FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Kompresní metoda Zstandard
Student:	Ondřej Vladyka
Vedoucí:	Ing. Jan Baier
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Seznamte se s metodami komprese z rodiny LZ algoritmů. Prozkoumejte metodu Zstandard [1] a vysvětlete její princip. Vytvořte efektivní implementaci této metody v rámci knihovny ExCom [2], otestujte výkon této metody na standardních testovacích datech [3] a porovnejte s ostatními metodami uvnitř knihovny.

Seznam odborné literatury

[1] http://facebook.github.io/zstd/
[2] http://www.stringology.org/projects/ExCom/
[3] http://www.stringology.org/projects/PragueCorpus/

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Kompresní metoda Zstandard

Ondřej Vladyka

Katedra teoretické informatiky
Vedoucí práce: Jan Baier

13. května 2018

Poděkování

Chci poděkovat svému vedoucímu, Ing. Janu Baierovi, za jeho rady a rychlé odpovědi na mé emaily. Děkuji také své rodině za nepřetržitou emocionální a monetární podporu během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2018 .....

Odkaz na tuto práci

Vladyka, Ondřej. *Kompresní metoda Zstandard*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

V práci se zaměřuji na nový algoritmus pro bezeztrátovou kompresi dat zvaný Zstandard. Teoretická část se zabývá úvodem do kompresních algoritmů a analýzou algoritmu Zstandard. V praktické části je algoritmus implementován v jazyce C++ do knihovny ExCom a otestován na standardních datech Pražského Korpusu. Výsledek je porovnán s ostatními kompresními metodami knihovny ExCom.

Klíčová slova Porovnání kompresních algoritmů, Zstandard, knihovna ExCom, Pražský korpus, zstd, kompresní algoritmy, bezeztrátová komprese dat

Abstract

This thesis is focused on the new lossless compression algorithm called Zstandard. The theoretical part of this thesis serves as introduction to data compression and analyzes the Zstandard algorithm. Zstandard is then implemented in C++ into the ExCom library and tested on the Prague Corpus dataset. The measurement is compared to the other compression methods imlpemented in ExCom.

Keywords Compression algorithm comparison, Zstandard, ExCom library, Prague corpus, zstd, compression algorithms, lossless data compression

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Bez kompresních algoritmů bychom dnes už jen stěží dokázali používat počítače. Ať už si stahujeme z internetu naši oblíbenou distribuci Linuxu, posloucháme hudbu, díváme se na filmy nebo si prohlížíme fotografie z dovolené, je dost pravděpodobné, že nám při tom skrytě pomáhají kompresní algoritmy.

Některé z nich nám nabízejí vysokou efektivitu komprese za cenu ztráty části původních informací (tzv. ztrátová komprese), jiné nám umožňují po kompresi kompletně zrekonstruovat původní data (tzv. bezeztrátová komprese).

V této bakalářské práci se zabývám novou metodou pro bezeztrátovou kompresi zvanou Zstandard, kterou vyvinul Yann Collet v roce 2016 ve firmě Facebook. Jeho záměrem bylo vytvořit algoritmus s velmi rychlou kompresí i dekompresí, a to i při zachování vysoké efektivity.

Tuto metodu implementuji do knihovny ExCom. Jedná se o knihovnu ČVUT určenou pro testování a porovnávání kompresních algoritmů vyvinutou Filipem Šimkem v diplomové práci *Data compression library* [2]. Knihovna už obsahuje několik kompresních algoritmů a její modulární design umožňuje snadné rozšíření o nové. Zstandard implementovaný v knihovně ExCom dále testuji na standardních datech Pražského korpusu a porovnávám s ostatními metodami.

Výsledek mého testování bude prospěšný všem jednotlivcům i velkým firmám při hledání vhodného kompresního algoritmu pro jejich potřeby. Rozšířením ExCom knihovny také pomůžu ČVUT v Praze a jeho studentům, kteří budou ExCom v budoucnu vyvíjet.

Práce je rozdělena na pět kapitol. První kapitola slouží jako úvod do kompresních algoritmů, obsahuje jejich rozdělení a základní definice. Druhá kapitola se věnuje popisu základních metod komprese z rodiny LZ algoritmů. Analýza metody Zstandard se nachází v kapitole třetí. Čtvrtá kapitola popisuje implementaci Zstandard do knihovny ExCom. Pátá kapitola se zabývá jejím testováním a následným porovnáním s ostatními metodami. Na závěr je shrnutí výsledků testování a rozepsání možností pro budoucí vylepšení.

Kapitola 1

Komprese dat

Tato kapitola slouží jako úvod do komprese dat, popisuje základní pojmy a definice a obsahuje rozdělení kompresních metod. Pro stanovení definic jsem čerpal z [3].

1.1 Základní definice

Definice 1.1. (Abeceda)
Abeceda je konečná množina symbolů (například písmena, čísla, znaky, ...).

Definice 1.2. (Zdrojové slovo)
Zdrojové slovo je symbol abecedy nebo řetězec symbolů abecedy.

Definice 1.3. (Kódové slovo)
Kódové slovo je řetězec bitů.

Definice 1.4. (Kód)
Kód K je uspořádaná trojice $K = (S, C, f)$, kde:

- S je konečná množina zdrojových slov,
- C je konečná množina kódových slov,
- f je injektivní zobrazení $S \mapsto C^+$.

Platí tedy $\forall a_1, a_2 \in S, a_1 \neq a_2 \implies f(a_1) \neq f(a_2)$. Tato vlastnost zobrazení f je nutnou podmínkou pro dekódování kódu K.

Definice 1.5. (Jednoznačné dekódování)
Řetězec kódových slov $y \in C^+$ je *jednoznačně dekódovatelný*, jestliže existuje nejvýše jeden řetězec zdrojových slov $x \in S^+$ takový, že $f(x) = y$.

Definice 1.6. (Prefixový kód)
Kód K je *prefixový kód*, pokud žádné kódové slovo $u \in C$ není prefixem žádného jiného kódového slova $v \in C, u \neq v$.

1.2 Redundance a entropie

V teorii informace [4] *Entropie* vyjadřuje množství informací obsažených ve zprávě (v tomto případě ve vstupních datech).

Definice 1.7. (Entropie)
Mějme kód $K = (S, C, f)$, kde S je konečná množina zdrojových slov $S = \{x_1, x_2, x_3, \ldots, x_n\}$ s pravděpodobnostmi $P = \{p_1, p_2, p_3, \ldots, p_n\}$. *Průměrná entropie* zdrojových slov z množiny S je:

$$H_{\text{avg}}(S) = -\sum_{i=1}^{n} p_i \log_2 p_i \text{ bitů.}$$

Průměrná entropie udává minimální počet bitů, který musíme v průměru spotřebovat pro zakódování zdrojových slov z S.

Definice 1.8. (Délka)
Průměrná délka kódového slova $c \in C$ kódu K je:

$$L_{\text{avg}}(K) = -\sum_{i=1}^{n} d_i p_i \text{ bitů.}$$

Kde d_i je bitová velikost kódového slova $c_i = f(x_i)$.

Definice 1.9. (Redundance)
Průměrná redundance kódu K je:

$$R_{\text{avg}}(K) = L_{\text{avg}} - H_{\text{avg}} \text{ bitů.}$$

Průměrná redundance kódu udává průměrný počet bitů, které jsou ve zprávě zakódované přebytečně (mohly by se odstranit a zpráva by obsahovala stejné množství informace).

1.3 Komprese a dekomprese

Definice 1.10. (Komprese)
Komprese je proces zakódování vstupních dat pomocí určitého algoritmu na data výstupní s cílem zmenšit jejich velikost.

Definice 1.11. (Dekomprese)
Dekomprese je inverzní proces ke kompresi, při kterém se vstupní data dekódují na data originální (při bezeztrátové kompresi) nebo jim podobná (při ztrátové kompresi).

Kompresní algoritmy jsou schopné redukovat velikost vstupu pomocí eliminace redundance a maximalizování entropie v datech.

1.4 Výkon komprese

Existují tři hlavní měřítka výkonu kompresních algoritmů:

- kompresní poměr,
- rychlost komprese,
- rychlost dekomprese.

Je nemožné je optimalizovat všechny, tvůrci algoritmů tedy musí mezi nimi při návrhu často volit kompromis (zejména mezi rychlostí komprese a kompresním poměrem).

1.4.1 Kompresní poměr

Efektivita komprese, nebo-li míra, do jaké se podařilo zredukovat vstupní data, se nejčastěji měří pomocí *kompresního poměru*. Může se také měřit *kompresním faktorem*. Ani jedna z těchto metrik teoreticky nezávisí na přístroji, na kterém je komprimace prováděna (v praxi může vadit například velikost paměti, se kterou algoritmus pracuje).

Definice 1.12. (Kompresní poměr)
Kompresní poměr je podíl velikosti zkomprimovaných dat k velikosti originálních dat.

$$\text{Kompresní poměr} = \frac{\text{Velikost zkomprimovaných dat}}{\text{Velikost originálních dat}}$$

Jestliže měříme velikosti zkomprimovaných i originálních dat v bitech, kompresní poměr vychází v jednotkách *bpb* (bit per bit). Pokud je poměr roven například $\frac{1}{2}$, znamená to, že se velikost originálních dat zmenšila o polovinu bitů. Při poměru větším než 1 velikost originálních dat při kompresi vzrostla (což se může stát u některých jednodušších kompresních algoritmů).

Definice 1.13. (Kompresní faktor)
Kompresní faktor je inverzní hodnota kompresního poměru.

$$\text{Kompresní faktor} = \frac{\text{Velikost originálních dat}}{\text{Velikost zkomprimovaných dat}}$$

1.4.2 Rychlost komprese

Rychlost komprese (resp. dekomprese) udává časovou dobu, kterou kompresní algoritmus strávil kódováním (resp. dekódováním) vstupních dat na data výstupní. Porovnávání rychlostí různých kompresních algoritmů se musí provádět na stejném přístroji, protože je rychlost závislá na jeho výkonu.

1.5 Druhy kompresních algoritmů

1.5.1 Ztrátovost

Algoritmy je možné dělit podle ztrátovosti na:

- *Ztrátové* – Při kompresi se ztratí část informací ze vstupních dat, dekompresí je tudíž nelze kompletně zrekonstruovat. Používají se převážně pro multimediální kompresi, kde mírná ztráta informací nevadí nebo není rozpoznatelná.

- *Bezeztrátové* – Při dekompresi jsou původní data zcela obnovena. Používají se například při kompresi textových souborů nebo programů.

1.5.2 Symetričnost

Podle symetričnosti lze algoritmy dělit na:

- *Symetrické* – Dekomprese provádí podobné kroky jako kompresní algoritmus, pouze v opačném směru. Rychlosti komprese i dekomprese jsou tedy velice podobné.

- *Asymetrické* – Pro kompresi a dekompresi se používají jiné algoritmy a proto jsou jejich rychlosti různé. To je výhodné v případě, když je jeden proces využíván častěji než druhý (např. balíčkovací systém jednou zkomprimuje balíček s programy, který je poté dekomprimován větším počtem uživatelů).

1.5.3 Typy metod

Bezeztrátové algoritmy se dělí podle typu na:

- *Statistické* – Hlavní myšlenkou je vytvořit prefixový kód, který zakóduje nejčastěji se vyskytující symboly ve vstupních datech do nejkratšího kódového slova a nejméně časté symboly do nejdelšího. Jejich nevýhodou je nutnost znát frekvence všech symbolů předem, což většinou znamená projít vstupní data nadvakrát. Příklady statistických metod jsou Huffmanovo kódování, Shannon-Fanovo kódování nebo Aritmetické kódování.

- *Slovníkové* – Tyto metody používají *slovník*, datovou strukturu, ve které se při kompresi zdrojového slova vyhledává shoda. Pokud je shoda nalezena, slovo se zakóduje jako odkaz na shodu do slovníku. Většina slovníkových metod si vytvářejí slovník při kompresi z předešlých vstupních slov a pro kompresi využívají opakování slov ve zdrojových datech. Paří mezi ně například LZ algoritmy (LZ77, LZ78, LZSS, ...).

- *Kontextové* – Využívají faktu, že pravděpodobnost jednotlivých symbolů na vstupu se mění podle jejich kontextu (podle předchozích N symbolů). Jejich zástupci jsou například PPM, DCA nebo ACB.

Některé algoritmy můžou používat kombinace výše uvedených typů, například populární Deflate kombinuje principy LZ77 a Huffmanova kódování.

1.5.4 Model

- *Statický model*– Komprese či dekomprese používá statický model dat, který se na základě vstupu nemění. V praxi se to může být například četnost jednotlivých písmen v anglických textech. Model se v zkomprimovaných datech nepřenáší, algoritmus ho musí mít připravený před začátkem procesu. Díky tomu stačí pouze jeden průchod vstupními daty.

- *Semi-adaptivní model* – Komprese vyžaduje dva průchody. V prvním se vytvoří vlastní model dat, který je závislý na vstupních datech, poté jsou podle něj v druhém průchodu data zakódována. Dekomprese dopředu nezná model dat, proto je nutné jej přidat k zkomprimovanému výstupu.

- *Adaptivní model* – Kombinuje výhody statického a semi-adaptivního modelu. Jedním průchodem se při kompresi i dekompresi datový model postupně vytváří, používá a upravuje podle vstupních dat, není tedy třeba ho přenášet spolu se zkomprimovanými daty.

Kapitola 2

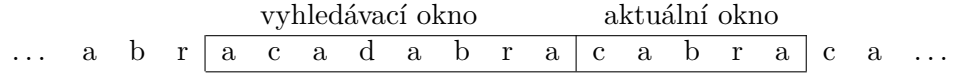
LZ77

2.1 Úvod

LZ77 je bezeztrátový asymetrický slovníkový adaptivní kompresní algoritmus. Poprvé jej popsali popsali Abraham Lempel a Jacob Ziv v roce 1977 v článku *A universal algorithm for sequential data compression* [5]. Vycházeli z předpokladu, že se schodné řetězce zdrojových slov nacházejí ve vstupních datech relativně blízko u sebe. Na výstupu jsou pak duplikátní řetězce nahrazeny za ukazatele na jejich první výskyt.

Algoritmus si vytváří za chodu vlastní slovník, používá k tomu princip *klouzavého okénka* (angl. *sliding window*), paměťového bufferu s konstantní velikostí rozděleného na dvě části – *aktuální okno* (angl. *look-ahead buffer*) a *vyhledávací okno* (angl. *search buffer*). Slovník je tvořen všemi řetězci, které začínají ve vyhledávacím okně (můžou přesahovat i do aktuálního okna).

Aktuální okno bývá relativně malé (desítky až stovky bajtů) a obsahuje začátek nezpracované části vstupu. Vyhledávací okno by mělo být mnohem větší (tisíce až deseti tisíce bajtů) a obsahuje konec již zpracované části vstupu. Algoritmus pracuje pouze s daty obsaženými v klouzavém okénku, jeho paměťová náročnost díky tomu neroste s velikostí vstupních dat.



Obrázek 2.1: Ukázka klouzavého okénka

2.2 Komprese

Před začátkem komprese se inicializuje klouzavé okénko – aktuální okno je prázdné a do vyhledávacího okna se zkopíruje začátek vstupních dat.

Poté se algoritmus pokusí nalézt první symbol z aktuálního okna ve vyhledávacím okně (prohledává ho zprava doleva). Pokud ho najde, začne vyhledávat první dva symboly z aktuálního okna. To se opakuje, dokud algoritmus nenajde nejdelší řetězec začínající ve vyhledávacím okně, který je schodný s prefixem (předponou) slova v aktuálním okně. Řetězec se zakóduje do uspořádané trojice (i, j, X), kde:

- i je vzdálenost začátku řetězce ve vyhledávacím okně od začátku aktuálního okna,
- j je délka řetězce (počet jeho symbolů),
- X je symbol následující za řetězcem v aktuálním okně.

Pokud se žádný řetězec nenajde, na výstup se zapíše trojice $(0, 0, X)$, kde X je první symbol v aktuálním okně. Klouzavé okénko poté načte dalších $j+1$ symbolů ze vstupu (celé se posune o $j + 1$ symbolů „doprava“).

Algoritmus skončí, pokud je aktuální okno prázdné (nezbývají žádná data k zakódování).

2.2.1 Ukázka komprese

Ukázku provedu na vstupu $T =$ „banana␣bandana␣ano“ s velikostí vyhledávacího okna $S = 7$ a aktuálního okna $L = 4$.

Krok		Vyhledávací okno							Aktuální okno					Výstup
1.									b	a	n	a	...	(0, 0, b)
2.								b	a	n	a	n	...	(0, 0, a)
3.							b	a	n	a	n	a	...	(0, 0, n)
4.						b	a	n	a	n	a	␣	...	(2, 3, ␣)
5.		b	a	n	a	n	a	␣	b	a	n	d	...	(7, 3, d)
6.	...	n	a	␣	b	a	n	d	a	n	a	␣	...	(3, 2, a)
7.	...	b	a	n	d	a	n	a	␣	a	n	o		(0, 0, ␣)
8.	...	a	n	d	a	n	a	␣	a	n	o			(4, 2, o)
9.	...	a	n	a	␣	a	n	o						

Obrázek 2.2: Ukázka LZ77 komprese

V zakódované trojici slouží i jako ukazatel do vyhledávacího okna, jeho hodnoty tedy můžou být $\{1, 2, \ldots, S-1, S\}$ a jeho velikost musí být minimálně $\lceil \log_2 S \rceil$ bitů.

Jeden symbol z aktuálního okna se vždy musí zakódovat do trojice jako X, proto j může nabývat pouze hodnot $\{1, 2, \ldots, L-2, L-1\}$. Minimální velikost j je tudíž $\lceil \log_2 L - 1 \rceil$ bitů.

Pokud předpokládáme velikost jednoho symbolu 8 bitů, velikost vstupu s 18 symboly je $18 * 8 = 144$ bitů. Velikost jedné uspořádané trojice je $\lceil \log_2 7 \rceil + \lceil \log_2 3 \rceil + 8 = 3 + 2 + 8 = 13$ bitů. Při kompresi vzniklo 8 trojic, celý zkomprimovaný výstup tedy tvoří $8 * 13 = 104$ bitů. Na ukázce dosáhl algoritmus LZ77 kompresního poměru $\frac{104}{144} = 0,7\overline{2}$.

2.3 Dekomprese

LZ77 je asymetrický algoritmus, jeho dekomprese je mnohem rychlejší. Nemusí se v ní totiž provádět nejsložitější část kompresního algoritmu – hledání nejdelšího prefixu aktuálního okna v okně vyhledávacím.

Musí se použít stejná velikost klouzavého okénka (i obou jeho částí) jako v kompresi, oproti ní však neslouží pro čtení vstupu, ale pro zápis dekódovaného výstupu. Na začátku je okénko prázdné.

V každém kroku dekomprese se pak načte uspořádaná trojice (i, j, X) ze vstupu, ve vyhledávacím okně se najde začátek řetězce podle hodnoty i, dalších j znaků se zkopíruje do aktuálního okna a za ně se připíše symbol X. Obsah aktuálního okna se zapíše na výstup a klouzavé okénko se poté posune o $j + 1$ symbolů (celý obsah aktuálního okna se posune do vyhledávacího okna).

Algoritmus se opakuje, dokud jsou na vstupu nějaké uspořádané trojice.

2.3.1 Ukázka dekomprese

Ukázka dekomprese je provedena na výstupu z ukázky komprese. Vstup je $T =$ „$(0, 0, \mathrm{b}), (0, 0, \mathrm{a}), (0, 0, \mathrm{n}), (2, 3, \sqcup), (7, 3, \mathrm{d}), (3, 2, \mathrm{a}), (0, 0, \sqcup), (4, 2, \mathrm{o})$“, velikost vyhledávacího okna je $S = 7$ a velikost aktuálního okna je $L = 4$.

Krok	Vstup	Vyhledávací okno							Aktuální okno				Výstup
1.	$(0, 0, \mathrm{b})$								b				b
2.	$(0, 0, \mathrm{a})$							b	a				a
3.	$(0, 0, \mathrm{n})$						b	a	n				n
4.	$(2, 3, \sqcup)$					b	a	n	a	n	a	$\sqcup$	ana$\sqcup$
5.	$(7, 3, \mathrm{d})$	b	a	n	a	n	a	$\sqcup$	b	a	n	d	band
6.	$(3, 2, \mathrm{a})$	n	a	$\sqcup$	b	a	n	d	a	n	a		ana
7.	$(0, 0, \sqcup)$	b	a	n	d	a	n	a	$\sqcup$				$\sqcup$
8.	$(4, 2, \mathrm{o})$	a	n	d	a	n	a	$\sqcup$	a	n	o		ano

Obrázek 2.3: Ukázka LZ77 dekomprese

Je třeba dát pozor na situace, kdy je $j > i$ (jak je tomu ve 4. kroku ukázky), řetězec ke zkopírování přesahuje z vyhledávacího okna do aktuálního. Algoritmu to nevadí, jen je třeba kopírovat řetězec po jednotlivých symbolech a ne celý najednou.

2.4 Další metody

2.4.1 LZ78

Zabýval jsem se hlavně metodou LZ77, protože z ní vychází i metoda Zstandard. Další důležitou metodou z rodiny LZ algoritmů je ale i LZ78.

Jedná se o slovníkovou metodu vyvinutou autory LZ77 Abrahamem Lempelem a Jacobem Zivem [6]. Místo principu klouzavého okénka využívá rostoucí slovník reprezentovaný jako *trie* (prefixový strom). Oproti LZ77 nemá konstantní velikost a obsahuje všechny dosud zpracované řetězce. Umožňuje tím efektivnější kompresi (rozezná vzdálenější shody řetězců), ale je velmi náročný na paměť.

2.4.2 LZSS

Do rodiny LZ algoritmů spadá i velké množství dalších metod, které vycházejí z LZ77 nebo LZ78 a nějakým způsobem je upravují či vylepšují. Jednou takových metod navazujících na LZ77 je LZSS.

Místo uspořádané trojice (i, j, X) se na výstupu může objevit (s jednobitovým indikátorem) uspořádaná dvojice (i, j) nebo symbol X. Když se při kompresi řetězec R o velikosti p bitů zakóduje do dvojice (i, j) o velikosti q bitů a pokud $p < q$, na výstup se místo dvojice zkopírují symboly řetězce. Tím se zabraňuje *negativní kompresi* (kompresní poměr je větší než 1 – data se kompresí zvětšila). Dále jsou využité efektivnější datové struktury (*binární vyhledávací strom* a *kruhová fronta*) umožňující rychlejší vyhledávání a posouvání symbolů v klouzavém okénku.

2.4.3 Deflate

Deflate patří v současnosti mezi nejpoužívanější bezeztrátové kompresní algoritmy. Popis implementace byl jeho autorem Philipem Kratzem uveden licenčně jako *volné dílo* (angl. *public domain*), může se tedy volně šířit, upravovat a používat i pro komerční účely.

Princip algoritmu vychází z LZ77, podobně jako LZSS se při kompresi na výstupu může objevit uspořádaná dvojice (i, j) nebo symbol X. Výstup se propojí se statistickou metodou *Huffmanovo kódování* [7] pro zvýšení entropie a dosažení lepšího kompresního poměru.

Kapitola 3

Analýza Zstadard

3.1 Úvod

Zstandard je bezeztrátový asymetrický slovníkový adaptivní kompresní algoritmus vydaný Yannem Colletem v roce 2016. Vychází z metody LZ77, patří tedy do rodiny LZ kompresních algoritmů [8]. Jeho zdrojové soubory, implementované v jazyce C, jsou dostupné pod dvojitou licencí BSD / GNU GPLv2, tudíž mohou být volně používány a dále upravovány.

Na rozdíl od ostatních kompresních algoritmů zaměřujících se pouze na rychlost (např. LZ4) nebo na efektivitu (např. LZMA), Zstandard se zaměřuje na obojí. Jeho hlavním cílem je dosažení větší rychlosti komprese i dekomprese při zachování podobného kompresního poměru, jako má knihovna zlib (používající algoritmus Deflate) [9].

Podobně jako u jiných pokročilých LZ algoritmů má komprese u Zstandard dvě fáze. V první fázi (*modeling stage – modelovací fáze*) se opakující se řetězce symbolů ve vstupních datech zakódují do tokenů (uspořádaných n-tic) na principu LZ77, v druhé fázi (*entropy stage – entropická fáze*) se tokeny zakódují nejmenším počtem bitů na výstup. V entropické fázi je použita nová a efektivní metoda FSE (*Finite State Entropy*) nebo rychlá moderní implementace Huffmanova kódování umožnující paralelní dekódování [10]. Obě fáze probíhají při stejném průchodu vstupními daty.

Vedle normální komprese či dekomprese Zstandard nabízí i „slovníkový režim“, určený pro kompresi malých dat. Termín *slovník* zde nevyjadřuje klouzavé okénko, ale nějaký externí slovník vytvořený na datech podobných těm, které chceme komprimovat. Tento slovník je nutné algoritmu poskytnout při kompresi i při dekompresi. Drasticky se tím vylepší kompresní poměr na malých datech, což obvykle dělá LZ77 metodám problém (dat není dost na to, aby se v nich pořádně využilo častého opakování řetězců symbolů). Jeho velikou nevýhodou je nutnost předat si slovník mezi stranou, která komprimuje, a stranou, která dekomprimuje. Tento režim není určený pro běžné používání, ale spíše pro specifické případy (např. pro kompresi v databázích).

3.2 Formát .zst

Mimo implementace algoritmu Zstandard je v oficiálním repozitáři i definice formátu zkomprimovaných dat (přípona *.zst*), kterou bu měly všechny jiné implementace komprese či dekomprese dodržovat [1].

Hlavní jednotkou formátu .zst je *rámec* (angl. *frame*). Zkomprimovaná data jsou tvořena jedním nebo více rámci. Každý rámec je kompletně nezávislý na ostatních, při kompresi jich tedy může vznikat více najednou (např. ve více vláknech) a při dekompresi se můžou najednou dekódovat. V každém rámci jsou definované parametry, které určují jeho typ, velikost nebo jakým způsobem se má dekomprimovat.

Každý rámec je tvořen jedním nebo více *bloky*. Jednotlivé bloky v posloupnosti jsou na sobě závislé, pro dekompresi jednoho je tedy potřeba nejprve provést dekompresi všech jeho předchůdců (nemusí se však čekat na dokončení dekomprese jeho následníků).

Všechna metadata (například *Magické číslo*, *Hlavička rámce/bloku* nebo *Kontrolní součet*) jsou uloženy ve formátu little endian. Názvy nepovinných polí jsou v obrázcích v této kapitole ohraničené hranatými závorkami (např. [Kontrolní součet] v obrázku 3.1).

3.2.1 Rámec

Formát rozlišuje dva typy rámce:

- *Zstandard rámec* – obsahuje zkomprimovaná data (v blocích),
- *Přeskočitelný rámec* (angl. *Skippable frame*) – neobsahuje žádná data, používá se pro uchování metadat.

3.2.1.1 Zstandard rámec

Hlavní typ rámce. Obsahuje všechna zkomprimovaná data, která se při dekompresi načtou a dekódují.

Magické číslo	Hlavička rámce	Blok	[Další bloky]	[Kontrolní součet]
4 bajty	2–14 bajtů	n bajtů	...	0–4 bajty

Obrázek 3.1: Zstandard rámec [1]

Popis Zstandard rámce na obrázku 3.1:

- *Magické číslo* – Slouží jako identifikátor Zstandard rámce. V současné verzi v1.3.4 má hodnotu 0xFD2FB528.
- *Hlavička rámce* – Obsahuje parametry rámce, více popsané v sekci 3.2.2.

- *Blok* – Blok slouží pro uchování zkomprimovaných dat.
- *Další bloky* – Rámec může obsahovat jeden nebo více bloků.
- *Kontrolní součet* – Slouží pro kontrolu integrity dat. Ukázková implementace používá nejnižší 4 bajty z xxh64, rychlou hashovací funkci od Yanna Colleta.

3.2.1.2 Přeskočitelný rámec

Přeskočitelný rámec je možné použít například pro uložení velikosti jednotlivých Zstandard rámců, které tuto informaci neobsahují (obsahují pouze velikost jejich bloků). To se hodí zejména při pro vypočítání a nalezení začátku nějakého konkrétního rámce, kterého chceme dekomprimovat (jinak bychom museli blok po bloku projít všechny rámce předchozí).

Magické číslo	Velikost rámce	Metadata
4 bajty	4 bajty	n bajtů

Obrázek 3.2: Přeskočitelný rámec [1]

Popis přeskočitelného rámce na obrázku 3.2:

- *Magické číslo* – Slouží jako identifikátor Přeskočitelného rámce. V současné verzi v1.3.4 má libovolnou hodnotu v rozmezí od 0x184D2A50 do 0x184D2A5F.
- *Velikost rámce* – Udává velikost následujících metadat (v bajtech). Pole je typu *celé číslo bez znaménka* (angl. *unsigned integer*), což znamená, že velikost metadat může být maximálně $2^{32} - 1$ bajtů.
- *Metadata* – Zde může být uloženo cokoliv. Slouží pro předání parametrů v nějaké specifické implementaci komprese/dekomprese.

3.2.2 Hlavička rámce

Hlavička rámce (angl. *Frame Header*) má variabilní velikost 2–14 bajtů.

Deskriptor hlavičky	[Deskriptor okénka]	[ID slovníku]	[Velikost obsahu rámce]
1 bajt	0–1 bajt	0–4 bajty	0–8 bajtů

Obrázek 3.3: Hlavička rámce [1]

Popis hlavičky rámce na obrázku 3.3:

- *Deskriptor hlavičky* – Popisuje, které další pole hlavička obsahuje. Podle jeho hodnoty jde spočítat velikost hlavičky rámce. Více informací v sekci 3.2.2.1
- *Deskriptor okénka* – Udává minimální velikost bufferu nutnou pro dekompresi rámce (pro klouzavé okénko).
- *ID slovníku* – Obsahuje ID slovníku nutného pro dekompresi. Pokud pole chybí, dekodér se nějak musí umět rozhodnout, který slovník použít. Používá se pouze při „slovníkovém režimu“. Hodnota je 0, pokud žádný slovník není třeba použít.
- *Velikost obsahu rámce* – Velikost originálních (nezkomprimovaných) dat.

3.2.2.1 Deskriptor Hlavičky

Deskriptor hlavičky obsahuje příznaky (angl. flag) pro různé parametry rámce. Pokud je nastaven příznak pro *jeden segment*, *deskriptor okénka* v hlavičce chybí a při dekompresi se musí alokovat klouzavé okénko o velikosti dané hodnotou pole *velikost obsahu rámce*. To může být maximálně $2^{64} - 1$ bajtů, tedy 16 exabajtů.

Číslo bitu	význam
7–6	Příznak pro *velikost obsahu rámce*
5	Příznak pro *jeden segment*
4	nepoužívaný bit
3	rezervovaný bit
2	Příznak pro *kontrolní součet*
1–0	Příznak pro *ID slovníku*

Tabulka 3.1: Deskriptor Hlavičky [1]

Číslo bitu určuje pořadí bitů v bajtu, například 0 značí nejméně významný bit.

Popis deskriptoru hlavičky z tabulky 3.1:

- Příznak pro *velikost obsahu rámce* – Dva bity, které značí přítomnost a velikost pole *velikost obsahu rámce* v hlavičce podle tabulky 3.2. Pokud je hodnota příznaku 0, velikost pole je dána podle příznaku pro *jeden segment* – je-li příznak 1, velikost je 1, jinak je velikost 0.
- Příznak pro *jeden segment* – Určuje přítomnost pole *velikost obsahu rámce* (pokud je 1) nebo pole *deskriptor okénka* (pokud je 0) v hlavičce rámce.
- *nepoužívaný bit* – Měl by být 0, při dekompresi je ignorován.

- *rezervovaný bit* – Musí být 0, je rezervovaný pro budoucí funkcionalitu.
- Příznak pro *kontrolní součet* – Určuje přítomnost kontrolního součtu na konci rámce.
- Příznak pro *ID slovníku* – Dva bity, které značí přítomnost a velikost pole *ID slovníku* podle tabulky 3.3

Hodnota příznaku	0	1	2	3
Velikost pole *velikost obsahu rámce* v bajtech	0 nebo 1	2	4	8

Tabulka 3.2: Hodnoty příznaku pro *velikost obsahu rámce* [1]

Hodnota příznaku	0	1	2	3
Velikost pole *ID slovníku* v bajtech	0	1	2	4

Tabulka 3.3: Hodnoty příznaku pro *ID slovníku* [1]

3.2.3 Blok

Každý Zstandard rámec je tvořen alespoň jedním blokem, horní limit na počet bloků v rámci není. První 3 bajty bloku tvoří jeho hlavičku rozdělenou na tři parametry. Po ní následuje obsah bloku.

Poslední blok	Typ bloku	Velikost bloku	Obsah bloku
1 bit	2 bity	21 bitů	n bajtů

Obrázek 3.4: Blok [1]

Popis bloku na obrázku 3.4:

- *Poslední blok* – Signalizuje, zda-li se jedná o poslední blok v rámci.
- *Typ bloku* – 2 bity udávající typ bloku.
- *Velikost bloku* – Značí velikost *obsahu bloku* v bajtech.
- *Obsah bloku* – Obsahuje data podle typu bloku.

Typy bloků jsou následující:

- *Nezkomprimovaný blok* – *Obsah bloku* jsou nezkomprimovaná data. Používá se pro zamezení *negativní komprese* (kompresí se zvětší velikost původních dat).

- *RLE blok* – *Obsah bloku* je jeden bajt. Při dekompresi se musí tento bajt zopakovat $n$krát, kde n je *velikost bloku*.
- *Zkomprimovaný blok* – Více v sekci 3.2.4.
- *Rezervované* – Hodnota je rezervována pro budoucí definici nového typu bloku.

3.2.4 Zkomprimovaný Blok

Veškerá zkomprimovaná data Zstandard formátu se uchovávají ve zkomprimovaných blocích. Pro dekompresi zkomprimovaného bloku je třeba mít n posledních bajtů z předchozích bloků (nejen z těch zkomprimovaných) uložené v paměťovém bufferu klouzavého okénka, kde n je velikost bufferu definovaná v hlavičce rámce (pole *deskriptor okénka* nebo *velikost obsahu rámce*). Dále je potřeba mít FSE dekódovací tabulku a Huffmanův strom používaný předchozími bloky v rámci.

Zkomprimovaný blok se skládá ze svou částí:

- *Sekce literálů*,
- *Sekce sekvencí*.

Dekódováním a zkombinováním obou částí dostaneme originální data.

3.2.5 Sekce literálů

Začátek bloku tvoří sekce literálů. Literály jsou symboly, které nebyly při kompresi nalezeny v klouzavém okénku (nelze se na ně odkázat do minulosti). Mohou být uloženy v nezkomprimované podobě, nebo zakódované pomocí Huffmanova kódování, konkrétně metodou Huff0 vyvinutou autorem Zstandardu, Yannem Colletem. Tato metoda umožňuje paralelní dekódování.

Hlavička sekce literálů	[Popis Huffmanova stromu]	Proud 1	[Proudy 2–4]

Obrázek 3.5: Sekce literálů [1]

Popis sekce literálů z obrázku 3.5:

- *Hlavička sekce literálů* – Obsahuje parametry této sekce literálů. Více informací v sekci 3.2.5.1.
- *Popis Huffmanova stromu* – Obsahuje informace o Huffmanově stromu, který byl použitý pro zakódování této sekce literálů. Pokud jsou literály uložené v nezkomprimované podobě, toto pole chybí. Pokud jsou literály zkomprimované a toto pole chybí, pro dekompresi je třeba použít Huffmanův strom z posledního bloku, který popis stromu obsahoval.

- *Proudy 1–4* – Pokud jsou literály nezkomprimované, v proudu 1 jsou uchovaná jejich data. Pokud jsou zkomprimované, data jsou uchovaná v proudech 1–4 podle parametrů v hlavičce sekce. Více proudů umožňuje paralelní dekódování.

Dekódované literály uložené v paměťovém bufferu jsou dále využívané sekvencemi.

3.2.5.1 Hlavička sekce literálů

V hlavičce sekce literálů jsou uložené všechny parametry. Pokud má sekce více proudů, hodnoty *původní velikost* a *zkomprimovaná velikost* udávají velikosti všech proudů dohromady.

Typ sekce literálů	Formát sekce	Původní velikost	[Zkomprimovaná velikost]
2 bity	1–2 bity	5–20 bitů	0–18 bitů

Obrázek 3.6: Hlavička sekce literálů [1]

Popis hlavičky sekce literálů na obrázku 3.6:

- *Typ sekce literálů* – Značí, zda-li je sekce nezkomprimovaná, zkomprimovaná s *popisem Huffmanova stromu* nebo zkomprimovaná bez *popisu Huffmanova stromu*.
- *Formát sekce* – Určuje velikosti polí *původní velikost* a *zkomprimovaná velikost* a kolik obsahuje sekce proudů.
- *Původní velikost* – Hodnota udává velikost nezkomprimovaných literálů.
- *Zkomprimovaná velikost* – Označuje velikost zkomprimovaných literálů s velikostí *popisu Huffmanova stromu*, pokud je přítomen. Pole je přítomné pouze, pokud jsou literály zkomprimované.

3.2.6 Sekce sekvencí

Za sekcí literálů v bloku následuje sekce sekvencí. Její velikost není nikde přímo definovaná, jde ale odvodit rozdílem velikosti celého bloku a velikosti sekce literálů.

Sekvence fungují na principu klouzavého okénka z algoritmu LZ77. Každá sekvence obsahuje tři hodnoty:

- *Délka literálů* – Kolik bajtů se má zkopírovat z bufferu literálů.
- *Vzdálenost shody* (angl. *offset*)– Ukazatel na začátek kopírování v klouzavém okénku.

- *Délka shody* – Kolik bajtů se má zkopírovat z klouzavého okénka.

Po načtení a dekódování této trojice se do aktuálního okna zkopíruje *délka literálů* bajtů z bufferu literálů (z předchozí sekce). Poté se do aktuálního okna zkopíruje *délka shody* bajtů ve vyhledávacím okně, začínající bajtem vzdáleným *vzdálenost shody* bajtů od začátku aktuálního okna. Aktuální okno se vypíše na výstup, klouzavé okénko se posune a načte se další sekvence. Po provedení poslední sekvence se na výstup (do aktuálního okna) zkopírují všechny zbývající literály z bufferu a algoritmus pokračuje dalším blokem.

Podobně jako u algoritmu Deflate, i zde jsou hodnoty sekvence dále zakódovány. Místo Huffmanova kódování se ale používá metoda FSE (*Finite State Entropy*) vyvinutá Yannem Colletem, který se inspiroval článkem *Asymmetric numeral systems* [11]. FSE je o něco pomalejší než Huffmanovo kódování, dosahuje však mnohem větší efektivity, zejména pro často se opakující symboly. Všechny tři typy hodnot (*Délka literálů*, *Vzdálenost shody* a *Délka shody*) jsou zakódované vlastním FSE (nesdílí stejný kód).

Hlavička sekce sekvencí	[FSE tabulky pro dekódování hodnot]	Proud

Obrázek 3.7: Sekce sekvencí [1]

Popis sekce sekvencí na obrázku 3.7:

- *Hlavička sekce sekvencí* – Udává počet sekvencí v sekci. Dále určuje, podle jaké FSE tabulky se mají jednotlivé hodnoty dekódovat. Možnosti jsou: použít předdefinovanou tabulku, použít jednu z tabulek z následujícího pole nebo použít tabulku z minulého zkomprimovaného bloku.
- *FSE tabulky pro dekódování* – Obsahuje tři FSE tabulky pro dekódování *délek literálů*, *vzdáleností shod* nebo *délek shod* (tabulky jsou v tomto pořadí).
- *Proud* – Obsahuje zakódované sekvence. Při dekódování se musí číst odzadu.

Kapitola 4

Implementace Zstandard do knihovny ExCom

V této kapitole popisuji praktickou část bakalářské práce, tedy implementaci algoritmu Zstandard do knihovny ExCom.

4.1 Knihovna ExCom

ExCom (*Extensible Compression Library*) je knihovna určená pro porovnávání kompresních algoritmů. Obsahuje již implementované různé statické, slovníkové i kontextové metody a její modulární design umožňuje snadné rozšíření o metody nové. Je v ní také implementované prostředí pro testování rychlostí komprese i dekomprese.

Knihovna byla vyvinuta Filipem Šimkem v rámci jeho diplomové práce *Data compression library* [2]. Zdrojové soubory implementace v jazyce C++ jsou na oficiálních stránkách [12] dostupné ke stažení pod licencí GNU LGPL. Na stránkách se rovněž vyskytuje podrobný návod pro použití knihovny i pro přidávání nových metod.

4.2 Příprava

Se zdrojovými soubory jsou v balíčku s knihovnou obsaženy i konfigurační soubory `configure.ac` a `Makefile.am` umožnující snadné sestavení (angl. build) a kompilaci pomocí GNU Autotools. S aktuální verzí automake 1.15 však nejsou tyto soubory plně funkční, respektive soubor `configure.ac` se musí mírně upravit. Pro automake od verze 1.12 musí obsahovat makro `AM_PROG_AR`, pro verze starší však makro obsahovat nesmí, protože ho starší verze neznají. To lze vyřešit přidáním makra `m4_ifdef([AM_PROG_AR], [AM_PROG_AR])` na začátek souboru `configure.ac` [13].

Po úspěšné kompilaci knihovny jsem si dále všiml, že implementace metod LZ77 a LZSS nefungují (při spuštění sice zkompilují určený soubor, ale poté skončí chybou *double free or corruption*). Z toho důvodu jsem tyto metody v kapitole 5 nepoužíval.

4.3 Implementace Zstandard

Zdrojové soubory implementace algoritmu Zstandard si lze stáhnout z oficiálního repozitáře [14] pod dvojitou licencí BSD / GNU GPLv2. Repozitář obsahuje zdrojové soubory napsané v jazyce C, jsou však kompatibilní i s C++ a jejich implementování do knihovny ExCom tedy nebude problém.

Při přidávání nové metody do ExCom jsem postupoval podle návodu na stránkách knihovny ExCom [15]. Nejprve jsem vytvořil adresář `lib/method/zstandard` se soubory `zstandard.cpp` a `zstandard.hpp`, které definovaly novou třídu *CompZstandard*. Dále jsem tuto třídu přidal do zbytku knihovny podle výše zmíněného návodu (přidání `Automake.am`, úprava `configure.ac`, atd...).

Následovně bylo třeba v souboru `zstandard.cpp` implementovat metody pro kompresi a dekompresi.

Pokračoval jsem tedy zkopírováním adresářů `lib/common`, `lib/compress` a `lib/decompress` z repozitáře Zstandard do adresáře `lib/method/zstandard`. Tyto adresáře obsahují referenční implementaci komprese a dekomprese algoritmu Zstandard, neobsahují však například implementaci „slovníkového režimu“ zmíněného v kapitole 3. Z repozitáře jsem zkopíroval i soubor `lib/zstd.h`, ve kterém je rozhraní (angl. API) implementace.

Rozhraní je rozděleno na dvě části: stabilní a nestabilní (experimentální). Ze zřejmých důvodů jsem se zaměřil pouze na část stabilní. Ta obsahuje dva základní režimy komprese:

- *Bloková komprese*,
- *Proudová komprese*.

4.3.1 Bloková komprese

Pro blokovou kompresi a dekompresi se používají následující funkce:

```
size_t ZSTD_compress(void* dst, size_t dstCapacity,
        const void* src, size_t srcSize,
        int compressionLevel);

size_t ZSTD_decompress(void* dst, size_t dstCapacity,
        const void* src, size_t compressedSize);
```

Při kompresi se data na ukazateli *src* o délce *compressedSize* zkomprimují do jednoho Zstandard rámce. Rámec se v paměti uloží na ukazatel *dst*. Dekomprese funguje analogicky stejně.

Pokud bych chtěl použít blokové funkce, musel bych znát velikost dat určených k dekompresi předem. To ale v ExCom kvůli návrhu I/O modulu nelze. Mohl bych data ze vstupu načítat podle předem dané velikosti, vytvářet z nich zkomprimované rámce a ty ukládat na výstup, nejednalo by se ale o příliš efektivní řešení. Rozdělením vstupních dat na menší bloky bych totiž omezoval kontext, se kterým může Zstandard pracovat – LZ77 i Entropické kódování spoléhá na různé opakující se vzorce, které se v menších datech vyskytují méně.

4.3.2 Proudová komprese

Pro implementaci Zstandard v knihovně ExCom jsem zvolil proudovou kompresi, protože je efektivnější než bloková komprese. Používají se pro ní následující funkce:

```
ZSTD_CStream* ZSTD_createCStream(void);

size_t ZSTD_initCStream(ZSTD_CStream* zcs,
        int compressionLevel);

size_t ZSTD_compressStream(ZSTD_CStream* zcs,
        ZSTD_outBuffer* output, ZSTD_inBuffer* input);

size_t ZSTD_endStream(ZSTD_CStream* zcs,
        ZSTD_outBuffer* output);

size_t ZSTD_freeCStream(ZSTD_CStream* zcs);
```

Před kompresí se musí pomocí funkce `ZSTD_createCStream` vytvořit *zcs*, struktura představující proud. Dále se struktura před začátkem musí inicializovat pomocí funkce `ZSTD_initCStream`. Při inicializaci se nastaví i požadovaná úroveň komprese (angl. *compression level*, viz 5.3).

V průběhu komprese se do ní pomocí funkce `ZSTD_compressStream` postupně posílají části vstupu uložené v bufferu *input*, *zcs* z nich postupně vytváří zkomprimovaný rámec a zapisuje jej do bufferu *output*.

Po spotřebování všech vstupních dat se použije funkce `ZSTD_endStream`, která signalizuje *zcs* konec vstupu. Do bufferu *output* se uloží konec Zstandard rámce. Nakonec se proud *zcs* ukončí a uvolní svoje prostředky pomocí funkce `ZSTD_freeCStream`.

Celý vstup předem neznámé velikosti se zkomprimuje do jednoho Zstandard rámce. Dekomprese je analogicky podobná.

Kapitola 5

Testování Zstandard

5.1 Pražský korpus

Aby mělo testování kompresních algoritmů vypovídající hodnotu, jako testovací data se používají různé *korpusy*. Korpus je množina souborů zahrnující vše, s čím by se měl běžný uživatel setkat při používání komprese (textové nebo binární soubory, obrázky, fotografie, PDF soubory, atd ...).

V dnešní době nejčastěji používaný korpus pro testování kompresních algoritmů se nazývá *Slezský korpus* (angl. *Silesia Corpus*) z roku 2003 [16]. Korpus má však několik nevýhod – je relativně velký (celkem nad 200 MB) a pro výběr jeho obsahu nebyla použita žádná sofistikovaná metoda.

Proto místo něj použiji nový Pražský korpus vyvinutý na ČVUT v roce 2010, který má menší velikost a modernější obsah souborů [17]. Celý korpus, obsahující třicet souborů z osmi různých typů dat, se nachází na přiloženém DVD.

5.2 Testovací prostředí

Testování jsem prováděl na notebooku s procesorem Intel® Core™ i5-7300HQ 2,50 GHz a 7,66 GB operační paměti. Na notebooku byl nainstalovaný operační systém Ubuntu 16.04 LTS s linuxovým kernelem verze 4.10.0-42-generic. Pro sestavení a zkompilování knihovny ExCom jsem použil programy autoconf verze 2.69, automake verze 1.15 a gcc verze 5.4.0.

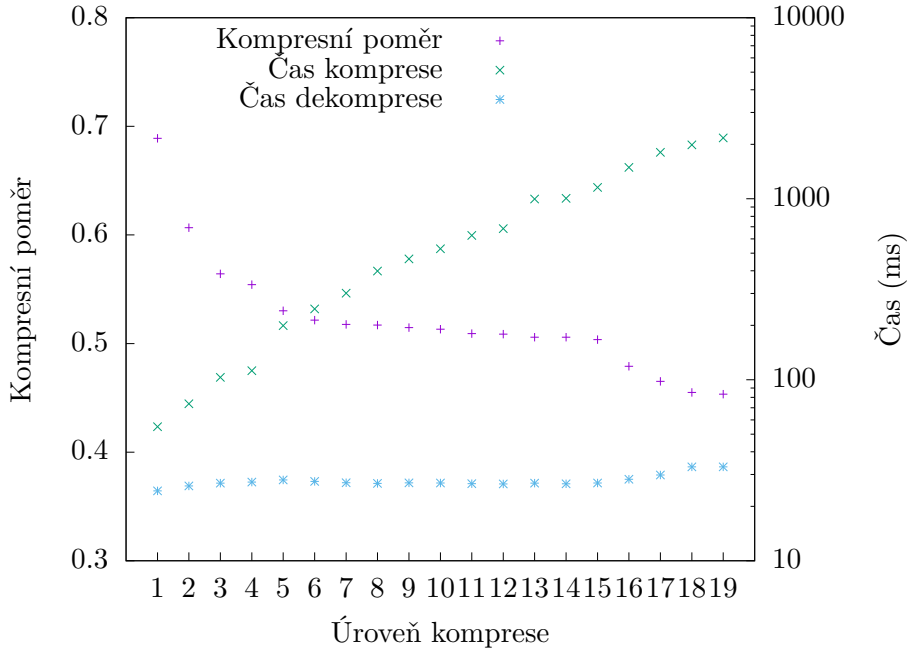
Pro minimalizování vlivu vedlejších efektů na výsledky měření jsem každý proces komprese a dekomprese provedl třicetkrát a uložil nejmenší naměřenou hodnotu. Dále jsem vypnul nepotřebné aplikace a odpojil notebook od internetu.

Z knihovny ExCom jsem použil tyto metody: acb, bwt, dhuff, lz78, lzap, lzmw, lzw, lzy, mtf, ppm, rle_n, sfano, shuff a zstd. Každou metodu jsem s defaultními hodnotami parametrů použil pro kompresi a dekompresi každého souboru z Pražského korpusu a uložil si čas komprese, čas dekomprese a

kompresní poměr. Všechny výsledky se nacházejí na přiloženém DVD, v následujících sekcích budu uvádět jen výsledky vybraných metod a souborů. Na DVD se nacházejí i dva skripty (`clevel-script` a `perf-script`), které jsem při testování použil. Byly napsané čistě pro mé potřeby a při spuštění na jiných zařízeních pravděpodobně nebudou bez mírné úpravy fungovat.

5.3 Porovnání úrovní komprese Zstandard

Zstandard je implementovaný v knihovně ExCom s jedním parametrem nazvaným *úroveň komprese* (angl. *compression level*). Jeho hodnota je přirozené číslo z intervalu 1–19, ve výchozím nastavení má hodnotu 3 (stejně jako v ukázkové implementaci od Yanna Colleta). Zvětšením úrovně komprese se zvětšuje velikost použitého klouzavého okénka. Měla by se tím zvýšit efektivita a zároveň by se měl prodloužit čas komprese.



Obrázek 5.1: Vliv úrovní komprese na kompresi a dekompresi souboru `flower`

Parametr jsem testoval na souboru `flower` z Pražského korpusu, protože na něm byl dobře vidět vliv jednotlivých úrovní na kompresní poměr a čas. Soubor je relativně veliký (10 MB) a obsahuje celkem obtížná data pro kompresi (jedná se o fotografii).

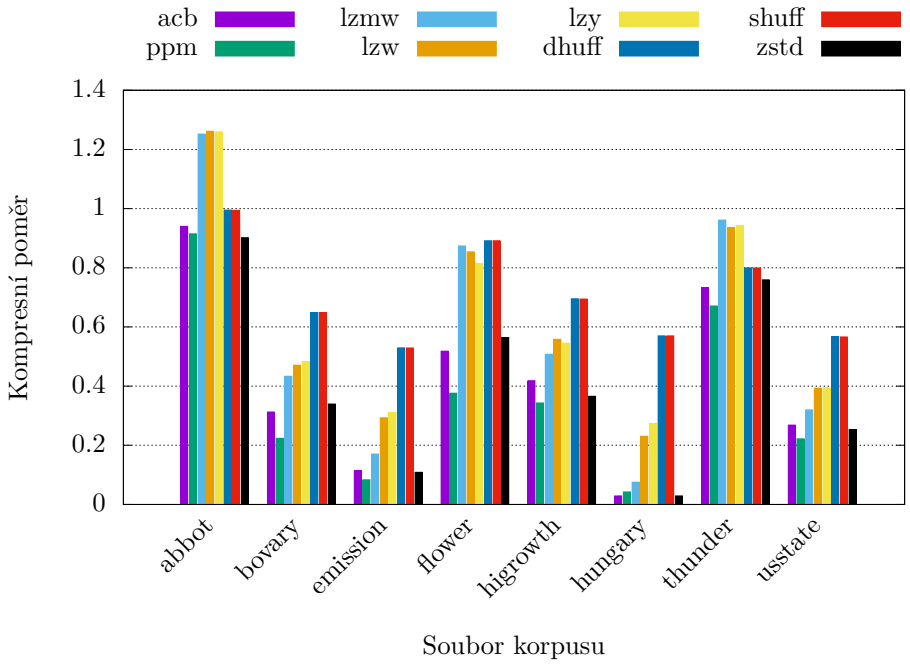
Z obrázku 5.1 je zřejmé, že s větší úrovní komprese roste její efektivita (kompresní poměr klesá), roste ale i čas potřebný pro její dokončení. Na čas

potřebný pro dekompresi nemá úroveň téměř žádný vliv. Úroveň 3 představuje celkem dobrý kompromis mezi rychlostí a efektivitou.

5.4 Porovnání Zstandard s ostatními metodami

Ze všech naměřených dat jsem si vybral hodnoty kontextových metod acb a ppm, slovníkových metod lzmw, lzw, lzy a Zstandard (v grafech zstd) a statistických metod implementujících Huffmanovo kódování dhuff a shuff naměřených na souborech `abbot`, `bovary`, `emission`, `flower`, `higrowth`, `hungary`, `thunder` a `usstate`. Každý z těchto souborů reprezentuje jednu kategorii dat v Pražském korpusu.

5.4.1 Kompresní poměr

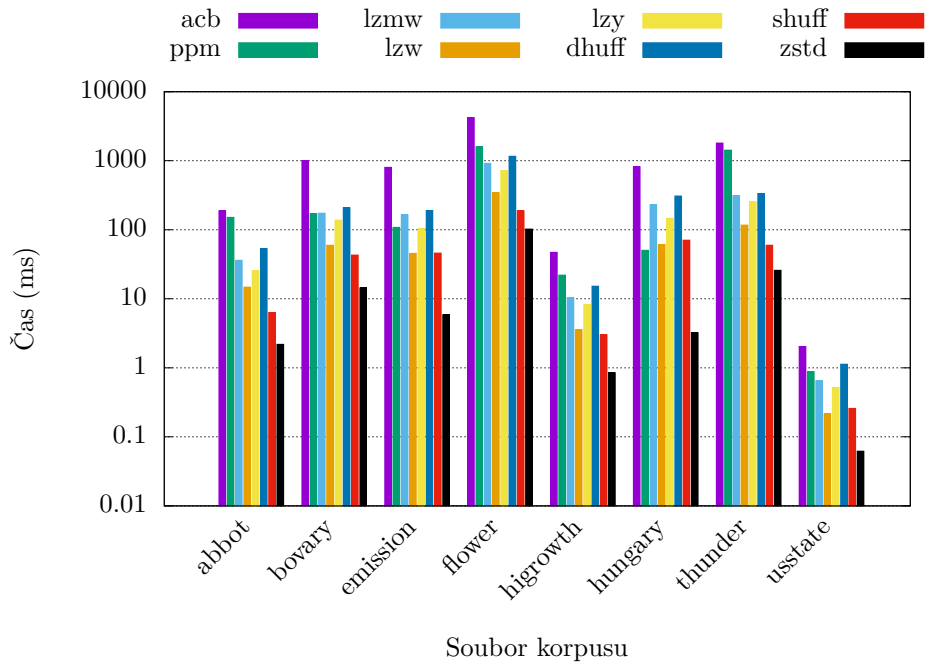


Obrázek 5.2: Kompresní poměry metod na souborech Pražského korpusu

Moderní metoda Zstandard v kompresním poměru překonala podle očekávání ostatní slovníkové i statistické metody, které jsou už relativně zastaralé a méně sofistikované. Zajímavé je však porovnání Zstandard s kontextovými metodami acb a ppm. Metoda acb se na všech typech dat držela blízko hodnot Zstandardu a metoda ppm jej dokonce překonávala, což je nejvýrazněji vidět na obrázku 5.2 u souboru `flower`.

Dále jsou zajímavé výsledky u souboru `abbot` obsahujícího velice obtížná data pro kompresi. Slovníkové algoritmy lzmw, lzw a lzy nemají žádné mechanismy proti negativní kompresi, takže jejich výstupní data měla vetší velikost než data vstupní.

5.4.2 Čas komprese

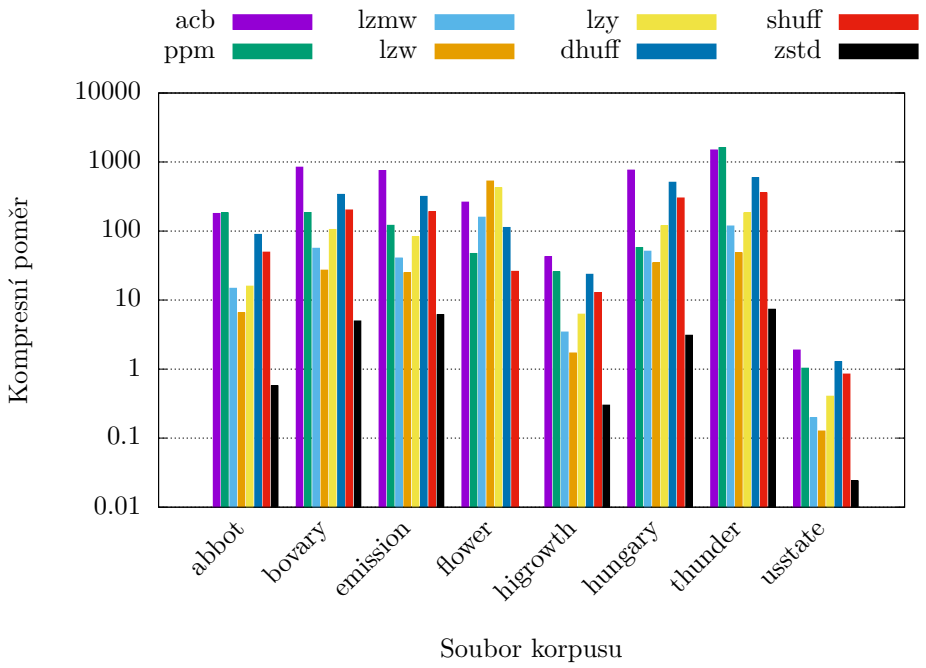


Obrázek 5.3: Časy strávené kompresí u metod na souborech Pražského korpusu

Přestože kontextové metody měly stejné nebo lepší výsledky u kompresního poměru, na obrázku 5.3 je odhalen jejich problém. V porovnání se Zstandard jsou při kompresi výrazně pomalejší.

Slovníkové metody lzmw, lzw i lzy vycházejí z algoritmu LZ78, proto jsou také dost pomalé (LZ78 se oproti LZ77 zaměřuje spíše na kompresní poměr než na rychlost). Zajímavější by bylo porovnat Zstandard s metodami na principu LZ77, metody lz77 a lzss implementované v ExComu však nebyly funkční.

5.4.3 Čas dekomprese



Obrázek 5.4: Časy strávené dekompresí u metod na souborech Pražského korpusu

Při porovnání časů potřebných pro dekompresi u jednotlivých metod vychází Zstandard ještě lépe než v předchozí sekci. Princip klouzavého okénka z rychlého algoritmu LZ77, moderní implementace Huffmanova kódování Huff0 a nová metoda efektivního entropického kódování FSE umožňují algoritmu Zstandard dosáhnout nejrychlejších dekompresí na všech měřených souborech.

Závěr

V práci jsem se zabýval kompresními metodami z rodiny LZ algoritmů. Dále jsem analyzoval novou metodu Zstandard a popsal její princip. V rámci praktické části jsem implementoval metodu Zstandard do knihovny ExCom, otestoval na datech Pražského korpusu a porovnal s ostatními metodami uvnitř knihovny.

První dvě kapitoly obsahují základní definice a podrobný popis algoritmu LZ77, ze kterého Zstandard vychází. Druhá kapitola obsahuje také krátké popisy algoritmů LZ78, LZSS a Deflate.

Třetí kapitola se zabývá charakteristikou algoritmu Zstandard a analýzou jeho formátu. V další kapitole popsána implementace do knihovny ExCom.

V poslední kapitole se nachází porovnání různých hodnot parametru *úroveň komprese* metody Zstandard implementované v knihovně ExCom. Kapitola se dále zaměřuje na porovnání kompresního poměru, rychlosti komprese a dekomprese Zstandard s jinými metodami.

V budoucnosti bylo možné metodu Zstandard v knihovně ExCom rozšířit o vícevláknovou kompresi a dekompresi nebo implementovat „Slovníkový režim“ (popsaný v 3.1) a otestovat jej na malých datech.

Literatura

[1] Collet, Y.: Zstandard Compression Format. *Github repository [online]*, 2016, [cit. 2018-04-22]. Dostupné z: `https://github.com/facebook/zstd/blob/dev/doc/zstd_compression_format.md`

[2] Šimek, F.: *Data compression library*. Diplomová práce, České vysoké učení technické v Praze, 2010.

[3] Holub, J.: *Přednášky k přemdětu MI-KOD (Komprese dat)*. České vysoké učení technické v Praze, 2018.

[4] Shannon, C. E.: A Mathematical Theory of Communication. *Bell System Technical Journal*, ročník 27, 1948.

[5] Lempel, A.; Ziv, J.: A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, ročník 23, 1977.

[6] Lempel, A.; Ziv, J.: Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, ročník 24, 1978.

[7] Huffman, D. A.: A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the Institute of Radio Engineers*, ročník 40, 1952.

[8] Collet, Y.: Zstandard Overview. *RealTime Data compression – Development blog on compression algorithms [online]*, 2018, [cit. 2018-04-22]. Dostupné z: `https://fastcompression.blogspot.cz/2018/01/zstandard-overview.html`

[9] Collet, Y.: Zstandard: A stronger compression algorithm. *RealTime Data compression – Development blog on compression algorithms [online]*, [cit. 2018-04-22]. Dostupné z: `http://fastcompression.blogspot.cz/2015/01/zstd-stronger-compression-algorithm.html`

[10] Collet, Y.; Turner, C.: Smaller and faster data compression with Zstandard. *Facebook Code [online]*, 2016, [cit. 2018-04-22]. Dostupné z: `https://code.facebook.com/posts/1658392934479273/smaller-and-faster-data-compression-with-zstandard/`

[11] Duda, J.: Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. *Computing Research Repository [online]*, 2013, [cit. 2018-04-22]. Dostupné z: `https://arxiv.org/pdf/1311.2540.pdf`

[12] Šimek, F.; Řezníček, J.: Knihovna ExCom. *Stringology – The Prague Stringology Club [online]*, 2009–2010, [cit. 2018-04-22]. Dostupné z: `http://www.stringology.org/projects/ExCom/`

[13] Autoreconfing packages – A Maintainers' Guide. *Debian Wiki [online]*, [cit. 2018-04-22]. Dostupné z: `https://wiki.debian.org/Autoreconf#Updating_macros`

[14] Collet, Y.: Zstandard – Fast real-time compression algorithm. *Github repository [online]*, [cit. 2018-04-22]. Dostupné z: `https://github.com/facebook/zstd`

[15] Holub, J.: New compression method. *Stringology – The Prague Stringology Club [online]*, 2013, [cit. 2018-04-22]. Dostupné z: `http://www.stringology.org/projects/ExCom/new_method.html`

[16] Deorowicz, S.: *Universal lossless data compression algorithms*. Dizertační práce, Silesian University of Technology, 2003.

[17] Řezníček, J.: *Corpus for comparing compression methods and an extension of a ExCom library*. Diplomová práce, České vysoké učení technické v Praze, 2009.

Příloha **A**

Seznam použitých zkratek

ACB Associative Coder of Buyanovsky

BSD Berkeley Software Distribution

ČVUT České vysoké učení technické v Praze

DCA Data Compression using Antidictionaries

ExCom Extensible Compression Library

FSE Finite State Entropy

GNU GNU is Not Unix

GPL General Public License

LGPL Lesser General Public License

LZ Lempel-Ziv

LZMA Lempel-Ziv-Markov-Chain

LZSS Lempel–Ziv–Storer–Szymanski

PDF Portable Document Format

PPM Prediction by Partial Matching

zstd Zstandard

Příloha B

Sestavení a spuštění knihovny ExCom

Na unixových lze knihovnu ExCom snadno sestavit a zkompilovat pomocí GNU autotools. V kořenovém adresáři ExComu se nachází potřebné skripty `configure.ac` a `Makefile.am`, pro sestavení a nainstalování knihovny tedy stačí pouze zadat do terminálu následující příkazy:

```
autoreconf -i -s -f
./configure --enable-perf-measure
make
make install
```

Pro odinstalování knihovny ze systému stačí příkaz:

```
make uninstall
```

Pro spuštění knihovny slouží konzolová aplikace `app` umístěným v adresáři `/Excom/src/app`. Aplikace se používá následujícím způsobem:

```
# Popis všech přepínačů pro ExCom
./app -h

# Zobrazení seznamu všech metod obsažených v knihovně
./app -m ?

# Zobrazení a popis parametrů metody Zstandard
./app -m zstd -p ?

# Ukázka použití metody Zstandard na kompresi
# souboru /tmp/input-file s úrovní komprese 19
./app -m zstd -p l=19 -i /tmp/input-file -o /tmp/input-file.zst
```

Příloha C

Obsah přiloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `ExCom` ..... zdrojové soubory knihovny ExCom
- `Performance` ..... testovací skripty a všechny výsledky testování
  - `clevel-script` ..... skript pro měření úrovní komprese Zstandard
  - `perf-script` ..... skript pro měření všech metod v ExComu
  - `Data` ..... všechny výsledky mých měření
- `PragueCorpus` ..... soubory Pražského korpusu
- `Text` ..... text práce
  - `bakalarska-prace.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojová forma práce ve formátu LaTeX