VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

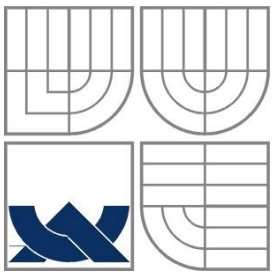
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLOVNÍKOVÉ METODY KOMPRESE DAT

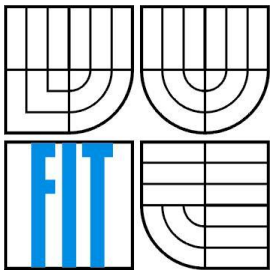
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Sergij Černičko
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLOVNÍKOVÉ METODY KOMPRESE DAT

DICTIONARY METHODS OF DATA COMPRESSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Sergij Černičko
AUTHOR

VEDOUCÍ PRÁCE — Ing. David Bařina
SUPERVISOR

BRNO 2011

Abstrakt

Náplní této bakalářské práce je analýza slovníkových kompresních metod a jejich vlastností. Práce se zaměřuje a podává podrobný přehled slovníkových metod LZ77, LZ78, LZSS, LZW, DEFLATE a LZX. Jsou zde popsány principy těchto metod, a obecná teorie komprese. Závěrem jsou všechny metody otestovány a je provedena analýza výsledků.

Abstract

The aim of this thesis is to analyse dictionary methods of data compression and their characteristics. The work is focused on and hands out detail overview of dictionary methods LZ77, LZ78, LZSS, LZW, DEFLATE and LZX. Principles of these methods and also general theory of compression is described. In conclusion all methods are tested and results analyzed.

Klíčová slova

Komprese dat, komprese, bezztrátová komprese, slovníkové metody komprese dat, DEFLATE, LZ77, LZ78, LZSS, LZX, LZW.

Keywords

Data compression, compression, lossless compression, dictionary methods of data compresion, DEFLATE, LZ77, LZ78, LZSS, LZX, LZW.

Citace

Černičko Sergij: Slovníkové metody komprese dat, bakalářská práce, Brno, FIT VUT v Brně, 2011

Slovníkové metody komprese dat

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.

……
Sergij Černičko
16. května 2011

Poděkování

Děkuji vedoucímu práce panu Ing. Davidu Bařinovi za odbornou pomoc, kterou mi poskytl při vypracování této práce.

Obsah

1 Úvod

V současnosti se setkáváme s kompresí dat poměrně často, nejvíce asi v informačních technologiích. Komprese se využívá hlavně k ušetření paměti v počítačích nebo k urychlení přenosů dat. Metod kompresí existuje celá řada, některé zde budou vysvětleny, avšak v této práci se budeme zabývat podrobně pouze slovníkovými metodami komprese dat. I těchto metod je poměrně mnoho, a proto v práci rozebereme jen ty základní (LZ77, LZSS, LZ78, LZW, DEFLATE a LZX).

Celá práce je rozdělena do kapitol a podkapitol dle tematických okruhů. Ve druhé kapitole této bakalářské práce bude čtenář podrobněji obeznámen s pojmem komprese dat, jsou zde vysvětleny základní pojmy a také krátká historie. Tady se čtenář dozví, kdy, jak a proč komprese dat vznikla a její rozvoj v průběhu let.

Třetí kapitola se zabývá základními technikami a metodami komprese dat. Zde jsou pouze okrajově vysvětleny některé metody komprese, jako jsou například statistické metody.

Čtvrtá kapitola popisuje slovníkové metody komprese dat. Toto téma zde je rozebráno podrobněji než předchozí metody. Kapitola čtenáři objasní pojem slovníková komprese a jsou zde vysvětleny vybrané způsoby implementací této komprese.

Tato bakalářská práce zahrnuje kromě tohoto textu také knihovnu, ve které jsou implementovány zadané způsoby slovníkové komprese. Proto se v páté kapitole věnuji srovnání těchto způsobů na různých datech.

2 Komprese dat

V této kapitole je vysvětlen pojem komprese dat a jsou zde také uvedeny některé důležité pojmy a vlastnosti z tohoto oboru. V závěru je krátká historie počátku komprese. Při vytvoření této kapitoly byly použity publikace [1] a [2].

2.1 Charakteristika komprese

Komprese je postup při ukládání nebo transportu dat, jehož úkolem je zmenšit velikost vstupního datového toku. Jinak řečeno komprese provádí zmenšení nějakých dat, tak aby zabírala co nejmenší prostor.

Pomocí komprese lze ukládat mnohem více dat na disk nebo na paměťová média. Také při transportu dat komprese značně sníží délku trvání přenosu. Příkladem nějaké komprese jsou například obrázky, videa nebo streamované online přenosy videí.

Komprese odstraňuje redundantní (opakující se) data ze zadaného datového toku, dokud není bez redundance. Různé typy dat vyžadují různé typy komprese, avšak princip zůstává stejný. Komprimovaná data zabírají sice méně místa nebo umožňují rychlejší přenosy, avšak pro jejich správné přečtení je potřeba tato data dekomprimovat.

2.2 Základní pojmy a vlastnosti komprese dat

Zde jsou uvedeny již zmíněné pojmy, které budou použity v popisech jednotlivých kompresních metod.

Kodér neboli *kompresor* je program, který komprimuje vstupní datový tok na výstupní. Naopak *dekodér* neboli *dekompresor* je program, který komprimovaná data převede zpět. *Kodek* je program, který provádí kompresi i dekompresi.

Kód je způsob reprezentace dat a informací při jejich uložení do paměti nebo při jejich přenosu. Příkladem je ACSII kód, který znaky reprezentuje jako 1 bytové číslo. Proces převodu datového toku na *kód* se nazývá *kódování*.

Komprese můžeme rozdělit dle několika různých pohledů. *Neadaptivní kompresní metoda* nemění svoje operace, má pevný postup a je pouze pro určitý druh dat, protože na některých by nedosahovala dobré komprese. Jiný případ je *adaptivní metoda*, která přizpůsobuje svoje operace dle vstupních dat. Některé metody jsou *jednoprůchodové* a zaberou méně času než metody *dvojprůchodové*.

Některé kompresní metody mohou dosahovat lepších výsledků za cenu vypuštění některých dat. Dekomprimovaná data nejsou tedy shodná se vstupními. Těmto kompresím se říká *ztrátové*. Tyto komprese jsou použity u obrázků, videí, zvuků. Je zde využito nedokonalosti lidských smyslů, jako je zrak nebo sluch. Avšak vypuštění dat by mělo být takové, aby ho člověk nezpozoroval. Tomuto se říká *perceptivní komprese*. Opakem *ztrátových kompresí* jsou *komprese bezztrátové*. U nich nedochází k vypouštění informací, a tedy dekomprimovaná data se rovnají původním datům.

Textové soubory by měly být komprimovány bezztrátově. Zde by vadila jakákoliv ztráta znaků kromě opakujících se bílých znaků. Kdybychom měli knihu, na které by byla provedena komprese, tak při odstranění např. písmene „a“ dostaneme nečitelnou knihu. Jestliže bychom ale odstranili jednu ze dvou mezer, které jsou vedle sebe, pořád by byla kniha čitelná.

Další možností jak lze komprese rozdělit je na *symetrické* a *asymetrické*. Jestliže kompresor a dekompresor používají stejný algoritmus, ale dekompresor pracuje opačným směrem (inverzně), jedná se o *komprese symetrické*. V případě *asymetrických kompresí* je algoritmus jiný a buď kompresor, nebo dekompresor pracuje výrazně déle. Toho se využívá například u zálohování, kde se nepředpokládá častá dekomprese, ale měla by být rychlá komprese. Mnoho moderních metod je asymetrických a jejich formální popis se skládá z dekodéru a popisu formátu komprimovaného toku. Každý kodér generující správný komprimovaný tok se potom označuje jako *pracující ve shodě* (compliant) a stejně tak i dekodér. Výhodou tohoto způsobu je, že každý si může vytvořit vlastní kodér i dekodér. Takový vytvořený kodér je potom *algoritmický* a dekodér *deterministický*. Kompresní metoda se označuje jako *univerzální*, pokud kompresor a dekompresor nevyužívá statistik vstupního toku.

Dalším důležitým pojmem je *rozdílování souborů*. Tento pojem udává činnost, při které chceme komprimovat jen některé rozdíly v souborech a ne celý soubor. Například mějme dva identické soubory. Jeden změníme a chceme, aby byl druhý soubor identický. Místo komprimování celého souboru se zašlou pouze změny v prvním souboru.

Většina metod pracuje v *streaming módu* neboli v *proudovém režimu*, kde se pracuje pouze s určitým počtem bytů, než se dojde na konec souboru. Další možností je práce v *blokovém režimu*. V tomto případě se data zpracovávají po blocích a každý blok se kóduje zvlášť. Velikost bloku je nastavitelná a ovlivňuje účinnost kompresní metody.

Kompresní metody mohou být *fyzické* a *logické*. Většina metod je fyzických, tedy zpracovávají vstupní datový tok a nahrazují ho výstupním. *Logické metody* vstupní tok ještě člení a nahrazují jeho datové položky kratšími položkami. Tyto metody se dají využít jenom u konkrétního typu dat, a proto nejsou tolik rozšířené.

2.3 Parametry komprese

Kvalita a výkon komprese je určena několika veličinami.
Kompresní poměr je definován jako:

$$Kompresní\ poměr = \frac{Velikost\ výstupního\ toku}{Velikost\ vstupního\ toku} \qquad \textbf{(2.1)}$$

Jestliže je hodnota poměru větší než 1, znamená to, že nedošlo ke kompresi. Výsledný soubor by byl větší než původní a toto je nežádoucí. Výsledek 0,5 znamená, že výsledný soubor zabírá 50 % původního. Kompresní poměr se udává také jako počet bitů na bit (bpb), to nám říká kolik výstupních bitů je potřeba na bit vstupní. U obrázků se používá označení počet bitů na pixel (bpp) a u textů je to počet bitů na znak (bpc). Tyto pojmy se označují jako *bitová rychlost*. Dalším parametrem je *bitová režie*. Například máme komprimovaný tok a v něm 15 % bitů nese záznam s popisem tabulek a 85% jsou vlastní data. Tedy bitová režie je 15 %.

Kompresní faktor je definován jako:

$$Kompresní\ faktor = \frac{1}{Kompresní\ poměr} = \frac{Velikost\ vstupního\ toku}{Velikost\ výstupního\ toku} \quad \textbf{(2.2)}$$

Jedná se tedy o převrácenou hodnotu kompresního poměru. A pro kompresi musí být hodnota větší jak 1. Čím větší číslo, tím lepší komprese. Pro měření *míry komprese* lze použít výraz:

$$100 * (1 - Kompresní\ poměr) \quad \textbf{(2.3)}$$

Při dosaženém výsledku 60 zjistíme, že bylo ušetřeno 60 % původního objemu nebo to můžeme brát tak, že výstupní tok zabírá 40 % původního obsahu.

Referenční velikost odpovídá velikosti vstupního toku nebo velikosti výstupního toku, který byl vytvořen nějakou standardní bezztrátovou kompresní metodou.

Provádí-li se komprese pomocí logických obvodů, tak zavádíme další veličinu a tou je *rychlost komprese*. Měří se v počtu cyklů na bit (cpb). Udává střední počet strojových cyklů potřebných ke kompresi jednoho bitu.

Dalším pojmem je *entropie*. Je to základní a důležitý pojem co se týče statistických metod. Entropie závisí na četnosti (počet stejných znaků), či pravděpodobnosti (počet stejných znaků / celkový počet znaků) výskytu prvku. V informatice nese název také *Shannonova entropie* podle Clauda Shannona, který tento termín převzal z termodynamiky.
Definice entropie: Nechť se data skládají z *n* různých prvků

$$a_1, a_2, a_3, \ldots\ldots\ldots\ldots, a_n$$

a tyto prvky se v datech vyskytují s pravděpodobnostmi

$$p_1, p_2, p_3, \ldots\ldots\ldots\ldots, p_n$$

Pak množství informace, která je reprezentována prvkem a_i, udává jeho *entropie*

$$E_i = -\log_2 p_i \quad \textbf{(2.5a)}$$

$$E = -\sum_1^n \log_2 p_i \quad \textbf{(2.5b)}$$

Entropie E_i vyjadřuje, jak velkou informaci nese výskyt prvku a_i. Je zřejmé, že čím je pravděpodobnost výskytu prvku a_i menší, tím je jeho entropie a tedy i míra informace větší.
Pro lepší pochopení je to vysvětleno na příkladu 2.1.

Příklad 2.1: Pro zjednodušení uvažujme, že se datový tok skládá jen ze 4 znaků. Kdyby byly znaky obsaženy v textu rovnoměrně, tak by každý z nich dostal 2bitový kód a střední entropie by byla rovna 2. To je vyznačeno v tabulce 2.1. K zakódování takového řetězce by bylo potřeba celkem 16 bitů.

Znak	a	b	c	d
Četnost	2	2	2	2
Pravděpodobnost	0,25	0,25	0,25	0,25
Kód	00	11	01	10

Tabulka 2.1: Tabulka pro výpočet entropie s rovnoměrnou pravděpodobností

Máme-li znaky s různou četností výskytu, tak si spočítáme z rovnice (2.5b) střední hodnotu entropie, což vychází 1,75. K zapsání každého znaku je tady potřeba 1,75 bitu. Příklad zakódování je v tabulce 2.2. Jestliže si spočítáme celkovou velikost zapsaných bitů na výstup, tak dostaneme hodnotu 14. Což odpovídá průměrné hodnotě 1.75 bitů na znak.

Znak	a	b	c	d
Četnost	4	1	1	2
Pravděpodobnost	0,5	0,125	0,125	0,25
Kód	0	110	111	10

Tabulka 2.2: Tabulka pro výpočet entropie s nerovnoměrnou pravděpodobností

2.4 Historie komprese dat

V roce 1838 vznikla Morseova abeceda, která se používala pro kódování textů při telegrafii. Toto kódování využívalo pouze krátké a dlouhé signály. Sekvence těchto signálů vyjadřovala znak, ale často používané znaky byly vyjádřeny kratšími signály.

Roku 1949 Claude Shannon a Robert Fano vymysleli způsob, jak přiřadit kódovaná slova znakům nebo blokům dle pravděpodobnosti jejich výskytu. Tato metoda se jmenuje Shannon-Fanovo kódování a řadí se mezi metody statistické. David Huffman optimalizoval tuto metodu a vzniká tak Huffmanovo kódování, které dosahuje lepších výsledků. Hlavním rozdílem mezi těmito dvěma metodami je konstrukce kódů. Zatímco Shannon-Fanovo kódování konstruuje kódy shora dolů, tak Huffmanovo kódování konstruuje kódy zdola nahoru. Dřívější kódování bylo pouze součástí hardwaru a až v roce 1970 vzniklo softwarové kódování, založené na adaptivním Huffmannově kódování.

1977 Abraham Lempel a Jacob Ziv objevili způsob komprese dat pomocí ukazatelů (tzv. posuvné okno) a to byl vznik slovníkové metody LZ77. O rok později autoři vymysleli metodu LZ78, která si vytváří slovník použitých frází. Obě tyto metody se staly později základem dalších i dnes používaných metod. Roku 1984 upravil Terry Welch metodu LZ78, a stvořili tak metodu LZW. V roce 1990 vzniklo aritmetické kódování (potřeba matematický koprocesor), které přidělí jeden kód (velmi dlouhý) celému vstupnímu souboru. Tím vylepšuje Huffmanovo kódování, protože to přiděluje kódy pouze celočíselné. Jestliže jsme dostali entropii jednoho znaku např. 1.6, tak Huffmanova metoda mu přidělí kód dlouhý 1 nebo 2 bity.

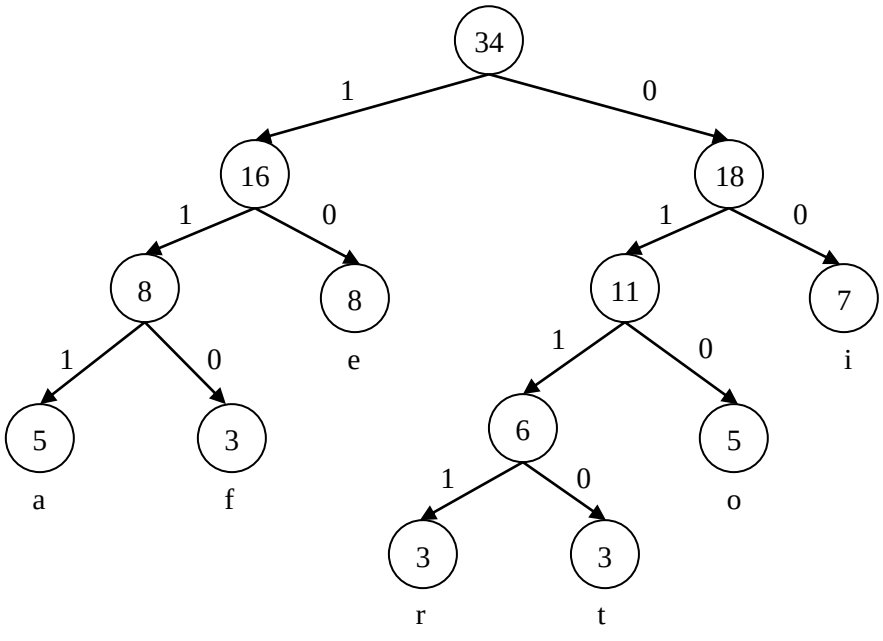
S vyvíjející se technikou rostla potřeba komprimovat data rychleji (např.: mediální přenosy), a proto v roce 1990 byla vyvinuta ztrátová komprese dat (pro obrázky, videa, hudbu). Dnes se tato komprese využívá na počítačích převážně v multimédiích. Nejznámější formáty využívající ztrátovou kompresi jsou např.: JPEG, MPEG, MP3, WMA. Mezi bezztrátové komprese používané v současnosti patří např.: GIF, PNG, PDF.

3 Statistické metody

Tyto metody používají při kompresi *statistický model dat*. Většinou se jedná o četnost výskytu jednotlivých položek. Četnosti výskytů jsou vypočítány předem nebo v průběhu komprese a potom jsou použity pro všechna data. Četnosti nebo pravděpodobnosti výskytu znaků mohou být známy předem, v tom případě *kodér* sestaví „tabulku kódů“ z těchto dat a může rovnou zahájit kompresi. Kodér i dekodér tedy pracují se stejným modelem dat. Nejznámější metoda je Morseova abeceda, která kóduje znaky na sekvenci dlouhých a krátkých signálů. Tato kapitola byla sestavena z publikací [1] a [2].

3.1 Huffmanovo kódování

Jak už bylo uvedeno dříve, rok vzniku tohoto algoritmu je 1951 a patří k nejstarším kompresním metodám. Tato metoda kóduje znaky s nevětší pravděpodobností výskytu do krátkých bitových řetězců, zatímco znaky s menší pravděpodobností výskytu mají delší bitový řetězec. Jde o metodu jednoprůchodovou i dvouprůchodovou. Jestliže známe dopředu četnosti nebo pravděpodobnosti výskytu, potom můžeme tabulku s kódy sestavit ihned. Tomuto způsobu se říká *statický model*. *Semiadaptivní model* sestaví tabulku výskytu znaků po prvním průchodu, a ve druhém průchodu znaky zakóduje dle sestavené tabulky. Pro sestavení *Huffmanova kódu* se používá *binární strom*. Dekompresor potom dekóduje tento binární strom a získá zdrojová data. Tato metoda je bezztrátová a často se používá společně s nějakou optimalizací. Na obrázku 3.1 máme příklad binárního stromu pro *Huffmenovo kódování*, čísla u znaků udávají počet jednotlivých znaků ve vstupním datovém toku. Tabulka 3.1 ukazuje kódy z obrázku 3.1.



Obrázek 3.1: Huffmanův strom

Znak	e	i	a	o	f	r	t
Četnost	8	7	5	5	3	3	3
Pravděpodobnost	0,24	0,21	0,14	0,14	0,09	0,09	0,09
Kód	10	00	111	010	110	0111	0110

Tabulka 3.1: Huffmanovy kódy

V praxi je velice nepraktické procházet datový tok dvakrát. Jednou pro sestavení kódu a po druhé pro kódování. Proto se zrodilo *adaptivní Huffmanovo kódování*. Kodér i dekodér začínají s prázdným Huffmanovým stromem a v průběhu komprese si sestavují svůj vlastní. Důležité je, aby kodér i dekodér v kterémkoliv místě používali stejný kód, přestože se může v průběhu procesu měnit. Tomuto říkáme, že kodér i dekodér jsou synchronní nebo, že dekodér zrcadlí operace kodéru.

Znak nenacházející se ve stromu se posílá na výstup normálně. Kodér ho potom přidá do stromu, a musí opět zkontrolovat celý strom. Kontrola slouží ke zjištění, zda jsou nejkratší kódy přiřazené znakům s nejvyšší četností. V opačném případě je nutné strom přeskládat. Dekodér pracuje stejným způsobem.

3.2 Aritmetické kódování

Tato metoda je obdobně jako Huffmanovo kódování entropická metoda s proměnlivou délkou kódového slova. Vytváří ale intervaly, které jsou umístěny v intervalu $\langle 0, 1)$. Velikosti těchto dílčích intervalů jsou určeny pravděpodobností výskytu znaku. Rozdělení intervalu:

$$(0,p_1), (p_1,p_1+p_2), (p_1+p_2,p_1+p_2+p_3), \ldots, (p_1+p_2+\ldots+p_{n-1},p_1+p_2+\ldots+p_{n-1}+p_n)$$

Zápis lze zjednodušit pomocí tzv. *kumulativních pravděpodobností*:

$$q_0=0,\ q_1=p_1,\ q_2=p_1+p_2,\ \ldots,\ q_n=p_1+p_2+\ldots+p_n \qquad \textbf{(3.1)}$$

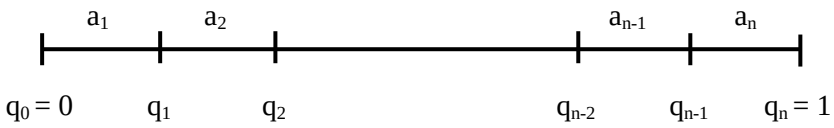
Můžeme popsat rozdělení do intervalu pomocí 3 kroků:

1) Hodnota intervalu I = ⟨0; 1)
2) Dle načteného znaku přiřadíme interval $\langle q_{i-1}; q_i)$ a ze stávající hodnoty intervalu I = ⟨L; H) provedeme jeho přepočet:

$$I = (L+q_{i-1}*(H-L),\ L+q_i*(H-L))$$

3) Jestliže jsme nezpracovali celý vstupní soubor, tak se vracíme opět na 2. krok

Interval potom vypadá nějak takto:

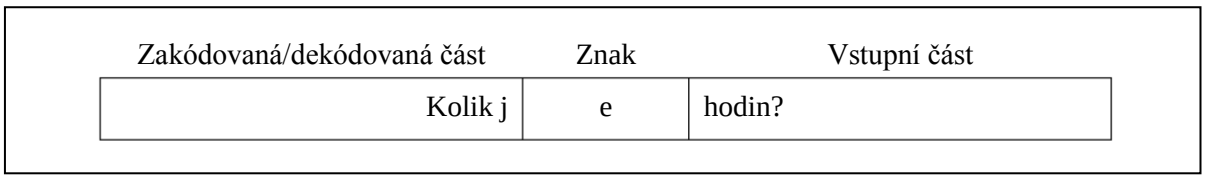


Obrázek 3.2: Interval aritmetického kódování

Dekódování probíhá obdobně jako kódování. Toto bylo kódování u desetinných čísel. Kódování s celými čísly probíhá téměř stejně jenom je zde použit interval I = ⟨0; M⟩, kde M je celé číslo.

3.3 Metoda konečného kontextu

Předchozí metody nepoužívaly *kontextu* pro své kódování a dekódování, můžeme je tedy nazývat bezkontextové. Oproti tomu metoda konečného kontextu používá pro své kódování kontextu, to znamená, že jsou brány v úvahu i jiné znaky nežli právě kódované.



Obrázek 3.3: Metoda konečného kontextu

Jestliže chceme tuto metodu považovat za adaptivní, tak je třeba kódovat/dekódovat znak dle již zakódovaných/dekódovaných znaků a musíme vynechat vstupní část, která obsahuje posloupnost načtených znaků (obrázek 3.3). Potom má každý znak jinou pravděpodobnost výskytu dle již zpracovaných znaků.

Např. na obrázku 3.3 se zpracovává znak **e** a před ním byl znak **j**. Teprve dle tohoto provedeme kódování nebo dekódování. Kdyby se před **e** vyskytoval znak **s,** tak by byla pravděpodobnost výskytu jiná, a tudíž by se provedlo i jiné kódování či dekódování.

Problém zde nastává u délky předchozích znaků. Jestliže máme zakódovat 256 různých znaků, tak u bezkontextových metod máme pouze 256 pravděpodobností výskytu. Avšak u kontextových metod, které by braly v potaz 1 předchozí znak, je 256^2 možných kombinací. Toto číslo roste s rostoucím počten znaků v kontextu (256^3, 256^4, …, 256^n). Proto se bere v úvahu jenom určitý počet znaků v kontextu, a tudíž se tato metoda nazývá *metoda konečného kontextu*.

Další podmínkou pro úspěšné kódování je zvolit si počáteční pravděpodobnosti výskytů znaků, jsou-li načteny poprvé. Toto nelze matematicky nijak odvodit, a proto bylo navrženo několik možných speciálních řešení.

4 Slovníkové metody komprese dat

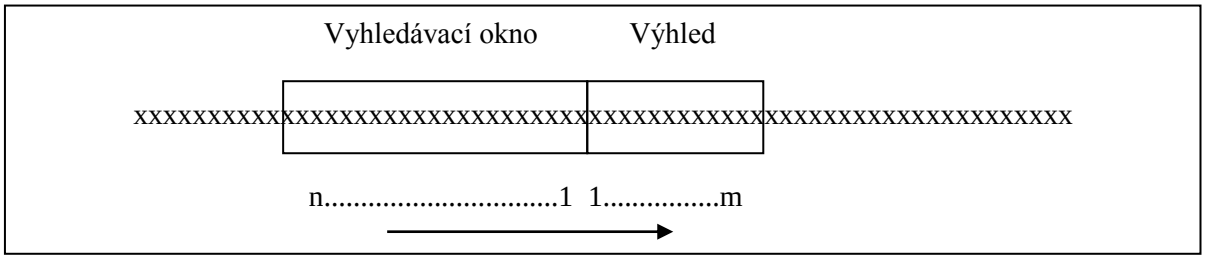
Tyto metody na rozdíl od metod statistických nezpracovávají data po znacích, ale snaží se kódovat nebo dekódovat množinu znaků. U většiny případů jsou to metody jednoprůchodové a adaptivní. Jako první vznikla metoda LZ77, kde písmena L a Z jsou první písmena příjmení zakladatelů (Lempel a Ziv) a číslo je rok vzniku (1977). Další v pořadí byla metoda LZ78. Podle toho jak tyto metody pracují, je můžeme přiřadit do dvou skupin:

- Výskyt skupiny znaků je dán pozicí v již zpracované části a délkou shodných znaků (LZ77)
- Je vytvářen slovník a skupiny znaků jsou nahrazeny odkazem do tohoto slovníku (LZ78)

Ostatní slovníkové metody jsou pouze různou modifikací těchto dvou prvních metod.

4.1 LZ77

Jak již bylo zmíněno na začátku, metoda LZ77 [3] nahrazuje skupiny znaků odkazem na již zpracovaná data. Využívá *posuvné okno* (*sliding window*), které se posouvá po vstupním datovém toku a je rozděleno na dvě části (obrázek 4.1). *Vyhledávací okno* (*search window*) obsahuje část již zpracovaných znaků a v praxi se používá velikost až několik desítek tisíc znaků. *Výhledové okno* (*look-ahead window*) obsahuje část vstupního toku a bývá veliké desítky až stovky znaků. V praxi se ale využívá velikost jen několik desítek znaků, protože delších shod se moc nedosahuje.



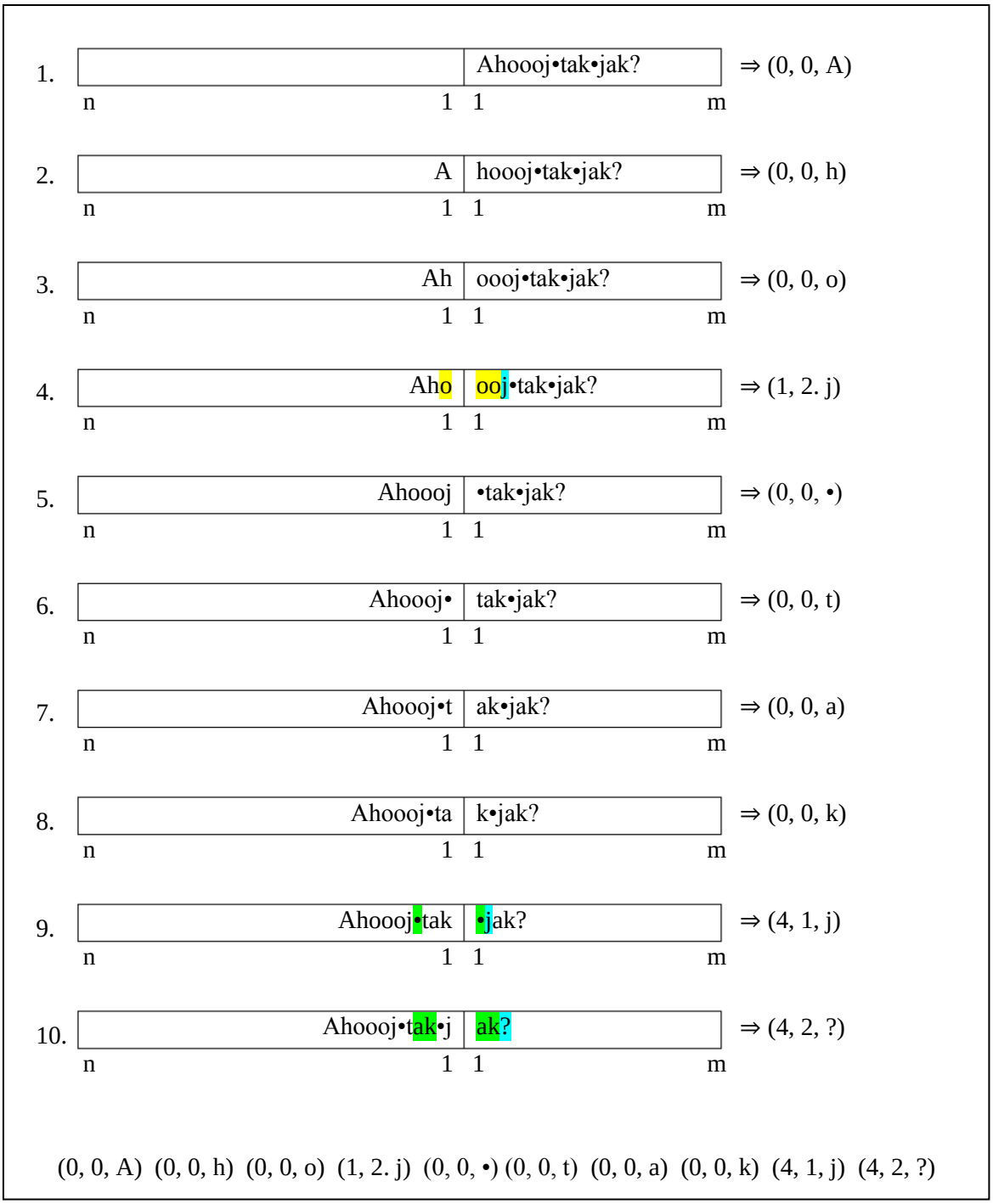
Obrázek 4.1: Posuvné okno

Skupiny znaků se potom zakódují jako např. (5, 4, a). Kde první číslo udává pozici nalezeného prvního znaku z výhledu ve vyhledávacím okně. Druhé číslo vyjadřuje délku shodných znaků a jako poslední je znak, který následuje ve výhledu po skupině shodných znaků. Okno se potom posune doprava o velikost shodných znaků + 1. V tomto případě je délka menší nebo rovna pozici (obrázek 4.2).

Avšak jsou-li ve výhledu opakující se stejné znaky, může být délka větší než je nalezená pozice (obrázek 4.2 - 4. řádek). Toto je zapsáno ve tvaru (1, 5. a), kde tečka udává, že se jedná o tento způsob. LZ77 využívá dva buffery, které reprezentují posuvné okno.

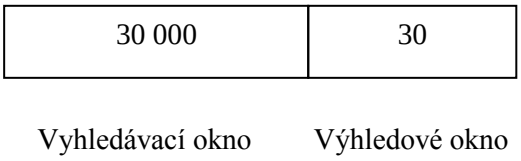
Kodér dle velikosti vyhledávacího a výhledového okna, zjistí nejmenší mocniny 2, které by byly potřeba pro zapsání pozice a délky v kódu. Pro lepší pochopení je to vysvětleno na příkladu 4.1.

Tato metoda je jednoprůchodová, adaptivní a asymetrická. Dekodér pracuje rychleji než kodér. Metoda nevyžaduje mnoho paměťového prostoru, protože se data vyskytují pouze v posuvném okně. Největší problém je vyhledání nejdelšího *prefixu* (předpony) z výhledu ve vyhledávacím okně a také to, že metoda posílá nulovou shodu. Proto je na jeden znak, který se v textu ještě nevyskytl, potřeba stejně bitů, jako na nějakou nalezenou posloupnost. Tento princip je dost neefektivní.



Obrázek 4.2: Princip metody LZ77

Příklad 4.1:



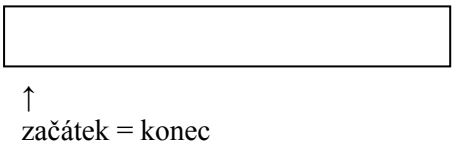
Jelikož je vyhledávací okno velké 30 000 bytů, tak pro jeho zakódování je potřeba 15 bitů Nejmenší mocnina dvojky, aby se do ní vešlo číslo 30 000, je 2^{15} (2^{15} = 32 768). Stejně se to provede u délky, tzn. 2^5 = 32. Pro hodnotu délky je tedy potřeba 5 bitů. Celková velikost kódu je v tomto případě 28 bitů (15 + 5 + 8).

4.2 LZSS

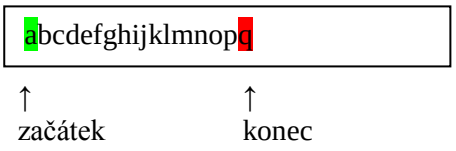
Metoda LZSS [5] vznikla vylepšením LZ77, kterou vytvořili pánové Storer a Szimanski v roce 1982. Oproti LZ77 výsledný kód nevypadá (pozice, délka, znak), ale pouze (flag, pozice, délka) nebo (flag, znak). *Flag* je bitová hodnota, která určuje, zda se jednalo o posloupnost nebo pouze o znak. Podle této hodnoty potom dekodér zpracuje následující bity. Jestliže je hodnota flagu rovna jedné, tak se jedná o posloupnost. V opačném případě se jedná o znak.

U metody LZ77, jak bylo zmíněno v předchozí podkapitole, je jeden z hlavních nedostatků, že zakóduje i posloupnost, která je rovna nule, do celého kódu. To nám ale zabere víc bitů než zakódování samotného znaku. A proto další změnou u LZSS je, že se posloupnost znaků kóduje jako fráze, je-li délka shodných znaků větší jak 2, jinak se kóduje jako znak. Vyhledávání shod se implementuje pomocí binárního vyhledávacího stromu nebo pomocí hashovací tabulky.

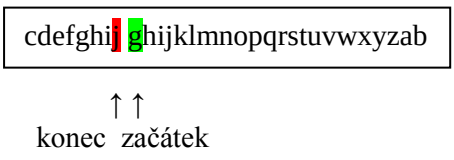
Dalším rozdílem je vyhledávání ve vyhledávacím okně. Celé okno je implementováno jako *cyklický buffer* (obrázek 4.3 – obrázek 4.5) nebo je možné použít seznam. Princip cyklického bufferu je poměrně jednoduchý a spočívá pouze ve vytvoření ukazatelů na první a poslední znak. Když je buffer prázdný, tak ukazují oba ukazatele na začátek. Při plnění se posouvá pouze ukazatel na konec. Jestliže se dostaneme na konec bufferu, tak se začne pohybovat i ukazatel na začátek. Znaky, které se potom nachází mezi ukazateli konec – začátek, se nahradí novými a ukazatel konec se posune před ukazatel začátek.



Obrázek 4.3: Prázdný buffer



Obrázek 4.4: Nenaplněný buffer



Obrázek 4.5: Plně naplněný buffer

Metoda LZSS vytváří *binární vyhledávací strom* (zkráceně *bvs*), do kterého se ukládají odkazy z okna (příklad 4.2). Pomocí tohoto řešení by se měla zkrátit doba vyhledávání nejdelších shod. Slovo měla, zde bylo použito záměrně, protože doba vyhledávání záleží ještě na uspořádání listů stromu (obrázek č. 10).

Binární vyhledávací strom je strom, u kterého má každý uzel nejvýše 2 potomky. Každý uzel obsahuje také klíč, který je v našem případě textový řetězec, podle něhož jsou uzly seřazeny. V levé větvi se ukládají hodnoty menší i stejné a v pravé větvi hodnoty větší.

Binární vyhledávací strom v metodě LZSS má předem daný počet uzlů, který je potřeba dodržet. Počet uzlů se vypočítá z rovnice:

$$Počet\ uzlů\ v\ bvs = velikost\ vyhledávacího\ okna - velikost\ výhledového\ okna + 1 \quad \textbf{(4.1)}$$

Příklad 4.2:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

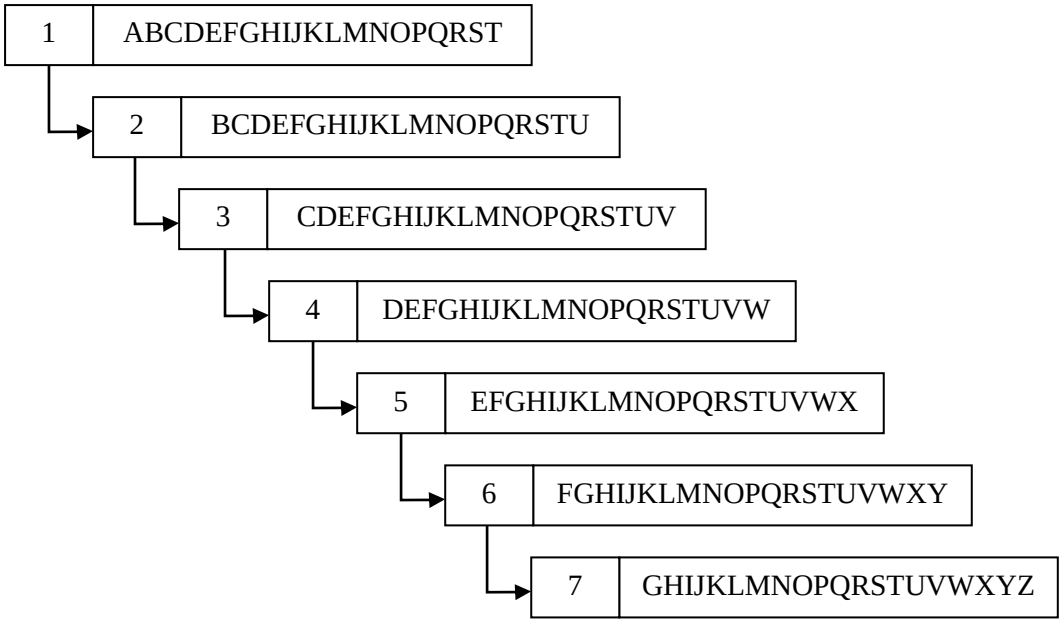
vyhledávací buffer

Velikost vyhledávacího okna = 26
Velikost výhledového okna = 15
Počet uzlů v bvs = 26 - 20 +1 = 7

Náš strom bude obsahovat, jak jsme vypočítali z rovnice (4.1), 7 uzlů.

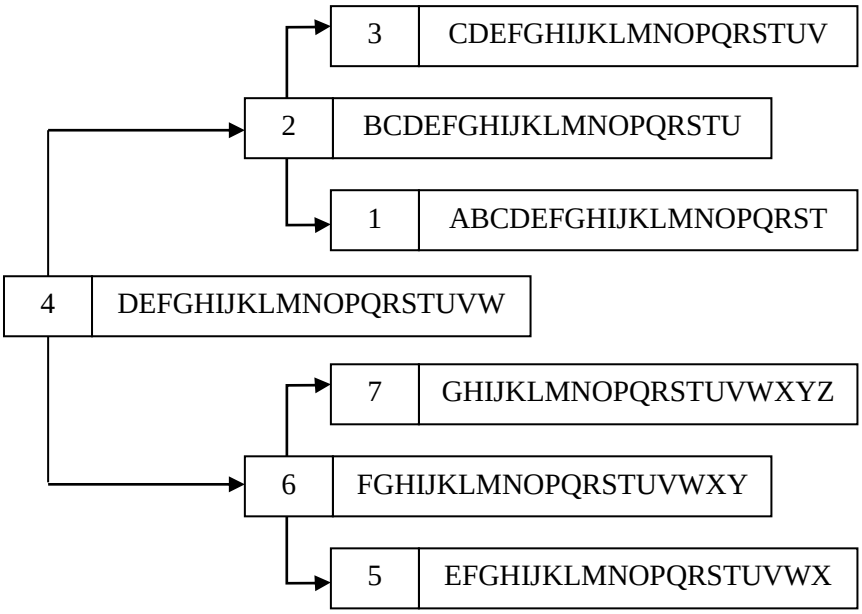
Pozice	Řetězec	Pozice	Řetězec
1	ABCDEFGHIJKLMNOPQRST	5	EFGHIJKLMNOPQRSTUVWX
2	BCDEFGHIJKLMNOPQRSTU	6	FGHIJKLMNOPQRSTUVWXY
3	CDEFGHIJKLMNOPQRSTUV	7	GHIJKLMNOPQRSTUVWXYZ
4	DEFGHIJKLMNOPQRSTUVW		

Tabulka 4.1: Řetězce do uzlů



Obrázek 4.6: Nevyvážený binární vyhledávací strom

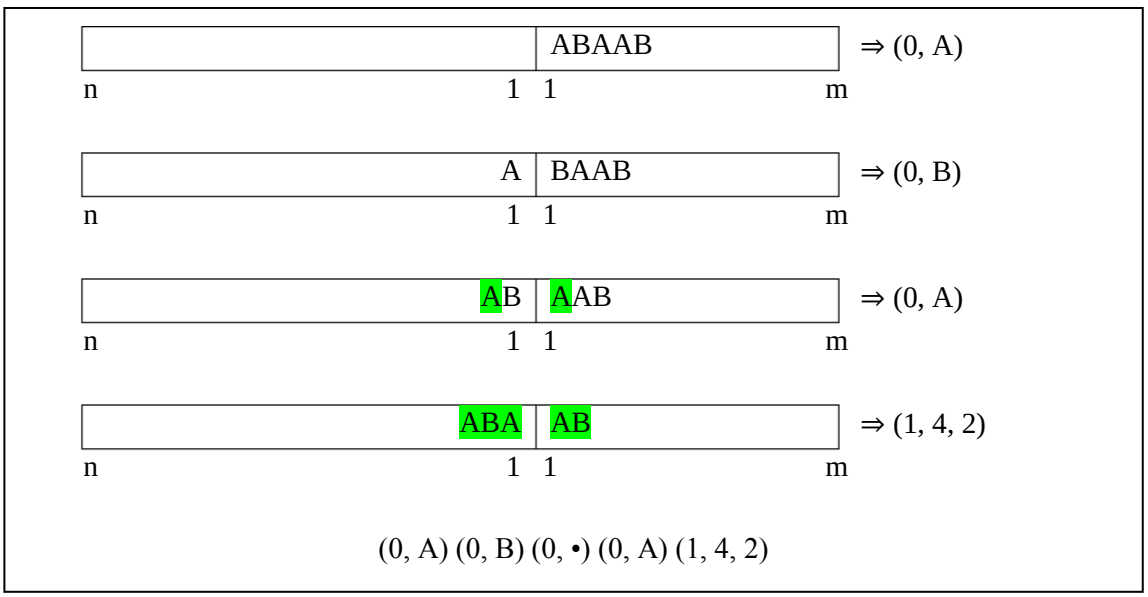
Na obrázku 4.6 vidíme sestavený binární vyhledávací strom. Tento strom není vyvážený a vyhledávání v něm může trvat déle nežli ve vyváženém stromu, který je na obrázku 4.7. Chceme-li vyhledat řetězec “GHIJKLMNOPQRSTUVWXYZ“ tak v prvním stromu musíme projít předchozích šest uzlů, kdežto ve vyváženém nalezneme shodu už ve třetím kroku. Toto opatření sníží čas komprese.



Obrázek 4.7: Vyvážený binární vyhledávací strom

Jestliže dojde k překročení velikosti, tak se musí smazat prvně uložené uzly. Strom musí zůstat ale uspořádaný. Jestliže mazaný uzel měl potomky, tak je musíme začlenit do stromu tak, aby platilo pravidlo, že na levé straně jsou uzly menší a stejné a na pravé pouze větší.

Metoda LZSS ukládá kód stejně jako LZ77. Znaky na 8 bitů a posloupnosti dle potřebných bitů pro uložení daného okna (pozice – velikost vyhledávacího okna, délka – velikost výhledového okna).

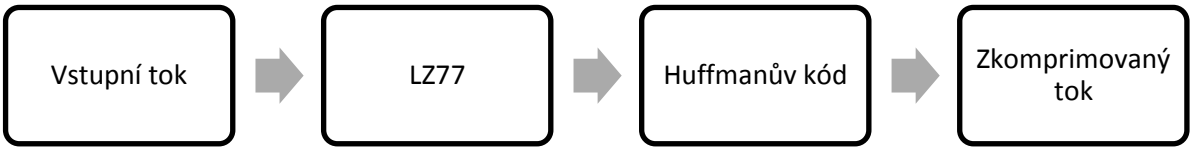


Obrázek 4.8: Princip metody LZSS

4.3 DEFLATE

Stejně jako předchozí dvě metody využívá i DEFLATE [8] posuvné okno. První verzi napsal Phil Katz pro svůj program PKZIP. Tato metoda se stala snad nejpoužívanější a byla základem známých softwarů ZIP, GZIP. Je využita v kompresi obrázků, dokumentů (PNG, PDF) a také v protokolu HTTP. Stala se základem algoritmu LZMA, který je použit v aplikaci 7-Zip.

DEFLATE využívá metodu LZ77 a Huffmanův kód (obrázek č. 13). Stejně jako LZSS nekóduje fráze, které jsou kratší nežli požadovaná délka. U této metody jsou považovány za fráze pouze shody větší než 2 znaky.



Obrázek 4.9: Diagram metody DEFLATE

V metodě LZ77 bylo vyhledávání řetězců neefektivní, proto DEFLATE využívá tabulku, ve které jsou uloženy fráze. Aby bylo vyhledávání rychlé, tak se jako klíč tabulky musí zvolit jedinečná hodnota. Tuto hodnotu získáme tzv. *hashovací funkcí*. Hashovací funkce na základě textu fráze určí pozici v tabulce, na které se bude nacházet informace o pozici dat v posuvném okně. Jak už bylo uvedeno, není výhodné kódovat fráze, které mají velikost menší než tři znaky. Proto se hashovací funkce vypočítá z prvních tří znaků.

Velikost tabulky volíme buď prvočíslo, nebo mocninu čísla 2. Prvočíslo přispívá k rovnoměrnému rozložení hodnot v hashovací tabulce, zatímco mocnina 2 je výhodná pro výpočet operace modulo. Druhá možnost bývá ale častější, protože operaci modulo lze nahradit bitovou operací logického součinu (AND).

V našem případě byla zvolena velikost $2^{15} = 32768$ s minimální délkou fráze 3. Hashovací funkce bude tedy zobrazením prvních tří znaků fráze $z_1z_2z_3$ do intervalu $\langle 2^{15}; -1\rangle$:

$$hash(z_1z_2z_3) \rightarrow \langle 2^{15}; -1\rangle$$

Je mnoho možností jak by mohla funkce vypadat, avšak je výhodné vybrat nějakou, jejíž přepočítání nezabere mnoho času a je jednoduchá. Ve vytvořené knihovně byla použita funkce:

$$hash(z_1z_2z_3) = ((((z_1 * 2^5) \oplus z_2) * 2^5) \oplus z_3) \bmod (2^{15} - 1) \qquad \textbf{(4.2)}$$

Po úpravě:

$$hash(z_1z_2z_3) = ((((z_1 * 2^5) \oplus z_2) * 2^5) \oplus z_3) \circledast (2^{15} - 1) \qquad \textbf{(4.3)}$$

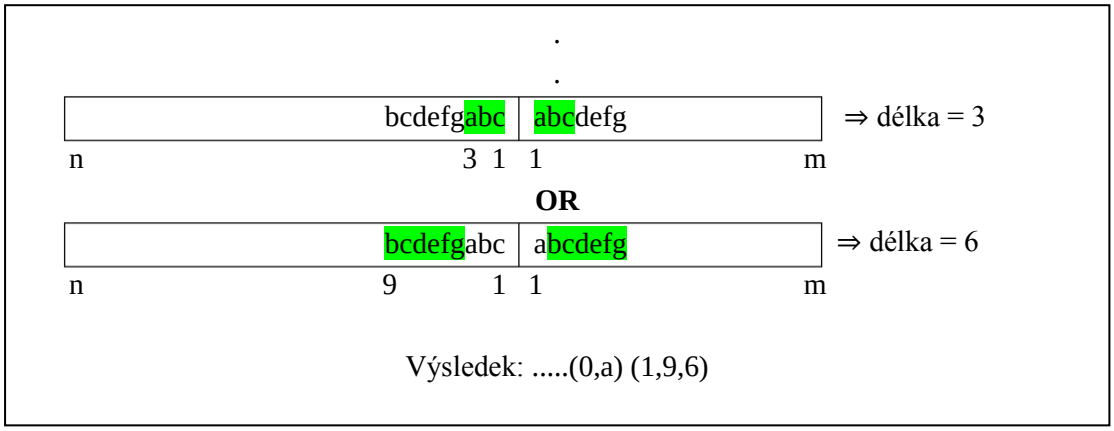
Znak $\oplus$ označuje bitovou operaci nonekvivalence (XOR), $\circledast$ označuje bitovou operaci logického součinu (AND), $*2^5$ označuje bitový posun vlevo o pět bitů (násobení mocninou 2) a *mod* značí operaci modulo. Výhoda uvedené volby hashovací funkce spočívá v možnosti snadného spočítání hodnoty funkce při posunu o jeden znak vpravo na další frázi. Jestliže máme znaky $z_1z_2z_3z_4$, tak

spočítáme hashovací funkcí hodnotu prvních tří znaků (hash($z_1z_2z_3$)). Chceme-li spočítat hodnotu následující, tzn. $z_2z_3z_4$, použijeme výsledek z předchozí funkce a provedeme opět bitový posun vlevo a operaci nonekvivalence:

$$hash(z_2z_3z_4) = ((hash(z_1z_2z_3) * 2^5) \oplus z_4) \circledast (2^{15} - 1) \qquad \textbf{(4.4)}$$

Takto vytvořená tabulka by platila pouze pro unikátní hodnoty. My však máme několik stejných hodnot na různých pozicích, proto nám obyčejná hashovací tabulka nepostačí. Pro naši potřebu je nutné vytvořit tabulku, která nemá jenom jednu hodnotu u klíče, ale obsahuje seznam adres. Seznam je obousměrný. Ze začátku se přidávají nové adresy, potom z konce mažeme staré neaktuální adresy.

Jedna z možností jak dosáhnout nižšího kompresního poměru je, že zvolíme možnost „*lazy matching*“ (obrázek 4.10). Tato možnost nám provede dvě vyhledávání nejdelší společné shody. První je od začátku výhledového okna, a potom to provede se začátkem na následujícím znaku. Kodér pak zjistí, zda byla první shoda vyšší než druhá. Je-li první shoda vyšší, tak si kodér vybere tuto posloupnost a dále ji zpracuje. Ve druhém případě se první znak odešle jako jedna hodnota, posune se okno o jedno doprava a je provedeno přesně to samé co předtím. Takto se postupuje, než se narazí na menší shodu a teprve potom kodér odešle posloupnost. Tento mód dosahuje sice vyšší komprese, ale je zase časově náročnější.



Obrázek 4.10: „lazy matching“

Další atribut ovlivňující míru a rychlost komprese je minimální požadovaná shoda. Je-li nastavena na nulu, tak se provádí „úplná komprese“ (vyhledává se nejdelší možná shoda). Je-li větší jak dva, potom se provede hledání v hashovací tabulce, které vrací první nalezenou možnost, která by odpovídala délce minimální shody nebo by byla vyšší.

Mějme například minimální požadovanou shodu nastavenou na 5. Kodér si vypočítá hashovací funkci z prvních tří znaků ve výhledovém okně. Dle hashovací tabulky najde pozici první shody. Od té pozice provede vyhledání nejdelší shody. Je-li délka shody menší nežli délka požadovaná např.: 4, potom se pokračuje dál a zkoumají se další shody. Jestli by byly délky dalších nalezených shod menší než požadovaná délka, potom se vrací i ta menší jako nejdelší možná shoda (v našem příkladu hodnota 4). Byla by však nalezená shoda vyšší než požadovaná např.: 8, tak se přeruší veškeré hledání a vrací se tato hodnota.

Nyní je již provedené vyhledání v posuvném okně a je také zjištěno, zda se nalezla nebo nenalezla nějaká shodná data. V případě nálezu známe jak pozici, tak délku opakujících se dat.

Dalším krokem metody DEFLATE jak si můžeme všimnout na obrázku 4.8 je převedení našeho výsledku do Huffmanova kódu a následné poslání na výstup.

Metoda DEFLATE používá dvě možnosti jak data zakódovat. Pomocí statického Huffmanova kódování nebo adaptivního Huffmanova kódování. V první variantě se využívají tabulky, které jsou pro jednotlivé délky, pozice a znaky předem známy. Druhá varianta si vytváří vlastní tabulky, které převádí délky, pozice a znaky taky do Huffmanova kódu, ale je zde počítáno ještě s četností výskytu jednotlivých hodnot.

4.3.1 Statický model

Statický model se nemění a má pevně stanovené uspořádání. Nerozlišuje se, zda se jedná o znak nebo o posloupnost a obojí kóduje dle stejné tabulky, která má rozsah ⟨0; 285⟩. Hodnoty 0 až 255 z tohoto intervalu se používají pro označení znaků. Jestliže chceme na výstup poslat zakódovanou frázi, tak použijeme hodnoty v rozmezí od 257 do 285. Zbylá hodnota 256 nám slouží k označení konce bloku dat.

Hodnoty 257 až 285 nám nejen označují, že se jedná o frázi, ale i délku dané fráze. Fráze je dlouhá minimálně 3 až 285 znaků. U kratších délek vyjadřuje číslo přímo délku fráze, jedná se o hodnoty 257 až 264. Např.: 257 → délka 3. Delší fráze jsou potom kódovány čísly od 265 do 285. U těchto hodnot jsou přidány pomocné bity, které nám potom pomohou určit přesnější délku. Kódy, jim přiřazené délky a pomocné bity jsou uvedeny v tabulce 4.2.

Kód	Bity	Délka	Kód	Bity	Délka
257	0	3	272	2	31 - 34
258	0	4	273	3	35 - 42
259	0	5	274	3	43 - 50
260	0	6	275	3	51 - 58
261	0	7	276	3	59 - 66
262	0	8	277	4	67 - 82
263	0	9	278	4	83 - 98
264	0	10	279	4	99 - 114
265	1	11 - 12	280	4	115 - 130
266	1	13 - 14	281	5	131 - 162
267	1	15 - 16	282	5	163 - 194
268	1	17 - 18	283	5	195 - 226
269	2	19 - 22	284	5	227 - 257
270	2	23 - 26	285	0	258
271	2	27 - 30			

Tabulka 4.2: Kódy délek pro DEFLATE

V tabulce 4.2 jsou uvedeny pouze hodnoty, ale tyto „kódy“ (0 až 285) se musí ještě zakódovat. Potom se teprve mohou poslat na výstup. K zakódování nám slouží tabulka 4.3, kde jsou jednotlivým rozmezím přiřazené určité Huffmanovy kódy.

Hodnoty	Kódování	Hodnoty	Kódování
0 - 143	00110000 - 10111111	256 - 279	0000000 - 0010111
144 - 255	110010000 - 111111111	280 - 287	11000000- 11000111

Tabulka 4.3: Intervaly a jejich kódy pro metodu DEFLATE

Pozice nalezené fráze v posuvném okně se kóduje také dvěma hodnotami. Číslem z intervalu ⟨0; 29⟩, které je zakódováno do 5 bitů ($2^5 = 32$; $29 < 32$). Navíc se k tomuto číslu přiřazují pomocné bity, které určují přesnou pozici. Hodnoty 0 až 3 se opět používají bez pomocného bitu a značí nám přesnou pozici. U hodnot 4 až 29 je nutné zpracovat i pomocné bity, abychom určili přesnou polohu. Tyto kódy jsou zobrazeny v tabulce 4.4.

Kód	Bity	Pozice	Kód	Bity	Pozice
0	0	1	15	6	193 - 256
1	0	2	16	7	257 - 384
2	0	3	17	7	385 - 512
3	0	4	18	8	513 – 768
4	1	5 - 6	19	8	769 - 1024
5	1	7 - 8	20	9	1025 - 1536
6	2	9 - 12	21	9	1537 - 2048
7	2	13 - 16	22	10	2049 - 3072
8	3	17 - 24	23	10	3073 - 4096
9	3	25 - 32	24	11	4097 - 6144
10	4	33 - 48	25	11	6145 - 8192
11	4	49 - 64	26	12	8193 - 12288
12	5	65 - 96	27	12	12289 - 16384
13	5	97 - 128	28	13	16382 - 24576
14	6	129 - 192	29	13	24577 - 32768

Tabulka 4.4: Kódy pozic pro DEFLATE

Pro lepší pochopení tohoto kódování se můžete podívat na příklad 4.3 a na příkladu 4.4 vidíte, jak se zakóduje pouhý znak.

Příklad 4.3: Zakódujeme frázi, která byla nalezena na pozici 1 200 a má délku 50 znaků.

Nejprve si zjistíme z tabulky 4.2, že délce 50 znaků odpovídá kód 274. Hodnota 274 dle tabulky 4.3 má 7 bitů a hodnotu 274 zjistíme tak, že od ní odečteme 256. Dostaneme tak číslo 18 (274 – 256 = 18). Toto číslo má binární hodnotu 0010010b. Jenomže hodnota 274 má tři pomocné bity, které určují rozdíl délek. Proto odečteme naši délku a základní délku z tabulky. Rozdíl těchto čísel je 7 (50 – 43 = 7) a tento výsledek je nutné zapsat pomocí těch tří bitů.

Další krok je určit kód pro zakódování pozice. Číslu 1 200 odpovídá dle tabulky 4.4 číslo 20. Rozdíl skutečné pozice a základní z tabulky je 175.

Kód 274	+ 3 bity	kód 20	+ 9 bitů
0010010	111	10100	010101111

Příklad 4.4: Zakódování znaku je velice jednoduché. Mějme například znak „A“.

ASCII hodnota znaku „A“ je 65. V tabulce 4.3 dle této hodnoty nalezneme příslušný kód. „A“ je v rozsahu 0 až143, a proto má kód 8 bitů a je potřeba k 65 přičíst 48. Jejich součet, což je 113, zapíšeme binárně pomocí 8 bitů. Kód, který pošleme na výstup, bude vypadat takto:

01110001

4.3.2 Adaptivní model

Tento model využívá stejné tabulky jako statický model. Znaky jsou kódovány společně s délkou (tabulka 4.2) a pozice využívá také stejnou tabulku (tabulka 4.4). Avšak tabulka 4.5 se nahradí jinou, která nám udává četnost výskytu znaků. Dekodér si tedy vytváří vlastní strom, ve kterém často používané znaky nebo naopak délky jsou zakódovány nejkratším kódem a nejméně používané mají kód delší. Stejně je to s tabulkou pro pozice frází.

Největší problém je jak takový model poslat dekodéru, aby přijatý kód správně dekódoval. Nemůžeme posílat všechna data, protože by byl komprimovaný soubor pak větší, což by vedlo k horšímu kompresnímu poměru. A proto posíláme pouze délky kódu jednotlivých znaků a dekodér si musí sestavit tabulku pro dekódování sám. Nejdůležitější je v tomto případě, aby si dekodér při stejných délkách sestavil tabulku správně a aby neprohodil nějaké znaky. Tento postup je vysvětlen na následujícím příkladu.

Příklad 4.5: Mějme 8 znaků (A, B, C, D, E, F, G, H), které budeme kódovat Huffmanovým kódem.

Krok 1.
Máme osm znaků, proto bude i osm Huffmannových kódů (00, 010, 011, 100, 101, 1100, 1101, 111). Zjistíme si počet kódů na každou délku (tabulka 4.5).

Délka kódu	**Počet kódů**
1	0
2	1
3	5
4	2

Tabulka 4.5: Počty kódů

Krok 2.
Nyní počítáme výskyty znaků v datovém toku. Výsledky jsou v tabulce 4.6.

Znak	A	B	C	D	E	F	G	H
Četnost	10	5	2	3	12	4	5	1

Tabulka 4.6: Výskyt znaků

Krok 3.
Teď přidáme kratším kódům znaky, které se vyskytovali častěji (4.7). Tady nám poslouží tabulka 4.5, ve které jsou délky kódu a jejich počet. Budeme přiřazovat od začátku a budeme se v tabulce postupně snižovat.

Znak	A	B	C	D	E	F	G	H
Délka	3	3	4	3	2	3	3	4

Tabulka 4.7: Výskyt znaků

Krok 4.
Nyní si musíme určit pravidla, která zabrání špatnému dekódování.

Pravidlo: Jestliže budeme mít více znaků se stejnou požadovanou délkou, tak se budou Huffmanovy kódy přiřazovat vzestupně z hodnot znaků.

Bez tohoto pravidla by mohl například znak „A“ dostat kód 010, ale dekodér by znaku „A“ mohl přiřadit kód 011. Tady vidíme, že znak „A“ dostane nižší hodnotu kódu než znak „D“. Přiřazené kódy jednotlivým znakům si můžete prohlédnout a tabulce 4.8.

Znak	**Četnost**	**Délka**	**Kód**
A	10	3	010
B	5	3	011
C	2	4	1100
D	3	3	100
E	12	2	00
F	4	3	101
G	5	3	111
H	1	4	1101

Tabulka 4.8: Přiřazení Huffmanova kódu

4.4 LZX

Metoda LZX [7] je založena na metodě LZ77, která používá statické Huffmanovo kódování a posuvné okno s danou velikostí. Znaky v datech jsou kódovány buď jako nekomprimované znaky, nebo pomocí kódu (offset, délka), což naznačuje, že by měly být znaky určené délky nalezeny v posuvném okně od aktuální pozice vstupního toku dat. Hodnota offsetu je menší, než je velikost vyhledávacího okna. K vyhledávání shodných řetězců byla použita hashovací tabulka, která je popsána v předchozí kapitole.

4.4.1 Opakující offset

LZX rozšiřuje tradiční LZ77 formát několika způsoby, z nichž jeden zahrnuje používání opakujících se offsetů v kódu. Tři poslední shody offsetů, pojmenované jako opakující se kódy offsetů, jsou vyhrazeny k určení shody, která má offset stejný jako jeden ze tří předešlých kódů. Těmto třem offsetům jsou přiřazené hodnoty 0, 1 a 2 (tj. offset 0 znamená, že to je naposled použitý offset v kódu). Všechny zbývající hodnoty offsetů jsou posunuty o tři pozice. Toto posunutí je znázorněno v následující tabulce.

Zakódovaný offset	**Skutečný offset**
0	Naposled použitý offset
1	Druhý naposled použitý offset
2	Třetí naposled použitý offset
3	1
4	2
5	3
6	4
7	5
8	6
500	498
X+2	X
VELIKOST_OKNA-1 (maximální hodnota)	VELIKOST_OKNA-3

Tabulka 4.9: kódování posledních offsetů v metodě LZX

Tyto tři offsety jsou uloženy v seznamu, jejich chování je popsáno níže:

Nechť R0 je definován jako naposled použitý ofset.
Nechť R1 je definován jako druhý naposled použitý offset.
Nechť R2 je definován jako třetí naposled použitý offset.

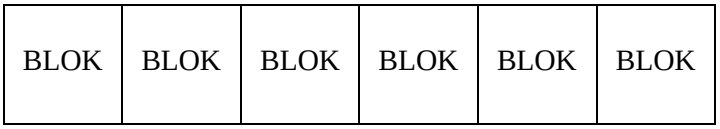
Seznam je řízen podobně jako LRU (least recently used) fronta, s výjimkou případů, kdy R1 nebo R2 je výstup. V těchto případech, které nejsou moc obvyklé, se prostě použitý prvek prohodí s R0, což nevyžaduje tolik operací jako LRU fronta. Počáteční hodnoty prvků R0, R1 a R2 jsou nastaveny na 1. Operaci s těmito prvky je ukázána v tabulce 4.10.

Offset X, který…	Operace
X ≠ R0 && X ≠ R1 && X ≠ R2	R2 ← R1 R1 ← R0 R0 ← X
X = R0	Nic
X = R1	Prohoď R0 ⇔ R1
X = R2	Prohoď R0 ⇔ R2

Tabulka 4.10: Operace s naposled použitými offsety

4.4.2 Formát dat

LZX komprese se skládá ze sledu zkomprimovaných datových bloků, které obsahují hlavičku a potom vlastní data. Hlavička udává, zda se jedná o komprimovaný tok, nebo zda je tok nekomprimovaný. Složení hlaviček je popsáno níže.



Obrázek 4.10: Struktura výstupního datového toku u LZX

0	Nedefinováno
1	Přesný blok
2	Zarovnaný blok
3	Nekomprimovaný blok
4 – 7	Nedefinováno

Tabulka 4.11: 3bitové číslo pro různé typy bloků

Nekomprimovaný blok

Nekomprimovaný blok začíná 1 až 16 bity s nulami, které zarovnávají datový tok pro 16bitové rozhraní. V tomto momentě končí bitové posílání dat a posílá se po bytech. Následujících 12 bytů obsahuje hodnoty R0, R1 a R2. Potom následuje proud bytů nekomprimovaných dat.

1 – 16 bitů	4 byty	4 byty	4 byty	n bytů
Zarovnání nulami	R0	R1	R2	Nekomprimovaná data

Tabulka 4.12: Struktura nekomprimovaného bloku dat

Přesný blok

Vstup	Poznámka	Velikost
Počet nekomprimovaných znaků	Rozsah od 1 do 2^{24}	24 bitů
Pre-tree pro prvních 256 prvků z main tree	20 prvků po 4 bitech	80 bitů
Cesta délek pro prvních 256 prvků z main tree	Kódováno pomocí pre-tree	Proměnná
Pre-tree pro zbývající prvky z main tree	20 prvků po 4 bitech	80 bitů
Cesta délek pro zbývající prvky z main tree	Kódováno pomocí pre-tree	Proměnná
Pre-tree pro length tree	20 prvků po 4 bitech	80 bitů
Cesta délek pro prvky z length tree	Kódováno pomocí pre-tree	Proměnná
Kódovaná data		Proměnná

Tabulka 4.13: Struktura přesného bloku dat

Zarovnaný blok

Vstup	Poznámka	Velikost
Počet nekomprimovaných znaků	Rozsah od 1 do 2^{24}	24 bitů
Pre-tree pro prvních 256 prvků z main tree	20 prvků po 4 bitech	80 bitů
Délka cest pro prvních 256 prvků z main tree	Kódováno pomocí pre-tree	Proměnná
Pre-tree pro zbývající prvky z main tree	20 prvků po 4 bitech	80 bitů
Délka cest pro zbývající prvky z main tree	Kódováno pomocí pre-tree	Proměnná
Pre-tree pro length tree	20 prvků po 4 bitech	80 bitů
Délka cest pro prvky z length tree	Kódováno pomocí pre-tree	Proměnná
Zarovnaný offset tree	8 prvků po 3 bitech	
Kódovaná data		Proměnná

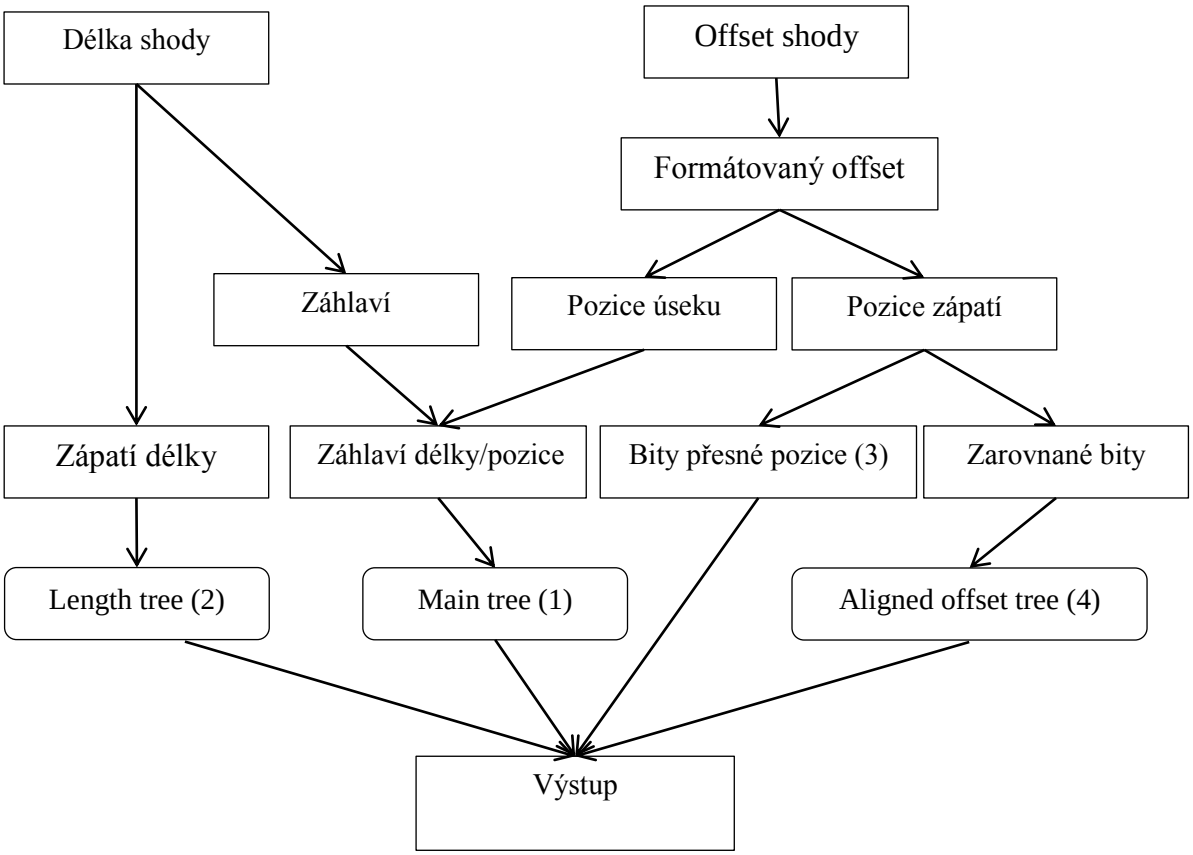
Tabulka 4.14: Struktura zarovnaného bloku dat

Offset tree obsahuje pouze 8 prvků, z nichž každý je zakódován jako 3bitová délka cesty. Protože velikost tohoto stromu je tak malá, není potřeba další komprese.

4.4.3 Kódování tree a pre-tree

Všechny stromy použité v LZX jsou tvořeny formou kanonického Huffmanova stromu, délka cest každého prvku ve stromu je tedy dostatečná k rekonstrukci původního stromu. Main tree je zakódován do dvou částí, jako by to byly dva samostatné stromy. První strom odpovídá prvním 256 prvkům stromu (nekomprimované znaky), a druhý strom odpovídá zbývajícím prvkům (shody).
Jelikož jsou stromy výstupem několikrát během komprese, LZX optimalizuje kompresi kódováním pouze rozdílu délky předcházející a délky aktuální. Jedná-li se o první strom, potom se jako přechozí délka cesty bere hodnota 0.
Každý prvek může mít délku cesty 0 až 16 (včetně) u nichž nulová délka cesty naznačuje, že má prvek nulovou frekvenci, a tudíž se ve stromě nenachází. Prvky stromu jsou na výstup posílány v postupném pořadí od prvního prvku. Jestliže následuje několik délek cest, které jsou stejné, potom

se provede kódování délky, jinak se provede rozdíl mezi délkami cest modulo 17. Je důležité, aby při dekompresi souhlasil počet nekomprimovaných znaků z hlavičky s celkovým počtem dekomprimovaných znaků.



Obrázek 4.11 : Zpracování offsetů délek v LZX

Na obrázku je znázorněn diagram zpracování shod. Nyní když máme nalezený offset, převedeme ho pomocí pseudokódu 4.1 na formátovaný offset.

```
if offset == R0 then
        formátovaný_offset ← 0
else if offset == R1 then
        formátovaný_offset ← 1
else if offset == R2 then
        formátovaný_offset ← 2
else
        formátovaný_offset ← offset + 2
endif
```

Pseudokód 4.1: Převod nalezeného offsetu na formátovaný offset

Dále se formátovaný offset dělí na sloty a zápatí. Sloty definují nejvýznamnější bity offsetu, které udávají základ polohy. Zápatí potom definuje méně významné bity, pomocí kterých lze získat přesnou pozici. V pseudokódu 4.2 je zobrazeno vypočítání slotu a zápatí.

Slot	Bity zápatí	Pozice	Kód	Bity	Pozice
0	0	0	15	6	192 - 255
1	0	1	16	7	256 - 383
2	0	2	17	7	384 - 511
3	0	3	18	8	512 – 767
4	1	4 - 5	19	8	768 - 1023
5	1	6 - 7	20	9	1024 - 1535
6	2	8 - 11	21	9	1536 - 2047
7	2	12 - 15	22	10	2048 - 3071
8	3	16 - 23	23	10	3072 - 4095
9	3	24 - 31	24	11	4096 - 6143
10	4	32 - 47	25	11	6144 - 8191
11	4	48 - 63	26	12	8192 - 12287
12	5	64 - 95	27	12	12288 - 16383
13	5	96 - 127	28	13	16381 - 24575
14	6	128 - 191	29	13	24576 - 32767

Tabulka 4.15: Offsety a pomocné bity pro metodu LZX

```
slot ← formátovaný_offset
zápatí_bity ← bity_zápatí[ slot ]
if zápatí_bity > 0
        pozice_zápatí ← formátovaný_offset & ((2^ zápatí_bity) - 1)
else
        pozice_zápatí ← null
```

Pseudokód 4.2: Výpočet slotu a zápatí offsetu

Pokud je aktuální blok zarovnaný, pak spodní 3 bity v zápatí jsou posunuty o 3 bity vpravo. Toto se provede pouze, je-li zápatí veliké alespoň ty 3 bity. Následující pseudokód popisuje tuto situaci.

```
if block_type = aligned_offset_block then
        if formátovaný_offset <= 15 then
                verbatim_bits ← bity_zápatí
                aligned_offset ← null
        else
                aligned_offset ← bity_zápatí
                verbatim_bits ← bity_zápatí >> 3
        endif
else
        verbatim_bits ← bity_zápatí
        aligned_offset ← null
endif
```

Pseudokód 4.3: Výpočet slotu a zápatí offsetu při zarovnaném bloku

Délka shody je také zapsána do dvou hodnot. Záhlaví nabývá hodnot 0 až 7 (včetně), což označuje délku shody 2, 3, 4, 5, 6, 7, 8 nebo délku větší nežli 8. Jestliže je délka menší nežli 8, potom se nepřidávají žádné bity ze zápatí. Jinak je hodnota zápatí délky rovna skutečné délce mínus 9.

Nalezená délka	Délka záhlaví	Délka zápatí
2	0	NIC
3	1	NIC
4	2	NIC
5	3	NIC
6	4	NIC
7	5	NIC
8	6	NIC
9	7	0
257 (MAX)	7	248 (257 - 9)

Tabulka 4.16: Záhlaví a zápatí délky shody

Následující pseudokód ukazuje jak dostat z těchto hodnot záhlaví délky, zápatí délky a záhlaví délky/pozice.

```
if nalezená_délka <= 8
        délka_záhlaví ← nalezená_délka - 2
        délka_zápatí ← null
else
        délka_záhlaví ← 7
        délka_zápatí ← nalezená_délka - 9
endif
délka/pozice_záhlaví ← (slot << 3) + délka_záhlaví
```

Pseudokód 4.4: Výpočet slotu a zápatí offsetu při zarovnaném bloku

Shoda se nakonec pošle na výstup ve čtyřech prvcích:

1. Výstupní prvek (délka/pozice_záhlaví + 256) z main tree
2. Pokud délka_zápatí ≠ null, pak pošli na výstup prvek délka_zápatí z length tree
3. Pokud verbatim_bits ≠ null, pak pošli na výstup verbatim_bits
4. Pokud aligned_offset_bits ≠ null, pak pošli na výstup prvek aligned_offset z aligned offset tree

4.5 LZ78

Metoda LZ78 [4] vytváří slovník v podobě stromu, ve kterém vyhledává nejdelší existující frázi a vytvoří novou přidáním následujícího znaku. Důležité je, že nejdříve nalezneme frázi ve slovníku, potom ji zakóduje a na konec teprve vytvoříme novou frázi. Jestliže by to proběhlo obráceně, tak by nebylo možné data dekomprimovat. Princip kódování je vysvětlen na příkladu 4.6.

Slovník začíná prázdný a postupně se zaplňuje. Tato metoda je rychlejší než u pohyblivého okna, ale slovním má tendenci růst (přidávání nových frází), a tudíž je to paměťově náročné. Tento problém se v praxi dá řešit několika způsoby. Můžeme celý slovník zmrazit, jakmile překročí stanovenou velikost. To znamená, že při dosažení uzlu s hodnotou třeba 30 000 už do slovníku nic nepřidáváme, ale pouze provádíme vyhledávání. Další možností je celý slovník smazat a začít vytvářet nový. Ale nejlepší možností, která je i nejsložitější, by bylo vymazávat ze slovníku fráze, které jsou nejméně používané.

LZ78 je metoda jednoprůchodová, adaptivní a symetrická. V tabulce 4.17 je zapsáno kódování a na obrázku 4.17 je toto kódování převedeno do stromu (výsledný strom pro vzorový řetězec).

Vstupní text	Výstup	Číslo ve slovníku	Fráze ve slovníku
Ahoooj•tak•jak?	(0,A)	1	A
hoooj•tak•jak?	(0,h)	2	h
oooj•tak•jak?	(0,o)	3	o
ooj•tak•jak?	(3,o)	4	oo
j•tak•jak?	(0,j)	5	j
•tak•jak?	(0, •)	6	•
tak•jak?	(0, t)	7	t
ak•jak?	(0,a)	8	a
k•jak?	(0,k)	9	k
•jak?	(6,j)	10	•j
ak?	(8,k)	11	ak
?	(0,?)	12	?

Tabulka 4.17: Tabulka LZ78 komprese

Příklad 4.6:

Pro každý krok platí, že načítáme ze vstupu, dokud se znaky nachází ve slovníku. Jestliže přijde znak, který ve slovníku není, tak přestaneme načítat. Příklad je vytvořen pro řetězec z tabulky 4.17.

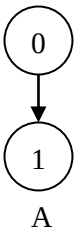
Inicializace slovníku

Slovník je prázdný a obsahuje pouze kořenový uzel s hodnotou 0.

Obrázek 4.12: Inicializovaný strom

Krok 1.

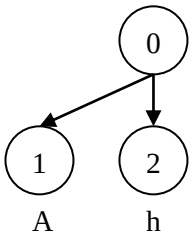
Načteme „A“, které není ve slovníku. Pošleme na výstup tedy: (0, A). Přidáme znak do slovníku.



Obrázek 4.13: 1. krok metody LZ78

Krok 2.

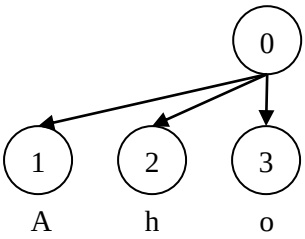
Načteme „h“, které není ve slovníku. Pošleme na výstup tedy: (0, h). Přidáme znak do slovníku.



Obrázek 4.14: 2. krok metody LZ78

Krok 3.

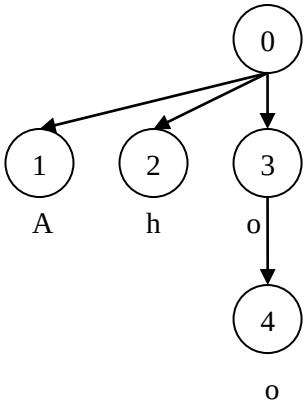
Načteme „o“, které není ve slovníku. Pošleme na výstup tedy: (0, o). Přidáme znak do slovníku.



Obrázek 4.15: 3. krok metody LZ78

Krok 4.

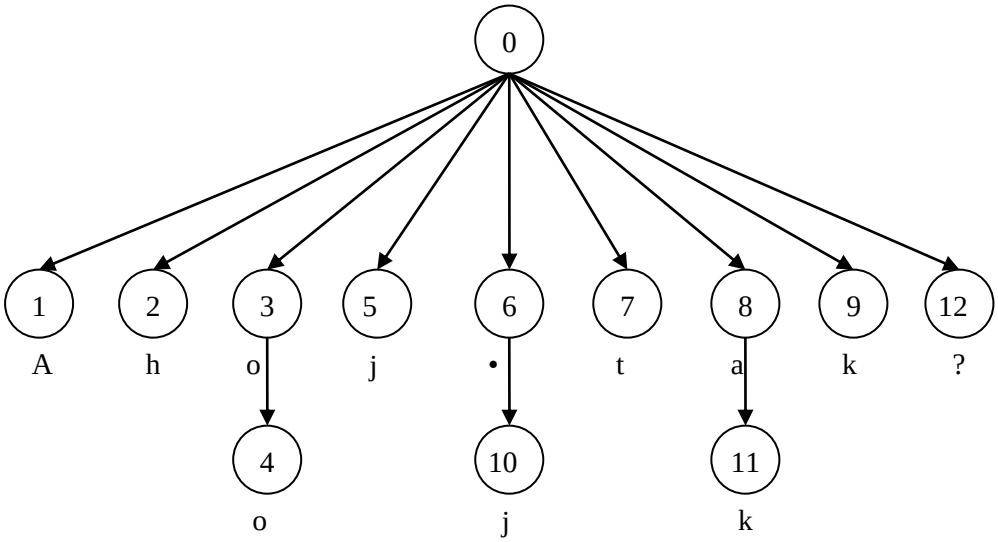
Načteme „o“, které je ve slovníku, proto načteme další znak. Načetli jsme opět „o“, protože jsme v uzlu č. 3, tak ten nemá žádného potomka, proto se znak „o“ už ve slovníku nevyskytuje. Pošleme na výstup tedy: (3, o). Přidáme znak do slovníku.



Obrázek 4.16: 4. krok metody LZ78

Výsledek

Pro pochopení principu této metody by měly první čtyři kroky postačit. Další postup by byl stejný jako v předchozích krocích. Na obrázku 4.17 vidíte výsledný strom sestavený ze vzorového řetězce.



Obrázek 4.17: Výsledný strom metody LZ78

4.6 LZW

Metoda LZW [6] vychází z LZ78, ale posílá na výstup pouze číslo fráze ve slovníku a vynechává následující znak. Proto se musí slovník před použitím inicializovat, tzn. nahrát všechny hodnoty ASCII tabulky (zde záleží na množině použitých znaků). Díky tomuto opatření se na výstup posílá méně bitů a kompresní poměr je nižší než u LZ78. Pro odeslání kódu je potřeba tolik bitů, aby pokrylo hodnoty ve slovníku. Jestliže máme 300 hodnot ve slovníku je potřeba 9 bitů ($2^9 = 512$). Dekodér si potom sám počítá, kolik bitu musí načíst, aby tok správně dekódoval.

Stejně jako u metody LZ78 dochází i zde k problémům s pamětí. Pro jejich odstranění se využívá stejných způsobů jako u LZ78. Tato metoda je jednoprůchodová, adaptivní a symetrická. Její použití vidíte v tabulce 4.18 a podrobně je vysvětleno na příkladu 4.7.

Vstupní text	Výstup	Číslo ve slovníku	Fráze ve slovníku
Inicializace slovníku			
Ahoooj•tak•jak?	65	256	Ah
hoooj•tak•jak?	104	257	ho
oooj•tak•jak?	111	258	oo
ooj •tak•jak?	258	259	ooj
j •tak•jak?	106	260	j•
•tak•jak?	32	261	•t
tak•jak?	116	262	ta
ak•jak?	97	263	ak
k•jak?	107	264	k•
•jak?	32	265	•j
jak?	106	266	ja
ak?	263	267	ak?
?	63		

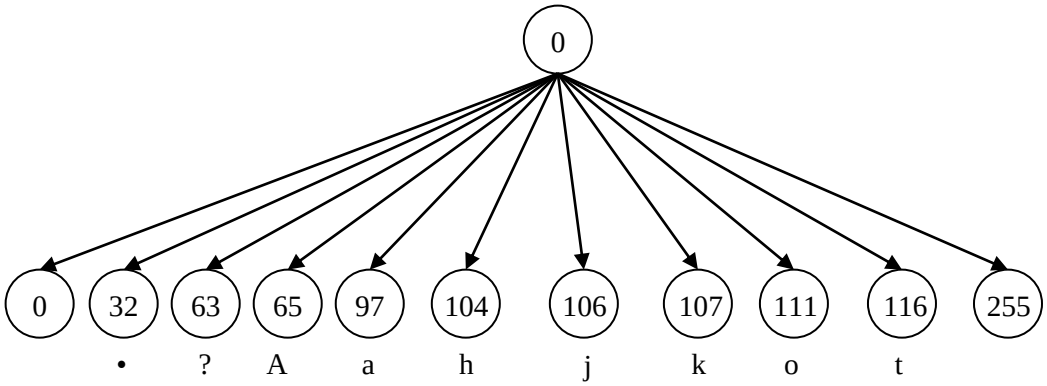
Tabulka 4.18: Tabulka LZW komprese

Příklad 4.7:

Pro každý krok platí, že načítáme ze vstupu, dokud se znaky nachází ve slovníku. Jestliže přijde znak, který ve slovníku není, tak přestaneme načítat. Po přidání znaku do slovníku musíme ale nově přidaný znak znovu zpracovat, jako by byl teprve načtený. Příklad je vytvořen pro řetězec z tabulky 4.18.

Inicializace slovníku

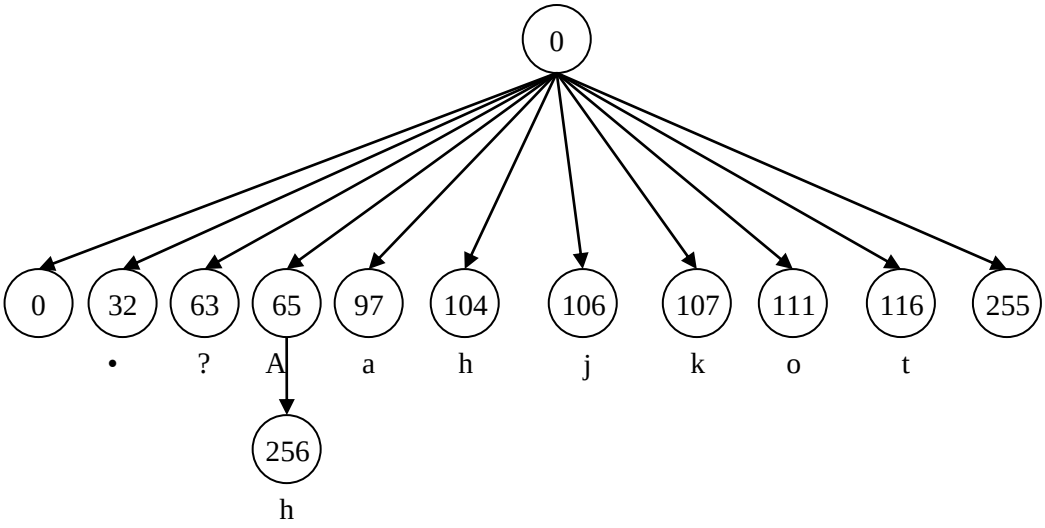
Slovník obsahuje kořenový uzel s hodnotou 0 a jeho potomky s hodnotami použité abecedy. V našem přikladu je jako abeceda použita ASCII tabulka hodnot.



Obrázek 4.18: Inicializovaný strom

Krok 1.

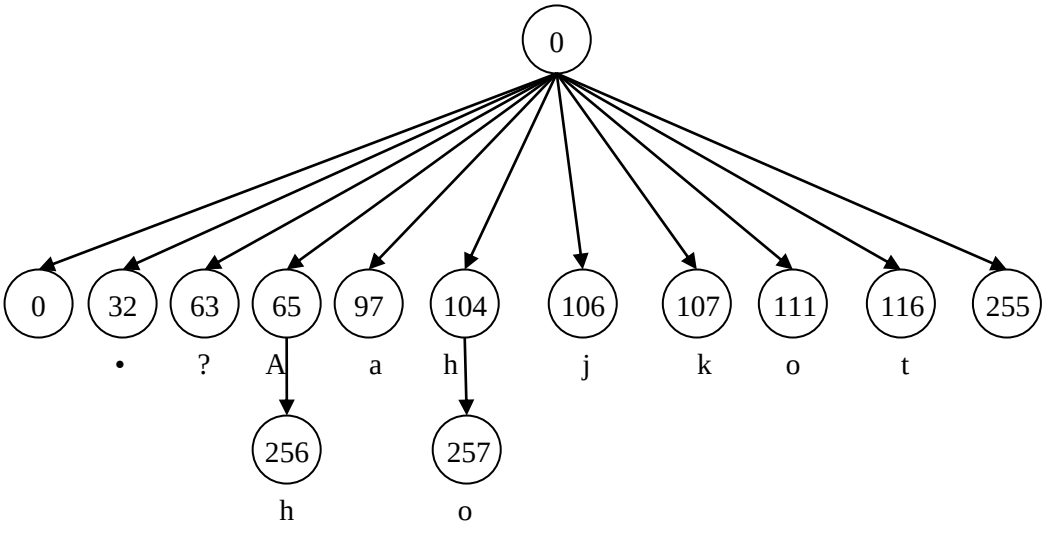
Načteme „A“, které je slovníku, proto načteme další znak „h“. Prohledáme potomky „A“ a zjistíme, že „h“ tam není. Pošleme na výstup tedy: (65). Přidáme znak do slovníku.



Obrázek 4.19: 1. krok metody LZ78

Krok 2.

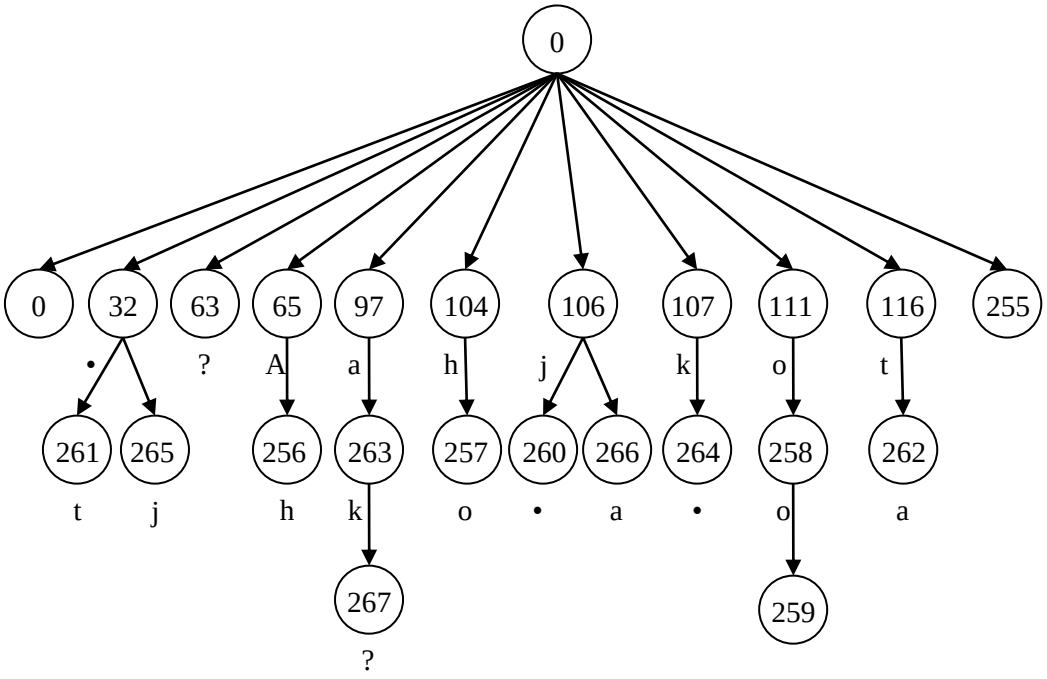
Načteme „h“, které je slovníku, proto načteme další znak „o“. Prohledáme potomky „h“ a zjistíme, že „o“ tam není. Pošleme na výstup tedy: (104). Přidáme znak do slovníku.



Obrázek 4.20: 2. krok metody LZ78

Výsledek

Pro pochopení principu této metody by prvních pár kroků postačit. Další postup by byl stejný jako v předchozích krocích. Na obrázku 4.21 vidíte výsledný strom sestavený ze vzorového řetězce.



Obrázek 4.21: Výsledný strom metody LZW

5 Testování

V dnešní době jsou kompresní metody velice různorodé, proto se na jejich testování používají tzv. *korpusy*. Korpus je zde chápán jako kolekce několika neměnných souborů, které jsou sestavené právě pro tento účel. Hlavním výsledkem komprese je dosáhnout co nejmenšího kompresního poměru, který byl popsán v kapitole 2.3. Pro testování byl použit Silensia [9] corpus a společně byly testovány podobné metody.

5.1 Silesia corpus

Tento korpus sestavil v roce 2003 Sebastian Deorowicz. Hlavním důvodem byla nedostačující velikost již známých korpusů (Calgary, Canterbury) a také nedostačující typ dat. V následující tabulce jsou vypsány soubory z korpusu a informace o nich.

Název souboru	Popis	Typ	Zdroj	Velikost [B]
dickens	Sbírka prací Charlese Dickense	Anglický text	Project Gutenberg	10,192,446
mozilla	Mozilla 1.0 (UNIX edition)	exe	Mozilla Project	51,220,480
mr	Obrázek magnetické rozonance	obrázek	Nemocniční obrázek	9,970,564
nci	Chemická databáze struktur	databáze	CACTVS Chemical Information Services at LMC/NCI	33,553,445
ooffice	dll z Open Office.org 1.01	exe	Open Office	6,152,192
osdb	Vzorová databáze v MySQL z Open Source Database Benchmark	databáze	Open Source Database Bechmark Project	10,085,684
reymond	Text knihy Chłopi od Władysław Reymont	Polské pdf	Virtual Library of Polish Literature	6,627,202
Samba	Zdrojový kód Samby 2-2.3	src	Samba Project	21,606,400
Sao	The SAO star catalog	bin data	Astronomical Catalogs and Catalog Formats	7,458,703
Webster	The 1913 Webster Unabridged Dictionary	html	Project Gutenberg	41,458,703
Xml	Sbírka XML souborů	html	XMLPPM: XML – Conscious PPM Compression	5,345,280

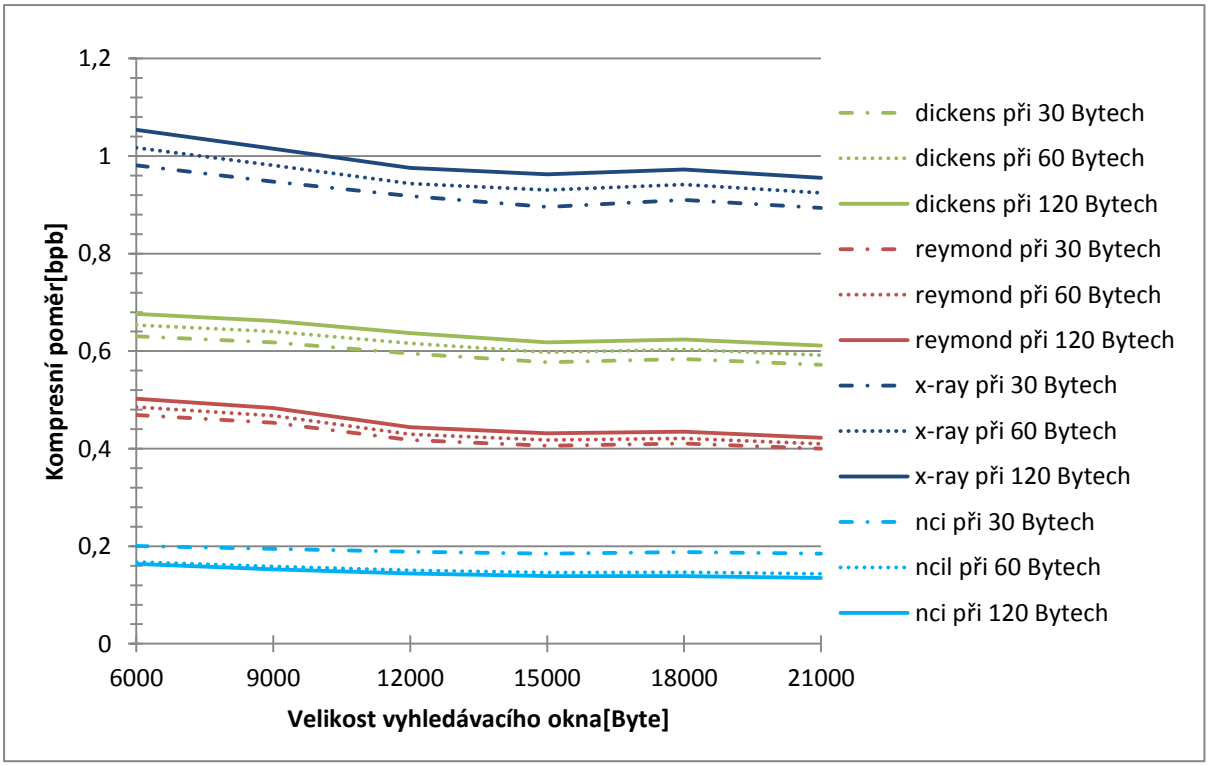
x-ray	Rengenový obrázek	obrázek	Nemocniční obrázek	8,474,240

Tabulka 5.1: Tabulka informací o souborech v Silesia korpusu

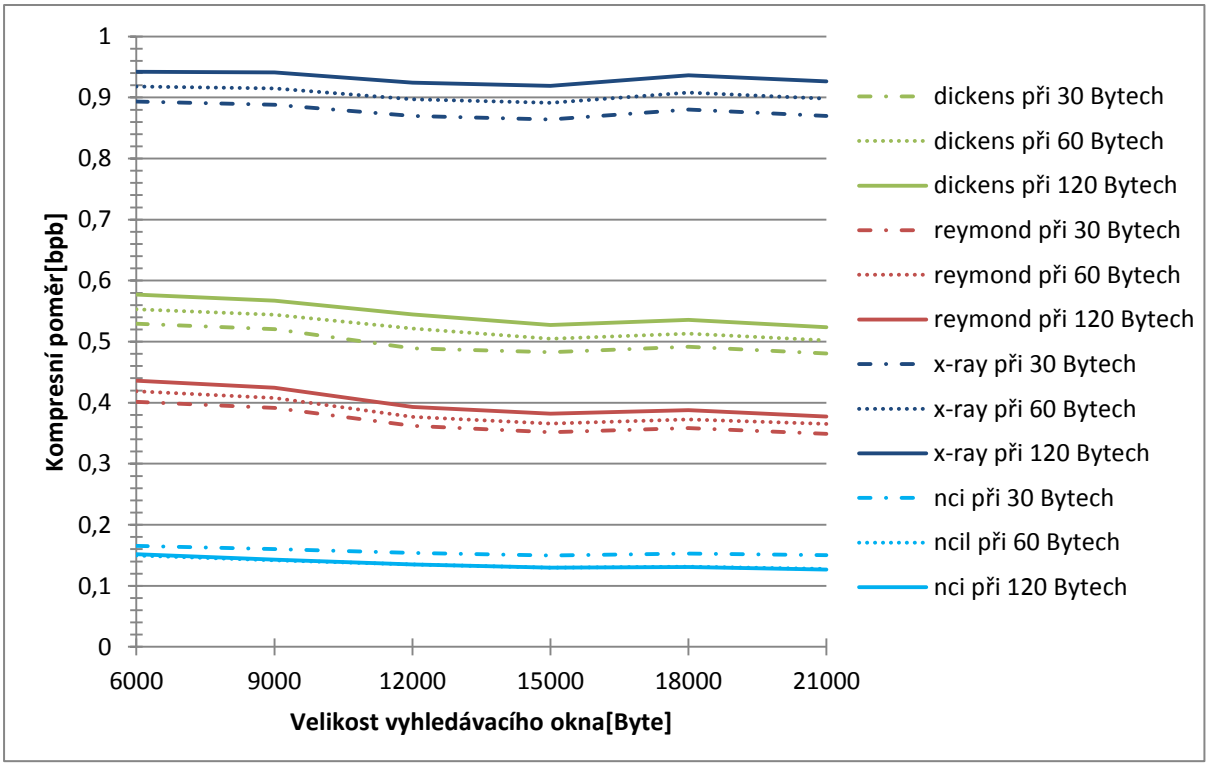
5.2 Metody LZ77 a LZSS

Obě tyto metody jsou si velice podobné. Metoda LZ77 využívá buffery k realizaci posuvného okna a LZSS využívá taktéž buffery, které jsou kruhové, navíc vyhledávání je realizováno pomocí binárního vyhledávacího stromu. U obou metod je možné tedy zadat dva parametry. První parametr je velikost vyhledávacího okna a druhý je velikost výhledového okna. V testech byly použity velikosti pro vyhledávací okno: 6 000 B, 9 000 B, 12 000 B, 15 000 B, 18 000 B a 21 000 B. Velikosti výhledového okna jsou tyto: 30 B, 60 B a 120 B.

V následujících dvou grafech jsou vyznačeny kompresní poměry, které odpovídají jednotlivým velikostem výhledového okna. V grafech 5.1 a 5.2 vidíme, že větší hodnoty než 30 pro výhledové okno zmenšují kompresní poměr. V každém souboru jsou totiž posílány velikosti shodných řetězců pomocí bitů. Počet těchto bitů je proměnný. Na velikost 30 je potřeba pouze 5 bitů, kdežto na velikosti 60 a 120 je už potřeba bitů 6 a 7. V datech se větší velikosti objevují méně nežli velké, a proto je kompresní poměr u většiny souborů lepší při maximální velikosti 30 bytů. Další testy jsou právě proto provedeny při velikosti výhledového okna 30 bytů. Grafy nejsou tvořeny všemi soubory z korpusu, ale jsou tam vyneseny jen některé, aby se dosáhlo lepší přehlednosti.



Graf 5.1: Rozdíl kompresí metody LZ77 při různých velikostí výhledového okna



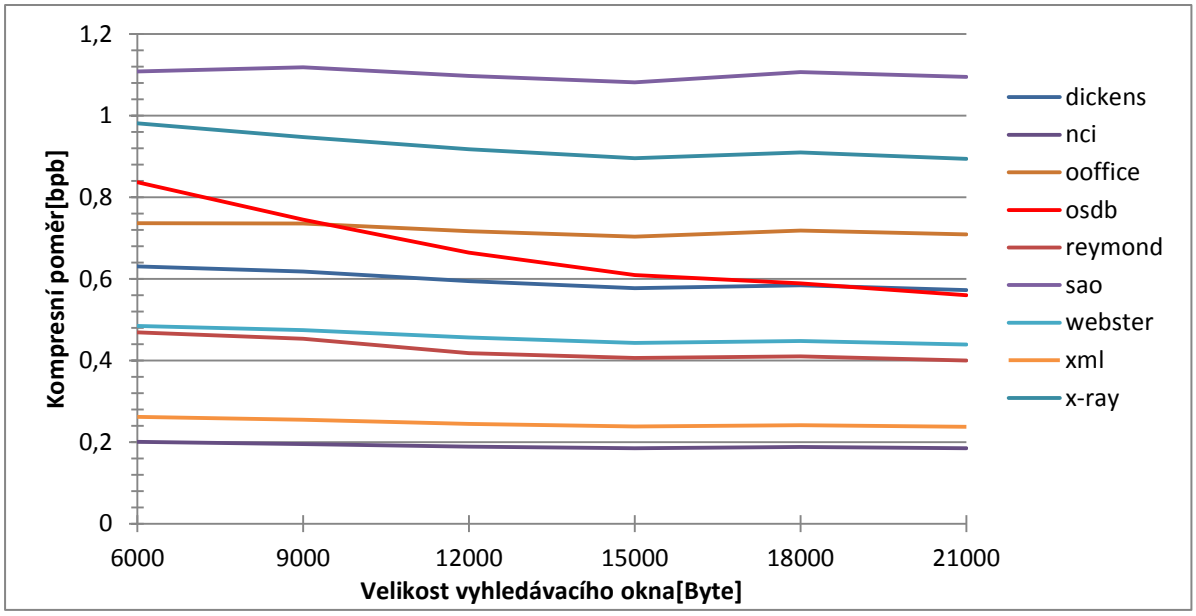
Graf 5.2: Rozdíl kompresí metody LZSS při různých velikostí výhledového okna

Grafy 5.3 a 5.4 udávají kompresní poměr dosažený metodami LZ77 a LZSS při velikosti výhledového okna 30 bytů. Z grafů vyplývá, že metoda LZSS má lepší kompresní poměr než metoda LZ77. Toho bylo dosaženo změnou formátu výstupního toku. Metoda LZ77 ukládala na výstup trojici (offset, délka, následující znak) a to i když se jednalo pouze o znak nebo i komprimovaný tok. Naproti tomu LZSS posílá na výstup pouze bit, který určuje znak nebo posloupnost, společně s kódem, a proto zabere méně bitů. Tento rozdíl je vysvětlen v následujícím příkladu.

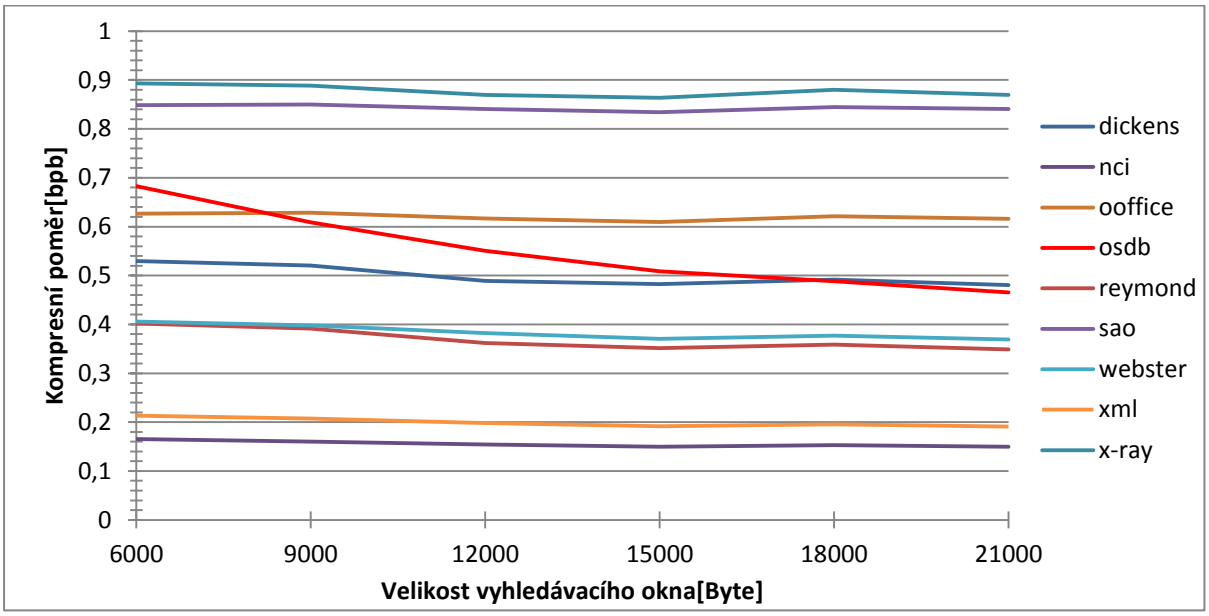
Příklad 5.1: Mějme výhledové okno na 30 bytů, vyhledávací okno je pro 21 000 bytů.

LZ77
Kód znaku (0, 0, a): 15 bitů + 5 bitů + 8 bitů = 28 bitů
Kód posloupnosti (20, 5, a): 15 bitů + 5 bitů + 8 bitů = 28 bitů

LZSS
Kód znaku (0, a): 1 bit + 8 bitů = 9 bitů
Kód posloupnosti (1, 20, 5): 1 bit + 15 bitů + 5bitů = 21 bitů

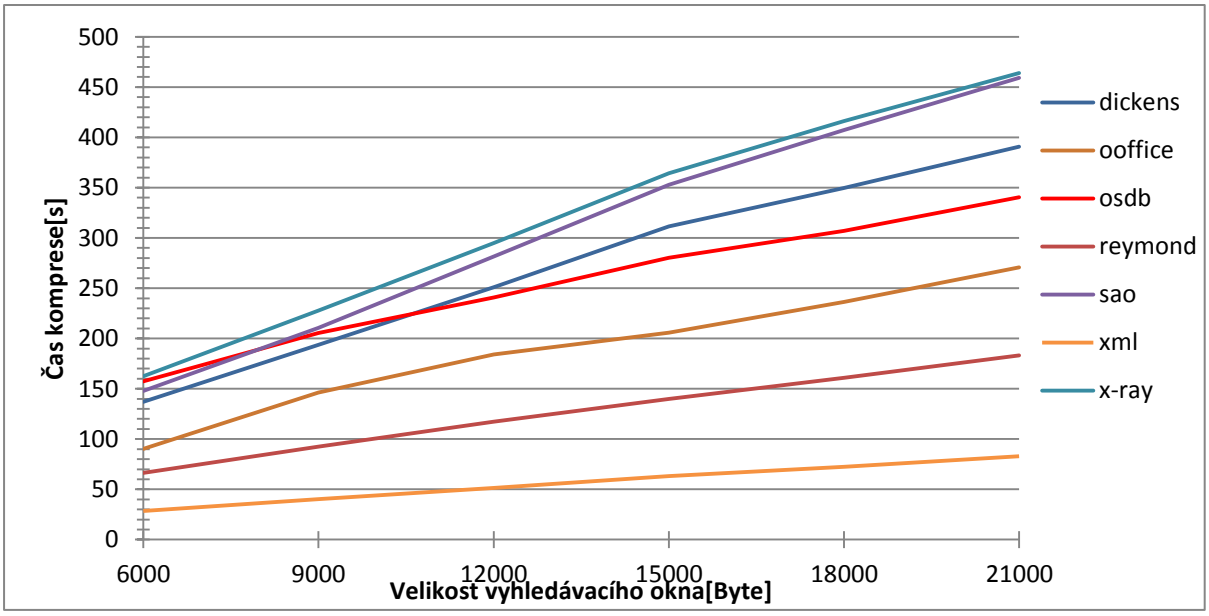


Graf 5.3: Kompresní poměr metody LZ77 při 30 B velikosti výhledového okna

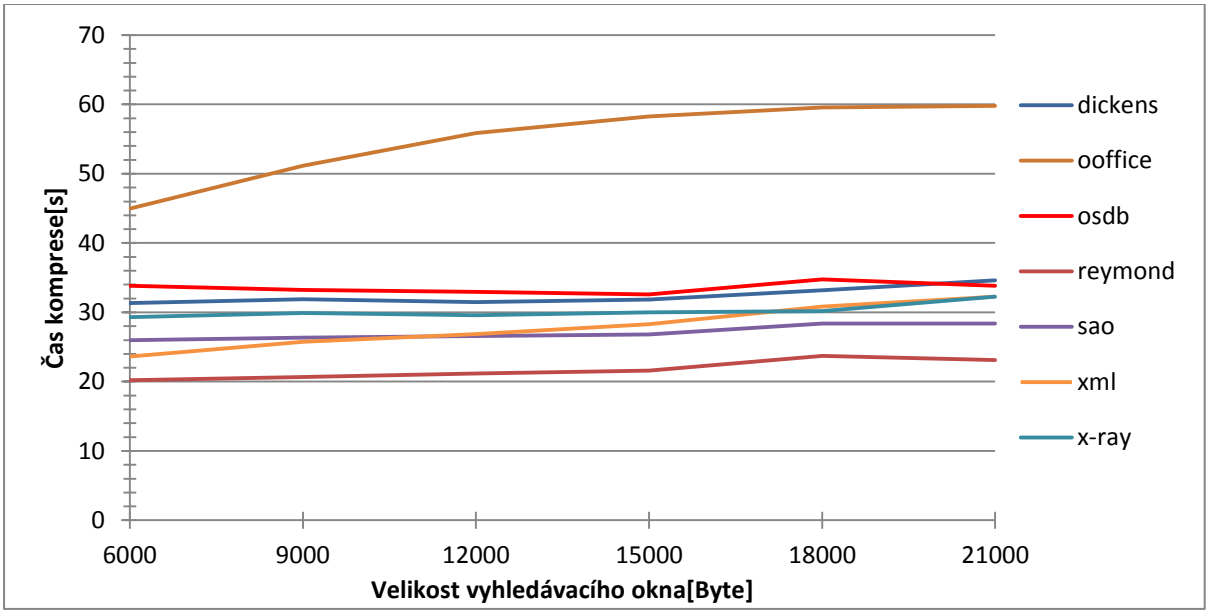


Graf 5.4: Kompresní poměr metody LZSS při 30 B velikosti výhledového okna

Na grafech 5.5 a 5.6 je znázorněn čas, který byl potřeba pro kompresi jednotlivých souborů. Časové výsledky jsou u těchto metod poněkud vyšší, protože by bylo potřeba tyto metody ještě optimalizovat. U metody LZ77 se vyhledává ve dvou buffrech nejdelší společný prefix. Optimalizací tohoto vyhledávání by bylo možné dosáhnout lepších časových výsledků. V metodě LZSS je použit pro vyhledávání binární vyhledávací strom. Opět by bylo možné dosáhnout lepších časů komprese za použití některých vylepšení tohoto stromu (např.: přestavovat strom aby byl vyvážený). Jinak je z grafů patrné, že metoda LZSS je rychlejší než metoda LZ77.

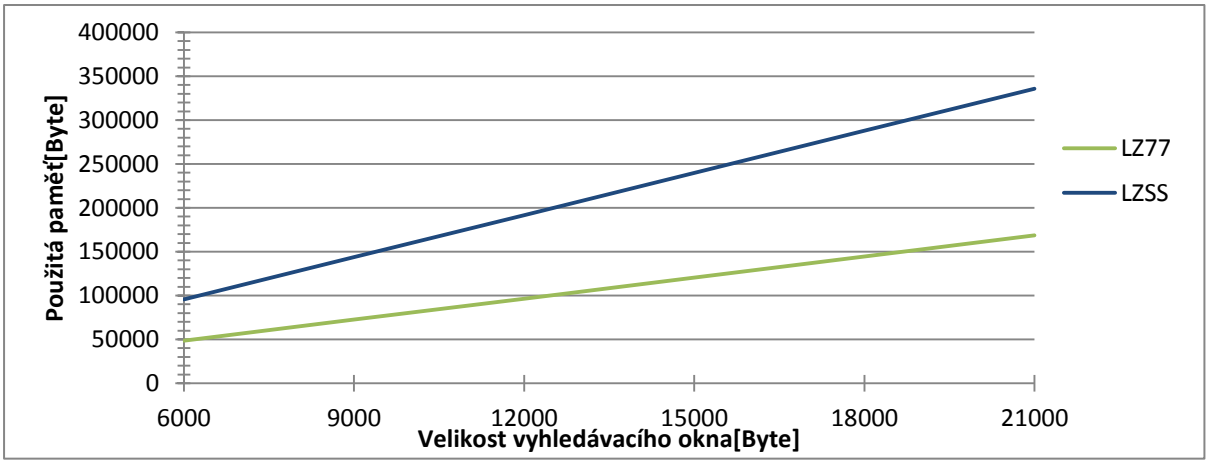


Graf 5.5: Čas komprese metodou LZ77



Graf 5.6: Čas komprese metodou LZSS

Jako poslední výsledek dosažený při testování těchto metod je maximální použitá paměť při kompresi. V metodě LZSS je vytvářen strom, ve kterém se nachází jednotlivé řetězce pro porovnávání, proto je využitá paměť vyšší než u metody LZ77.

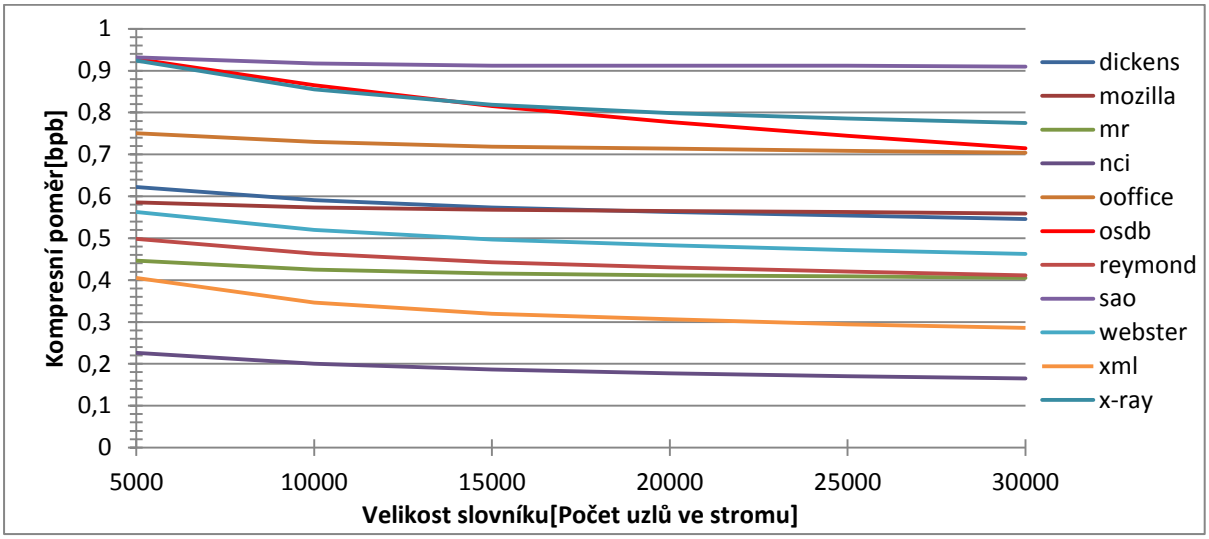


Graf 5.7: Použitá paměť aplikace u metod LZ77 a LZSS

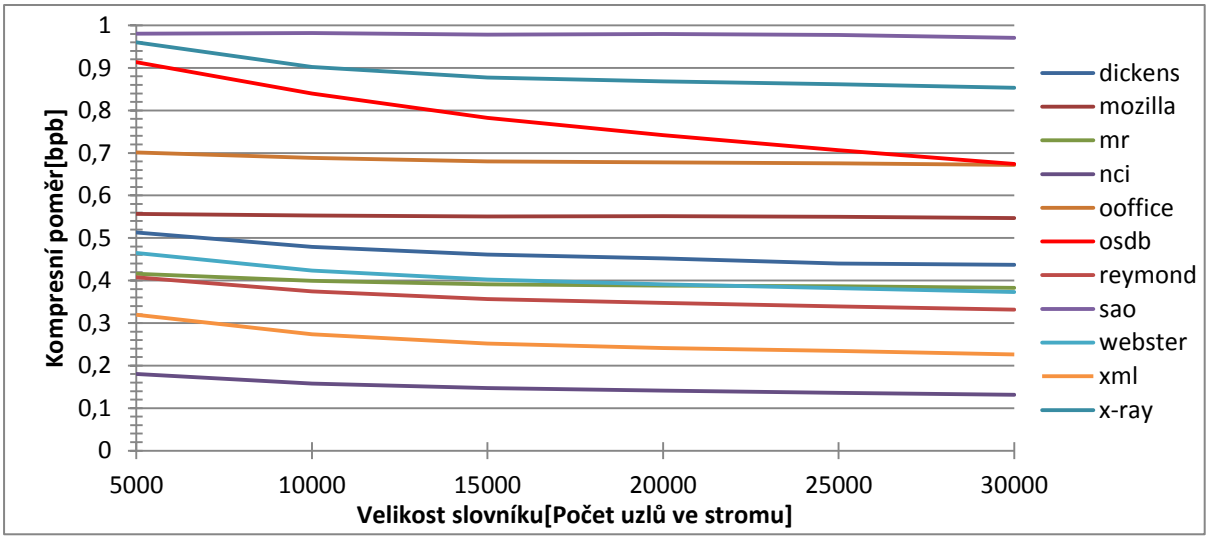
5.3 Metody LZ78 a LZW

Tyto metody nepoužívají žádné posuvné okno, ale vytvářejí si slovník již použitých znaků, ve kterém vyhledávají nejdelší možnou shodu. Princip zde uveden není, protože je vysvětlen výše. Jenom je třeba vědět rozdíl mezi těmato dvěma metodami, který spočívá pouze v tom, že LZ78 si vytváří celý slovník od začátku a LZW už má ve slovníku použitou abecedu (v tomto případě znaky od hodnoty 0 do 255). Grafy 5.8 a 5.9 udávají kompresní poměr metod na testovacím korpusu. Z grafů je možné vyčíst, jaký význam má velikost použitého slovníku na změnu velikosti kompresního poměru. Dále je patrné, že metoda LZW je pro většinu souborů účinnější než metoda LZ78. Některé grafy neobsahují všechny testovací soubory z korpusů, protože jinak by byly méně čitelné.

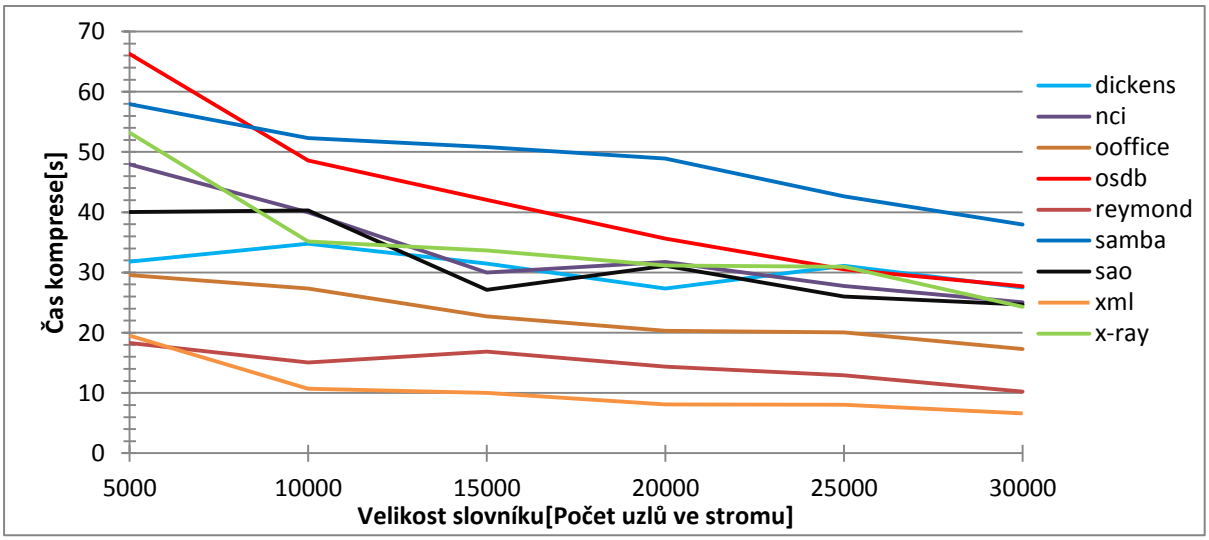
V grafech 5.10 a 5.11 jsou vyobrazeny časy kompresí jednotlivých souborů. Jak je na nich vidět, potřebný čas klesá s rostoucí velikostí slovníku. Toto se děje jednak proto, že se slovník promazává méně, a jednak proto, že jsou nalezeny delší řetězce. Tyto metody jsou mnohem rychlejší než předchozí, avšak by se zde zase dalo dosáhnout lepších výsledků nějakou optimalizací. Většina profesionálních komprimačních metod proto používá k vyhledávání jazyk assembler.



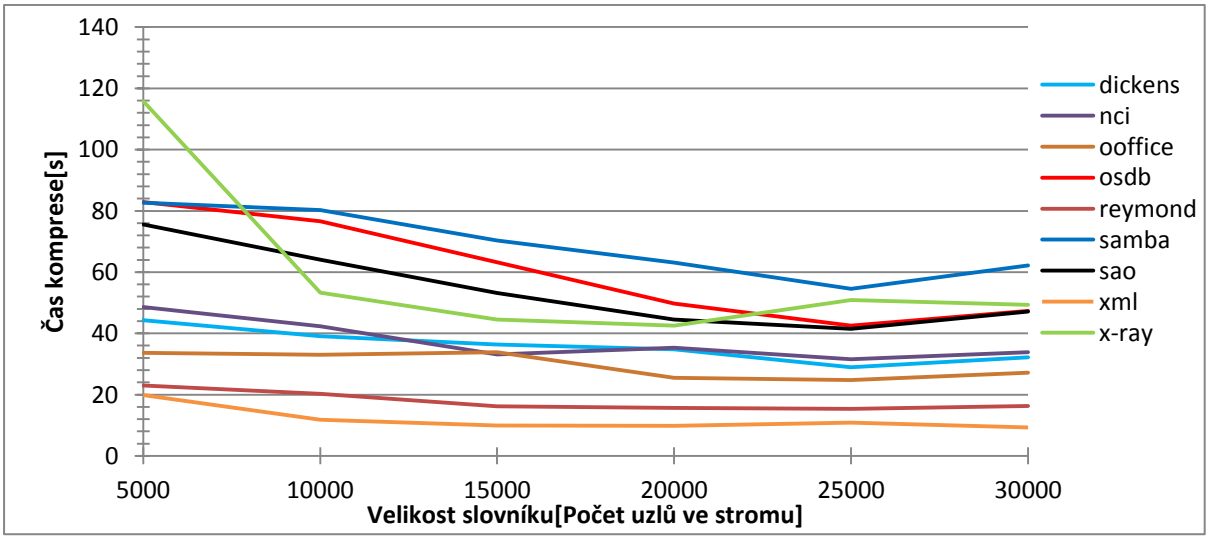
Graf 5.8: Kompresní poměr metody LZ78 při změnách velikostí slovníku



Graf 5.9: Kompresní poměr metody LZW při změnách velikostí slovníku

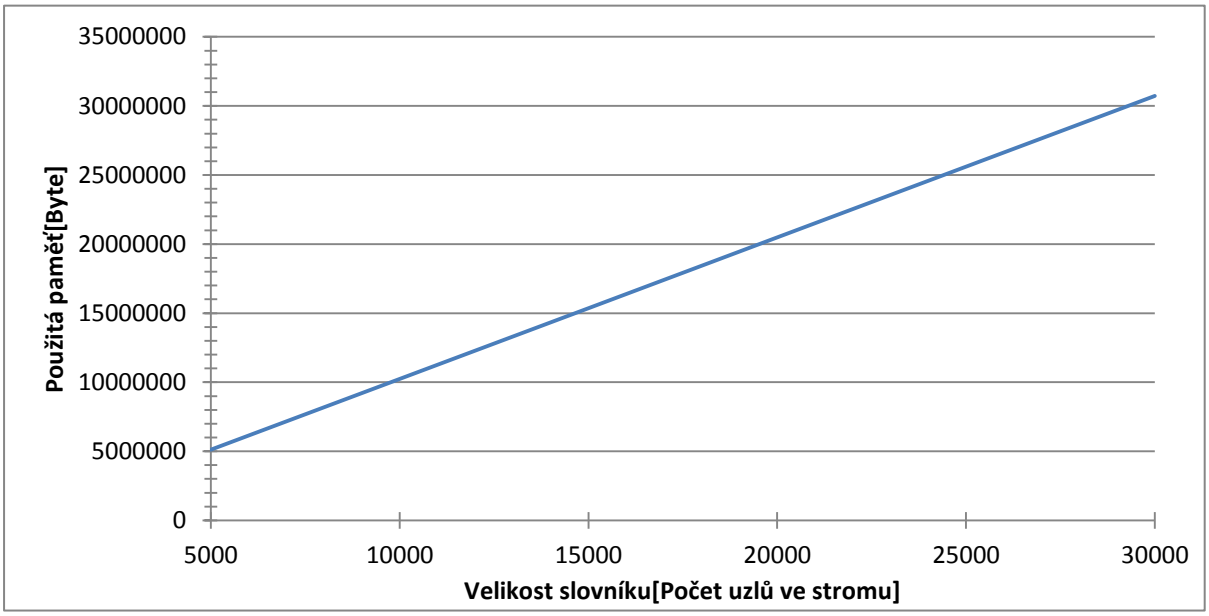


Graf 5.10: Čas komprese metodou LZ78



Graf 5.11: Čas komprese metodou LZW

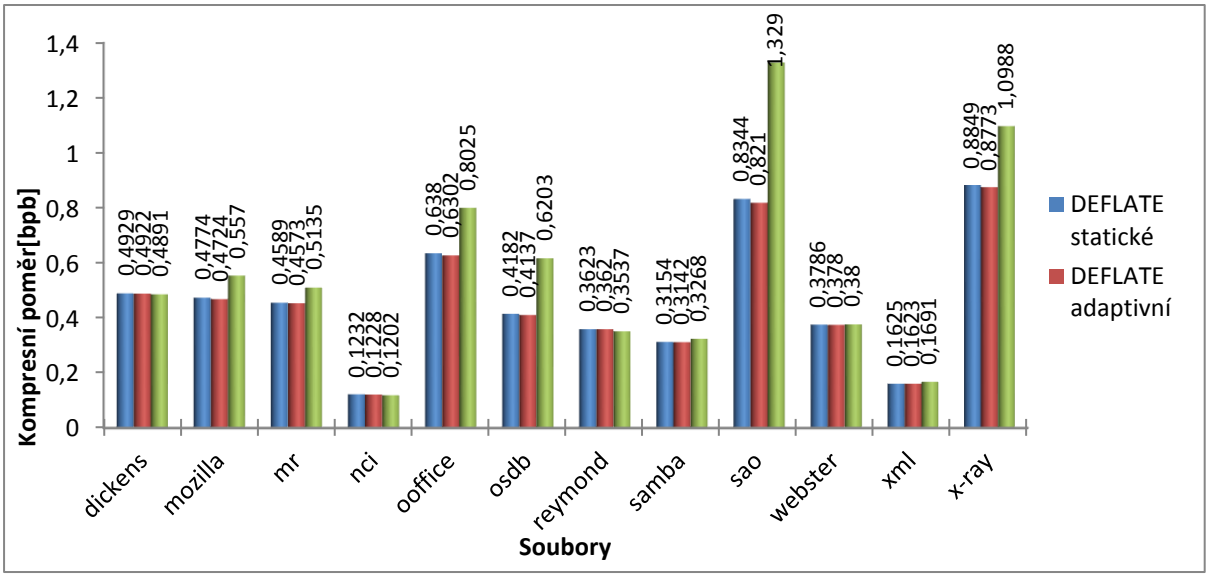
Problémem těchto metod jsou paměťové nároky, které rostou společně s velikostí slovníku. V grafu 5.12 je znázorněna použitá paměť při našich velikostech slovníku. Knihovna využívá pro tyto dvě metody strukturu tree, která si alokuje vždy potřebnou paměť pro uzel a jeho potomky. Toto řešení zabírá více paměti, avšak je rychlejší. Mohla by se vytvořit struktura tree, která by si alokovala paměť pouze pro požadovaný uzel, a jeho potomky by bylo potřeba alokovat až při jejich použití. Tento způsob by zabral méně paměti.



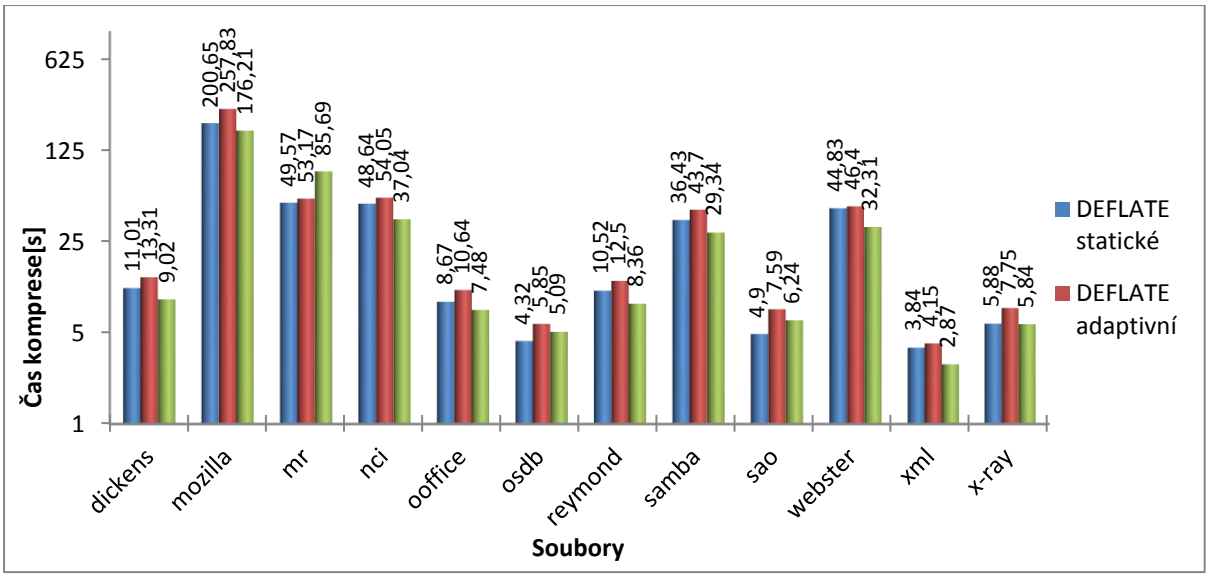
Graf 5.12: Paměťové nároky metod LZ78 a LZW

5.4 Metody DEFLATE a LZX

Tyto metody využívají k vyhledávání nejdelšího prefixu hashovací tabulku, což je činí rychlejšími nežli předchozí metody. DEFLATE využívá ke kódování dva modely, podle kterých data kóduje. Adaptivní model je sice pomalejší, ale dosahuje lepších výsledů nežli statický model. Metoda LZX data zpracuje a teprve při určitém okamžiku je zakóduje a pošle. Tyto kódovaná data jsou posílána společně s hlavičkou, která obsahuje potřebné informace k jejich dekódování. U metody LZX je dosaženo na některých typech dat horších výsledků. Jedná se zejména o data málo se opakující. Tento problém by bylo možné dále řešit například počítáním celkového množství znaků, jejich ukládáním (i s kódovanými daty) a následným vyhodnocením, které by nám řeklo, jestli data poslat s hlavičkou jako komprimovaný tok nebo zda data poslat jako nekomprimované znaky. V této práci je to řešeno tak, že se data pošlou jako nekomprimovaná pouze tehdy, jestli nebyla nalezena žádná komprese a opakující se znaky by nebylo výhodné komprimovat. V grafech 5.13 a 5.14 jsou zobrazeny výsledky měření, konkrétně kompresní poměr a rychlost komprese.



Graf 5.13: Kompresní poměry metod DEFLATE a LZX

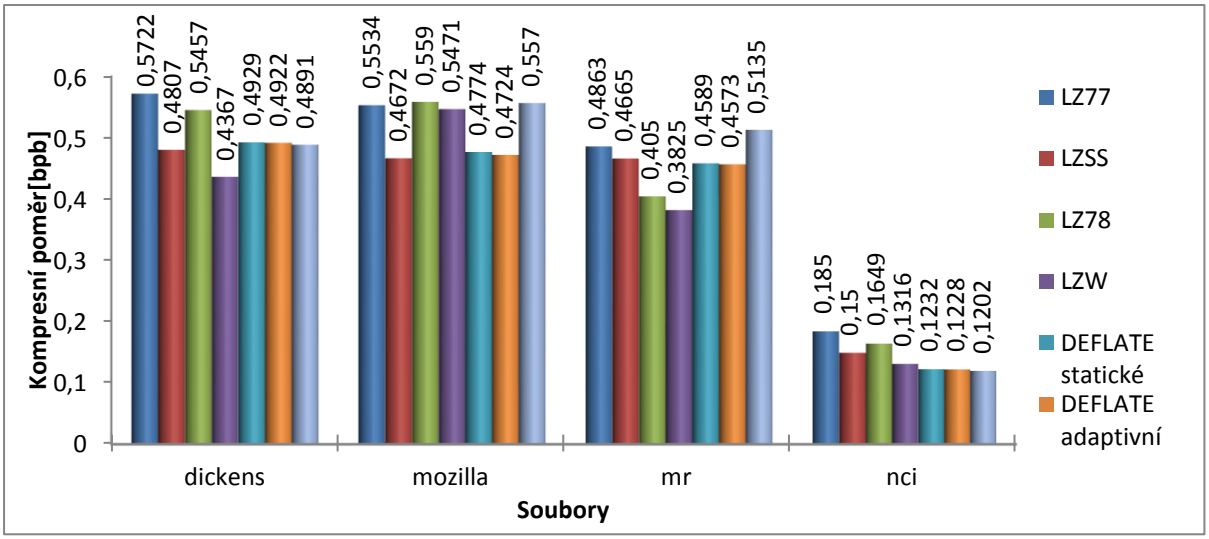


Graf 5.14: Čas komprese u metod DEFLATE a LZX

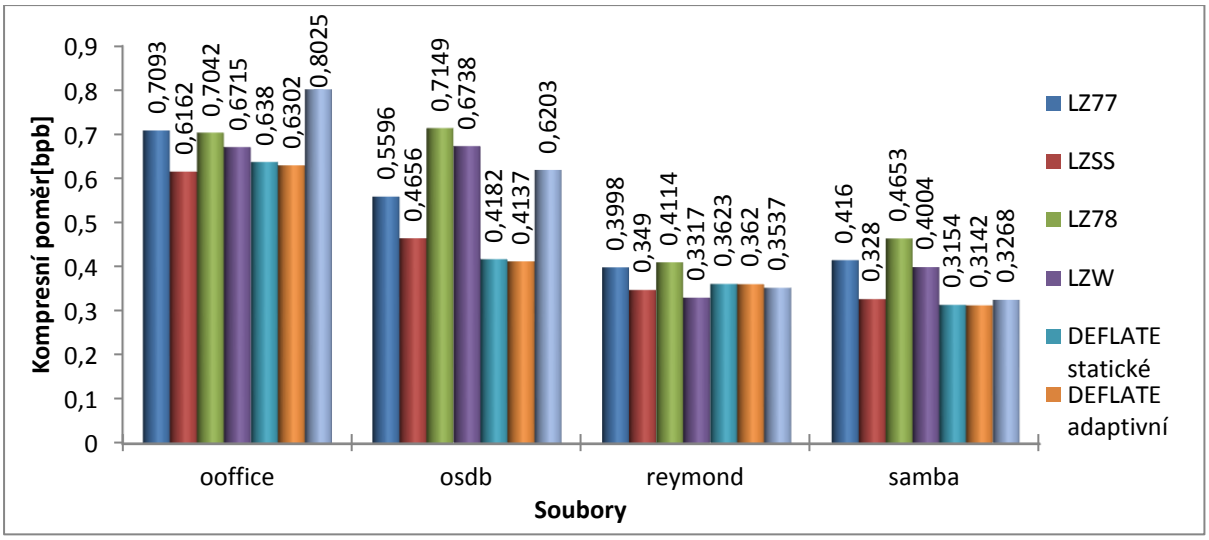
Na závěr této kapitoly jsou přidány grafy 5.15 – 5.16 s výsledky všech použitých metod se všemi soubory z použitého korpusu. Jak je z grafů patrné každá metoda dosáhla lepších výsledků na různých typech dat. Metoda LZ77 v těchto provedených testech nikde neskončila jako první. Ale LZSS dosáhla dobrých výsledků téměř na všech souborech. Nicméně by potřeboval optimalizovat binární vyhledávací strom, protože ten proces komprese při větších souborech zpomaluje.

Metody LZ78 a LZW si vedly v porovnávacím testu velice dobře. LZ78 dosáhla oproti LZW lepší komprese na souborech sao a x-ray (tady měla nejlepší kompresní poměr ze všech), jinak metoda LZW předčila LZ78 a na souborech dickens, mr a reymond skončila jako první. Největším problémem těchto metod byla ale paměťová náročnost, kterou měly ze všech nejvyšší.

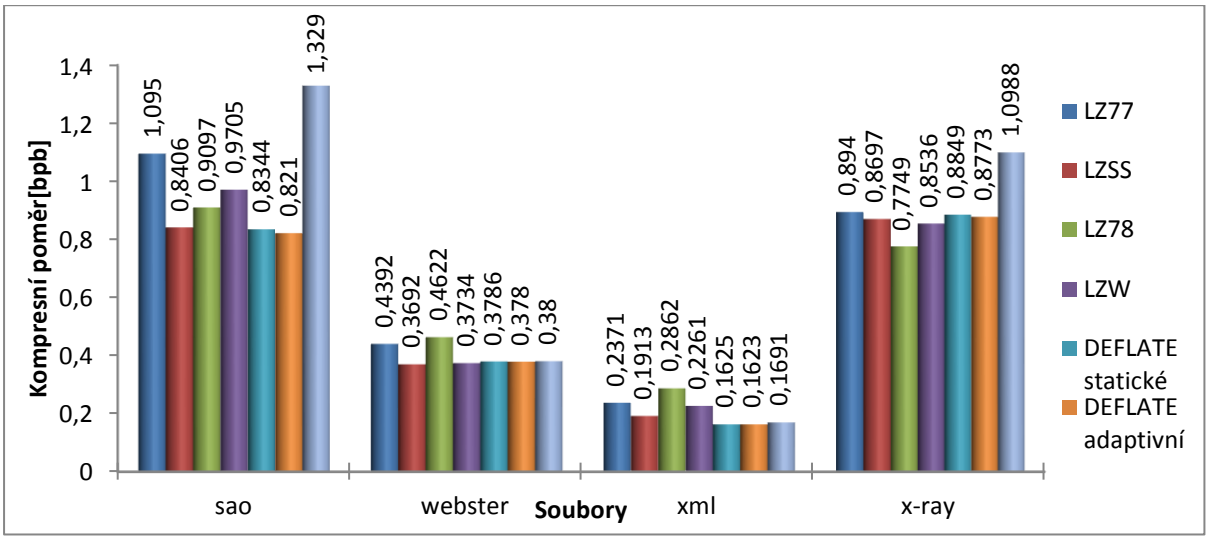
DEFLATE a LZX byli metody nejrychlejší a nejlepších výsledků dosáhly na datech, která se často opakovala (nci a xml).



Graf 5.15: První část kompresního poměru z celého korpusu



Graf 5.16: Druhá část kompresního poměru z celého korpusu



Graf 5.16: Třetí část kompresního poměru z celého korpusu

6 Závěr

Cílem této práce bylo implementovat slovníkové kompresní metody jako knihovnu v jazyce C/C++ a konzolovou aplikaci pro otestování této knihovny. Knihovna obsahuje metody LZ77, LZ78, LZSS, LZW, DEFLATE a LZX. Principy metod jsou detailně popsány v předchozím textu a na příkladech. Ovládání aplikace je popsané v dokumentaci programu.

Po implementaci této knihovny a aplikace bylo vše otestováno na korpusu Silensia. U metod LZ77 a LZSS jsou dva povinné parametry, na kterých závisí velikost komprese (kompresní poměr), délka trvání a využitá paměť. Parametry udávají velikost vyhledávacího a výhledového okna. Metoda LZSS dosáhla v těchto testech lepších výsledků než LZ77.

Jako další jsem testovat metody LZ78 a LZW, které pracují na stejném principu. LZ78 začíná s prázdným slovníkem, zatímco LZW už má ve slovníku uloženou abecedu pro testování (v aplikaci byli použity hodnoty 0 až 255). Zpočátku byly metody bez omezení velikosti slovníku a při testování na Canterbury korpusu dosahovaly metody lepších výsledků. Při testování na Silesia korpusu, který obsahuje větší soubory, byl slovník moc velký a zaplnil celou paměť. Proto jsem u těchto metod zavedl povinný parametr, který označuje maximální velikost slovníku. Při vyšších hodnotách byla míra komprese vyšší a čas komprese se snižoval. Na většině souborů bylo lepšího kompresního poměru dosaženo metodou LZW.

Poslední jsem otestoval metody DEFLATE a LZX. LZX neobsahuje žádné parametry a u metody DEFLATE je možné si vybrat, zda testovat se statickým modelem nebo s adaptivním modelem. Tato metoda obsahuje ještě další dva parametry, které ale zlepšují pouze délku trvání komprese na úkor kompresního poměru. Varianty těchto dvou metod jsou dnes používány v kompresních programech (DEFLATE – PKZIP, 7-Zip, PuTTY; LZX – microsoft cabinet files). Metoda LZX dosáhla lepších výsledků na datech, která se hodně opakovala.

Na testovacím korpusu bylo dosaženo kompresního poměru v rozmezí $\langle 0{,}1623; 1{,}329\rangle$. V několika případech z provedených testů bylo místo komprese dosaženo expanze. Podrobnější výsledky testů jsou uvedeny v předchozí kapitole a tabulkách B.1 – B.14, které jsou v příloze.

Práci by bylo možné rozšířit vypracováním dalších slovníkových metod (např. LZFG, LZRWx, LZMW, LZAP, LZY, LZPx, LZC, LZJB a LZMA). Dalším vývojem práce by mohlo být provedení optimalizací již vytvořených metod a doděláním aplikace. Aplikace by mohla mít i grafické rozhraní a dle komprimovaných souborů by vybírala nejvhodnější metody ke kompresi.

7 Literatura

[1] SALOMON, D. *Data Compression : The Complete Reference*. 3rd Edition. Springer, 2004. 899 s. ISBN 0-387-40697-2.

[2] BELL, T, CLEARY, J, WITTEN, I. *Text Compression*. New Jersey (USA) : Prentice Hall, 1990. 318 s. ISBN 0139119914.

[3] ZIV, J, LEMPEL, A. A Universal Algorithm for Sequential Data Compression. *IEEE TRANSACTIONS ON INFORMATION THEORY*. Květen 1997, IT-23, NO. 3, s. 337-343.

[4] ZIV, J, LEMPEL, A. Compression of Individual Sequences via Variable-Rate Coding. *IEEE TRANSACTIONS ON INFORMATION THEORY*. Září 1998, IT-24, NO. 5, s. 530-536

[5] STORER, J, SZYMANSKI, T. Data Compression via Textual Sustitution. *Journal of the ACM*. Říjen 1982, Vol-29, NO. 4, s. 928-951

[6] WELCH, T. A Technique for High-Performance Data Compression. *IEEE Computer*. Červen 1984, IT-17, NO. 6, s. 8-19

[7] MICROSOFT CORPORATION. *Microsoft LZX Data Compression Format*. Microsoft Corporation, 1997.

[8] DEUTSCH, P. *DEFLATE Compressed Data Format Specification version 1.3* [online] c1996. <ftp://ftp.vse.cz/pub/docs/rfc/rfc1866.txt>. [cit. 2011-05-01].

[9] DEROWICZ, S. *Zespół Oprogramowania* [online]. 2010. Silesia compression corpus. Dostupné z WWW: <http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia>. [cit. 2011-05-15].

Příloha A

Obsah přiloženého CD

- Technický text
- Readme
- Knihovna s aplikací
- Testovací korpusy Silesia a Canterbury

Příloha B

Tabulky s výsledky testů

	Statický model		Adaptivní model		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,4929	11,01	0,4922	13,31	1578888
mozilla	0,4774	200,65	0,4724	257,83	1578792
mr	0,4589	49,57	0,4573	53,17	1578720
nci	0,1232	48,64	0,1228	54,05	1578528
ooffice	0,638	8,67	0,6302	10,64	1578792
osdb	0,4182	4,32	0,4137	5,85	1577520
reymond	0,3623	10,52	0,362	12,5	1577976
samba	0,3154	36,43	0,3142	43,7	1578720
sao	0,8344	4,9	0,821	7,59	1578792
webster	0,3786	44,83	0,378	46,4	1578720
xml	0,1625	3,84	0,1623	4,15	1579248
x-ray	0,8849	5,88	0,8773	7,75	1579200

Tabulka B.1: Výsledky metody DEFLATE

Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,4891	9,02	1578888
mozilla	0,557	176,21	1578792
mr	0,5135	85,69	1578720
nci	0,1202	37,04	1578528
ooffice	0,8025	7,48	1578792
osdb	0,6203	5,09	1577520
reymond	0,3537	8,36	1577976
samba	0,3268	29,34	1578720
sao	1,329	6,24	1578792
webster	0,38	32,31	1578720
xml	0,1691	2,87	1579248
x-ray	1,0988	5,84	1579200

Tabulka B.2: Výsledky metody LZX

	5000			10000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,6219	31,79	5120000	0,5911	34,77	10240000
mozilla	0,5856	185,81	5120000	0,5736	149,66	10240000
mr	0,4463	34,59	5120000	0,4249	23,59	10240000
nci	0,2262	47,96	5120000	0,2001	40	10240000
ooffice	0,7504	29,54	5120000	0,7301	27,33	10240000
osdb	0,9282	66,28	5120000	0,8654	48,59	10240000
reymond	0,4985	18,32	5120000	0,4629	15,04	10240000
samba	0,5273	57,93	5120000	0,5007	52,33	10240000
sao	0,9315	40,01	5120000	0,9175	40,3	10240000
webster	0,5627	130,32	5120000	0,5198	120,13	10240000
xml	0,4052	19,53	5120000	0,3462	10,69	10240000
x-ray	0,9239	53,23	5120000	0,8554	35,14	10240000

Tabulka B.3: Výsledky metody LZ78 při velikosti slovníku 5000 a 10000

	15000			20000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,5732	31,49	15360000	0,5626	27,36	20480000
mozilla	0,568	144,59	15360000	0,5648	134,24	20480000
mr	0,4156	20,6	15360000	0,4113	19,72	20480000
nci	0,1862	30,01	15360000	0,1773	31,74	20480000
ooffice	0,7183	22,72	15360000	0,7136	20,33	20480000
osdb	0,8155	42,03	15360000	0,7772	35,61	20480000
reymond	0,4427	16,89	15360000	0,43	14,37	20480000
samba	0,4856	50,8	15360000	0,4779	48,89	20480000
sao	0,9118	27,11	15360000	0,9121	31,09	20480000
webster	0,4965	100,72	15360000	0,4827	94,12	20480000
xml	0,3195	10,02	15360000	0,3064	8,12	20480000
x-ray	0,8185	33,68	15360000	0,799	31,18	20480000

Tabulka B.4: Výsledky metody LZ78 při velikosti slovníku 15000 a 20000

	25000			30000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,5538	31,1	25600000	0,5457	27,48	30720000
mozilla	0,5629	120,1	25600000	0,559	115,48	30720000
mr	0,4086	19,22	25600000	0,405	16,86	30720000
nci	0,1706	27,76	25600000	0,1649	25,03	30720000
ooffice	0,7087	20,08	25600000	0,7042	17,31	30720000
osdb	0,7442	30,53	25600000	0,7149	27,72	30720000

reymond	0,4204	12,95	25600000	0,4114	10,24	30720000
samba	0,4719	42,66	25600000	0,4653	37,98	30720000
sao	0,9118	26	25600000	0,9097	24,7	30720000
webster	0,4717	90,23	25600000	0,4622	80,21	30720000
xml	0,2941	8,02	25600000	0,2862	6,61	30720000
x-ray	0,7858	30,95	25600000	0,7749	24,32	30720000

Tabulka B.5: Výsledky metody LZ78 při velikosti slovníku 25000 a 30000

	5000			**10000**		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,513	44,29	5120000	0,4794	39,1	10240000
mozilla	0,5564	241,24	5120000	0,5532	220,54	10240000
mr	0,4163	47,73	5120000	0,39927	35,26	10240000
nci	0,1802	48,54	5120000	0,1579	42,36	10240000
ooffice	0,7011	33,66	5120000	0,6882	33,05	10240000
osdb	0,9136	82,83	5120000	0,8394	76,57	10240000
reymond	0,4076	22,95	5120000	0,3743	20,25	10240000
samba	0,4555	82,63	5120000	0,4322	80,22	10240000
sao	0,9801	75,6	5120000	0,9817	64,06	10240000
webster	0,4652	192,15	5120000	0,4236	151,3	10240000
xml	0,3198	19,89	5120000	0,2736	11,8	10240000
x-ray	0,96	115,83	5120000	0,9023	53,32	10240000

Tabulka B.6: Výsledky metody LZW při velikosti slovníku 5000 a 10000

	15000			**20000**		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,4611	36,35	15360000	0,4518	34,76	20480000
mozilla	0,5505	180,25	15360000	0,5515	190,9	20480000
mr	0,391	26,71	15360000	0,3883	26,46	20480000
nci	0,1475	33,11	15360000	0,1414	35,32	20480000
ooffice	0,6803	33,9	15360000	0,6778	25,49	20480000
osdb	0,7828	63,2	15360000	0,7417	49,74	20480000
reymond	0,3567	16,23	15360000	0,3471	15,63	20480000
samba	0,4192	70,28	15360000	0,4117	63,14	20480000
sao	0,9785	53,21	15360000	0,9796	44,54	20480000
webster	0,4024	114,79	15360000	0,391	121,77	20480000
xml	0,2519	9,92	15360000	0,2413	9,86	20480000
x-ray	0,877	44,47	15360000	0,8681	42,53	20480000

Tabulka B.7: Výsledky metody LZW při velikosti slovníku 15000 a 20000

	25000			30000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,44	28,99	25600000	0,4367	32,16	30720000
mozilla	0,5499	184,71	25600000	0,5471	180,12	30720000
mr	0,3864	24,35	25600000	0,3825	25,87	30720000
nci	0,1363	31,51	25600000	0,1316	33,88	30720000
ooffice	0,6754	24,79	25600000	0,6715	27,13	30720000
osdb	0,7061	42,53	25600000	0,6738	47,31	30720000
reymond	0,3395	15,31	25600000	0,3317	16,28	30720000
samba	0,4071	54,54	25600000	0,4004	62,18	30720000
sao	0,9772	41,45	25600000	0,9705	47,12	30720000
webster	0,3818	128,72	25600000	0,3734	110,5	30720000
xml	0,2342	10,88	25600000	0,2261	9,28	30720000
x-ray	0,8613	50,88	25600000	0,8536	49,29	30720000

Tabulka B.8: Výsledky metody LZW při velikosti slovníku 25000 a 30000

	6000			9000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,6303	137,12	48480	0,6183	193,8	72480
mozilla	0,5699	646,92	48480	0,5711	918,57	72480
mr	0,5107	163,28	48480	0,5073	235,73	72480
nci	0,2003	217,16	48480	0,1947	300,6	72480
ooffice	0,7363	90,24	48480	0,7353	146,36	72480
osdb	0,8369	157,42	48480	0,7451	205,36	72480
reymond	0,4688	66,41	48480	0,453	92,37	72480
samba	0,4549	204,5	48480	0,4478	286,98	72480
sao	1,108	147,69	48480	1,1186	210,64	72480
webster	0,4849	402,88	48480	0,4744	575,62	72480
xml	0,2621	28,54	48480	0,255	40,37	72480
x-ray	0,981	162,52	48480	0,9475	227,77	72480

Tabulka B.9: Výsledky metody LZ77 při velikosti výhledového okna 30B

	12000			15000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,5948	250,96	96480	0,5774	311,43	120480
mozilla	0,5583	1150,38	96480	0,5487	1542,92	120480
mr	0,4933	306,64	96480	0,4834	393,5	120480
nci	0,1885	401,71	96480	0,1845	506,7	120480

ooffice	0,7165	184,26	96480	0,7035	205,79	120480
osdb	0,6642	240,92	96480	0,6091	280,43	120480
reymond	0,4179	117,31	96480	0,4061	139,79	120480
samba	0,4305	367,06	96480	0,4193	451,33	120480
sao	1,0975	281,65	96480	1,0814	352,83	120480
webster	0,4561	730,17	96480	0,4428	900,06	120480
xml	0,2449	51,5	96480	0,2383	63,03	120480
x-ray	0,918	294,79	96480	0,8958	364,21	120480

Tabulka B.10: Výsledky metody LZ77 při velikosti výhledového okna 30B

	18000			**21000**		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,5839	349,84	144480	0,5722	390,68	168480
mozilla	0,5599	1724,64	144480	0,5534	1980,65	168480
mr	0,4927	475,95	144480	0,4863	518,88	168480
nci	0,1878	590,69	144480	0,185	667,88	168480
ooffice	0,7181	236,19	144480	0,7093	270,68	168480
osdb	0,5892	306,87	144480	0,5596	340,52	168480
reymond	0,4103	160,89	144480	0,3998	183,06	168480
samba	0,4245	552,29	144480	0,416	600,4	168480
sao	1,1066	407,36	144480	1,095	459,09	168480
webster	0,4479	1027,99	144480	0,4392	1165,01	168480
xml	0,2415	72,48	144480	0,2371	83,02	168480
x-ray	0,9101	416,36	144480	0,894	463,94	168480

Tabulka B.11: Výsledky metody LZ77 při velikosti výhledového okna 30B

	6000			**9000**		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,5294	31,33	95776	0,5207	31,88	143776
mozilla	0,4739	1392,21	95776	0,4743	1792,52	143776
mr	0,48	964,14	95776	0,4814	1503,95	143776
nci	0,1655	258,46	95776	0,1604	333,15	143776
ooffice	0,6266	44,94	95776	0,6282	51,14	143776
osdb	0,6827	33,85	95776	0,6091	33,23	143776
reymond	0,4017	20,21	95776	0,3915	20,68	143776
samba	0,3587	392,11	95776	0,3519	553,43	143776
sao	0,8483	26	95776	0,8497	26,35	143776
webster	0,4058	128,76	95776	0,3982	132,79	143776
xml	0,2131	23,63	95776	0,2072	25,73	143776
x-ray	0,8934	29,28	95776	0,8882	29,92	143776

Tabulka B.12: Výsledky metody LZSS při velikosti výhledového okna 30B

	12000			15000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,4889	31,48	191776	0,4826	31,84	239776
mozilla	0,4671	1974,84	191776	0,4619	2223,22	239776
mr	0,4694	1926,68	191776	0,4604	2518,87	239776
nci	0,1541	402,5	191776	0,1498	480,69	239776
ooffice	0,617	55,87	191776	0,6092	58,27	239776
osdb	0,5503	32,96	191776	0,5086	32,56	239776
reymond	0,3621	21,18	191776	0,3515	21,6	239776
samba	0,3386	627,33	191776	0,3304	745,92	239776
sao	0,8407	26,59	191776	0,8341	26,8	239776
webster	0,3821	133,49	191776	0,3702	136,47	239776
xml	0,198	26,85	191776	0,1919	28,28	239776
x-ray	0,8696	29,57	191776	0,8638	29,98	239776

Tabulka B.13: Výsledky metody LZSS při velikosti výhledového okna 30B

	18000			21000		
Název souboru	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**	**Kompresní poměr [bpb]**	**Čas komprese [s]**	**Využitá paměť [B]**
dickens	0,4917	33,18	287776	0,4807	34,6	335776
mozilla	0,4704	2270,5	287776	0,4672	2205,35	335776
mr	0,4725	3025,4	287776	0,4665	3334,9	335776
nci	0,153	520,1	287776	0,15	600,79	335776
ooffice	0,6214	59,55	287776	0,6162	59,77	335776
osdb	0,4882	34,73	287776	0,4656	33,85	335776
reymond	0,3585	23,7	287776	0,349	23,13	335776
samba	0,3342	871,5	287776	0,328	917,99	335776
sao	0,8449	28,36	287776	0,8406	28,4	335776
webster	0,3769	144	287776	0,3692	206,46	335776
xml	0,1953	30,82	287776	0,1913	32,24	335776
x-ray	0,8801	30,18	287776	0,8697	32,25	335776

Tabulka B.14: Výsledky metody LZSS při velikosti výhledového okna 30B

	x-ray	xml	webster	sao	samba	reymond	osdb	ooffice	nci	mr	mozilla	dickens
LZ77	0,894	0,2371	0,4392	1,095	0,416	0,3998	0,5596	0,7093	0,185	0,4863	0,5534	0,5722
LZSS	0,8697	0,1913	0,3692	0,8406	0,328	0,349	0,4656	0,6162	0,15	0,4665	0,4672	0,4807
LZ78	0,7749	0,2862	0,4622	0,9097	0,4653	0,4114	0,7149	0,7042	0,1649	0,405	0,559	0,5457
LZW	0,8536	0,2261	0,3734	0,9705	0,4004	0,3317	0,6738	0,6715	0,1316	0,3825	0,5471	0,4367
DEFLATE statické	0,8849	0,1625	0,3786	0,8344	0,3154	0,3623	0,4182	0,638	0,1232	0,4589	0,4774	0,4929
DEFLATE adaptivní	0,8773	0,1623	0,378	0,821	0,3142	0,362	0,4137	0,6302	0,1228	0,4573	0,4724	0,4922
LZX	1,0988	0,1691	0,38	1,329	0,3268	0,3537	0,6203	0,8025	0,1202	0,5135	0,557	0,4891

Tabulka B.15: Výsledky všech metod na celém testovacím korpusu