způsob uložení informací o slovech a jejich výskytech

O velikosti výstupních dat lze prohlásit, že je zhruba polovinou velikosti dat vstupních, viz. tabulka 1.

```
Vstupy:   Informace o souborech
          Hashovací tabulka

Výstupy:  Tři datové soubory

Implementace v C#:
```

```
textBoxPrubeh.Text = "Creating index files";
Application.DoEvents();
zapocalzapis = true;
```

```
slova = new ArrayList(serazenaslova.Count);
using (StruktBinaryWriter sbw =new StruktBinaryWriter(new
FileStream(PomFunkce.NormujCestu(textBoxCesta.Text)+
                                                        textBoxJmeno.Text + ".ixw",
FileMode.Create)))
{
    using (BinaryWriter bw = new BinaryWriter(new FileStream(PomFunkce.NormujCestu(textBoxCesta.Text) +
                                                        textBoxJmeno.Text + ".ixr",
FileMode.Create)))
    {
        uint i = 0;
        foreach (string slovo in serazenaslova)   //zapis informaci o slovech a jejich vyskytech na disk
        {
             pomslovo.slovo=slovo;
             pomslovo.poloha = i;
             slova.Add(pomslovo);                 //ulozeni do pameti k pripadnemu pozdejsimu vyuziti
             sbw.Write(pomslovo);                 //zapis informaci o slovu na disk
             al = (ArrayList)vyskyty[slovo];
             bw.Write(al.Count);                  //pocet vyskytu slova
             i++;
             foreach (uint p in al)
             {
                 bw.Write(p);                     //jednotlive vyskyty slova
                 i++;
             }
        }
    }
}
using (StreamWriter sw = File.CreateText(PomFunkce.NormujCestu(textBoxCesta.Text)+ textBoxJmeno.Text+
".ixf"))
{
    sw.WriteLine(cesta);                           //zapis informaci o souborech na disk
    sw.WriteLine("{0:X} {1:X} {2:X}
{3:X}",soubory.Count,velikost,vyskyty.Count,checkBoxPodadresare.Checked);
    foreach (Soubor pomsoub in soubory)
    {
             sw.WriteLine("{0:X} {1:X} {2:X}", pomsoub.zacatek, pomsoub.delka, pomsoub.relcesta);
    }
}
Index pomindex = new Index(cesta, PomFunkce.NormujCestu(textBoxCesta.Text) + textBoxJmeno.Text + ".ixr",
                                                    soubory, slova, velikost,
checkBoxPodadresare.Checked);
fHO.indexy.Add(pomindex);
fHO.aktualniindex = fHO.indexy.Count - 1;
progressBarSoubor.Value = progressBarSoubor.Maximum;
progressBarCelkem.Value = progressBarCelkem.Maximum;
textBoxPrubeh.Text = "Indexing Complete";
```

Komentář k implementaci:

Slova i jejich výskyty jsou do souborů zapisována v cyklu přes abecedně seřazená slova.

3.8 Převod dat pro další využití

Po skončení indexace a zápisu dat na disk je ještě možné data v paměti převést do tvaru, který bude využitelný pro práci s indexem. Alternativou by bylo data pro práci načíst z právě zapsaných souborů, ale to by bylo pomalejší.

Vstupy: Hashovací tabulka

Výstupy: Seznam slov

Kapitola 4 – Práce s indexy

V této kapitole se zaměříme na zpracování dotazů uživatele a třídění výsledků

4.1 Načtení indexů slov ze souborů

Potom, co byly vytvořeny datové soubory, jsou data potřebná pro práci s indexy uložena v paměti. Po opětovném spuštění programu je již nutné data načíst z datových souborů. Ze tří datových souborů využijeme v této fázi jen první dva – z prvního získáme informace o souborech, které byly indexovány a z druhého seznam indexovaných slov.

```
Vstupy:    Název indexu

Výstupy:   Seznam souborů
           Seznam slov

Implementace v C#:
```

```
/// <summary>
/// Nacte index daneho jmena z datovych souboru
/// </summary>
/// <param name="jmeno"> jmeno indexu</param>
/// <returns> true pokud byly uspesne nacteny informace o alespon jednom vstupnim souboru</returns>
public bool NactiIndexZeSouboru(string jmeno)
{
     Index pomindex=new Index();
     try
     {
          if (listViewIndexy.Items.Count > 0)
          {
               int i = 0;
               foreach (ListViewItem pomitem in listViewIndexy.Items)
               {
                   if (jmeno == pomitem.SubItems[2].Text + "\\" + pomitem.SubItems[1].Text + ".ixf")
                   {
                       indexy.RemoveAt(i);
                       listViewIndexy.Items.RemoveAt(i);
                   }
                   i++;
               }
          }
          int souboru;
          int unikatnichslov;
          string pomstring;
          string[] polstr;
          Soubor pomsoubor;
          ArrayList pomal = new ArrayList();
          using (StreamReader sr = File.OpenText(jmeno))
          {
               pomindex.cesta = sr.ReadLine();
               pomindex.souborvyskytu = jmeno.Remove(jmeno.Length - 1, 1) + "r";
               pomstring = sr.ReadLine();
               polstr = pomstring.Split(null, 4);
               souboru = System.Convert.ToInt32(polstr[0], 16);
```

```
                pomindex.velikost = System.Convert.ToUInt32(polstr[1], 16);
                unikatnichslov = System.Convert.ToInt32(polstr[2], 16);
                pomindex.podadresare = System.Convert.ToBoolean(polstr[3]);
                pomindex.soubory = new ArrayList(souboru);
                pomindex.slova = new ArrayList(unikatnichslov);
                pomindex.vyskyty = new Hashtable();
                while (sr.Peek() >= 0)
                {
                    pomstring = sr.ReadLine();
                    polstr = pomstring.Split(null, 3);
                    pomsoubor.zacatek = System.Convert.ToUInt32(polstr[0], 16);
                    pomsoubor.delka = System.Convert.ToUInt32(polstr[1], 16);
                    pomsoubor.relcesta = polstr[2];
                    pomindex.soubory.Add(pomsoubor);
                }
            }
            using (StruktBinaryReader sr = new StruktBinaryReader(new
                                    FileStream(jmeno.Remove(jmeno.Length - 1, 1) + "w",
FileMode.Open)))
            {
                Slovo pomslovo;
                while (sr.BaseStream.Position < sr.BaseStream.Length)
                {
                    pomslovo = sr.ReadSlovo();
                    pomindex.slova.Add(pomslovo);
                }
            }
            indexy.Add(pomindex);
            aktualniindex = indexy.Count - 1;
            ListViewItem item = new ListViewItem(" - ");
            item.SubItems.Add(System.IO.Path.GetFileNameWithoutExtension(jmeno));
            item.SubItems.Add(System.IO.Path.GetDirectoryName(jmeno));
            item.Selected = true;
            listViewIndexy.Items.Add(item);
            listViewIndexy.Select();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Unable to add index. \n" + ex.ToString());
            return false;
        }
        return (pomindex.soubory.Count!=0);
}
```

Komentář k implementaci:

Při načítání většího indexu nereaguje aplikace na zprávy zasílané operačním systémem, bylo by proto vhodné zprávy zpracovávat během načítání.

4.2 Způsob uložení indexů v paměti

Pro reprezentaci indexu v paměti byla zavedena zvláštní třída, která se může nacházet v dynamické datové struktuře, což uživateli umožňuje pracovat s více indexy najednou. Tato třída obsahuje neměnné seznamy indexovaných souborů a slov a také hashovací tabulku, která slouží k uložení informací o výskytech slov a která se s počtem uživatelem vyhledaných slov zvětšuje.

Tato hashovací tabulka strukturou odpovídá té, kterou jsme používali pro vytvoření indexu. Tabulku, která by obsahovala všechna slova a jejich výskyty, by šlo z druhého a třetího datového souboru sice znovu vytvořit, ale použití této tabulky by znamenalo enormní nároky na

paměť a rychlost vyhledávání by se zvýšila jen zanedbatelně. Stejně tak by bylo možné nepoužívat hashovací tabulku pro výskyty slov vůbec a potřebná data vždy načíst z datového souboru.

4.3 Vyhledání výskytů slova

Nejzákladnějším požadavkem uživatele je vyhledání výskytů nějakého slova ve vstupních datech. Po zadání hledaného slova se nejprve ověří, zda se slovo nachází na seznamu indexovaných slov. Pokud se tam opravdu nachází, tak je u něj číslo, udávající pozici, na které jsou ve třetím datovém souboru data, týkající se tohoto slova. Následuje načtení čísla na této pozici. Toto číslo nám říká, kolikrát se slovo ve vstupních datech vyskytlo a tedy kolik následujících čísel (přesných pozic výskytů slova ve vstupních datech) ještě načteme. Takto získané pozice výskytů slova uložíme do hashovací tabulky, která je součástí každého používaného indexu, a zároveň nalezené výskyty zobrazíme.

Jedním ze způsobů, jak zúžit prostor vyhledávání, je podmínka na výskyt jiného slova v souboru. Zobrazíme tedy pouze ty výskyty slova, které se nacházejí v souborech, které obsahují jiné uživatelem zadané slovo. Postup je shodný jako v předešlém případě, s tím rozdílem, že s nalezením všech výskytů slova naše práce ještě nekončí. Nalezneme totiž ještě výskyty druhého slova a z nich určíme soubory, ve kterých se toto slovo vyskytuje. Potom zbývá již jen pro každý výskyt hledaného slova ověřit, zda se nalézá v některém ze souborů, které obsahují druhé slovo a výskyty splňující tuto podmínku zobrazit.

Dalším užitečným omezením je omezení na blízkost jiného slova. Opět postupujeme jako v prvním případě a potom pro každý výskyt prohledáme okolí slova ve vstupních datech, zda druhé zadané slovo obsahuje. Poznamenejme, že k tomuto typu omezení je již zapotřebí mít k dispozici původní vstupní data.

Obě výše uvedená omezení je možno uplatnit současně, tedy zadáním tří slov A, B a C nalezneme slova A, vyskytující se v blízkosti slova C v souborech obsahujících slovo B.

```
Vstupy:   Hledané slovo
          (Slovo, které musí být ve stejném souboru)
          (Slovo, které musí být v blízkosti hledaného slova)

Výstupy:  Výskyty slova

Implementace v C#:
```

```
/// <summary>
/// Vrati index souboru ve kterem je pozice
/// </summary>
/// <param name="pozice"> pozice slova v imaginarnim poli</param>
/// <param name="dolni"> dolni hranice intervalu na kterem hledame</param>
/// <param name="horni"> horni hranice intervalu na kterem hledame</param>
/// <returns> cislo souboru ve kterem je hledana pozice</returns>
public int NajdiSouborzPozice(uint pozice, int dolni, int horni)
{
    ArrayList soubory = ((Index)indexy[aktualniindex]).soubory;
```

```
    if (dolni == horni) return dolni;
    if (horni - dolni == 1)               //osetreni dvou sousedicich slov
        if (pozice >= ((Soubor)soubory[horni]).zacatek)
            return horni;
        else return dolni;
    int i = dolni + (horni - dolni) / 2;
    uint zacatek = ((Soubor)soubory[i]).zacatek;
    if (pozice < zacatek)
        return NajdiSouborzPozice(pozice, dolni, i);
    else
    {
        uint delka = ((Soubor)soubory[i]).delka;
        if (pozice < zacatek + delka)
            return i;
        else
            return NajdiSouborzPozice(pozice, i, horni);
    }
}


/// <summary>
/// Nacte prefixy, sufix, slovo a sirsi okoli slova pro vyskyty splnujici podminky
/// 1. podminka: vyskyt musi byt v nekterem ze souboru tretiho parametru
/// 2. podminka: vyskyt musi mit zacatek sufixu se ctvrtym parametrem
/// 3. podminka: vyskyt musi byt blizko jineho slova
/// </summary>
/// <param name="pomindex"> index ve kterem budeme hledat</param>
/// <param name="alpozic"> pozice vyskytu slova</param>
/// <param name="soub"> soubory ve kterych budeme hledat - 1. podminka</param>
/// <param name="zacateksufixu"> jaky tvar musi mit sufix - 2. podminka</param>
private void NactiPouzitiDynamicky(Index pomindex, ArrayList pomal, ArrayList soub, string zacateksufixu)
{
    ListViewItem item;
    List<ListViewItem> listitemu = new List<ListViewItem>();
    int aktcislosoub = -1;
    Soubor pomsoub = new Soubor();
    byte[] pompole = new byte[delkaprefixu + vybraneslovo.slovo.Length + delkasufixu];
    char[] pomslovo = new char[delkaprefixu];
    char[] pomsufix = new char[delkasufixu];
    string prvniprefix = null;
    string druhyprefix = null;
    string sufix = null;
    FileStream fs = null;
    int precteno = 0;       //kolik znaku ze streamu bylo precteno
    foreach (uint pozice in pomal)
    {
        int cislosoub = NajdiSouborzPozice(pozice, 0, pomindex.soubory.Count - 1);
        if (soub.Contains(cislosoub) || (soub.Count == 0))
        {
            if (cislosoub != aktcislosoub)
            {
                aktcislosoub = cislosoub;
                pomsoub = (Soubor)pomindex.soubory[cislosoub];
                fs = new FileStream(pomindex.cesta + pomsoub.relcesta, FileMode.Open, FileAccess.Read);
                precteno = 0;
            }
            int relpozice = (int)(pozice - pomsoub.zacatek);            //pozice prvniho znaku slova v souboru
            int offset = relpozice - delkaprefixu > 0 ? relpozice - delkaprefixu : 0;   //od kolikateho znaku
                                                                                        v souboru zacneme cist
            int delkasuf = relpozice + delkasufixu < pomsoub.delka ? delkasufixu : ((int)pomsoub.delka -
                                                                                    relpozice - vybraneslovo.slovo.Length);
            item = new ListViewItem(cislosoub.ToString());
            pomslovo = new char[delkaprefixu];
            if (precteno <= offset)     //je treba cast souboru preskocit nez zacneme cist do pole
            {
                fs.Seek(offset, SeekOrigin.Begin);
                precteno = offset;
                fs.Read(pompole, 0, relpozice - offset + vybraneslovo.slovo.Length + delkasuf);
                precteno += relpozice - offset + vybraneslovo.slovo.Length + delkasuf;
            }
            else
```

```
            {
                if (precteno >= pompole.Length)    //pole je jiz cele zaplnene -> je treba provest posun vlevo
                    Array.Copy(pompole, pompole.Length - (precteno - offset), pompole, 0, precteno - offset);
                fs.Read(pompole, precteno - offset, pompole.Length - (precteno - offset)); //docteni znaku do
                                                                                            konce pole
                precteno += pompole.Length - (precteno - offset);
            }
            int i = offset == 0 ? relpozice - 1 : delkaprefixu - 1;
            int k = 0;      //nacteni sufixu
            int zacateksufixuvpoli = relpozice - offset + vybraneslovo.slovo.Length;
            for (int j = zacateksufixuvpoli; j < zacateksufixuvpoli + delkasuf; j++)
            {
                if (pompole[j] != '\r')
                    if (pompole[j] != '\n')
                        pomsufix[k++] = (char)pompole[j];
                    else
                        pomsufix[k++] = ' ';
            }
            if (k > 0) pomsufix[k - 1] = '\0';
            sufix = new string(pomsufix);
            if (((zacateksufixu==null) || (sufix.StartsWith(zacateksufixu)))&& ((comboBoxBlizko.Text == "") ||
                            (System.Text.Encoding.Default.GetString(pompole).Contains(comboBoxBlizko.Text))))
            {
                k = 0;  //nacteni prvniho prefixu
                while ((i >= 0) && ((pompole[i] == ' ') || (pompole[i] == '\n') || (pompole[i] == '\r'))) i--;
                while ((i >= 0) && ((pompole[i] != ' ') && (pompole[i] != '\n') && (pompole[i] != '\r')))
                {
                    pomslovo[k++] = (char)pompole[i--];
                }
                if (pomslovo[0] == '\0')
                    prvniprefix = " - ";
                else
                {
                    Array.Reverse(pomslovo, 0, k);
                    prvniprefix = new string(pomslovo);
                }
                i--;
                k = 0;      //nacteni druheho prefixu
                pomslovo = new char[delkaprefixu];
                while ((i >= 0) && ((pompole[i] == ' ') || (pompole[i] == '\n') || (pompole[i] == '\r'))) i--;
                while ((i >= 0) && ((pompole[i] != ' ') && (pompole[i] != '\n') && (pompole[i] != '\r')))
                {
                    pomslovo[k++] = (char)pompole[i--];
                }
                if (pomslovo[0] == '\0')
                    druhyprefix = " - ";
                else
                {
                    Array.Reverse(pomslovo, 0, k);
                    druhyprefix = new string(pomslovo);
                }
                k = 0;      //nacteni slova
                pomslovo = new char[vybraneslovo.slovo.Length];
                for (int j = relpozice - offset; j < relpozice - offset + vybraneslovo.slovo.Length; j++)
                {
                    pomslovo[k++] = (char)pompole[j];
                }
                item.SubItems.Add(druhyprefix);
                item.SubItems.Add(prvniprefix);
                item.SubItems.Add(new string(pomslovo));
                item.SubItems.Add(sufix);
                item.SubItems.Add((String.Copy(pozice.ToString())));
                listitemu.Add(item);
            }
        }
    }
    if (fs!=null) fs.Close();
    itemyListViewPouziti = listitemu.ToArray();
    listViewPouziti.VirtualListSize = itemyListViewPouziti.Length;
}
```

```
private void buttonHledani_Click(object sender, System.EventArgs e)
{
   if (!comboBoxHledani.Items.Contains(comboBoxHledani.Text)) comboBoxHledani.Items.Add(comboBoxHledani.Text);
   if (!comboBoxObsahuje.Items.Contains(comboBoxObsahuje.Text))
                                                        comboBoxObsahuje.Items.Add(comboBoxHledani.Text);
   if (!comboBoxBlizko.Items.Contains(comboBoxBlizko.Text)) comboBoxBlizko.Items.Add(comboBoxHledani.Text);
   if (listBoxSlova.Items.Count > 0)
   {
        string[] vstup = comboBoxHledani.Text.Split(null,2);
        if (listBoxSlova.Items.Contains(vstup[0]))
        {
            Index pomindex = (Index)indexy[aktualniindex];
            itemyListViewPouziti = null;
            listViewPouziti.VirtualListSize = 0;
            if (!pomindex.vyskyty.Contains(vybraneslovo.slovo)) NactiVyskytyZeSouboru(pomindex, vybraneslovo);
            ArrayList pomal =(ArrayList) pomindex.vyskyty[vybraneslovo.slovo];
            ArrayList soub = new ArrayList();
            if (VyberSoubory(soub)) try
            {
                if ((vstup.Length == 1) && (comboBoxObsahuje.Text == "") && (comboBoxBlizko.Text == ""))
                    NactiPouziti(pomindex, pomal, soub);
                else
                {
                    if (vstup.Length == 1)
                        NactiPouzitiDynamicky(pomindex, pomal, soub, null);
                    else
                        NactiPouzitiDynamicky(pomindex, pomal, soub, " " + vstup[1]);
                }
                listViewPouziti.Invalidate();
                listViewPouziti.Select();
                if (listViewPouziti.Items.Count > 0)
                {
                    listViewPouziti.Items[0].Selected = false;
                    listViewPouziti.Items[0].Selected = true;
                }
            }
            catch (System.IO.FileNotFoundException ex)
            {
                MessageBox.Show(ex.Message);
            }
            textBoxVysledky.Text = listViewPouziti.Items.Count.ToString();
            foreach (ColumnHeader c in listViewPouziti.Columns)
            {
                c.ImageKey = null;
            }
            listViewPouziti.Columns[0].ImageIndex = 0;
        }
   }
}
```

Komentář k implementaci:

Pozice každého výskytu ve vstupních datech je uložena nikoliv jako číslo, ale jako řetězec znaků jako SubItem třídy ListViewItem, což znamená zjednodušení kódu za cenu efektivity. Alternativou by bylo pozice ukládat do zvláštní datové struktury, ale problém by byl v tom, že by tato struktura musela reflektovat změny pořadí prvků v důsledku třídění.

Zde uvedená funkce pro načítání je dynamická, protože dopředu nevíme, kolik výskytů splní omezující podmínky. Pro situace, kdy žádná omezení nebyla zadána a budou tedy zobrazeny všechny výskyty hledaného slova, je v programu implementována i statická verze této funkce, která je rychlejší, ale je třeba dopředu znát počet výskytů.

4.4 Zobrazení nalezených výskytů

Výskyty zobrazíme spolu s jejich bezprostředním okolím, které se skládá z prefixů a sufixu. Slovo předcházející hledanému slovu označíme jako první prefix, slovo předcházející prvnímu prefixu jako druhý prefix. Slova, která následují po hledaném slovu označíme jako sufix. Zobrazení může vypadat třeba tak, jako v příloze 2.

Uživatel má tedy přehled o bezprostředním okolí výskytů slov. Vybráním jednoho výskytu, se mu zobrazí slovo, zvýrazněné v části vstupních dat a také jméno a cesta k souboru, ve kterém se daný výskyt nachází.

K zobrazení okolí slova je již třeba pracovat s původními vstupními daty. Při vytváření položek seznamu se ze vstupních dat načtou nejenom prefixy a sufix, ale znovu i slovo samotné, protože při indexaci se nehledělo na velká a malá písmena.

Vstupy: Výskyty slova

Výstupy: Prefixy a sufix pro každý výskyt

Implementace v C#:

```
if (listViewPouziti.SelectedIndices.Count == 1)
{
    Index pomindex = (Index)indexy[aktualniindex];
    Soubor pomsoub =
(Soubor)pomindex.soubory[System.Convert.ToInt32(listViewPouziti.Items[listViewPouziti.
                                                                    SelectedIndices[0]].SubItems[0].Text)
];
    uint pozice = System.Convert.ToUInt32(listViewPouziti.Items[listViewPouziti.SelectedIndices[0]].
                                                                                SubItems[5].Text
);
    int relpozice = (int)(pozice - pomsoub.zacatek);
    int offset = relpozice - pocetznakupred > 0 ? relpozice - pocetznakupred : 0;
    int delka = relpozice + pocetznakuza < pomsoub.delka ? pocetznakuza : ((int)pomsoub.delka -
relpozice);
    char[] charpole = new char[pocetznakupred + pocetznakuza];
    if (pomsoub.relcesta.EndsWith(".tex"))
    {
        formNovyIndex.NactiStopWords(true);
        StreamReader sr = new StreamReader(pomindex.cesta + pomsoub.relcesta,
System.Text.Encoding.Default,
                                                                                            true
);
        string pomstr = formNovyIndex.UpravTex((TextReader)sr).Substring(offset, relpozice - offset+
delka);
        richTextBoxSoubor.Text = pomstr;
        charpole = pomstr.ToCharArray();
    }
    else
    {
        byte[] pompole = new byte[pocetznakupred + pocetznakuza];
        using (FileStream fs = new FileStream(pomindex.cesta + pomsoub.relcesta, FileMode.Open,
                                                                                FileAccess.Read
))
        {
            fs.Seek(offset, SeekOrigin.Begin);
            fs.Read(pompole, 0, relpozice - offset + delka);
        }
        for (int i = 0; i < pompole.Length; i++)
            charpole[i] = (char)pompole[i];
        richTextBoxSoubor.Text = new string(charpole);
```

```
    }
    int radku = 0;
    int zacatekoznaceni = relpozice - pocetznakupred > 0 ? pocetznakupred : relpozice;
    bool obsahujecarriagereturn = false;
    for (int i = 0; i < zacatekoznaceni; i++)
    {
        if (charpole[i] == '\n') radku++;
        if (charpole[i] == '\r') obsahujecarriagereturn=true;
    }
    if ((charpole[0] == '\n')&&(obsahujecarriagereturn)) radku--;    //osetreni stavu kdy pri souboru s
                                                                     koncem radku CRLF je prvni znak textboxu
LF
    if (obsahujecarriagereturn) zacatekoznaceni -= radku;
    richTextBoxSoubor.SelectionStart = zacatekoznaceni;
    richTextBoxSoubor.SelectionLength = listViewPouziti.Items[listViewPouziti.SelectedIndices[0]].
                                                                                SubItems[3].Text.Length
;
    statusBarHokno.Text = pomindex.cesta + pomsoub.relcesta;
}
```

Komentář k implementaci:

Chceme-li v RichTextBoxu označit hledané slovo, je třeba spočítat na kolikátém řádku se v něm slovo nachází a tímto údajem upravit polohu začátku označení.

Soubory vytvořené programem LaTeX projdou před zobrazením stejnou úpravou, kterou prošly při vytváření indexu.

4.5 Třídění nalezených výskytů

Po zobrazení jdou jednotlivé výskyty za sebou tak, jako je tomu v datovém souboru, tj. v tom pořadí, ve kterém se vyskytly ve vstupních datech. Třídit lze podle prefixů, sufixu, i podle samotného slova, což má smysl v případě, kdy se výskyty slova od sebe liší velkými a malými písmeny.

Po setřídění například podle prvního prefixu, se často dostaneme do situace (zvlášť u slov s velkým počtem výskytů), kdy ve výpisu za sebou následuje mnoho stejných prefixů. To je v pořádku, pokud již víme, který prefix nás zajímá, ale pokud se nejprve chceme podívat, jaké prefixy se vyskytly, můžeme ponechat první vyskytnuvší se prefix a jeho další výskyty přesunout na konec seznamu. Tak získáme snadno přehled o všech unikátních prefixech.

Vstupy: Výskyty slova s prefixy a sufixem

Výstupy: Vstup setříděný podle nějakého kriteria

Implementace v C#:

```
private void listViewPouziti_ColumnClick(object sender, ColumnClickEventArgs e)
{
    foreach (ColumnHeader c in listViewPouziti.Columns)
    {
        if (c.Index!=e.Column) c.ImageKey = null;
    }
```

```
    if (listViewPouziti.Columns[e.Column].ImageIndex == 0)
    {
        Array.Sort(itemyListViewPouziti, new ListViewItemSorterDolu(e.Column,
                                                                    StringComparison.CurrentCulture))
;
        listViewPouziti.Columns[e.Column].ImageIndex = 1;
    }
    else
    {
        Array.Sort(itemyListViewPouziti, new ListViewItemSorterNahoru(e.Column,
                                                                    StringComparison.CurrentCulture))
;
        listViewPouziti.Columns[e.Column].ImageIndex = 0;
    }
    listViewPouziti.Invalidate();
}

private void buttonPrvniPrefix_Click(object sender, EventArgs e)
{
    if (listViewPouziti.Items.Count > 2)
        {
            ListViewItem item;
            List<ListViewItem> items = new List<ListViewItem>();
            List<ListViewItem> uitems = new List<ListViewItem>(itemyListViewPouziti);
            int hranice = itemyListViewPouziti.Length;
            for (int i = 1; i < hranice; )
            {
                if (uitems[i].SubItems[2].Text == uitems[i - 1].SubItems[2].Text)
                {
                    item = uitems[i];
                    uitems.RemoveAt(i);
                    items.Add(item);
                    hranice--;
                }
                else
                    i++;
            }
                uitems.AddRange(items);
                itemyListViewPouziti = uitems.ToArray();
                listViewPouziti.Invalidate();
        }
}
```

Komentář k implementaci:

Po setřídění je směr třídění indikován ikonkou v záhlaví sloupce.

4.6 Přehled souborů a slov

Během práce s indexem je můžeme zobrazit informace o souborech, které tvořily jeho vstupní data. Stejně tak je možné zobrazit i přehled všech slov. Tento seznam se dá setřídit podle četnosti výskytu slova a zjistit tak nejvíce používaná slova.

Vstupy: Název indexu

Výstupy: Přehled souborů
Přehled slov

Kapitola 5 – Server–klient verze programu

V této kapitole se zamyslíme nad možnou úpravou programu na architekturu server-klient.

5.1 Motivace

Úprava programu na architekturu server-klient by se mohla na první pohled zdát v rozporu s v kapitole 1 vytýčenou oblastí působnosti programu. Nicméně pokud by byla skupina uživatelů, která by měla zájem o kontextové prohledávání nějakých specifických dat o velkém objemu, pak by stačilo provést indexaci na těchto datech jednou a takto získaná data dát k dispozici serveru. Uživatelé by potom mohli s daty pracovat na svých pracovních stanicích pomocí klientského programu. Pro příklad nemusíme chodit daleko – katedra by mohla jako vstupní data použít například diplomové práce studentů, či odborné články ostatních pracovníků.

Tedy oba hlavní rysy programu zůstávají zachovány – úplná kontrola nad vstupními daty (ovšem v tomto případě na straně správce serveru) a možnost třídit výsledky vyhledávání.

5.2 Předpoklady

Kromě samotné změny architektury by bylo třeba rozšířit funkčnost programu, zejména by měl umět zpracovat soubory ve formátu PDF (portable document format), což je formát v současnosti velice rozšířený. Také podpora písmen z jiných abeced než anglické by byla téměř nezbytností.

Klientský program by bylo vhodné implementovat jako webové rozhraní, tedy aby pro práci s indexem stačil internetový prohlížeč.

Kapitola 6 – Závěr

Program lze použít k rychlému vyhledání kontextu slova v textových souborech, nalezené výsledky je možné třídit. Rychlost je však vykoupena velikostí datových souborů, které mají obvykle polovinu objemu vstupních dat.

Výhledově by se program dal upravit z verze pro lokální počítač na webovou službu pro potřeby oboru. Díky technologii ASP.NET [9] by se této úpravy dalo docílit jen s malými změnami zdrojového kódu. Vznikla by tak pomůcka pro psaní vědeckých prací.

Ačkoliv je program napsán v jazyce C#, není ještě multiplatformní, neboť prostředí Mono[10] dosud nepodporuje všechny programem používané funkce. Výsledek testu přenositelnosti v programu MoMA [11] je v příloze C.

Použité zdroje

[1] Grep. Domovská stránka:
http://www.gnu.org/software/grep/doc

[2] Google Guide: How Google Works. Domovská stránka:
http://www.googleguide.com/google_works.html

[3] Google Desktop. Domovská stránka:
http://desktop.google.com/features.html

[4] British National Corpus. Domovská stránka:
http://www.natcorp.ox.ac.uk

[5] Český národní korpus. Domovská stránka:
http://ucnk.ff.cuni.cz

[6] LaTeX. Domovská stránka:
http://www.latex-project.org

[7] Project Gutenberg. Domovská stránka:
http://www.gutenberg.org/wiki/Main_Page

[8] MSDN Library for Visual Studio .NET 2005. Domovská stránka:
http://msdn1.microsoft.com/en-us/default.aspx

[9] ASP.NET. Domovská stránka:
http://www.asp.net

[10] Mono. Domovská stránka:
http://www.mono-project.com/Main_Page

[11] MoMA. Domovská stránka:
http://www.mono-project.com/Moma

[12] Gunnerson, Eric: Začínáme programovat v C#. Computer Press 2001

Použitý software

Microsoft Visual Studio 2003

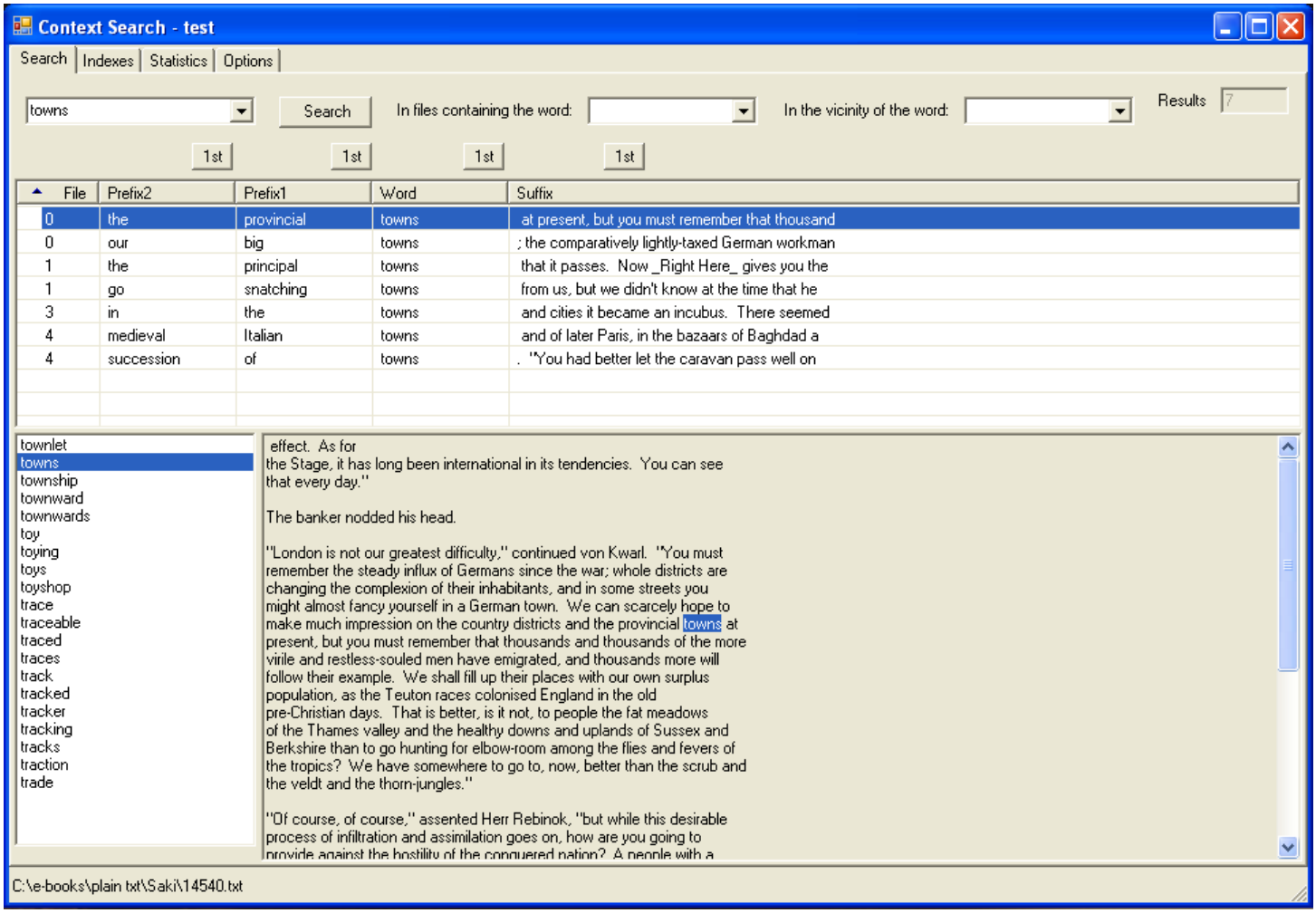
Microsoft Visual Studio 2005

Přílohy

Příloha A – 50 nejběžnějších anglických slov delších než dva znaky

the	some	their
and	what	time
you	there	will
that	can	way
was	out	about
for	other	many
are	were	then
with	all	them
his	your	would
they	when	write
one	use	like
have	word	these
this	how	her
from	said	long
had	each	make
hot	she	thing
but	which	

Příloha B – ukázka zobrazení nalezených výskytů slova



Příloha C – test přenositelnosti do prostředí Mono v programu MoMA

MoMA Scan Results

Scan time: 9.1.2007 18:25:58

For descriptions of issues and what to do, see http://www.mono-project.com/MoMA_-_Issue_Descriptions.

Context.exe

Methods missing from Mono

Calling Method	Method not yet in Mono
Class Context.FormNacitani:	
void InitializeComponent()	void Control.set_UseWaitCursor(bool)
Class Context.FormHlavniOkno:	
bool VyberSoubory(ArrayList)	void ListView/SelectedIndexCollection.Clear()
bool VyberSoubory(ArrayList)	int ListView/SelectedIndexCollection.Add(int)
void NactiPouziti(Index, ArrayList, ArrayList)	void ListView.set_VirtualListSize(int)
void NactiPouzitiDynamicky(Index, ArrayList, ArrayList, string)	void ListView.set_VirtualListSize(int)
void InitializeComponent()	void MainMenu..ctor(IContainer)
void InitializeComponent()	void ListView.set_VirtualMode(bool)
void InitializeComponent()	void ListView.add_RetrieveVirtualItem(RetrieveVirtualItemEventHandler)
void InitializeComponent()	void Control.set_ContextMenuStrip(ContextMenuStrip)
void InitializeComponent()	void ListView.set_VirtualMode(bool)
void InitializeComponent()	void ListView.add_RetrieveVirtualItem(RetrieveVirtualItemEventHandler)
void InitializeComponent()	void Form.add_Shown(EventHandler)
void buttonZobrazSoubory_Click(Object, EventArgs)	void ListView.set_VirtualListSize(int)
void buttonZobrazSlova_Click(Object, EventArgs)	void ListView.set_VirtualListSize(int)
void listViewIndexy_SelectedIndexChanged(Object, EventArgs)	void ListView.set_VirtualListSize(int)
void buttonHledani_Click(Object, EventArgs)	void ListView.set_VirtualListSize(int)
void buttonHledani_Click(Object, EventArgs)	void ColumnHeader.set_ImageKey(string)
void buttonHledani_Click(Object, EventArgs)	void ColumnHeader.set_ImageIndex(int)
void notifyIconHlavniOkno_DoubleClick(Object, EventArgs)	void Form.Show(IWin32Window)
void toolStripMenuItemOdstranit_Click(Object, EventArgs)	void ListView.set_VirtualListSize(int)
void listViewPouziti_ColumnClick(Object, ColumnClickEventArgs)	void ColumnHeader.set_ImageKey(string)
void listViewPouziti_ColumnClick(Object, ColumnClickEventArgs)	int ColumnHeader.get_ImageIndex()
void listViewPouziti_ColumnClick(Object, ColumnClickEventArgs)	void ColumnHeader.set_ImageIndex(int)
void listViewPouziti_ColumnClick(Object, ColumnClickEventArgs)	void ColumnHeader.set_ImageIndex(int)
void listViewStatistika_ColumnClick(Object, ColumnClickEventArgs)	void ColumnHeader.set_ImageKey(string)
void listViewStatistika_ColumnClick(Object, ColumnClickEventArgs)	int ColumnHeader.get_ImageIndex()
void listViewStatistika_ColumnClick(Object, ColumnClickEventArgs)	void ColumnHeader.set_ImageIndex(int)
void listViewStatistika_ColumnClick(Object, ColumnClickEventArgs)	void ColumnHeader.set_ImageIndex(int)

Methods called that throw NotImplementedException

Calling Method	Mono method that throws NotImplementedException
Class Context.FormNovyIndex:	
void buttonCesta_Click(Object, EventArgs)	void ManagementObject.Get()
void buttonCesta_Click(Object, EventArgs)	Object ManagementBaseObject.get_Item(string)
void buttonCesta_Click(Object, EventArgs)	Object ManagementBaseObject.get_Item(string)
Class Context.FormHlavniOkno:	
void buttonIndexyNovy_Click(Object, EventArgs)	void ManagementObject.Get()
void buttonIndexyNovy_Click(Object, EventArgs)	Object ManagementBaseObject.get_Item(string)

Methods called marked with [MonoTodo]

Calling Method	Method with [MonoTodo]	Reason
Class Context.FormNovyIndex:		
void buttonCesta_Click(Object, EventArgs)	void ManagementObject..ctor(string)	Not Specified
void buttonCesta_Click(Object, EventArgs)	void ManagementObject.Get()	Not Specified
void buttonCesta_Click(Object, EventArgs)	Object ManagementBaseObject.get_Item(string)	Not Specified
void buttonCesta_Click(Object, EventArgs)	Object ManagementBaseObject.get_Item(string)	Not Specified
Class Context.FormHlavniOkno:		
void InitializeComponent()	void StatusBar..ctor()	Change cursor when mouse is over grip
void buttonIndexyNovy_Click(Object, EventArgs)	void ManagementObject..ctor(string)	Not Specified
void buttonIndexyNovy_Click(Object, EventArgs)	void ManagementObject.Get()	Not Specified
void buttonIndexyNovy_Click(Object, EventArgs)	Object ManagementBaseObject.get_Item(string)	Not Specified

Příloha D – obsah přiloženého CD

CD obsahuje zdrojový kód programu v adresáři *kod*. Spustitelná verze programu se nachází v adresáři *release*.