LZW algoritem – stisni me krepko

↓↓↓
Martin Duh

→ **Ste si kdaj zaželeli, da bi vaše besedilo zasedlo manj pomnilnega prostora, ne da bi pri tem kak podatek izgubili? Ste želeli kdaj stisniti sliko formata GIF ali TIFF, ne da bi slika izgubila svojo kakovost? Obstaja rešitev, ki vse to in še več naredi ob preprosti implementaciji. Rešitev je Lempel-Ziv-Welchov algoritem.**

Lempel-Ziv-Welchov algoritem

Skorajda vsi algoritmi za stiskanje besedila so bili izpeljani iz dveh zelo znanih algoritmov za kodiranje, LZ77 in LZ78.

Leta 1977 sta avtorja Abraham Lempel in Jakob Ziv ustvarila algoritem LZ77, leto kasneje pa LZ78. Priljubljeno različico algoritma LZ78 je leta 1984 predstavil Terry Welch in ga poimenoval Lempel-Ziv-Welch (LZW).

Algoritem stiska besedila na podlagi vnaprej določenega slovarja, ki ga sproti posodablja. Slovar si lahko predstavljamo kot seznam različnih znakov oz. nizov, s pomočjo katerih lahko zapišemo celotno besedilo. Algoritem za implementacijo ni zahteven in je zaradi te svoje preprostosti eden izmed najbolj priljubljenih algoritmov za stiskanje besedil.

Če za primerjavo vzamemo Huffmanovo kodiranje [6] je najprej potrebno vse znake v besedilu urediti po verjetnosti (t. j. kako pogosto se v besedilu nahajajo) in na podlagi teh verjetnosti zgraditi ustrezno drevo. Šele ko je drevo zgrajeno, lahko besedilo zakodiramo. Postopek stiskanja oz. kodiranja ter postopek razširjanja oz. dekodiranja LZW algoritma bomo predstavili v nadaljevanju.

Kodiranje

Pri kodiranju bomo dano besedilo zakodirali na takšen način, da bo rezultat krajši ali enak začetnemu besedilu, pri tem pa ne bomo izgubili nobenih informacij. Za kodiranje danega besedila je potrebno najprej razložiti pojem slovarja. Slovar predstavlja seznam izrazov, kjer ima vsak izraz svoj enolično določen indeks. Slovar se sproti posodablja, kar pomeni, da se vsak izraz v besedilu, ki ni v seznamu, doda na ustrezno mestu in se mu priredi ustrezen indeks.

Implementacija slovarja med programerji ni enotna oz. enolično določena, saj si lahko vsak programer sam izbere, na kakšen način bo implementiral svoj slovar in koliko vnosov bo na začetku le-ta imel. V večini primerov ima slovar 256 vnosov, ki predstavljajo razširjeno ASCII tabelo.[1] Ni pa nujno, da slovar predstavlja razširejno ASCII tabelo, saj ima lahko tudi manj vnosov (poljubna druga abeceda, ki vsebuje vse znake iz besedila).

LZW algoritem deluje v naslednjih treh korakih:

- **Pridobivanje izraza**
  LZW pretvori dano besedilo v manjša besedila, ki jim pravimo izrazi. Da se poišče naslednji izraz v besedilu, LZW jemlje na vsakem koraku po en znak oz. po eno črko (kar predstavlja osem bitov) ter ga doda k izrazu. Ta proces traja tako dolgo, dokler tako dobljeni izraz v slovarju več ne obstaja. Izraz brez zadnjega dodanega znaka zako-

[1] American Standard Code for Information Interchange (ASCII) tabela je nabor različnih znakov. Uporablja se zato, ker računalniki razumejo le števila, v ASCII tabeli pa imajo vsi znaki zapisano svojo enolično določeno število, kar omogoča računalniku, da jih razume. Vsak znak v razširjeni ASCII tabeli je velikosti osmih bitov.

diramo, saj je v slovarju še definirani, nedefiniran izraz pa se doda v slovar (postane definiran). Iskanje naslednjega izraza poteka od vključno zadnjega dodanega znaka naprej.

- **Posodabljanje slovarja**
  Kakor hitro naletimo na izraz, ki še v našem slovarju ni definiran, je potrebno le-tega zapisati v slovar. Običajno dodajamo izraze na konec slovarja. V praksi je slovar implementiran kot uravnoteženo drevo, saj nam tako v logaritemskem, in ne v linearnem, številu korakov poišče potrebni izraz, če za osnovni korak štejemo eno primerjavo dveh nizov. Pri stiskanju daljšega besedila velikost slovarja zelo hitro narašča in zaradi tega je potrebno velikost slovarja omejiti, saj na tak način preprečimo preveliko zasedenost pomnilnika. Na velikost slovarja vpliva velikost binarnega (dvojiškega) zapisa[2] posameznega izraza. Primer: če se odločimo, da bomo vse izraze zakodirali v 12-bitnem zapisu, se velikost slovarja omeji na 4096, saj je $2^{12} = 4096$.
- **Kodiranje izraza**
  Vsakemu izrazu v slovarju pripada število (indeks) oz. koda tega izraza. To pomeni, da namesto izraza zapišemo le-temu pripadajočo kodo.

Algorithm 1 LZW kodiranje

```
beseda ← ""
while EOF = false do
    x ← preberi_naslednjo_crko()
    if beseda + x je v slovarju then
        beseda ← beseda + x
    else
        izpisi indeks iz slovarja za beseda
        dodaj beseda + x v slovar
        beseda ← x
    end if
end while
izpisi indeks iz slovarja za beseda
```

Algoritem 1 predstavlja psevdokodo za kodiranje.[3]

[2]Binarni (dvojiški) zapis je predstavitev kombinacije stanj v digitalnih računalnikih z vrednostmi 0 in 1. Vsak izraz je zapisan kot zaporedje števk 0 in 1 in vsako tako zaporedje enolično določa posamezni izraz. Količino informacije, ki jo lahko predstavimo z eno binarno števko, imenujemo bit.

Preprost primer uporabe

Naj bo naša naloga zakodirati oz. stisniti naslednje besedilo: *AAABBBABBA*. Za slovar uporabimo razširjeno ASCII tabelo, ne da bi karkoli preindeksirali (začetni indeks je 0). Za zgornje besedilo sta za nas pomembni le prvi dve veliki črki angleške abecede, ki se v razširjeni ASCII tabeli nahajata na 65. in 66. mestu.

V tabeli 1 predstavljamo delovanje algoritma, kjer je prvi stolpec zaporedni korak delovanja algoritma, v drugem stolpcu je preostalo besedilo, ki ga še moramo zakodirati. V tretjem stolpcu je prva črka oz. znak preostalega besedila, v četrtem stolpcu je lepljenje črke s trenutnim izrazom (spremenljivka *beseda*). V petem stolpcu preverjamo, ali je izraz, ki mu dodamo prvo črko preostalega besedila (izraz v tretjem stolpcu), v slovarju. V naslednjem stolpcu je izpis kod. Celotni stolpec bo predstavljal zakodirano besedilo (bran od zgoraj navzdol). V naslednjem stolpcu so izrazi, ki so na novo dodani v slovar, saj so pred tem bili nedefinirani, zadnji stolpec pa predstavlja trenutni izraz.

Tako je končni izpis (zakodirano besedilo) danega besedila: 65 256 66 258 65 259 in novi vnosi v slovar so: *AA*(256) *AAB*(257) *BB*(258) *BBA*(259) *AB*(260).

Za zapis originalnega besedila je potrebno 80 bitov ($10 \cdot 8 = 80$ - število 10 predstavlja velikost besedila, za vsak znak pa je potrebnih osem bitov). Pri zakodiranem besedilu pa je za zapis izrazov potrebnih $3 \cdot 8 + 3 \cdot 9 = 51$ bitov. Osem bitov je uporabljenih za osnovni slovar, vse nove izraze zapišemo z devetimi biti. V primeru, da vse izraze (tudi osnovni slovar) zapišemo z devetimi biti, potem dobimo $6 \cdot 9 = 54$ bitov, kar predstavlja 67,5 % originalnega besedila.

Dekodiranje

Ker iz zakodiranega besedila ne moremo razbrati njegovega pomena, je potrebo za tovrstne namene zakodirano besedilo dekodirati. Zelo pomembno je, da za dekodiranje uporabimo enako implementacijo osnovnega slovarja kot pri kodiranju, kajti le tako bomo dobili pravo originalno besedilo in ustrezni pomen.

[3]Znak za seštevanje + v algoritmu pomeni lepljenje nizov, npr. $beseda + x$ pomeni, da se besedi *beseda* doda na konec še beseda x.

korak	preostalo besedilo	naslednja črka x	$beseda + x$	je v slovarju?	izpis	nov vnos v slovar	$beseda$
1							''
2	*AAABBBABBA*	*A*	*A*	Da			*A*
3	*AABBBABBA*	*A*	*AA*	Ne	65	*AA* (256)	*A*
4	*ABBBABBA*	*A*	*AA*	Da			*AA*
5	*BBBABBA*	*B*	*AAB*	Ne	256	*AAB* (257)	*B*
6	*BBABBA*	*B*	*BB*	Ne	66	*BB* (258)	*B*
7	*BABBA*	*B*	*BB*	Da			*BB*
8	*ABBA*	*A*	*BBA*	Ne	258	*BBA* (259)	*A*
9	*BBA*	*B*	*AB*	Ne	65	*AB* (260)	*B*
10	*BA*	*B*	*BB*	Da			*BB*
11	*A*	*A*	*BBA*	Da			*BBA*
12		EOF			259		

TABELA 1.
Primer kodiranja.

Algoritem za dekodiranje deluje v obratnih korakih kot pa algoritem za kodiranje.

Algorithm 2 LZW dekodiranje

```
preberi kodo x iz zakodiranega besedila
poisci v slovarju element na mestu x
izpisi element
beseda ← element
while EOF = false do
    preberi x
    poisci v slovarju element na mestu x
    if indeks x v slovarju ne obstaja then
        element ← beseda + prvaCrkaOdBeseda
    end if
    izpisi element
    dodaj beseda+prvaCrkaOdElement v slovar
    beseda ← element
end while
```

Algoritem 2 predstavlja psevdokodo za dekodiranje.

Preprost primer uporabe

S pomočjo dekodiranja dekodirajmo prejšnje zakodirano besedilo: 65 256 66 258 65 259. Pričakujemo, da nam bo algoritem zakodirano besedilo dekodiral v besedilo *AAABBBABBA*. Uporabimo enak slovar kot pri kodiranju, kar pomeni, da uporabljamo razširjeno ASCII tabelo, ne da bi karkoli preindeksirali. Slovar šteje 256 znakov, koda 65 predstavlja veliko črko *A*, koda 66 pa veliko črko *B*. Ostale kode so izrazi, ki so v slovar bili dodani v fazi kodiranja.

V tabeli 2 je prikazano delovanje algoritma za dekodiranje, kjer predstavljajo prvi trije stolpci podobno kot v tabeli 1, četrti stolpec pa podobno kot peti stolpec v tabeli 1. Peti stolpec v tabeli 2 predstavlja element v slovarju, ki ima za indeks število iz drugega stolpca. Če je indeks iz drugega stolpca v slovarju definiran, potem je element izraz s tem indeksom. Če pa indeks iz drugega stolpca ni definiran, potem postane element prejšnji izraz, ki mu še prilepimo prvo črko tega izraza. Ta dobljen izraz dodamo v slovar na ustrezno zaporedno mesto (kar predstavlja sedmi stolpec). Na vsakem koraku izpišemo dobljeni izraz, kar predstavlja šesti stolpec. Zadnji stolpec pa predstavlja trenutni izraz.

Dekodirano besedilo je *AAABBBABBA*, kar je enako kot originalno besedilo.

Novi vnosi v slovar so: 256 (*AA*), 257 (*AAB*), 258 (*BB*), 259 (*BBA*) in 260 (*AB*).

Vprašanja in odgovori

- Ali je potrebno za dekodiranje dodati celotni slovar v kodirano besedilo? Ne, ker se tudi pri dekodiranju slovar gradi sproti. Potrebno je le, da sta osnovna slovarja pri kodiranju in dekodiranju enaka.

korak	preostalo besedilo	naslednja koda	je že v slovarju?	element	izpiši	nov vnos v slovar	beseda
1	65 256 66 258 65 259	65	Da	A	A	/	A
2	256 66 258 65 259	256	Ne	$beseda + x = AA$	AA	256 (AA)	AA
3	66 258 65 259	66	Da	B	B	257 (AAB)	B
4	258 65 259	258	Ne	$beseda + x = BB$	BB	258 (BB)	BB
5	65 259	65	Da	A	A	259 (BBA)	A
6	259	259	Da	BBA	BBA	260 (AB)	BBA
7	/	EOF	/	/	/	/	/

TABELA 2.
Primer dekodiranja. Oznaka x v petem stolpcu je krajši zapis spremenljivke $prvaCrkaOdBeseda$.

- Ali lahko velikost slovarja omejimo? Omejitev velikosti slovarja je priporočljiva, saj tako omejimo število bitov, ki so potrebni za zapis izrazov. Recimo, če želimo porabiti za zapis izrazov 12 bitov, potem omejimo velikost slovarja na 4096 izrazov, saj je $2^{12} = 4096$.
- Kaj se zgodi, ko omejeni slovar zapolnimo? Kakor hitro slovar zapolnimo, bodisi pri kodiranju bodisi pri dekodiranju, izbrišemo vse nove vnose, tako da ostane le osnovni slovar. Nato ga začnemo ponovno graditi. Tega nam ni potrebno nikjer označiti, saj se bo slovar pri kodiranju in dekodiranju izbrisal na istem mestu oz. koraku. Če v slovarju izbrišemo vse nove vnose, se nam zgodi, da bodo novi vnosi v slovar imeli enake kode kot prejšnji (zdaj že izbrisani) vnosi. To ne vpliva na končni rezultat, saj noben prejšnji vnos več ne bo obstajal. To pomeni, da bo še vedno vsak izraz imel svojo enolično določeno kodo.
- Kako shraniti zakodirano besedilo? Datoteko, kamor shranjujemo zakodirano besedilo, je potrebno odpreti kot binarno datoteko, kajti le tako bo zakodirano besedilo zasedlo manj prostora.

Literatura

[1] Ida M. Pu, *Fundamental Data Compression*, Butterworth-Heinemann publications, Burlington, 2006.

[2] G. Lakhani, *Introduction to LZW, Reducing coding redundancy in LZW*, 2005, 1418–1420.

[3] T. A. Welch, *LZW compression algorithm, A Technique for High-Performance Data Compression*, 1984, 8–12.

[4] `http://en.wikipedia.org/wiki/Lempel-Ziv-Welch`, (citirano 20. 08. 2013).

[5] J. Nieminen, *An efficient LZW implementation*, `http://warp.povusers.org/EfficientLZW`, (citirano 10. 08. 2013).

[6] T. Kos, *Huffmanovo kodiranje*, Presek **39** (2011/12), 3, 26–29.