Komprese dat

Cvičení

4. dubna 2006

Obsah

1 Základní pojmy, entropie, redundance

Se stále rostoucím množstvím ukládaných a přenášených dat roste potřeba komprese těchto dat, tedy způsobu jak zmenšit prostor, který zabírají. Metody komprese dat dělíme na ztrátové a bezeztrátové. Ztrátové kompresní metody nezachovávají všechnu informaci uloženou v datech, jsou vhodné pro kompresi obrázků, zvuku a videa. Toto skriptum se zaměřuje na bezeztrátovou kompresi.

Definice 1.1 (Kód)

Kód K je uspořádaná trojice (S, C, f), kde S je množina zdrojových jednotek, C je množina kódových slov a f je zobrazení z S do C. Zobrazení f přiřazuje každé zdrojové jednotce z množiny S právě jedno *kódové slovo* z množiny C. Toto zobrazení musí být *prosté*, to znamená každé dvě různé zdrojové jednotky jsou zobrazovány na dvě různá kódová slova. Toto je nutnou nikoli dostačující podmínkou pro jednoznačnou dekódovatelnost kódu K.

Zdrojová jednotka je tedy definována jako prvek nějaké množiny. Nejčastěji jsou jako zdrojové jednotky brány symboly nějaké abecedy (textové soubory), ale tato definice připouští použití složitějších zdrojových jednotek, např. slov.

Zobrazení f může být zobecněno na řetězce zdrojových jednotek (*zdrojová data, vstupní data*):

$$f(S_1 S_2 \dots S_k) = f(S_1) f(S_2) \dots f(S_k).$$

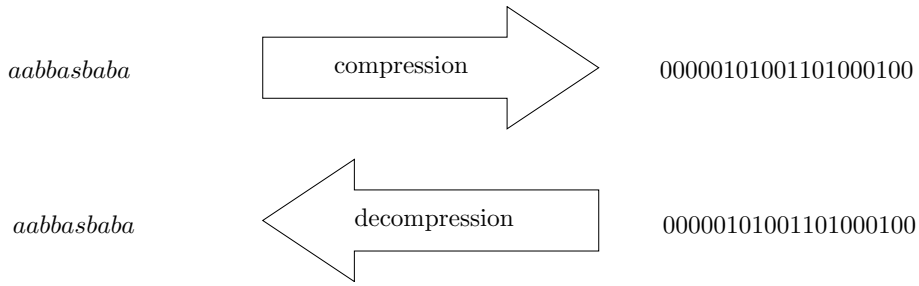
Zakódovaná vstupní data budeme označovat pojmem *výstupní data*.

Prvky množiny C označujeme jako *kódy* nebo *kódová slova*. Jestliže výstupní data obsazují méně místa než vstupní data, pak proces kódování označujeme pojmem *komprese* a výstupní data označujeme jako *komprimovaná*.

Opačný proces ke kompresi dat označujeme jako *dekompresi*. Oba procesy jsou názorně zobrazeny na obrázku 1.1.

Definice 1.2 (Jednoznačně dekódovatelný kód)
Mějme nějaký kód $K = (S, C, f)$. Řetězec $x \in C^+$ je *jednoznačně dekódovatelný* vzhledem k zobrazení f, jestliže existuje právě jeden řetězec $y \in S^+$ takový, že $f(y) = x$. Kód (S, C, f) je *jednoznačně dekódovatelný kód* právě tehdy když všechny možné řetězce z C^+ jsou jednoznačně dekódovatelné. Symbolem C^+ označujeme množinu všech řetězců nenulových délek vzniklých ze symbolů z C.

Příklad 1.3
Mějme kód $K = (S, C, f)$, kde $S = \{a, b, c, d\}$, $C = \{12, 122, 1211, 2121\}$ a zobrazení f je definováno následující tabulkou:



Obrázek 1.1: Grafické znázornění procesu komprese a dekomprese. Množina zdrojových jednotek je $S = \{a, b, s\}$, množina kódových slov C je $\{00, 01, 11\}$.

zdrojová jednotka	kódové slovo
a	122
b	21211
c	12
d	1211

Jeden z možných výstupních řetězců je řetězec 12221211212111221211. Tento kódový řetězec je jednoznačně dekódovatelný, zdrojový text byl *abbad*.

Jiným výstupním kódovým slovem je 12111221211. Tento řetězec není jednoznačně dekódovatelný, protože možné zdrojové texty, které zobrazení f převede na naše kódové slovo jsou *dad* a *dcb*.

Příklad 1.4
Mějme kód $K = (S, C, f)$, kde $S = \{a, b, c, d\}$, $C = \{121, 122, 12211, 2222\}$ a zobrazení f je definováno následující tabulkou:

zdrojová jednotka	kódové slovo
a	122
b	122111
c	121
d	2222

Jeden z možných výstupních řetězců je řetězec 122122111122121122111. Tento kódový řetězec je jednoznačně dekódovatelný, zdrojový text byl *abacb*.

Určení, zda libovolný kód je jednoznačně dekódovatelný je algoritmicky neřešitelný problém, nicméně existují třídy kódů, které jsou jednoznačně dekódovatelné, mezi ně patří prefixové, afixové a blokové kódy.

Definice 1.5 (Prefixový, afixový a blokový kód)
Kód (S, C, f) je *prefixový kód* jestliže žádné kódové slovo z C^+ není předponou jiného kódového slova z C^+. Zdrojová data zakódována prefixovým

kódem jsou jednoznačně dekódovatelná dekódováním jednotlivých kódových slov během čtení zleva doprava. To nám umožní začít s dekódováním aniž bychom znali celý kódový text. Mezi prefixové kódy patří například kód UTF-8.

Pro *afixový kód* platí, že žádné kódové slovo není příponou jiného kódového slova. Afixový kód je dekódovatelný znak po znaku během čtení zprava doleva. Kód z příkladu 1.4 je afixovým kódem, proto je jednoznačně dekódovatelný.

Pokud mají všechna kódová slova stejnou délku, pak tento kód označujeme jako *blokový*. Jedním z kódů pro kódování znaků je ASCII [1] kód. Základní ASCII kód používá 7 bitů pro 95 grafických (znaky anglické abecedy, číslice, apod.) a 33 řídících znaků, rozšířený ASCII kód je 8-bitový dovolující kódovat 256 znaků.

Je si nutné uvědomit, že neexistuje jednoznačná bezeztrátová komprese, která by každý vstupní řetězec zkomprimovala na nějaký kratší řetězec. Důvodem je rozdíl v mohutnosti množin vstupních řetězců (delší řetězce) a komprimovaných řetězců (kratší řetězce).

Definice 1.6 (Kompresní poměr)
Mějme kód (S, C, f), $x \in S^+, y \in C^+ : y = f(x)$. Řetězec x je komprimován na řetězec y. *Kompresní poměr* cr je:

$$cr = \frac{|y|}{|x|}.$$

Definice 1.7 (Optimální kód)
Mějme kód (S, C, f). Pravděpodobnostní rozložení jednotlivých prvků z S je p, $\sum_{i \in S} p(i) = 1$. Kód (S, C, f) je optimální, pokud neexistuje kód (S, C', f') takový, že $\sum_{i \in S} p(i)|f'(i)| < \sum_{i \in S} p(i)|f(i)|$.

S kompresí dat úzce souvisí pojmy entropie (neuspořádanost, neurčitost, nejistota) a redundance (nadbytečnost). Claude Elwood Shannon označovaný jako otec teorie informace v roce 1948 definoval entropii[2] (střední hodnotu informace na jeden symbol zprávy) takto:

Definice 1.8 (Entropie)
Předpokládejme existenci nějakého systému. Dále předpokládejme samostatnou událost (nezávislou na předchozích událostech), která způsobí přechod

[1] American Standard Code for Information Interchange - americký standardní kód pro výměnu informací

[2] Vypráví se, že když Shannon uvažoval o pojmenování této veličiny (pojem informace byl již přetížený), řekl mu John von Neumann přibližně toto:„You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, no one really knows what entropy really is, so in a debate you will always have the advantage.“

systému do nového stavu. Předpokládejme n vzájemně se vylučujících stavů x, pravděpodobnost stavu i je $p(i)$, *entropie* stavu x je

$$H(x) = -\sum_{i=1}^{n} p(i) \log_2 p(i).$$

Jednotkou entropie je jeden bit[3]. Jeden bit je použit pro zprávu, která byla vybrána ze dvou stejně pravděpodobných možností[4].

Tato definice předpokládá, že pravděpodobnost jednoho symbolu v textu je nezávislá na pravděpodobnosti předcházejících symbolů. Tato vlastnost pro texty v přirozeném jazyce není splněna, například v jazyce anglickém je pravděpodobnost znaku 'u' po symbolu 'q' větší než 99%.

Pokud není tato podmínka splněna, je nutné definovat entropii odlišným způsobem. Tato definice bude uvedena u kontextových metod komprese dat.

Pro definici entropie se zavádí statistický model dat. Pro zjednodušení budeme pro účely klasifikace statistických modelů uvažovat, že zdrojovými jednotkami jsou znaky. Pak můžeme zavést tyto modely:

- Statistický model nultého stupně. Každý znak vstupního textu je statisticky nezávislý na jiném znaku, pravděpodobnosti výskytu všech znaků jsou stejné. Tomuto modelu odpovídá náhodně generovaný text a přibližují se mu DNA sekvence.

- Statistický model prvního stupně. Každý znak vstupního textu je statisticky nezávislý na jiném znaku, pravděpodobnosti výskytu všech znaků jsou různé.

- Statistický model druhého stupně. Tento model zohledňuje pravděpodobnosti výskytu dvojic znaků. Jednotlivé pravděpodobnosti se nepřiřazují jednotlivým znakům, ale jejich dvojicím.

- Statistický model n-tého stupně. Tento model pracuje s pravděpodobnostmi n-tic znaků.

Entropie zprávy vyjadřuje míru informace, která je v ní uložená. Pokud mají znaky ve zprávě různý počet výskytů, nevede použití statistického modelu nultého stupně k přesnému odhadu entropie. Pokud je text kontextově závislý, je nutné použít statistický model vyššího stupně, který lépe odhadne entropii zprávy. Pro anglický text s mezerou statistický model nultého stupně odhaduje entropii na 4,75 bitů na znak, model třetího stupně již tento odhad zpřesňuje na 2,77 bitů na znak.

[3]bit - zkratka z anglických slov **bi**nary digi**t**

[4]Počet možností ovlivňuje základ logaritmu, pro základ 2 je jednotka jeden bit, pro přirozený logaritmus (základ e) je jednotkou jeden nat a pro základ 10 jeden Hartley. Pro převod logaritmů se používá následující vztah $\log_2 x = \frac{\log_{10} x}{\log_{10} 2}$

Definice 1.9 (Redundance)
Předpokládejme kódování zdrojové jednotky x a značme ji jako $C(x)$. Délku kódového slova $C(x)$ označme $L(x)$. Tato délka se měří v bitech. Teorie informace ukazuje, že $H(x) \leq L(x)$. Rozdíl mezi délkou kódu a entropií x je *redundance* (nadbytečnost) kódu $C(x)$, značíme ji $R(x)$. Formálně:

$$R(x) = L(x) - H(x).$$

Příklad 1.10
V tomto příkladu si ukážeme vztah mezi průměrnou entropií symbolu a pravděpodobnostním rozložením symbolů. Mějme tři typy zpráv X_1, X_2, X_3 nad abecedou $\Sigma = \{a, b, c, d\}$. Rozložení pravděpodobností jednotlivých symbolů pro jednotlivé typy řetězců definuje následující tabulka:

symbol	P_1	P_2	P_3
a	0,25	0,5	1
b	0,25	0,25	0
c	0,25	0,125	0
d	0,25	0,125	0

První typ zpráv odpovídá rovnoměrnému rozložení všech symbolů, reprezentant by mohl vypadat například takto *abacdcbd*. Druhou třídu zpráv reprezentuje řetězec *aabaabcd* a poslední typ zprávy má jediného reprezentanta *aaaaaaaa*. Následující tabulka ukazuje hodnoty entropií jednotlivých symbolů pro tři různé distribuce pravděpodobností.

symbol	P_1	H_1	P_2	H_2	P_3	H_3
a	0,25	2	0,5	1	1	0
b	0,25	2	0,25	2	0	∞
c	0,25	2	0,125	3	0	∞
d	0,25	2	0,125	3	0	∞
H_{avg}		2		1,75		0

Vztah $H_{avg}(S) = -\sum_{i \in S} p(i) \log_2 p(i)$ definuje průměrnou entropii. Z tabulky je vidět, že entropie jednoho symbolu roste pokud klesá počet jeho výskytů ($\sim$ pravděpodobnost). Průměrná entropie je maximální pro rovnoměrné rozložení výskytů symbolů a naopak nejmenší (nulová) pro řetězce obsahující pouze jeden symbol.

Příklad 1.11
Mějme text dlouhý 160 znaků. V tomto textu se 80 krát opakuje symbol 'a', 40 krát symbol 'e', 20 krát symbol 'c'. Dále tento text obsahuje po deseti symbolech 'g' a 'b'. Předpokládejme, že výskyty jednotlivých symbolů jsou

nezávislé na předcházejících symbolech. Určete entropii každého symbolu a entropii celé zprávy.

Pro výpočet entropie zdrojových jednotek nejprve vypočítáme jejich pravděpodobnosti a poté určíme entropii podle vztahu:

$$H(x_i) = -\log_2 p(i), x_i \in S$$

Výsledné entropie shrnuje následující tabulka:

symbol	počet	pravděpodobnost	entropie (bit)
a	80	0,5	1
b	10	0,0625	4
c	20	0,125	3
e	40	0,25	2
g	10	0,0625	4

Je vidět, že čím výjimečnější symbol (malá pravděpodobnost) tím nese více informace (větší redundance). Pokud je pravděpodobnost nějakého symbolu zápornou mocninou čísla dvě, vychází entropie celočíselná a odpovídá počtu bitů potřebných pro uložení jednotlivých symbolů.

Průměrná entropie jednoho symbolu z S je definována vztahem:
$H_{avg}(S) = -\sum_{i \in S} p(i) \log_2 p(i) = \sum_{i \in S} p(i) H(x_i) = 1{\cdot}0,5 + 4{\cdot}0,0625 + 3{\cdot}0,125 + 2{\cdot}0,25 + 4{\cdot}0,0625 = 1,875$ bitu

Entropie této zprávy X délky k znaků je definována vztahem:
$H(X) = -\sum_{i=1}^{k} \log_2 p(i) = \sum_{i=1}^{k} H(x_i) = k{\cdot}H_{avg}(S) = 300$ bitů.

Příklad 1.12

Mějme zprávu s pravděpodobnostním rozložením definovaným v příkladu 1.11. Uvažujme dva kódy definované následující tabulkou.

symbol, S_i	kód, C_i^1	kód, C_i^2
a	0	01
b	1110	00
c	110	100
e	10	11
g	1111	101

Vypočtěte průměrnou redundanci obou kódů. Určete redundanci pro typickou zprávu délky 160 znaků. Typická zpráva má frekvence výskytu jednotlivých znaků odpovídající jejich pravděpodobnosti výskytu. Pro výpočet redundance kódu musíme zjistit délku zakódované zprávy. Pro její výpočet využijeme četností z příkladu 1.11. Délka kódu zprávy délky k symbolů lze vypočítat podle vztahu

$$L(x) = \sum_{j=1}^{k} d_{i_j},$$

kde $i \in S$ a $f(x_i)$ má d_i bitů. Výsledné délky kódů pro C^1 a C^2 jsou $L^1 = 300$ bitů a $L^2 = 350$ bitů. Redundance kódu je $R(X) = L(X) - H(X)$. Typická zpráva v prvním kódu má nulovou redundanci a typická zpráva ve druhém kódu obsahuje 50 redundantních bitů na 160 bitů zprávy. Průměrná redundance druhého kódu je tedy 0,3125 bitu na znak.

Příklad 1.13
Mějme typickou zprávu odpovídající pravděpodobnostnímu rozložení z příkladu 1.11. Dále uvažujme dva kódy definované v příkladu 1.12. Předpokládejme, že typická zpráva byla zakódována v rozšířeném ASCII kódování (8 bitů na znak). Určete kompresní poměr pro oba kódy.

1.1 RLE komprese

Mezi nejjednodušší metody komprese dat patří metoda označovaná RLE (z anglického Run-length encoding). Tato metoda využívá skutečnosti, že některá data obsahují dlouhé posloupnosti (řada $\sim$ run) stejných znaků. Typickým příkladem použití jsou obrázky v nízkém barevném rozlišení. Tato metoda je využita například ve formátu obrázků PCX a může být použita ve formátu TIFF.

Základní varianta RLE komprese kóduje každou posloupnost dvojicí (počet opakování, znak). Tato metoda je výhodná pro černobílé obrázky. Předpokládejme, část obrázku

$$X = bbbbbbbwwwwwwwwwwwwwwwwwwwwbbbbbbbbbbbbbbbbbbwwwww,$$

symbolem b značíme černou barvu, symbolem w barvu bílou. RLE komprese by tento řetězec zakódovala do posloupnosti

$$7b20w18b5w.$$

Tento kód je nevhodný pro texty, které obsahují vedle dlouhých posloupností i jednoznakové posloupnosti. Pro tyto texty vznikla varianta, která pro neopakující se symboly (či krátké posloupnosti, např. dvou znaků) neprovádí kódování počtu, ale přímo přepisuje vstup. Pak je nutné použít nějaký speciální symbol, který „přepne“ do módu dvojic, (počet, znak).

Příklad 1.14
Nyní se pokusíme vypočítat kompresní poměr pro kompresi části obrázku z úvodu této kapitoly. V obrázku mohou být pouze dvě hodnoty, $\{b, w\}$. Pro zakódování jednoho znaku použijeme jeden bit, délka nezakódované zprávy je pak 50 bitů.

Délku zakódovaného obrázku výrazně ovlivní použité kódování na reprezentaci počtu opakování. Pro jednoduchost budeme uvažovat, že nejdelší posloupnost nepřekročí 256 symbolů, pak použijeme binární kód pevné délky, 8 bitů.

1.2 Příklady na cvičení

Příklad 1.15
Mějme kód definovaný následující tabulkou.

zdrojová jednotka	kódové slovo
a	123
b	321
c	12
d	312

Mějme kódové řetězce:

1. 31212321,
2. 1232112123,
3. 32131212312,
4. 3211212,
5. 12321.

Pokuste se dekódovat jednotlivé řetězce. Rozhodněte, který kódový řetězec je jednoznačně dekódovatelný. Je tento kód jednoznačně dekódovatelný? Je tento kód prefixový, blokový či afixový?

Příklad 1.16
Mějme kód definovaný následující tabulkou.

zdrojová jednotka	kódové slovo
a	01
b	111
c	0
d	011

Mějme kódové řetězce:

1. 01001111101,
2. 010110111011,
3. 0111101101,
4. 111001010,
5. 0011000.

Pokuste se dekódovat jednotlivé řetězce. Rozhodněte, který kódový řetězec je jednoznačně dekódovatelný. Je tento kód jednoznačně dekódovatelný? Je jednoznačně dekódovatelný po znacích? Je tento kód prefixový, blokový či afixový?

Příklad 1.17
Mějme tři typy zpráv s četnostmi znaků definovanými následující tabulkou.

zdrojová jednotka	četnost 1	četnost 2	četnost 3
a	40	80	160
b	40	40	0
c	40	20	0
d	40	20	0

Určete entropii jednotlivých symbolů a průměrnou entropii pro oba typy zpráv.

Příklad 1.18
Předpokládejme zprávu s pravděpodobnostním rozdělením četností symbolů definovaným následující tabulkou.

zdrojová jednotka	pravděpodobnost
a	$\frac{1}{4}$
b	$\frac{1}{16}$
c	$\frac{1}{8}$
d	$\frac{1}{8}$
e	$\frac{1}{4}$
f	$\frac{1}{16}$
g	$\frac{1}{16}$
h	$\frac{1}{16}$

Určete entropii jednotlivých symbolů a průměrnou entropii jednoho symbolu. Jaká je entropie typické zprávy o délce 16 znaků. Jak taková zpráva vypadá?

Jaké jsou redundance (absolutní pro zprávu délky 160 znaků a relativní redundance) blokového kódu kódujícím jeden znak 8 bity a blokového kódu kódujícím jeden znak 3 bity.

Navrhněte kód s minimální redundancí.

Příklad 1.19
Mějme část černobílého obrázku:

$$X = bbbbwwwwwwbbbbbbbwwwwwwwwwwwbbbbbbbbbbbbwbbbbbwbbwbbbbb$$

Zakódujte jej pomocí RLE komprese. Zhodnoťte dosažený kompresní poměr. Diskutujte různé typy RLE.

1.3 Další příklady

Příklad 1.20
Dokažte, že každý prefixový kód je jednoznačně dekódovatelný.

Příklad 1.21
Mějme zprávu nad abecedou o pěti symbolech. Pro jaké pravděpodobnostní rozdělení symbolů nad touto abecedou je blokový kód délky 3 optimální?

Příklad 1.22
Dokažte, že neexistuje bezeztrátová komprese, která by každý řetězec zakódovala kratším kódem, než byla původní zpráva.

Příklad 1.23
Pro kódování znaků národních abeced se používá standard Unicode. Verze Unicode 4.0 obsahuje 96 382 znaků, mezi kterými jsou interpunkční znaménka, matematické a technické symboly, geometrické obrazce, apod. Standard Unicode je možné rozšiřovat o nové znaky, např. symbol pro měnu Euro byl přidán ve verzi 2.1. Významnou vlastností Unicode je možnost skládání složitějších symbolů z jednotlivých částí (háček + c = č), nebo z jednotlivých znaků (české ch = c+h).

Tento standard definuje symboly na základě jejich významu, nerozlišuje mezi jejich různými zápisy - grafickými reprezentacemi. Definice znaků jsou obohaceny o další vlastnosti, např. lexikografické uspořádání, proto nejde pouze o kódování.

Mezi základní reprezentace Unicode patří kódování UCS-4[5] a UCS-2, který obsahuje pouze prvních 64K znaků[6]. Protože ani nové nástroje nedokáží efektivně pracovat s UCS-4 (Java používá UCS-2) byly definovány kódy pro transformaci UCS, nazvané UTF[7] Mezi významné patří UTF-8, UTF-16 a UTF-32.

Následující tabulka ukazuje převodní vztah mezi UCS-4 ($\sim$ UTF-32) a kódováním UTF-8.

UCS-4 kód od - do	Binární zápis znaku v UTF-8
0000 0000 - 0000 007F[8]	0xxxxxxx
0000 0080 - 0000 07FF [9]	110xxxxx 10xxxxxx
0000 0800 - 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000 - 001F FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000 - 03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000 - 7FFF FFFF	1111110x 10xxxxxx ... 10xxxxxx

[5]Universal Character Set (UCS), definováno normou ISO/IEC 10646-1
[6]Kóduje základní sadu - Basic Multilingual Plane (BMP)
[7]UCS transformation formats (UTF).

Kódování UTF-8 je „zpětně kompatibilní“, tedy všechny znaky, které byly obsaženy v sadě ASCII mají tentýž kód, ve starších editorech jsou tedy zobrazeny korektně. České znaky jsou reprezentovány pomocí 16 bitů.

Předpokládejme existenci zprávy $X =$ Žába leze do bezu, já tam za ní polezu.

Kolik bitů obsadí tato zpráva v UCS-4, kolik v UTF-8 a kolik v rozšířeném ASCII (např. ISO 8859-2, osm bitů). Sedmibitový ASCII nemůže reprezentovat „nabodeníčka“. Některé systémy umožňují přepis takových symbolů pomocí speciálních sekvencí. LaTeX by tuto zprávu přepsal do tvaru $X =$ \v{Z}{\'a}ba leze do bezu, j{\'a} tam za n{\'\i{}} polezu.

Kolik bitů spotřebuje tento přepis s použitím ASCII(7 bitů)? Jaký je kompresní poměr mezi nejúspornější a nejhorší reprezentací této zprávy?

2 Reprezenatce celých čísel

V metodách komprese dat je častou úlohou reprezentace celých čísel, která reprezentují počet opakování, index ve slovníku, apod.

2.1 Blokový kód

Pro počítače je přirozená reprezentace celých čísel pomocí pevně dané délky, která je většinou násobkem délky jednoho bytu. V současné době (přelom dvacátého a dvacátého prvního století) je pro reprezentaci celého čísla používáno 32 nebo 64 bitů. To nám umožní reprezentovat $2^{32} = 4294967296 \simeq 4{\cdot}10^9$ (nebo $2^{64} \simeq 1,8{\cdot}10^{19}$) různých hodnot.

Příklad 2.1
Mějme zprávu obsahující nezáporná celá čísla. Určete, kolik bitů by musel mít blokový kód, aby mohl reprezentovat tuto zprávu, pokud víme, že největší číslo ve zprávě je

1. 8,
2. 127,
3. 2 500 000.

Počet bitů lze určit pomocí následujícího vztahu:

$$b = \lceil \log_2 n \rceil,$$

kde n odpovídá počtu možných hodnot. Protože nezáporná celá čísla menší nebo rovna 8 připouští hodnoty $\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$, tedy devět hodnot je pro první případ potřeba 4 bity. Obdobně pro druhý případ 7 bitů a pro třetí 22 bitů.

Blokový kód je optimální pro reprezentaci hodnot, které mají ve zprávě stejnou pravděpodobnost a jejichž počet je celočíselnou mocninou čísla 2.

2.2 Kódování s oddělovačem

Kódování s oddělovačem (byte coding, comma coding) rozděluje každý blok (obvykle 8 nebo 16 bitů) kódového slova na dvě části, část, která nese informaci. Druhou částí je jeden bit označující konec kódového slova. Původní číslo vznikne zřetězením významových částí.

Příklad 2.2
Pomocí kódování s oddělovačem zakódujte čísla:

1. 0,

2. 203,

3. 2 500 000.

Pro kódování použijte blok o délce jeden byte, bitem 1 označte poslední blok.

Nejprve musíme kódované číslo vyjádřit pomocí binární reprezentace, poté jej rozdělíme po 7 bitech a zakódujeme. Celý proces shrnuje následující tabulka:

číslo	binární reprezentace	kód s oddělovačem
0	0	00000001
203	11001011	00000010 10010111
2 500 000	10011000100101101000000	00000010 00110000 10010110 01000001

Tento kód umožňuje přirozeným způsobem zakódování čísla 0. Obdobně je tomu i pro jiné kódy, např. Golombovy, ale protože budeme uvažovat kódování přirozených čísel (celá čísla větší než nula), nebudeme nulu kódovat a celý kód o jedničku posuneme.

2.3 Fibonacciho kódy

Fibonacciho kódy jsou prefixové kódy, které umožňují kódovat celá kladná čísla. Apostolico a Fraenkel v roce 1987 představili kód založený na Fibonnaciho číslech druhého řádu. Fibonacciho čísla jsou pojmenovány po italském matematikovi Leonardu z Pisy[10] (1175 - 1250), známém též jako Fibonacci (filius Bonacci, syn Bonacci*ho*). První zmínka o těchto číslech (maatraameru) však spadá do 5 století před naším letopočtem[11], tato čísla zavedl indický matematik Pingala ve svém díle Chhandah-shastra[12].

Tento kód zobrazuje malá čísla na krátká kódová slova. Každé Fibonacciho číslo končí řetězcem „11“. Tento řetězec se nevyskytuje nikde v kódovém slově. 2.1.

Fibonacciho kód se vytváří pomocí Fibonacciho čísel, která jsou definována rekurzívně:

- $F_1 = 1$;
- $F_2 = 1$;
- $F_i = F_{i-1} + F_{i-2}$; for $i > 2$

Posloupnost Fibonacciho čísel začíná čísly 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 ....

[10] Zasloužil o zavedení původem arabské desítkové soustavy.

[11] V roce 509 př. n. l. byla v Římě svržena monarchie a založena republika.

[12] V tomto díle je mimo jiné i první zmínka o čísle nula (označované .), o Pascalově trojúhelníku a zavedení binární číselné soustavy.

Fibonacci zavedl tuto posloupnost jako ideální řadu pro množení králíků. Limita $\frac{F_n}{F_{n-1}}$ pro n jdoucí do nekonečna je číslo označované jako zlatý řez ($\frac{1+\sqrt{5}}{2}$), číslo významné v umění, biologii a výpočetní technice.

Každé přirozené číslo n může být kódováno binární reprezentací:

$$RF(n) = \sum_{i=1}^{k} b_i F_{i+1},$$

kde $b_i \in \{0, 1\}$, $k < n$, $F_i, 2 \leq i \leq k$ je $i-$té Fibonacciho číslo. Tato reprezentace neobsahuje dvě jedničky vedle sebe (jsou nahrazeny jednou na vyšším „řádu“).

Fibonacciho kód[13] čísla n je:

$$F^2(n) = b_0 b_1 b_2 b_3 \dots b_k 1,$$

kde $b_0 b_1 b_2 b_3 \dots b_k$ je reverzí reprezentace $RF(n)$.

Pro dekódování původního čísla se odtrhne poslední jednička a původní číslo se vypočte podle vzorce:

$$\sum_{i=1}^{|cw|} b_i F_{i+1},$$

kde F_i je $i-$té Fibonacciho číslo a b_i je $i-$tý bit v kódovém slově cw.

2.4 Fibonacciho kódy vyšších řádů

Pro urychlení růstu Fibonacciho posloupnosti může být definice Fibonacciho čísel rozšířena použitím delší posloupnosti čísel (k předchozích čísel) pro výpočet Fibonacciho čísla. Tato čísla jsou nazývána Fibonacciho čísla k-tého řádu. Tato čísla byla intenzivně studována V. Schlegelem.

Posloupnost Fibonacciho čísel třetího řádu (též nazývána Tribonacci numbers) začíná 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705 ....

Yamamoto a Ochi (1991) odvodili Fibonacciho kódy vyšších řádů. Pro jejich odvození je potřeba nejprve definovat Fibonacciho čísla vyšších řádů:

- $F_i^r = 0$, jestliže $i \leq 0$,
- $F_i^r = 1$, jestliže $i = 1$,
- $F_i^r = \sum_{j=i-r}^{i-1} F_j^r$, jestliže $i \geq 2$.

Dále označme S_i^r i-tý součet Fibonacciho čísel stupně r:

$$S_i^r = \sum_{j=1}^{i} F_j^r, i \geq 1$$

[13] Budeme jej označovat $F^2(n)$, protože dále zavedeme Fibonacciho kódy založené na Fibonacciho číslech vyšších řádů

n						$RF(n)$			$Fib^2(n)$	$\lvert Fib^2(n)\rvert$
1								1	11	2
2							1	0	011	3
3						1	0	0	0011	4
4						1	0	1	1011	4
5					1	0	0	0	00011	5
6					1	0	0	1	10011	5
7					1	0	1	0	01011	5
8				1	0	0	0	0	000011	6
50	1	0	1	0	0	1	0	0	001001011	9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1 000 000								...	$0^8 1010^{11} 1010^3 1^2$	30
	34	21	13	8	5	3	2	1		

Tabulka 2.1: Fibonacciho kódy pro malá čísla. Číslo n vznikne součtem RF koeficientů násobených příslušným Fibonacciho číslem, jak je zobrazeno na posledním řádku. Výsledný kód $Fib^2(n)$ vznikne otočením kódu $RF(n)$ a jeho prodloužením jedničkou.

Dále označme M_i^r i-tý součet součtů Fibonacciho čísel stupně r:

$$M_i^r = \sum_{j=1}^{i} S_j^r, i \geq 1$$

Výsledný binární kód stupně r je:

- $Fib^r(n) = 01^{r-1}$, jestliže $n = 1$
- $Fib^r(n) = 0Fib^r(n - S_{i-1}^{r-1})$, jestliže $M_{i-1}^{r-1} < n \leq M_{i-1}^{r-1} + F_i^{r-1}$
- $Fib^r(n) = 1Fib^r(n - S_i^{r-1})$, jestliže $M_{i-1}^{r-1} + F_i^{r-1} < n \leq M_i^{r-1}$

Příklad 2.3
Určete $Fib^3(31)$. Pro určení Fibonacciho kódu třetího řádu jsou důležité Fibonacciho čísla druhého řádu, jejich součty a součty těchto součtů (tzv. mega součty). Nejbližší nižší mega součet M_{i-1}^2 je číslo 26. Příslušné Fibonacciho číslo (F_i^2) se nachází na dalším řádku a je to číslo 8. Protože $26 + 8 \geq 31$ bude výsledný kód začínat nulou a zbytek kódu bude $Fib^3(31 - 12)$, protože $S_{i-1}^2 = 12$. Bereme tedy součet ze stejného řádku jako mega součet. Pokud bychom kódovali jedničku, odečítali bychom součet z následujícího řádku. Lze tedy psát: $Fib^3(31) = 0Fib^3(31 - 12) = 0Fib^3(19)$.

n	S_n^2	M_n^2	$Fib^3(n)$
1	1	1	011
2	2	3	0 011
3	4	7	1 011
4	7	14	0 0 011
5	12	26	0 1 011
6	20	46	1 0 011
7	33	79	1 1 011
8	54	133	0 0 0 011
50	32951280098	86267571219	0 0 0 1 0 0 011
			↓ ↓ ↓↙ ↓ ↓
			20 12 7 4 2 1

Tabulka 2.2: Fibonacciho kódy třetího řádu pro malá čísla. Významové nuly značí, že do výsledného součtu se započítává částečný součet S_x^2, jedničky značí, že do výsledného součtu se započítává částečný součet z vyšší úrovně.

n	F_n^2	S_n^2	M_n^2	F_n^3	S_n^3	M_n^3	$Fib^3(n)$	$Fib^4(n)$
1	1	1	1	1	1	1	011	0111
2	1	2	3	1	2	3	0 011	0 0111
3	2	4	7	2	4	7	1 011	1 0111
4	3	7	14	4	8	15	00 011	00 0111
5	5	12	26	7	15	30	01 011	01 0111
6	8	20	46	13	28	58	10 011	10 0111
7	13	33	79	24	52	110	11 011	11 0111
8	21	54	133	44	96	206	000 011	000 0111

Tabulka 2.3: Fibonacciho čísla, jejich součty a Fibonacci kódy třetího a čtvrtého stupně pro malá čísla

Číslo 19 je větší než $14 = M_4^2$ a zároveň je menší či rovno číslu $19 = M_4^2 + F_5^2$, proto bude další bit opět 0 a budeme odečítat $S_4^2 = 7$. Tedy $Fib^3(31) = 0Fib^3(19) = 00Fib^3(19-7) = 00Fib^3(12)$.

Pro zakódování čísla 12 jsou rozhodující čísla $7 = M_3^2$ a $10 = M_3^2 + F_4^2$. Protože 12 je více než 10, bude dalším bitem 1 a od čísla 12 se bude odečítat $S_4^2 = 7$. Stejným postupem získáme výsledný kód:

$$Fib^3(31) = 0Fib^3(19) = 00Fib^3(12) = 001Fib^3(12-7) = 001Fib^3(5) =$$
$$= 0010Fib^3(3_{=5-2}) = 00101Fib^3(1_{=3-2}) = 00101\ 011$$

Příklad 2.4
$Fib^3(50) = 000100\,011$.

číslo	unární kód	délka unárního kódu
1	1	1
2	01	2
3	001	3
4	0001	4
5	00001	5
⋮	⋮	⋮
1 000 001	$0^{1\,000\,000}1$	1 000 001

Tabulka 2.4: Příklad unárního kódu

Délka Fibonacciho kódu stupně r je $L_{Fib^r}(n) = i + r - 1$, $M_{i-1}^{r-1} < n \leq M_i^{r-1}$.

Z Fibonacciho kódu $Fib^r(n) = b_m b_{m-1} \dots b_2 b_1 01^{r-1}$ lze číslo n dekódovat pomocí:

$$n = 1 + \sum_{i=1}^{m} S_{i+b_i}^{r-1}.$$

Příklad 2.5
$000100\,011 \longrightarrow 1 + (S_{1=1+0}^2 + S_2^2 + S_{4=3+1}^2 + S_{4=4+0}^2 + S_5^2 + S_6^2) = 1 + (1 + 2 + 7 + 7 + 12 + 20) = 50$. Tabulka 2.3 ukazuje příklady čísel potřebných pro výpočet Fibonacciho čísel r-tého stupně.

2.5 Unární kód

Unární kód kóduje nezáporné celé číslo i pomocí bitového řetězce $0^{i-1}1$. Tento kód je optimální pro pravděpodobnostní rozdělení splňující

$$p(i) = 2^{-(i+1)}.$$

Příklady unárního kódu pro malá čísla ukazuje tabulka 2.4.

2.6 Golombovo kódování

Golombovo kódování je pojmenováno po svém tvůrci Solomonu Wolfu Golombovi, který toto kódování představil v roce 1966. Toto kódování je optimální pro zdrojové jednotky s geometrickým rozložením, tedy malé hodnoty jsou mnohem více pravděpodobnější než vyšší hodnoty.

Používá nastavitelný parametr b pro rozdělení vstupní hodnoty na dvě části: výsledek po dělení číslem b, a zbytek po tomto dělení.

$$q = \lfloor \frac{n-1}{b} \rfloor, \quad r = n - qb - 1$$

číslo	$b = 3$	$b = 5$
1	0|0	0|00
2	0|10	0|01
3	0|11	0|10
4	10|0	0|110
5	10|10	0|111
6	10|11	10|00
7	110|0	10|01
8	110|10	10|10
⋮	⋮	⋮
1 000 000	$1^{333\,333}0\|0$	$1^{199\,999}0\|111$

Tabulka 2.5: Příklady Golombových kódů pro dva různé parametry $b = 3$ a $b = 5$

První částí kódu je unární kód čísla $q + 1$ (zde jsou zaměněny jedničky a nuly). Následuje zbytek po dělení v zkráceném binárním kódování. Výběr $b = 3$ generuje tři možné zbytky od 0 do 2, které jsou zakódovány pomocí řetězců 0, 10 a 11. Pro volbu zbytku 5 vznikají zbytky od 0 do 4, které jsou postupně kódovány řetězci 00, 01, 10, 110 a 111.

Tabulka 2.5 ukazuje Golombovo kódování pro dvě různé hodnoty koeficientu b (3 a 5). Obě části kódu jsou v tabulce 2.5 odděleny symbolem |. Je vidět, že tento kód je nepoužitelný pro velká čísla.

2.7 Riceovo kódování

Toto kódování poprvé popsal Robert Rice v 1979. Jde o speciální případ Golombova kódování, kde je koeficient b volen jako mocnina dvou. Tento kód je velice rychle kódovatelný a dekódovatelý na počítačích. Pro uložení zbytku lze s výhodou použít blokový kód.

Příklad 2.6
Pomocí Riceových kódů s parametry 4 a 16 zakódujte číslo 50. Číslo 50 lze zapsat jako $\mathbf{1} + 12{\cdot}4 + 1$, výsledný kód je $Rice_4(50) = 1111111111110|01$.

Obdobně pro koeficient 16 dostáváme $50 = \mathbf{1} + 3{\cdot}16 + 1$, $Rice_{16}(50) = 1110|0001$.

2.8 Eliasovy kódy

Tyto kódy jako první popsal sovětský vědec Levenstein (1968), ale pozdější, Eliasův (1975) popis je zejména v anglické literatuře citován.

Eliasův $\alpha(i)$ kód je unární kód.

číslo i	$\gamma(i)$	$\gamma'(i)$	δ'
1	1	1	1
2	001	01 0	001 0
3	011	01 1	001 1
4	00001	001 00	011 00
5	00011	001 01	011 01
6	01001	001 10	011 10
7	01011	001 11	011 11
8	0000001	0001 000	00001 000
⋮	⋮	⋮	⋮
1 000 000	$39bits$	$0^{19}1\ 1^3010^410^210^6$	$000010011\ 1^3010^410^210^6$

Tabulka 2.6: Příklady Eliasových γ, γ' a δ' kódů pro malá čísla

Příklad 2.7
Eliasův $\alpha(i)$ kód čísla 50 je $0^{49}1$. Jeho délka je 50 bitů.

Eliasův $\beta(i)$ kód přirozená binární reprezentace čísla i začínající první významnou jedničkou. Tento kód není jednoznačně dekódovatelný, např. $\beta(6) = 110 = \beta(1)\beta(2)$.

Příklad 2.8
Eliasův $\beta(i)$ kód čísla 50 je 110010. Jeho délka je 6 bitů. Tento kód neumí reprezentovat číslo 0.

Eliasův $\beta'(i)$ kód vznikne z Eliasova $\beta(i)$ kódu odtržením nejvýznamnější jedničky. Formálněji:

- $\beta'(0) = \varepsilon$
- $\beta'(2i) = \beta'(i)0$
- $\beta'(2i+1) = \beta'(i)1$

Eliasův $\beta'(i)$ kód je nejednoznačný stejně jako $\beta(i)$ kód.

Příklad 2.9
Eliasův $\beta'(i)$ kód pro číslo 50 je 10010. Jeho délka je 5 bitů.

Eliasův $\gamma(i)$ kód zapisuje jednotlivé bity $\beta'(i)$ kódu a každý uvede bitovým příznakem. Poslední příznak je jednička, ostatní jsou nuly. Na tento kód se dá nahlížet jako na blokový kód s oddělovačem, kde délka bloku je jedna.

Eliasův $\gamma'(i)$ kód vznikne permutací $\gamma(i)$ kódu, s bitovými příznaky tvořícími unární kód umístěnými před datové bity ($\beta'(i)$ kód).

Formálně:

$$\gamma'(i) = \alpha(|\beta(i)|)\beta'(i)$$

Příklad 2.10
Pomocí Eliasova $\gamma'(i)$ kódu zakódujte číslo 38.

1. Nejprve převedeme číslo 38 do dvojkové soustavy a odtrhneme první jedničku, čímž vytvoříme Eliasův $\beta'(38)$ kód, tedy 00110.

2. Počet bitů tohoto kódu je 5, $\alpha(5) = 00001$. Výsledný Eliasův $\gamma'(i)$ kód vznikne zřetězením obou kódů. Výsledný kód je:

   $$\overbrace{00000}^{}\underbrace{1}_{}00110$$

   Všimněte si, že výsledný kód má lichou délku, uprostřed se nachází jednička, která uzavírá unární kód délky druhé části a zároveň může být považována za první bit dvojkového zápisu původního čísla.

Příklad 2.11
Mějme binární řetězec 000000100100101110001.... Předpokládejme, že reprezentuje posloupnost čísel v γ' kódu. Dekódujte první číslo.

1. Kdyby řetězec začínal jedničkou, pak je tato jednička celým kódem a reprezentuje číslo 1.

2. V našem případě kód začíná číslicí 0, proto spočteme počet nul před první jedničkou, zde jich je šest.

3. Tento počet zvětšíme o jedna (zde 7) a tolik bitů vyjmeme(1001001) a převedeme z dvojkové soustavy. Původní číslo bylo 73 a jeho kód byl 0000001001001.

Další příklady γ a γ' kódů ukazuje tabulka 2.6. Tyto kódy reprezentují číslo i pomocí $2\lceil log(i) \rceil$ bitů.

Eliasův $\gamma(i)$ kód je dlouhý pro velká čísla, což je způsobeno reprezentací délky β' kódu pomocí α (unárního) kódu. **Eliasův** $\delta(i)$ **kód** používá pro délku β kódu γ kód. Formálně:

$$\delta(i) = \gamma(|\beta(i)|)\beta'(i)$$

Obdobně **Eliasův** $\delta'(i)$ **kód** používá pro délku β kódu γ' kód. Formálně:

$$\delta'(i) = \gamma(|\beta(i)|)\beta'(i)$$

Příklad 2.12
Eliasův $\gamma'(i)$ kód pro číslo 50 je 000001 10010. Jeho permutací vznikne Eliasův $\gamma(50)$ kód, 01000001001. Délka každého tohoto kódu je 11 bitů.

číslo i	$\omega(i)$	$\omega'(i)$
1	0	0
2	$\overline{10}\,0$	$\overline{010}\,0$
3	$\overline{11}\,0$	$\overline{011}\,0$
4	$\overline{10}\,\overline{100}\,0$	$\overline{100}\,0$
7	$\overline{10}\,\overline{111}\,0$	$\overline{111}\,0$
8	$\overline{11}\,\overline{1000}\,0$	$\overline{011}\,\overline{1000}\,0$
15	$\overline{11}\,\overline{1111}\,0$	$\overline{011}\,\overline{1111}\,0$
16	$\overline{10}\,\overline{100}\,\overline{10000}\,0$	$\overline{100}\,\overline{10000}\,0$
$\vdots$	$\vdots$	$\vdots$
1 000 000	$\overline{10}\,\overline{101}\,\overline{10011}\,\overline{1^4010^4010^210^6}\,0$	$\overline{101}\,\overline{10100}\,\overline{1^4010^4010^210^6}\,0$

Tabulka 2.7: Příklady Eliasových ω a ω' kódů

Eliasův $\delta'(1\,000\,000)$ je 000010011 11101000010010000000 a má délku 28 bitů místo 39 bitů $\gamma(1\,000\,000)$.

Dalšími Eliasovými kódy jsou Eliasův ω a ω' kód. Příklady těchto kódů ukazuje tabulka 2.7. $\omega(i)$ kód je ukončen nulou. Tuto nulu předchází $\beta(i)$ kód čísla i. Další části výsledného kódu jsou β kódy reprezentující délku následujících částí. Tato rekurze končí první (nejkratší) částí, která je dlouhá 2 bity (ω) popřípadě 3 bity (ω'). Tyto reprezentace jsou pro velká čísla úspornější než ostatní Eliasovy kódy, např. $\omega(1\ 000\ 000)$ zabírá 30 bitů a $\omega'(1\ 000\ 000)$ pouze 28 bitů.

Příklad 2.13
Eliasův $\omega(i)$ kód čísla 50 je 10 101 110010 0. Eliasův $\omega'(50)$ je 101 110010 0. Eliasův $\omega(50)$ zabírá 12 bitů, délka $\omega'(50)$ kódu je 10 bitů.

Příklad 2.14
Dekódujte posloupnost čísel uloženou v řetězci

2.9 Trojkový kód s oddělovačem

Všechny prezentované kódy používaly binární kódování. Použijme páry bitů pro reprezentaci čtyř hodnot, prvků množiny $\{0, 1, 2,$ oddělovač$\}$. Pak pomocí binárního kódu můžeme lehce reprezentovat posloupnost trojkových čísel oddělených oddělovačem. Tento kód je označován trojkový kód s oddělovačem (ternary comma code). S touto myšlenkou přišel v roce 1993 Australan Peter Fenwick. Tabulka 2.8 ukazuje příklady trojkových kódů s oddělovačem.

Příklad 2.15
Pomocí trojkového kódu s oddělovačem zakódujte číslo 52. Trojkový kód

číslo i	trojkový kód(i) s oddělovačem	bitů
1	,	2
2	0,	4
3	1,	4
4	2,	4
5	10,	6
6	11,	6
7	12,	6
8	21,	6
⋮	⋮	⋮
1 000 000	1212210201222,	28

Tabulka 2.8: Příklady trojkových kódů s oddělovačem

kód c	1	2	5	10	10^2	10^3	10^4	10^5	10^6
binární	1	2	3	4	7	10	14	17	20
Golombův$_{b=3}$	2	3	4	5	35	335	3 335	33 335	333 335
Golombův$_{b=5}$	3	3	4	5	23	203	2 003	20 003	200 003
Fibonacciho	2	3	5	6	11	16	18	23	30
unární	2	3	6	11	101	1 001	10 001	100 001	1 000 001
Eliasův γ	1	3	5	7	13	19	27	33	39
Eliasův δ'	1	4	5	8	11	16	20	25	28
Eliasův ω	1	3	6	7	13	17	21	28	31
Fib^3	3	4	5	6	10	15	19	24	29
Fib^4	4	5	6	7	10	14	18	22	25
trojk. s odd.	4	4	6	8	12	16	20	24	28

Tabulka 2.9: Délky kódových slov pro vybraná čísla

s oddělovačem je posunutý o 2, proto nejprve převedeme číslo 50 (50 = 52 - 2) do trojkové soustavy (1212) a přidáme oddělovač Délka výsledného kódu je 10 bitů, protože pro kódování jednotlivých symbolů abecedy $\{0, 1, 2,$oddělovač$\}$ musíme použít dvou bitů.

2.10 Porovnání kódů

Tabulka 2.9 ukazuje délky kódů pro reprezentaci celých nezáporných čísel. Nejlepší je samozřejmě binární kód, ale není jednoznačně dekódovatelný. Unární a Golombovy kódy jsou použitelné pro malá čísla, jejich délka výrazně roste s rostoucí zdrojovou jednotkou.

2.11 Příklady na cvičení

Příklad 2.16
Mějme zprávu obsahující kladná celá čísla. Určete, kolik bitů by musel mít blokový kód, aby mohl reprezentovat tuto zprávu, pokud víme, že největší číslo ve zprávě je

1. 1,
2. 16,
3. 524,
4. 3 333 333.

Příklad 2.17
Odhad počtu částic ve známém vesmíru je po několik málo posledních desetiletí 10^{82}. Kolik bitů by musel mít blokový kód, abychom mohli očíslovat každou částici?

Příklad 2.18
Vyplňte tabulky v příloze.

Příklad 2.19
Mějme tyto zprávy obsahující posloupnosti čísel:

1. 0, 1, 1, 0, 2, 0, 0, 0, 0, 2, 0, 1, 3, 1
2. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
3. 10, 20, 30, 40, 50, 60, 70, 80, 90, 100

Zjistěte kolik bitů zabere zakódování těchto zpráv pomocí těchto kódování:

1. unárního,
2. blokového,
3. trojkového s oddělovačem,
4. Fibonacciho,
5. Fibonacciho třetího řádu,
6. Fibonacciho čtvrtého řádu,
7. Eliasova γ,

8. Eliasova δ',
9. Eliasova ω,
10. Eliasova ω'.

Které kódování je nejlepší pro jednotlivé zprávy?

2.12 Další příklady

Příklad 2.20
Princip kódování čísel pomocí jiné číselné soustavy uplatněný u trojkového kódu s oddělovačem lze použít i pro další číselné soustavy. Jako základ použijte soustavu o základu 7. Pro zakódování jednotlivých číslic a oddělovače použijte tento kód:

hodnota	kód	hodnota	kód
0	000	4	100
1	001	5	101
2	010	6	110
3	011	odd.	111

Zakódujte čísla od nuly do 20 a dále čísla 100, 1000 a 1000 000.

Příklad 2.21
Výše zmíněný princip lze použít pro binární abecedu rozšířenou o oddělovač. Porovnejte následující kódy:

hodnota	kód1	kód2
0	0	01
1	10	00
oddělovač	11	1

Zakódujte čísla od nuly do 20 a dále čísla 100, 1000 a 1000 000.

Příklad 2.22
Ve cvičení 1.19 jsme měli část černobílého obrázku:

$$X = bbbbwwwwwbbbbbbbwwwwwwwwwwwbbbbbbbbbbbbwbbbbbwbbwbbbbb.$$

Protože se pravidelně střídají části černé a bílé, lze použít RLE kompresi, která bude kódovat pouze délky jednotlivých úseků. Výsledná posloupnost bude $X' = 4, 6, 7, 11, 12, 1, 5, 1, 2, 1, 5$. Jaké kódování čísel je pro tuto zprávu nejvhodnější a jakého kompresního poměru tímto kódováním dosáhneme?

4 Statistické metody komprese dat

Statistické metody komprese dat vycházejí z modelu první úrovně, který předpokládá, že pravděpodobnosti zdrojových jednotek ve zprávě jsou různé a nezávislé na výskytu okolních symbolů.

4.1 Shannon-Fanovo kódování

V roce 1949 publikovali na sobě nezávisle Claude Elwood Shannon s Warrenem Weaverem a Robert Mario Fano metodu na vytvoření prefixového kódu proměnné délky pro množinu symbolů a jejich četností. Tato metoda je označována jako Shannon-Fanovo kódování. Způsob vytvoření Shannon-Fanova kódu popisuje algoritmus 4.1.

Algoritmus 4.1 Shannon-Fanovo kódování

Vstup: vstupní text nad abecedou zdrojových jednotek S;
Výstup: kód $K = (S, C, f)$, kódové slovo pro každý symbol ze vstupního textu;

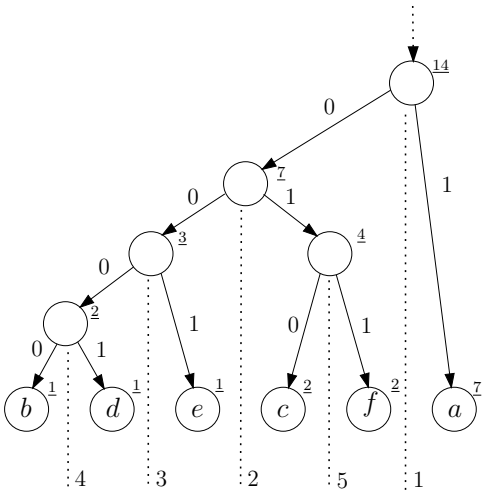
1: pro každé $x \in S : f(x) \leftarrow \varepsilon$;
2: zjisti četnosti jednotlivých symbolů;
3: symboly seřaď do neklesající posloupnosti s podle jejich četností;
4: $Split(s)$;

Procedura **Split**(**s**):

1: **if** $|s| > 1$ **then**
2: rozděl s na s_1 a s_2 se součtem četností nejbližšími polovině;
3: **for all** $x \in s_1$ **do**
4: $f(x) \leftarrow f(x)\mathbf{0}$
5: **end for**
6: **for all** $x \in s_2$ **do**
7: $f(x) \leftarrow f(x)\mathbf{1}$
8: **end for**
9: $Split(s_1)$;
10: $Split(s_2)$;
11: **end if**

Příklad 4.1
Pomocí Shannon-Fanova kódování zakódujte zprávu $X = abcdefafaaacaa$. Shannon-Fanův kód je reprezentován stromem vybudovaným nad symboly vstupní abecedy seřazenými do neklesající posloupnosti podle jejich četností. Tento strom vznikne rekurzivním dělením této posloupnosti na dvě části s četnostmi nejbližšími polovině. Výsledný strom je zobrazen na obrázku 4.1. Jednotlivé kroky dělení jsou znázorněny pomocí tečkovaných čar.

Dělení číslo tři a čtyři mohla být provedena v opačném pořadí, na délku výsledného kódu nemá toto pořadí vliv.



Obrázek 4.1: Shannon-Fanův kód pro vstupní text z příkladu 4.1

Tento strom je vhodný pro dekódování, není však příliš vhodný pro kódování, pro které je lepší převodní tabulka zobrazená v tabulce 4.1.

Symbol, $x \in S$	počet	Shannon-Fanův kód, $f(x)$
a	7	1
b	1	0000
c	2	010
d	1	0001
e	1	001
f	2	011

Tabulka 4.1: Převodní tabulka pro Shannon-Fanův z příkladu 4.1

Vstupní text $X = abcdefafaaacaa$ je zakódován do posloupnosti $Y =$ 1 0000 010 0001 001 011 1 011 1 1 1 010 1 1. Délka výsledného kódu je 30 bitů. K této délce by bylo nutné přidat režii vzniklou při ukládání kódového stromu, popřípadě informací nutných k jeho vytvoření.

4.2 Statické Huffmanovo kódování

David A. Huffman v roce 1952 publikoval metodu na vytvoření kódů s minimální redundancí. Tato metoda vytváří kódovací strom opačným směrem než Shannon-Fanova metoda, tedy směrem od spoda nahoru. Tento strom je nazýván Huffmanovým stromem. Algoritmus 4.2 popisuje vytváření tohoto stromu.

Algoritmus 4.2 Statické Huffmanovo kódování

Vstup: n zdrojových jednotek s pravděpodobnostmi $p(i)$, $1 \leq i \leq n$, $\sum_{i=1}^{n} p(i) = 1$.

Výstup: n kódových slov, automat $M = (Q, \{0,1\}, \delta, q_0, F)$.

1: $Q \leftarrow \emptyset$; $F \leftarrow \emptyset$;
2: **for** $i = 1$ to n **do**
3: vytvoř nový stav q_i;
4: $Q \leftarrow Q \cup \{q_i\}$; $F \leftarrow F \cup \{q_i\}$; $p(q_i) \leftarrow p(i)$;
5: **end for**
{Stav (list Huffmanova stromu) q_i zdrojové jednotky i je označen svou pravděpodobností $p(i)$.}
6: $k \leftarrow n$;
7: **while** $k \geq 2$ **do**
8: najdi dva stavy r, s; $r \neq s$ s nejmenšími pravděpodobnostmi $p(r)$, $p(s)$;
9: vytvoř nový stav q;
10: $Q \leftarrow Q \cup \{q\}$; $p(q) \leftarrow p(r) + p(s)$;
11: $\delta \leftarrow \delta \cup \{\delta(q, \mathbf{0}) \rightarrow r,\ \delta(q, \mathbf{1}) \rightarrow s\}$;
12: $p(r) \leftarrow 0$; $p(s) \leftarrow 0$;
13: **if** k=2 **then**
14: označ stav q jako počáteční;
15: **end if**
16: $k \leftarrow k - 1$;
17: **end while**
18: kódové slovo $f(i)$ pro každou zdrojovou jednotku i je označení přechodů na cestě od počátečního stavu do stavu q_i, $1 \leq i \leq n$;

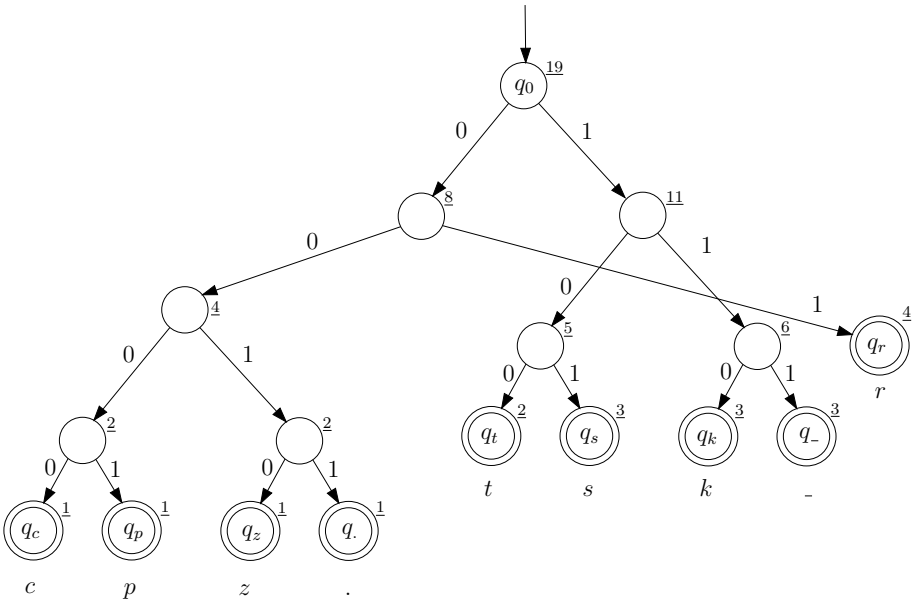
Místo pravděpodobností zdrojových jednotek lze použít počty opakování zdrojových jednotek. Výsledek bude stejný.

Příklad 4.2

Mějme text $X = strč\_prst\_skrz\_krk$. Zakódujte jej s použitím statického Huffmanova kódování. Zhodnoťte kompresi. Následující tabulka ukazuje shrnutí zdrojových jednotek s jejich frekvencemi ve zdrojovém textu.

Zdrojová jednotka	frekvence	Huffmanův kód
s	3	101
t	2	100
r	4	01
$č$	1	0000
_	3	0001
p	1	0010
z	1	110
k	3	111
.	1	0011

Tato tabulka je využita při konstrukci Huffmanova stromu. Prvním krokem konstrukce je vytvoření stavů pro zdrojové jednotky. Tyto stavy jsou ohodnoceny příslušnými počty opakování.Výsledný Huffmanův strom ukazuje obrázek 4.2. Počty opakování jsou znázorněny pomocí malých, podtržených čísel. Výsledné kódy vzniknou přečtením cesty od počátečního stavu do listu odpovídajícího zdrojové jednotce.



Obrázek 4.2: Huffmanův strom vzniklý algoritmem 4.2. Množina zdrojových jednotek je $S = \{s, t, r, č, \_, p, z, k, .\}$

Zdrojová zpráva $X = str č\_prst\_skrz\_krk$ bude s použitím kódovací tabulky převedeno na zprávu $Y = 10110001000011100010110110011110111001001011111100110011100011$.

Délka tohoto kódu je 57 bitů. Předpokládejme, že pro původní zprávu byl použit standardní osmibitový kód. Potom je délka původní zprávy 19

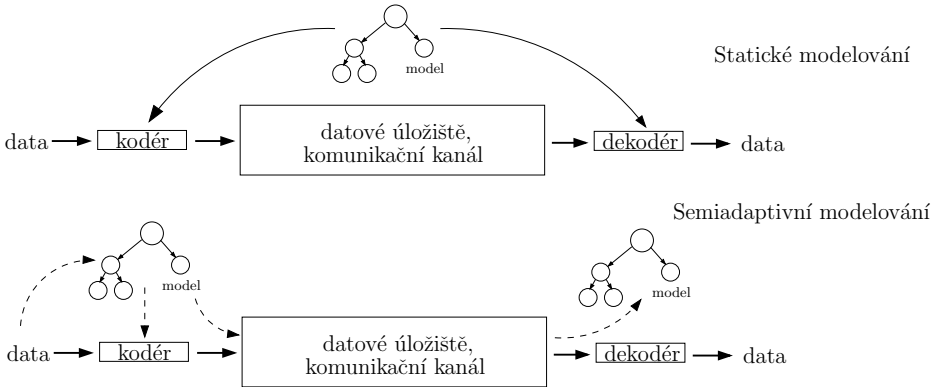
bytů, 152 bitů. Zanedbejme potřebu uložení Huffmanova stromu, pak bude kompresní poměr $cr = \frac{57}{152} = 0,375$.

4.3 Modelování komprese dat

Statický model komprese dat předpokládá, že kodér i dekodér používají stejný model dat, v našem případě Huffmanův strom. Toho lze dosáhnout v případě, že je tento model pro velké množství dat neměnný, například pro české texty ve velké knihovně.

Častějším případem je nutnost přizpůsobení modelu vstupním datům. Pak je nutné provést jeden průchod vstupním textem, abychom napočítali četnosti jednotlivých symbolů, které slouží pro konstrukci modelu - Huffmanova stromu. Druhým průchodem se provede komprese dat. Tento model je označován jako semiadaptivní.

Porovnání obou typů modelování znázorňuje obrázek 4.3



Obrázek 4.3: Statické a semiadaptivní modelování

Další možností je vytváření modelu během čtení vstupních dat, model se během komprese vyvíjí. Dekodér vytváří model na základě dekódovaných dat. Tento model je adaptivní. Adaptivní variantou Huffmanova kódu se zabývá následující kapitola.

4.3.1 Uložení Huffmanova stromu

Příklad 4.3
Předpokládejme semiadaptivní verzi statického Huffmanova kódování z příkladu 4.2. Zakódujte příslušný Huffmanův strom.

1. První možností pro přenesení Huffmanova stromu je přenesení zdrojových jednotek spolu s jejich četnostmi. Dekodér by měl dostatek informací pro jeho vytvoření. Výsledný kód by začínal počtem zdrojových

jednotek následovaný páry (zdrojová jednotka, počet opakování). Pro kódování čísel použijme Fib^3 kód. Výsledek bude $Fib^3(9)$ *č* $Fib^3(1)$ *p* $Fib^3(1)$ *z* $Fib^3(1)$. $Fib^3(1)$ *r* $Fib^3(4)$ *t* $Fib^3(2)$ *s* $Fib^3(3)$ *k* $Fib^3(3)$ _ $Fib^3(3)$. Při použití osmi bitů na jednu zdrojovou jednotku bude délka kódu pro Huffmanův strom $9 \cdot 8 + 39 = 111$ bitů. Tento způsob je velmi neefektivní pro velké hodnoty četností jednotlivých symbolů.

2. Prostorově výhodnější je metoda využívající výpis Huffmanova stromu do bitového proudu bez potřeby ukládání počtů opakování. Implementace začíná počtem zdrojových jednotek následovaných jejich výčtem. Pořadí musí odpovídat jejich umístění v Huffmanově stromu, zleva doprava. Dále následuje rekuzivní výpis ohodnocení hran stromu, kdy se nejprve vypisuje levý podstrom a potom pravý. Náš příklad produkuje následující výstup $Fib^3(9)$*č p z . r t s k* _0000110111001101. Vlastní struktura stromu zabírá pouze 16 bitů. Celková délka kódu pro strom je $5 + 9 \cdot 8 + 16 = 93$ bitů. Výsledný kompresní poměr je $\frac{57+93}{152} \cong 0.99$. S rostoucí délkou kódované zprávy klesá neblahý vliv nutnosti uložení Huffmanova stromu na výsledný kompresní poměr.

4.4 Adaptivní Huffmanovo kódování

Adaptivní verzi Huffmanova kódování poprvé na sobě nezávisle navrhli Faller (1973) a Robert G. Gallager (1978). Donald E. Knuth v roce 1985 tento algoritmus vylepšil, výsledná varianta adaptivního Huffmanova kódování je nazývána FGK algoritmem.

Jeff Vitter tuto metodu v roce 1987 vylepšil (po zakódování jednoho symbolu dojde nejvýše k jedné úpravě kódového stromu oproti až $log_2 n$). Tato metoda je složitější.

FGK algoritmus adaptivního Huffmanova kódování popisuje algoritmus 4.1. Celý algoritmus se opírá o významnou vlastnost Huffmanových stromů, kterou je sourozenecká vlastnost. Tato vlastnost je pro binární stromy definována takto:

- každý uzel kromě kořene má sourozence,
- existuje uspořádání uzlů v pořadí neklesajícího ohodnocení tak, že každý uzel sousedící v seznamu s nějakým uzlem je jeho sourozenec (leví synové na lichých místech a praví synové na sudých místech v seznamu).

Takové uspořádání je na obrázku 4.4 v části III zobrazeno lomenou tečkovanou šipkou.

Příklad 4.4
Text $X = abbaacddc$ zakódujte pomocí adaptivního Huffmanova kódování.

Algoritmus 4.3 Adaptivní Huffmanovo kódování

Vstup: vstupní text nad abecedou zdrojových jednotek S;
Výstup: zakódovaný text;

```
1: vytvoř strom obsahující pouze uzel zero;
2: while není zakódován celý text do
3:    c ← další symbol vstupního textu;
4:    if první výskyt symbolu c then
5:       kód prodluž o kód uzlu zero;
6:       zakóduj c;
7:       uzel zero nahraď novým uzlem s následníky zero a novým uzlem u
         pro symbol c, ohodnocení u ← 0;
8:       aktualizujStrom(u);
9:    else
10:      kód prodluž o kód uzlu c;
11:      aktualizujStrom(c)
12:   end if
13: end while
```

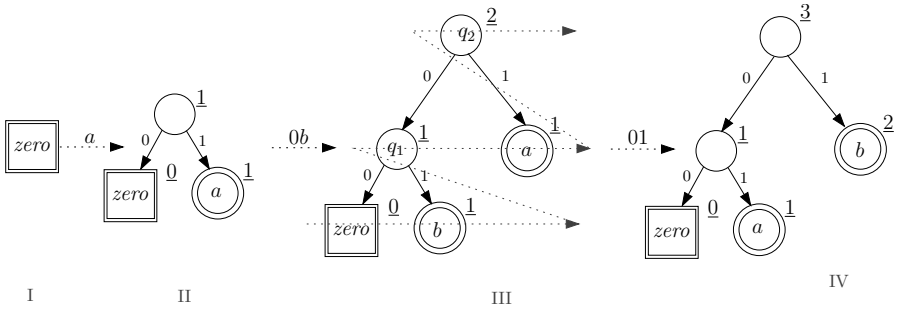
Procedura **aktualizujStrom(uzel u)**:

```
1: while u není kořen do
2:    if ∃ uzel u' se stejným ohodnocením výše v pořadí sourozenců, pokud
      jich je více, vyber ten nejvýše položený then
3:       prohoď uzly u a u';
4:       zvyš ohodnocení u o 1;
5:       u ← předek(u);
6:       aktualizujStrom(u);
7:    end if
8:    zvyš ohodnocení u o 1;
9: end while
```

Kódovat budeme pomocí FGK varianty popsané algoritmem 4.3. Celé kódování začíná s Huffmanovým stromem obsahujícím pouze uzel *zero*. Nejprve je do výstupu poslán kód prvního znaku, pro který je použit nějaký jednoznačně dekódovatelný kód, na kterém se dohodne kodér i dekodér. Takovým kódem je například ASCII kód.

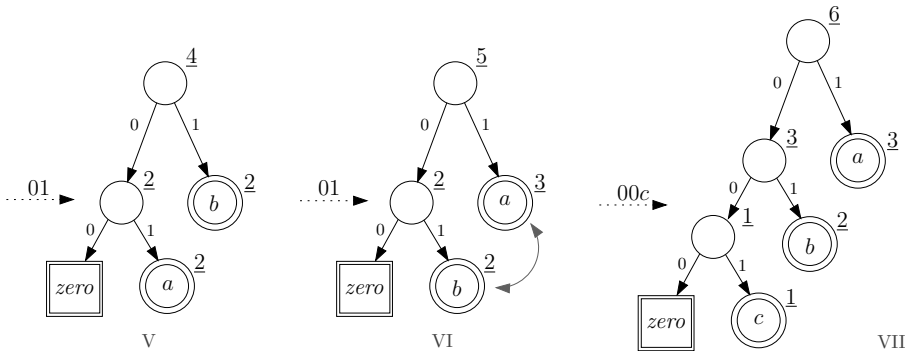
Po odeslání symbolu a je upraven Huffmanův strom nahrazením uzlu *zero* novým uzlem se dvěma potomky, uzlem *zero* a novým uzlem odpovídajícím symbolu a a s ohodnocením (četností symbolů) 1. Situace po zakódování symbolu a ukazuje obrázek 4.4, část II. Ohodnocení uzlu je zobrazeno pomocí podtrženého čísla u uzlu.

Dalším symbolem vstupního tetu je symbol b. I toto je zatím nezakódovaný symbol, proto je do výstupu poslán kód uzlu *zero* následovaný kódem nového symbolu. Opět dojde k rozdělení uzlu *zero*.



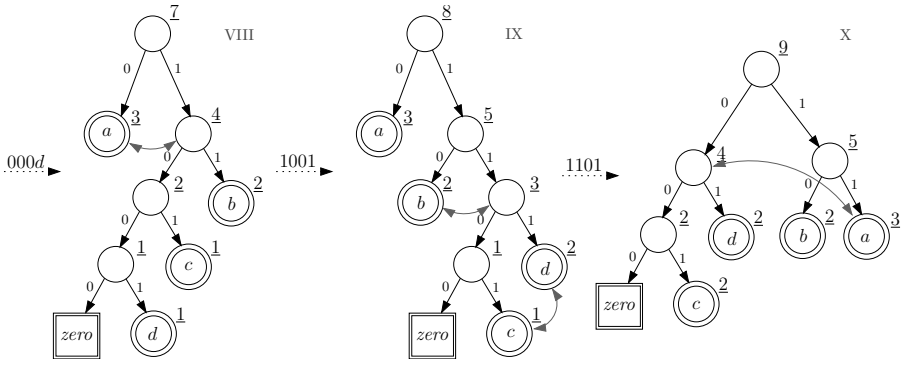
Obrázek 4.4: První čtyři kroky kódování FGK adaptivního Huffmanova kódování řetězce $X = \mathbf{abb}aacddc$ z příkladu 4.4

Dalším symbolem ve vstupu je opět b. Tento symbol je již ve stromu obsažen, do výstupu je poslán jeho kód, tedy 01. V této fázi kódování dojde k prohození uzlů. Jelikož jsme kódovali symbol b, budeme u tohoto uzlu zvyšovat ohodnocení, ale musíme zajistit, že výsledný strom neporuší sourozeneckou vlastnost. Uspořádání uzlů do neklesající posloupnosti je zobrazeno v části III obrázku 4.4 tečkovanou lomenou čarou. Toto uspořádání včetně četností je $zero0$, $b1$, $q_1 1$, $a1$, $q_2 2$. Poslední vkládaný symbol je b, jeho četnost je 1. V této posloupnosti napravo od symbolu b jsou dva uzly s četností 1, uzel q_1 a a. Vybereme ten bližší kořeni, tedy a a oba uzly prohodíme. Navýšíme ohodnocení uzlu b a přesuneme se k jeho předkovi. Tím je kořen stromu, aktualizace stromu končí navýšením jeho četnosti.



Obrázek 4.5: Další kroky kódování FGK adaptivního Huffmanova kódování řetězce $X = abb\mathbf{aac}ddc$ z příkladu 4.4

Obdobným způsobem jsou zakódovány další tři symboly vstupního řetězce. Modifikace Huffmanova stromu během těchto kroků ukazuje obrázek 4.5. Závěr kódování shrnuje obrázek 4.6. Všimněte si, že ve fázi IX dojde během kódování jednoho symbolu k dvěma záměnám podstromů. Varianta FGK připouští možnost až $log_2 n$ záměn během kódování jednoho symbolu.



Obrázek 4.6: Poslední kroky kódování FGK adaptivního Huffmanova kódování řetězce $X = abbaac\mathbf{ddc}$ z příkladu 4.4

Příklad 4.5
Zhodnoťte dosažený kompresní poměr v příkladu 4.4.

Vstupní text je $X = abbaacddc$, jeho délka 9 symbolů. Při použití rozšířeného ASCII kódu s osmi bity na symbol je výsledná délka vstupního textu $9{\cdot}8 = 72$ bitů.

Zakódovaná zpráva je $Y = a0b01010100c000d10011101$. Pokud pro kódování symbolů použijeme stejný kód jako pro vstupní text, bude délka zakódované zprávy $4{\cdot}8 + 20 = 52$ bitů. Výsledný kompresní poměr je $\frac{52}{72} \doteq 0,72$.

4.5 Příklady na cvičení

Příklad 4.6
Mějme zprávu s následujícími četnostmi jednotlivých symbolů:

Symbol, $x \in S$	počet
a	105
b	49
c	42
d	42
e	35

Vypočtěte entropii jednotlivých symbolů.

Pro danou zprávu vytvořte Shannon-Fanův a Huffmanův kód. Porovnejte jejich redundanci nad danou zprávou. Jaké jsou kompresní poměry pro oba kódy. Diskutujte možnosti implementace obou stromů.

Příklad 4.7
Předpokládejme kódy z příkladu 4.6. Dekódujte tento fragment zprávy $Y =$ 01010 01110 10111 00101 10110 11101 00101 00000 01011 ... Jak se změní dekódovaná zpráva, pokud zaměníme první bit za jedničku?

Příklad 4.8
Mějme zprávu s následujícími pravděpodobnostmi jednotlivých symbolů:

Symbol, $x \in S$	počet
a	0.35
b	0.17
c	0.17
d	0.16
e	0.15

Vypočtěte entropii jednotlivých symbolů.

Pro danou zprávu vytvořte Shannon-Fanův a Huffmanův kód. Porovnejte jejich redundanci nad danou zprávou. Jaké jsou kompresní poměry pro oba kódy. Diskutujte možnosti implementace obou stromů.

Příklad 4.9
Zprávu $X = mamamelemaso$ zakódujte pomocí adaptivního Huffmanova kódování. Zhodnoťte dosaženou kompresi.

Příklad 4.10
Kód $Y = b10a0100c110$ dekódujte pomocí adaptivního Huffmanova kódování. Zhodnoťte dosaženou kompresi.

Příklad 4.11
Zprávu $X = babaaabba$ zakódujte pomocí adaptivního Huffmanova kódování. Zhodnoťte dosaženou kompresi. Stejnou zprávu zakódujte pomocí statického Huffmanova kódování. Porovnejte kompresní poměry.

Příklad 4.12
Pomocí adaptivního Huffmanova kódování zakódujte zprávu $X = a^{30}bbc$. Zhodnoťte dosaženou kompresi. Stejnou zprávu zakódujte pomocí statického Huffmanova kódování. Porovnejte kompresní poměry.

4.6 Další příklady

Příklad 4.13
Pomocí Huffmanova kódování zakódujte zprávu $X = abcdefafaaacaa$. Porovnejte délku výsledného kódu s Shannon-Fanovým kódem z příkladu 4.1.

Příklad 4.14
Předpokládejme, že vstupní abeceda má 32 symbolů. Jaká je nejmenší délka vstupní zprávy, aby nad ní vytvořený Huffmanův strom měl maximální hloubku (odpovídá délce nejdelšího kódu):

1. 5,

2. 6,

3. 7,

4. 31?

Příklad 4.15
Zprávu $X = telemelesele$ zakódujte pomocí adaptivního Huffmanova kódování. Zhodnoťte dosaženou kompresi. Stejnou zprávu zakódujte pomocí statického Huffmanova kódování. Porovnejte kompresní poměry.

5 Aritmetické kódování

5.1 Statické aritmetické kódování

Huffmanovo statické kódování je optimální pokud pravděpodobnosti jednotlivých symbolů zápornými mocninami čísla dvě. Poté jsou entropie jednotlivých symbolů celočíselné a odpovídají délce příslušného Huffmanova kódu.

Pokud jsou však pravděpodobnosti jednotlivých symbolů nejsou zápornými mocninami čísla dvě, tak dochází při kódování jednotlivých symbolů ke vzniku redundance.

Příklad 5.1
Určete entropii symbolu a s pravděpodobností $p(a) = 0,3$.

Tento symbol má entropii $H(a) = -log_2(p(a)) \doteq 1,74$. Pokud bychom pro tento symbol použili kód o délce 2 bity, byla by redundance vzniklá tímto symbolem $0,26$ bitu.

Tuto redundanci lze snížit použitím aritmetického kódování, které nekóduje jednotlivé symboly, ale používá jeden kód pro zakódování celé zprávy. Kódem celé zprávy je pak jedno číslo z intervalu $< 0; 1)$. Postup kódování ukazuje algoritmus 5.1.

Algoritmus 5.1 Aritmetické kódování

Vstup: vstupní text nad abecedou zdrojových jednotek S;
Výstup: zakódovaný text;

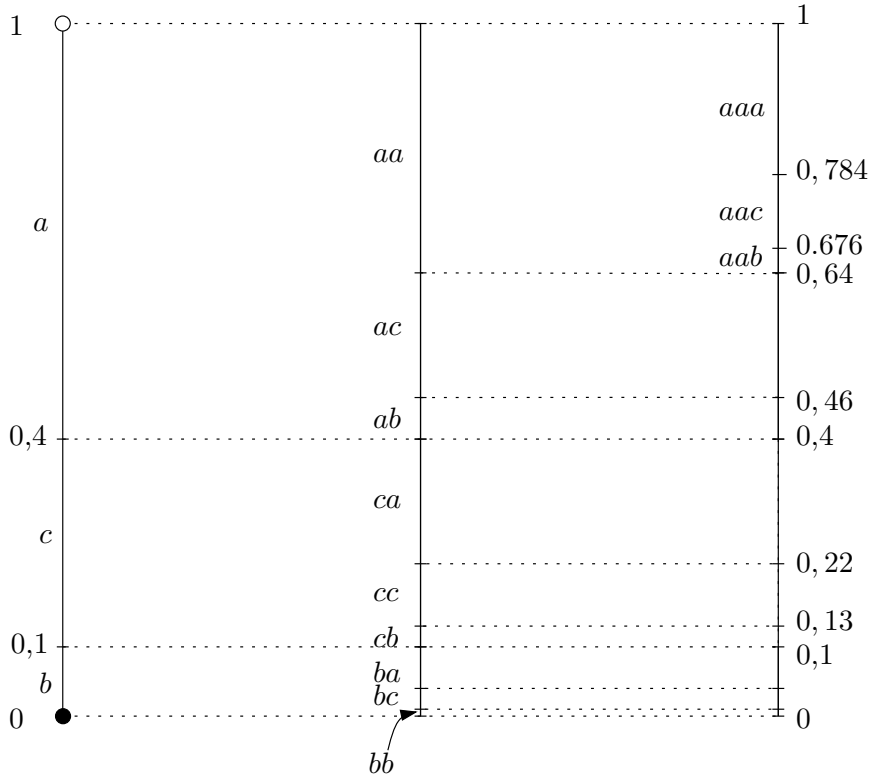
1: do výstupu ulož pravděpodobnosti nebo četnosti symbolů;
2: nastav interval $I \leftarrow < 0; 1)$;
3: **while** není zakódován celý text **do**
4: přečti další symbol c;
5: Rozděl interval I na podintervaly, jejichž velikosti jsou úměrné pravděpodobnostem symbolů;
6: $I \leftarrow$ podinterval odpovídající c;
7: **end while**
8: Výstupem je libovolné číslo z intervalu I;

Přestože výsledkem kódování je libovolné číslo z posledního intervalu, je ještě potřeba zakódovat informaci o ukončení kódu, protože dekodér musí vědět, kdy má skončit s dělením intervalu. Existují dvě varianty řešení tohoto problému:

1. před vlastním kódem čísla z intervalu je nějakým prefixovým kódem zakódován počet symbolů,

2. použít speciální symbol pro označení konce vstupu. Tento symbol bude mít nejnižší pravděpodobnost. Pokud na něj dekodér narazí, pozná, že dekódování končí.

Příklad 5.2
Mějme text nad abecedou $A = \{a, b, c\}$. Pravděpodobnosti jednotlivých symbolů jsou $p(a) = 0,6$, $p(b) = 0,1$, $p(c) = 0,3$. Ukažte jak se vytvářejí aritmetické kódy pro různé řetězce. Obrázek 5.1 ukazuje postupné dělení intervalu $< 0; 1)$ na podintervaly.



Obrázek 5.1: Dělení intervalu při aritmetickém kódování

Z obrázku je zřejmé, že velikost intervalu je přímo úměrná pravděpodobnostem odpovídající zprávy. Symbol a zabírá interval $I(a) =< 0,4; 1)$, jehož velikost je 0,6. Oproti tomu symbol b zabírá interval $I(a) =< 0; 0,1)$ o šířce 0,1. Šířky intervalů klesají s rostoucí délkou zprávy, $I(aaa) =< 0,784; 1)$, velikost 0,216; $I(bbb) =< 0; 0,001)$, velikost 0,001.

Přestože šířka intervalu odpovídá pravděpodobnosti zprávy, délka čísla nemusí úplně přesně záviset na šířce intervalu.

Nejkratšími binárními kódy čísel z intervalu $< 0; 1)$ jsou binární čísla $0,0_2 = 0$ a $0,1_2 = 0,5$. Všechna čísla z intervalu $< 0; 1)$ začínají 0, proto tuto nulu nekódujeme. Jediný bit může kódovat následující zprávy:

bit	číslo (dekadicky)	původní zpráva
0	0	*b*, *bb*, *bbb*, *bbbb*,...
1	0,5	*a*,*ac*, *acc*, *acca* ...

Na druhou stranu pro zakódování zprávy $X = aaa$ je potřeba třech bitů, 111, protože $0,111_2 = \frac{7}{8} = 0,875 \in < 0,784; 1)$.

Příklad 5.3
Text $X = abacaacbc$ zakódujte statickým aritmetickým kódem.

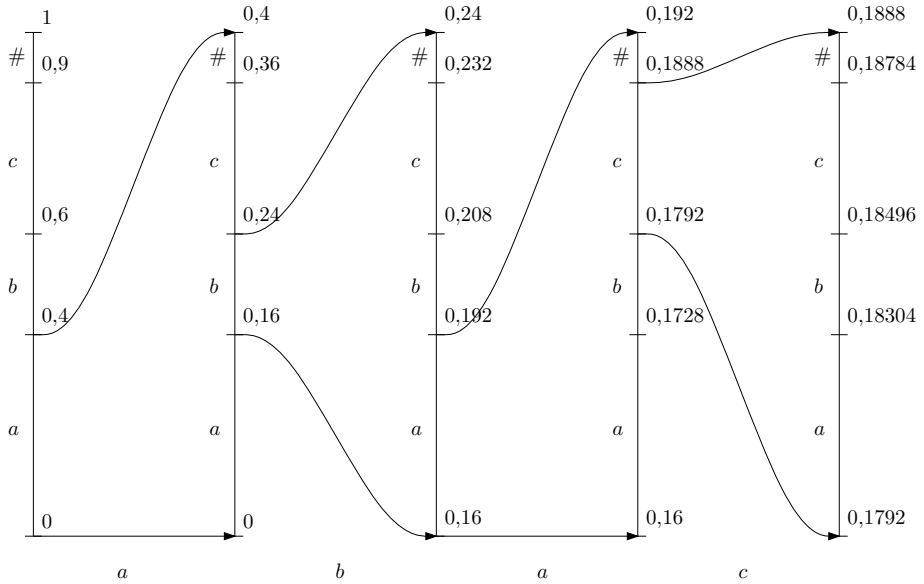
Text X rozšíříme o ukončovací symbol, $X' = abacaacbc\#$. Pro správnou funkci aritmetického kodéru si spočítáme pravděpodobnosti a kumulativní pravděpodobnosti jednotlivých symbolů. Ty nám poslouží jako základ pro dělení intervalu. Kumulativní pravděpodobnost zdrojové jednotky x v uspořádání $x_1, x_2, \ldots x_{|S|}$ získáme podle vztahu

$$cp_i = \sum_{j=1}^{i-1} p_j$$

Tyto hodnoty zobrazuje následující tabulka.

Symbol x_i	četnost $f(x_i)$	pravděpodobnost p_i	kumulativní p. cp_i	interval $I(x_i)$
a	4	$\frac{4}{10} = 0,4$	0	$< 0; 0,4)$
b	2	$\frac{2}{10} = 0,2$	0,4	$< 0,4; 0,6)$
c	3	$\frac{3}{10} = 0,3$	0,6	$< 0,6; 0,9)$
$\#$	1	$\frac{1}{10} = 0,1$	0,9	$< 0,9; 1)$

První kroky komprese řetězce $X' = abacaacbc\#$ jsou zobrazeny na obrázku 5.1.



Obrázek 5.2: První kroky kódování řetězce $X' = abacaacbc\#$

krok	symbol	*Low*	*Range*
0		0	1
1	a	0	0,4
2	b	0,16	0,8
3	a	0,16	0,032
4	c	0,1792	0,0096
5	a	0,1792	0,00384
6	a	0,1792	0,001536
7	c	0,1801216	0,0004608
8	b	0,18030592	0,00009216
9	c	0,180361216	0,000027648
10	#	0,1803860992	0,0000027648

Tabulka 5.1: Komprese řetězce $X' = abacaacbc\#$

Aritmetické kódování nemusí počítat všechny dělící hodnoty intervalu, stačí si pamatovat pouze dolní mez, označme ji *Low* a šířku intervalu, *Range*. Nové hodnoty se vypočítávají podle vztahů:

$$Low \leftarrow Low + Range \cdot cp_i$$

$$Range \leftarrow Range \cdot p_i$$

Jednotlivé kroky komprese shrnuje tabulka 5.1. Výsledný interval je $I(abacaacbc\#) =< 0,1803860992; 0,180388864)$. Vhodným kandidátem se jeví být číslo

$$0,1803874969482421875 = \frac{94575}{2^{19}} = 0,00101110001011011111_2.$$

Výsledný kód obsahující ukončovací znak je bez úschovy četností symbolů dlouhý 19 bitů.

5.2 Adaptivní aritmetické kódování

Tak jako statická verze Huffmanova kódování musí nejprve určit pravděpodobnosti jednotlivých symbolů, tyto frekvence předat dekodéru a teprve pak kódovat, musí toto provést i statická verze aritmetického kódování. Pokud opomineme možnost použití připraveného pravděpodobnostního modelu (např. pro jeden typ dat, knihy v češtině) je řešením adaptivní verze aritmetického kodéru.

Adaptivní verze aritmetického kodéru na základě pravděpodobností kóduje jednotlivé symboly stejně jako statické aritmetické kódování, ale model - pravděpodobnostní rozdělení se mění po zakódování každého symbolu.

	metoda 1				metoda 2			
Sym.	a	b	c, d, e, f, g	$\#$	a	b	c, d, e, f, g	$\#$
	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
a	$\frac{2}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{2}\cdot(1+\frac{1}{8}) = \frac{9}{16}$	$\frac{1}{16}$	$\frac{1}{2}\cdot\frac{1}{8} = \frac{1}{16}$	$\frac{1}{16}$
a	$\frac{3}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{3}\cdot(2+\frac{1}{8}) = \frac{17}{24}$	$\frac{1}{24}$	$\frac{1}{3}\cdot\frac{1}{8} = \frac{1}{24}$	$\frac{1}{24}$
b	$\frac{3}{11}$	$\frac{2}{11}$	$\frac{1}{11}$	$\frac{1}{11}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
a	$\frac{4}{12}$	$\frac{2}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
a	$\frac{5}{13}$	$\frac{2}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
b	$\frac{5}{14}$	$\frac{3}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
$\#$	$\frac{5}{15}$	$\frac{3}{15}$	$\frac{1}{15}$	$\frac{2}{15}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$

Tabulka 5.2: Vývoj pravděpodobností během aritmetického kódování

Příklad 5.4
Mějme text $X = aabaab\#$. Zakódujte jej pomocí adaptivního aritmetického kódování.

Předpokládejme, že vstupní abeceda obsahuje pouze symboly $S = \{a, b, c, d, e, f, g\}$.

Na začátku kódování mají všechny symboly vstupní abecedy stejnou pravděpodobnost. V našem případě, každý symbol zabírá $\frac{1}{|S|+1} = \frac{1}{8}$, protože jeden podinterval rezervujeme pro ukončovací znak #. Pokud bude interval pro symbol a první, tedy $< 0; 0,125$ bude tímto intervalem zakódován první symbol. Po jeho zakódování je nutné vypočítat nové pravděpodobnosti. Pro tento výpočet existuje mnoho způsobů, tady jsou dva základní:

1. Každý symbol měl na začátku četnost rovnu 1. Zakódování jednoho symbolu jeho četnost zvýší o jedna.

2. Četnost všech symbolů dohromady byla na začátku rovna 1. Zakódování jednoho symbolu jeho četnost zvýší o jedna.

První způsob výpočtu četností je jednodušší, druhý pružněji reaguje na vstupní text[14].

Postupné změny pravděpodobností jednotlivých symbolů během kódování řetězce $X = aabaab\#$ ukazuje tabulka 5.2.

5.3 Příklady na cvičení

Příklad 5.5
Text $aaab$ zkomprimujte pomocí aritmetického kódování. Zhodnoťte dosa-

[14]V praxi je nutné řešit otázky související s implementací v binárním kódu, zejména omezená přesnost čísel a omezený rozsah

ženou kompresi.

Příklad 5.6
Text *baaabaa#* zkomprimujte pomocí aritmetického kódování. Zhodnoťte dosaženou kompresi.

Příklad 5.7
Mějme pravděpodobnostní rozdělení symbolů z příkladu 5.6. Jaký řetězec kóduje číslo 0,0110?

Příklad 5.8
Text *cabccbabb#* zkomprimujte pomocí aritmetického kódování. Zhodnoťte dosaženou kompresi.

Příklad 5.9
Text *aaab* zkomprimujte pomocí adaptivního aritmetického kódování. Zhodnoťte dosaženou kompresi.

Příklad 5.10
Text *baaabaa#* zkomprimujte pomocí adaptivního aritmetického kódování. Zhodnoťte dosaženou kompresi.

Příklad 5.11
Mějme text obsahující symboly $S = \{a, b, c, \}$. Tento text je zakončen symbolem #. Jaký řetězec kóduje číslo 0,0110?

Příklad 5.12
Text *cabccbabb#* zkomprimujte pomocí adaptivního aritmetického kódování. Zhodnoťte dosaženou kompresi.

5.4 Další příklady

6 Slovníkové metody komprese dat

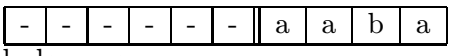
6.1 LZ77

Metoda LZ77 je nazývána metodou posuvného okna. Posuvné okno je použito pro zakódování textu. Je rozděleno na dvě části, zakódovanou část a nezakódovanou část. Dohromady tvoří celkovou velikost okna. Předpony z nezakódované části jsou kódovány pomocí podřetězců začínajících v zakódované části okna.

Do výstupního proudu je v každém kroku uložena trojice (i, j, a). Předpokládejme, že jsme nalezli nejdelší řetězec (označme jej s) začínající v zakódované části a shodující se s předponou řetězce z nezakódované části posuvného okna (označme jej p). Pak i je vzdálenost prvního znaku řetězce s od hranice mezi zakódovanou a nezakódovanou částí okna, j je délka řetězce s a a je první znak za předponou p. Po uložení trojice (i, j, a) je celé okno posunuto o $j + 1$ znaků doprava.

Příklad 6.1
Metodou LZ77 zakódujte text T = aabaacaaaaaababababbab. Použijte okno velikosti 10 znaků, velikost nezakódované části volte čtyři znaky. Zhodnoťte dosaženou kompresi.

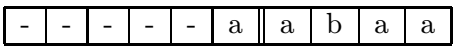
Kódování začíná s posuvným oknem položeným na text tak, že hranice mezi zakódovanou a nezakódovanou částí leží na začátku textu. V zakódované části se nenachází jediný znak, slovník pro vyhledávání je tedy prázdný. Tuto situaci ukazuje následující obrázek.



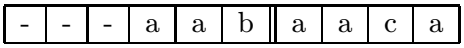
Trojice je (0,0,a), posun okna o jeden znak doprava.

Jelikož nejdelší předpona z nezakódované části začínající v zakódované části má délku 0, je do výstupního proudu vložena trojice (0, 0,a), kde a je první znak kódovaného řetězce. Celé okno je posunuto o jeden znak doprava.

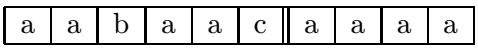
Kódování pokračuje v následujících krocích:



(1,1,b)



(3,2,c)



(3,2,a)

Nejdelší předpona v nezakódované části okna na předcházejícím obrázku byla aa. Tato předpona se nachází v zakódované části okna na dvou pozicích a to pozici 3 a pozici 6. Proto jsou dvě možnosti zakódování následující trojice : (3,2,a) a (6,2,a). Na jednoznačnost dekódování nemá výběr jedné

trojice vliv, nicméně volíme trojici (3,2,a), neboť obsahuje nižší čísla a dá se předpokládat, že budou kódována kratšími kódy.

a	a	c	a	a	a	a	a	a	b	(1,3,b)

Připomeňme, že nalezený řetězec nemusí končit v zakódované části okna. Na základě úmluvy z předcházejícího odstavce použijeme trojici (1,3,b) z možných trojic (1,3,b),(2,3,b) a (3,3,b).

a	a	a	a	a	b	a	b	a	b	(2,3,b)

Zde nastává situace, kdy je celá nezakódovaná část okna ve slovníku - v zakódované části, ale protože se do výstupního proudu musí dát jeden symbol (ten by byl právě za oknem), je do výstupního proudu dána trojice (2,3,b).

a	b	a	b	a	b	a	b	b	a	(2,2,b)

b	a	b	a	b	b	a	b	-	-	(3,2,b)

Závěr kódování. Nezakódovaná část okna není zcela zaplněna a poslední znak musí být uveden jako následující znak za nalezeným podřetězcem. Metoda LZ77 nepotřebuje ukončovací znak, protože binární kód každé trojice je delší než jeden byte.

Výsledný kód je tedy:

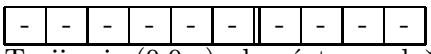
$$(0, 0, a)$$
$$(1, 1, b)$$
$$(3, 2, c)$$
$$(3, 2, a)$$
$$(1, 3, b)$$
$$(2, 3, b)$$
$$(2, 2, b)$$
$$(3, 1, b)$$

První číslo v uspořádané trojici adresuje nezakódovanou část okna, interval přípustných hodnot je tedy $< 0, 6 >$. Při použití binárního kódu bude potřeba pro uložení této adresy třech bitů.

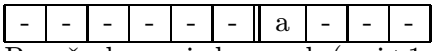
Druhé číslo určuje délku nalezeného podřetězce. Maximální délka je rovna délce nezakódované části okna zmenšené o jedničku (znak do výstupu), v našem případě tři znaky. Nejkratší délka je pro případ, kdy již první znak v nezakódované části okna není obsažen ve slovníku, pak je délka rovna nule. Tedy interval $< 0, 3 >$, binární kód délky dva. Pro kódování znaků použijeme standardní ASCII kód, tedy 8 bitů.

Celková délka kódu bude $7 * (3 + 2 + 8) = 91$ bitů. Byl-li původní text v ASCII kódování, byla jeho délka $22 * 8 = 176$ bitů. Kompresní poměr je $91/176 \doteq 0, 52$.

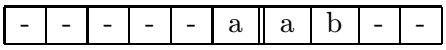
Příklad 6.2
Proveďte dekódování textu z předchozího příkladu. Postup je obdobný jako u kódování. Dekodér bude používat stejné okno, na základě indexů bude doplňovat znaky do výstupního proudu. Potom přesune okno tak, aby hranice mezi zakódovanou a nezakódovanou částí byla za posledním znakem. Vše je znázorněno na následujících obrázcích.



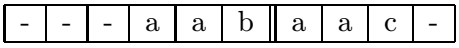
Trojice je (0,0,a), do výstupu vlož symbol a.



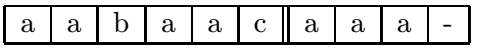
(1,1,b)
Posuň okno o jeden znak (o j+1, zde 0+1).



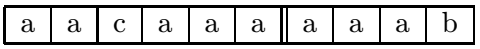
(1,1,b) Vlož a a přidej b.



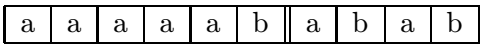
(3,2,c) Vlož aa a přidej c.



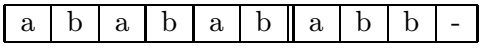
(3,2,a) Vlož aa a přidej a.



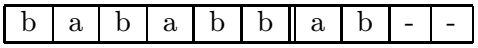
(1,3,b)



(2,3,b)



(2,2,b)



(3,1,b)

6.2 LZ78

Příklad 6.3
Metodou LZ78 zakódujte text T = aabaacaaaaaababababbab. Zhodnoťte dosaženou kompresi.

Metoda LZ78 produkuje v každém kroku jednu dvojici (i, a), kde i je index fráze ve slovníku a a je symbol nacházející se bezprostředně za ukládanou frází. Slovník je reprezentován jako strom s očíslovanými uzly, každý uzel reprezentuje frázi na cestě z kořene do tohoto uzlu. Po vložení jedné uspořádané dvojice do výstupního proudu je slovník rozšířen o novou frázi, která vznikne rozšířením ukládané fráze o právě ukládaný symbol. Tato fráze bude označena nejnižším novým číslem a začleněna do kódovacího stromu.

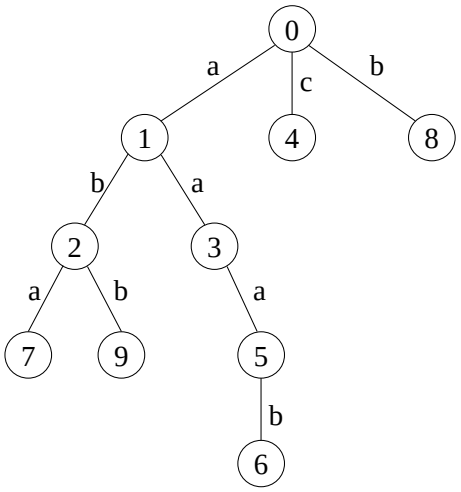
Kódování začíná se stromem obsahujícím jediný uzel, uzel nula. Nejdelší předpona nalezená v tomto stromu je ε, do výstupu jde dvojice (0,a) a strom

je rozšířen o uzel 1 a hranu ohodnocenou symbolem a. Uzel 1 tedy reprezentuje frázi a. Hranice mezi zakódovanou a neazakódovanou částí vstupního řetězce je posunuta:

$$a|abaacaaaaaababababbab$$

Protože nejdelší předpona nezakódované části řetězce nalezená v kódovacím stromu je a, tj. uzel 1, do výstupu se předá (1,b). Strom je rozšířen o novou frázi $2 = 1b = ab$. Hranice je posunuta za třetí symbol:

$$aab|aacaaaaaababababbab$$



Obrázek 6.1: Stromová reprezentace slovníku metody LZ78.

Algoritmus komprese postupně vytvoří strom na obrázku 6.1. Hranice mezi zakódovanou a nezakódovanou částí budou:

$$a|ab|aa|c|aaa|aaab|aba|b|abb|ab$$

Každý podřetězec uzavřený sousedícími hranicemi je kódovací fráze, její pořadí odpovídá číslu uzlu, jež ji reprezentuje. Výjimku tvoří poslední podřetězec, který zde nevytváří novou frázi a je kódován dvojicí (1,b), protože algoritmus dekódování vyžaduje vložení symbolu. Výsledný kód je:

Kód	velikost slovníku	počet bitů pro index fráze
(0,a)	0	0
(1,b)	1	1
(1,a)	2	2
(0,c)	3	2
(3,a)	4	3
(5,b)	5	3
(2,a)	6	3
(0,b)	7	3
(2,b)	8	4
(1,b)	9	4

Pro výpočet kompresního poměru předpokládejme, že původní text používal 8 bitů na jeden symbol, d0lka původního textu byla $22 * 8 = 176$ bitů. Jednotlivé symboly budeme kódovat také 8 bity. Indexy frází budeme kódovat binárním kódem jehož délka bude závislá na velikosti slovníku, tj. při kódování první dvojice, je slovník prázdný, není potřeba kódovat index fráze, po vložení první fráze je potřeba jeden bit (0 ... nová fráze, 1 ... fráze a). Celková délka kódu bude $10*8+1+2*2+4*3+2*4 = 105$ bitů. Kompresní poměr je $105/176 \doteq 0,60$.

Příklad 6.4
Dekódujte výstup z předchozího příkladu. Pro reprezentaci slovníku je při dekódování výhodnější tabulka. Její inicializace je:

Index fráze	fráze
0	ε

První dvojice je $(0, a)$, do výstupu je přidán řetězec odpovídající indexu fráze 0, tedy ε a symbol a. Tabulka frází je rozšířena o frázi vzniklou zřetězením fráze 0 a symbolu a, tedy a. Výsledná tabulka včetně dokódování bude

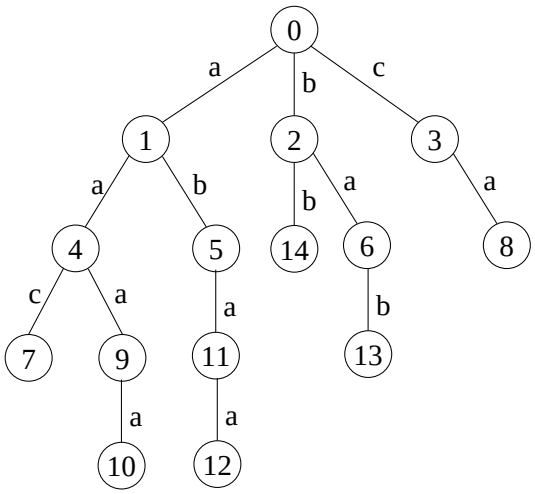
Index fráze	fráze	do výstupu
0	ε	
1	$0a \ldots a$	a
2	$1b \ldots ab$	ab
3	$1a \ldots aa$	aa
4	$0c \ldots c$	c
5	$3a \ldots aaa$	aaa
6	$5b \ldots aaab$	$aaab$
7	$2a \ldots aba$	aba
8	$0b \ldots b$	b
9	$2b \ldots abb$	abb
–	–	ab

6.3 LZW

Metoda LZW vymyšlená Welchem vychází z metody LZ78. Jejím výstupem jsou pouze indexy, slovník je inicializován všemi symboly vstupní abecedy, poslední symbol posledně vložené fráze je počátečním symbolem vkládané fráze.

Příklad 6.5
Metodou LZW zakódujte text $T = aabaacaaaaaababababbab$. Zhodnoťte dosaženou kompresi.

Metoda LZW předpokládá slovník inicializovaný všemi symboly abecedy. Pro jednoduchost předpokládejme, že vstupní abeceda obsahuje pouze symboly a, b, c[15]. Slovník bude obsahovat fráze a, b a c, indexované čísly $1, 2$ a 3. Celý slovník je zobrazen na obrázku 6.2.



Obrázek 6.2: Stromová reprezentace slovníku metody LZW.

Následující tabulka shrnuje jednotlivé kroky kódování. Symbolem | je označena nová hranice mezi zakódovanou a nezakódovanou částí textu, fráze ukládaná do slovníku je zvýrazněna tučně.

[15]Obecně nevíme jaké symboly budou v textu obsaženy, proto se slovník inicializuje všemi možnými symboly. Jinou možností by bylo zjišťování pravdy, které by si vyžádalo jeden průchod zakódovávaným textem a předání inicializovaného stromu dekodéru.

Text	index do výstupu	index nové fráze
a|**a***baacaaaaaaababababbab*	1	4
*a***a**|**b***aacaaaaaaababababbab*	1	5
*aa***b**|**a***acaaaaaaababababbab*	2	6
*aab***aa**|**c***aaaaaaababababbab*	4	7
*aabaa***c**|**a***aaaaababababbab*	3	8
*aabaac***aa**|**a***aaababababbab*	4	9
*aabaacaa***aaa**|**a***bababababbab*	9	10
*aabaacaaaaa***ab**|**a***bababbbab*	5	11
*aabaacaaaaaab***aba**|**b***abbab*	11	12
*aabaacaaaaaababa***ba**|**b***bab*	6	13
*aabaacaaaaaabababa***b**|**b***ab*	2	14
*aabaacaaaaaababababab***bab**|	13	–

Výsledná posloupnost indexů je 1,1,2,4,3,4,9,5,11,6,2,13. Pro kódování jednotlivých indexů použijeme binární kód, jehož délka bude $\lceil \log_2(sl) \rceil$, kde sl je velikost slovníku, tj. první a druhý index fráze budou zakódovány dvěma bity (inicializovaný slovník obsahuje 3 fráze). Délka výsledného kódu bude $2 * 2 + 4 * 3 + 6 * 4 = 40$ bitů. Pokud byl původní text v ASCII kódu, je kompresní poměr $40/176 \doteq 0,23$.

Příklad 6.6
Dekódujte posloupnost z předchozího příkladu. Uvažujte stejný počáteční slovník. Počáteční slovník obsahuje tři fráze:

Index fráze	fráze
1	*a*
2	*b*
3	*c*

Připomeňme, že každá nová fráze má první symbol shodný s posledním symbolem předchozí fráze.

Pro dekódování si je nutné uchovávat dva odkazy do textu, místo kam půjde další symbol (vždy za poslední symbol) a poslední symbol v poslední uložené frázi.

První index je číslo jedna, odpovídající fráze je a. Druhý index je jedna, odpovídající fráze je a, celý výstup je aa. Nová fráze je $\overline{aa}$, číslo 4. Tento a další kroky shrnuje následující tabulka, nová fráze je zvýrazněna pruhem nad dekódovaným textem, poslední dekódovaná fráze je zvýrazněna tučným písmem.

Index fráze ze vstupu	výstup	nová fráze
1	$\overline{a\mathbf{a}}$	$4 \ldots aa$
2	$aa\overline{\mathbf{b}}$	$5 \ldots ab$
4	$aa\overline{b\mathbf{aa}}$	$6 \ldots ba$
3	$aab\overline{aa\mathbf{c}}$	$7 \ldots aac$
4	$aabaa\overline{c\mathbf{aa}}$	$8 \ldots ca$

V tomto okamžiku je na vstupu index 9, tato fráze ještě nebyla vložena do slovníku. Tento případ nastává pro fráze typu axa, kde x je libovolný řetězec a fráze ax je již ve slovníku. Tato fráze je rozšířením posledně dekódované fráze. V našem případě jde o frázi $4 \ldots aa$, první a poslední symbol nové fráze se musí shodovat, fráze 9 bude aaa. Stejný problém bude dekodér řešit s frází 11.

Index fráze ze vstupu	výstup	nová fráze
9	$aabaac\overline{aa\mathbf{a}}\mathbf{aa}$	$9 \ldots aaa$
5	$aabaacaa\overline{a\mathbf{aaa}}\mathbf{b}$	$10 \ldots aaaa$
11	$aabaacaaaaaa\overline{b\mathbf{ba}}$	$11 \ldots aba$
6	$aabaacaaaaaab\overline{aba\mathbf{ba}}$	$12 \ldots abab$
2	$aabaacaaaaaababa\overline{ba\mathbf{b}}$	$13 \ldots bab$
13	$aabaacaaaaaababab\overline{b\mathbf{bab}}$	$14 \ldots bb$

6.4 Příklady na cvičení

Příklad 6.7
Text $X = aabaabbcabacaabcbaa$ zakódujte metodou LZ77. Velikost posuvného okna a nezakódované části volte:

velikost zakódované části	velikost nezakódované části
5	3
10	4
15	5

Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

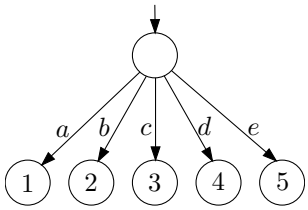
Příklad 6.8
Text $X = aaaaaaaaaabaaaaaaaaaa = a^{10}ba^{10}$ zakódujte metodou LZ77. Velikost posuvného okna a nezakódované části volte stejně jako v příkladu 6.7. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.9
Metoda LZ77 s velikostí zakódované části 6 symbolů a velikostí výhledu 4 symboly vytvořila výstup Y =(0,0,a)(0,0,b)(2,3,b)(2,1,a)(4,2,b)(4,2,b). Určete vstupní text.

Příklad 6.10
Text $X = aabaabbcabacaabcbaa$ zakódujte metodou LZ78. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.11
Text $X = aaaaaaaaaabaaaaaaaaaa = a^{10}ba^{10}$ zakódujte metodou LZ78. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.12
Metoda LZ78 vytvořila výstup $Y =$(0,a)(0,b)(1,b)(3,a)(1,a)(2,b)(5,b). Určete vstupní text.



Obrázek 6.3: Počáteční strom pro metodu LZW z příkladů 6.13, 6.14, 6.15, 6.16 a 6.19

Příklad 6.13
Text $X = aabaabbcabacaabcbaa$ zakódujte metodou LZW. Předpokládejme, že počáteční strom je stejný jako na obrázku 6.3. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.14
Text $X = aaaaaaaaaabaaaaaaaaaa = a^{10}ba^{10}$ zakódujte metodou LZW. Předpokládejme, že počáteční strom je stejný jako na obrázku 6.3. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.15
Text $X = aaaaaaaaaabaaaaaaaaaa = a^{10}ba^{10}$ zakódujte metodou LZW. Předpokládejme, že počáteční strom je stejný jako na obrázku 6.3. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.16
Text $X = aaaaaaaaaabaaaaaaaaaa = a^{10}ba^{10}$ zakódujte metodou LZW. Předpokládejme, že počáteční strom je stejný jako na obrázku 6.3. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

6.5 Další příklady

Příklad 6.17
Text $X = abababababababababab = (ab)^{10}$ zakódujte metodou LZ77. Velikost posuvného okna a nezakódované části volte stejně jako v příkladu 6.7. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.18
Text $X = abababababababababab = (ab)^{10}$ zakódujte metodou LZ78. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.19
Text $X = abababababababababab = (ab)^{10}$ zakódujte metodou LZW. Předpokládejme, že počáteční strom je stejný jako na obrázku 6.3. Zvolte vhodné kódování použitých čísel a zhodnoťte dosaženou kompresi.

Příklad 6.20
Určete, jak se změní kódování a výsledný kompresní poměr v příkladech 6.13, 6.14, 6.15, 6.16 a 6.19 pokud počáteční strom bude obsahovat pouze symboly, které se vyskytují v kódovaném textu.