V prednáške sa budeme venovať niektorým praktickým metódam kompresie dát.

1 Aritmetické kódovanie

Aritmetické kódovanie je alternatívou k Huffmanovmu kódovaniu. Odstraňuje niektoré jeho nedostatky – oddeľuje pravdepodobnostný model zdroja od procesu kódovania (preto sa dá ľahšie upraviť na adaptívnu verziu) a nevyžaduje celý počet bitov na kódovanie každého znaku (preto je v situáciach ako $p_a = \frac{1}{10}$, $p_b = \frac{9}{10}$ výhodnejšie). Spoločnou črtou aritmetického a Huffmanovho kódovania je rovnaký model zdroja – pravdepodobnosti výskytov znakov zdrojovej abecedy (nezávislé). Keďže pri kódovaní nevyužívajú žiadne kontextové informácie (pozičné závislosti znakov), zvyknú sa označovať ako „zero-order coders“. Do tejto skupiny patrí aj Shannonov-Fanov kód.

1.1 Kompresia (kódovanie)

Pri kompresii najskôr určíme pravdepodobnosti výskytu jednotlivých znakov zdrojovej abecedy. Nech $\{c_1, c_2, \dots c_n\}$ je zdrojová abeceda a nech $p_1, p_2, \dots p_n$ sú príslušné pravdepodobnosti. Proporčne, podľa pravdepodobností rozdelíme interval $\langle 0, 1)$ na n častí:

$$\begin{aligned} I_1 &= \langle 0, p_1) \\ I_2 &= \langle p_1, p_1 + p_2) \\ I_3 &= \langle p_1 + p_2, p_1 + p_2 + p_3) \\ &\vdots \\ I_n &= \langle p_1 + p_2 + \dots + p_{n-1}, 1). \end{aligned}$$

Označme $p'_k = \sum_{i=1}^{k} p_i$, pričom $p'_0 = 0$. Zároveň symbolom $|\langle d, h)|$ označíme dĺžku intervalu, t.j. hodnotu $h - d$. Algoritmus na začiatku vychádza z intervalu $I = \langle 0, 1)$. Po prečítaní prvého znaku sa interval zúži na príslušnú časť, podľa tohto znaku. Teda ak je znak na vstupe c_k, zúžime interval na I_k. Čítaním ďalších znakov naďalej interval zužujeme:

$$I = \langle d, h) \xmapsto{c_k} \langle d + p'_{k-1}|I|, d + p'_k|I|). \qquad (1)$$

Výstupom je ľubovoľné číslo z výsledného intervalu (najlepšie to, ktoré má najkratší zápis). Hlavná myšlienka spočíva v tom, že znaky, ktoré majú vysokú pravdepodobnosť, zužujú interval najmenej. Čím je interval menší, tým viac bitov potrebujeme na zápis niektorého z čísel, ktoré doň patria (očakávaný počet potrebných bitov je $\log_2 \frac{1}{|I|}$).

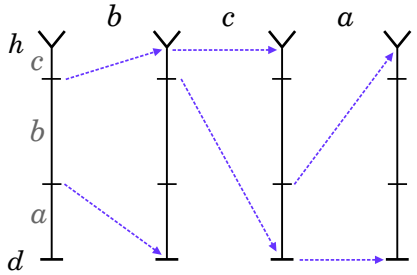
1. $I \leftarrow \langle 0, 1)$
2. pokiaľ nie sme na konci vstupu
   (a) načítame ďalší znak – c_k
   (b) upravíme interval I podľa (1)
3. dáme na výstup ľubovoľné číslo z intervalu I

Príklad: Nech vstupným textom je reťazec $aababbcbababcb$. Potom pravdepodobnosti výskytu jednotlivých znakov zdrojovej abecedy $\{a, b, c\}$ sú $p_a = \frac{5}{14}$, $p_b = \frac{7}{14}$ a $c = \frac{2}{14}$. Nasledujúca tabuľka ukazuje zmenu intervalu I počas spracúvania vstupu. Dolné a horné hranice sú počítané na 14 desatinných miest.

c	I
	$\langle 0\ ,\ 1)$
a	$\langle 0,00000000000000\ ,\ 0,35714285714286)$
a	$\langle 0,00000000000000\ ,\ 0,12755102040816)$
b	$\langle 0,04555393586006\ ,\ 0,10932944606414)$
a	$\langle 0,04555393586006\ ,\ 0,06833090379009)$
b	$\langle 0,05368856726364\ ,\ 0,06507705122865)$
b	$\langle 0,05775588296543\ ,\ 0,06345012494794)$
c	$\langle 0,06263666180758\ ,\ 0,06345012494794)$
b	$\langle 0,06292718435771\ ,\ 0,06333391592789)$
a	$\langle 0,06292718435771\ ,\ 0,06307244563277)$
b	$\langle 0,06297906338452\ ,\ 0,06305169402205)$
a	$\langle 0,06297906338452\ ,\ 0,06300500289792)$
b	$\langle 0,06298832749645\ ,\ 0,06300129725315)$
c	$\langle 0,06299944443076\ ,\ 0,06300129725315)$
b	$\langle 0,06300010615304\ ,\ 0,06300103256424)$

Potom napríklad číslo $0,063000679016\dots$ je z výsledného intervalu a jeho binárny rozvoj je $0,0001000000010000011101$. Teda výsledkom kódovania je uvedený dvadsaťbitový reťazec (bez 0 pred desatinnou čiarkou). Len pre porovnanie, Huffmanov kód potrebuje 21 bitov ($a = 10$, $b = 0$, $c = 11$).

Spracovanie vstupu bca ilustruje aj nasledujúci obrázok:



1.2 Dekompresia (dekódovanie)

Pri dekódovaní postupujeme analogicky, ako pri kódovaní. Na začiatku nastavíme interval $I = \langle 0, 1)$. Na vstupe máme číslo $x \in I$. Pre zistenie prvého znaku je potrebné určiť, v ktorom z potenciálnych n intervalov $I_1, \dots, I_n$ sa x nachádza. Príslušný interval (povedzme I_k) určuje znak (c_k) a zároveň umožní zúžiť I ($I \mapsto I_k$).

Podobne postupujeme ďalej. Pre aktuálny interval I určujeme k také, že x je prvkom intervalu, ktorý toto k „vyrobí“ z intervalu I podľa (1). Na výstup dáme c_k a zúžime I.

Otázkou je, ako zistiť, že sme už dekódovali posledný znak. Možné sú dve riešenia:

1. Pridáme do abecedy špeciálny znak „EOF“ (koniec súboru) a pri dekódovaní postupujeme až dovtedy, kým tento znak nedekódujeme.

2. Spolu s výsledným číslom dáme pri kódovaní na výstup aj dĺžku kódovaného reťazca. Teda dekóder skončí po vypísaní daného počtu znakov.

1.3 Implementačné poznámky

Aritmetické kódovanie môžeme efektívne realizovať pomocou celočíselnej aritmetiky. Interval $\langle 0, 1)$ nahradíme intervalom celých čísel, napríklad $\langle$0x0000, 0xffff) (vyjadrené hexadecimálne, teda $\langle 0, 65535)$. Základné pozorovanie, ktoré umožní ostať počas celého výpočtu len v tomto intervale: ak sa začiatočné čísla (napr. bity) dolnej a hornej hranice aktuálneho intervalu zhodujú, už ostanú rovnaké. Dôvod je prostý: interval sa počas kódovania zužuje, preto zhody na začiatku sa už nezmenia. To znamená, že ak takúto zhodu zaznamenáme, môžme ju z oboch hraníc intervalu odstrániť (a dať na výstup). Napríklad, ak po úprave intervalu dostaneme $h =$ 0x6807 a $d =$ 0x4af1, tieto sa zhodujú na prvých dvoch bitoch (01), ktoré dáme na výstup a následne upravíme hranice – $h =$ 0xa01f, $d =$ 0x2bc4. Hornú hranicu dopĺňame jednotkami a dolnú nulami. Prirodzene, dekóder musí pri práci s intervalmi aplikovať rovnaký postup.

Problém môže nastať, ak pri zužovaní intervalu dostávame (binárne) $h = 1000zzz$ a $d = 0111www$. V takomto prípade nielen nevieme čo dať na výstup (kam sa nakoniec „preklopí“ najvyšší bit), ale aj strácame presnosť (dĺžku intervalu). Riešenie spočíva v tom, že v týchto situáciach odstránime úvodné nulové bity z h a úvodné jednotkové z d: $h = 1zzz$ a $d = 0www$. Popritom si zapamätáme počet takto odstránených bitov a keď sa najbližšie zhodnú najvyššie bity hraníc, budeme vedieť, koľko a akých bitov dať na výstup.

1.4 Poznámky

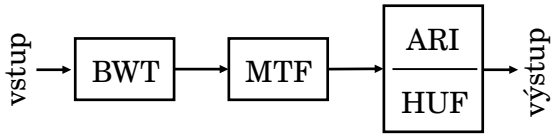
Aritmetické kódovanie, podobne ako Huffmanovo sa zvyčajne nepoužíva samostatne, ale vystupuje ako súčasť zložitejších kompresných algoritmov (najčastejšie ako záverečná fáza). Môžeme ho kombinovať so slovníkovými metódami (napr. LZARI je kombinácia LZSS a aritmetického kódovania), v štatistických metódach vyšších rádov (napr. PPM) aj v iných metódach (pozri napr. časť 2).

Problémom aritmetického kódovania je rýchlosť – ktorá nie je príliš veľká. Kompresný pomer je zvyčajne o čosi lepší ako pri Huffmanovom kódovaní (ale nie o veľa). Jednoduché je modifikovať aritmetické kódovanie na adaptívnu verziu. Jednoducho začneme s rovnomerne distribuovanými pravdepodobnosťami a každý načítaný znak zo vstupu najskôr spracujeme (zúžime interval), a potom príslušne upravíme pravdepodobnosti (teda zväčšíme početnosť tohto znaku). Samozrejme, aj Huffmanovo kódovanie je možné upraviť na adaptívne, avšak nie tak priamočiaro.

Adaptívne verzie aritmetického alebo Huffmanovho kódovania majú výhodu v tom, že nie je potrebné prenášať frekvenčnú tabuľku znakov a pri kódovaní nemusíme čítať vstup dvakrát (najskôr na zistenie frekvenčnej tabuľky a potom na samotné kódovanie).

2 BWT

BWT (Burrows-Wheeler Transformation) v podstate nie je algoritmus na kompresiu dát. Je to invertovateľná transformácia, ktorá reťazec znakov transformuje na iný reťazec znakov. Výstupný reťazec je potom vhodnejší na kompresiu ako pôvodný reťazec. Metóda kompresie dát využíva BWT ako úvodnú transformáciu, nasledovanú napríklad MTF (pozri časť 2.3) a štatistickým kóderom nultého rádu tak, ako je to zobrazené na obrázku. Prirodzene, možné sú aj ďalšie modifikácie.



2.1 Kódovanie

Transformácia pracuje nad blokom dát (reťazcom znakov) dĺžky n. Vytvoríme z reťazca n reťazcov dĺžky n tak, že vstupný reťazec rotujeme. Získané reťazce utriedime. Výstupom transformácie je reťazec pozostávajúci z posledných znakov v reťazcoch (teda ak prejdeme v poradí utriedenia po reťazcoch a vypíšeme ich posledné znaky) a z pozície pôvodného vstupného reťazca medzi utriedenými reťazcami.

Hlavná myšlienka BWT spočíva v tom, že rovnaké kontexty (podreťazce vstupu) sú zvyčajne uvádzané rovnakými znakmi (je to podobná úvaha ako pri znakoch za kontextami). Rovnaké kontexty dáme k sebe triedením. Znaky, ktoré sú pred týmito kontextami sú na konci reťazcov. Teda na konci reťazcov môžeme očakávať častý výskyt rovnakých znakov vedľa seba.

Príklad: Nech vstupným textom je reťazec *aababbcbababcb*. Potom utriedenie jednotlivých rotácií dopadne takto (i označuje pozíciu začínajúceho znaku v pôvodnom reťazci):

i	reťazec
0	aababbcbababcb
1	ababbcbababcba
8	ababcbaababbcb
3	abbcbababcbaab
10	abcbaababbcbab
13	baababbcbababc
7	bababcbaababbc
2	babbcbababcbaa
9	babcbaababbcba
4	bbcbababcbaaba
11	bcbaababbcbaba
5	bcbababcbaabab
12	cbaababbcbabab
6	cbababcbaababb

Výstupom z BWT je v tomto prípade reťazec *babbbccaaaabbb* a pozícia, na ktorej sa nechádza pôvodný reťazec, teda 0.

2.2 Dekódovanie

Pri dekódovaní máme k dispozícii reťazec zložený z posledných znakov utriedených reťazcov a index, kde treba hľadať pôvodný reťazec. Modelujme dekódovanie na tabuľke, akú sme použili v príklade kódovania. Teda poznáme posledný stĺpec znakov. Poznáme aj prvý stĺpec, stačí znaky len utriediť.

Pozrime sa na posledný znak v prvom riadku (inými slovami prvý znak v poslednom stĺpci). Nech je to x. Vieme, že toto x je ten istý znak, ktorý sa ako prvé x vyskytne v prvom stĺpci. Dôvod je ten, že za ním v reťazci nasleduje lexikograficky najmenší reťazec (inak by nebol v prvom riadku) a najmenší reťazec spomedzi ostatných začínajúcich x musí nasledovať aj za znakom, ktorý je prvým výskytom x v prvom stĺpci (inak by to nebol prvý výskyt). Túto pozíciu x v prvom stĺpci si označme ako obsadenú.

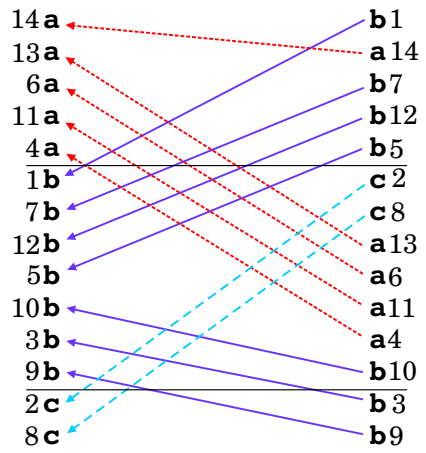
Zoberieme druhý znak v poslednom stĺpci, nech je to y. Opäť hľadáme v prvom stĺpci prvý neobsadený výskyt znaku y. Postupujeme takto ďalej, až kým neurčíme pre všetky znaky z posledného stĺpca, ktorým znakom z prvého stĺpca zodpovedajú.

Teraz rekonštruujeme reťazec v prvom riadku. Keďže vieme, že posledný znak v riadku je v reťazci pred prvým znakom v riadku, dokážeme spätne prejsť a odzadu rekonštruovať požadovaný reťazec. Potom ho stačí len zrotovať, utriediť a vybrať výstupný reťazec zo správnej pozície.

Drobný problém v prezentovanej rekonštrukcii by mohol nastať, ak sú prvý znak a posledný v prvom riadku zhodné. Potom ale celý reťazec obsahuje len tento znak a môžeme sa podľa toho zariadiť.

Poznamenajme, že v praktickej implementácii BWT sa dekódovanie dá robiť na jeden prechod v lineárnom čase $O(n)$.

Príklad: Ilustrujme dekódovanie na výstupe príkladu kódovania, teda máme na vstupe reťazec *babbbccaaaabbb* a pozíciu 0. Rekonštruujeme prvý stĺpec a potom dekódovanie prebieha naznačeným spôsobom podľa šípiek (šípky označujú zodpovedajúcim si znakom). Čísla pri znakoch hovoria o poradí znakov pri spätnej rekonštrukcii.



2.3 MTF

MTF (Move to front) je heuristika, ktorou sa snažíme poziční blízkosť rovnakých znakov pretransformovať do štatistickej významnosti znakov. MTF nekomprimuje text, ale vytvára predpoklady na úspešnú aplikáciu štatistických kóderov nultého rádu tým, že sa snaží znižovať entropiu.

MTF má pole, v ktorom sú usporiadané znaky. Po prečítaní znaku dá na výstup index (pozíciu) tohto znaku v poli a zároveň znak presunie v poli na začiatok. Takto pokračuje, až kým nevyčerpá celý vstup.

Príklad: Demonštrujme si MTF na príklade výstupu z BWT, teda na reťazci *aababbcbababcb*. Stĺpec „pole“ ukazuje poradie prvkov v poli *po* spracovaní príslušného znaku.

	pole	výstup
	abc	
b	*bac*	1
a	*abc*	1
b	*bac*	1
b	*bac*	0
b	*bac*	0
c	*cba*	2
c	*cba*	0

pokr.

	pole	výstup
a	*acb*	2
a	*acb*	0
a	*acb*	0
a	*acb*	0
b	*bac*	2
b	*bac*	0
b	*bac*	0

Hoci náš príklad nie je ideálnym na demonštráciu výhod MTF, porovnajme entropie pôvodného a no-

vého reťazca:

$$H\left(\frac{5}{14},\frac{7}{14},\frac{2}{14}\right)\approx 0,4371$$
$$H\left(\frac{8}{14},\frac{3}{14},\frac{3}{14}\right)\approx 0,4318$$

Teda MTF sa snaží posúvať aktuálne spracúvané znaky na začiatok poľa a zabezpečiť častý výskyt nízkych indexov vo výstupe. Keď vo vstupe prejdeme na iný blok rovnakých znakov, až na prvý index opäť dostávame na výstup nízke hodnoty. Výsledkom je zníženie entropie a môže nasledovať úspešná aplikácia štatistického kódovania.

Dekódovanie MTF prebieha podobne ako kódovanie. Začneme s utriedeným poľom znakov. Prečítame index zo vstupu. Príslušný znak z poľa dáme na výstup. Zároveň presunieme znak na začiatok poľa a načítame ďalší index zo vstupu. Toto opakujeme, až kým neprečítame celý vstup.

2.4 Poznámky

Iná možnosť pri spracovaní výstupu BWT (namiesto MTF, prípadne navyše k MTF) je použiť RLE. RLE (Run Length Encoding) je jednoduchý spôsob kódovania, keď namiesto reťazca rovnakých znakov dávame na výstup len jeden znak a dĺžku reťazca.

Časovo najnáročnejšou operáciou v BWT je triedenie pri kódovaní. Dá sa napríklad použiť kombinácia radix-sortu (napr. na prvé dva znaky) s následným quicksortom (na utriedenie vnútri skupín). Prirodzene, pri kódovaní nie je ani potrebné vytvárať ďalšie reťazce – stačí správne indexovať do pôvodného reťazca.

Príkladom praktického použitia BWT je program Bzip2, ktorý je kombináciou BWT, MTF a Huffmanovho kódovania.