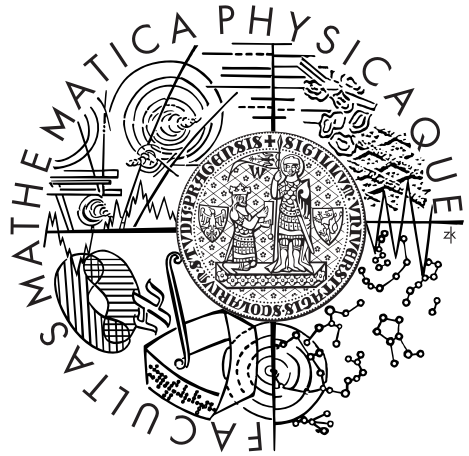
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta
DIPLOMOVÁ PRÁCE



Tomáš Urban

Semi-adaptivní slovníkové kompresní metody

Katedra softwarového inženýrství
Vedoucí diplomové práce: Mgr. Jan Lánský
Studijní program: Informatika, softwarové systémy
2008

Rád bych na tomto místě poděkoval vedoucímu diplomové práce Mgr. Janu Lánskému za ochotu a čas, který mi při psaní této práce věnoval.

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

V Praze, dne 6.12.2008

Tomáš Urban

Obsah

Název práce: Semi-adaptivní slovníkové kompresní metody
Autor: Tomáš Urban
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: Mgr. Jan Lánský
e-mail vedoucího: lansky@ksi.ms.m.cuni.cz

Abstrakt: Cílem této diplomové práce bylo navrhnout a experimentálně ověřit postupy vedoucí k vytvoření a kompresi slovníku ze vstupního souboru případně proudu dat, tak aby vstupní soubor po rozdělení na elementy tohoto slovníku byl optimalizován pro následné kompresní algoritmy. Všechny uvažované metody procházejí vstup nadvakrát. Informace o vstupu získané při prvním průchodu využívají k zefektivnění druhého průchodu. Vytvořený slovník je součástí komprimovaného výstupu, proto je velmi důležitá jeho velikost. První část naší práce popisuje rozšíření již existujících metod TD2 a TD3[4] pro použití nad neomezenou abecedu. Metodu TD2 využíváme k uložení slovníku. Druhá část zkoumá možnosti vytvoření optimálního slovníku a způsob rozdělení vstupu pomocí toho slovníku bez rozlišování slov, slabik. Všechny operace jsou uvažovány nad relativně neomezenou abecedou unicodu. Pro experimenty byl použit modulární program XBW, navržené postupy byly realizovány a otestovány jako modul toho programu.

Klíčová slova: komprese, trie, slovník

Title: Semi-adaptivni dictionary compression methods
Author: Tomáš Urban
Department: Department of Software Engineering
Supervisor: Mgr. Jan Lánský
Supervisor's e-mail address: lansky@ksi.ms.m.cuni.cz

Abstract: Goal of this work was to design and test methods for creating and parsing input data from file or stream in a way they are optimalized for following compression algorithms. We used two runs through data, first run collected analytical data and second run was for parsing. First part of this document is focused on methods TD2 and TD3 [4] for trie compression and their extension for unicode alphabets. In second part there are presented methods for creating dictionary without considering partitiong on words or syllabels. All methods are considered over unicode alphabet and implemetned as part of modular program XBW.

Keywords: compression, trie, dictionary

Kapitola 1

Úvod

U některých typů dokumentů můžeme výrazně zkrátit čas komprese a zlepšit kompresní poměr předzpracováním vstupních dat. Může to být zaměněním logických nebo syntaktických celků za jednotky. Nejběžnějším předzpracováním je rozdělení na slova, slabiky, či v případě strukturovaných dokumentů, na syntaktické elementy. Tato dělení jsou ve většině kompresních programů dána dopředu, méně častěji zvolena uživatelem před zahájením komprese nebo vybrána na základě hlavičky či koncovky souboru. Pokud tedy uživatel nebo program špatně zvolí typ souboru, není zaručeno, že bude použito vhodné dělení. Naším cílem bylo najít takové dělení, které by nebylo závislé na znalosti struktury ani syntaxe jazyka vstupu a zároveň by bylo co nejvýhodnější z hlediska následné komprese. Motivací pro naše zkoumání byla hypotéza, že rozdělení vstupu na základě předešlé analýzy bude dosahovat lepších výsledků než pevně daná dělení. Nevýhodou, která bude zřejmě ve většině případů bránit praktickému nasazení, je nutnost procházet vstupní data nadvakrát.

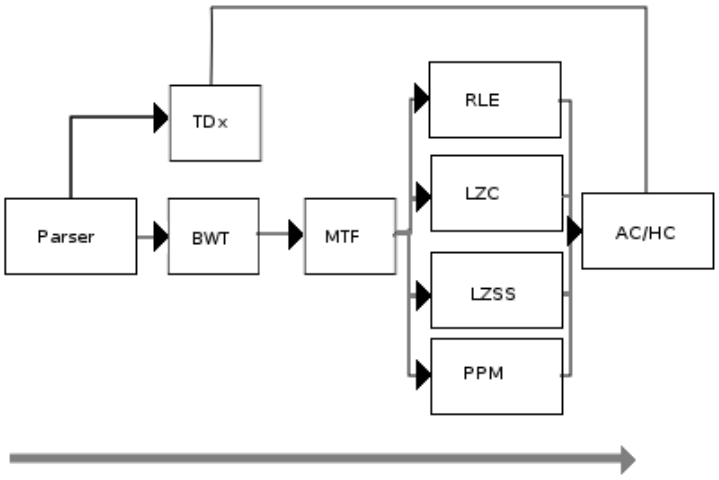
Postup, který jsme zvolili k získání výsledků, byl následující. Nejprve jsme vytvořili úplný slovník, slovník zpočátku obsahoval všechny možné řetězce ze vstupu do určité délky a četnosti jejich výskytu. Po analýze slovníku jsme navrhli několik optimalizací a kriterií, jež by měly řetězce slovníku splňovat. Podle těchto kriterií jsme slovník redukovali a zkoumali, jaký mají vliv na celkovou kompresi při použití hladového parseru. V další fázi jsme hladový parser nahradili parserem s výhledem.

Pro uchování slovníku jsme zvolili strukturu trie, ta přirozeně reprezentuje stavbu slovníku, sama o sobě je poměrně úsporná pokud jde o místo a umožnila nám provádět nejčastější operace nad slovy velice rychle. Vytvořený slovník je pro každý vstup jiný, proto je nutné, aby byl také dobře zkomprimován a mohl být součástí výstupu. Ke kompresi trie jsme použili metod TD autorů Lánského a Žemličky [4]. Tyto metody jsme museli rozšířit pro použití nad abecedou unicodu. Implementovali jsme verzi TD2 i TD3 pro unicode. Všechny navržené postupy jsme implementovali jako modul pro modulární program XBW, ten také posloužil pro testovací účely.

Kapitola 2

XBW

XBW [5] je bezztrátový kompresní program optimalizovaný pro kompresi nevalidních XML souborů. Program byl vyvinut pro systém Egothor, fulltextový vyhledávač, který indexuje a uchovává velké objemy HTML dokumentů. Program XBW je složen z jednotlivých modulů, zapojení těchto modulů je možné ovlivnit pomocí parametrů předávaných programu. Architektura programu je na obrázku 2.



Obrázek 2.1: Architektura programu XBW.

Parser umožňuje práci v několika režimech. Pomocí parametrů příkazové řádky lze zapnout režim parsování textu nebo režim parsování XML struktury, přičemž není vyžadován validní dokonce ani dobře formovaný XML vstup. V režimu XML zkracuje vstup tím, že vytváří dynamicky slovník XML značek, atributů a vynechává ukončovací znaky, text mezi značky parsuje na znaky, slabiky nebo slova stejně jako v textovém režimu parsování. Naparsované řetězce předá slovníkovému kompresoru, který každému slovu přidělí identi-

fikátor a slovník uloží. Další operace jsou již jen nad polem těchto identifikátorů.

Slovník je uschováván v paměti ve struktuře trie a po ukončení parsování je uložen metodami TDx. Implementovány jsou metody TD2 a TD3 v kombinaci se statickým i adaptivním Huffmanovým i aritmetickým kodérem. Zapojení slovníku téměř ve všech případech vylepší kompresní poměr, ale prodlužuje čas běhu. Jen v případě zapnuté transformace vynahradí čas ušetřený zkrácením vstupu čas na práci se slovníkem. Parser je schopen pomocí knihovny `iconv` zpracovat velké množství kódování vstupu.

Při výchozím nastavení po parseru následují metody Burrows-Wheelerovy transformace a MTF (move to front). Burrows-Wheelerova transformace je implementována v několika variantách, jejich volba neovlivňuje poměr komprese, ale pouze a jen rychlost zpracování. Výhodou XBW oproti jiným kompresním programům je to, že provádí BWT nad celým souborem najednou. MTF nahrazuje výstupní symboly z BWT indexy, přičemž naposledy použitý symbol staví na začátek seznamu, ze kterého indexy bere. Výsledkem je posloupnost opakujících se nul a malých čísel.

Po metodách upravujících entropii vstupu následují kompresní algoritmy. Výchozí je použití RLE, jež zaměňuje posloupnosti stejných symbolů za symbol a počet jeho opakování. Dále jsou implementovány algoritmy LZC a LZSS, vycházející z algoritmů LZ78 a LZ77. LZC zapisuje na výstup index slova ve slovníku, který vytváří za běhu. LZ77 hledá v již zpracovaném textu nejdelší shodu s řetězcem na vstupu a jeho výstup je pozice shody a její délka. Tyto algoritmy jsou poměrně rychlé, nepotřebují mnoho paměti a s jejich použitím XBW dosahuje podobných výsledků jako `gzip`.

Pro zápis finálního výstupu je použit kodér. Implementován je aritmetický i Huffmanův kodér, oba v statické i adaptivní verzi. Aritmetický kodér je implementován s Moffatovou datovou strukturou, Huffmanův kodér v kanonické verzi. Volba kodéru je parametrem při kompilaci XBW, není možné ji měnit pomocí parametrů příkazové řádky před spuštěním. Aritmetický kodér se osvědčil jako rychlejší a zároveň efektivnější, proto je nastaven jako výchozí volba.

Výchozí nastavení programu používá slabikový parser, XBW, MTF, RLE a statický aritmetický kodér.

Kapitola 3

TD2 a TD3

3.1 Trie

Trie [3] je stromová datová struktura, ve které každý uzel představuje jeden znak. Každý uzel může mít maximálně tolik synů, kolik je prvků použité abecedy. Položky slovníku jsou reprezentovány jako cesta začínající v kořeni trie, reprezentované slovo vznikne spojením znaků uzlů na této cestě. Tedy pokud je uzel A synem uzlu B, slovo představované uzlem A vznikne spojením slova, které představuje uzel B a znaku u uzlu A. Pro všechna slova reprezentovaná trií platí následující: pokud mají 2 slova stejný prefix délky n, cesty, které je reprezentují, mají n prvních uzlů stejných. Kořen trie představuje prázdný znak. Jednotlivé uzly obsahují:

- hodnotu znaku, který zastupují,
- počet svých synů,
- pole ukazatelů na své syny,
- příznak určující zda se jedná o slovo ze slovníku.

3.1.1 Rozšíření trie pro neomezené abecedy

Každý uzel trie může obsahovat v případě práce s abecedou unicodu teoreticky tolik synů, kolik je znaků této abecedy, zároveň v praxi většina uzlů má pouze několik desítek synů v závislosti na jazyku textu. Je proto výhodné pole synů reprezentovat jako hashované pole s proměnlivou délkou. Na začátku práce s trii pole odkazů synů alokujeme na výchozí velikost N, pokud přesáhne počet synů $N/2$ prvků, pole zvětšíme na velikost $2N$. Prvky v poli umísťujeme pomocí funkce H mod M, kde H je hodnota znaku syna a M je velikost pole, viz Algoritmus 1.

Algoritmus 1 Vkládání do trie.

```
void vlož_do_trie(kořen_trie, vkládané_slovo,délka_slova)

t=kořen_trie
nalezeno=false
for (i = 0; i < délka_slova; i++) do
  index=vkládané_slovo[i];
  if (t.počet_synů >= (t.synů_max / 2)) then
    rozšiř_pole(t,t.synů*2)
  end if
  for (j = index mod t.synů_max; t;j=(j+1) mod t.synů_max) do
    if (t.pole_synů[j]==NULL) then
      break;
    end if
    if  (t.pole_synů[j]->data == index)  then
      t=t.pole_synů[j];
      t.četnost++;
      nalezeno=true
      break;
    end if
  end for
  if (!nalezeno)  then
    založ_nový uzel trie a přidej ho do pole synů t.
  end if
end for
```

3.2 TD1

3.2.1 TD1 kodér

Zkratka TD zastupuje anglický výraz Trie-based Dictionary compression, což je volně přeloženo komprese slovníku pomocí struktury trie. Tato metoda byla navržena Lánským a Žemličkou [4], popisuje převod trie z operační paměti do souboru či proudu dat a zpět. Na straně kodéru i dekodéru předpokládá znalost struktury a pravidel pro tvorbu trie. Kodér trii prochází nejprve do hloubky a u každého uzlu zakóduje počet jeho synů, hodnotu True/False značící, zda uzel představuje slovo nebo ne, hodnotu znaku, který představuje první syn a pro další syny jen rozdíly jejich hodnot. Nejjednodušší variantu představuje Algoritmus 2.

Algoritmus 2 Komprese trie

```
kodér(uzel_trie){
zapiš(trie.počet synů);
seřaď_podle_hodnoty_znaku(trie.pole_synů);
hodnota=0;
for syn=trie.pole_synů[i];i<trie.počet_synů;i++ do
  zapiš(syn.hodnota-hodnota)
  hodnota=syn.hodnota
  if syn.Rep then
    zapiš(TRUE)
  else
    zapiš(FALSE)
  end if
  kodér(syn)
end for
}
```

3.2.2 TD1 dekodér

Dekodér využívá dvou zásobníků, jeden zásobník slouží pro ukládání počtu synů na cestě od kořene trie až po právě zpracovávaný uzel, druhý představuje znaky uzlů na této cestě. Podle poslední hodnoty na zásobníku s počtem synů vždy víme, zda bude následující načtený uzel syn nebo bratr předchozího. Kladná hodnota znamená, že následující uzel je syn, 0 značí bratra. Nulu ze zásobníků počtu synů po přečtení odstraníme, kladnou hodnotu snížíme o 1. Viz pseudokód Algoritmu 3.

3.3 Metoda TD2 a její rozšíření pro unicode

Metoda TD2, jak je popsána autory Lánským a Žemličkou [4], využívá poznatku, že některé skupiny znaků se vyskytují vedle sebe častěji. Tato optimalizace přerovnává znaky abecedy tak, aby znaky těchto skupin bylo blízko u sebe, a při kompresi kóduje místo rozdílů hodnot znaků rozdíly v pořadí znaků. Pro použití nad abecedou unicodu jsme tuto metodu museli upravit. Různé národní abecedy mohou mít naprosto odlišné skupiny znaků, které se vyskytují často spolu. Ve vstupním souboru nemusí být ani všechny symboly zastoupeny, proto je třeba nejprve zjistit, jak velké jsou jednotlivé skupiny. Příslušnost znaku do skupiny určíme pomocí předgenerované tabulky pro všechny znaky unicodu. Při vytváření trie ukládáme do hashovaného pole všechny různé znaky a před kódováním trie tuto množinu rozdělíme na skupiny:

- malých písmen,
- velkých písmen,

Algoritmus 3 Dekomprese trie

```
trie_dekodér(vstup){
načti_kořen_trie()
if kořen_trie.počet_synu then
  zásobník_poctu_synu.přidej( kořen_trie.počet_synu)
  while zásobník_poctu_synu.délka do
    if zásobník_poctu_synu.poslední_hodnota>0 then
      zásobník_počtu_synů.poslední_hodnota−;
    else
      odeber všechny 0 hodnoty z zásobníku_počet_synů a o stejný počet hodnot zkraťzásobník_hodnot_synů
    end if
    syn=načti_uzel()
    zásobník_hodnot.přidej(syn.hodnota);
    zásobník_počtu_synů.přidej(syn.počet_synů);
    }
  end while
end if
```

- číslic,
- symbolů,
- ostatních znaků.

Každé této skupině přidělíme hodnotu H_p pro první prvek, kde H_p je suma počtu prvků předchozích skupin. Pořadí skupin volíme libovolně. V rámci skupin přidělujeme hodnoty prvkům vzestupně podle četnosti výskytu v textu, první prvek ve skupině získá hodnotu H_p, následující o jedna větší. Tuto hodnotu používáme při kódování trie místo hodnoty představovaného symbolu.

3.4 Metoda TD3

Tato optimalizace je také navržena autory Lánským a Žemličkou [4], vychází z metody TD2, přidává podmínku na parser. Všechny výstupní řetězce z parseru musí být tvaru:

- První symbol velké písmeno a následují pouze malá písmena.
- Celý řetězec z velkých písmen.
- Celý řetězec z malých písmen.
- Celý řetězec z číslic.

- Celý řetězec z symbolů.
- Celý řetězec z ostatních znaků.

Za této podmínky můžeme při kódování trie pro uzly ležící od druhé úrovně níž směrem od kořene odhadnout z jaké skupiny znaků bude jejich první syn. Pokud víme, že znak, který představuje prvního syna právě kódovaného uzlu, musí být například malé písmeno, můžeme namísto celé hodnoty tohoto znaku zakódovat pouze rozdíl mezi prvním znakem ve skupině malých písmen a touto hodnotou.

3.4.1 Porovnání TD2 a TD3

Ke srovnání obou metod jsme použili program XBW ve výchozím nastavení a testovací množinu souborů popsanou na straně 27 v sekci 5.2.

Sloupec tabulky 3.1 označený jako K.p.TD2 představuje procentuální velikost výstupu oproti velikosti vstupního souboru při použití TD2, následující sloupec analogicky pro TD3. Další dva sloupce udávají kolik procent z velikosti zkomprimovaného souboru zabírá slovník při kompresi pomocí metody TD2 a TD3. V posledním sloupci je v bytech vyjádřen rozdíl velikostí slovníku při použití obou metod. Všechny hodnoty jsou v posledním sloupci kladné, což znamená, že s TD3 jsme vždy ušetřili místo. V tabulce není uveden čas běhu. Ten byl pro obě metody stejný, ze stejného důvodu není uveden kompresní poměr bez započítaného slovníku, protože jsme v obou případech použili stejný parser. Průměrná hodnota je počítána jako průměr jednotlivých hodnot, nejedná se o vážený průměr přes velikosti.

Soubor	Velikost [MB]	K.p.TD2	K.p.TD3	P. slov TD2	P. slov TD3	rozdíl
CZ01	23,46	10,77%	10,76%	1,91%	1,85%	1647
CZ02	22,15	9,53%	9,53%	2,00%	1,94%	1334
CZ03	19,22	10,32%	10,31%	2,24%	2,18%	1280
CZ04	16,97	9,27%	9,26%	2,39%	2,32%	1073
CZ05	18,1	10,45%	10,44%	2,81%	2,76%	980
CZ06	18,55	7,78%	7,78%	2,62%	2,52%	1498
CZ07	18,42	8,33%	8,32%	3,01%	2,96%	839
CZ08	18,75	9,07%	9,06%	2,83%	2,75%	1360
CZ09	19,21	7,84%	7,83%	3,62%	3,55%	1208
CZ10	18,22	8,08%	8,07%	2,37%	2,30%	1085
EN01	13,82	10,51%	10,50%	2,04%	1,94%	1527
SL01	20,07	8,38%	8,38%	2,36%	2,29%	1182
SL02	27,17	5,66%	5,66%	2,85%	2,77%	1296
SL03	23,48	4,23%	4,23%	3,31%	3,20%	1103
SL04	22,67	7,97%	7,97%	2,60%	2,52%	1371
SL05	22,81	4,31%	4,31%	2,11%	2,03%	906
SL06	17,64	4,43%	4,42%	2,66%	2,51%	1206
SL07	23,55	7,27%	7,26%	2,76%	2,71%	975
SL08	17,29	8,94%	8,94%	2,41%	2,32%	1518
SL09	19,12	6,44%	6,44%	2,31%	2,20%	1399
SL10	21,22	8,39%	8,38%	2,40%	2,35%	978
SI01	32,0	4,41%	4,41%	0,27%	0,26%	78
SI02	39,54	19,17%	19,17%	0,45%	0,42%	2108
Průměr	**22495913,96**	**8,33**	**8,32**	**2,36**	**2,29**	**1215**

Tabulka 3.1: Výsledky TD2 a TD3 rozšířené nad unicode.

Kapitola 4

Prořezání trie

4.1 Úvod

Vše, co bylo v předchozích kapitolách popsáno má souvislost s ukládáním slovníku ze struktury trie do souboru. V této kapitole jsme se zaměřili na tvorbu slovníku, optimalizaci a také na způsob, jakým je vstup dělen na slova slovníku. Prvním úkolem bylo určit vhodnou maximální délku řetězců ve slovníku, hloubku trie, poté jsme zkusili navrhnout a otestovat metodu na prořezání slovníku tak, abychom odstranili řetězce, které nejsou použity ve výstupu nebo jsou použity s malou četností. Poslední části jsme probrali možnosti při dělení vstupu na elementy slovníku nejprve pomocí hladového parseru, poté jsem otestovali parser s výhledem.

4.2 Tvorba slovníku, PA parser

Pro naše účely bylo třeba implementovat vhodný parser, protože v programu využíváme více typů parserů, nazvali jsme tento PA. Parser PA slouží při prvním průchodu vstupem k získání všech různých podřetězců délky 1 až maximální délky slova-hloubky trie. Implementovaný parser PA prochází vstupní soubor pomocí zásobníku sekvenčně po znacích, do trie ukládá slova délky jedna až maximální definovaná délka slova počínaje od právě zpracovávané pozice. Pro každý podřetězec uložený do trie chceme znát i jeho četnost, proto jsme strukturu trie rozšířili o proměnnou frekvence, při vytvoření uzlu v trii ji pro nový uzel nastavíme na 1, pro každý další výskyt řetězce, jež představuje tento uzel, zvedneme její hodnotu o jedna. Po prvním průchodu vstupu máme tedy naplněnou trii všemi podřetězci, které lze v souboru najít a u každého známe jeho četnost. Pro všechny cesty v trii začínající v kořeni trie platí:

- Posloupnost četností jednotlivých uzlů od kořene k listům je nerostoucí.
- Trie má hloubku H - max délka slova při parsování.

- Pro všechny úseky libovolné cesty od kořene k listu platí, pokud první uzel tohoto úseku má stejnou četnost jako poslední, platí pro všechny řetězce reprezentované cestou od kořene do jednoho z uzlů toho úseku, že se ve vstupním textu vyskytují pouze jako podřetězec řetězce reprezentovaného uzly na cestě od kořene do nejvzdálenějšího uzlu z toho úseku.
- Počet synů kořene trie odpovídá velikosti abecedy vstupního souboru.
- Součet četností všech uzlů v trii je roven délce vstupu násobené maximální délkou slova v trii.

4.3 Hladový parser

Další typ parseru, který jsme implementovali, byl hladový parser, je použit pro druhý průchod vstupním souborem. Hledá nejdelší shodný řetězec v trii s řetězcem na vstupu od aktuální pozice dál. Jeho výhodou je velmi rychlý běh, má lineární složitost s délkou vstupu, a snadno se implementuje. Nevýhodou hladového parseru je možnost zablokování se. Pokud parser nenajde odpovídající řetězec v trii pro řetězec na vstupu, neumí se vrátit a zkusit jinou kombinaci a zasekne se. Parsování na co nejdelší možné úseky nemusí být vždy nejlepší způsob, my se pokusíme upravit trii tak, abychom s hladovým parserem dosáhli co nejlepších výsledků. Náš hladový parser je implementován jako while cyklus, který probíhá dokud je možné číst ze vstupního souboru, vstup porovnává s trii a pokud narazí na konec v trii odešle na výstup ukazatel na poslední procházený uzel trie a posune se ve vstupu o hloubku toho uzlu.

4.4 Optimalizace

4.4.1 Hloubka trie

Základním parametrem charakterizujícím trii je její hloubka. Pokud bychom nad trii neprováděli žádné optimalizace, tak hloubka trie exponenciálně ovlivňuje velikost slovníku a zároveň při použití hladového parseru nepřímo úměrně zkracuje délku výstupu z parseru. První experimenty s hladovým parserem bez optimalizací ukázaly, že se s rostoucí hloubkou trie zhoršuje jak kompresní poměr výstupního souboru, tak i kompresní poměr výstupu bez započítaného slovníku i přestože délka výstupu z parseru je podstatně kratší. Tento narůst lze vysvětlit velikostí abecedy, tedy různých listů trie, které jsou předány z parseru dalším modulům XBW. Abeceda, která vstupovala z parseru neměla od sta do tří set znaků jako vstupní soubory, ale až řádově miliony znaků v závislosti na hloubce trie. Přesné výsledky jsou uvedeny v abulce 4.1 na straně 17, kde je také vidět, že s rostoucí hloubkou trie se zkracuje čas běhu XBW (modulu BWT) a prodlužuje běh metody TD2. Bez optimalizace trie jsme dosáhli nejlepších výsledků s trii velikosti 2, což zhruba odpovídá parsování na slabiky. Od hloubky trie 9 při průměrné velikosti vstupu 22 MB zakódovaný slovník zabíral

více jak 60% výstupního souboru. Testy s hloubkou trie větší jak 9 bohužel nebylo možné dokončit kvůli vysokým nárokům na paměť, počet uzlů u testovaných souborů dosahoval při hloubce trie 9 přes 20 milionů uzlů, přitom jen malá část z nich byla skutečně použita ve výstupu z parseru. Výsledky, ze kterých jsme vycházeli, jsou v tabulce 4.1 označeny jako PT0A.

4.4.2 PT0B

Pokusíme se odstranit parserem nepoužité a tedy zbytečné uzly z trie. První navržená optimalizace je velice snadná, do struktury trie přidáme proměnnou, jež bude představovat četnost uzlu ve výstupu z parseru. Při tvorbě trie bude tato proměnná nastavena na 0 a budeme ji zvětšovat o jedna vždy, když uzel odešleme jako součást výstupu. Po skončení parsování z trie odstraníme všechny uzly, jež mají tuto proměnnou rovnu 0 a zároveň nemají žádného potomka s touto proměnnou nenulovou, tedy odstraníme uzly, jejichž synové a uzly samy nebyly použity ve výstupu. Metodu jsme nazvali PT0B. Tímto krokem významně zmenšíme velikost zakódovaného slovníku. V případě hloubky trie 9 na testované množině klesne velikost slovníku o 13% z velikosti výstupu a o 3% se zlepší kompresní poměr bez započítaného slovníku ve výstupu. Podrobné výsledky jsou v tabulce 4.1.

4.4.3 PT0C

Struktura trie obsahuje proměnnou *Rep*, která říká zda daný uzel reprezentuje slovo slovníku nebo zda se jedná pouze o podřetězec. V trii, jak ji získáváme my, je tato proměnná nevyužita, respektive je nastavena u všech uzlů na hodnotu True. Při kódování trie metoda TD2 přidělí každému uzlu s hodnotou True v proměnné *Rep* nový symbol z výstupní abecedy i přestože uzel není použit ve výstupu. Důležité je, že má nastavenu proměnnou *Rep*. Pokud proměnnou *Rep* nastavíme na False u uzlů, které parser nepoužil, zmenší se tím výstupní abeceda, čímž by se měl nepatrně zlepšit kompresní poměr. Výsledky této úpravy jsou označeny jako PT0C.

Optimalizace PT0C vyžaduje ještě drobnou úpravu hladového parseru. Po přidání podmínky na proměnnou*Rep* si parser musí pamatovat poslední výskyt shody vstupu s trií, kde měl uzel představující shodu proměnnou *Rep* True, pokud doparsuje do konce trie a není shoda, nebo *Rep* je rovno False, musí jako výstup vrátit zapamatovaný uzel.

4.4.4 Výsledky optimalizací

Tabulka 4.1 ukazuje výsledky optimalizací PTOA, PT0B a PT0C. Testy byly provedeny s programem XBW, místo původního parseru byl použit parser PA a hladový parser. Trie byla komprimována metodou TD2. Použili jsme stejnou testovací množinu souborů jako v případě metod TD2 a TD3, více o této množině je na straně 27. Tato množina byly použita i pro všechny následující testy. Každý řádek představuje průměrné hodnoty výsledků

testů nad všemi soubory z testované množiny. První sloupec značí použitou metodu k ořezání trie, druhý hloubku trie. Třetí sloupec kompresní poměr výstupu bez započítaného slovníku, sloupec vedle je kompresní poměr včetně slovníku. Sloupce t. BWT a t. TD představují čas potřebný k provedení BWT a čas potřebný ke kompresi slovníku. Předposlední údaj značí délku výstupu z parseru a poslední počet různých slov na výstupu z parseru. Na výsledcích je vidět, že s rostoucí trii se zkracuje délka výstupu, zvětšuje slovník. Každá metoda prořezání trie vylepšila kompresní poměr jak samotného výstupu, tak i se započítaným slovníkem. Při použití optimalizace PT0C je zajímavé pozorování, že od hloubky trie 5 klesá kompresní poměr výstupu bez započítaného slovníku.

Metoda	Hl.t.	K.p. bez sl	K.p	P. slovník	t. BWT	t. TD	l výstupu	růz. slov
PT0A	2	10,42%	10,44%	0,15%	12,91	0,57	11205466	8810
PT0A	3	11,84%	12,13%	2,44%	7,20	0,30	7470311	104907
PT0A	4	13,06%	14,86%	11,98%	4,67	0,48	5602735	452063
PT0A	5	13,99%	19,43%	27,56%	3,34	1,22	4482192	1176212
PT0A	6	14,76%	25,97%	42,45%	2,57	2,32	3735167	2309090
PT0A	7	15,47%	34,23%	53,96%	2,06	3,80	3201579	3840299
PT0A	8	16,29%	44,07%	62,21%	1,66	5,42	2801387	5757952
PT0A	9	17,24%	55,36%	68,07%	1,49	7,36	2490137	8052010
PT0B	2	10,42%	10,43%	0,14%	13,18	0,50	11205466	8281
PT0B	3	11,79%	12,04%	2,09%	7,32	0,28	7470311	85029
PT0B	4	12,88%	14,14%	8,91%	4,67	0,43	5602735	304967
PT0B	5	13,54%	16,68%	18,71%	3,29	1,09	4482192	662030
PT0B	6	13,88%	19,28%	27,72%	2,58	1,65	3735167	1097867
PT0B	7	13,99%	21,72%	35,13%	2,03	2,44	3201579	1573036
PT0B	8	14,02%	24,00%	41,08%	1,62	3,33	2801387	2061968
PT0B	9	13,99%	26,17%	45,96%	1,47	4,25	2490137	2558273
PT0C	2	10,42%	10,43%	0,14%	13,09	0,43	11205466	8036
PT0C	3	11,76%	12,01%	2,09%	7,23	0,26	7470311	77057
PT0C	4	12,71%	13,97%	9,03%	4,71	0,43	5602735	226197
PT0C	5	13,08%	16,23%	19,25%	3,35	1,10	4482192	378496
PT0C	6	13,07%	18,48%	28,96%	2,54	1,69	3735167	486793
PT0C	7	12,82%	20,55%	37,16%	1,94	2,46	3201579	556396
PT0C	8	12,50%	22,49%	43,85%	1,55	3,29	2801387	602402
PT0C	9	12,15%	24,33%	49,44%	1,38	4,21	2490137	632006

Tabulka 4.1: Výsledky optimalizací PT0.

4.5 PT1

4.5.1 PT1A a PT1B

Po ořezání trie pomocí předchozích metod již z trie ve fázi po průchodu hladovým parserem nemůžeme nic odebrat. Takto prořezaná trie by neměla obsahovat žádný zbytečný řetězec, jež by nebyl nutný při dekompresi. Nyní se zaměříme na optimalizaci trie, ještě než je zpracována parserem. Našim cílem bude upravit trii tak, aby výstupní abeceda z parseru, tj. počet různých slov, na které parser rozdělí vstup, byl co nejmenší a zůstala jen ta slova, která se vyskytují ve vstupu nejčastěji. Použijeme opět proměnné *Rep*. Hladový parser nastavíme tak, aby parsoval jen na řetězce, jež mají nastavenu tuto proměnnou na hodnotu True. Uzly s touto proměnnou False bude vyřazeny.

Jako první kriterium pro vyřazení uzlu z trie zkusíme použít četnost uzlu. Prvním pokusem bylo použít pevnou hranici, např. odstranit z trie všechny uzly s menší četností jak 100. Tento postup je ale příliš závislý na délce vstupu, a proto nevhodný. Jako poměrně úspěšné se ukázalo kriterium, zda je četnost větší než průměrná četnost pro každou úroveň trie. Nejprve rekurzivně projdeme trii a pro každou úroveň trie spočítáme průměrnou četnost. Z trie poté vyřadíme všechny uzly s četností menší než průměrná. Pokud v trii zachováme všechny uzly na úrovni 1, můžeme si být jisti, že hladový parser vždy doparsuje vstup. Tuto optimalizaci jsme nazvali PT1A.

Pojďme se nyní podívat jak takto prořezaná trie vypadá uvnitř, jaké řetězce a s jakou četností zůstaly. Pro tento účel jsme napsali proceduru, která u zvoleného uzlu vypíše všechny jeho syny a jejich četnosti a proceduru, která přijímá jako parametry minimální četnost a hloubku a vypíše všechny uzly, jež splňují tyto podmínky. Zde je kus výstupu, který nás zaujal:

- fu(109)
- fun(109)
- func(109)
- funct(109)
- functi(109)
- functio(109)

Hodnota v závorce za řetězcem označuje jeho četnost ve vstupním souboru. Na tomto příkladu vidíme, že prefixy slova function se ve vstupu vyskytují jen a pouze jako součást slova *function*, jinak by měly různé četnosti. Z principu fungování hladového parseru dále víme, že tyto řetězce nebudou nikdy použity při parsovaní, aby byly, parser by musel začít parsovat vstup před řetězcem function a v tomto případě vždy doparsuje až k poslednímu znaku pokud samozřejmě jinou optimalizací nevyřadíme i slovo function. Proto můžeme všechny prefixy function z trie vyřadit označením proměnné *Rep* na False. Neočekáváme

velké zlepšení, tímto krokem zmenšíme alespoň velikost výstupní abecedy. Slovo function není výjimečný příklad, takto jsou reprezentovaná všechna slova, jejichž prefix není součástí jiných frekventovaných slov jako také třeba řetězce: *http://*, *selection* a další. Tuto optimalizaci jsme nazvali PT1B.

Nyní se pojďme podívat na jednotlivé uzly a hlavně četnosti jejich synů. Pokud si necháme vypsat kořen trie libovolného souboru testované množiny, zjistíme:

- součet četností synů kořene je přesně velikost vstupního souboru
- znak mezery se v vstupních souborech vyskytuje o řád častěji než znaky abecedy
- v souboru webster z korpusu Silesia se stejně často jako mezera vyskytují ještě znaky <,>.

U ostatních uzlů klesá počet různých synů se vzdáleností od kořene, jak se dalo očekávat, u uzlů vzdálenějších od kořene trie pozorujeme nepoměr četností jejich synů. Jedna se o uzly představující slova s různými koncovkami, např. slovo funkce a funkci, kde první tvar slova je mnohem častější. Oba tvary zůstaly v trii zachovány i po prořezání, protože se vyskytují častěji než ostatní řetězce.

4.5.2 Výsledky

Význam sloupců tabulky 4.2 je stejný jako u tabulky 4.1. Řádky s výsledky optimalizace PT1A ukazují, že odstraněním uzlů s podprůměrnou četností jsme o polovinu vylepšili kompresní poměr. Řádky označené PT1B ukazují výsledky komprese při kombinaci odstranění podprůměrných hodnot a odstranění prefixů se stejnou četností, PTC1 je předchozí kombinace v opačném pořadí. Je vidět, že optimalizace PT1B odstraňuje z trie stejné uzly jako optimalizace PT0C a prakticky tedy žádné zlepšení nepřináší, to se dostavilo až při kombinaci s PT1A, kdy jsme PT1B zapojili před PT1A.

4.6 PT2

Tato metoda z trie odstraní všechny uzly trie, jež mají podprůměrnou četnost oproti svým bratrům. Předpokládáme, že pro řetězce, jež jsou reprezentovány těmito uzly, by mohlo být výhodnější z hlediska komprese naparsování na prefix těchto slov, představovaný cestou v trii z kořene do otce takového uzlu s mnohem vyšší četností a na sufix naparsovaný zvlášť jako syn kořene trie případně jako prefix dalšího řetězce. Tato metoda prochází trii rekurzivně do hloubky, v každém uzlu spočte nejprve sumu četností synů, poté v cyklu prochází syny, pokud narazí na syna s podprůměrnou četností, odstraní ho a zároveň uvolní všechny jeho potomky, na ostatní syny se rekurzivně zavolá.

Metoda	Hl, t,	K,p, bez sl	K,p	P, slovník	t, XBW	t, TD	l výstupu	růz, slov
PT1A	2	8,63%	8,64%	0,08%	14,22	0,23	11860175	1248
PT1A	3	8,86%	8,90%	0,53%	8,53	0,34	8015537	9958
PT1A	4	9,17%	9,39%	2,43%	5,94	0,19	6105933	38791
PT1A	5	9,41%	9,99%	5,92%	4,6	0,54	4977600	83528
PT1A	6	9,55%	10,62%	10,20%	3,78	1,03	4241924	129840
PT1A	7	9,62%	11,27%	14,78%	3,12	1,55	3723665	174225
PT1A	8	9,63%	11,92%	19,22%	2,71	2,21	3336751	212998
PT1A	9	9,57%	12,45%	23,05%	2,38	2,88	3042705	242224
PT1B	2	8,63%	8,64%	0,08%	14,33	0,23	11860175	1248
PT1B	3	8,86%	8,90%	0,53%	8,57	0,33	8015537	9958
PT1B	4	9,17%	9,39%	2,43%	5,97	0,19	6105933	38791
PT1B	5	9,41%	9,99%	5,92%	4,54	0,53	4977600	83527
PT1B	6	9,55%	10,62%	10,20%	3,75	1,06	4241924	129840
PT1B	7	9,62%	11,27%	14,78%	3,11	1,56	3723665	174224
PT1B	8	9,63%	11,92%	19,22%	2,75	2,22	3336751	212997
PT1B	9	9,57%	12,45%	23,05%	2,43	2,9	3042706	242223
PT1C	2	8,63%	8,63%	0,07%	14,46	0,19	11860175	1248
PT1C	3	8,85%	8,90%	0,53%	8,63	0,32	8044692	9830
PT1C	4	9,13%	9,34%	2,42%	6,07	0,18	6175740	35867
PT1C	5	9,33%	9,89%	5,84%	4,8	0,5	5078545	71880
PT1C	6	9,44%	10,46%	9,81%	3,91	1,08	4372761	105308
PT1C	7	9,51%	11,04%	13,94%	3,36	1,48	3884021	136011
PT1C	8	9,54%	11,61%	17,85%	2,88	2,14	3526054	161480
PT1C	9	9,51%	12,05%	21,04%	2,64	2,74	3264451	178401

Tabulka 4.2: Výsledky optimalizací PT1.

4.6.1 Výsledky

Výsledky v tabulce 4.3 k této optimalizaci ukazují, že srovnávání četností v rámcí trie není příliš zajímavým kritériem.

4.7 PT3

Další optimalizace, která vyšla z pozorování obsahu trie, zkusí v trii prosadit parsování na slova. Využijeme poznání, že oddělovací znak, ať už je to mezera, tabulátor nebo středník, má v testované množině největší četnost, proto ho můžeme snadno identifikovat. Pomocí toho znaku zkusíme v trii rozdělit všechny řetězce na dvě části a zahodit část za tímto znakem.

Metoda	Hl. t.	K.p. bez sl	K.p	P. slovník	t. XBW	t. TD	l výstupu	růz. slov
PT2	2	10,21%	10,22%	0,15%	13,44	0,61	11215259	7946
PT2	3	11,43%	11,67%	2,13%	7,5	0,4	7479645	75248
PT2	4	12,32%	13,54%	9,10%	4,9	0,43	5610328	219607
PT2	5	12,65%	15,70%	19,35%	3,43	1,01	4488903	366352
PT2	6	12,64%	17,86%	29,03%	2,64	1,62	3740865	471016
PT2	7	12,43%	19,91%	37,20%	2,17	2,44	3206613	539752
PT2	8	12,14%	21,81%	43,88%	1,66	3,28	2805880	585190
PT2	9	11,81%	23,62%	49,45%	1,5	4,18	2494324	614793

Tabulka 4.3: Výsledky optimalizací PT2.

4.7.1 Výsledky

Tato optimalizace měla jako první lepší kompresní poměr bez započítaného slovníku ve srovnání s původním parserem programu XBW a to v průměru na testované množině o jedno procento. Tento výsledek by měl odpovídat rozdílu mezi parsováním na slova (PT3) a slabiky, jež využívá původní parser XBX. Celkový kompresní poměr je vykompenzován v neprospěch PT3 velikostí slovníku, výsledky této optimalizace jsou shrnuty v tabulce 4.4.

Metoda	Hl. t.	K.p. bez sl	K.p	P. slovník	t. XBW	t. TD	l výstupu	růz. slov
PT3	2	8,59%	8,61%	0,17%	16,53	0,58	13096482	7512
PT3	3	8,57%	8,77%	2,37%	11,65	0,7	10004041	57698
PT3	4	8,41%	9,17%	8,65%	9,53	0,29	8496188	131326
PT3	5	8,14%	9,68%	16,4%	8,56	0,56	7589066	181028
PT3	6	7,81%	10,05%	22,9%	7,62	0,9	6938866	205753
PT3	7	7,55%	10,42%	28,14%	7,52	1,21	6522161	220437
PT3	8	7,29%	10,7%	32,52%	6,71	1,56	6205294	229797
PT3	9	7,09%	11,01%	36,08%	6,39	1,9	5977098	236819

Tabulka 4.4: Výsledky optimalizací PT3.

4.8 PT4

Zkusili jsme upravit PT1 tak, aby ořezávala podle průměrné četnosti uzlů celé trie, což upřednostní uzly blíže ke kořeni. To nám pomohlo udělat si představu, jestli je výhodnější spíše široká a hluboká trie, nebo trie menších rozměrů s velkou hustotou uzlů těsně pod kořenem. Lépe se osvědčila druhá varianta.

4.8.1 Výsledky

Na výsledcích této metody v tabulce 4.5 a výsledcích PT3 (Tabulka 4.4) je poměrně zajímavé pozorování, i přestože výstup z parseru po optimalizaci PT4 je kratší a slovník obsahuje méně slov, tak kompresní poměr bez započítaného slovníku je podstatně horší oproti PT3.

Metoda	Hl. t.	K.p. bez sl	K.p	P. slovník	t. XBW	t. TD	l výstupu	růz. slov
PT4	2	8,56%	8,56%	0,07%	14,98	0,14	12339858	904
PT4	3	8,69%	8,71%	0,33%	8,61	0,29	8163346	6294
PT4	4	8,89%	9,01%	1,32%	6,02	0,17	6193676	23953
PT4	5	9,08%	9,39%	3,38%	4,54	0,48	5035871	55012
PT4	6	9,21%	9,81%	6,29%	3,75	0,86	4281504	92240
PT4	7	9,29%	10,27%	9,70%	3,09	1,4	3752006	131467
PT4	8	9,33%	10,73%	13,24%	2,73	2,09	3360198	169583
PT4	9	9,31%	11,19%	16,83%	2,33	2,63	3054343	205193

Tabulka 4.5: Výsledky optimalizací PT4.

4.9 PT5

Jako poslední optimalizaci s hladovým parserem jsme vyzkoušeli zapojení nejúspěšnějších metod za sebou, tedy PT3 a PT4.

4.9.1 Výsledky

Výsledky této kombinace jsou v tabulce 4.6, touto optimalizací jsme získali konečně lepší výsledky oproti slabikovému parseru použitému v programu XBW.

4.10 Parser s výhledem

Nevýhodou předchozího postupu a hladového parseru je, že nemůžeme ovlivnit způsob naparsovaní. Například pokud se nám na vstupu vyskytuje velmi často slovo abeceda a je obklopeno náhodnými krátkými řetězci, tak předchozími optimalizacemi v trii ponecháme s velkou pravděpodobností slovo abeceda a slova tvaru aabeced nebo becedaa, tj. všechna slova s velkou četností. Hladový parser bohužel neupřednostní tato dlouhá slova a pokud z trie neodstraníme řetězce, jejichž sufix je prefixem slova abeceda, tak dokonce tato slova začne parsovat na písmena. Přitom z hlediska komprese by bylo výhodné rozdělit vstup na slovo abeceda a ostatní krátké řetězce na slabiky. Zkusíme navrhnout a otestovat parser, který bude používat jistý výhled na vstupu, vyzkouší různé způsoby naparsovaní a pomocí

Metoda	Hl. t.	K.p. bez sl	K.p	P. slovník	t. XBW	t. TD	l výstupu	růz. slov
PT5	2	8,4%	8,41%	0,07%	18,3	0,19	14188593	843
PT5	**3**	**8,27%**	**8,29%**	**0,3%**	**12,57**	**0,29**	**10665904**	**5348**
PT5	**4**	**8,24%**	**8,33%**	**1,04%**	**10,19**	**0,29**	**9066340**	**17300**
PT5	5	8,22%	8,4%	2,29%	9,0	0,35	8133836	33453
PT5	6	8,17%	8,47%	3,72%	8,12	0,56	7488582	48134
PT5	7	8,11%	8,54%	5,17%	7,64	0,87	7076822	60978
PT5	8	8,08%	8,64%	6,63%	7,53	1,24	6764654	72685
PT5	9	8,04%	8,72%	7,98%	7,02	1,48	6535320	82549

Tabulka 4.6: Výsledky optimalizací PT5.

ohodnocující funkce vybere tu nejlepší variantu. Všechny doposud uvedené optimalizace mohou být použity i pro tento parser.

Námi implementovaný parser načítá vstup do zásobníku o proměnné délce zvolené při kompilaci programu. Pomocí levé a pravé zarážky určuje jaký kus zásobníku budeme zpracovávat. Pravá zarážka je neměnná a označuje konec zásobníku. Ke zpracování zásobníku slouží rekurzivní funkce, která v aktuálním rozsahu od levé zarážky hledá odpovídající řetězec v trii. Pro shodu v uzlu, který je označen jako reprezentující, posune levou zarážku právě o délku shody, uloží si poslední uzel a zavolá sebe sama na zkráceny úsek. Pokud přiblíží levou zarážku pravé, došel do konce, ohodnotí aktuální variantu naparsování a výsledek porovná s dosavadním nejlepším, pokud je lepší, uloží jej, jinak zahodí. Poté se vrací do stavu před rekurzivním voláním a zkouší delší shodu. Takto projde všechna možná naparsovaní pro každý úsek a jako výstup vrátí podle zvoleného ohodnocení naparsování nejlepší.

V pseudokódu Algoritmu 4 není uvedena optimalizace, kterou jsme v implementovaném algoritmu použili. Do cyklu, ve kterém srovnáváme trii proti vstupu, jsme vložili podmínku, která zabrání rekurzivnímu volání v případě shody délky jedna, naparsovaní po jednom znaku se použije jen v případě, kdy jiná shoda není nalezena.

4.10.1 Určení skóre

Skóre pro každou variantu naparsování počítáme jako sumu přes ohodnocení všech obsažených uzlů. Ohodnocení jednotlivých uzlů násobíme hodnotou odpovídající délce řetězce, jež uzel představuje. Pokud bychom to neudělali, podstatně bychom zvýhodnili naparsování po kratších úsecích, a tedy s větším počtem obsažených uzlů. Pro parser s výhledem je stěžejní správně navržená ohodnocující funkce. Nejprve jsme si museli zvolit jaké řetězce má upřednostňovat, zvolili jsme následující kategorie:

- řetězce, která jsme již použili jako výstup parseru,

Algoritmus 4 Parser s výhledem

```
struktura napársování={
trie **pole_použitých_uzlů;
UInt32 *pole_dílčích_délek_úseků;
UInt32 délka_zpracovaného_úseku;
UInt32 skóre;}
parser_rec(vstup,od,do,nej_napársování,akt_napársování){
if od==do then
  akt_napársování.score=spočti_scóre(akt_napársování)
  if nej_napársování.score<akt.napársování.skóre then
    nej_napársování=akt_napársování;
  end if
  return();
end if
for uzel in všechny_shody(vstup[od],trie) do
  akt_napársování.přidej(uzel);
  parser_rec(vstup.vstup.pozice+uzel.hloubka,do,nej_napársování,akt_napársování);
  akt_napársování.odeber(uzel);
end for
}
parser(vstup){
while vstup.pozice<vstup.délka do
  inicializuj nej_naparsovaní, aktualní_napársovaní;
  parser_rec(vstup, vstup.pozice,vstup.pozice+l_výhledu,nej_napársování,aktualní_napársování)
  vstup.pozice+=napársování.délka_zpracovaného_úseku;
  }
end while
```

- dlouhé řetězce,
- slova, která jsou ve vstupu často.

Slova z první kategorie jsou pro nás výhodná, protože nám nezvětšují velikost slovníku a následná komprese častých slov je také výhodnější. Výhody dlouhých a častých slov jsou zřejmé. Proměnnými, které vstupovaly do ohodnocující funkce, byly: hloubka uzlu v trii, četnost ve vstupním souboru a četnost, se kterou se uzel již objevil jako výstup z parseru. Obě skupiny vstupů i preferovaného výstupu jsou prakticky identické, proto jsme první parametry ohodnocující funkce zvolili náhodně, viz algoritmus 5.

Tabulka 4.7 ilustruje, jak jsou při této ohodnocující funkci ohodnoceny uzly tří kategorií podle délky reprezentovaného řetězce.

Dalším důležitým parametrem parseru s výhledem je délka výhledu, pro začátek jsme zvolili délku 10 znaků.

Algoritmus 5 Parametry ohodnocující funkce.

```
skore=hloubka_uzlu(uzel)*4
skore+=(!!uzel.frekvence_použití_parserem)*10
skore+=(!!uzel.četnost_na_vstupu)*5
```

Tabulka 4.7: Hodnoty skóre

Délka řetězce	1	3	5	8
již použitý	19	27	35	47
nepoužitý, četný	9	17	25	37
nepoužitý	4	12	20	32

4.10.2 Výsledky

Testy jsme provedli na stejné testovací množině jako u předchozích metod, řádky opět představují průměrné hodnoty z celé testované množiny, optimalizaci pro prořezání trie jsme zvolili PT1A, tedy takovou, aby trie nebyla příliš prořezaná a parser měl širší výběr. Tato první navržená ohodnocující funkce bohužel nedala moc dobré výsledky. Jak ukazují řádky označené PTV1 v tabulce 4.8, získané hodnoty jsou při porovnání s hladovým parserem a PT1A mnohem horší. Řádky označené PTV1B jsou výsledky testů při stejných parametrech jako PTV1, jen výhled parseru byl rozšířen na 14 znaků. Rozšířením výhledu o 4 znaky jsme získali zlepšení kompresního poměru o desetiny procenta, ale čas běhu parseru se zhruba zdvojnásobil. Pod označením PTV1c jsou řádky odpovídající výsledkům při kombinaci prořezání trie PT3 a PT4. Zkusili jsme různé obměny parametrů ohodnocující funkce, ale změna se projevila jen prodloužením výstupu a zkrácením slovníku nebo opačně. Kompresní poměr jsme příliš neovlivnili.

Metoda	Hl. t.	K.p. bez sl	K.p	P. slovník	t. XBW	t. TD	l výstupu	růz. slov
PTV1	2	8,84%	8,84%	0,04%	34,1	0,12	22410932	267
PTV1	3	12,57%	12,58%	0,05%	11,93	0,28	12068999	1248
PTV1	4	16,53%	16,56%	0,17%	7,75	0,25	10061298	4659
PTV1	5	17,02%	17,12%	0,58%	5,92	0,44	8284797	14149
PTV1	6	16,70%	17,03%	1,91%	4,42	0,8	6675073	35282
PTV1	7	17,69%	18,46%	4,07%	3,98	1,4	6251257	71664
PTV1	8	17,14%	18,46%	6,93%	3,29	2,12	5463863	109532
PTV1	9	16,68%	18,91%	11,53%	3,13	2,82	5167717	169426
PTV1B	2	8,84%	8,84%	0,04%	34,14	0,16	22410932	267
PTV1B	3	12,63%	12,63%	0,05%	11,35	0,21	12041598	1248
PTV1B	4	16,44%	16,47%	0,18%	7,31	0,29	9737677	4657
PTV1B	5	16,98%	17,08%	0,58%	5,55	0,44	7987574	14221
PTV1B	6	17,24%	17,52%	1,56%	4,41	0,81	6828730	30642
PTV1B	7	17,19%	18,13%	5,08%	3,18	1,43	5474478	87300
PTV1B	8	16,78%	18,55%	9,29%	2,34	2,15	4390897	152434
PTV1B	9	17,36%	19,41%	10,32%	2,24	2,77	4241886	178628
PTV1C	2	8,84%	8,84%	0,04%	34,26	0,12	22410932	267
PTV1C	3	13,15%	13,15%	0,06%	13,72	0,28	13536384	2103
PTV1C	4	16,27%	16,30%	0,19%	9,71	0,27	11453706	7330
PTV1C	5	16,22%	16,29%	0,39%	8,82	0,32	10435766	12729
PTV1C	6	16,34%	16,48%	0,85%	8,17	0,52	9587945	21609
PTV1C	7	16,55%	16,91%	2,10%	7,23	0,85	8662883	40798
PTV1C	8	16,64%	17,23%	3,37%	6,85	1,22	8134914	61498
PTV1C	9	16,81%	17,55%	4,08%	6,76	1,6	7970045	75962

Tabulka 4.8: Výsledky při použití parseru s výhledem.

Kapitola 5

Výsledky

5.1 Testy na binárních datech

Nejlépe vycházející metody jsme otestovali i na binárních datech, na nichž by se podle našeho očekávání měla nejvíce projevit výhoda parsování na základě prořezání trie oproti parsování na slova, slabiky. Z počátku parser nebyl schopen zpracovat binární data, funkce `iconv` nedokázala rozpoznat znakovou sadu vstupu, proto jsme ji pro tyto testy zakomentovali. K testování jsme použili vstupní soubory popsané v sekci 5.3. V tabulkách 5.1 a 5.2 jsou nejlepší získané výsledky.

Soubor	K.p. bez sl	K.p	P. slovník	t. BWT	t. TD	l.výstupu	růz. slov
dickens	26,93%	26,93%	0,01%	5,43	0,0	6755601	299
mozilla	37,42%	37,43%	0,02%	34,75	0,8	38270959	3665
mr	25,97%	25,99%	0,05%	6,9	0,4	9429636	363
nci	4,51%	4,51%	0,01%	42,38	0,0	30435233	88
ooffice	48,94%	48,99%	0,11%	2,27	0,12	4062806	3819
osdb	28,01%	28,13%	0,01%	3,54	0,4	5876620	4002
reymont	19,24%	19,26%	0,13%	2,73	0,4	3926793	835
samba	22,23%	22,24%	0,04%	10,56	0,12	13191302	2292
sao	71,68%	71,71%	0,05%	3,18	0,8	5345033	2310
x-ray	52,87%	52,88%	0,03%	4,42	0,0	7603703	621
Průměr	33,79%	33,81%	0,06%	11,62	0,3	12489769	1829

Tabulka 5.1: Výsledky komprese binárních souborů, hloubka trie 2.

5.2 Testovací vstupní data

Experimenty jsme prováděli nad testovacími korpusy z projektu XBW a korpusem Silesia[1]. Korpusy z XBW obsahují XML soubory sestavené z českých, slovinských a anglických

Soubor	K.p. bez sl	K.p	P. slovník	t. BWT	t. TD	l.výstupu	růz. slov
dickens	26,16%	26,18%	0,05%	4,48	0,0	5399030	1488
mozilla	38,10%	38,27%	0,46%	23,37	2,28	29196206	142832
mr	26,31%	26,33%	0,09%	7,32	0,24	9030358	2549
nci	4,35%	4,35%	0,03%	38,19	0,0	27788448	364
ooffice	51,13%	51,70%	1,10%	1,42	0,34	3065496	46512
osdb	28,35%	28,46%	0,37%	2,42	0,3	4492263	12794
reymont	19,00%	19,05%	0,25%	1,94	0,12	3069324	2368
samba	22,59%	22,69%	0,43%	7,68	0,56	10073844	30708
sao	73,37%	73,81%	0,59%	1,18	1,34	3211139	103003
x-ray	51,53%	51,63%	0,18%	2,37	0,2	4836722	11587
Průměr	34,09%	34,25%	0,36%	9,04	0,54	10016283	35421

Tabulka 5.2: Výsledky komprese binárních souborů, hloubka trie 3.

HTML stránek. Z korpusu Silesia jsme vybrali pouze dva soubory, soubor webster, který obsahuje text s HTML značkami, a soubor xml.

5.3 Testovací vstupní data-binární

Jako vstupní data pro testy nad binárními soubory jsme použili soubory z korpusu Silesia, takové, které nebyly v kódování unicodu. V tabulce 5.4 jsou popsány blíže.

5.4 Použitý hardware

Pro běh všech experimentů jsme použili počítačovou sestavu:

- Intel(R) Core(TM)2 CPU 6420 @ 2.13Ghz
- 2GB RAM
- SATA II pevný disk Seagate

Operační systém Linux, distribuce Gentoo 32bit, verze jádra 2.6.25.r9.

Zkratka	Korpus a jméno	Velikost
CZ01	xbw_test_xml_cz/998001.txt	24 027Kb
CZ02	xbw_test_xml_cz/998002.txt	22 685Kb
CZ03	xbw_test_xml_cz/998003.txt	19 676Kb
CZ04	xbw_test_xml_cz/998004.txt	17 378Kb
CZ05	xbw_test_xml_cz/998005.txt	18 539Kb
CZ06	xbw_test_xml_cz/998006.txt	18 996Kb
CZ07	xbw_test_xml_cz/998007.txt	18 863Kb
CZ08	xbw_test_xml_cz/998008.txt	19 203Kb
CZ09	xbw_test_xml_cz/998009.txt	19 668Kb
CZ10	xbw_test_xml_cz/998010.txt	18 652Kb
EN01	xbw_test_xml_en/999010.txt	14 153Kb
SL01	xbw_test_xml_sl/997001.txt	20 556Kb
SL02	xbw_test_xml_sl/997002.txt	27 820Kb
SL03	xbw_test_xml_sl/997003.txt	24 039Kb
SL04	xbw_test_xml_sl/997004.txt	23 216Kb
SL05	xbw_test_xml_sl/997005.txt	23 358Kb
SL06	xbw_test_xml_sl/997006.txt	18 061Kb
SL07	xbw_test_xml_sl/997007.txt	24 112Kb
SL08	xbw_test_xml_sl/997008.txt	17 703Kb
SL09	xbw_test_xml_sl/997009.txt	19 583Kb
SL10	xbw_test_xml_sl/997010.txt	21 727Kb
SI01	Silesia/nci	32 767Kb
SI02	Silesia/webster	40 487Kb

Tabulka 5.3: Testovací množina souborů.

Jméno	obsah	velikost
dickens	kolekce prací CH. Dickense	9954Kb
mozilla	zatarováné spustitelné soubory Mozilly 1.0	50020Kb
mr	obrázek z mag. rezonance	9737Kb
nci	databáze	32768Kb
ooffice	knihovna z OpenOffice1.01	6008Kb
osdb	MySql databaze	9850Kb
reymont	pdf soubor	6472Kb
samba	zdrojové kódy Samby 2-2.3	21100Kb
sao	SAO katalog	7082Kb
x-ray	obrázek z rentgenu	8276Kb

Tabulka 5.4: Testovací množina souborů s binárním kódováním.

Kapitola 6

Závěr

Naším cílem bylo navrhnout a otestovat metody vedoucí k vytvoření slovníku tak, aby vstup naparsován na elementy toho slovníku byl optimalizován pro následné kompresní algoritmy a abychom zároveň dosáhli co nejlepšího kompresního poměru při zahrnutí slovníku do výstupu. Představili jsme metody PT01A, B a C, které ze slovníku odstraní slova, jež nebyla použita, metody PT1A, B a C, které odstraňují slova s podprůměrnou četností v rámci úrovní v trii a na základě porovnávání četností mezi úrovněmi. Metodu PT2, která odstraňuje řetězce s podprůměrnou četností v rámci uzlů. Metodu PT3, jež ve vstupu najde znak s největší četností a v trii rozdělí podle tohoto znaku slova. Metodu PT4, která odstraňuje slova s menší četností než průměr nad celou trií a jako poslední metodu PT5, jež je kombinací PT3 a PT4.

Nejlepších výsledků jsme dosáhli při kombinaci optimalizace PT5 a hladového parseru při hloubce trie 3. V tabulkách 6.1 a 6.2 uvádíme porovnání s existujícími kompresními programy. Výsledky programu Rar a XBW jsou získané při výchozích parametrech. Programům Gzip a Bzip2 jsme předali parametr -9 značící nejlepší kompresi. Hodnota za podtržítkem v prvních řádcích tabulek značí hloubku trie. Sloupec označený XBW_znak obsahuje výsledky programu XBW bez použití parseru. Optimalizací PT5 se nám podařilo zkrátit čas běhu programu XBW oproti verzi s původním parserem. Tohoto zrychlení jsme docílili podstatným zmenšením slovníku. S původním parserem XBW měl výstup 25 tisíc slov v průměru na testované množině. Náš parser pouze 5 tisíc. Přitom rozdíl délky výstupu z parseru byl 4 miliony, což se neprojevilo zásadně na době běhu BWT (viz tabulka 7.1 v Přílohách). Metoda PT5 v kombinaci s hladovým parserem odpovídá zhruba parsování po slabikách, které je výhodné pro textové dokumenty. Proto náš parser dosahuje podobných výsledků nad textovými daty jako program XBW s původním parserem. Výhoda našeho parseru oproti parsování na slabiky se projevila při binárním vstupu, kde jsme dosáhli v průměru o 4% lepších výsledků než program XBW. Bohužel se nám nepodařilo přiblížit se k výsledkům programu bzip2[2], který používá podobné kompresní techniky jako program XBW, ovšem bez využití parseru a slovníku. Program XBW při vypnutém parsování na slabiky dosahoval průměrného kompresního poměru 34.21%.

Další možnosti vývoje spočívají v rozvoji metod prořezání trie např. na základě analýzy

užitečnosti jednotlivých uzlů ku nárokům na jejich uložení v rámci slovníku.

Soubor	bzip2	gzip	rar	PT5_2	PT5_3	XBW	XBW_znak
dickens	27,47%	37,79%	30,64%	26,93%	26,18%	26,49%	27,51%
mozilla	34,98%	37,08%	30,38%	37,43%	38,27%	39,92%	37,29%
mr	24,48%	36,85%	32,39%	25,99%	26,33%	43,87%	26,09%
nci	5,40%	8,90%	6,46%	4,51%	4,35%	4,41%	4,48%
ooffice	46,53%	50,23%	37,62%	48,99%	51,70%	53,75%	48,62%
osdb	27,79%	36,85%	32,52%	28,13%	28,46%	30,39%	28,16%
reymont	18,80%	27,48%	23,60%	19,26%	19,05%	19,44%	19,04%
samba	21,06%	25,03%	19,53%	22,24%	22,69%	22,90%	23,08%
sao	68,13%	73,46%	76,81%	71,71%	73,81%	82,76%	74,49%
x-ray	47,81%	71,25%	48,80%	52,88%	51,63%	60,89%	53,3%
Průměr k,p,	32,24%	40,49%	33,88%	33,81%	34,25%	38,48%	34,21%
Čas celkem	41s	32s	45s	2m30s	2m35s	3m19s	3m18s

Tabulka 6.1: Srovnání TD5 s ostatními programy nad binárním vstupem.

Soubor	bzip2	gzip	rar	PT5_2	XBW
CZ01	17,57%	21,20%	16,11%	10,98%	10,76%
CZ02	16,71%	20,08%	14,89%	9,72%	9,53%
CZ03	16,99%	20,41%	15,76%	10,50%	10,31%
CZ04	16,56%	19,98%	15,06%	9,44%	9,26%
CZ05	16,07%	19,15%	14,68%	9,16%	10,44%
CZ06	15,37%	18,92%	13,42%	7,95%	7,78%
CZ07	15,74%	18,95%	13,35%	8,48%	8,32%
CZ08	15,83%	19,26%	13,76%	7,99%	9,06%
CZ09	15,71%	18,80%	13,83%	8,03%	7,83%
CZ10	15,47%	18,96%	13,37%	8,25%	8,07%
EN01	16,64%	21,19%	11,50%	10,79%	10,50%
SL01	14,08%	17,11%	12,53%	8,56%	8,38%
SL02	12,05%	18,80%	8,84%	5,79%	5,66%
SL03	13,27%	17,80%	4,41%	4,10%	4,23%
SL04	13,49%	16,79%	11,33%	7,13%	7,97%
SL05	14,40%	18,14%	4,84%	4,41%	4,31%
SL06	12,61%	16,25%	4,72%	4,49%	4,42%
SL07	13,50%	16,93%	11,49%	7,42%	7,26%
SL08	16,99%	20,20%	9,81%	9,08%	8,94%
SL09	14,09%	18,05%	7,72%	6,60%	6,44%
SL10	14,31%	17,46%	12,82%	8,56%	8,38%
SI01	5,40%	8,90%	6,46%	4,35%	4,41%
SI02	20,85%	29,09%	23,40%	18,92%	19,17%
Průměr	14,94%	18,80%	11,92%	8,29%	8,32%
čas	2m9s	37s	1m46s	6m20s	8m39s

Tabulka 6.2: Srovnání nad soubory v unicodu.

Kapitola 7

Přílohy

7.1 Seznam zkratek

PA Parser použitý při prvním průchodu vstupu sloužící k vytvoření trie, viz strana 14.

PT0[A|B|C] Metody prořezání trie předcházející kompresi trie, viz strana 15.

PT1[A|B|C] Metody prořezání trie před parsovanim vstupu, viz strana 18.

PT2 Metoda prořezání trie srovnáním oproti průměrné četnosti pro každou délku, viz strana 19.

PT3 Metoda prořezání trie pomocí na řetězce ohraničené nejčetnějším znakem na vstupu, viz strana 20.

PT4 Metoda prořezání trie srovnáním oproti celkové průměrné četnosti, viz strana 21.

PT5 Metoda prořezání trie kombinující PT3 a PT4 22.

TD[1|2|3] Zastupuje anglický výraz Trie-based Dictionary compression, viz strana 9.

XBW Modulární program optimalizovaný pro kompresi nevalidních XML, viz strana 6.

7.2 Výsledky XBW

Soubor	K.p.bez slov	délka	růz slov
CZ01	10,56%	6921280	34383
CZ02	9,34%	6394066	31349
CZ03	10,09%	5527476	32438
CZ04	9,05%	4775627	26944
CZ05	10,10%5	6268216	31901
CZ06	7,58%	5339039	26437
CZ07	8,08%	5164828	27947
CZ08	8,81%	5824709	28499
CZ09	7,55%	5301372	29880
CZ10	7,89%	5188355	26111
EN01	10,30%	3582952	19881
SK01	8,19%	5134690	26040
SK02	5,50%	7178640	25175
SK03	4,09%	9032914	19619
SK04	7,77%	8028592	27929
SK05	4,22%	6050120	15654
SK06	4,31%	4333590	15331
SK07	7,07%	6060207	27132
SK08	8,73%	4673205	24362
SK09	6,29%	4958832	19531
SK10	8,18%	5434780	27484
SI01	4,40%	11625285	11424
SI02	19,09%	17070364	32681
Průměr	**8,14%**	**6516049,52**	**25570,96**

Tabulka 7.1: XBW s původním perserem: kompresní poměr, délka výstupu z parseru, počet různých slov.

Literatura

[1] Silesia compression corpus.

[2] Wikipedia - bzip2.

[3] Wikipedia - trie.

[4] Michal Žemlička Jan Lánský. Compression of a dictionary. *Proceedings of the Dateso 2006*, 176:11–20, 2006.

[5] Jan Lánský Radovan Šesták. Compression of concatenated web pages using xbw. *SOFSEM 2008*, 4910:743–753, 2008.