



STISKANJE PODATKOV

Stiskanje (kompresija) podatkov

- Stiskanje podatkov je **ključnega pomena** za multimedijske aplikacije
- “Surovi” podatki so **preveliki** za obdelavo/prenos:
 - minuta nestisnjenega zvoka zavzame:

	44.1 kHz	22.05 kHz	11.025 kHz
16 bit stereo	10.1 MB	5.05 MB	2.52 MB
16 bit mono	5.05 MB	2.52 MB	1.26 MB
8 bit stereo	2.52 MB	1.26 MB	630 kB

- nestisnjene slike:

tip	velikost
512x512 8 bit	0.25 MB
3872x2592 24 bit	30.1 MB

Stiskanje podatkov

- Video = zvok + slike
 - sekunda videa, 25 slik na sekundo, 640x480, bitna globina 24 bitov
 $25 * 640 * 480 * 3 = \mathbf{23.04\ MB}$
ena ura je torej ~ **83 GB**
 - HD DV kodiranja, skupaj s kompresijo so nekje med 2-12 MB na sekundo!
 - brez kompresije = 155 MB
- Stiskanje **mora biti** sestavni del predstavitve zvoka, slike in videa



www.dilbert.com



- S stiskanjem želimo vhodno zaporedje **simbolov zakodirati** tako, da bo v stisnjeni obliki predstavljeno s čim manj biti
 - simbol je lahko npr. en črka, 8x8 velik kos slike ...
- Stiskanje **simbol preslika** v ustrezno kodno besedo, dobimo **kodo**
- Raztezanje (dekompresija) iz kode sestavi zaporedje simbolov
- **Kompresijsko razmerje**
 - $(\text{število bitov kode}) / (\text{število bitov vhodnih podatkov})$

Stiskanje podatkov





Stiskanje: primer

- Imamo ASCII besedilo *EIEIO*
- Nestisnjena ASCII predstavitev

E I E I E O
01000101 01001001 01000101 01001001 01000101 01001111
*6*8 = 48 bitov*

- Stisnjena predstavitev s kodiranjem simbolov:

E = 0, I=10, O=11:

E I E I E O
0 10 0 10 0 11
9 bitov

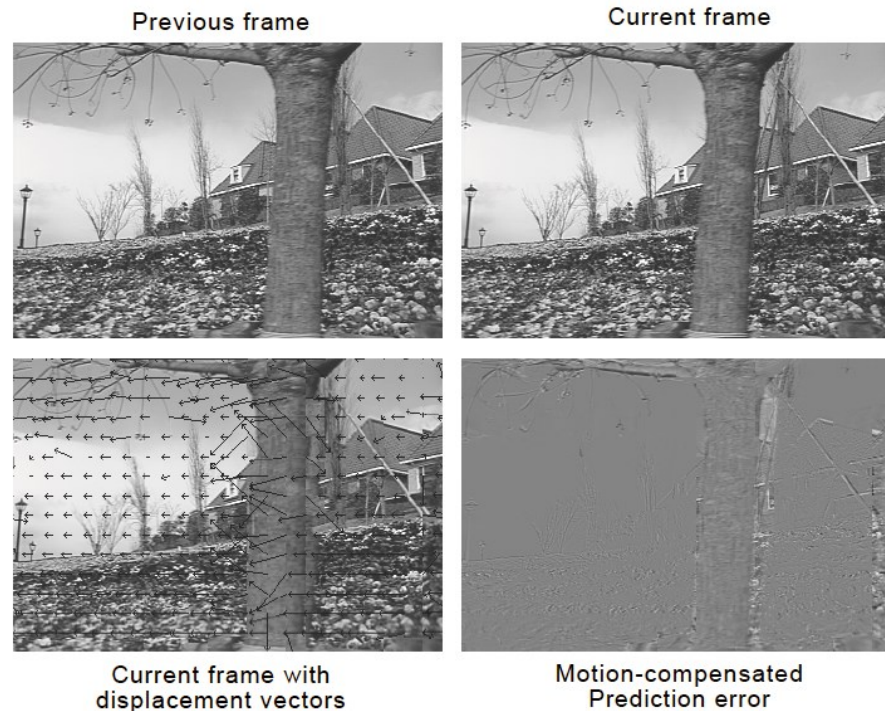
- Kompresijsko razmerje je torej $9:48 = 1:5.33$



Stiskanje multimedijskih podatkov

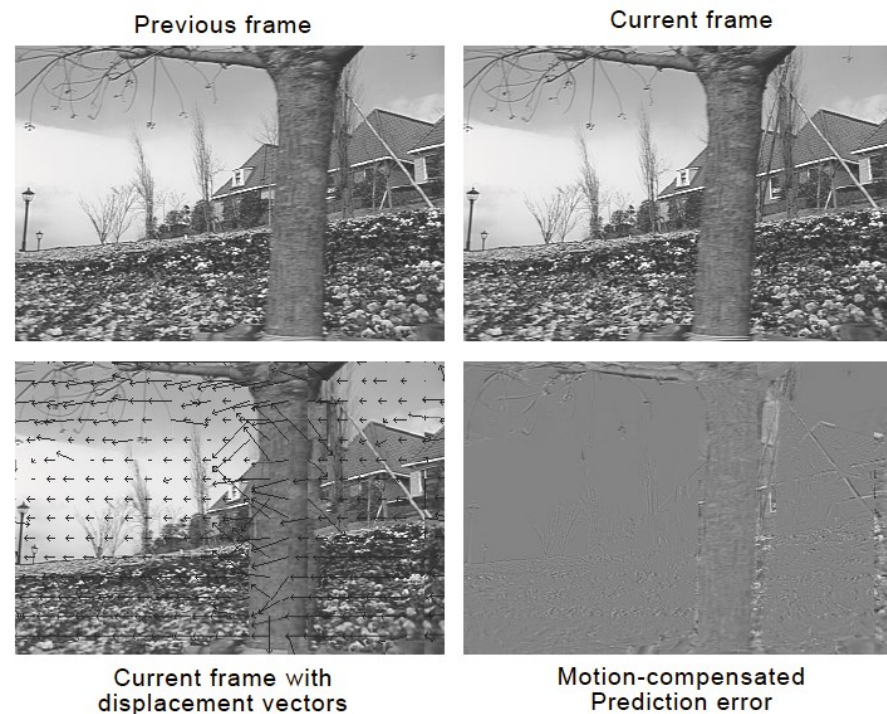
- Izkorišča
redundantnost
podatkov

- ▣ časovna
- ▣ prostorska
- ▣ spektralna
- ▣ zaznavna



Časovna redundantnost

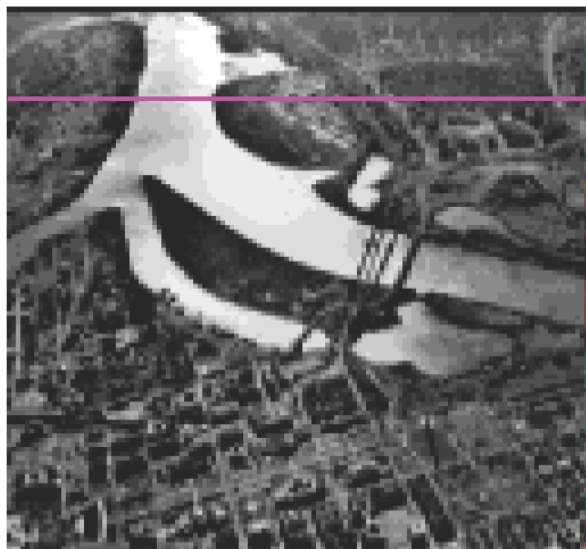
- Izkorišča “zvezno” spreminjanje zvoka, videa **v času**
- Lahko “napovemo” prihodnost na podlagi trenutnih podatkov in kodiramo le razliko do napovedanega (flac, mpeg ...)
 - navadno porabimo manj bitov za kodiranje razlik



*časovna redundantnost:
zaporedne slike so si podobne*

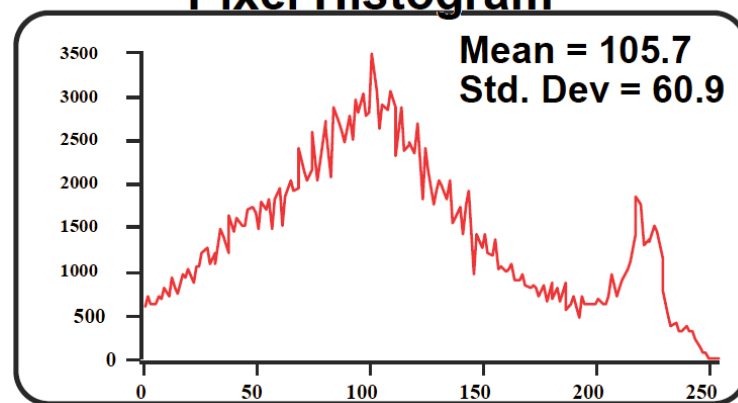
Prostorska redundantnost

- Pri slikah izkorišča lastnost, da so sosednji piksli korelirani
 - lahko npr. kodiramo le razlike v sosednjih pikslih



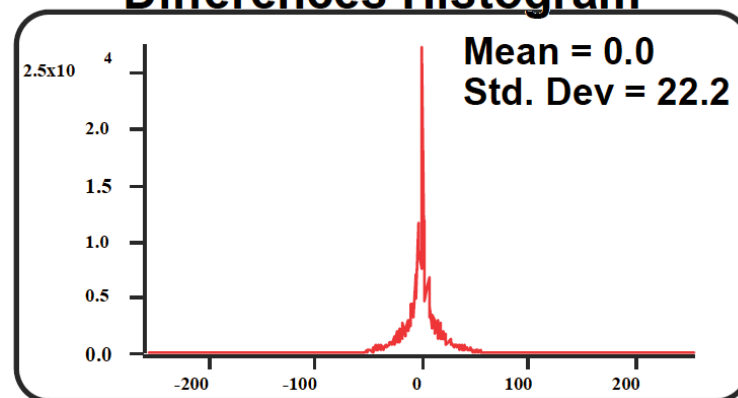
sosednji piksli v vrstici so si večinoma podobni

Pixel Histogram



histogram pikslov slike

Differences Histogram

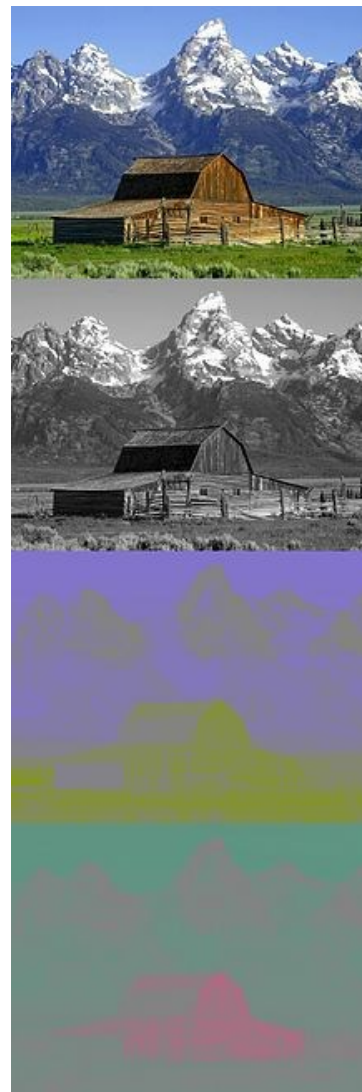


histogram razlik med piksli



Spektralna redundantnost

- Vidni sistem je bolj občutljiv na spremembe v svetlosti (*luminance*) kot v barvi (*chrominance*)
 - RGB pretvorimo v ustrezno alternativno predstavitev, npr. YC_bC_r
 - barvne kanale predstavimo z manj biti kot svetlost (NTSC, JPEG, MPEG)
 - nizko-nivojske lastnosti percepcije



slika RGB

Y

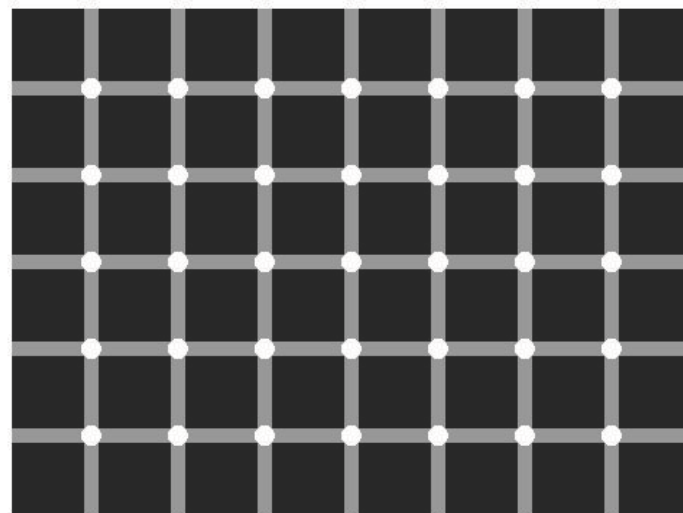
C_b

C_r



Zaznavna redundantnost

- Slušni in vidni sistem sta **kompleksna**
- Ne zaznata vseh podrobnosti, ki jih lahko nek vzorčen signal vsebuje
 - ne vidimo prehitrih sprememb kontrastov v slikah, podrobnosti hitro premikajočih se predmetov itn.
- Izkoristimo pri stiskanju (mp3, jpeg ...)



Vizualna iluzija



Hibridne slike



Stiskanje z oz. brez izgub

- Stiskanje **brez izgub**: raztegnjeni podatki so enaki originalu
 - zip, flac ...
- Stiskanje **z izgubo**: raztegnjeni podatki so dober približek originala (predvsem s stališča percepcije)
 - jpeg, mp3 ...
 - navadno lahko določimo **razmerje med kvaliteto/velikostjo**
- Glavni razlog uporabe stiskanja z izgubo je, da stiskanje brez izgub **nima dovolj velikega** kompresijskega razmerja
 - nekje do 1:2 na multimedijskih podatkih (zvok, slika)



Lena Söderberg



Izrez slike Lene:
242x212 pikslov

original
164 kB

LZW stiskanje
brez izgube
148 kB

JPG stiskanje
25 kB
vidno popačenje

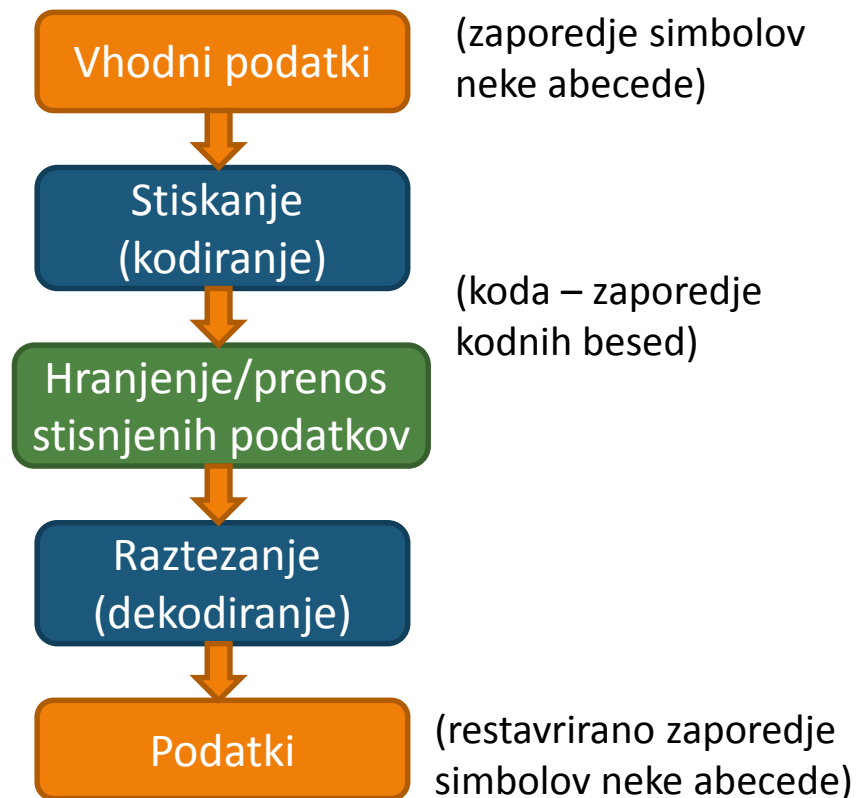


BREZIZGUBNO STISKANJE



- Raztegnjeni podatki so **enaki** vhodnim podatkom
- Pregledali bomo nekaj standardnih brezizgubnih algoritmov
 - jih uporabljamo neposredno (zip)
 - so sestavni del izgubnih algoritmov (npr. JPEG uporablja RLE)

Brezizgubno stiskanje





- Enostavna metoda:
ponovitve nekega simbola zamenjamo s **simbolom** in **številom pojavitev**
 - navadno potrebujemo še posebno oznako, ki določa kdaj se simbol pojavi
- Kompresijsko razmerje je odvisno od tipa podatkov
- Uporaba npr. za **stiskanje ničel** (*zero length supression*)
 - stiskanje tišine v zvočnih posnetkih
 - čb slike

Stiskanje ponavljanj

Primer:

89400000000000000000
0000000000000000

stisnemo kot:

894f32

kjer je *f* oznaka, da se
začenja zaporedje ničel

razmerje je 5:35





Run-length encoding (RLE)

- Podobno kot prej ponovitve simbolov zamenjamo s **simbolom** in **številom pojavitev**
- V najslabšem primeru, ko ni ponavljanj, dobimo precej večjo kodo (dve vrednosti namesto ene)
- Uporaba:
 - del JPEG algoritma
 - stiskanje slik (po vrsticah), npr. Microsoft RLE

Primer:

*1111222333333111122
22*

stisnemo kot:

*(1,4),(2,3),(3,6),(1,4),(2,
4)*

razmerje je 10:21





- Simbole zakodiramo v **različno dolge** kodne besede
 - dolžina kodnih besed je povezana z verjetnostjo pojavitve simbola - **bolj verjetni** simboli imajo **krajšo** kodno besedo
- **Shannonov teorem** o brezšumnem izvornem kodiranju:
 - optimalna dolžina kodne besede simbola je enaka **lastni informaciji** simbola:
$$I(x) = -\log_2 P(x)$$
 - lastna informacija = vrednost informacije, ki jo prejmemo, ko sprejmemo nek simbol

Entropijska kodiranja

Primer:

AABACABAAB

$P(A)=6/10$

$P(B)=3/10$

$P(C)=1/10$

$I(A) = -\log_2 6/10 \cong 0.74$ bita

$I(B) = -\log_2 3/10 \cong 1.74$ bita

$I(C) = -\log_2 1/10 \cong 3.32$ bita





(Shanonova) Entropija

- Entropija je **povprečna** lastna informacija podatkov
 - če so podatki sestavljeni iz množice simbolov $\{x_1, x_2, \dots, x_n\}$
 - pojavitve simbolov so statistično **neodvisne**
 - je entropija v bitih:

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

- Entropija podaja **spodnjo mejo** števila bitov, s katerim lahko podatke predstavimo
 - ob predpostavki neodvisnosti

Primer:

AABACABAAB

$$\begin{aligned} H &= -6/10 \cdot \log_2 6/10 \\ &\quad -3/10 \cdot \log_2 3/10 \\ &\quad -1/10 \cdot \log_2 1/10 \\ &= 1.295 \text{ bitov} \end{aligned}$$

Spodnja meja pri stiskanju brez izgub je torej 1.295 bita na simbol



(Shanonova) Entropija

■ Primeri:

- imamo sivinsko sliko, 256 nivojev sivine, če predpostavimo, da so piksli neodvisni, vse sivine so enako verjetne:

$$H = -256 * 1/256 * \log_2 1/256 = 8 \text{ bitov}$$

- kot zgoraj, le da je polovica pikslov 0, ostali pa so enako verjetni

$$H = -0.5 * \log_2 0.5 - 255 * 1/510 * \log_2 1/510 \cong 5 \text{ bitov}$$





- Kako **določiti kodne besede**, ki bodo čimbolj optimalno stisnile podatke in se bomo z njimi **približali spodnji meji** velikosti (entropiji)?

- Primer: kako zakodirati

$X=\{A,B,C,D\}$

$P(A)=0.125$

$P(B)=0.125$

$P(C)=0.25$

$P(D)=0.5$

$H(X) = 1.75$ bitov

Entropijsko kodiranje

- **Kodiranje 1:**

A=01 B=11

C=1 D=0

Kako dekodiramo 1101?

Dekodiranje ni enolično, torej izbrane kode niso smiselne!

- **Kodiranje 2:**

A=110 B=111

C=10 D=0

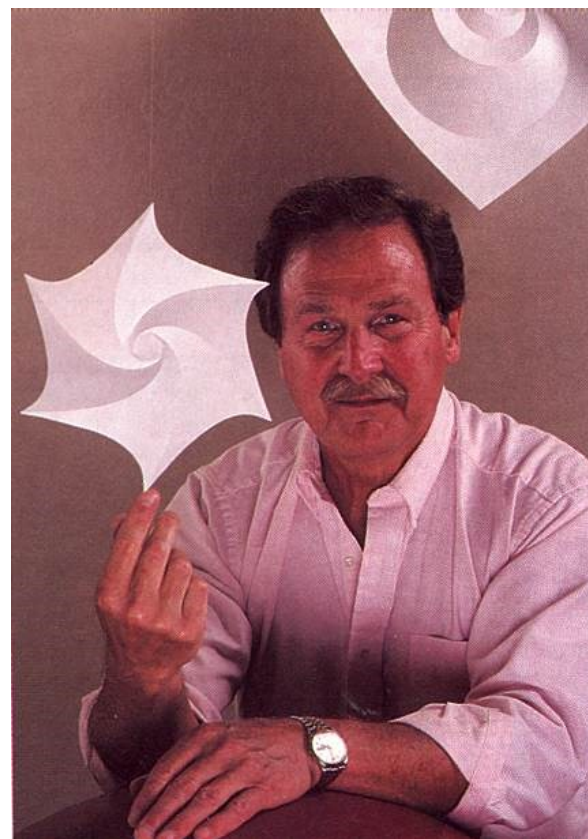
- Predpone kodnih besed so unikatne, torej je dekodiranje enolično.
- Dosežena je tudi spodnja meja, v povprečju porabimo 1.75 bitov ($3*0.125+3*0.125+2*0.25+0.5$)





Huffmanovo kodiranje

- Učinkovito entropijsko kodiranje
 - dolžina kodnih besed je povezana z verjetnostjo pojavitve simbola - **bolj verjetni** simboli imajo **krajšo** kodno besedo
 - dve najdaljši kodni besedi se razlikujeta le v zadnjem bitu
- Za izračun potrebujemo **verjetnosti** simbolov
 - ni vedno mogoče dobiti teh verjetnosti, npr. če so vsebine pretočne
- Uporaba v DEFLATE (zip), JPEG, MP3 ...



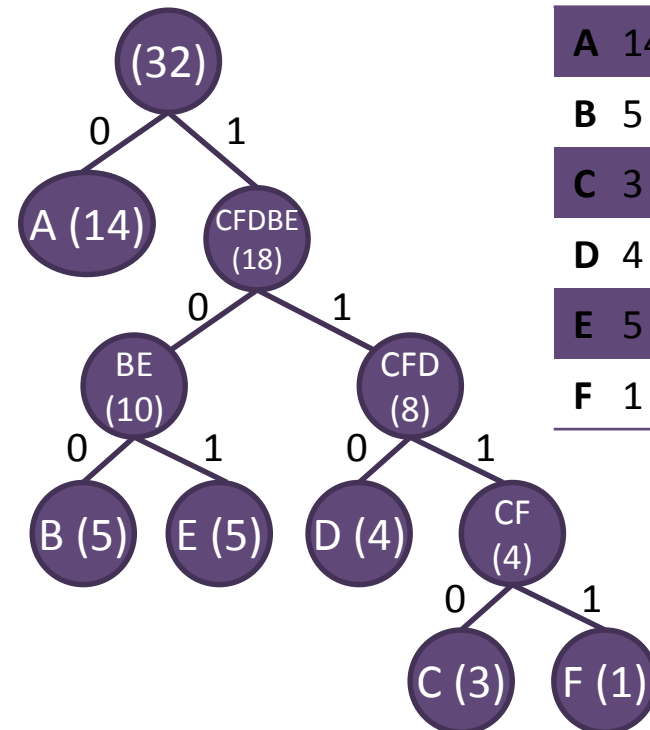
David A. Huffman

Huffmanovo kodiranje

- Gradimo **binarno drevo** od spodaj navzgor:
 - vse simbole dodamo v vrsto V
 - ponavljamo, dokler v vrsti V ne ostane en sam element:
 - iz vrste vzamemo dva elementa e_1 in e_2 z **najmanjšima** verjetnostma
 - naredimo nov element drevesa n , ki ima za naslednika e_1 in e_2 in katerega verjetnost je vsota verjetnosti e_1 in e_2
 - vejama drevesa med n in e_1 in e_2 določimo oznaki 0 in 1
 - n dodamo v V
 - iz V izbrišemo e_1 in e_2
- Kodo simbola dobimo iz oznak na vejah drevesa, ki vodijo do simbola

Primer:

*ABEAAAACDEAAABBBDDDEE
AAACABCDEF*



N kodna beseda		
A	14	0
B	5	100
C	3	1110
D	4	110
E	5	101
F	1	1111



Huffmanovo kodiranje

- Dekodiranje je enostavno, poznati je potrebno **tabelo kodnih besed** (*code book*)
 - smiselno je, da je shranjena oz. prenesena pred samo kodo
 - tabela kodnih besed tudi zavzema nekaj prostora, vendar navadno malo v primerjavi s podatki
- Nobena kodna beseda ni predpona kaki drugi kodni besedi – predpone so **unikatne**
 - zagotavlja unikatno dekodiranje
- Pri slikah (npr. JPEG) se kodira transformirane 8x8 velike bloke slike (glej npr. <http://www.impulseadventure.com/photo/jpeg-huffman-coding.html>)

Primer:

0100011010101111111010
000101

ABADEAFCBAAE

**kodna
beseda**

A 0

B 100

C 1110

D 110

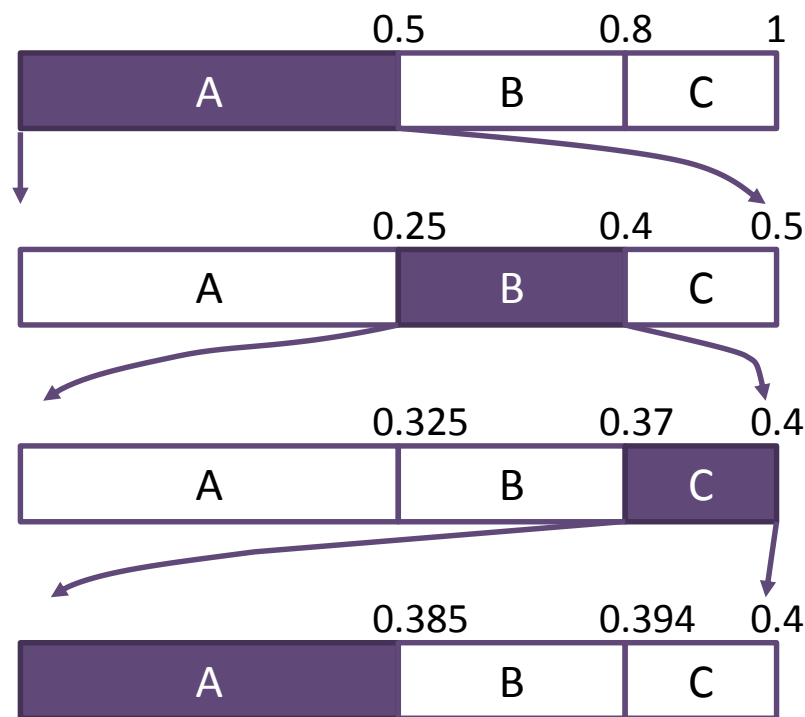
E 101

F 1111



- Problemi pri Huffmanu
 - kodne besede za 1 simbol imajo vedno **celo število** bitov (1,2,3 itn.) – ni optimalno
- Aritmetično kodiranje
 - entropijsko kodiranje
 - **zaporedje** vhodnih simbolov zakodiramo kot **eno realno število**
 - število bitov za kodno besedo na simbol ni nujno celo število
 - lahko nekoliko bolje stisne kot Huffman

Aritmetično kodiranje





- Aritmetično kodiranje temelji na **deljenju** številčnega **intervala** glede na verjetnosti simbolov
 1. imamo n dolgo zaporedje simbolov $s_1, s_2, s_3 \dots s_n$
 2. interval $I=[0,1)$
 3. for $i = 1..n$
 - a) I razdelimo proporcionalno glede na verjetnosti $p(s_1), p(s_2) \dots p(s_n)$
 - b) I =podinterval simbola s_i
 4. koda je število znotraj I , ki zasede najmanj bitov

Aritmetično kodiranje

Primer:

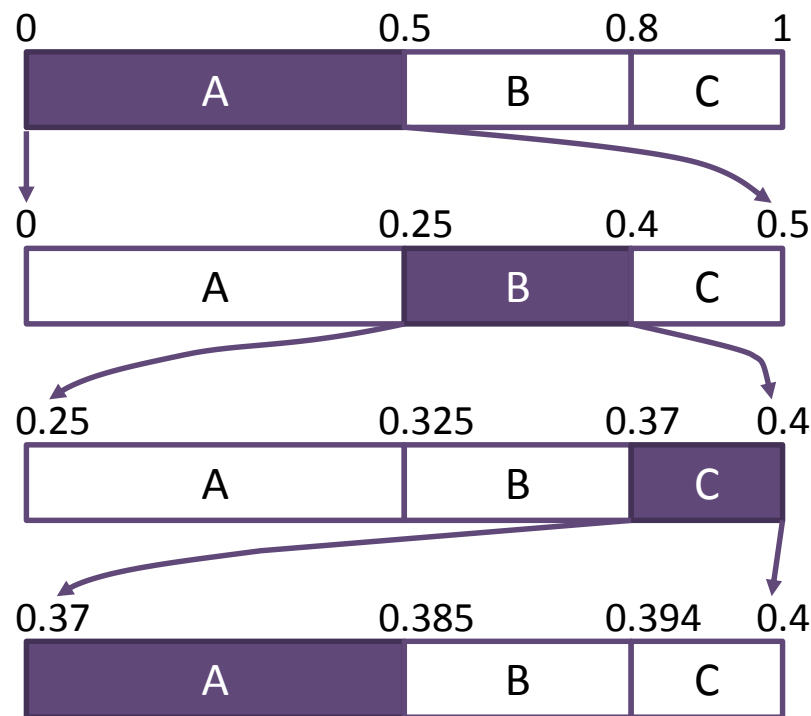
kodiramo ABCA

$p(x)$

A 0.5

B 0.3

C 0.2



Rezultat je število v $[0.37 \text{ in } 0.385)$, ki zasede najmanj bitov, npr. 0.38



- Dekodiranje: imamo kodo (število), število simbolov in verjetnosti
 1. interval razdelimo po verjetnostih, po vrsti pogledamo v kateri interval število sodi
 2. dekodiramo simbol
 3. izberemo podinterval simbola in ponavljamo

Aritmetično kodiranje

Primer:

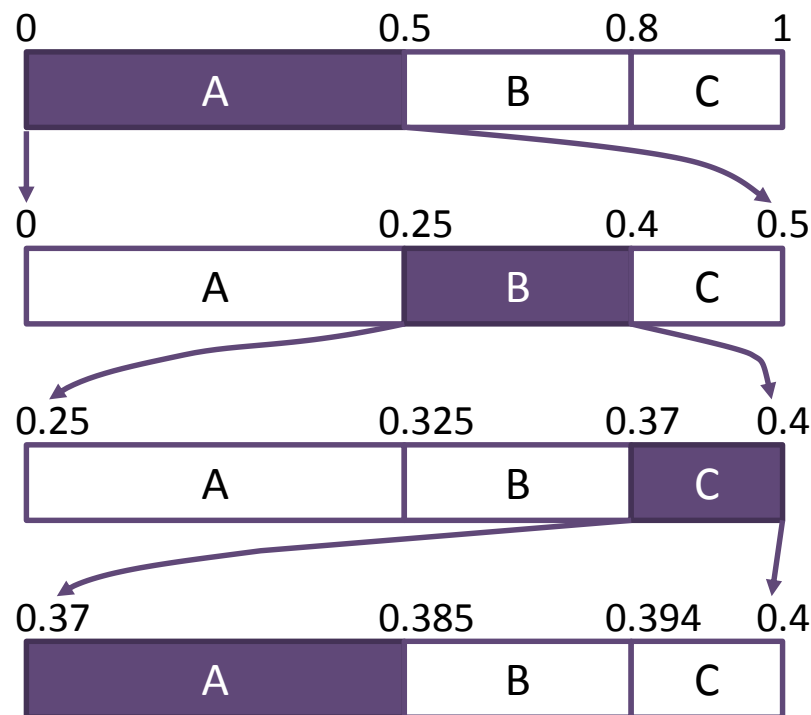
*dekodiramo 0.38,
št. simbolov je 4*

$p(x)$

A 0.5

B 0.3

C 0.2



Rezultat je ABCA





- Navadno ne uporabljamo decimalnih števil s plavajočo vejico

- uporabimo npr. binarne ulomke:
 $0.11 = 1/2 + 1/4$
(glej tudi [konverter](#))

- Lahko tudi vnaprej pripravimo tabelo kodiranja vseh možnih kratkih zaporedij simbolov

- hitro, ampak pomnilniško potratno

Aritmetično kodiranje

Primer:

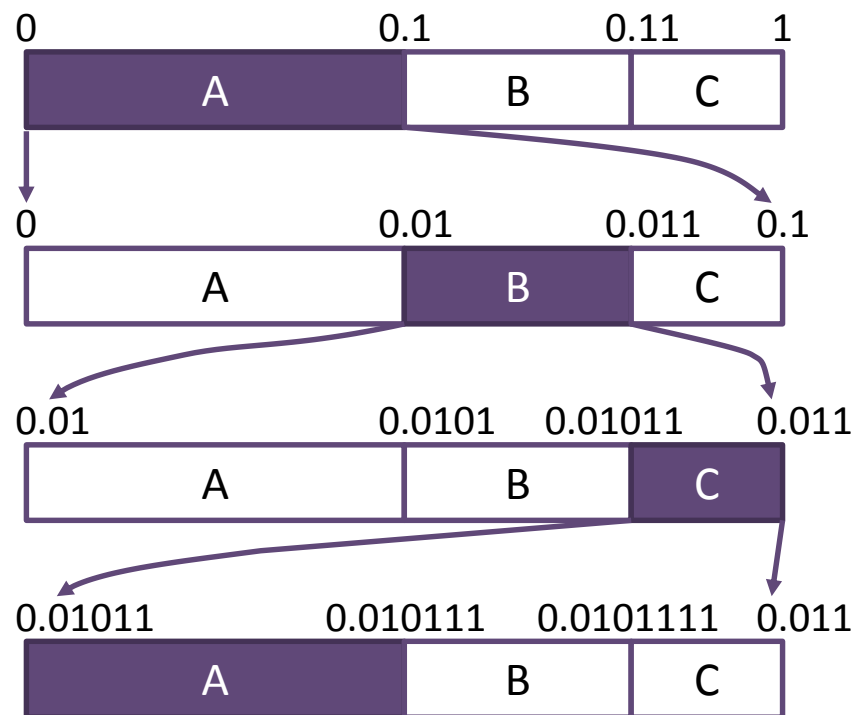
kodiramo 0.38

$p(x)$

A 0.5

B 0.3

C 0.2



Koda je npr. 0.01011




- Velikokrat **ne poznamo** natančnih verjetnosti simbolov (npr. pretočne vsebine itn.)
- Obstaja vrsta entropijskih kodiranj, ki določajo kodne besede celih števil
 - predpostavka, da so **večja števila manj verjetna**
 - tabela kodnih besed je **fiksna** (ni jih potrebno računati)
 - hitrejša, ker je tabela fiksna, poleg tega je ni treba prenašati skupaj s kodo
- Veliko variant
 - Golomb, Rice, Fibonacci, Elias ...
- Dosegajo nekoliko slabšo stiskanje kot Huffman oz. aritm. kodiranje, so pa lahko hitrejša
- Uporaba v FLAC, UTF-8, Apple lossless, H.264...

Univerzalne kode idr.

simbol	kodna b.
0	1 00
1	1 01
2	1 10
3	1 11
4	0 1 00
5	0 1 01
6	0 1 10
7	0 1 11
8	00 1 00
9	00 1 01
10	00 1 10
11	00 1 11
12	000 1 00
13	000 1 01
14	000 1 10
15	000 1 11

Rice kodne besede za števila med 0 in 15 (če je parameter kodiranja enak 4)



- Če ne kodiramo le posameznih simbolov, ampak zaporedja simbolov, lahko ustvarimo **slovarje**
 - pojavitev zaporedja simbolov zamenjamo z referenco na slovar
- **Statični slovarji** 
 - fiksni, določeni vnaprej
- **Dinamični slovarji**
 - jih ustvarimo iz podatkov
 - LZ77, LZW itn.

Kodiranje s slovarji

Primer:

This is an example

beseda	koda
an	1
as	2
is	3
This	4
example	5
case	6
question	7

Kodiramo kot par koda slovarja, položaj
 $(4,0)(3,5)(1,8)(5,11)$



LZ77 (Lempel-Ziv 1977)

- Uporablja se npr. v DEFLATE (ZIP)
- Ko kodiramo, iščemo preteklo **najdaljšo** pojavitev **zaporedja** simbolov
- Če najdemo, **shranimo**:
 - razdaljo do trenutnega položaja
 - dolžino zaporedja
 - naslednji znak
- Pretekle pojavitve iščemo znotraj **omejenega okna**, npr. 4096 simbolov
 - prvih n znakov ne kodiramo
- Slovar je **impliciten** kot referenca na preteklost
- Dekodiranje je enostavno, samo sledimo referencam

Primer:

ABCD CDABCEBCDA

Kodiramo kot:

ABCD(2,2,A)(6,2,E)(9,3,A)

Kako dekodiramo:

ABCE(3,2,D)(6,3,E)(8,2,C)

ABCEBCDBCEEEBC



Lempel-Ziv-Welsch (LZW)

- compress, GIF, TIFF ...
- Sproti gradi slovar**
 - začnemo z začetnim slovarjem, npr. za vsako posamezno črko

```
dict = { a,b,c,...,z }; // initial dictionary
code = empty;
word = nextSymbol();
while (!EOF)
{
    c = nextSymbol();
    if (dict.hasWord(word + c))
        word = word + c;
    else
    {
        code+=dict.getCode(word);
        dictionary.addWord(word +c);
        word = c;
    }
}
code+=dictionary.getCode(word);
```

Primer: *ababcbabab*

	beseda	koda
začetni slovar {	<i>a</i>	1
	<i>b</i>	2
	<i>c</i>	3
	<i>ab</i>	4
	<i>ba</i>	5
	<i>abc</i>	6
	<i>cb</i>	7
	<i>bab</i>	8

Koda:

1 2 4 3 5 8

Lempel-Ziv-Welsch (LZW)

- **Dekodiranje:** ni potrebno prenašati slovarja (razen začetnega), ga izgradimo iz kodiranih podatkov!

```
dict = { a,b,c,...,z };
oldWord = empty;
decode=empty;
while ( (code = nextCodeWord()) != EOF )
{
    if (dict.hasCode(code))
        word = dict.getWord(code);
    else
        word = oldWord + oldWord[0];

    decode = decode + word;

    if (oldWord is not empty)
        dict.addWord(oldWord + word[0]);

    oldWord = word;
}
```

Primer: 1 2 4 3 5 8

začetni slovar	beseda	koda
	<i>a</i>	1
	<i>b</i>	2
	<i>c</i>	3
	ab	4
	ba	5
	abc	6
	cb	7
	bab	8

Dekodirani simboli:

a b ab c ba bab



Kodiranje razlik

- Pri multimedijskih signalih lahko izkoriščamo tudi časovno in prostorsko **redundantnost**
 - namesto da signal neposredno kodiramo, poskušamo “napovedati” prihodnost, in **kodiramo le razlike**
 - razlike imajo navadno **manjšo entropijo**, zato jih lahko bolj stisnemo

Primer:

1 2 3 4 5 4 3 2 1

$H(x)=2.281$

Predpostavimo, da bo naslednja vrednost enaka prejšnji, napoved je torej:

0 1 2 3 4 5 4 3 2

Razlika med dejanskim signalom in napovedjo je:

1 1 1 1 1 -1 -1 -1 -1

$H(x)=0.9911$



8 bitna sivinska slika

original

H=7.64

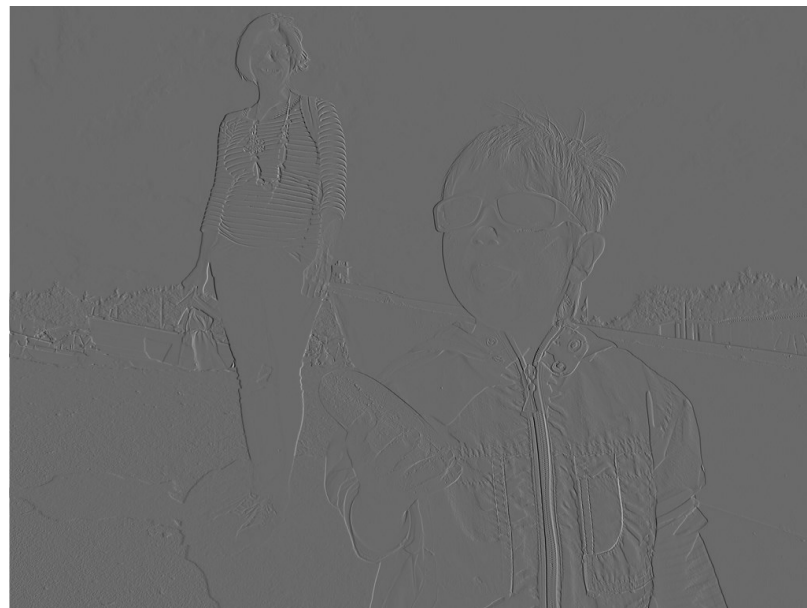


Kodiranje razlik

Enostavna funkcija za razliko:

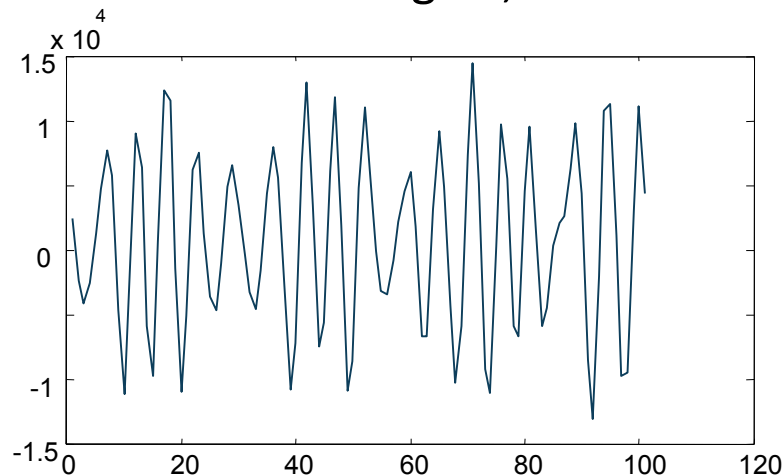
$\text{pix}(x,y) = \text{pix}(x,y) - \text{p}(x-1,y)$

H=3.43

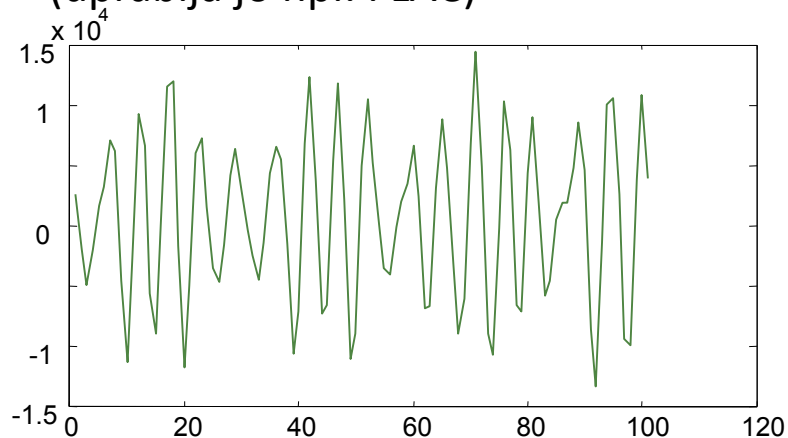




16 bitni zvočni signal, $H=14.02$

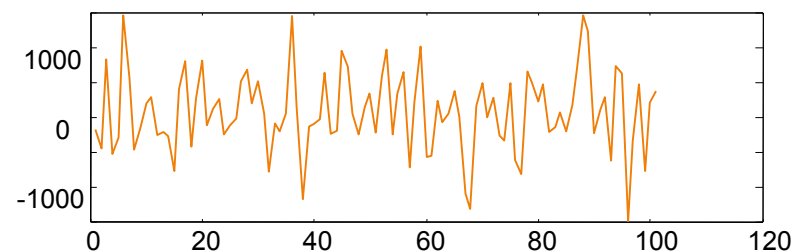


Bolj kompleksna funkcija za
napovedovanje – Linear Predictive Coding
(upravlja jo npr. FLAC)



Kodiranje razlik


Razlika med originalom in LPC napovedjo
 $H=11.17$



Podobno kodiranje uporablja FLAC



Konkretna uporaba stiskanja brez izgub

- Nekateri brezizgubni formati:
 - DEFLATE (libz: zip, gzip, PNG, PDF ...) uporablja LZ77+Huffman
 - TIFF uporablja lahko RLE ali LZW
 - GIF in compress uporabljata LZW
 - FLAC uporablja Rice kode (+ LPC za napovedovanje)
 - Nekateri formati z izgubo:
 - mp3, AAC, JPEG uporabljata RLE in Huffman
 - H.264 uporablja aritmetično kodiranje ali Golomb kode
- 



REFERENCE

- David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1
 - DEFLATE Compressed Data Format Specification version 1.3
- 