VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

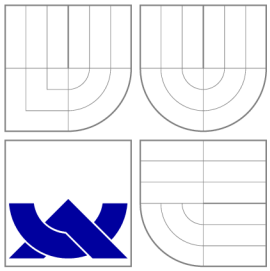
KOMPRESE OBRAZOVÝCH DAT V MEDICÍNĚ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

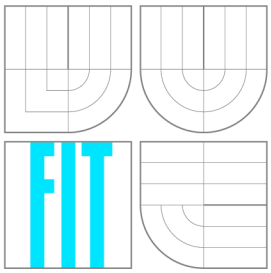
AUTOR PRÁCE
AUTHOR

JIŘÍ ŠEBEK

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMPRESE OBRAZOVÝCH DAT V MEDICÍNĚ

MEDICAL IMAGE DATA COMPRESSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ ŠEBEK
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. PŘEMYSL KRŠEK, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá metodami bezeztrátové komprese obrazových dat. Cílem práce je dosáhnout co nejlepšího kompresního poměru při kompresi obrazových medicínských dat. Práce zahrnuje uvedení do teoretických základů komprese, návrh komprese pro obrazová medicínská data a popis implementace kompresních modulů. V závěru práce je obsaženo ověření vhodnosti navržené metody a porovnání dosažených výsledků s dalšími kompresními metodami.

Abstract

This thesis deals with several lossless methods of image compression. The main goal is to achieve the best compression ratio for set of medical images. This paper provides brief introduction into the data compression theory. It also contains design of image compression and description of designed compression modules implementation. Also the verification of suitability of designed method for medical images compression is involved.

Klíčová slova

Komprese, komprese obrazu, bezeztrátová komprese, neztrátová komprese, medicínská data, slovníkové metody, statistické metody, transformace, RLE, metoda potlačování nul, delta kódování, MTF, BWT, Shannon–Fanovo kódování, Huffmanovo kódování, aritmetické kódování, LZ–77, LZ–78, LZW.

Keywords

Compression, image comression, lossless compression, medical images, dictionary methods, statistical methods, transformation, RLE, zero elimination, delta coding, MTF, BWT, Shannon–Fano coding, Huffman coding, arithmetic coding, LZ–77, LZ–78, LZW.

Citace

Jiří Šebek: Komprese obrazových dat v medicíně, bakalářská práce, Brno, FIT VUT v Brně, 2012

Komprese obrazových dat v medicíně

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Přemysla Krška, Ph.D..

.....
Jiří Šebek
14. května 2012

Poděkování

Rád bych na tomto místě poděkoval mému vedoucímu za věcné rady a konstruktivní kritiku. Dalším, koho bych nerad opomenul, je Ing. Michal Španěl, Ph.D., který mi poskytl dodatečné informace o nástroji *MDSTk*.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

S dnešním pojetím moderní medicíny jsou již neoddělitelně spjaty možnosti vyšetření pomocí některé z radiologických metod. Ať již se jedná o magnetickou rezonanci, počítačovou tomografii, rentgenové prosvěcování, ultrazvuk nebo další úzce specializované metody, je třeba výsledek vyšetření vždy zaznamenat na určité médium.

V současné době výpočetní techniky jsou tyto informace, často obrazového charakteru, uchovávány v digitální podobě. S pokračujícím zdokonalováním těchto neinvazivních lékařských vyšetření se však postupně zvyšují i datové nároky na přenos a uchovávání výsledků vykonaných vyšetření. Na druhou stranu je třeba brát v potaz, že i při dnes již značně pokročilých technických možnostech integrace, jsou kapacity médií pro uchování dat stále omezené.

Schopnost uložit na ekvivalentní médium větší množství dat je tedy jednou z hlavních motivací pro vznik kompresních algoritmů. Další motivací je například snížení nutného přenosového pásma při přenosu dat přes počítačovou síť. Tato práce se zabývá bezeztrátovými kompresními metodami, konkrétně jejich aplikací v prostředí komprese obrazových medicínských dat.

První část práce je věnována osvětlení teoretické podstaty komprese a představení jednotlivých kompresních metod. Jedná se o tři kapitoly, kde v kapitole 2 naleznete vysvětlení nejdůležitějších pojmů z teorie informace a teorie kódování, které jsou dále v textu využívány. Kapitola 3 představuje rozdělení kompresních metod z několika pohledů a popisuje také důležité vlastnosti, kterých mohou jednotlivé kompresní metody nabývat. Závěrečnou kapitolu teoretického bloku (kapitola 4) tvoří popis jednotlivých logicky odlišných přístupů ke kompresi a představení významných zástupců těchto rodin kompresních algoritmů.

Další část práce se skládá z prakticky orientovaného bloku, který sestává ze tří kapitol. V kapitole 5 je představen návrh komprese, který vychází z vlastností typických právě pro medicínská data. Následuje kapitola 6, kde je uveden popis implementace vybraných kompresních metod. Jsou zde také popsány případné odchylky od jejich představení v teoretické části. Poslední a současně nejvýznamnější kapitolou je kapitola 7 praktického bloku popisuje způsob testování jednotlivých navržených kompresních metod a obsahuje vyhodnocení výsledků, kterých bylo pomocí těchto metod dosaženo.

Kapitola 2

Teorie informace a kódování

Následující kapitola ve stručnosti představuje základy teorie informace, teorie kódování a vymezuje smysl pojmů, které jsou používány dále v textu. Jako prerekvizita se také předpokládá znalost pojmů z oblasti teorie pravděpodobnosti. Nicméně, pokud je čtenář s výše uvedenou problematikou obeznámen, může volně přejít ke kapitole 2, aniž by se musel obávat ztráty kontextu. Informace zde uvedené jsem čerpal z [9],[7] a [8].

2.1 Data a informace

Terminologie dat a informací se v běžném životě často zaměňuje, ač mají tyto pojmy v informatice odlišné významy.

Data jsou jakékoliv vyjádření skutečnosti. Jedná se tedy o statická fakta, která jsou časově nezávislá a z počítačového hlediska se jedná jen o hodnoty datových typů. Samotná data nemusejí mít, a zpravidla také nemají, sémantiku. Mezi základní vlastnosti dat tedy patří schopnost přenosu, uchování, strojového zpracování a v neposlední řadě interpretace.

Právě interpretací dat získáváme informace. Pojem informace lze tedy přisoudit datům, kterým byla přiřazena sémantika. Tento proces je do značné míry subjektivní, neboť interpretaci dat provádí uživatel.

2.2 Entropie dat

Z teorie informace plyne, že má-li určitá zpráva nést informaci, musí tomu předcházet neznalost jejího obsahu. Postupné čtení jednotlivých symbolů lze s pomocí teorie pravděpodobnosti označit za náhodné jevy. Entropie nastoupení jevu, tedy výskytu daného symbolu, je tím vyšší, čím nižší je pravděpodobnost, že se právě tento symbol vyskytne. Celkovou entropii zprávy lze získat dle vztahu 2.1, kde $P(S_i)$ značí pravděpodobnost výskytu daného symbolu.

$$H(S) = -\sum_{i=1}^{n} P(S_i) * \log_2 P(S_i) \tag{2.1}$$

$$\sum_{i=1}^{n} P(S_i) = 1 \tag{2.2}$$

Při uvažování předchozích faktů můžeme entropii označit za míru neurčitosti. Tedy i za míru informace, kterou průměrně nese jeden elementární jev. Ve smyslu obrazových dat se jedná o střední hodnotu informace, kterou nese jeden kódovaný symbol, tedy pixel.

2.3 Redundance dat

Na základě pojmů z teorie kódování, blíže specifikovaných v kapitolách 2.4 a 2.5, lze formálně definovat redundanci dat, která je pro obor komprese pojmem klíčovým.

Předpokládejme kódování zdrojové zprávy x a označme jeho kódové slovo $C(x)$. Délku kódového slova $C(x)$ označme $L(x)$. Délka kódového slova je, stejně jako entropie dat, udávána v bitech. Z teorie informace vyplývá, že $H(x) \leq L(x)$. Rozdíl mezi délkou kódu a entropií $H(x)$ je tedy vždy nezáporný a nazývá se redundance. Redundanci můžeme považovat za nadbytečnost kódového slova $C(x)$. Je značena $R(x)$ a formálně lze její hodnotu získat dle vztahu 2.3.

$$R(x) = L(x) - H(x) \tag{2.3}$$

2.4 Kódování

Kódováním se rozumí proces vyjádření téže informace jinou formou. Zpravidla se jedná o převod prvků vstupní abecedy na symboly výstupní abecedy. Abeceda je v tomto smyslu definována jako konečná a neprázdná množina prvků, které nazýváme symboly.

Formálně řečeno se jedná o zobrazení ze zdrojové abecedy S do kódové, neboli výstupní, abecedy C. Aby bylo dané zobrazení kódováním, je třeba, aby splňovalo podmínky injektivního zobrazení.

2.5 Kód

Kód K je možné definovat jako uspořádanou trojici (S, C, f), kde S je abeceda zdrojových symbolů, C je abeceda výstupních symbolů a funkce f je zobrazením s vlastnostmi kódování, diskutovanými výše.

2.5.1 Jednoznačně dekódovatelný kód

Nechť C^+ je množinou všech řetězců nenulových délek, vzniklých nad abecedou C. Řetězec $x \in C^+$ je jednoznačně dekódovatelný vzhledem k zobrazení f, jestliže existuje právě jeden řetězec $y \in S^+$ takový, že $f(y) = x$. Kód (S, C, f) je jednoznačně dekódovatelný kód právě tehdy, když všechny možné řetězce z C^+ jsou jednoznačně dekódovatelné.

2.5.2 Prefixový kód

Kód je možné považovat za prefixový, je-li jeho základem kódování f a neexistují-li různé X_i a X_j takové, že $f(X_i)$ je prefixem $f(X_j)$.

Definice 2.1 *Nechť L je libovolný jazyk. L je* bezkontextový jazyk, *když a jen když $L = L(G)$, kde G je libovolná bezkontextová gramatika.*

2.5.3 Optimální kód

Za optimální kód lze považovat takový kód, jehož výstup se blíží entropii kódovaných dat ze všech kódů nejvíce. Dle formální definice je pak kód (S, C, f) optimální, pokud neexistuje kód (S, C, f') takový, aby platila nerovnost ve vztahu 2.4, kde p představuje pravděpodobnostní rozdělení jednotlivých symbolů.

$$\sum_{i \in S} P_i * |f'(i)| < \sum_{i \in S} P_i * |f(i)| \tag{2.4}$$

Kapitola 3

Komprese

Komprese, někdy také zvaná komprimace, je speciálním případem kódování (vizte kapitola 2.4), které má za účel snížit náročnost vstupních dat na zdroje.

Dosažení komprimace dat je založeno na snižování míry jejich redundance (vizte kapitola 2.3), případně na odstraňování irelevance v datech. Tyto dva přístupy mají diametrálně odlišné vlastnosti, které jsou diskutovány dále v textu.

3.1 Ztrátová komprese

Tato kategorie metod se zabývá snižováním irelevance dat, přičemž je určitá část dat a informace kterou nesou nenávratně ztracena. Princip ztrátové komprese spočívá v nalezení méně významné složky dat, která je pak potlačena, či zcela odstraněna. Po rekonstrukci je dosaženo pouze přibližných hodnot původních dat.

Ztrátové kompresní metody je možné využít v situacích, kdy je třeba dosáhnout vyšších kompresních poměrů a současně je tolerovatelná určitá míra zkreslení dat. Typickým příkladem nasazení ztrátových metod jsou multimédia, kdy se využívá nedokonalosti lidských smyslů.

3.2 Bezeztrátová komprese

Bezeztrátová komprese dat pracuje na principu snižování redundance dat. U těchto metod nelze očekávat příliš velký kompresní poměr, neboť maximální teoretická hranice bezeztrátové komprese je určena mírou entropie (vizte kapitola 2.2) komprimovaných dat. Tento zdánlivý nedostatek je však kompenzován možností provedení přesné rekonstrukce dat do podoby před komprimací. Jedná se o vlastnost, která předurčuje bezeztrátové techniky komprese k využití v oblastech, kde nelze tolerovat jakékoli zkreslení dat. Pro bezeztrátové kompresní programy je typické využívání kaskádové komprese, tedy hned několika vhodně seřazených bezeztrátových algoritmů. Častým případem je situace, kdy nejprve dochází k transformaci vstupních dat. Tato transformace sama o sobě nemusí nutně data komprimovat, ale je prováděna za účelem získání dat v podobě vhodnější pro kompresi, kde je pak možné docílit vyššího kompresního poměru. Při dekompresi pak dekodér pro získání původních dat využívá jednotlivých dekompresních metod v zrcadlově otočeném pořadí.

3.3 Další vlastnosti komprese

Kompresní metody a jejich vlastnosti lze posuzovat z mnoha dalších úhlů. V následující podkapitole jsem popsal vybrané vlastnosti, které mohou ovlivnit vhodnost algoritmu pro použití v dané úloze.

3.3.1 Symetrická a asymetrická komprese

Určení, zda je konkrétní kompresní metoda symetrická či asymetrická, probíhá na základě zhodnocení asymptotické složitosti komprese a dekomprese. Pokud se od sebe obě složitosti významněji liší, pak je daná metoda označena za asymetrickou.

Asymetrických algoritmů se využívá zejména při zvýšeném požadavku na jednu z činností. Například při archivaci dat, ke kterým je velmi často přistupováno pouze pro čtení, je velmi výhodné použít asymetrickou metodu, kde bude nižší složitost dekomprese, třeba i za cenu vyšší složitosti komprese.

3.3.2 Proudová a bloková komprese

Proudovou metodou je taková kompresní metoda, která dokáže svůj vstup generovat průběžně po zpracování jednotlivých vstupních symbolů. Naopak bloková metoda pracuje nad vstupním blokem dat určité velikosti. Tento blok svým kompresním algoritmem transformuje a následně vygeneruje odpovídající výstup.

Mezi nevýhody blokových algoritmů patří vyšší paměťová náročnost jak při kompresi, tak při následné dekompresi. Tento fakt je způsoben nutností uchovávat v paměti celý datový blok, který je zpracováván. Nicméně lze vhodnou transformací bloku přispět ke zvýšení kompresního poměru.

3.3.3 Statická a adaptivní komprese

Pokud je kompresní metoda statická, pak se model jejího algoritmu v čase nemění. Naopak adaptivní metody upravují svůj výpočetní model dle vstupních dat.

Adaptivní metody je možné dále rozlišit na lokálně adaptivní a semiadaptivní metody. Semiadaptivní metody jsou víceprůchodové a svůj výpočetní postup upravují mezi jednotlivými průchody. Oproti tomu lokálně adaptivní metody nemusejí být víceprůchodové a upravují svůj model algoritmu průběžně po zpracování jednotlivých symbolů vstupní abecedy.

3.4 Hodnocení účinnosti komprese

Existuje samozřejmě celá řada metrik pro vyjádření účinnosti komprese. V této kapitole jsou vybrány ty nejdůležitější a je popsán jejich vzájemný vztah. Dalšími důležitými kritérii pro hodnocení kompresních metod jsou univerzálnost, paměťová náročnost a časová složitost metod. Tato kritéria jsou diskutována přímo u konkrétních metod.

3.4.1 Kompresní poměr

Za základní metriku pro zjišťování účinnosti komprese lze považovat kompresní poměr. Tato metrika udává poměr velikosti originálních dat ku velikosti dat komprimovaných. Pokud je hodnota kompresního poměru menší než jedna, pak došlo k expanzi vstupních dat, což je nežádoucí jev.

Princip výpočtu kompresního poměru $P(X)$ ukazuje vzah 3.1, kde $O(X)$ představuje velikost originálních dat a $L(X)$ velikosti komprimovaných dat. Z povahy výpočtu této veličiny také plyne její nezápornost a bezrozměrnost. Existují další příbuzné metriky, například zisk komprese (vztah 3.2) a míra úspory místa (vztah 3.3).

$$P(X) = \frac{O(X)}{L(X)} \tag{3.1}$$

$$Z(X) = \frac{O(X) - L(X)}{O(X)} \tag{3.2}$$

$$U(X) = 1 - \frac{L(X)}{O(X)} \tag{3.3}$$

3.4.2 Počet bitů k vyjádření jednotky dat

V anglicky psané literatuře se setkáváme s pojmem bits per byte (bpb), který udává počet bitů nutných pro reprezentaci jednoho bytu vstupních dat. Často je tato míra účinnosti zvána také bits per character (bpc). Typické je to při řešení problematiky komprese textu, kdy se takto značí počet bitů nutných pro vyjádření jednoho znaku. V části literatury je pak bpc použito i ve významu, kde znak nemusí nutně reprezentovat jen textový znak, ale je chápán jako symbol nad vstupní abecedou použitého kódování.

Při kompresi grafických dat je pak možné zavedení pojmu bits per pixel (bpp). Tedy počtu bitů nutných pro reprezentaci jednoho obrazového bodu – pixelu[12]. Tento způsob vyjádření je motivován skutečností, že se k popisu pixelu často využívá barevných schémat, která mají jiný rozsah, než je právě jeden byte. Například pro medicínská obrazová data ve formátu *DICOM* je hodnota bpp dvojnásobná oproti bpb. Tato skutečnost je způsobena využíváním rozsahu dvou bytů pro popis hodnoty jednoho pixelu.

Kapitola 4

Algoritmy bezeztrátové komprese

Jednotlivé algoritmy je možné logicky rozdělit do tříd dle teoretických úvah, na jejichž základě vznikly. V následující kapitole tedy představím hlavní proudy kompresních metod a jejich typické zástupce. U každé z následujících kompresních metod je diskutován jak princip dané metody, tak její význačné vlastnosti. Informace zde uvedené jsem čerpal z [8] a [5]. Doplňkové informace a jsem získal z [11] a [10].

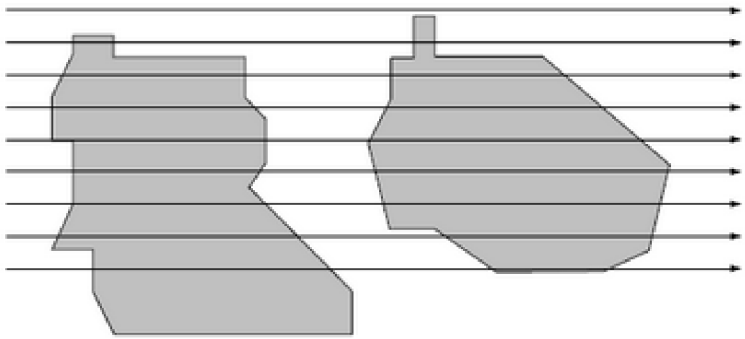
4.1 Základní metody komprese

4.1.1 RLE

Jedná se o jednoprůchodovou metodu pro proudovou kompresi vstupních dat. Myšlenka tohoto přístupu ke kompresi je poměrně intuitivní a je založena na možnosti nahradit sekvenci n-krát se opakujícího symbolu d jedinou dvojicí (n, d). V tomto zápisu pak d značí symbol a n počet jeho bezprostředních opakování.

Účinnost komprese je silně závislá na charakteru vstupních dat, kde je vhodným předpokladem, aby vstup obsahoval delší sekvence stejných symbolů. Jak z principu tohoto kódování vyplývá, dojde k úspěšné kompresi sekvence symbolů jen pokud je $n \geq 3$. V případech, kdy je $n = 2$ tato metoda není efektivní a pokud je zakódována jednosymbolová sekvence, dochází dokonce k nežádoucí expanzi.

RLE je využívána v počítačové grafice jako pomocná metoda komprese dat při vícestupňové (kaskádové) kompresi, například u standardu JPEG.



Obrázek 4.1: Ilustrace použití metody RLE pro komresi obrazu. Příklad převzat z [8]

4.1.2 Metoda potlačení nul

Metoda potlačení nul je jednou z nejdéle využívaných kompresních metod. Dnes je tato jednoprůchodová metoda v praxi využívána převážně v kombinaci s dalšími stupni komprese. Ač se tato metoda nazývá metodou potlačení nul, jedná se o potlačování jednoho určitého symbolu vstupní abecedy, který nemusí být vždy roven právě nule.

V opačném případě výrazně klesá účinnost komprese a v extrémních případech může dojít i k nárůstu velikosti vstupních dat.

Konstrukce výstupního kódu
Princip je velmi podobný jako u metody RLE, která vznikla rozšířením této metody. U metody potlačení nul jsou ovšem všechny nesledované znaky zapisovány na výstup v nezměněné podobě. Jakmile kodér nalezne vyhledávaný symbol, pokusí se zakódovat nejdelší sekvenci tvořenou jeho bezprostředním opakováním. Následně je výskyt této sekvence nahrazen dvojicí (n, d), kde d je daným symbolem a n představuje počet jeho bezprostředních opakování. Příklad zahrnující metodu potlačení nul je uveden na obrázku 4.2.

$$\begin{matrix} 79 & 0 & 3 & -1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 0 & -2 \\ -1 & 3 & 0 & 0 & 0 & -1 \\ 0 & 3 & 2 & -1 & -1 & 0 \\ 0 & 2 & -1 & 1 & -2 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & -1 \end{matrix} \xrightarrow{\textit{Metoda potlačení nul}} \begin{matrix} 79 & (1,0) & 3 & -1 & (3,0) & 2 \\ 2 & (2,0) & -2 & -1 & 3 & (3,0) \\ -1 & (1,0) & 3 & 2 & -1 & -1 \\ (2,0) & 2 & -1 & 1 & -2 & (3,0) \\ -1 & (3,0) & -1 & (3,0) & 1 & -1 \end{matrix}$$

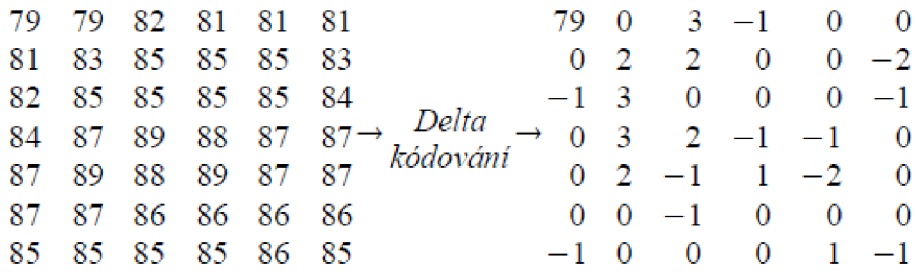
Obrázek 4.2: Ilustrace použití metody potlačení nul.

Zpětná dekomprese
Dekomprese je poměrně přímočará, neboť při průchodu komprimovaných dat dochází u nesledovaných symbolů k jejich nezměněnému odeslání na výstup. Při nalezení dvojice (n, d) pak dojde k nahrazení symbolu d jeho n výskyty.

4.1.3 Delta kódování

Principem tohoto jednoprůchodového kódování je ukládání rozdílů mezi hodnotami vstupních symbolů, namísto jejich původních hodnot. Tím je možné dosáhnout snížení průměrné hodnoty výstupních symbolů oproti vstupním.

První symbol je odeslán na výstup v nezměněné podobě, zatímco při zpracování dalších symbolů se jedná o hodnotu rozdílu oproti předchozímu symbolu. Následná rekonstrukce probíhá obdobným způsobem, kdy je nejprve načten první (tudíž nezměněný) symbol. Hodnoty následujících symbolů je možné rekonstruovat vždy pomocí odečtení zaznamenaného rozdílu od vypočtené hodnoty předchozího symbolu.



Obrázek 4.3: Ilustrace aplikace delta kódování.

4.1.4 MTF

Rodina algoritmů typu move to front je často používána jako mezistupeň po provedení Burrows-Wheelerovy transformace (vizte kapitola 4.2.1) a před kompresí pomocí entropického kódování.

Princip překódování spočívá v udržování abecedy A ve formě zásobníku. Jedná se o lokálně adaptivní metodu, kde je výstupem překódování vzdálenost aktuálně zpracovávaného symbolu od vrcholu zásobníku. Po každém kroku je právě zpracovaný symbol přesunut na vrchol zásobníku.

Algoritmus 1: MTF

Input: Symboly
Output: Vzdálenost aktuálního symbolu od vrcholu zásobníku

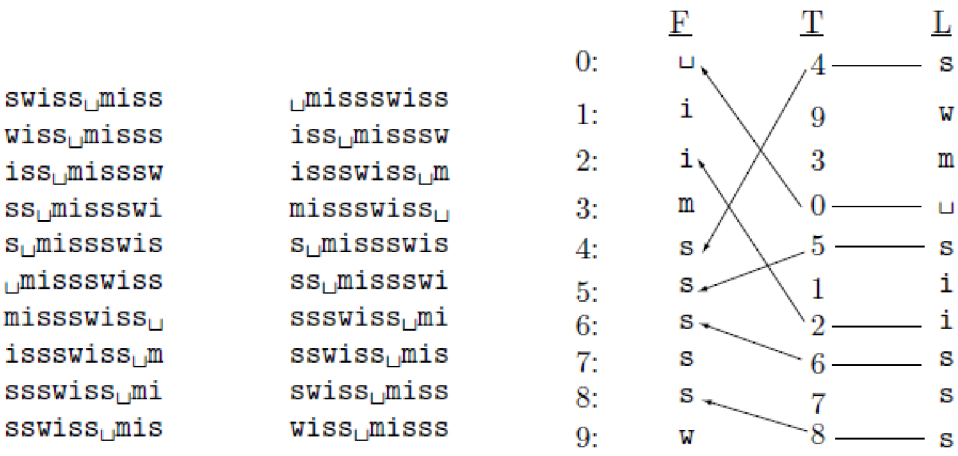
1: *inicializuj zásobník vložením všech symbolů vstupní abecedy*
2: **while** *existuje nezakódovaný symbol* **do**
3: *načti_symbol(***symbol***)*
4: *tiskni_výstup(***pozice symbolu v zásobníku***)*
5: *přesuň* **symbol** *na začátek zásobníku*
6: **end while**

4.2 Transformační kompresní techniky

4.2.1 BWT

Jedná se o blokový transformační algoritmus, který je velmi univerzální a využívá se během komprese obrazových, zvukových i textových dat. Burrows-Wheelerova transformace vstupní data přímo nekomprimuje, ale převádí je do podoby vhodnější pro následnou kompresi. Hlavní myšlenkou této metody je transformovat každý blok, který je definován jako řetězec S o n znacích, na jiný řetězec L o n znacích. Smyslem je transformovat vstupní řetězec S tak, aby výstupní řetězec L obsahoval shluky stejných symbolů a současně zůstala zachována možnost zpětného získání původního řetězce S. Shluků stejných znaků v řetězci L se dá následně využít při komprimaci, například výše uvedenou metodou RLE (vizte kapitola 4.1.1).

Konstrukce řetězce L
Prvním krokem je vytvoření matice $n \times n$ znaků, kde bude na prvním řádku vstupní řetězec S. Na následujících n řádcích budou řetězce vzniklé postupnou cyklickou rotací původního řetězce S. Pro názornost je postup ukázán na obrázku 4.4.



Obrázek 4.4: Ilustrace principu BWT. Příklad převzat z [8]

Řádky takto vzniklé matice jsou následně lexikograficky seřazeny, což představuje druhá matice na obrázku 4.4. V tuto chvíli je důležité si povšimnout, že jak každý řádek, tak každý sloupec seřazené matice, jsou jednotlivými permutacemi původního řetězce S. Jako řetězec L je kodérem vybrána permutace, která se nachází v posledním sloupci seřazené matice. Z pouhého řetězce L není možné určit původní řetězec S, proto je nutné jej doplnit o informaci I. Tato informace je definována číslem řádku, na kterém se nachází původní řetězec v seřazené matici (ve výše uvedeném příkladu se jedná o číslo 8).

Rekonstrukce řetězce S
K rekonstrukci původního řetězce S je třeba znát hodnotu I a dva sloupce seřazené matice, která vznikla při transformaci řetězce S na L. Jedním z těchto sloupců je samotný řetězec L, který vznikl právě jako poslední sloupec seřazené matice. K možnosti rekonstrukce tedy zbývá získat další sloupec seřazené matice. Je třeba si uvědomit, že se v každém sloupci

nachází určitá permutace původního řetězce S. Vzhledem k tomu, že matice vznikla lexikografickým seřazením, je jisté, že její první sloupec F vznikne stejnou metodou, tentokrát aplikovanou na řetězec L. Během řazení posledního sloupce vytvoříme pomocný vektor T, který bude vyjadřovat vztah mezi elementy obsaženými v L a F. Samotná rekonstrukce pak probíhá dle vztahů 4.1, 4.2 a 4.3.

$$S[n-1-i] = L[T^i[I]],\ pro\ i \in \{0,1,\ldots,n-1\} \tag{4.1}$$
$$T^0[j] = j \tag{4.2}$$
$$T^{i+1}[j] = T[T^i[j]] \tag{4.3}$$

Pro ilustraci je zde uveden kompletní příklad rekonstrukce dříve zakódovaného řetězce „swiss miss“, vizte pole rovnic 4.4.

$$\begin{aligned}
S[10-1-0] &= L[T^0[I]] = L[I] &&= L[8] = \text{"}s\text{"} \\
S[10-1-1] &= L[T^1[I]] = L[T[8]] = L[7] = \text{"}s\text{"} \\
S[10-1-2] &= L[T^2[I]] = L[T[7]] = L[6] = \text{"}i\text{"} \\
S[10-1-3] &= L[T^3[I]] = L[T[6]] = L[2] = \text{"}m\text{"} \\
S[10-1-4] &= L[T^4[I]] = L[T[2]] = L[3] = \text{"}_\text{"} \\
S[10-1-5] &= L[T^5[I]] = L[T[3]] = L[0] = \text{"}s\text{"} \\
S[10-1-6] &= L[T^6[I]] = L[T[0]] = L[4] = \text{"}s\text{"} \\
S[10-1-7] &= L[T^7[I]] = L[T[4]] = L[5] = \text{"}i\text{"} \\
S[10-1-8] &= L[T^8[I]] = L[T[5]] = L[1] = \text{"}w\text{"} \\
S[10-1-9] &= L[T^9[I]] = L[T[1]] = L[9] = \text{"}s\text{"}
\end{aligned} \tag{4.4}$$

4.3 Statistické metody

Pro metody, které nepracují se statistickým modelem vstupních dat, je typický jeden společný znak, kterým je přiřazování kódového slova stejné délky pro všechny vstupní symboly. Z těchto vlastností vyplývá zásadní nedostatek takto pracujících metod, kdy je i pro symboly s velmi častým výskytem použito stejné množství bitů jako pro symboly se statisticky mizivou pravděpodobností výskytu.

Statistické metody využívají odlišný přístup, neboť vytvářejí výstupní kód s variabilní délkou. A to takovým způsobem, kdy pro nejčastěji se vyskytující vstupní symboly je generována kratší bitová sekvence výstupního kódu.

Stejně jako u všech ostatních kompresních metod je nutné zajistit možnost zpětné dekomprese vygenerovaného kódu. U statistických metod může dojít ke splnění tohoto požadavku například konstrukcí prefixového kódu, jehož vlastnosti jsou detailněji popsány v kapitole 2.5.2. Neméně důležitým předpokladem pro nasazení statistické metody v praxi je generování výstupního kódu s minimální střední délkou. Způsob konstrukce výstupního kódu se u různých metod do značné míry liší, a proto je vždy diskutován u konkrétní statistické metody.

4.3.1 Shannon-Fanovo kódování

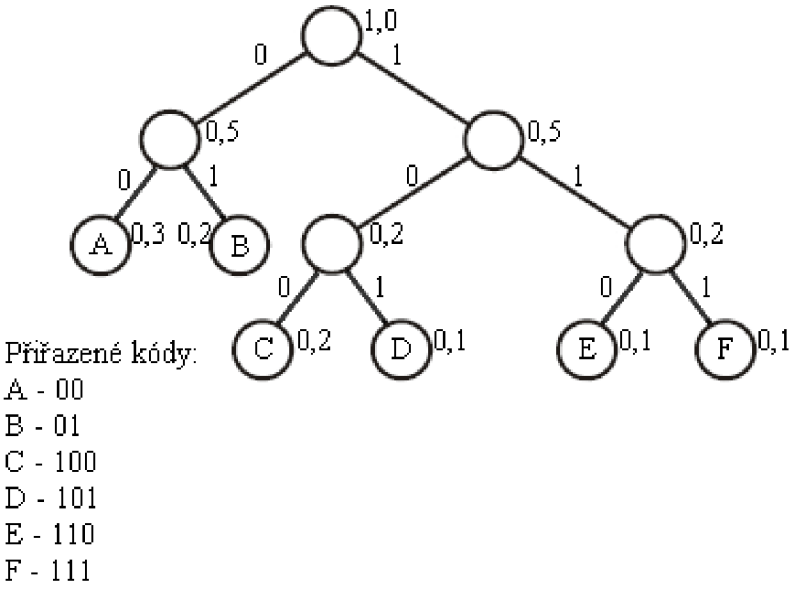
Shannon-Fanovo kódování je prvním algoritmem, který byl navržen pro konstrukci dobrých kódů s proměnnou délkou, současně splňujících vlastnosti prefixového kódu. Shannon-Fanův kód je možné reprezentovat acyklickým grafem, konkrétně binárním stromem, vybudovaným nad symboly vstupní abecedy.

Konstrukce stromu Shannon-Fanova kódování
Pro vybudování tohoto stromu a následný návrh kódu je třeba seřazení symbolů vstupní abecedy do nerostoucí posloupnosti dle četnosti jejich výskytu. Dalším krokem je rozdělení této posloupnosti do dvou podmnožin tak, aby rozdíl součtů pravděpodobností výskytu jednotlivých symbolů v rámci obou podmnožin byl co nejnižší.

Následuje ohodnocení jednotlivých symbolů. Všem symbolům patřícím do první množiny je přiřazen kód začínající nulovým bitem, zatímco všem symbolům z druhé množiny je přiřazeno ohodnocení bitem jedna. Každá z dříve vzniklých podmnožin je dále rekurzivně dělena a symbolům jsou přiřazovány další bity kódu. Proces dělení podmnožin končí ve chvíli, kdy podmnožina obsahuje jediný prvek.

Jak již z principu konstrukce Shannon-Fanových kódů plyne, jedná se o prefixové kódování, které ale nemusí vykazovat optimální výsledky ve smyslu definice optimálnosti kódu (vizte kapitola 2.5.3). Tato skutečnost je způsobená vlastností Shannon-Fanova kódování, kde je kvalita komprese velmi závislá na způsobu rozdělení vstupní abecedy do podmnožin. Lze dokázat, že pokud jsou podmnožiny děleny ideálně, tedy tak, že rozdíl součtů pravděpodobnosti výskytu prvků v nich obsažených je nulový, pak tato metoda vykazuje optimální výsledky.

Ideální dělení podmnožin je možné, pokud mají symboly vstupní abecedy pravděpodobnost výskytu v kódované zprávě rovnu záporné mocnině čísla dvě. V takové situaci kódování dosahuje přímo hodnoty entropie zprávy. Obrázek 4.5 ilustruje princip konstrukce Shannon-Fanova kódu na příkladu zprávy, která obsahuje šest symbolů. Střední délku takto sestaveného kódu, označenou $overline L(X)$, je možné získat pomocí vztahu 4.5, kde p_x představuje pravděpodobnost výskytu určitého symbolu a $L(x)$ délku jeho kódu. Ve výše



Obrázek 4.5: Postup sestavení Shannon-Fanova stromu.

uvedeném příkladu činí 2,5 bitu na symbol.

$$\overline{L(X)} = \sum_{x \in S} p_x \cdot L(x) \tag{4.5}$$

4.3.2 Huffmanovo kódování

Huffmanovo kódování je jedním z nejznámějších algoritmů v oblasti statistických metod komprese dat. Jeho rozšíření je způsobeno relativně jednoduchým principem a skutečností, že generuje nejkratší prefixový kód ze všech statistických metod, které se zabývají kódováním jednotlivých znaků.

Huffmanovo kódování je v určitých ohledech podobné již dříve uvedenému Shannon-Fanovu kódování. Huffmanova metoda, stejně jako Shannon-Fanova, generuje optimální kód, pokud mají symboly vstupní abecedy pravděpodobnost výskytu v kódované zprávě rovnu záporné mocnině čísla dvě. V takovém případě totiž entropie jednotlivých symbolů dosahuje celočíselnou hodnotu, která je rovna délce odpovídajícího Huffmanova kódu.

Hlavním rozdílem oproti Shannon-Fanovu kódování je ovšem způsob konstrukce kódovacího stromu, a tím i výstupního kódu. Tato metoda vytváří binární strom opačným směrem, tedy zdola nahoru. Ukazuje se, že Huffmanova metoda konstrukce kódu vykazuje obecně lepší výsledky než Shannon-Fanova.

Konstrukce Huffmanova stromu
Pro konstrukci Huffmanova stromu je třeba znát pravděpodobnosti nebo frekvence výskytu jednotlivých znaků. Tuto potřebu je možné řešit předem dohodnutým tvarem stromu, který je znám jak straně kodéru, tak dekodéru. Tento způsob řešení je ale vhodný pouze pro určitou skupinu dat, pro kterou je možné statisticky určit frekvence vyskytujících se symbolů.

Příkladem může být komprese zdrojových textů, které obsahují převážně sadu klíčových slov.

Univerzálnějším způsobem je zvolit dvouprůchodové zpracování vstupní zprávy, kdy jsou při prvním průchodu vypočítány frekvence výskytu jednotlivých symbolů vstupní abecedy. Na druhou stranu je tento způsob zatížen nutností přenést informace pro vytvoření stromu i na stranu dekodéru, kde pak podle něj probíhá dekomprese.

Samotná konstrukce stromu probíhá směrem od listů ke kořeni. První vytvořený list reprezentuje symbol s nejnižší frekvencí výskytu v dané zprávě. Dalším krokem je vytvoření jeho rodičovského uzlu, ke kterému bude následně připojen listový uzel reprezentující symbol s nejnižší výskytovou frekvencí z množiny dosud nezpracovaných symbolů. Takto vzniklý rodičovský uzel získává ohodnocení rovné součtu obou jeho synovských uzlů.

Stejným způsobem pokračuje konstrukce Huffmanova stromu až do chvíle, kdy dojde ke zpracování všech symbolů vstupní abecedy a vytvoření kořenového uzlu, který má celkové ohodnocení rovno jedné. Tento postup je formálně popsán algoritmem 2 a jeho použití je ukázáno na obrázku 4.6.

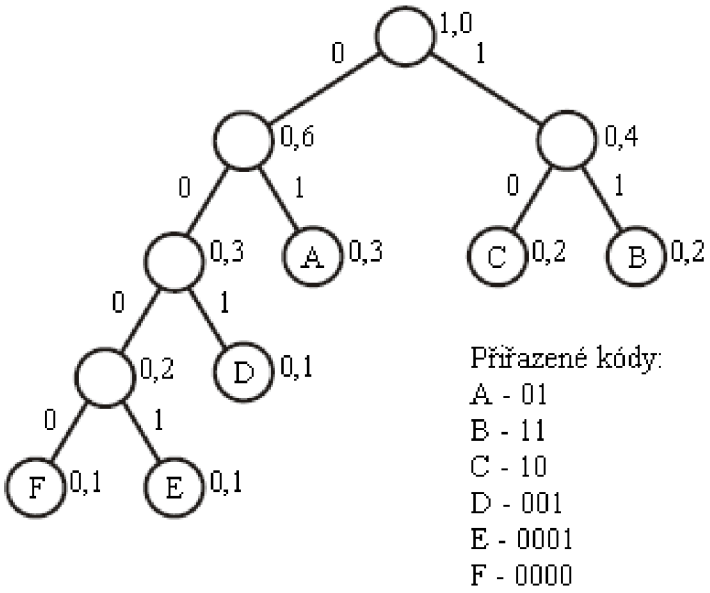
Algoritmus 2: Huffmanovo kódování

Input: Symboly
Output: Binární kódy uzlů jednotlivých symbolů

1: *spočítej frekvence výskytu symbolů vstupní abecedy*
2: *seřaď symboly dle četnosti výskytu* **C** *do neklesající posloupnosti*
3: **while** *existuje nezpracovaný vstupní symbol* **do**
4: *načti_symbol(***symbol***)*
5: **U** = *vytvoř_uzel(***symbol***,* **C***,* **levý_syn** = *NULL,* **pravý_syn** = *NULL)*
6: *vlož uzel* **U** *do fronty* **F**
7: **end while**
8: **while** *nedošlo k vytvoření kořenového uzlu* **do**
9: *z čela fronty vyber dva uzly* **U1**, **U2** *s četnostmi* **C1** *a* **C2**
10: *vytvoř_uzel(***uzel ohodnocený součtem četností C1 a C2***)*
11: *tomuto rodičovskému uzlu přiřaď za následníky uzly* **U1** *a* **U2**
12: *zatřiď nový uzel zpět do fronty* **F**
13: **end while**
14: *ohodnoť cesty od kořene ke všem listům*
15: **while** *dokud existuje nezpracovaný vstupní* **symbol do**
16: *načti_symbol(***symbol***)*
17: *tiskni_výstup(***kód listu reprezentujího symbol***)*
18: **end while**

Kód vyprodukovaný Huffmanovou metodou nemusí být ve všech případech unikátní. Tato metoda totiž určuje způsob konstrukce stromu pouze na základě pravděpodobností výskytu jednotlivých symbolů. Pokud má tedy více symbolů vstupní abecedy stejnou pravděpodobnost výskytu, pak můžeme v daném kroku konstrukce Huffmanova stromu použít libovolný z nich. Tato arbitrální rozhodnutí při konstrukci Huffmanova stromu ovlivňují podobu jednotlivých kódů. Vzhledem k tomu, že zaměnitelné symboly mají stejnou frekvenci výskytu, tak tato rozhodnutí neovlivňují průměrnou délku kódu. Tento fakt je viditelný i ze vztahu 4.5. Při volbě z možných variant kódů pro danou množinu symbolů je pak často vhodné volit kód s nejnižší hodnotou rozptylu 4.6. Zejména důležitá je tato volba při

přenosu dat s využíváním vyrovnávací paměti, kterou je pak možné lépe dimenzovat.



Obrázek 4.6: Příklad konstrukce Huffmanova stromu.

$$\sigma^2 = \sum_{i=1}^{n} [x_i - E \cdot (X)]^2 \cdot p_i \tag{4.6}$$

Dekódování Huffmanova kódu
Dekódování sekvence probíhá opět pomocí procházení Huffmanova stromu. Tento binární strom procházíme směrem od kořene a přesun do odpovídajícího synovského uzlu se vždy řídí dle dalšího bitu přijatého kódu. Vzhledem k tomu, že se jedná o prefixový kód, můžeme okamžitě po dosažení listového uzlu zapsat odpovídající symbol na výstup dekodéru.

4.3.3 Adaptivní modifikace Huffmanova kódování

Tato modifikace umožňuje jednoprůchodové zpracování vstupních dat i bez předem dohodnutého tvaru Huffmanova stromu. Dochází zde totiž k dynamickému sestavování Huffmanova stromu v závislosti na aktuálně načítaných a následně kódovaných symbolech. Tato modifikace je jak časově, tak implementačně náročnější než statická varianta, na druhou stranu zde odpadá nutnost ukládat vytvořený strom. K jeho opětovnému sestrojení totiž dojde i na straně dekompresoru.

Konstrukce výstupního kódu
Při začátku komprese je Huffmanův strom prázdný. Prvním zpracováním vstupního symbolu dojde k jeho zapsání na výstup kodéru beze změny – tedy bez komprese. Současně však dojde k vytvoření odpovídajícího listového uzlu Huffmanova stromu, čímž je danému symbolu přidělen kód.

Jakmile dojde k opětovnému výskytu symbolu, který se již nachází v Huffmanově stromu, je na výstup zapsán jeho aktuální přidělený kód. Také dojde k zaznamenání tohoto výskytu, to znamená ke zvýšení frekvence výskytu právě kódovaného symbolu. Vzhledem k tomu, že i při adaptivní modifikaci musí být zachována platnost pravidel pro generování Huffmanova kódu, může toto zvýšení frekvence výskytu znamenat požadavek na změnu tvaru Huffmanova stromu. Tento fakt může, dle zvolené implementace, vyvolat buď stavbu nového stromu, případně přeuspořádání stromu stávajícího.

Z možnosti změny tvaru Huffmanova stromu plyne nejen zvýšená časová náročnost metody, ale také skutečnost, že kód každého symbolu se může v čase měnit.

Algoritmus 3: Adaptivní Huffmanovo kódování

Input: Symboly
Output: Binární kódy jednotlivých symbolů

```
1:  vytvoř uzel escape
2:  while existuje nezpracovaný symbol, pak opakuj do
3:      načti_symbol(symbol)
4:      if jde o první načtení symbol then
5:          tiskni_výstup(kód listu escape)
6:          tiskni_výstup(symbol)
7:          vytvoř nový uzel U s následníky escape a symbol
8:          aktualizuj_strom
9:      else
10:         tiskni_výstup(kód listu reprezentujícího symbol )
11:         aktualizuj_strom
12:     end if
13: end while
```

Dekomprese adaptivního Huffmanova kódování

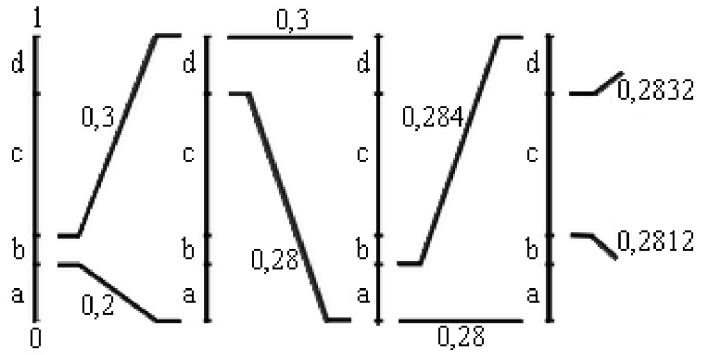
Při dekompresi dochází ke stavbě Huffmanova stromu stejným způsobem, jako se tomu děje na straně kompresoru. Je tak překonáván i problém proměnného kódu jednotlivých symbolů. I když se v čase může kód měnit, je zajištěno, že v daném kroku dekomprese je používán Huffmanův strom ve stejném tvaru, v jakém byl za tohoto kroku na straně kompresoru. Dekompresor však musí mít informaci o tom, zda aktuálně načtená data představují Huffmanův kód symbolu, či se jedná o symbol vstupní abecedy v nekomprimované podobě. Tato situace se často řeší zavedením tzv. *escape* sekvence. Z tohoto důvodu je k Huffmanovu stromu přidán další list. Jedná se o list s nulovou frekvencí výskytu a představuje právě tento *escape* symbol. Jak ukazuje algoritmus 3, bude hodnota escape sekvence vždy předcházet nekomprimované podobě symbolu.

4.3.4 Aritmetické kódování

Aritmetické kódování je dvouprůchodovou statistickou metodou komprese dat. Předchozí uvedené statistické metody pracovaly na principu odděleného kódování jednotlivých symbolů zprávy. Tím vznikala nutnost jednotlivým symbolům přiřazovat kódové sekvence o celočíselné délce, čímž mohlo docházet ke vzniku redundance.

Vznik této nežádoucí redundance je možné částečně omezit použitím aritmetického kódování. Aritmetické kódování pracuje, stejně jako Huffmanova metoda, nad statistickým modelem kódovaných dat, který si vytváří při jejich prvním průchodu.

Tato metoda však volí jiný přístup, kdy nekóduje jednotlivé symboly, ale používá jeden kód pro zakódování celé zprávy. Principem aritmetického kódování je rozdělení intervalu $\langle 0, 1)$ na n disjunktních intervalů, kde n je určeno počtem symbolů vstupní abecedy. Velikosti jednotlivých intervalů jsou přímo úměrné frekvenci výskytů symbolů.



Obrázek 4.7: Princip specifikace intervalu u aritmetického kódování.

Metoda startuje s celým intervalem $\langle 0, 1)$ a postupně jej specifikuje dle četnosti výskytu aktuálně zpracovávaného symbolu. Vyjádření zúženého intervalu vyžaduje další bity desetinného čísla. Délka generovaného čísla tedy s upřesňováním intervalu průběžně roste.

Základní princip tohoto přístupu ke kompresi je založen na myšlence, že symbol s vyšší frekvencí výskytu specifikuje výsledný interval méně, a tak i délka generovaného kódu roste v menší míře. Naproti tomu zakódování symbolu s malou frekvencí výskytu způsobí výraznější specifikaci intervalu a razantnější zvětšení délky výstupního kódu.

Konstrukce výstupního kódu

V každém kompresním kroku dojde k rozdělení aktuálně specifikovaného intervalu podle frekvence výskytů symbolů. Následuje přidělení podintervalů o poměrných šířkách jednotlivým symbolům a následná specifikace intervalu. Interval zvolený pro specifikaci, je vybrán vždy dle příslušnosti načteného symbolu do některého z podintervalů. Obdobným způsobem dochází k rekurzivnímu dělení intervalu na disjunktní podintervaly, dokud nedojde k zakódování celé vstupní zprávy.

Výsledkem kódování je pak libovolné číslo náležící do takto vzniklého intervalu. Vzhledem k této neurčitosti je u aritmetického kódování třeba řešit problém předání informace dekodéru o skutečnosti, kdy je možné přestat interval dělit.

Existuje poměrně intuitivní řešení tohoto problému, jímž je zavedení speciálního ukončovacího symbolu. Vzhledem k tomu, že se tento symbol vyskytne ve zprávě právě jednou, pak bude mít nejnižší pravděpodobnost výskytu a neovlivní významným způsobem pravdě-

podobnosti výskytu ostatních symbolů. Další variantou řešení je využití prefixového kódu, s jehož pomocí by byla dekodéru předána informace o počtu symbolů k dekompresi.

Algoritmus 4: Aritmetické kódování - komprese

Input: Symboly
Output: Číslo v intervalu $\langle 0, 1)$

```
1:  spočítej frekvence výskytu symbolů vstupní abecedy
2:  interval I= ⟨0, 1)
3:  rozděl I podle poměrů frekvencí výskytu jednotlivých symbolů
4:  načti_symbol(symbol)
5:  while symbol není ukončovacím symbolem do
6:      I = podinterval I, kam náleží symbol
7:      rozděl I podle počtu početnosti jednotek
8:      načti_symbol(symbol)
9:  end while
10: tiskni_výstup(číslo z intervalu I)
```

Dekomprese aritmetického kódu

Dekódování pracuje způsobem, kdy z obdrženého kódu postupně určuje zakódované symboly. To je možné díky známému přiřazení symbolů k vzájemně disjunktním podintervalům. Jakmile dojde k určení symbolu, je odstraněn jeho vliv na zbývající část aritmetického kódu. Tento proces je rekurzivně opakován až do chvíle, kdy dojde k identifikaci speciálního ukončovacího symbolu, případně do navršení počtu symbolů (hodnota předána prefixovým kódem).

4.4 Slovníkové metody

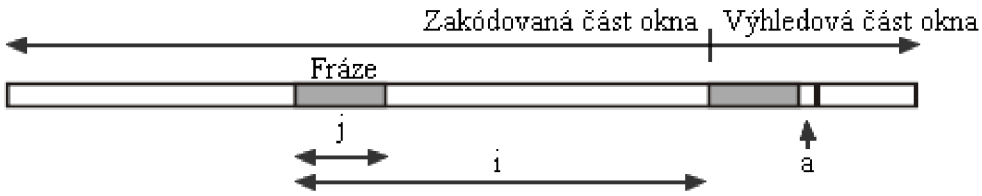
Metody, které řadíme mezi slovníkové, generují výstupní kód fixní délky. Základní myšlenkou slovníkového přístupu ke kompresi je vyhledávání opakujících se řetězců vstupních symbolů a následné překódování tohoto řetězce do podoby výstupního odkazu za použití slovníku. Při kompresi se na výstup zapisuje pouze první výskyt dané sekvence symbolů, která je ve slovníkových metodách označovaná jako fráze. Další výskyty této fráze se nahradí odkazem.

Zde existují dva principy řešení tvorby odkazu. První třída algoritmů nahrazuje výskyt fráze odkazem na místo v textu, kde již byla dříve použita. Odlišný přístup pak využívá třída algoritmů, které udržují v paměti slovník a další výskyty frází nahrazují odkazy do svého slovníku, nikoli na místo v textu.

Teoretickou hranicí komprese, kterou nabízejí slovníkové metody, je $n \cdot H$ bitů, kde n je počtem symbolů v řetězci a H je entropií daného řetězce. Principiálně lze tedy slovníkové metody označit za entropické kódování. Tato skutečnost ovšem platí pouze pro velké vstupní soubory.

4.4.1 LZ-77

Princip této metody spočívá ve využívání části dříve zakódovaného textu jako slovníku. Metoda LZ–77 je někdy také zvána metodou klouzavého okna. Toto okno je rozděleno na dvě části, kde první část je tvořena již zakódovanými frázemi. Ty jsou dále využívány jako kódovací slovník. Druhou část okna, v anglické literatuře zvanou $look-aheadbuffer$, tvoří část dosud nezakódované vstupní zprávy.



Obrázek 4.8: Ukázka rozdělení okna u metody LZ–77.

V praxi jsou velikosti těchto částí okna nesymetrické a značně převládá velikost kódovacího slovníku oproti části obsahující nezakódované sekvence symbolů. Tato skutečnost je poměrně logická, neboť je výhodné mít obsáhlejší slovník a tedy i větší šanci na úspěšné zakódování, byť za cenu omezenějšího výhledu do nezakódované části zprávy.

Konstrukce výstupního kódu

Při zakódovávání vstupní sekvence symbolů prochází kodér slovníkovou část okna směrem zprava doleva a vyhledává nejdelší posloupnost znaků, kterou lze úspěšně zakódovat sekvenci znaků nacházejících se na začátku výhledové části okna.

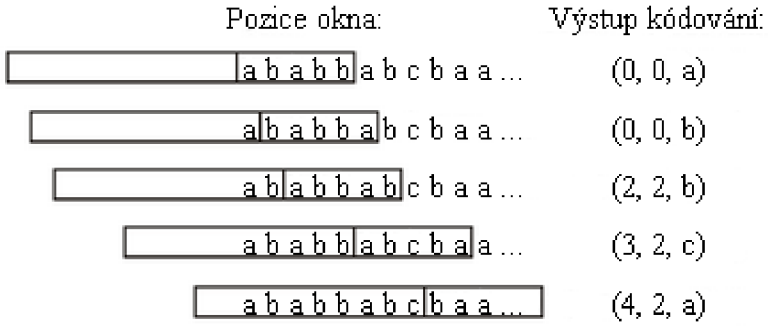
Na výstup kodéru je v každém kroku zapsána trojice (i, j, a), kde i představuje vzdálenost prvního znaku nalezené kódovací fráze od hranice mezi slovníkovou a výhledovou částí okna. Dále pak j značí délku shody obou frází a a je symbolem, který se nachází za kódovanou sekvencí znaků ve výhledové části okna. Po vygenerování trojice (i, j, a) je celé okno posunuto o $j + 1$ znaků vpravo. Postup je popsán algoritmem 5 a jako příklad kódování slouží obrázek 4.9.

Algoritmus 5: LZ–77

Input: Řetězce symbolů
Output: Trojice (i, j, a)

1: *naplň nezakódovanou část okna ze vstupu*
2: **while** *existuje nezakódovaný vstupní* **symbol do**
3: *najdi ve slov. části okna nejdelší frázi* **f** *pro aktuální vstup*
4: **i** = *pozice fráze* **f** *v okně*
5: **j** = *délka fráze* **f**
6: **a** = *první znak za aktuálně zpracovaným vstupem*
7: *tiskni_výstup(***i,j,a***)*
8: *posuň okno doprava o* **j+1** *znaků*
9: **end while**

Pokud je vhodných kódovacích frází nalezeno více, pak je vybrána ta z nich, která disponuje nejdelší shodnou částí s kódovaným řetězcem. V situaci, kdy je nalezeno několik takových frází se stejnou délkou, je výběr z nich implementačně závislý. Principiálně vhodnějším řešením je zvolení fráze, která je blíže výhledové části okna, neboť bude k jejímu popsání třeba nižšího čísla j. Na druhou stranu je v některých implementacích vybrána poslední taková nalezená fráze. Je to způsobeno zjednodušením implementace, kdy si stačí pamatovat informace vždy jen o poslední nalezené frázi. Pokud nedojde k nalezení vhodné fráze ve slovníku, pak je na výstup zapsána trojice, která je tvořena informacemi o nulovém posunu, nulové délce a nese také nezakódovaný symbol.



Obrázek 4.9: Postup kódování LZ–77.

Dekomprese kódu LZ–77

Dekomprese je velmi snadná a rychlá, což řadí tuto metodu mezi asymetrické kompresní algoritmy. Opět je zde využíváno principu klouzavého okna, kam dekodér přistupuje na základě indexu získaného z výše definované trojice (i, j, a). Po jejím načtení je schopen získat použitou kódovací frázi bez nutnosti procházet celou slovníkovou část. Následně provede odpovídající kopírování ze slovníku na výstup a přidání znaku a.

4.4.2 LZ–78

Metoda LZ–78 patří do kategorie metod, které nevyužívají posuvného okna. Důsledkem je skutečnost, že místo odkazování na předchozí pozici v textu využívá v paměti postupně vybudovaného slovníku. Tento slovník dává metodě LZ–78 oproti výše uvedené LZ–77 výhodu v tom, že je možné ke kompresi využít všech dříve použitých frází, kdežto u posuvného okna LZ–77 jejich počet závisí na zvolené velikosti slovníkové části okna. Současně je tento fakt i nevýhodou, neboť takto tvořený a doplňovaný slovník má tendenci k nárůstu a metoda tedy vykazuje nezanedbatelné paměťové nároky.

Metoda LZ–78 produkuje v každém kroku jednu dvojici (i, a), kde i je index fráze ve slovníku a a je, stejně jako u metody LZ–77, symbol nacházející se za právě ukládanou frází. Oproti metodě LZ–77 odpadá nutnost udávat délku fráze, neboť každá fráze je uložena ve slovníku samostatně.

Konstrukce výstupního kódu
Na počátku komprese je slovník prázdný, respektive obsahuje frázi nulové délky s přiřazeným číslem nula. Tohoto záznamu ve slovníku je využíváno při kódování vstupu, který nemůže být zakódován pomocí žádné dosud použité fráze.

V každém kompresním kroku se ve slovníku vyhledává nejdelší fráze, pomocí níž lze zakódovat aktuálně zpracovávaný vstup. Po zakódování dvojice (i, a) je do slovníku přidána nová fráze, která je tvořena frází aktuálně použitou v tomto kroku doplněnou o symbol a. Pakliže nebyla nalezena vhodná fráze, pak je do slovníku uložen zpracovávaný symbol a na výstup kodéru je generována dvojice, kde je číslo fráze rovno nule.

Krok:	Nezakódovaný vstup:	Slovník:	Kódování:
0			
1	a b a b b a b c b a a	a	(0, a)
2	b a b b a b c b a a	b	(0, b)
3	a b b a b c b a a	a b	(1, b)
4	b a b c b a a	b a	(2, a)
5	b c b a a	b c	(2, c)
6	b a a	b a a	(4, a)

Obrázek 4.10: Příklad kódování LZ–78, v kroku 0 je slovník prázdný.

Dekomprese kódu LZ–78
Dekomprese je méně časově náročnou než komprese, stejně jako u metody LZ-77. Zpětná rekonstrukce komprimované zprávy je možná za pomoci opětovného vytvoření dekódovacího slovníku. Tento slovník na počátku dekomprese obsahuje pouze prázdný řetězec na pozici nula. Postupným načítáním dvojic $(i,\ a)$, dochází k budování dekódovacího slovníku a generování odpovídajícího výstupu.

4.4.3 LZW

Metodu LZW lze zařadit mezi semiadaptivní slovníkové metody a její návrh vychází z výše představené metody LZ–78. Její výstup je však tvořen pouze indexem fráze nalezené ve slovníku. Tato modifikace vychází z myšlenky, že kódování znaku za frází je neefektivní. Dochází tak pouze ke kódování sekvencí vstupních symbolů za pomocí frází již obsažených ve slovníku.

Tato myšlenka ovšem přináší oproti metodě LZ–78 nutnost vždy nalézt ve slovníku alespoň jednu frázi využitelnou k zakódování aktuálně zpracovávaného vstupu. Z tohoto důvodu je slovník metody LZW na počátku kódování inicializován všemi symboly vstupní abecedy. Ačkoli nepatří metoda LZW mezi nejúčinnější, je poměrně rozšířena.

Konstrukce výstupního kódu
Díky inicializovanému slovníku je v každém kompresním kroku nalezena kódovací fráze. Výstup kodéru je pak číslem této fráze, které značí její pořadí ve slovníku. Při každém kroku, vyjma prvního, je také slovník doplněn o novou frázi. K vytvoření nové fráze dochází pomocí konkatenace naposledy využité fráze a aktuálně zpracovávaného symbolu.

Algoritmus 6: LZW - komprese

Input: Řetězce symbolů
Output: Číslo fráze

```
1:  inicializuj kódovací_slovník
2:  while existuje nezpracovaný vstupní symbol do
3:      načti_symbol(symbol)
4:      if (fráze+symbol) existuje v kódovacím_slovníku then
5:          fráze += symbol
6:      else
7:          tiskni_výstup(číslo fráze v kódovacím_slovníku)
8:          přidej (fráze+symbol) do kódovacího_slovníku
9:          fráze = symbol
10:     end if
11: end while
```

Dekomprese kódu LZW
Pro dekompresi je nutné inicializovat překladový slovník frázemi tvořenými všemi symboly vstupní abecedy, stejně jako je tomu u komprese. Naopak není třeba předávat vytvořený slovník, neboť ho lze v průběhu samotné dekomprese rekonstruovat. Důležitým faktem je, že vytváření frází ve slovníku je vzhledem ke kompresní části o krok zpožděno. Tato skutečnost může mít za následek, že v daném kroku dekomprese ještě nemusí být požadovaná fráze ve slovníku. Pokud tato situace nastane, pak je první znak fráze shodný s prvním znakem fráze použité v předchozím kroku. Na základě tohoto poznatku lze frázi do slovníku doplnit a je tak možné pokračování v dekompresi.

Algoritmus 7: LZW - dekomprese

Input: Číslo fráze
Output: Řetězce symbolů

```
1:  inicializuj kódovací_slovník
2:  předchozí_fráze = ""
3:  while existuje nezpracovaný kód do
4:      načti_kód(kód)
5:      if existuje fráze ve slovníku s kódem kód then
6:          fráze = získej frázi s odpovídajícím kódem kód
7:          tiskni_výstup(fráze)
8:          přidej_do_slovníku_frázi(předchozí fráze + první znak z fráze)
9:      else
10:         if kód ⟩ maximální index slovníku + 1 then
11:             ukonči dekompresi
12:         else
13:             fráze = předchozí_fráze + první znak z předchozí_fráze
14:             tiskni_výstup(fráze)
15:             přidej_do_slovníku_frázi(fráze
16:         end if
17:         předchozí_fráze = fráze
18:     end if
19: end while
```

Kapitola 5

Návrh komprese medicínských dat

Obrazová medicínská data jsou ve formátu *DICOM* tvořena 12–ti bitovými celočíselnými hodnotami, které reprezentují pixel ve stupních šedi. I když se k zakódování hodnoty pixelu reálně využívá jen 12 bitů, jsou kvůli zarovnávání v paměti ukládány jako 16–ti bitové hodnoty.

Za jednu z vlastností medicínských dat je možné označit časté opakování stejných, či podobných hodnot pixelů, které spolu sousedí. Je to dáno povahou lidského těla a metodami snímání, kdy jsou stejné tkáně zaznamenány pomocí podobných hodnot pixelů. Samozřejmě dochází k drobným odlišnostem a tvoří se tak regiony pixelů s podobným, ne však nutně stejným ohodnocením. Naopak výrazné změny v hodnotách sousedních pixelů nastávají v oblastech snímku, kde došlo k zachycení změn povahy tkáně v lidském těle. Typickým představitelem popisovaného faktu je snímkování regionu lidského těla, kde se setkávají měkké a kostní tkáně. Další znalosti o vlastnostech medicínských dat lze získat z [4], kde jsou popsány technologie snímání. Příklad podoby snímku je možné vidět na obrázku 5.1, kde se nachází snímek lidského mozku, pořízený pomocí počítačové tomografie.

Obrázek 5.1: Ukázka snímku z CT ve formátu *DICOM*.

5.1 Sestavení vícestupňové komprese

V následujících částech této kapitoly představím svůj návrh rozložení kompresních metod do jednotlivých stupňů komprese. Při tomto návrhu jsem vycházel z výše uvedených faktů známých o medicínských obrazových datech a kladl jsem důraz na skutečnost, aby každý následující stupeň využil tvaru výstupu předchozího stupně tak, aby byl výsledný kompresní poměr co nejvyšší.

5.2 Snížení rozsahu hodnot

V prvním stupni komprese jsem navrhl využít delta kódování, kdy dojde k uložení pouze rozdílu mezi sousedícími pixely. Delta kódování bude tedy mít dva významné efekty, kdy ve většině případů dojde ke snížení ukládaných hodnot, a také bude docházet ke generování sekvencí nul v místech, kde se opakovala hodnota předchozího pixelu.

5.3 Transformace před kompresí

Navazujícím krokem navrhované komprese je rozdělení 16–ti bitových hodnot na dvojice osmibitových, kdy budou nejprve odeslány všechny nižší byty a poté všechny vyšší byty. Tato transformace bude mít za následek zvýšení pravděpodobnosti opakující se hodnoty symbolu. K tomuto jevu dochází ve chvíli, kdy jsou sousední pixely jen mírně odlišné, a to tak, aby se odlišnost projevila jen v nižším bytu a vyšší byte zůstane neovlivněn. V takových případech pak vznikají posloupnosti nulových hodnot v oblasti uložení vyšších bytů. Výše uvedené vlastnosti se dá úspěšně využít v dalších stupních komprese.

5.4 Omezení vzniklých sekvencí

Vzhledem k předchozímu faktu se bude následující stupeň pokoušet omezovat generované sekvence stejných hodnot. Tradiční přístup by volil metodu RLE, ovšem v obrazových medicínských datech nelze příliš často očekávat výskyt gradientu. Tudíž nelze ani očekávat delší sekvence obecných hodnot po průchodu delta kódováním. Výjimku tvoří sekvence nul, které vznikají v oblastech se stejnou hodnotou sousedních pixelů. Proto bude v tomto stupni komprese pravděpodobně výhodnější použití metody eliminace nul.

5.5 Získání a zakódování shluků symbolů

Výhoda omezení sekvencí stejných hodnot se projeví i v dalším stupni komprese, kde dojde k Burrows–Wheelerově transformaci vstupních dat. Výhoda spočívá v rychlejším seřazení vstupních bloků, ke kterému dojde bez dlouhých sekvencí stejných symbolů rychleji. Výstupem Burrows–Wheelerovy transformace bude řetězec o stejné délce, ovšem dojde ke zvýšení pravděpodobnosti výskytu shluků stejných symbolů.

V praxi velmi používanou metodou je kombinace BWT s Move-to-Front, kdy jsou výstupem pouze indexy symbolů, nikoliv symboly samotné (vizte princip MTF v kapitole 4.1.4). Alternativním přístupem může být stav, kdy dojde k nahrazení MTF pomocí RLE, neboť zde již lze předpokládat shluky obecných symbolů, nikoli pouze nul.

5.6 Finální stupeň komprese

V posledním stupni komprese je vhodné zařadit výkonný kompresní algoritmus, který bude hlavním zdrojem kompresního poměru výsledné vícestupňové komprese. V předchozích stupních nedocházelo vždy k přímé kompresi, neboť bylo některých stupňů využito čistě k transformaci dat do podoby vhodnější k následné kompresi.

Jedním z možných řešení je zařazení statistického kompresního algoritmu, v tomto případě se bude jednat o aritmetické kódování. Jako jednu z alternativních cest je pak možné zařadit na místo tohoto stupně slovníkový algoritmus, případně kombinaci obou výše zmíněných možností, kdy je možné zakódovat výstup slovníkového algoritmu pomocí aritmetického kódování.

Kapitola 6

Implementace zvolených metod

V následující kapitole popisuji prostředky, které byly použity k implementaci praktické části této práce a dále pak důležité implementační detaily, které je nutné brát v potaz u jednotlivých kompresních metod.

Pro implementaci všech kompresních metod jsem zvolil jazyk C++, který umožňuje objektově orientovaný přístup, jak je specifikováno v zadání. Současně je využíváno prostředků MDSTk, především pak pro získávání čistě obrazových dat ze souboru ve formátu *DICOM* [17]. Informace o netriviálních implementačních problémech, které je nutné řešit v rámci jednotlivých metod, jsem čerpal z [3] a [8].

6.1 Vlastnosti společné všem modulům

Jednotlivé kompresní metody jsou umístěny do samostatných modulů. K tomuto rozdělení jsem přistoupil jednak z důvodu přehlednosti a udržovatelnosti kódu, ale také pro usnadnění následného využívání, kdy je možné jednotlivé moduly libovolně spojovat pomocí nepojmenovaných rour.

Obecnou vlastností všech modulů je režim, ve kterém pracují. Snahou bylo umožnit jejich využívání jako standardní filtry v Linuxu. Proto daným způsobem zpracovávají data ze vstupu a jejich výsledný tvar zapisují na svůj standardní výstup. Pokud uživatel přesto chce předat danému modulu vstupní či výstupní soubor ve formě parametru, dojde k internímu znovuotevření tohoto souboru jako popisovače standardního vstupu či výstupu. Tento přístup zajišťuje jednotnost práce se soubory v rámci všech modulů a současně nechává uživateli možnost zvolit pro něj v dané chvíli výhodnější variantu.

6.2 Získání obrazových dat

K získání obrazových dat ze souboru ve formátu *DICOM* jsem využil modul *mdsLoadDicom*, který patří mezi nástroje šířené v prostředí *MDSTk*. Modul očekává na vstupu data ve formátu *DICOM* a při použití přepínače *-raw* dojde k postupnému průchodu všech pixelů a zapsání jejich šestnáctibitových hodnot v nezměněné podobě na výstup. Při tomto procesu dochází k potlačení veškerých hlaviček, které soubor *DICOM* obsahuje [17] a jsou tak získána čistě obrazová data, se kterými je dále nakládáno.

6.3 Delta kódování

Využití delta kódování se primárně předpokládá v úvodní fázi komprese. Z tohoto důvodu jsem se jej rozhodl implementovat v jeho šestnáctibitové podobě, aby odpovídalo šestnáctibitové reprezentaci jednotlivých pixelů. Při implementaci byl využit řádkový prediktor, který pracuje na principu výpočtu rozdílu hodnot sousedních pixelů. Jako výsledek je tedy ukládán vždy jen jejich rozdíl.

6.4 Rozdělení na vyšší a nižší byty

Při rozdělování šestnáctibitových dat na dvojici osmibitových hodnot dochází k přímému načítání šestnáctibitových hodnot ze vstupu a postupnému zápisu dvojnásobného počtu osmibitových dat na výstup modulu.

Získání nižšího bytu je docíleno pomocí maskování a k zapsání jeho hodnoty do výstupního proudu dochází přímo při jeho zpracování. Pomocí maskování a využití bitového posuvu je i hodnota vyššího bytu převedena do osmibitové podoby, která je následně uložena do standardního vektoru C++[13], který zde funguje jako buffer. K vyprázdnění tohoto bufferu dochází až po zpracování celého vstupu. Důsledkem tedy je zapsání nejprve všech nižších bytů a následně všech vyšších bytů na výstup modulu.

6.5 RLE a metoda eliminace nul

Vzhledem k podobnosti metody eliminace nul a RLE jsem pro implementaci zvolil společný modul, jehož účel je určen parametrem již při překladu. Podmíněný překlad jsem zvolil proto, aby nebylo nutné testovat zadaný parametr za běhu programu, čímž by mělo dojít ke zrychlení metody.

Pro zvýšení kompresního poměru obou metod jsem zvolil implementační heuristiku, kdy dochází k vytváření kódové dvojice až při nalezení sekvence stejných symbolů v délce alespoň dvou symbolů. Je tím omezeno generování kódové dvojice v případech, kdy by docházelo k nulové kompresi, případně až k expanzi souboru.

6.6 BWT

V kapitole 4.2.1 jsem popsal teoretický postup Burrows-Wheelerovy transformace, která zpracovává bloky dat o n prvcích a u jednotlivých bloků dochází k vytváření matice o velikosti $n \times n$. V reálné implementaci by ovšem bylo vytváření takto rozlehlé matice v paměti velmi neefektivní a je nahrazeno postupnou rotací řetězce vždy o jeden znak. Díky tomu je dosaženo požadovaného efektu, kterým je vytvoření všech permutací vstupního řetězce, s využitím paměťového místa n–krát menšího, než tomu bylo v původním návrhu metody.

Při zápisu jednotlivých výstupních bloků dochází nejen k zápisu samotných dat, ale je nutné je doplnit o určitou hlavičku. Výstupní data jsou tvořena seřazenými symboly sloupce L (vizte kapitolu 4.2.1). Hlavička pak obsahuje doplňující informace, které budou následně využity při dekompresi. Jedná se o počet bytů v aktuálním bloku, což je informace čistě technického rázu pro dekompresor, ale také je nutné zde doplnit informace nutné k samotné funkčnosti dekompresního algoritmu. Tyto položky, o kterých v kapitole 4.2.1 hovořím jako o dodatečné informaci I, jsou v praxi tvořeny indexem pozice, kterou obsadil první prvek

v seřazeném výstupu a také indexem místa, na kterém se nachází speciální ukončovací prvek bloku.

6.7 MTF

Metoda MTF existuje v několika variantách, které se liší metodikou přeskupování zásobníku. Já jsem zvolil implementaci, kdy dojde vždy k přesunu aktuálně zpracovávaného prvku na vrchol zásobníku. Sleduji tím vyšší množství metodou generovaných nul na výstupu, pokud se budou jednotlivé vstupní symboly opakovat. Tyto sekvence nul budou generovány již od dvou stejných symbolů za sebou.

Naopak postupné přibližování symbolu vrcholu zásobníku, které je praktikováno v odlišných implementacích, by nedávalo při zpracovávání obrazových medicínských dat valný smysl, jelikož by docházelo při stejné situaci ke generování postupně se snižujících hodnot, což nevede na přímočaré využití dalších kompresních technik.

6.8 Aritmetické kódování

Při implementaci aritmetického kódování jsem zvolil jeho adaptivní formu, kdy k aktualizaci datového modelu dojde vždy po zpracování aktuálního symbolu. Výjimku tvoří zakódování posledního symbolu, po jehož zpracování již k aktualizaci modelu nedochází.

Zakódování jednotlivých symbolů pak spočívá v generování bitů kódu dle umístění intervalu, do kterého spadá daný symbol.

6.9 LZ–77 a LZW

Dle doporučení pana docenta Krška jsem metody LZ–77 a LZW neimplementoval od základu sám, ale vycházel jsem ze zdrojových kódů dostupných na [16] a [6]. Tyto moduly nejsou primárně součástí navrhované komprese a slouží především pro porovnání mnou dosažených výsledků s jistou alternativní cestou. Tyto zdrojové kódy jsem upravil do podoby, kdy jsou schopné spolupracovat s ostatními, mnou implementovanými, moduly.

Jak již bylo uvedeno v příslušné kapitole 4.4.1, tak se v praktické implementaci metody LZ–77 často volí nesouměrné rozdělení okna. Ve zde využité implementaci je tento nepoměr 256 : 1, kde zakódovanou část okna tvoří 4096 osmibitových symbolů a aktuálně zpracovávanou část tvoří šestnáct 8–mi bitových symbolů.

Při implementaci metody LZW je zvoleno dvanáctibitové zakódování symbolu. Na tento fakt je navázána i velikost použitého slovníku, kterou je vhodné volit jako prvočíslo vyšší než 2^n, kde n značí počet bitů použitých k zakódování symbolu. V tomto případě je vybráno prvočíslo 5021. Pokud při kompresi dojde k vyčerpání slovníku, tak již nadále nedochází k přidávání dalších kódovacích frází, nicméně proces komprese pokračuje s dosavadním slovníkem dále.

Kapitola 7

Vyhodnocení výsledků

Následující kapitola obsahuje vyhodnocení výsledků kompresních metod aplikovaných na obrazová medicínská data. K reprezentaci těchto výsledků jsem zvolil tabulky obsahující hodnoty kompresního poměru pro extrémní dosažené případy a také průměrnou hodnotu kompresního poměru, která byla na dostupné sadě testovacích dat dosažena. Jako další formu grafické reprezentace výsledků jsem zvolil kvartilové grafy, aby bylo možné dosáhnout ucelenějšího pohledu na danou kompresní techniku. Z těchto grafů je možné vyčíst pro každou metodu oba extrémy a je zde vyznačena i hodnota mediánu pro danou sadu testovacích dat.

V kapitole se věnuji nejprve ověření předpokladů jednotlivých kompresních metod. Dále pak vyhodnocuji testování vícestupňových kompresí, včetně výběru nejvhodnější z nich. V závěru kapitoly poté rozeberu nejlepší a naopak nejhorší případ dosažený navrženou kompresní metodou, včetně důvodu, proč daná situace nastala.

7.1 Sada testovacích dat

Použitá sada testovacích dat obsahuje celkem 189 obrazů ve formátu *DICOM*. Jedná se o kombinaci snímků různých částí lidského těla, které byly získány technologií magnetické rezonance a počítačové tomografie. Všechny testovací soubory jsou dostupné na CD přiloženém k této práci. Jedná se o podmnožinu dat dostupných v anonymizované podobě z rozsáhlých archivů [1], [14] a [15]. Automatizovaném testování je popsáno v příloze B.2.

7.2 Jednostupňová komprese

Tabulka 7.1 ukazuje chování jednotlivých nezřetězených metod na množině testovacích dat. Jak je z této tabulky pozorovatelné, tak metoda rozdělování dat z 16–ti bitových na osmibitové má kompresní poměr rovný právě jedné, tudíž nedochází ke změně objemu dat. Je to dáno jejím principem, kdy vždy dojde k nahrazení jedné 16–ti bitové hodnoty právě jednou dvojicí osmibitových hodnot. Stejného výsledku kompresního poměru je dosaženo i u delta kódování a metody MTF, kde se také počet vstupních dat a jejich velikost nemění. U Burrows–Wheelerovy transformace si lze povšimnout dokonce mírné expanze vstupního souboru ve všech testovaných případech. Je to však očekávané chování, neboť se nejedná o kompresní metodu jako takovou, ale o pouhou transformaci dat do podoby vhodnější pro následnou kompresi. U BWT tedy dochází k transformaci bloků vstupních dat na výstupní bloky o stejné velikosti, které jsou navíc doplněny o údaje nutné ke zpětnému dekódování

(vizte kapitolu 4.2.1). Z důvodu doplňování těchto dodatečných informací dochází k drobné expanzi vstupních dat.

U ostatních metod vidíme jejich přímou účinnost na sadě testovacích obrazových dat. Ovšem je třeba přiznat, že pro řadu z nich se jedná o čistě ukázková data, neboť se očekává jejich zařazení do jiných stupňů komprese, kde bude jejich potenciál lépe využit.

	Dosažený kompresní poměr		
Metoda	**Průměrný**	**Nejlepší**	**Nejhorší**
Aritmetické kódování	1,8970	4,0259	1,3606
BWT	0,9999	0,9999	0,9999
Delta kódování	1,0000	1,0000	1,0000
LZ–77 (lzss)	1,8453	4,1446	1,2808
LZ–78 (lzw)	1,4919	7,6817	0,8902
MTF	1,0000	1,0000	1,0000
RLE	1,0841	1,9242	0,9982
Metoda eliminace nul	0,9395	1,7938	0,6755
16to8	1,0000	1,0000	1,0000

Tabulka 7.1: Přehled účinnosti jednotlivých metod

7.3 Vícestupňové komprese

Při návrhu vícestupňových kompresních metod jsem postupoval s výběrem metod tak, aby každý následující stupeň mohl využít tvaru výstupu stupně předchozího.

Samozřejmě by bylo možné vytvářet velmi rozsáhlé kompresní metody, čítající mnoho stupňů, ovšem časová náročnost takových metod by byla enormní, při nepříliš významném vylepšení kompresního poměru. Proto jsem testoval metody uvedené v tabulce 7.6, které mají omezený počet stupňů. Jejich návrh jsem realizoval na základě získaných poznatků o specifičnosti medicínských dat, což přispělo k vylepšení jejich kompresního poměru.

Vliv postupného zapojování jednotlivých stupňů komprese ukazují tabulky 7.2 až 7.5, kde je použito zkratkovitého názvu kompresních metod. Použitý název je určen dle pořadí aplikace jednotlivých stupňů komprese.

	Dosažený kompresní poměr		
Metoda	**Průměrný**	**Nejlepší**	**Nejhorší**
rle_ari	1,9390	4,0077	1,3593
rle_lzss	1,8894	4,1644	1,2787
rle_lz78	1,4966	7,7164	0,8881
ze_ari	1,8518	3,3707	1,3066
ze_lzss	1,7542	3,9988	1,1235
ze_lz78	1,4642	6,6343	0,8881

Tabulka 7.2: Omezení sekvencí a finální kódování

V tabulce 7.2 vidíme spojení metod pro omezení sekvencí s využitím finálního kódování. Oproti jednostupňové kompresi došlo ke zlepšení kompresního poměru, ovšem například

využití metody LZ–78 v implementaci LZW v nejhorších případech stále selhává.

Dalšího vylepšení kompresního poměru bylo dosaženo přidáním delta kódování jako prvního stupně komprese. Tento fakt je dokumentován tabulkou 7.3. Po přidání delta kódování již došlo k úspěšné kompresi u všech metod i v nejhorších testovaných případech.

	Dosažený kompresní poměr		
Metoda	**Průměrný**	**Nejlepší**	**Nejhorší**
dc_rle_ari	2,4815	9,1378	1,6772
dc_rle_lzss	2,1538	8,6648	1,4199
dc_rle_lz78	2,2131	9,9507	1,3781
dc_ze_ari	2,3219	8,5830	1,5610
dc_ze_lzss	1,9870	8,5804	1,3151
dc_ze_lz78	2,1517	9,7587	1,3533

Tabulka 7.3: Ukázka vlivu použití delta kódování

V situaci, kdy byla jako první stupeň komprese využita transformace dat ze 16–ti bitových na 8–mi bitová (tabulka 7.4) došlo k vylepšení kompresního poměru většiny metod, ale u metody *16to8_ze_lz78* došlo v nejhorším případě k selhání a expanzi souboru. Po demon-

	Dosažený kompresní poměr		
Metoda	**Průměrný**	**Nejlepší**	**Nejhorší**
16to8_rle_ari	2,7690	10,3088	1,8084
16to8_rle_lzss	2,5197	10,0416	1,6665
16to8_rle_lz78	1,9114	8,6691	1,2843
16to8_ze_ari	2,4555	7,6833	1,6512
16to8_ze_lzss	2,3231	6,5436	1,6403
16to8_ze_lz78	1,1536	8,7476	0,8009

Tabulka 7.4: Ukázka vlivu použití transformace 16–ti bitových dat na 8–mi bitová

straci předchozích kroků je logickým krokem předřazení delta kódování a transformace dat ze 16–ti bitových na 8–mi bitová jako počátečních stupňů komprese. Tento případ, který již částečně bere v úvahu vlastnosti medicínských dat, je dokumentován tabulkou 7.5.

	Dosažený kompresní poměr		
Metoda	**Průměrný**	**Nejlepší**	**Nejhorší**
dc_16to8_rle_ari	2,7123	9,6835	1,8632
dc_16to8_rle_lzss	2,2974	8,7934	1,5270
dc_16to8_rle_lz78	1,7861	7,5786	1,1876
dc_16to8_ze_ari	2,7331	9,7268	1,8412
dc_16to8_ze_lzss	2,3332	8,9247	1,5436
dc_16to8_ze_lz78	1,7581	7,6519	1,2106

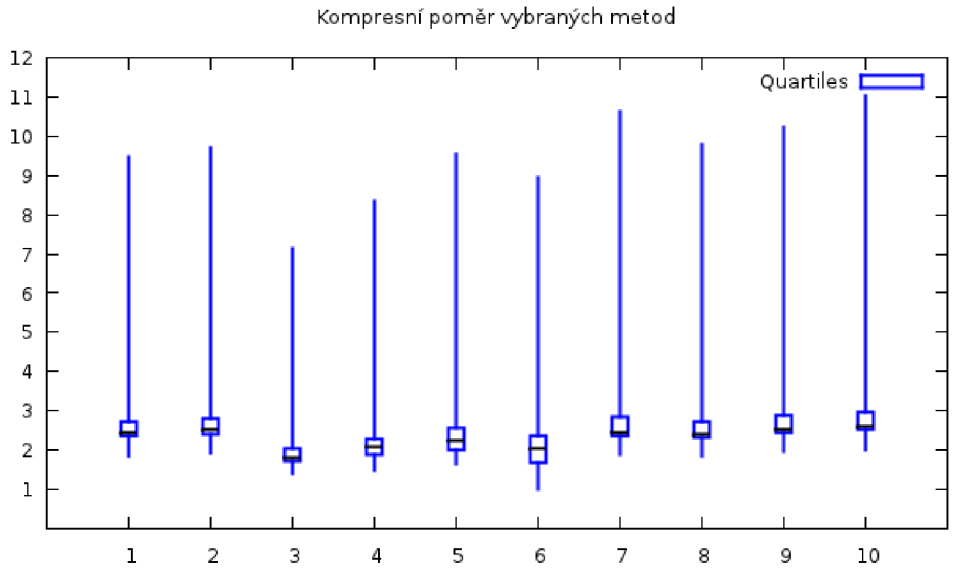
Tabulka 7.5: Ukázka vlivu delta kódování a transformace 16–ti bitových dat na 8–mi bitová

V další části testování došlo k ověření metod navržených v kapitole 5, jmenovitě se jedná o metody *dc_16to8_rle_bwt_ze_ari*, *dc_16to8_ze_bwt_ze_ari* a *dc_16to8_ze_bwt_rle_ari*. Výsledky těchto metod jsou uvedeny v třetí části tabulky 7.6 a dle předpokladu patří k nejúspěšnějším testovaným metodám. Pro srovnání jsou v prvních dvou částech tabulky 7.6 i další možné kompresní metody.

Metoda	**Dosažený kompresní poměr**		
	Průměrný	**Nejlepší**	**Nejhorší**
dc_16to8_rle_bwt_ari	2,7250	9,4825	1,8590
dc_16to8_ze_bwt_ari	2,8175	9,7031	1,9183
dc_16to8_ze_bwt_ari_lz78	2,0274	7,1449	1,3950
dc_16to8_ze_bwt_lzss	2,2357	8,3580	1,4709
dc_16to8_ze_bwt_lz78_ari	2,3442	9,5446	1,6329
dc_16to8_ze_bwt_lz78	1,9703	8,9410	1,0076
dc_16to8_ze_bwt_mtf_ze_ari	2,8036	10,6325	1,8905
dc_16to8_rle_bwt_ze_ari	2,6986	9,7895	1,8497
dc_16to8_ze_bwt_ze_ari	2.8511	10.2317	1.9607
dc_16to8_ze_bwt_rle_ari	2,9256	11,0482	2,0119

Tabulka 7.6: Přehled testovaných komplexních metod

Z výše uvedené tabulky 7.6 a grafu 7.1 plyne porovnání komplexních kompresních metod. Jako nejvhodnější metodu pro kompresi obrazových medicínských dat z formátu *DICOM* jsem zvolil metodu *dc_16to8_ze_bwt_rle_ari*, která dosahuje nejlepších hodnot kompresního poměru na dostupné sadě testovacích dat. Bližší rozbor této metody a jejích vlastností následuje v kapitole 7.4.



Obrázek 7.1: Kvartilový graf kompresního poměru metod z tabulky 7.6 (číslování metod odpovídá jejich pozici v tabulce).

7.4 Výsledná kompresní metoda

Výsledná kompresní metoda *dc_16to8_ze_bwt_rle_ari* odpovídá návrhu uvedeném v kapitole 5 a dosahuje průměrné hodnoty kompresního poměru $2,9256 : 1$. Stejně jako v této hodnotě, dosahuje nejlepších výsledků i v porovnání nejlepších a nejhorších případů komprese s ostatními testovanými metodami.

7.4.1 Princip komprese

Ke snížení rozsahu hodnot a případnému generování nul se v prvním stupni komprese využívá delta kódování. Na něj navazuje stupeň, který překóduje 16–ti bitová data na dvojice 8–mi bitových. Vzhledem ke sníženému rozsahu hodnot se tím zvyšuje pravděpodobnost výskytu nul v oblasti obsahující horní byty. Tohoto faktu využívá třetí stupeň komprese, jímž je metoda eliminace nul. Čtvrtý stupeň je tvořen Burrows–Wheelerovou transformací, jenž má za úkol seskupit bloky dat tak, aby vznikaly shluky stejných hodnot. Tohoto faktu opět využívá pátý stupeň, tvořený metodou RLE, která je schopna právě vytvořené shluky komprimovat. Jako finální zakódování se prosadila metoda aritmetického kódování.

7.4.2 Extrémní případy komprese

Na obrázku 7.2b je uveden příklad lékařských dat, na kterých bylo dosaženo nejlepších výsledků navrženou kompresí, konkrétně byl dosažen kompresní poměr $11,0482 : 1$. Stejně tak na obrazu 7.2a bylo dosaženo velmi dobrého kompresního poměru v hodnotě $9,5531 : 1$. Jak si můžeme povšimnout, oba obrazy mají vhodnou koncepci snímku, která odpovídá předpokladům, podle nichž jsem kompresní techniku navrhoval. Konkrétně zde tedy existují regiony obrazu, které mají stejné ohodnocení pixelů. Také relativně nižší členitost obrazu, tedy počet přechodů mezi druhy tělesné tkáně, tomuto předpokladu napomáhá.

(a) (b) (c)

Obrázek 7.2: Ukázka vhodných (7.2a a 7.2b) a zašuměných (7.2c) testovacích dat.

Naopak obrázek 7.3 představuje protiklad, tedy ukázku obrazových lékařských dat, u nichž nejsou dosaženy očekávané výsledky, konkrétně zde kompresní poměr klesl na hodnotu $2,0119 : 1$. Kvalita komprese tohoto snímku utrpěla jeho poměrně velkou členitostí, zejména ve středu snímku v oblasti plic pacienta.

Dalším případem snímku s kompresním poměrem pod hodnotou průměru této metody je snímek 7.2c. V tomto případě je nižší hodnota kompresního poměru způsobena zejména šumem v pozadí snímku, kde nevznikají potřebně velké regiony se stejnou hodnotou pixelu, což ovlivňuje další stupně komprese. I u tohoto snímku však došlo ke kompresi, a to v poměru $2,0675 : 1$, tudíž ani v nejhorších testovaných případech nedošlo k nežádoucí expanzi dat.

Obrázek 7.3: Ukázka snímku s vysokou členitostí

Kapitola 8

Závěr

Cílem bakalářské práce bylo nalézt vhodný kompresní algoritmus pro obrazová medicínská data. V této práci jsem k problematice komprese medicínských obrazových dat přistoupil z pohledu možnosti využití obecných kompresních algoritmů. Specializace na výše uvedený druh dat jsem dosahoval vhodným výběrem kompresních metod a návrhem vícestupňové komprese, který jsem realizoval na základě získaných poznatků o specifičnosti medicínských dat.

Dosažený průměrný kompresní poměr $2,9256 : 1$ řadí navrženou metodu do účinnější poloviny v tabulce přehledu kompresních metod aplikovaných na medicínská obrazová data [2]. Tato hodnota je srovnatelná s metodou JPEG-SV 5 $(2,92 : 1)$.

V budoucím rozvoji projektu bude vhodné specializovat prediktor použitý v delta kódování takovým způsobem, aby dokázal lépe využít vlastností lidského těla při snímkování. Další možností pro budoucí rozvoj v oblasti komprese medicínských dat spočívá také ve využití vlnkové transformace, která v dnešní době zažívá značný rozvoj a v některých implementacích umožňuje i progresivní škálování ztrátovosti obrazu, který je uživateli poskytnut při dekompresi.

Literatura

[1] BARRÉ, S.: Medical Imaging: Samples. [online], 01-Dec-2003, [cit. 2012-04-29].
URL http://barre.nom.fr/medical/samples/#s-ct

[2] CLUNIE, D. A.: Lossless Compression of Grayscale Medical Images - Effectiveness of Traditional and State of the Art Approaches. [online], [cit. 2012-05-12].
URL http://dclunie.com/papers/spie_mi_2000_compression.pdf

[3] DIPPERSTEIN, M.: Michael Dipperstein's Page O'Stuff. [online], October 4, 2010, [cit. 2012-03-15].
URL http://michael.dipperstein.com/

[4] DRASTICH, A.: *Zobrazovací systémy v lékařství*. Brno: Rektorát VUT v Brně, první vydání, 1990, ISBN 80-214-0220-2, 512 s.

[5] DRÁBEK, V.: *Kódování a komprese dat: KKO*. Brno: Fakulta informačních technologií VUT v Brně, 2008, 236 s.

[6] NELSON, M.: LZW Data Compression. [online], October 1st, 1989, [cit. 2012-03-25].
URL http://marknelson.us/1989/10/01/lzw-data-compression/

[7] PASEKA, J.: *Kódování*. Brno: Ústav matematiky a statistiky Přírodovědecké fakulty MU v Brně, 20011, 149 s.

[8] SALOMON, D.: *Data compresiion: the complete reference*. New York: Springer, třetí vydání, 2004, ISBN 0-387-40697-2, 898 s.

[9] VAJDA, I.: *Teorie informace*. Praha: Vydavatelství ČVUT, 2004, ISBN 80-01-02986-7.

[10] VEČERKA, A.: *Komprese dat*. Olomouc: Katedra informatiky Přírodovědecké fakulty Univerzity Palackého v Olomouci, 2008, 65 s.

[11] VLČEK, K.: *Komprese a kódová zabezpečení v multimediálních komunikacích*. Praha: BEN – technická literatura, druhé vydání, 2004, ISBN 80-7300-134-9, 258 s.

[12] ŽÁRA, J.; BENEŠ, B.; FELKEL, P.: *Moderní počítačová grafika*. Praha: Computer press, první vydání, 1998, ISBN 80-7226-049-9, 448 s.

[13] C++ STL Vector reference.
URL http://www.cplusplus.com/reference/stl/vector/

[14] DICOM sample image sets. [online], [cit. 2012-04-29].
URL http://pubimage.hcuge.ch:8080/DATA/BRAINIX.zip

[15] Die Universität Oldenburg. [online], 16.10.02, [cit. 2012-04-29].
URL ftp://dicom.offis.uni-oldenburg.de/pub/dicom/images/

[16] LZ-77 Implementation. [online], 16-Jul-2008, [cit. 2012-3-23].
URL http://my.execpc.com/~geezer/code/lzss.c

[17] *Digital Imaging and Communications in Medicine (DICOM)*. Rosslyn (Virginia): National Electrical Manufacturers Association, 2011.
URL http://medical.nema.org/standard.html

Příloha A

Obsah CD

CD	Obsahuje text práce a podadresář workspace
workspace	Obsahuje soubor *Makefile* a všechny dále uvedené podadresáře
bin	Obsahuje přeložené spustitelné binární soubory kompresních modulů
doc	Obsahuje soubor *Doxyfile* a je cílovým adresářem generované dokumentace
inputs	Obsahuje vstupní data ve formátu *DICOM*
MDSTk	Obsahuje nástroj MDSTk
outputs	Cílový adresář výstupních komprimovaných souborů
plot	Cílový adresář pro log (jen data) výsledků komprese
raw	Cílový adresář pro vygenerování RAW souborů ze vstupních dat
results	Cílový adresář pro log (s popisem) výsledků komprese
scripts	Obsahuje testovací skripty a konfigurační soubor
src	Obsahuje zdrojové soubory kompresních modulů

Příloha B

Manual

B.1 Ovládání modulů

Ovládání jednotlivých modulů probíhá standardní formou, běžnou u filtrů v Linuxu. Každý modul tedy zpracovává svůj standardní vstup a data zasílá na standardní výstup. Je možné modulu i určit vstupní a výstupní soubor. V takovém případě se první parametr považuje za jméno vstupního souboru a druhý parametr je považován za jméno výstupního souboru.

Ukázka očekávaného využití modulů v konzoli:

upřednostňovaná varianta:

```
workspace/bin$ mdsLoadDicom <../inputs/80.dcm | ./modul1 | ./modul2 >../outputs/80.bin
```

nepraktická varianta:

```
workspace/bin$ mdsLoadDicom <../inputs/80.dcm >../mds/80.bin
workspace/bin$ ./modul1 ../mds/80.bin ../outputs/80.bin
```

B.2 Automatizované testování

Pro testování jednotlivých modulů i celých kaskádových kompresí jsem vytvořil sadu skriptů pro Linuxový shell, která tuto úlohu značně automatizuje. V nižší úrovni obsahuje skripty pro jednotlivé navržené kompresní procesy. Dále je možné spustit skript *main.sh*, který otestuje všechna dostupná data na všech navržených metodách. Tento skript udržuje statistiky výsledků jednotlivých metod na testovacích obrazech. Ještě vyšší možnost automatizace pak umožňuje testování pomocí pravidel v *Makefile* (vizte B.2.1).

Podporované přepínače skriptu *main.sh*	
-v	Vypisuje informace o dosažených hodnotách komprese
-a	Vypisuje informace o dosažených hodnotách pro každý testovaný obrázek zvlášť
-p	Generuje data ve formátu bez popisu do složky *plot*
-c	Po otestování maže vygenerované komprimované soubory
-f	Otestuje pouze finální metodu (*dc_16to8_ze_bwt_rle_ari*)

B.2.1 Spuštění přes *make*

Pro usnadnění testování jsem zakomponoval pravidla pro automatické testování přímo do *Makefile*:

Pravidla podporovaná v *Makefile*	
make	Provede překlad všech modulů
make [metoda]	Provede překlad modulu konkrétní metody
make clean	Vyčistí adresář od souborů vzniklých překladem
make doxygen	Vygeneruje dokumentaci zdrojových kódů
make pack	Zabalí zdrojové soubory do formátu *tar.gz*
make test	Vypisuje informace pro sadu testovacích dat v agregované podobě
make test_verbose	Vypisuje informace pro každý testovaný obrázek zvlášť
make test_final	Generuje data pro *GNUPLOT*

B.2.2 Konfigrační soubor

Konfigurační soubor automatické testovací základny se nachází v umístění *workspace/scripts/config.sh* a obsahuje cesty k ostatním pracovním adresářům. Při případných problémech je zde třeba upravit nastavení položky *$workspace*. Všechny ostatní cesty jsou vztahovány relativně právě k tomuto adresáři.