

Komprese binárních textových a grafických dat Compression of binary, text and graphical data

Marek Kojecký

Bakalářská práce
2005



Univerzita Tomáše Bati ve Zlíně
Fakulta technologická

Abstrakt

Práce pojednává o metodách komprese binárních, textových a grafických dat. První část je zaměřena na bezztrátovou kompresi RLE, LZW a další. Druhá část se týká okrajově ztrátové komprese, která je reprezentována metodou komprese JPEG. Třetí částí je popis nejpoužívanějších komprimačních programů.

Abstract

This thesis deals with methods of compression of binary, text and graphical data. The first part is focused to lossless compression RLE, LZW and other. The second part concerns about loss-making compression, which is represented by compression method JPEG. The third part is a description of the most used archivers.

Rád by jsem touto cestou poděkoval vedoucímu bakalářské práce Ing. Marku Čandíkovi, Ph.D. za odborné vedení, rady a pomoc při řešení problémů souvisejících s danou prací.

Souhlasím s tím, že s výsledky mé práce může být naloženo podle uvážení vedoucího bakalářské práce. V případě publikace budu uveden jako spoluautor.

Prohlašuji, že jsem na celé bakalářské práci pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně 9.6.2005

.....

Podpis

OBSAH

ÚVOD.....	6
1 KOMPRESI DAT.....	7
2 BEZZTRÁTOVÁ KOMPRESI.....	10
2.1 NEJLEPŠÍ MOŽNÁ BEZZTRÁTOVÁ KOMPRESI.....	10
2.2 NEJLEPŠÍ BEZZTRÁTOVÝ ALGORITMUS	11
2.3 KOMPRESI RLE (RUN-LENGTH ENCODING)	11
2.3.1 Bitová úroveň.....	14
2.3.2 Bajtová úroveň	14
2.3.3 Pixelová úroveň.....	15
2.4 KOMPRESI LZW (LEMPER-ZIV-WELCH)	15
2.4.1 LZ77	16
2.4.2 LZ78	17
2.5 HUFFMANOVO KÓDOVÁNÍ	20
2.6 SHANNON – FANOV KÓDOVÁNÍ.....	24
2.7 ARITMETICKÉ KÓDOVÁNÍ	25
2.8 BURROWS-WHEELEROVA TRANSFORMACE	28
3 ZTRÁTOVÁ KOMPRESI.....	31
3.1 JPEG.....	31
3.1.1 Progresivní JPEG	33
3.1.2 Transparentní JPEG	34
3.1.3 Postup komprese	35
3.2 KRÁTCE O DALŠÍCH FORMÁTECH ZTRÁTOVÉ KOMPRESI.....	37
3.2.1 Formát MPEG-1 audio	37
3.2.2 Windows Media Audio (WMA)	37
4 PŘEHLED POUŽÍVANÝCH PROGRAMŮ PRO KOMPRESI DAT	38
4.1 WINZIP.....	38
4.1.1 Instalace.....	39
4.1.2 Vzhled programu.....	39
4.1.3 Integrace do Windows.....	40
4.1.4 Průvodce.....	41
4.2 POWER ARCHIVER.....	41
4.3 WINRAR.....	42
ZÁVĚR.....	44
SEZNAM POUŽITÉ LITERATURY	45
SEZNAM OBRÁZKŮ	46
SEZNAM TABULEK.....	47

ÚVOD

Vytváření, uchovávání, sdílení informací a jejich přenos, ať již pomocí drátové či bezdrátové technologie, přenos údajů pomocí médií, se v posledních letech stává pevnou součástí našich životů. Nárůst využívání Internetu jako zdroje poučení, informací i zábavy a elektronického obchodování, mobilní telefonie a postupné přerůstání a sbližování komunikačních a informačních technologií a využívání multimediálních informací, vede ke zvyšování nároků na rychlost, kvalitu a bezpečnost přenosu.

Použití a přenos běžně používaných informací, jako je zvuk, obraz a video v digitálním tvaru vyžaduje použití rozsáhlých pamětí a velkých šířek přenosového pásma. To by mělo za následek zvyšování nákladů na uchovávání a přenos informací v této podobě.

Pokud si chceme uchovat výhody digitálního tvaru informace a omezit náklady na uchovávání a přenos informací, máme k dispozici technologii komprimace. Komprimace umožňuje efektivní digitální reprezentaci zdrojových signálů, jako je text, zvuk, obraz nebo video, použitím redukovaného počtu prvků digitální informace (bitů), než má originál. Pokud má být použití komprimace efektivní, musí umožňovat reprodukci komprimované informace v požadované kvalitě.

Požadovaná kvalita může být v rozpětí od požadavku úplné rekonstrukce původní informace (například u textové informace), až po omezení přenášeného frekvenčního pásma (zvuková informace), počtu zobrazovaných barev (obrazová informace) nebo počtu rozlišitelných zobrazovaných prvků informace (obraz, video).

Současný rozvoj digitálních přenosových zařízení a mobilních telefonů, u kterých dochází k prorůstání s Internetem, a poskytování nejen hlasových, ale i textových a multimediálních služeb mobilním uživatelům přinesl zvýšený zájem o metody efektivního přenosu informací v digitální podobě a tím i o metody komprimace.

Většina metod komprese funguje na fyzické úrovni vstupních dat, tj. nahrazuje vstupní proud bitů nebo bajtů a nestará se o jejich informační obsah a souvislosti. Takovéto metody transformují vstupní proud informací na jiný, pokud možno kratší výstupní proud podle kódovacího algoritmu, bez jakéhokoliv využití vědomostí o složení a možné následnosti kódovaných informací. Metody komprimace na logické úrovni tyto znalosti umí využívat a díky tomu mohou dosahovat vyšších poměrů komprimace. Obvykle jsou použitelné jen pro jistý specifický typ dat.

1 KOMPRESSE DAT

Zjednodušeně lze říct, že komprimace (komprese) je metodou efektivní reprezentace zdrojového signálu v digitálním tvaru. Tuto stručnou definici můžeme rozšířit přidáním požadavku na zachování věrnosti tohoto signálu. Komprimace je tedy metodou reprezentace zdrojového signálu v digitálním tvaru nejmenším možným počtem bitů při zachování požadavku přijatelné ztráty věrnosti. Zdrojovým signálem, jak už bylo uvedeno, může být text, obraz, zvuk, video anebo jakýkoliv jiný signál, který potřebujeme uchovávat a přenášet.

Před samotnou komprimací je důležité uvážit, které algoritmy se pro daný druh souboru budou hodit. Mnohem účinnější je použít několik algoritmů najednou. Proto se také algoritmy komprese dělí na primární a sekundární. Pořadí algoritmů není pevně dáno, proto se provádí důkladná analýza souboru. Měli bychom vědět, který algoritmus použít jako první a který použít na závěr.

Další druh kompresních algoritmů tvoří tzv. "Slovníkové (kontextové) metody", které vytváří slovníkovou databázi posloupností bajtů. Při dalším výskytu oné posloupnosti stačí uvést jenom odkaz do slovníku. Existuje také spousta algoritmů, které samotný soubor nekomprimují, ale které ho připraví pro další komprimaci (Burrows Wheeler, Move To Front). Většina algoritmů (včetně jejich modifikací) je patentována.

Na vyjádření efektivnosti metod komprimace můžeme použít několik ukazatelů:

- Poměr komprimace je definován následovně

$$\text{Poměr komprimace} = \frac{\text{Rozměr vstupního proudu}}{\text{Rozměr výstupního proudu}} \quad (1)$$

- Faktor komprimace je převrácenou hodnotou poměru komprimace

$$\text{Faktor komprimace} = \frac{\text{Rozměr výstupního proudu}}{\text{Rozměr vstupního proudu}} \quad (2)$$

Komprimace se dá rozdělit podle několika základních hledisek:

- bezztrátová a ztrátová komprimace
- fyzická a logická komprimace
- symetrická a asymetrická komprimace
- adaptivní a neadaptivní komprimace

Bezztrátová a ztrátová komprimace

Bezztrátová komprimace se používá všude tam, jak už název napovídá, kde si nelze dovolit jakoukoliv ztrátu dat. Tento typ používají známé pakovací (od angl. pack - zabalit) programy jako Winrar či Winzip. Stejně tak se používá při přenosu dat na internetu.

Ztrátová komprese se používá tam, kde ztráta některých informací nevadí. Používá se například u obrázků, videa nebo zvuku. Její princip je založen na nedokonalosti lidských smyslů, tedy uší a očí. Není to tak, že by jste ve své oblíbené muzice přišli třeba o bicí. Ztrátovou kompresi většinou ani nepoznáte.

Fyzická a logická komprimace

Rozdíl mezi fyzickou a logickou komprimací spočívá v tom, zda komprimační algoritmus při komprimaci přihlíží nebo nepřihlíží k logické informační hodnotě komprimovaných dat.

Logická komprimace používá logické substituce sekvence znaků jinou, úspornější řadou. Konkrétním příkladem jsou zkratková slova jako Čedok (nahrazující někdejší plný název Československá dopravní kancelář) nebo Svazarm (Svaz pro spolupráci s armádou).

Fyzická komprese probíhá bez zřetele na logiku dat, se kterými se manipuluje. Vytváří se nová sekvence znaků (bajtů, bitů atd.), jejíž vztah k původním datům lze rozpoznat výhradně s použitím nekomprimačního algoritmu. Bez znalosti tohoto algoritmu je informační hodnota komprimovaných dat nulová.[4]

Symetrická a asymetrická komprimace

Toto rozdělení je založeno na porovnání množství práce, která se podle algoritmu vykoná při kompresi a dekompresi dat.

Pokud je doba (a tím většinou i počet a druh operací) potřebná pro kompresi a dekompresi dat přibližně stejná jedná se o symetrickou kompresi.

Některé algoritmy jsou však záměrně konstruovány jako asymetrické. Většina takových komprimačních algoritmů provede větší množství operací při kompresi dat. Krátká doba dekomprimace je výhodná například tam, kde soubor dat ukládáme na disk sice v komprimované podobě, ale často s ním pracujeme.

Adaptivní a neadaptivní komprimace

Tyto algoritmy se rozlišují podle své schopnosti přizpůsobit se charakteru dat, se kterými pracují.

Neadaptivní algoritmy jsou určeny výhradně pro komprimaci specifického druhu dat. Většinou obsahují předdefinované slovníky nebo řetězce znaků, o kterých je známo, že jejich pravděpodobnost výskytu v souborech dat je vysoká.

Adaptivní algoritmus je naproti tomu schopen dosáhnout určité nezávislosti na komprimovaných datech. Takové algoritmy neobsahují žádné statické slovníky řetězců, ale budují si je pro každý komprimovaný soubor dat znovu dynamicky v průběhu kódování.

Obecně lze říci, že adaptivní algoritmy platí za svou přizpůsobivost a větší šíři použití menší rychlostí ve srovnání se specializovanými neadaptivními algoritmy. To však jistě nikoho nepřekvapí.

2 BEZZTRÁTOVÁ KOMPRESSE

Bezztrátová komprimace (lossless compression) spočívá v tom, že je možné rekonstruovat dokonale původní signál zdroje informací. Bezztrátová komprese nám zajistí, že po rozbalení o žádná data nepřijde. S bezztrátovou kompresí se můžeme setkat hlavně při datových přenosech (usnadňuje přístup k Internetu většině z nás) a zálohování dat.

2.1 Nejlepší možná bezztrátová komprese

Představa, že s vylepšováním kompresních algoritmů budou soubory stále více zabalitelné, je naprosto mylná, protože nelze udělat komprimátor, který by byl univerzální, protože hodnota informace obsažená ve výstupním souboru je vždy stejná, nebo větší té původní. Dnešní algoritmy se pouze snaží zapsat daný druh souboru co možná nejefektivnějším způsobem. Proto se také setkáváme s jednoúčelovými algoritmy pro daný druh souboru. Ani v tomto případě však k žádné skutečné kompresi nedochází, protože hodnota informace původního nezabaleného souboru se nezmění ani v případě, že pro daný soubor použijeme dokonalou kompresi.

Dokonalá komprese je v obecném případě ponechat soubor nekomprimovaný. Při běžné komprimaci však k dokonalé kompresi v 99% nedochází. Pokud komprese není dokonalá, tak dochází ke zbytečnému přidávání informací a znehodnocování souboru. V případě, že použijeme nedokonalou kompresi (např. Huffman), se může stát, že zmenšíme velikost původního souboru, ale zdvojnásobíme informaci v souboru obsaženou. Pokud bychom nejprve použili dokonalou kompresi, tak bychom dosáhli mnohem menší velikosti. Po této nedokonalé metodě se zhorší komprimovatelnost souboru.[6]

Naší snahou je vytvořit dokonalou kompresi pro určitý druh souborů. O žádném kompresním algoritmu nemůžeme říci, že je dokonalý, pokud to není přímo zřetelné a matematicky odvoditelné. Můžeme to také teoreticky tvrdit o aktuálně nejlepším algoritmu. Přestože použijeme nedokonalý algoritmus, můžeme dosáhnout dokonalého výsledku - např. řetězec XYYY, kde X a Y je náhodné číslo od 0 do 255 a X je různé od Y, půjde v nejlepším případě zabalit na 3 bajty. V tomto případě můžeme říci, že se jedná o 100% dokonalou kompresi pro daný řetězec. Pro řetězec XYZ, kde X,Y a Z se sobě nerovnají a nabývají hodnot od 0 do 255, je však dokonalá komprese XYZ. Z uvedených příkladů je jasné, že na stále se zlepšující algoritmy můžeme zapomenout.

2.2 Nejlepší bezztrátový algoritmus

Účinnost většiny bezztrátových metod se hodnotí podle schopnosti komprimovat text. Snahou je maximálně využít vztahů mezi jednotlivými bajty souboru. Pokud nám jde o závislost mezi bajty, musíme její vstupní informace stavět na té samé úrovni. Pokud víme o vztahu bajtů mezi sebou navzájem a rozhodneme se vstup převést např. do binární reprezentace, pouze zhoršíme kompresní poměr. V dnešní době dosahuje nejlepších výsledků při komprimaci textu algoritmus PPM.

Za nejlepší komprimátor dneška je považován např. DURILCA Dmitrye Shkarina. Známý je rovněž PPMonstr. Pro komprimaci velkých souborů se docela hodí Burous-Wheelerova transformace. Následné zpracování takto transformovaného souboru sice nedosahuje kvalit PPM, ale stále se objevují nové lepší a lepší metody (hlavně díky tomu, že transformace stále není dostatečně prozkoumána a důsledky promyšleny). Úspěšnost většiny algoritmů se porovnává většinou na normovaných archívech - např. Calgary Corpus.

2.3 Komprese RLE (Run-length Encoding)

Jedná se o komprimační algoritmus, který lze použít na jakýkoli druh dat. Na charakteru těchto dat ovšem velmi silně závisí dosažený kompresní poměr. Ačkoli metoda RLE se ve většině případů nemůže pochlubit tak dobrými kompresními poměry jako jiné složitější algoritmy, jeho výhoda spočívá ve snadné instalaci, jednoduchosti celého algoritmu a s tím související poměrně vysoké rychlosti komprimace a dekomprimace.

Základním principem komprimace metodou RLE je zhuštění opakovaných znaků, které se v souboru vyskytují hned za sebou. Řetězec opakujících se znaků se nazývá proud (odtud se také RLE nazývá proudové kódování). Tento proud znaků je vždy zkomprimován do formy jednoho paketu RLE. Paket musí vždy obsahovat dvě informace:

- Proudové číslo udávající počet znaků proudu snížený o jedničku a následující
- proudovou hodnotu, jež se shoduje s hodnotou opakujícího se znaku v proudu

Pokud soubor obsahuje následující sekvenci znaků,

AAAkkRRRRRm

Zakóduje se tato sekvence do podoby:

3A1k4R0m

V uvedené posloupnosti znaků rozpozná komprimační algoritmus čtyři proudy a přiřadí jim čtyři pakety. Tyto pakety jsou:

- 3A
- 1k
- 4R
- 0m

Prvním údajem v paketu je vždy proudové číslo (počet znaků proudu snížený o jedničku). Tento údaj je následován proudovou hodnotou znaku.

Zmenšení velikosti souboru bylo dosaženo výhradně díky proudům znaků obsahujícím více než dva znaky. Pokud se proud skládá pouze z jediného znaku, komprimovaný paket RLE je dokonce delší než původní proud.

Metoda RLE není vhodná pro komprimaci textových souborů. Kompresní poměr by byl v takovém případě velice špatný. Stejně by dopadla komprimace většiny binárních souborů. Zbývají tedy jen obrázky.

Metoda RLE je vhodná pro komprimaci jednoduchých obrázků s malou barevnou hloubkou (maximálně 256 barev). Takové obrázky se vytvářejí například v programu Paintbrush (Malování). Tyto obrázky mají často jednobarevné pozadí, které zaručuje dostatečnou velikost jednotlivých proudů. U těchto obrázků je možné dosáhnout kompresního poměru až 70%, tj. kompresní poměr odpovídá komprimaci dat na 30% původní velikosti.

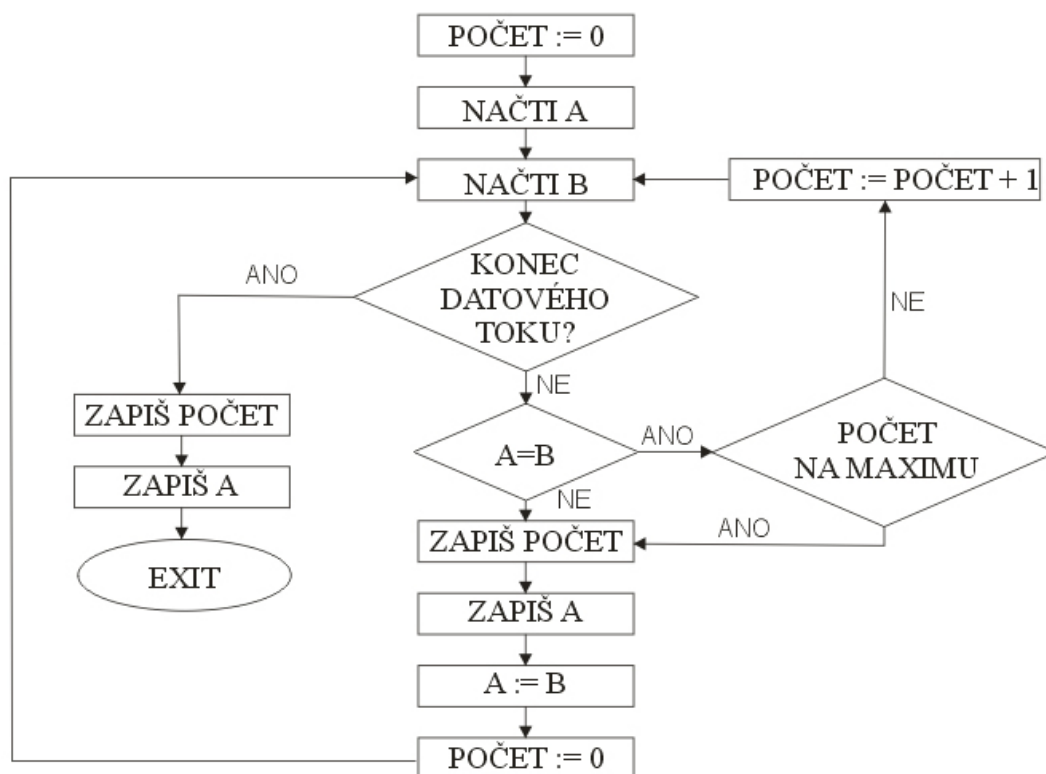
Pro komprimaci fotografií se komprimační metoda RLE nehodí. Fotografie totiž na první pohled nesplňují podmínku dlouhých proudů.

Vzhledem k tomu, že praktický efekt má metoda RLE téměř výhradně při jejím použití na jednoduché obrázky s malou barevnou hloubkou, zaměřím se při dalším popisu rozdílných variant algoritmu RLE výhradně na tato grafická data.

Existují tři základní přístupy k praktické realizaci algoritmu RLE ve vztahu k bitovým mapám jednoduchých obrázků. Ve všech těchto přístupech jsou obvykle původní data předlohy proudově kódována sekvenčním postupem, který přistupuje k datům jako k jednorozměrnému poli dat a nikoli jako k dvourozměrné mapě dat.

Bitová mapa se začíná číst a komprimovat od levého horního rohu a postupuje se buď po řádcích nebo sloupcích jednotlivých pixelů až do pravého dolního rohu.[1]

Při komprimaci je však vhodné udržovat informaci o koncích jednotlivých řádků (respektive sloupců) bitové mapy. Tyto značky lze potom s výhodou použít, pokud je potřeba rozbalit pouze určitou část obrázku (např. 101. až 180. řádek původní předlohy). V takovém případě dekomprimační algoritmus nemusí dekomprimovat prvních 100 řádků obrázku pouze proto, aby našel požadovaný začátek výřezu. Dekomprimace může začít přímo na 101. řádku. Bohužel některé programy podporující grafické formáty s kompresí RLE informaci o konci řádků či sloupců do komprimované podoby obrázků neukládají.



Obr. 1. Základní schéma algoritmu RLE

Praktické přístupy ke kódování bitových map se dělí podle typu základních datových prvků tvořící proudy. Podle typů těchto základních prvků existují tři praktické modifikace:

- bitová úroveň
- bajtová úroveň
- pixelová úroveň

2.3.1 Bitová úroveň

Bitová úroveň kódování metodou RLE rozeznává pouze dva proudy znaků – proud jedniček a proud nul. Taková metoda je účinná prakticky pouze na monochromatické jednobitové obrázky. Tato data obsahují dostatečně dlouhé proudy jedniček i nul k tomu, aby metoda RLE v této modifikaci dosáhla dobrých kompresních poměrů.

Jedná se o jediný bajt logicky rozdělený do dvou částí. Nejvýznamnější bit každého bajtu určuje proudovou hodnotu (tj. 1 nebo 0) a sedm méně významných bitů představuje proudové číslo udávající počet opakování proudové hodnoty snížený o 1. Je zřejmé, že délka proudu musí být vždy v rozsahu 1 – 128 znaků, neboť do sedmi bitů lze zapsat číslo 0 – 127. To je také důvod, proč se jako proudové číslo používá počet znaků proudu snížený o jedničku a nikoli přímo počet znaků proudu. Pokud původní nekomprimovaná data obsahují proud delší než 128 znaků, musí být tento proud při kódování rozdělen na dva nebo více proudů. Na závěr je nutné podotknout, že při komprimaci tato metoda nerozeznává hranice jednotlivých bajtů. Ty nejsou v tomto případě zajímavé.

2.3.2 Bajtová úroveň

Tato modifikace kóduje proudy opakujících se bajtových hodnot a naopak si nevšímá dělení na bity nebo hranic šestnáctibitových (případně 32 bitových) slov.

Paket RLE se v tomto případě skládá ze dvou bajtů. První bajt udává proudové číslo v rozsahu 0-255 a druhý bajt proudovou hodnotu. Její rozsah je samozřejmě opět 0 – 255. Zde tedy může velikost jediného proudu, kterému bude odpovídat jeden RLE paket, dosahovat délky až 256 znaků.

Pro dosažení lepších kompresních poměrů bývá vhodné doplnit bajtovou metodu o schopnost ukládat přesné nekódované jednobajtové proudy do kódovaného toku dat. V této modifikaci bajtové úrovně se pro vyjádření proudového čísla využívá pouze sedm méně významných bitů prvního bajtu paketu. Maximální velikost proudu je tak opět snížena na 128 znaků. Nejvýznamnější bit je indikátorem typu proudu. Pokud je tento bit nastaven na 1, jedná se o klasický dvoubajtový paket RLE, kde druhý bajt obsahuje proudovou hodnotu. Při nastavení nejvýznamnějšího bitu prvního bajtu na hodnotu 0 následuje za proudovým číslem přesný proud znaků, který se čte v té podobě, v jaké je zapsán. Jedná se o tzv. přímý paket. Používáním přímých paketů se zamezí tzv. záporné kompresi, při které kódování jednobajtových proudů dat představuje větší velikost komprimovaných dat oproti

jejich původní formě. Metoda RLE na úrovni bajtů je vhodná pro obrázky, které jsou uloženy ve formě jeden bajt na pixel.

2.3.3 Pixelová úroveň

Tato metoda se používá u složitějších obrázků, kdy je jediný pixel reprezentován více než jedním bajtem dat. Velikost paketu RLE se v tomto případě liší podle počtu bajtů na pixel. Pokud např. obrázek podporuje schéma dat „3 bytes per pixel“ (3 bajty na pixel), bude paket RLE čtyřbajtový – první bajt jako proudové číslo a následující tři bajty jediné proudové hodnoty. Kódovací metoda zůstává stejná jako u jednobytové úrovně a opět lze využít modifikace s indikátorem přímého paket

2.4 Komprese LZW (Lempel-Ziv-Welch)

Komprimační algoritmus Lempel-Ziv-Welch (LZW) je jednou z nejrozšířenějších komprimačních metod, kterou používají (v různé formě) jak kompresní programy, tak i různé grafické formáty obrázků. Jedná se o tzv. substituční (adaptivní slovníkovou) metodu.

Existují dvě základní schémata tohoto komprimačního algoritmu. Jeho počátky jsou spojeny se jmény Abraham Lempel a Jakob Ziv. Tito pánové vypracovali v letech 1977 a 1978 kompresní algoritmy, které vešly do podvědomí pod zkratkami LZ77 a LZ78. V roce 1984 na jejich práci navázal Terry Welch, který modifikoval kompresní algoritmus LZ78 pro potřeby hardwarových zařízení, konkrétně diskových řadičů. Tak vznikla podoba Lempel-Ziv-Welch algoritmu známého pod zkratkou LZW.

Základním principem tohoto kompresního algoritmu je vyhledávání stejných posloupností bajtů v originálním souboru. Pomocí odkazů na tyto posloupnosti dat algoritmus buduje datový slovník. Komprimace pak probíhá podle následujícího schématu:

- Pokud se posloupnost bajtů (řetězec) ve vytvářeném slovníku nevyskytuje, je tato posloupnost přidána do slovníku a v nezměněné formě zapsána do komprimovaného výstupního toku dat.
- Pokud se vstupní posloupnost bajtů ve slovníku již nachází, zapíše se do výstupního toku dat pouze zástupná hodnota odpovídající nalezené vstupní posloupnosti. Díky tomu, že zástupná hodnota je vždy menší než čtená posloupnost, dochází ke kompresi dat.

2.4.1 LZ77

Komprimační část algoritmu LZ77 funguje tak, že se pokouší vyhledat co nejdelší opakující se posloupnosti znaků. Pokud takovou opakující se posloupnost nalezne, zapíše na výstup pouze odkaz na předcházející výskyt řetězce.

Například vstupní řetězec „Leze po železe“ se zakóduje do podoby „Leze po že[10,4]“. Znaky [10,4] je třeba považovat za schématicky zapsaný offset udávající, že dekodér má z předcházejících deseti znaků vybrat první čtyři.

Podstatou tohoto typu komprese je tzv. posuvné okno (sliding window), které obsahuje koncovou část již přečteného (a zkomprimovaného) textu. V tomto okně se kompresní algoritmus snaží nalézt co nejdelší podřetězec odpovídající řetězi na vstupu. Pokud se to podaří, zakóduje jej v podobě odkazu na tento výskyt. Odkaz musí obsahovat ukazatel na začátek podřetězce a jeho délku. Posuvné okno obsahuje dvě části: prohlížecké okno a aktuální okno (look ahead buffer). Na začátku algoritmu nastaví posuvné okno tak, aby začátek vstupu obsahovalo aktuální okno. Potom pokaždé v posuvném okně najdeme co nejdelší počáteční podřetězec (předponu) řetězce z aktuálního okna začínající v prohlížeckém okně. Tento podřetězec se pak zakóduje v podobě ukazatele na počátek podřetězce v prohlížeckém okně a jeho délku.[6]

Zde je uveden nástin algoritmu popsaného v předcházejícím odstavci:

```
1 while(aktuální okno není prázdné)
2 {
3     najdi informaci (ukazatel, délka) nejdelší předpony aktuálního okna
4     if (délka > minimální délka)
5         {
6             write (ukazatel, délka)
7             posuň posuvné okno o délku znaku doleva
8         }
9     else
10        {
11            write (první znak aktuálního okna)
12            posun posuvné okno o 1 znak doleva
13        }
14 }
```

Uvedený algoritmus je základním algoritmem, od kterého se odvozuje velká skupina modifikací nazývaná třída algoritmů LZ77. Dekomprimace souboru zkomprimovaného metodou LZ77 je velice jednoduchá a rychlá. Vždy, když dekomprimační algoritmus narazí na offset udávající ukazatel a délku řetězce, prostě tento řetězec zkomprimuje na výstup.

2.4.2 LZ78

Zatímco metoda LZ78 vytváří svůj dynamický slovník pomocí odkazů do již komprimovaného textu, vylepšený algoritmus LZW patřící do třídy algoritmů LZ78 používá slovníkové odkazy odlišně. Metoda LZW vytváří dynamicky slovník opakujících se řetězců v průběhu komprimace. Různé modifikace algoritmů třídy LZ78 se liší ve způsobu vytváření slovníku.

Slovníková metoda LZW je založena na následující velmi jednoduché strategii. Algoritmus postupně rozpoznává a ukládá do tabulky řetězce znaků a tyto řetězce nahrazuje ve výstupním textu přirozenými čísly z předem definovaného intervalu. Definice intervalu je závislá na charakteru komprimovaných dat. Například při kódování řetězce znaků zobrazených v osmibitovém zobrazení (znaků ASCII) je prvních 255 čísel vyhrazeno pro zobrazení samostatných znaků z původního souboru. Čísla nad 255 se pak přidělují

jednotlivým nalezeným řetězcům. Přitom se vytváří slovník (tabulka) již rozeznáných řetězců, který se v průběhu komprimace udržuje v paměti počítače.

Běh algoritmu LZW začíná s prázdným slovníkem a řetězcem W obsahujícím první znak zdrojového souboru. Vždy po přečtení dalšího znaku c zjistí, jestli se řetězec $W+c$ vyskytuje ve slovníku. Pokud ano, pouze se prodlouží řetězec W o znak c , jinak zapíše nový odkaz na řetězec do slovníku. Pokud řetězec W obsahuje jediný znak, bude do slovníku zanesen pouze jediný znak. Vzhledem k tomu, že do slovníku se zapisují čísla větší než 255, je nutné i tento jediný znak zapsat v příslušné podobě. K vyjádření odkazu na řetězec se většinou používá 12bitová hodnota. V tomto případě tedy čísla 0 – 255 jsou jednotlivé znaky a čísla 256 – 4095 kódy řetězců do slovníku.

Nástin popsaného algoritmu v předcházejícím odstavci: uvádějí následující řádky:

```
1 set W = NIL
2 while(!eof(input))
3 {
4   přečti znak c
5   if (Wc ve slovníku) W = Wc
6   else
7     {
8       Zapiš kódovou hodnotu pro W
9       Přidej Wc do slovníku
10      W = c
11    }
12 }
```

Pro větší názornost uvedeného algoritmu si uvedeme konkrétní příklad zakódování vstupního řetězce. Mějme vstupní řetězec následujících znaků: WEB/WEB/WEB!

Tab. 1. Zakódování vstupního řetězce WEB/WEB/WEB!

Řetězec W	Přečtený znak	Výstup	Nová položka ve slovníku
		W	
W	E	W	(256) = WE
E	B	E	(257) = EB
B	/	B	(258) = B/
/	W	/	(259) = /W
W	E		
WE	B	-256	(260) = WEB
B	/		
B/	W	-258	(261) = B/W
W	E		
WE	B		
WEB	!	-260	(262) = WEB!
!	(EOF)	!	

Jednou z důležitých předností, kterou je třeba vyzdvihnout, je využitelnost algoritmů LZW u takových zařízení, kterými data pouze procházejí a kde je žádoucí data zpracovat rychle. Takovými zařízeními jsou například modemy. Zde je totiž třeba rychlé zpracování dat a vyslání po lince k příjemci. Výhoda komprimace LZW oproti jiným spočívá v tom, že modemy používající tuto kompresi nemusí zdrojová ani zkomprimovaná data uchovávat. Zdrojová data se okamžitě vyšlou příjemci. Nezanedbatelnou výhodou je přitom fakt, že většina jiných komprimačních algoritmů pracuje po blocích, tedy že zkomprimovaná data

mohou být vyslána až po kompresi celého bloku, kdežto za použití algoritmu LZW může být proces vysílání plynulý.

Mezi další výhody algoritmu LZW patří: velmi dobrý kompresní poměr, rychlá komprese i dekomprese a nevelké nároky na paměť. Je to adaptivní metoda vytvářející dynamický substituční slovník. Tento slovník není nutné ukládat pro potřeby dekomprese, protože dekompresní část algoritmu vytvoří duplicitní slovník přímo pro dekompresi.

Mezi nevýhody patří rychlý nárůst slovníkových kódů odkazujících na řetězce původního souboru. Ten vede v některých případech k zaplnění paměti určené pro slovníkové kódy. Z toho pak vyplývá smazání aktuálního slovníku a jeho nové vytváření bez možnosti použití „zapomenutých“ odkazů. V případě větších slovníků zase narůstá doba vyhledání řetězce v tabulce.

2.5 Huffmanovo kódování

Huffmanovo kódování je nejznámějším zástupcem skupiny algoritmů, které pracují na základě různých četností znaků v kódovaných datech. Již z této úvodní věty je zřejmé, že Huffmanovo kódování a příbuzné komprimační techniky jsou určeny především pro komprimaci textových souborů. Při komprimaci se postupuje tak, že nejprve komprimační algoritmus zjistí pravděpodobnosti výskytů jednotlivých znaků (případně jejich kombinací) a každému znaku (kombinaci znaků) přiřadí jedinečný kód. Takovéto kódy se liší svou bitovou délkou. Tato část algoritmu je nejdůležitější a musí být navržena tak, aby přiřazení kódů znakům s častějším výskytem a bitově delším znakům s méně častým výskytem. Pak již jen algoritmus postupně načítá znaky vstupního souboru, nachází odpovídající předem přiřazené kódy a tyto kódy zapisuje na výstup.

Předpokládejme, že potřebujeme zkomprimovat soubor, který obsahuje pouze znaky *A*, *8*, *0*, *K* a *R*. Dále předpokládejme, že proběhla analýza pravděpodobnosti výskytu těchto znaků v souboru určeném pro komprimaci s následujícím výsledkem:

znak *A* se v souboru vyskytuje s pravděpodobností 50 %

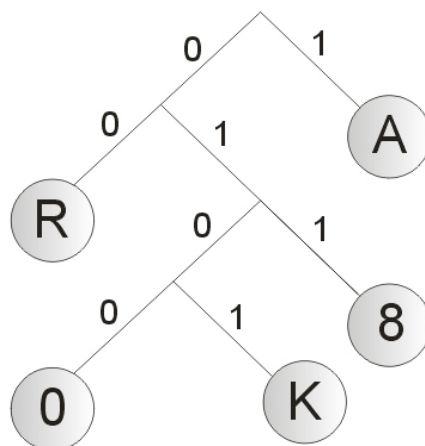
znak *8* se v souboru vyskytuje s pravděpodobností 12,5 %

znak *0* se v souboru vyskytuje s pravděpodobností 6,25 %

znak *K* se v souboru vyskytuje s pravděpodobností 6,25 %

znak *R* se v souboru vyskytuje s pravděpodobností 12 %

Binární strom podle zásad Huffmanova algoritmu potom tvoříme od koncových „listů“ stromu, tj. od znaků s nejmenší pravděpodobností (viz obrázek 2). Vybereme nejprve znaky s nejmenší pravděpodobností výskytu a uděláme z nich dva koncové listy stromu. Potom vybereme znak s nejmenší pravděpodobností z dosud nezpracovaných znaků, a tak postupujeme až ke kořeni stromu. Každý uzel binárního stromu představuje součet pravděpodobností všech „listů“, které jsou ve stromové struktuře pod ním.



Obr. 2. Příklad binárního stromu
Huffmanova kódování

Uvedený binární strom není samozřejmě jedinou možností přiřazení Huffmanových kódů jednotlivým znakům při uvedené distribuci. Binárních stromů lze uvedeným způsobem vytvořit více, avšak celková komprimovaná délka souboru bude vždy stejná, jelikož počet bitů přiřazených jednotlivým znakům bude ve všech případech stejný.

Vytvořením binárního stromu jsou definovány kódy přiřazené podle četnosti výskytu jednotlivých znaků souboru. Tyto znaky se přiřadí jednotlivým znakům tak, že ke každému znaku se postupuje stromovou strukturou. Při pohybu od uzlu k uzlu zaznamenáváme jedničku vždy při cestě vpravo dolů a nulu při cestě vlevo dolů.

Pokud se budeme držet binárního stromu vytvořeného na obrázku 2, budou kódy přiřazené jednotlivým znakům vypadat takto:

A: 1

8: 011

0: 0100

K: 0101

R: 00

Komprese na základě vytvořeného binárního stromu probíhá tak, že komprimační algoritmus čte sekvenčně znaky ze vstupního souboru a do výstupního pak zapisuje jim příslušné sekvence bitů.

Dekomprese sice vyžaduje znalost původního binárního stromu, ale je také velmi jednoduchá. Pokud jsou kódy ve zkomprimované podobě souboru uloženy bezprostředně za sebou, bude již dekomprimační algoritmus schopen rozpoznat znak příslušející ke kódu. Algoritmus bude prostě číst kód (o kterém ještě není známo, jak je dlouhý) bit po bitu a podle hodnoty bitu bude postupovat po binárním stromě od kořene až k „listu“ stromu představujícímu znak. Tento znak je již původním znakem, který byl při komprimaci zakódován.

Slabinou tohoto algoritmu je mimo jiné i to, že binární strom lze bez problémů sestavit pouze v případě, že pravděpodobnosti výskytu všech znaků vstupního souboru jsou mocninou čísla 0,5. Pouze v tom případě každé vyšší patro vytvářeného stromu obsahuje uzel nebo „list“ stromu s pravděpodobností výskytu dvojnásobnou oproti „listům“ či uzlům ležící o patro níž. Je zřejmé, že ideálního kompresního poměru může být dosaženo pouze při dodržení této zásady. V praxi však s něčím takovým nelze počítat vlastně nikdy, a proto je zapotřebí zaokrouhlovat pravděpodobnosti výskytu jednotlivých znaků.

Zaokrouhlování ovšem musí být prováděno při dodržení dvou základních zásad:

- Součet všech procentuálních hodnot musí samozřejmě dávat hodnotu 100 %
- Zároveň je nutné dbát na to, že v každém patře vytvářeného stromu může být maximálně určitý počet „listů“ a uzlů. Tento počet je dále omezen již vytvořenými patry stromu.

Složitosti vznikající při dodržování poslední zásady lze demonstrovat na následujícím příkladu. Předpokládejme, že máme opět stejný soubor znaků jako v předcházejícím případě, tentokrát ovšem s jinou pravděpodobnostní distribucí:

znak *A* se v souboru vyskytuje s pravděpodobností 46 %

znak *8* se v souboru vyskytuje s pravděpodobností 22 %

znak *0* se v souboru vyskytuje s pravděpodobností 5 %

znak *K* se v souboru vyskytuje s pravděpodobností 5 %

znak *R* se v souboru vyskytuje s pravděpodobností 22 %

Při vytváření binárního stromu podle zásad Huffmanova algoritmu nalezneme nejprve dva znaky s nejnižší pravděpodobností (*0* a *K*). Jejich pravděpodobnosti zaokrouhlíme na 6,25%, což je mocnina 0,5. Následují znaky *8* a *R* se stejnou pravděpodobností. Zaokrouhlení obou příslušných pravděpodobností na 25 % je nepřípustné, protože na jediný zbývajících znak by zbyla pravděpodobnost 37,5 % - ta je ovšem v binárním stromě nepřípustná. Je tedy nutné pravděpodobnost jednoho z těchto znaků zaokrouhlit na 12,5 % a pravděpodobnost druhého zaokrouhlit na 25 %. To znamená, že ačkoli oba znaky se ve vstupním souboru vyskytují se stejnou pravděpodobností, jeden z nich bude v komprimované formě reprezentován třemi bity a druhý dvěma bity.

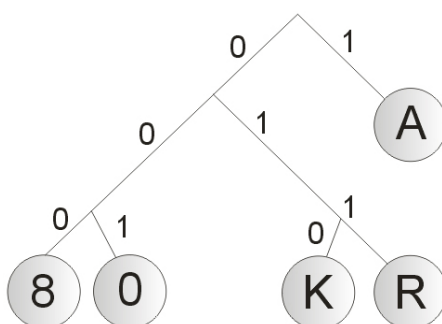
Tento jednoduchý příklad nedemonstruje všechny problémy spojené s tvorbou binárního stromu, ale nastiňuje příčinu, proč se do kompresního schématu zavádějí nežádoucí nepřesnosti, které zhoršují výsledný kompresní poměr. Zároveň se algoritmus tvorby binárního stromu stává složitější.

2.6 Shannon – Fanovo kódování

Shanon-Fanovo kódování je velmi podobné Huffmanovu. Rozdíl mezi oběma algoritmy spočívá v konstrukci binárního stromu. Tvorba binárního stromu v Shanon-Fanově modifikaci je poněkud jednodušší. Lze ji shrnout do dvou následujících kroků:

- Rozděl soubor symbolů na dvě skupiny se stejnou nebo co nejpodobnější celkovou pravděpodobností znaků obsažených v obou skupinách. Pokud se vrátíme k druhému příkladu z předcházejícího odstavce, obsahovala by po takovém rozdělení znaků jedna skupina pouze znak *A* (46 %) a druhá skupina znaky *8*, *0*, *K* a *R* (celkem 54 %). První skupině se přiřadí binární symbol nuly a druhé binární symbol jedničky.
- Opakuj první krok na všechny dosud vytvořené skupiny, dokud každá skupina nebude obsahovat jediný znak.

Rozdíl mezi způsoby vytváření binárních stromů v Huffmanově a Shanon-Fanově variantě je v tom, že Huffmanovo kódování vytváří strom od koncových listů směrem ke kořenu, zatímco Shanon-Fanova metoda postupuje obráceně – od kořene k listům. Zatímco Huffmanova varianta při správném použití generuje vždy optimální kódy pro jednotlivé znaky, jednodušší Shannon-Fanova metoda může v některých případech použít v některých kódech pár bitů navíc. Všimněte si, jakým způsobem vytvoří Shanon-Fanova varianta binární strom pro druhý příklad z předcházejícího odstavce (viz obrázek 3). Kód příslušející ke znaku *A* bude reprezentován jediným bitem, zatímco všem ostatním znakům bude přiřazen kód tříbitový. Zcela se tak smaže rozdíl mezi pravděpodobnostmi výskytu znaku 5 % a 22 %.



Obr. 3. Příklad binárního stromu vytvořeného Shannon-Fanovou metodou

Pokud se budeme držet binárního stromu vytvořeného na obrázku 3, budou kódy přiřazené jednotlivým znakům vypadat takto:

A: 1

8: 000

0: 001

K: 010

R: 011

2.7 Aritmetické kódování

Jak bylo uvedeno výše, Huffmanovo kódování dosahovalo optimálních výsledků pouze v případě, kdy pravděpodobnosti jednotlivých znaků byly celočíselnými mocninami 0,5. V ostatních případech by bylo optimální na zakódování jednoho znaku použít neceločíselný počet bitů, což však samozřejmě binární soustava neumožňuje.

Aritmetické kódování tento nedostatek odstraňuje velmi zajímavým způsobem – kóduje celou zprávu jako jediné kódové slovo.

Základní myšlenku použitou v této komprimační metodě lze popsat ve stručnosti takto: aritmetické kódování reprezentuje celou zprávu jako číslo z intervalu $<0,1$). Na začátku kódování uvažujeme celý tento interval. Jak se zpráva prodlužuje, zužujeme postupně interval – přibližují se k sobě horní a dolní mez nově vytvářeného intervalu. Na konec stačí zapsat libovolné číslo z výsledného intervalu – to samo o sobě reprezentuje celou zprávu.[5]

Algoritmus komprese lze nastínit jako následující sekvenci kroků:

- Zjištění pravděpodobnosti výskytu jednotlivých znaků ve zdrojovém souboru
- Rozdělení intervalu $<0,1$) na podintervaly, jejichž vzájemný poměr velikostí odpovídá poměru pravděpodobností jednotlivých znaků (seřazených dle abecedy).
- Uložení tohoto základního rozdělení intervalu $<0,1$).

- Vlastní komprese víceznakové zprávy. Komprese víceznakové zprávy bude probíhat tak, že se nejprve vybere první znak vstupního souboru. Ten zúží interval $<0,1)$ na podinterval příslušející tomuto znaku tak, jak mu byl přidělen v druhém bodě celkového algoritmu aritmetického kódování. Tento podinterval bude rozdělen stejným způsobem jako dříve celý interval $<0,1)$. Po načtení dalšího znaku bude podinterval dále zúžen podle načteného znaku. Tak to půjde dále až do načtení posledního znaku zprávy.
- Posledním bodem je vybrání kteréhokoliv zlomku náležejícího do výsledného nejmenějšího podintervalu a jeho převedení do binární formy

Nejjednodušeji lze tento algoritmus pochopit na příkladu.

Naším úkolem bude za pomoci aritmetického kódování zakódovat tříbajtovou zprávu obsahující znaky „ XXY “.

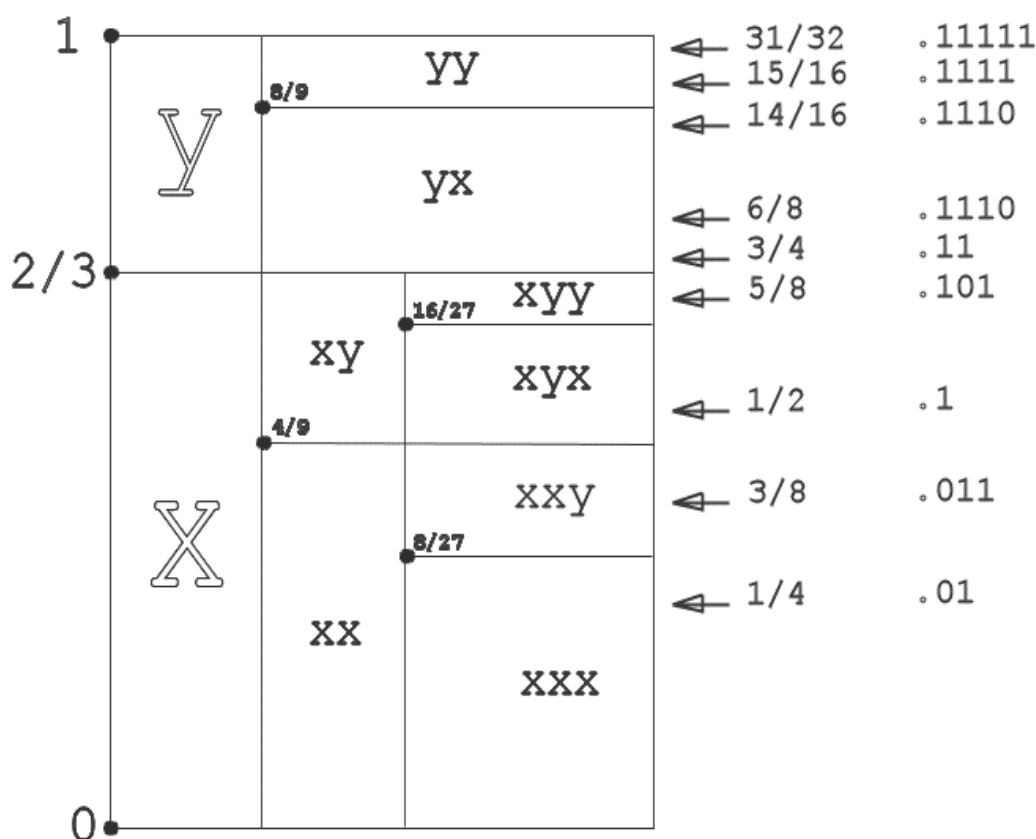
V souladu s prvním bodem algoritmu aritmetického kódování zjistíme, že zpráva obsahuje pouze dva znaky:

Pravděpodobnost výskytu znaku X je $2/3$,

pravděpodobnost výskytu znaku Y je $1/3$.

V dalším kroku rozdělíme v tomto poměru (tedy 2:1) interval $<0,1)$ a rozdělení si zapamatujeme. Nyní již přistoupíme ke kódování vstupního souboru.

Načteme první znak souboru, kterým je písmeno X a zjemníme původní interval na podinterval $<0, 2/3)$. Tento interval opět rozdělíme v poměru 2:1. Dělicím bodem bude v tomto případě bod $4/9$. Načteme další znak – je jím opět písmeno X , které nám podinterval opět zúží na $<0,4/9)$. Tento interval opět rozdělíme v poměru 2:1 – dělicím bodem je v tomto případě zlomek $8/27$. Následujícím znakem zprávy je písmeno Y , jež opět zúží interval, o který se budeme zajímat, na $<8/27, 4/9)$. Vzhledem k tomu, že třetí znak je v našem vstupním souboru také znakem konečným, je konečný i tento interval. Zbývá již jen vybrat libovolnou hodnotu z tohoto intervalu, která převedena do binární formy bude reprezentovat celou vstupní sekvenci. Vyberme číslo $3/8$.



Obr. 4. Tabulka Aritmetického kódování

Pro potřeby dekomprimace zakódované zprávy je nezbytné předat dekomprimační části algoritmu (kromě kódové hodnoty) také poměry pravděpodobností jednotlivých znaků a počet znaků původní zprávy.

Tím jste se seznámili se základní myšlenkou komprimace obecně známé pod jménem *arithmetic coding* (aritmetické kódování). Při převádění této chytré a jednoduché myšlenky do počítačové praxe se však setkáme s celou řadou problémů – nejzávažnější je asi potřeba konverze reálných čísel na bitové vyjádření. Výsledný kód reprezentující vstupní posloupnost je zapotřebí převést do formy vyjádřitelné ve dvojkové reprezentaci. Navíc s každým reálným číslem rozdělujícím základní interval $<0,1)$ je nezbytné pracovat s vyloučením jakéhokoli zaokrouhlování.

Velkým problémem je u popsaného komprimačního algoritmu rychlý růst počtu platných desetinných míst v racionálních číslech, která představují dolní a horní meze intervalů. Ve většině případů již u souborů o deseti znacích roste počet platných cifer těchto dělících bodů na dvacet až třicet číslic. Z toho je zřejmé, že celé soubory o velikosti mnoha

megabajtů nelze kódovat jediným číslem, neboť počet platných míst u čísel v mezi výpočtech by rostl prakticky neomezeně.

Dalším problémem tohoto druhu komprimace je složitost způsobu zjišťování, jak dlouhé bude výsledné bitové vyjádření kódu reprezentujícího vstupní posloupnost znaků

2.8 Burrows-Wheelerova transformace

Burrows-Wheelerova transformace (BWT) je metoda, jak přeměnit hůře komprimovatelná data na lépe komprimovatelná. Odtud také název "transformace". Jejími tvůrci jsou Michael Burrows a David Wheeler, kteří v roce 1994 publikovali zprávu o své práci u Digital Systems Research, ve které podrobně rozebrali celý postup komprese pomocí BWT a několika doplňujících algoritmů. Vlastní BWT objevil Wheeler již v roce 1983, ale nikde ji nepublikoval. V současné době se používá např. v bzipu2, který se postupně stává hlavním komprimačním nástrojem pro Linux (a nejen pro něj).

Transformace

Celá BWT spočívá ve vratném setřídění bloku dat. Výstup obsahuje ta samá data jako byla ta původní, ale jejich struktura je mnohem vhodnější pro kompresi klasickými algoritmy (např. Huffmanovým kódováním).

Jako vstupní blok dat zvolme například řetězec "ema ma maso" (v reálné implementaci se transformuje najednou mnohem větší blok, ale pro naše účely stačí těchto 11 písmen). Předpokládejme, že tento řetězec je umístěn v kruhovém bufferu a umístíme postupně stejně dlouhé řetězce začínající na pozici 0 až 10 do bufferu takto :

- 1 ema ma maso
- 2 ma ma masoe
- 3 a ma masoem
- 4 ma masoema
- 5 ma masoema
- 6 a masoema m
- 7 masoema ma
- 8 masoema ma
- 9 asoema ma m
- 10 soema ma ma
- 11 oema ma mas

Ve skutečné implementaci se samozřejmě tento buffer nevytváří a reprezentuje se například pomocí ukazatelů do vstupního bloku dat.

Nyní tuto tabulku lexikograficky (tj. dle abecedy) seřídíme po řádcích.

	F	L	
1	0	ma masoema	
2	1	masoema ma	
3	2	a ma masoem	
4	3	a masoema m	
5	4	asoema ma m	
6	5	ema ma maso	<--
7	6	ma ma masoe	
8	7	ma masoema	
9	8	masoema ma	
10	9	oema ma mas	
11	10	soema ma ma	

Šipkou je označen původní blok dat. Jak vidno, v tabulce jsou navíc ještě dvě značky. F označuje první sloupec, kde jsou seříděny všechny znaky vstupních dat, v našem případě "aaaemmmos". L označuje poslední sloupec, který není seříděn, zato každý znak v něm obsažený je prefixovým znakem pro řetězec začínající ve stejném řádku ve sloupci F (což bude později důležité). Výstupem transformace je sloupec L a tzv. primární index, což je číslo řádku s původními daty. V našem případě ("aammoe sa", 5). Povšimněte si seskupení stejných písmen k sobě (důvod, proč se výstup tak krásně komprimuje).[8]

Zpětná transformace

Ač se to na první pohled nezdá, tak BWT je skutečně vratná. Nejprve se seřídí sloupec L, čímž vznikne sloupec F (neboli první sloupec seříděného bufferu). Nyní nastává ta horší část.

Máme sloupec F a sloupec L, který je prefixovým sloupcem sloupce F. Tato informace nám stačí, abychom spočetli transformační vektor T. T je jakýmsi řetězcem určujícím návaznost jednotlivých písmen na sebe (tedy před sebe, protože se postupuje odzadu). Podívejme se tedy na způsob, jakým se tvoří.

Podíváme-li se znovu na setříděný buffer,

	F	L
1	ma	masoema
2	masoema	ma
3	a	ma masoem
4	a	masoema m
5	asoema	ma m
6	ema	ma maso
7	ma	ma masoe
8	ma	masoema
9	masoema	ma
10	oema	ma mas
11	soema	ma ma

a vezmeme-li v úvahu, že známe pouze sloupce F a L, jak určíme vektor T? Jednak víme, že prvky z L předchází prvkům z F, tzn. že máme dvojice následník (z F) - předchůdce (z L). Ale, jak je spojit do jednoho řetězce? Samozřejmě, mohou se propojit přes stejné prvky (z m-a a a-s vznikne m-a-s). Leč má to háček. Přes jaké prvky, neboť je jich tam více stejných, že? Stačí si všimnout, že jak řetězce začínající v F na nějaký znak (např. "a"), tak ty začínající v L na ten samý znak, jsou řazeny stejně, od 2. znaku (řetězce v F jsou řazeny i dle 1., ale ten je stejný), tedy, jejich pořadí je shodné, pouze mohou být poněkud vzdáleny od sebe. Tím pádem, hledáme-li předchůdce pro druhé "a" z L, najdeme ho v 2. řádku začínajícím na "a" (s "a" ve sloupci F).

V našem případě je výsledkem transformační vektor T :

	0	1	2	3	4	5	6	7	8	9	10
L	a	a	m	m	m	o	e		s	a	
F		a	a	a	e	m	m	m	o	s	
T	2	3	6	7	8	9	5	0	1	10	4

Vytvořili jsme řetězec "ameosam am " (začali jsem ve sloupci L na pozici 0) převedeno do správného pořadí " ma masoema". Nyní vezmeme primární index, který vygenerovala původní transformace, tedy číslo 5. Ten určoval pozici originálního textu v setříděném bufferu, což znamená, že kam se transformuje prvek z pozice 5 ve sloupci L, tam končí vstupní text. V tomto případě je vstupní text "ema ma maso" (ve skutečnosti se primární index bere do úvahy již při vytváření řetězce, kdy se začíná od něj).

3 ZTRÁTOVÁ KOMPRESSE

Ztrátové komprimační techniky provádějí snížení datového objemu odstraněním irelevantní části informace ze signálu. Protože je obvykle nastavitelné, která část se odstraňuje a v jak velké míře, jejich komprimační poměr je proměnlivý, a proto může být i relativně vysoký. Tento nastavitelný a vysoký kompresní faktor ztrátových komprimačních metod je velice vítán při multimediálních aplikacích s cílem v co největší míře snížit nároky na bitovou rychlost, resp. šířku pásma.

Spousta typů dat zabírá hodně prostoru na disku – jde o video, obrázky a hudbu. Přitom se jedná o data, u nichž určitou ztrátu kvality nepostřehneme nebo u nichž je taková, že jsme ji ochotni akceptovat. První ztrátová komprese se objevila pravděpodobně u obrázků. Vezmeme-li v úvahu, že obrázek o rozlišení 1024×768 bodů s barevnou hloubkou 24 bitů zabírá na disku 2,25 MB místa v nezkomprimované podobě, je nasnadě, že bylo třeba něco vymyslet. Jako základ se využívá bezztrátové komprese a hledají se oblasti, které mají stejné barvy, například bílé pozadí obrázku. Informace o nich zabírají mnohem méně místa než plný datový výpis. Takto lze tedy obrázky výrazně zmenšit, aniž se kvalita sníží. Toho se hojně využívá v profesionální práci s obrazovými daty (fotografové, grafici). Pro běžné domácí použití jsou však takové soubory stále příliš velké. Ke slovu se tedy dostává ztrátová komprese. V případě zmíněných barevných oblastí je to tak, že jsou tolerovány určité odchylky barev, které oko běžného uživatele nepostřehne. Barvy podobného odstínu jsou považovány za tutéž barvu, a soubor tedy může být i několikanásobně menší. Podobně to funguje s ostroť. Míru oné tolerance lze nastavit při ukládání obrázku.

3.1 JPEG

Zkratka JPEG znamená Joint Photographic Experts Group, což je komise pro standardy pod ISO (International Standard Organization). JPEG představuje spolehlivou a rychlou kompresi pro obrázky se spojitými barevnými tóny v barevném rozlišení 24 bitů.

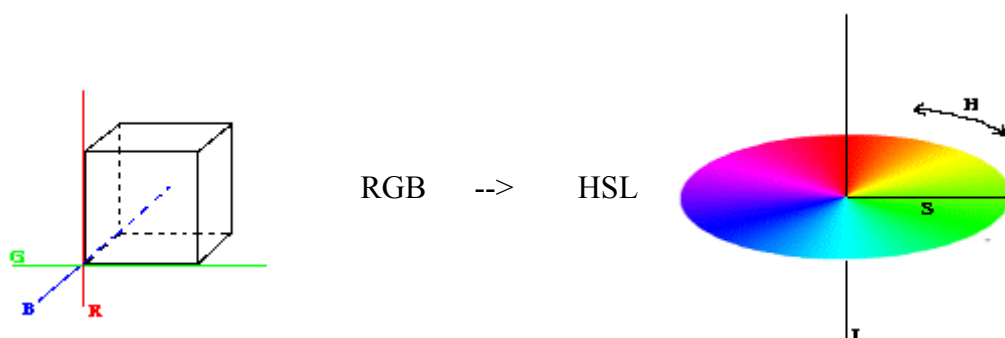
Uživatel může kvalitu komprese JPEG ovlivnit parametrem, který se nazývá *nastavení kvality* nebo též *Q faktor*. V různých aplikacích se setkáte s různým odstupňováním Q faktoru. Většinou je však stupnice v rozmezí 1-100. Hodnota 1 odpovídá nejmenší velikosti, ale také nejmenší kvalitě obrazu. Čím vyšší je Q faktor, tím je obrázek kvalitnější. Zároveň však platí, že čím je obrázek kvalitnější, tím také narůstá jeho

velikost. Obecně lze tedy říci, že účelem je najít nejmenší hodnotu Q faktoru s ještě vizuálně akceptovatelným výsledkem. Samozřejmě jiná bude kvalita webových fotografií a např. fotografií určených k tisku.

JPEG byl navržen tak, aby se ztrácely pouze informace, které jsou lidským okem přehlédnutelné. Malé změny barev není snadné lidským okem rozeznat tak, jako třeba změny v jejich intenzitě. Proto se také ztrátová komprese metody JPEG chová mnohem šetrněji k šedé složce obrazu, než k barvám. JPEG se hodí především na fotografie, a proto se také jeho hardwarová implementace velice často používá u digitálních fotoaparátů.

Protože JPEG standard je chráněn ISO a ITU Copyright–em není možné uveřejnit přesný popis. Ale pro pochopení se mi zdá zcela dostačující schéma, které kdysi popsal Tom Lane.

Smyslem této procedury je fakt, že lidské oko je schopno rozlišit velice přesně změny na ose L (lightness na obrázku vpravo) - v odstínu barvy, ale velice špatně vnímá změny na osách S a H (saturation a hue), které definují barvu jako takovou. Důsledkem takovéto transformace je skutečnost, že pak si můžeme dovolit velice silnou kompresi a ztráty dat na osách S a H aniž by si toho pozorovatel obrázku všiml. Také již při této transformaci barevných systémů vzniká jistá ztráta informací v důsledku zaokrouhlování.



Obr. 5. Transformace obrázku do vhodného systému barev

Běžná fotografická předloha může být zkomprimována v poměru až 25:1 bez výraznější změny kvality. Zcela nevhodný se JPEG naopak jeví pro vektorovou grafiku nebo pro obrázky, na kterých jsou velké plochy se stejnými barvami. Plocha pak při bližším pohledu

obsahuje barevné fleky ve tvaru čtverců. Při vyšší kompresi se na těchto místech potom objevují nepříjemné zuby. Je vždy třeba pečlivě zvážit, na jaký obrázek je vhodné JPEG použít a na jaký ne.

Standard JPEG byl po několikaletém vývoji definitivně schválen roku 1988. Čtrnáct let vydržel a přežíval. Je to opravdu mnoho. Nicméně všechno jednou končí, a proto skupina JPEG vyvíjí od roku 1995 nový standard *JPEG 2000*. Nový algoritmus obohacuje současné metody. Je založen na vlnkové transformaci. Vývojáři uvádí zlepšení kompresního poměru o 20 až 30% oproti nynějšímu standardu JPEG. JPEG také neumožňuje zpracovávat obrázky větší jak 64 000 x 64 000 obrazových bodů (pixelů). JPEG 2000 toto omezení ruší a navíc je zde možnost využití i ostatních barevných módů jako například CMYK (současné JPEG oficiálně podporuje pouze RGB). Novinek, které JPEG 2000 s sebou přináší je mnoho. O jeho úspěchu či neúspěchu však rozhodnou především výrobci softwaru a hardwaru. Ti si jistě možnosti, které jim JPEG 2000 nabízí, velice dobře uvědomují. Současně se však např. u digitálních fotoaparátů musí počítat se zvýšenými nároky na hardware.[7]

3.1.1 Progresivní JPEG

Jak jsem již na začátku naznačil, JPEG není ani klasickým grafickým formátem, ani přesně daným kompresním algoritmem pro komprimaci grafických dat. Nyní si rozšířme chápání termínu JPEG o další pojem – progresivní JPEG. Základní verze kompresního schématu JPEG funguje tak, že celý obrázek je dekomprimován z formátu JPEG během jediného skenu od horního k dolnímu okraji obrázku.

Progresivní JPEG je určen speciálně pro publikování na WWW stránkách. Při použití progresivního JPEGu se celý obrázek zobrazuje v několika po sobě následujících skenovacích přebězích, během nichž se zlepšuje kvalita obrázku. Po prvním skenu se zobrazí pouze hrubé kontury obrázku atd. Kvalita obrázku se tak progresivně zlepšuje. Výhodou tohoto přístupu je fakt, že k prvnímu náhledu na celý obrázek postačí zpracování malého zlomku celkových dat kódujících původní grafickou předlohu.

Progresivní JPEG je ohromnou výhodou při používání pomalého dial-up (po telefonní lince) připojení k serveru WWW. Již po přenesení malého množství dat uživatel vidí základní kontury celého obrázku, aby zjistil, co grafická data zobrazují.

Určitou nevýhodou tohoto druhu kódování je fakt, že při každém jednotlivém skenovacím přeběhu musí počítač provést prakticky stejné množství matematických operací jako při jediném přeběhu normálního JPEGu.

3.1.2 Transparentní JPEG

Pokud chcete vytvořit transparentní (s prosvítajícím pozadím) obrázek, vězte, že nepochodíte. Implantace této vlastnosti do komprimačního schématu JPEG je natolik složitá, že se s ní zatím nepočítá ani do budoucna.

Důvodem je systém, jakým se transparentnosti obvykle dosahuje. Ostatní grafické formáty vyčleňují jednu určitou barvu pro identifikaci pixelů, které mají být transparentní. Tato barva pak není v obrázcích použita. Vzhledem ke ztrátovosti formátu JPEG podobný přístup není možný. Ztrátové komprimační schéma totiž při komprimaci často průměruje barvu blízko ležících bodů, a tak by vyčlenění jediné barvy pro identifikaci transparentních pixelů vedlo k velice špatným výsledkům co do kvality obrázku. Tento problém lze částečně řešit použitím určitého intervalu barev jako identifikátoru transparentních pixelů. „Průhlednost“ pozadí by potom byla procentuálně odstupňovaná.

I v tomto případě by se vyskytla spousta problémů. Kvalita zobrazení prosvítajícího pozadí by se sice zlepšila, ale stále by byly patrné drobné chyby v zobrazení. Použití takového přístupu ke grafickým datům by také vedlo k podstatnému zvětšení nároků na diskovou paměť. Transparentní obrázky JPEG by byly často dvakrát i vícekrát větší než stejné předlohy komprimované do standardní netransparentní podoby. Navíc je třeba si uvědomit, že obrázky, u kterých se vyplatí vytvářet transparentní formu, jsou přesně ty, na které ztrátové komprimační schéma funguje nejhůře – velké jednobarevné plochy se skokovými změnami barev.

Z tohoto pohledu se jeví jediným možným řešením použití nějakého neztrátového algoritmu pro uchování masky transparentního obrázku. Vývoj, standardizace a popularizace takového formátu by však stála mnoho úsilí, které si podle expertů sdružených do Joint Photographic Experts Group transparentní obrázky nezaslouží.[6]

3.1.3 Postup komprese

transformace barev

RGB $\rightarrow$ YC_BC_R (1 bajt na složku - tři bajty na pixel, zatím žádná ztráta dat)

redukce barev

- týká se pouze barevných složek C_B,C_R; jasová složka Y zůstává v původním stavu (lidské oko je totiž citlivější na vnímání světlosti)
- dvě varianty: průměrování sousedních dvojic pixelů (původně 3+3=6 bajtů se redukuje na 4 bajty - 2 bajty pro nezměněné složky Y, 2 bajty pro společné složky C_B,C_R obou pixelů); průměrování čtyř sousedních pixelů do čtverce (původních 4×3 bajtů se redukuje na 6 bajtů - 4 na Y, 2 na C_B,C_R)
- dochází tak k nenávratné ztrátě dat (která je ovšem většinou vizuálně málo patrná)

dopředná diskrétní kosinová transformace

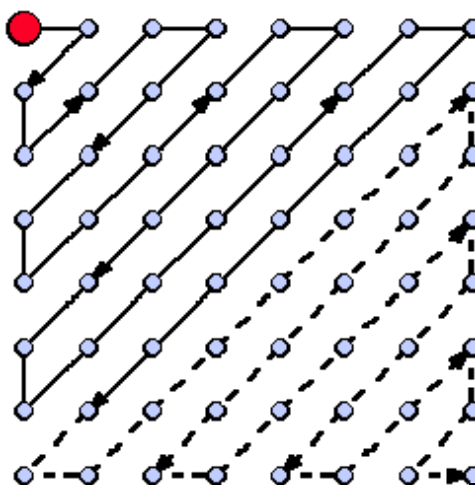
- aplikuje se postupně na čtverce po 8×8 pixelech
- provádí se zvlášť pro každou složku Y,C_B,C_R
- příklad transformace jasových složek Y (vlevo nahoře je tzv. **DC člen**, ostatní jsou **AC členy**)

kvantování koeficientů DCT

- kvantizační činitelé jsou stanoveny komisí JPEG (pro kvalitu 75% originálu) a uloženy v tabulkách pro každou složku Y,C_B,C_R
- v této fázi lze ovlivnit kvalitu výsledku komprese koeficientem kvality Q (nabývá hodnot 1 až 100), kterým se upraví kvantizační činitelé z dané tabulky - takto získaná tabulka kvantizačních činitelů se použije na všechny čtverce 8×8 pixelů pro všechny tři barevné složky a uloží se do výstupního souboru pro možnost zpětné dekomprese

kódování do výstupního souboru

- provádí se cik-cak Huffmanovým nebo aritmetickým kódováním



Obr. 6. Kódování do výstupního souboru

- DC členy se ukládají samostatně (lze tak rychle zobrazit náhled, který je $8 \times$ menší než původní obrázek)

Tab. 2. Postup při kompresi metodou JPEG

Komprese JPEG						
Zdrojový soubor s obrazem	Barevná transformace do optimálního barevného modelu	Redukce počtu barevných pixelů (např. $2 \times 2 \Rightarrow \text{pixel}$)	DCT - diskrétní kosinová transformace	Kvantizace - zde se uplatní Q faktor a dochází k jediným ztrátám	Neztrátová komprese pomocí Huffmanova kódování	Výsledný soubor JPEG
Dekomprese JPEG						
Vstupní soubor JPEG	Dekomprese Huffmanova kódování	Dekvantizace na základě uloženého Q-faktoru	Zpětná diskrétní kosinová transformace (zpětná DCT)	Navzorkování barevných pixelů	Zpětná barevná transformace do původního barevného modelu	Obrázek

3.2 Krátce o dalších formátech ztrátové komprese

3.2.1 Formát MPEG-1 audio

Pro přenos audia v normě MPEG-1 slouží tři „vrstvy“ (od angl. layer). Tyto vrstvy jsou jakýmsi označením verzí, kde Layer 1 je nejstarší a Layer 3 nejnovější. Jednotlivé vrstvy se od sebe liší hlavně kvalitativně. Všechny společně využívají frekvenční maskování (silnější zvuk je překryt slabším) a poslední dva navíc ještě Temporal Masking, který využívá nedostatečné rychlosti přizpůsobení sluchu při přechodu z hlasitého signálu na tichý.

Layer 3 (tedy mp3) je v současnosti nejrozšířenějším kodekem ztrátové komprese a je využíván i pro komerční účely jako je obchodování s hudbou na internetu.

3.2.2 Windows Media Audio (WMA)

Firma Microsoft pro své video formáty WMV a ASF (používaný zejména pro streamování videa a hudby po internetu) vyvinula zvukový kodek WMA. Tento formát patří mezi nejrozšířenější hlavně díky politice firmy Microsoft, která je jej implementuje do svých operačních systémů. Využívá toho, že průměrnému uživateli stačí jednoduché rozhraní a přítomnost přehrávače hned po instalaci, než aby instalovaly jiný, třeba i lepší přehrávač. Hlavní rozdíl mezi WMA a ostatními formáty je možnosti zachování autorských práv (v praxi to ovšem často znamená spíš omezení než výhodu).

Formát WMA dosahuje lepších výsledků než mp3 pouze při nízkých datových tocích. Dokazuje to hlavně několik testů mezi mp3 datového toku 128 kbps, jenž se kvalitou rovná WMA datovému toku 64 kbps. Navzdory tomuto faktu tento formát na ostatních „liniích“ prohrál.

4 PŘEHLED POUŽÍVANÝCH PROGRAMŮ PRO KOMPRESI DAT

Komprimačních programů je několik. Většina z nich užívá svůj specifický způsob kódování (tedy způsob jak data zmenšit). Mezi nejznámější komprimační programy patří WinZIP, WinRAR, Power Archiver, ACE atd. WinZIP nedosahuje nejlepších výsledků, ovšem je asi nejznámější a data pomocí něj zabalené je schopen rozbalit snad každý. WinRAR už má daleko lepší kompresní poměry a zároveň také solidní uživatelskou základnu. Dále uvedu ještě třeba JAR, ARJ, TAR, které jsou známější ještě méně, avšak dosahují rovněž dobrých poměrů a měl by je být schopen rozbalit snad každý solidní archivační program.

Kromě toho, že pak můžeme s komprimovanými soubory manipulovat jako s jedním objektem, poskytují archivační programy další výhody:

- data se mohou 'zmáčknot', aby nezabírala tolik místa
- mohou se přitom i zašifrovat, aby je nezneužila nepovolaná osoba
- výsledný archiv lze rozdělit na díly například na velikost diskety
- data lze vybavit kontrolními součty tak, aby po případném poškození diskety bylo možné chybné místo dopočítat beze ztráty informace
- archiv lze doplnit údaji o tom, kdo jej vytvořil (obdoba digitálního podpisu)

Protože se ke komprimaci v současné době používá nejvíce program WinZip, zaměřím se hlavně na něj. O ostatních programech se jen zmíním.

4.1 WinZip

Komprimační program Winzip je už po několik let nejoblíbenějším nástrojem pro práci s komprimovanými nebo archivními soubory ve formátu ZIP i v několika dalších populárních formátech. Winzip je nyní k dispozici v nové verzi s pořadovým číslem 9. I v této nové verzi se program drží staré tradice a zachovává si své osvědčené uživatelské rozhraní, které vypadá podobně, jako rozhraní Průzkumníka ve Windows. Vedle něj zůstávají k dispozici Průvodci pro pohodlné vykonávání nejčastějších úloh, kteří usnadní práci s programem především počítačovým začátečníkům. Winzip nyní také mnohem lépe

spolupracuje s populárními e-mailovými a antivirovými programy. Díky tomu se zjednodušuje přístup ke komprimovaným souborům, které vám vaši kolegové nebo přátelé pošlou elektronickou poštou a zrychluje se jejich antivirová kontrola. A v neposlední řadě je důležité i to, že Winzip i ve své poslední verzi zůstává věrný sharewarovému konceptu, což mj. znamená, že program si může kdokoli volně stáhnout a vyzkoušet.

4.1.1 Instalace

Instalace Winzipu je jednoduchá a přímočará: Klikněte na instalační soubor a dále pokračujte podle pokynů, které vám instalátor vypíše. program vás nejprve požádá, abyste odsouhlasili licenční ujednání a pak vám nabídne volbu složky, do níž se má nainstalovat.

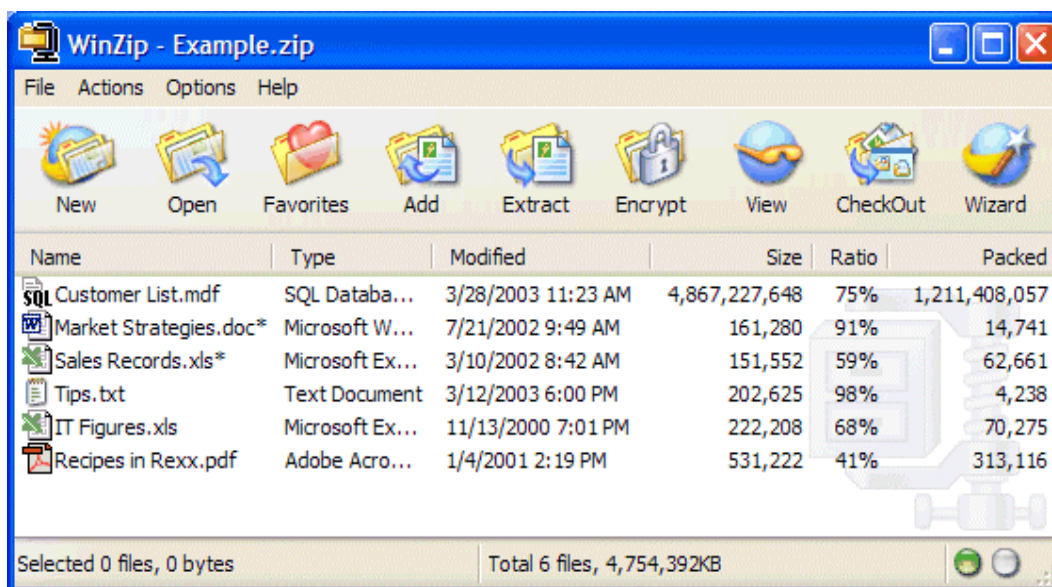
V dalším kroku se instalátor zeptá, zda má s programem Winzip asociovat přípony souborů pro nejpoužívanější komprimační formáty. Pokud na svém počítači nepoužíváte jinou utilitu pro komprimaci souborů, můžete odpovědět kladně pro všechny nabídnuté typy souborů.

Instalace končí dotazem, zda chcete používat Winzip s klasickým rozhraním nebo Winzip v podobě průvodce. Klasické rozhraní se hodí pro zkušené počítačové uživatele, průvodce je naproti tomu vhodnější pro začátečníky.

Po skončení instalace Winzip vyhledá na vašem disku všechny ZIP soubory a sestaví z nich seznam oblíbených nebo často používaných položek.

4.1.2 Vzhled programu

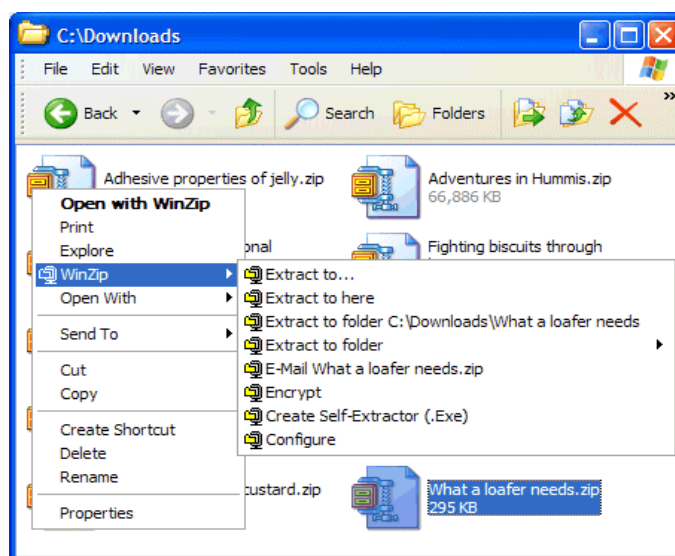
Po otevření Winzipu se na obrazovce objeví jeho hlavní okno. Pro základní ovládání programu je nejvhodnější použít velká tlačítka na ovládací liště. Chceme-li komprimovat soubory, založíme nejprve nový archiv stiskem tlačítka New a poté do něj přidáváme soubory pomocí tlačítka Add. Potřebujeme-li naopak vyextrahovat jednotlivé soubory ze zkomprimovaného ZIP archivu, musíme nejprve archiv otevřít pomocí tlačítka Open. Obsah archivu se vypíše v okně, v seznamu zvolíme požadované položky a poté klikneme na Extract, čímž je z archivu vykopírujeme.



Obr. 7. Vzhled programu WinZip

4.1.3 Integrace do Windows

Jednou z nejdůležitějších předností Winzipu je jeho dokonalá integrace s operačním systémem MS Windows. Winzip dobře zvládá práci se soubory pomocí jejich přetahování myší (drag&drop), přidává své funkce do kontextových nabídek Průzkumníka i některých jiných programů a pro provádění nejpoužívanějších funkcí nabízí Průvodce ve stylu Windows. Na následujícím obrázku je vidět, jak se změní kontextová nabídka Průzkumníka Windows. V nabídce se objeví nová položka WinZip, která umožňuje pohodlně sbalit vybrané soubory do ZIP archivu nebo naopak zvolený ZIP archiv rozbalit.



Obr. 8. Kontextová nabídka programu WinZip

4.1.4 Průvodce

Průvodce je ve Winzipu ideální volbou pro začínající uživatele. Provede vás krok za krokem typickými úkony, jakými je například sestavení komprimovaného archivu ze souborů ve zvoleném adresáři nebo rozbalení ZIP souboru získaného z internetu do zvolené složky. Jakmile získáte konkrétnější představu o tom, co program umí a jak se s ním pracuje, můžete při jeho používání přejít ke klasickému rozhraní a bezesbytku využít všech jeho možností.[4]

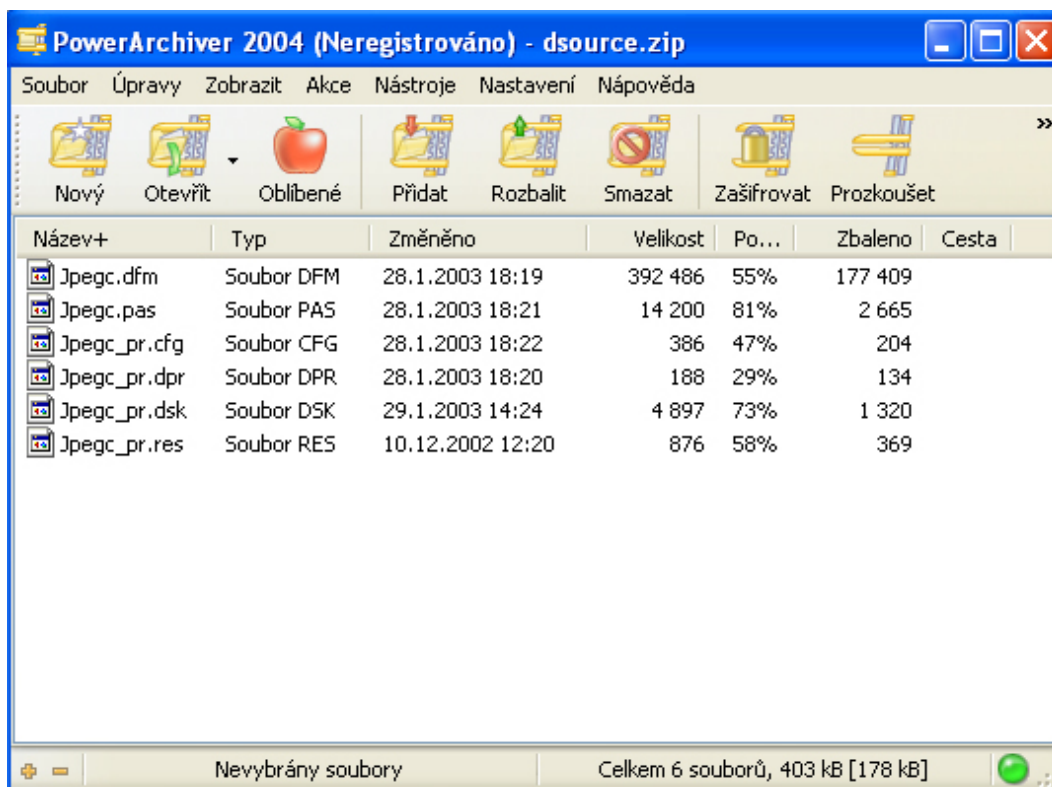
4.2 Power Archiver

Tento program je velmi dobrým archivačním programem, který je svým vzhledem velice podobný WinZipu. Svými vlastnostmi z něj vychází a dále je rozšiřuje. Na rozdíl od něj je freewarový (volně šířitelný). Také se od něj liší některými novými funkcemi a integrací ostatních druhů komprese, které jsou v něm obsaženy (například ACE nebo CAB). Kompresi je možné samozřejmě nastavit podle vašich přání, od nejmenší a nejrychlejší po největší a nejpomalejší. Při dnešní rychlosti procesorů je však čas, který strávíte u "pakovacího programu" minimální.

Jeho ovládání je přehledné. V horní části se nachází menu a ikony, pomocí kterých můžete využívat základní funkce programu. Na spodní liště vidíte, kolik souborů je v archivu a kolik z nich jste vybrali. Další operace se pak týkají pouze vybraných souborů. To, co tu popisují, není pevně stanovené, ale je možno to změnit v nabídce Options(Možnosti) pod položkou Configuration(Konfigurace). Pod touto položkou nastavujete i další věci, například integrované ostatní formáty archivů apod.

Můžete vytvářet nové a otevírat již existující archivy. V nich můžete nahrazovat novější soubory staršími. Také máte možnost vytvořit samorozbalovací archivy. Jakékoli archivy mohou být zablokovány zvoleným heslem nebo rozděleny na více libovolně velkých částí, například pokud potřebujeme něco přenést pomocí diskety. Můžete nahrazovat staré soubory novějšími, mazat soubory uvnitř archivu, nebo archiv v případě podezření na poškození zkontrolovat. Stejně jako WinZip nabízí výběr mezi klasickým rozhraním a takzvaným "wizardem", který má přidáné nové funkce a automaticky vám na disku vyhledá všechny archivy. Mezi nimi si potom můžete vybrat ten, s kterým budete pracovat.

Toto by mělo usnadnit práci, pokud však rutinně ovládáte počítač, bude vás to spíše zdržovat.



Obr. 9. Vzhled programu PowerArchiver

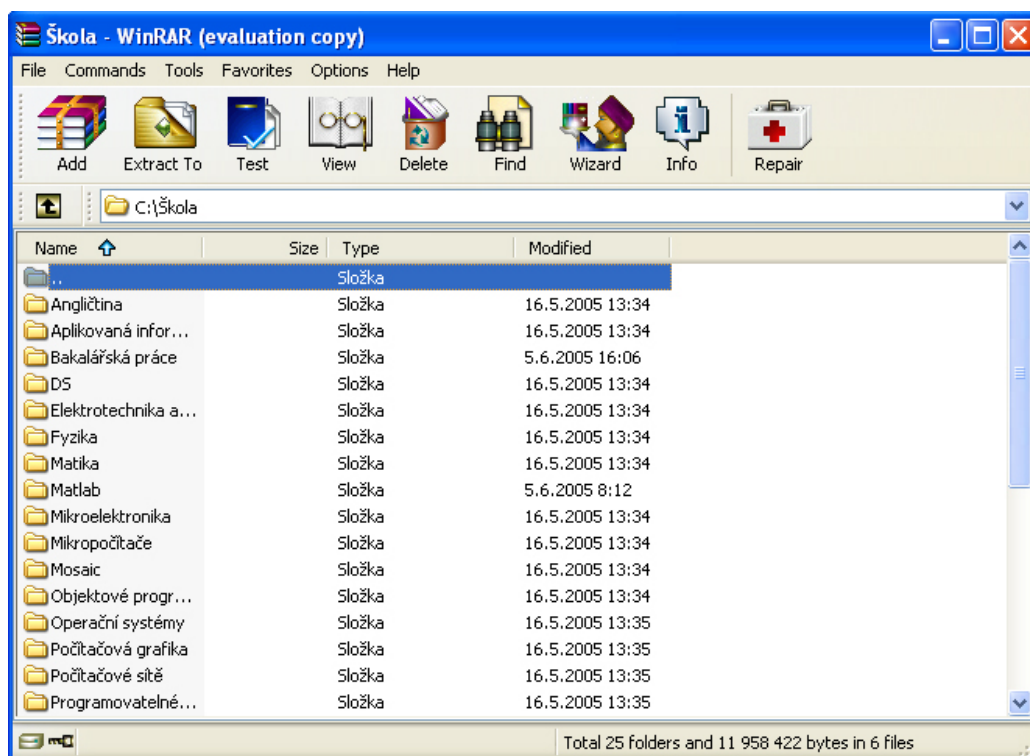
4.3 WinRAR

Jako oba výše uvedené komprimační programy i WinRAR poskytuje kompletní podporu archivů RAR a ZIP. Tento program používá vysoce výkonný kompresní algoritmus, který je o poznání lepší než u WinZipu. Používá také speciální algoritmy, optimalizované pro kompresi textových, grafických a audio dat. Při otevření nás nepřekvapí nic jiného než grafické uživatelské rozhraní. Po následném prozkoumání zde nechybí ani průvodce pro základní operace.

Dále WinRAR poskytuje

- Rozhraní z příkazového řádku
- Správa jiných než RAR archivů (CAB, ARJ, LZH, TAR, GZ, ACE, UAE, BZ2, JAR a ISO)

- Solid kompresi, která může zvýšit kompresní poměr o 10% - 50% oproti běžným kompresním metodám, zvláště při kompresi velkého množství malých, podobných souborů
- Dělené archivy
- Vytváření samorozbalovacích (SFX) archivů - i dělených - s použitím základních nebo volitelných SFX modulů
- Opravu fyzicky poškozených archivů
- Záchranné segmenty umožňující obnovit poškozenou či chybějící část děleného archivu
- Podporu znaků Unicode ve jménech souborů
- Jiné obslužné funkce, například šifrování archivů, komentáře, zaznamenávání chyb atd.



Obr. 10. Vzhled programu WinRAR

ZÁVĚR

Hlavním cílem mé bakalářské práce bylo nastudování tématu „Komprese binárních, textových a grafických dat“ a následná tvorba literární rešerše.

V první kapitole jsem obecně popsal kompresi dat, včetně dělení komprese do dalších podskupin. Také jsem zde uvedl dva vztahy, které popisují míru efektivnosti dané kompresní metody.

V druhé kapitole jsem podrobně uvedl problematiku bezztrátové komprese, která se především týká metod RLE, LZW, Huffmanova kódování, Aritmetického kódování a Burrows-Wheelerova transformace. U každé z těchto metod jsem vysvětlil princip kódování a ve většině případů jsem uvedl příklad, který ještě více napomáhá k pochopení dané problematiky. U metod bezztrátové komprese RLE a LZW jsem uvedl nástin algoritmu.

Třetí kapitola této práce pojednává o ztrátové kompresi. Jako zástupce této kategorie jsem uvedl ztrátovou metodu JPEG, která je podrobně vysvětlena. Zmínil jsem se také o progresivním JPEGu, který se používá hlavně pro prezentaci na WWW stránkách. JPEG není jedinou ztrátovou metodou komprese, patří mezi ně také MPEG, WMA a další, o kterých jsem se v této práci pouze zmínil. JPEG jsem si vybral z toho důvodu, že k samotné kompresi používá bezztrátové metody např. Huffmanovo kódování. Ztráty jsou zapříčiněny zaokrouhlováním.

Poslední zpracovanou kapitolou jsou nejpoužívanější komprimační programy. Zařadil jsem zde opravdu jen ty nejpoužívanější, protože těchto komprimačních programů existují desítky. Mezi ně jsem uvedl WinZip, WinRAR a PowerArchiver. U každého z nich jsem se snažil uvést, jaké daný program podporuje formáty, jeho přednosti a další parametry. Ke každému z nich jsem umístil obrázek.

V této práci jsem se snažil o to, aby veškerá zde popisovaná tematika byla snadno pochopitelná a o to, abych i neznalého čtenáře uvedl do světa ať už bezztrátové komprese či komprese obecně.

SEZNAM POUŽITÉ LITERATURY

- [1] JAIN, A., K. *Fundamentals of Digital Image Processing*. Prentice-Hall: London, 1997.
- [2] GONZALEZ, R.C., WOODS, R.E. *Digital Image Processing*. Prentice-Hall: London, 1997.
- [3] PRATT, W.K. *Digital Image Processing*, Prentice-Hall: London, 2000.
- [4] PECINOVSKÝ, J. *Archivace a komprimace dat - snadno a rychle*, Grada Publishing a.s.: Praha, 2003.
- [5] ČAPEK, J., FABIAN, P. *Komprimace dat - principy a praxe*, Computer Press: Praha, 2000.
- [6] MORKES, D. *Komprimační a archivační programy*, Computer Press: Brno, 1998.
- [7] PC SVĚT: *Co je JPEG?* [online]. [cit. 2005-04-12]. Dostupný z WWW: <<http://www.pcsvet.cz/art/article.php?id=2449>>.
- [8] ŠTĚPÁN ROH: *Burrows-Wheelerova transformace* [online]. [cit. 2005-05-28]. Dostupný z WWW: <<http://srnet.cz/~stepan/st/bwt.html>>.

SEZNAM OBRÁZKŮ

Obr. 1. Základní schéma algoritmu RLE	13
Obr. 2. Příklad binárního stromu Huffmanova kódování	21
Obr. 3. Příklad binárního stromu vytvořeného Shannon-Fanovou metodou	24
Obr. 4. Tabulka Aritmetického kódování	27
Obr. 5. Transformace obrázku do vhodného systému barev	32
Obr. 6. Kódování do výstupního souboru	36
Obr. 7. Vzhled programu WinZip	40
Obr. 8. Kontextová nabídka programu WinZip	40
Obr. 9. Vzhled programu PowerArchiver	42
Obr. 10. Vzhled programu WinRAR	43

SEZNAM TABULEK

Tab. 1. Zakódování vstupního řetězce WEB/WEB/WEB!	19
Tab. 2. Postup při kompresi metodou JPEG	36