Kompresia dát

Zdroje redundancie:

- priveľa znakov
- frekvencie výskytu znakov sú rôzne
- podmienené pravdepodobnosti rôznych postupností znakov sú rôzne
- dá sa vyjadriť vzorcom alebo algoritmom (malá kolmogorovská zložitosť)
- možnosti interpolácie a aproximácie

Kompresný pomer $k = \dfrac{\textit{dĺžka kompresovaného súboru}}{\textit{dĺžka pôvodného súboru}}$

Niekedy sa ako kompresný pomer uvádza hodnota $1 - k$.

Vlastnosti kódov

- Jednoznačná dekódovateľnosť (Každú postupnosť vieme jednoznačne rozdeliť na kódové slová.)
- Prefixová vlastnosť (Žiadne kódové slovo nie je prefixom iného kódového slova.)
- Vlastnosť okamžitej rozhodnuteľnosti (Pri prečítaní posledného znaku poznáme koniec kódového slova.)

Príklad:

	kód 1	kód 2
A	0	0
B	10	01
C	110	011
D	1110	0111

Kód 1 má prefixovú vlastnosť. Kód 2 je jednoznačne dekódovateľný, ale nemá prefixovú vlastnosť, ani vlastnosť okamžitej rozhodnuteľnosti. Prefixová vlastnosť implikuje obe ostatné vlastnosti.

Kompresia rozptýlených záznamov

- Opakovací znak a počet opakovaní
- Bitová mapa
- Separácia konštánt

Príklad: $d_1d_2000000d_3d_4d_5d_6000d_7\boxed{d_8}d_9000$

Opakovací znak: $d_1d_2\Re6d_3d_4d_5d_6\Re3d_7d_8d_9\Re3$

Bitová mapa: 110000001111000111000: $d_1d_2d_3d_4d_5d_6d_7d_8d_9$

Separácia: $\underbrace{26699(12)}$: $d_1d_2d_3d_4d_5d_6d_7d_8d_9$

Distribučný vektor

Na nepárnych miestach počet dátových položiek. Na párnych miestach počet konštánt. Súčet znamená celkový počet predchádzajúcich.

Nájdenie L-tej položky z pôvodných dát v kompresovanom súbore: Nájdeme najmenšie i také, že $L \leq v_{i-1}+v_i$. Ak i je párne konštanta. Ak i je nepárne $p = L - v_{i-1}$.

Diferenčné metódy kompresie

Princíp pamätá sa len rozdiel od predchádzajúceho záznamu. Metóda je vhodná na menné zoznamy, bibliografické údaje, slovníky, tabuľky hodnôt funkcií. Možno ju kombinovať so slovníkovou metódou.

Nevýhodou je citlivosť na chyby.

Pr.:					
	Michalčík	0	Michalčík	(0,3)	Michalč
	Michna	4	na	(4,0)	na
	Mikelka	2	kelka	(2,4)	kel
	Mikletič	3	letič	(3,2)	let
	Mikulcová	3	ulcová	(3,7)	ul
	Mikulič	4	lič	(4,2)	l
	Mikuličová	7	ová	(7,7)	
	Mikulová	5	ová	(5,7)	

Pridáním slovníka 15 najbežnejších koncoviek mien: er, ič, ík, ka, ná, ný, ová, ...
zlepší sa kompresný pomer.

Frekvenčná analýza – Huffmanové kódy

Princíp minimalizácia očakávanej dĺžky kódu. Nech p_i je pravdepodobnosť výskytu i - tého kódového znaku a l_i je dĺžka. Potom očakávaná dĺžka kódu H je daná vzorcom:

$$H = \sum_{i=1}^{n} p_i l_i ,$$ kde n je počet znakov.

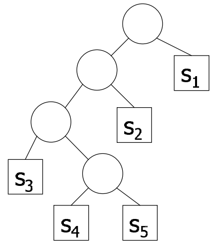
Algoritmus konštrukcie Huffmanovho kódu (Knuth):

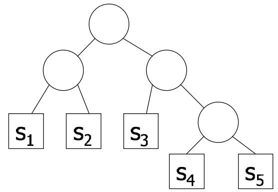
Utriedíme znaky podľa p_i. Zlúčime znaky dva znaky s najmenšími hodnotami p_i. Riešime problém pre $n-1$ hodnôt.

Algoritmus nie je jednoznačný. V niektorých prípadoch sa môžeme rozhodnúť, ktoré uzly budeme spájať. Možno je výhodne v takomto prípade minmalizovať súčet vnútorných ciest. Dostaneme tak kódy, ktoré sa dĺžkou najmenej odlišujú.

Konštrukcia Huffmanovho kódu

znak, p_i;	znak, p_i;	znak, p_i;	znak, p_i;	znak, p_i;	Kód
s_1 0.4	0.4	0.4	s_{3452} 0.6	s_{34521} 1	1
s_2 0.2	0.2	s_{345} 0.4	s_1 0.4		01
s_3 0.2	0.2	s_2 0.2			000
s_4 0.1	s_{45} 0.2				0010
s_5 0.1					0011





00
01
10
110
111

Iný strom a kód

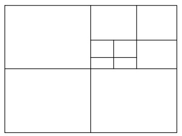
Run length coding

Niekedy sa stáva, že pravdepodobnosti výskytu znakov sú približne rovnaké, ale je veľká pravdepodobnosť, že nasledujúci údaj bude ako predošlý. Vznikajú dlhé sekvencie (runs) rovnakých znakov. (Je to typické pre grafické dáta.) Kódovanie pomocou znaku opakovania a počtu opakovaní.

Príklad: aaaabbbbccabcccccbbbaaaa *l=25*
aℜ4bℜ4cℜ3abcℜ5bℜ3aℜ4 *l=20*

Účinnosť tejto metódy je tým väčšia, čím sú sekvencie dlhšie.

Quad trees a oct trees v počítačovej grafike sú založené na podobnom princípe.



Kompresia s využitím kódového slovníka

Daný text t máme kompresovať pomocou slovníka n fráz $d = \{ p_i : 1 \le i \le n \}$. Predpokladáme, že slovník obsahuje všetky elementárne (jednoznakové) frázy. Nech $|t| = N$ označuje dĺžku vstupného textu a t_j j-tý symbol vstupného textu. Označme $B(j) = \{p_i : \forall(0 \le k < | p_i |) \ t_{j+k} = p_{ik}\}$, kde p_{ik} je k-tý symbol frázy p_i. Nech ω_i je cena frázy p_i. (Obvykle $\omega_i = 1$ pre všetky i.)

Cenu $F(1)$ minimálneho pokrytia textu t dostaneme, ako riešenie systému rekuretných rovníc:

$$F(N+1) = 0$$

$$F(j) = \min_{p_i \in B(j)} \{F(j + |p_i|) + \omega_i\}$$

$F(j)$ je cena pokrytia textu od pozície j (včítane) do konca. Na pokrytie použijeme tie frázy, pomocou ktorých sa F(1) vypočítalo.

Príklad - kompresia pomocou slovníka

Slovník: { A, AR, ARG, E, EN, G, GU, M, ME, MENT, N, R, U, UM, T }
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Text: ARGUMENT

B(1)={1,2,3} B(4)={13,14} B(7)={11}
B(2)={12} B(5)={8,9,10} B(8)={15}
B(3)={6,7} B(6)={4,5}

F(9)=0, F(8)=1, F(7)=2, F(6)=2, F(5)=1, F(4)=2,
15 15,8 15,14 10 10, 13

F(3)=2, F(2)=3, F(1)=3
10,7 12,10,7 10,7,2 (10,13,3)

AR-GU-MENT 2,7,10; ARG-U-MENT 3,13,10.

Napriek zdanlivej zložitosti, kompresovať podľa statického slovníka sa dá v „lineárnom" čase. Konštrukcia optimálneho slovníka, k danému textu je NP ťažký problém.

Adaptívne metódykompresie

Hoci optimálne pokrývanie sa dá robiť efektívne, vyžaduje najprv prečítanie celého vstupu. Konštrukcia slovníka je tak, či tak ťažký problém. Nápad vzdať sa optimálnosti, použiť nejaký „pažravý" (greedy) algoritmus a robiť všetko za letu (on the fly).

Algoritmus: **while** not koniec **do**
begin zakóduj čo najdlhší úsek vstupu;
modifikuj slovník
end;

Výhodou týchto metód je, že pri dekódovaní sa použije ten istý algoritmus modifikácie slovníka a slovník nie je potrebné prenášať a uchovávať s dátami.

Ziv – Lempel – Welch

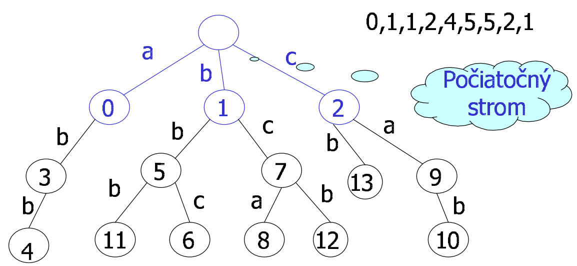
Slovník je strom zostavený z podreťazcov prečítaného úseku textu. Maximálna výška tohoto stromu *h* a najväčší možný počet vrcholov *n* sú parametre metódy.

Vrcholy sú označené číslami kódmi a hrany symbolmi vstupnej abecedy. Na počiatku sú všetky symboly listy v hĺbke 1. Tieto postupne rozširujeme o suffixy prečítaného textu (dĺžky menšej alebo rovnej *h*) pokiaľ strom nemá *n* vrcholov. Uzol stromu je kód pre cestu od vrcholu k nemu.
Ak sa strom naplní nemusíme ho už ďalej modifikovať, ale môžeme ho reorganizovať a prečíslovať vrcholy.

Kódovanie sa robí tak, že nájdeme najdlhšiu cestu od koreňa k uzlu stromu, ktorej hrany sa zhodujú so vstupom.

Príklad konštrukcie adaptívneho slovníku

Σ={a,b,c}, h=4, n=16, t=abbcabbbbbbcb



Aritmetické kódovanie

Huffmanovo kódovanie nie je optimálne pretože dĺžka kódu musí byť celé číslo. Idea kódovať pomocou intervalov podintervalov <0, 1). Celý text jeden interval.

Symbol	pravdepodobnosť	interval
a	0.2	<0, 0.2)
e	0.3	<0.2, 0.5)
i	0.1	<0.5, 0.6)
o	0.2	<0.6, 0.8)
u	0.1	<0.8, 0.9)
!	0.1	<0.9, 1)

Text je eaii!.

Pri kompresii i dekompresii začíname s intervalom <0, 1). Vždy po načítaní symbolu zúžime interval.

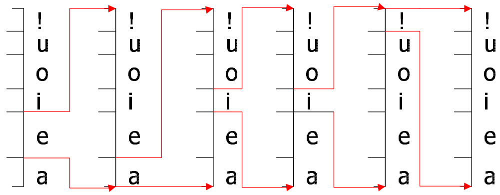
Aritmetické kódovanie - pokračovanie

Proces kódovania (kompresie)

	<0,	1)
e	<0.2,	0.5)
a	<0.2,	0.26)
i	<0.23,	0.236)
i	<0.233,	0.2336)
!	<0.23354,	0.2336)

Ak posledný znak je znak konca textu, môžeme kompresovaný text zakódovať ľubovolným číslom z výsledného intervalu.

Proces dekódovania (dekompresie)



Technické detaily aritmetického kódovania

- Presná aritmetika (integer, fixed)
- kumulatívne frekvencie namiesto pravdepodobností
- underflows - scaling

Metóda bude adaptívna, keď po každom prečítaní symbolu, aktualizujeme kumulatívne frekvencie, či pravdepodobnosti.

Metóda je vhodná na:
- adaptívnu kompresiu textu
- kompresiu postupností celých čísiel
- kompresiu čierno-bielých obrázkov