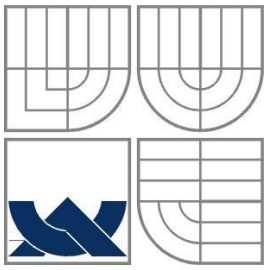


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVĚ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# IMPLEMENTACE STATISTICKÝCH KOMPRESNÍCH METOD

IMPLEMENTATION OF STATISTICAL COMPRESSION METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETER FTOREK
AUTHOR

VEDOUCÍ PRÁCE — Ing. DAVID BAŘINA
SUPERVISOR

BRNO 2013

# Abstrakt

Cílem této práce je popsat statistické metody komprese dat. Úvod pokrývá teoretické minimum komprese dat. Těžiště práce tvoří popis jednotlivých metod a implementace Burrows-Wheelerovho kompresního algoritmu v programovacím jazyce C. Obsahuje výsledky testů jednotlivých metod a jejich vyhodnocení.

# Abstract

The aim of this thesis is to describe statistical methods for data compression. Introduction covers theoretical minimum of data compression. Center of the work is about description of each method and implementation of Burrows-Wheeler compression algorithm in C programming language. It contains test results of each method and their evaluation.

# Klíčová slova

Komprese dat, bezztrátová komprese, BWT, MTF, WFC, IFC, RLE, aritmetické kódování, Huffmanovo kódování

# Keywords

Data compression, lossless compression, BWT, MTF, WFC, IFC, RLE, arithmetic coding, Huffman coding

# Citace

Peter Ftorek: Implementace statistických kompresních metod, bakalářská práce, Brno, FIT VUT v Brně, 2013

# Implementace statistických kompresních metod

# Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Davida Bařiny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Peter Ftorek

15. mája 2013

# Poďakovanie

Chcel by som poďakovať vedúcemu práce Ing. Davidovi Bařinovi za poskytnutie odborných rád a potrebných materiálov k riešeniu práce, rovnako aj za jeho trpezlivosť a pochopenie.

# Obsah

# 1 Úvod

Vývoj informačných technológii sa každý rokom stupňuje veľmi rýchlym tempom. Týka sa to nielen osobných počítačov či serverov, ale aj herných konzolí, prehrávačov rôznych druhov médií, mobilov či iných vstavaných zariadení. Ich spoločným znakom je práca s dátami, ktoré treba ukladať a prenášať. Veľkosť týchto dát narastá spolu s pokrokom IT a vzhľadom na obmedzené kapacity úložných médií a prenosových liniek je čoraz výhodnejšie ich objem zmenšovať pomocou rôznych kompresných metód.

Ich výber a použitie závisí od konkrétnych dát. Základným typom je bezstratová kompresia, kde sa dáta pretransformujú na iné, pričom dôjde k zmenšeniu veľkosti. Opačným postupom je možné získať pôvodné dáta. V určitých prípadoch však nepotrebujeme byť schopní zrekonštruovať originálne údaje. Pri multimédiách sa stretávame hlavne so stratovou kompresiou, kde sa spoliehame na nedokonalosť ľudských zmyslov (zrak a sluch). Z obrazu a zvuku je možné odstrániť určité údaje alebo ich pretransformovať, čím dôjde k zhoršeniu kvality, ale aj zmenšeniu objemu dát. Miera kompresie sa volí tak, aby výsledné zhoršenie nebolo príliš evidentné.

V prvej časti rozoberiem teoretické minimum ako základné delenie metód, možnosti ich porovnávania, či použitie kompresie mimo výpočtovej techniky. Jej obsahom je tiež ťažisko práce, popis jednotlivých častí a metód Burrows-Wheelerovho kompresného algoritmu. Implementácia jeho vybraných častí, ich testovanie a následné zhrnutie získaných výsledkov je obsahom poslednej časti.

# 2 Teória

Definície pojmov v tejto kapitole sú prevzaté z [1] a [5].

Pod pojmom kompresia rozumieme prevod vstupného toku dát na výstupný tok, ktorý má menšiu veľkosť, teda je skomprimovaný, ale zachováva si svoju informačnú hodnotu. Tento prevod nazývame kódovanie, čo je proces vzájomne jednoznačného priradenia symbolov jednej množiny symbolom druhej množiny. Kompresia je dosiahnuteľná redukciou alebo odstránením prebytočných informácií (redundancie). Redundancia vzniká nedokonalým kódovaním alebo pridávaním dodatočných informácií, napríklad pre zabezpečenie alebo rýchlejšie spracovanie dát. Zo skomprimovaných dát je opačným postupom (dekompresiou) možné získať pôvodné dáta. Kompresný a dekompresný algoritmus spolu tvoria kompresnú metódu. Pre rôzne typy dát sú vhodné rozdielne metódy s konkrétnymi parametrami.

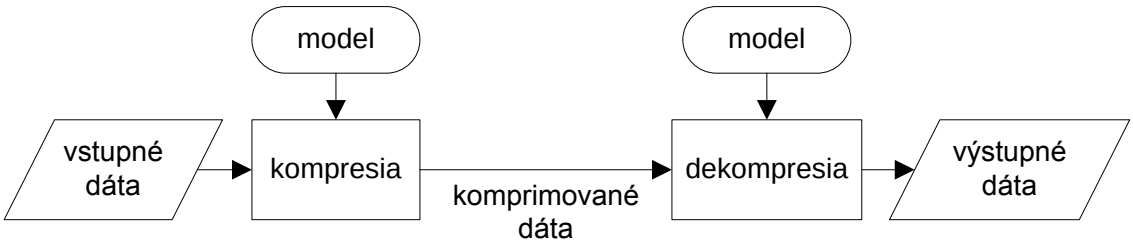
Bezstratová kompresia umožňuje zmenšenie veľkosti bez straty informácie. Po dekompresii získame pôvodný dátový tok. Väčšina základných kompresných metód je bezstratových. Ich použitie nájdeme všade, kde dáta pred kompresiou a po dekompresii musia byť rovnaké – napríklad binárne súbory alebo text. Strata informácie je neprípustná, pretože by viedla k nenávratnej zmene (poškodeniu) súboru. Základné bezstratové typy sú slovníkové a štatistické metódy.

Stratová kompresia je sprevádzaná stratou informácie, čo obecne vedie k lepšiemu kompresnému pomeru. Neexistuje však dekompresný algoritmus pre spätné získanie pôvodného dátového toku. Ich uplatnenie nájdeme pri obrázkoch, videu a zvuku, kde využívajú nedokonalosť ľudského zraku a sluchu rozlíšiť stratu informácie. V daných prípadoch je možné tolerovať aj určitú mieru vnímania rozdielu v kvalite dát, napríklad horšia kvalita telefónneho zvuku verzus kvalita CD.

Štatistické metódy sú založené na matematickej pravdepodobnosti výskytov jednotlivých symbolov. Určité symboly sa vo vstupných dátach vyskytujú častejšie a práve tento fakt je využívaní pri komprimácii, kde sú nahradzované kratšími sekvenciami. Patrí sem aritmetické a Huffmanovo kódovanie. Radíme sem aj kontextové metódy, ktoré využívajú znalosti predchádzajúcich symbolov na predikciu nasledujúcich.

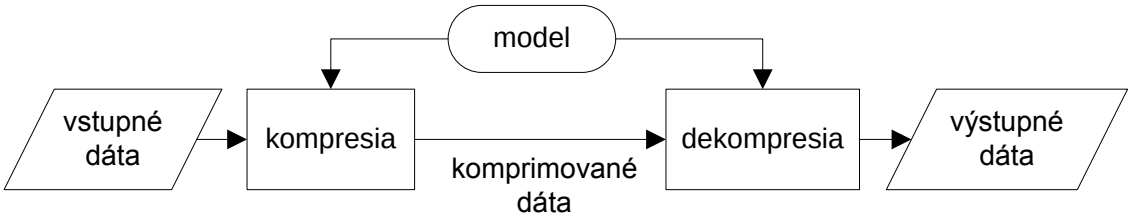
Slovníkové metódy vyhľadávajú vo vstupe opakujúce sa reťazce, ktoré zapisujú do špeciálnej dátovej štruktúry – slovníka. Pokiaľ je pri kompresii vo vstupe nájdený reťazec zapísaný v slovníku, je na výstupe nahradzovaný odkazom na príslušný záznam v slovníku. Patria sem metódy LZ77, LZ78, LZW a LZMA.

Adaptívne metódy svoje parametre a operácie menia počas samotnej kompresie na základe skúmania vstupných dát. Z tohto dôvodu sú veľmi univerzálne a z časového hľadiska vyžadujú len jeden prechod vstupnými dátami.



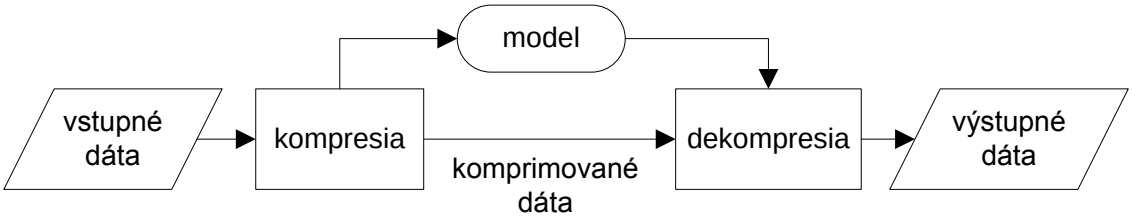

**Obrázok 2.1:** Adaptívny model

Statické (neadaptívne) metódy používajú rovnaké parametre a operácie počas celej doby kompresie, nezávisle na vstupných dátach. Preto je konkrétna metóda s konkrétnymi parametrami vhodná len pre istý typ dát, kde dosahuje najlepšie výsledky. Vhodné nastavenie sa zvyčajne vytvára na určitej vzorke vstupných dát veľkého objemu, preto ak sa skutočný vstupný tok bude príliš líšiť od exemplárneho, tak môže dôjsť k extrémnemu zhoršeniu kompresie.




**Obrázok 2.2:** Statický model

Semiadaptívne metódy sú typicky dvojfázové. Pri prvom prechode vstupnými tokom sa parametre a operácie prispôsobia dátam. V druhom prechode už prebieha samotná kompresia podľa napevno nastavených údajov. Tieto metódy sú však zvyčajne časovo najpomalšie a v istých prípadoch úplne nepoužiteľné. Nastavenie metódy sa často prikladá k samotným výstupným dátam, aby bolo dostupné pre dekompresný algoritmus.




**Obrázok 2.3:** Semiadaptívny model

## 2.1 Metriky výkonnosti

Kompresné metódy sa líšia algoritmickou, pamäťovou aj časovou zložitosťou. Tieto veličiny sú nedostatočné pre porovnávanie týchto metód a preto bolo zavedených niekoľko ďalších veličín.

Kompresný pomer je pomer medzi celkovou veľkosťou výstupných dát po kompresii a celkovou veľkosťou vstupných dát pred kompresiou.

$$kompresný\ pomer = \frac{veľkosť\ výstupných\ dát}{veľkosť\ vstupných\ dát} \tag{2.1}$$

Pokiaľ je pomer v intervale 0 až 1, znamená to úspešnú kompresiu. Napríklad pomer 0.72 znamená, že sme ušetrili 28% miesta. Hodnota 1 označuje, že výsledné dáta majú rovnakú veľkosť ako pôvodné a hodnoty nad 1 sú neželané, namiesto kompresie sme získali expanziu.

Kompresný faktor získame obrátenou hodnotou kompresného pomeru. Faktor väčší ako 1 značí úspešnú kompresiu a hodnoty pod 1 naopak neželané zväčšenie výstupného toku dát.

$$kompresný\ faktor = \frac{veľkosť\ vstupných\ dát}{veľkosť\ výstupných\ dát} \tag{2.2}$$

Veličina bpc (bits per character) má podobný výpočet ako kompresný pomer. Udáva priemerný počet výstupných bitov potrebných k reprezentácii jedného vstupného znaku.

$$bpc = \frac{8 \times veľkosť\ výstupnych\ dát}{veľkosť\ vstupných\ dát} \tag{2.3}$$

Kompresný zisk slúži na porovnávanie metód, jednotkou je percentuálny logaritmický pomer (percent log ratio) značený ako o/o. Vďaka prítomnosti logaritmu je možné porovnať pomocou rozdielu hodnôt dva kompresné zisky. Ako referenčnú veľkosť môžeme použiť veľkosť výstupného toku štandardnej bezstratovej kompresnej metódy alebo veľkosť vstupného toku.

$$kompresný\ zisk = 100 \times log_e \times \frac{referenčná\ veľkosť}{komprimovaná\ veľkosť} \tag{2.4}$$

Počet cyklov na byte (cpb) sa používa pre meranie rýchlosti kompresie. Udáva stredný počet strojových cyklov potrebných na kompresiu jedného bytu. Dôležitá je v prípadoch, kde vyžadujeme rýchlu kompresiu pomocou dedikovaných obvodov.

Entropia symbolu ($E_i$), najmenší počet bitov potrebných na jeho reprezentáciu, je závislá od pravdepodobnosti ($P_i$) výskytu symbolu vo vstupných dátach. Vstupné dáta nemajú ideálnu štruktúru z pohľadu efektívnosti využívania miesta, pretože obsahujú redundanciu. Mnohé informácie sa vyskytujú viackrát ako ostatné, prípadne sa opakujú celé úseky. Kompresia má za úlohu zefektívniť uloženie, aby veľkosť zodpovedala obsiahnutým informáciám.

$$E_i = -\log_2 P_i \tag{2.5}$$

Entropia dát je priemerná entropia, ktorá sa počíta z entropie symbolov. Ak sú pravdepodobnosti výskytov symbolov zhodné, tak hodnota je najvyššia.

$$E_{avg} = -\sum_{i=1}^{n} P_i \log_2 P_i \tag{2.6}$$

Redundanciu je možné určiť ako rozdiel priemernej a skutočnej entropie dát. Tú možno získať ako hodnotu bpc.

$$R = bpc - E_{avg} \tag{2.7}$$

## 2.2 Vstupné dáta

Aby sa kompresné metódy mohli porovnávať uvedenými veličinami, je nutné zaistiť rovnaké vstupné dáta. Pre tieto potreby boli vytvorené korpusy, čo sú kolekcie súborov vybraných podľa určitých kritérií. Obsahy súborov by mali byť odlišného typu, aby nebola zvýhodnená žiadna konkrétna metóda vhodná pre určitý typ vstupných dát. Typy zvolených súborov by mali reprezentovať skutočné dáta používané v reálnej výpočtovej technike, napríklad text, video alebo databázy. Ich veľkosť by mala byť zvolená primerane a nesmú byť chránené autorským zákonom, teda môžu byť verejne šírené.

### 2.2.1 Calgary korpus

Calgary korpus [2] je pomenovaný podľa svojho miesta vzniku, na Univerzite Calgary. Pochádza z roku 1987 a ako štandard sa používal vyše desať rokov. V dnešnej dobe je už zastaraný, no je možné ho použiť pre porovnanie súčasných metód s výsledkami starších algoritmov. Pozostáva zo štrnástich textových a binárnych súborov, z pôvodnej osemnásť súborovej verzie boli odstránené súbory paper3 až paper6 pre vzájomnú podobnosť.

| Súbor | Typ dát | Veľkosť v B |
|---|---|---|
| bib | ASCII text v UNIX-ovom formáte referencií | 111261 |
| book1 | Neformátovaný ASCII text – kniha | 768771 |
| book2 | ASCII text v UNIX-ovom formáte „troff“ – kniha | 610856 |
| geo | Geofyzikálne dáta tvorené 32-bitovými desatinnými číslami | 102400 |
| news | ASCII text USENET dávkového súboru | 377109 |
| obj1 | Kompilovaný program pre VAX | 21504 |
| obj2 | Kompilovaný program pre Macintosh | 246814 |
| paper1 | UNIX-ový formát „troff“ – technický dokument | 53161 |
| paper2 | UNIX-ový formát „troff“ – technický dokument | 82199 |
| pic | Čiernobiely obrázok z faxu (1728 x 2376 pixelov) | 513216 |
| progc | Zdrojový kód jazyka C | 39611 |
| progl | Zdrojový kód jazyka Lisp | 71646 |
| progp | Zdrojový kód jazyka Pascal | 49379 |
| trans | Záznam prenosu dát terminálu vrátane špeciálnych znakov | 93695 |
| **Celková veľkosť** | | 3141622 |

**Tabuľka 2.1:** Používané súbory Calgary korpusu

| Súbor | Typ dát | Veľkosť v B |
|---|---|---|
| paper3 | UNIX-ový formát „troff“ – technický dokument | 46526 |
| paper4 | UNIX-ový formát „troff“ – technický dokument | 13286 |
| paper5 | UNIX-ový formát „troff“ – technický dokument | 11954 |
| paper6 | UNIX-ový formát „troff“ – technický dokument | 38105 |
| **Celková veľkosť (spolu s používanými súbormi)** | | 3251493 |

**Tabuľka 2.2:** Nepoužívané súbory Calgary korpusu

### 2.2.2 Canterbury korpus

Canterbury korpus [3] je následníkom zastaraného Calgary korpusu. Vytvorený bol v roku 1997 na Univerzite Canterbury. Zahŕňa jedenásť súborov, ktoré boli zvolené tak, aby lepšie reprezentovali „priemerné“ reálne súbory daného typu. Ďalším dôvodom pre jeho vytvorenie bolo postupné zanikanie niektorých typov súborov z Calgary korpusu a ich nahradzovanie novými typmi. Niektoré metódy boli navyše špeciálne upravené alebo vytvorené priamo pre dáta Calgary korpusu, čo skresľovalo objektívne hodnotenie daných algoritmov.

| Súbor | Skratka | Typ dát | Veľkosť v B |
|---|---|---|---|
| alice29.txt | text | Anglický text | 152089 |
| asyoulik.txt | play | Text – dielo Shakespeara | 125179 |
| cp.html | html | Zdrojový kód jazyka HTML | 24603 |
| fields.c | Csrc | Zdrojový kód jazyka C | 11150 |
| grammar.lsp | list | Zdrojový kód jazyka LISP | 3721 |
| kennedy.xls | Excl | Tabuľky z Excel-u | 1029744 |
| lcet10.txt | tech | Technický dokument | 426754 |
| plrabn12.txt | poem | Poézia | 481861 |
| ptt5 | fax | Súbor obrázkov z faxu | 513216 |
| sum | SPRC | Kompilovaný program pre SPARC | 38240 |
| xargs.1 | man | Manuálové stránky GNU | 4227 |
| **Celková veľkosť** | | | 2810784 |

**Tabuľka 2.3:** Súbory Canterbury korpusu

Verzia Large Corpus [3] existuje s tromi súbormi, ktoré sú však omnoho väčšie. Používa sa na porovnávanie algoritmov, ktoré dosahujú lepšie výsledky pri veľkých súboroch. Pre testovanie najhorších prípadov vstupných dát (napríklad extrémne malá alebo naopak extrémne veľká frekvencia opakovania symbolov) sa používa Artificial Corpus. Výsledky dosiahnuté na tomto korpuse sa zväčša používajú na vyhľadávanie abnormálneho chovania kompresných metód. Miscellaneous Corpus určený pre dopĺňanie momentálne pozostáva iba z jedného súboru pi.txt, ktorý obsahuje jeden milión začiatočných číslic hodnoty π.

| Súbor | Skratka | Typ dát | Veľkosť v B |
|---|---|---|---|
| bible.txt | bible | Kniha – Biblia | 4047392 |
| E.coli | E.coli | Génom baktérie E.Coli | 4638690 |
| world192.txt | world | Kniha údajov | 2473400 |
| **Celková veľkosť** | | | 11159482 |

**Tabuľka 2.4:** Súbory Large korpusu

### 2.2.3 Silesia korpus

Silesia korpus [4] pochádza z roku 2003 zo Sliezskej Univerzity Technológií. Jeho cieľom je poskytnúť typické dátové súbory používané v súčasnej výpočtovej technike, napríklad databázy

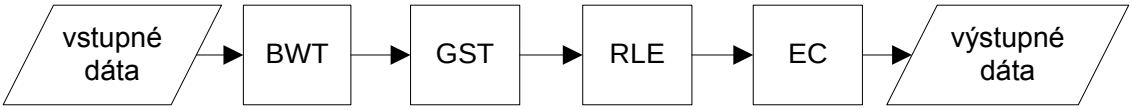
a archívy dát aplikácií. Vzhľadom na bežné používanie stratovej kompresie pri multimediálnych súboroch nie je tento typ dát zahrnutý pre testovanie bezstratových metód. Výnimku tvoria medicínske obrazy, u ktorých strata kvality (teda informácie) nie je prípustná. Súbory sú väčšie ako pri starších korpusoch, v súčasnosti sa však pri istých typoch súborov bežne pracuje s omnoho väčšími veľkosťami.

| **Súbor** | **Typ dát** | **Veľkosť v B** |
|---|---|---|
| dickens | Anglický text – diela Charlesa Dickensa | 10192446 |
| mozilla | TAR archív – program Mozilla 1.0 | 51220480 |
| mr | 3D obraz – magnetická rezonancia | 9970564 |
| nci | Text – databáza chemických štruktúr | 33553445 |
| ooffice | Windows DLL knižnica – OpenOffice 1.01 | 6152192 |
| osdb | Vzorková MySQL databáza | 10085684 |
| reymont | Nekomprimované PDF – poľský text | 6627202 |
| samba | TAR archív – program Samba 2.2.3a | 21606400 |
| sao | Binárna databáza – informácie o hviezdach | 7251944 |
| webster | HTML - anglický slovník | 41458703 |
| x-ray | Obrázok v šedých odtieňoch – röntgen | 8474240 |
| xml | TAR archív – XML súbory | 5345280 |
| **Celková veľkosť** | | 211938580 |

**Tabuľka 2.5:** Súbory Silesia korpusu

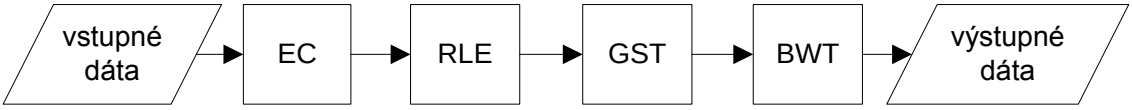
## 2.3 Burrows-Wheelerov algoritmus

Burrows-Wheelerov kompresný algoritmus [6] pozostáva z viacerých postupov, ktoré nasledujú postupne po sebe tak, že výstup predchádzajúcej fázy je vstupom nasledujúcej. Dáta sú spracovávané po samostatných blokoch, čo ponúka možnosti paralelného spracovania pri viacjadrových procesoroch. Prvou fázou je Burrows-Wheeler Transformation (BWT), ktorá zhlukuje symboly s podobným kontextom. V tejto fáze nedochádza ešte ku kompresii, naopak pribudne index, ktorý je potrebný pre spätný proces. Po nej nasleduje Global Structure Transformation (GST), ktorá mení lokálny kontext symbolov na globálny. Aj v tomto prípade ostáva počet symbolov rovnaký. Táto fáza má niekoľko rôznych implementácií. Treťou fázou je Run Length Encoding (RLE), ktorá znižuje počet symbolov. Poslednou fázou je Entropy Coding (EC), ktorá komprimuje symboly pomocou adaptovaného modelu.




**Obrázok 2.4:** Schéma kompresného postupu

Pri dekompresii sa poradie fáz musí otočiť, teda začíname entropickým kodérom EC a končíme Burrows-Wheelerovou transformáciou BWT.




**Obrázok 2.5:** Schéma dekompresného postupu

### 2.3.1 Burrows-Wheeler Transformation

Tento algoritmus namiesto načítavania a spracovávania jednotlivých znakov (prúdový režim) pracuje v blokovom režime [6] [1], teda vstupom sú celé reťazce symbolov s presne danou dĺžkou. Tieto bloky sú následne transformované tak, aby obsahovali zhluky rovnakých symbolov. Väčšie bloky poskytujú lepšie šance pre vytváranie väčších zhlukov a tým pádom možnosť lepšej komprimácie. Takéto koncentrácie symbolov sú vhodnejšie pre nasledujúce algoritmy GST (napríklad MFT) a RLE. Metóda je veľmi univerzálna, dosahuje dobré výsledky pri audio, video aj textových dátach. Priebeh kódovania pomocou tejto metódy bude ukázané na nasledujúcom príklade:

1. Zo vstupu je načítaný reťazec symbolov „TROLOLO“ s veľkosťou n (tu n = 7).

| T | R | O | L | O | L | O |
|---|---|---|---|---|---|---|

**Tabuľka 2.6:** Vstupný reťazec

2. Vytvorí sa matica s rozmermi n krát n. Nultý riadok tvorí vstupný reťazec „TROLOLO“, každý nasledujúci riadok vznikne cyklickou rotáciou svojho predchodcu o jeden symbol doprava. Prvý stĺpec matice označíme X a posledný Y.

| Index | X | | | | | | Y |
|---|---|---|---|---|---|---|---|
| **0** | T | R | O | L | O | L | O |
| **1** | O | T | R | O | L | O | L |
| **2** | L | O | T | R | O | L | O |
| **3** | O | L | O | T | R | O | L |
| **4** | L | O | L | O | T | R | O |
| **5** | O | L | O | L | O | T | R |
| **6** | R | O | L | O | L | O | T |

**Tabuľka 2.7:** Matica s rotovanými reťazcami

3. Riadky matice sa abecedne zoradia. Index pôvodného vstupného reťazca značíme Z.

| Index | X | | | | | | Y |
|---|---|---|---|---|---|---|---|
| **0** | L | O | L | O | T | R | O |
| **1** | L | O | T | R | O | L | O |
| **2** | O | L | O | L | O | T | R |
| **3** | O | L | O | T | R | O | L |
| **4** | O | T | R | O | L | O | L |
| **5** | R | O | L | O | L | O | T |
| **6 -> Z** | T | R | O | L | O | L | O |

**Tabuľka 2.8:** Abecedne zoradená matica

4. Výstupom je index Z a nový stĺpec Y, ktorý obsahuje zhluky symbolov vhodné pre nasledujúce kroky kompresie. Index Z je potrebný pre spätnú transformáciu Y na pôvodný vstup.

Praktické implementácie nepoužívajú celú maticu z dôvodu vysokej pamäťovej náročnosti pri veľkých vstupoch, navyše rotácie by tiež boli časovo zdĺhavé. Namiesto toho sa pracuje iba so vstupným reťazcom a poľom ukazateľov (indexov) odkazujúcich do vstupného reťazca na začiatky rotovaných reťazcov (riadky matice).

Spätný proces je zložitejší, pretože algoritmus na spätné preusporiadanie abecedne zoradených symbolov všeobecne neexistuje. V tomto prípade je to možné vďaka stĺpcu Y a indexu Z, pričom je tento proces jednoduchší a rýchlejší ako pôvodná transformácia. Algoritmus je ukázaný na dátach z predchádzajúceho príkladu:

1. Stĺpce Y a X obsahujú tie isté symboly, to využijeme na opätovné zostavenie X z Y obyčajným abecedným zoradením symbolov z Y.
2. Začíname na pozícii udanej indexom Z.
3. Výstupom je symbol z X na aktuálnej pozícii. Presunieme sa na riadok, ktorý má rovnakú pozíciu v Y vo vnútri množiny (zhluku) posledného symbolu ako pozícia posledného symbolu vo vnútri množiny (zhluku) posledného symbolu v X. Tento krok opakujeme, pokiaľ aktuálna pozícia nedosiahne znova index Z.

| **Index** | **X** | **Y** | **Krok** |
|---|---|---|---|
| **0** | L | O | 4. |
| **1** | L | O | 6. |
| **2** | O | R | 3. |
| **3** | O | L | 5. |
| **4** | O | L | 7. |
| **5** | R | T | 2. |
| **6 -> Z** | T | O | 1. |

**Tabuľka 2.9:** Matica dekompresného postupu s vyznačenými krokmi

Pri prvom kroku (pozícia 6) je aktuálnym (posledným) symbolom „T“. Presunieme sa na riadok 5, pretože symbol „T“ sa vyskytuje len raz, práve na tejto pozícii v stĺpci Y. Výstupom je „R“, presunieme sa na riadok 2 (znova unikátny symbol). Získame symbol „O“, ktorý je prvý v svojej množine rovnakých symbolov v X. Prvý symbol „O“ v Y sa nachádza na pozícii 0, preto sa presunieme tam. Získaný symbol „L“ je znova prvý v svojej množine, ďalšia pozícia je teda riadok 3. Symbol „O“ je teraz druhý v svojej množine a druhý symbol „O“ v Y je na pozícii 1. Získané „L“ je tiež druhé, jeho druhý výskyt v Y nájdeme na riadku 4. Posledným výstupom je symbol „O“, ktorého tretí výskyt nás odkáže na pozíciu 6, čo je začiatočný index Z a teda algoritmus končí.

### 2.3.2 Global Structure Transformation

Druhá fáza označovaná ako GST spracováva výstup predchádzajúceho BWT. Tieto dáta obsahujú zhluky rovnakých symbolov, ktorých pravdepodobnostné rozdelenie a lokálny kontext sa môže veľmi líšiť v každom spracovávanom bloku vstupných dát. Úlohou tejto fázy je zmeniť lokálny kontext symbolov (v blokoch) na globálny kontext (v celom vstupe), aby dáta získali

lepšiu formu pre spracovanie nasledujúcimi fázami. GST nie je jeden algoritmus, ale súhrnné označenie pre niekoľko rôznych algoritmov.

### 2.3.3 Move-To-Front

MTF [6] [1] je algoritmus používaný v mnohých implementáciách Burrows-Wheelerovej komprimačnej metódy. Symboly vstupného toku prevádza na výstupnú sekvenciu indexov. Tieto indexy určujú posledný výskyt daného symbolu, čím menšie číslo, tým bližšie je posledná známa pozícia. Aby sme tieto indexy mohli vytvárať, musí existovať abeceda vstupných symbolov vo forme usporiadaného zoznamu, kde často sa vyskytujúce symboly umiestňujeme na jej začiatok. Abeceda je inicializovaná tak, že symboly sú abecedne usporiadané. Výstupný index udáva, koľko symbolov sa nachádza v aktuálnej abecede pred práve spracovávaným symbolom. Algoritmus je pomenovaný Move-to-Front podľa procesu, keď po vypočítaní aktuálneho výstupného indexu dôjde k presunu súčasného symbolu na začiatok abecedy. Predpokladá sa, že aktuálny symbol sa na vstupe objaví viackrát za sebou a preto bude zakódovaný nižším číslom.

Spätný postup dekódovania pracuje na rovnakom princípe. Na začiatku máme k dispozícii rovnakú abecedu symbolov. Zo vstupu preberáme indexy označujúce poradie symbolu v abecede, ktorý pošleme na výstup a následne ho v abecede presunieme na jej začiatok.

| **Vstup** | **Abeceda** | **Index** | **Abeceda po MFT** |
|---|---|---|---|
| O | L, O, R, T | 1 | O, L, R, T |
| O | O, L, R, T | 0 | O, L, R, T |
| R | O, L, R, T | 2 | R, O, L, T |
| L | R, O, L, T | 2 | L, R, O, T |
| L | L, R, O, T | 0 | L, R, O, T |
| T | L, R, O, T | 3 | T, L, R, O |
| O | T, L, R, O | 3 | O, T, L, R |

**Tabuľka 2.10:** Proces kódovania reťazca z ukážky činnosti BWT

| Index | Abeceda | Výstup | Abeceda po MFT |
|---|---|---|---|
| 1 | L, O, R, T | O | O, L, R, T |
| 0 | O, L, R, T | O | O, L, R, T |
| 2 | O, L, R, T | R | R, O, L, T |
| 2 | R, O, L, T | L | L, R, O, T |
| 0 | L, R, O, T | L | L, R, O, T |
| 3 | L, R, O, T | T | T, L, R, O |
| 3 | T, L, R, O | O | O, T, L, R |

**Tabuľka 2.11:** Proces dekódovania

Táto metóda je jednoduchá a relatívne rýchla. Nedostatkom je, že na začiatok abecedy presúva každý symbol nezávisle na počte jeho predchádzajúcich výskytov. Pokiaľ je symbol zriedkavý, zbytočne odsúva častejšie sa vyskytujúce symboly zo začiatku abecedy, čo má za následok zvyšovanie hodnôt výstupných indexov. Vyššie čísla zaberajú pri kódovaní viacej miesta a to je pre komprimáciu nežiaduci efekt.

Preto bolo vytvorených niekoľko jednoduchých modifikácií, ktoré čiastočne tento problém odstraňujú. MTF-1 presúva na prvú pozíciu abecedy (index 0) len symboly z druhej pozície. Symboly presúvané z pozície tri a viac sa teda presúvajú namiesto prvej na druhú pozíciu. MTF-2 je vylepšenie MTF-1 založené na princípe, že symboly z druhej pozície sa môžu presunúť na prvú len vtedy, ak predchádzajúci index nebol nula. Ten by značil opakujúci sa symbol, teda je výhodnejšie ho nechať na prvej pozícii. Ďalšie modifikácie sú napríklad M1 a M2, ktoré používajú indikátory na dosiahnutie väčšieho počtu indexov 0 na výstupe za cenu ostatných indexov.

Vhodné vstupné dáta sú tie, ktoré spĺňajú koncentračnú vlastnosť, čo je predpokladaný výskyt viacero rovnakých symbolov v slede. Práve takto dokážeme získať veľa nízkych hodnôt výstupných indexov, ktoré sa lepšie a úspornejšie zakódujú Huffmanovým alebo aritmetickým kódovaním. Z toho vyplýva vlastnosť lokálnej adaptívnosti. Abeceda a z nej vyplývajúce indexy sa prispôsobujú výskytom symbolov v spracovávanej lokálnej časti vstupných dát.

### 2.3.4 Timestamp

Algoritmus časovej značky [7] podobne ako metóda MTF vytvára na výstupe hodnoty podľa indexov aktuálnych vstupných symbolov v abecede. Aj v tomto postupe nájdeme presúvanie aktuálneho znaku na začiatok abecedy. Tento presun je však upravený, daný symbol sa presunie iba pred prvý symbol abecedy, ktorý bol najviac jeden krát spracovaný od posledného

spracovania aktuálneho symbolu. K presunu nedôjde, ak aktuálny symbol ešte nebol zatiaľ nikdy spracovávaný. Z dôvodu uchovávania informácií o počte spracovávaní jednotlivých symbolov je potrebné dvojrozmerné pole s rozmermi veľkosti abecedy na druhú. To sa prejavuje na zvýšenej pamäťovej náročnosti oproti ostatným metódam. Postup bude predvedený na vstupnom reťazci „TROLOLO“:

| Vstup | Abeceda | Výstup | Nová abeceda |
|---|---|---|---|
| T | L, O, R, T | 3 | L, O, R, T |
| R | L, O, R, T | 2 | L, O, R, T |
| O | L, O, R, T | 1 | L, O, R, T |
| L | L, O, R, T | 0 | L, O, R, T |
| O | L, O, R, T | 1 | O, L, R, T |
| L | O, L, R, T | 1 | O, L, R, T |
| O | O, L, R, T | 0 | O, L, R, T |

**Tabuľka 2.12:** Proces kódovania reťazca „TROLOLO“

### 2.3.5 Inversion Frequencies

Táto metóda pracuje na základe merania vzdialenosti výskytov symbolov [6]. Pre každý symbol abecedy postupne vytvára výstupnú sekvenciu analýzou vstupu. Pokiaľ je aktuálne skúmaný symbol vstupu zhodný s aktuálne skúmaným symbolom abecedy, výstup je rovný počtu symbolov väčších ako aktuálne skúmaný symbol abecedy, ktoré sa nachádzajú medzi aktuálnou a poslednou pozíciou výskytu tohto symbolu na vstupe. Aby sme získaný výstup boli schopní dekódovať, potrebujeme poznať počet výskytov symbolov abecedy vo vstupných dátach alebo musíme použiť špeciálny ukončovací symbol za každým symbolom abecedy. Týmto sa však zvyšuje celková veľkosť výstupných dát.

Výhodou je, že posledný symbol abecedy sa nemusí kódovať, pretože výstupom v tomto prípade sú samé nuly. Preto tento symbol nie je potrebný pre spätné dekódovanie a tým sa zmenšuje veľkosť výstupu. V porovnaní s MTF poskytuje lepšie kompresné pomery vo väčšine prípadov. Výstup IF neobsahuje toľko sledov núl, ktoré pri MTF znamenajú sled rovnakého symbolu. Väčšie symboly vstupnej abecedy sú kódované sekvenciami menších čísel v porovnaní s menšími symbolmi. Priemerný podiel núl na výstupe narastá, ako sa blížime ku koncu vstupných dát. Na konci dosiahne až 100%, v porovnaní s MTF, kde táto hodnota kolíše priemerne okolo 60%. Postup je ukázaný na príklade vstupného reťazca „TROLOLO“.

| Vstupný symbol | Výstupná sekvencia |
|---|---|
| L | 0, 1 |
| O | 0, 0, 0 |
| R | 0 |
| T | nekóduje sa |

**Tabuľka 2.13:** Vstupné a výstupné údaje pri kódovaní

Vylepšením tejto metódy je Sorted Inversion Frequencies (SIF). Problémom IF je fakt, že vo výstupnej sekvencii pre každý symbol abecedy sa počítajú iba väčšie symboly vyskytujúce sa medzi poslednou a aktuálnou pozíciou. Pokiaľ na vstupe budú najprv spracovávané symboly s vysokým počtom výskytov, tak nasledujúce symboly s menším výskytom budú mať menšie hodnoty. Usporiadanie abecedy a frekvencia výskytov jednotlivých symbolov má preto veľký vplyv na výsledný kompresný pomer. Usporiadať ju môžeme zostupne alebo vzostupne podľa frekvencie výskytov symbolov, smer zoradenia volíme podľa konkrétneho vstupného súboru. So vzostupným zoradením sa v praxi stretávame častejšie.

Pre zistenie optimálneho smeru zoradenia pre aktuálny vstupný súbor je možné použiť nasledujúci výpočet. Pre každý symbol abecedy zistíme počet jeho výskytov vo vstupných dátach. Určíme množinu symbolov, pre ktoré je tento počet väčší ako dvojnásobok priemerného počtu výskytov.

$$G = \left\{ a \,\middle|\, f_a > 2 \times \frac{n}{|A_{in}|} \right\} \tag{3.1}$$

$$S = 100 \times \frac{|G|}{|A_{in}|} \tag{3.2}$$

Požadovaný výsledok získame ako percentuálny podiel týchto symbolov z celkovej abecedy. Pokiaľ je táto hodnota rovná alebo vyššia ako 10, zoradíme abecedu vzostupne, v opačnom prípade zostupne.

### 2.3.6 Weighted Frequency Count

WFC [6] sa svojim algoritmom približuje viac metóde MTF ako IF. Každý vstupný symbol nahradzuje na výstupe príslušnou hodnotiacou váhou. Túto hodnotu počíta hodnotiaca funkcia, ktorá sa v prípade MTF udáva index súčasného vstupného symbolu v aktuálnej abecede. Abeceda sa aktualizuje pri každom vstupnom symbole jeho posunom na začiatok abecedy. Táto zmena neberie do úvahy predchádzajúcu početnosť výskytu tohto symbolu, takže častejšie sa

vyskytujúce symboly môžu byť odsunuté tými menej početnými. Tento nedostatok majúci sa následok menej efektívnejšiu kompresiu sa dá odstrániť zavedením posuvného okna, v ktorom sa počítajú výskyty daných symbolov a vzdialenosti medzi týmito výskytmi. Každej pozícii v tomto posuvnom okne je pridelená váha, pričom bližšie vzdialenosti majú väčšie váhy ako tie vzdialenejšie. Každý symbol získa súčet váh vyplývajúci z jeho výskytov v posuvnom okne. Tieto súčty sa zoradia zostupne, teda najnižší súčet bude na poslednej pozícii. Symboly s vyššou frekvenciou výskytu budú mať vyšší súčet a teda nižší index ako menej početné symboly. Nevýhodou je vyššia časová náročnosť z dôvodu opätovného procesu váženia a zoraďovania pre každý spracovávaný symbol. V priemere tým ale získame nižšie hodnotiace indexy ako pri MTF, čo vedie k lepším výsledkom kompresie.

Vylepšením je Advanced Weighted Frequency Count, ktorá pri hodnotení nepoužíva fixné váhové koeficienty. Vstupné dáta sa obecne veľmi líšia štruktúrou a výskytom symbolov v každom type súborov zvlášť, preto fixné koeficienty nevedú k optimálnemu kompresnému pomeru pre každý súbor. Pri určitých súboroch sú vhodnejšie približne rovnaké váhové hodnoty pre staršie symboly ako pre nedávne, naopak v istých prípadoch dosahujú lepšie výsledky omnoho vyššie váhy prideľované nedávnym symbolom v porovnaní s váhami starších symbolov. Tento nedostatok sa odstráni dynamickým počítaním váh v závislosti na výskyte symbolov.

### 2.3.7 Incremental Frequency Count

Metóda WFC [8] [9] poskytuje kvalitné kompresné pomery, problémom ostáva vysoká časová náročnosť kvôli opätovnému prepočítavaniu váh symbolov v posuvnom okne a zoraďovaní výsledkov pre každý spracovávaný symbol. IFC ponúka podobne dobré kompresné pomery, ale je podstatne rýchlejšia. Princíp posuvného okna a súboru súčtov váh pre každý symbol ostáva, zjednodušenie spočíva v zoraďovaní týchto súčtov. Iba jeden súčet váh pre aktuálne spracovávaný symbol sa aktualizuje, takže namiesto triedenia všetkých súčtov stačí správne zatriediť len tento jeden konkrétny súčet, preto poskytuje lepšie časové výsledky ako WFC.

Aby boli aktuálne symboly lepšie váhovo hodnotené ako tie staršie, súčty môžu byť príslušne zvyšované alebo znižované. Metóda je pomenovaná Incremental Frequency Count podľa faktu, že v priemere dochádza práve k ich zvyšovaniu. Aby sa zamedzilo pretečeniu súčtov, často dochádza k ich preškálovaniu.

Pre každý vstupný symbol je vytvorená výstupná hodnota rovná indexu jeho súčtu v zozname súčtov. Na začiatku sú všetky súčty vynulované a vzostupne abecedne zoradené podľa príslušných symbolov. Zoznam je udržovaný zostupne zoradený podľa hodnôt súčtov, súčet s najvyššou hodnotou je teda na indexe nula. Po každom spracovaní vstupu sa počíta

rozdiel medzi priemerom aktuálneho indexu a priemerom minulého indexu. Priemer indexu je priemerná hodnota posledných indexov so zameraním na súčasne spracovávaný symbol. Priemer sa počíta:

$$mean_i = \frac{(mean_{i-1} \times (ifc\_size - 1)) + index}{ifc\_size} \tag{3.3}$$

Malé hodnoty veľkosti posuvného okna zrýchľujú algoritmus, pretože pri väčších hodnotách pomalšie reaguje na zmeny kontextu. Ďalej sa počíta inkrement, ktorý upraví hodnotu súčtu daného symbolu. Aby bol výstup kvalitnejší, hodnota inkrementu nesmie byť zvolená konštantne, lineárne alebo exponenciálne. Získame ho analýzou štatistických vlastností posledných indexov. Vypočítame rozdiel posledného priemeru a aktuálneho priemeru:

$$diff_i = mean_i - mean_{i-1} \tag{3.4}$$

Aby sa malé zmeny dostatočne prejavovali a naopak veľké rozdiely zbytočne neprevažovali, výsledok je dodatočne upravovaný:

$$diff_i = \min\left(|diff_i|,\ ifc\_diff\right) \times sign\left(diff_i\right) \tag{3.5}$$

Toto limitovanie maximálnej hodnoty rozdielu znižuje vplyv veľkých indexov, ktoré sa vyskytujú pri zmenách kontextu. V poslednom kroku sa počíta samotný inkrement. Pokiaľ sa kontext začne meniť, jeho hodnota znižuje, naopak pri stabilnom kontexte sa zvyšuje. Takto sú časté symboly v stabilnom kontexte váhovo hodnotené viac ako nové symboly v meniacom sa kontexte.

$$change_i = change_{i-1} - \frac{(change_{i-1} \times diff_i)}{ifc\_scale} \tag{3.6}$$

Získaný inkrement sa pričíta k danému súčtu. Pokiaľ súčet prekročí zvolenú hodnotu, inkrement aj všetky súčty sa vydelia dvoma, aby sa zabránilo ich pretečeniu. Zoznam súčtov je nakoniec znova zoradený, ale vzhľadom na fakt, že iba jeden súčet zmenil hodnou, dôjde iba k jednému presunu na správnu pozíciu. Pokiaľ je hodnotou rovný iným súčtom na novej pozícii, je posunutý na najnižší index z tejto skupiny.

### 2.3.8 M03

Táto metóda nie je štandardnou metódou GST [6], pretože jej výstup nemá globálnu štruktúra, ale naopak stále má veľa lokálnych vlastností. Navyše v sebe zahŕňa aj fázu EC, aby sa čo

najviac využili tieto vlastnosti. Hlavnou myšlienkou je rozdeliť výstupné dáta BWT na niekoľko kontextovo menších častí. Tento proces delenia sa opakuje dovtedy, až kým každá časť neobsahuje iba jeden rovnaký symbol. V prípade použitia tejto metódy nasleduje hneď po BWT, aby bol zachovaný celý kontext jej výstupu, teda nie je možné použiť medzi nimi RLE.

Metóda postupne delí vstupný vstupné dáta na menšie sekvencie. V každom kroku je každá výsledná sekvencia predchádzajúceho delenia znova rozdelená na menšie časti. Počet získaných častí z delenia každej sekvencie je rovný počtu rôznych symbolov vo vnútri príslušného intervalu vstupných dát pre danú delenú časť. Veľkosť každej novej časti je daná počtom výskytov príslušného symbolu v danom intervale. Ak nová časť obsahuje len jeden rovnaký symbol, už sa ďalej nedelí.

### 2.3.9 Interval Encoding

Rýchla a jednoduchá metóda IE [11] pracuje so vzdialenosťami výskytov symbolov. Pre každý nový symbol zo vstupu je potrebné uchovať index jeho prvého výskytu a potom sú hľadané jeho ďalšie výskyty. Výstupom sú počty symbolov medzi aktuálne nájdeným a posledným výskytom daného symbolu, teda vzdialenosť medzi nimi. Výstupom je nula v prípade, že dosiahneme koniec vstupných dát. Na vstupnom reťazci „TROLOLO“ získame nasledujúce údaje:

| **Symbol** | **Prvý výskyt** | **Vzdialenosti** |
|---|---|---|
| T | 1 | 0 |
| R | 2 | 0 |
| O | 3 | 2, 2, 0 |
| L | 4 | 2, 0 |

**Tabuľka 2.14:** Kódovací proces

### 2.3.10 Distance Coding

Metóda DC [10] vznikla vylepšením algoritmu IE. Pre každý symbol vstupných dát je výstupom vzdialenosť k jeho ďalšiemu výskytu na vstupe. Pokiaľ sa už symbol nevyskytne do konca vstupných dát, výstupom je nula. Aj v tomto prípade musíme uchovávať index prvého výskytu každého symbolu. Vylepšením je, že vzdialenosť k ďalšiemu výskytu symbolu počítame namiesto počtu všetkých symbolov medzi aktuálnou a ďalšou pozíciou symbolu len ako počet zatiaľ neznámych (neanalyzovaných) symbolov. Ďalším vylepšením je, že výstup sa negeneruje v prípade, ak je aktuálny symbol rovnaký ako posledný, teda opakujúce sa symboly sú preskočené.

### 2.3.11 Run Length Encoding

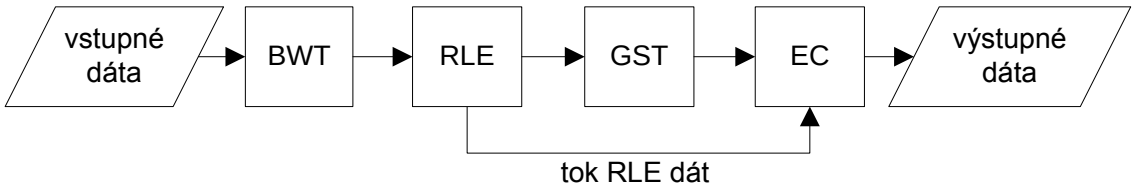
Treťou fázou je rýchly a jednoduchý algoritmus RLE [1], často používaný pri kompresii obrazu aj textu. Princípom je nahradzovanie sekvencií rovnakých symbolov na vstupe kratšími dátami, zvyčajne jedným alebo viacerými symbolmi a počtom opakovaní daného symbolu. V niektorých implementáciách sa pridáva aj špeciálny ukončovací symbol.

Aby bol dosiahnutý vyšší kompresný pomer, zavádza sa hodnota prahu kompresie. Udáva, od akého počtu rovnakých symbolov v slede sa má začať kódovať. Pre jeden symbol tvoriaci sled by výstup narástol dvojnásobne pridaním hodnoty opakovania, preto je nutné tento prípad vylúčiť. Zvyčajne sa volí hodnota 4 alebo 3. Pokiaľ je teda veľkosť vstupného sledu menšia ako zvolený prah, nedochádza ku kódovaniu a nahradzovaniu symbolov. Ak je však dĺžka sledu rovná alebo väčšia ako daný prah, začne kódovanie. V tomto prípade je potrebné odlíšiť zakódovanú výstupnú informáciu od zvyšku nezakódovaných dát. Prvou možnosťou je použiť sled daného symbolu o dĺžke rovnej nastavenému prahu. Ak pri dekódovaní narazíme na dlhý sled o danej dĺžke, vieme že nasleduje informácia o počte opakovaní symbolu a tým získame všetky potrebné informácie pre opätovné zostavenie pôvodného vstupu. Druhou možnosťou je použitie špeciálneho symbolu indikujúceho kódovú sekvenciu. Zvyčajne po ňom nasleduje daný symbol sledu a počet jeho opakovaní. Tento špeciálny symbol však nesmie byť súčasťou abecedy vstupných symbolov, aby sme ho nezamenili za obyčajný symbol, prípadne ho musíme špeciálne zakódovať. Pri informácii o počte opakovaní je taktiež viacero možností, ako ju na výstupe zakódovať. Zvyčajne nasleduje za úvodným sledom symbolov, respektíve za dvojicou špeciálneho znaku a symbolu.

Obmedzenie spočíva v samotných vstupných dátach. Ak použijeme ako vstup štandardný text, na sekvencie aspoň dvoch rovnakých znakov narazíme veľmi zriedkavo, pričom trojice a viacznakové sekvencie sú ešte vzácnejšie. Ďalším obmedzením je maximálna hodnota počtu opakovaní rovná 255, čo je maximálne veľké číslo uložiteľné na jeden bajt. V prípade dlhších sekvencií ich musíme rozdeliť na niekoľko kratších. Taktiež použitý špeciálny symbol na odlíšenie kódovaných sekvencií sa nesmie vyskytovať na vstupe. V prípade štandardného textu môžeme použiť napríklad niektoré ASCII riadiace znaky.

Vylepšení tejto metódy je hneď niekoľko. Zero Run Transformation (RLE0) kóduje iba sekvencie núl vzhľadom na to, že práve nula je symbol s najčastejšími sledmi. Sledy ostatných symbolov sa nekódujú. V niektorých prípadoch sa RLE fáza môže umiestniť pred GST, čo má za následok lepší kompresný pomer výstupu metódy GST. Druhou výhodou je zrýchlenie celého kompresného procesu, pretože RLE je rýchlejšia metóda ako GST a jej výstup je obecne menší, takže GST nemusí venovať toľko času jeho transformácii. V tomto prípade sú však výstupom RLE až dva toky dát. Prvý obsahujúci transformované vstupné dáta bez sledov

opakujúcich sa symbolov je napojený na vstup GST. Druhý obsahujúci informácie o dĺžkach jednotlivých sledov je vedľajším vstupom fázy EC. Tieto údaje preskakujú fázu GST, aby nenarúšali kontext symbolov spracovávaných dát.




**Obrázok 2.6:** Upravený kompresný postup

Kompresný faktor RLE je možné vypočítať podľa nasledujúceho vzorca:

$$kompresný\ faktor = \frac{N}{N - X \times (L - 3)} \tag{3.7}$$

N je počet symbolov vstupného reťazca, kde sa vyskytuje X sledov s priemernou dĺžkou Y. Metóda nahradzuje tieto opakovania dvojicou symbolov a číslom udávajúcim dĺžku opakovania alebo špeciálnym znakom spolu so symbolom a dĺžkou opakovania, takže spolu troma symbolmi.

#### 2.3.11.1 Alternatívy RLE

Iným vylepšením je metóda relatívneho kódovania [1]. Ak vstupné dáta tvoria postupnosti čísel, ktoré medzi sebou nemajú príliš veľké rozdiely, prípadne vzájomne podobné reťazce, je možné kódovať tieto špecifické prípady len ako rozdiely medzi nimi. Výstupom je teda prvé číslo nasledované rozdielmi medzi ním a nasledujúcimi číslami. Rozdiely sa môžu počítať medzi prvým a každým nasledujúcim číslom, alebo medzi aktuálnym a nasledujúcim. Ak by bol rozdiel príliš veľký, na výstup sa zašle celé číslo. Tieto čísla je nutné pre proces dekódovania rozlíšiť od ostatných výstupov udávajúcich rozdiely. Ku každému normálnemu číslu sa preto pridáva špeciálny príznakový bit, ktoré kompresor zhromažďuje a v istých momentoch ich posiela dekompresoru. Ak sa rozdiely posielajú vo forme jedného bajtu, kompresor pripája ku skupinám ôsmich výstupných bajtov jeden bajt obsahujúci príznakové bity. Iným riešením je odosielanie dvoch bajtov. Môžu obsahovať dva jedno bajtové rozdiely alebo jednu dvoj bajtovú hodnotu. Pre každý pár je vytvorený príznakový bit, ktoré sa akumulujú do dvoj bajtovej informácie odosielanej po každých šestnástich pároch bajtov.

Ďalšou metódou je kódovanie digramov. Použiteľná je len v špecifických prípadoch obmedzenej abecedy vstupných dát, napríklad v prípade textových dát (písmená, číslice,

interpunkčné znaky). V tomto prípade sa dopredu určené, často sa vyskytujúce dvojice symbolov dajú nahradiť jedným symbolom, ktorý sa na vstupe nikdy nevyskytne, napríklad určité ASCII riadiace znaky.

Na podobnom princípe funguje metóda substitúcie vzoriek. V tomto prípade sa nahradzujú celé slová určenými symbolmi. Použiteľné je to pri kompresii zdrojových kódoch programov, kde sa často vyskytujú vyhradené slová programovacieho jazyka (napríklad print, for, int, repeat).

### 2.3.12 Entropy Coding

EC je poslednou fázou celého Burrows-Wheelerovho kompresného algoritmu. Na výber máme niekoľko rôznych algoritmov, medzi najznámejšie patria aritmetické kodéry a rôzne modifikácie Huffmanovho kódovania. Huffmanovo kódovanie ponúka rýchlejšiu kompresiu, v prípade lepšieho kompresného pomeru zvolíme aritmetické kódovanie.

### 2.3.13 Aritmetické kódovanie

Aritmetické kódovanie [5] [1] funguje na princípe priradzovania výstupného kódu celým vstupným sekvenciám na rozdiel od kódovania jednotlivých symbolov. Kódové slovo je tvorené reálnym číslom z intervalu nula až jedna vrátane nuly. Každým zakódovaným symbolom dôjde k zúženiu tohto intervalu úmerne svojej pravdepodobnosti výskytu. Častejšie sa vyskytujúce symboly sú zakódované menším počtom výstupných bitov, týmto dochádza ku kompresii. Početnejšie symboly zužujú interval menej ako symboly s malou frekvenciou výskytu, teda nové medze intervalu sú reprezentované menším počtom bitov. Vzhľadom na použitie reálnych čísiel sa v intervale nachádza nekonečný počet unikátnych hodnôt, takto je možné priradzovať každej vstupnej postupnosti úplne odlišnú výstupnú hodnotu.

Túto metódu je vhodné použiť v prípade, keď sa niektoré symboly vyskytujú mnohonásobne viac ako iné, teda dochádza ku skoseniu pravdepodobností výskytov jednotlivých symbolov. Pomocou tejto metódy je možné znížiť zbytočnú redundanciu a priblížiť sa tak viac k skutočnej entropii kódovaných dát. Huffmanovo kódovanie dosahuje lepších výsledkov len v prípade, ak pravdepodobnosti výskytov jednotlivých symbolov budú rovné hodnotám záporných mocnín čísla dva. Vtedy výsledky dosahujú hodnôt skutočnej entropie daných dát.

V základnej konfigurácii je algoritmus dvojpriechodový, čo ho radí medzi semiadaptívne metódy. Pre potreby druhého hlavného prechodu je nutné zistiť množstvo výskytov jednotlivých symbolov vo vstupných dátach a to je úlohou prvého prechodu. Z týchto údajov sa vytvorí pravdepodobnostný model pre postup kódovania v druhej fáze. Tento model

sa musí pre potreby dekódovania vzhľadom na jeho semiadaptívnu vlastnosť distribuovať spolu s výstupnými dátami. Základný koncept algoritmu funguje na nasledujúcom princípe:

1. V prvom kroku je definovaný počiatočný interval <0; 1).
2. Následne pre každý symbol abecedy vstupných dát rozdelíme aktuálny interval na podintervaly, ktorých veľkosť sa rovná pravdepodobnostiam výskytov jednotlivých symbolov. Vyberieme podinterval zodpovedajúci aktuálnemu symbolu vstupných dát a určíme ho ako aktuálny interval. Posunieme sa na ďalší symbol vstupu.
3. Tento postup opakujeme až do konca vstupných dát. Výstupom je číslo z aktuálneho intervalu, ktoré ho jednoznačne identifikuje. Vzhľadom na použitie začiatočného intervalu <0; 1) sa nemusíme starať o celé číslo a stačí použiť iba desatinnú časť.

Pri vstupnom reťazci „TROOLOLO“ získame pravdepodobnostný model s nasledujúcimi intervalmi, ktorý bude pre potreby jeho znovu zostavenia pri procese dekódovania zapísaný do výstupných dát:

| **Symbol** | **Frekvencia** | **Pravdepodobnosť** | **Interval** |
|---|---|---|---|
| O | 4 | 4 / 8 = 0,5 | <0; 0,5) |
| L | 2 | 2 / 8 = 0,25 | <0,5; 0,75) |
| T | 1 | 1 / 8 = 0,125 | <0,75; 0,875) |
| R | 1 | 1 / 8 = 0,125 | <0,875; 1) |

**Tabuľka 2.15:** Získaný pravdepodobnostný model

V prvom prechode bol teda zhotovený pravdepodobnostný model. Teraz môžeme začať vykonávať druhú fázu, samotný kódovací proces. Na uchovávanie dolnej a hornej hranice aktuálneho intervalu využijeme premenné limit_a, respektíve limit_b. Inicializujeme ich na prvotné hodnoty nula a jedna, pričom ďalšie hodnoty sa budú počítať v krokoch spracovávania vstupných symbolov podľa nasledujúcich vzorcov:

$$limit\_a_n = limit\_a_{n-1} + (limit\_b_{n-1} - limit\_a_{n-1}) \times range\_a(X) \quad (3.8)$$

$$limit\_b_n = limit\_a_{n-1} + (limit\_b_{n-1} - limit\_a_{n-1}) \times range\_b(X) \quad (3.9)$$

Pri vstupnom reťazci „TROOLOLO“ budú nadobúdať pre jednotlivé symboly a ich podintervalov nasledujúce hodnoty:

| Symbol | Premenná | Výpočet hodnoty |
|---|---|---|
| T | limit_a | 0 + (1 - 0) * 0,75 = 0,75 |
| | limit_b | 0 + (1 - 0) * 0,875 = 0,875 |
| R | limit_a | 0,75 + (0,875 - 0,75) * 0,875 = 0,859375 |
| | limit_b | 0,75 + (0,875 - 0,75) * 1 = 0,875 |
| O | limit_a | 0,859375 + (0,875 - 0,859375) * 0 = 0,859375 |
| | limit_b | 0,859375 + (0,875 - 0,859375) * 0,5 = 0,8671875 |
| O | limit_a | 0,859375 + (0,8671875 - 0,859375) * 0 = 0,859375 |
| | limit_b | 0,859375 + (0,8671875 - 0,859375) * 0,5 = 0,86328125 |
| L | limit_a | 0,859375 + (0,86328125 - 0,859375) * 0,5 = 0,861328125 |
| | limit_b | 0,859375 + (0,86328125 - 0,859375) * 0,75 = 0,8623046875 |
| O | limit_a | 0,861328125 + (0,8623046875 - 0,861328125) * 0 = 0,861328125 |
| | limit_b | 0,861328125 + (0,8623046875 - 0,861328125) * 0,5 = 0,86181640625 |
| L | limit_a | 0,861328125 + (0,86181640625 - 0,861328125) * 0,5 = = 0,861572265625 |
| | limit_b | 0,861328125 + (0,86181640625 - 0,861328125) * 0,75 = = 0,8616943359375 |
| O | limit_a | 0,861572265625 + (0,8616943359375 - 0,861572265625) * 0 = = 0,861572265625 |
| | limit_b | 0,861572265625 + (0,8616943359375 - 0,861572265625) * 0,5 = = 0,86163330078125 |

**Tabuľka 2.16:** Proces kódovania

S každým ďalším spracovaným vstupným symbolom dochádza k rekurzívnemu deleniu aktuálneho intervalu na nové podintervaly. Z tohto dôvodu dochádza k zmenšovaniu šírky intervalu a na uchovávanie a reprezentáciu hraničných hodnôt sú potrebné čísla s vyššou presnosťou.

Na zakódovanie vstupného reťazca „TROOLOLO“ môžeme použiť ľubovoľnú hodnotu z výsledného intervalu <0,861572265625; 0,86163330078125). Z praktických dôvodov pri dekódovaní sa použije dolná hranica intervalu 0,861572265625. V binárnej reprezentácii má hodnotu 0,110111001001. Výsledné hraničné hodnoty sa budú pohybovať vždy v rozmedzí začiatočného intervalu <0, 1), preto nikdy nedosiahnu hodnotu jedna a celočíselnú časť výsledku (ktorá bude vždy nula) nemusíme kódovať. Výsledkom kódovania je po tejto úprave hodnota 110111001001. Vzhľadom na pomalosť výpočtov aritmetiky s pohyblivou desatinnou

čiarkou sa v skutočným implementáciách používa upravený postup s rýchlejšou celočíselnou aritmetikou.

Pri procese dekódovania je nutné najprv načítať zo vstupu príslušný pravdepodobnostný model tvorený symbolmi abecedy a ich príslušnými počtami výskytov. Z týchto údajov sme schopní znova zostaviť celý pravdepodobnostný model. V prvom kroku sa znova inicializujú hodnoty hraníc počiatočného intervalu na nula a jedna. Aj v tomto procese bude dochádzať k postupnému deleniu aktuálneho intervalu na jeho podintervaly.

Z predchádzajúceho príkladu bola výstupom kódovania hodnota 110111001001. Tú po pridaní celočíselnej časti rovnej hodnote nula prevedieme späť na pôvodnú dekadickú reprezentáciu 0,861572265625. Pre dekódovanie prvého symbolu je potrebné nájsť v pravdepodobnostnom modeli príslušný podinterval, v ktorého hraniciach leží táto hodnota. Aby sa eliminoval vplyv aktuálne dekódovaného symbolu na kódovaný vstup, z hodnoty sa odpočíta dolná hranica nájdeného intervalu a následne sa vydelí rozsahom (šírkou) tohto intervalu. Postup opakujeme až do dekódovania všetkých vstupných dát. Na konci procesu získame hodnotu rovná nule.

$$value = \frac{value - range\_a(X)}{range} \tag{3.10}$$

Aby sme zabránili neustálemu vytváraniu symbolu „O“ na konci dekódovania, musí dekodér dostať informáciu o konci procesu prostredníctvom špeciálneho symbolu. Tento symbol sa však nesmie vyskytovať v abecede symbolov pôvodného vstupu kodéru. Inou možnosťou je dodanie informácie o dĺžke dekódovaných dát..

| **Symbol** | **Interval** | **Rozsah** |
|---|---|---|
| O | <0; 0,5) | 0,5 |
| L | <0,5; 0,75) | 0,25 |
| T | <0,75; 0,875) | 0,125 |
| R | <0,875; 1) | 0,125 |

**Tabuľka 2.17:** Obnovený pravdepodobnostný model

| Hodnota | Interval | Symbol | Nová hodnota |
|---|---|---|---|
| 0,861572265625 | <0,75; 0,875) | T | (0,861572265625 - 0,75) / 0,125 = = 0,892578125 |
| 0,892578125 | <0,875; 1) | R | (0,892578125 - 0,875) / 0,125 = 0,140625 |
| 0,140625 | <0; 0,5) | O | (0,140625 - 0) / 0,5 = 0,28125 |
| 0,28125 | <0; 0,5) | O | (0,28125 - 0) / 0,5 = 0,5625 |
| 0,5625 | <0,5; 0,75) | L | (0,5625 - 0,5) / 0,25 = 0,25 |
| 0,25 | <0; 0,5) | O | (0,25 - 0) / 0,5 = 0,5 |
| 0,5 | <0,5; 0,75) | L | (0,5 - 0,5) / 0,25 = 0 |
| 0 | <0; 0,5) | O | (0 - 0) / 0,5 = 0 |

**Tabuľka 2.18:** Proces dekódovania

Tento princíp aritmetického kódovania pomocou aritmetiky plávajúcej desatinnej čiarky je z pohľadu možností dnešnej výpočtovej techniky nepraktický. Problémom je nielen ukladanie desatinných čísel s nekonečnou presnosťou, ale aj vysoká časová náročnosť, najmä pri delení veľkých desatinných čísiel. Riešením je upravený postup s použitím celočíselnej aritmetiky. Presnosť kódovania je aj v tomto prípade obmedzená, a to počtom bitov využívaných pri ukladaní celočíselných hodnôt. Zvyčajne sa stretávame s 32-bitovými systémami.

Pravdepodobnosti výskytov symbolov sa už nepoužívajú na výpočet rozsahov intervalov, namiesto toho sú zavedené kumulované frekvencie. Tie sú v tomto prípade reprezentované celými číslami. Kumulovaná frekvencia daného symboly je rovná súčtu frekvencií ostatných symbolov nachádzajúcich sa pred ním v danom pravdepodobnostnom modeli. Pre vstupný reťazec „TROOLOLO“ by pravdepodobnostný model vyzeral nasledovne:

| Symbol | Frekvencia | Kumulované frekvencie |
|---|---|---|
| O | 4 | 0 |
| L | 2 | 4 |
| T | 1 | 6 |
| R | 1 | 7 |
| **Celková kumulovaná frekvencia** | | 8 |

**Tabuľka 2.19:** Pravdepodobnostný model pri použití celočíselnej aritmetiky

Hranice intervalu sú aj v tomto prípade vyjadrené hodnotami limit_a a limit_b, pribudne však tretia hodnota limit_c udávajúca celkovú kumulovanú frekvenciu, teda súčet všetkých výskytov symbolov vstupnej abecedy. Hodnota limit_a udáva súčet výskytov symbolov umiestnených v pravdepodobnostnom modeli pred daným symbolom. Hodnota limit_b vznikne súčtom čísla limit_a a počtom výskytov aktuálneho symbolu.

Pravdepodobnostný model je reprezentovaný jednorozmerným poľom, ktorého veľkosť závisí od počtu symbolov vstupnej abecedy. Jedna pozícia navyše je vyhradená pre špeciálny ukončovací symbol. Pre obecnú počítačovú reprezentáciu vstupných dát bude mať abeceda veľkosť 256 symbolov, teda rozsah indexu poľa modelu bude po pridaní špeciálneho symbolu od nuly až po 256.

Prvým krokom algoritmu je inicializácia hodnôt hraníc intervalu. Dolná hranica je nastavená na štandardnú nulu, problém však nastáva pri hornej hranici. Hodnoty hraníc sa musia reprezentovať ako desatinné zlomky z intervalu <0,1), preto nie je možné použiť pre inicializáciu hornej hranice číslo jedna. Namiesto toho použijeme polovicu maximálnej možnej hodnoty premennej zaokrúhlenú smerom dole. V každom kroku spracovania vstupného symbolu sa interval neustále zmenšuje, čo by nakoniec viedlo vďaka obmedzeným možnostiam rozsahu celočíselných hodnôt k nesprávnej reprezentácii malých čísiel a výpočet by prestal fungovať. Preto sa interval v presne daných momentoch zväčšuje. Informácie o týchto zmenách sú na výstupe reprezentované binárnou hodnotou.

Pre tieto potreby sú zavedené štyri konštanty udávajúce maximálne rozsahy odvodených intervalov. Maximálny rozsah označíme ako INT_1. V praktickej implementácii je rovný polovici maximálnej hodnoty premennej. Štvrtinu maximálneho rozsahu reprezentuje konštanta INT_0_25:

$$INT\_0\_25 = 1 + \frac{INT\_1}{4} \tag{3.11}$$

Polovicu maximálneho rozsahu reprezentuje konštanta INT_0_5:

$$INT\_0\_5 = 2 \times INT\_0\_25 \tag{3.12}$$

Tri štvrtiny maximálneho rozsahu reprezentuje konštanta INT_0_75:

$$INT\_0\_75 = 3 \times INT\_0\_25 \tag{3.13}$$

Hranice intervalu sa menia iba v prípadoch spĺňajúcich určené podmienky. Ak sa interval nachádza v dolnej polovici maximálneho rozsahu, teda v intervale <0; INT_0_5), hodnoty jeho hraníc sú zdvojnásobené a k hornej je navyše pripočítaná jednotka. Pokiaľ sa interval nachádza naopak v hornej polovici maximálneho rozsahu, teda v intervale <INT_0_5; INT_1), tak hodnoty sú pred zdvojnásobením znížené o hodnotu INT_0_5. K hodnote hornej hranice je pripočítaná dodatočne jednotka. Posledným prípadom je umiestnenie intervalu medzi prvou a treťou štvrtinou maximálneho rozsahu, teda v intervale <INT_0_25; INT_0_75). Hodnoty hraníc sú v tomto prípade znížené o konštantu INT_0_25, následne zdvojnásobené a k hornej hranici je znova pripočítaná jednotka.

Po vytvorení pravdepodobnostného modelu a inicializácii hraníc sa môže začať samotný proces kódovania. V každom kroku je načítaný vstupný symbol a hranice sú aktualizované podľa nasledujúcich vzorcov:

$$limit\_a_n = limit\_a_{n-1} + (1 + limit\_b_{n-1} - limit\_a_{n-1}) \times \frac{cum\_high\_freq(X)}{limit\_c} \quad (3.14)$$

$$limit\_b_n = limit\_a_{n-1} + (1 + limit\_b_{n-1} - limit\_a_{n-1}) \times \frac{cum\_low\_freq(X)}{limit\_c} \quad (3.15)$$

Následne sa skontroluje, či nové hodnoty hraníc intervalu spĺňajú podmienky pre preškálovanie. Ak nastane prvý prípad, hodnoty sa upravia popísaným spôsobom a výstupom bude nula nasledovaná určitým počtom jednotiek. Ten je rovný počtu výskytov tretieho prípadu preškálovania. Tento počet následne vynulujeme. V prípade výskytu intervalu v hornej polovici maximálneho rozsahu sa tiež príslušne upravia hranice intervalu. Výstupom je však jednotka nasledovaná počtom núl rovných počtu výskytov tretieho prípadu preškálovania, ktorý potom vynulujeme. Ak je splnená podmienka pre tretí prípad, dôjde k úprave hodnôt hraníc intervalu a zvýši sa evidovaný počet výskytov tohto preškálovania. Tento krok preškálovania intervalu sa opakujem, až kým nové hodnoty hraníc nespĺňajú podmienky ani jedného z uvedených troch prípadov.

Celý proces kódovania je tvorený načítavaním vstupného symbolu, určením nového intervalu a jeho prípadným preškálovaním. Ukončí sa po vyčerpaní a spracovaní všetkých vstupných symbolov.

Pri dekódovaní musíme byť schopní rozoznať koniec vstupných dát. Jednou možnosťou je uloženie počtu zakódovaných symbolov, v tomto prípade však môže dôjsť k pretečeniu tejto hodnoty pri príliš veľkých vstupných dátach s veľkým množstvom symbolov abecedy. V praxi sa používa druhý spôsob, kde používame špeciálny symbol označujúci koniec vstupných dát.

Ten je pri procese kódovania zakódovaný spolu s ostatnými vstupnými dátami a pri spätnom procese dekódovania je algoritmus ukončený v momente detekcie tohto symbolu.

V prvom kroku je znova vytvorený pravdepodobnostný model zhodný s modelom použitým pri kódovaní. Hranice intervalu sú inicializované na rovnaké hodnoty ako pri kódovaní. Zo vstupu je načítaný počet bitov rovný počtu bitov reprezentujúcich hodnoty hraníc intervalu. Z týchto načítaných dát je vypočítaná pomocná hodnota:

$$tmp = \frac{(value - limit_a + 1) \times limit\_c - 1}{limit\_b - limit\_a + 1} \tag{3.16}$$

Po jej úprave k najbližšiemu celému číslu je porovnaná s kumulovanými frekvenciami symbolov v pravdepodobnostnom modeli. Výstupom je symbol, ktorého interval obsahuje túto hodnotu. Hodnoty hraníc sú aktualizované podľa rovnakých vzorcov ako pri kódovaní. Proces sa opakuje až do opätovného zostavenia pôvodných vstupných dát.

V každom kroku musíme kontrolovať hodnoty limit_a a limit_b. Pokiaľ nové hranice intervalu spĺňajú podmienky preškálovania ako pri postupe kódovania, musia byť upravené. V prvom prípade sú hodnoty upravené podľa popísaného postupu a aktuálna hodnota načítaná zo vstupu je zdvojnásobená a zvýšená o ďalší vstupný bit. Pri splnení druhej podmienky je načítaná hodnota pred zdvojnásobením najskôr znížená o konštantu INT_0_5. Nakoniec je zvýšená o ďalší vstupný bit. Pokiaľ sa interval nachádza medzi hodnotami INT_0_25 vrátane a INT_0_75, je od načítanej hodnoty odpočítaná konštanta INT_0_25. Následne je zdvojnásobená a zvýšená o ďalší vstupný bit.

#### 2.3.13.1 Adaptívne aritmetické kódovanie

Nevýhodou aritmetického kódovania je jeho semiadaptívnosť, teda nutnosť prvého prechodu vstupnými dátami pre vytvorenie pravdepodobnostného modelu a následný druhý prechod pre samotné kódovanie. Prvá fáza sa dá nahradiť inými spôsobmi výpočtu pravdepodobnostného modelu, no tie zvyčajne vedú k zhoršeniu výsledného kompresného pomeru.

Riešením je adaptívne aritmetické kódovanie [1]. Pravdepodobnostný model sa vytvára pri samotnom procese kódovania a je tvorený zoznamom symbolov, ich frekvenciou výskytu a kumulovanými frekvenciami. Zoznam je udržovaný zoradený podľa frekvencií výskytov. Pri procese kódovania vstupného symbolu sa evidovaná početnosť výskytu daného symbolu zvýši o jedna. Následne sa nová hodnota správne zatriedi, aby bola dodržaná podmienka zoradenia zoznamu. Nakoniec dôjde k aktualizácii kumulovaných frekvencií. Algoritmus dekódovania pracuje na rovnakom princípe.

Pravdepodobnostný model sa inicializuje priradením jedného výskytu všetkým symbolom vstupnej abecedy. Tento krok je nutný pre proces správneho zakódovania symbolu, ktorý vyžaduje pravdepodobnosť aspoň jedného výskytu daného symbolu. Problém pretečenia sa rieši kontrolou a prípadným následným delením všetkých frekvencií výskytov symbolov číslom dva. Po tomto kroku je nutné príslušne upraviť aj kumulované frekvencie.

### 2.3.14 Huffmanovo kódovanie

Huffmanovo kódovanie [1] je jedna z najstarších kompresných metód. Princíp je založený na vytváraní prefixového kódu. Častejšie sa vyskytnúcim symbolom vstupných dát sa priradzujú kratšie výstupné kódy ako zriedkavejším symbolom. Ako pravdepodobnostný model vstupnej abecedy môžeme použiť niektorý statický, už dopredu pripravený model, alebo ho vytvoriť prechodom vstupnými dátami (semiadaptívny prístup). Tento model musí byť zoradený v zostupnom poradí. Metóda dosahuje najlepšie výsledky v prípade, ak sa pravdepodobnosti výskytov symbolov vstupnej abecedy rovnajú záporným mocninám čísla dva, preto sa v praktických implementáciách takmer vôbec nepoužíva.

Po získaní pravdepodobnostného modelu je potrebné vytvoriť Huffmanov strom. Pri jeho konštrukcii sa postupuje od symbolov s najnižšou pravdepodobnosťou výskytu. Listy obsahujúce tieto symboly sú spájané v jednotlivých krokoch do vetiev, pričom súčet ich frekvencií vytvára nový nadradený list s novým zástupným symbolom. Hodnota tohto dočasného symbolu nie je podstatná, respektíve nemusí vôbec existovať, dôležitá je iba získaná frekvencia. Takto spracované dva symboly sú z modelu odstránené a nahradené týmto zástupným symbolom, ktorý je spolu s novou frekvenciou výskytu správne zatriedený. Tento proces sa opakuje, až kým v pravdepodobnostnom modeli nezostane jediný zástupný symbol, ktorý tvorí koreň celého získaného Huffmanovho stromu. V poslednom kroku pridelíme hranám stromu hodnoty jedna a nula, ktoré budú kódovať pôvodné symboly abecedy.

Na vstupnom reťazci „TROOLOLO“ by sme získali nasledujúce údaje:

| Symbol | Frekvencia | Pravdepodobnosť |
|---|---|---|
| O | 4 | 0,5 |
| L | 2 | 0,25 |
| T | 1 | 0,125 |
| R | 1 | 0,125 |

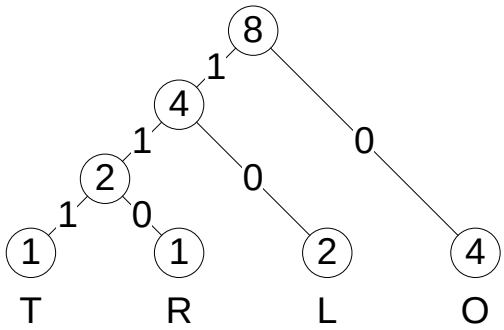
**Tabuľka 2.20:** Pravdepodobnostný model pred vytváraním Huffmanovho stromu

Po prvom kroku vytvárania Huffmanovho stromu by sa model upravil a získali by sme prvú časť stromu:

| Symbol | Frekvencia | Pravdepodobnosť |
|---|---|---|
| O | 4 | 0,5 |
| L | 2 | 0,25 |
| TR | 2 | 0,25 |

**Tabuľka 2.21:** Pravdepodobnostný model po prvom kroku

Popísaným algoritmom spracujeme celý pravdepodobnostný model. Z výsledného stromu odvodíme kódovacie sekvencie pre jednotlivé symboly:




**Obrázok 2.7:** Výsledný Huffmanov strom

| Symbol | Kód |
|---|---|
| O | 0 |
| L | 10 |
| T | 111 |
| R | 110 |

**Tabuľka 2.22:** Výsledné kódové sekvencie

Symboly s vyššou pravdepodobnosťou výskytu majú priradené kratšie kódy. Vstupný reťazec „TROOLOLO“ zakódujeme pomocou výstupnej sekvencie 11111000100100. Takto získaný kódovací model je možné použiť aj pre iné vstupné dáta pozostávajúce iba zo symbolov „L“, „O“, „R“ a „T“, najlepšie výsledky však bude podávať iba v prípade, ak polovicu symbolov bude tvoriť „O“, štvrtinu „L“ a po jednej osmine „R“ a „T“. Na tomto princípe sú

založené statické modely napríklad pre textové dáta, odlišujú sa rôznymi frekvenciami výskytov symbolov abecied národných jazykov.

Výsledný kódovací model musí byť v prípade semiadaptívneho prístupu pre potreby dekódovania priložený k výstupným dátam, napríklad vo forme pravdepodobností výskytov symbolov. Z týchto údajov je možné spätne vytvoriť Huffmanov strom. Pri procese dekódovania sa načítavajú jednotlivé bity a dekodér sa na základe nich pohybuje v strome od koreňa smerom k listom. Ak narazí na koncový list, výstupom je príslušný symbol a proces sa opakuje znova od koreňa stromu.

#### 2.3.14.1 Vylepšenia Huffmanovho kódovania

Kanonické Huffmanove kódy [1] sú vyberané z viacero modelov Huffmanových kódov. Podmienkou je rýchle a jednoduché použitie. V prípade veľkej abecedy dosahujú výborné časy pri procese dekódovania, rýchlosť kódovania nie je až tak dôležitá. Reálne použitie je vhodné napríklad pri zbierkach dokumentov, ktoré sú komprimované adaptívnym Huffmanovým kódovaním pracujúcim so slovami namiesto znakov.

Adaptívne Huffmanovo kódovanie [1] rieši problém dvojnásobného prechodu pri semiadaptívnom prístupe aj neefektívnosť statických pravdepodobnostných modelov. Huffmanov strom je inicializovaný ako prázdny strom bez symbolov s pridelenými kódmi. Prvý vstupný symbol je v nezakódovanej podobne skopírovaný na výstup a zapísaný do Huffmanovho stromu. Pri ďalšom výskyte tohto symbolu na vstupe je už výstupom jeho príslušná kódová sekvencia a jeho frekvencia výskytu je zvýšená. Po tejto inrementácii sa overí konzistentnosť Huffmanovho stromu. V prípade potreby sa strom preusporiada, čím vzniknú nové kódové sekvencie symbolov.

Problémom je možnosť pretečenia hodnôt frekvencií výskytov jednotlivých symbolov. Riešenie ponúka preškálovanie celého stromu, ak hodnota frekvencie koreňa dosiahne maximálnu hodnotu. Prakticky je to riešené podelením hodnôt listov číslom dva a následnou postupnou aktualizáciou ich nadradených uzlov. Toto delenie však znižuje presnosť a výsledný strom nemusí splňovať podmienky Huffmanovho stromu, preto sa musí upraviť.

Veľký počet symbolov vstupnej abecedy môže vytvoriť vysoký Huffmanov strom, čo má za následok problém preplnenia kódu. V tomto prípade sa pri prechode kodéra od koreňa stromu až k príslušnému listu akumulujú jednotlivé bity výstupného kódu. Ich počet však môže prekročiť veľkosť použitej premennej a dôjde k neželanému pretečeniu. Výsledkom je nesprávny kód. Ak nechceme obmedziť algoritmus udaním maximálnej veľkosti výšky stromu, tak ako riešenie môžeme použiť lineárne zviazaný zoznam premenných, nevýhodou je však pomalší algoritmus.

Dekodér používa rovnaký postup ako kodér. Nekódované symboly sú od zakódovaných odlíšené úvodným špeciálnym kódom zmeny s premenlivou dĺžkou. Výstupom nezakódovanej reprezentácie vstupného symbolu je tento symbol, pričom je pridaný do Huffmanovho stromu. Kódovaná vstupná sekvencia je vyhľadaná v strome a výstupom je príslušný symbol. Následne je Huffmanov strom upravený rovnako ako pri kódovacom procese.

# 3 Implementácia a testovanie

Pre potreby testovania a overenia teoretických podkladov bola implementovaná jednoduchá konzolová aplikácia FCOMP v jazyku C. Program komprimuje vstupné dáta pomocou Burrows-Wheelerovej komprimačnej metódy s nastaviteľnou veľkosťou spracovávaného bloku dát. Samozrejmosťou je možnosť spätnej dekomprimácie. Ako variant entropického kodéra bolo zvolené aritmetické kódovanie. Pri fáze GST je na výber päť metód: Incremental Frequency Count, Weighted Frequency Count, Move To Front spolu s jej dvoma úpravami MTF1 a MTF2. Každá metóda je implementovaná v samostatnom module, čo prináša možnosť opätovného použitia kódu.

## 3.1 Parametre kompresie

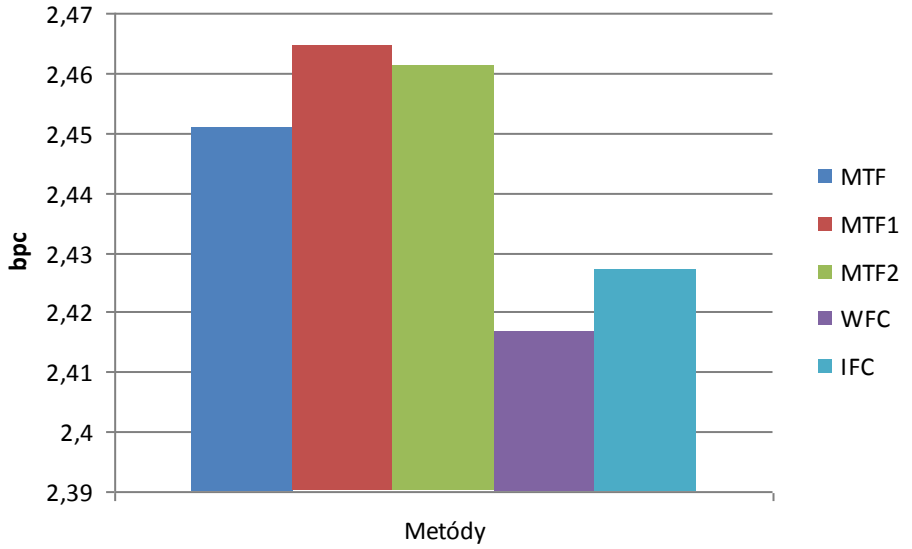
Jeden z hlavných faktorov ovplyvňujúcich výsledný kompresný pomer je zvolená veľkosť spracovávaného bloku dát. Ako vstupné dáta bol zvolený Silesia korpus, keďže z uvedených korpusov sa najviac približuje v súčasnosti používaným typom dát. Väčšie bloky by mali viesť k lepším výsledkom vzhľadom na zvyšujúcu sa pravdepodobnosť vytvorenia väčšieho zhluku rovnakých symbolov. Tento teoretický fakt bol na získaných výsledkoch prakticky potvrdený.

Nevýhodou je zvyšujúca sa časová náročnosť procesu. Kompresný pomer sa pri vyšších hodnotách veľkosti bloku výrazne nezlepšuje. Za povšimnutie stojí aj fakt, že ak nastavená hodnota prekročí veľkosť spracovávaného súboru, tak jej ďalším zvyšovaním už nezískame lepšie kompresné výsledky.

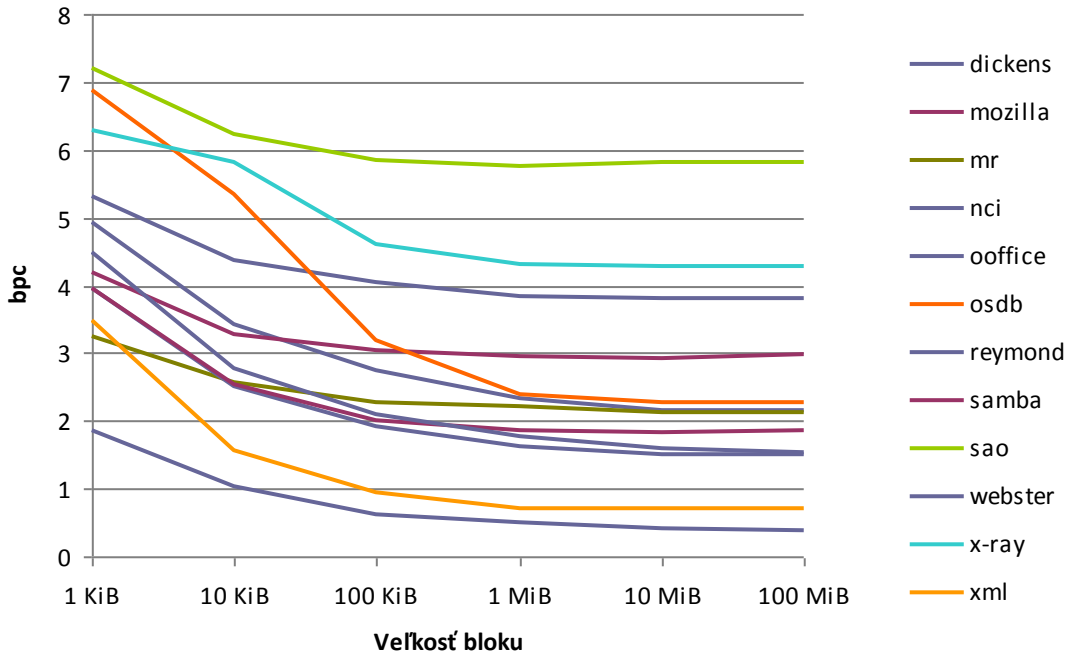
Pamäťová náročnosť programu je úmerná zvolenej veľkosti bloku, závisí tiež od použitej metódy GST. Pri vstupnom súbore menšom ako veľkosť bloku nedochádzalo k zbytočnému nárastu využitia pamäte. Z týchto pozorovaní vyplýva, že skutočné využitie pamäte sa odvíja od reálnej veľkosti bloku namiesto od nastavenej.

Časové výsledky tohto implementovaného riešenia nie je braná do úvahy vzhľadom k limitáciám použitého programovacieho jazyka. Pri testovaní čas kompresie narastal spolu s používanou veľkosťou bloku, najviac sa to odrazilo na spracovávaní najväčšieho súboru korpusu „mozilla“. Túto stratu výkonu možno pripísať algoritmom s neustálymi vyhľadávaniami v poli údajov, prípadne ich triedenie. Metóda Move-To-Front a jej variácie dosahovali pri komprimácii lepšie časové výsledky ako zvyšné metódy. Proces dekompresie bol pri všetkých metódach rýchlejší.

Z uvedených výsledkov vyplýva, že najlepšie kompresné pomery pri porovnávaní celkovej veľkosti skomprimovaného korpusu dosahuje metóda WFC, a to pri všetkých testovaných veľkostiach bloku.. Túto metódu možno označiť za najlepšiu, ak pri kompresii nezáleží na jej čase. Naopak v prípade potreby rýchlej kompresie je vhodné zvoliť MTF pri veľkosti bloku do 1 MiB, pri vyšších hodnotách dosahovali variácie MTF1 a MTF2 lepšie kompresné pomery.




**Graf 3.1:** Porovnanie výkonu jednotlivých metód, celý Silesia korpus, veľkosť bloku 100 KiB




**Graf 3.2:** Porovnanie kompresie jednotlivých súborov metódou IFC

## 3.2 Nastavenie WFC

Pri metóde WFC je možné cez rozhranie nastaviť váhové hodnotenia a ich rozsah. Jeden parameter je však konštantný, a to veľkosť posuvného okna. Po každej zmene tejto hodnoty je nutné program znova zostaviť z dôvodu limitácií jazyka C. Pri experimentovaní s týmto parametrom a testovaní na Calgary korpuse boli namerané nasledujúce hodnoty:

| | *SIZE = 512* | *SIZE = 1024* | *SIZE = 2048* | *SIZE = 4096* |
|---|---|---|---|---|
| *bib* | 47695 | 47301 | 43402 | 43402 |
| *book1* | 383422 | 383001 | 350919 | 350919 |
| *book2* | 264141 | 268724 | 250601 | 250601 |
| *geo* | 73165 | 70867 | 65225 | 65225 |
| *news* | 183311 | 183382 | 173031 | 173031 |
| *obj1* | 12573 | 11103 | 10544 | 10544 |
| *obj2* | 112765 | 99075 | 91033 | 91033 |
| *paper1* | 23705 | 23526 | 21893 | 21893 |
| *paper2* | 37538 | 37304 | 33979 | 33979 |
| *pic* | 59268 | 57092 | 55704 | 55704 |
| *progc* | 17726 | 16512 | 15226 | 15226 |
| *progl* | 24026 | 22127 | 20387 | 20387 |
| *progp* | 17375 | 15347 | 14266 | 14266 |
| *trans* | 32966 | 31646 | 29710 | 29710 |
| ***KORPUS*** | **1289676** | **1267007** | **1175920** | **1175920** |

**Tabuľka 3.1:** Vplyv veľkosti okna na kompresiu metódou WFC pri veľkosti bloku 10 KiB

| | *SIZE = 512* | *SIZE = 1024* | *SIZE = 2048* | *SIZE = 4096* |
|---|---|---|---|---|
| *bib* | 46493 | 47524 | 33611 | 33611 |
| *book1* | 449786 | 388430 | 299873 | 299873 |
| *book2* | 245643 | 244187 | 205026 | 205026 |
| *geo* | 73114 | 72547 | 62993 | 62993 |
| *news* | 182117 | 183498 | 147153 | 147153 |
| *obj1* | 12711 | 11810 | 10577 | 10577 |
| *obj2* | 120524 | 111061 | 82934 | 82934 |
| *paper1* | 20880 | 20708 | 17872 | 17872 |
| *paper2* | 33697 | 33027 | 28354 | 28354 |
| *pic* | 67257 | 60399 | 52911 | 52911 |
| *progc* | 19240 | 17228 | 12979 | 12979 |
| *progl* | 28344 | 23923 | 17573 | 17573 |
| *progp* | 19446 | 17404 | 11382 | 11382 |
| *trans* | 30769 | 25449 | 22078 | 22078 |
| ***KORPUS*** | **1350021** | **1257195** | **1005316** | **1005316** |

**Tabuľka 3.2:** Vplyv veľkosti okna na kompresiu metódou WFC pri veľkosti bloku 50 KiB

Z výsledkov vyplýva, že zmenšením veľkosti posuvného okna dôjde k zhoršeniu výsledného kompresného pomeru. Naopak pri dvojnásobnom zvýšení z pôvodnej hodnoty 2048 na novú 4096 nedošlo k žiadnemu zlepšeniu výsledného pomeru ani pri jednej veľkosti okna. Zmenšenie okna malo pri menšom vstupnom bloku 10 KiB menší vplyv na kompresiu ako rovnaké zmenšenie pri vstupnom bloku 50 KiB.

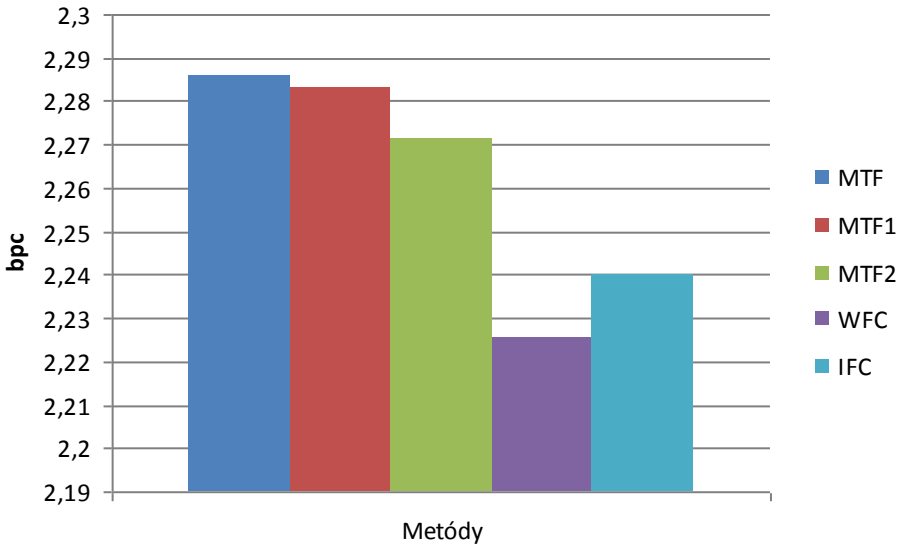
## 3.3 Nastavenie IFC

Táto metóda je nastaviteľná vo väčšom rozsahu. Hodnota ifc_size má väčší vplyv na výsledný kompresný pomer netextových vstupných údajov, ak sa zároveň ostatné parametre nemenia. Hodnota ifc_limit rovnako ako ifc_size ovplyvňuje najviac netextové vstupné údaje, Jedine nastavenie ifc_scale vplýva vysokou mierou na kompresiu textu, ifc_diff nemá takmer žiadny vplyv. Samozrejme nastavovanie parametrov jednotlivo nemá takmer žiaden význam, je nutné zvoliť zmenu viacerých alebo všetkých hodnôt naraz. Kombinácii je obrovské množstvo, voľba tých správnych je predmetom množstva diskusií a publikácií. Pre potreby testovania boli zvolené nasledujúce hodnoty:

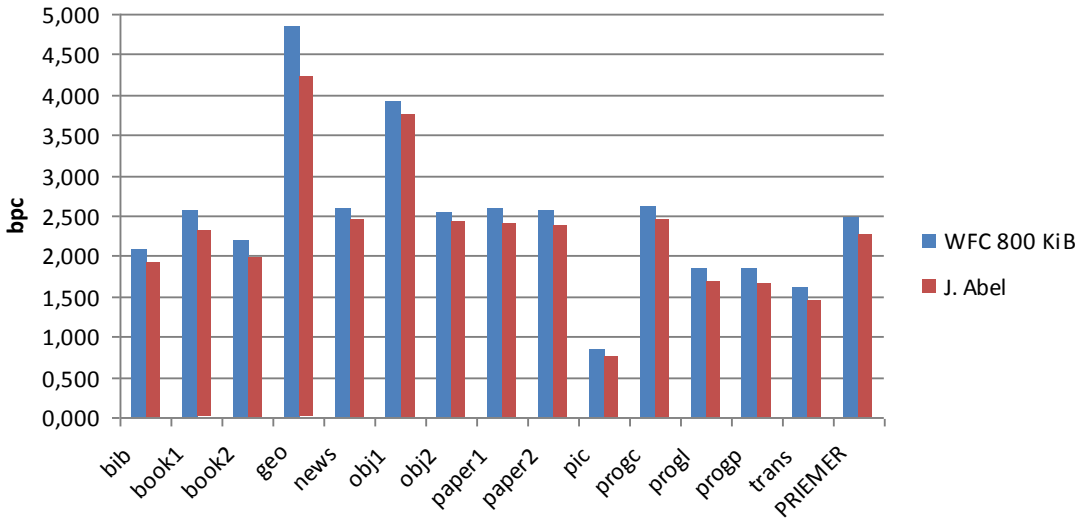
ifc_diff = 8
ifc_size = 32
ifc_limit = 64
ifc_scale = 128

## 3.4 Porovnanie s výsledkami J. Abela

Vstupné dáta v tomto porovnávaní sú reprezentované Calgary korpusom. Pred samotným porovnávaním bolo nutné vybrať správny typ GST metódy a veľkosť bloku. Niekoľkými meraniami bola ako najlepšia možnosť vybraná metóda WFC, ktorá na tomto korpuse dosahovala najlepšie celkové výsledky. Zvolená veľkosť bloku 800 KiB prevyšuje najväčší súbor tohto korpusu, čo by teoreticky malo viesť k najlepším kompresným pomerom. Zo získaných údajov vyplýva, že aplikácia J. Abela dosahuje lepších výsledkov. Hlavným rozdielom je fáza RLE. J. Abel premiestnil túto fázu pred GST, zatiaľ čo implementačné riešenie tejto práce používa klasický model. Implementovaná metóda RLE navyše nepoužíva hodnotu prahu pre určenie začiatku spustenia kódovania, čo vedie k obecne horším výsledkom ako pri ostatných implementáciách.




**Graf 3.3:** Porovnanie výkonu jednotlivých metód, celý Calgary korpus, veľkosť bloku 800 KiB



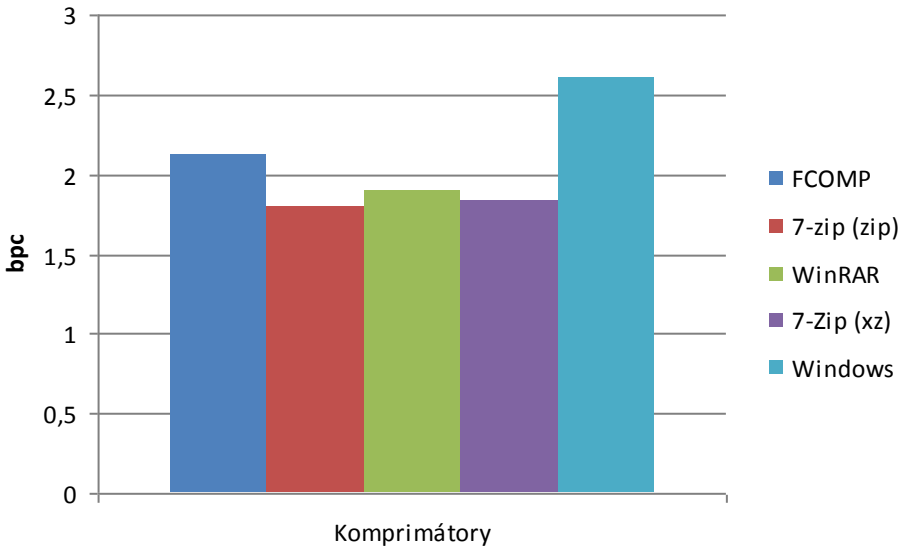

**Graf 3.4:** Porovnanie kompresie jednotlivých súborov

## 3.5 Porovnanie s inými programami

Pre porovnávanie bol aj v tomto prípade zvolený Silesia korpus. Testovanie prebiehalo na počítači s operačným systémom Microsoft Windows XP Professional SP3 (32-bit). Ako konkurenti boli vybrané rozšírené komprimátory 7-Zip a WinRAR, spolu so základným nástrojom integrovaným vo Windows XP pre kompresiu a dekompresiu formátu ZIP Tento test bol koncipovaný ako záťažový pre dosiahnutie najlepšieho výsledného kompresného pomeru, preto boli nastavené najlepšie dostupné parametre. Implementovaný program bol použitý s metódou WFC, ktorá sa v predchádzajúcich testoch javila ako najlepšia. Veľkosť bloku bola nastavená na 10 MiB. Program WinRAR (verzia 5.00 beta 3) použil formát archívu RAR

s úrovňou kompresie Best, veľkosťou slovníka 4096 KB a s povolenou kompresiou textu. Program 7-Zip bol použitý s dvoma rôznymi nastaveniami. Pri formáte ZIP použil úroveň kompresie Ultra, kompresnú metódu PPMd a 256 MB veľký slovník s veľkosťou slova 16. Pri formáte XZ bola nastavená úroveň kompresie Ultra, kompresná metóda LZMA2 a 64 MB veľký slovník s veľkosťou slova 273. Komprimačný nástroj systému Windows XP nemá žiadne nastavenia a bol použitý len ako referencia.

Profesionálne programy dosiahli jednoznačne najlepšie výsledky, implementovaný program bol lepší iba v porovnaní s nástrojom systému Windows. Jeho časová náročnosť je však príliš vysoká pre bežné používanie, pri časovo neobmedzených úlohách ako kompresia záloh alebo komprimovanie na pozadí pri slabom vyťažení procesora by sa po úpravách mohol použiť na tieto špecializované úlohy.




**Graf 3.5:** Porovnanie výkonnosti programov

# 4 Záver

Táto práca sa zaoberala teoretickým úvodom do problematiky kompresie dát, vybranými štatistickými metódami a ich následnou analýzou, implementáciou a testovaním. Opisované metódy boli súčasťou viacstupňového Burrows-Wheelerovho kompresného algoritmu. Pre implementačnú a testovaciu časť bolo vybraných päť zástupcov fázy GST, ako entropický kodér bolo použité aritmetické kódovanie.

Demonštračný program je implementovaný v programovacom jazyku C. Zdrojové kódy jednotlivých kompresných metód sú v oddelených zdrojových súboroch, takže sú ľahko prístupné pre opätovné použitie v inej aplikácii. Každá z metód je reprezentovaná funkciou en_METODA pre kódovací proces, o dekódovanie sa postará funkcia de_METODA.

Výkonnosť bola testovaná a porovnávaná na korpuse Silesia, v prípade porovnávania s výsledkami J. Abela bol použitý Calgary korpus. Ako rozhodujúcim faktorom pre výsledný kompresný pomer sa ukázala veľkosť bloku načítavaných dát. Väčšia hodnota viedla obecne k lepšej kompresii, nevýhodou však bola vyššia časová náročnosť komprimačného procesu. Pri možnostiach dnešnej výpočtovej techniky si môžeme dovoliť vyššiu záťaž pamäte aj procesora, čo povedie k väčšej úspore miesta. Dekompresné procesy trvali kratšiu dobu ako ich časovo náročnejšie kompresné ekvivalenty. Najlepšie kmpresné pomery z vybraných metód dosahovala Weighted Frequency Count, v rýchlosti však zaostávala za Move-To-Front.

Implementovaný program bol horší v porovnaní so známymi programami 7-Zip a WinRAR, tie však boli nastavené tak, aby dosahovali čo najlepšiu kompresiu. Priestor pre zlepšenie ponúka prechod na vyšší programovací jazyk (napríklad C++), ktorý by priniesol značné časové zrýchlenie pri alokovaní potrebnej pamäte. Rozhranie umožňuje jednoduché pridávanie nových metód, či preusporiadanie tých súčasných. Tieto kroky môžu nielen urýchliť celý algoritmus, ale zlepšiť aj výsledný kompresný pomer. Ďalšou možnosťou je použitie viacvláknového spracovania.

# Literatúra


[1] SALOMON, D.: *Data Compression: The Complete Reference*. Springer, 2004. 899 s. ISBN 0-387-40697-2.

[2] CLEARY, J. G.; BELL, T.; WITTEN, I. H.: *The Calgary Corpus*. Dostupné na URL: <http://corpus.canterbury.ac.nz/descriptions/>, 2001.

[3] BELL, T.: *The Canterbury Corpus*. Dostupné na URL: <http://corpus.canterbury.ac.nz/descriptions/>, 2001.

[4] DEOROWICZ, S.: *Silesia compression corpus*. Dostupné na URL: <http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia>, 2011.

[5] SAYOOD, K.: *Introduction to Data Compression*. Morgan Kaufmann, 2005. 704 s. ISBN 0-126-20862-X.

[6] ABEL, J.: Post BWT stages of the Burrows–Wheeler compression algorithm. *Software: Practice and Experience*,. August 2010. Dostupné na URL: <http://onlinelibrary.wiley.com/doi/10.1002/spe.982/pdf>, 2010.

[7] ALBERTS, S.: Improved randomized on-line algorithms for the list update problem. *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995. s. 412-419.

[8] ABEL, J.: Incremental frequency count - a post BWT-stage for the Burrows-Wheeler compression. *Software: Practice and Experience*, 2007. Dostupné na URL: <http://www.juergen-abel.info/Preprints/Preprint_IFC.pdf>, 2007.

[9] ABEL, J.: A fast and efficient post BWT-stage for the Burrows-Wheeler Compression Algorithm. *Proceedings of the IEEE Data Compression Conference 2005*, 2005. s. 449.

[10] BINDER, E.: *Distance Coder*. Dostupné na URL: <http://groups.google.com/groups?selm=390B6254.D5113AD2%40T-Online.de>, 2000.

[11] ELIAS, P.: Interval and Recency Rank Source Coding: Two On-Line Adaptive Variable-Length Schemes. *IEEE Transactions on Information Theory*, 1987. s. 194-203.

# Zoznam príloh

**A** Obsah CD

**B** Tabuľky s výsledkami testov

# A Obsah CD

- **bin** – Priečinok so spustiteľnými súbormi aplikácie pre rôzne systémy
- **src** – Priečinok so zdrojovými súbormi, vrátane Makefile
- **data** – Priečinok s číselnými údajmi vo formáte XLS (Microsoft Excel)
- **readme.txt –** Dokumentácia k aplikácii
- **thesis.pdf** – Text bakalárskej práce vo formáte PDF
- **thesis.doc** – Text bakalárskej práce vo formáte DOC (Microsoft Word)

# B Tabuľky s výsledkami testov

| | *FCOMP* | *J. Abel* |
|---|---|---|
| *bib* | 2,076 | 1,912 |
| *book1* | 2,577 | 2,320 |
| *book2* | 2,190 | 1,981 |
| *geo* | 4,858 | 4,236 |
| *news* | 2,599 | 2,449 |
| *obj1* | 3,935 | 3,765 |
| *obj2* | 2,536 | 2,423 |
| *paper1* | 2,603 | 2,414 |
| *paper2* | 2,564 | 2,373 |
| *pic* | 0,830 | 0,748 |
| *progc* | 2,621 | 2,454 |
| *progl* | 1,847 | 1,683 |
| *progp* | 1,844 | 1,665 |
| *trans* | 1,611 | 1,446 |
| ***PRIEMER*** | **2,478** | **2,276** |

**Tabuľka B.1:** Porovnanie výkonnosti [bpc] programu J. Abela a FCOMP - metóda WFC, blok 800 KiB

| | *FCOMP* | *7-Zip (zip)* | *WinRAR* | *7-Zip (xz)* | *Windows* | |
|---|---|---|---|---|---|---|
| *dickens* | 2729847 | 2269499 | 2383033 | 2831076 | 3959942 | 10192446 |
| *mozilla* | 18558933 | 15773795 | 15342774 | 13376300 | 19159059 | 51220480 |
| *mr* | 2634149 | 2311912 | 2784713 | 2753692 | 3695509 | 9970564 |
| *nci* | 1744976 | 2040888 | 2079199 | 1450032 | 3399926 | 33553445 |
| *ooffice* | 2903410 | 2506703 | 2298856 | 2427324 | 3118482 | 6152192 |
| *osdb* | 2864369 | 2370494 | 3224183 | 2842692 | 3799968 | 10085684 |
| *reymont* | 1243006 | 1022713 | 1073624 | 1315076 | 1941056 | 6627202 |
| *samba* | 4854602 | 3805103 | 3901279 | 3738988 | 5510056 | 21606400 |
| *sao* | 5395291 | 4777152 | 5538956 | 4422424 | 5371844 | 7251944 |
| *webster* | 8268815 | 6449332 | 7172183 | 8369796 | 12483395 | 41458703 |
| *x-ray* | 4551002 | 3848653 | 4137423 | 4492448 | 6053959 | 8474240 |
| *xml* | 466988 | 398077 | 403937 | 434304 | 731547 | 5345280 |
| ***KORPUS*** | **56215388** | **47574321** | **50340160** | **48454152** | **69224743** | **211938580** |

**Tabuľka B.2:** Veľkosti súborov [B] komprimovaných jednotlivými kompresormi

| | *MTF* | *MTF1* | *MTF2* | *WFC* | *IFC* | |
|---|---|---|---|---|---|---|
| *dickens* | 6321483 | 6375479 | 6374090 | 6240345 | 6275934 | 10192446 |
| *mozilla* | 26905425 | 27281854 | 27317477 | 26743312 | 26852578 | 51220480 |
| *mr* | 4009118 | 4006547 | 3963219 | 4011520 | 4048060 | 9970564 |
| *nci* | 7842103 | 8031551 | 8098414 | 7815227 | 7822576 | 33553445 |
| *ooffice* | 4093662 | 4129199 | 4130495 | 4071467 | 4085983 | 6152192 |
| *osdb* | 8714641 | 8737376 | 8743405 | 8668369 | 8662399 | 10085684 |
| *reymont* | 3308279 | 3342241 | 3355393 | 3288783 | 3287258 | 6627202 |
| *samba* | 10650745 | 10962197 | 10987045 | 10602895 | 10657669 | 21606400 |
| *sao* | 6543757 | 6566151 | 6565671 | 6510634 | 6528591 | 7251944 |
| *webster* | 23286987 | 23926992 | 24010379 | 23068483 | 23244460 | 41458703 |
| *x-ray* | 6668738 | 6639000 | 6607687 | 6636402 | 6670632 | 8474240 |
| *xml* | 2330136 | 2439544 | 2452307 | 2320551 | 2329082 | 5345280 |
| ***KORPUS*** | **110675074** | **112438131** | **112605582** | **109977988** | **110465222** | **211938580** |

**Tabuľka B.3:** Veľkosti súborov [B] komprimovaných jednotlivými metódami, veľkosť bloku 1 KiB

| | *MTF* | *MTF1* | *MTF2* | *WFC* | *IFC* | |
|---|---|---|---|---|---|---|
| *dickens* | 4434566 | 4450825 | 4437391 | 4353035 | 4381464 | 10192446 |
| *mozilla* | 21085945 | 21233405 | 21235800 | 20929858 | 21001411 | 51220480 |
| *mr* | 3213198 | 3215095 | 3216011 | 3197363 | 3197758 | 9970564 |
| *nci* | 4353111 | 4483259 | 4500510 | 4326635 | 4349576 | 33553445 |
| *ooffice* | 3405405 | 3419064 | 3418242 | 3344522 | 3363672 | 6152192 |
| *osdb* | 6739431 | 6842902 | 6846432 | 6659520 | 6726744 | 10085684 |
| *reymont* | 2121017 | 2112245 | 2110700 | 2087045 | 2078876 | 6627202 |
| *samba* | 6850872 | 7030960 | 7056236 | 6827977 | 6872853 | 21606400 |
| *sao* | 5718328 | 5707725 | 5701379 | 5660137 | 5653080 | 7251944 |
| *webster* | 14416415 | 14645729 | 14643614 | 14179302 | 14340147 | 41458703 |
| *x-ray* | 6081210 | 6205395 | 6209591 | 6098623 | 6161016 | 8474240 |
| *xml* | 1040946 | 1081746 | 1092898 | 1033125 | 1038197 | 5345280 |
| ***KORPUS*** | **79460444** | **80428350** | **80468804** | **78697142** | **79164794** | **211938580** |

**Tabuľka B.4:** Veľkosti súborov [B] komprimovaných jednotlivými metódami, veľkosť bloku 10 KiB

| | *MTF* | *MTF1* | *MTF2* | *WFC* | *IFC* | |
|---|---|---|---|---|---|---|
| *dickens* | 3563341 | 3549707 | 3522296 | 3482039 | 3489622 | 10192446 |
| *mozilla* | 19662448 | 19707508 | 19695985 | 19410251 | 19531788 | 51220480 |
| *mr* | 2898995 | 2900590 | 2902801 | 2852209 | 2846090 | 9970564 |
| *nci* | 2660124 | 2716623 | 2722631 | 2637627 | 2659482 | 33553445 |
| *ooffice* | 3159036 | 3152413 | 3149235 | 3075270 | 3098941 | 6152192 |
| *osdb* | 4018338 | 4153322 | 4159108 | 3960242 | 4001291 | 10085684 |
| *reymont* | 1621502 | 1604600 | 1597005 | 1577033 | 1578549 | 6627202 |
| *samba* | 5346331 | 5457473 | 5481239 | 5347523 | 5396370 | 21606400 |
| *sao* | 5414614 | 5386894 | 5378554 | 5339975 | 5305628 | 7251944 |
| *webster* | 11025402 | 11063446 | 11011373 | 10797229 | 10875586 | 41458703 |
| *x-ray* | 4932827 | 4967612 | 4947031 | 4932043 | 4890431 | 8474240 |
| *xml* | 626060 | 636934 | 640646 | 618239 | 623750 | 5345280 |
| ***KORPUS*** | **64929018** | **65297122** | **65207904** | **64029680** | **64297528** | **211938580** |

**Tabuľka B.5:** Veľkosti súborov [B] komprimovaných jednotlivými metódami, veľkosť bloku 100 KiB

| | *MTF* | *MTF1* | *MTF2* | *WFC* | *IFC* | |
|---|---|---|---|---|---|---|
| *dickens* | 3065146 | 3031625 | 2991925 | 2972457 | 2979747 | 10192446 |
| *mozilla* | 19051084 | 19053470 | 19029273 | 18749958 | 18975867 | 51220480 |
| *mr* | 2785998 | 2782467 | 2768913 | 2737439 | 2755830 | 9970564 |
| *nci* | 2070654 | 2098819 | 2106302 | 2048680 | 2105850 | 33553445 |
| *ooffice* | 3019623 | 3001436 | 2996498 | 2925410 | 2949120 | 6152192 |
| *osdb* | 3048428 | 3037166 | 3034615 | 3001074 | 3007851 | 10085684 |
| *reymont* | 1396560 | 1379514 | 1369624 | 1354471 | 1356355 | 6627202 |
| *samba* | 4978512 | 5052753 | 5075876 | 4965134 | 5036071 | 21606400 |
| *sao* | 5382262 | 5348223 | 5336054 | 5312071 | 5231306 | 7251944 |
| *webster* | 9425959 | 9369570 | 9273806 | 9173421 | 9220158 | 41458703 |
| *x-ray* | 4648112 | 4648611 | 4592585 | 4608644 | 4571563 | 8474240 |
| *xml* | 485501 | 492352 | 495631 | 478024 | 481175 | 5345280 |
| ***KORPUS*** | **59357839** | **59296006** | **59071102** | **58326783** | **58670893** | **211938580** |

**Tabuľka B.6:** Veľkosti súborov [B] komprimovaných jednotlivými metódami, veľkosť bloku 1 MiB

| | *MTF* | *MTF1* | *MTF2* | *WFC* | *IFC* | |
|---|---|---|---|---|---|---|
| *dickens* | 2835392 | 2789555 | 2739689 | 2729847 | 2741899 | 10192446 |
| *mozilla* | 18876628 | 18848402 | 18811572 | 18558933 | 18804385 | 51220480 |
| *mr* | 2685792 | 2677964 | 2663366 | 2634149 | 2638851 | 9970564 |
| *nci* | 1760956 | 1774959 | 1781195 | 1744976 | 1781240 | 33553445 |
| *ooffice* | 3002793 | 2977707 | 2970876 | 2903410 | 2927186 | 6152192 |
| *osdb* | 2905622 | 2884533 | 2880610 | 2864369 | 2847468 | 10085684 |
| *reymont* | 1283561 | 1266792 | 1254580 | 1243006 | 1242788 | 6627202 |
| *samba* | 4872476 | 4938260 | 4966358 | 4854602 | 4922053 | 21606400 |
| *sao* | 5466373 | 5435228 | 5419220 | 5395291 | 5280058 | 7251944 |
| *webster* | 8543794 | 8447610 | 8326184 | 8268815 | 8336230 | 41458703 |
| *x-ray* | 4587532 | 4583362 | 4518469 | 4551002 | 4529084 | 8474240 |
| *xml* | 475231 | 482011 | 486050 | 466988 | 477374 | 5345280 |
| ***KORPUS*** | **57296150** | **57106383** | **56818169** | **56215388** | **56528616** | **211938580** |

**Tabuľka B.7:** Veľkosti súborov [B] komprimovaných jednotlivými metódami, veľkosť bloku 10 MiB

| | *MTF* | *MTF1* | *MTF2* | *WFC* | *IFC* | |
|---|---|---|---|---|---|---|
| *dickens* | 2835392 | 2789555 | 2739689 | 2729847 | 2741899 | 10192446 |
| *mozilla* | 19237469 | 19155740 | 19107279 | 18866130 | 19060722 | 51220480 |
| *mr* | 2685792 | 2677964 | 2663366 | 2634149 | 2638851 | 9970564 |
| *nci* | 1611157 | 1623805 | 1629148 | 1597293 | 1623086 | 33553445 |
| *ooffice* | 3002793 | 2977707 | 2970876 | 2903410 | 2927186 | 6152192 |
| *osdb* | 2905622 | 2884533 | 2880610 | 2864369 | 2847468 | 10085684 |
| *reymont* | 1283561 | 1266792 | 1254580 | 1243006 | 1242788 | 6627202 |
| *samba* | 4990901 | 5057335 | 5083001 | 4965009 | 5033134 | 21606400 |
| *sao* | 5466373 | 5435228 | 5419220 | 5395291 | 5280058 | 7251944 |
| *webster* | 8146803 | 8041998 | 7910722 | 7865465 | 7945101 | 41458703 |
| *x-ray* | 4587532 | 4583362 | 4518469 | 4551002 | 4529084 | 8474240 |
| *xml* | 475231 | 482011 | 486050 | 466988 | 477374 | 5345280 |
| ***KORPUS*** | **57228626** | **56976030** | **56663010** | **56081959** | **56346751** | **211938580** |

**Tabuľka B.8:** Veľkosti súborov [B] komprimovaných jednotlivými metódami, veľkosť bloku 100 MiB