V prednáške si ukážeme príklady slovníkových metód kompresie dát. Tieto metódy sa snažia využiť na kompresiu skutočnosť, že v dátach sa častokrát opakujú rovnaké postupnosti znakov.

1 LZ77

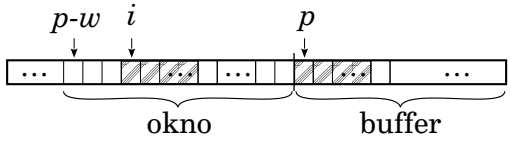
Autormi tohto algoritmu sú Lempel a Ziv (v roku 1977). Mnohé ďalšie slovníkové algoritmy boli inšpirované práve LZ77. Pri popise algoritmu budeme používať nasledujúce pojmy:

- Pozícia – aktuálna pozícia, na ktorej sa pri kódovaní/dekódovaní nachádzame. Začíname na prvom znaku a postupne pokračujeme až k poslednému znaku vstupného textu.
- Okno – posledných w spracovaných znakov (pri kompresii). Znak na pozícii sa do okna už nepočíta.
- Buffer – postupnosť znakov vo vstupnom texte začínajúca pozíciou.

1.1 Kompresia (kódovanie)

Označme vstupný (komprimovaný) text T a nech jeho dĺžka je n. Nech $p \in \{0, \ldots, n-1\}$ označuje pozíciu. To znamená, že okno je postupnosť (podreťazec) $T[p-w, \ldots, p-1]$ a buffer $T[p, \ldots, n-1]$. Ak $p < w$, tak je okno, prirodzene, kratšie. Hlavná myšlienka algoritmu spočíva v tom, že hľadáme najdlhší reťazec v okne, ktorý je zároveň prefixom buffra. Teda hľadáme maximálne $k \leq n-p$ také, že existuje $i \in \{p-w, \ldots, p-k\}$:

$$T[i, \ldots, i+k-1] = T[p, \ldots, p+k-1]. \qquad (1)$$



Postup pri kompresii je nasledujúci:

1. $p = 1$
2. pokiaľ je $p < n$ opakujeme:
   (a) nájdeme i a k podľa (1)
   (b) výstupom je trojica $\langle i-(p-w)+1, k, T[p+k]\rangle$
   (c) $p \leftarrow p+k+1$

Prvý člen vo výstupnej trojici označuje index v okne, číslovaný pre aktuálne okno od 1, kde začína nájdený najdlhší zhodný reťazec. Tento spôsob zaručuje, že index je vždy prvok z $\{1, \ldots, w\}$. Uvedený postup nerieši niektoré situácie, ktoré pri kompresii môžu nastať. Prípad $k = 0$ nastane, ak sa v okne nenachádza znak $T[p]$. Vtedy dáme na výstup trojicu $\langle 0, 0, T[p]\rangle$. Ak najdlhší podreťazec „vyčerpá“ všetky znaky z buffra, t.j. $k = n-p$, tak zhodu skrátime o 1 a na výstup dáme trojicu $\langle i-(p-w)+1, k-1, T[n-1]\rangle$.

Príklad: Nech $T = aababbcbababcb$ ($n = 14$). Tabuľka ukazuje priebeh činnosti algoritmu, pričom zhoda udáva najdlhší nájdený reťazec v okne. Vzhľadom na malý rozsah príkladu nie je veľkosť okna obmedzená.

p	$T[p]$	zhoda	výstup
0	a	–	$\langle 0, 0, a\rangle$
1	a	a	$\langle 1, 1, b\rangle$
3	a	ab	$\langle 2, 2, b\rangle$
6	c	–	$\langle 0, 0, c\rangle$
7	b	bab	$\langle 3, 3, a\rangle$
11	b	bcb	$\langle 6, 2, b\rangle$

1.2 Dekompresia (dekódovanie)

Dekóvanie je jednoduché. Podobne ako pri kódovaní si budujeme a udržiavame okno, v tomto prípade to bude posledných w znakov daných na výstup. Na začiatku je, prirodzene, prázdne. Postupne čítame trojice zo vstupu a na výstup dáme príslušný reťazec z okna (určený indexom a dĺžkou) doplnený znakom z trojice. Ak má vstupná trojica tvar $\langle 0, 0, x\rangle$, tak je výstupom samotný znak x.

1.3 Poznámky

LZ77 je relatívne rýchly algoritmus, pričom dekódovanie je oveľa rýchlejšie ako kódovanie, keďže nie je potrebné vyhľadávať najdlhší zhodný podreťazec a len sa „vypisuje“. Podstatnou z hľadiska rýchlosti kódovania a dosiahnutého kompresného pomeru je voľba veľkosti okna. Dlhé okno umožňuje nachádzať lepšie zhody, ale zároveň podstatne zvyšuje časovú zložitosť kódovania. Navyše, dlhšie okno zvyšuje počet bitov potrebných na zápis indexu a dĺžky vo výstupe (prvý a druhý prvok výstupnej trojice). Krátke okno naopak „zahadzuje“ potenciálne cenné informácie o prechádzajúcej podobe textu skoro, čím obmedzuje možnosť nájsť dlhšie reťazce zhody a predlžuje výstupný text. Na druhej strane krátke okno znižuje časovú zložitosť a dovoľuje použiť na zápis indexu a dĺžky menší počet bitov.

Použitie okna v LZ77 zabezpečuje, že algoritmus je orientovaný na využitie posledne videných znakov. Teda charakteristiky zo začiatku textu neberieme do úvahy. To môže byť pri niektorých typoch dát podstatnou výhodou.

LZ77 je možné rôzne modifikovať – použiť cyklické okno, variabilne meniť veľkosť okna a podobne. Medzi významnejšie úpravy patrí spracovanie výstupu ďalšími algoritmami. Dá sa očakávať,

že žiadna z troch súradníc výstupnej trojice nie je rovnomerne distribuovanou náhodnou premennou. Preto môžeme použiť štatistické kódovanie (napr. Huffmanovo, aritmetické) na následnú transformáciu jednotlivých „stôp“ výstupu.

1.4 LZSS

Dôležitým algoritmom odvodeným z LZ77 je LZSS. LZSS rieši problém znakov, ktoré sa nevyskytujú v okne inak ako LZ77. LZSS si stanoví, akú minimálnu dĺžku musí mať reťazec zhody. Ak je nájdený reťazec kratší, tak na výstup ide samotný znak ($T[p]$) a posunieme pozíciu ďalej. Ak má reťazec dostatočnú dĺžku, dáva na výstup dvojicu $\langle i-(p-w), k\rangle$ (bez ďalšieho znaku, preto aj posun p je iný: $p \leftarrow p+k$). Teda LZSS dáva na výstup dva typy informácií. Aby ich bolo možné rozlíšiť pri dekódovaní, pridáme pred oba typy identifikačný bit.

Uveďme príklad činnosti LZSS pre vstupný text $T = aababbcbababcb$ (rovnaký ako v príklade pre LZ77). Minimálna požadovaná veľkosť zhody je 2.

p	$T[p]$	zhoda	výstup
0	a	–	$0, a$
1	a	a	$0, a$
2	b	–	$0, b$
3	a	ab	$1, \langle 1, 2\rangle$
5	b	b	$0, b$
6	c	–	$0, c$
7	b	bab	$1, \langle 2, 3\rangle$
10	a	ab	$1, \langle 1, 2\rangle$
12	c	cb	$1, \langle 6, 2\rangle$

LZSS vo všeobecnosti dosahuje lepšie kompresné pomery ako LZ77, s porovnateľnými pamäťovými a časovými nárokmi. Dekódovanie je veľmi jednoduché a rýchle. Preto slúži ako základ pre ďalšie známe algoritmy – ARJ (kombinácia LZSS s Huffmanovým kódovaním), PKZip a pod. Odlišnosti iných algoritmov môžu spočívať vo veľkosti okna, v spracovaní výstupov (pozícií) a znakov štatistickým kódovaním (napr. Huffmanovým), v spôsobe posunu okna a jeho premenlivej veľkosti, v spôsobe určenia minimálnej dĺžky reťazca zhody a pod.

2 LZW

Autorom algoritmu je Welch (1984). LZW je v podstate vylepšením algoritmu LZ78. Špeciálna implementácia LZW sa používa na kompresiu v grafickom formáte GIF. Iný variant používa utilita `compress` v UNIXových systémoch. Algorimus si počas spracovávania vstupu buduje slovník, ktorý využíva na kódovanie. Pri popise algoritmu budeme používať nasledujúce pojmy:

- Slovník – postupnosť reťazcov
- Kódové slovo – index (pozícia) konkrétneho reťazca v slovníku

2.1 Kompresia (kódovanie)

Symbolom $+$ označíme zreťazenie dvoch reťazcov, teda $x+y$ označuje zreťazenie reťazcov x a y. Nech D je slovník kódových slov a nech $D(x)$ označuje kódové slovo reťazca x v slovníku D. Postup pri kódovaní je nasledujúci:

1. zaradíme všetky znaky abecedy do D
2. $p \leftarrow$ "" (inicializujeme p ako prázdny reťazec)
3. pokiaľ nie sme na konci vstupu:
   (a) $c \leftarrow$ ďalší znak zo vstupu
   (b) ak je $p+c$ v D:
   $p \leftarrow p+c$
   (c) inak:
   dáme na výstup $D(p)$
   pridáme $p+c$ do D
   $p \leftarrow c$
4. dáme na výstup $D(p)$

Poznamenajme, že uvedený postup nepočíta s prázdnym vstupom. Demonštrujme si postup kódovania LZW na príklade.

Príklad: Nech $T = aababbcbababcb$ je vstupný text. Tabuľka ukazuje priebeh činnosti algoritmu, pričom stĺpec „slovník“ uvádza kódové slovo a prislúchajúci reťazec pridaný do slovníka v danom kroku výpočtu. Na začiatku obsahuje slovník tri znaky. Stĺpec pre hodnotu p udáva túto hodnotu *na konci* spracovania vstupného znaku v premennej c.

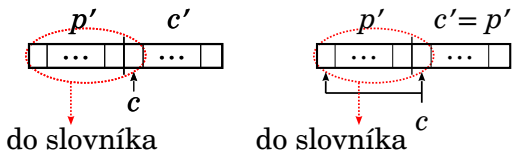
c	slovník	p	výstup
	$0, a$		
	$1, b$		
	$2, c$	""	
a	–	a	
a	$3, aa$	a	0
b	$4, ab$	b	0
a	$5, ba$	a	1
b	–	ab	
b	$6, abb$	b	4
c	$7, bc$	c	1
b	$8, cb$	b	2
a	–	ba	
b	$9, bab$	b	5
a	–	ba	
b	–	bab	
c	$10, babc$	c	9
b	–	cb	
	–		8

2.2 Dekompresia (dekódovanie)

Dekódovanie prebieha analogicky ako kódovanie. Konštruujeme slovník, ktorý následne používame na dekódovanie a reťazce zodpovedajúce kódovým slovám dávame na výstup. Na začiatku je opäť slovník naplnený všetkými znakmi abecedy.

Pri dekódovaní môžu nastať dva prípady: kódové slovo sa v slovníku nachádza alebo nie. Ak je vstupné kódové slovo už v slovníku, vieme dať na výstup príslušný reťazec. Zároveň vieme, že *pred* kódovaním tohto reťazca sme do slovníka zaradili reťazec, ktorý aj teraz potrebujeme do D dostať. Tento reťazec pozostáva z reťazca určeného prechádzajúcim kódovým slovom a prvým znakom reťazca určeného aktuálnym kódovým slovom.

Druhý prípad (vstupné kódové slovo nie je v slovníku) môže nastať, lebo budovanie slovníka je pri dekódovaní oneskorené. Nastane však len v tom prípade, keď pri kódovaní dáme na výstup kódové slovo, ktoré bolo do slovníka pridané ako posledné. To však znamená, že v texte sa vyskytoval za sebou dvakrát rovnaký reťazec. Preto je prvý znak reťazca prislúchajúceho vstupnému kódovému slovu zhodný s prvým znakom reťazca určeného predchádzajúcim kódovým slovom. Teda vieme, aký reťazec potrebujeme pridať do slovníka a následne aj vypísať na výstup. Nasledujúce obrázky ilustrujú oba uvedené prípady (označenia sú prebrané z nižšie uvedeného popisu algoritmu).



Pre zjednodušenie zápisu označme $D(x')$ reťazec prislúchajúci kódovému slovu x', teda opačné zobrazenie ako pri kódovaní. Postup pri dekódovaní je nasledujúci (premenné s čiarkou označujú kódové slová, bez čiarky sú to reťazce):

1. zaradíme všetky znaky abecedy do D
2. $c' \leftarrow$ prvé kódové slovo zo vstupu
3. dáme na výstup $D(c')$
4. pokiaľ nie sme na konci vstupu:
   (a) $p' \leftarrow c'$
   (b) $c' \leftarrow$ ďalšie kódové slovo zo vstupu
   (c) ak je $D(c')$ v D:
   dáme na výstup $D(c')$
   $p \leftarrow D(p')$
   $c \leftarrow$ prvý znak z $D(c')$
   pridáme $p + c$ do D
   (d) inak:
   $p \leftarrow D(p')$
   $c \leftarrow$ prvý znak z $D(p')$
   dáme na výstup $p + c$ (zhodné s $D(c')$)
   pridáme $p + c$ do D

Príklad: Nech abeceda je $\{a, b, c\}$ a nech postupnosť kódových slov $(0, 0, 1, 4, 1, 2, 8, 9, 3)$ je vstupný text. Poznamenajme, že postupnosť je na začiatku zhodná s výstupom predchádzajúceho príkladu (pre kontrolu) a na záver sa odlišuje (pre lepšiu demonštráciu postupu). Tabuľka ukazuje priebeh činnosti algoritmu pri dekódovaní. Stĺpce p, p' a c zobrazujú hodnoty premenných *na konci* spracovania vstupného kódového slova.

vstup	p'	p	c	slovník	výstup
				0, a	
				1, b	
				2, c	
0				–	a
0	0	a	a	3, aa	a
1	0	a	b	4, ab	b
4	1	b	a	5, ba	ab
1	4	ab	b	6, abb	b
2	1	b	c	7, bc	c
8	2	c	c	8, cc	cc
9	8	cc	c	9, ccc	ccc
3	9	ccc	a	10, $ccca$	aa

2.3 Poznámky

Výhoda LZW oproti LZ77 spočíva najmä v rýchlosti kódovania, pretože sa porovnáva menší počet reťazcov. Modifikácie LZW môžu zahŕňať premenlivú dĺžku zápisu kódových slov (v závislosti na aktuálnej veľkosti D), odstraňovanie starých reťazcov zo slovníka a podobne.