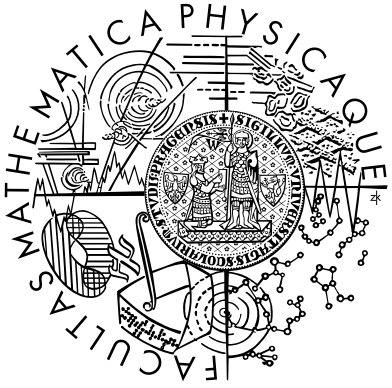
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Blažek

Komprese krátkých textových zpráv

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Dvořák, CSc.

Studijní program: Informatika,
Obecná informatika

2007

Děkuji svému vedoucímu RNDr. Tomáši Dvořákovi, CSc. za odborné konzultace a účastníkům referativního semináře z komprese dat za konstruktivní kritiku.

Prohlašuji, že jsem bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Se zapůjčováním práce souhlasím.

V Praze dne 6. srpna 2007 Jan Blažek

Obsah

Obsah

Název práce: Komprese krátkých textových zpráv

Autor: Jan Blažek

Katedra (ústav): Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Dvořák, CSc.

e-mail vedoucího: dvorak@ksvi.mff.cuni.cz

Abstrakt: Práce popisuje implementaci algoritmu pro kompresi krátkých textových zpráv založenou na PPM. Pro reprezentaci kontextového modelu je zde použita datová struktura Optimal Tree Machine, vybudovaná na základě statistického rozboru trénovacích dat. Algoritmus je optimalizován pro zařízení s omezenou výpočetní kapacitou a omezenou operační pamětí. Práce shrnuje experimentální výsledky algoritmu.

Klíčová slova: SMS, OTM, PPM, kontext

Title: Compression of short text messages

Author: Jan Blažek

Department: Kabinet of software and informatics teaching

Supervisor: RNDr. Tomáš Dvořák, CSc.

Supervisor's e-mail address: dvorak@ksvi.mff.cuni.cz

Abstract: This paper describes implementation of short text message compression algorithm based on PPM. For representation of context model algorithm is used the Optimal Tree Machine data structure built on statistics of training data. Algorithm is optimized for low-level hardware device. This paper summarizes experimental results of this algorithm.

Keywords: SMS, OTM, PPM, context

1. Úvod

Impulsem pro řešení problému komprese krátkých textových zpráv bylo jejich zavedení v masovém měřítku v mobilních komunikacích. Nejde jen o běžně používané SMS, ale i další služby, jako například MMS, WAP nebo mobilní internet. Komprese SMS zde ovšem stojí v popředí díky největší rozšířenosti.

Při použití stávajících protokolů v mobilních zařízeních má software umožňující kompresi posílaných paketů dvojí uplatnění. Lze jej implementovat jako uživatelský software[1)] a nebo jako software operátora, jenž šetří radiové pásmo vymezené pro konkrétní službu. Ve své práci jsem se soustředil spíše na uživatelskou aplikaci.

Motivací pro řešení problému komprese krátkých textových zpráv je neefektivní aplikace stávajících algoritmů. Z těchto dnes používaných algoritmů bezztrátové komprese a jejich různých variací práce vybírá algoritmus a upravuje některé jeho vlastnosti tak, aby byl co nejefektivnější a aby umožnil uživatelům mobilních telefonů využít poskytovanou službu co nejlépe. Služby mobilních operátorů se samozřejmě velmi různí, ovšem princip krátkých zpráv je zachován u téměř všech těchto mobilních služeb.

Práce se zabývá implementací návrhu kompresního algoritmu publikovaného v [2] a jeho experimentálním testováním. Cílem je též upozornit na oblast oboru komprese, kterou je třeba se dále zabývat a ve které je možné dosáhnout efektivních výsledků. Z různých pramenů a dnes používaných algoritmů vybírá a vyzdvihuje některé, jež je možné pro kompresi krátkých zpráv použít.

Součástí práce je i přehled problémů spojených s kompresí krátkých zpráv, popis některých stávajících kompresních algoritmů, postup implementace navrženého algoritmu a jeho experimentální výsledky.

[1)] např. Java applet

2. Současný stav problematiky

Kompresi krátkých zpráv se zatím příliš nevěnovala pozornost. Přirozeným způsobem úspory krátkých zpráv bylo sdružování do větších souborů a jejich následná komprese, čímž se odstranily nevýhody, jež se u komprese krátkých zpráv vyskytují. Tento způsob však dnes již není postačující, neboť novým cílem není úspora místa na disku nebo pásce, ale úspora přenosové kapacity.

Prací, které by se touto problematikou přímo zabývaly, v současné době není nijak mnoho. Jedná se hlavně o [7], kde autoři svůj návrh dovedli až ke komerční implementaci a [2], ze které jsem čerpal při své implementační práci. Vývoj nového algoritmu má smysl ze dvou důvodů: Malá efektivita stávajících nejlepších metod komprese a úzké spektrum navržených algoritmů. Navíc některá data v mobilních sítích vůbec komprimována nejsou. Algoritmy komprese neformálně rozdělme do dvou kategorií:

Definice 1: (adaptivní kompresní algoritmus)
je kódovací algoritmus upravující kódy pro znaky přijímané abecedy v *průběhu komprese* (např. podle jejich četnosti výskytu).

Definice 2: (statický kompresní algoritmus)
je kódovací algoritmus s konstantními kódy pro kódovaní znaků přijímané abecedy.

Převážná část dnes používaných kompresních algoritmů jsou algoritmy adaptivní, jejichž užití je u krátkých zpráv nemožné nebo neefektivní. Jako použitelné se tak jeví statické kompresní algoritmy, avšak ty nedosahují tak dobré komprese. Nabízí se tedy otázka, zda je možné vyvinout algoritmus, který bude efektivnější pro kompresi krátkých zpráv než stávající statické i adaptivní metody.

Nejprve podrobněji shrnu požadavky na software pro použití na mobilních telefonech.

2.1. Technické parametry

Technické parametry zařízení, pro která je software určen, jsou dnes mezi $200MHz - 700MHz$ pro rychlost procesorů a $64kB - 128MB$ pro velikost operační paměti. Od zařízení lze očekávat možnost připojení k internetu (HTTP a další protokoly), základní práce s persistentní pamětí.

Pro vývoj na mobilních zařízení existuje několik vývojových prostředí. Nejrozšířenější je *JavaME*, ale silnou konkurencí je *C/C++* podporované operačními systémy *Windows Mobile, OS Symbian* nebo *Linux*. Variantou programovacího jazyka je též *C#* a programování pro určitý typ telefonu bez jakékoli podpory (různé firemní vývojářské nástroje). Rozhodl jsem se pro *JavaME* kvůli největšímu rozšíření mezi uživateli. V *JavaME* existují dvě konfigurace: Conected Device Configuration a Conected Limited Device Configuration (dále jen *CDC* a *CLDC*). *CDC* má plnou podporu Java Virtual Machine (dále jen *JVM*), ale od zařízení vyžaduje velikost paměti alespoň $2MB$ a proto nepřipadá v úvahu. *CLDC* je nejmenší podporovaná konfigurace *JavaME* a vyžaduje alespoň $128kB$ persistentní paměti, $32kB$ operační paměti a alespoň 16 bitový procesor. Tyto parametry odpovídají cílovému hardware. *CLDC* ovšem nepodporuje kompletní *JVM* ale pouze omezenou verzi tzv. K Virtual Machine viz [8]. Využil jsem též možností pro *CLDC* ve formě profilů určených pro konkrétní zařízení. Jsou to *PDAP* pro PDA, *IMP* pro prodejní automaty (bez grafického rozhraní) a *MIDP* pro mobilní telefony. V Bakalářské práci je tedy použita *JavaME* v konfiguraci *CLDC* a profilem pro mobilní telefony *MIDP* ve verzi 2.0.

Tyto možnosti a parametry se již dnes stávají zastaralými, nicméně požadavky na nízkou výpočetní složitost algoritmu a velikost operační paměti zůstávají nadále aktuální. Do budoucna lze počítat nejen s HTTP protokoly, ale i s různým blue-tooth, infrared nebo wireless připojením, které podstatně zvýší přístupnost různých externích dat (adaptivních modelů, kompresních

slovníků ...).

2.2. Délka zpráv

Pro textovou kompresi je problémem délka zpráv na mobilních zařízeních. Obvykle se komprese provádí na velkých datech, jejichž kompresí se ušetří mnoho místa. Komprese krátkých textových zpráv má požadavky na kompresní algoritmy přinejmenším odlišné. Efektivita komprese krátkých zpráv bude vůči objemnějším datům menší jak v absolutním, tak v relativním měřítku. Použití adaptivních algoritmů komprese je navíc málo efektivní díky délce zpráv (různé úpravy kódu založené na statistikách získaných ze vstupních dat jsou víceméně náhodné kvůli malému vzorku statisticky zpracovaných dat).

Úspory je dnes dosahováno prostřednictvím provozovatele sítě, který šetří radiové pásmo tím, že délku zpráv omezuje. Na oplátku zákazníci neustávají ve vývoji alternativního srozumitelného jazyka, který vyhovuje parametrům operátorů lépe. Je tedy nutné počítat s vyšší entropií užitého jazyka a s malým objemem vstupních dat.

2.3. Slovník

Jestliže pro kompresi chceme využít nějakou pomocnou strukturu trénovanou na vzorových datech, je třeba brát v úvahu odlišnosti jazyka používaného v krátkých zprávách. Uživatelé nevyužívají spisovného jazyka, ale jazyka s vyšší informační hustotou. Navíc jazyk, který je používán pro SMS služby, jistě obsahuje knižní ani odborné výrazy jen výjmečně. Prostředkem uživatelů pro úsporu jsou různé metody vypuštění mezer mezi slovy, používání zkratek místo frází a pod. Proto by slovníková metoda měla odlišnost jazyka zohledňovat.

Slovník lze využívat například pro kódování v závislosti na kontextu (podobně jako PPM) nebo kódování na základě frekvenční analýzy jazyka (adaptivní Huffmanovo kódování, slovník používaný službou *T9*). Aplikace tohoto charakteru je pak nutné specializovat pro konkrétní data (např. dle světových jazyků).

Druhým problémem použití slovníku je jeho omezenost. Slovník nesmí být rozhodně větší než $64kB$, neboť větší již nemusí být akceptován na některých zařízeních. [2)] Efektivita malého slovníku bude pravděpodobně nízká. Při dosavadním vývoji elektronického zařízení, však problém malé paměti nebude zřejmě tak zásadní.

Pro použití statického slovníku (slovník se v průběhu komprese nemění) mluví i možnost jeho přípravy na výkonnějším zařízení (např. PC) a samozřejmě alespoň minimální adaptace na předkládaná data.

2.4. Symetrie

Symetrie označuje vlastnost kompresního algoritmu, srovnává složitost algoritmu komprese a algoritmu dekomprese. Symetrie se vyznačuje pro různé aplikace různými požadavky. Komunikace dvou uživatelů prostřednictvím SMS vyžaduje symetričnost algoritmu (komprese a dekomprese trvají přibližně stejně dlouho). Naopak komunikace klient-server může složitost výpočtu přesunout směrem k serveru (jedná se zřejmě o výkonnější hardware). Takové zohlednění ovšem vyžaduje spíše více druhů kompresních algoritmů než návrh jediného univerzálního. Symetrii je vhodné spíše definovat nějakým protokolem.

Nyní je možné začít konstruovat algoritmus zohledňující co nejvíce výše zmíněných parametrů. Jde nám hlavně o nízkou složitost algoritmu a velmi nízké nároky na paměť. Zároveň je třeba navrhnout algoritmus, který již po vytvoření nebude zastaralý. Potenciál programu by měl být externě rozšiřitelný. Kompresní algoritmus by měl být statický. A v případě využití slovníku použít typická trénovací data pro jeho výrobu (vzorek komprimovaných dat a pod.).

[2)] pro zde vybranou platformu *JavaME*

3. Bezztrátové kompresní algoritmy

Ze známých výsledků používaných v bezztrátové kompresi bude sestaven algoritmus, kombinující jejich přednosti pro CLDC. Podrobně budou rozebrány požadavky na paměť a výpočetní složitost, na něž jsou u *CLDC* kladeny velké nároky. Neméně důležitým srovnávacím kritériem je také kompresní poměr a u adaptivních metod délka stlačitelné zprávy.

Definice 3: (stlačitelná zpráva)
Termínem označuje průměrnou délku *textové* zprávy, pro niž algoritmus již není expanzivní.

Pozn.: Definice je značně neformální, ale čtenáři snad termín dostatečně přibližuje.

Metod bezztrátové komprese dat je nepřeberné množství a shrnovat zde principy všech, i jen dnes používaných metod, by bylo na samostatné dílo. Budu proto předpokládat, že čtenář má alespoň minimální znalost bezztrátových kompresních algoritmů (více například v [1]) a budu se podrobněji zabývat implementačními detaily a použitými datovými strukturami.

Pro naše účely budou pravděpodobně adaptivní kompresní algoritmy nepoužitelné nejen kvůli zmiňovanému malému vzorku dat, ale i kvůli časové složitosti. Jejich statické variace by mohly splňovat naše požadavky. Úprava adaptivního algoritmu na statický tak, aby zachovávala kompresní schopnosti co nejlépe, by mohla vypadat například následovně. Datová struktura definující kódy komprimovaných znaků upravovaná v průběhu komprese bude statická, definovaná před začátkem komprese a v jejím průběhu neměnná. Takovouto strukturu budu nazývat statickým slovníkem.

Definice 4: (statický slovník)
Pomocná datová struktura kompresního algoritmu definující kódy jednotlivých znaků vstupní abecedy neměnná po celý průběh kompresního algoritmu

Nyní k popisu jednotlivých algoritmů.

3.1. PPM

Kompresní metoda nazvaná Predictive Partial Matching byla poprvé publikována v [9] a má několik variant různě upravujících nároky na výpočet a paměť (například [3] nebo [4]). Hlavní myšlenka spočívá v predikci následujícího znaku v závislosti na předchozích písmenech (kontextu). Metoda je adaptivní, vytváří slovník v průběhu komprese. Proto se kvalita komprese na delších zprávách zlepšuje (data musí mít typický charakter).

Práci s kontexty lze shrnout do několika málo operací. Každý nalezený kontext (přípona zpracované části vstupu) maximální zadané délky se zapíše do slovníku a k němu se postupně připisují jeho následníci (četnost se zde přepočítává na pravděpodobnost). Každý kontext také obsahuje jako následníka escape znak (zastupující všechny ostatní znaky, které se v kontextu nevyskytly). Komprese znaku pak probíhá následovně: vezme se maximálně dlouhý kontext a v něm se hledá písmeno určené ke kompresi. Pokud se zde vyskytuje, kóduje se pravděpodobnost tohoto znaku (například aritmetickým kodérem), pokud se nevyskytuje, kóduje se escape a přechází se na kratší kontext. V případě, že kontext klesne pod minimální hranici, používá se uniformní rozdělení pravděpodobností.

Variabilita PPM spočívá ve velikosti a obnově slovníku. Lze též omezovat maximální délku kontextu, popřípadě zvolit různé modely určení pravděpodobnosti escape znaku.

Pro použití PPM na krátké textové zprávy je nutné z adaptivního algoritmu vytvořit statický (vyhledávací slovník natrénovat na vzorových datech viz úvodní odstavec kapitoly) a značně

omezit velikost statického slovníku (podle 2.3.). Nevýhodou adaptivního PPM je "dlouhý rozjezd", kdy je zpráva stlačitelná od cca $110B$ (viz experimentální výsledky 5.1.3.).

3.2. Huffmanovo a aritmetické kódování

Tyto dva algoritmy jsou nejrozšířenějšími entropickými kodéry vůbec. Nepracují s kontexty, ale s četnostmi znaků vstupní abecedy. Základním principem obou algoritmů je zkrátit délku kódu častěji se vyskytujících znaků na úkor délky kódu "řídkých znaků". Obě metody tak dokáží sestrojit podle dané frekvenční tabulky ideální kód pro jejich kompresi. Liší se pouze svým přístupem.

Huffmanovo kódování vychází z binárního stromu, kde pravý syn má označení 1 a levý 0. Jednotlivé znaky jsou reprezentovány listy stromu. Kódy znaků jsou tedy prefixově jednoznačné, tj. žádný znak není prefixem jiného a jejich rozlišení v komprimovaném signálu je jednoznačné. Nevýhodou Huffmanova kódování je reprezentace znaku celým počtem bitů (přestože optimální může být i 1,5 bitu pro daný znak). Nevýhodou je též nutnost bezprefixového kódu pro jednotlivé znaky.

Základní ideou aritmetického kódování je zápis zprávy jako čísla z intervalu $(0, 1\rangle$, každý znak na tomto intervalu má svůj podinterval (otevřený zleva a uzavřený zprava). V případě kódování znaku se interval zúží na daný podinterval a zde se opět pokračuje v dělení dle zadaných četností. Tento algoritmus s sebou přináší nemálo implementačních problémů (reprezentace reálných čísel pomocí přirozených, problémy se zaokrouhlováním, množství dat držených v paměti a pod.). Tímto algoritmem lze generovat kódy se skutečně ideální délkou kódu.

Použití obou metod je velmi vhodné pro *CLDC* (doplněný o transformace nebo o statistický model). Algoritmy jsou rychlé, s minimálními nároky na paměť (pouze statická frekvenční tabulka popřípadě prostor pro vybudování stromu pro určení Huffmanova kódu), ovšem v případě aritmetického kódování s nezanedbatelnými nároky na rychlost výpočtu.

3.3. Použití Optimal Tree Machine

Bezztrátový kompresní algoritmus používající Optimal Tree Machine byl publikován v roce 2005 - autoři Rissanen, Tabus a Korodi. Celá tato podkapitola je shrnutím článku [2]. Algoritmus je speciálně navržen pro kompresi krátkých textových zpráv a kombinuje v sobě výhody známých kompresních algoritmů. Při návrhu byla akceptována omezenost hardwaru. Ta se projevuje v parametru, kterým je možné omezit velikost pomocné datové struktury.

Navrhovaný algoritmus upravený pro mou implementaci budu nazývat metodou Mobile U Machine (dále jen MUM). MUM predikuje následující znak zprávy v závislosti na kontextu, který je právě zpracováván (podobně jako u PPM). Metoda PPM oproti MUM je ale adaptivní. Slovník metody MUM je vytvořen rozborem vzorových dat a poté podle prořezávacího algoritmu prořezán na "potřebnou" velikost (bude vysvětleno níže). Takto předpřipravený a prořezaný slovník je implementován do samotného kompresního algoritmu (kvůli omezenosti výkonu *CLDC* se složitá část algoritmu přesune na výkonnější hardware). Kontext predikovaného znaku je klíčem ve struktuře slovníku a v případě úspěšného vyhledávaní (znak se v kontextu vyskytl) je tento znak komprimován aritmetickým kódováním (podle parametrů kontextu). V případě neúspěšného vyhledávání se kóduje speciální znak escape a přechází na jiný slovník (detaily níže).

3.3.1. Maximal Tree Machine

Budování slovníku OTM probíhá přes tzv. Maximal Tree Machine (dále jen MTM), která je následně prořezána. Konstrukce MTM využívá zvláštních kontextů:

Definice 5: (minimální unikátní kontext)

Je posloupnost znaků $x_{k-m} \dots x_{k-1}$ taková, že pro celou vstupní množinu je unikátní a nemůže být kratší (vůči unikátnosti).

MTM tedy zahrnuje všechny minimální unikátní kontexty vyskytující se v trénovacích datech plus četnosti znaků pro každý takový kontext. Jelikož se jedná o stromovou strukturu, MTM obsahuje též všechny podmnožiny unikátních kontextů a u nich četnosti znaků odpovídající jejich kontextu.

Konstrukce MTM probíhá jedním průchodem vstupních dat. Do slovníku se přidá každý kontext, který je již unikátní. Pokud není unikátní prodlouží se a pokud nelze prodloužit vloží se aktuální. Při konstrukci uzlů a následném průchodu struktury se zaznamenávají četnosti (přičítá se 1 k četnosti znaku x_k). Z četností se později určí pravděpodobnost znaku v konkrétním kontextu nutná pro aritmetické kódování.

V MTM se kontexty vkládají odzadu (v případě potřeby kratšího kontextu stačí použít otce uzlu a není nutné procházet strukturou znovu). Takováto struktura je ovšem pro použití na *CLDC* příliš velká (při akceptovatelné velikosti MTM jako pomocného slovníku je zpracovaných dat příliš málo) a je vhodné ji prořezat.

3.3.2. Prořezávací parametr

Prořezávací operace vytváří z MTM jeden, a nebo i celou sadu OTM. Prořezávací parametr zmenšuje slovník tak, aby byl použitelný pro *CDLC*. Navíc definuje maximální povolenou odchylku pro statisticky získaná data (ostatní data, která se liší o více než tuto odchylku od průměru, se odříznou). Účelnost prořezávaní bude pravděpodobně patrnější po vysvětlení algoritmu.

Základem algoritmu je DFS,[3)] který provádí výpočet na každém uzlu. Je-li uzlem list, pak se spočte koeficient L tohoto listu:

$$L(s) := -\sum_{i \in A(s)} n_{i|s} log P_{i|s}$$

Pokud je uzel vnitřní, pak:

$$L(s) := 0$$

s určuje pro tento vztah kontext, který uzel reprezentuje, i je hodnota znaku abecedy, $A(s)$ je sada všech znaků vyskytujících se v kontextu s, $n_{i|s}$ je četnost výskytu znaku i v kontextu s a $P_{i|s}$ je pravděpodobnost jeho výskytu v kontextu s.[4)] Pro otce listu se navíc počítá důležitost každého jeho potomka (listu) I:

$$I_r(s) := -\sum_{i \in A(s)} n_{i|s} log P_{i|r}$$

Poté se porovná

$$I_r(s) \quad \text{a} \quad (1+p)L(s),$$

kde p je hodnota prořezávacího parametru. Pokud je $I_r(s)$ větší, list se zachová a otci se nastaví

$$L(r) := L(r) + L(s),$$

jinak

$$L(r) := L(r) + I_r(s)$$

a list se odřízne [2]. r značí kontext rodiče.

[3)] prohledání stromu do hloubky. Jde o průchod všech uzlů stromu tak, že se nejprve projdou všichni potomci uzlu a teprve uzel samotný

[4)] Oproti $n_{i|s}$ je v $P_{i|s}$ započtená i pravděpodobnost escape

Algoritmus tedy spočte relevanci všech listů. Pokud jejich parametry nevyhovují, odříznou se a relevance se přičte jejich otci. Takto se postupuje dokud strom nemá všechny listy relevantní (tj. jeden průchod DFS).

3.3.3. Optimal Tree Machine kodér

Poté, co je připravena pomocná slovníková struktura, zbývá již jen samotné kódování. Postup kodéru je velmi podobný procesu PPM. Máme na vstupu znak ch, předchozí kompresní kontext byl $x_1x_2 \dots x_{k-1}$. Nyní určíme kontext pro kódování ch. Ten bude maximální možný vyskytující se v slovníkové struktuře, ale jistě ne delší než $x_1x_2 \dots x_k$ (tomu mělo zabránit i prořezávání samotné). Vezmeme tedy kratší z $x_1 \dots x_k$ a celkové délky prefixu (až do začátku souboru). Nyní máme kontext komprese $x_{l-k+1} \dots x_k$ takový, že se vyskytuje ve struktuře slovníku. Zde nalezneme uzel reprezentující tento kontext. Pokud tento uzel má nenulovou pravděpodobnost výskytu znaku ch, kóduje se pravděpodobnost aritmetickým nebo jiným kodérem. Pokud pravděpodobnost výskytu ch je 0, pak se kóduje escape a přechází se do jiného kompresního stromu (s mírnějším parametrem prořezání a nebo jinými zdrojovými daty), kde se tento postup opakuje se stejným kontextem.

Jsou-li vyčerpány všechny kompresní stromy, kóduje se znak s uniformním rozdělením. Navíc do tohoto rozdělení nejsou zahrnuty znaky, které se vyskytly v předchozích uzlech kódovaných escape (je totiž zřejmé, že nejde o žádný z těchto znaků).

3.3.4. Metoda MUM

Při implementaci metody [2] jsem narazil na několik problémů, které metoda MUM musí řešit. Výsledky publikované v [2] byly získány na slovníku, který byl trénován na datech velikosti přes $700kB$, což je pro *CLDC* nepřijatelné. Každý uzel stromu OTM uchovává minimálně jeden znak a četnosti mnoha znaků. Velikost těchto základních údajů, které pomocná struktura musí obsahovat, jsou $2B$ pro reprezentaci znaku a $1-255B$ pro jednotlivé četnosti uložené jako *Byte* na jeden uzel. Empiricky zjištěná průměrná velikost je $8-10B$. Struktura použitelná na *CDLC* může tedy obsahovat maximálně $400-1200$ uzlů. V kapitole 5 jsou tyto empiricky získané výsledky podrobněji rozebrány. Zde je třeba zmínit, že trénovací data pro *CLDC* by neměla přesáhnout $2kB$.

Pravděpodobnost výskytu escape je v [2] určována podle modelu Krichevsky-Trofimov (viz 4.3.1.). S tou je možno dále pracovat a testovat výsledky jiných modelů. Implementaci shrnuje následující část práce.

Algoritmy popsané v 3.1., 3.2. shrnují část dnes používaných metod, které lze pro kompresi krátkých textových zpráv použít. Bylo by vhodné zmínit ještě varianty LZ77 a LZW, popřípadě jejich možné použití pro kompresi krátkých zpráv, ovšem jde o metody adaptivní, bez efektivní statické implementace. Více o těchto metodách [1].

4. Implementace metody MUM

Při implementaci metody MUM jsem se samozřejmě setkal s některými problémy a vlastnostmi samotného algoritmu nedefinovanými v [2]. Tyto detaily metody MUM bych rád v této kapitole popsal a zdůvodnil jejich použití, popřípadě alternativní varianty, jež jsem mohl také využít.

4.1. Datová struktura

Datová struktura na *CLDC* vyžaduje precizní návrh pro co největší úsporu paměti a současně minimální nároky na složitost vyhledávání. Variantami, jež je možné použít pro reprezentaci Optimal Tree Machine, jsou např. regulární strom, strom se spojovým seznamem potomků, jiná slovníková struktura (hašování podle vyhledávaného kontextu).

Jelikož metoda MUM využívá statický slovník pro kódování, nemusí nás zajímat složitost budování slovníku, ale pouze operace spojené s vyhledáváním ve struktuře[5)] a velikost struktury (optimalizované pro vyhledávání).

Následující požadavky určují datovou strukturu:

(i) **požadavky na údaje ve struktuře** - V datové pomocné struktuře slovníku je potřeba uchovávat **kontexty**. Pro každý kontext se uchovává **pravděpodobnost jednotlivých znaků**, které se v kontextu vyskytují, pravděpodobnost **escape** znaku pro daný kontext a **příslušnost** ke konkrétnímu OTM.

(ii) **požadavky na operace ve struktuře** - Algoritmus MUM v sobě obsahuje tyto požadavky na práci s daty: vyhledání maximálního povoleného kontextu, určení intervalu pro aritmetické kódovaní (tj. součet četností výskytu vybraných znaků v kontextu, četnosti výskytu jednotlivých znaků), příslušnost kontextu k danému OTM a test výskytu znaku v kontextu (tj. zda je třeba použít escape).

Nyní vyberme nejvhodnější strukturu pro reprezentaci našich požadavků s ohledem na slovník. Ten musí v paměti zabírat co nejméně místa. Zároveň složitost jednotlivých operací nesmí být příliš velká. Nejprve je ovšem třeba ozřejmit některé algoritmy.

4.1.1. Určení maximálního kontextu

Výběr kontextu pro komprimovaný znak se určuje z posledního použitého kontextu. Je zřejmé, že aktuální kontext nemá smysl prodlužovat "před" poslední použitý kontext.[6)] Maximální kontext pro kompresi se tedy vybere prodloužením předchozího kontextu o poslední komprimovaný znak a reprezentace tohoto kontextu se najde v příslušném OTM (pokud maximální neexistuje, použije se nejdelší jeho možný suffix). Po escapu se pokračuje obdobně v jiném OTM stromě. Pro stromy s postupně snižovaným parametrem prořezání to tedy znamená zkracování kontextu. Naopak při výměně kompresního stromu má smysl zvážit prodloužení kontextu na nejdelší poslední kontext v tomto stromě použitý.

Z hlediska tohoto postupu výběru je vhodné procházet kontexty lineárně podle počtu písmen, která obsahují. Nejdelší suffix vyskytující se ve stromě dostáváme tímto postupem "zdarma". Takováto struktura je stromová a proto se ji i následně budu držet.

[5)] nutno říci, že nemusí jít ani o stromovou strukturu, ta je potřeba především pro budování slovníku

[6)] Pokud by prodloužení před poslední kontext bylo obsaženo ve stromě, pak poslední použitý kontext mohl být delší (tj. nebyl vybrán maximálně dlouhý) neboť i jeho prodloužení musí být obsaženo ve stromě (pokud v tomto stromě existuje jako podřetězec nějakého kontextu, zde aktuálního kontextu), což je spor.

4.1.2. Parametry aritmetického kódování

Po určení kontextu komprese je třeba získat parametry pro aritmetické kódování. Konkrétně se jedná o podinterval intervalu $(0, 1\rangle$. Hranice jsou závislé na kontextu a kódovaném znaku (čím častěji se znak v trénovacích datech v daném kontextu vyskytl, tím větší pravděpodobností, a tedy i intervalem, bude reprezentován). Také závisí na počtu escape znaků kódovaných bezprostředně před kompresí aktuálního znaku. Všechny znaky vyskytující se v kontextech předchozích stromů se mohou předem z intervalu odstranit. Přístupů se proto nabízí hned několik.

Uzel stromu musí obsahovat své následníky a také četnosti znaků, které se v kontextu výskytují (dále jen četnosti). Z těchto četností je možné dopočítat pravděpodobnosti (poměr součtu všech četností + escape ku četnosti znaku), ale naopak to nelze. Všechny znaky totiž nemusí být zařazeny do kódované abecedy (a to ty, které se vyskytly v předchozích escape sekvencích, neboť žádný znak z nich nevyhovoval). Z tohoto pohledu je reprezentace pomocí četností dostačující. Jakákoli jiná struktura by obsahovala redundantní data (variantou je například trojrozměrné pole - kontext, znak, počet escape sekvencí), ale tato reprezentace by byla nejen redundantní, ale ještě navíc velmi "řídká",[7] což je při požadavku na co nejmenší pomocnou strukturu nepřípustné.

Další variantou je dvourozměrné pole znaků a escapů s ukazatelem na stromově setříděné kontexty. Nevýhodou takovéto struktury je opět náročnost na paměť kvůli redundanci kontextů v dané struktuře (přestože by některé údaje byly z uzlu odstraněny, řetězcová reprezentace kontextu by byla příliš náročná).

Po několika pokusech se jako ideální jeví vyhledávat nejprve podle kontextu, což je obvyklé ve většině kompresních algoritmů. Zvolme tedy jako základní datovou strukturu kontextový strom. V uzlu reprezentujícím daný kontext je následně nutné získat požadovaný interval. Alternativou k poli četností je dvourozměrné pole - znak×počet escapů. Reprezentace jedné četnosti zvětší svůj objem o

$$2 * \text{ počet OTM } \times \text{ počet následníků uzlu.}$$

Je nutné uložit zde i rozsah [8] Avšak při výpočtu dosáhneme konstantního času oproti $O(n)$ při lineárním průchodu četností a postupném načítání. Při použití dvourozměrného pole by bylo vhodné omezit se pouze na znaky, které se v kontextu vyskytují (v opačném případě by se nám zmnožily i tak zbytečné nuly) a počet escapů, který může nastat (v některých OTM se uzel vůbec vyskytovat nemusí). Dvourozměrné pole je vhodnou reprezentací pouze pro malý počet OTM a "řídký" strom (strom s nízkou aritou).

Variantou s ještě větší úsporu paměti představuje dynamický výpočet intervalu podle pomocných údajů (výše zmiňovaná složitost $O(n)$). Při tvorbě OTM se pracuje s četnostmi znaků v daném kontextu a právě ty lze při výpočtu intervalu použít. V programu by kód pro výpočet intervalu mohl vypadat například takto:

```
int[] UrciInterval(Uzel kontext, char[] abeceda, char znak) {
      char ch=getFirst();
      int mensi;
      while (ch<znak) {
            mensi+=kontext.znak[ch];
            ch=getNext(ch);
      }
      while (kontext.znak[ch]!=null) {
            vetsi+=kontext.znak[ch];
            ch=getNext(ch);
      }
      return int[] interval = {mensi, mensi+kontext.znak[znak], mensi+vetsi}
}
```

[7] Obsahující větší počet nul než skutečných dat.

[8] tj. součet četností.

Takovýto výpočet ušetří paměť, neboť pro každý znak, který se v kontextu vyskytuje, neuchovává několik intervalů, ale pouze jednu četnost. Výpočet bohužel vyžaduje průchod všech následníků.

4.1.3. Shrnutí

Pro implementaci některých návrhů z předchozí kapitoly je nutné využít hašovací schémata a další pomocné algoritmy, aby bylo potřebné složitosti dosaženo. Shrnutí navrhovaných metod:

metoda	**nalezení kontextu**	**parametry pro kodér**	**pamět**
3D pole	O(1)	O(1)	O(Esc·Con·Zn)
2D pole	O(n)	O(1)	O(Esc·Zn·(nutné Con))
kontextové 2D pole	O(n)	O(1)	O(Con·(nutné Zn)·Esc)
výpočet intervalu	O(n)	O(nutné Zn)	O(Con·(nutné Zn))

Con	- počet všech kontextů
Esc	- počet možných escapů (tj. počet stromů)
Zn	- velikost abecedy
nutné Con	- pouze kontexy, které pro daný počet escapů a znak existují
nutné Zn	- velikost části abecedy reprezentované v uzlu (s četností >0)
n	- délka kontextu

Při implementaci jsem se rozhodl slovníky budovat na stejných vzorových datech a jejich počet omezit na pět (podrobnosti viz 5. kapitola). Pro tuto konfiguraci bylo vhodnější zvolit reprezentaci slovníku pomocí uložených četností a dynamického načítání.

4.2. Slovník

Výsledky statických kompresních metod používajících slovník velmi závisí na trénovacích datech a modelech.[9] Výjimkou není ani metoda MUM. I zde je třeba zohlednit cílovou skupinu dat určených ke kompresi. Ideálními trénovacími daty je větší množství SMS od různých uživatelů. Kompresní slovník je vhodné vytvořit pro každý světový jazyk samostatně. Originální návrh OTM publikovaný v [2] navíc počítá s několika různými slovníky natrénovanými na různých datech. Z hlediska použitelnosti však v metodě MUM bylo nutné omezit velikost slovníku na možné minimum. Více slovníků trénovaných na stejných datech zabere o 1 bit více za každý kontext. Více slovníků získaných z různých dat by potřebovalo n-krát více paměťového prostoru. Proto jsem zvolil větší trénovací data a pouze slovníky konstruované na těchto datech (z jednoho MTM).

Vyšší relevanci slovníku lze získat přísnějším prořezávacím parametrem[10] se současně většími trénovacími daty. Na malých datech se mohou vyskytnout krajní případy výskytů některých znaků, které mají díky velikosti dat větší relevanci. V kombinaci s velkým prořezávacím parametrem MTM se tato data dostávají až do kompresního OTM, kde způsobují neefektivní kompresi (zabírají větší podinterval než je nutné).

Pro co nejefektivnější metodu je tedy nutné vzít velká vzorová data (velký balík SMS, mobile paketů) a malé prořezávací parametry. Vhodné je též více různých tréninkových dat (vhodně se doplňujících).

4.3. Pravděpodobnost escape znaku

Největší komplikací u prediktivních kompresních metod je pravděpodobnost escape. Metoda MUM není výjimkou. Zde je třeba zkonstruovat pravděpodobnost escape pro jednotlivé kontexty. Hledáme-li vhodný model pro efektní kompresi, je možné se opět inspirovat u již známých metod

[9] Například komprese mp3 závisí na konkrétním použitém psychoakustickém modelu, nebo statické Huffmanovo kódování na tabulce četností jednotlivých znaků

[10] Funkce prořezávacího parametru viz 3.3.2.

PPM, nebo čerpat z čistě teoretických pramenů. Cílem mé práce není ovšem najít ideální konfigurace. Do implementace MUM jsem proto zařadil Kritchevsky-Trofimov estimator doporučený i v [2]. Popis této metody i dalších variant následuje v dalším textu

4.3.1. Model Krichevsky-Trofimov

Model určuje pravděpodobnost všech znaků, jež se v kontextu nevyskytly (tj. pravděpodobnost escape znaku), jako $\left(m+\frac{1}{2}\right)/(n+1)$, kde m je počet různých symbolů, které se v kontextu vyskytly a n počet znaků abecedy. Pro potřeby statického OTM je pravděpodobnost upravena na

$$P_{ESC|s} = \begin{cases} \frac{m+\frac{1}{2}}{n+1} & \text{pro } 0 < m < \frac{d}{2} \text{ a } n > 1 \\ \frac{d-m}{n} & \text{pro } \frac{d}{2} < m < d \\ \frac{d-1}{d} & \text{pro } n = 1 \end{cases}$$

viz [2] str. 4, vztah (1). Velikost kódované abecedy je d a s je kontext, který uzel reprezentuje.

Model je odvozen v [5].

4.3.2. Model PPM

Při kompresi metodou PPM se používaný slovník mění podle komprimovaných dat a přístup k pravděpodobnostem je adaptivní. Tedy po každém průchodu se mění. To je hlavní rozdíl mezi PPM a MUM. Z adaptivních algoritmů PPM můžeme tedy vyvodit statický algoritmus pro metodu MUM.

Při klasickém přístupu odhadu escape podle četností ostatních znaků a předchůdců ve stromě je statická metoda velmi triviální. Ze získaných dat se jednoduše pravděpodobnost escape spočte a zafixuje. [11] Existují ale také složitější varianty odhadu escape. V PPMZ [6] je použita jednoduchá metoda SEE (Secondary escape estimation), která při druhém a každém dalším použití escape pro daný kontext se zvýší četnost u indikátoru escape (více viz [6]). Způsob jakým vyrobit z takovéhoto adaptivního kompresního algoritmu statický je netriviální. Metoda SEE je v tomto ohledu jiná, a proto jsem ji vybral jako model pro složitější postupy. Při stavbě stromu algoritmem OTM nemáme možnost ukládat výskyty escape, ovšem po vytvoření slovníku je možné tyto četnosti nacvičit přímo na dalších trénovacích datech, která mohou mít různé parametry: velikost, typ a pod. Takto se zachová charakter statického slovníku. Při této úpravě pravděpodobnosti escape dostaneme kompresní algoritmus velmi dobře reagující na data typu obou trénovacích slovníků. Průměrné výsledky metody však budou horší než pomocí optimalizovaného modelu Krichevsky-Trofimov.

Jinou úpravu escape by bylo vhodné otestovat empiricky na datech určených ke kompresi[12] Z důvodů omezené paměti pro slovník jsem dal přednost navrhované metodě Krichevsky-Trofimov.

[11] Takto například pracuje Krichevsky-Trofimov estimator.

[12] Z důvodu jejich specifičnosti

5. Výsledky implementace

Implementace se skládá z tříd pro práci se slovníkovou metodou a z aritmetického kodéru jehož autorem je © Bob Carpenter, 2002 zdroj [11]. Jeho kód je oddělen ve speciálním balíčku ArithColloquial. Na základě hotových tříd (ty jsou v programech různě variovány podle potřeb *JVM* a *KVM*) pak vzniklo několik programů, jejichž popis je v příloze 8.1 - 8.4.

Aby bylo možné provádět testy ve velkých objemech dat, pro kompresi a dekompresi byl použit program Tester, který implementuje MUM pro JVM. Z tohoto důvodu není zaznamenána doba komprese a dekomprese (neodpovídá cílovému zařízení). Další pomocný program je pak DictionaryMaker, který ze zadaného vstupního souboru vygeneruje pomocný slovník. Program MUM je software pro mobilní telefony, vybavený metodami pro připojení k internetu a případné stažení nových slovníků. Odpovídá nastavení *CDLC* a *MIDP 2.0*. Třídy použité ve všech třech programech jsou buď zachovány, nebo jen s nutnou úpravou pro dané zařízení a uživatelské rozhraní.

5.1. Testování

Bylo již zmíněno v předchozích kapitolách, že vytvořit objektivní srovnání pro metodu MUM je netriviálním problémem, který se mi doufám podařilo alespoň z části vyřešit. Metoda je specializována na konkrétní data, na kterých je nacvičen kompresní pomocný slovník. O adaptabilitě na konkrétní data by měla vypovídat část první. Druhá část se pak snaží určit optimální prořezávací parametry a počet OTM slovníků. Třetí část ukazuje výsledky na krátkých zprávách vůči jiným kompresním algoritmům.

Souhrnné výsledky jsou umístěny v příloze, stejně tak popis testovacích dat a trénovacích dat.

5.1.1. Adaptabilita

Pro testování adaptability algoritmu jsem zvolil Calgary Corpus a jako trénovací množinu dat vždy část některého ze souborů. Konkrétně se jedná o paper1, news zastupující textové soubory, geo, obj1, trans a pic zastupující speciální soubory. Protože tyto soubory jsou pro trénování slovníku příliš velké byl z nich vybrán náhodný úsek. Výsledky byly pořízeny programem Tester s prořezávacími parametry 0.0, 0.1, 0.25, 0.40 a 0.60 a jsou shrnuty v následující tabulce (celá tabulka viz příloha 8.6.1.):

seznam	size	geo	news	obj1	paper1	pic	trans	260sms
geo	102400	74964	141045	95813	143561	114516	112320	130030
news	377109	436949	256673	466147	265206	601589	284300	292871
obj1	21504	19803	28793	17089	29249	24044	23391	26293
paper1	53161	61939	35405	65510	35401	84891	38491	40264
pic	513216	176007	755442	241840	764536	75996	335934	676948
trans	93695	105286	81861	111100	80686	143723	66999	87455

První dva sloupce tabulky definují komprimovaný soubor. Další sloupce jsou pak výsledky pro konkrétní trénovací data (vždy pouze část krom 260.sms).

Na výsledcích je dobře vidět, že opravdu závisí na trénovacích datech (ovšem tento výsledek byl očekáván), která pokud se vhodně vyberou mohou rapidně zvýšit kvalitu komprese. Navíc tabulka ukazuje možnost využití algoritmu i v jiných oblastech komprese než jsou krátké textové zprávy.

5.1.2. Optimální prořezávací parametry

Pro odhad prořezávacích parametrů a počet OTM stromů mi pomohly výsledky uvedené v [2].

Zde je ovšem slovník natrénován na mnohem větším objemu dat. Testování probíhalo nejprve na jediném OTM stromě s horní hranicí velikosti struktury $2kB$. Jako optimální pro takovouto konfiguraci se ukazuje, že velikost statistických dat získaných slovníkem, je příliš malá. Graf (viz příloha 8.6.2.) ukazuje nejlépe vývoj vzhledem k prořezávacímu parametru. Jako nejefektivnější se jeví co "nejpřísnější" parametr. Z tohoto lze usuzovat, že bez prořezání jsou ve slovníku irelevantní data (okrajové možnosti s nízkou četností).

Test probíhal se dvěma různými slovníky *paper1.part* a *ObchodSRakvemi.part*, kde se postupně měnil prořezávací parametr. Tabulka shrnuje průběh testování (celá tabulka viz příloha 8.6.2.):

seznam	size	o0.0	o0.05	o0.1	o0.2	o0.5	o1.0	o2.0
020.sms	21	22	22	22	22	24	26	26
050.sms	51	40	40	39	40	46	49	50
080.sms	81	61	61	60	62	69	76	79
090.sms	91	61	61	59	61	78	85	85
100.sms	102	78	78	77	80	95	101	102
130.sms	131	85	85	84	87	102	114	115
160.sms	161	104	104	100	108	136	151	157
190.sms	191	121	121	117	127	146	165	173
220.sms	221	141	141	138	151	179	193	204
240.sms	241	157	157	154	162	192	215	229
250.sms	251	164	164	160	166	201	216	232
260.sms	261	174	174	167	175	207	231	239
mff.txt	11053	7418	7426	7370	7719	9179	10020	10231
pohadka1.txt	3788	2297	2297	2456	2571	2921	3211	3243
pohadka4.txt	5833	3531	3531	3815	3947	4597	5021	5072
pohadka6.txt	4630	2972	2972	2920	3096	3760	4161	4229
pohadka7.txt	11464	7379	7392	7254	7626	9190	10110	10388
zarovky.txt	2118	1379	1386	1371	1439	1694	1895	1974
celkem	40689	26184	26212	26363	27639	32816	36040	36828

seznam	size	p0.0	p0.05	p0.1	p0.2	p0.5	p1.0	p2.0
020.sms	21	24	24	24	24	25	25	25
050.sms	51	42	42	42	42	44	49	48
080.sms	81	65	65	65	64	72	75	74
100.sms	102	79	79	79	78	91	98	99
130.sms	131	95	95	95	100	108	112	113
160.sms	161	114	114	114	117	126	138	142
190.sms	191	133	133	133	136	147	163	165
220.sms	221	153	153	153	159	181	193	195
240.sms	241	170	170	170	169	188	206	206
250.sms	251	175	175	175	181	195	213	215
260.sms	261	189	189	189	193	214	219	223
mff.txt	11053	8306	8306	8306	8455	8937	9321	9553
pohadka1.txt	3788	2414	2414	2415	2656	2847	3043	3193
pohadka4.txt	5833	3739	3739	3738	4161	4433	4688	4886
pohadka6.txt	4630	3120	3120	3120	3241	3581	3868	4048
pohadka7.txt	11464	8546	8546	8545	8690	9128	9538	9732
zarovky.txt	2118	1641	1641	1641	1645	1737	1811	1844
celkem	40598	29005	29005	29004	30111	32054	33760	34761

Horní řádek tabulky obsahuje označení souboru trénovacích dat (p odpovídá *paper1.part.txt*, o odpovídá *ObchodSRakvemi.part.txt*) a prořezávací parametr.

Dalším testem bylo testování na více OTM stromech, za účelem určení jejich ideálního počtu. Roli zde hrála velikost slovníku a kvalita komprese. Výsledky shrnuje tabulka:

seznam	size	mecT1	mecT9	newsT1	newsT9
GardenOfEden.txt	356090	233925	233925	216520	216459
GreenHillsOfAfrica.txt	369165	235618	235618	218031	217949
ObchodSRakvemi.txt	7980	4949	4949	5742	5714
050.sms	51	40	40	40	39
100.sms	102	78	78	76	75
120.sms	121	81	81	83	82
160.sms	161	104	104	109	109
190.sms	191	122	122	125	126
220.sms	221	142	142	150	149
260.sms	261	175	175	180	178

Z tohoto testu je zřejmé, že vzorek trénovacích dat je příliš malý. Přestože slovník s devíti stromy (mecT9, newsT9) je několikrát větší než s jedním stromem, rozdíl v kompresi je zanedbatelný. Jev je patrně důkazem nedostatku trénovacích dat (data, která mají malou váhu jsou zastoupena ve velké míře, ale pro kompresi jsou nepodstatná).

Z výsledků testu jsem se snažil zkonstruovat ideální nastavení kodéru. Výsledků není příliš mnoho a navíc závislost velikosti slovníku a prořezávacího parametru spolu souvisí. Odhad je tedy jen velmi hrubý a vychází ze zde uvedených výsledků. Finální nastavení kodéru je pro dva stromy s prořezávacími parametry 0.0 a 0.2.

5.1.3. Komprese krátkých textových zpráv

Hlavní část testování se týkala komprese krátkých zpráv a srovnání s některými stávajícími kompresními algoritmy. Z nich jsem jako zástupce vybral PPMd, PPMonster, bzip, gzip, zip (všechny s nejlepším kompresním poměrem a nejnižší rychlostí). Jako testovací data jsem zvolil sadu SMS zpráv a některé rozřezané dlouhé soubory (po 200B, 300B, 500B, 700B a zde jsem jako konkurenci uvažoval pouze PPMd a PPMonster). Předpokládaným výsledkem testu je konstantní kompresní poměr pro program tester a zlepšující se tendence "klasických" algoritmů (všechny výsledky viz 8.6.4.):

file	size	bzip2	gzip	PPMd	PPMonster	zip	animal	obchod
020.sms	21	62	49	48	48	133	22	22
050.sms	51	85	75	67	68	161	40	39
080.sms	81	108	104	93	92	190	61	59
110.sms	111	125	119	112	110	205	79	75
140.sms	141	145	136	127	126	222	96	87
170.sms	171	162	156	149	145	242	124	121
200.sms	201	184	175	166	164	261	135	129
220.sms	221	194	182	173	170	268	148	139
230.sms	231	201	188	175	171	274	150	142
250.sms	251	219	209	189	185	295	176	161
260.sms	261	225	210	193	193	296	185	168

file	size	obchod	animal	PPMd	PPMonster
garden200	356000	254236	229176	270028	273625
garden300	355800	249015	223987	235656	239072
garden500	356000	245083	220045	203535	206693
garden700	355600	243080	218086	186758	189887
konec200	70600	47933	56067	61214	61012
konec300	70500	46863	54981	54712	54751
konec500	70500	46052	54172	48440	48777
konec700	70000	45375	53441	44957	45436

garden200 je po $200B$ rozdělený soubor *GardenOfEden.txt* a *konec200* po $200B$ rozdělený soubor *KonecSveta.txt*. Ostaní analogicky. Výsledky byly pořízeny pro konfiguraci programu tester: 5 stromů 0.0, 0.1, 0.25, 0.4 a 0.6.

Tabulka neukazuje velikost využité paměti, neboť u některých programů ji nelze jednoduše

změřit, přestože by srovnání bylo vhodné. Ze získaných výsledků je možné konstatovat, že metoda je použitelná a její mobilní verzi má smysl používat. Srovnání s metodou [7] se bohužel nepodařilo získat. Lze uvažovat pouze porovnání hodnot uvedených v [7].

5.1.4. Testovací data

Pro testování metody jsem zvolil několik druhů souborů, převážně však soubory textové (cílová skupina, na kterou se programy soustřeďují). Jako standard pro testování kompresních algoritmů je běžně používán Calgary Corpus, proto jsem jej využil také. Výsledky na něm získané slouží spíše pro srovnání s "klasickými" kompresními programy. Další data jsou knihy od Ernesta Hamingwaye Green Hills of Africa a Garden of Eden (anglicky), Haškův Obchod s Rakvemi (česky), knihy Andzreje Sapkowského Meč Osudu, Něco končí, něco začíná, Konec světa (česky), knihy George Orwella 1984 a Animal Farm (anglicky) poslední druh testovacích dat jsou kratší texty Matfyzáckých pohádek a soubor SMS zpráv různé délky (1 – 260 znaků). Tyto soubory lze nalézt na přiloženém CD.

Trénovací data jsou uložena v adresáři příloh dictionaries a jsou to části některých souborů (lze rozpoznat podle názvu nebo instrukcí v README).

6. Závěr

Moje práce otevírá novou problematiku oblasti komprese dat. Snažil jsem se nastínit několik standardních postupů a upozornit na nové pohledy na problematiku krátkých zpráv a navíc implementovat jedno z navrhovaných řešení. Mé výsledky je ovšem těžké hodnotit, neboť neexistuje příliš alternativ k mé práci.[13] V příloze 7.1. jsou shrnuty výsledky na vzorových datech (sebraných vzorcích SMS a Calgary Corpusu). Algoritmus lze porovnat s běžně používanými archivačními programy například podle délky zpráv, na které je komprese metody MUM efektivnější.

Vývojem aplikace vznikl software simulující kompresi SMS na mobilních zařízení. Software samotný je nutné uzpůsobit hardwaru, pro který je určen a proto vznikla aplikace pouze pro Java Virtual Maschine. Další vývoj mobilní aplikace pak závisí hlavně na uživatelsky příjemné implementaci do mobilního zařízení.

Trendem v mobilních technologiích je v dnešní době rozšiřování o nejrůznější aplikace známé z PC. Alternativou algoritmu MUM není jen úprava slovníku pro SMS, ale dnes již také rozšíření databáze slovníků o slovníky pro mobilní internet (internetové pakety), slovníky pro MMS, případně různá rozšíření bezdrátových protokolů. V tomto ohledu je metoda MUM univerzální a lze ji s úspěchem upravit pro konkrétní data. Vývoj aplikace do budoucna by proto měl reagovat na komerční požadavky.

[13] Pokud vím jedinou alternativou je zatím [7]

7. Použitá literatura

[1] J. Čapek, P. Fabian, *Komprimace dat, principy a praxe*, ISBN 80-7226-231-9, Computer Press, vydání první, Praha 2000

[2] Gergely Korodi, Jorma Rissanen, Ioan Tabus, *Lossless Data Compression Using Optimal Tree Machines*, Proc. IEEE Data Compression Conference 2005, ISBN 0-7695-2309-9, ISSN 1068-0314, str. 348-357

[3] A. Moffat, *Implementing the PPM Data Compression Scheme*, Proc. IEEE Transactions on Communications 1990, vol. 38, no. 11, str. 1917-1921

[4] D. Shkarin, *PPM: One Step to Practicality*, Proc. IEEE Data Compression Conference 2002, ISBN 0-7695-1477-4, ISSN 1068-0314, str. 202-211

[5] R. Krichevsky, V. Trofimov, *The Performance of Universal Encoding*, IEEE Trans. Information Theory 1981, vol. IT-27, no. 2, str. 199-207

[6] Zijun Wu, *A Tutorial of The Context Tree Weighting Method: Basic Properties*, http://www. .nd.edu/j̃nl/ee80653/tutorials/zijun.pdf, 2005

[7] S. Rain, C. Guehmann, F. Fitzek, *Low Complexity Compression of Short Messages*, Proc. IEEE Data Compression Conference 2006, ISBN 0-7695-2545-8, ISSN 1068-0314, str. 123-132

[8] Sun Microsystems, *dokumentace JavaME, CLDC a MIDP 1.0*, http://java.sun.com/javame/ /index.jsp

[9] T. Bell, J. Cleary, I. Witten, *Data compression using adaptive coding and partial string matching*, IEEE Transactions on Communications 1984, Vol. 32 (4), str. 396-402

[10] M. Nelson, *Arithmetic coding and statistical modeling*, http://dogma.net/markn/articles/ /arith/part1.htm, 1991

[11] B. Carpenter, *Arithmetic coder library*, http://www.colloquial.com/ArithmeticCoding/, 2002

8. Příloha

Tato kapitola obsahuje dokumentaci ke všem programům (MUM, Tester a DictionaryMaker). Druhá část dokumentace, tj. automaticky generovaný dokument *javadoc*, lze nalézt na přiloženém CD (adresář dokumentace). Další kapitolou v tomto oddílu jsou pak kompletní tabulky výsledků testování, které byly ve zkrácené podobě zařazeny do textu práce v kapitole 5.

8.1. Úvod k programům MUM, Tester a DictionaryMaker

Mou implementační část práce tvoří tři programy. Cílem všech je komprese krátkých textových zpráv. První zde popisovaný MUM je software pro mobilní telefony. Tester je program testující algoritmus na PC, je tedy variantou MUM. Poslední je DictionaryMaker, který vytváří zásuvné moduly z trénovacích dat, je určen jako podpora programu MUM.

Pro správné pochopení běhu programů je nutné přečtení [2]. Zde pouze velmi stručně shrnu práci algoritmů v programech. Popis datových struktur je pak v kapitole 4.

Všechny programy pracují pouze s platformou *JVM* (program MUM s platformou *KVM*). Proto je pro jejich spuštění nutná nainstalovaná java. Spuštění se pak provádí příkazem `java -jar "nazev a cesta k programu" <parametry programu>`. Pro program MUM je spouštěcím programem i `mum.jad`, ovšem je nutná podpora *KVM*.

8.1.1. Zadání práce

Cílem práce je vývoj softwaru pro kompresi SMS na mobilních telefonech. Vstpní data programu tvoří krátká textová zpráva. Výstupem je komprimovaná zpráva. Nebo naopak. Kompresní algoritmus pracuje jako algoritmus popisovaný v [2]. Algoritmus používá pomocnou datovou strukturu (dále slovník), která má být zpracována jako externí zásuvný modul. Jako součást programu je povolen externí aritmetický kodér.

8.1.2. Výběr platformy

Celý ročníkový projekt je zpracován v programovacím jazyce Java, jednotlivé programy se však liší. Program Tester a DictionaryMaker pracují na standardním *JVM*. Program MUM pracuje na *KVM* s konfigurací *CDLC* 1.0 a platformou *MIDP* 2.0. Více o těchto platformách a konfiguracích viz [8].

8.1.3. Licence

Součástí programu je externí aritmetický kodér jehož autorem je © Bob Carpenter 2002. Ve všech programech je uložen v balíčku **ArithColloquial**.

8.2. Program MUM

8.2.1. Programátorská dokumentace

Součástí této části dokumentace je i generovaný *javadoc*.

Algoritmus: Program je určen ke kompresi krátkých textových zpráv, je omezen $32kB$ operační paměti a $128kB$ persistentní paměti. Procesor má alespoň 16bit. Těmto parametrům podléhají všechny pomocné datové struktury i složitost algoritmu. Program importuje slovník z internetu. Tento slovník obsahuje parametry pro aritmetické kódování pro jednotlivé znaky

abecedy. Slovník vychází z kontextového modelu. Jde o stromovou strukturu kontextů, kde v každém uzlu je vždy uložena četnost výskytu všech jeho následníků. Z těchto četností se parametry pro aritmetické kódování dopočítávají. Při escape události (znak s v kontextu nevyskytuje) se přechází na jiný slovník (což by mělo znamenat i kratší kontext). Podrobnější popis je právě v [2] nebo v mé bakalářské práci.

Program je rozdělen do několika balíčků:

ArithColloquial - licencovaný externí aritmetický kodér spravuje třídy ArithCoder, ArithEncoder, ArithDecoder, BitInput, BitOutput dokumentace jejich funkce viz autorovy stránky [11].

MIDlet - obsahuje pouze jednu třídu CMidlet, která spravuje veškerou interakci s uživatelem.

coder - obsahuje třídy, které zajišťují kompresi a dekompresi zpráv.

support - pomocné třídy obstarávající připojení k internetu, práci s databází zařízení apod.

Konstanty použité v programu: součást *javadoc*.

Formáty vstupů a výstupů jednotlivých programů:

Program MUM dostává několik vstupů. Hlavním vstupem je neformátovaný text zadaný uživatelem. Dalšími vstupy jsou:

Slovník - slovník je stromová datová struktura (slouží pro vyhledávání kontextových závislostí textu), je uložen průchodem DFS (prohledáváním do hloubky, ukládá se nejprve uzel samotný a poté jeho potomci) u každého uzlu jsou zaznamenávány postupně parametry:

typ	počet	popis
int	1	počet znaků vyskytujících se v kontextu uzlu
CFreqPocket	předchozí údaj	CFreqPocket obsahuje dva chary udávající označení znaku a četnost výskytu
char	1	četnost pro escape znak
boolean	TREES_NUM (tj. 5)	toto pole booleanu určuje náležitost k jednotlivým OTM stromům
char	1	hodnota posledního znaku kontextu, který uzel obsahuje (prodloužení otce)
int	1	počet synů uzlu

Tento slovník je vstupem programu MUM a zároveň výstupem programu CDictionaryMaker.

Slovníková databáze - je soubor na internetové adrese, který spravuje slovníky. Program z tohoto souboru dostane informace o nabízených slovnících na dané adrese. Každý slovník má unikátní číslo až do *Integer.MAX_VALUE* a podle něj se určuje název souboru slovníku (desetimístné číslo s příponou `.lng`). Ve slovníkové databázi se potom uchovávají slovníky jako jejich identifikační číslo a slovní popis tj.:

typ	počet	popis
int	1	identifikační číslo slovníku
char[]	1	slovní popis slovníku ukončený (char)0

Jiné vstupy do programu nezasahují.

Program nemá žádné externí výstupy, pouze výstup na obrazovku. V tomto výstupu je zahrnut průběh komprese, tj. doba, délka původní a délka komprimované zprávy, dekomprimovaná zpráva a hlášení o průběhu komprese. Program je ale připraven k úpravě tak, aby na výstupu byla komprimovaná zpráva. Tento výstup není v této verzi programu dostupný, neboť závisí na implementaci na konkrétní hardware (ten nebyl při vývoji k dispozici). Výstupní šifrovaná zpráva pak má formát:

typ	počet	popis
int	1	počet znaků zprávy
int	1	identifikace slovníku
byte[]	1	kód generovaný kodérem

8.2.2. Uživatelská dokumentace

Při spuštění programu na mobilním zařízení je uživatel nucen interagovat na několika obrazovkách. Popis jednotlivých funkcí následuje:

Úvodní obrazovka - spuštění MIDletu

Výběr jazyka - tato obrazovka se načte na začátku pokud MIDlet má v databázi alespoň jeden slovník. Zde je možné si vybrat, kterým slovníkem se bude zpráva komprimovat (tj. zásuvný modul z databáze zařízení). Rozdělení by mělo být intuitivní podle jazyka uživatele. Další možností je také stažení slovníku z internetové databáze.

Výběr internetové databáze slovníků - tato obrazovka umožňuje uživateli vybrat externí internetovou databázi kompresních slovníků. Je předvyplněna na databázi výrobce. Potvrzení výběru znamená stažení obsahu externí databáze. Zrušení výběru znamená ukončení programu. Databáze je uložena zatím na adrese http://gimli.matfyz.cz/language/

Žádný slovník v externí databázi - obrazovka se zobrazí pokud nebyl nalezen soubor "all" na internetové adrese. Možnosti uživatele jsou návrat na zadání adresy a ukončení aplikace.

Výběr slovníku z internetové databáze - umožnění uživateli výběru slovníku v externí internetové databázi.

Zadání textové zprávy - Uživatel zadá text komprimované zprávy.

Výstupní informace - Obrazovka shrnuje výsledky komprese. Čas komprese a výpise dekomprimované zprávy (test identicity). Zároveň informuje o průběhu komprese. Možnost návratu k napsané zprávě (ta již ale nelze změnit, neboť jedno spuštění aplikace je dovolena pouze jedna komprese zprávy).

8.3. Program Tester

Součástí této dokumentace je i *javadoc*.

8.3.1. Programátorská dokumentace

Algoritmus: Z trénovacích dat je zpracován pomocný slovník podobně jako u DictionaryMaker ovšem bez úsporné reprezentace (na PC stačí reprezentace pomocí třídy COtm. Poté komprimuje/dekomprimuje soubory podobně jako program MUM (viz popis MUM a [2]). Rozdílná je datová struktura slovníku, který nevyužívá platformy *KVM* ale *JVM*. Proto místo RMS balíčků (RMS je způsob zaznamenávání persistentních dat na mobilních zařízeních) využívá spojové seznamy ad.

Použité konstanty - jsou v *javadoc* v části konstanty.

Formát vstupu a výstupu:
Program umí komprimovat i dekomprimovat textové soubory. Na vstupu tedy musí být testovací data ve formátu textového souboru a to buď komprimovaná data vygenerovaná programem nebo textová data určená ke kompresi. Postup zadání je v uživatelské dokumentaci.

8.3.2. Uživatelská dokumentace

Protože program simuluje běh programu MUM na textových souborech, je uživatelské rozhraní navrženo pro co nejjednodušší obsluhu z příkazové řádky. Program s parametrem `--help` generuje návod k použití. Parametry programu jsou `-cd <soubor slovnku> [datove soubory]`. Volba

`-d` indikuje dekompresi zatímco `-c` indikuje kompresi. Slovník je zde ve formátu trénovacích dat (je vyroben, podobně jako v programu DictionaryMaker). Komprimované soubory předkládané k dekompresi nemusí mít příponu *.mum*, která je vygenerovaná pří kompresi, ale musí pro ně být použit stejný slovník (to aplikace nekontroluje).

8.4. Program DictionaryMaker

Součástí této dokumentace je i *javadoc*.

8.4.1. Programátorská dokumentace

Algoritmus: Ze vstupního textového souboru jsou vybrány unikátní podřetězce (unikátní v celém souboru) s minimální délkou (při zkrácení o jeden znak již nejsou unikátní). Ty jsou uloženy do MTM. Zároveň při této operaci je jejich následníku (znak následující kontextu) zvýšena četnost ve všech uzlech slovníku, kterými se prochází. Průchod se provádí proudovým způsobem, tj. soubor se prochází od začátku a každý kontext se prodlužuje (na začátku má délku jeden znak) dokud není unikátní. Takto vznikne MTM.

MTM je třeba následně prořezat podle prořezávacího parametru uloženého v konstruktoru třídy COtm. Tento konstruktor za pomoci metody otmInic prořezává stromovou strukturu MTM podle algoritmu v [2]. Třída CDictionary a CDictionaryNode je již mým vlastním rozšířením pro maximální úsporu paměti. Reprezentace znaků a jejich četností pomocí pole indexovaného znaky (v jednotlivých uzlech) se mění na pole CFreqPocket *setříděných podle četnosti* (vícečetné znaky jsou na začátku pole neboť při vyhledávání se pole prochází lineárně). Vše ostatní zůstává zachováno. CDictionary se poté vyexportuje do souboru. Podrobnější popis viz kapitola 4.

Použité konstanty: *javadoc*, část konstanty.

Formát vstupu a výstupu:

Vstup - Nároky na vstup programu nejsou téměř žádné. DictionaryMaker generuje z neformátovaného textového souboru (pro Windows přípona *.txt*) požadovaný zásuvný modul použitelný v programu MUM.

Výstup - Formát výstupního souboru je blíže popsán v kapitole Program MUM - formát vstupu a výstupu. Stromová struktura slovníku je kompaktně uložena průchodem DFS algoritmu. Nejprve se ukládá uzel, poté jeho potomci a u každého uzlu je zaznamenán počet znaků v kontextu, četnosti těchto znaků, poslední znak kontextu a počet synů uzlu.

Testovací data - Jsou k nalezení v CD přílohách, jinak lze samozřejmě použít libovolný textový soubor. Doporučuji velikost max $2kB$, větší slovníky nemusí mobilní aplikace zvládnout (experimentální výsledky - více v mé bakalářské práci).

8.4.2. Uživatelská dokumentace

Užívání programu pro generování slovníků je velmi jednoduché, uživatel se může omezit pouze na příkazovou řádku, kde jako *první parametr* zadá cestu ke vstupnímu neformátovanému textovému souboru (který v sobě shrnuje charakteristická data určená ke kompresi) a jako *druhý parametr* uvede cestu k cílovému souboru, kam má být slovník vygenerován. Cílový soubor musí mít v názvu své identifikační unikátní jméno, což je deseticiferné číslo (všechny cifry nemusí být platné) a přípona *.lng*. Příklad názvu: *0000050124.lng*.

Takto připravený slovník je možné použít v programu MUM, po umístění na nějakou webovou adresu a doplnění o správce slovníků (formát správce viz kapitola Program MUM).

8.5. Zhodnocení

Všechny tři programy poskytují komplexní přehled práce algoritmu z [2]. Program MUM testuje schopnost přenositelnosti na mobilní zařízení. Program Tester testuje kvalitu algoritmu na větším množství dat. Program DictionaryMaker poskytuje programu MUM vyměnitelný zásuvný modul, díky němu lze i nadále zlepšovat efektivitu software.

8.5.1. Nedodělky

Bohužel se software nepodařilo otestovat na hardware, pro který byl vyvíjen. Tento nedostatek způsobuje i nedostatečně kvalitně navržené parametry algoritmu.

Celou dokumentaci včetně *javadoc* lze najít na přiloženém CD.

8.6. Tabulky

8.6.1. Test adaptability

Testováno na datech Calgary Corpusu, v adresáří TestingData/calgary. Konfigurace programu tester je $p = [0.0, 0.1, 0.25, 0.4, 0.6]$. Trénovací soubory jsou v hlavičce tabulky.

seznam	size	anim	castaslova	geo	news	obchod	obj1	paper1	pic	trans	260sms
bib	111261	88966	93878	129598	80725	95109	135154	82795	177969	85098	89807
book1	768771	455157	527037	889557	467310	508437	963906	465544	1235358	535401	542535
book2	610856	391388	435821	712035	385435	430677	757163	384332	974893	429865	441243
geo	102400	155156	131172	74964	141045	155638	95813	143561	114516	112320	130030
news	377109	273255	288676	436949	256673	301151	466147	265206	601589	284300	292871
obj1	21504	32346	26507	19803	28793	31408	17089	29249	24044	23391	26293
obj2	246814	361774	301475	253670	318689	352680	247460	329457	305249	279785	298727
paper1	53161	36888	39882	61939	35405	40516	65510	35401	84891	38491	40264
paper2	82199	49669	56519	95598	49890	55206	102774	49782	131312	57089	58121
paper3	46526	28764	32565	54375	28504	31725	57823	28343	74305	32299	33141
paper4	13286	8259	9251	15471	8185	9189	16506	8153	21201	9200	9374
paper5	11954	8190	8805	13905	7915	8872	14773	7952	19046	8724	8904
paper6	38105	27322	29106	44221	26275	29779	46819	26210	60988	28149	29038
pic	513216	847074	677115	176007	755442	822458	241840	764536	75996	335934	676948
progc	39611	31575	32557	45402	29824	35622	48437	29979	63178	31641	33289
progl	71646	58139	58289	82935	54008	65951	86642	52749	113750	54551	61548
progp	49379	37257	38721	56021	36955	47337	59899	35191	79157	36697	43430
trans	93695	91930	88474	105286	81861	98562	111100	80686	143723	66999	87455
celkem	3251493	2983109	2875850	3267736	2792934	3120317	3534855	2819126	4301165	2449934	2903018

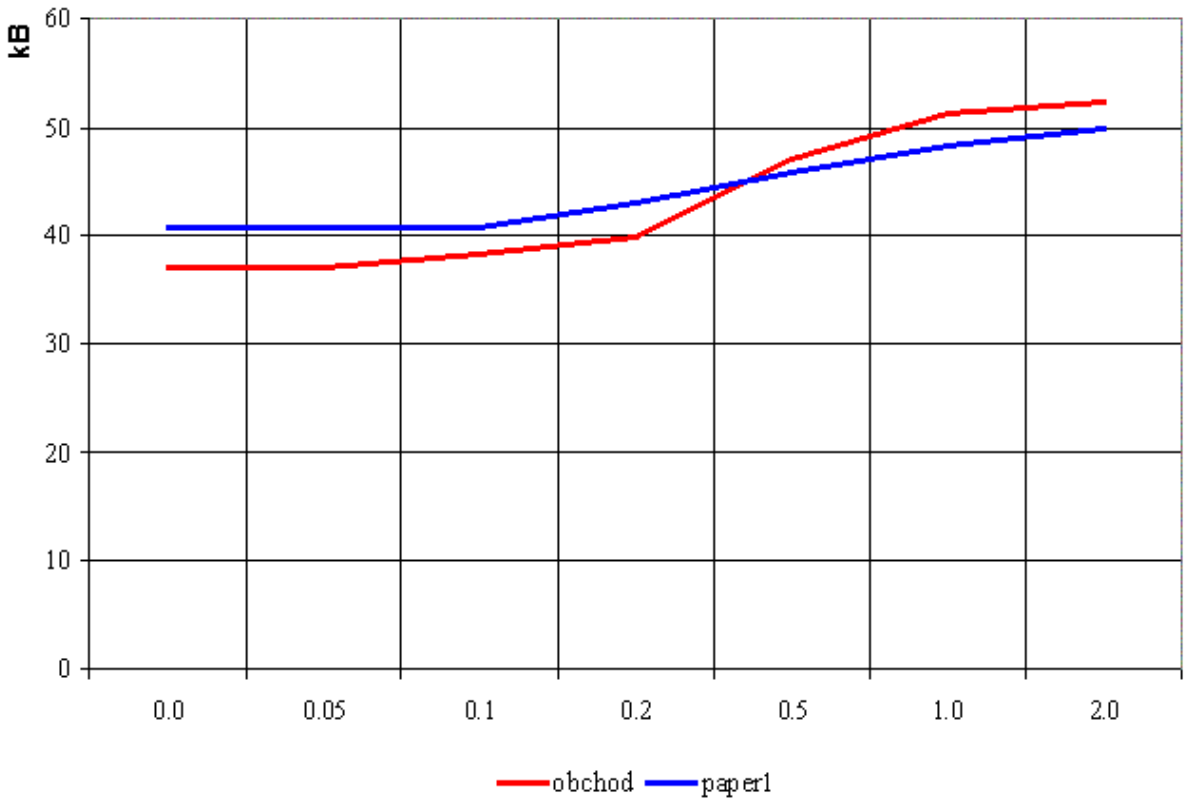
8.6.2. Testy parametrů

Konfigurace programu tester jsou p = [0.0], [0.05] ... viz hlavička tabulky. Trénovací soubory ObchodSRakvemi.part.txt a paper1.part.txt.

seznam	size	o0.0	o0.05	o0.1	o0.2	o0.5	o1.0	o2.0
001.sms	2	11	11	11	11	11	11	11
002.sms	3	12	12	12	12	12	12	12
010.sms	11	18	18	17	17	18	19	19
020.sms	21	22	22	22	22	24	26	26
030.sms	31	28	28	27	29	36	38	38
040.sms	41	33	33	33	35	43	43	45
050.sms	51	40	40	39	40	46	49	50
060.sms	61	44	44	43	44	52	58	60
070.sms	71	52	52	51	54	63	73	76
080.sms	81	61	61	60	62	69	76	79
090.sms	91	61	61	59	61	78	85	85
100.sms	102	78	78	77	80	95	101	102
110.sms	111	75	75	75	82	98	103	106
120.sms	121	81	81	79	83	100	111	113
130.sms	131	85	85	84	87	102	114	115
140.sms	141	89	89	87	91	115	125	129
150.sms	151	95	95	94	102	121	128	130
160.sms	161	104	104	100	108	136	151	157
170.sms	171	122	122	120	129	146	163	167
180.sms	181	120	120	113	119	148	158	163
190.sms	191	121	121	117	127	146	165	173
200.sms	201	133	133	128	139	157	172	176
210.sms	211	134	134	129	137	163	180	189
220.sms	221	141	141	138	151	179	193	204
230.sms	231	145	145	142	147	194	207	216
240.sms	241	157	157	154	162	192	215	229
250.sms	251	164	164	160	166	201	216	232
260.sms	261	174	174	167	175	207	231	239
mff.txt	11053	7418	7426	7370	7719	9179	10020	10231
pohadka1.txt	3788	2297	2297	2456	2571	2921	3211	3243
pohadka2.txt	4341	2782	2782	2982	3073	3490	3786	3836
pohadka3.txt	3159	1872	1872	2043	2092	2438	2626	2671
pohadka4.txt	5833	3531	3531	3815	3947	4597	5021	5072
pohadka5.txt	8443	5062	5062	5676	5829	6683	7315	7401
pohadka6.txt	4630	2972	2972	2920	3096	3760	4161	4229
pohadka7.txt	11464	7379	7392	7254	7626	9190	10110	10388
zarovky.txt	2118	1379	1386	1371	1439	1694	1895	1974
celkem	58371	37092	37120	38225	39864	46904	51368	52386

seznam	size	p0.0	p0.05	p0.1	p0.2	p0.5	p1.0	p2.0
001.sms	2	11	11	11	11	11	11	11
002.sms	3	11	11	11	11	12	12	12
010.sms	11	18	18	18	17	17	18	18
020.sms	21	24	24	24	24	25	25	25
030.sms	31	30	30	30	30	34	37	36
040.sms	41	36	36	36	38	39	44	44
050.sms	51	42	42	42	42	44	49	48
060.sms	61	50	50	50	50	52	53	54
070.sms	71	59	59	59	60	64	69	73
080.sms	81	65	65	65	64	72	75	74
090.sms	91	67	67	67	70	74	78	79
100.sms	102	79	79	79	78	91	98	99
110.sms	111	80	80	80	81	87	100	102
120.sms	121	88	88	88	92	97	101	103
130.sms	131	95	95	95	100	108	112	113
140.sms	141	99	99	99	101	114	124	126
150.sms	151	104	104	104	105	114	127	131
160.sms	161	114	114	114	117	126	138	142
170.sms	171	127	127	127	130	141	150	154
180.sms	181	136	136	136	139	144	155	159
190.sms	191	133	133	133	136	147	163	165
200.sms	201	142	142	142	143	154	168	171
210.sms	211	148	148	148	151	171	186	188
220.sms	221	153	153	153	159	181	193	195
230.sms	231	156	156	156	164	184	204	206
240.sms	241	170	170	170	169	188	206	206
250.sms	251	175	175	175	181	195	213	215
260.sms	261	189	189	189	193	214	219	223
mff.txt	11053	8306	8306	8306	8455	8937	9321	9553
pohadka1.txt	3788	2414	2414	2415	2656	2847	3043	3193
pohadka2.txt	4341	2874	2874	2874	3152	3340	3592	3740
pohadka3.txt	3159	1985	1985	1987	2208	2371	2499	2594
pohadka4.txt	5833	3739	3739	3738	4161	4433	4688	4886
pohadka5.txt	8443	5432	5432	5431	6209	6556	6912	7234
pohadka6.txt	4630	3120	3120	3120	3241	3581	3868	4048
pohadka7.txt	11464	8546	8546	8545	8690	9128	9538	9732
zarovky.txt	2118	1641	1641	1641	1645	1737	1811	1844
celkem	58371	40658	40658	40658	43073	45830	48400	49996

Následující graf zachycuje vliv parametru prořezání na výslednou kompresi (shrnutí předchozích tabulek, v grafu je zachycen řádek obou tabulek *celkem*). Obě funkce jsou neklesající pro testovaná data:



8.6.3. Testování počtu použitých OTM stromů

Konfigurace programu tester je p = [0.0] [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6]. Trénovací soubory jsou MecOsudu.part.txt a news.part.txt.

seznam	size	mecT1	mecT9	newsT1	newsT9
GardenOfEden.txt	356090	233925	233925	216520	216459
GreenHillsOfAfrica.txt	369165	235618	235618	218031	217949
ObchodSRakvemi.txt	7980	4949	4949	5742	5714
001.sms	2	11	11	11	11
002.sms	3	12	12	11	11
010.sms	11	17	17	18	17
020.sms	21	22	22	23	22
030.sms	31	27	27	29	29
040.sms	41	34	34	34	34
050.sms	51	40	40	40	39
060.sms	61	46	46	48	47
070.sms	71	53	53	56	55
080.sms	81	61	61	61	60
090.sms	91	62	62	65	65
100.sms	102	78	78	76	75
110.sms	111	75	75	78	78
120.sms	121	81	81	83	82
130.sms	131	86	86	92	92
140.sms	141	90	90	98	98
150.sms	151	96	96	101	100
160.sms	161	104	104	109	109
170.sms	171	121	121	120	119
180.sms	181	121	121	131	129
190.sms	191	122	122	125	126
200.sms	201	132	132	137	136
210.sms	211	135	135	143	143
220.sms	221	142	142	150	149
230.sms	231	145	145	153	152
240.sms	241	158	158	163	163
250.sms	251	166	166	172	171
260.sms	261	175	175	180	178

8.6.4. Srovnání se standardními kompresními algoritmy

Konfigurace programu tester je p = [0.0,0.1,0.25,0.4,0.6]. Trénovací soubory jsou AnimalFarm.part.txt a ObchodSRakvemi.part.txt. Ostatní algoritmy jsou volně dostupné PPMd a PPMonster, lze nalézt na přílohovém CD v adresáři algoritmy. Nastavení ostatních algoritmů je nejpomalejší = nejlepší kompresní poměr.

file	size	bzip2	gzip	PPMd	PPMonster	zip	animal	obchod
001.sms	2	39	30	28	29	114	11	11
002.sms	3	42	31	30	30	115	12	12
010.sms	11	53	39	38	38	123	18	17
020.sms	21	62	49	48	48	133	22	22
030.sms	31	69	57	54	56	143	28	27
040.sms	41	76	69	63	64	153	34	33
050.sms	51	85	75	67	68	161	40	39
060.sms	61	95	87	77	77	173	48	43
070.sms	71	96	94	82	82	180	56	51
080.sms	81	108	104	93	92	190	61	59
090.sms	91	111	103	94	92	189	65	59
100.sms	102	130	122	109	109	208	79	77
110.sms	111	125	119	112	110	205	79	75
120.sms	121	129	125	115	113	211	84	79
130.sms	131	132	129	123	120	215	91	84
140.sms	141	145	136	127	126	222	96	87
150.sms	151	139	140	131	128	226	100	94
160.sms	161	159	150	141	139	236	109	100
170.sms	171	162	156	149	145	242	124	121
180.sms	181	168	164	151	150	250	129	113
190.sms	191	178	167	158	154	253	125	117
200.sms	201	184	175	166	164	261	135	129
210.sms	211	193	180	166	163	266	143	129
220.sms	221	194	182	173	170	268	148	139
230.sms	231	201	188	175	171	274	150	142
240.sms	241	218	202	188	184	288	168	155
250.sms	251	219	209	189	185	295	176	161
260.sms	261	225	210	193	193	296	185	168

Konfigurace programu tester viz výše. Komprimované soubory GardenOfEden.txt rozdělená po $200B$, $300B$, $500B$ a $700B$ a KonecSveta.txt rozdělený analogicky. Výsledky jsou sumou přes všechny komprimované soubory.

file	size	obchod	animal	PPMd	PPMonster
garden200	356000	254236	229176	270028	273625
garden300	355800	249015	223987	235656	239072
garden500	356000	245083	220045	203535	206693
garden700	355600	243080	218086	186758	189887
konec200	70600	47933	56067	61214	61012
konec300	70500	46863	54981	54712	54751
konec500	70500	46052	54172	48440	48777
konec700	70000	45375	53441	44957	45436