VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

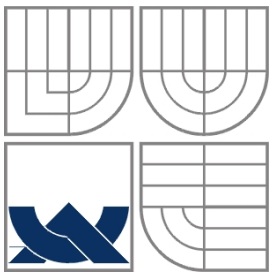
VYHĽADÁVANIE NAJČASTEJŠÍCH N-TÍC SLOV

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

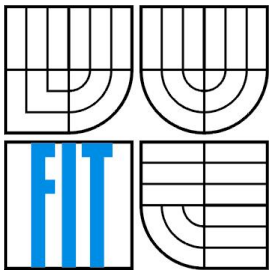
AUTOR PRÁCE
AUTHOR

Matúš Holec

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ NEJČASTĚJŠÍCH N-TIC SLOV

THE MOST FREQUENT WORD N-GRAMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATÚŠ HOLEC

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2009

Abstrakt

Tato práce se zabývá návrhem a implementací efektivního systému vyhledávání n-tic slov v textu. Systém je založen na principu dávkového zpracování, což umožňuje zpracování rozsáhlých textů. V první části práce jsou shrnuty principy stávajících metod sloužících pro extrakci n-gramů. V další části je popsán implementovaný systém a následně i jeho urychlení pomocí paralelizace dávkového zpracování. V závěru je uvedeno srovnání výkonnosti dostupných implementací s navrženým systémem, jakož i porovnání časové náročnosti sekvenčního přístupu s paralelním.

Abstract

This thesis deals with design and implementation of effective system for word n-grams extraction from texts. System is based on batch processing therefore it is able to process large text corpuses. The first part contains principles of existing methods for an n-gram extraction. The next part includes description of the implemented system as well as the approach of acceleration system by paralelizing the batch processing. The last part contains efficiency comparison between available implementations and designed system and time complexity comparison between sequential and paralelized approach.

Klíčová slova

zpracování přirozeného jazyka, extrakce n-gramů, dávkové zpracování, rozsáhlé textové korpusy

Keywords

natural language processing, n-gram extraction, batch processing, text corpuses of large size

Citace

Matúš Holec: Vyhľadávanie najčastejších n-tíc slov, bakalářská práce, Brno, FIT VUT v Brně, 2009

Vyhľadávanie najčastejších n-tíc slov

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Doc. RNDr. Pavla Smrža, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Matúš Holec
12.5.2009

Poďakovanie

Rád by som poďakoval svojmu vedúcemu, Doc. RNDr. Pavlovi Smržovi, Ph.D za hodnotné pripomienky a rady, ktoré mi poskytoval počas priebehu tvorby práce.

Obsah

1 Úvod

Spracovanie prirodzeného jazyka je obor na rozhraní lingvistiky a počítačovej vedy. Tento pojem rovnako symbolizuje spracovanie či už písaného, alebo hovoreného jazyka pre praktické a užitočné účely: preklad jazykov, získavanie znalostí z webových portálov, vedenie konverzácie s počítačom za účelom získania nových, zmysluplných informácii. V konečnom dôsledku ide o dosiahnutie lepšieho porozumenia prirodzeného jazyka počítačom [1].

Pri riešení úloh z oblasti počítačového spracovania prirodzeného jazyka sa na reprezentáciu textu využívajú vo väčšine prípadov slová. Do popredia sa však dostáva využívanie viacerých, po sebe nasledujúcich n-tíc slov, ktoré sa označujú ako n-gramy. Táto práca sa zaoberá systémom vyhľadávania najčastejších n-tíc slov v rozsiahlom texte. Jej cieľom je navrhnutie a popísanie efektívneho algoritmu, slúžiaceho na extrakciu n-tíc slov z textu, ako aj porovnanie s už existujúcimi postupmi, a možné zefektívnenie navrhnutého systému pomocou paralelizácie.

V súčasnej dobe existuje niekoľko metód, ktoré sa využívajú na extrakciu slovných n-gramov z textu. Ich najväčším nedostatkom je neschopnosť práce s textom, ktorého veľkosť presahuje veľkosť dostupnej operačnej pamäte. To znamená, že pri spracovaní rozsiahlych textov sú tieto algoritmy nepoužiteľné. Popisovaný systém je schopný pracovať s textovým súborom akejkoľvek veľkosti, čo je riešené postupným, dávkovým spracovaním vstupných dát. Systém dokáže spracovať ľubovoľne veľký textový súbor, bez ohľadu na výkon počítača.

Kapitola 2 sa zaoberá vymedzením pojmu n-gram, takisto aj zhrnutím sfér, v ktorých sa n-gramy používajú a zaradením využitia n-gramov v rámci spracovania prirodzeného jazyka.

V kapitole 3 sú zhrnuté princípy existujúcich metód pre vyhľadávanie n-gramov. Ďalej tu je popísané porovnanie dvoch najvyužívanejších metód a ich výpočtová zložitosť.

Obsahom kapitoly 4 je detailný popis implementovaného systému. Sú tu zhrnuté princípy, na ktorých algoritmus funguje, popisy jednotlivých častí a ich fáz, a prípadné vstupy a výstupy každej etapy.

Urýchlenie systému pomocou paralelizácie, ako aj princípy samotnej paralelizácie sú popísané v kapitole 5.

Výsledky testov, porovnanie paralelizovaného prístupu s klasickým prístupom sú zaznamenané v kapitole 6 vo forme grafov.

Táto práca bola vyvíjaná v rámci projektu KiWi - Znalosti ve Wiki, na ktorom participuje Výskumná skupina spracovania prirodzeného jazyka na Fakulte Informačných Technológií VUT v Brne. Pre implementáciu boli dostupné servery výskumnej skupiny, takisto aj rozsiahle textové korpusy, ktoré boli využívané ako množina testovacích dát.

2 N-gramy a ich využitie

2.1 Definícia n-gramu

N-gram je definovaný v [2] ako sled *n* po sebe idúcich položiek z danej sekvencie. Zo sémantického pohľadu môže byť táto postupnosť buď postupnosťou slov alebo písmen. V praxi sa častejšie stretávame s n-gramom ako sledom slov. Sled dvoch po sebe nasledujúcich položiek býva často označovaný ako bigram, pre sled troch položiek je zaužívaný pojem trigram. Od štyroch a vyššie sa používa označenie N-gram, kde N je nahradené počtom za sebou nasledujúcich elementov.

2.2 Využitie n-gramov

2.2.1 Všeobecné využitie n-gramov

N-gramy vo všeobecnosti nachádzajú svoje uplatnenie v širokej sfére zameraní. Uplatniť sa dokážu napríklad v oblasti teoretickej matematiky, biológie, kartografie ale aj v oblasti hudby. Ako je uvedené v [3], medzi najzaujímavejšie využitia n-gramov patrí:

- extrahovanie čŕt pre zhlukovanie sérií satelitných snímkov Zeme, a následné rozhodovanie, ktorú konkrétnu časť Zeme daný snímok zobrazuje,
- používajú sa pri vyhľadávaní genetickej sekvencie,
- na poli genetiky sú využívané pre určenie, ktorému konkrétneho živočíšnemu druhu odobraná vzorka DNA patrí,
- z počítačovej vedy sú uplatňované pri kompresii,
- pomocou n-gramov bývajú indexované dáta, súvisiace so zvukom.

2.2.2 Využitie n-gramov pre potreby spracovania prirodzeného jazyka

Širokú aplikáciu majú n-gramy v oblasti spracovania prirodzeného jazyka. Slúžia ako vstupný element pre riešenie obšírneho celku problémov.

Ako je uvedené v [4], vo sfére spracovania prirodzeného jazyka sa n-gramy využívajú hlavne na predikciu slov, kde sa využívajú pravdepodobnostné modely zvané N-gram modely. N-gram model počíta pravdepodobnosť výskytu posledného slova n-gramu z predchádzajúcich n-gramov. Pokiaľ je tento prístup použitý pre modelovanie jazyka, tak výskyt každého slova závisí len na

predchádzajúcich *n* slovách. N-gram model je tým pádom Markovský model, čo značne uľahčuje strojové získanie znalostí o modele jazyka z textu.

Ďalším využitím n-gramov je odhaľovanie plagiátov, a to rozdelením textu na viacero malých fragmentov, reprezentovaných n-gramami, tie sa dajú ľahko medzi sebou porovnať, a tým odvodiť mieru podobnosti kontrolovaných dokumentov. Bližšie sa tejto problematike venuje [5]. N-gramy sú často úspešne využívané pre kategorizáciu textu alebo jazyka. Ako je spomenuté v [3], n-gramy bývajú využívané pre vytváranie funkcií, ktoré umožňujú algoritmom strojového učenia získavať znalosti z textových dát. Pomocou n-gramov je takisto možné efektívne vyhľadanie kandidátov pre náhradu za pravopisne chybne napísané slovo.

2.2.3 Výskumné projekty Google

Podľa [6] vo výskumných centrách Google boli využívané n-gram modely pre širokú paletu výskumných a vývojárskych projektov. Medzi tieto projekty patrí napríklad štatistický preklad z jedného jazyka do druhého, rozpoznávanie reči, oprava pravopisných chýb, extrakcia informácií, detekcia entít a mnoho ďalších. Pre účely týchto projektov boli použité textové korpusy s obsahom niekoľko biliónov slov.

Spoločnosť Google sa rozhodla svoje tréningové korpusy nahromadiť a spojiť. Z toho vznikol tzv. Google teracorpus, ktorý obsahuje 1 024 908 267 229 slov zozbieraných z verejných webových stránok. Pre tento teracorpus bola takisto zverejnená početnosť všetkých piatich po sebe nasledujúcich slov v texte, 5-gramov. Google distribuuje teracorpus prostredníctvom Linguistic Data Consortium za istý poplatok. Tým pádom nie je teracorpus dostupný úplne každému, a ani v tejto práci som s ním nemal možnosť experimentovať.

3 Metódy na extrakciu n-gramov

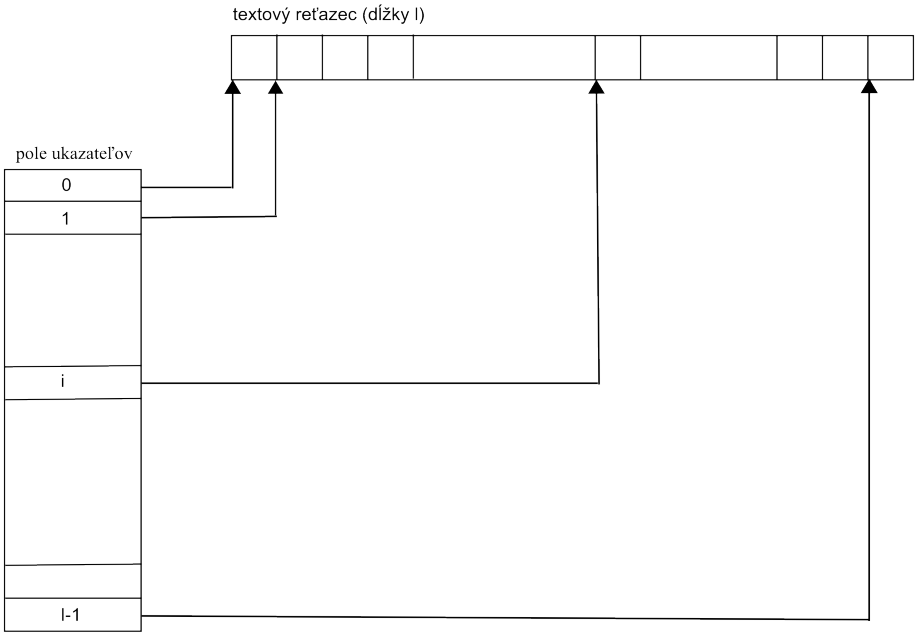
Kvôli častému využívaniu n-gramov pre riešenie rôznych úloh popísaných v predchádzajúcich podkapitolách, je potrebný spoľahlivý a rýchly algoritmus pre ich extrakciu z textu. Vhodný nástroj slúžiaci na extrakciu n-gramov z textu by mal dokázať pracovať s textom neobmedzenej veľkosti, pracovať rýchlo a efektívne využívať dostupnú operačnú pamäť. Existuje viacero metód extrahujúcich n-gramy z textu. Tieto metódy sú založené na rôznych princípoch:

- algoritmus Nagao '94,
- algoritmus Lempel-Ziv-Welch pre kompresiu,
- sufixové pole,
- sufixový strom,
- invertovaný index .

Vyššie zmienené metódy a princípy budú bližšie popísané v tejto kapitole.

3.1 Nagao '94 algoritmus

V [7] je tento algoritmus detailne popísaný. Metóda sa používa na extrakciu n-gramov z japonského textu, kde je každé slovo reprezentované jedným znakom. Určuje frekvenciu výskytu postupnosti znakov. Tento postup pozostáva z dvoch etáp. V prvej etape je vstupný text reprezentovaný ako dlhý reťazec pozostávajúci zo slov, bielych znakov atď. Tento text je uložený v jednom poli. Druhé pole, rovnakej veľkosti ako prvé pole, obsahuje ukazatele na podreťazce v prvom poli, tak ako je znázornené na obrázku 3.1.



Obrázok 3.1: Prvá etapa algoritmu Nagao '94

Podreťazec, na ktorý ukazuje i-1 ukazateľ je definovaný ako kompozícia znakov od i-tej pozície po koniec textového reťazca. Pole ukazateľov je následne abecedne zoradené, čím dostaneme zhodné podreťazce pri sebe. Radením poľa ukazateľov sa nezmení obsah textového poľa.

Po zoradení poľa sú porovnávané každé dva susedné podreťazce a vypočíta sa dĺžka prefixu, ktorý je zhodný pre oba podreťazce. Napríklad ak sú dva susedné podreťazce “na žltej zahneme vpravo“ a “na žltej zahneme vľavo“, tak je dĺžka zhodného prefixu rovná sedemnásť. Toto číslo je potom uložené do tabuľky súčasných výskytov prefixov podreťazcov.

Druhá etapa je výpočet frekvencie n-gramov. Používame pole ukazateľov a tabuľku súčasných výskytov prefixov znakov vytvorené v predchádzajúcej etape. Pre zvolené *n* prechádzame pole ukazateľov, načítame prvých *n* znakov prvého slova a skontrolujeme číslo v tabuľke. Ak je toto číslo väčšie alebo rovné ako *n*, tak to znamená, že druhé slovo má najmenej *n* spoločných prefixových znakov s prvým slovom. Následne skontrolujeme ďalší záznam v tabuľke a porovnáme ho s *n*. Tento postup opakujeme až kým číslo z tabuľky nie je menšie ako *n*. Počet takto prejdených slov od prvého slova je frekvencia výskytu *n* znakov z prvého slova. Tento istý postup sa opakuje pre všetky slová.

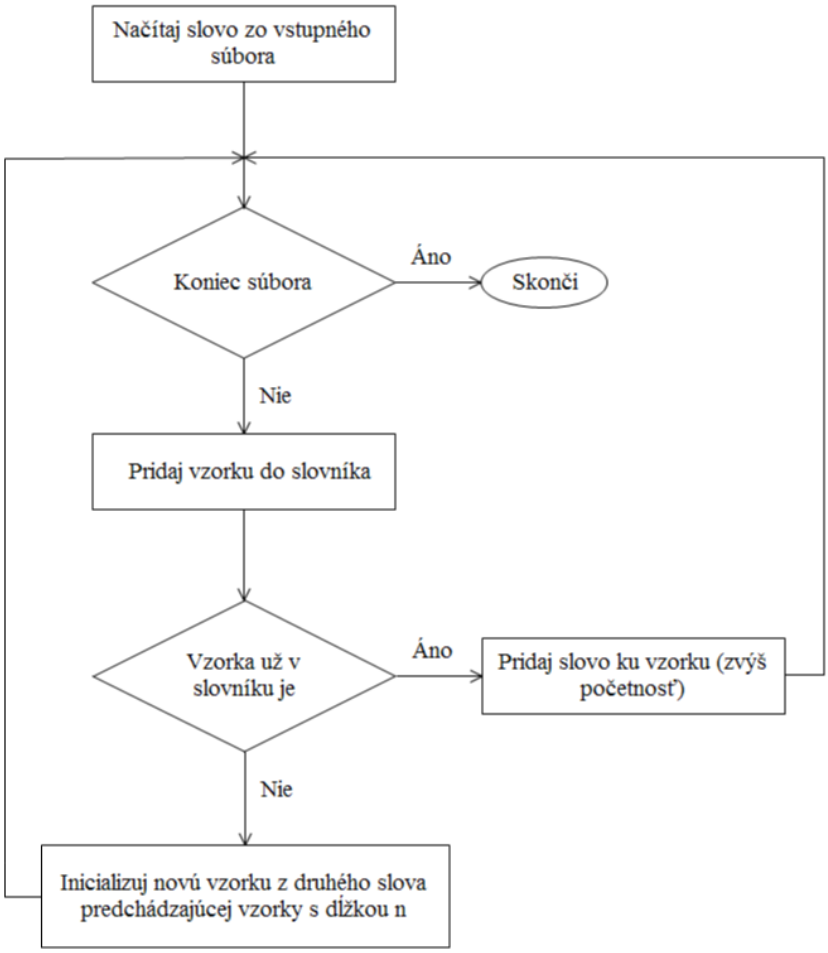
3.2 Metóda využívajúca LZW algoritmus

Táto metóda je inšpirovaná Lempel-Ziv-Welch (LZW) algoritmom bezstratovej dátovej kompresie. Princíp LZW algoritmu spočíva v postupnom vytváraní kódovacej tabuľky zo slov

použitých v už zakódovanom texte. V tejto tabuľke je každý vstup mapovaný na slová s pevne stanovenou dĺžkou. Pri začiatku kódovania je tabuľka zvyčajne inicializovaná pomocou všetkých znakov z ASCII tabuľky. Algoritmus potom sekvenčne prehľadáva text a ukladá si do tabuľky každé unikátne dvojznakové slovo ako spojenie kódu a vzoru. Po uložení všetkých dvojznakových slov je na výstup poslaný kód prvého znaku na vstupu. Algoritmus pokračuje v kódovaní, keď narazí na už zakódované slovo tak na výstup odošle zodpovedajúci kódový znak s prvým znakom slova. V [8] je LZW algoritmus bližšie popísaný.

V [9] je predstavený algoritmus na extrakciu n-gramov z textu, ktorý je založený na vyššie popísanom princípe LZW. Princíp algoritmu je znázornený vývojovým diagramom na obrázku 3.2.

Tento algoritmus používa binárny strom na uloženie už extrahovaných vzorkov. Jednotlivé vzorky sú v tomto prípade chápané ako slová.



Obrázok 3.2: Vývojový diagram algoritmu extrahujúceho n-gramy z textu na základe LZW algoritmu

3.3 Sufixové pole

Ďalším, veľmi často používaným prístup pre získanie n-gramov a ich početností z textu je metóda založená na práci so sufixovým poľom.

Podobná dátová štruktúra pre extrakciu n-gramov bola popísaná v kapitole 3.1, tento postup je však vhodný len pre získanie frekvencie výskytu podreťazcov z japonského textu. V tejto časti bude tento princíp popísaný názornejšie. Sufixové pole je dátová štruktúra obsahujúca N sufixov, abecedne zoradených. Sufix, s[i], je reťazec, ktorý začína na i-tej pozícii v korpuse a pokračuje až do konca korpusu. Obrázok 3.3 znázorňuje rozdelenie vstupného reťazca “to be or not to be“ do poľa znakov a jeho indexáciu v sufixovom poli.

Vstupný reťazec rozdelený na slová:

to	be	or	not	to	be

Suffixové pole pre daný vstupný reťazec:

index	Sufix na konkrétnom indexe
0	to be or not to be
1	be or not to be
2	or not to be
3	not to be
4	to be
5	be

Obrázok 3.3: Obsah sufixového poľa pre vstupný reťazec "to be or not to be"

Predchádzajúci obrázok ilustruje obsah sufixového poľa pre reťazec “to be or not to be“. Po načítaní reťazca a vytvorení sufixového poľa nasleduje zoradenie sufixov podľa abecedy. Stav sufixového poľa po zoradení demonštruje obrázok 3.4.

Frekvencia jednotlivých termov sa vypočíta ako rozdiel indexu prvého výskytu a indexu posledného výskytu daného termu (obrázok 3.4). K tomuto rozdielu sa pripočíta jednotka. Frekvencia výskytov jednotlivých termov sa uloží do tzv. LCP (longest common prefix) poľa.

Obrázok 3.4 znázorňuje obsah sufixového poľa pred a po abecednom zoradení jednotlivých sufixov. Výpočet frekvencie pre term “to be“ by bol prevádzaný pokiaľ by nás zaujímala extrakcia

bigramov. Pre zvolenú dĺžku n-gramu by sa načítal príslušný počet termov a tieto by boli následne vyhľadávané v prefixoch nasledujúcich sufixov.

Po naplnení LCP poľa je vhodné vytvoriť štruktúru tzv. n-gram deskriptorov [10], ktorá obsahuje dvojicu (koncová pozícia, frekvencia). Vďaka uloženiu do štruktúr deskriptorov je možné prevádzať nad n-gramami niekoľko vhodných operácií pre konečný výstup. Prvou je vytriedenie menej frekventovaných deskriptorov, toto je užitočné pokiaľ nepožadujeme do výstupu zahrnúť n-gramy s nízkou frekvenciou výskytu. Druhou, nemenej podstatnou operáciou je zoradenie deskriptorov podľa koncovej pozície, čím dostaneme abecedné zoradenie deskriptorov. Posledným krokom je extrakcia n-gramov z deskriptorov, čo pomocou koncovej pozície, frekvencie a požadovanej dĺžky n-gramov je triviálna úloha.

Sufixové pole pred zoradením:

index	Sufix na konkrétnom indexe
0	to be or not to be
1	be or not to be
2	or not to be
3	not to be
4	to be
5	be

Sufixové pole po zoradení:

index výskytu	index slova	Sufix na konkrétnom indexe
0	5	be
1	1	be or not to be
2	3	not to be
3	2	or not to be
4	4	to be
5	0	to be or not to be

Počiatočný index výskytu termu "to be" na začiatku sufixu (j) : 4
Konečný index výskytu termu "to be" na začiatku suffixu (k): 5

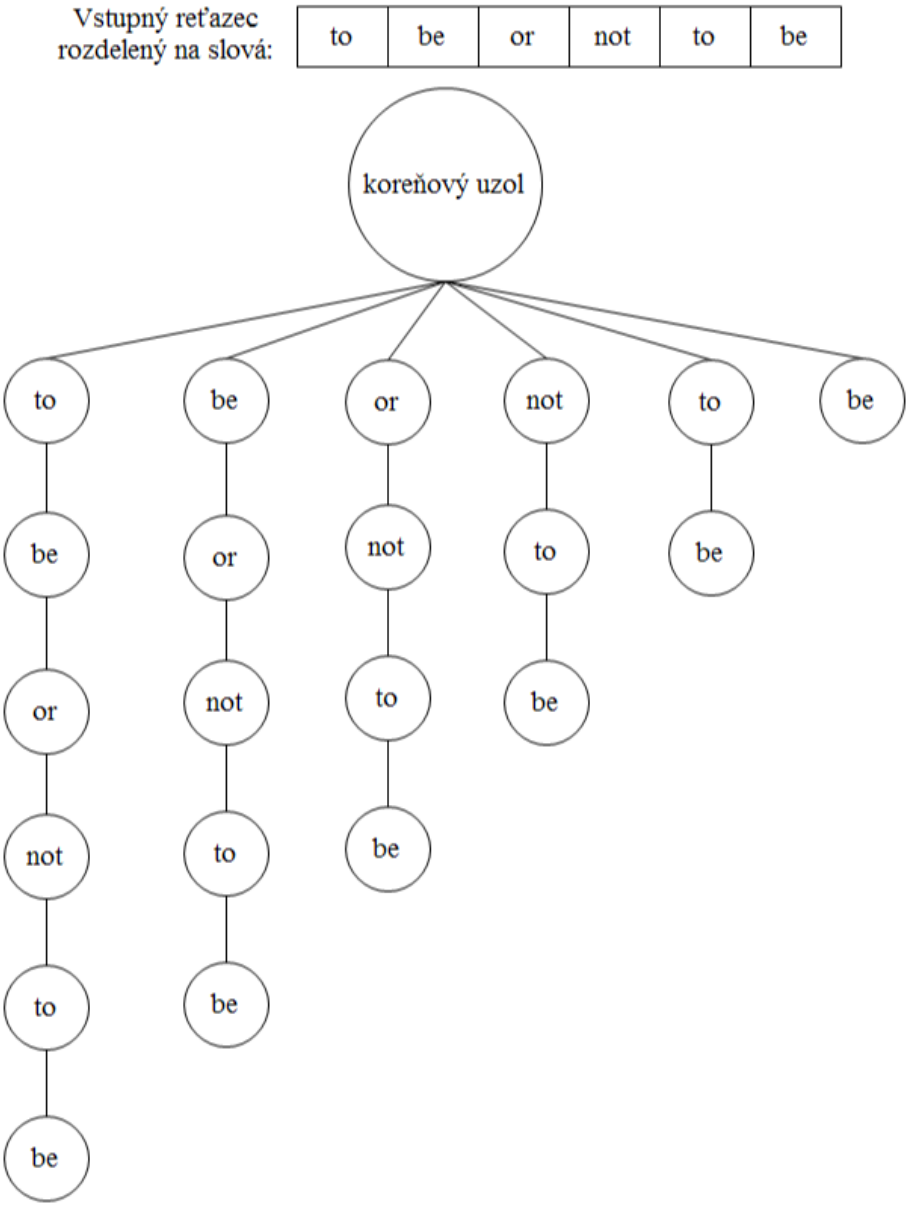
Frekvencia termu "to be" = k - j + 1 = **2**

Obrázok 3.4: Sufixové pole pred a po zoradení, príklad výpočtu frekvencie termu

3.4 Sufixový strom

Sufixový strom je taká dátová štruktúra, v ktorej každý vrchol *v* reprezentuje časť podreťazca (príp. slova), ktorý dostaneme ako zreťazenie hrán na ceste z koreňa do listu. [11] Metóda na extrakciu n-gramov pomocou sufixových stromov je veľmi podobná prístupu so sufixovým poľom. Rozdiel je práve v dátovej štruktúre, do ktorej sú ukladané ukazateľe na jednotlivé slová vstupného

reťazca. Uloženie vstupného reťazca “to be or not to be“ do sufixového stromu znázorňuje obrázok 3.5.



Obrázok 3.5: Uloženie vstupného reťazca "to be or not to be" do sufixového stromu.

Na obrázku 3.5 je zobrazený obsah sufixového stromu po načítaní vstupného reťazca. V uzloch nie sú uložené kompletné slová, ale každý synovský uzol koreňového uzla ukazuje na podreťazec slov v poli obsahujúcom vstupný textový reťazec. Ďalší postup pre extrakciu slovných n-gramov je zhodný s postupom popísaným v kapitole 3.3. Postupy pre extrakciu slovných n-gramov z textu bývajú často založené práve buď na práci so sufixovým poľom, alebo práci so sufixovým stromom.

Porovnaniu týchto dvoch metód sa venuje nasledujúca kapitola 3.5. Okrem extrakcie slovných n-gramov majú sufixové stromy využitie v mnoho iných oblastiach práce s textom, medzi ne patrí [11]:

- určenie, či je dané slovo *y* podreťazcom (príp. podslovom) iného reťazca *x* (príp. slova)
- nájdenie všetkých výskytov slova *y* v reťazci *x,*
- vyhľadávanie dvojíc opakovaných výskytov,
- vyhľadávanie maximálnych opakovaní,
- vyhľadávanie najdlhšieho palindromu v texte,
- kompresia dát,
- identifikácia osôb pomocou DNA.

Hlavnou nevýhodou použitia sufixových stromov je ich priestorová náročnosť.

3.5 Porovnanie metód založených na sufixovom poli a sufixovom strome

Ako bolo spomenuté v predchádzajúcej podkapitole, metódy extrakcie n-gramov založené na sufixovom poli a sufixovom strome bývajú v praxi často využívané. V [12] je vypracované porovnanie týchto dvoch metód. Stručné zhrnutie postupu a výsledkov bude popísané v tejto podkapitole.

Hlavným cieľom bolo zistenie možnosti zefektívnenia extrakcie n-gramov pomocou sufixového poľa ako aj vzájomné porovnanie s implementáciou metódy využívajúcej sufixový strom. Výpočtová zložitosť pre sufixové pole je $\Theta(N \cdot \log(N))$, a pre sufixový strom je zložitosť logaritmická, $\Theta(\log(N))$, kde N je veľkosť vstupného textového korpusu v počte slov. Preto autori porovnávali práve tieto dve metódy.

Sufixové pole bolo zoraďované pomocou algoritmu quick sort. Časová zložitosť bola v tomto prípade $\Theta(N \cdot \log(N))$. Pre sufixový strom autori použili dve implementácie:

1. implementácia sufixového stromu založená na zreťazenom zozname,
2. implementácia sufixového stromu založená na hashovacej tabuľke.

V prvom prípade bola časová zložitosť pre vytvorenie sufixového stromu $\Theta(N \cdot K)$, kde K vyjadruje veľkosť abecedy. Druhý algoritmus umožňuje veľmi rýchle vytvorenie sufixového stromu s časovou zložitosťou $\Theta(N)$, avšak tento postup nie je vhodný pre extrakciu n-gramov. Pre extrakciu n-gramov je potrebný čas rovný $\Theta(N \cdot K)$.

Každý algoritmus sa skladá z dvoch fáz, prvá fáza je načítanie vstupného korpusu a vytvorenie dátovej štruktúry s jej následným naplnením. Druhá fáza je vlastná extrakcia n-gramov pomocou naplnených dátových štruktúr. Vzhľadom na vytvorenie dátovej štruktúry je najrýchlejšou

metódou sufixové pole. Pre nízky počet slov, je rýchlejšia metóda využívajúca sufixový strom, avšak s narastajúcou komplexnosťou výpočtu a zložitosťou algoritmu efektivita tejto metódy klesá.

V druhej fáze je najrýchlejšia metóda opäť tá, ktorá využíva sufixové pole, najpomalšou je implementácia sufixového stromu pomocou hashovacej tabuľky, práve kvôli jej zložitosti $\Theta(N \cdot K)$.

3.6 Invertovaný index

Veľmi často využívanou dátovou štruktúrou pre potreby spracovania prirodzeného jazyka je invertovaný index. Táto štruktúra je významná nie tak pre extrakciu n-gramov z textu, ale hlavne pre jej efektivitu pri vyhľadávaní podreťazcov v texte. Pre vyhľadávanie podreťazcov sa využívajú práve n-gramy. Dátová štruktúra invertovaného indexu spravidla obsahuje dvojicu:

- zoznam dokumentov, ktoré obsahujú konkrétny term,
- pre každý výskyt termu v dokumente obsahuje frekvenciu výskytu v danom dokumente a pozície, na ktorých sa term v dokumente vyskytuje.

Na extrakciu n-gramov z textu nie je invertovaný index vhodným prístupom, hlavne kvôli jeho priestorovým nárokom ako aj neefektívnosti v prípade vyššej dĺžky n-gramov. Svoje uplatnenie však okrem spomenutého vyhľadávania nachádza v oblasti získavania informácií z textu, bioinformatike. Vďaka jeho nezávislosti na jazyku je často využívaný pre prácu s japonským, kórejským a čínskym textom.

3.7 Obmedzenia metód

Všetky vyššie popisované metódy majú spoločnú negatívnu a obmedzujúcu vlastnosť, a tou je fakt, že dokážu pracovať len s takým množstvom textu, koľko sa zmestí do operačnej pamäte. Respektíve, efektívne dokážu pracovať len v tom prípade, ak je celý spracovávaný text načítaný v operačnej pamäti. Vzhľadom ku práci s diskom nie je možné metódy využívajúce tieto štruktúry implementovať efektívne. Hlavným dôvodom je vysoký počet nutných prístupov na disk, čím sa značne znižuje použiteľnosť algoritmu.

Pri použití sufixového poľa na extrakciu n-gramov, kde by časť poľa bola načítaná v pamäti a časť uložená na disku, by bolo pri fáze vyhľadávania spoločných prefixov nutné pre každú položku načítanú v pamäti pristúpiť na disk. Časté prístupy na disk však značne zneefektívňujú algoritmus. Tento princíp platí pre všetky štruktúry popísané v predchádzajúcich kapitolách.

V [2] bol navrhnutý a popísaný algoritmus Teraman, ktorý je schopný extrahovať n-gramy z textov neobmedzenej veľkosti. Je tu použitý princíp dávkového spracovania a ukladanie medzivýsledkov na

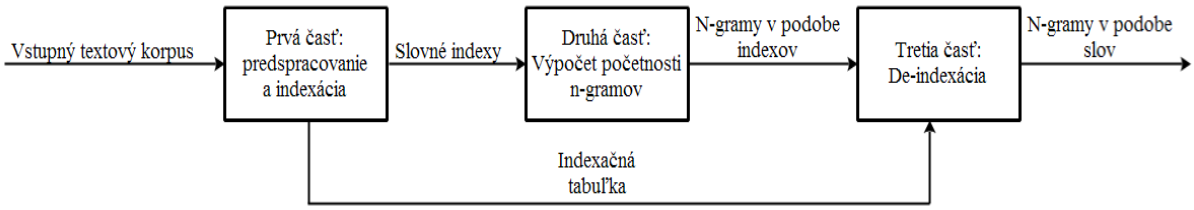
disk, takisto dokáže efektívne využívať operačnú pamäť. Jeho princípy, implementácia, zrýchlenie pomocou paralelizácie a vyhodnotenie výsledkov sú popísané v nasledujúcich kapitolách.

4 Implementovaný systém

Táto kapitola obsahuje detailný popis princípu, na akom implementovaný systém pracuje. Tento systém bol predstavený v [2] ako algoritmus Teraman. Tento algoritmus dokáže extrahovať n-gramy z textu akejkoľvek veľkosti bez ohľadu na výkon počítača, na ktorom algoritmus vykonáva svoju činnosť.

4.1 Popis systému

Základnou črtou algoritmu Teraman je dávkové spracovanie, ktoré vstupné dáta rozdelí na menšie celky, a tie následne spracuje v dostupnej operačnej pamäti. Pre dosiahnutie čo najlepšej rýchlosti algoritmu sa nepracuje priamo s textovými reťazcami. Jednotlivé slová sú prevádzané na unikátne 32-bitové čísla, vďaka tomu je zaberané menej operačnej pamäte ako aj samotná práca s číslami je podstatne rýchlejšia ako práca s reťazcami. Systém sa skladá z troch hlavných častí, každá časť potom z jednotlivých fáz. Popis týchto etáp je obsahom nasledujúceho textu. Komplexnú činnosť ilustruje obrázok 4.1



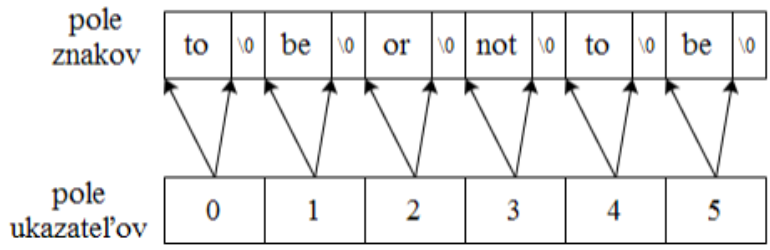
Obrázok 4.1: Hlavné časti algoritmu Teraman

4.1.1 Predspracovanie a indexácia

Predspracovanie a indexácia je úvodnou časťou implementovaného systému. Jej úlohou je transformácia vstupného textu na číselnú reprezentáciu. Táto časť pozostáva z piatich fáz.

4.1.1.1 Načítanie slov

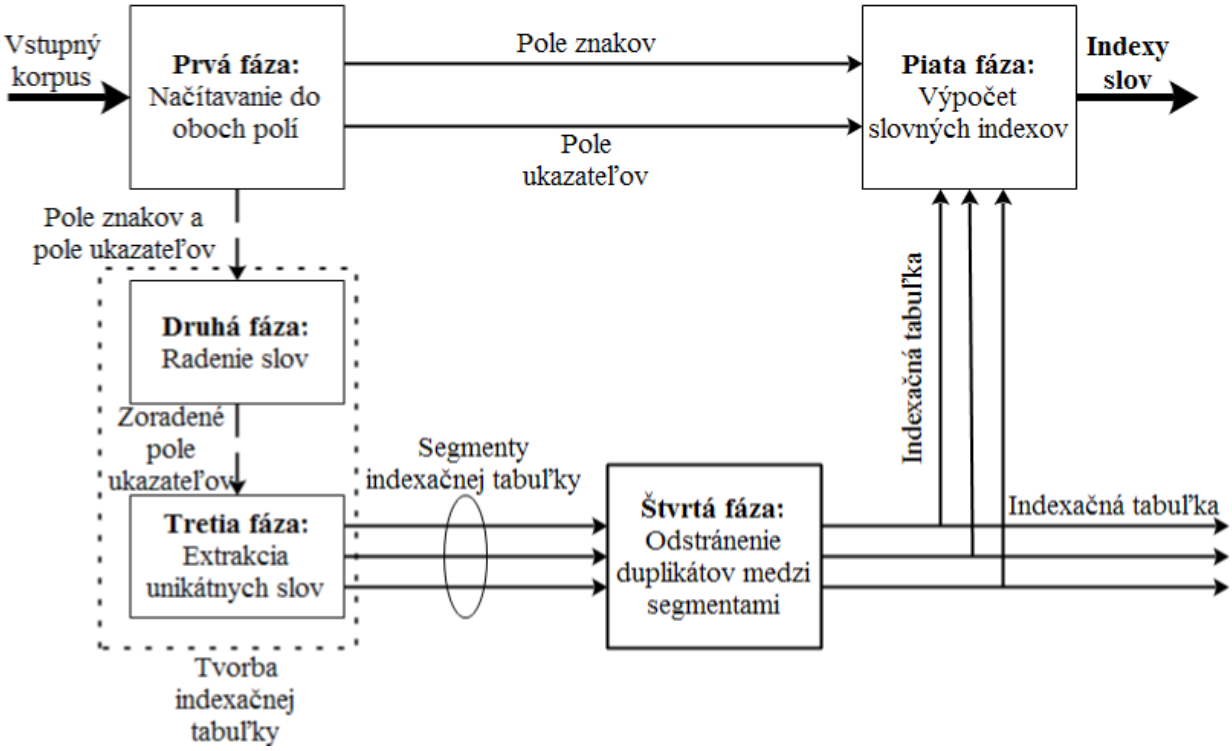
Na začiatku sa alokujú dve polia. Jedno pole, pole znakov, udržuje načítané znaky zo vstupného textového korpusu. Druhé pole obsahuje ukazatele na počiatky slov v poli znakov, toto pole sa nazýva pole ukazateľov. V poli znakov je za každé slovo vložený znak ukončujúci slovo, tým je dosiahnuté to, že položky v poli ukazateľov pri práci s textom obsahujú len jednotlivé slová tak, ako je to znázornené na obrázku 4.2.



Obrázok 4.2: Obsah poľa ukazateľov a poľa znakov

Pri načítaní je vhodné celkový text prevádzať na malé písmená. Vstupný textový korpus teda nie je uložený a spracovávaný v rovnakej podobe ako bol pôvodne vytvorený. Ďalšia zmena je automatické vynechávanie bielych znakov. Ľubovoľný počet v bielych znakov je chápaný ako separátor dvoch slov. Posledný filter, ktorý je implicitne aplikovaný, je vynechávanie interpunkcie.

Veľkosť oboch polí ako aj pomer ich veľkostí som stanovil empiricky, podľa toho, kedy mal program najlepšiu výkonnosť. V prípade dosiahnutia konca jedného alebo druhého poľa algoritmus prejde do druhej a následne tretej fázy prvej časti, a obsah poľa znakov uloží na disk. Pre lepšiu ilustráciu činnosti predspracovania a indexácie, sú jednotlivé fázy so svojimi vstupmi a výstupmi znázornené na obrázku 4.3.



Obrázok 4.3 : Schematické zobrazenie prvej časti algoritmu Teraman, predspracovanie a indexácia

4.1.1.2 Radenie slov

Druhá fáza prvej časti úzko súvisí s prvou fázou, načítaním slov. Pri dosiahnutí konca ľubovoľného z polí napĺňaných vo fáze načítanie, dochádza ku abecednému zoradeniu načítaných slov vzostupne. Zoraďované je len pole ukazateľov, čiže žiadne zmeny v poli načítaných znakov nenastávajú. Táto fáza je prvým krokom ku vytvoreniu segmentu indexačnej tabuľky. Zložitosť fázy radenia slov je $O(n \cdot log_2(n/k))$, kde n je celkový počet slov v korpuse a k je počet blokov vytvorených dávkovým spracovaním [2].

4.1.1.3 Extrakcia unikátnych slov

Po vzostupnom zoradení slov v predchádzajúcej fáze sú v poli ukazateľov rovnaké slová umiestnené bezprostredne za sebou. Úlohou tejto fázi je jedným priechodom poľa ukazateľov extrahovať unikátne slová a tieto následne uložiť na disk ako segment indexačnej tabuľky. Na disk sú uložené aj slová z poľa znakov, ktoré budú potrebné pre neskorší výpočet indexov slov. Časová zložitosť pre prechod všetkých polí a uloženie indexačnej tabuľky je $O(n)$, kde n je celkový počet slov korpusu [2].

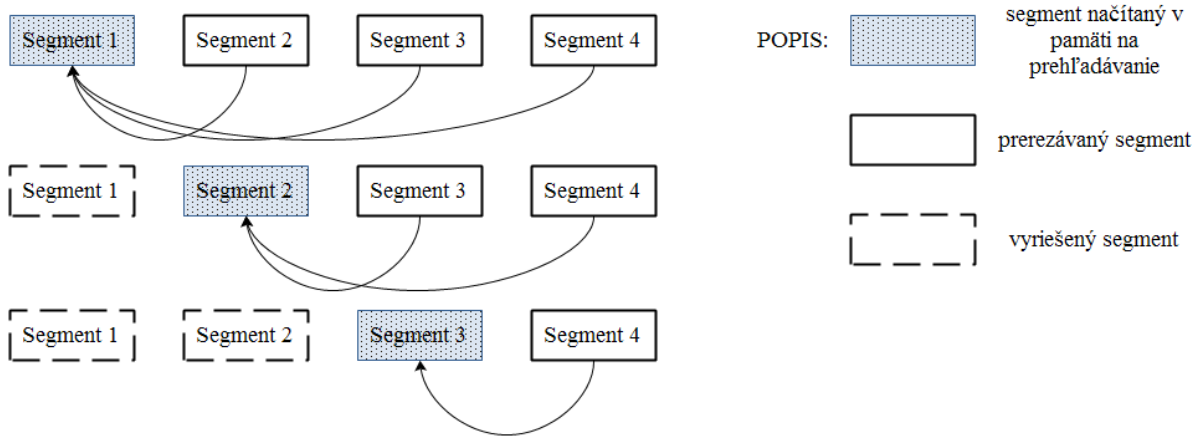
4.1.1.4 Odstraňovanie duplikátov medzi segmentami indexačnej tabuľky

V predchádzajúcich fázach bolo v dôsledku dávkového spracovania vytvorených viacero segmentov indexačnej tabuľky. Z tohto dôvodu je možné, že jednotlivé segmenty obsahujú duplicitné slová. Indexačná tabuľka obsahuje všetky slová zo vstupného textu práve raz, čo nám umožní jednoznačnú identifikáciu slov číselným indexom. Z toho dôvodu je nutné porovnať všetky segmenty navzájom a prípadné duplicitné výrazy odstrániť. Pri mazaní duplicity načítame do pamäte segment i, kde $i \in <1,k>$, kde k reprezentuje celkový počet vytvorených segmentov. Po tomto kroku postupne porovnávame všetky slová zo zvyšných j segmentov, kde $j \in <i+1,k>$. V druhej fáze predspracovania a indexácie boli slová abecedne zoradené, to nám umožňuje pri vyhľadávaní prípadného výskytu slova v segmentoch využiť algoritmus binárneho vyhľadávania.

Binárne vyhľadávanie je vyhľadávací algoritmus, ktorý slúži na nájdenie požadovanej hodnoty v usporiadanom zozname pomocou skracovania zoznamu o polovicu v každom kroku. Binárne vyhľadávanie nájde medián, urobí porovnanie a na základe tohto sa rozhodne o pokračovaní v hornej alebo dolnej časti zoznamu a rekurzívne sa pokračuje od začiatku [13].

Ak je v segmente j nájdené rovnaké slovo ako je načítané v pamäti, tak je zo segmentu j toto slovo zmazané. Príklad postupu pri odstraňovaní duplikátov zo štyroch segmentov je zachytený na obrázku 4.4.

Časová zložitosť tejto fázy závisí od rozloženia slov v segmente *j*.



Obrázok 4.4: Ilustrácia prehľadávania segmentov indexačnej tabuľky, za účelom odstránenia duplikátov

4.1.1.5 Výpočet indexov slov

Vstupmi do poslednej fázy prvej časti algoritmu je pole znakov s poľom ukazateľov ukazujúcich na začiatky slov v prvom poli, ďalším vstupom je indexačná tabuľka rozdelená do segmentov. Po odstránení duplicitných výskytov slov je možné pre každé slovo vypočítať unikátny index pomocou nasledujúceho vzorca:

$$index_w = p_{w,m} + \sum_{j-1}^{m-1} c_j \tag{1}$$

kde $p_{w,m}$ je poradie slova v segmente m, uvažované od začiatku segmentu a c_j je celkový počet slov v segmente j.

Úlohou tejto fázy je nahradenie každého slova unikátnym číslom, ktoré vyjadruje index slova v rámci indexačnej tabuľky. Na začiatku načítame do pamäte *i*-te pole znakov a *i*-te pole ukazateľov, ktoré boli vytvorené vo fázy načítania slov. Následne načítavame do pamäte segmenty indexačnej tabuľky, vzhľadom na dostupnú operačnú pamäť. Tu môžeme zvoliť dva prístupy, buď načítame vždy jeden segment indexačnej tabuľky, v prípade malej dostupnej operačnej pamäte aj časť jedného segmentu, alebo načítame do pamäte čo najviac segmentov. Po prevedení testov a meraní som dospel k záveru, že je výkonnostne lepšie do pamäte načítavať po jednom segmente indexačnej tabuľky.

Ďalším krokom je prechádzanie ukazateľov na začiatky slov, a hľadanie týchto slov v načítanom segmente indexačnej tabuľky. Pre toto vyhľadávanie môžeme opäť použiť algoritmus binárneho vyhľadávania, keďže slová v tabuľke sú usporiadané podľa abecedy. Pri úspešnom vyhľadaní slova v tabuľke je jeho index vypočítaný pomocou vzorca (2). Pre zaznamenanie

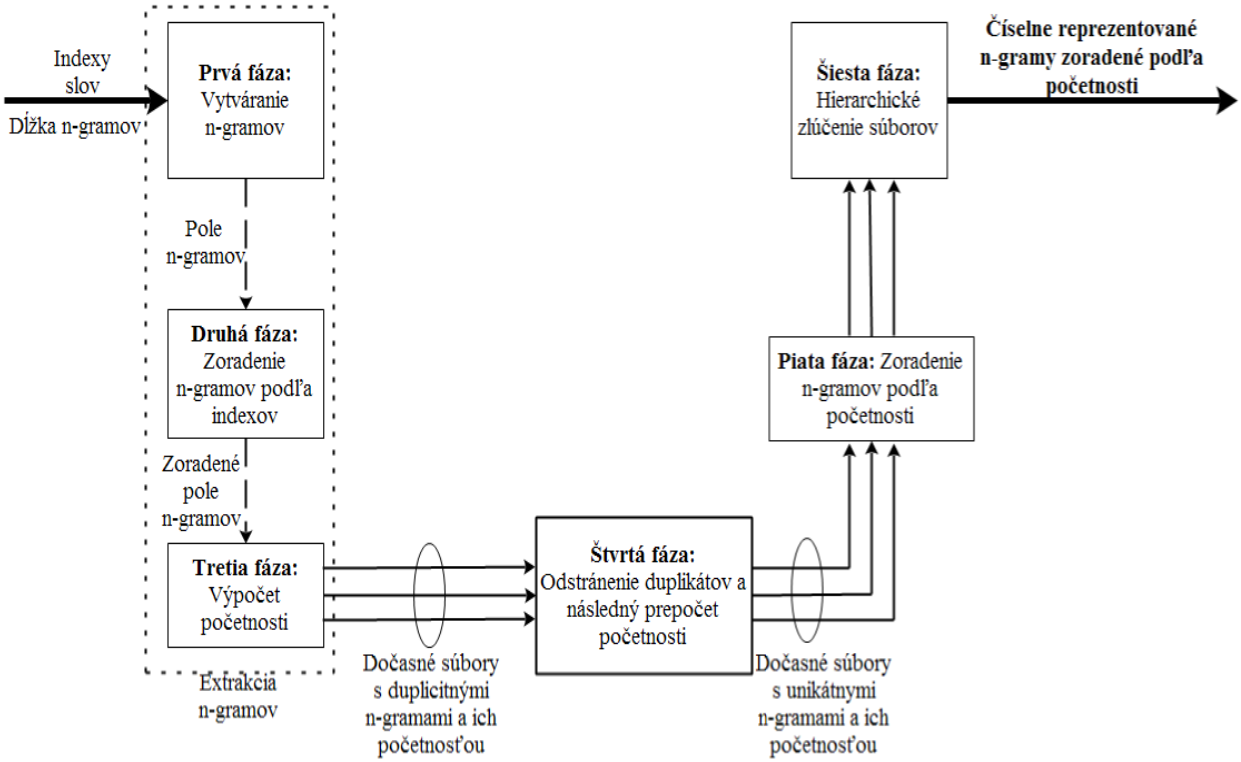
vypočítaného indexu som testoval dva prístupy. Prvým bolo uloženie zápornej hodnoty indexu slova do poľa ukazateľov. Druhým bol prístup pomocou bitového poľa, kde pomocou nastavovania jednotlivých bitov poľa bol indikovaný index poľa ukazateľov, ktorý obsahoval slovo, s už vypočítanou hodnotou indexu. Pre vyššiu efektivitu a nižšie pamäťové nároky som zvoli prístup pomocou zapisovania zápornej hodnoty vypočítaného indexu.

Následne, po načítaní ďalšieho segmentu indexačnej tabuľky do pamäte, sú do výpočtu zahrnuté len ukazateľe s nezápornou hodnotou. Po nahradení všetkých ukazateľov je ich hodnota prevedená naspäť na kladnú, a pripojená do súboru ktorý obsahuje vstupný korpus reprezentovaný číselnými indexmi slov. Tento súbor je zároveň aj vstupom do nasledujúcej časti počítania početnosti n-gramov.

Výsledná časová zložitosť tejto fázy je $0(k \cdot n \cdot log_2(t / k))$, kde n predstavuje celkový počet slov vo vstupnom textovom korpuse, t je celkový počet slov nachádzajúci sa v indexačnej tabuľke, k je počet segmentov indexačnej tabuľky [2].

4.1.2 Výpočet početnosti n-gramov

Táto časť tvorí jadro implementovaného systému. Jej vstupom je súbor vytvorený predchádzajúcou časťou, ktorý obsahuje slovné indexy. Výstup tvorí súbor s početnosťami jednotlivych n-tíc slov, ktoré sú v tomto prípade reprezentované číselnými indexmi. N-gramy sú extrahované postupne, čo nám zaručuje možnosť práce aj za hranicou dostupnej operačnej pamäte. Výpočet početnosti n-gramov je rozdelený do šiestich fáz, ich činnosť znázorňuje obrázok 4.5. Na obrázku sú znázornené jednotlivé fázy so svojimi vstupmi a výstupmi.



Obrázok 4.5: Schéma druhej časti algoritmu Teraman

4.1.2.1 Vytváranie n-gramov

Vstupom do prvej fázy druhej časti je indexový súbor, kde sú jednotlivé slová zakódované ako čísla. Z tohto faktu vyplýva, že je na nás zvoliť rozsah indexov. Postačujúcim je určite 32-bitový rozsah, teda <0, 2^{32}-1>, pretože žiaden jazyk neobsahuje viac ako 4 294 967 296 unikátnych a zmysluplných slov. Pre ilustráciu, Google teracorpus obsahuje 13 588 391 unikátnych slov, ktoré sa v ňom vyskytujú aspoň dvestokrát.

Pre možnosť extrakcie n-gramov neobmedzenej dĺžky sa načítavajú do dvojrozmerného poľa s rozmermi $(m \cdot n)$, kde m predstavuje počet n-gramov, a n vyjadruje dĺžku n-gramu zvýšenú o jeden, kvôli potrebe uloženia početnosti výskytu daného n-gramu.

Pri počte n-gramov m presahujúcom hodnotu h, $m > h$, ktorú som empiricky stanovil pre dosiahnutie najvyššej výkonnosti systému, sa n-gramy načítajú iteračne. V prípade nedostatku pamäte pre alokáciu poľa na uloženie n-gramov danej dĺžky n a pevne zadanej veľkosti h, je veľkosť poľa zmenšovaná až do úspešnej alokácie.

Časová zložitosť tejto fázy je $O(s \cdot n)$, kde s je dĺžka n-gramu a n je celkový počet slovných indexov.

4.1.2.2 Zoradenie n-gramov

Pri dosiahnutí konca alokovaného poľa pre n-gramy, prípadne načítaní všetkých n-gramov je nutné toto pole zoradiť. N-gramy sú zoraďované podľa hodnôt slovných indexov ktoré obsahujú. Značné urýchlenie radenia prináša fakt, že pracujeme len s číslami, radenie prebieha pomocou algoritmu quick sort.

Časová zložitosť radenia je $O(s \cdot n \cdot log2(s \cdot n / k))$, kde s vyjadruje dĺžku n-gramu, n je celkový počet slovných indexov a k je celkový počet iterácií nutných na načítanie všetkých n-gramov [2].

4.1.2.3 Výpočet početnosti

Po zoradení dochádza ku výpočtu početnosti n-gramov. Vstupom do tretej fázy druhej časti je zoradené pole n-gramov uložených v poli, kde identické n-gramy ležia vedľa seba. Vďaka tejto skutočnosti početnosť n-gramov získame jedným priechodom poľom s n-gramami.

Po prejdení celého poľa a výpočte početnosti sa toto pole pre každú iteráciu uloží do dočasného súboru. Na začiatku súboru je informácia o počte n-gramov v súbore, nasledovaná slovnými indexmi a ich frekvenciou výskytu. Na obrázku 4.6 je zobrazený prípad extrakcie trigramov z textu pred a po fáze výpočtu početnosti.

Stav poľa s n-gramami po zoradení, pred výpočtom početnosti:

slovné indexy			početnosť
0	1	2	1
0	1	2	1
0	1	2	1
2	8	3	1
2	8	3	1
2	9	1	1
2	9	1	1
3	5	3	1
4	1	6	1

Stav poľa s n-gramami po výpočte početnosti:

slovné indexy			početnosť
0	1	2	3
0	1	2	0
0	1	2	0
2	8	3	2
2	8	3	0
2	9	1	2
2	9	1	0
3	5	3	1
4	1	6	1

Obrázok 4.6: Obsah poľa s n-gramami pred a po výpočte početnosti

V [2] je časová zložitosť tejto fázy uvedená ako $O(s \cdot n)$, kde s vyjadruje dĺžku n-gramu a n je celkový počet slovných indexov.

4.1.2.4 Odstránenie duplikátov z dočasných súborov a prepočet početností

V predchádzajúcich fázach extrakcie n-gramov a následnom počítaní ich frekvencie v texte boli vytvorené dočasné súbory popísané v kapitole 4.1.2.3. Tieto dočasné súbory sú vstupom do štvrtej fázy druhej časti, pretože môžu obsahovať duplicitné n-gramy. Úlohou tejto fázy je odstránenie akýchkoľvek duplicít medzi pomocnými súbormi s n-gramami a upraviť početnosť takto zmazaných duplicít.

Algoritmus tejto fázy je veľmi podobný tomu, ktorý bol popísaný v kapitole 4.1.1.4, s tým rozdielom, že namiesto slov sú v jednotlivých segmentoch vyhľadávané a odstraňované číselné indexy. Pomocný súbor i, $i \in <1,k>$, kde k vyjadruje počet pomocných súborov vytvorených predchádzajúcimi fázami, je načítaný do pamäte. Následne sú z pomocného súboru j, $j \in <i+1,k>$, načítavané n-gramy, ktoré sú porovnávané s n-gramami načítanými v pamäti. Pre vyhľadanie n-gramu v pamäti je opäť možné použiť algoritmus binárneho vyhľadávania, keďže n-gramy sú zoradené podľa číselných indexov. Pri vyhľadaní n-gramu zo súboru j v súbore i, ktorý je načítaný v pamäti, je početnosť tohto n-gramu zvýšená v súbore i, a zo súboru j je n-gram odstránený.

Časová zložitosť odstraňovania duplicitných n-gramov je podobná časovej zložitosti štvrtej fázy prvej časti algoritmu, a teda závisí od rozloženia n-gramov v dočasných súboroch.

4.1.2.5 Zoradenie n-gramov podľa početnosti

Kvôli lepšej využiteľnosti algoritmu pre praktické riešenie úloh z oblasti spracovania prirodzeného jazyka, je vhodné mať jeho výstup, teda n-gramy slov z textu a ich početnosť zoradené podľa početnosti. Toto zoradenie je úlohou piatej fázy druhej časti algoritmu.

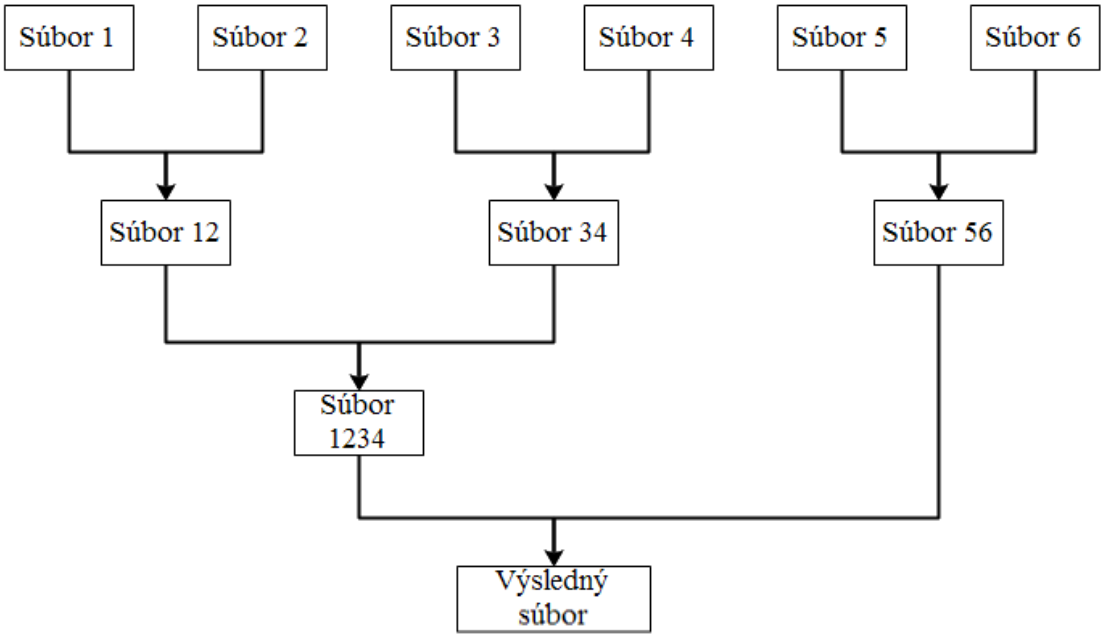
N-gramy sú z jednotlivých dočasných súborov načítané do pamäte, následne prebehne ich zoradenie podľa početnosti, využívajúce algoritmus quick sort. Po zoradení je starý dočasný súbor prepísaný novým, s už zoradenými n-gramami. Tento postup sa opakuje pre všetky dočasné súbory.

V [2] je časová zložitosť rovná $O(n \cdot log2(n / k))$, kde n je celkový počet n-gramov a k je celkový počet dočasných súborov.

4.1.2.6 Zlúčenie dočasných súborov

Úlohou poslednej fázy druhej časti je zlúčenie dočasných súborov s n-gramami zoradenými podľa početnosti do jedného súboru.

Kompletná veľkosť všetkých súborov dokopy vo väčšine prípadov presahuje veľkosť dostupnej operačnej pamäte, preto nie je možné spojiť naraz všetky súbory. Pre zlúčenie je teda využívané hierarchické spojenie súborov, ktorého princíp je zobrazený na obrázku 4.7.



Obrázok 4.7: Hierarchické zlúčenie súborov

Ako príklad je uvedené zlúčenie šiestich dočasných súborov, F1, F2, F3, F4, F5 a F6. Najprv bude zlúčený súbor F1 a F2, následne súbory F3 a F4, a F5 s F6. Takto sa bude postupovať do vytvorenia jediného súboru, tak ako je naznačené na obrázku 4.6. Zlučovanie súborov rešpektuje usporiadanie n-gramov podľa ich početností.

Postup pre zlúčenie dvoch susedných súborov F1 a F2 do súboru F12 popisuje nasledujúci postup [2]:

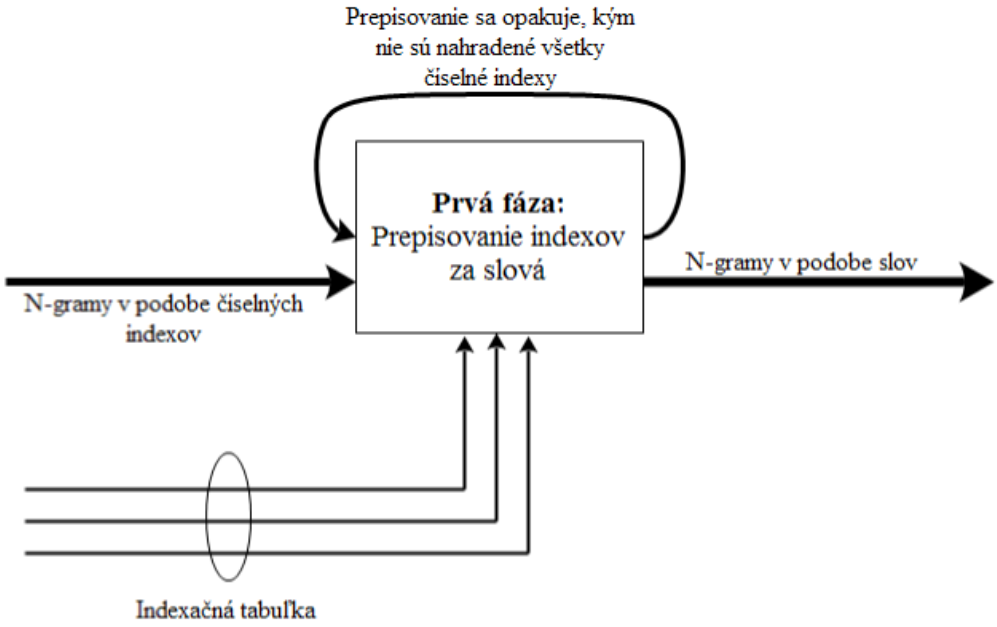
1. Načíta sa prvý n-gram so svojou početnosťou do premennej *a* zo súboru F1, a zároveň sa zo súboru F2 načíta n-gram s početnosťou do premennej *b.*
2. Porovná sa početnosť n-gramu *a* s početnosťou n-gramu *b,* kde n-gram s vyššou početnosťou je zapísaný do súboru F12.
3. Ak bola početnosť v *a* väčšia, nasledujúci n-gram sa načíta zo súboru F1. Ak súbor F1 neobsahuje už ďalšie n-gramy, je obsah súboru F2 prekopírovaný do výsledného súboru F12, v tomto prípade sa postupuje na bod číslo 5, inak sa algoritmus vráti do bodu 2.
4. Ak bola početnosť v *b* väčšia, nasledujúci n-gram sa načíta zo súboru F2. Ak súbor F2 neobsahuje už ďalšie n-gramy, je obsah súboru F1 prekopírovaný do výsledného súboru F12, v tomto prípade sa postupuje na bod číslo 5, inak sa algoritmus vráti do bodu 2.

5. Dosiahlo sa zlúčenie oboch súborov F1 a F2 do súboru F12 so zachovaným poradím n-gramov podľa početnosti. Súbory F1 a F2 sú zmazané z disku.

Po tejto fáze sú všetky n-gramy reprezentované číselným indexom so svojimi početnosťami uložené v jednom súbore, a je možný ich prevod na slová.

4.1.3 De-indexácia

Vstupom do tejto časti je súbor s n-gramami zoradenými podľa ich početnosti, ktorý bol vytvorený predchádzajúcou fázou. Hlavnou úlohou tejto fázy je prevod slovných indexov v n-gramoch na slová, ktoré boli takto zakódované v prvej časti. Činnosť de-indexácie je zobrazená na obrázku 4.8.



Obrázok 4.8: Schéma činnosti de-indexácie.

Táto fáza môže mať dva rôzne priebehy. V prvom prípade, pokiaľ sa do operačnej pamäte zmestí celá indexčná tabuľka, tak sa slová môžu priamo vypísať. Indexácia podľa (1) zaručuje, že po postupnom načítaní segmentov do tabuľky reprezentovanej napr. vektorom, príp. poľom, slová ležia vo vektore presne na pozícii, ktorá im bola vypočítaná. Tým pádom načítaním slovných indexov zo súboru s n-gramami vytvorenom v predchádzajúcej časti budeme indexovať indexačnú tabuľku a slová vypíšeme.

Druhý prípad nastane, pokiaľ sa celá indexačná tabuľka do operačnej pamäte nevojde. Ak takáto situácia nastane, tak do pamäte načítame niekoľko segmentov indexačnej tabuľky, vzhľadom na dostupnú operačnú pamäť, a de-indexáciu prevádzame vo viacerých iteráciach. V tomto prípade je nutné vhodne si označiť už vyriešené slovo od číselného indexu. Toto sa dá dosiahnuť napríklad

vložením znaku pred nevypočítaný index, ktorý sa v načítanom texte nemôže vyskytnúť. Keďže pri načítavaní znakov v prvej časti sa aplikuje filter prevádzajúci všetky písmená na malé, tak v našom prípade sú týmito znakmi veľké písmená. Týmto dosiahneme indikáciu nevyriešeného indexu, pri prevedenom indexe slovo uložíme do súboru s čiastočne prepísanými indexmi. Túto situáciu zachytáva obrázok 4.9.

N-gramy reprezentované číselnými indexami

slovné indexy			početnosť
19	3	4	3
8	4	9	2
8	4	11	1

Čiastočne prevedené n-gramy

A19	is	the	3
see	the	A9	2
see	the	A11	1

Kompletne prevedené n-gramy

this	is	the	3
see	the	lion	2
see	the	sky	1

Obrázok 4.9: Príklad čiastočného prevádzania slovných indexov na slová

Po dokončení indexácie sú na výstupe zobrazené slovné n-gramy zadanej dĺžky s ich frekvenciou výskytu v texte.

4.2 Implementácia

Implementácia popisovaného systému sa drží následnosti jednotlivých procedúr z kapitoly 4.1. Algoritmus bol vyvíjaný v jazyku C, pričom pre prácu s reťazcami a inými dátovými štruktúrami boli využité objekty zo štandardnej knižnice jazyka C++, dodržiavajúc pravidlá normy ISO C++98 a paradigmatá procedurálneho programovania.

V [2] bol systém implementovaný pre možnosť vyhľadávania n-gramov dĺžky 1 – 4, v jazyku C#. N-gramy sú tu ukladané do štruktúr, optimalizovaných pre ukladanie n-gramov danej dĺžky. Mnou implementovaný systém dokáže extrahovať n-gramy neobmedzenej dĺžky, vďaka ukladania n-gramov do dvojrozmerného poľa. Táto skutočnosť znamená mierne spomalenie oproti systému v [2], avšak takisto aj širšiu škálu použiteľnosti, s tým, že spomalenie nie je výrazné.

Pre implementáciu radenia bola zvolená posixová implementácia algoritmu quick sort, ktorá dosahovala najrýchlejšie výsledky radenia z testovaných možností.

4.3 Zrýchlenie činnosti systému

Systém je navrhnutý tak, aby jednotlivé kroky mohli byť vykonávané na jednom počítači, bez potreby využitia iných zdrojov. Keďže táto práca bola vyvíjaná v rámci výskumnej Skupiny

spracovania prirodzeného jazyka, boli mi pre vývoj dostupné viaceré stroje skupiny. Z toho dôvodu je vhodné navrhnúť a vyvinúť spôsob urýchlenia činnosti systému paralelizáciou niektorých jeho úloh. Tomuto problému sa venuje nasledujúca kapitola.

5 Paralelizácia

Riešenie výpočtových úloh v jednom slede na jednom procesore sa nazýva sekvenčné. Tento prístup je dostačujúci len pre niektoré typy úloh. Pokiaľ potrebujeme spracovať obrovské množstvo dát za krátky čas, je sekvenčné riešenie silno obmedzujúce. Z dôvodu urýchlenia výpočtov je čoraz častejšie využívaná paralelizácia.

Paralelný výpočet by sa dal definovať ako postup zamestnávajúci niekoľko na sebe nezávislých a vzájomne spolupracujúcich procesorov, za účelom rýchleho prevedenia rozsiahleho výpočtu. Po dekompozícii problému sú jednotlivé čiastkové úlohy rozdelené medzi dostupné procesory. Zoskupenie kompletných počítačov, pripojených do internetu, využívaných na riešenie paralelných úloh sa nazýva grid computing. Paralelizácia nachádza využitie napríklad v riešení úloh z grafiky, simulácii, bioinformatiky, spracovania obrazu, umelej inteligencie atď.

Na základe komunikácie medzi procesmi bežiacimi na procesoroch a správou ich pamäte existuje niekoľko modelov pre paralelné programovanie, medzi ne patria [14]:

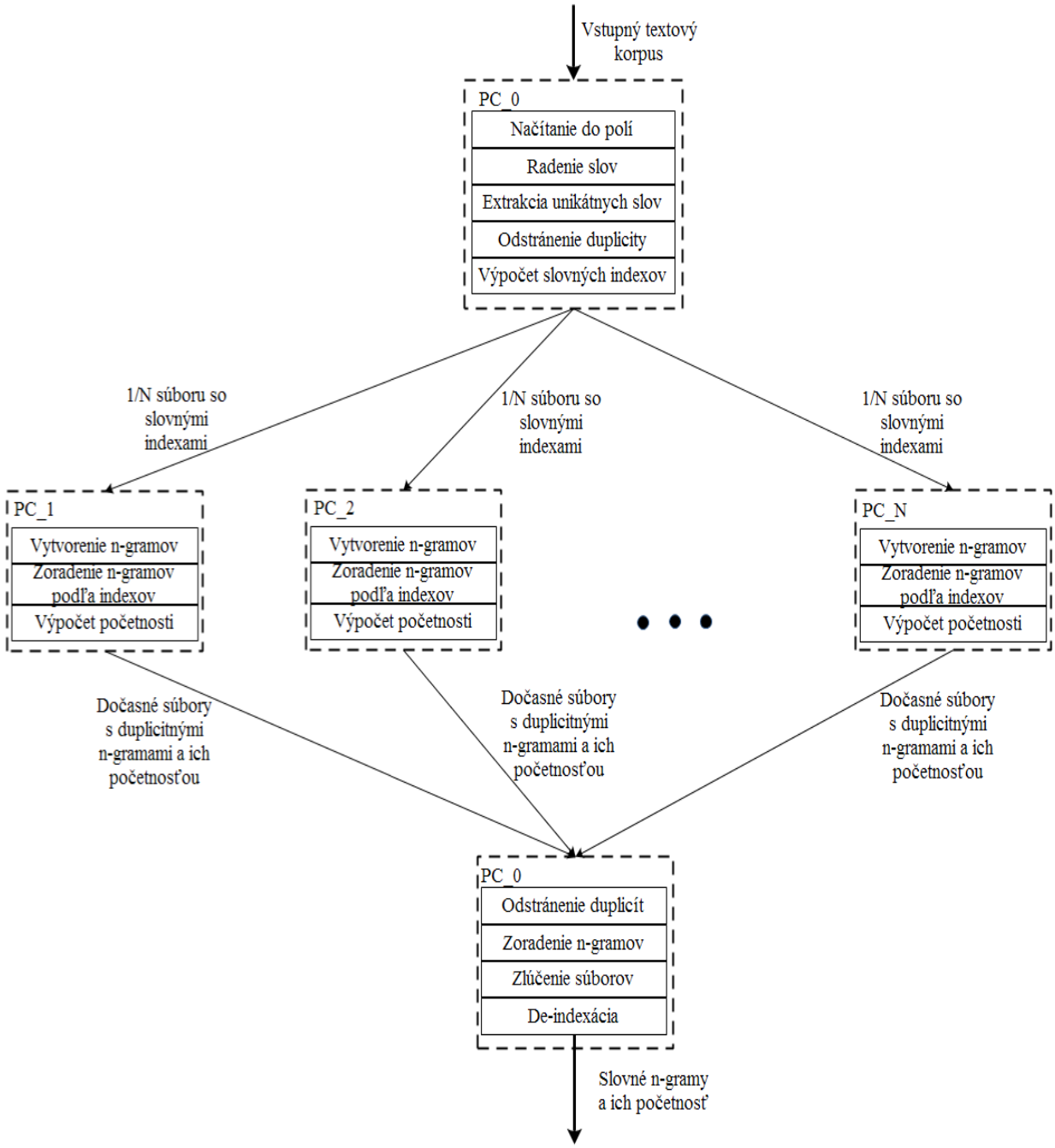
- posielanie správ: procesy využívajú len lokálnu pamäť a medzi sebou komunikujú pomocou zasielaných správ,
- paralelné dáta: každý proces pracuje s rozdielnou časťou rovnakej dátovej štruktúry, dáta sú na procesory distribuované,
- zdieľaná pamäť: viacero procesov zdieľa rovnakú časť pamäte,
- vzdialené operácie s pamäťou – sada procesov, v ktorých môže jeden proces pracovať s pamäťou druhého, na základe vlastnej réžie (neobmedzuje druhý proces),
- vlákna – jeden proces je realizovaný vo viacerých sledoch,
- kombinované modely – zložené z viacerých modelov.

5.1 Paralelizácia algoritmu Teraman

Pre paralelné algoritmy je v laboratóriach FIT dostupný SGE – Sun Grid Engine, nástroj na správu viacerých výpočtových zdrojov, gridov. Pre potrebu bakalárskej práce som zvolil využitie strojov výskumnej skupiny spracovania prirodzeného jazyka, hlavne kvôli jednoduchej réžii a problému, ktorý nevyžaduje natoľko sofistikované riešenie.

Najdôležitejším krokom ku paralelizovaní výpočtu bolo zvolenie čo najširšej množiny tých častí algoritmu, popisovaných v kapitole 4.1, ktoré môžu byť prevádzané na rôznych počítačoch. Tieto časti musia byť na sebe nezávislé, aby ich bolo možné prevádzať samostatne. Táto voľba musela takisto zohľadniť aj objem posielaných dát a frekvenciu posielania a to tak, aby boli dáta na

každý stroj posielané práve raz, prípadne v čo najmenšom možnom počte opakovaní, a takisto z tohto stroja čo najmenej krát prijímané. Využívaným modelom paralelného programovania je model paralelných dát. Dátová štruktúra, s ktorou jednotlivé procesy narábajú, je súbor obsahujúci indexy slov. Činnosť paralelizovaného riešenia je zobrazená na obrázku 5.1. PC_0 je tu reprezentovaný riadiaci počítač, na ktorom bol algoritmus spustený a ktorý má na starosti celkovú réžiu výpočtu.



Obrázok 5.1: Schéma paralelizovaného algoritmu.

5.1.1 Popis paralelizovaného systému

Ako je ilustrované na obrázku 5.1, pre paralelizáciu bola zvolená prvá, druhá a tretia fáza druhej časti algoritmu Teraman. Algoritmus si najprv overí dostupnosť počítačov zadaných v konfiguračnom súbore (viď príloha 2). Počet týchto strojov *N,* je následne uložený pre neskoršie použitie. Priebeh časti predspracovania a indexácie je takmer zhodný s priebehom popísaným v kapitole 4.1. Odlišuje sa len v tej časti, kedy je prevedených na slovné indexy viac ako 1/*N*

celkového počtu slov. V tomto momente je doposiaľ vytvorený súbor s vypočítanými slovnými indexmi odoslaný na spracovanie na stroj uvedený ako prvý v konfiguračnom súbore. Tento postup sa s ďalšími strojmi opakuje až do prevedenia všetkých slov na slovné indexy.

Na vzdialených počítačoch prebieha časť výpočtu početnosti n-gramov, konkrétne fáza vytvárania n-gramov, ich zoradenie podľa indexov a vypočítanie početnosti. Po vykonaní týchto operácií sú výsledné súbory odoslané nazad, na počítač riadiaci celkový výpočet. Tieto súbory obsahujú n-gramy spolu s ich početnosťou, avšak vyskytujú sa v nich aj duplicitné n-gramy.

Z dôvodu duplicity n-gramov v dočasných súboroch sa algoritmus, bežiaci na riadiacom počítači, po prijatí dočasných súborov dostáva do fázy odstraňovania duplikátov. Ďalší priebeh je zhodný s priebehom popísaným v kapitole 4.1.

Miera urýchlenia a porovnanie výkonnosti paralelizovaného riešenia so sekvenčným výpočtom je zhrnutá v kapitole 6.

5.1.2 Implementácia paralelne pracujúceho algoritmu

Systém je, tak ako sekvenčné riešenie, implementovaný v jazyku C, využívajúc niektoré objekty zo štandardnej knižnice jazyka C++, podľa normy ISO C++98. Réžia paralelizmu je prevádzaná pomocou skriptu písanom v unixovom príkazovom interprete – bashi. Na pripojenie využíva službu ssh, aby bola odtienená práca skriptu, je nutná výmena verejného kľúča užívateľa so vzdialenými strojmi na ktorých sa má algoritmus vykonávať (viď príloha 2). Toto zabezpečí pripojenie cez ssh bez nutnosti zadávania hesla.

Pred vlastným pripojením sú potrebné súbory kopírované pomocou služby scp. Po nadviazaní pripojenia je pomocou niekoľkých jednoduchých príkazov spustený kód na vzdialenom počítači vykonávajúci požadovaný výpočet.

Komunikácia so skriptom je v programe riešená pomocou funkcie *popen(command,...),* ktorá slúži na exekúciu príkazu, obsiahnutom v reťazci *command.* Takisto je tu veľmi jednoduché predávanie parametrov, ktoré obsahujú informáciu o názve vzdialeného stroja na pripojenie, dĺžke n-gramov atď.

6 Testovanie a výsledky

Prevádzanie testov bolo rozdelené do dvoch fáz, prvou fázou bolo testovanie a porovnávanie výkonnosti dostupných implementácií pre extrakciu n-gramov s algoritmom Teraman. Druhú fázu tvorilo porovnanie rýchlosti sekvenčného riešenia s paralelizovanou variantou Teramana. Ako vstupné dáta slúžili textové korpusy nachádzajúce sa na servery *minerva1.fit.vutbr.cz*. Testy prebiehali

na *pcnlp5*, ktoré nebolo zaťažované inými procesmi a všetky jeho zdroje boli plne dostupné. Konfigurácia *pcnlp5* je Intel Core 2 Duo E6750 2,66 GHz, 2 GB RAM, 214 GB veľkosť diskového priestoru a operačný systém CentOS 5.3.

6.1 Porovnanie výkonnosti viacerých metód

Porovnanie výkonnosti dostupných implementácií a algoritmu Teraman tvorilo prvú časť testovania. Podarilo sa mi vyhľadať tri vhodné, voľne dostupné, implementácie algoritmov na extrakciu n-gramov z textu. Ich zdrojové kódy ako aj spustiteľné binárne súbory sú zverejnené v [12]. Prvý algoritmus je založený na práci so sufixovým poľom, druhý pracuje so sufixovým stromom a zreťazeným zoznamom a tretí využíva takisto sufixový strom v kombinácii s hashovacou tabuľkou. Podrobnejšie porovnanie týchto prístupov navzájom je popísané v kapitole 3.5.

Všetky tri implementácie využívajú pre vytváranie slovníka minimalizáciu konečných automatov v kombinácií s perfektným hashovaním. Bližšie informácie o minimalizovaných konečných automatoch sú dostupné v [15].

Tabuľka 6.1 zachytáva porovnanie rýchlosti extrakcie bigramov z textu pomocou metódy využívajúcej sufixové pole, sufixový strom a algoritmus Teraman. Pre porovnanie týchto metód som zvolil maximálnu veľkosť vstupného korpusu 1 000 000 slov, z toho dôvodu, že čas potrebný pre exekúciu metódy používajúcej sufixový strom narastá príliš rýchlo, a na vzájomné názorné porovnanie týchto metód je to počet postačujúci.

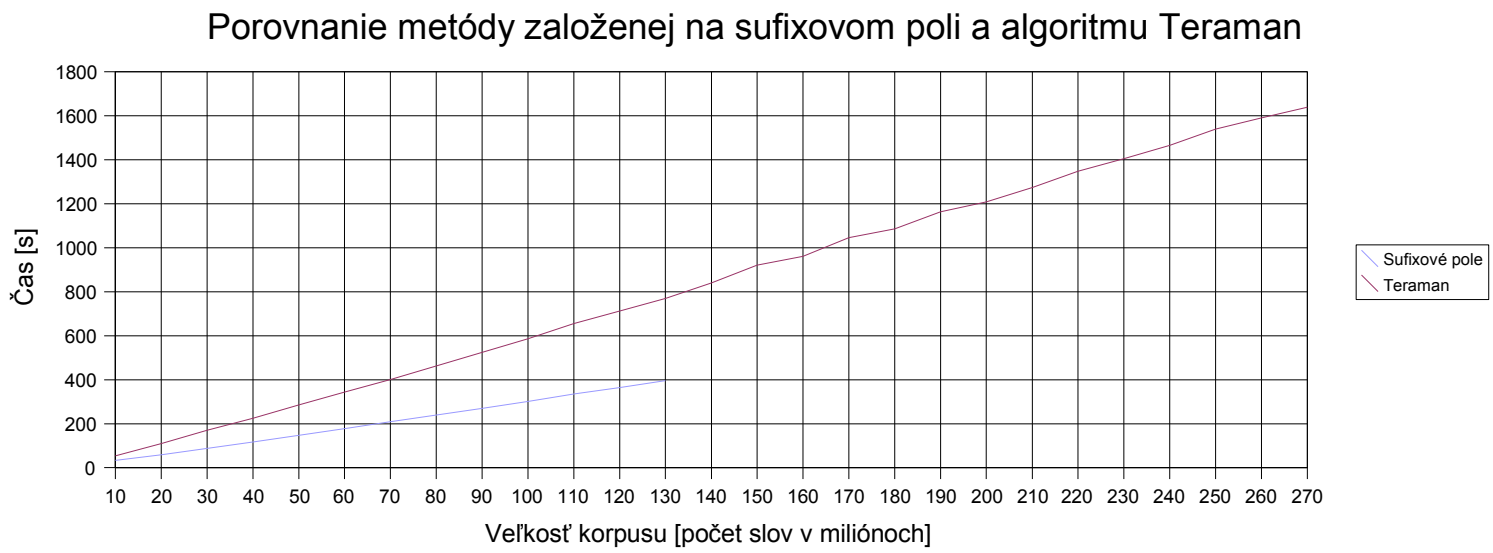
Z nameraných údajov vyplýva, že metóda používajúca sufixový strom v kombinácii so zreťazeným zoznamom je na extrakciu n-gramov neefektívna, naopak, veľmi rýchlou a efektívnou sa javí byť metóda pracujúca so sufixovým poľom. Prístup postavený na sufixovom strome s využitím hashovacej tabuľky nebol do meraní zahrnutý, pretože už pre milión slov nebola táto metóda schopná n-gramy extrahovať.

Veľkosť korpusu	**Čas extrakcie [s]**		
[počet slov]	Teraman	Sufixové pole	Sufixový strom
100000	0,87	0,46	8,45
200000	1,43	0,80	18,3
300000	1,95	1,14	29,09
400000	2,49	1,47	40,37

500000	2,99	1,83	51,43
600000	3,53	2,12	63,35
700000	3,92	2,44	79,18
800000	4,44	2,75	94,43
900000	5,09	3,05	112,16
1000000	5,60	3,36	130,46

Tabuľka 6.1: Porovnanie času extrakcie bigramov z textu skúmanými metódami.

Nasledujúci graf 6.1 znázorňuje časový priebeh extrakcie bigramov z textu pomocou sufixového poľa. Veľkosť textového korpusu bola periodicky zväčšovaná o 10 000 000, až kým táto metóda prestala byť schopná extrahovať bigramy z dôvodu nedostatočnej pamäte. Na grafe je ďalej znázornený časový priebeh extrakcie bigramov pomocou algoritmu Teraman. Vstupný korpus obsahoval 270 000 000 slov.



Graf 6.1: Časový priebeh metód extrahujúcich bigramy.

Z grafu 6.1 je možné vyčítať, že metóda využívajúca sufixové pole je takmer o polovicu rýchlejšia ako algoritmus Teraman. Jej časový priebeh však končí pri spracovaní 130 000 000 slov, z dôvodu nedostatočnej voľnej operačnej pamäte. Pri dostupnej pamäti o veľkosti 2 GB je pomocou metódy využívajúcej sufixové pole možné spracovať vstupný textový korpus s veľkosťou 722 MB, ktorá zodpovedá 130 000 000 slovám. Vyššia časová náročnosť algoritmu Teraman je spôsobená dávkovým spracovaním, ktoré zaručuje spracovanie textu ľubovoľnej veľkosti.

6.2 Porovnanie sekvenčného a paralelizovaného prístupu

Testovanie sekvenčnej verzie prebiehalo, ako aj predchádzajúce testy, na pcnlp5. Testy prebiehali na 40 GB vstupnom textovom korpuse, a počet slov z neho bol periodicky zvyšovaný pre dostačujúci počet testov. V grafe 6.2 je zaznamenaná spotreba času sekvenčnou a paralelnou verziou programu, pre korpusy rôznych veľkostí, kde maximálna veľkosť bola päť miliárd slov, čo zodpovedá textovému súboru veľkosti približne 40 GB. Pri paralelizovanom výpočte bolo využívaných päť vzdialených staníc.

Graf 6.2: Časový priebeh metód extrahujúcich bigramy.

Miera urýchlenia výpočtu paralelizovaným prístupom je nepriamo úmerná počtu slov. S narastajúcim počtom slov sa pomer urýchlenia znižuje. Tento fakt je spôsobený tým, že pri vyššom počte slov rastie spotreba času časťou algoritmu, ktorá ma na starosti odstraňovanie duplicity, čo je prevádzané na jednom, riadiacom, počítači. Pri testovaní varianty, kde boli odstránené duplicity zo všetkých dočasných súborov v rámci jedného vzdialeného počítača a až následne odoslané naspäť na riadiacu stanicu, som zistil, že výpočet, oproti pôvodnej navrhnutej variante, nebol urýchlený.

7 Záver

Cieľom práce bolo navrhnúť a implementovať systém, ktorý dokáže efektívne vyhľadávať n-tice slov v rozsiahlych textoch. Tento systém dokáže spracovať textové korpusy neobmedzenej veľkosti vďaka dávkovému spracovaniu. Jeho implementácia je efektívna vzhľadom na prácu s diskom. Po implementácii tohto systému pracujúceho na jednom počítači som navrhol mierne urýchlenie algoritmu pomocou paralelizácie dávkového spracovania medzi viacero vzdialených počítačov. Implementácia oboch variant systému je v jazyku C, kde pre efektívnu prácu s reťazcami sú využívané niektoré objekty štandardnej knižnice jazyka C++. Paralelizácia je riešená pomocou skriptu, ktorý na distribúciu dát a komunikáciu so vzdialenými počítačmi využíva protokol Secure Shell (ssh).

Pre extrakciu slovných n-gramov existuje viacero metód, ktoré sú podrobne popísané v kapitole 3 tejto práce. Z porovnania ich výkonností vyplýva, že metóda postavená na používaní sufixového poľa ako dátovej štruktúry na uloženie n-gramov, je veľmi rýchla a efektívna. V porovnaní s implementovaným systémom je rýchlejšia o viac ako 40%. Jej nevýhoda je neschopnosť spracovať vyššie množstvo textu, ako sa zmestí do operačnej pamäte, tým pádom je využitie tejto metódy značne obmedzené. Naopak, implementovaný systém zvládne pracovať so súbormi ľubovoľnej veľkosti. Avšak táto vlastnosť je z väčšej miery zodpovedná za pomalšiu extrakciu, ako metóda so sufixovým poľom. Metóda využívajúca sufixový strom sa ukázala ako najmenej efektívna, kde pri textovom korpuse obsahujúcom stotisíc slov dosahovala niekoľkokrát vyšší čas exktrakcie ako ostatné skúmané metódy.

Paralelizáciou algoritmu Teraman je možné znížiť čas potrebný na jeho exekúciu o 15% – 25%, v nepriamej závislosti od počtu slov vstupného textu. Miera urýchlenia klesá s narastajúcim počtom slov vstupného korpusu z dôvodu zvyšujúcich sa nárokov fáz prevádzaných sekvenčne.

Práca by sa dala rozšíriť o vyhľadávanie nespojitých n-tíc v texte, teda tých, ktoré nenasledujú bezprostredne za sebou, ale vyskytujú sa v jednej vete v rámci kontextu.

Výsledný implementovaný systém tejto práce môže byť využitý ako aparát pre získavanie početnosti slov v rozsiahlych dokumentoch, ktoré sa ďalej využívajú pre široké spektrum úloh spracovania prirodzeného jazyka.

Literatúra

[1] 2009, NLP group, University of Sheffield [online]. 13 December 2006 [cit. 2009-04-22]. Dostupný z WWW: <http://nlp.shef.ac.uk/>.

[2] Proceedings of the 7th Annual Conference ZNALOSTI 2008, Bratislava, Slovakia, pp. 54-65, February 2008. ISBN 978-80-227-2827-0.

[3] Wikipedia: N-Gram [online]. 22:47, 14 September 2004, 9 April 2009, at 05:14 [cit. 2009-04-23]. Dostupný z WWW: <http://en.wikipedia.org/wiki/N-gram>.

[4] URAFSKY, Daniel, MARTIN, James H. Speech And Language Processing : An Introduction To Natural Language Processing, Computational Linguistics, And Speech Recognition. 2nd edition. Upper Saddle River: Prentice Hall, 2008. 1024 s. Dostupný z WWW: <http://books.google.com/books?id=fZmj5UNK8AQC&dq=Speech+and+language+processing+:an+introduction+to+natural+language+processing&printsec=frontcover&source=bl&ots=LqS8_-HLQI&sig=0hNFclP0wlsKmjUtfyShEm437ws&hl=en&ei=sjrvSZaHCImI_QbE_cjDDw&sa=X&oi=book_result&ct=result&resnum=9>. ISBN 0-13-504196-1.

[5] Proceedings of the ITAT 2008, Information Technologies - Applications and Theory, Hrebienok, Slovakia, pp. 23-26, September 2008. ISBN 978-80-969184-8-5

[6] FRANZ, Alex, BRANTS, Thorsten. Official Google Research Blog : All Our N-gram are Belong to You [online]. Thursday, August 03, 2006 at 8/03/2006 11:26:00 AM [cit. 2009-04-23]. Dostupný z WWW: <http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html>.

[7] M. Nagao and S. Mori. A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese. *In Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, Kyoto, Japan, 1994.

[8] Wikipedia : Lempel-Ziv-Welch [online]. 17:18, 8 October 2001, 26 April 2009, at 09:15 (UTC) [cit. 2009-04-27]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Lempel-Ziv-Welch>.

[9] MOSTAFA, Hatem. The code project : N-gram and Fast Pattern Extraction Algorithm [online]. 10 Sep 2007 , 31 Oct 2007 [cit. 2009-04-29]. Dostupný z WWW: <http://www.codeproject.com/KB/recipes/Patterns.aspx>.

[10] TANG , Haixia. A Suffix Array Based N-gram Extraction Algorithm [online]. November 2005 [cit. 2009-04-27]. Dostupný z WWW: <http://flame.cs.dal.ca/~htang/proj/suffixngram.html>.

[11] Wikipedia : Suffix tree [online]. 18:16, 7 July 2004 , 25 April 2009, at 23:52 (UTC) [cit. 2009-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Suffix_tree>.

[12] KOZLOWSKI, Sebastian. Ngram counting solution comparison [online]. December 2003 [cit. 2009-04-28]. Dostupný z WWW: <http://nlp.strefa.pl/ngrams/linux/index.htm>.

[13] Wikipedia : Binary search algorithm [online]. 18:44, 2 July 2006 , 04:11, 1 May 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Binary_search>.

[14] Maui High Performance Computing Center. SP Parallel Programming Workshop : parallel programming introduction [online]. Sat, 06 Sep 2003 03:25:03 GMT [cit. 2009-05-02]. Dostupný z WWW: <http://www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html>.

[15] DACIUK, Jan. Jan Daciuk : Personal homepage [online]. [cit. 2009-05-05]. Dostupný z WWW: <http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/>.

Zoznam príloh

Príloha A: návod na preklad, použitie a spustenie sekvenčnej verzie programu.
Príloha B: návod na preklad, použitie a spustenie paralelnej verzie programu.
Príloha C: CD obsahujúce zdrojové súbory paralelnej aj sekvenčnej verzie programu, Makefile, konfiguračný súbor, skripty potrebné pre paralelnú verziu, súbor README s návodom a text technickej správy v pdf formáte.