Teorija informacij

Entropija

Entropija je mera za nedoločenost (neurejenost) sistema. Predpostavimo, da v nekem sistemu lahko nastopi n različnih stanj z verjetnostmi $p_1, p_2, \ldots, p_n$, V tem primeru je entropija sistema:

$$H(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Poseben primer: Vzemimo, da sistem sestavljata samo dve možni stanji. Največja nedoločenost sistema z dvema stanjema nastopi v primeru, ko sta obe stanji enako verjetni – najtežje napovemo, kaj se bo v sistemu zgodilo.

– Primer 1 –

Definicija enote za količino informacije:

> **En bit** je količina informacije, ki jo dobimo, ko dobimo odgovor na vprašanje na katerega sta možna natanko dva enako verjetna odgovora.

– Primer 2 –

$$X = \begin{pmatrix} a & b & c & d & e & f & g & h \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \end{pmatrix}$$

$$H(p_1, p_2, \ldots, p_8) = -\sum_{i=1}^{8} p_i \log_2 p_i = -8\frac{1}{8}(-3) = \boxed{3}$$

– Primer 3 –

$$X = \begin{pmatrix} a & b & c & d & e & f & g & h \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{8} & \frac{1}{8} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \end{pmatrix}$$

$$H(p_1, p_2, \ldots, p_8) = -\sum_{i=1}^{8} p_i \log_2 p_i =$$

$$-2\frac{1}{4}(-2) - 2\frac{1}{8}(-3) - 4\frac{1}{16}(-4) = \boxed{2.75}$$

Entropija sistema predstavlja *spodnjo mejo števila bitov*, ki so potrebni za predstavitev vseh elementov tega sistema.

Mera za povezanost dveh spremenljivk na osnovi entropije

Za merjenje povezanosti poljubnih dveh spremenljivk (lahko tudi nominalnih) lahko uporabljamo Cramerjev koeficient (*Cramer's V*).

$$V = \sqrt{\frac{\chi^2}{n(k-1)}}$$

Poleg njega obstaja še koeficient Rajskega (1964), ki je zgrajen na osnovi entropije:

Imejmo dve spremenjivki X in Y. Spremenljivka X naj zavzame n različnih vrednosti, spremenljivka Y pa m različnih vrednosti.

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

$$H(Y) = -\sum_{i=1}^{m} p(y_i) \log_2 p(y_i)$$

in

$$H(XY) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i, y_j)\log_2 p(x_i, y_j)$$

Informacija med spremenljivkama X in Y je definirana takole:

$$I(X, Y) = H(X) + H(Y) - H(XY)$$

Informacija $I(X, Y)$ doseže vrednost 0, natanko takrat ko za vsak par x_i in y_j velja $p(x_i, y_j) = p(x_i)p(y_j)$, kar pomeni, da sta spremenljivki neodvisni.

Informacija $I(X, Y)$ doseže največjo vrednost, natanko takrat ko med spremenljivkama obstaja funkcijska zveza – v vsakem stolpcu in vsaki vrstici ustrezne kontingenčne tabele je največ en od nič različen element. Tedaj velja:

$$H(X) = H(Y) = H(XY) = I(X, Y)$$

Torej je informacija I(X,Y) mera funkcijske odvisnosti (določenosti) med spremenljivkama X in Y.

Koeficienti Rajskega:

$$R(X \leftrightarrow Y) = \frac{I(X,Y)}{H(XY)}$$

$$R(X \rightarrow Y) = \frac{I(X,Y)}{H(Y)}$$

$$R(X \leftarrow Y) = \frac{I(X,Y)}{H(X)}$$

Vsi koeficienti zavzamejo vrednosti med 0 in 1. Vrednost 0 zavzamejo, ko sta spremenljivki neodvisni.

$R(X \rightarrow Y) = 1$, ko je Y funkcija X,
$R(X \leftarrow Y) = 1$, ko je X funkcija Y in
$R(X \leftrightarrow Y) = 1$, ko obe spremenljivki ena drugo natanko določata.

Primer 1:

	y_1	y_2	y_3	Sum
x_1	2	2	1	5
x_2	2	1	2	5
Sum	4	3	3	10

$p(x_i, y_j)$	y_1	y_2	y_3	$p(x_i)$
x_1	0.2	0.2	0.1	0.5
x_2	0.2	0.1	0.2	0.5
$p(y_j)$	0.4	0.3	0.3	1

$$R(X \leftrightarrow Y) = 0.0194$$

$$R(X \rightarrow Y) = 0.0312$$

$$R(X \leftarrow Y) = 0.0490$$

Vrednosti vseh treh koeficientov so majhne, na osnovi vrednosti ene spremenljivke ne moremo napovedati vrednosti druge spremenljivke.

Primer 2:

	y_1	y_2	y_3	Sum
x_1	0	3	0	3
x_2	4	0	3	7
Sum	4	3	3	10

$p(x_i, y_j)$	y_1	y_2	y_3	$p(x_i)$
x_1	0	0.3	0	0.3
x_2	0.4	0	0.3	0.7
$p(y_j)$	0.4	0.3	0.3	1

$$R(X \leftrightarrow Y) = 0.5610$$

$$R(X \rightarrow Y) = 0.5610$$

$$R(X \leftarrow Y) = 1$$

Spremenljivka X je funkcija Y: če poznamo vrednost spremenljivke Y lahko natanko napovemo vrednost spremenljivke X, obratno pa ni res.

Primer 3:

	y_1	y_2	y_3	Sum
x_1	4	0	0	4
x_2	0	0	3	3
x_3	0	3	0	3
Sum	4	3	3	10

$$R(X \leftrightarrow Y) = 1$$

$$R(X \rightarrow Y) = 1$$

$$R(X \leftarrow Y) = 1$$

Spremenljivki X in Y ena drugo natanko določata.

Primer 4:

	y_1	y_2	Sum
x_1	2	2	4
x_2	2	2	4
Sum	4	4	8

$$R(X \leftrightarrow Y) = 0$$

$$R(X \rightarrow Y) = 0$$

$$R(X \leftarrow Y) = 0$$

Spremenljivki X in Y sta neodvisni

Kodiranje podatkov

Naloga: Kako predstaviti elemente, ki nastopajo v nekem sistemu, tako da bo povprečna dolžina zapisa (*kode*) čimkrajša. Ta problem je zelo pogost, npr.

- pri shranjevanju podatkov na disk
  (porabiti čimmanj prostora)
- pri prenosu podatkov
  (podatke čimhitreje prenesti)

V nadaljevanju se bomo omejili na *dvojiško ali binarno kodiranje* – elemente sistema želimo predstaviti samo z znakoma 0 in 1.

Označimo z d_i dolžino kode i-tega elementa, z $\overline{d}$, pa povprečno dolžino kode (matematično upanje slučajne spremenljivke).

$$\overline{d} = \sum_{i=1}^{n} d_i p_i$$

Velja: $\overline{d} \geq H(p_1, p_2, \ldots, p_n)$.

Kodiranje je **optimalno**, če je $\overline{d} = H(p_1, p_2, \ldots, p_n)$.

Učinkovitost kodiranja:

$$\xi = \frac{H(p_1, p_2, \ldots, p_n)}{\overline{d}} \in [0, \ldots, 1]$$

$1 - \xi$ se imenuje **odvečnost** kodiranja.

– Primer 4 –

$$X = \begin{pmatrix} a & b & c & d & e & f & g & h \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \\ 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{pmatrix}$$

$$\overline{d} = \sum_{i=1}^{8} d_i p_i = 8 * 3 * \frac{1}{8} = \boxed{3}$$

$$\overline{d} = H(p_1, p_2, \ldots, p_8), \quad \xi = 1, \quad 1 - \xi = 0$$

$\rightarrow$ Kodiranje je optimalno

– Primer 5a –

$$X = \begin{pmatrix} a & b & c & d & e & f & g & h \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{8} & \frac{1}{8} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\ 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{pmatrix}$$

$$\overline{d} = \sum_{i=1}^{8} d_i p_i = \boxed{3} > \boxed{2.75} = H(p_1, p_2, \ldots, p_8)$$

$$\xi = \frac{2.75}{3} = 0.92, \quad 1 - \xi = 0.08$$

$\rightarrow$ Kodiranje ni optimalno

– Primer 5b –

$$X = \begin{pmatrix} a & b & c & d & e & f & g & h \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{8} & \frac{1}{8} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\ 00 & 01 & 100 & 101 & 1100 & 1101 & 1110 & 1111 \end{pmatrix}$$

$$\overline{d} = \sum_{i=1}^{8} d_i p_i = 2 * 2 * \frac{1}{4} + 2 * 3 * \frac{1}{8} + 4 * 4 * \frac{1}{16} = \boxed{2.75}$$

$$\overline{d} = H(p_1, p_2, \ldots, p_8), \quad \xi = 1, \quad 1 - \xi = 0$$

$\rightarrow$ Kodiranje je optimalno

Pri kodiranju v primeru 5a imajo vse kode elementov enako dolžino (3 dvojiški znaki). Tako kodiranje imenujemo *kodiranje s stalno dolžino zapisa*.

Kodiranje s stalno dolžino dekodiramo tako, da beremo po toliko znakov naenkrat kolikor je dolžina kode:

Zaporedje zakodirano po kodiranju 5a:

$$111000001100 \rightarrow 111\ 000\ 001\ 100 \rightarrow habe$$

Pri kodiranju v primeru 5b pa so dolžine različne (od 2 do 4 bitov) – *kodiranje s spremenljivo dolžino zapisa*. V tem primeru moramo dodatno zahtevati, da je kodiranje **enopomensko** – nobena koda ne sme nastopati kot predpona kaki drugi kodi.

– Primer 6 –

Vzemimo kodiranje v primeru 5b in dekodirajmo naslednje zaporedje: 111100011100

Edina rešitev je $habe$ ($1111\ 00\ 01\ 1100$).

Kodiranje v primeru 5b je enopomensko: nobena koda ne nastopa kot predpona kaki drugi kodi.

Kodiranje

$$X = \begin{pmatrix} a & b & c & d & e & f & g & h \\ 00 & 01 & 000 & 101 & 1100 & 1101 & 1110 & 1111 \end{pmatrix}$$

pa ni enopomensko: koda znaka a nastopa kot predpona kodi znaka c. Tako lahko npr. zaporedje 0001101, dekodiramo na dva načina: abd ali cf

Če kodiranje s spremenljivo dolžino ni enopomensko, lahko enolično dekodiranje zagotovimo s presledki med kodami, vendar pomeni to dodatne bite.

Naloga: Kako priti do kodiranja, ki bo čimbliže optimalnemu kodiranju (optimalnega ni mogoče vedno doseči)? Eden od postopkov je Huffmanovo kodiranje (*Huffman coding*, 1952).

Huffmanovo kodiranje

je način dvojiškega kodiranja s spremenljivo dolžino, ki se skuša čimbolj približati spodnji meji potrebnih bitov določenih z entropijo.

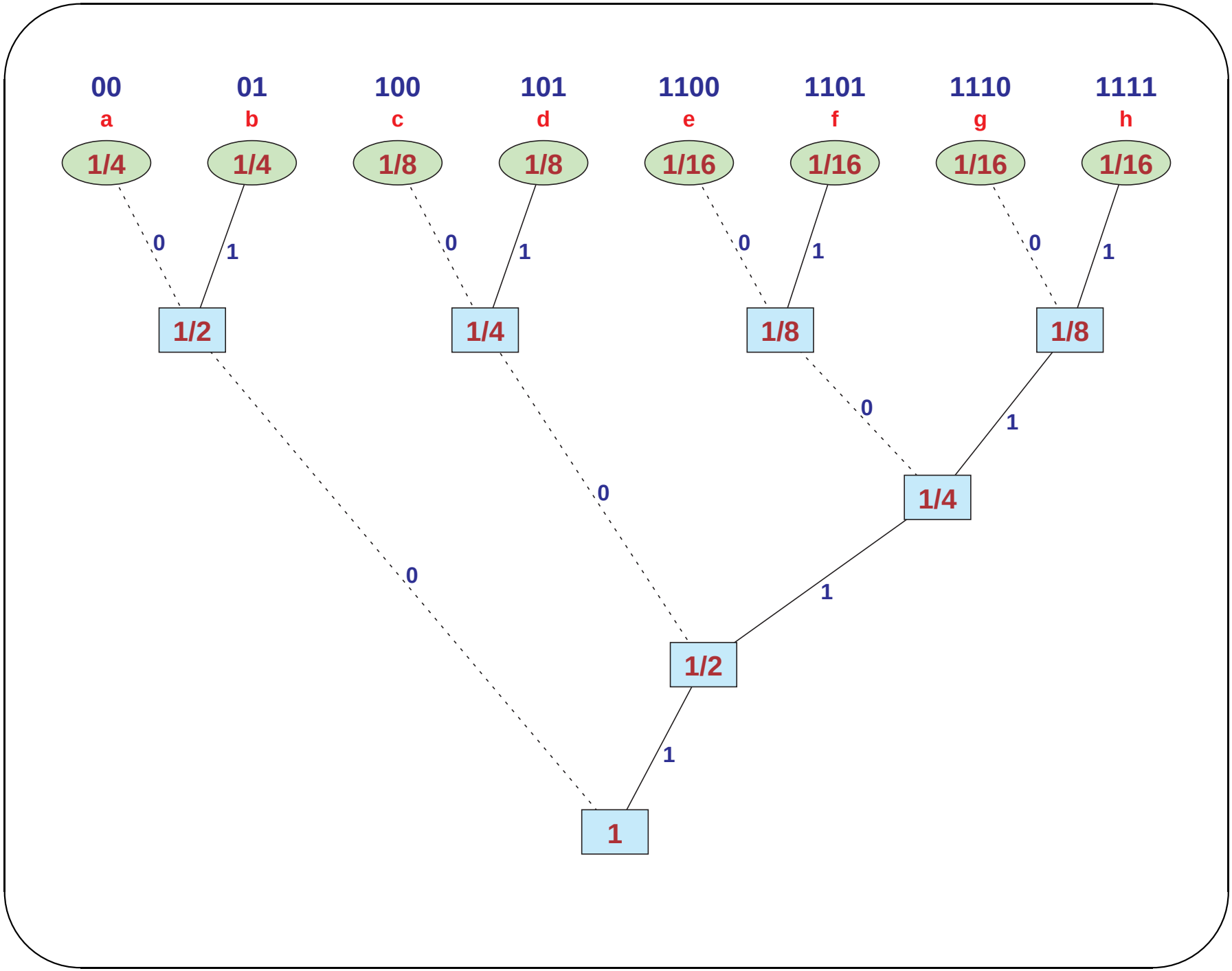
Kodiranje je naslonjeno na izgradnjo dvojiškega drevesa (vsako vozlišče ima dva naslednika).

V določenih primerih lahko s pomočjo tega algoritma dobimo več enako dobrih rešitev (dreves).

Algoritem:

1. Uredimo elemente (simbole) po padajoči vrednosti nastopa posameznih znakov. To bodo listi dvojiškega drevesa, ki ga gradimo.
2. V drevesu poiščemo vozlišči z najmanjšima verjetnostima.
3. Iz dobljenih dveh vozlišč zgradimo novo vozlišče z verjetnostjo enako vsoti verjetnosti vozlišč.
4. Ponavljamo koraka 2 in 3, dokler ne ostane le eno vozlišče – koren drevesa.
5. Določimo posamezne bite za vse elemente: Začnemo pri korenu, če se premaknemo levo dodamo bit 0 sicer bit 1. Ponavljamo, dokler niso obiskana vsa vozlišča v drevesu.
6. Preberemo kode znakov, tako da se od korena sprehodimo do posameznega znaka.

Kodiranje 5b je dobljeno na naslednji način:


00
a
1/4
01
b
1/4
100
c
1/8
101
d
1/8
1100
e
1/16
1101
f
1/16
1110
g
1/16
1111
h
1/16
0
1
0
1
0
1
0
1
1/2
1/4
1/8
1/8
0
1
0
1
1/4
0
1
1/2
1
1

– Primer 7 – – Primer 8 –

Stiskanje (zgoščanje) podatkov

Huffmanovo kodiranje je ena od metod za *brezizgubno stiskanje podatkov*. Brezizgubno stiskanje podatkov pomeni, da lahko zakodirano besedilo v popolnosti (brez izgub) pretvorimo v originalno besedilo. Sem spadata še *LZW* in *aritmetično kodiranje*. Zaradi hitrosti dekodiranja se v praksi upošteva le nekaj nivojev drevesa, preostanek pa se zakodira s stalno dolžino (vsi manj verjetni znaki bodo enako dolgi) – rezanje drevesa.

Poleg tega obstaja tudi *stiskanje z izgubami*, ki se v glavnem uporablja pri shranjevanju slik, zvoka in videa, kjer izguba nekaterih podatkov ni kritična (npr. *fraktalno zgoščevanje*, *JPEG*, *Waved compression*, *MPEG*). S tem, da dovolimo nekaj izgube, lahko zelo povečamo zgoščevanje, napake pa navadno oko (uho) niti ne opazi.

Stiskanje podatkov postaja vse bolj pomembno na področju shranjevanja in prenosa podatkov (Internet) saj bistveno zniža stroške.

`http://www.brunel.ac.uk/~dc95anm/`

Morsejeva abeceda

Primer kodiranja, ki ni enopomensko.

Avtorja: *Samuel Morse* in *Alfred Vail* v začetku 19. stol.

Vsak znak (črke, številke in posebni znaki) je predstavljen z zaporedjem . in – Uporabljala se je (se) pri prenosu podatkov na velike razdalje (telegrafija).

Bolj pogoste črke imajo krajše kode, (primer Morse.htm)

E . $p(E) = 0.105$ v angleškem besedilu

T –

A .– $p(A) = 0.063$

N –. $p(N) = 0.059$

I ..

F ..–.

O –––

..–...–.––– =??? (EA, IT, IN, FI, ...)

Kodiranje torej ni enopomensko, zato znake ločimo s presledki: .. | −. | .. − . | − − − (INFO)

... | − − − | ... (SOS)

Shema komunikacijskega kanala

*** Slika ***

V praksi imamo ponavadi komunikacijske kanale s šumom (motnjami), zato se lahko zgodi, da $K(S) \neq S'$. V tem primeru uporabljamo še dodatne metode:

- metode za odkrivanje prisotnosti napak
- metode za odpravljanje napak

V primeru, da želimo besedilo še dodatno zaščititi pred napadalci, tako da bo besedilo razumel le tisti, ki mu je namenjeno, si pomagamo s *kriptografijo*.

Odkrivanje prisotnosti napak

Namen tega postopka je odkriti, da napaka obstaja, ne pa najti mesto napake. Ena od teh metod je *kontrola parnosti*:

Celotno kodirano sporočilo razdelimo na bloke (npr. po 7 bitov). Za vsakim blokom dodamo še en bit, katerega vrednost bo taka, da bo skupno število enic v paketu sodo (*soda pariteta*). Včasih se namesto sode uporablja tudi *liha pariteta* (skupno število enic mora biti liho).

Primer: sproročilo: 10010111001101010011 0

10010111001101010011 0 $\rightarrow$ 1001011 1001101 0100110

10010110100110100100110 1 (soda pariteta)

10010111100110110100110 0 (liha pariteta)

Tako dopolnjeno sporočilo pošljemo naslovniku. Na njegovi strani ustrezna naprava razbije sporočilo na bloke po 8 bitov in preveri, če je število enic res sodo (soda pariteta) oziroma liho v primeru lihe paritete. Če pride do napake, zahteva naprava ponoven prenos ustreznega bloka.

Odpravljanje napak

Če želimo poleg tega, da ugotovimo prisotnost napake, napako tudi odpraviti brez ponovnega pošiljanja istega sproročila, si pomagamo z *redundantnimi biti* – odvečnimi biti, ki niso potrebni za razumevanje besedila ampak za nek drug namen (v našem primeru – odkrivanje napak). Ena od možnosti je. da vsak bit pošljemo večkrat, npr.:

sproročilo: 101

pošljemo kot: 111000111

(vsak bit sporočila pošljemo trikrat)

Ko naprava dobi tako sporočilo, je sposobna odkrivati vse enojne napake. V splošnem:

Če vsak bit pošljemo 2e+1 krat, lahko *odpravimo* e-kratne napake.

Če vsak bit pošljemo 2e-krat lahko *odpravimo* (e-1)-kratne napake, *samo odkrijemo* pa še prisotnost e-kratne napake.

Hammingova razdalja $d(b1, b2)$ med dvema besedama je število bitov v katerih se besedi razlikujeta.

Če pošiljamo vsak bit 5 krat in dobimo na izhodu 00110, proglasimo to besedo za znak 0, ker

$$d(00110, 00000) = 2$$

$$d(00110, 11111) = 3$$

Po Hammingovi razdalji je beseda bliže 00000, ki predstavlja znak 0.

Če pošiljamo vsak bit 4 krat in dobimo na izhodu 0001, proglasimo to besedo za znak 0, če pa dobimo na izhodu 0011, pa ne vemo katera beseda je to, vemo le, da je nekaj narobe (niso vsi znaki enaki).

Huffmanovo kodiranje – lastnosti

- vsak znak v besedilu zakodiramo posebej
- kode znakov so **cela** (dvojiška) števila
- optimalni rezultati kadar so verjetnosti znakov potence števila $\frac{1}{2}$
- dolžine kod bolj verjetnih znakov so krajše

V praksi so verjetnosti znakov zelo redko potence števila $\frac{1}{2}$.

V tem primeru so boljši algoritmi, ki vsakemu znaku namesto celega števila bitov dodelijo neko realno število bitov – zaporedni znaki si razdelijo isti bit.

Aritmetično kodiranje (Rissanen)

- celo besedilo obravnavamo kot enoto in ga zakodiramo z enim **realnim** številom z intervala $[0.37, \ldots, 0.46)$, ali $[0 \ldots 1)$
- daljša kot je beseda bolj natančno realno število potrebujemo (več decimalk)
- bolj verjetne črke zahtevajo manj natančne intervale – manj zožijo interval in zato se število potrebnih bitov manj poveča
- ker obravnavamo celo besedilo kot enoto dosežemo spodnjo mejo potrebnih bitov določenih z entropijo pri vsakem besedilu
- aritmetično kodiranje je računsko zahtevnejši postopek

Zaradi enostavnosti bomo v nadaljevanju uporabljali interval $[0 \ldots 1)$.

Postopek kodiranja:

1. Vzamemo vrstni red vseh različnih črk, ki nastopajo v besedilu po abecedi.
   Interval $[0 \ldots 1)$ razdelimo v razmerju verjetnosti posameznih črk v besedilu.
   Dobimo verjetnostne intervale posameznih črk.
   Označimo spodnjo mejo verjetnostnega intervala trenutne črke z L_i, zgornjo mejo pa z U_i.

2. Trenutno spodnjo mejo označimo z L, zgornjo pa z U.
   Postavimo: $L = 0$ in $U = 1$.

3. Zapišemo celotno besedilo v stolpcu, poleg vsake črke izpišemo njeno spodnjo mejo L_i in zgornjo mejo U_i.

4. Dokler obstaja naslednja črka v besedilu ponavljamo:
   - Izračunamo novo spodnjo mejo:
     $L := L + (U - L) * L_i$
     in novo zgornjo mejo:
     $U := L + (U - L) * U_i$.

Realno število x, ki predstavlja dano besedilo je L (oziroma katerokoli število iz zadnjega dobljenega in-

tervala).

Postopek dekodiranja:

1. Vzamemo število x, ki predstavlja dano besedilo.
2. Ponavljamo
   - Poiščemo črko za katero velja: $L_i \leq x < U_i$.
     Črko izpišemo.
   - Če je $x = L_i$, potem konec (izpisali smo celotno besedilo)
     sicer
     $$x := \frac{x - L_i}{U_i - L_i}$$

Kodiranje LZW (*Lempel-Ziv-Welch*)

LZW je način zgoščevanja podatkov, ki izkorišča ponavljanje istih zaporedij znakov (zaporedij istih črk v besedilu ali zaporedij točk istih barv na sliki).

Modificirana metoda za zgoščevanje se uporablja v formatu za shranjevanje slik **GIF**.

Metoda je patentirana (Unisys).

Poznana zaporedja črk bomo v nadaljevanju imenovali *slovar*.

Postopek kodiranja:

1. Na začetku postavimo v slovar samo posamezne črke, ki nastopajo v vhodni besedi in jih zaporedoma oštevilčimo.
   Postavimo $P = \emptyset$.

2. C postane naslednji znak iz vhodne besede.

3. Ali je zaporedje $P + C$ že prisotno v slovarju?
   (a) **Če je potem** $P := P + C$ (P podaljšamo za C)
   (b) **sicer**
   - izpišemo številko kode zaporedja, ki jo predstavlja P na izhod;
   - dodamo zaporedje $P + C$ v slovar;
   - $P := C$ (P vsebuje samo znak C).

4. Če še nismo na koncu vhodne besede, se vrnemo na korak 2, sicer izpišemo zaporedno številko kode zaporedja, ki jo predstavlja P, na izhod.

Postopek dekodiranja:

1. Na začetku postavimo v slovar samo posamezne črke, ki nastopajo v besedi in jih zaporedoma oštevilčimo.
2. cW (codeWord) postane prva koda v zakodiranem zaporedju.
3. Izpišemo znak, ki v slovarju ustreza kodi cW na izhod.
4. $pW := cW$ (previousWord).
5. cW postane naslednja koda v zakodiranem zaporedju.
6. Ali je zaporedje, ki ustreza kodi cW že v slovarju?
   (a) **če je potem**
   - izpišemo ustrezno zaporedje na izhod
   - P postane zaporedje, ki ustreza kodi pW;
   - C postane prvi znak zaporedja, ki ustreza kodi cW;
   - zaporedje $P + C$ dodamo v slovar

   (b) **sicer**
   - P postane zaporedje, ki ustreza kodi pW;
   - C postane prvi znak zaporedja, ki ustreza pW;
   - izpišemo zaporedje $P + C$ na izhod in ga

dodamo v slovar (sedaj pripada kodi cW);

7. Če še nismo na koncu kodiranega zaporedja, se vrnemo na korak 4.

Zapisi in zgoščanje digitalnih slik

Zgoščanje slik je veliko bolj pomembno kot zgoščanje besedila, saj slike zavzamejo precej več prostora kot običajno besedilo.

Za predstavitev barv se uporablja 8, 16 ali 24 bitov. Če uporabimo za predstavitev 8 bitov lahko predstavimo $2^8 = 256$ različnih barv, če pa uporabimo 24 bitov pa lahko predstavimo $2^{24} = 16.7$ milijonov barv.

Primerjave potrebnega prostora za shranjevanje besedila in slik različnih podrobnosti (resolucij):

- en ekran besedila (25 vrstic, v vsaki vrstici 40 znakov) v obliki ASCII zavzame približno
  25 * 40 * 1= 1000 byte-ov = 0.001 MB
- slika na ekranu VGA (256 barv, resolucija 640 * 480) zavzame
  640 * 480 * 1 = 0.3 MB
  (300 krat več kot navadno besedilo)

- slika na ekranu SVGA (16.7 milijonov barv, resolucija 800 * 600) zavzame
  800 * 600 * 3 = 1.4 MB
  (1400 krat več kot navadno besedilo)
- slika na ekranu XGA (16.7 milijonov barv, resolucija 1024 * 768) zavzame
  1024 * 768 * 3 = 2.3 MB
  (2300 krat več kot navadno besedilo)

Formati za breizgubno predstavitev slik:

- **BMP** – *Bitmap* (Microsoft, Windows) – za vsako piko na sliki povemo kakšne barve je, torej za vsako piko porabimo v 24 bitnem načinu 24 bitov. Pomanjkljivost tega zapisa je ravno v tem, da ne uporablja nobenih zgoščevalbih tehnik, zato slike v tem zapisu zavzamejo zelo veliko prostora.

  Slika dimenzije 500*400 zavzame
  500*400*3=600,000 byteov.

- **PCX** (ZSoft, PaintBrush) – uporabljena je preprosta zgoščevalna tehnika **RLE** (Run Length Encoding), ki na sliki šteje zaporedne pike enake barve in jih zgosti. Namesto zapisa vseh pik enake barve, se v datoteko zapiše le barva in število zaporednih pik take barve.

  Slika dimenzije 500*400 zavzame v 24 bitnem načinu od 150,000 do 600,000 byteov.

- **TIFF** – *Tag Image File Format* (Aldus in Microsoft) Obstaja šest različnih specifikacij tega zapisa in sicer zapis brez zgoščevanja, zapis z uporabo Huffmanovega, LZW, RLE, Fax Group 3 in Fax Group 4 zgoščevanja. Faktor zgoščevanja, ki ga dosežemo s to vrsto zapisa je okoli 4.5.

Na splošno lahko z brezizgubnim stiskanjem dosegamo zgoščevalna razmerja do 6:1.

Formati za zgoščevanje slik z izgubami

- **JPEG** – *Joint Photographic Expert Group*

  Uporablja diskretno kosinusno transformacijo (varianta Fourierove transformacije).

  Sliko razdeli na bloke 8*8 pik in nad vsakim izvrši DCT s čimer dobimo neka števila.

  Dobljena števila zaokrožimo glede na željeno kvaliteto slike (100 – skoraj originalna kvaliteta, 1 – najslabša kvaliteta).

  Po zaokrožitvi ponavadi postane veliko koeficientov enakih 0, zato se uporabi zgoščevalna tehnika RLE.

  Zgoščevalni proces se konča s kodiranjem blokov z aritmetičnim ali Huffmanovim kodiranjem.

  Tako dobljeno zaporedje se shrani na datoteko.

  Pri prikazu (branju iz datoteke) se uporabi obratni postopek.

Slike v formatu JPEG imajo v primeru kvalitete 70 do 80 zgoščevalno razmerje večje od 20:1, v primeru zelo slabe kvalitete pa tudi do 100:1. Slabost zapisa JPEG je v tem, da postanejo pri večjih zgoščevalnih razmerjih na sliki vidni bloki 8*8 pik.

- **FIF** – *Fractal Image Format* – (M. Barnsley, Georgia)

  Postopek fraktalnega zgoščevanja temelji na iskanju podobnosti med manjšimi deli slike in opisu le teh s preslikavami.

  Zaradi tega je ta zapis nedvisen od ločljivosti – sliko lahko poljubno povečamo.

  Zgoščevalna razmerja tudi nad 200:1.

  Zgoščevanje po tem postopku vzame precej časa, zato se lahko poleg programskega uporabi tudi aparaturno zgoščevanje s posebno kartico, medtem ko prikaz ostane le programski.

  Slika: fraktal.gif