VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

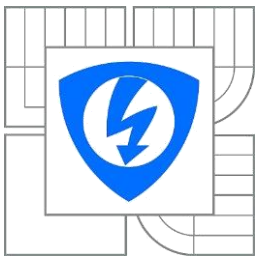
VIZUALIZACE ALGORITMŮ BEZZTRÁTOVÉ KOMPRESE
VISUALIZATION OF LOSSLESS COMPRESSION ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JIŘÍ MADEJA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL ŠILHAVÝ, Ph.D.
SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Jiří Madeja	***ID:***	154794
Ročník:	3	***Akademický rok:***	14/2015

NÁZEV TÉMATU:

Vizualizace algoritmů bezztrátové komprese

POKYNY PRO VYPRACOVÁNÍ:

V prostředí programu Matlab realizujte výukovou aplikaci demonstrující principy algoritmů pro bezztrátovou kompresi dat. Aplikace bude realizována pomocí grafického rozhraní GUI a bude krok za krokem vizualizovat jednotlivé části výpočtů. Dílčí kroky budou pro snadné porozumění doplněny textem. Aplikace bude rovněž umožňovat porovnání dosažené komprese pro uživatelem vybrané algoritmy. Aplikace bude minimálně obsahovat Huffmanovo, aritmetické a LZW kódování. Aplikaci koncipujte tak, aby ji bylo možno snadno doplnit o další algoritmy s jednotným rozhraním pro předávání vstupních a výstupních proměnných u jednotlivých funkcí implementovaných algoritmů. Ošetřete chybné zadání či volbu.

DOPORUČENÁ LITERATURA:

[1] Dušek, F., Honc, D. Matlab a Simulink: úvod do používání. Pardubice: Univerzita Pardubice, 2005. ISBN 80-7194-776-8.
[2] Zaplatílek, K., Doňar, B. MATLAB: tvorba uživatelských aplikací.Praha: BEN, 2004. ISBN 80-7300-133-0.
[3] Biggs, N. Codes: an introduction to information communication and cryptography. London: Springer, 2008, 273 s. ISBN 978-1-84800-272-2.

Termín zadání:	9.2.2015	***Termín odevzdání:***	2.6.2015

Vedoucí práce: Ing. Pavel Šilhavý, Ph.D.
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zaměřuje především na vývoj softwaru pro vizualizaci několika nejpoužívanějších algoritmů bezztrátové komprese. První část práce představuje teoretické poznatky z oblasti bezztrátových kompresních algoritmů. Druhá část práce je věnován popisu vývoje softwaru v prostředí Matlab. Návrh se věnuje především vytvoření prostředí pro vkládání dat a jejich zakódování pomocí nejběžnějších algoritmů bezztrátové komprese, porovnání jednotlivých algoritmů vzhledem k účinnosti kódování a vizualizace průběhu jednotlivých algoritmů a jejich vlastností při samotném průběhu kódování. V závěru práce je prezentován výsledný software.

ABSTRACT

This thesis focuses mainly on the development of visualization software for some of the most used lossless data compression algorithms. The first part of the thesis is devoted to the theoretical findings from the lossless data compression field. The second part of the thesis deals with the description of practical development of the software using Matlab. The draft is mainly focused on creating such environment to input data which will be encoded using most common lossless data compression algorithms, the comparison based on efficiency of those algorithms and visualization of the algorithm characteristics while in the process of encoding. Last part of the thesis is focused on presenting the final result of the software development.

KLÍČOVÁ SLOVA

bezztrátová komprese, kódování, LZW, RLE, DEFLATE, algoritmus, Huffman, aritmetické kódování, vizualizace, Matlab

KEYWORDS

lossless compression, coding, LZW, RLE, DEFLATE, algorithm, Huffman, arithmetic coding, visualisation, Matlab

CITACE PRÁCE

MADEJA, J. *Vizualizace algoritmů bezztrátové komprese*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 55 s. Vedoucí bakalářské práce Ing. Pavel Šilhavý, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vizualizace algoritmů bezztrátové komprese“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne . . . .

podpis autora

PODĚKOVÁNÍ

Děkuji vedoucímu semestrální práce panu Ing. Pavlu Šilhavému, Ph.D. za věcné rady a nasměrování správným směrem pro tvorbu této práce a výukové aplikace.

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Cílem této práce je vytvořit výukovou aplikaci demonstrující principy algoritmů pro bezztrátovou kompresi dat.

V teoretické části se budu věnovat teorii informace, informační entropii, nadbytečnosti a vysvětlení principu prefixových kódů. Srovnám a popíšu také nejběžnější algoritmy bezztrátové komprese, které budou zpracovány v praktické části této práce při vývoji aplikace. V popisu se zaměřím na kompresi bloků dat, RLE, LZW, DEFLATE, Huffmanovo kódování a aritmetické kódování, jejich vlastnosti a princip a následně krok po kroku naznačím jejich algoritmus a ukáži jej na příkladu.

V další části práce se budu věnovat podrobnému rozboru programu z hlediska jeho funkcí a způsobu jeho návrhu. Bude také popsán způsob implementace jednotlivých funkcí programu a vysvětleny jejich vzájemné vztahy.

V závěru práce představím výsledný program. Především pak jeho grafické prostřední a hlavní funkce.

Program byl vytvořen v programu Matlab ve verzi R2013a.

1 ÚVOD

1.1 Základní definice a vztahy

Informace

Je míra množství neurčitosti nebo nejistoty v nějakém náhodném ději. Množství informace vyjadřuje rozdíl mezi entropií informace před a po zprávě [1].

Z výše uvedeného vyplývá, že podle pravděpodobnosti výskytu dané informace můžeme také určit její množství, jinak řečeno její významovou důležitost. Je-li informace vysoce pravděpodobná, množství informace je menší, nežli v případě, kdy je generována zpráva o nízké pravděpodobnosti. Této vlastnosti využívají statistické (entropické) kompresní algoritmy.

Výpočet množství informace lze provést dle vztahu 1.1.

$$I_i = -log_2 p_i \ \text{[Sh]}, \tag{1.1}$$

kde p_i je pravděpodobnost výskytu symbolů ve zprávě.

Zdroj informace

Generátor zpráv. Můžeme jej chápat jako entitu produkující množinu zpráv s určitou pravděpodobností, tvořených ze symbolů určité abecedy [2].

Symbol/Znak

Nejmenší stavební prvek abecedy [2].

Abeceda

Konečná množina symbolů [2].

Zpráva

Zpráva v nějaké konkrétní abecedě je konečnou množinou jejích symbolů. Je posloupností prvků tvořící ucelenou informaci [2], [3].

Kódové slovo/Kód

Je prostým zobrazením symbolů jedné abecedy zprávou složenou ze symbolů abecedy druhé. Tedy kódovací funkce nepřiřazuje stejné kódy dvěma různým symbolům [2].

Entropie

Průměrná hodnota informace jedné zprávy či symbolu. Entropie určuje míru neurčitosti systému a jako takovou ji lze chápat jako množství informace systému. Při vysoké entropii se bavíme o informacích s podobnou pravděpodobností (významností), čím nižší entropie, tím je rozložení pravděpodobností nerovnoměrnější [2], [4].

Při stejné pravděpodobnosti výskytu zpráv je entropie systému maximální a tvoří limit pro entropické kompresní algoritmy. Pro výpočet entropie platí vztah 1.2.

$$H = \sum_{i=1}^{q} p_i I_i = -\sum_{i=1}^{q} p_i log_2 p_i \quad [\mathrm{Sh/symbol}], \tag{1.2}$$

kde q je počet symbolů abecedy.

Maximální entropie

Nejvyšší možná průměrná hodnota informace jedné zprávy či symbolu. Maximální entropie je dosaženo, mají-li symboly či zprávy stejnou pravděpodobnost výskytu. Výpočet maximální entropie je dán vztahem 1.3.

$$H_{max} = -log_2 \frac{1}{q} = log_2 q \quad [\mathrm{Sh/symbol}] \tag{1.3}$$

Relativní entropie

Poměr mezi entropií zprávy a maximální entropií, jaké může dosáhnout. Výpočet relativní entropie lze vyjádřit vztahem 1.4.

$$H_r = \frac{H}{H_{max}} \quad [-] \tag{1.4}$$

Redundance/Nadbytečnost

Vyjadřuje množství prvků zprávy nad minimální potřebnou mez pro utvoření užitečné informace. Redundanci, nebo také nadbytečnost získáme rozdílem relativní entropie od jedničky. Nadbytečnost může vznikat zcela přirozeně, např.: většina obrazových dat obsahuje mnoho nadbytečnosti, protože taková je jejich přirozená povaha.

Nadbytečnost se však vytváří i uměle, a to za účelem tvorby kódů zjišťujících chyby nebo kódů, co si dokáží opravit chyby samy. Redundantní data obsahuje i lidská řeč proto, aby si lidé dobře rozuměli i v přirozeně ztrátovém a rušeném prostředí. Z redundantních dat si pak lze při přeslechu nějaké hlásky či slabiky mluveného slova snadno domyslet původní význam zprávy (řečené slovo), aniž by hrozila jeho záměna za několik významově jiných slov.

V případě kompresních algoritmů se snažíme nadbytečnost snížit na minimum a chápeme ji jako přesný opak účinnosti algoritmu. Dokonalý entropický algoritmus by měl nadbytečnost rovnou nule a účinnost rovnou jedné. Z důvodu absence nadbytečnosti je nutné přenášet takto komprimovaná data po bezchybných kanálech, či je zakódovat i nějakým protichybovým algoritmem [5].

Pro výpočty redundance nepřizpůsobeného a přizpůsobeného zdroje platí vztahy 1.5 a 1.6.

$$R = 1 - H_r \quad [-] \tag{1.5}$$

$$R = 1 - \mu \quad [-] \tag{1.6}$$

Entropie přizpůsobeného zdroje

Entropie zdroje snížená o informace nutné k samotnému praktickému zakódování zprávy. V informatickém pojetí jde o vyjádření symbolu posloupností bitů, kde vyjadřujeme jeden symbol jednou a více značkami. Výpočet entropie přizpůsobeného zdroje lze vyjádřit vztahem 1.7.

$$H_x = \frac{H}{n} \quad \text{[Sh/symbol]}, \tag{1.7}$$

kde n je počet znaků.

Zprávu o abecedě „abc“ například můžeme kódovat pomocí ASCII tabulky, kde je každý znak vyjádřen sedmi bity, tedy n=7 [6]. Neznali-li bychom délku značky n (u rovnoměrného kódování), můžeme její minimální nutnou délku vypočítat zaokrouhlením maximální entropie nepřizpůsobeného zdroje nahoru. Tím také můžeme snadno zjistit, je-li dané rovnoměrné kódování efektivní. Tedy pro zprávu o abecedě „abc“ vychází potřeba alespoň 3 různých kódů, což splňuje pro dvojkové vyjádření $2^2 = 4$. Onen exponent dvojky je n, čímž můžeme snadno zjistit, že kódovali-li bychom

„abc“ pomocí ASCII tabulky, měli bychom 3,5x nižší efektivitu oproti vyjádření pouhými dvěma bity [9].

Maximální entropie přizpůsobeného zdroje

Maximální možná entropie, které může nabývat přizpůsobený zdroj. Tvoří limit pro entropické kódy. Lze vypočítat vztahem 1.8.

$$H_{x_{max}} = logF \qquad \text{[Sh/symbol]}, \tag{1.8}$$

kde *F* je počet stavů, kterých může nabývat kód.

Obvyklé je binární kódování a proto F = 2, kdy pro výpočet maximální entropie přizpůsobeného zdroje platí 1.9.

$$H_{x_{max}} = 1 \qquad \text{[Sh/symbol]} \tag{1.9}$$

Střední délka značky

Pro nerovnoměrné kódy platí, že jsou délky značek různé, střední délkou je tedy průměrná délka pro zdroj zpráv. Tedy záleží na délce značek a jejich pravděpodobností výskytu. U rovnoměrných kódů záleží na daném kódování.

Výpočet střední délky značky se provádí vztahem 1.10.

$$\bar{n} = E\{n_i\} = \sum_{i=1}^{q} p_i n_i \qquad [-] \tag{1.10}$$

Účinnost kompresního algoritmu

Ekvivalent relativní entropie pro přizpůsobené zdroje. Účinnost kódování je vyjádřena číslem od nuly po jedničku včetně. Nemůže být tedy nikdy větší, než jedna.

Pro výpočet účinnosti kompresního algoritmu platí 1.11.

$$\mu = \frac{H_x}{H_{x_{max}}} \qquad [-] \tag{1.11}$$

1.2 Bezztrátové kompresní algoritmy

U kompresních algoritmů se využívá nerovnoměrného kódování. To znamená, že každému symbolu může být přiřazen kód o jiné délce. To přináší možné snížení redundance, avšak pro rozlišení jednotlivých symbolů je třeba využít oddělovačů či nějak jinak odlišit navzájem symboly. Oddělovačů se nepoužívá z důvodu neefektivity, oddělovač může být například unikátní sekvence symbolů. Při použití oddělovačů by ztrácelo smysl mít komprimovaná data, proto se používá prefixových kódů.

Prefixový kód je takový, kde kód žádné značky nezačíná jako kód značky jiné. Díky tomu nelze zaměnit znaky mezi sebou. Nevýhodou prefixového kódu je to, že jakmile dojde k chybě, dojde k nenapravitelné ztrátě synchronizace a celý zbytek kódu se dekóduje chybně. V případě neprefixového kódu by došlo jen k chybnému přečtení toho znaku, ve kterém došlo k chybě [2].

Kompresní algoritmy využívají různých kroků ke zmenšení objemu dat beze ztráty informací. Jejich výhodou je i jistá univerzalita vůči ztrátovým algoritmům, které jsou vždy určeny pro konkrétní aplikaci.

Komprimují se s pomocí nich různé typy dat, často ve spojení se ztrátovými algoritmy (Např. obraz komprimovaný metodou JPEG se ještě komprimuje pomocí bezztrátového algoritmu RLE a následně Huffmanovým kódováním, případně aritmetickým kódováním).

Tyto algoritmy například kódují zprávu podle opakujících se posloupností znaků (Např. RLE kódování, ve spojení s nímž je výhodné využít BWT transformaci).
Nejčastěji používanými způsoby komprese jsou slovníkové a statistické algoritmy [9]..

Slovníkové algoritmy

Vytvářejí slovník hodnot - lze chápat jako jakési vzorkování všech frází obsažených ve zprávě, která se následně dekomprimuje za pomocí slovníku, ve kterém jsou obsaženy právě vzorky a jejich aliasy. Některé slovníkové algoritmy nepotřebují ukládat slovník zvlášť a při dekomprimaci jsou schopny si slovník vypočítat ze zakódované zprávy.

Statistické algoritmy

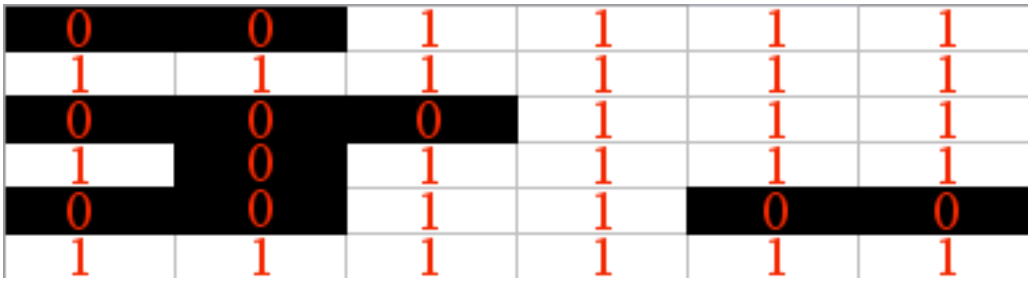
Jsou založeny na neuspořádanosti zpráv. To vyplývá ze situací z reálného světa, kde většina těchto dat nemá stejné rozložení pravděpodobností. Statistické algoritmy můžeme chápat tak, že přiřadí častějším jevům méně objemu na úkor méně častým jevům. V počítačové vědě to konkrétně znamená, že místo toho, aby měl každý znak zprávy alokovaný stejný počet bitů, statisticky významnějším znakům se přiřadí menší počet bitů a těm, které se vyskytují ve zprávě méně často naopak větší počet bitů.

Výsledkem pak je snížení celkového počtu bitů (díky nižší průměrné délce značky) a tím snížení redundance [9].

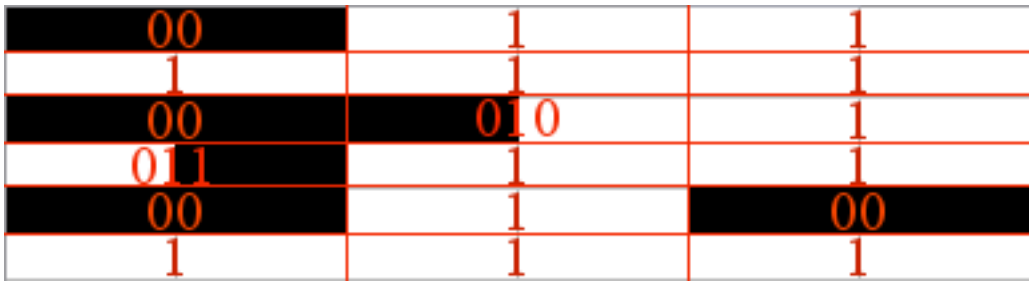
1.2.1 Komprese bloků dat

Tento algoritmus lze s výhodou využít u zdrojů informací se zdrojovou abecedou příliš malou pro efektivní realizování jiného kompresního algoritmu. Jde v podstatě o vytvoření abecedy s větším množstvím prvků, kde přidané prvky jsou složenými prvky původní abecedy [2].

Typickým příkladem je černobílý obrázek (ne škála šedé), kde se přirozeně (nebo neintuitivně je takový kód výsledkem Huffmanova algoritmu) nabízí přiřadit černé „0“ a bílé „1“. Je-li ale jedné barvy přítomno více, než druhé, toto kódování přestává být efektivní. V takovém případě je výhodné použít komprese bloků dat. Bloky dat chápejme jako n-tice hodnot. V tomto případě můžeme vytvořit dvojice hodnot Bílá-Bílá, Černá-Černá, Bílá-Černá a Černá-Bílá. Jsou-li v obraze přítomny převážně spojitá bílá místa, výsledkem takto zakódované abecedy Huffmanovým kódováním by byl například kód Bílá-Bílá = 1, Černá-Černá = 00, Bílá-Černá = 011 a Černá-Bílá = 010.



Obr.1.1: Data před kompresí do bloků



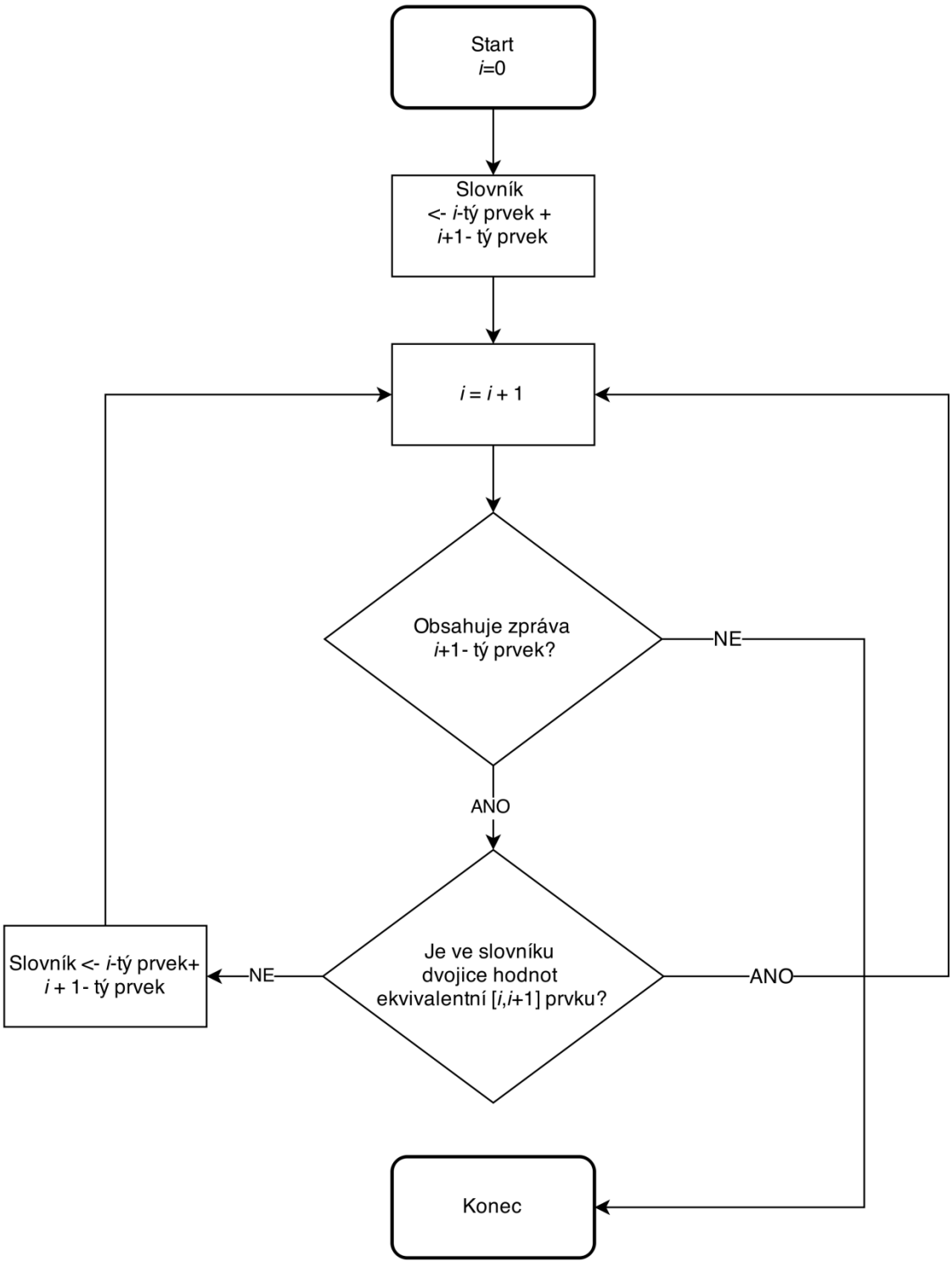
Obr. 1.2: Data po kompresi do bloků dat

Spočítáme-li celkový součet bitů v obrazci, pro nekomprimovaný obrazec nám vyjde celkem 36 bitů.

V komprimovaném obrazci je součet bitů 26.

Je tedy zřejmé, že i na této jednoduché ukázce došlo k efektivní kompresi – komprimovaný obrázek zabírá 72% původní velikosti.

Takto lze spojovat data do stále větších bloků, dokud je to výhodné.



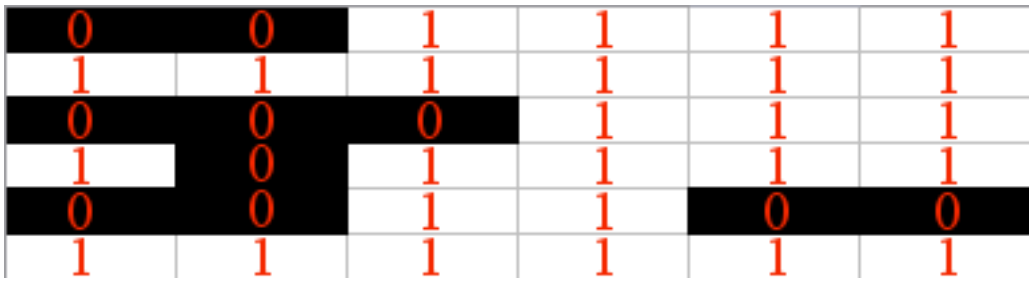
Obr. 1.3: Vývojový diagram komprese bloků dat - dvoubloková

1.2.2 LZW KÓDOVÁNÍ

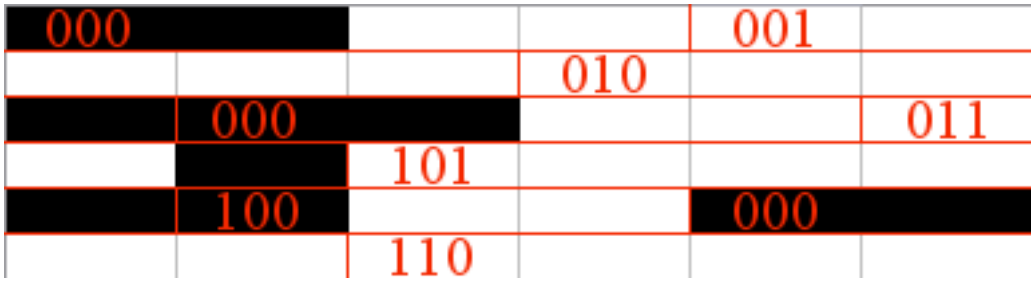
Jedná se o dynamický slovníkový algoritmus. Postup kódování zprávy je takový, že procházíme zprávu, dejme tomu řetězec znaků, pak: zapíšeme do jednoho sloupce 1. znak, do druhého sloupce 2. znak, zkontrolujeme, je-li dvojice 1. - 2. znak ve slovníku a není-li, zapíšeme do třetího sloupce 1. znak a do slovníku přidáme dvojici 1.- 2. znak. Pokračujeme dále tak, že zapíšeme do prvního sloupce na novém řádku 2. znak a do druhého sloupce 3. znak, zkontrolujeme, je-li dvojice 2. - 3. znaku ve slovníku a pokud ano, zapíšeme do třetího sloupce dvojici 2.-3. znak. Na dalším řádku pak zapíšeme do prvního sloupce dvojici 2. - 3. znak a do druhého sloupce 4. znak, opět zkontrolujeme, nenachází-li se kombinace 2.-3.-4. znaků ve slovníku [2].

Takto pokračujeme do konce řetězce.

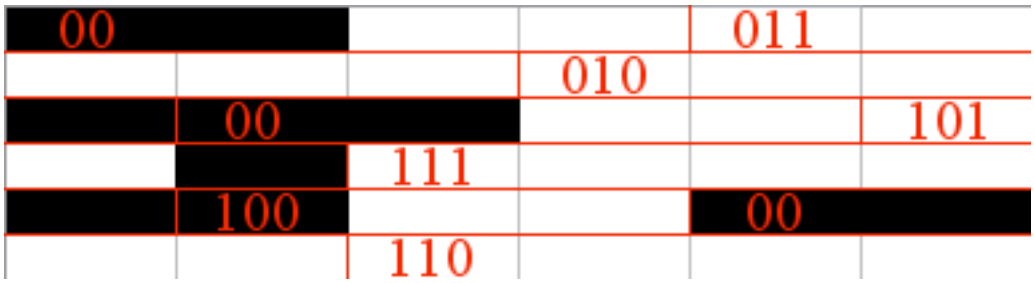
Výhodou LZW algoritmu je snadná aplikovatelnost na hardwarové prostředky (lze efektivně vytvořit dedikovaná zařízení pro LZW kompresi) [8]. Pro dekódování není třeba slovník ukládat, stačí jen určit, kolika bitů dosahuje nově vytvořený slovník.



Obr. 1.4: Data před LZW kompresí



Obr. 1.5: Data po LZW kompresi s pevnou šířkou kódu

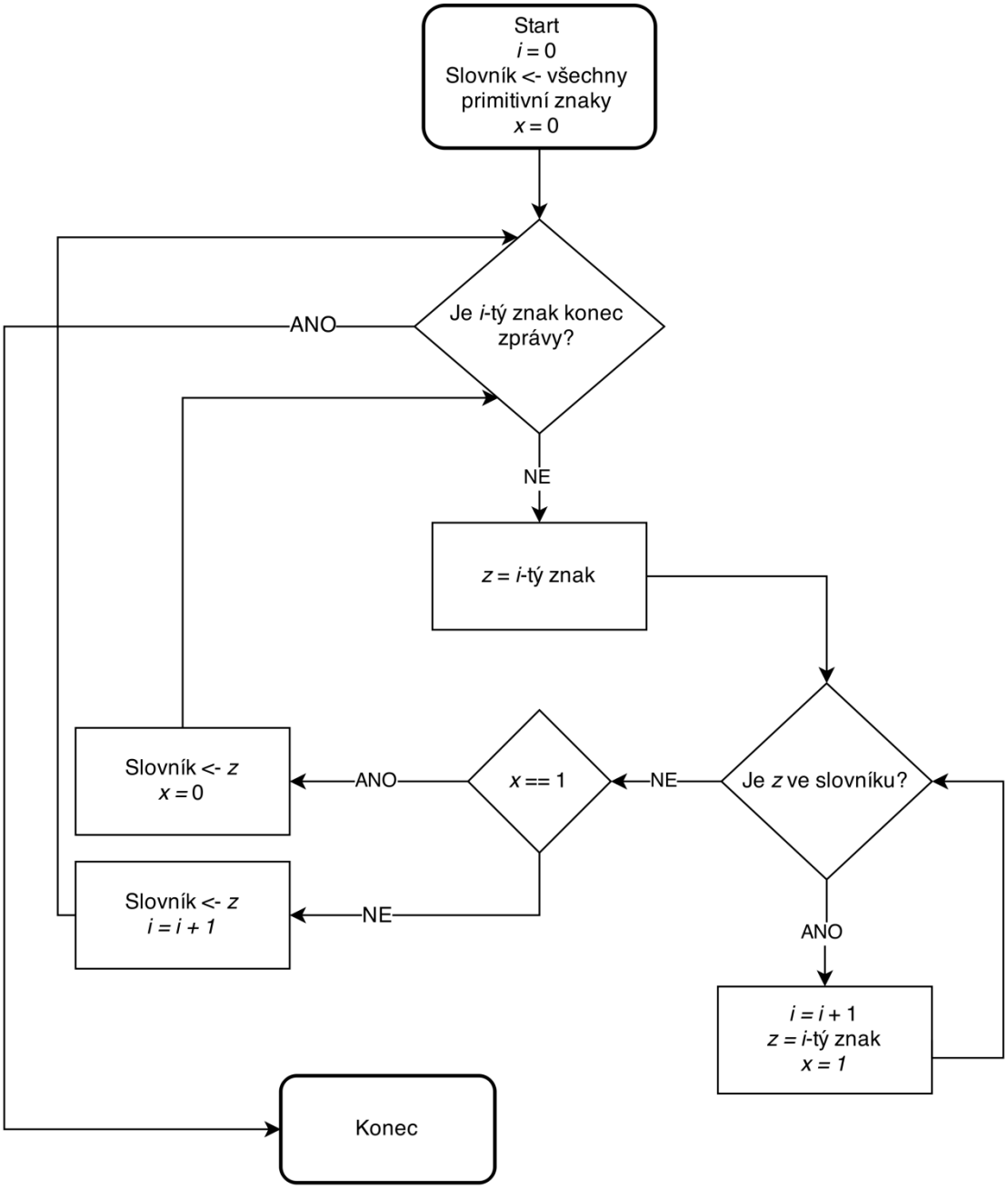


Obr. 1.6: Data po LZW kompresi s Huffmanovým kódováním slovníku

Nekomprimovaný obrazec zabírá 36 bitů.

Obrazec zakódovaný pomocí LZW kódování, kde prvky slovníku seřadíme dle pravděpodobnosti a nepoužité prvky odstraníme ze slovníku a zároveň používáme pevnou délku kódu, dostaneme celkový počet 27 bitů na obrazec.

Pokud vezmeme hodnoty z předchozího případu a zakódujeme je do prefixového kódu proměnlivé délky, znázorněný obrazec bude mít délku 24 bitů.



Obr. 1.7: Vývojový diagram LZW kódování

1.2.3 RLE KÓDOVÁNÍ

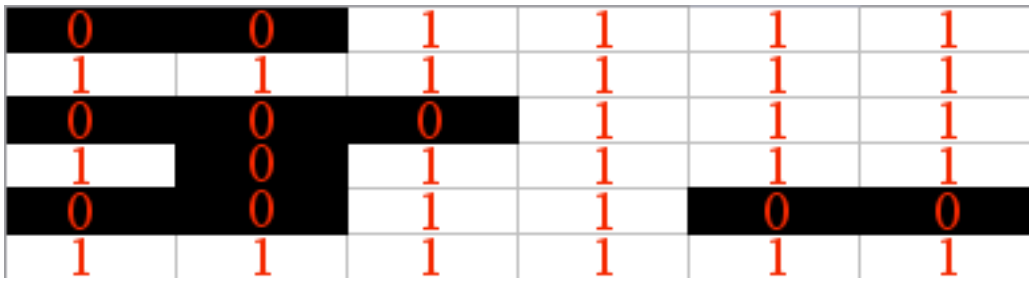
Jedná se o velmi jednoduchý kompresní algoritmus. Obzvláště pro obrazová data je výhodné použít RLE kódování, tedy run-length encoding, které zjednodušeně řečeno pracuje na principu toho, že místo vyjádření každého prvku individuálně vyjádří počet stejných za sebou jdoucích prvků a jejich hodnotu. Je realizován dvojicemi hodnot, kde jedna hodnota udává délku posloupnosti a druhá její hodnotu, tedy: „5 červená“.

Například namísto „červená červená červená červená zelená červená“ můžeme říci: „4krát červená 1krát zelená 1krát červená“.

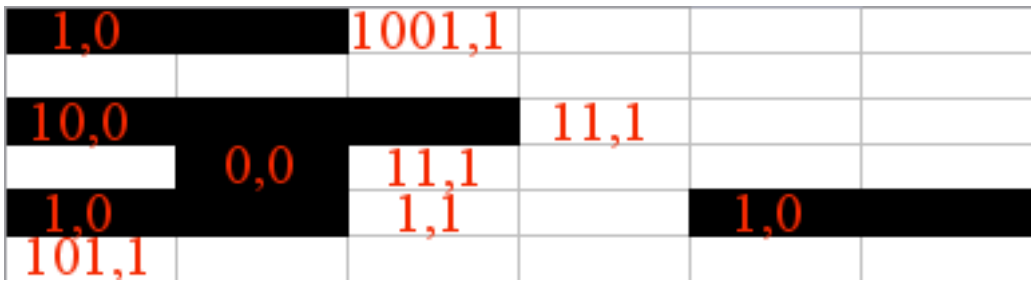
Tato metoda je nejvhodnější pro použití na různých ikonách a ilustracích, které neobsahují nějaké komplexní vzory, avšak může být vhodná i pro fotografie při současném použití JPEG algoritmu, díky tomu, že ten ze své povahy sdružuje podobné barvy do stejné [10].

Známým formátem využívající RLE kódování je například Truevision TGA, jinak také TARGA.

Protože je přidána přídavná informace o délce posloupnosti, nejsou-li posloupnosti dostatečně dlouhé (bez přerušení jinou hodnotou), může dojít k výraznému zvýšení nadbytečnosti. Proto je účinnost tohoto algoritmu silně závislá na charakteru vstupních dat.



Obr. 1.8: Data před RLE kompresí

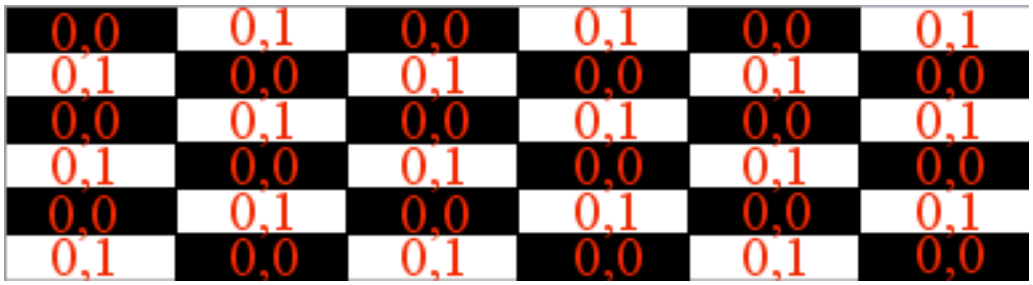


Obr. 1.9: Data po RLE kompresi

Dejme tomu, že první číslo každé části symbolizuje počet opakování nějakého znaku - 1 a druhé číslo (číslo za čárkou) symbolizuje jeho hodnotu (pixelu v tomto symbolickém obrázku). Obojí zapsané v dvojkové soustavě - pak zápis „1,0“, který lze vidět v první části (2 černé „pixely“) symbolizuje $1_2 = 1$ násobek. Avšak pozor, vzhledem k tomu, že nemá smysl vyjadřovat, že je jakákoliv hodnota přítomna nulakrát, pak lze vyjadřovat násobky číslem n-1, tedy tento „1 násobek“ je ve skutečnosti dvojnásobkem. 0 za čárkou pak symbolizuje hodnotu - černou. Zápis „1,0“ tedy znamená, že jsou za sebou dva sousedící černé pixely (obecně řečeno n-krát za sebou stejné hodnoty).
„1001,1“ pak znamená 10 za sebou jdoucích bílých pixelů.

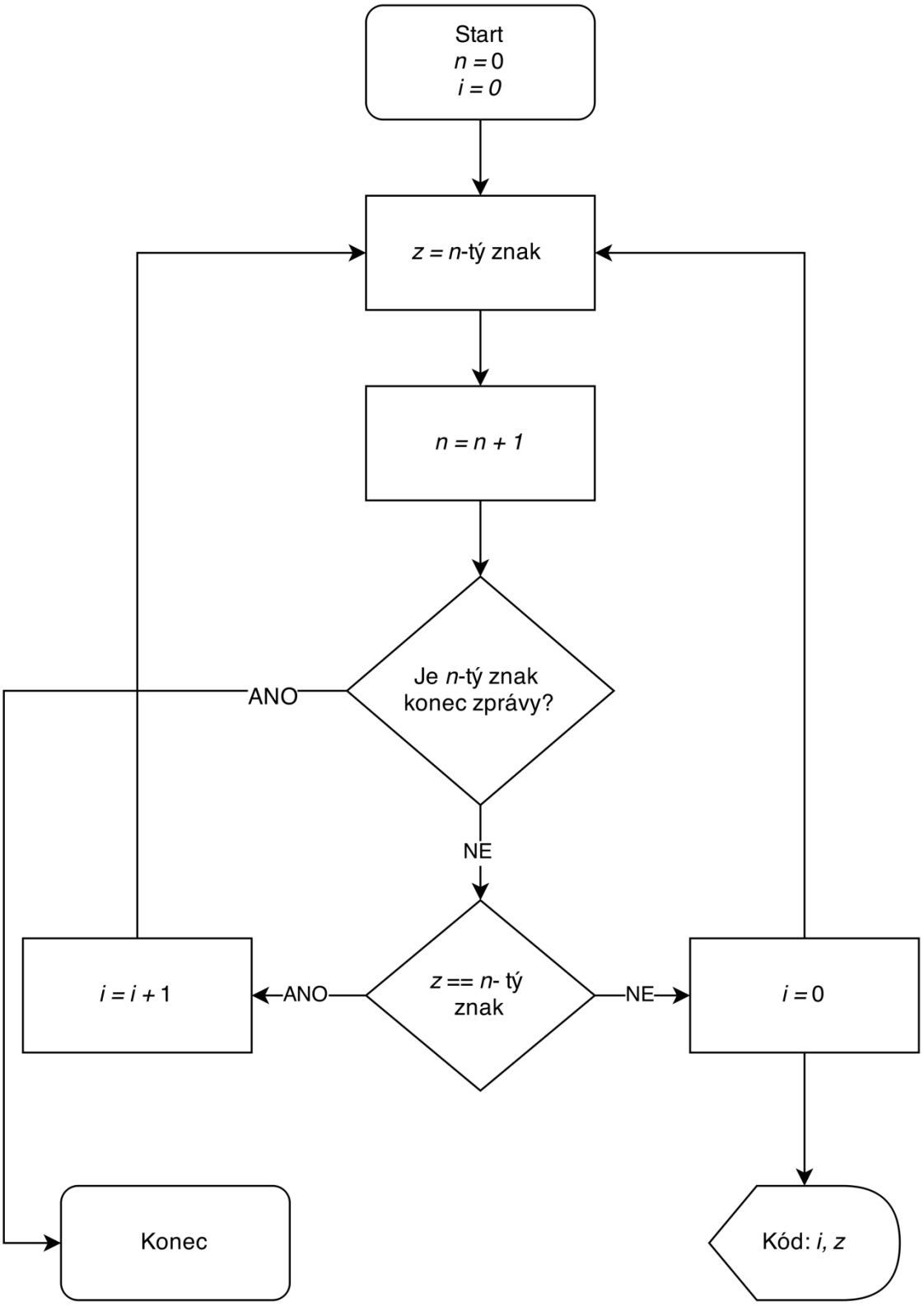
Spočítáme-li celkový počet bitů v obrázku, pak u nekomprimovaného vyjde 36. U komprimovaného obrázku je součet bitů 28, což je jen málo účinná komprimace oproti kompresi bloky dat.

Logickým úsudkem můžeme dojít k tomu, že střídaly-li by se černé pixely s bílými vždy ob jeden pixel, pak by se (v případě černo-bílého obrázku) zvýšil celkový počet pixelů na dvojnásobek, tedy 72 bitů.



Obr. 1.10: Data po RLE kompresi (pro nevhodný zdroj informace)

Vzniklá redundance je způsobena nutností vyjádřit počet opakování daných hodnot, i přesto, že v tomto případě tato informace není vůbec využita.



Obr. 1.11: Vývojový diagram RLE kódování

1.2.4 DEFLATE KÓDOVÁNÍ

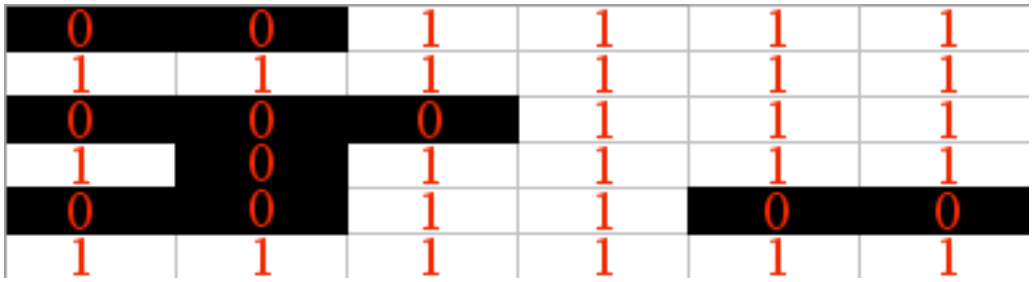
DEFLATE je algoritmus spojující LZ77 kódování a Huffmanovo kódování. LZ77 je podobně jako ostatní algoritmy z rodiny LZ slovníkovým algoritmem. Jeho princip je založen na trojici (případně dvojici) hodnot reprezentujících jeden znak. Při prvním nálezu znaku ve zprávě se uloží hodnoty 0 (reprezentující vzdálenost znaku od námi porovnávaného), 0 (délka řetězce shodného s námi hledaným řetězcem) a znak.

Najdeme-li ten samý znak dále a bude-li se jednat o jeden znak, bude mít hodnotu například 5 (znak je vzdálený 5 znaků od porovnávaného) a 1 (jedná se o jeden znak). Zde se odkazujeme na jeden znak vzdálený pět znaků před porovnávaným, a proto nemusíme určovat, o jaký znak se jedná [11].

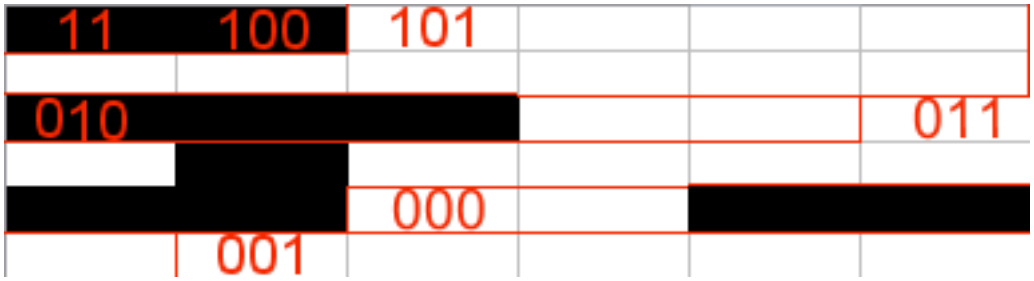
Například pro řetězec „MADEJAMATLAB“ vytvoříme slovník:

M	A	D	E	J	A	MA	T	L	A	B
0,0,M	0,0,A	0,0,D	0,0,E	0,0,J	4,0	6,2	0,0,T	0,0,L	3,1	0,0,B

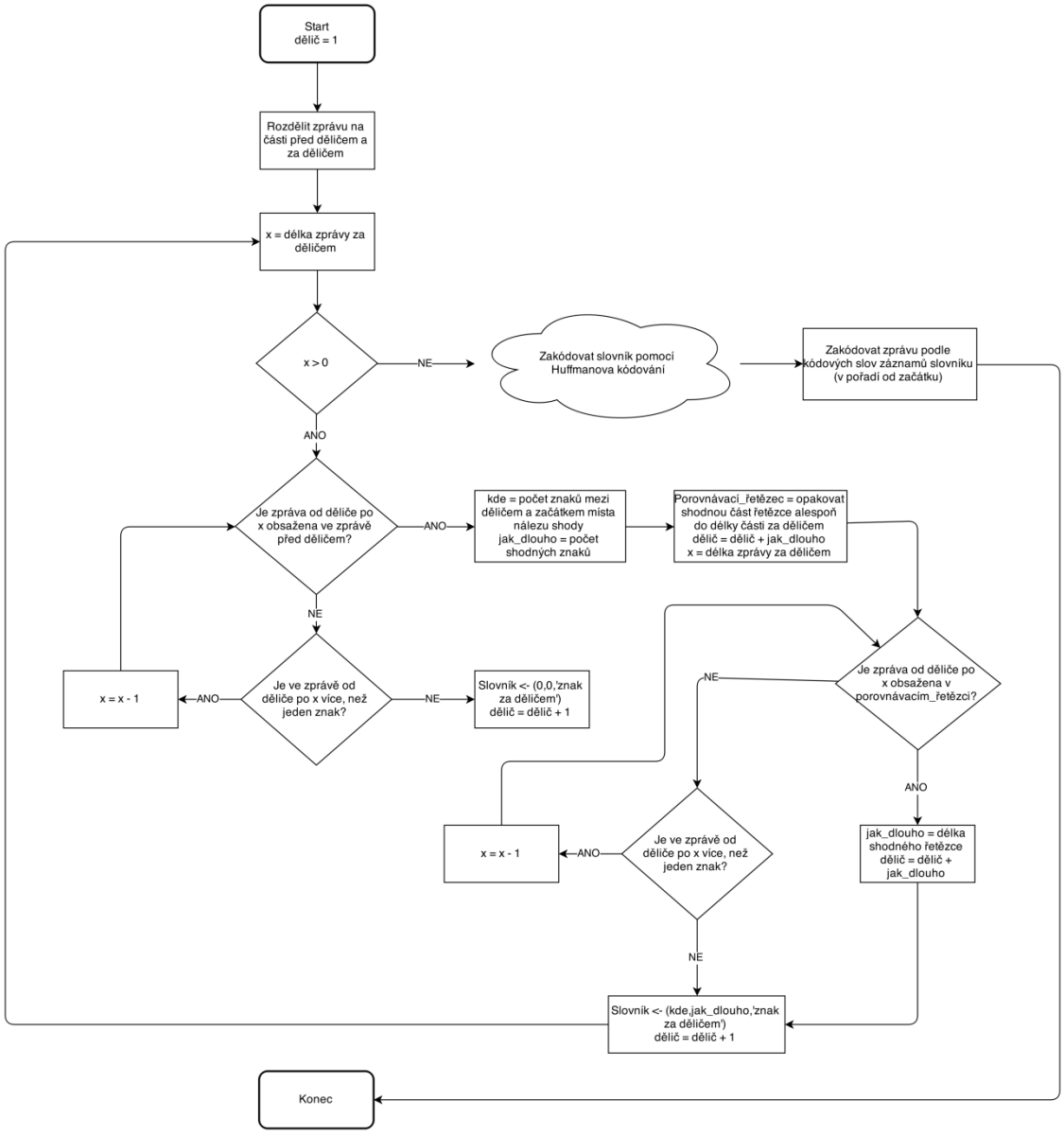
Tento kód následně s výhodou zakódujeme pomocí Huffmanova kódování.
Používá se v PNG souborech, ZIP a gzip archivech.



Obr. 1.12: Data před DEFLATE kompresí



Obr. 1.13: Data po DEFLATE kompresi



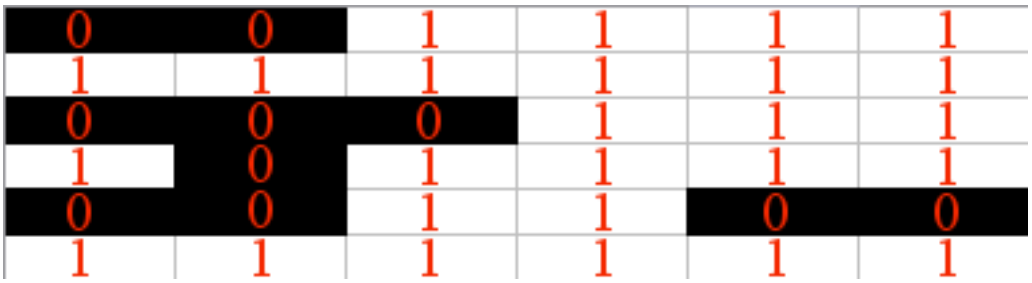
Obr. 1.14: Vývojový diagram DEFLATE kódování

1.2.5 Huffmanovo kódování

Jak již bylo řečeno, Huffmanovo kódování je entropickým kódováním. Vyvinul jej David Albert Huffman. Využívá prefixového kódu, což znamená, že začátek žádného kódového slova nemůže obsahovat jiné kódové slovo, protože by pak nešlo jednoznačně dekódovat zprávu [2] [7].

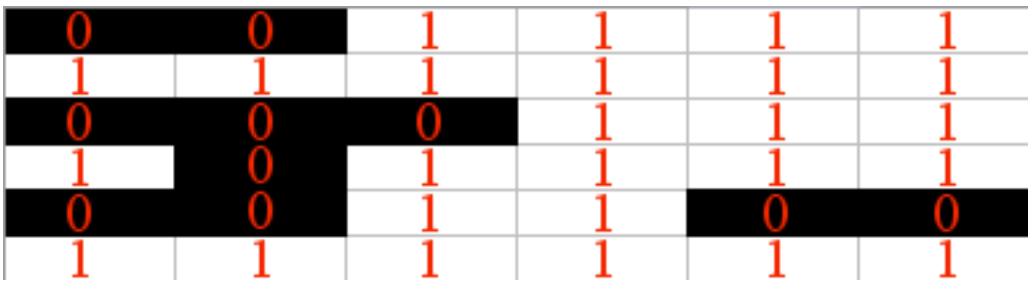
Při Huffmanově kódování je nejprve nutné projít celou zprávu a spočítat četnost výskytu jednotlivých znaků. Podle této četnosti se následně vytvoří strom, podle kterého se přiřazují kódová slova jednotlivým znakům tak, aby bylo navržení nejoptimálnější a aby vznikl prefixový kód. To je velkou slabinou Huffmanova kódování, protože nelze kódovat zprávu, pokud jí nemůžeme přečíst celou až do konce. [2]

Huffmanovo kódování je málo efektivní, obsahuje-li abeceda málo symbolů, například jen „1“ a „0“, které používám v ukázkových příkladech, nedokáže Huffmanovo kódování efektivně zakódovat. Prakticky jen zamění symbol za jedničku nebo nulu podle toho, jestli je jeho výskyt pravděpodobnější, než výskyt druhého symbolu. Tímto tedy nedojde k žádné kompresi.



Obr. 1.15: Data před Huffmanovou kompresí

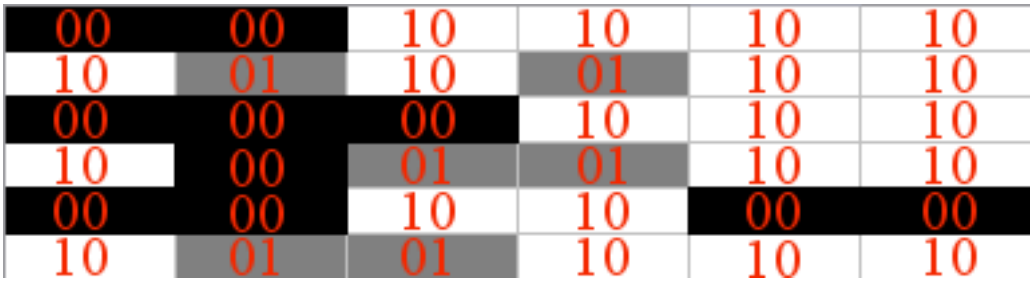
Budeme-li kódovat symboly s vyšší pravděpodobností „1“ a s nižší „0“, zakódovaný obrázek bude zcela identický s nezakódovaným.



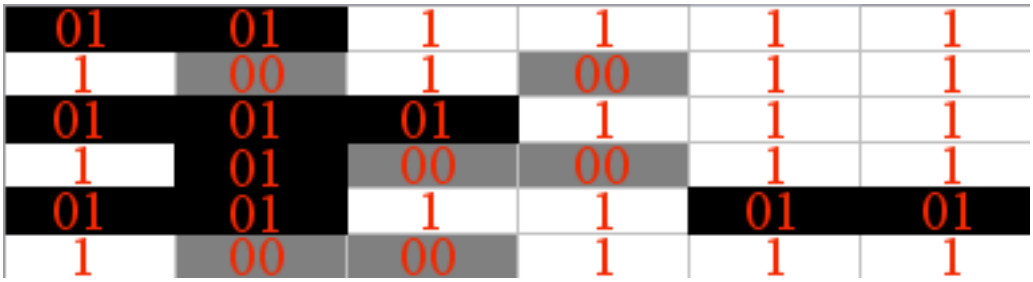
Obr. 1.16: Data po Huffmanově kompresi

Aby bylo možné naznačit princip Huffmanova kódování, ukážu obrázek o větší abecedě.

Hned letmým pohledem lze vidět, že jde o kódování s pevnou šířkou kódového slova, a to 2 bity – minimum nutné k zakódování 3 různých symbolů. Avšak dva byty mohou reprezentovat celkem $2^2 = 4$ kombinací. Vzhledem k tomu, že používáme jen 3 symboly ze 4 možných, už tímhle můžeme vidět, že kód má redundanci. A právě Huffmanovým kódováním navrhneme efektivní kód s proměnlivou velikostí kódu.



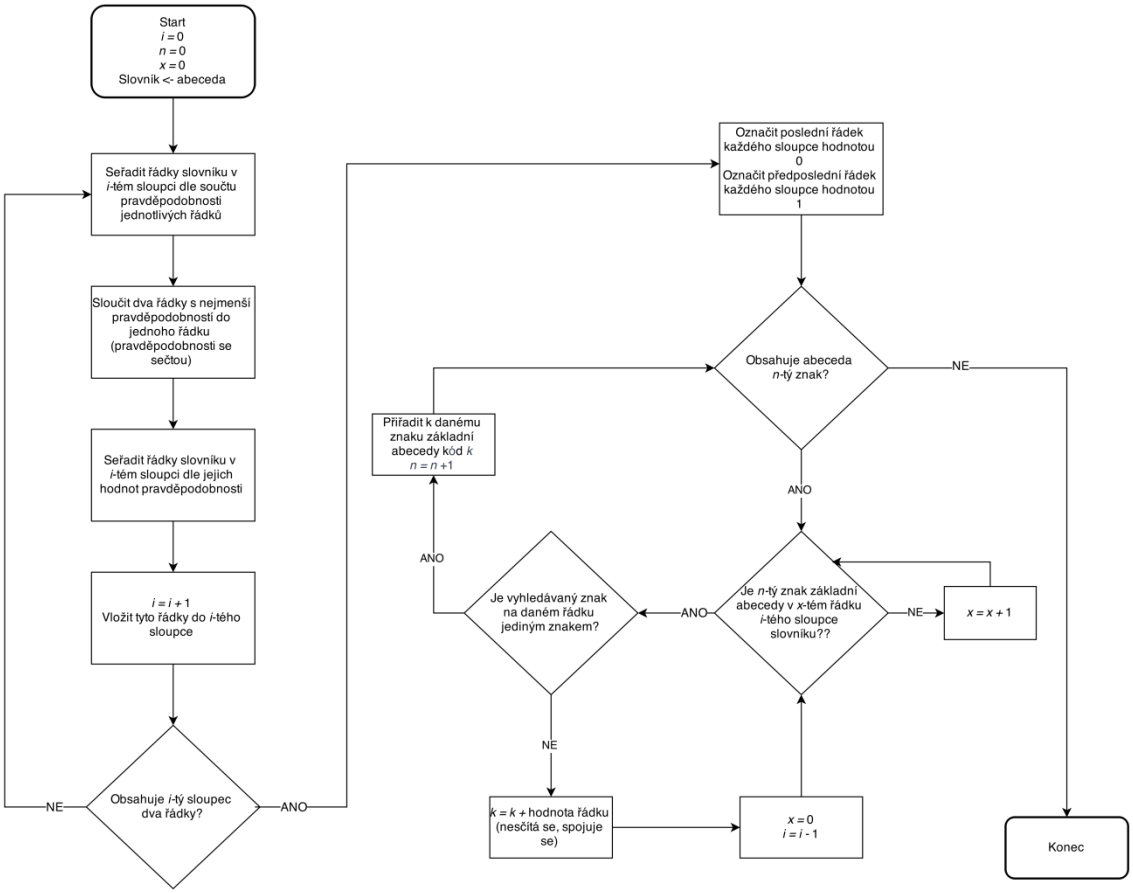
Obr. 1.17: Data s rozšířenou abecedou před Huffmanovou kompresí



Obr. 1.18: Data s rozšířenou abecedou po Huffmanově kompresi

Zpráva před kompresí byla tvořena celkem 72 bity. Zpráva po kompresi je tvořena celkem 52 bity. Zpráva byla zkomprimována na 72% původní velikosti. Ani to však není limitem entropického kódování, Huffmanovo kódování není příliš efektivní pro zprávy tvořené abecedou s malým počtem symbolů.

Právě tu neschopnost Huffmanova kódování kódovat zprávy s malými abecedami řeší aritmetické kódování.

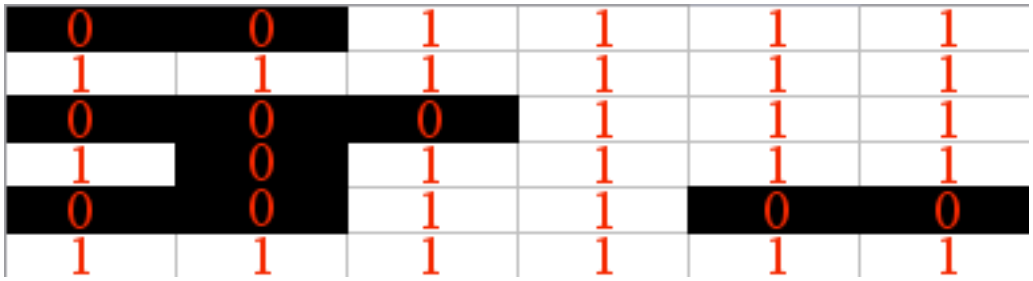


Obr. 1.19: Vývojový diagram Huffmanova kódování

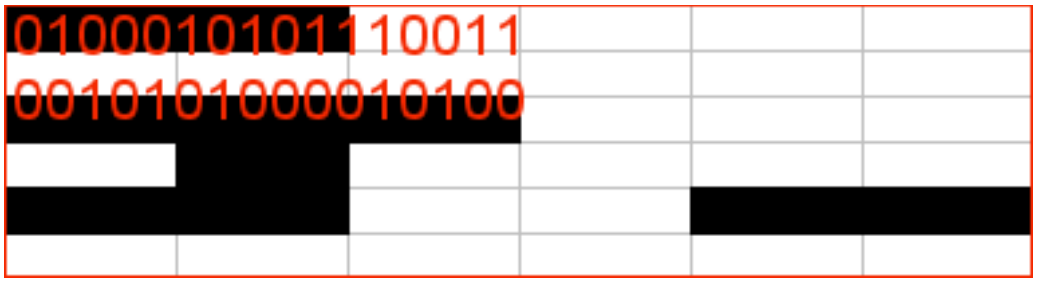
1.2.6 Aritmetické kódování

Aritmetické kódování je podobně, jako Huffmanovo kódování založeno na proměnlivé délce kódových slov. Dle pravděpodobnosti znaků jim přiřadí rozsah v intervalu (0,1), kde první znak zprávy obsahuje ve svém intervalu opět v poměru pravděpodobnosti dalších znaků, další znak opět další znaky a tak dále [2]. Pro přesné určení konce zprávy se obvykle (ale ne nutně) přidává speciální znak, který jasně určuje, kde končí zpráva.

Výhodou aritmetického kódování je zejména jeho individuální přístup pro každý znak abecedy, který má za následek, že při změně pravděpodobností znaků lze přepočítat jejich kódy bez nutnosti vytvářet celý kódovací strom jako v případě Huffmanova kódování [2]..

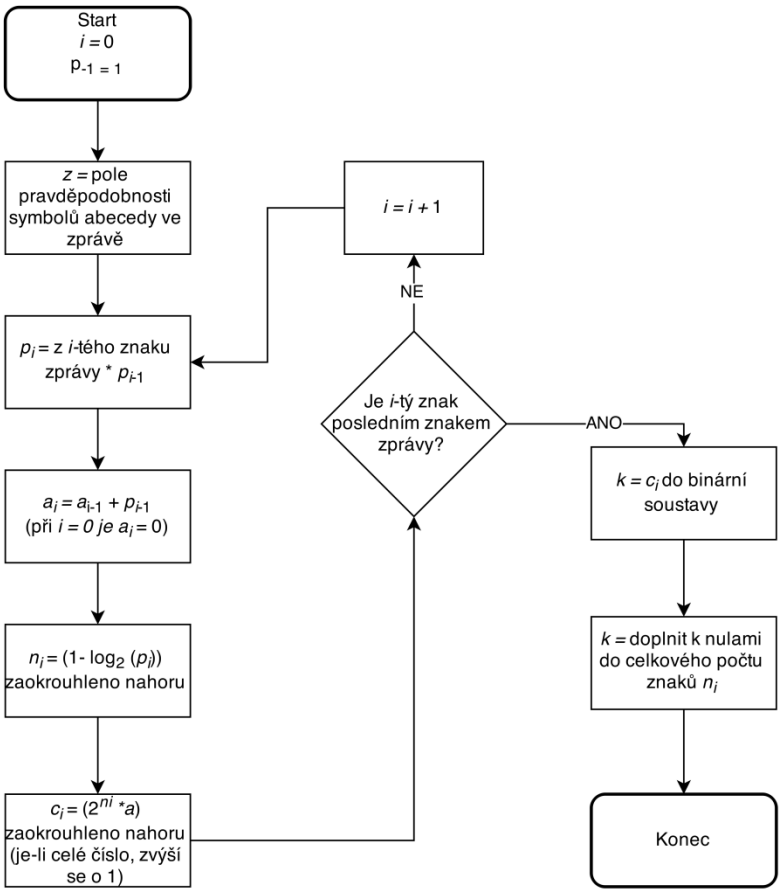


Obr. 1.20: Data před kompresí aritmetickým kódováním



Obr. 1.21: Data po kompresi aritmetickým kódování

Zpráva před kompresí byla tvořena celkem 36 bity. Zpráva po kompresi je tvořena celkem 32 bity. Zpráva byla zkomprimována na 89% původní velikosti. Na tomto příkladu lze vidět, že byť není aritmetické kódování vhodné pro tento typ zprávy (zejména problém malé abecedy), poradilo si s kompresí lépe, než Huffmanovo kódování (11% komprese aritmetického kódování proti 0% kompresi Huffmanova kódování).



Obr. 1.22: Vývojový diagram pro aritmetické kódování

2 NÁVRH PROGRAMU

2.1 Návrh funkčnosti programu

Pro efektivní výstavbu programu je důležité si nejdříve ujasnit programátorské cíle, funkce, jaké má program umět a hlavně jaká data a v jaké formě bude program přijímat a jaká data a v jaké formě bude program ukazovat uživateli po provedení požadovaných funkcí.

Program bude umožňovat výběr z několika typů kódování (komprese bloků dat, RLE, LZW, DEFLATE, Huffmanova kódování a aritmetického kódování). Dále také bude umět porovnat tato kódování dle předem určených parametrů. Následně umožní zadat uživateli vlastní definované parametry kódování, tedy základní abecedu a zprávu, určenou k zakódování. Umožní také přepsat pravděpodobnosti symbolů ve zprávě manuálně, a to jak četností výskytu znaků, tak jejich pravděpodobností. Tyto parametry bude uživatel zadávat buďto manuálně, nebo bude mít možnost načíst zprávu ze souboru a nechat tak zbytek parametrů kódování vygenerovat program.

Program bude umožňovat ovládání provádění jednotlivých operací programu, uživatel bude konkrétně schopen spustit provádění kódování, bude moci zobrazit kódovací tabulku, bude si moci nechat zobrazit vývojový diagram ukazující právě probíhající algoritmus. Dále bude umožňovat nenechat program provést celé kódování v jeden okamžik, ale kódovat po programátorem předem definovaných krocích – tzv. provádět krokování. Během každého kroku si uživatel bude moci zobrazit vývojový diagram a program mu naznačí, kde ve vývojovém diagramu se právě algoritmus nachází.

Další funkčností programu je zobrazení konečných výsledků. Program zobrazí jak samotnou zakódovanou zprávu, tak i parametry kódování (např. účinnost daného kódování). Dále program zobrazí dodatkové informace, které může uživateli poskytnout o daném algoritmu, tyto informace tedy budou intuitivně co nejblíž právě provedenému algoritmu (vzhledem k odlišnostem mezi algoritmy nelze mít univerzální způsob zobrazení těchto dat, např. stromová struktura, která se hodí pro Huffmanovo kódování, je zcela nevhodná pro RLE apod.).

Program bude kontrolovat vstupy uživatelů, a to zejména tak, aby uživatele nemystifikoval a uživatel si tak byl jist, že kódování chápe zcela správně. Program by tedy měl uživatele upozornit na (pravděpodobně) špatně zadané hodnoty, jako např. dva

stejné symboly v rámci jedné abecedy. Dále by měl program uživatele vhodně upozornit na funkce, které má, nebo naopak nemá, u kterých si uživatel může domýšlet pravý opak. Program by neměl opravovat různé špatné vstupy a další pochybné ovládání programu ze strany uživatele, aby uživatel nezískal dojem, že nějaká funkčnost dělá něco, co ve skutečnosti nedělá. Na tyto skutečnosti by měl program uživatele upozornit a nechat jej napravit dané nesrovnalosti.

Program bude poskytovat nápovědu pro uživatele v přívětivé formě – je tedy vhodné rozdělit nápovědu do více celků, a to podle logické souvislosti.

2.2 Návrh grafického uživatelského prostředí

Prvním krokem po navržení funkčnosti programu je návrh grafického uživatelského prostředí, dále GUI.

Z hlediska práce na programu je tento krok logický, protože se utváří hlubší představa o funkčnosti programu, logických souvislostech a je to proces, který lze udělat poměrně intuitivně bez hlubší znalosti implementace jednotlivých algoritmů, což je důležité vzhledem k tomu, že nelze získat hlubší představu implementace jednotlivých algoritmů hned na začátku tvorby programu.

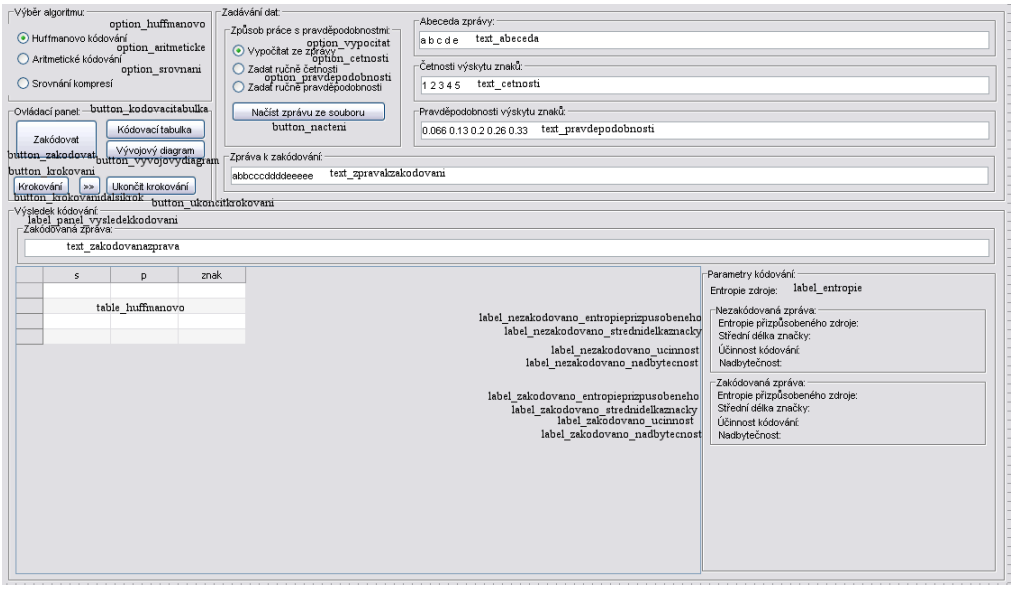
GUI by mělo být navrženo dle známých doporučení. S ohledem na lidské schopnosti a možnosti.

Z důvodů jednoduchosti je třeba omezit počet nastavení a různých možností ovládání programu na nezbytné minimum. Jakékoliv prvky, které uživatel v daném kontextu či chvíli nevyužije, je třeba zneaktivnit, aby nerozptylovaly uživatelovu pozornost, týká se to například různých tlačítek a polí. Toto řešení je mnohonásobně vhodnější, než přímo skrývání daných prvků, protože dává uživateli informaci o tom, že tam taková funkčnost je, ale nemůže ji v danou chvíli využít (ostatně proto se zneaktivnění, na rozdíl od například zneviditelnění prvků používá téměř ve všech aplikacích).

Z hlediska přehlednosti je třeba program rozdělit do několika funkčních celků, které spolu logicky souvisejí. U programů s komplexním ovládáním a s velkým množstvím informací je vhodné tyto bloky neukazovat v jednu chvíli, ale mít možnost mezi nimi přepínat, jako je tomu například u moderních operačních systémů nebo internetových prohlížečů zvykem. Ovšem z výčtu funkcí, které má navrhovaný program

umět, lze usoudit, že nebude přecházení mezi funkčními celky třeba a tyto se pohodlně vejdou na jednu souvislou plochu, aniž by došlo k znepřehlednění programu pro uživatele. Je však stále třeba funkční celky vizuálně oddělit a zároveň vhodné je náležitě pojmenovat. V GUI Matlabu se nabízí využít prvku „panel“, který je zároveň standardem pro řešení tohoto problému i v jiných prostředích. Jak už to bývá u návrhu různých ovládání, většina těchto doporučení se netýká jen prostředí počítačového, ale zejména i obsluh různých mechanických zařízení z praktického života. Dále by mělo GUI reflektovat časovou posloupnost prováděných úkonů, a to tak, jak je (alespoň v západní civilizaci) zvykem – jako při čtení textu, tedy zleva doprava/shora dolu, či jejich kombinace. Uživatel je zvyklý na toto uspořádání a tak mu bude připadat intuitivní i obsluha programu.

Na základě prvotního návrhu bylo jednoduché obrázkové schéma, které umožní programátorovi přehlednou a intuitivní práci s grafickým prostředím. Jde o tzv. screenshot grafického prostředí programu, který je navíc opatřen identifikátory jednotlivých prvků, se kterými bude programátor pracovat. Tyto prvky by jinak musel programátor ručně individuálně vyhledávat pokaždé, kdy by s nimi chtěl pracovat.



Obr. 2. 1: Referenční schéma uživatelského grafického prostředí

2.3 Řešení programu z hlediska algoritmizace

Programátor při řešení programátorského problému musí vhodně navrhnout stavbu programu a způsob, jakým jej vybuduje. Vhodným nástrojem je navržení základní funkcionality dekompozicí a abstrakcí algoritmického problému. Toho se využívá, protože při tvorbě složitějších programů je nad lidské možnosti obsáhnout celou problematiku naráz.

Využívá se dekompozice, kde se rozkládají komplexní problémy na elementární podproblémy, které můžeme řešit (ať už z pohledu programátora nebo procesoru počítače). Z hlediska abstrakce jde o zjednodušení složitého problému ignorováním detailů v danou chvíli nepodstatných.

Zásadní je vhodně rozvrhnout budoucí kód. Vzhledem k průmyslovému standardu v tomto ohledu by bylo vhodné využít objektového paradigmatu. Ten spočívá v tom, že programátor pracuje s jednotlivými „objekty“, které mezi sebou komunikují a mají různé vlastnosti. Objektové paradigma umožňuje velkou míru abstrakce, což znamená, že programátorovi stačí, aby věděl o tom, co metody objektů dělají a k čemu jsou různé objekty, ale nemusí znát přesný způsob implementace dané funkcionality. To je výhodné pro současný i budoucí vývoj programu.

Bohužel Matlab není příliš zaměřen na objektově orientovaný styl programování, takže je třeba tvořit kód funkcionálním paradigmatem, který je v současnosti druhým nejpoužívanějším stylem programování. Ten spočívá v tom, že jednotlivé funkcionality programu se vnořují do funkcí, které mají nějaké vstupní a výstupní parametry. Do funkce se mohou vnořovat další funkce a tak dosahovat lepší dekompozice algoritmického problému.

Vzhledem k výše uvedenému, pro úspěšné a efektivní psaní kódu je důležité, aby si programátor definoval funkcionality programu (viz kapitola 2.1), dle těchto funkcionalit je vhodné navrhnout i vnitřní strukturu funkcí v programu. Funkcionality programu jsou značně abstraktní a není v silách programátora, aby navrhl a promyslel do detailů potřebné funkce.

Nezbývá tedy nic jiného, než se pustit do kódování a řešit dílčí problémy, až se vyskytnou. Aby byl kód přehledný a srozumitelný pro programátora a případné další generace, které budou kód revidovat a upravovat v budoucnu a budou se mu snažit

porozumět, je vhodné udržovat referenční seznam funkcí – jejích vstupních a výstupních parametrů, názvu dané funkce a popis toho, co dělá.

2.4 Způsob implementace nových funkcionalit programu a jejich vzájemné závislosti

2.4.1 Přidání nového kódovacího algoritmu do programu

Aby bylo možno algoritmus vybrat v GUI za chodu programu, je nutné jeho název přidat do prvku listbox s tagem `listbox_algoritmy`, kde přidáním dalšího řádku ve vlastnosti `String` se přidá další políčko výběru algoritmu, zde je vhodné napsat název algoritmu.

Funkce `proved_algoritmus(handles)` následně podmínkou s voláním funkce `vyber_algoritmu(handles)` zjišťuje, který algoritmus v prvku je označen. Funkce `vyber_algoritmu(handles)` vrací jen číselnou reprezentaci vybraného algoritmu, a to jeho pořadím odshora (počítáno od 1).

Do podmínky ve funkci `proved_algoritmus(handles)` je tedy nutné přidat další větev s novou hodnotou (je-li nový algoritmus na 5. pozici v listboxu, bude ona hodnota 5 (pro bližší pochopení konkrétních funkcí viz kapitola 2.5)). Nebyl-li algoritmus přidán na konec listboxu, je třeba přečíslovat ostatní větve, protože se změnilo jejich umístění v listboxu!

V dané větvi je pak kód, který se provádí, je-li vybrána požadovaná položka v listboxu. Pro tradiční volání kódovacího algoritmu, pro který je tento blok kódu určen, je pak nutné zavolat správně jak samotný algoritmus, tak funkce pro jeho zobrazení, jinak jeho provedení postrádá smysl. Tyto funkce jsou například:

- `zobrazit_zakodovanou_zpravu(zakodovana_zprava,handles)`
- `zobrazit_parametry(pravdepodobnosti,zprava_k_zakodovani,zakodovana_zprava, handles)`
- `zobrazit_kodovaci_tabulku(kod,handles)`

2.4.2 Tvorba nového kódovacího algoritmu

Při tvorbě nového algoritmu je třeba dodržet několik zásad, ať už se jedná o programátorské zásady či o závazná pravidla pro správnou funkci programu.

Z hlediska programátorských zásad je například nutné zejména do funkce provádějící a kódování vložit opravdu jen nutné minimální jádro. Podpůrné funkce pak musejí být zvlášť, tím se umožní opětovné využívání daných funkcí tam, kde budou znovu potřeba a zmenší se délka a nepřehlednost kódu. Podpůrným funkcím se nedá spolehlivě vyhnout, vzhledem k tomu, že každý algoritmus má trochu jiné požadavky na jeho vizualizaci a podobně, které nejdou dokonale standardizovat.

Dále je potřeba dodržovat jistá závazná pravidla pro to, aby program správně fungoval a aby bylo možno využívat již naprogramované funkce. Těmito funkcemi mohou být například funkce pro výpočet parametrů, zobrazení kódovací tabulky a podobně. Sice není nemožné, aby si programátor napsal tyto funkce vlastní a nevyužíval již napsaných funkcí. Je to ale zcela zbytečné a znepřehlední to kód. Celý proces práce s kódovacím algoritmem je navržen tak, aby byl kód v algoritmu co nejvíce samostatný a zároveň šlo využívat stejných funkcí pro rozdílné algoritmy. Pro to, aby toto mohlo fungovat, je třeba striktně dodržet vstupní a výstupní parametry funkce.

Každá funkce by měla vracet (alespoň) proměnné `kod` a `zakodovana_zprava`, ty jsou dále využívány pro zobrazení kódovací tabulky, zobrazení zakódované zprávy a zobrazení parametrů.

Vstupní parametry funkce jsou závislé na kódovacím algoritmu jako takovém, každá funkce musí znát hodnotu proměnné `zprava_k_zakodovani`, ne každá však potřebuje pro svůj chod znát hodnoty pravděpodobností apod.

Kromě vizualizace daného algoritmu, která je unikátní povahy, není třeba (a není to ani kýžené), aby programátor nastavoval, nebo četl hodnoty přímo z prvků GUI, k tomu jsou již vytvořené funkce, které vracejí nebo nastavují hodnoty blíže předem definovaným způsobem, viz kapitola 2.5. Chce-li tedy programátor využít ve svém algoritmu vektoru pravděpodobností, měl by použít proměnnou `pravdepodobnosti`, která je získána voláním funkce `[abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti]=nacist_data(handles)`, místo toho, aby si vytvářel vlastní vektor načítaný z nějakého prvku GUI.

2.5 Referenční rejstřík důležitých funkcí

2.5.1 nacist_data

Popis:

Funkce slouží k načítání vstupních parametrů pro kódování.

Převádí tyto načtené proměnné z textové podoby do podoby použitelné programem.

Výstupní proměnné:

Abeceda zprávy, četnosti, pravděpodobnosti a zpráva určená k zakódování.

Vstupní proměnné:

Interní proměnná **`handles`**, přes kterou se předávají informace k ovládání prvků GUI.

Dalšími vstupy jsou proměnné předávané skrze prvky GUI, a to konkrétně prvky s ID:

```
text_abeceda
text_zpravakzakodovani
text_cetnosti
text_pravdepodobnosti
```

Algoritmus:

Funkce načítá data z textových polí GUI a rozděluje vstupní řetězec na vektor prvků podle mezer. Ta buďto načte přímo tak, jak jsou zadaná, nebo je vypočítá ze zprávy určené k zakódování `(zprava_k_zakodovani)`. Jestli je má načíst přímo, nebo vypočítat pozná podle návratové hodnoty funkce `vyber_zadavani(handles)`.

Volá funkce:

```
vyber_zadavani()
```

2.5.2 vyber_algoritmu

Popis:

Vrací číselnou hodnotu vybraného algoritmu v pořadí, v jakém jsou algoritmy seřazeny v listboxu.

Výstupní proměnné:

`return_val` (1 – huffmanovo kódování, 2 – aritmetické kódování, 3 – rle kódování, 4 – komprese bloků dat, 5 – lzw kódování, 6 – deflate, 7 – srovnání algoritmů)

Vstupní proměnné:

```
handles
```

2.5.3 vyber_zadavani

Popis:

Vrací číselnou hodnotu vybrané metody zadávání a zároveň zneaktivňuje pole, která nemají být použita.

Výstupní proměnné:

`return_val` (1 – vypočítat, 2 – zadat četnosti, 3 – zadat pravděpodobnosti)

Vstupní proměnné:

`handles, prekreslit`

2.5.4 proved_algoritmus

Popis:

Funkce slouží k obecnému provádění algoritmů. Nejdříve vymaže obsah všech polí, vyplněných při výpočtech, následně načte data do proměnných a zjistí, který z algoritmů se má provést. Zavolá funkci daného algoritmu a zobrazí všechny příslušné hodnoty, jako jsou kódovací tabulky apod.

Výstupní proměnné: -

Vstupní proměnné:

`handles`

Algoritmus:

Data se načítají do proměnných, do proměnné se také načte hodnota algoritmu, který se má spustit a porovnává se ve switch podmínce.

Volá funkce:

```
nacist_data()
vyber_algoritmu()
huffmanovo_kodovani()
zobrazit_zakodovanou_zpravu()
zobrazit_parametry()
zobrazit_kodovaci_tabulku()
aritmeticke_kodovani()
rle_kodovani()
ucinnost_kodovani_pomerna()
komprese_bloku_dat()
lzw_kodovani()
deflate_kodovani()
srovnani_algoritmu()
```

2.5.5 komprese_bloku_dat

Popis:
Provádí kompresi bloků dat a vrací kódovací tabulku a zakódovanou zprávu.

Výstupní proměnné:

```
kod,zakodovana_zprava
```

Vstupní proměnné:

```
abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti,handles
```

Algoritmus:

Funkce prochází vstupní řetězec a spojuje jednotlivé znaky do dvojic. Je-li lichý počet znaků, poslední znak je spojen s prázdným řetězcem. Dvojice znaků jsou následně zakódovány rovnoměrným kódováním a výsledným kódem se zakóduje vstupní zpráva speciální funkcí, která kóduje dvojice podle slovníku.

Volá funkce:

```
zakodovat_rovnomerne()
zakodovat_dvojice_podle_slovniku()
```

2.5.6 lzw_kodovani

Popis:

Provádí kódování pomocí LZW algoritmu a vrací kódovací tabulku a zakódovanou zprávu.

Výstupní proměnné:

```
kod,zakodovana_zprava
```

Vstupní proměnné:

```
abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti,handles
```

Algoritmus:

Nejdříve načte základní abecedu zadanou uživatelem. Následně prochází řetězec a porovnává, jestli je znak obsažen ve slovníku, pokud je, pokračuje dál v procházení, přičemž si pamatuje všechny takto prošlé znaky. Ve chvíli, kdy narazí na řetězec, který není ve slovníku, zapíše jej do slovníku a vymaže tento řetězec z paměti, pokračuje následujícím znakem. Takto prochází řetězec, dokud nenarazí na konec. Až dojde na konec řetězce, je vytvořen slovník, ten pošle univerzální funkci pro rovnoměrné

kódování, která mu vrátí kód. Pomocí kódu pak zakóduje speciální funkcí na kódování pomocí slovníku vstupní řetězec.

Volá funkce:

```
zakodovat_rovnomerne()
zakodovat_podle_slovniku()
```

2.5.7 deflate_kodovani

Popis:

Provádí kódování pomocí metody DEFLATE, což je spojení algoritmu LZ77 a Huffmanova kódování. V této části se budu zabývat pouze LZ77. Funkce vrací kódovací tabulku a zakódovanou zprávu.

Výstupní proměnné:

```
kod, zakodovana_zprava
```

Vstupní proměnné:

```
abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti,handles
```

Algoritmus:

Na začátek vstupního řetězce vloží ukazatel, který rozděluje zprávu určenou k zakódování a již zakódovanou zprávu, na kterou se budou vytvářet odkazy. Program staticky přidělí prvnímu prvku hodnotu 0,0 a první znak, protože je zřejmé, že první znak nemůže být ve slovníku. Ukazatel se posune za tento první znak. Zjistí, je-li řetězec navazující na ukazatel obsažen v řetězci před ukazatelem. Toto zjišťuje tak, že porovnává celou délku řetězce a vyhledává podřetězec v části nalevo od ukazatele a při každé iteraci řetězec zkracuje o jeden znak, dokud nedojde ke shodě. Pokud není nalezena shoda, zapíše hodnotu 0,0 a první znak za ukazatelem a posune za něj ukazatel, hodnota ukazatele byla dočasně uložena. Pokud je nalezena shoda, rozkopíruje řetězec před ukazatelem spojený s onou shodou a opakuje proces. Do slovníku se zapíše nový záznam: první hodnotou slovníku je vzdálenost od dočasně uloženého ukazatele k prvnímu znaku shody, druhou hodnotou slovníku je délka shody až po místo, kde se řetězce rozcházejí, včetně rozkopírované zprávy, třetí hodnotou je následující znak. Na místo, kde se přestane shodovat rozkopírovaný řetězec se vstupním, se přesune ukazatel. Proces se opakuje, dokud nedojde ke konci řetězce. Jakmile je vytvořen celý slovník, aby bylo možné jej poslat k Huffmanovu kódování, zjistí se unikátní záznamy ve slovníku a jejich četnosti a pravděpodobnosti, záznamy se převedou na typ znak, se

kterým už může funkce Huffmanova kódování pracovat. Kód zakódovaný Huffmanovým kódováním převede zpět do původní podoby a zakóduje vstupní řetězec binárním vyjádřením (Huffmanovým kódem) LZ77 slovníku.

Volá funkce:

```
lz77()
huffmanovo_kodovani()
zprava_k_zakodovani()
```

2.5.8 nejvetsi_mozna_shoda

Popis:

Pomocný algoritmus k LZ77.

Výstupní proměnné:

```
delka, pozice
```

Vstupní proměnné:

```
zprava_k_zakodovani, plovouci_okno
```

Algoritmus:

Viz deflate_kodovani

2.5.9 aritmeticke_kodovani

Popis:

Provádí aritmetické kódování jednotlivých prvků.

Výstupní proměnné:

```
kod, zakodovana_zprava
```

Vstupní proměnné:

```
abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti,handles
```

Algoritmus:

Vkládá do tabulky vypočítané hodnoty podle teorie aritmetického kódování. Nejdříve pro první symbol abecedy, následně v cyklu pro zbývající symboly abecedy.

Volá funkce:

```
zakodovat_podle_slovniku()
```

2.5.10 pravdepodobnost_symboluf

Popis:

Vrací pravděpodobnost konkrétního symbolu s indexem n ve zprávě.

Výstupní proměnné:

```
pravdepodobnost_symbolu
```

Vstupní proměnné:

```
abeceda,zprava_k_zakodovani,pravdepodobnosti, n
```

2.5.11 huffmanovo_kodovani

Popis:

Provádí Huffmanovo kódování. Vrací kódovací tabulku a zakódovanou zprávu.

Výstupní proměnné:

```
kod, zakodovana_zprava
```

Vstupní proměnné:

```
abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti,handles
```

Algoritmus:

Seřadí dvojice znak-pravděpodobnost podle hodnot pravděpodobností. Následně spojí dvě dvojice s nejnižšími pravděpodobnostmi (jak znaky, tak pravděpodobnosti). Opakuje dokud nezbývají dva řádky. Ke každé dvojici na prvním řádku přiřadí hodnotu „1“, ke každé dvojici na druhém řádku přiřadí hodnotu „0“.
Následně podle abecedy prochází strom a zjišťuje, je-li znak přítomen v prvním nebo druhém řádku. Podle toho přiřazuje hodnotu danému znaku, není-li ani v jednom, nepřiřazuje žádnou. Strom prochází celý, nehledě na to, jestli se na řádku vyskytuje znak samotný, nebo společně s ním jsou na řádku přítomné i další znaky.

Volá funkce:

```
zakodovat_z_huffmanova_slovniku()
zakodovana_zprava = zakodovat_podle_slovniku()
```

2.5.12 zakodovat_z_huffmanova_slovniku

Popis:

Stará se o vytvoření kódu z Huffmanova stromu (slovníku).

Výstupní proměnné:

`kod`

Vstupní proměnné:

`slovnik,abeceda`

Algoritmus:

Viz huffmanovo_kodovani

2.5.13 rle_kodovani

Popis:

Provádí RLE kódování. Vrací kódovací tabulku a zakódovanou zprávu.

Výstupní proměnné:

`kod,zakodovana_zprava`

Vstupní proměnné:

`abeceda,zprava_k_zakodovani,cetnosti,pravdepodobnosti,handles`

Algoritmus:

Prochází řetězec a počítá počet opakování daného znaku. Jakmile najde jiný znak, zapíše počet opakování a znak do slovníku, opakuje se do konce řetězce.
Výsledný slovník – dvojice hodnot reprezentující opakování a znak, se rovnoměrně zakóduje.

Volá funkce:

```
zakodovat_rovnomerne()
zakodovat_podle_slovnikuRLE()
```

2.5.14 zakodovat_rovnomerne

Popis:

Vytváří rovnoměrné kódy ze sady symbolů.

Výstupní proměnné:

`kod`

Vstupní proměnné:

`sada_symbolu`

Algoritmus:

Zjišťuje unikátní hodnoty v sadě symbolů, ty očísluje od 0 a následně převede do binární podoby s min. počtem cifer (dle vztahu ceil(log2(počet unikátních hodnot))).

2.5.15 zakodovat_dvojice_podle_slovniku

Popis:

Podpůrný algoritmus pro blokové kódování

Výstupní proměnné:

`zakodovana_zprava`

Vstupní proměnné:

`zprava_k_zakodovani, kod`

2.5.16 zakodovat_podle_slovniku

Popis:

Kóduje vstupní řetězec dle přiložené kódovací tabulky.

Výstupní proměnné:

`zakodovana_zprava`

Vstupní proměnné:

`zprava_k_zakodovani, kod`

2.5.17 zobrazit_parametry

Popis:

Volá funkce pro výpočet různých parametrů a zobrazuje je.

Vstupní proměnné:

`pravdepodobnosti, zprava_k_zakodovani,zakodovana_zprava,handles,kod, entropicke`

Volá funkce:

```
entropie()
stredni_delka()
entropie_prizpusobeneho_zdroje()
ucinnost_kodovani()
nadbytecnost()
stredni_delka_zakodovano()
entropie_prizpusobeneho_zdroje_zakodovano()
```

ucinnost_kodovani_zakodovano()
nadbytecnost_kodovani_zakodovano()

2.5.18 nacist_soubor

Popis:

Načte ze souboru data pro kódování.

Načítá ve formátu:

```
pravdepodobnosti
0.066 0.13 0.2 0.26 0.33
zprava
abbcccddddeeeee
cetnosti
1 2 3 4 5
abeceda
a b c d e
```

kde nezáleží na pořadí vstupních dat, ani na jejich přítomnosti – nezadaná data se nenačtou. Je však důležité, aby byly navzájem odlišeny uvozujícím řádkem, který udává, jedná-li se o pravděpodobnosti, zprávu, četnosti, nebo abecedu. Jeden řádek tedy udává název proměnné a hned následující řádek jeho hodnotu.

Výstupní proměnné:
pravdepodobnosti, abeceda, cetnosti, zprava

Volá funkce:

nacist_soubor()
zobrazit_data_ze_souboru()

2.5.19 zakodovat_podle_slovnikuRLE

Popis:

Podpůrný algoritmus pro RLE kódování. Kóduje za pomocí kódovací tabulky RLE algoritmu zprávu.

Výstupní proměnné:

zakodovana_zprava

Vstupní proměnné:

zprava_k_zakodovani, kod

Algoritmus:

Viz rle_kodovani

2.5.20 zmenit_gui

Popis:

Stará se o chování GUI z hlediska viditelnosti a aktivity prvků prostředí v závislosti na proměnné volba (odlišuje algoritmy).

Vstupní proměnné:

`handles, volba`

Volá funkce:

```
restart_gui()
slovniky_gui()
transformace_gui()
srovnani_gui()
```

2.5.21 slovniky_gui

Popis:

Přizpůsobuje GUI pro použití zejména slovníkových kompresních algoritmů. (LZW, DEFLATE)

2.5.22 transformace_gui

Popis:

Přizpůsobuje GUI pro použití zejména transformačních kompresních algoritmů (RLE, komprese bloků dat).

2.5.23 srovnani_gui

Popis:

Přizpůsobuje GUI pro srovnání algoritmů.

2.5.24 restart_gui

Popis:

Uvádí GUI do startovacího (neutrálního) stavu.

2.5.25 ucinnost_kodovani_pomerna

Popis:

Vrací účinnost algoritmu vztaženou ku rovnoměrnému kódování pro danou zprávu.

2.5.26 srovnani_algoritmu

Popis:

Vytváří graf se srovnáním účinnosti jednotlivých algoritmů. Srovnává jen algoritmy, které jsou zavolány a uloženy do dané proměnné.

Volá funkce:

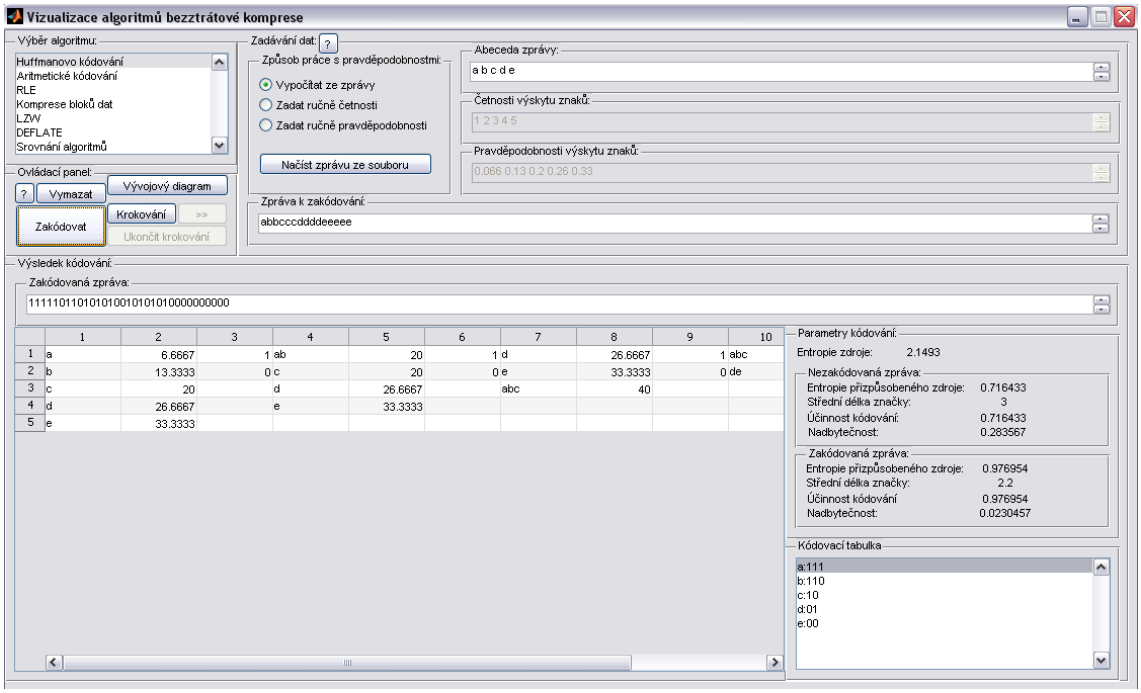
```
nacist_data()
huffmanovo_kodovani()
aritmeticke_kodovani()
rle_kodovani()
komprese_bloku_dat()
lzw_kodovani()
ucinnost_kodovani_pomerna()
```

3 VYTVOŘENÝ PROGRAM

3.1 Grafické prostředí

Grafické prostředí je navrženo tak, aby bylo jeho ovládání intuitivní a logické.

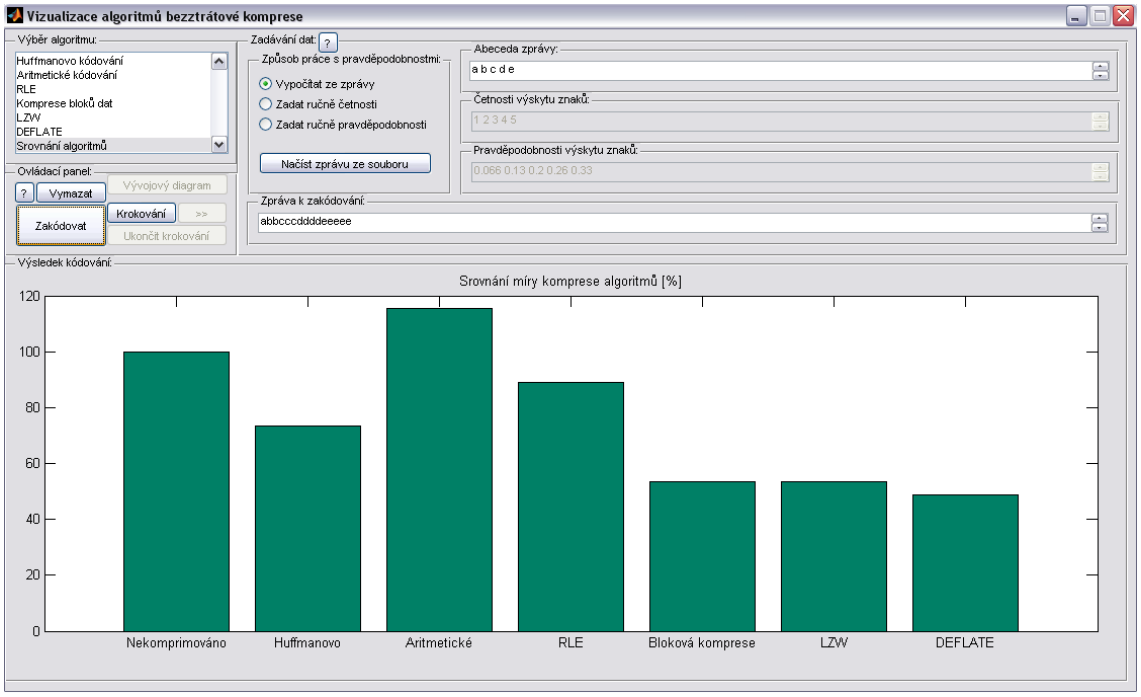
V levém horním rohu je seznam možných algoritmů, kterými může uživatel kódovat. Vedle něj je panel zadávání dat, což odpovídá časovém sledu úkonů. Postupně tak, jak uživatel vybírá možnosti, GUI se přizpůsobuje svým vzhledem tak, aby odpovídalo potřebám konkrétních algoritmů. Zároveň GUI uživatele navádí v tom, jaká políčka či tlačítka je potřeba vyplnit a kterých si nemusí všímat. U každého z bloků je tlačítko nápovědy, které poskytuje stručný popis a vysvětlivky příslušných funkcí.



Obr. 3. 1: Grafické prostřední vyvinutého programu

3.2 Srovnání algoritmů

Funkčnost srovnání algoritmů provádí pro zadanou zprávu o daných parametrech srovnání účinnost algoritmů v procentuálním vyjádření vůči nekomprimovanému vstupu. To umožňuje uživateli jednoduše porovnat dané algoritmy pro různé druhy zpráv a porovnat praktickou funkčnost algoritmů s teoretickými základy. Srovnání je pak zobrazeno graficky v podobě sloupcového grafu.



Obr. 3. 2: Srovnání míry komprese algoritmů ve vyvinutém prostředí

3.3 Vizualizace algoritmů

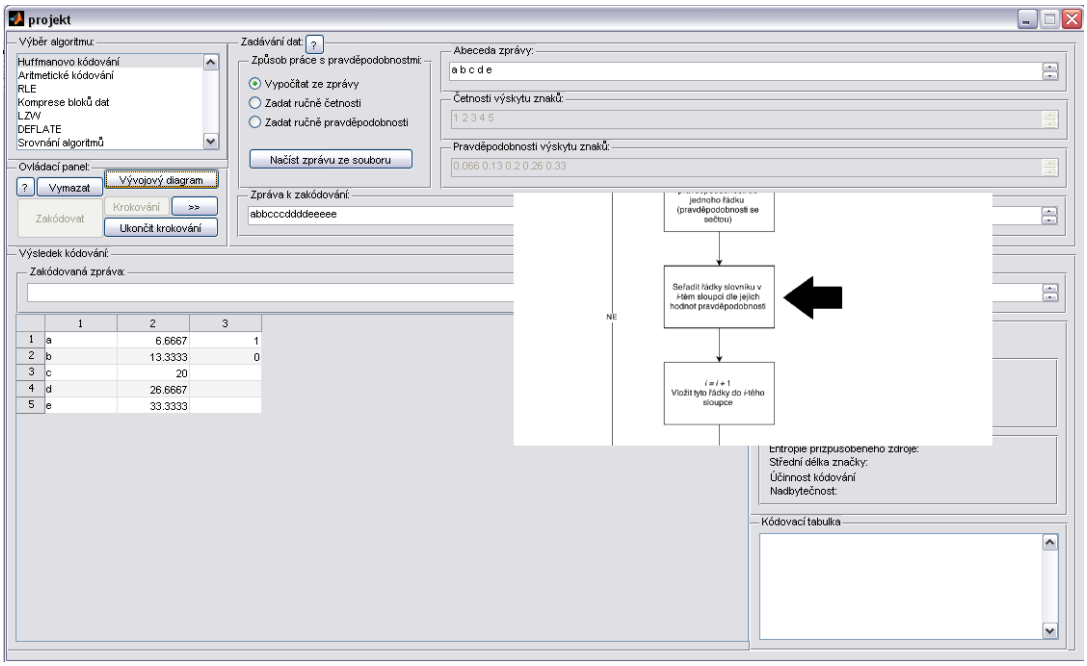
Program umožňuje vizualizovat algoritmus několika různými způsoby.

O každém z algoritmů poskytuje základní informace o parametrech a kódovací tabulku. A pro každý algoritmus lze samostatně zobrazit jeho celý vývojový diagram.

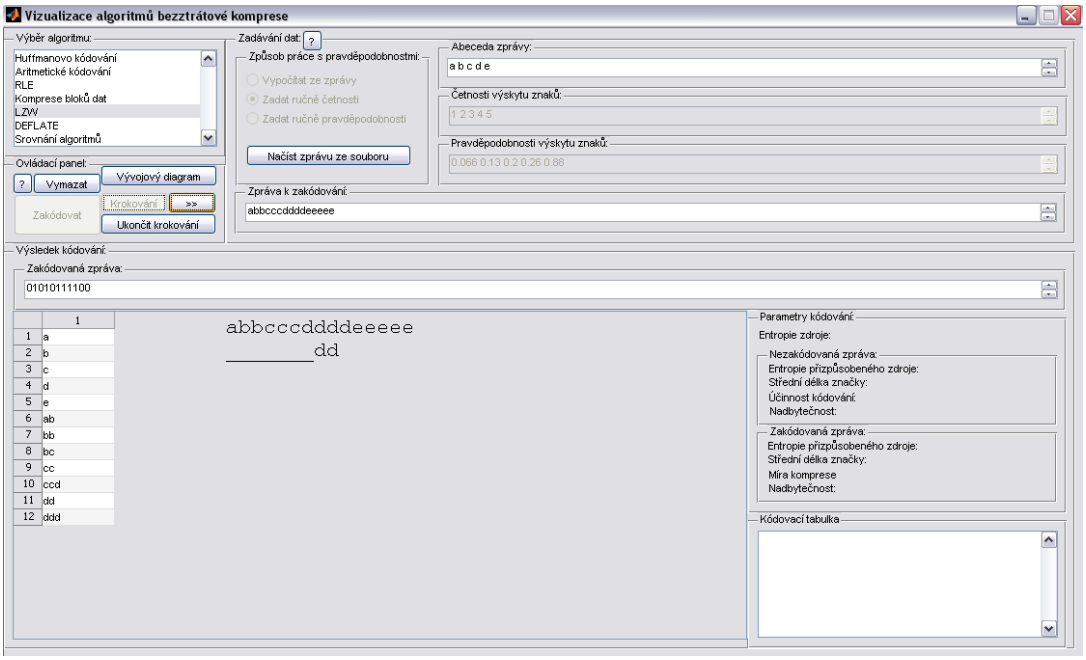
Pro každý z algoritmů je také možné zvolit možnost „Krokování“ a krokovat tak proces komprese po předem definovaných krocích. Na každém z jednotlivých kroků pak lze zobrazit aktuální stav algoritmu na vývojovém diagramu.

Vývojový diagram ukazuje černou šipkou místo, na kterém se algoritmus právě zastavil.

Pro Huffmanovo kódování, aritmetické a LZW je pak detailněji naznačen průběh graficky.



Obr. 3. 3: Krokování a ukázka vývojového diagramu



Obr. 3. 4: Ukázka grafické vizualizace algoritmu

4 ZÁVĚR

Byla vyvinuta aplikace provádějící a znázorňující několik typů kódování, a to Huffmanova kódování, aritmetického, RLE kódování, komprese bloků dat, LZW a DEFLATE. Důkladněji pak bylo vizualizováno Huffmanovo kódování, aritmetické kódování a LZW. Průběh algoritmů lze zobrazit na vývojových diagramech a krokováním. Při zobrazení vývojového diagramu během krokování je ukázán aktuální průběh algoritmu. Dále program zobrazuje parametry kódování a kódovací tabulku. Program také kóduje zadanou zprávu.

Tato aplikace bude moci být využita při výukách datových komunikací a kompresních algoritmů. Lze na ni názorně předvést silné a slabé stránky některých kompresních algoritmů.

Při vývoji aplikace byl kladen silný důraz na znovupoužitelnost, měla by tedy sloužit jako platforma pro vizualizaci dalších algoritmů či jiné podobné použití. Většina funkcionalit má unifikované vstupy a výstupy a při dodržení návrhových pravidel nemá programátor problém přidávat snadno další funkce.

SEZNAM LITERATURY

[1] NEČAS, J. a kol. *Aplikovaná matematika Díl I A-L, Oborové encyklopedie SNTL*. Praha: SNTL, 1977.

[2] BIGGS, N. L. *Codes, An Introduction to Information Comunnication and Cryptography*. Springer, 2008. ISBN 978-1-84800-273-913.

[3] NEČAS, J. a kol. *Aplikovaná matematika Díl II M-Ž, Oborové encyklopedie SNTL*. Praha: SNTL, 1978.

[4] SCHNEIER, B. *Applied Cryptography*. 2. vyd. John Wiley and Sons, 1996. 784 s. ISBN 978-1-119-09672-6.

[5] SCHÜRMANN T., GRASSBERGER P. *Entropy Estimation of Symbol Sequences*. Department of Theoretical Physics, University of Wuppertal, 1996, vol 6.

[6] AMERICAN STANDARDS ASSOCIATION. *American Standard Code for Information Interchange*. 1963.

[7] HUFFMAN, D. A. *Method for the Construction of Minimum-Redundancy Codes.* IRE, 1952, vol. 40, no. 9, s 1098 – 1101. ISSN 0096-8390.

[8] TANK, M.K. *Implementation of Limpel-Ziv algorithm for lossless compression using VHDL*. Springer India, 2011. s. 275-278. ISBN 978-81-8489-989-4.

[9] PROAKIS, J. G. *Digital Communications.* 4. vyd. New York: Mac Graw-Hill, 2000. ISBN 0-7-232111-3.

[10] MURRAY, J.D. a W. VAN RYPER. *Encyclopedia of Graphics File Formats.* 2. vyd. O'Reilly Media, 1996. 1152 s. ISBN: 1-56592-161-5.

[11] FELDSPAR, A. An Explanation of the DEFLATE Algorithm. *Gzip.org* [online]. [cit. 2015-05-31]. Dostupné z: http://www.gzip.org/deflate.html

SEZNAM ZKRATEK

ASCII	American Standard Code For Information Interchange
BWT	Burrows-Wheeler Transform
GUI	Graphical User Interface
GZIP	GNU Zip
ID	Identification
JPEG	Joint Photographic Experts Group
LZ	Lempel-Ziv
LZ77	Lempel-Ziv 77
LZW	Lempel–Ziv–Welch
PNG	Portable Network Graphics
RLE	Run-Length Encoding
TARGA	Truevision Advanced Raster Graphics Adapter
TGA	Truevision Graphics Adapter

OBSAH PŘILOŽENÉHO CD

Program.fig	Soubor grafického prostředí programu
Program.m	Zdrojový kód programu
Bakalářská Práce.pdf	Elektronická verze bakalářské práce
Návod.pdf	Návod k používání programu