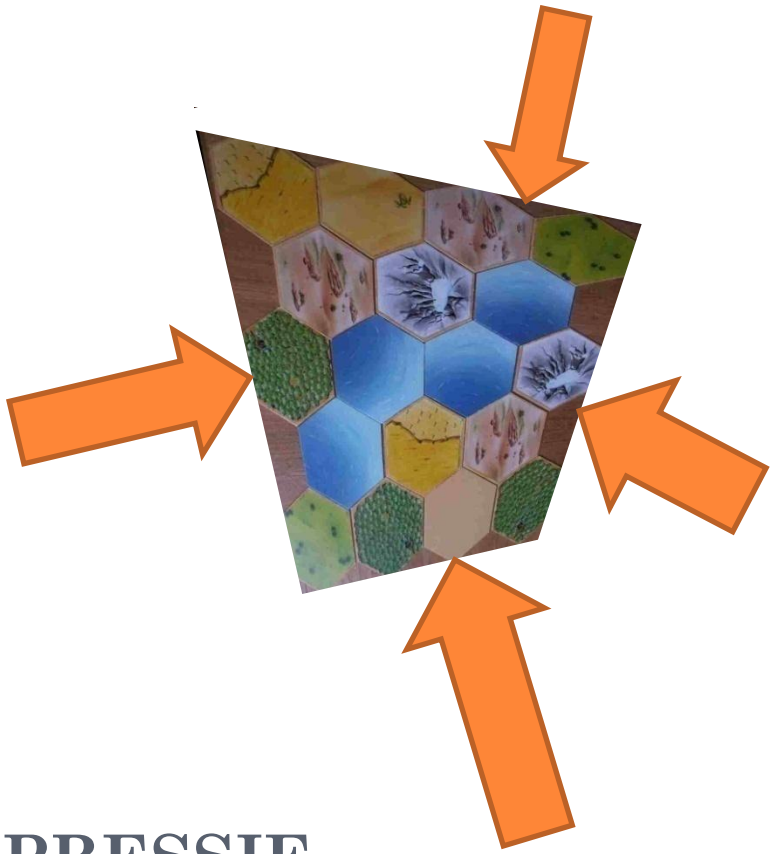


Data Compressie

datastructuren

Dr. D.P. Huijsmans

28 november 2012

Universiteit Leiden LIACS

Data Compressie waarom?

- Data opslaan kost ruimte (bits/bytes/words)
- Hoe minder ruimte hoe efficienter geheugen gebruik
- Hoe minder kosten voor opslag
- Data overzenden kost tijd (bits/sec)
- Hoe compacter gerepresenteerd hoe sneller en dus goedkoper overgezonden
  - **Data compressie is een echt datastructuren onderwerp**
- Voor processsor bewerkingen moeten data in een bepaald dataformat aangeboden worden, dit is vaak niet de meest compacte manier waarop die data opgeslagen zouden kunnen worden
- Nodig:
- standaard representatie  compacte representatie

Compressie ratio

N_o := aantal bytes data in oorspronkelijke vorm

N_c := aantal bytes data in gecomprimeerde vorm

compacter: gebruikt minder bytes $N_c < N_o$

Genormaliseerde Compressie Ratio: $0 < R_C = N_c / N_o <= 1.0$

Als $R_c >= 1.0$ dan neemt geheugenbeslag toe i.p.v. af

Twee tegenvoorbeelden

- Een bekend voorbeeld van een uitwisselformaat waarbij geen datacompressie plaatsvindt is postscript
- Om zeker te zijn dat elk byte in de postscript file een printbaar lowerASCII teken is, hakt postscript elke bytecodering (van 8 bits) in twee brokken van 4bits die afgebeeld worden op een printbaar 7-bits lowerASCII teken plus pariteitsbit
- De filegrootte in bytes van grijswaarden beelden b.v. wordt hierdoor 2x zo groot!!
- Ofwel $R_c = 2.0$ (>1.0 dus verlies i.p.v. winst)

Tegenvoorbeeld 2: priemgetalrepresentatie

- Omdat elk natuurlijk getal geschreven kan worden in de vorm van een som van priemgetallen verwacht een wiskundige dat een binair getalstelsel met een 0 of 1 voor een van hoog tot laag geordende reeks priemgetallen een compactere vorm dan binaire representatie op zal leveren:
- B.v. met 8 bits kan een willekeurige som van priemgetallen tussen 17 en 1 worden opgemaakt
- B.v. 10010101_{priem}=1*17+0*13+0*11+1*7+0*5+1*3+0*2+1*1=17+7+3+1=28
- Dit blijkt echter niet compacter te zijn dan de binaire representatie aangezien met 8 bits (1 byte) in de priemvorm [0..59] gerepresenteerd kan worden terwijl met 8-bits binair [0..127] kan worden weergegeven
- De R_c hiervoor is namelijk 128/60 = 2.13 >1.0

Lossless en Lossy Compression

We onderscheiden 2 hoofdvormen van data compressie:

Lossless (foutvrij):

na decompressie wordt precies de oorspronkelijke input data representatie verkregen

Lossy (benaderend):

na decompressie wordt een meer or minder goede benadering van de oorspronkelijke input data verkregen

Voor lossy compressie kan een veel lagere Compressie Ratio verkregen worden, maar hoe lager, hoe meer de benadering af gaat wijken van het origineel

We kijken vooral naar lossless compressie
Een belangrijke theoretische vraag daarbij is:
Wat is de limiet voor lossless compressie?

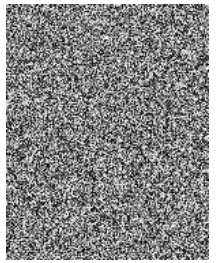
Redundantie en Entropie

- We gebruiken de term redundantie om aan te geven dat er meer bits/datum gebruikt wordt dan strict nodig zou zijn.

- Hoe maximaal compact een verzameling data waarden gerepresenteerd kan worden leert ons het begrip **Entropie** (ontleend aan de Natuurkunde)

- Entropie is een maat die gegeven een waarschijnlijkheidsverdeling van datawaarden het minimaal benodigde aantal bits/datum geeft:
- **Entropie = $\sum_{n=1,n}$ (-p(i).^{2}log(p(i)) bits**
- Met p(i) de fractie waarmee datum waarde i voorkomt in de verdeling i ε [1,n]

Pixel Entropie: minimum en maximum Waarde



In een grijswaarden beeld waar voor elk pixel een byte beschikbaar is voor opslag van de intensiteit ter plaatse hoort een grijswaarden verdeling $0 <= p(i) <= 1.0$ voor $i \varepsilon [0..255]$

De entropie van een bepaalde grijswaardeverdeling correspondeert met hoe chaotisch (hoe random) de verdelingsfunctie is

Minst chaotisch: alle pixels in dezelfde toestand j (egaal beeld):

$P(j)=1.0, P(i)=0.0, i<>j$

$$E_{min} = -0*^2\log(0) - ..1*^2\log 1... = 0 \text{ bit/pixel}$$

Maximale chaos: geen voorkeur bepaalde toestand (ruis beeld):

$P(I)=1/n$, $n=256$

$$E_{max} = - n*1/n*^2\log(1/256) = -^2\log(2^{-8}) = 8 \text{ bit/pixel}$$

$$E_{min} < E <= E_{max}$$

Het gebruikelijke 1 byte/pixel gaat dus uit van het slechtste geval

Entropie in een 2-toestanden systeem

The entropy in the case of two possibilities with probabilities p and $q = 1 - p$, namely

$$H = -(p \log p + q \log q)$$

is plotted in Fig. 7 as a function of p.

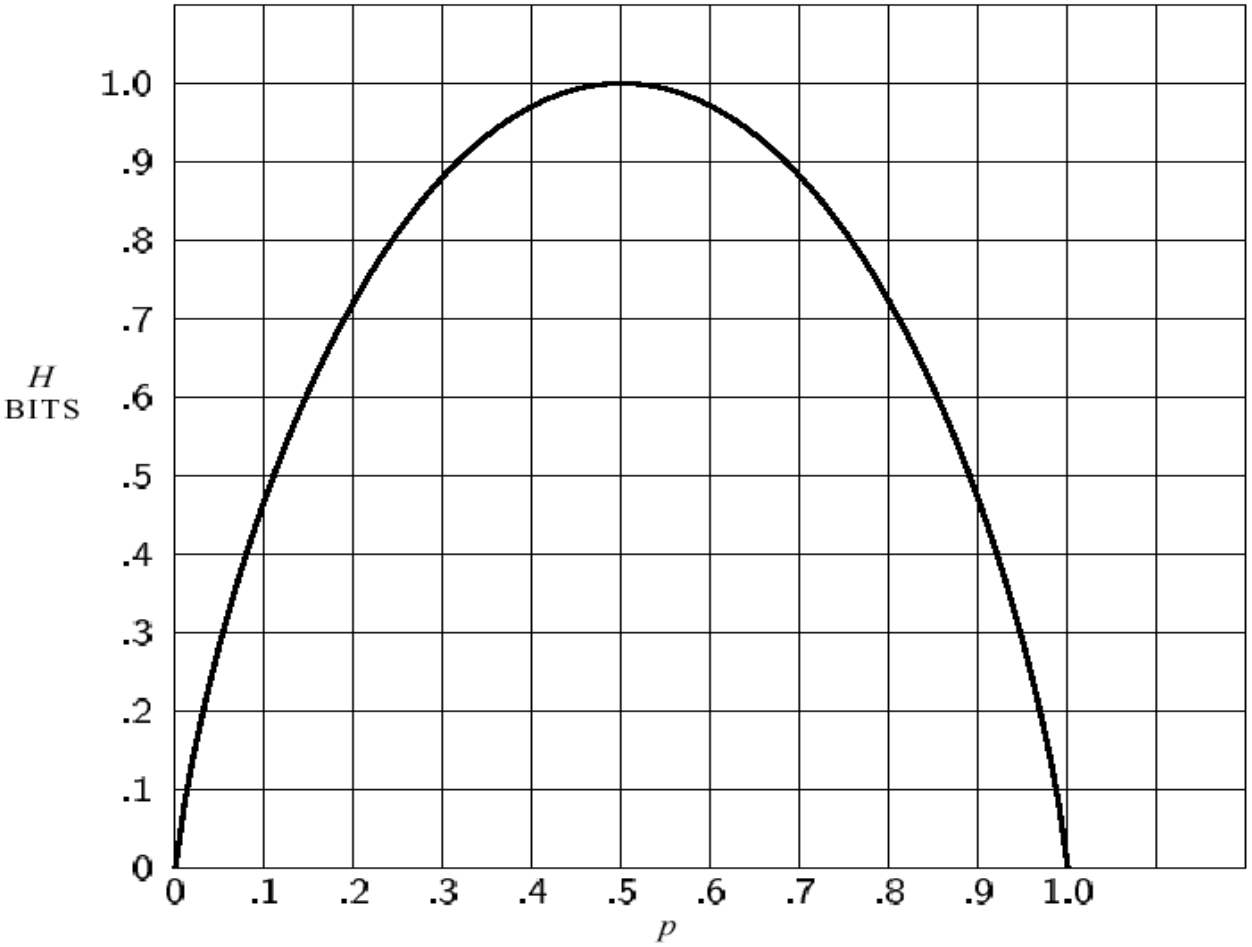


Fig. 7—Entropy in the case of two possibilities with probabilities p and $(1 - p)$.

Compactere opslag met Variabele lengte code woorden

Het leidende idee is:

Hoe frequenter de waarde voorkomt,
hoe korter de bit- code waarmee het wordt gerepresenteerd

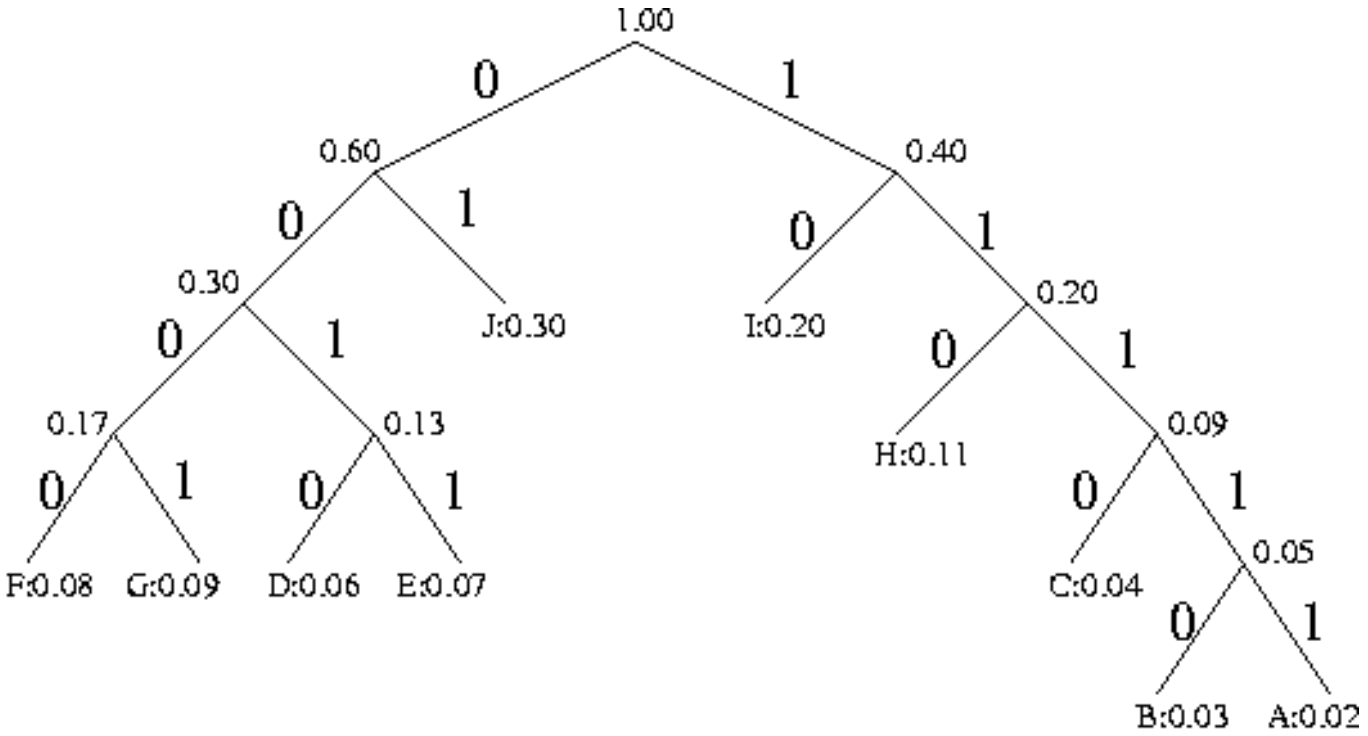
Alle mogelijke bit-string waardes gesorteerd op lengte:

- 0,1,00,01,10,11,000,001,010,011,100,101,110,111,0000,0001,...
- Omdat we niet meer een vast aantal bits/datum gebruiken, hebben we een manier nodig om de code-woorden uit elkaar te houden. Dat kan met een **scheidingsteken** (welke echter ook bits inneemt en waarvan het bitpatroon dan niet als (onderdeel) van een codewoord mag voorkomen
- **Of** we gebruiken een **Uniek Decodeerbare Deelverzameling**:
- Een langere code mag dan niet beginnen met een bitpatroon dat al als kortere code is uitgegeven:
- 0,1,00,01,10,11,000,001,010,011,100,101,110,111,0000,0001,...
- Of:0, -,---,---, 10,---,-----, ----,-----, ----,- ----,-----,110,111,------,-----,…
- 0,1,00,01,10,11,000,001,010,011,100,101,110,111,0000,0001,...
- Of: --,1, --, 01, ---,---,000,001,-----,----,-----,----,----,-----,------,------,…
- Alle gebruikte codes komen in een tabel die bij de gecomprimeerde data wordt opgeslagen

Huffman code

- Een Uniek decodeerbare, variabele woordlengte codering
- De meest efficiente van dergelijke coderingen
- **Algoritme voor codewoord vorming** bij een waarschijnlijkheidsverdeling van kenmerk waarden:
- - *maak een genormaliseerd histogram op (fracties)*
- - *registreer kenmerk-fractie verbindingen (graaf)*
- - ***while** (aantal fracties >1) **do***
- - - *sorteer fracties;*
- - - *combineer 2 laagste: registreer samenneming in graaf;*
- - ***enddo***
- *In samengenomen kenmerk graaf: ken 0 en 1 toe aan splitsingen*
- *Vorm code woorden door vanaf wortel 0/1 splitsingen te volgen tot aan oorspronkelijke fractie*

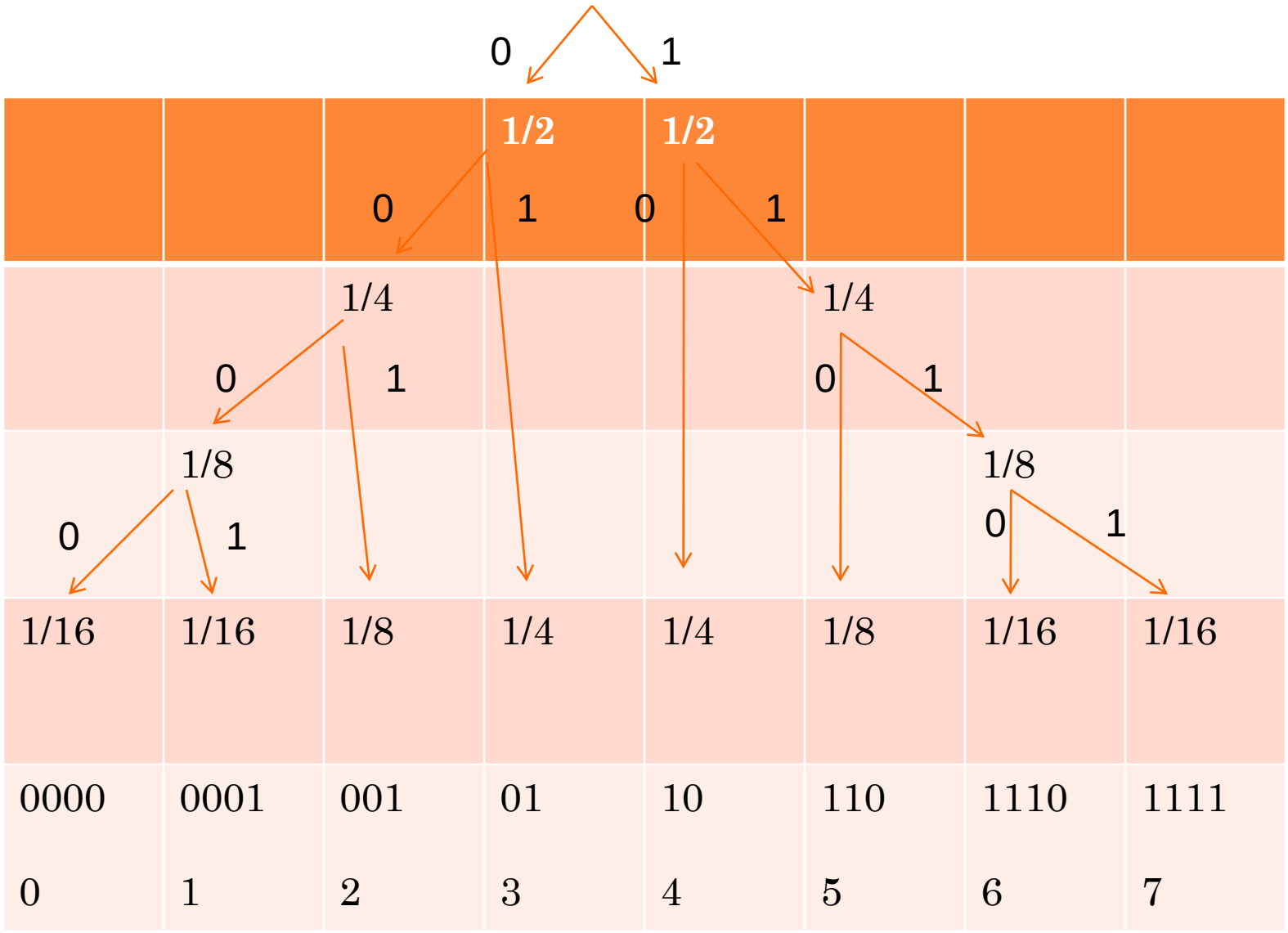
HUFFMAN CODE VOORBEELD



BIT CODE

A: 11111 F: 0000
B: 11110 G: 0001
C: 1110 H: 110
D: 0010 I: 10
E: 0011 J: 01

Huffman code voorbeeld 2



Een uitgewerkt Huffman voorbeeld 2

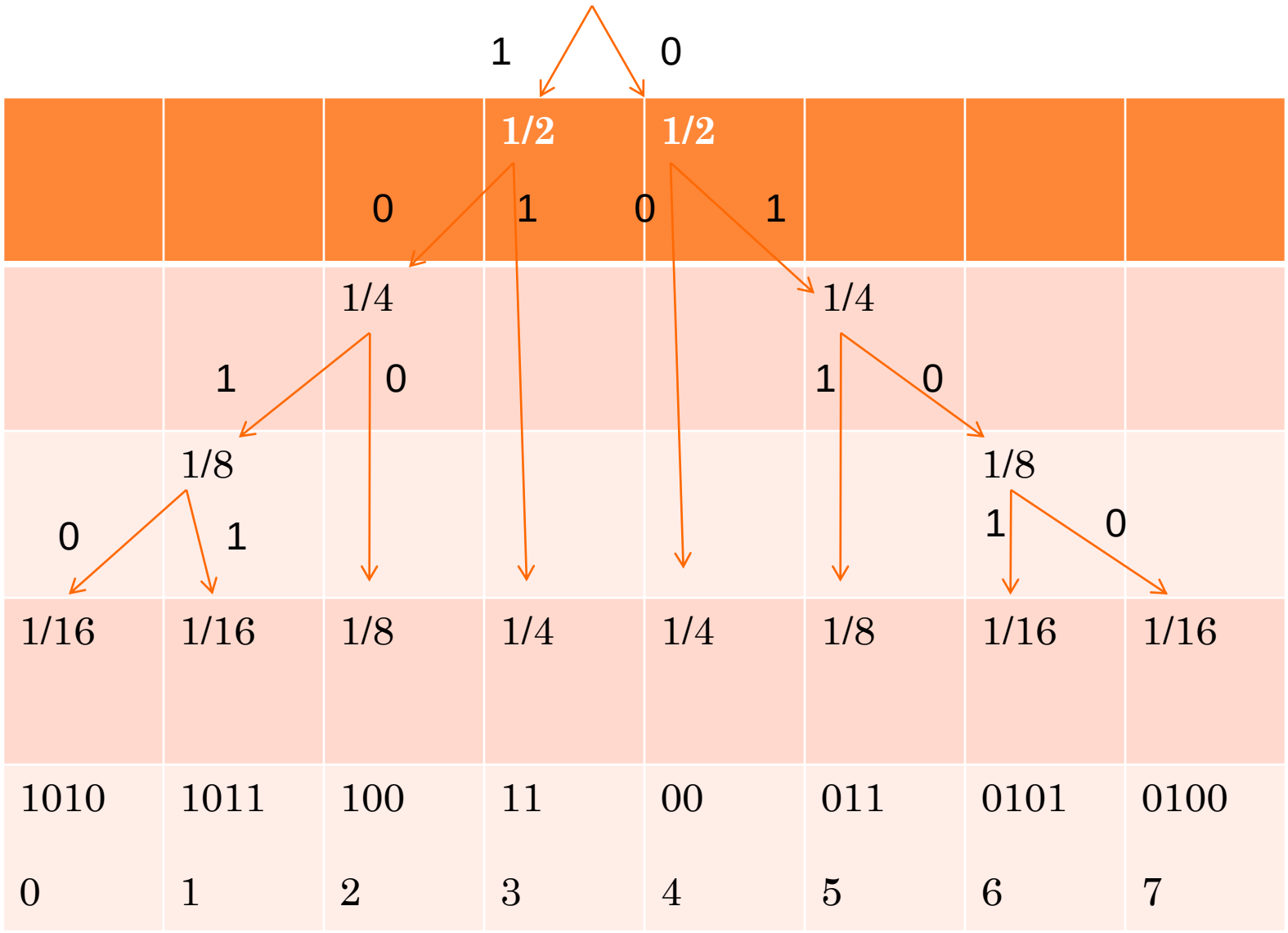
	fractie		Bijdrage Entropie			Bijdrage gem. woordlengte
grijswaarde	p(i)	$-^2\log p(i)$	$-p.^2\log p(i)$	H-code	l-H	l-H*p(i)
0	1/16	4	1/4	0000	4	1/4
1	1/16	4	1/4	0001	4	1/4
2	1/8	3	3/8	001	3	3/8
3	1/4	2	½	01	2	1/2
4	1/4	2	½	10	2	1/2
5	1/8	3	3/8	110	3	3/8
6	1/16	4	1/4	1110	4	1/4
7	1/16	4	1/4	1111	4	1/4
3 bits/pix			2.75 bit			2.75 bit

Entropie (E)

R=Gemiddelde woordlengte

Bij dit voorbeeld is het best denkbare resultaat behaald R=E

Alternatief Huffman code voorbeeld 2



Veel manieren om een Huffman code te vormen

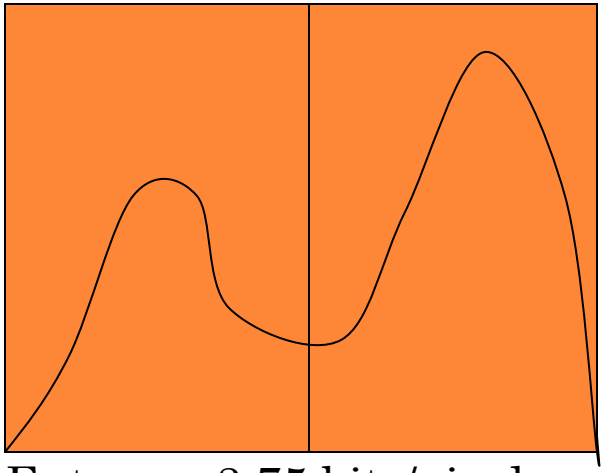
- Omdat bij elke splitsing naar believen een 0 en 1 over de twee takken verdeeld mag worden, zijn er alleen al voor dit voorbeeld honderden manieren om grijswaarden[0..7] te voorzien van Huffman codes.
- Door de manier van toekennen heeft elke langere Huffman code een begin dat niet als kortere code gebruikt is en dus uniek decodeerbaar is (gegeven de tabel van uitgedeelde codes) zonder een bit vooruit te kijken

Voorspellings Model

Resultaat Huffman codering hangt af van welke waarschijnlijkheids verdeling gebruikt wordt;
Als van een grijswaardenbeeld het histogram gebruikt wordt, is dat een pixeleigenschap die onafhankelijk van de buurwaarden is, terwijl er in een beeld meestal veel lokale coherentie aanwezig is
Beter is het de waarschijnlijkheidsverdeling te gebruiken van de lokale afwijkingen van een voorspellingsmodel

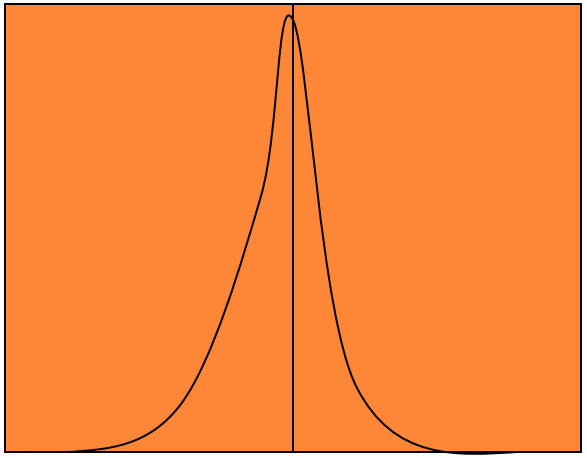
Hoe beter een voorspellingsmodel,
hoe gepiekter de verdeling van afwijkingen
van het voorspellingsmodel -> hoe lager entropie

Voorspelmodel:
Intensiteit= (maxint-minint)/2



Entropy=3.75 bits/pixel

Voorspelmodel:
Intensiteit= Intensiteit linker buur



Entropy=1.35 bits/pixel

Universeel codeer schema LZW

In principe moet voor Huffman codering de waarschijnlijkheidsverdeling van de te coderen waarden vooraf bekend zijn.

We gaan nu kijken naar een codeer schema waarbij de vorming van de code tabel gedreven wordt door de reeks input symbolen

Liv-Zempel-Welsh methode 1984 (verbetering van LZ77 en LZ78):

De codetabel telt 12bits codes (4096 stuks maximaal)
beginnend met alle (256) losse ASCII karakters in de codetabel[0..255]

```
Start met eerste input karakter -> s
while input karakter do
        c=input karakter;
        if (s+c in codetabel)
                s=s+c;
        else
                output codewoord(s);
                voeg s+c toe aan codetabel;
                s=c;
enddo
output codewoord(s);
```

LZW VOORBEELD

ct	a	m	a	n	a	p	l	a	n	a	c	a	n	a	l	p	a	n	a	m	a	out
(97)	a																					
256	**a**	m																				97
257		**m**	a																			109
258			**a**	n																		97
259				**n**	a																	110
260					**a**	p																97
262						**p**	l															112
263							**l**	a														108
264								**a**	**n**	a												258
265										**a**	c											97
266											**c**	a										99
267												**a**	**n**	**a**	l							264
268															**l**	p						108
269																**p**	a					112
270																	**a**	**n**	**a**	m		264
																				m	**a**	257

LZW VOORBEELD PREFIX OUTPUT

Het laatste patroon wordt als volgt behandeld:
- If s+c in codetabel output code
-else output codetabel(s) en codetabel(c)

In ons voorbeeld kwam **ma** in de codetabel voor en geeft **257** als output

De prefix van elke toegevoegde code wordt uitgevoerd:
97,109,97,110,97,112,108,258,97,99,264,108,112,264,257

Bij decodering zou aan de hand van een bijgesloten codetabel[0..270] elke code vertaald als:
a, m ,a, n, a ,p, l, an, a ,c, ana, l, p,ana, ma
het oorsponkelijke bericht weer opleveren

LZW loont natuurlijk pas bij langere files als er een codetabel met de file meegeschreven moet worden
Alleen de codes >255 zouden natuurlijk meegegeven hoeven worden (die van 0..255 zijn altijd gelijk en kunnen in het decodeer programma vast worden opgenomen)

Geheugen gebruik LZW voorbeeld

Input van ons voorbeeld bestond uit 21 karakters -> 21*8bit=168 bits

Codering neemt 15 codes -> 15*12bit=180 bits

Een meegezonden codetabel zou beperkt kunnen worden tot die codes die na vorming ook echt gebruikt zijn:
257, 258, 264 met hun strings ma, an, ana -> 3*12bit + 7*8bit=92 bits

Of van alle 14 codes boven 255 wordt aantal en strings opgeslagen:
1*12 + 35*8=292 bits

MAAR

LZW kan zonder codetabel!

De ontwikkelaars van LZW hebben als volgt gerealiseerd dat voor decompressie zelfs het meezenden van de codetabel boven code 255 niet nodig is, omdat deze tabel vanuit de gecodeerde uitvoer weer kan worden opgebouwd!!

LZW decompressie:

```
Maak het begin van de codetabel aan [0.255] (de losse ASCII tekens b.v.)
read priorcodewoord from input en output daarbij horend karakter
While  input do
   read codewoord;
   if codewoord not in codetabel
      enter in codetabel string(priorcodewoord)+ firstchar(string(priorcodewoord));
   else
      enter in codetabel string(priorcodewoord)+firstchar(string(codewoord));
      output string(codewoord);
   priorcodewoord=codewoord;
enddo
```

Universeel

LZW is universeel voor byte-geadresseerde geheugens:
- tekst ASCII latin1 of andere karaktercodering
- type byte pixels in beeldmatrices (ord van ASCII teken is grijswaarde)
- Data Types > 1 byte: na opdeling in bytes volstaat complete byte omzetting

Beperkingen LZW

De Codetabel die ook wel woordenboek (dictionary) genoemd wordt is beperkt tot 4096-256 extra strings

Er zijn varianten waarbij in de gecomprimeerde file wordt aangegeven dat er opnieuw met de opbouw van een dictionary begonnen gaat worden.

De winst van LZW is in het begin klein, maar naarmate de file langer is nadert LZW asymptotisch de maximaal mogelijke compressie.

Tekst files worden gemiddeld tot de helft ingedikt.

Voor beelden/videos bestaan lossy compressie methoden zoals JPEG en MPEG die tot veel grotere reducties leiden, maar exacte reconstructie van het oorspronkelijke beeld/video is daarmee niet meer mogelijk.

Toetje
Perfect Hash

Als afsluiting laat ik slides zien van een heel geslaagde ruimtelijke (2D en 3D) hash opzet zien zodat duidelijk wordt dat hashen niet beperkt is tot 1D lijsten

De presentatie PerfectHash is te vinden op:
Research.microsoft.com/en-us/um/people/hoppe/perfecthash.pdf

Voor het volgende (laatste) college krijg je een voorbeeld tentamen mee; uitwerking wordt volgende week gegeven