Data Compression Algorithms

Introduction

Marcus Hutter (*1967)
DeepMind

Data compression

The process of converting an input data stream (the source stream, the original raw data) into output data stream (the compressed stream, the bitstream) that has a smaller size.

Compression algorithm = *encoding* (compression)
+ *decoding* (decompression)

Compression

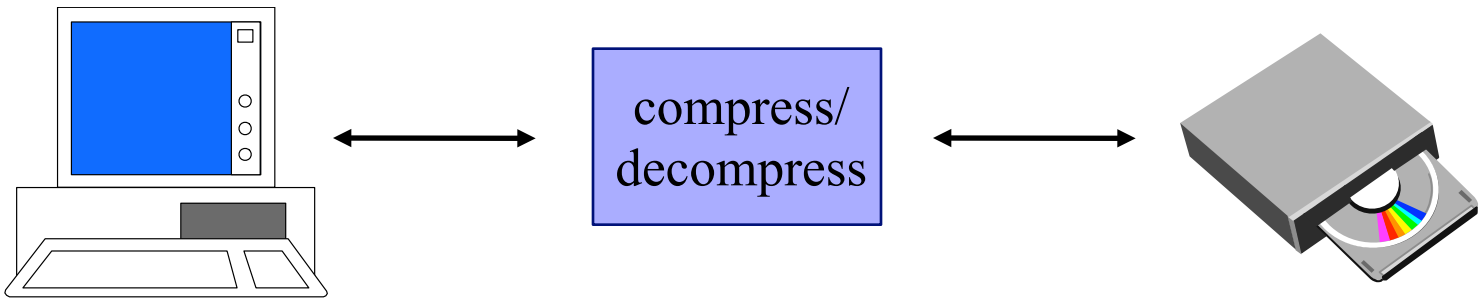
- *lossless*: the restored and original data are identical
- *lossy*: the restored data are a „reasonable“ approximation of the original

Methods

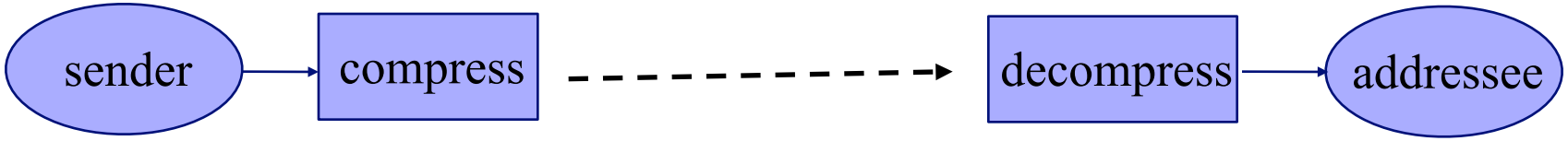
- *static* / *adaptive*
- *streaming* / *block*

Goals of data compression

✓ save storage



✓ reduce the transmission bandwidth



Measuring the performance

input data size u B

compressed data size k B (K bits)

measure	formula	example
compression ratio	$k / u * 100\%$	36 %
compression factor	$u : k$	3 : 1
compression gain	$(u-k)/u*100\%$	64 %
bpc (bits per char) bpp (bits per pixel) *average codeword length*	K / u	1.47 b/c
relative compression (*percent log ratio*)	$100 \ln (k / k')$	10.5

size of data compressed by a standard algorithm

Performance: data corpora

Comparing lossless compression algorithms

Calgary Corpus (1987)

- 14 files: text, graphics, binary files

Canterbury Corpus (1997)
http://corpus.canterbury.ac.nz

- 11 files + artifical c. (4) + large c. (3) + miscellaneous c. (1)

Silesia Corpus (2003)
Sebastian Deorowicz, Politechnika Śląska, Gliwice
http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia

- 18 files of sizes 6 – 51MB

Prague Corpus (2011)
Jan Holub et al., FIT ČVUT, Praha
http://www.stringology.org/projects/PragueCorpus/

- 30 files, 58 MB total

Canterbury corpus

soubor	kategorie	velikost (B)
alice29.txtd	English text	152089
asyoulik.txt	Shakespeare	125179
cp.html	HTML source	24603
fields.c	C source	11150
grammar.lsp	LISP source	3721
kennedy.xls	Excel Spreadsheet	1029744
lcet10.txt	Technical writing	426754
plrabn12.txt	Poetry	481861
ptt5	CCITT test set	513216
sum	SPARC Executable	38240
xargs.1	GNU manual page	4227

Canterbury corpus

The Artificial Corpus

a.txt	The letter 'a'	1
aaa.txt	The letter 'a', repeated 100,000 times.	100000
alphabet.txt	Enough repetitions of the alphabet to fill 100,000 characters	100000
random.txt	100,000 characters, randomly selected from [a-z|A-Z|0-9|!|] (alphabet size 64)	100000

Canterbury corpus

The Large Corpus

E.coli	Complete genome of the E. Coli bacterium	4638690
bible.txt	The King James version of the bible	4047392
world192.txt	The CIA world fact book	2473400

The Miscellaneous Corpus

pi.txt	The first million digits of pi	1000000

Data compression contests

Calgary Corpus Compression Challenge (1996)

- http://mailcom.com/challenge/
- (777,777.00 - X) / 333 for an archive of length X B that compresses the 14 file version of the Calgary corpus
- now (580,170.00 - X) / 111 $

Data compression contests

Calgary Corpus Compression Challenge (1996)

Size	Date	Name
759881	09/1997	Malcolm Taylor
692154	08/2001	Maxim Smirnov
680558	09/2001	Maxim Smirnov
653720	11/2002	Serge Voskoboynikov
645667	01/2004	Matt Mahoney
637116	04/2004	Alexander Rhatushnyak
608980	12/2004	Alexander Rhatushnyak
603416	04/2005	Przemysław Skibiński
596314	10/2005	Alexander Rhatushnyak
593620	12/2005	Alexander Rhatushnyak
589863	05/2006	Alexander Rhatushnyak
580170	07/2010	Alexander Rhatushnyak

Data compression contests II

Hutter Prize (2006) http://prize.hutter1.net

- create a self-extracting archive of the 100 MB prefix of English Wikipedia
- 500 € for each 1% improvement of the archive size

author	date	dec	size	comp. factor	RAM	time
Matt Mahoney	24.3.2006	paq8f	18'324'887	5.46	854MB	5h
Alexander Ratushnyak	25.7.2006	paq8hp5	17'073'018	5.86	900MB	5h
Alexander Ratushnyak	14.5.2007	paq8hp12	16'481'655	6.07	936MB	9h
Alexander Ratushnyak	23.5.2009	decmprs8	15'949'688	6.27	936MB	9h
Alexander Ratushnyak	4.11.2017	phda9	15284944	6.54	1048MB	5h

Data compression contests II

Hutter Prize : update

- create a self-extracting archive of the 1 GB prefix of English Wikipedia
- run in ≲50 hours, a single CPU core and <10GB RAM and <100GB HDD

author	date	dec	size	comp. factor	RAM	time
Alexander Ratushnyak	4.7.2019	phda9v1.8	116'673'681	8.58	6.3GB	23h
Artemiy Margaritov	31.5.2021	starlit	115'352'938	8.67	10GB	50h
Saurabh Kumar	16.7.2023	fast cmix	114'156'155	8.76	8.4GB	43h

Hutter prize

Goal: encourage research in AI

Marcus Hutter: being able to compress well is closely related to acting intelligently

- M. Hutter, *Towards a Universal Theory of Artificial Intelligence based on Algorithmic Probability and Sequential Decisions,* Proceedings of the 12th European Conference on Machine Learning, 226-238, 2000
- the optimal behaviour of a goal-seeking agent in an unknown but computable environment
- guess at each step that the environment is controlled by a shortest program consistent with all interaction so far

Kolmogorov complexity (algorithmic information theory)

Compressing natural language text – Turing imitation game alternative

Limits of lossless compression

Encoding f: $\{n\text{-bit strings}\} \rightarrow \{\text{strings of length} < n\}$

$|\operatorname{Dom} f| = 2^n$

$|\operatorname{Im} f| \leq 2^n - 1$

$\Rightarrow$ f cannot be injective!

Let $M \subseteq \operatorname{Dom} f$ such that $\forall s \in M, |f(s)| \leq 0.9n$

f injective on $M \Rightarrow |M| \leq 2^{1+0.9n} - 1$

$n = 100, |M|/2^n < 2^{-9}$

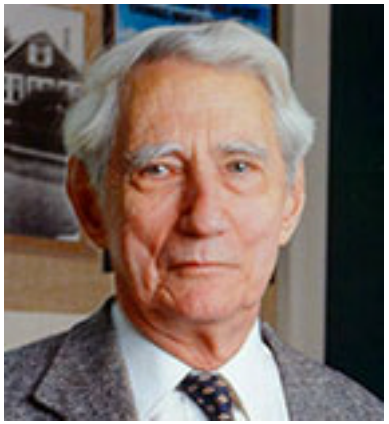
$n = 1000, |M|/2^n < 2^{-99} \approx 1.578 \cdot 10^{-30}$

Data Compression Algorithms

Statistical Methods

&

Shannon-Fano coding



Claude Shannon
(1916 – 2001)

Robert Fano
(1917– 2016)

Basic concepts

Alphabet is a finite set of symbols

String (*message*) over alphabet A

- is a finite sequence of letters of A
- the length of this sequence is called the *length of the string* s and denoted by $|s|$

The set of all strings over A is denoted by A^*.

Basic concepts

We are given

- a source alphabet A
- a coding alphabet A_C

Encoding is a function $f : A^* \to A_C^*$

If function f is an injection, the encoding is called *uniquely decodable*.

Put $A_C = \{0,1\}$

Phrases

A typical strategy

- input string $s \in A^*$ is factorized
- into (concatenation of) substrings $s = s_1\, s_2 \ldots\, s_k$
- strings s_i - *phrases*
- determine $C(s_1), \ldots, C(s_k) \in A_C^*$
- $C(s_i)$ - *codewords*
- $f(s) = C(s_1)\, C(s_2)\, \ldots\, C(s_k)$

We first study encodings with phrase length fixed to one

Codes

(Source) code is a function $K : A \to A_c^*$

$K(s)$ is a *codeword* for symbol $s \in A$

Encoding K^* *generated by a code* K is the mapping

$$K^*(s_1 s_2 \ldots s_n) = K(s_1)K(s_2) \ldots K(s_n)$$

$$\text{for every } s_1 s_2 \ldots s_n \in A^*$$

Code K is called *uniquely decodable* if it generates a uniquely decodable encoding K^*

Problem

<u>Input</u>: A set of codewords for source alphabet A

<u>Question</u>: Does it define a uniquely decodable code?

Example

- {0,01,11}
- {0,01,10}

Example

$A = \{a,b,c,d,e,f\}$

Code K_1

source alphabet	a	b	c	d	e	f
codeword	000	001	010	011	100	101

- fixed length codewords
- $K_1(abae) = 000001000100$

Code K_2

source alphabet	a	b	c	d	e	f
codeword	0	101	100	111	1101	1100

- variable length codewords
- $K_2(abae) = 010101101$
- Is K_2 uniquely decodable?

Prefix codes

String s' is a *prefix* of a string $s = a_1 a_2 \ldots a_n$ if $s' = a_1 a_2 \ldots a_k$ for some k ($0 \leq k \leq n$).

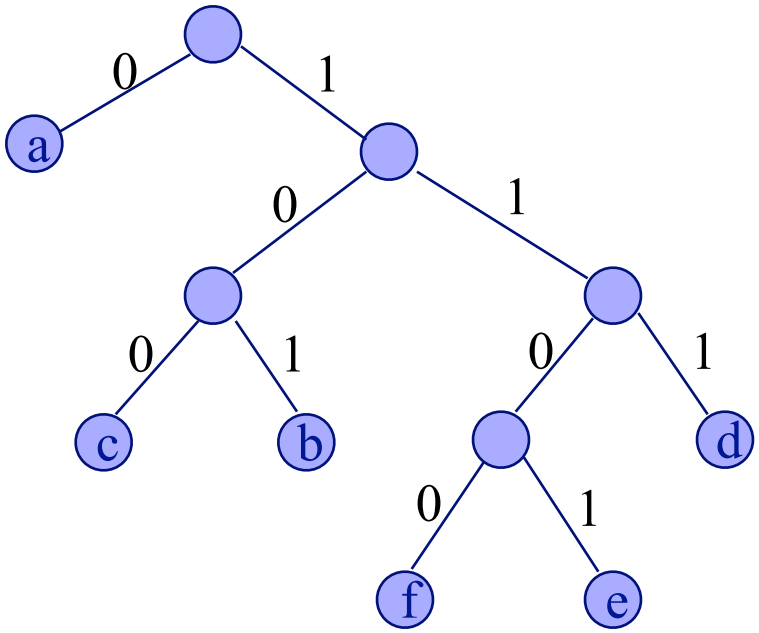
Prefix code is a code such that no codeword is a prefix of another codeword

Observation

- Every prefix code K (over a binary alphabet) may be represented by a binary tree = *prefix tree for K*
- Prefix tree may be used for decoding

Prefix tree

a	b	c	d	e	f
0	101	100	111	1101	1100



Statistical methods

Idea: some letters occur in the message to be encoded more frequently than the others

symbol	a	b	c	d	e
frequency	15	7	6	6	5
code K_1	000	001	010	011	100
code K_2	0	101	100	111	1101

Historical example: Morse code

Shannon-Fano coding

Claude Shannon, R. M. Fano (1949)

<u>Input:</u>

- source alphabet A
- frequency $f(s)$ for every symbol $s \in A$

<u>Output</u>: Prefix code for A

Shannon-Fano algorithm

(1) Sort the source alphabet by the symbol frequencies

(2) Divide the resulting sequence into two parts so that the sums of symbol frequencies in both parts are as close as possible to being equal

(3) Codewords of symbols from the first (second) part start with 0 (1)

(4) Apply steps 2 and 3 recursively to both parts.

(5) The process of division continues until each part reduces to a single symbol

Example

a	15
b	7
c	6
d	6
e	5

Example

a	15
b	7
c	6
d	6
e	5

Example

a	15	0
b	7	0
c	6	1
d	6	1
e	5	1

Example

a	15	0	0
b	7	0	1
c	6	1	
d	6	1	
e	5	1	

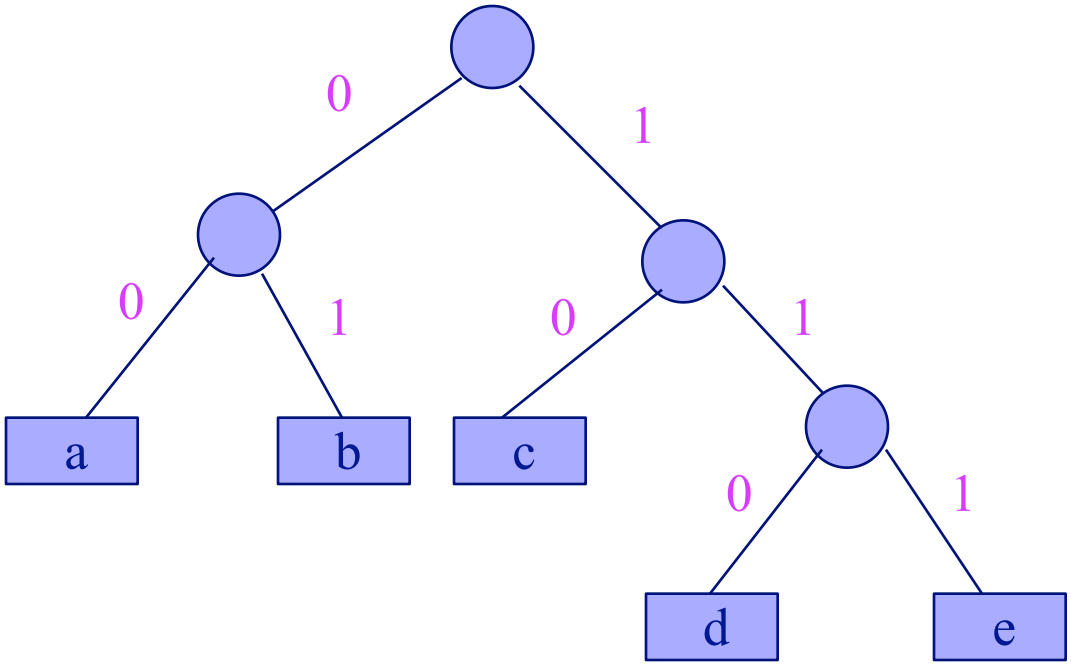
Example

a	15	0	0
b	7	0	1
c	6	1	0
d	6	1	1
e	5	1	1

Example

a	15	0	0	
b	7	0	1	
c	6	1	0	
d	6	1	1	0
e	5	1	1	1

Shannon-Fano code



Optimal codes – definition

Consider an algorithm which – given a source alphabet A with symbol frequencies $f(s)$ – constructs a prefix (uniquely decodable) code K for A.

We say that the algorithm constructs an *optimal* prefix (uniquely decodable) *code* K if

$$|K'^*(a_1a_2\ldots a_n)| \geq |K^*(a_1a_2\ldots a_n)|$$

- for every prefix (uniquely decodable) code K'
- and for every string $a_1a_2\ldots a_n \in A^*$ with symbol frequencies f

Optimal code

- Shannon-Fano algorithm **need not** always construct an optimal code

symbol	a	b	c	d	e
frequency	35	17	17	16	15

- A construction of optimal prefix code was described by **David Huffman** (1951)

Huffman coding

David A. Huffman
(1925 – 1999)

Optimal code

- Shannon-Fano algorithm does not always produce optimal prefix codes

symbol	a	b	c	d	e
frequency	35	17	17	16	15

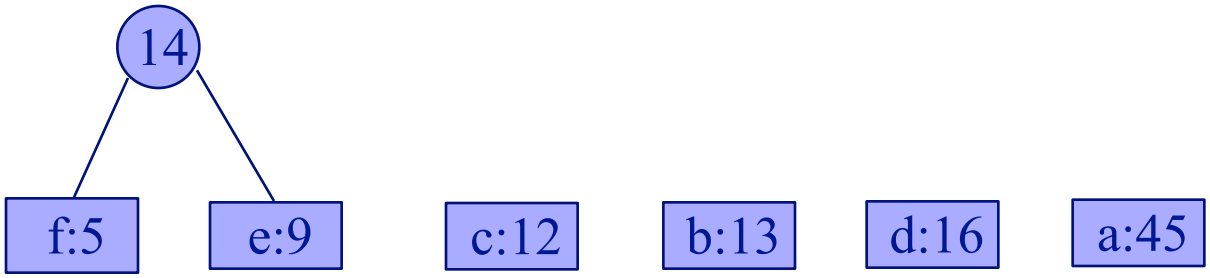
- A construction of optimal prefix code was described by **David Huffman** (1951)

Example

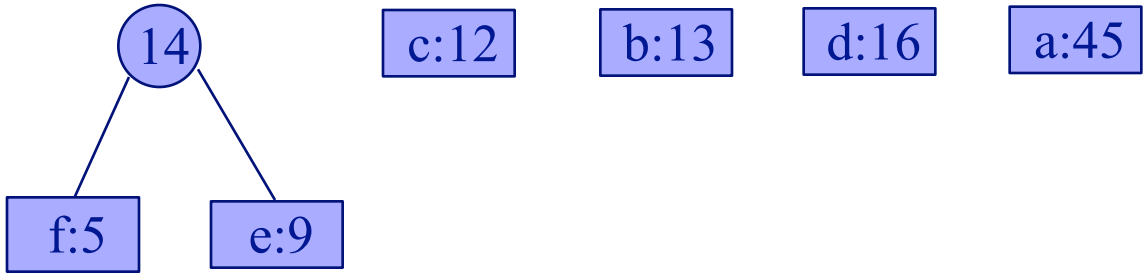
symbol	a	b	c	d	e	f
frequency	45	13	12	16	9	5

f:5 e:9 c:12 b:13 d:16 a:45

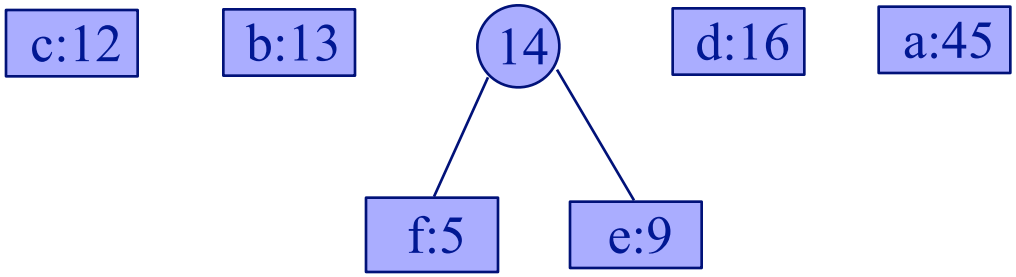
Example



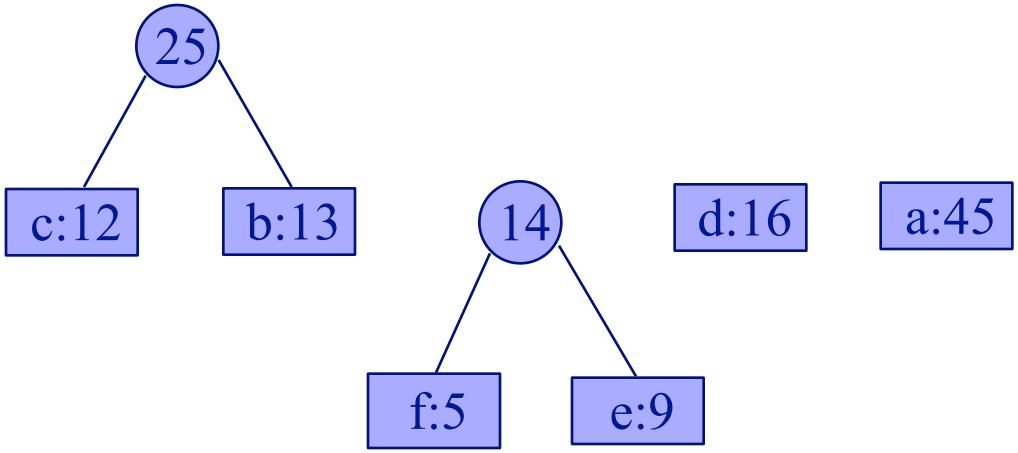
Example



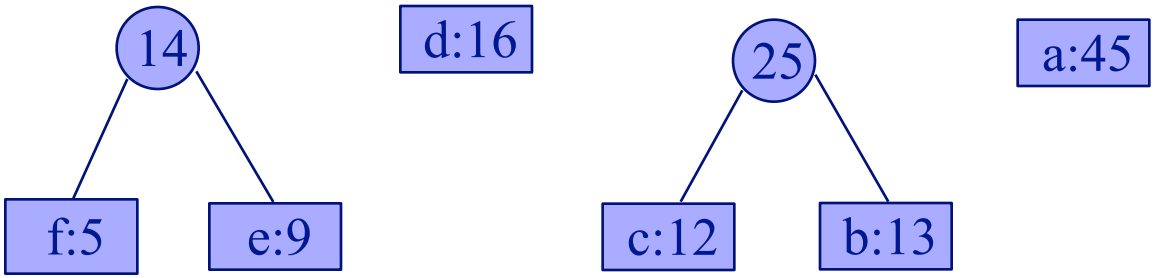
Example



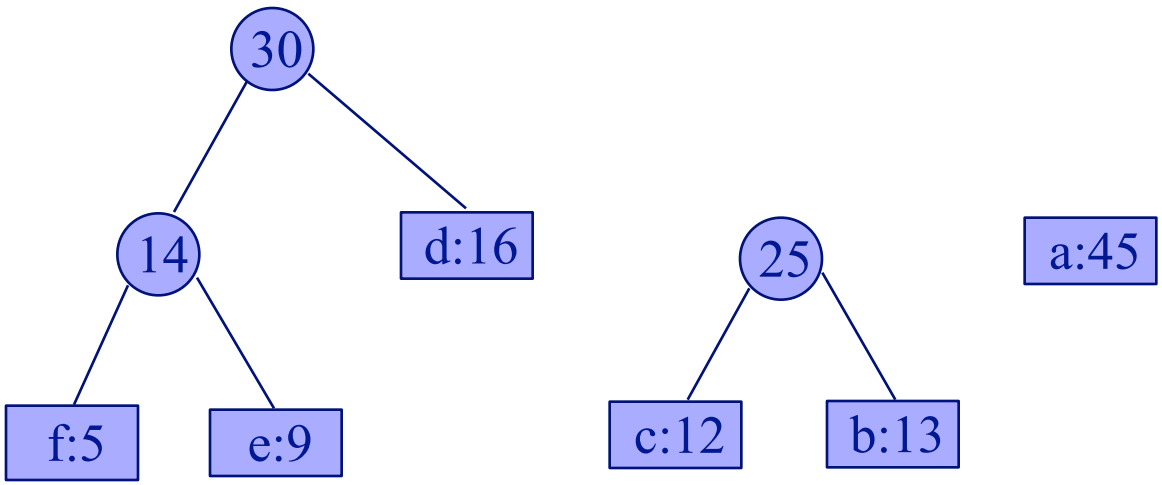
Example



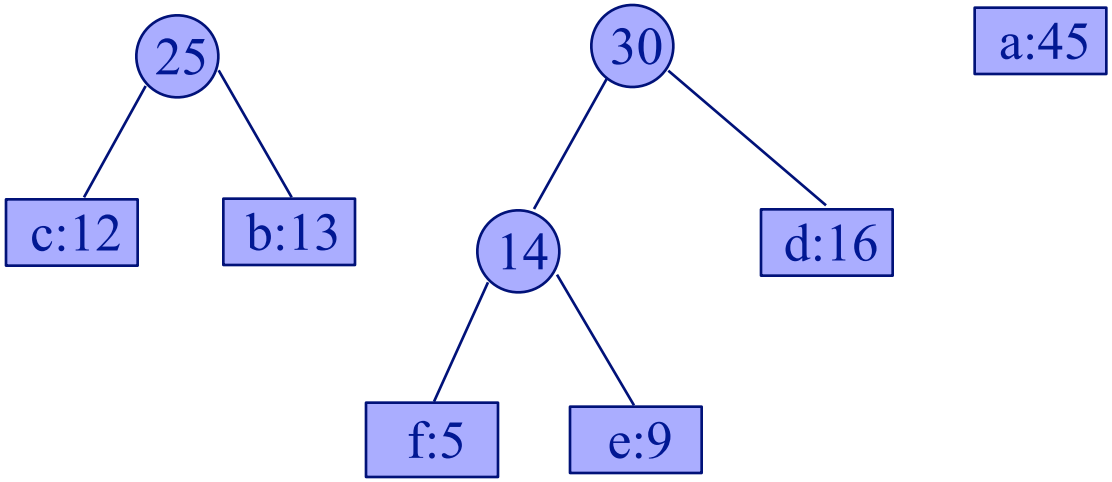
Example



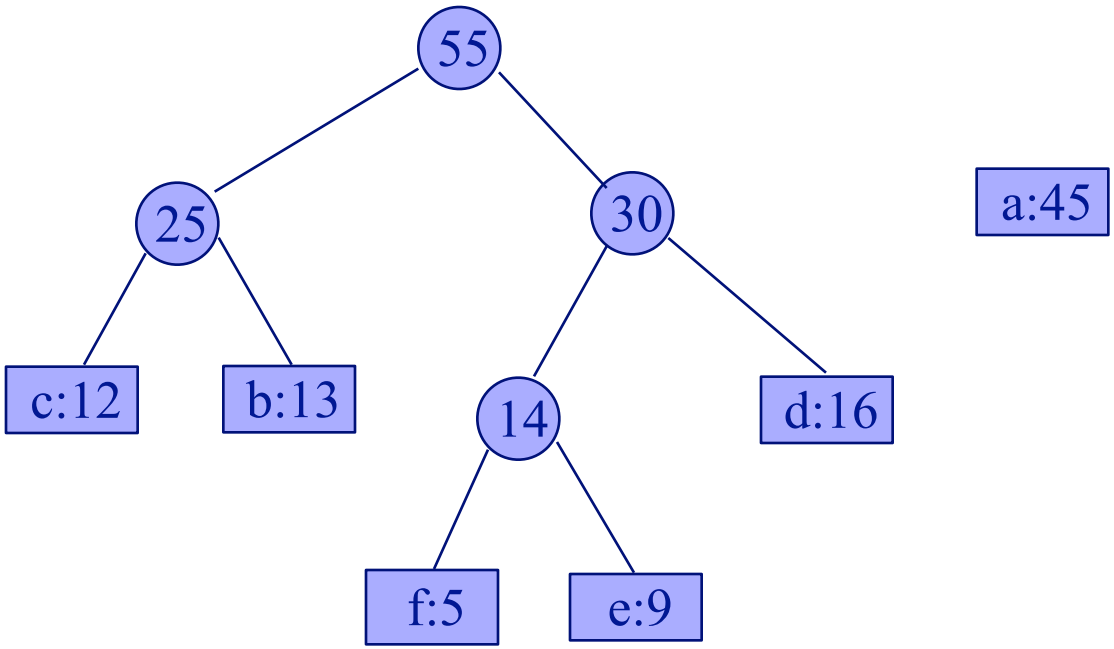
Example



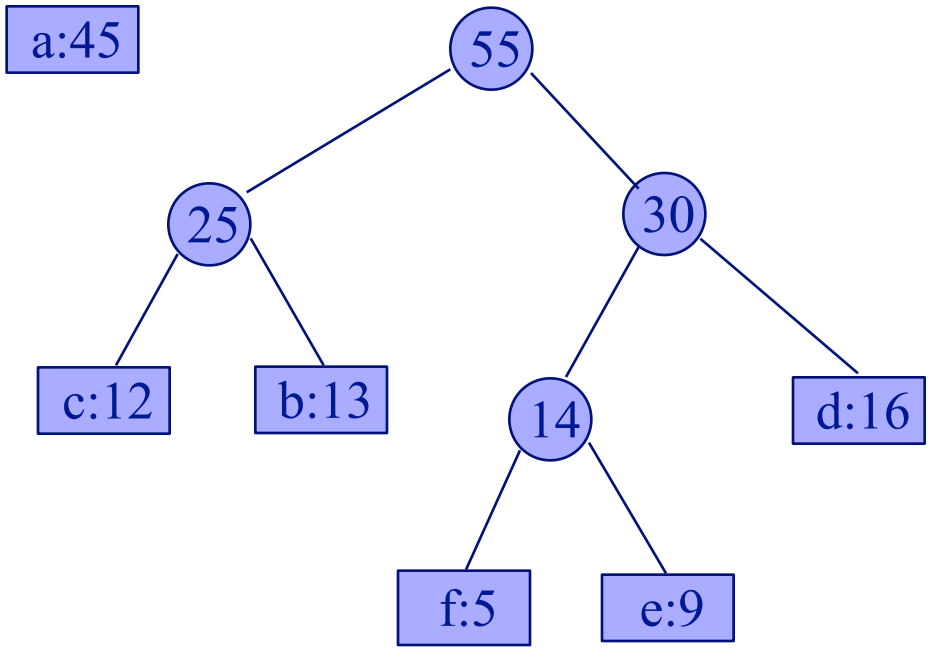
Example



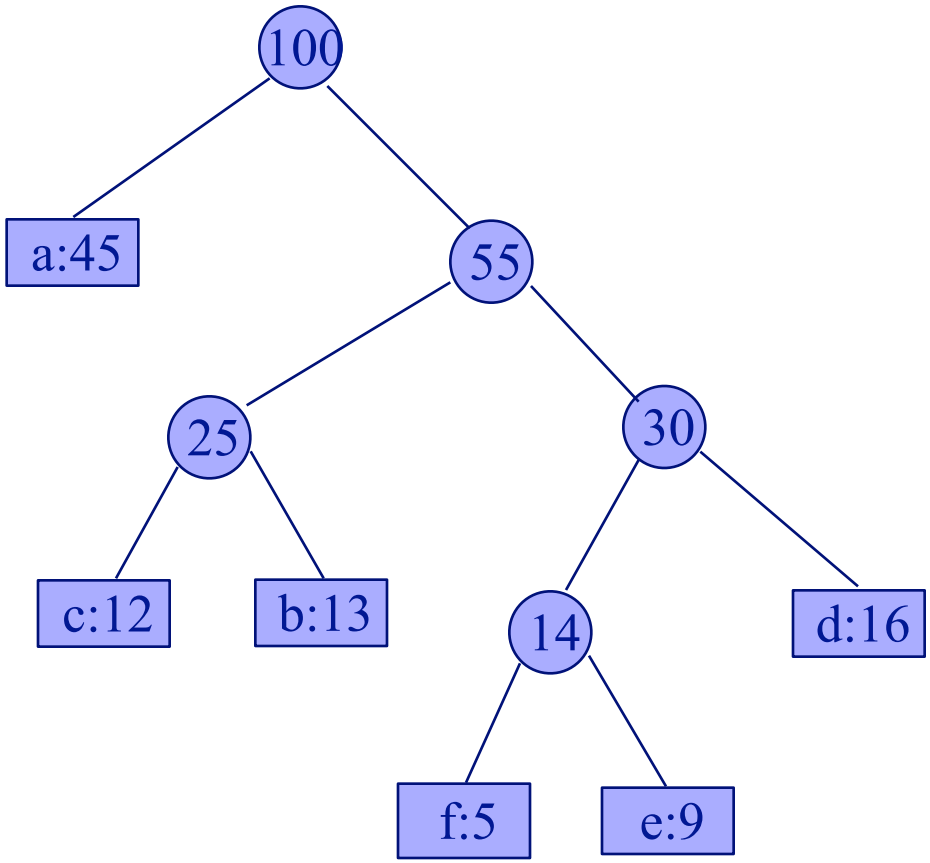
Example



Example



Example



Huffman coding: Algorithm

Input:

- alphabet A of symbols that occur in the input string
- frequency $f(s)$ ($f(s) \neq 0$) for each symbol $s \in A$

Output: prefix tree for A

Huffman coding: Algorithm

```
def Huffman(A,f):
 while |A| ≥ 2
        create new vertex v
        x = left-child(v) = ExtractMin(A)
        y = right-child(v) = ExtractMin(A)
        f(v) = f(x)+f(y)
        Insert(A,v)
 return ExtractMin(A).
```

Optimality of Huffman code: Notation

<u>We are given</u>

- alphabet A of symbols
- that occur in the input string
- frequency $f(s)>0$ for every symbol $s \in A$

Optimality of Huffman code: Notation

Let T be a prefix tree for alphabet A with frequencies f

- leaves of $T \leftrightarrow$ symbols of A

Let $d_T(s)$ denote the *depth of a leaf s* in the tree T

- $d_T(s)$ = length of the path from the root to s
- $d_T(s)$ = length of the codeword for symbol s

Cost $C(T)$ of tree T

- $C(T) = \sum_{z \in A} f(z) d_T(z)$
- $C(T)$ = length of the encoded message

Optimality of Huffman code: Proof

Lemma 0

If a prefix tree T ($|V(T)| \geq 3$) contains a vertex with exactly one child, then T is not optimal.

Lemma 1 Let

- A ($|A| \geq 2$) be an alphabet with frequencies $f(s)$ for every symbol $s \in A$
- $x,y \in A$ be symbols with the lowest frequencies.

Then there exists an optimal prefix tree for A in which x,y are siblings of maximum depth.

Optimality of Huffman code: Proof

Lemma 2 Let

- A be an alphabet with frequencies $f(s)$ for every symbol $s \in A$,
- $x,y \in A$ be symbols with the lowest frequencies,
- T' be an optimal prefix tree for alphabet $A'=A\setminus\{x,y\}\cup\{z\}$ where $f(z)=f(x)+f(y)$.

Then tree T obtained from T' by adding children x and y to vertex z is an optimal prefix tree for A.

Theorem Algorithm Huffman produces an optimal prefix code.

Optimality of Huffman code

☝ **Theorem (Kraft-McMillan inequality)**

Codeword lengths $l_1, l_2, \ldots, l_n$ of a uniquely decodable code satisfy

$$\sum_i 2^{-l_i} \leq 1.$$

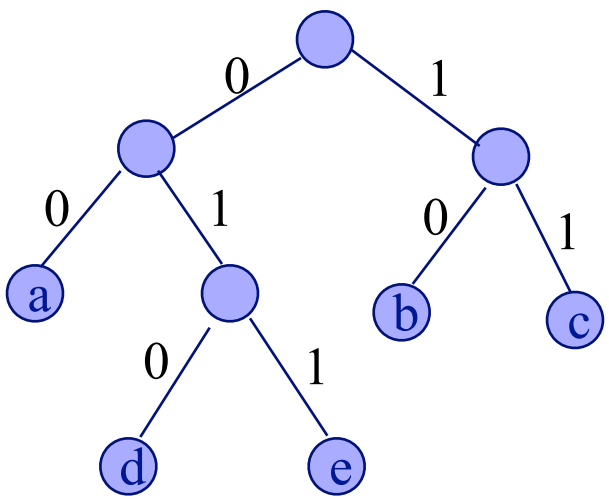
Conversely, if positive integers $l_1, l_2, \ldots, l_n$ satisfy the above inequality, then there is a prefix code with codeword lengths $l_1, l_2, \ldots, l_n$.

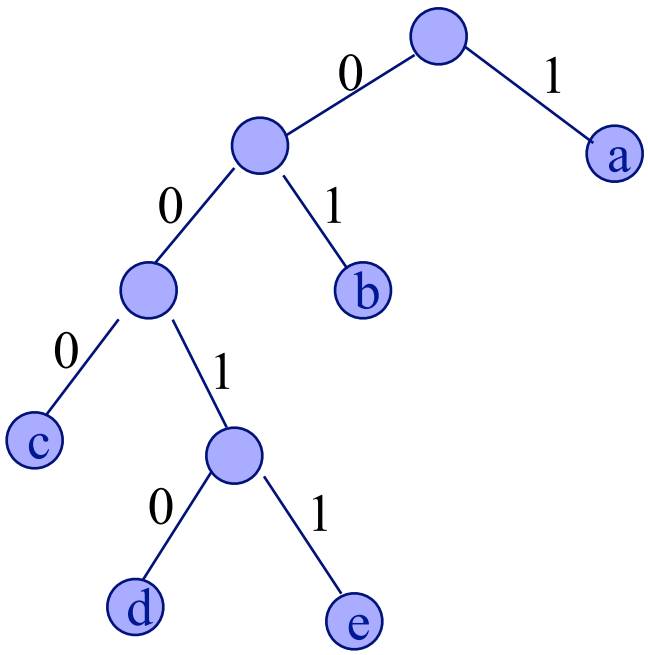
☞ **Corollary**

Huffman coding algorithm produces an optimal uniquely decodable code.

Two non-isomorphic Huffman trees for the same input

symbol	a	b	c	d	e
frequency	40	20	20	10	10





✎ Problems

① Can we modify Huffman algorithm to guarantee that the resulting code minimizes the maximum codeword length?

② What is the maximum height of a Huffman tree for an input message of length n?

③ Is there an optimal prefix code which cannot be obtained using Huffman algorithm?

④ Generalize the construction of binary Huffman code to the case of m-ary coding alphabet ($m > 2$)

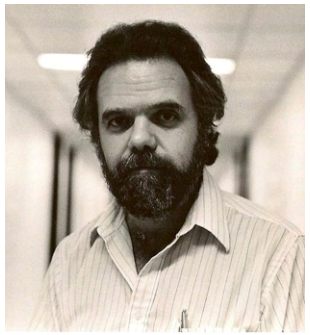
- $f(a_1)= f(a_3)= f(a_4)=20, f(a_5)=25, f(a_6)=10, f(a_2)=5$
- $m = 3$

Implementation notes

- ✓ bitwise input and output
- ✓ overflow problem
- ✓ code reconstruction necessary for decoding

Data Compression Algorithms

Adaptive Huffman coding



Newton Faller

Robert G. Gallager

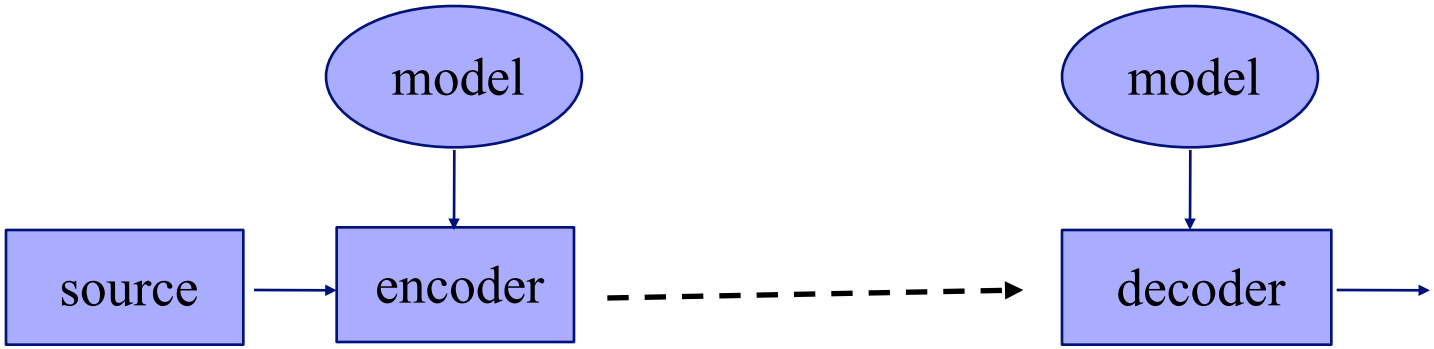
Donald Ervin Knuth

Static × adaptive methods

(Statistical) data compression

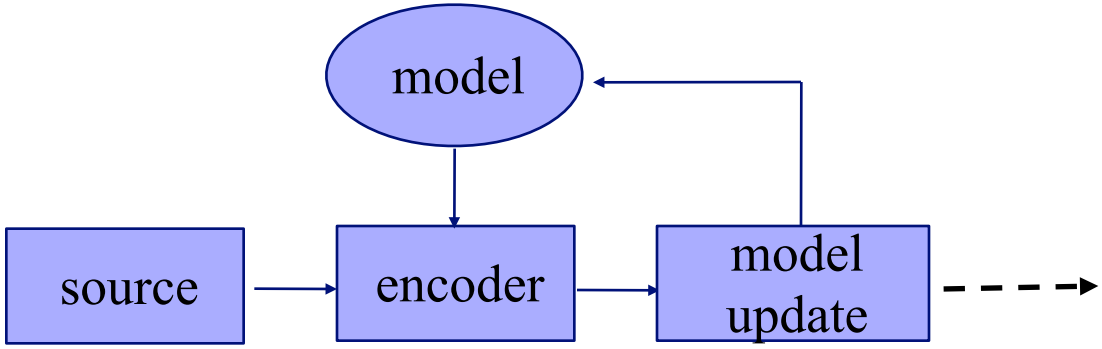
- modeling
- coding

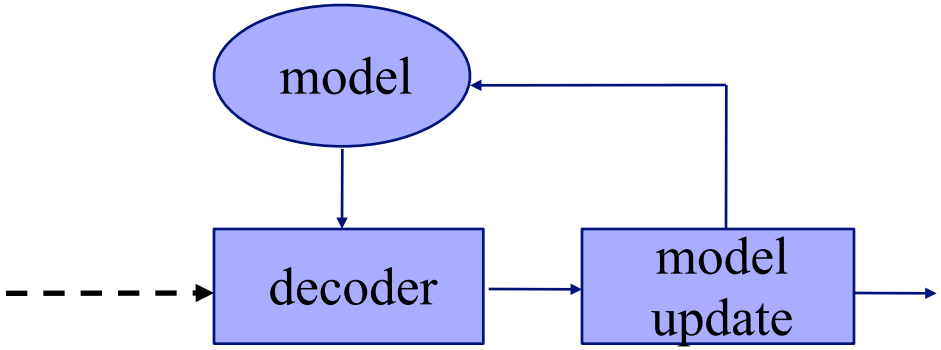
Static model



Static × adaptive methods

Adaptive model





Adaptive Huffman code

```
Algorithm AdaptiveHuffman: Encoding
Inicialize Huffman tree
while not EOF : read symbol
                encode symbol
                update tree
Algorithm AdaptiveHuffman: Decoding
Inicialize Huffman tree
while not EOF : decode symbol
                write symbol
                update tree
```

Adaptive Huffman code – brute force

Huffman tree reconstruction

- after each frequency change
- after reading next k symbols
- after change of order of symbols sorted by frequencies

Characterization of Huffman trees

Huffman tree – binary tree with nonnegative vertex weights

Two vertices of a binary tree with the same parent are called *siblings*

Binary tree with nonnegative vertex weights has a *sibling property* if

- $\forall$ *parent*: weight(*parent*) = Σ weight(*child*)
- each vertex except root has a sibling
- vertices can be listed in order of non-increasing weight with each vertex adjacent to its sibling

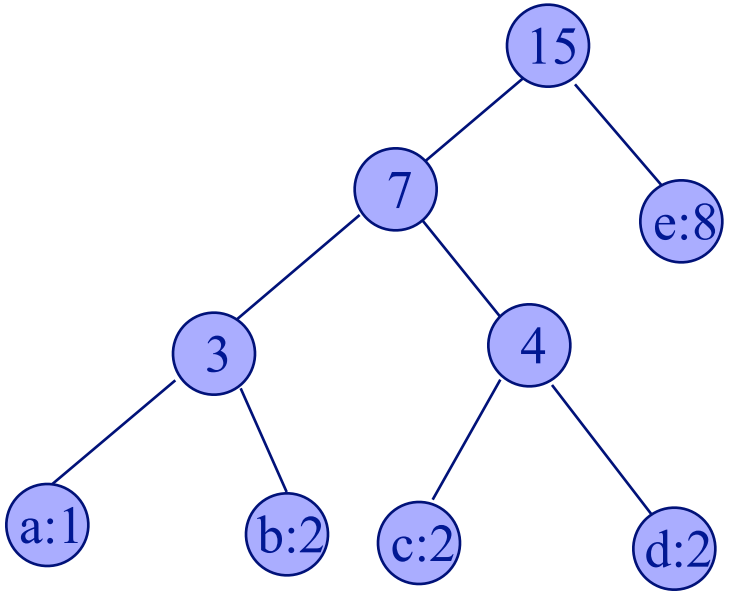
Characterization of Huffman trees

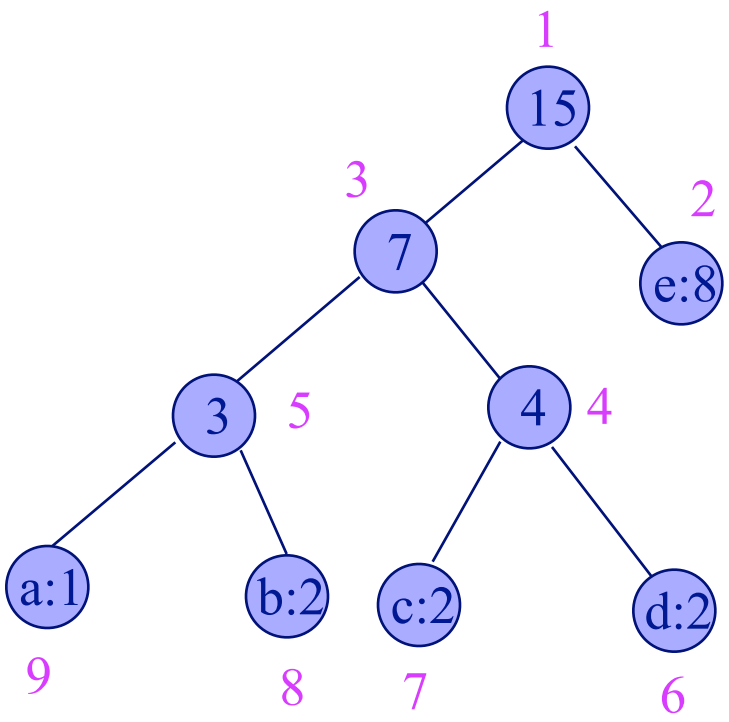
☝ **Theorem** (Faller 1973)

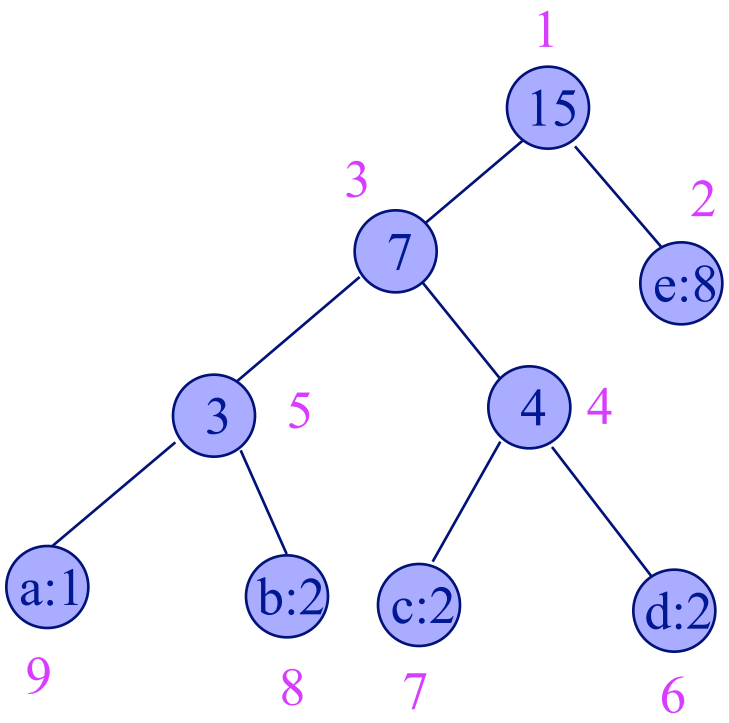
Binary tree with nonnegative vertex weights is a Huffman tree iff it has the sibling property.

☞ FGK algorithm

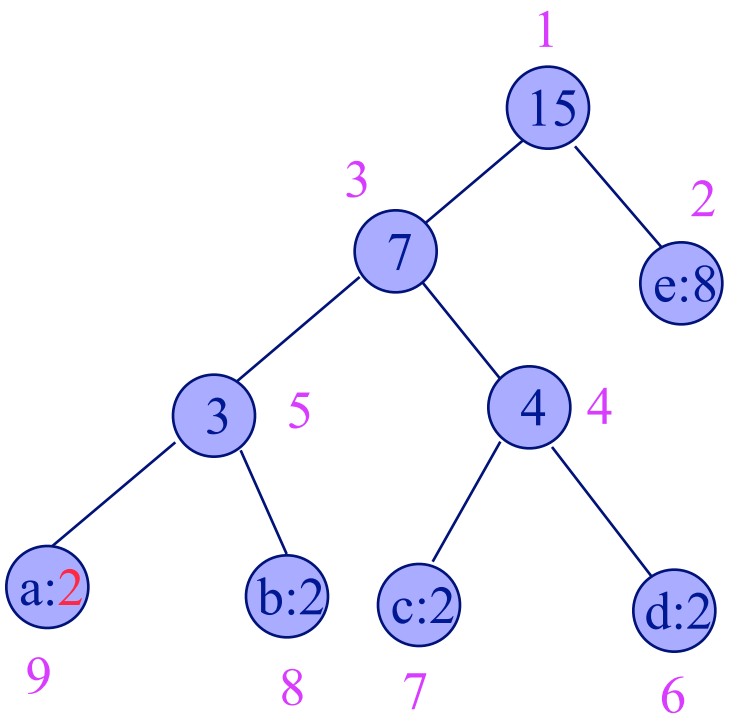
- Faller(1973), Gallagher(1978), Knuth(1985)

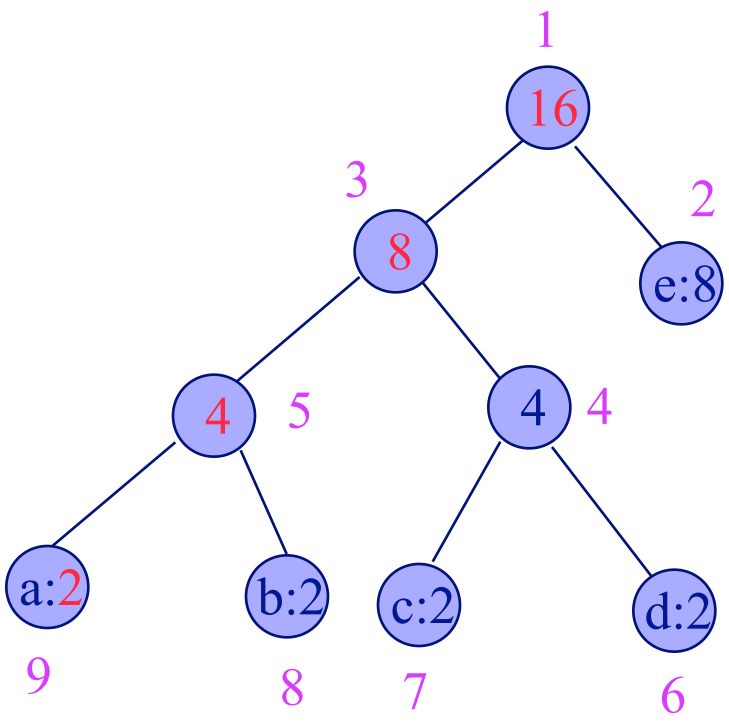

15
7
e:8
3
4
a:1
b:2
c:2
d:2

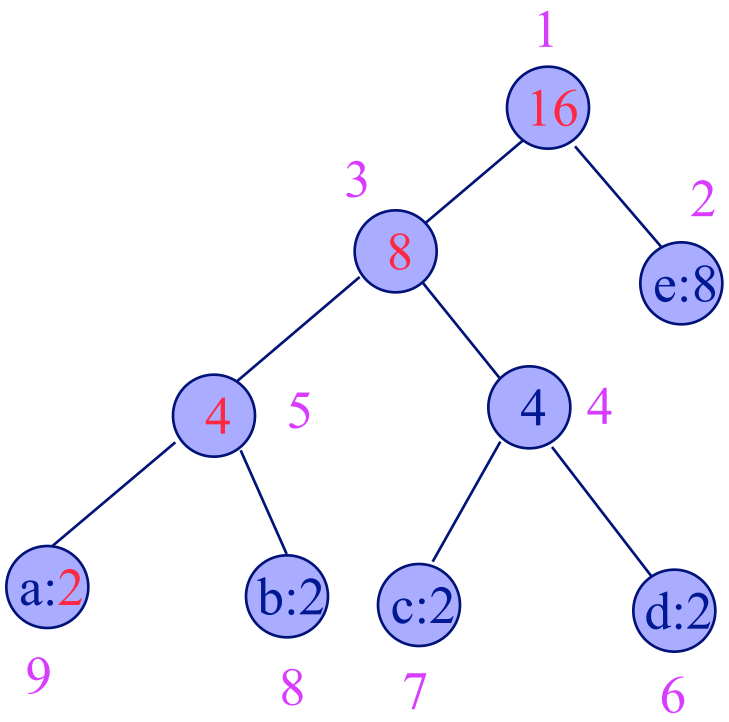

1
15
3
7
2
e:8
3
5
4
4
a:1
b:2
c:2
d:2
9
8
7
6



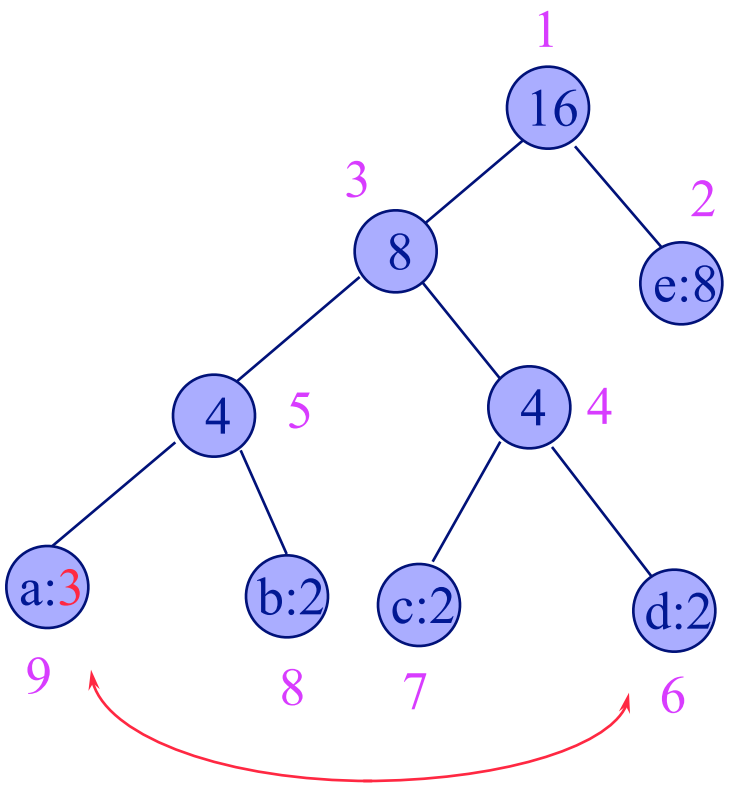
read symbol ´a´

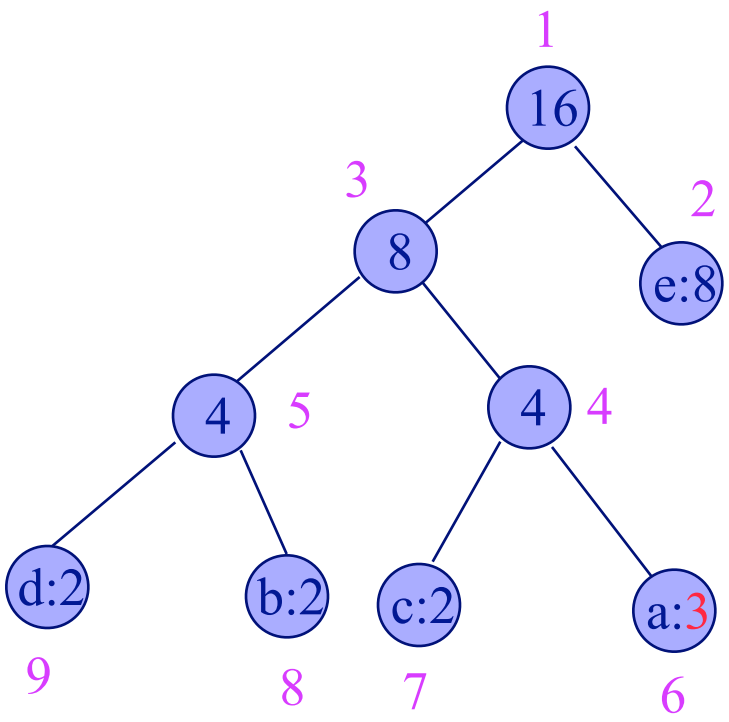

1
15
3
7
2
e:8
3
5
4
4
a:2
b:2
c:2
d:2
9
8
7
6

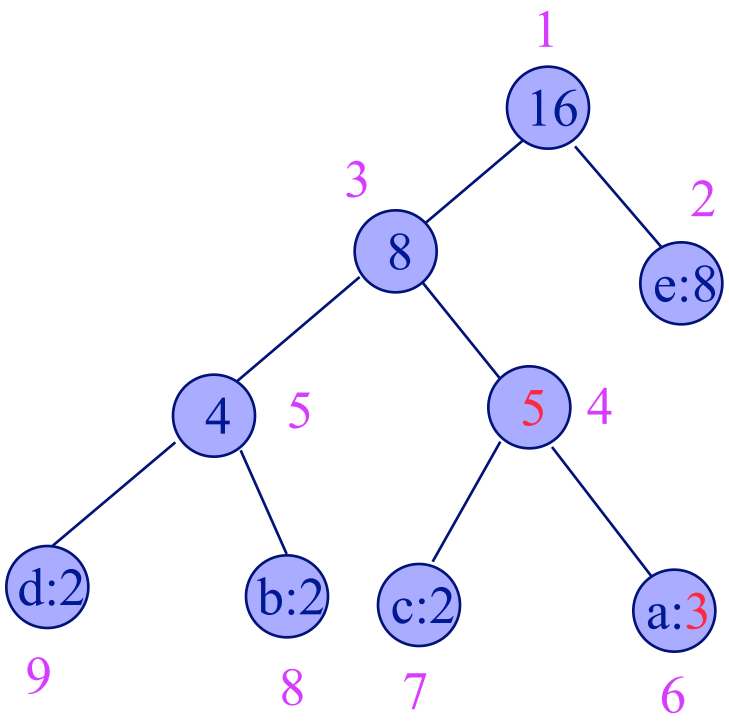

1
16
3
8
2
e:8
4
5
4
4
a:2
b:2
c:2
d:2
9
8
7
6

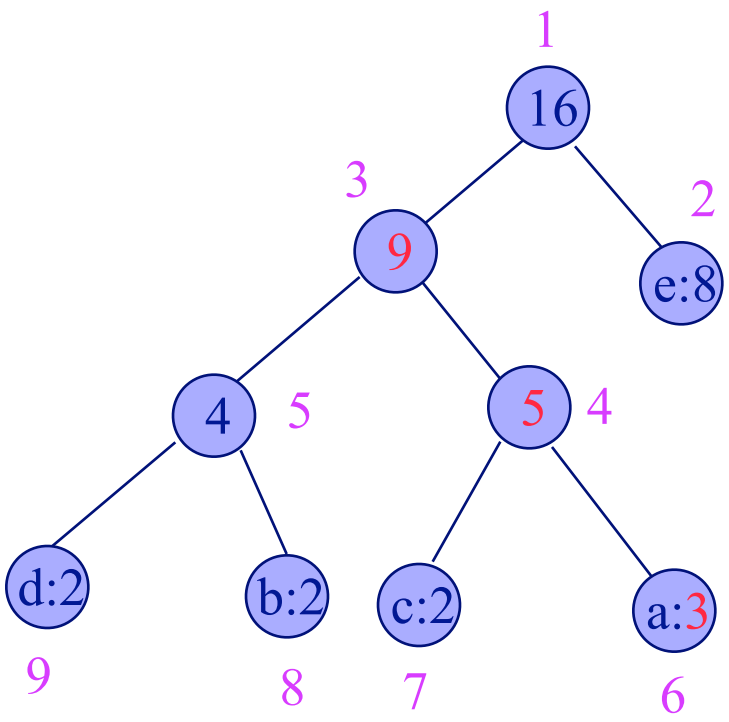


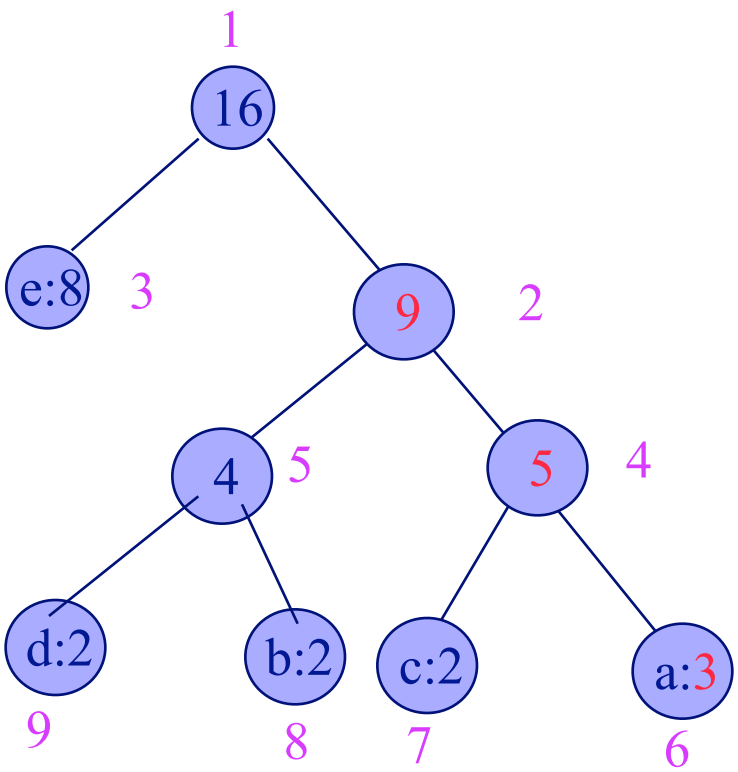
read symbol ´a´

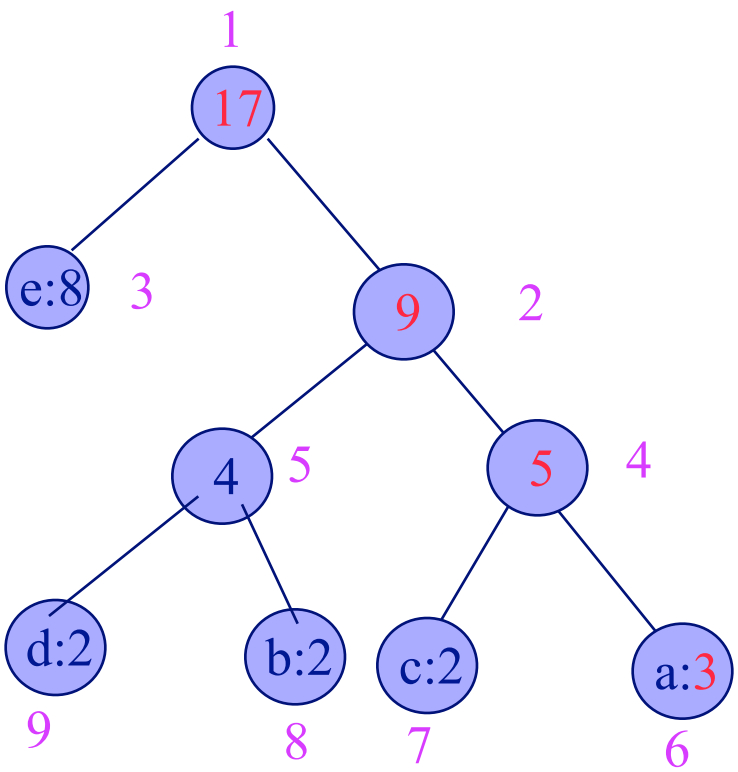

1
16
3
8
2
e:8
4
5
4
4
a:3
b:2
c:2
d:2
9
8
7
6


1
16
3
8
2
e:8
4
5
4
4
d:2
b:2
c:2
a:3
9
8
7
6


1
16
3
8
2
e:8
4
5
5
4
d:2
b:2
c:2
a:3
9
8
7
6


1
16
3
9
2
e:8
4
5
5
4
d:2
b:2
c:2
a:3
9
8
7
6


1
16
e:8
3
9
2
4
5
5
4
d:2
b:2
c:2
a:3
9
8
7
6


1
17
e:8
3
9
2
4
5
5
4
d:2
b:2
c:2
a:3
9
8
7
6

Zero-frequency problem

How to encode a novel symbol?

<u>1st solution</u>: Initialize Huffman tree with all symbols of the source alphabet, each with weight one.

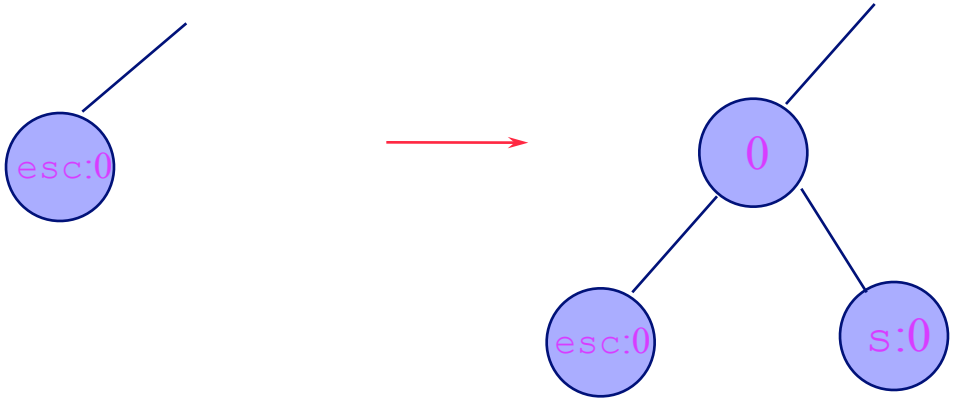
<u>2nd solution:</u> Initialize Huffman tree with a special symbol `esc`.

- encode the 1st occurrence of symbol s as a Huffman code of `esc` followed by s
- insert a new leaf representing s into the Huffman tree

Zero-frequency problem

Initialization

s is a novel symbol
which does not occur in the tree

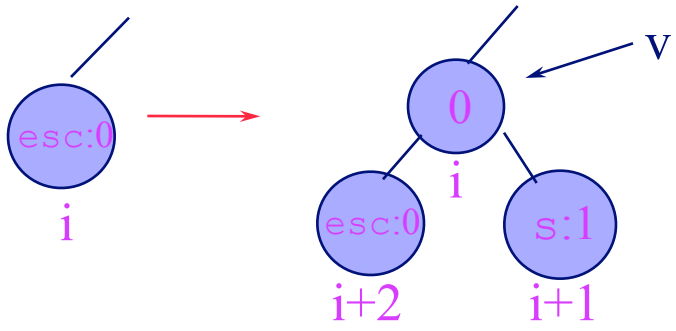


update the tree

Updating the Huffman tree

s is the current input symbol

if s does not occur in the tree :



else : v = leaf representing s

Updating the Huffman tree

```
while v ≠ None : # parent(root) = None
 exchange v with the vertex of
 the least order among the vertices
 of the same weight as v
 (subtrees exchanged with their roots)
 v.weight += 1
 v = parent(v)

the last symbol may be stored
directly into the esc vertex
```

FGK: Encoding

```
Initialize Huffman tree T
repeat read s
        if 1st occurrence of s :
                write(code(esc))
                write(s)
        else : write(code(s))
        update tree T with s
until EOF
```

FGK: Decoding

```
Initialize Huffman tree T
vertex = root of T
repeat
 while vertex is not a leaf : read bit
 if bit=0 : vertex = vertex.left-child
 else     : vertex = vertex.right-child
 if vertex.symbol == esc : read s
 else : s = vertex.symbol
 write s to output
 update tree T with s
until EOF
```

Average codeword length

l_A - average codeword length for algorithm A

$$l_{FGK} \leq l_H + O(1)$$

Implementation notes

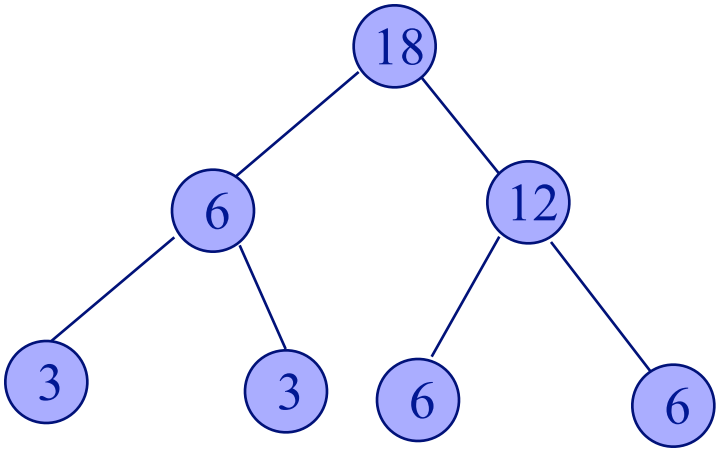
<u>Overflow problem</u>

- longest codeword length
- file length (root.weight)

<u>Solution</u>: multiply all weights by a coefficient

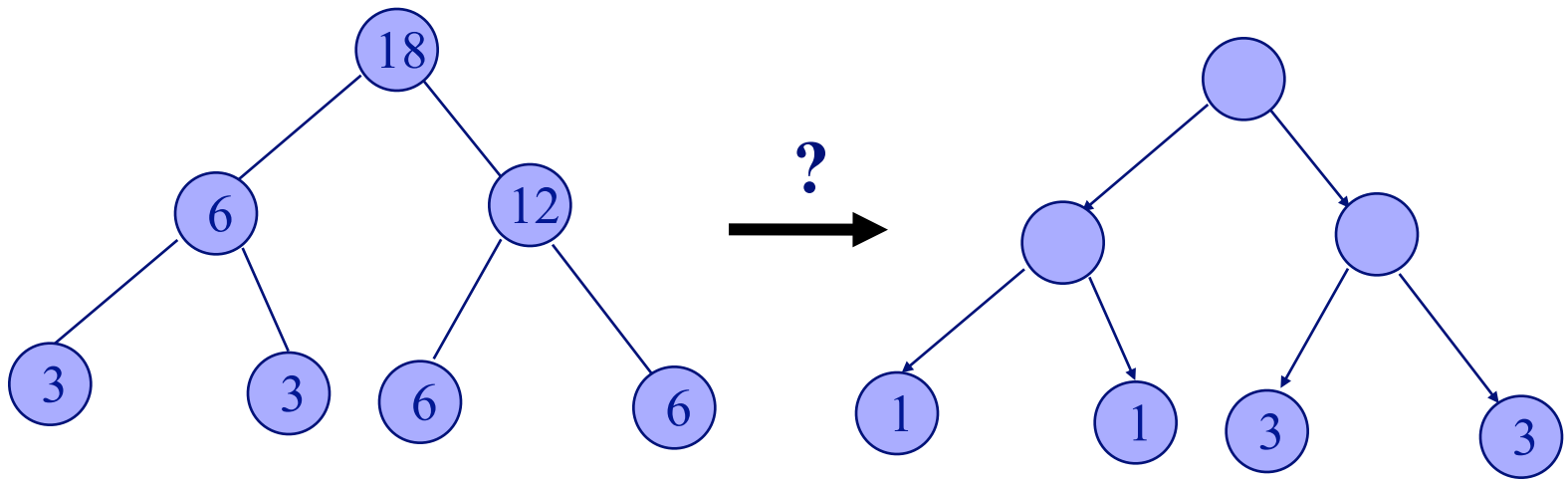
$$r < 1$$

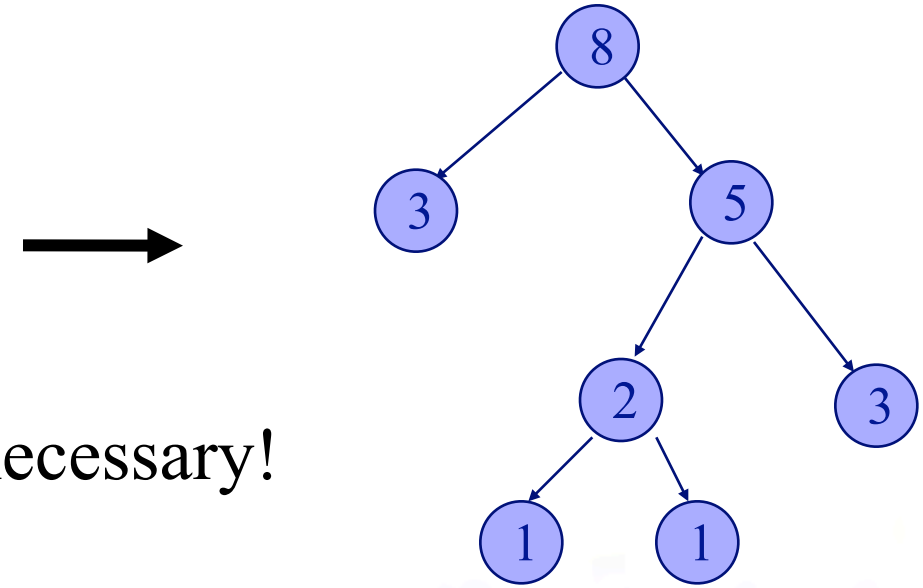
<u>Disadvantage</u>: tree reconstruction may be necessary



r = 1/2

Tree reconstruction





Tree reconstruction is necessary!

Statistics „aging“

each frequency /= 2

⇒ older frequencies have lower weight that new ones

⇒ better compression ratio

each frequency *= (1- x) in each step, $x \to 0$

current_symbol_frequency += $(1+x)^t$ at time t

<u>Example:</u> we need $(1+x)^d = 2$

- $x \approx \ln(1+x)$ for $x \to 0$
- $\Rightarrow x \approx (\ln 2)/d$ for $x \to 0$

<u>Choice of d</u>: stable probability distribution × frequent fluctuations

- $d = 1000$, $1+x = 1.00069$
- integral arithmetic ⇒ scaling
- 1st step: increment by 10000
- 2nd step: increment by $\lceil 10000(1+x) \rceil = 10007$
- 3rd step: increment by $\lceil 10000(1+x)^2 \rceil = 10014$
- when reaching the upper bound, each frequency /= 2

Data Compression Algorithms

Arithmetic coding

Alistair Moffat
University of Melbourne
Australia

Radford M. Neal
University of Toronto
Canada

Ian H. Witten
University of Waikato
New Zealand

Arithmetic coding – history

C.Shannon (1948)

P.Elias (1960?), Jelinek (1968)

Pasco, Rissanen (1976)

Pennebaker, Mitchell, Langdon, Arps (1988)

- Q-coder (IBM)

Witten, Neal, Cleary (1987)

Moffat, Neil, Witten (1998)

Arithmetic coding – idea

Each string $s \in A^*$ associate with a subinterval $I_s \subseteq \langle 0,1 \rangle$ such that

- $s \neq s' \Rightarrow I_s \cap I_{s'} = \emptyset$
- length of $I_s = p(s)$

$C(s)$ = number from I_s with the shortest code possible

✎ Problem: How can we estimate $p(s)$?

☞ Solution: Assume that

$$p(a_1 a_2 \ldots a_m) = p(a_1) \cdot p(a_2) \cdot \ldots \cdot p(a_m)$$

Example

Encode message BILL GATES

symbol	probability	interval
space	1/10	⟨0, 0.1)
A	1/10	⟨0.1, 0.2)
B	1/10	⟨0.2, 0.3)
E	1/10	⟨0.3, 0.4)
G	1/10	⟨0.4, 0.5)
I	1/10	⟨0.5, 0.6)
L	2/10	⟨0.6, 0.8)
S	1/10	⟨0.8, 0.9)
T	1/10	⟨0.9, 1.0)

symbol	lower	upper
	0.0	1.0
B	0.2	0.3

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0,256	0,258

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0,256	0,258
L	0.2572	0.2576

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0,256	0,258
L	0.2572	0.2576
space	0.2572	0.25724

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0,256	0,258
L	0.2572	0.2576
space	0.2572	0.25724
G	0.257216	0.257220

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0,256	0,258
L	0.2572	0.2576
space	0.2572	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168

symbol	lower	upper
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0,256	0,258
L	0.2572	0.2576
space	0.2572	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Encoding

```
l = 0   # initial lower bound
r = 1   # initial range
read(symbol)
while symbol ≠ EOF :
  l += r * lower_bound(symbol)
  r *= upper_bound(symbol)- lower_bound(symbol)
  read(symbol)
return the shortest code of a number
       from ⟨l,l+r)
```

Decoding

$code = 0.2572167754$

```
l = 0 # initial lower bound
r = 1 # initial range
repeat
 find a symbol such that
 lower_bound(symbol)≤(code-l)/r<upper_bound(symbol)
 output(symbol)
 l += r * lower_bound(symbol) # imitate the encoder
 r *= upper_bound(symbol)-lower_bound(symbol)
until whole input is decoded
```

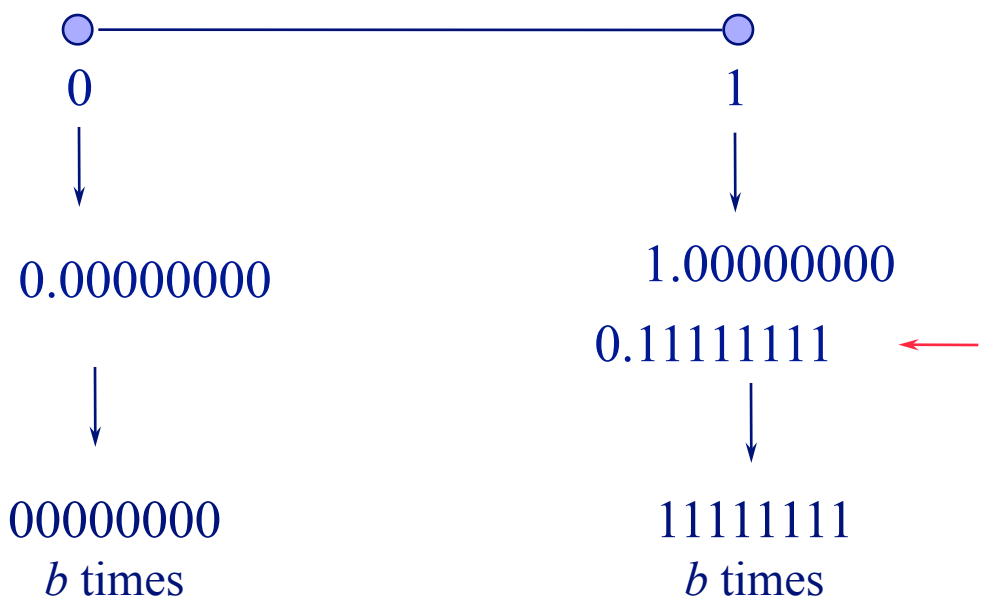
Decoding – alternative version

$code = 0.2572167754$

```
r = 1     # l is not needed
repeat
 find a symbol such that
 lower_bound(symbol) ≤ code/r < upper_bound(symbol)
 output(symbol)
 code -= r * lower_bound(symbol)
 r *= upper_bound(symbol)-lower_bound(symbol)
until whole input is decoded
```

Integer arithmetic implementation

interval ⟨0,1)



(Witten,Neal,Cleary) interval ⟨Lower,Upper) -> Lower, Upper-1

- $L = 0$; $U = 2^b-1$

(Moffat,Neil,Witten) interval ⟨Lower,Upper) -> ⟨L,L+R) -> L,R

- $L = 0$; $R = 2^{b-1}$
- invariant $2^{b-2} < R \leq 2^{b-1}$

Notation

Alphabet $A = \{1, 2, \ldots, n\}$

f_i = i^{th} symbol frequency in a message $S = s_1\, s_2 \ldots s_m$

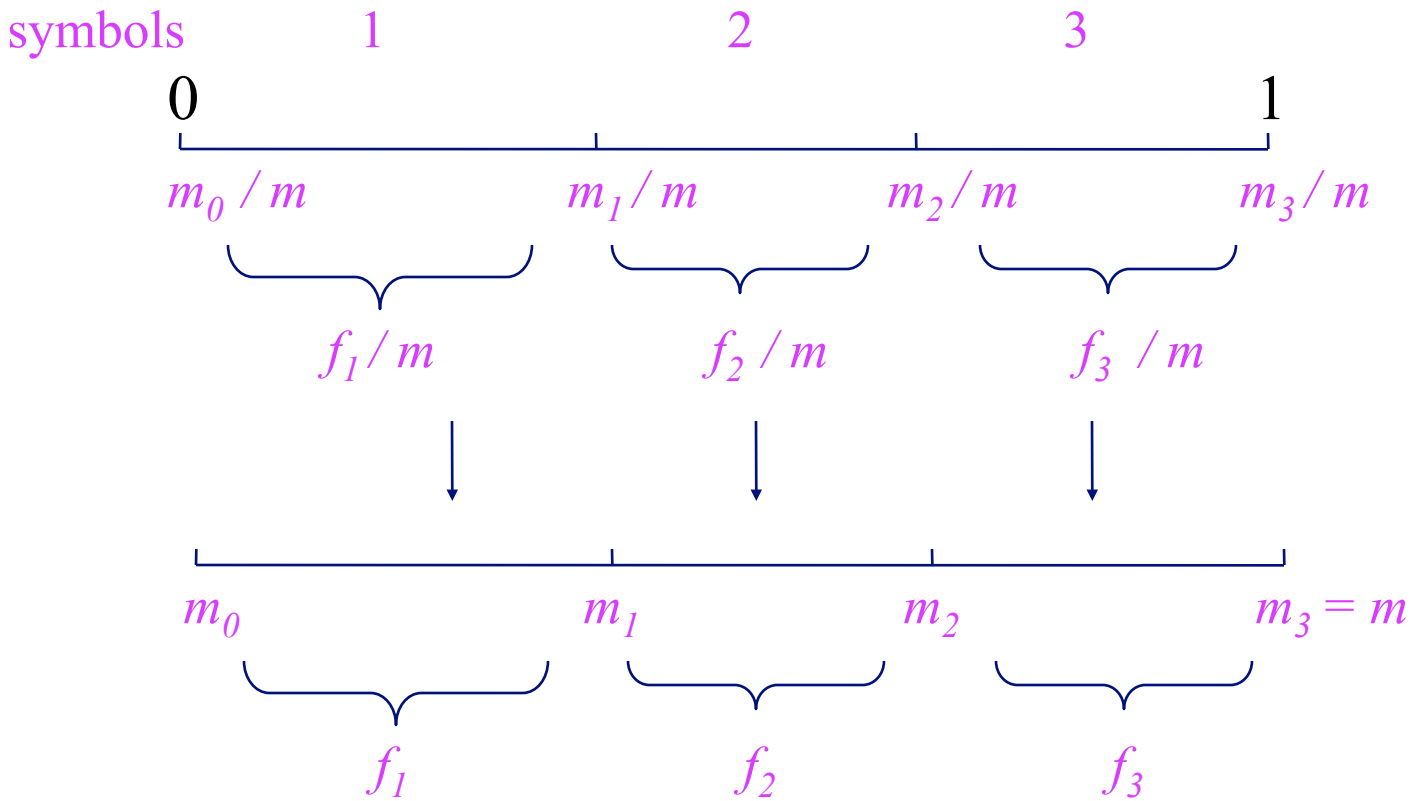
$$m = \sum_{i=1}^{k} f_i$$

m_i = *cumulative frequency* of ith symbol

$$m_i = \sum_{j=1}^{i} f_j \text{ for } i \in A \qquad m_0 = 0$$

Example

$$n = 3, \ A = \{1,2,3\}$$



Formulas for interval update

Real-valued: $\langle l, l+r) \subseteq \langle 0,1)$

Integer-valued: $\langle L, L + R) \subseteq \langle 0,2^b)$

Transform real-valued to integer-valued:

$l = L / m$, $r = R / m$

$$L \mathrel{+}= \left\lfloor \frac{R}{m} \right\rfloor \cdot m_{symbol-1}$$

if *symbol = n*

then $R \mathrel{-}= \left\lfloor \frac{R}{m} \right\rfloor \cdot m_{symbol-1}$

else $R = \left\lfloor \frac{R}{m} \right\rfloor \cdot (m_{symbol} - m_{symbol-1})$

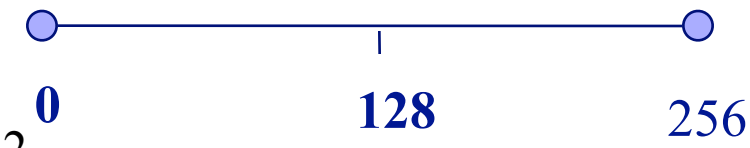
Example – encoding

symbol	a	b	c
f_i	40	1	9
m_i	40	41	50

encode input *acba*

$b = 8 \quad L^{(0)} = 0 = (00000000)_2$

$R^{(0)} = 128 = (10000000)_2$



$$L^{(1)} = 0 + \left\lfloor \frac{128}{50} \right\rfloor \cdot 0 = 0 = (00000000)_2$$

$$R^{(1)} = \left\lfloor \frac{128}{50} \right\rfloor \cdot 40 = 80 = (01010000)_2$$

Example – encoding acba – step 2

symbol	a	b	c
f_i	40	1	9
m_i	40	41	50

$$L^{(2)} = 0 + \left\lfloor \frac{80}{50} \right\rfloor \cdot 41 = 41 = (00101001)_2$$

$$R^{(2)} = 80 - \left\lfloor \frac{80}{50} \right\rfloor \cdot 41 = 39 = (00100111)_2$$

Step 2 output

$L^{(2)} = (00101001)_2 = 41$

$R^{(2)} = (00100111)_2 = 39 \leq 64$

$L^{(2)} + R^{(2)} = (01010000)_2 = 80$

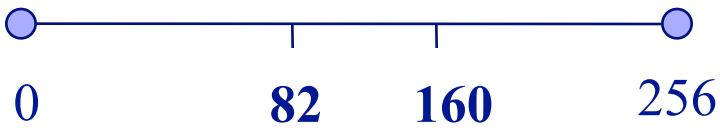
output(0)

$L^{(2)} <<= 1 \quad L^{(2)} = (01010010)_2 = 82$

$R^{(2)} <<= 1 \quad R^{(2)} = (01001110)_2 = 78 > 64$

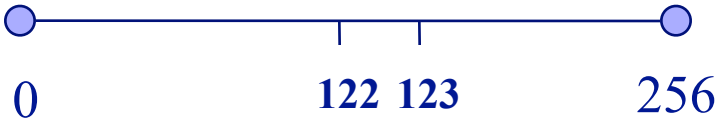
Example – encoding acba – step 3

symbol	a	b	c
f_i	40	1	9
m_i	40	41	50



$$L^{(3)} = 82 + \left\lfloor \frac{78}{50} \right\rfloor \cdot 40 = 122 = (01111010)_2$$

$$R^{(3)} = \left\lfloor \frac{78}{50} \right\rfloor \cdot 1 = 1 = (00000001)_2$$

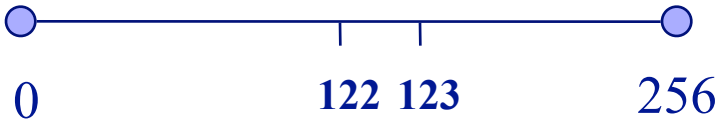


Step 3 output

$L^{(2)} = (01111010)_2 = 122$

$R^{(2)} = (00000001)_2 = 1 \leq 64$

$L^{(2)} + R^{(2)} = (01111011)_2 = 123$



output(0)

$L^{(2)} <<= 1 \qquad L^{(2)} = (11110100)_2 = 244$

$R^{(2)} <<= 1 \qquad R^{(2)} = (00000010)_2 = 2 \leq 64$

$L^{(2)} = (11110100)_2 = 244$

$L^{(2)} + R^{(2)} = (11110110)_2 = 246$

output(1)

Step 3 output

$L^{(3)} <<= 1$ $\quad$ $L^{(3)} = (11101000)_2 = 232$

$R^{(3)} <<= 1$ $\quad$ $R^{(3)} = (00000100)_2 = 4 \leq 64$

$L^{(3)} = (11101000)_2 = 232$

$L^{(3)} + R^{(3)} = (11101100)_2 = 236$

output(1)

$L^{(3)} <<= 1$ $\quad$ $L^{(3)} = (11010000)_2 = 208$

$R^{(3)} <<= 1$ $\quad$ $R^{(3)} = (00001000)_2 = 8 \leq 64$

output(1)

$L^{(3)} <<= 1$ $\quad$ $L^{(3)} = (10100000)_2 = 160$

$R^{(3)} <<= 1$ $\quad$ $R^{(3)} = (00010000)_2 = 16 \leq 64$

Step 3 output

output(1)

$L^{(3)} <<= 1$ $\quad L^{(3)} = (01000000)_2 = 64$

$R^{(3)} <<= 1$ $\quad R^{(3)} = (00100000)_2 = 32 \leq 64$

output(0)

$L^{(3)} <<= 1$ $\quad L^{(3)} = (10000000)_2 = 128$

$R^{(3)} <<= 1$ $\quad R^{(3)} = (01000000)_2 = 64 \leq 64$

output(1)

$L^{(3)} <<= 1$ $\quad L^{(3)} = (00000000)_2 = 0$

$R^{(3)} <<= 1$ $\quad R^{(3)} = (10000000)_2 = 128 > 64$

Encoding acba – step 4

znak	a	b	c
f_i	40	1	9
m_i	40	41	50

$$L^{(4)} = 0 + \left\lfloor \frac{128}{50} \right\rfloor \cdot 0 = 0 = (00000000)_2$$

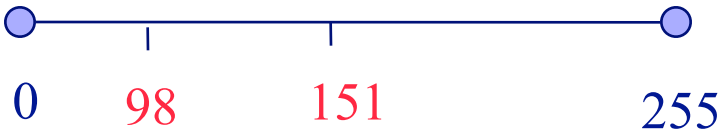
$$R^{(4)} = \left\lfloor \frac{128}{50} \right\rfloor \cdot 40 = 80 = (01010000)_2$$

Another example – critical situation

$L \quad = (01100010)_2 = 98$

$R \quad = (00110101)_2 = 53 \leq 64$

$L+R = (10010111)_2 = 151$



$L \quad = (01111111)_2$

$L+R = (10000000)_2$!

$L := (L - 64) <<= 1$

$L = (01100010)_2 \quad \Rightarrow \quad L = (01000100)_2 = 68$

$R <<= 1 \quad \Rightarrow \quad R = (01101010)_2 = 106 > 64$

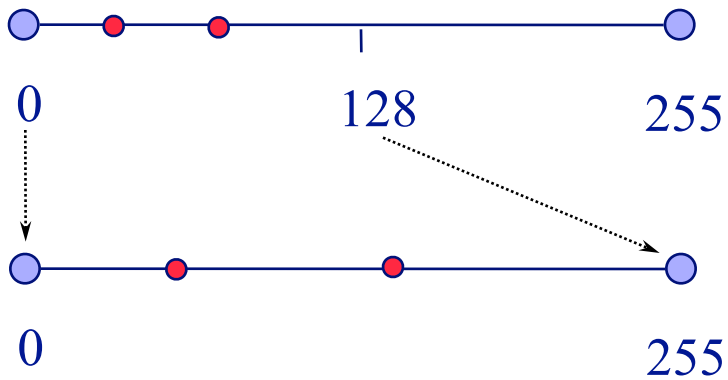
define a *counter*

counter := 0

counter ++

Encoding – interval expansion

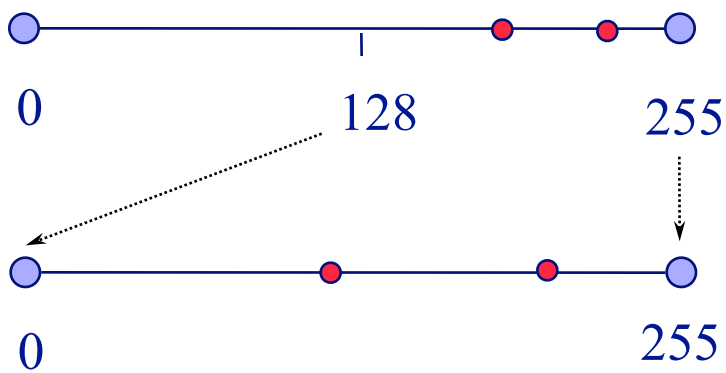
$E_1 : \langle 0, 0.5) \rightarrow \langle 0, 1) \quad E_1(x) = 2x$




```
if R ≤ 2^(b-2) and ⟨L,L+R) ⊆ ⟨0,2^(b-1)):
    output(0)
    while counter > 0: output(1)
                       counter --
    L = 2·L; R = 2·R
```

Encoding – interval expansion

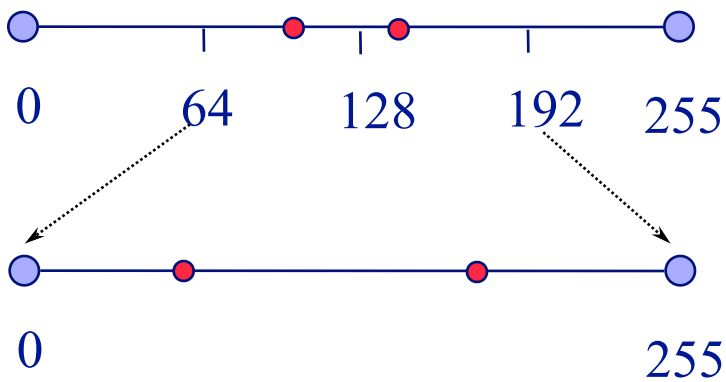
$E_2 : \langle 0.5, 1) \rightarrow \langle 0, 1)$ $\quad E_2(x) = 2(x\text{-}0.5)$




```
if R ≤ 2^(b-2) and ⟨L,L+R) ⊆ ⟨2^(b-1),2^b):
    output(1)
    while counter > 0 : output(0)
                        counter --
    L = 2·(L-2^(b-1)) ; R = 2·R
```

Encoding – interval expansion

$E_3 : \langle 0.25, 0.75) \rightarrow \langle 0, 1)$ $\quad E_3(x) = 2(x\text{-}0.25)$




```
if R ≤ 2^(b-2) and L < 2^(b-1) and L+R > 2^(b-1) :
    # ⟨L, L+R) ⊆ ⟨2^(b-2), 2^(b-1)+2^(b-2))
    counter++
    L = 2·L – 2^(b-1); R = 2·R
```

Arit_encode_symbol(s)

$r = \lfloor R / m \rfloor$; $L = L + r \cdot m_{s-1}$

if $m_s < m$: $R = r \cdot (m_s - m_{s-1})$

else : $R = R - r \cdot m_{s-1}$

while $R \leq 2^{b-2}$:

if $L+R \leq 2^{b-1}$: output_bits(0)

elif $2^{b-1} \leq L$: output_bits(1)

$L = L - 2^{b-1}$

else : $counter{+}{+}$; $L = L - 2^{b-2}$

$L = 2 \cdot L$; $R = 2 \cdot R$

output_bits(x)

```
output(x)
while counter > 0 : output(1-x)
                    counter--
```

Arit_encode_block(M,m)

```
{M = s1 s2 … sm}
for i in range(n+1)    : mi := 0
for i in range(1,m+1) : msi++
output(n); output(m)
for i in range(1,n+1) : output(mi)
                        mi = mi + mi-1
L = 0; R = 2^(b-1) ; counter = 0
for i in range(1,m+1) :
        Arit_encode_symbol(si)
output(b bits of L) using output_bits()
```

Arit_decode_block()

read(n); read(m); $m_0 = 0$

for i **in range** (1,n+1) **:** read(m_i)

$m_i = m_i + m_{i-1}$

$R = 2^{b-1}$; read b bits of code into L

for i **in range**(1,m+1):

$r = \lfloor R/m \rfloor$; $target = \min(m-1, \lfloor L/r \rfloor)$

find symbol $s \in \{1,..,n\}$ such that

$m_{s-1} \leq target < m_s$

output(s)

Arit_decode_symbol(s)

Arit_decode_symbol(s)

$L = L - r \cdot m_{s-1}$

if $m_s < m$: $R = r \cdot (m_s - m_{s-1})$

else : $R = R - r \cdot m_{s-1}$

while $R \leq 2^{b-2}$:

$R = 2 \cdot R$

$L = 2 \cdot L$ + read_one_bit_of_code

Time complexity

Encoding $O(n + c + m)$

- c = output code length in bits
- n = alphabet size
- m = input length

Decoding $O(n + c + m \log n)$

- if symbol s satisfying $m_{s-1} \leq target < m_s$ is found using binary search

Cumulative frequencies table → *implicit tree*

- P.Fenwick (1994), A.Moffat (1999)

⇒ decoding in time $O(n + c + m)$

- higher space complexity (n extra words of memory)

Frequency counts

Representation of

- variables L and R : b bits
- frequencies: f bits

$\Rightarrow$ input length $m < 2^f$

Relationship between b and f?

- we need $R \geq m$
- since $R > 2^{b-2}$, $R > m \Leftarrow 2^{b-2} \geq m$
- since $2^f > m$, $2^{b-2} > m \Leftarrow 2^{b-2} \geq 2^f$
- $2^{b-2} \geq 2^f \Leftrightarrow b-2 \geq f$

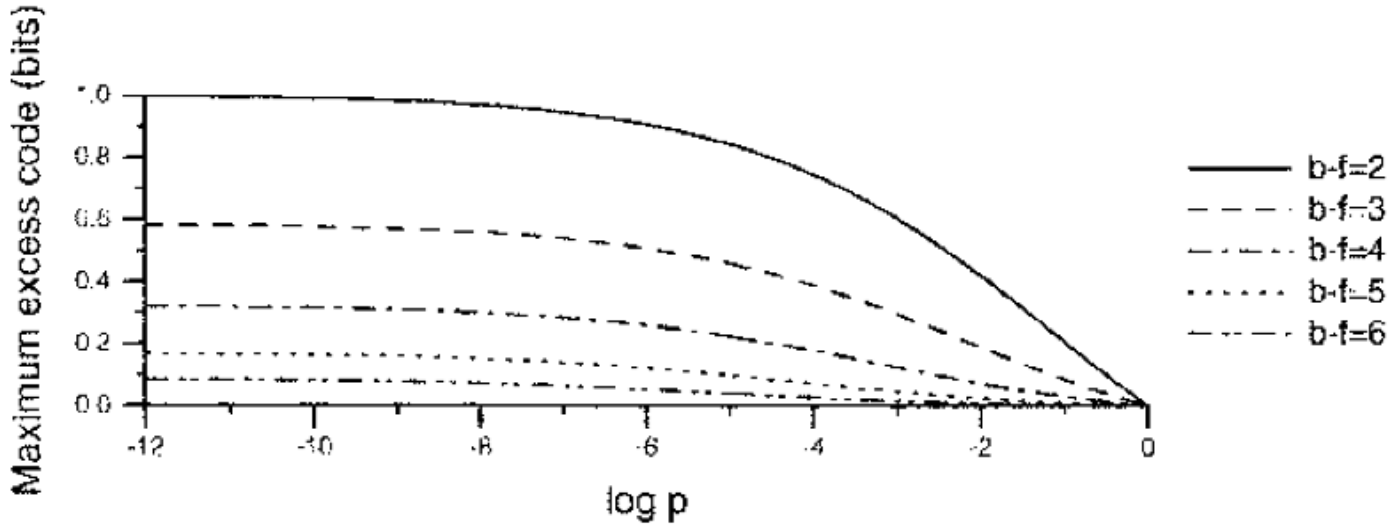
Hence we need $b - f \geq 2$

Precision loss in integer-valued operations

Moffat et al. (1998)

Roundoff errors may prolong the code length

- by less than $\approx \log_2 e \,/\, 2^{b-f-2}$ bits/symbol
- loss decreases with increasing p_n = probability of the nth symbol



Precision loss in integer-valued operations

$b = 32,\ f = 24 \Rightarrow m < 2^{24} \approx 16 \cdot 10^6$

Average code length increase

- < 0.01 bit/symbol

Adaptive version

Problem: updating cumulative frequencies

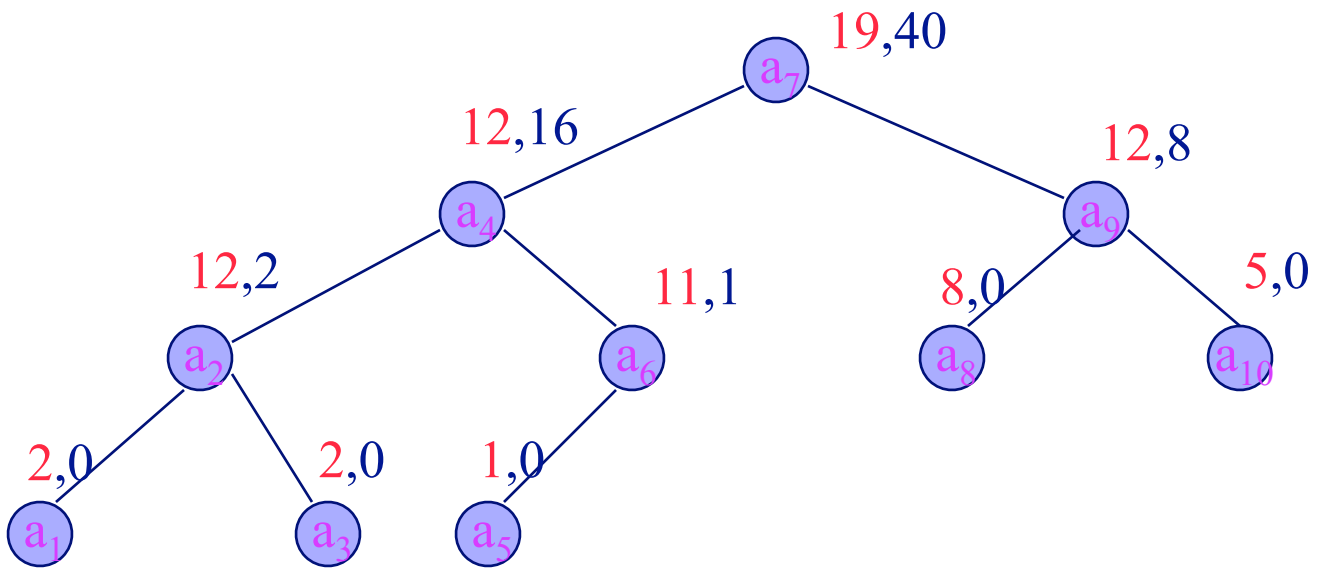
- decrease # updates → sort the symbols by frequencies
- periodical scaling of frequencies

Cumulative frequencies table as an implicit tree

- P.Fenwick (1994) update in time O(log n)
- A.Moffat (1999) update in time O(log (1+s)) for s^{th} symbol

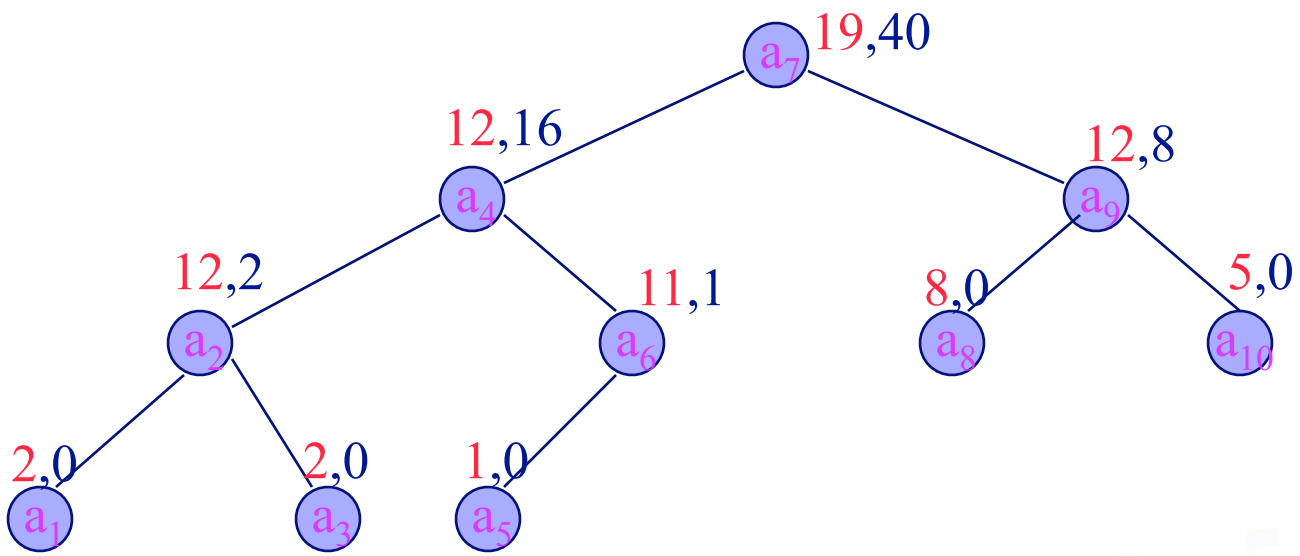
Fenwick implicit tree

alphabet	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
frequencies f_i	2	12	2	12	1	11	19	8	12	5
cumulative frequencies m_i	2	14	16	28	29	40	59	67	79	84

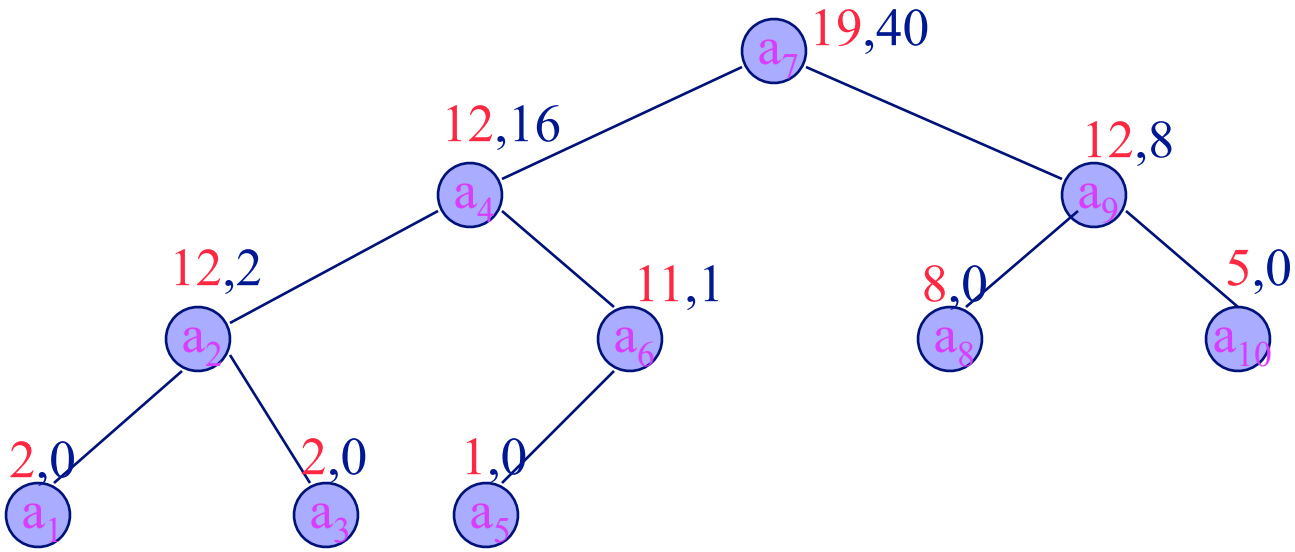


Fenwick implicit tree

alphabet	a_7	a_4	a_9	a_2	a_6	a_8	a_{10}	a_1	a_3	a_5
frequencies f_i	19	12	12	12	11	8	5	2	2	1
cumulative frequencies m_i	59	28	79	14	40	67	84	2	16	29
Fenwick frequencies	40	16	8	2	1	0	0	0	0	0

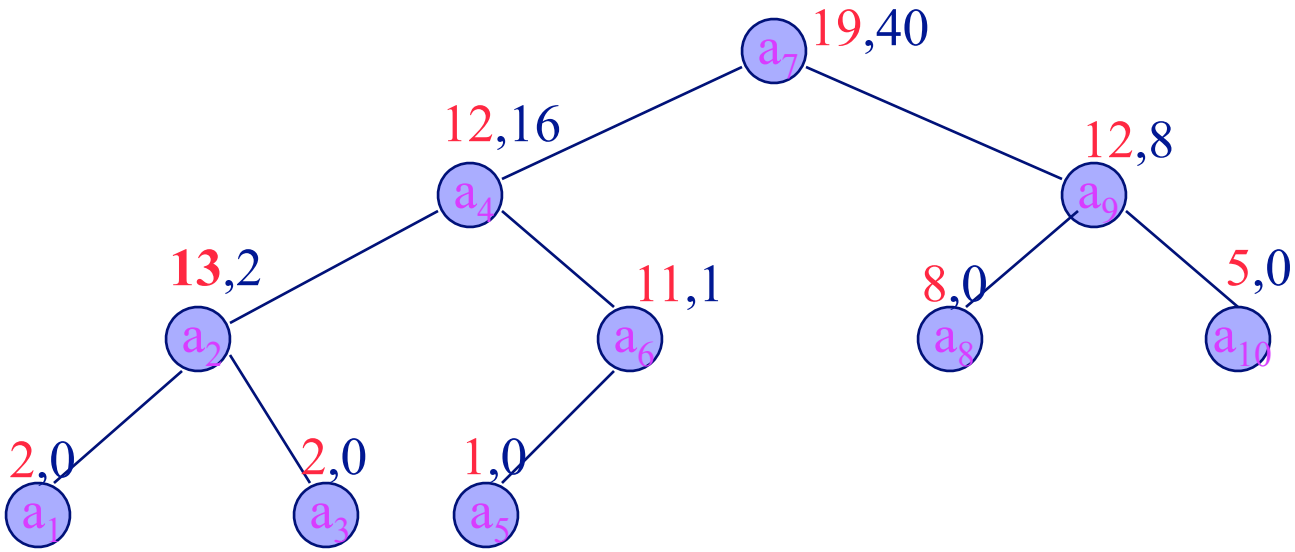


Tree update



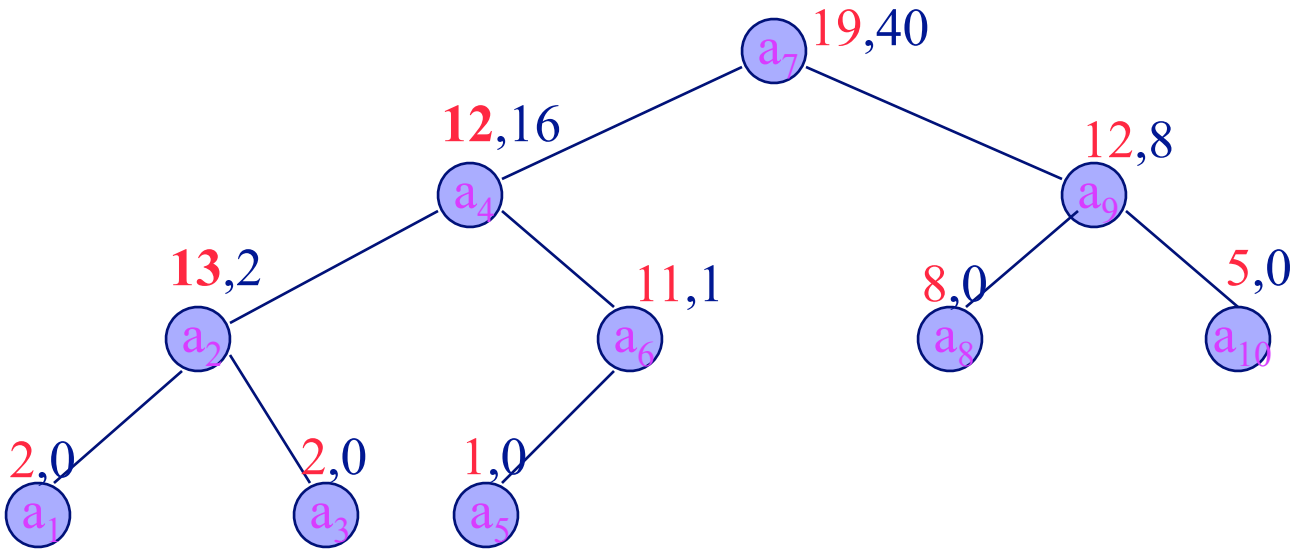
read a_2

Tree update



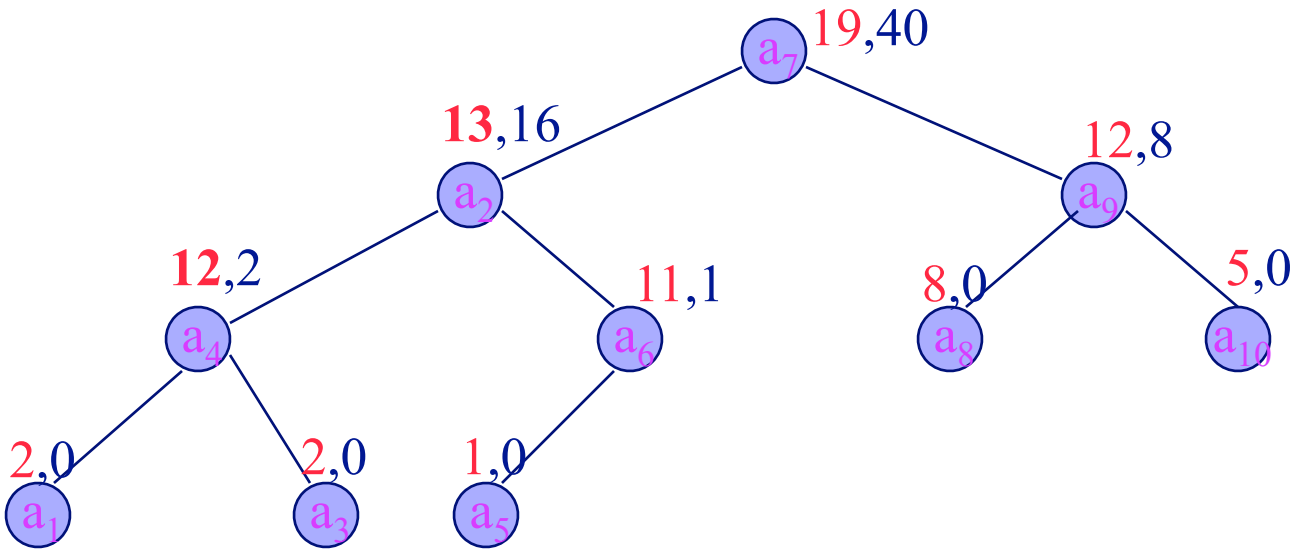
read a_2

Tree update



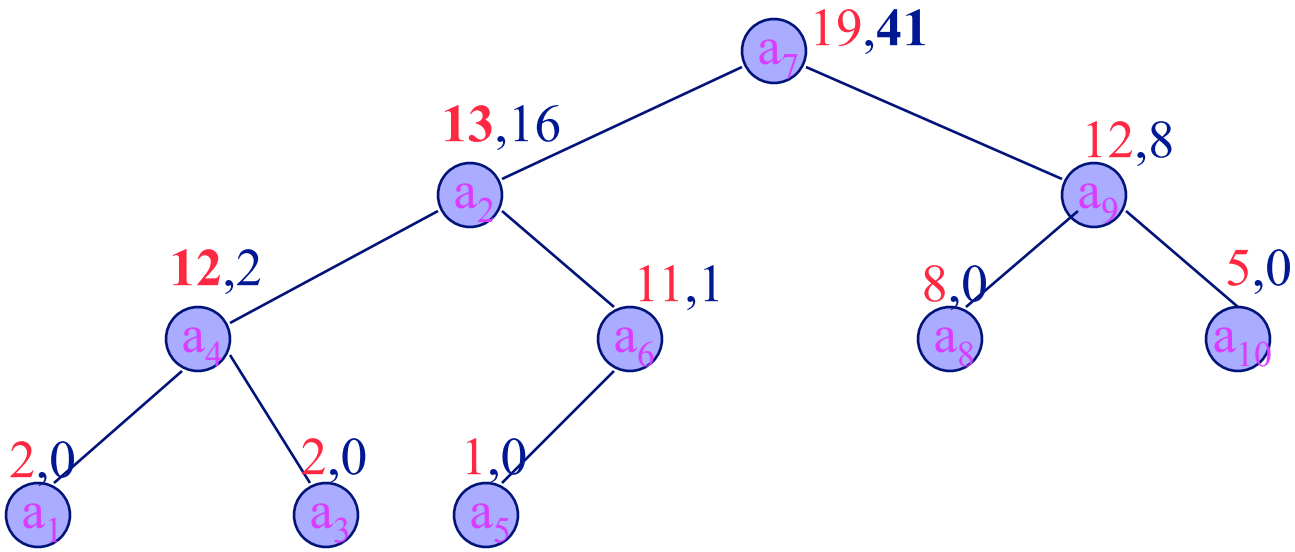
read a_2

Tree update



read a_2

Tree update



read a_2

Time complexity

Cumulative frequencies + tree update:

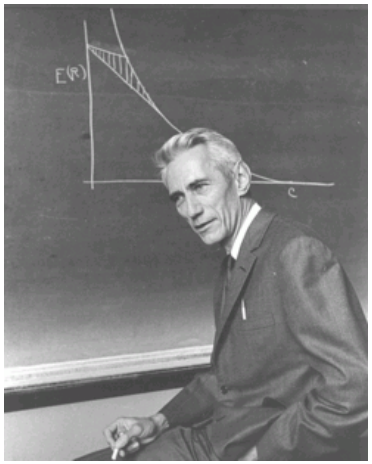
- Fenwick: $O(\log n)$ time
- Moffat: $O(z+1)$ time
  - » z = # bits necessary to encode the current symbol

$\Rightarrow$ Decoding time

- $O(n + c + m)$

Data Compression Algorithms

Theoretical limits to lossless compression



Claude Shannon
(1916 – 2001)

Information Theory

Information

- What is it?
- Is it measurable?
- Are there some theoretical limits to the length of a message with a given information?
- How close to these limits are methods that we have developed so far?

Hartley's formula

Motivation: Information in a message

- minimum # of YES/NO questions
- revealing the message content

Ralph Hartley (1928)

If x is a member of n element set, then x carries $\log_2 n$ bits of information.

Hartley's formula

$|R| = n$, $(x_1, \ldots, x_k) \in R^k$

Let S_k denote the least # of questions necessary to determine all x_i

$$\log_2 n^k \leq S_k < \log_2 n^k + 1$$

$$\log_2 n \leq S_k / k < \log_2 n + 1/k$$

☞ **Conclusion**

S_k / k = average # questions necessary to determine one element

Shannon's formula

Let X be a discrete random variable with range R and probability mass function $p(x) = P(X=x)$ for $x \in R$

☀ **Observation**: $|R| = n$, $p(x) = 1/n$ for every $x \in R$

$\Rightarrow \log_2 n = \log_2 1/p(x) = \Sigma_{x \in R}\, p(x) \log_2 1/p(x)$

Claude Shannon (1948): ***Entropy*** H(X) of a discrete random variable X with range R is defined by

$$\mathrm{H}(X) = -\sum_{x \in R} p(x) \log p(x)$$

Entropy

$$\mathrm{H}(X) = -\sum_{x \in R} p(x) \log p(x)$$

☀ Examples

- motivational game: R = set of candidates for guessing
- coding: R - source alphabet, X - source of information

☝ Justification

- statistical mechanics analogy (Ludwig Boltzmann, 1872)
- axiomatic approach
- relationship to coding & compression

Basic properties

Theorem. H(X) ≥ 0.

Lemma. Let $p_i, q_i > 0, i = 1,\ldots,k, \sum_{i=1}^{k} p_i = \sum_{i=1}^{k} q_i = 1$. Then

$$-\sum_{i=1}^{k} p_i \log p_i \le -\sum_{i=1}^{k} p_i \log q_i$$

with an equality iff $p_i = q_i$ for every $1 \le i \le k$.

Proof. Put $f(x) = x - \ln x - 1$ for $x > 0$.

Then $f'(x) = 1 - \dfrac{1}{x}, f''(x) = \dfrac{1}{x^2}$.

Hence f is convex, $f'(x) = 0$ for $x = 1, f(1) = 0$.

It follows that $f(x) \ge f(1) = 0$ for $x > 0$

and the equality holds iff $x = 1$.

Proof of the Lemma (continued)

Hence $\ln x \leq x - 1$ for $x > 0$ with an equality iff $x = 1$.

$$\ln p_i / q_i \leq p_i / q_i - 1 \qquad \text{for } i = 1,\ldots, k.$$

$$q_i \ln p_i / q_i \leq p_i - q_i \qquad \text{for } i = 1,\ldots, k.$$

$$q_i \ln p_i - q_i \ln q_i \leq p_i - q_i \qquad \text{for } i = 1,\ldots, k.$$

$$-\sum_{i=1}^{k} q_i \ln q_i \leq -\sum_{i=1}^{k} q_i \ln p_i$$

with an equality iff $p_i / q_i = 1$ for all $i = 1,\ldots, k$.

Basic properties

- **Theorem.** For a random variable X with a finite range R, $|R|=n$, we have

$$H(X) \leq \log n$$

with an equality iff $p(x)=1/n$ for every $x \in H$.

Entropy – axiomatic approach

Theorem. Let a sequence of functions $H_n(p_1, p_2, \ldots, p_n)$, $n = 2,3,\ldots$, satisfies the following:

- $H_2(1/2, 1/2) = 1$
- $H_2(p, 1-p)$ is continuous for $0 \le p \le 1$
- $H_{n+1}(p_1,\ldots,p_{n-1},q_1,q_2) = H_n(p_1,\ldots,p_n) + p_n H_2(q_1/p_n, q_2/p_n)$, where $p_n = q_1 + q_2 > 0$, $p_i \ge 0$, $p_1 + \ldots + p_n = 1$.

Then for $n = 2,3,\ldots$ we have

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

✎ Problems

① Define the entropy for a discrete random vector.

② Prove or disprove: For an n-dimensional random vector $\mathbf{X}=(X_1,\ldots,X_n)$ whose components are independent random variables with the same probability distribution, we have

$$H(\mathbf{X}) = n \cdot H(X_i).$$

Information × coding theory

Code C for a discrete random variable X with range R is a mapping $C: R \to \{0,1\}^*$.

Code C generates an encoding $C^*: R^* \to \{0,1\}^*$ defined by $C^*(x_1 \ldots x_k) = C(x_1) \ldots C(x_k)$.

Code C is *uniquely decodable* if

$$u,v \in R^*,\ C^*(u)=C^*(v) \Rightarrow u = v.$$

Average codeword length L(C) of a code C

- for a discrete random variable X
- with probability function $p(x)$

is defined by $L(C)=\Sigma_{x\in R}\, p(x)|C(x)|$.

Upper and lower bounds

Theorem (Kraft-McMillan)

Codeword lengths $l_1, l_2, \ldots$ of a uniquely decodable code C satisfy $\sum_i 2^{-l_i} \leq 1$.

On the other hand, if natural numbers $l_1, l_2, \ldots$ satisfy the inequality, then there is a prefix code with codewords of these lengths.

Proof

First put $R = \{x_1, \ldots, x_n\}$, $l_i = l(x_i) = |C(x_i)|$ for $i = 1, \ldots, n$. Then

Proof of Kraft-McMillan Theorem

$$
\begin{aligned}
(\sum_{i=1}^{n} 2^{-l_i})^k &= (\sum_{x \in R} 2^{-l(x)})^k \\
&= \sum_{x_{i_1},\ldots,x_{i_k} \in R} 2^{-l(x_{i_1})} \cdot 2^{-l(x_{i_2})} \cdot \ldots \cdot 2^{-l(x_{i_k})} \\
&= \sum_{x_{i_1},\ldots,x_{i_k} \in R} 2^{-(l(x_{i_1})+l(x_{i_2})+\cdots+l(x_{i_k}))} \\
&= \sum_{x_{i_1}\ldots x_{i_k} \in R^*} 2^{-l(x_{i_1} x_{i_2} \ldots x_{i_k})} \\
&= \sum_{i=1}^{k \cdot l_{max}} n(i) \cdot 2^{-i}
\end{aligned}
$$

Proof of Kraft-McMillan Theorem

$$= \sum_{i=1}^{k \cdot l_{max}} n(i) \cdot 2^{-i} \leq \sum_{i=1}^{k \cdot l_{max}} 2^{i} \cdot 2^{-i} = k \cdot l_{max}$$

$$l_{max} = \max\{l_1, \ldots, l_n\}$$

$$n(i) = |\{x_{i_1} \ldots x_{i_k} \mid l(x_{i_1} \ldots x_{i_k}) = i\}|$$

$$n(i) \leq 2^{i}$$

Hence $\sum_{i=1}^{n} 2^{-l_i} \leq (k \cdot l_{max})^{1/k}$ for $k = 1,2,\ldots$

$$\sum_{i=1}^{n} 2^{-l_i} \leq \lim_{k \to \infty} (k \cdot l_{max})^{1/k} = 1 .$$

Proof of Kraft-McMillan Theorem

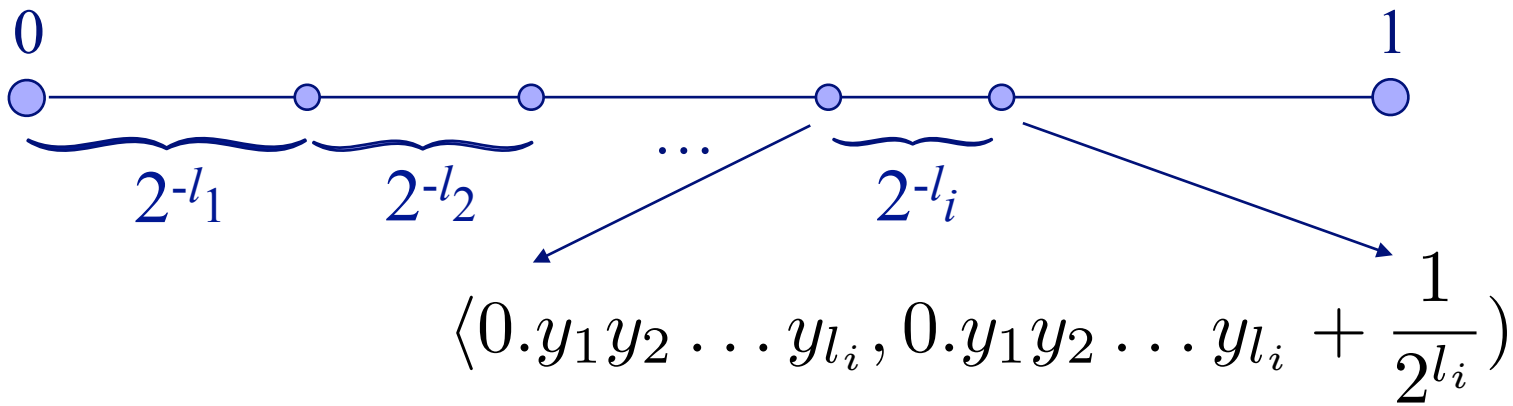
If R is infinite, then

$$\sum_{i=1}^{\infty} 2^{-l_i} = \lim_{N\to\infty} \sum_{i=1}^{N} 2^{-l_i} \leq 1\,.$$

Now let $\sum_i 2^{-l_i} \leq 1$.

We can assume that $l_1 \leq l_2 \leq \ldots$,

which means that $2^{-l_1} \geq 2^{-l_1} \geq \ldots$



$y_1y_2\ldots y_{l_i}$ is the desired codeword of length l_i

Upper and lower bounds

Theorem

Let C be a uniquely decodable code for a random variable X. Then

$$\mathrm{H}(\mathrm{X}) \leq \mathrm{L}(\mathrm{C}).$$

Proof

Let R be the range of X. First assume that $R = \{x_1, \ldots, x_n\}$.

Let

$$l_i = |C(x_i)|, p_i = P(X = x_i), q_i = \frac{2^{-l_i}}{\sum_{j=1}^{n} 2^{-l_j}}$$

for all $i = 1, \ldots, n$.

Note that then $$\sum_{i=1}^{n} q_i = 1\,.$$

Proof of Theorem

$$H(X) = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i \ \leq -\sum_{i=1}^{n} p_i \cdot \log_2 q_i \ =$$

$$= -\sum_{i=1}^{n} p_i \cdot \log_2 2^{-l_i} + \sum_{i=1}^{n} p_i \cdot \log_2 (\sum_{j=1}^{n} 2^{-l_j})$$

$$\leq \sum_{i=1}^{n} p_i \cdot l_i \ = L(C)$$

Infinite case

$$-\sum_{i=1}^{\infty} p_i \cdot \log_2 p_i = \lim_{N\to\infty} -\sum_{i=1}^{N} p_i \cdot \log_2 p_i \leq \sum_{i=1}^{N} p_i l_i$$

$$= L(C)\,.$$

Upper and lower bounds

- **Theorem.** An arbitrary optimal prefix code C for a random variable X satisfies

$$\mathrm{H}(\mathrm{X}) \leq \mathrm{L}(\mathrm{C}) < \mathrm{H}(\mathrm{X}) + 1\,.$$

- **Corollary.** Average codeword length L of the Huffman code satisfies

$$\mathrm{H} \leq L < \mathrm{H} + 1\,.$$

Proof of Theorem (upper bound)

Put $l_i = \lceil -\log_2 p_i \rceil$. Then

$$-\log_2 p_i \leq l_i < -\log_2 p_i + 1$$

$$p_i \geq 2^{-l_i}$$

$$1 \geq \sum 2^{-l_i}$$

By Kraft-McMillan Theorem there is a prefix code C with codewords of lengths l_i.

$$\sum p_i \cdot l_i < -\sum p_i \cdot \log_2 p_i + 1$$

$$L(C) < H(X) + 1$$

Example

symbol	frequency	entropy [b]	Huffman code length [b]
E	20	1.26	1
A	20	1.26	2
X	3	4	3
Y	3	4	4
Z	2	4.58	4

Huffman code over an extended alphabet

$$A = \{a_1, \ldots, a_m\}$$

$$A^{(n)} = \{\underbrace{a_1 a_1 \ldots a_1}_{n\text{-times}}, \underbrace{a_1 a_1 \ldots a_2}_{n\text{-times}}, \ldots, \underbrace{a_m a_m \ldots a_m}_{n\text{-times}}\}$$

✎ Problem

- derive lower and upper bounds on an average codeword length for Huffman code over an alphabet extended to n-grams
- you may assume that symbols in source message are
  - » independent
  - » with identical probability distribution
  - » and entropy H

Analysis of arithmetic coding

Let X be a discrete random variable with range $R = \{1,2,3,\ldots\}$ and a probability function p

Remark. Range R = set of messages to be encoded

Let $F(x) = P(X \leq x) = \Sigma_{y \leq x}\, p(y)$

$\overline{F(x)} = ½(F(x-1) + F(x)) = 0.y_1\, y_2\, y_3\ldots$

- $y_i \in \{0,1\}$

For $x \in R$ put $C(x) = y_1\, y_2 \ldots y_{l(x)}$

- $l(x) = \lceil \log_2 1/p(x) \rceil + 1$

Proposition. Then C is a prefix code for X satisfying

$$H(X) \leq L(C) < H(X) + 2\,.$$

Analysis of arithmetic coding (cont.)

Put $\lfloor \overline{F(x)} \rfloor_{l(x)} = 0.y_1 y_2 \ldots y_{l(x)}$, then $\lfloor \overline{F(x)} \rfloor_{l(x)} < F(x)$.

$$\overline{F(x)} - \lfloor \overline{F(x)} \rfloor_{l(x)} < \frac{1}{2^{l(x)}} = \frac{1}{2^{\lceil \log_2 \frac{1}{p(x)} \rceil + 1}}$$

$$\leq \frac{1}{2^{\log_2 \frac{1}{p(x)}} \cdot 2} = \frac{p(x)}{2} = \overline{F(x)} - F(x-1)$$

Hence $\lfloor \overline{F(x)} \rfloor_{l(x)} > F(x-1)$, i.e.

$$\lfloor \overline{F(x)} \rfloor_{l(x)} \in (F(x-1), F(x)).$$

$$F(x) - \lfloor \overline{F(x)} \rfloor_{l(x)} \geq F(x) - \overline{F(x)} = \frac{p(x)}{2} \geq \frac{1}{2^{l(x)}}$$

$$\langle \lfloor \overline{F(x)} \rfloor_{l(x)}, \lfloor \overline{F(x)} \rfloor_{l(x)} + \frac{1}{2^{l(x)}}) \subseteq (F(x-1), F(x))$$

Analysis of arithmetic coding (cont.)

It follows that $x \neq y \implies$

$$\langle \lfloor \overline{F(x)} \rfloor_{l(x)}, \lfloor \overline{F(x)} \rfloor_{l(x)} + \frac{1}{2^{l(x)}}) \cap \langle \lfloor \overline{F(y)} \rfloor_{l(y)}, \lfloor \overline{F(y)} \rfloor_{l(y)} + \frac{1}{2^{l(y)}})$$

$$= \emptyset$$

Consequently, C is a prefix code!

$$L(C) = \sum p(x) \cdot l(x) = \sum p(x)(\lceil \log_2 \frac{1}{p(x)} \rceil + 1)$$

$$< \sum p(x)(\log_2 \frac{1}{p(x)} + 2) = \sum p(x) \cdot \log_2 \frac{1}{p(x)} + 2$$

$$= H(X) + 2$$

Analysis of a probability model

In our description of arithmetic coding, we used the following assumption:

C is an encoding for a random vector $\mathbf{X} = (X_1,\ldots,X_m)$ such that $X_1,\ldots,X_m$ are

- independent random variables
- with equal probability distribution

For $\mathbf{X} = (X,\ldots,X)$ we then have

$$\mathrm{H}(\mathbf{X}) \leq L(\mathrm{C}) \leq \mathrm{H}(\mathbf{X}) + 2$$

$$\mathrm{H}(X) \leq L(\mathrm{C}) / m \leq \mathrm{H}(X) + 2 / m$$

Probabilistic model: a better solution

$$p(x_1\, x_2 \ldots x_m) = p(x_1)\, p(x_2) \ldots p(x_m)$$

$$p(x_1 x_2 \ldots x_m) = p(x_1)\frac{p(x_1 x_2)}{p(x_1)}\frac{p(x_1 x_2 x_3)}{p(x_1 x_2)}\cdots\frac{p(x_1 x_2 \ldots x_m)}{p(x_1 x_2 \ldots x_{m-1})}$$

$$= p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_1 x_2) \ldots p(x_m \mid x_1 x_2 \ldots x_{m-1})$$

✎ **Problem.** How to estimate $p(x_i \mid x_1\, x_2 \ldots x_{i-1})$?

☞**Solution.** Assume that

$$p(x_i \mid x_1\, x_2 \ldots x_{i-1}) = p(x_i \mid x_{i-k}\, x_{i-k+1} \ldots x_{i-1})$$

Model with a finite context of order k

Entropy of English: experimental estimate

English text – string over alphabet A = {26 letters}
∪ {space}

C.Shannon (1951)

# attempts	1	2	3	4	5	>5
frequency	79 %	8 %	3 %	2 %	2 %	5 %

T.M.Cover, R.C.King (1978)

Conclusion: 1.3b/symbol

For a comparison

- English text of bible (`bible.txt`, cca 4MB)
- PPMZ (C. Bloom): 1.47b/symbol

Data Compression Algorithms

Context methods

Matt Mahoney
Florida Institute of Technology

Data compression: 2 stages

① Modeling

② Coding

Zero-order model

- probabilities of occurrences of isolated symbols
- no correlation between adjacent symbols

Finite context methods

Probability of a symbol depends

- not only on its frequency
- but also on the contexts in which it has occurred

Originally designed for text compression

Model of order i - makes use of lengths i contexts

Finite context methods

Methods

- with a fixed length contexts
- combined – use contexts of various lengths
  - » complete (all contexts of lengths $i, i\text{-}1, \ldots, 0$)
  - » partial
- (static), adaptive

PPM - Prediction by Partial Matching

Cleary, Witten (1984) Moffat (1990)

- combines context modeling & arithmetic coding
- combined model of order i

Given a symbol s and context c

- determine $f(s|c)$ = frequency of symbol s in context c

PPM

Symbol s encoding:

```
Let c be a context of length i
read(s)
if f(s|c)>0 : encode s using f(…|c)
else        : output(code(ESC|c))
              try order i-1 context
```

Necessary assumption

- there must be an i such that f(s | c) is defined for all contexts of order i

Order $k = 2$					Order $k = 1$					Order $k = 0$					Order $k = -1$				
Predictions			c	p	Predictions			c	p	Predictions			c	p	Predictions			c	p
ab	$\rightarrow$	r	2	$\frac{2}{3}$	a	$\rightarrow$	b	2	$\frac{2}{7}$	$\rightarrow$		a	5	$\frac{5}{16}$	$\rightarrow$		A	1	$\frac{1}{\lvert A\rvert}$
	$\rightarrow$	Esc	1	$\frac{1}{3}$		$\rightarrow$	c	1	$\frac{1}{7}$	$\rightarrow$		b	2	$\frac{2}{16}$					
						$\rightarrow$	d	1	$\frac{1}{7}$	$\rightarrow$		c	1	$\frac{1}{16}$					
ac	$\rightarrow$	a	1	$\frac{1}{2}$		$\rightarrow$	Esc	3	$\frac{3}{7}$	$\rightarrow$		d	1	$\frac{1}{16}$					
	$\rightarrow$	Esc	1	$\frac{1}{2}$						$\rightarrow$		r	2	$\frac{2}{16}$					
					b	$\rightarrow$	r	2	$\frac{2}{3}$	$\rightarrow$		Esc	5	$\frac{5}{16}$					
ad	$\rightarrow$	a	1	$\frac{1}{2}$		$\rightarrow$	Esc	1	$\frac{1}{3}$										
	$\rightarrow$	Esc	1	$\frac{1}{2}$															
					c	$\rightarrow$	a	1	$\frac{1}{2}$										
br	$\rightarrow$	a	2	$\frac{2}{3}$		$\rightarrow$	Esc	1	$\frac{1}{2}$										
	$\rightarrow$	Esc	1	$\frac{1}{3}$															
					d	$\rightarrow$	a	1	$\frac{1}{2}$										
ca	$\rightarrow$	d	1	$\frac{1}{2}$		$\rightarrow$	Esc	1	$\frac{1}{2}$										
	$\rightarrow$	Esc	1	$\frac{1}{2}$															
					r	$\rightarrow$	a	2	$\frac{1}{3}$										
da	$\rightarrow$	b	1	$\frac{1}{2}$		$\rightarrow$	Esc	1	$\frac{1}{3}$										
	$\rightarrow$	Esc	1	$\frac{1}{2}$															
ra	$\rightarrow$	c	1	$\frac{1}{2}$															
	$\rightarrow$	Esc	1	$\frac{1}{2}$															

abracadabra

PPM – details

How to define f($s|c$)?

- # occurrences of symbol s in context c
- # cases in which context c was used to predict s

Exclusion principle

- x occurs in context abc for the 1st time
- f($y|abc$)>0 $\Rightarrow$ y may be excluded from the 2nd order model
- empirical results
  - » 5% compression gain

Problem: P(esc)=?

PPMA (Cleary,Witten)

- context c satisfies $f(c) = n$
- $\Rightarrow$ $P(esc|c)=1/(n+1)$

PPMB

- $f(z|c)' = f(z|c)-1$
- $abcx...abcx....abcy$
- $f(x|abc)' = 1$, $f(y|abc)'=0$, $f(esc|abc)'=2$
- $\Rightarrow$ $P(esc|c)=2/n$

Problem: P(esc)=?

PPMC (Moffat)

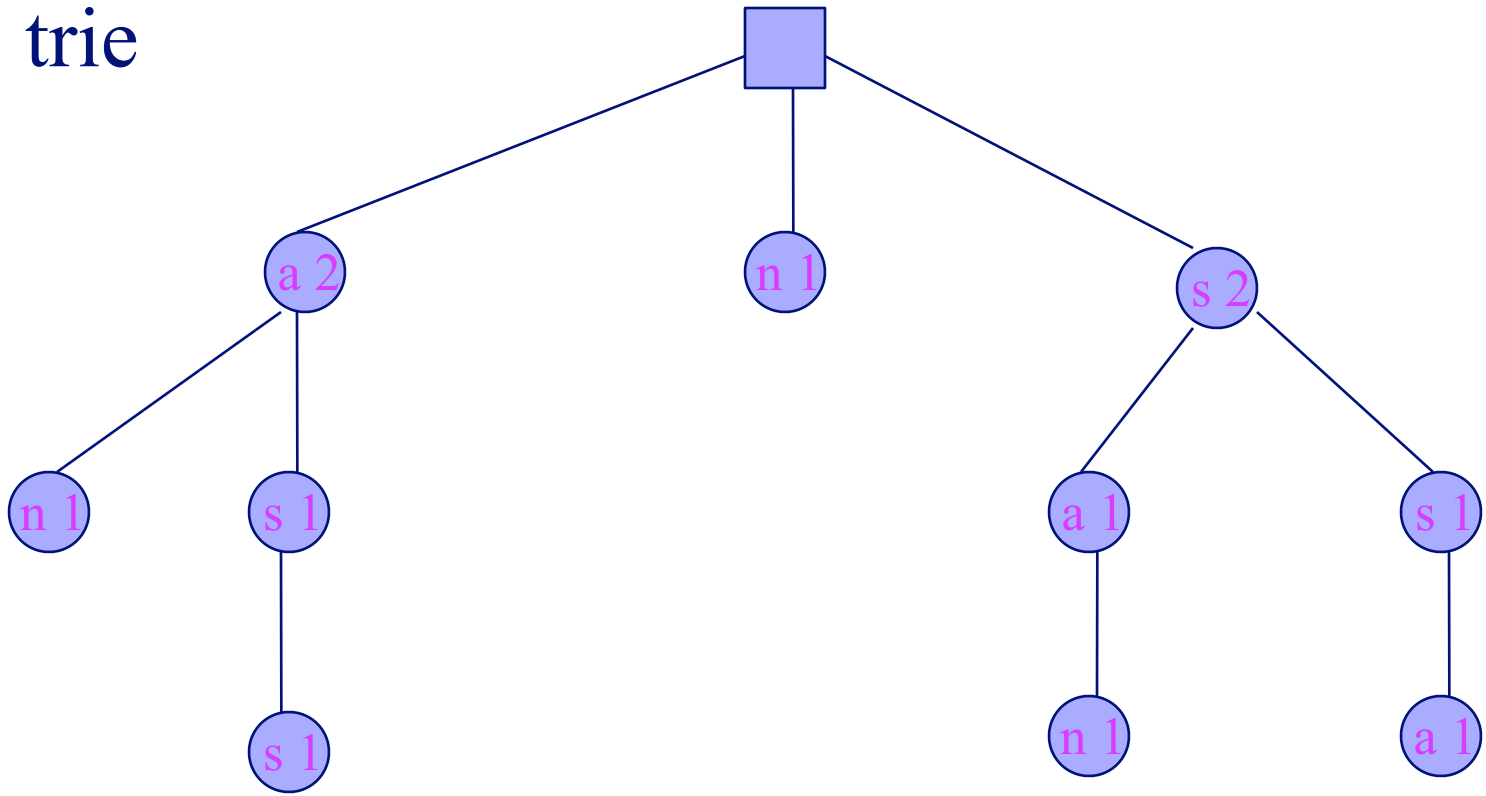
- $f(esc|c) := |\{x \mid f(x|c)>0\}|$
- $\Rightarrow$ $P(esc|c) = f(esc|c)/(n+f(esc|c))$

PPMD (P.G.Howard, J.Vitter)

- $f(esc|c) := |\{x \mid f(x|c)>0\}| / 2$
- $f(x|c)$:= 1st occurrence of x is assigned weight 1/2
  remaining occurrences weight 1

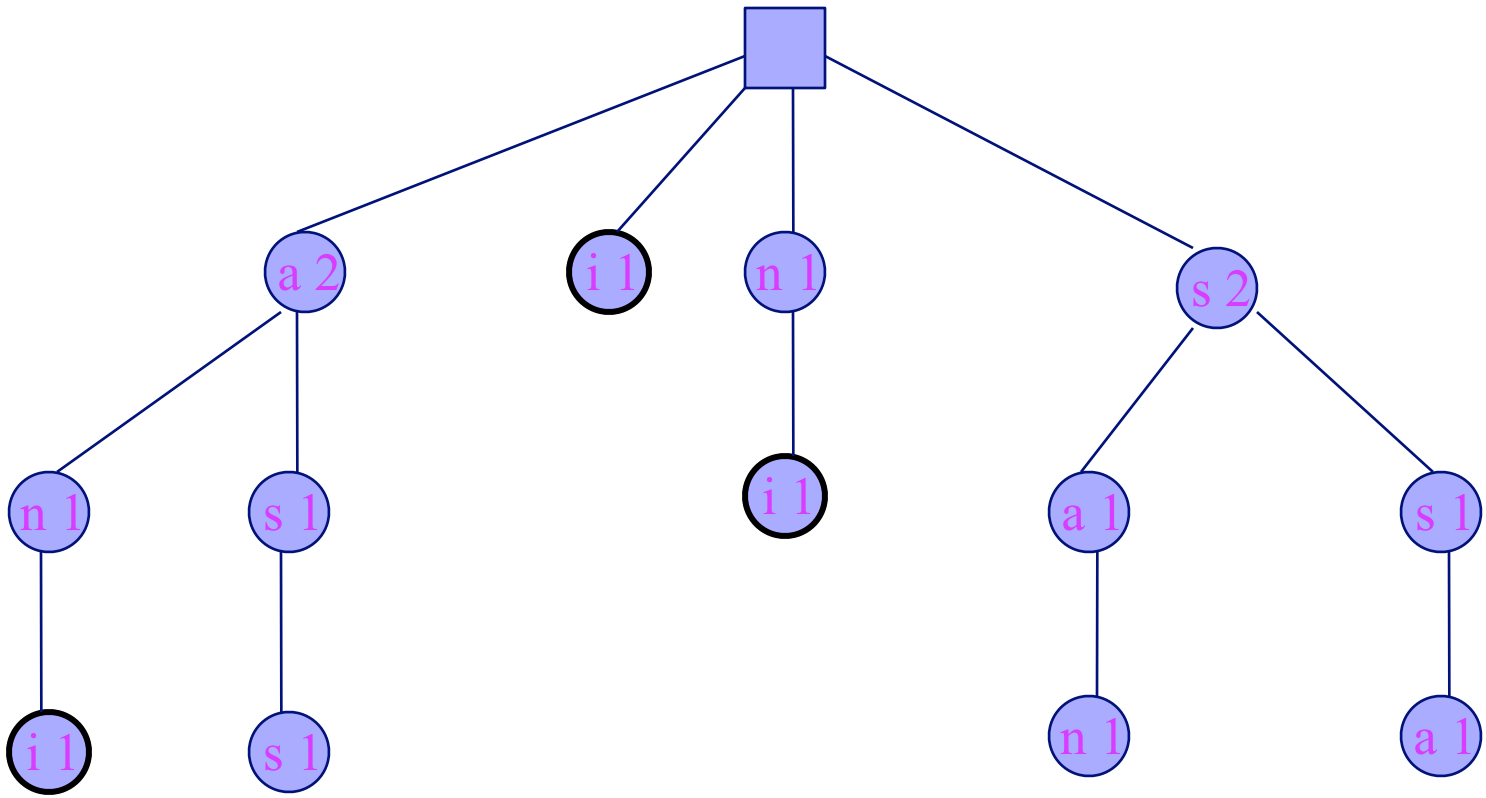
Data structures: context tree

trie



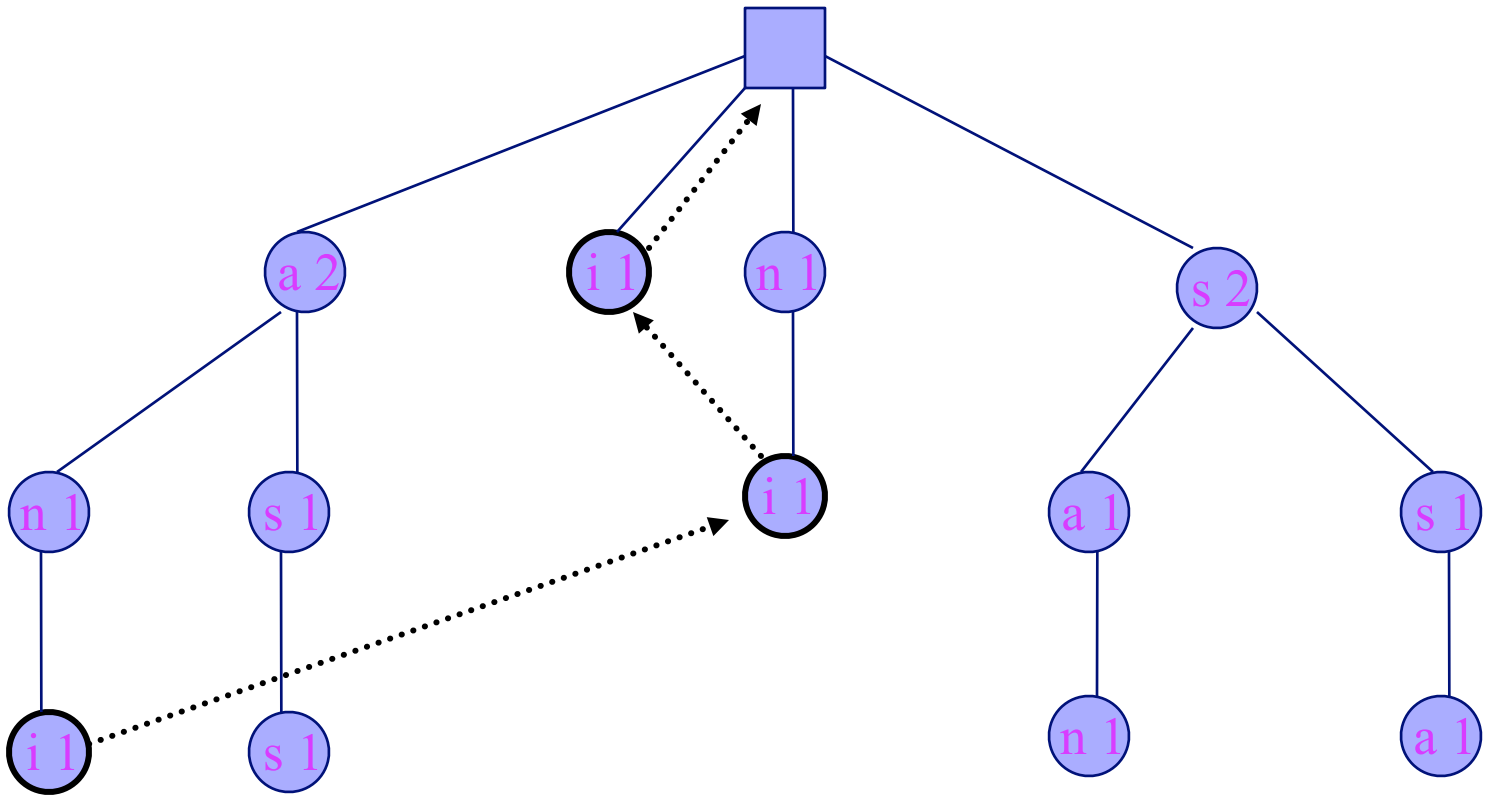
assan

Data structures: context tree



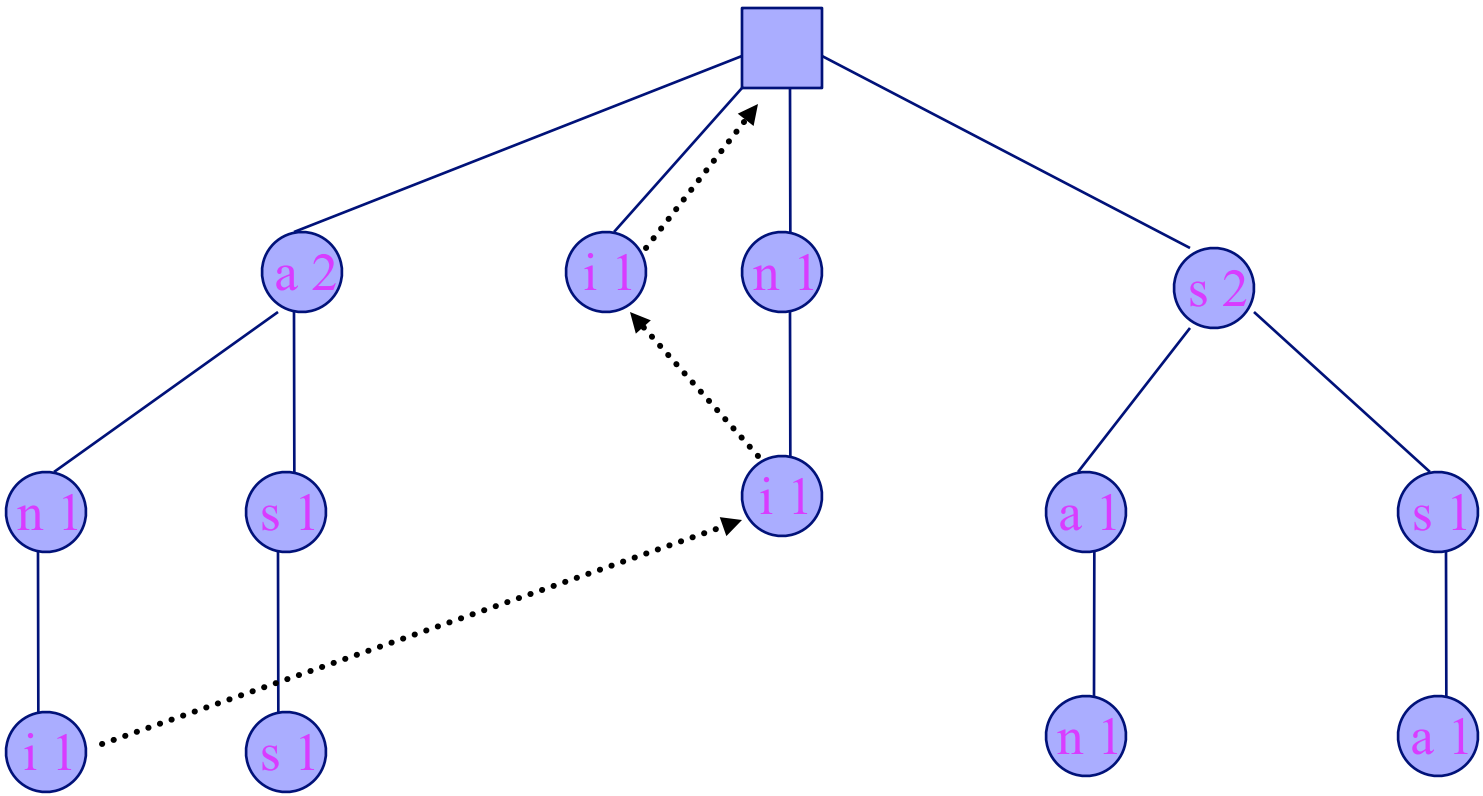
assani

Data structures: context tree



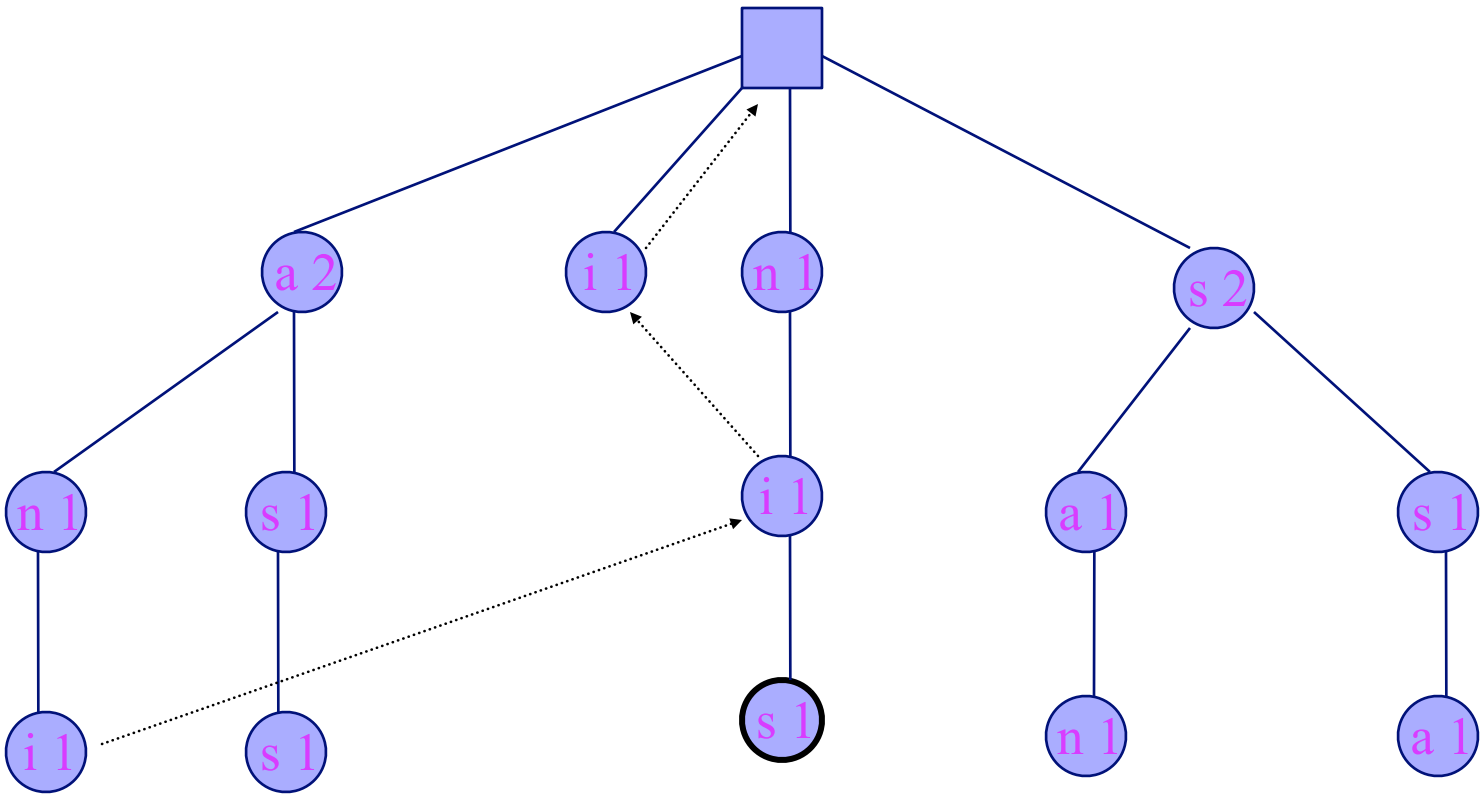
assani

Data structures: context tree



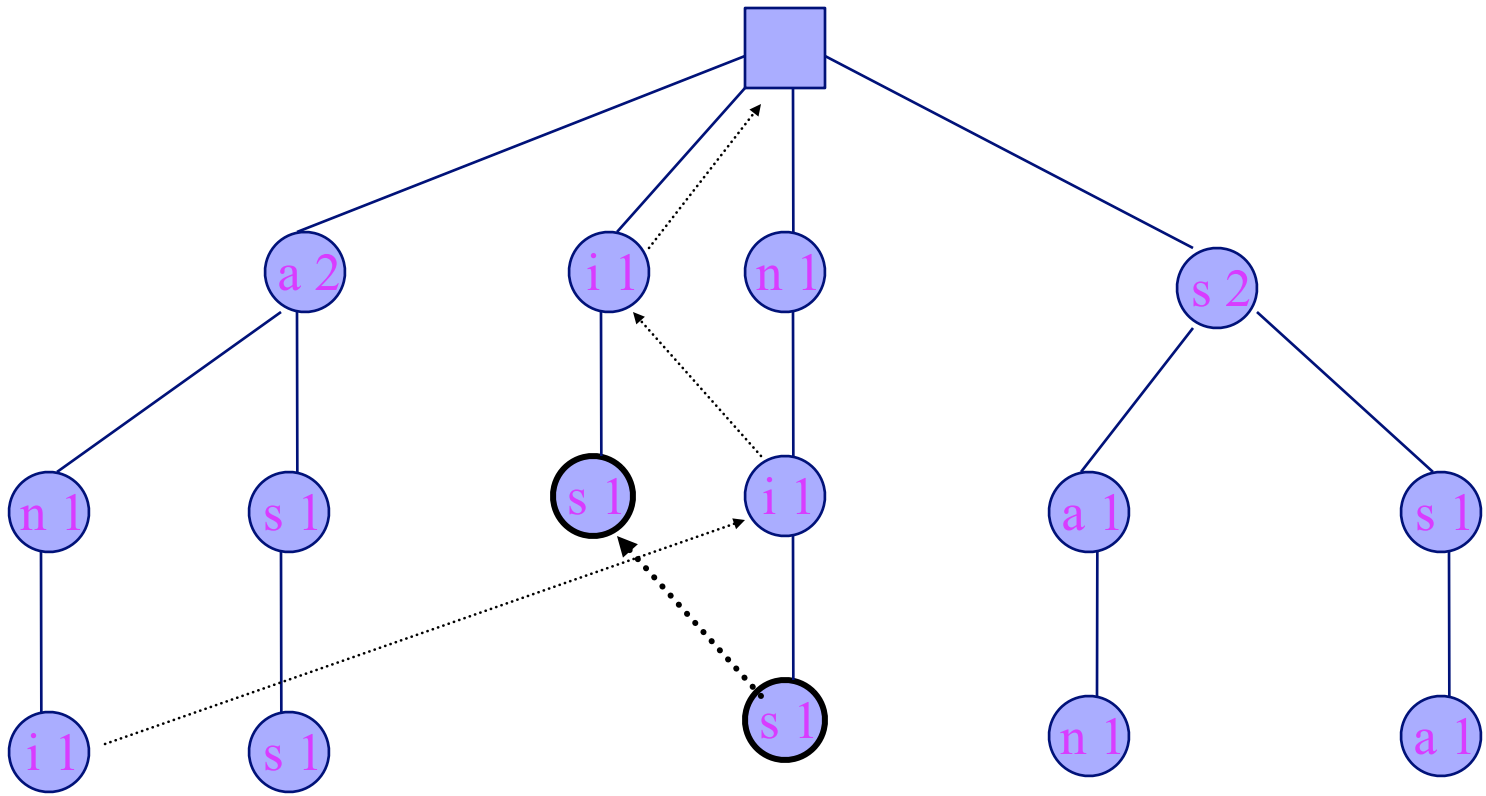
assanis

Data structures: context tree



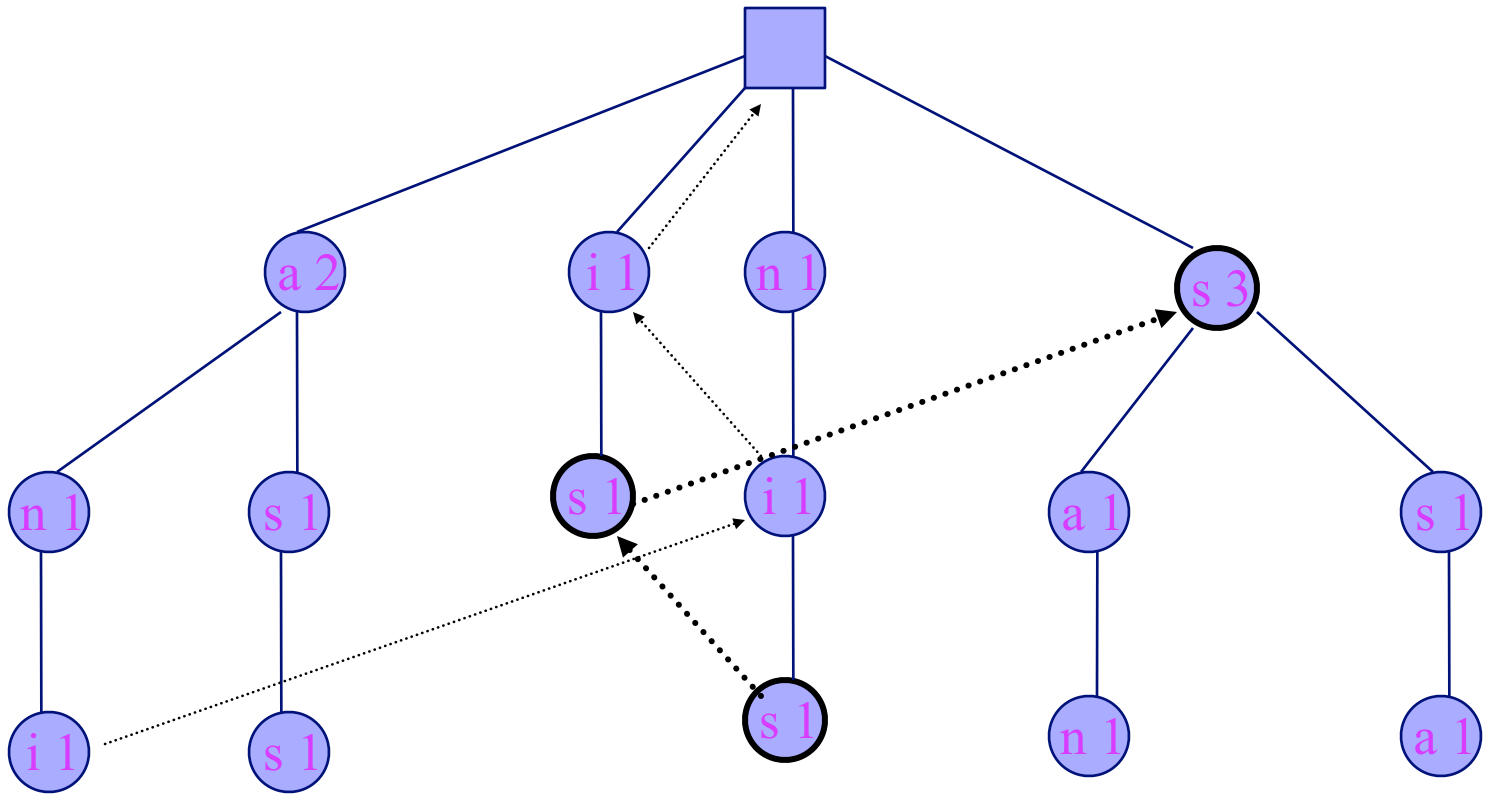
assanis

Data structures: context tree



assanis

Data structures: context tree



assanis

Space limitations

Monitor the free memory size

If it falls below a certain bound T

- $\Rightarrow$ freeze the model
  - » update frequencies in existing contexts
  - » ignore new contexts
- $\Rightarrow$ rebuild the model
  - » initialize with buffered history

Space limitations

Improvement

- monitor also the relative compression ratio
- if it starts decreasing $\Rightarrow$ *rebuild* the model

Advanced data structure: DAWG

- Directed Acyclic Word Graph
- equivalent contexts are identified

Experimental results (Moffat,Turpin,2002)

order	space (MB)						
	1	2	4	8	16	32	64
1	3,38						
2	2,44						
3	1,91	1,90					
4	**1,85**	**1,71**	**1,66**				
5	2,02	1,81	1,67	**1,60**	**1,58**		
6	2,18	1,96	1,78	1,66	1,59	**1,56**	
7	2,31	2,09	1,90	1,76	1,66	1,58	1,56

Units: b/symbol

File: bible.txt

Method: PPMD

Model rebuilding after free memory was exhausted

Other modifications

PPMII (D.Škarin, 2002)

- *PPM: One Step to Practicality*, DCC '02
- information inheritance
- contexts inherit from their ancestors
- three context classes with various models
- scaling of frequencies
- RAR

Other modifications

PAQ

- Matt Mahoney (PAQ1, 2002)
  - » PAQ8PX, 2009, Jan Ondruš
- binary alphabet
- context model + arithmetic coder
- context mixing
  - » weighted average of probability estimates from various models

Data Compression Algorithms

Integer Coding

Peter Elias
(1923 – 2001)

Integer coding

Encode a message $m \in A^*$

$A = \{1,2,\ldots,n\}$

$p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n$

$A = \{1,2,\ldots\}$

$p_1 \geq p_2 \geq p_3 \geq \ldots$

Unary code

Unary code α $\alpha(1)=1,\ \alpha(i+1)=0\ \alpha(i)$

Properties

- $|\alpha(i)| = i$

Optimal code for probability distribution

- $A = \{1,2,\ldots\}$
  - » $p(i)=1/2^i,\ \ i = 1,2,\ldots$
- $|A| = n$
  - » $p(i)=1/2^i,\ \ i = 1,2,\ldots,n-1$
  - » $p(n) = 1/2^{n-1}$

Binary code

Binary code β

$\beta(1)=1$, $\beta(2i)=\beta(i)0$, $\beta(2i+1)=\beta(i)1$

- $(\beta(0)=0)$

Properties

- $|\beta(i)| = \lfloor \log_2 i \rfloor + 1$
- β is not a prefix code!

Binary code

Binary code of fixed length

$|A| = n \Rightarrow$ fixed codeword length

- $\lceil \log_2 n \rceil$ bits

Optimal code for probability distribution

- $p(i)=1/n, \qquad i = 1,2,\ldots,n$

Binary code of increasing length

- n = current dictionary size
- dictionary size $\uparrow \Rightarrow$ codeword length $\uparrow$

Minimal binary code

What if $n = |A|$ is not a power of 2?

Minimal binary code β_n

- $i < 2^k - n \quad \Rightarrow$ $(k-1)$ bit binary code of i
- otherwise $\Rightarrow$ k bit binary code of $i+2^k-n$

int	code
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	1110
8	1111

Elias codes: an elegant compromise

Peter Elias (1975)

Reduced binary code β'

- $\beta'(i) = \beta(i)$ without the 1st bit
- $|\beta'(i)| = \lfloor \log i \rfloor$

$\gamma'(i) = \alpha(|\beta(i)|)\ \beta'(i)$

- $|\gamma'(i)| = 2\lfloor \log i \rfloor + 1$

$\gamma(i)$ = a permutation of $\gamma'(i)$

- bits of α interleave with bits of β'

Elias codes

$\delta(i) = \gamma(|\beta(i)|)\ \beta'(i)$

- $|\delta(i)| = 1 + \lfloor \log_2 i \rfloor + 2\lfloor \log_2(1 + \lfloor \log_2 i \rfloor) \rfloor$

Properties

- γ, γ' are long for large *i*
- δ is shorter for large *i*

Construction may continue analogically

- difference only for large *i*
- in practice δ suffices
- $|\gamma(10^9)| = 59$, $|\delta(10^9)| = 39$, $|\varepsilon(10^9)| = 39$

Elias codes: properties

Elias codes are **universal**

☀ **Observation**

Each probability distribution $p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n$ satisfies $p_i \leq 1/i$ pro $i = 1,2,\ldots,n$.

☞ **Corollary**: $-\log_2 p_i \geq -\log_2 (1/i) = \log_2 i$

Elias codes universality

- $|\gamma(i)| = \log_2 i + \Theta(\log_2 i)$
- $|\delta(i)| = \log_2 i + o(\log_2 i)$

Data Compression Algorithms

Dictionary methods

Phillip Walter Katz
(1962 - 2000)

Dictionary methods

☀ Idea

- repeated phrases stored in a dictionary
- phrase occurrence in the text → pointer

Problems and their solutions used in practice:

- construction of an optimal dictionary is NP-hard ⇒ greedy algorithm
- dictionary needed for decoding ⇒ dynamic methods

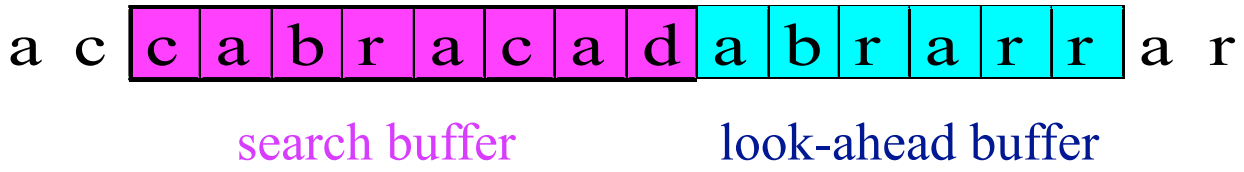
Jacob Ziv, Abraham Lempel

[LZ77] A universal algorithm for data compression. *IEEE Trans. Inform. Theory*, IT-23(3):337-343, 1977.

[LZ78] A compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory*, IT-24(5):530-536, 1978.

LZ77

Sliding window



Initialization

- search buffer filled with spaces
- input is read into the look-ahead buffer

Search for the longest string S such that

- S starts in the search buffer
- S matches a prefix of the string in the look-ahead buffer

LZ77

Sliding window

a	c	c	a	b	r	a	c	a	d	a	b	r	a	r	r	a	r

search buffer look-ahead buffer

String S is encoded as a triple $\langle i,j,z \rangle$ where

- i is the distance of the beginning of S from the look-ahead buffer
- $j = |S|$
- z is the symbol following S in the look-ahead buffer

offset

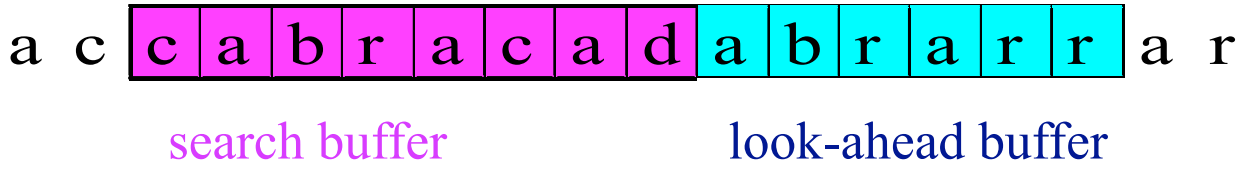
match length

$\langle 7,4,\mathrm{r} \rangle$

Window slides $j+1$ symbols to the right

LZ77

Sliding window



Sliding window size 2^{12} - 2^{16} B

- gzip 2^{15} B

Look-ahead buffer size tens - hundreds B

- gzip 256 B

Example

a c c a b r a c a d a b r a r r a r r a d

$\langle 0,0,d \rangle$

a c c a b r a c a d a b r a r r a r r a d

$\langle 7,4,r \rangle$

a c c a b r a c a d a b r a r r a r r a d

$\langle 3,5,d \rangle$

Example: decoding

a c c a b r a c a

$\langle 0,0,d \rangle$, $\langle 7,4,r \rangle$, $\langle 3,5,d \rangle$

a c c a b r a c a d

$\langle 7,4,r \rangle$, $\langle 3,5,d \rangle$

a c c a b r a c a d

a c c a b r a c a d a b r a r

Example: decoding

a c c a b r a c a d a b r a r

$\langle 3,5,d \rangle$

a c c a b r a c a d a b r a r r

a c c a b r a c a d a b r a r r a

a c c a b r a c a d a b r a r r a r

a c c a b r a c a d a b r a r r a r r

a c c a b r a c a d a b r a r r a r r a

a c c a b r a c a d a b r a r r a r r a d

Properties

Slow compression, fast decompression

Compression speed may be increased

- using special data structures
- $\Rightarrow$ higher storage demands

Typical application

- single compression
- multiple decompression

LZSS variants

Storer, Szymanski (1982)

- triple (i,j,z) replaced with (i,j) or literal z
- bit indicator to distinguish: pair of pointers × symbol
- if pair of pointers requires the same space as p symbols
  - » use pairs to encode only phrases of length > p
  - » otherwise copy symbols to the output
- sliding window → cyclic buffer [0..N]

b	c	b	a	c	b	a	b	a	b	c	x	y	z
5	6	7	8	9	10	0	1	2	3	4			

N=10

- possible implementation
  - » codeword 16b, $|i| = 11$, $|j|=5$
  - » 8 bit indicators in 1B

LZ77 variants

LZB (Bell, 1987)

- LZSS with more efficient encoding of (i,j)
- $i \rightarrow$ binary code with increasing length
- $j \rightarrow \gamma$ code

LZ77 variants

LZH (Brent, 1987)

- LZSS with Huffman - encoded pointers
- 1st pass: LZSS + symbol counts
- 2nd pass: static Huffman code

Problem of large alphabet

- i in (i,j) split into two parts (each with max value $\sqrt{N}$)
- one frequency table for items of all four types

What do they have in common?

.zip

.gz

.jar

.cab

.png

Deflate algorithm

- by Phil Katz, 1991
- originally for PKZIP 2 (1993)
  - » open format
  - » standard for file compression on various platforms
- LZSS + static Huffman coding

Deflate

Phil Katz (1991)

- Jean-loup Gailly, Mark Adler: zlib (1996)

http://www.gzip.org/zlib/rfc-deflate.html

- LZSS + Huffman coding
- search buffer size 32kB
- look-ahead buffer size 258B
  - » match length 3..258

Input divided into blocks, 3b header

- 1 bit: last block YES/NO
- 2 bits: block type

Deflate

3 block types

- 1: no compression
- 2: fixed coding tables for Huffman code
- 3: Huffman code based on input data statistics

Type 3 block

- starts with two Huffman trees
- one for literals (0..255) and match lengths (3..258)
  - » unique alphabet 0..285
  - » 0..255 for literals, 256 = end-of-block
  - » 257..285 for match length (+ extra bits)
- other for offset

Deflate: match length

Code	Extra Bits	Length(s)	Code	Extra Bits	Lengths	Code	Extra Bits	Length(s)
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

Deflate: offset

Code	Extra Bits	Dist	Code	Extra Bits	Dist	Code	Extra Bits	Distance
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

Deflate

String searching in search buffer → hashing

- hash table stores strings of length three
- strings in linked lists sorted by their age
- a parameter to limit the maximum list length

Greedy strategy extension

- first look for a primary match
- slide window one symbol to the right, find a secondary match, if better, encode as literal + secondary match
- fast mode: if primary match is satisfactory, secondary match test is waived

Compression ratio 30 - 40 % for text files

LZ77: disadvantages

Limited outlook of the search window

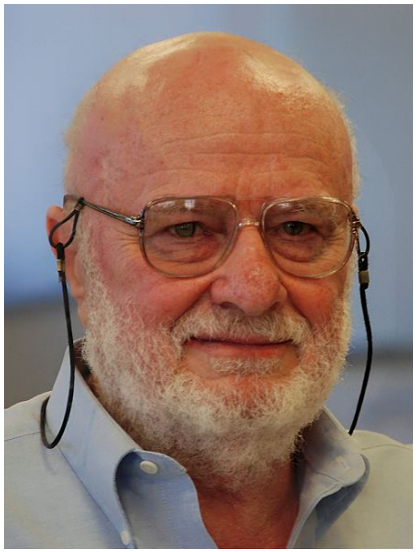
- longer window =>
  - » better compression,
  - » more complex search

Limited length of the look-ahead buffer

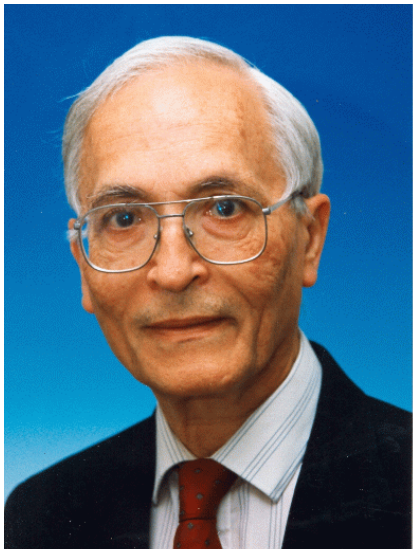
- match length limitation

Data Compression Algorithms

Dictionary methods: LZ78



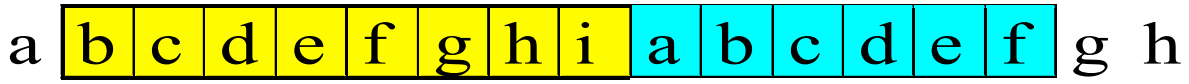
Abraham Lempel
1936 – 2023



Jacob Ziv
1931 – 2023

LZ78

Achilles' heel of LZ77:



Jacob Ziv, Abraham Lempel (1978)

Sliding window → explicit dictionary

- repeated phrases stored in a dictionary
- phrase occurrence in the text → pointer to dictionary
- dictionary not transmitted

LZ78

① initial dictionary empty

② read the longest prefix of the input
which matches a phrase f in the dictionary

③ output $\langle i, \mathrm{k}(s) \rangle$

- i points into the dictionary to phrase f
- $\mathrm{k}(s)$ is a codeword for symbol s
  which follows f in the input

④ insert new phrase fs into the dictionary

Example

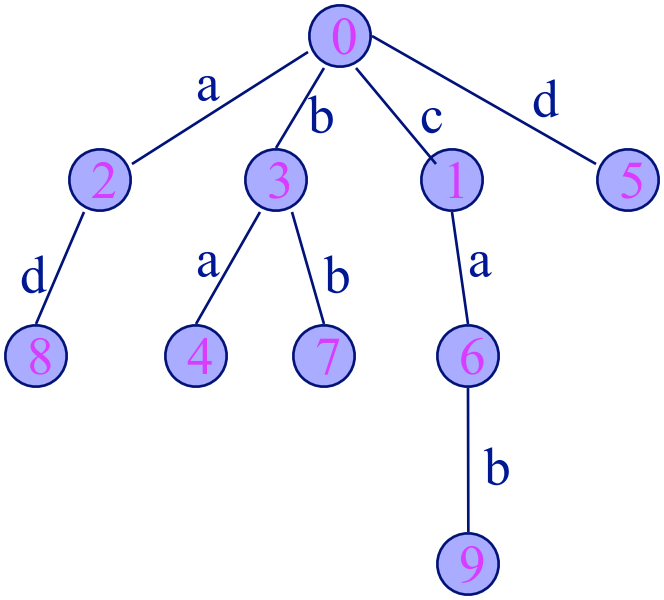
cabbadcabbadcab

$\langle 0,c\rangle\ \langle 0,a\rangle\ \langle 0,b\rangle\ \langle 3,a\rangle\ \langle 0,d\rangle\ \langle 1,a\rangle\ \langle 3,b\rangle\ \langle 2,d\rangle\ \langle 6,b\rangle$

index	phrase
1	c
2	a
3	b
4	ba
5	d
6	ca
7	bb
8	ad
9	cab

LZ78 - data structures

Dictionary → trie



index	phrase
1	c
2	a
3	b
4	ba
5	d
6	ca
7	bb
8	ad
9	cab

Dictionary reconstruction

Necessary when the dictionary space is exhausted

- delete the whole dictionary
- delete the phrases that
  - » occurred least frequently (LFU)
  - » have not occurred recently (LRU)

Need to preserve the prefix property

- if the dictionary $\ni$ phrase f
- then dictionary $\ni$ phrase all prefixes of f

Improvement

- if compression ratio starts getting worse $\Rightarrow$
- delete the dictionary and start over from the initial setting

LZ78 – decoding

$\langle 0,a \rangle \langle 0,b \rangle \langle 1,b \rangle \langle 3,a \rangle$

LZW

Terry Welch (1984)

LZ78 - citation & innovation

Innovation

- to encode the symbol following the phrase in the dictionary
- log n bits are needed
- n = alphabet size
- independent on the input

LZW = LZ78 with delayed innovation

LZW

Algorithm

- initialization
  - » insert all alphabet symbols into the dictionary
- read the longest prefix of the input which matches a phrase f in the dictionary
- output(index(f))
- fs where s is the symbol following f in the input is inserted into the dictionary

Example

Input: abcabcabcbcba

index	phrase
1	a
2	b
3	c

Example

Input: abcabcabcbcba

Output: 1 2 3 4 6 5 9 1

index	phrase
1	a
2	b
3	c
4	ab
5	bc
6	ca
7	abc
8	cab
9	bcb

LZW - decoding

<u>Code</u>: 1 2 3 4 6 5 9 1

index	phrase
1	a
2	b
3	c
4	ab
5	bc
6	ca
7	abc
8	cab
9	bcb

Problem

- input string *a*w*a*w*a*
- *a* - symbol, w - string
- *a*w is in the dictionary
- *a*w*a* is not in the dictionary

Solution

- missing phrase is *a*w*a*
- *a*w is the last decoded phrase

✎ Problem

- find the shortest input for which this occurs

LZC

compress 4.0 (Thomas et al, 1985)

- modified LZW

Pointers into dictionary

- binary code with increasing length
- 9 - 16 b

Max size is reached

- $\rightarrow$ compression with static dictionary

Compression ratio monitoring

- getting worse $\Rightarrow$ delete the dictionary and start over with the initial setting

LZMW

Miller, Wegman (1984)

☀ Idea

- new phrase = concatenation of *two* last ones
- phrase length increases faster

Dictionary full ⇒ LRU strategy

Prefix property fails for the dictionary

- this complicates searching
- ⇒ backtracking

LZAP

Storer (1988)

☀ Idea

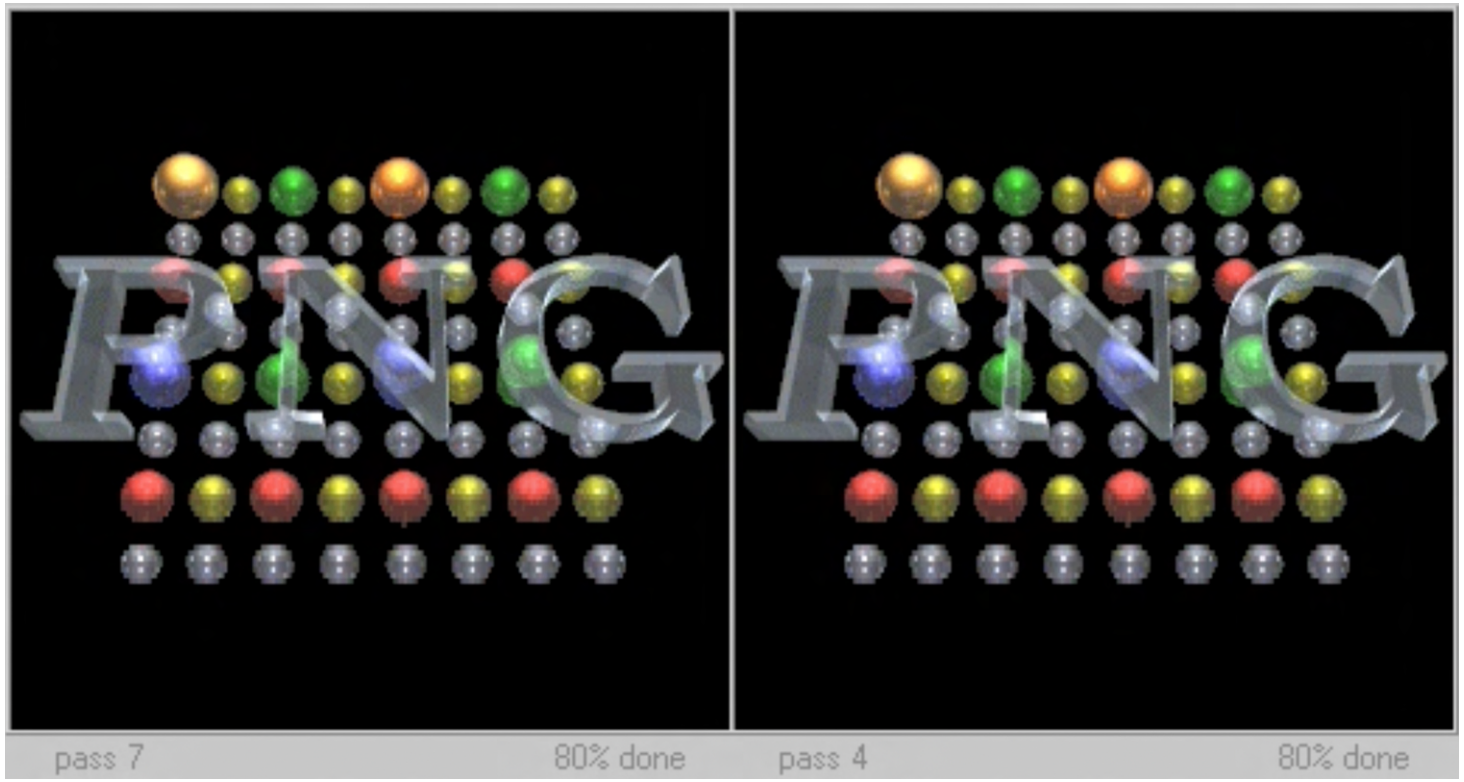
- instead of ST add all substrings Sp
- where p is a prefix of T

☞ Properties

- larger dictionary ⇒
  - » longer codewords
  - » wider choice for phrase selection
- faster searching
  - » backtracking not needed

Data Compression Algorithms

Lossless image compression



interlacing schemes: PNG & GIF

Image file formats

Digital images

- raster (bitmap)
- vector

Color spaces

- RGB
  - » red, green, blue
  - » (r,g,b), $r,g,b \in \langle 0,255 \rangle$
  - » additive composition of colors
  - » display of images in electronic systems (TV, PC)
- CMY (CMY K)
  - » cyan, magneta, yellow (black)
  - » subtractive composition
  - » color printing

Image file formats

Color spaces

- HSV (HLS)
  - » hue – dominant spectral color
  - » saturation – admixture of other colors
  - » value – brightness (lightness)
- YUV (YIQ, YC_BC_R)
  - » Y – luma (brightness)
  - » UV – chrominance components
  - » TV signal

Representation of bitmap images

Monochrome

- pixel = 1b

Shades of gray

- e.g. pixel ≈ 1B

Color

- pixel ≈ (r,g,b)
- α-channel

Representation of bitmap images

Color palette

- conversion table
  - » pixel = pointer into the table
- *grayscale*: refers to palette with shades of gray
- *pseudocolor*: refers to color palette (RGB)
- *direct color*: refers (r↑,g↑,b↑)
  to three color palettes
  - » easy change of colors
    without changing the raster image

Lossless compression of raster images

GIF: Graphics Interchange Format

- Compuserve Information Services (1987)
  - » GIF87a
  - » GIF89a (multimedia)

Usage

- originally for image transmission over phone lines
- suitable for WWW
- sharp-edged line art with a limited number of colors
  - » logos
- small animations
- not used for digital photography

GIF

Color palette

- table: color $\leftrightarrow$ RGB
- pixel $\approx$ 1B $\Rightarrow$ max 256 colors

Properties

- more images in one file
- interlacing scheme for transmission
  - » rough image after 25-50% data
- textual information (89a)
- control elements (89a)
  - » animation delays

LZW based compression

GIF

LZW

- b bits/pixel, dictionary size $2^{b+1} - 2^{12}$
- no free memory $\Rightarrow$ static dictionary
- if compression ratio starts getting worse
  - » $\Rightarrow$ delete the dictionary and start over from the initial setting
  - » $\Rightarrow$ *clear code* 2^b

Pointers

- binary code with increasing length

Blocks

- up to 255B
- *end-of-block 00000000*
- last block: *end-of-file* 2^{b+1}

Format PNG

PNG: Portable Network Graphics

Motivation: replace GIF with a format based on a non-patented method

- 1983: Terry Welch filed a patent application for LZW
- 1985: Patent granted to Sperry Corporation who later formed Unisys
- 1994: Unisys asked all companies employing LZW to license the technology from Unisys
- 1996: PNG 1.0
- 2004: international standard ISO/IEC 15948:2004
- W3C support, suitable for network transmission

Format PNG

Color space support

- grayscale
  - » 1,2,4,8,16b
- true color
  - » 8-8-8, 16-16-16 RGB
- palette-indexed color
  - » 1,2,4,8b → 8-8-8 RGB
- α- channel
- no support for other systems (CMYK)
  - » designed for transferring images over the network
  - » not for print

PNG compression

Two phases

- preprocessing
- dictionary compression

Preprocessing: model of linear prediction

- predict the value of the pixel to be encoded
- encode the difference between the real and predicted value (= prediction error)

Prediction based on neighboring pixels

PNG – prediction

None

Sub $I'(i,j) = I(i,j-1)$

Up $I'(i,j) = I(i-1,j)$

Average $I'(i,j) = \lfloor (I(i,j-1) + I(i-1,j))/2 \rfloor$

Paeth $I'(i,j) = \mathrm{PaethPredictor}(I(i,j-1), I(i-1,j), I(i-1,j-1))$

- Alan W. Paeth (1991)
- PaethPredictor(a,b,c) returns the value of $\{a,b,c\}$ closest to $a+b-c$

Each line may be processed with a different method

Compression phase: Deflate method (LZ77)

PNG – properties

True color

- up to 16b color depth per one color component

2D interlacing scheme for transmission

Single image only, no animation

- MNG

α-channel

Thomas Porter, Tom Duf (1984)

Pixel → value in $\langle 0,1 \rangle$

- 0 = transparent, 1 = opaque
- (0,1,0) → (0,1,0,½)

PNG: α-channel

GIF: one palette index may represent
a transparent background color

Interlacing schemes

PNG: 7 passes

Adam M. Costello: Adam7

1 6 4 6 2 6 4 6

7 7 7 7 7 7 7 7

5 6 5 6 5 6 5 6

7 7 7 7 7 7 7 7

3 6 4 6 3 6 4 6

7 7 7 7 7 7 7 7

5 6 5 6 5 6 5 6

7 7 7 7 7 7 7 7

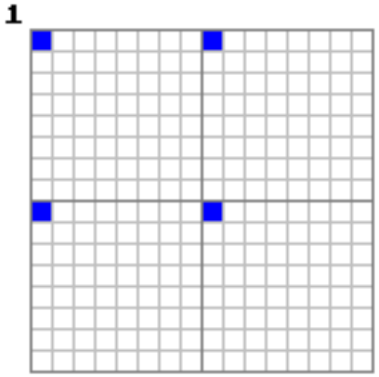
GIF: 4 passes

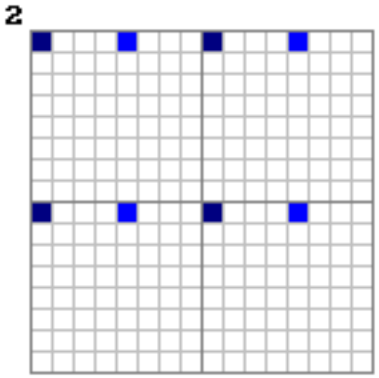
lines # 1,9,17,…

lines # 5,13,21,…

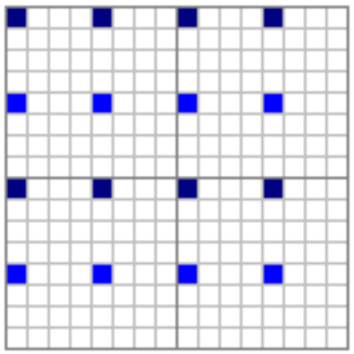
lines # 3,7,11,15,…

lines # 2,4,6,8,10,12,…

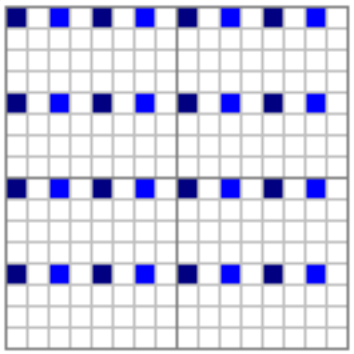

1


2

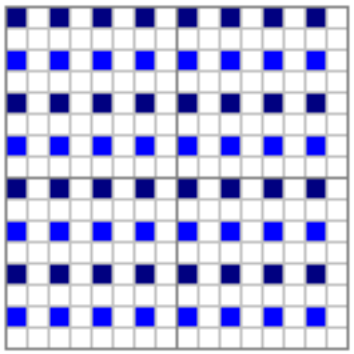
3



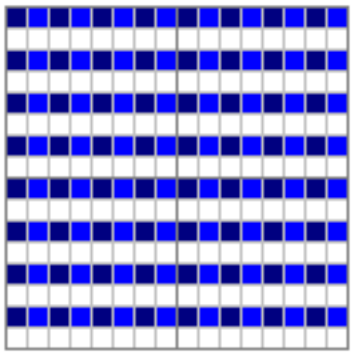
4

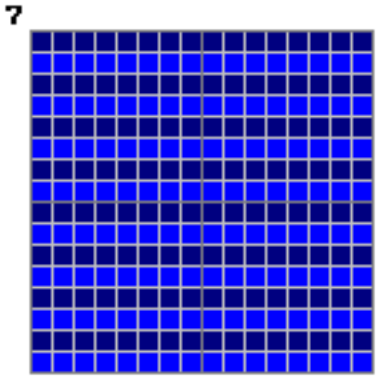


5

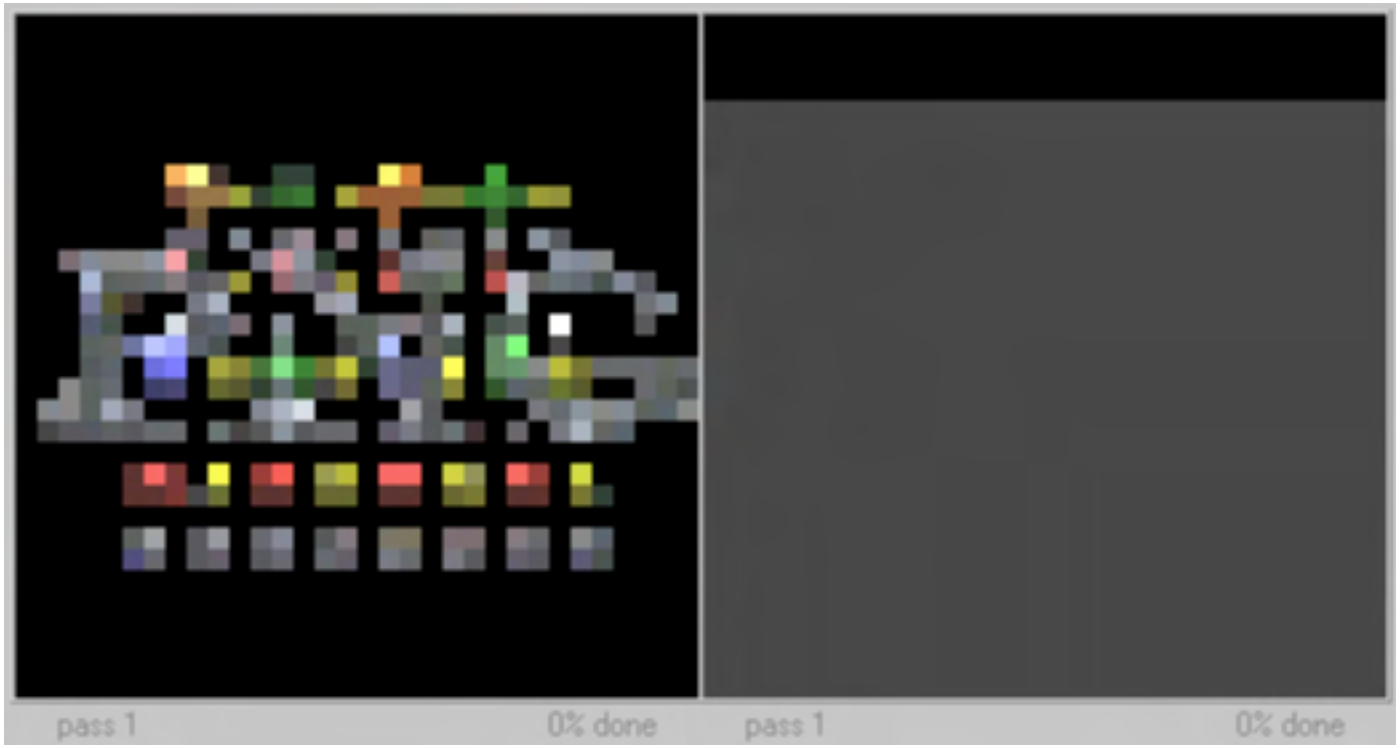


6




7

Interlacing schemes – comparison



Adam7 GIF89a

Data Compression Algorithms

Burrows-Wheeler transform

Michael Burrows
* 1963

David Wheeler
(1927 – 2004)

BSTW

Bentley, Sleator, Tarjan, Wei (1986)

Move to front heuristic to restructure the dictionary

Dictionary D is initially empty

```
s = input string
if s is in D on ith position :
      output(γ(i))
      move ith string in D
      to the front of D
else: output(γ(|D|+1)); output(s)
      insert s to the front of D
```

Burrows-Wheeler transform

M.Burrows, D.J.Wheeler, DEC, 1994

- David Wheeler invented the transformation in 1984 (unpublished)
- Michael Burrows (*1963)
  - » one of AltaVista creators
  - » now with Google
- Burrows & Wheeler (1994)
  - » published inverse transform

Burrows-Wheeler transform

String $\boldsymbol{x}$ of length n $\longrightarrow$ permutation L of string $\boldsymbol{x}$

- **if** L[i] = a
- **then** other occurrences of a in L
  are „close to“ the ith position

Matrix $n \times n$

- i^{th} row = cyclic shift of $\boldsymbol{x}$
  by i positions to the left
- sort rows lexicographically
- permutation L = last column of the sorted matrix

Example: $\boldsymbol{x}$ = `abrakadabra`

BWT: Example - *abrakadabra*

abrakadabra
brakadabraa
rakadabraab
akadabraabr
kadabraabra
adabraabrak
dabraabraka
abraabrakad
braabrakada
raabrakadab
aabrakadabr

aabrakadabr
abraabrakad
abrakadabra
adabraabrak
akadabraabr
braabrakada
brakadabraa
dabraabraka
kadabraabra
raabrakadab
rakadabraab

, 2

Encoding

Permutation L = last **column** of the sorted matrix

I = index of the row with the original string $\boldsymbol{x}$
in the sorted matrix

output(L,I) (`rdacraaaabb`,2)

✎ **Problem**: Does the last column satisfy our goal, i.e

- permutation of string S
- such that identical symbols occur close to each other?

Encoding

Move to front (MTF):

```
L = rdacraaaabb
A = ("a","b","c","d","r")
for i in range(|A|) :
 code(Li)= # preceding symbols in A
 move Li to the front of A
```

(RLE +) entropy encoding

- Huffman / arithmetic coding

Note: MTF alternatives

- IF (Arnavut & Magliveras, 1997)
- WFC (Deorowicz, 2001)

Decoding

	F	T	L
0	a	9	r
1	a	7	d
2	a	0	a
3	a	8	k
4	a	10	r
5	b	1	a
6	b	2	a
7	d	3	a
8	k	4	a
9	r	5	b
10	r	6	b

F = L sorted with a stable sorting algorithm

T[i] = position of L[i] in F

$F[T[i]] = L[i]$

decoder uses L, I and T to reconstruct $\boldsymbol{x}$

Decoding

```
for i in range(n) : x[n-1-i] = L[T_i[I]]
```

$T_0[j] = j$, $T_{i+1}[j]=T[T_i[j]]$

$L[i]$ precedes $F[i]$ in $\boldsymbol{x}$

$L[T[i]]$ precedes $F[T[i]]$ in $\boldsymbol{x}$

$F[T[i]]=L[i] \Rightarrow L[T[i]]$ precedes $L[i]$ in $\boldsymbol{x}$

$L[I]=L[2]=a$ (last symbol in $\boldsymbol{x}$)

$L[T[i]]= L[T[2]] =L[0]=r$

etc.

last but one symbol in $\boldsymbol{x}$

Application: bzip2

Julian Seward

- open source
- July 1996: version 0.15
- later maintained by
  - » Federico Mena (since 2019)
  - » Micah Snyder (since 2021)

bzip2 × gzip/ZIP

- more efficient compression
- slower
- only compression, no archiving (tar)

bzip2

Compression

- split the input into blocks of 100–900 kB
  - » determined by an input parameter
- Burrows – Wheeler transform (BWT)
- Move To Front heuristic
- Run Length Encoding
- arithmetic coding (patented)
  replaced with Huffman coding

Implementation notes

Method is efficient for large n

program	authors	block size	bible.txt (b/char)
Bzip2	J.Seward, J.Gailly (1999)	900kB	1.67
Szip	M.Schindler (1997)	4,3MB	1.53
PPMZ	Ch.Bloom		1.47

Implementation notes

Matrix $n \times n$ unnecessary

Sorting – suffix array

Suffix array for a string $\boldsymbol{x}$ is an array SA[0..n-1] such that

- SA[j] = i $\Leftrightarrow$ $\boldsymbol{x}[i:]$ is on j^{th} position
- in the lexicographic order of all suffixes of $\boldsymbol{x}$
- may replace rotations of the string $\boldsymbol{x}$\$

Example: suffix array for *abaabaab*

0	*abaaba*
1	*baabaa*
2	*aabaab*
3	*abaab*
4	*baab*
5	*aab*
6	*ab*
7	*b*

5	*aab*
2	*aabaab*
6	*ab*
3	*abaab*
0	*abaaba*
7	*b*
4	*baab*
1	*baabaa*

SA = [5,2,6,3,0,7,4,1]

Suffix array construction

Algorithm	Worst Case	Speed	Memory
Prefix-Doubling			
MM [MM93]	$O(n \log n)$	16	$8n$
LS [LS99]	$O(n \log n)$	1.7	$8n$
Recursive			
KA [KA03]	$O(n)$	2.2	13-14n
KS [KS03]	$O(n)$	2.8	10-13n
KSPP [KSPP03]	$O(n)$	–	–
HSS [HSS03]	$O(n)$	–	–
KJP [KJP04]	$O(n \log \log n)$	2.1	13-16n
Induced Copying			
IT [IT99]	$O(n^2 \log n)$	4	$5n$
S [S00]	$O(n^2 \log n)$	2.1	$5n$
BK [BK03]	$O(n \log n)$	2.1	5-6n
MF [MF04]	$O(n^2 \log n)$	1	$5n$
SS [SS05]	$O(n^2)$	1	9-10n
M [M05]	$O(n^2 \log n)$	1	5-7n
Suffix Tree			
K [K99]	$O(n \log \sigma)$	4	15-20n

Using suffix array for BWT

String $\boldsymbol{x} \rightarrow \boldsymbol{x}\$$

- sentinel \$ occurs in $\boldsymbol{x}\$$ exactly once
- instead of cyclic rotations of $\boldsymbol{x}\$ \rightarrow$ sort the suffixes of $\boldsymbol{x}$

abrakadabra$
brakadabra$a
rakadabra$ab
akadabra$abr
kadabra$abra
adabra$abrak
dabra$abraka
abra$abrakad
bra$abrakada
ra$abrakadab
a$abrakadabr
$abrakadabra

$abrakadabra
a$abrakadabr
abra$abrakad
abrakadabra$
adabra$abrak
akadabra$abr
bra$abrakada
brakadabra$a
dabra$abraka
kadabra$abra
ra$abrakadab
rakada$braab

a
abra
abrakadabra
adabra
akadabra
bra
brakadabra
dabra
kadabra
ra
rakada

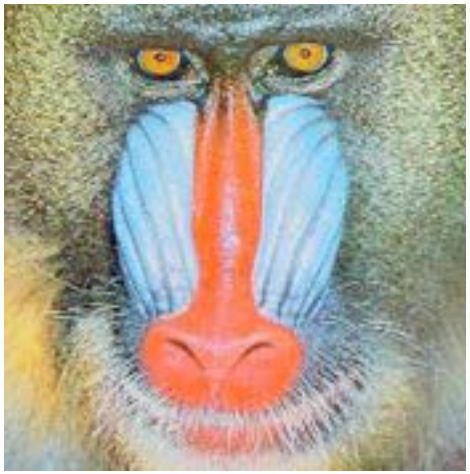
✎ Problem

Modify the decoding algorithm to return the symbols in the original order

<u>Hint</u>: Start with F[I] ...

Data Compression Algorithms

An introduction into lossy
data compression



Can we measure the loss of information?

$\{x_i\} \rightarrow \{y_i\}$

$$d_1 = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|$$

$$d_\infty = \max_i |x_i - y_i|$$

Mean Squared Error

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

Signal to Noise Ratio

$$\mathrm{SNR}(dB) = 10\log_{10}\frac{\frac{1}{n}\sum_{i=1}^{n} x_i^2}{\sigma^2}$$

Peak Signal to Noise Ratio

$$\mathrm{PSNR}(dB) = 10\log_{10}\frac{\max_i x_i^2}{\sigma^2}$$

Motivational example: digitizing sound

Sound – definition

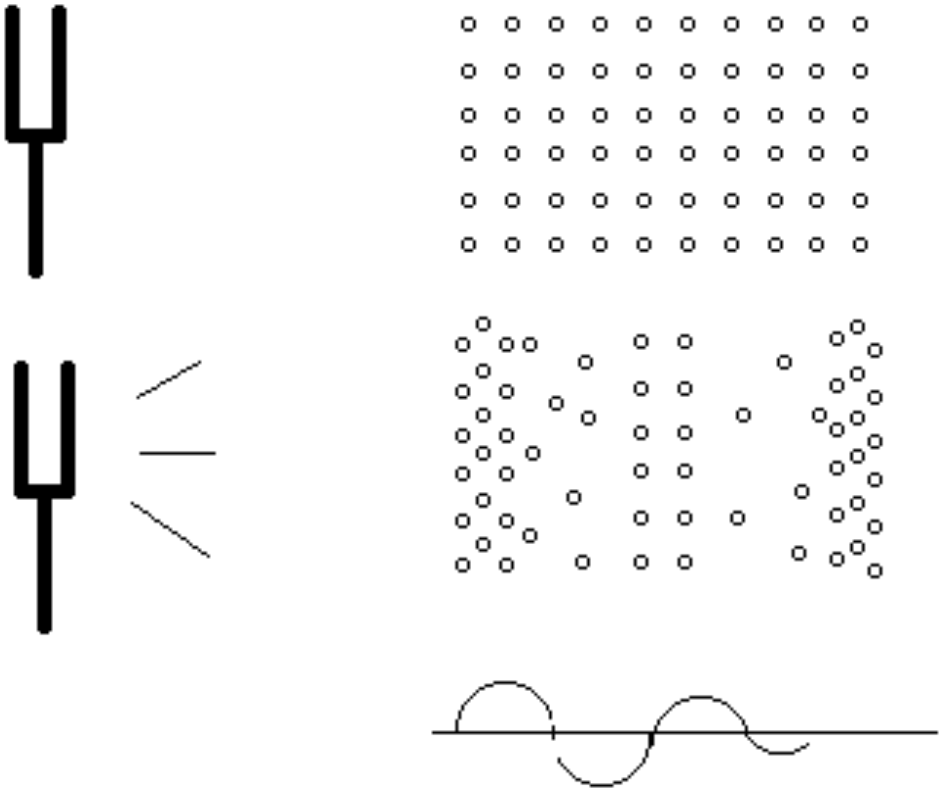
Intuitive: *Sound is the sensation detected by our ears and interpreted by our brain in a certain way.*

Scientific: *Sound is a physical disturbance in a medium. It propagates in the medium as a pressure wave by the movement of atoms or molecules.*

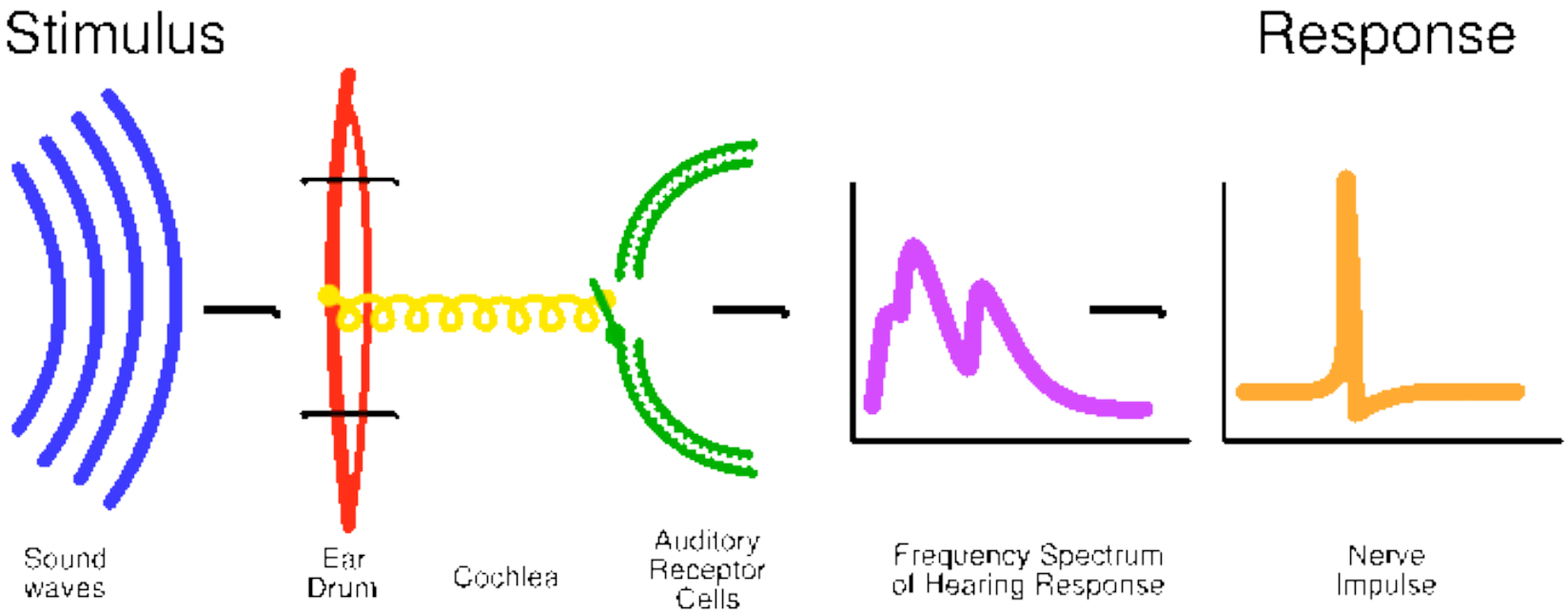
Official (ANSI/ASA S1.1-2013)

- (a) *Oscillation in pressure, stress, particle displacement, particle velocity, etc., propagated in a medium with internal forces (e.g., elastic or viscous), or the superposition of such propagated oscillation.*
- (b) *Auditory sensation evoked by the oscillation described in* (a).
- *Wave is the main source of sound. Man can only listen to sound in 20 Hz-20 kHz.*

Sound wave



Sound wave propagation

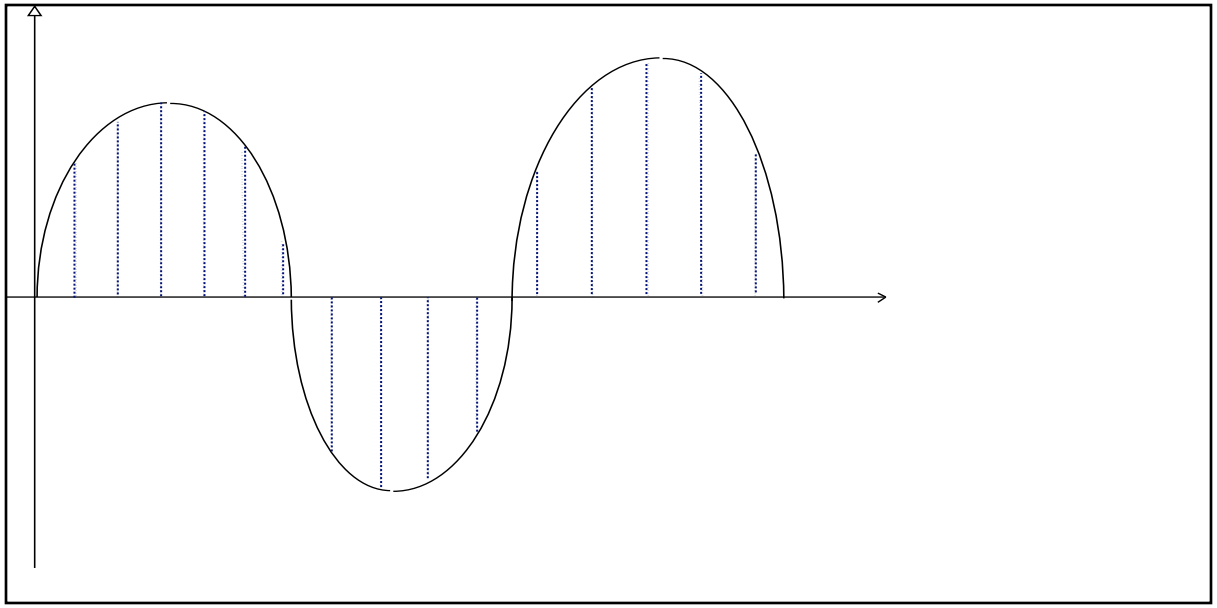


frequency [Hz] 20Hz - 20kHz

amplitude

Capturing audio

Sampling (sampling)



Nyquist–Shannon sampling theorem

Harry Nyquist (1928), Claude Shannon (1949)

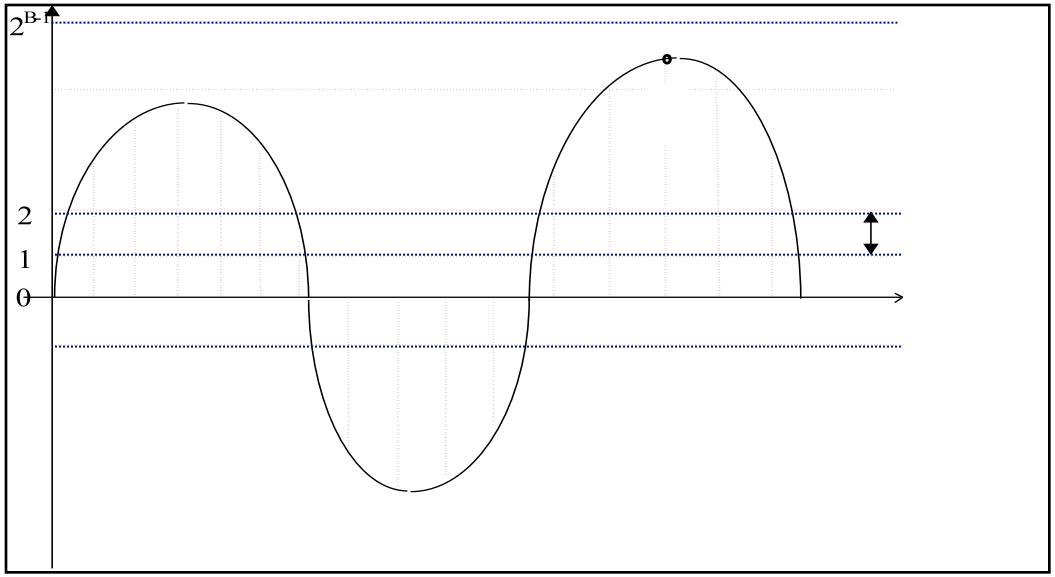
In order to accurately reconstruct an analog signal with highest frequency component F, the sampling frequency must satisfy $F_s >= 2F$.

Condition fails → aliasing: false frequency components appear that were not in the original analog signal, namely frequencies $f' = F_s - f$ where f the frequency component satisfying

$$f > 1/2F_s.$$

Capturing audio

Quantization



Sampling frequency

48 kHz

44.1 kHz

22.05 kHz

11.025 kHz

Bandwidth for various sources

sound source	bandwidth
radio (FM)	50 Hz - 15 kHz
radio (AM)	80 Hz - 5 kHz
CD player	20 Hz - 20 kHz
(cheap) microphone	80 Hz - 12 kHz
trumpet	180 Hz - 8 kHz
telephone	300 Hz - 3 kHz
child's ears	20 Hz - 20 kHz
rock fan ears	50 Hz - 10 kHz
male voice	120 Hz - 7 kHz
female voice	200 Hz - 9 kHz

Sampling frequency

48 kHz

44.1 kHz

22.05 kHz

11.025 kHz

Resolution

16b	96dB
8b	48dB
4b	24dB

Problem: SNR

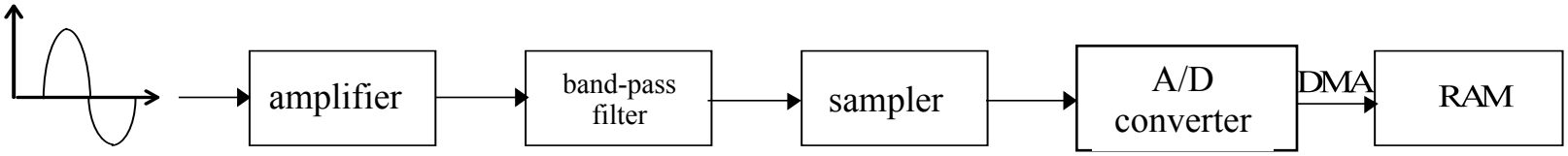
- Assumptions
  - uniform quantization
  - n – bit samples
  - uniform probability distribution of the input data
- Then SNR = $20\, n \log_{10} 2\,\mathrm{dB}$
  - $\approx 6.02\, n$ dB
  - 16b sample $\approx$ 96 dB

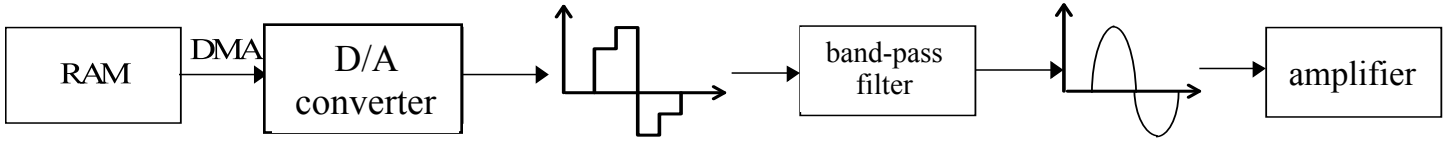
Intensity of sound – comparison

sound	dB	W/m^2
jet plane	165	30 000
pain threshold	145	300
factory hall	125	3
highway	105	$3 \cdot 10^{-2}$
screaming	85	$3 \cdot 10^{-4}$
conversation	65	$3 \cdot 10^{-6}$
quite home	45	$3 \cdot 10^{-8}$
whisper	25	$3 \cdot 10^{-10}$
audibility threshold	0	$1 \cdot 10^{-12}$

$$L_I = 10 \log_{10} \frac{I}{I_0} [dB] \quad I_0 = 1 \times 10^{-12} \frac{W}{m^2}$$

Audio processing





Quantization

{large set} $\rightarrow$ {smaller set}

Quantizer

- scalar
- vector

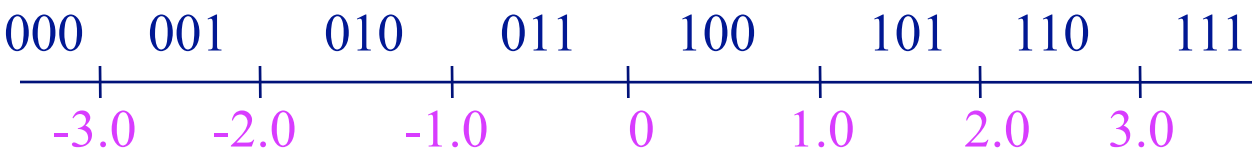
Quantization = 2 mappings

- encoding
- decoding

Source = analog signal

$\Rightarrow$ analog-to-digital (A/D) converter

Example: 3bit quantizer



code	000	001	010	011	100	101	110	111
output	-3.5	-2.5	-1.5	-0.5	0.5	1.5	2.5	3.5

Discrete and continuous random variables

X is a *discrete random variable* if

- ∃ at most countable set R
- tž. $P(X \in R) = 1$

Probability function $p(x) = P(X = x)$

Expected value $E(X) = \Sigma_{x \in \mathbf{R}}\, x\, p(x)$

Variance $\mathrm{Var}(X) = E(X-EX)^2$

X is *absolutely continuous random variable* if

- ∃ real-valued function f
- such that $P(x_1 \le X \le x_2) = \int_{x_1}^{x_2} f(x)\,\mathrm{d}x$

$$E(X) = \int x\, f(x)\, dx$$

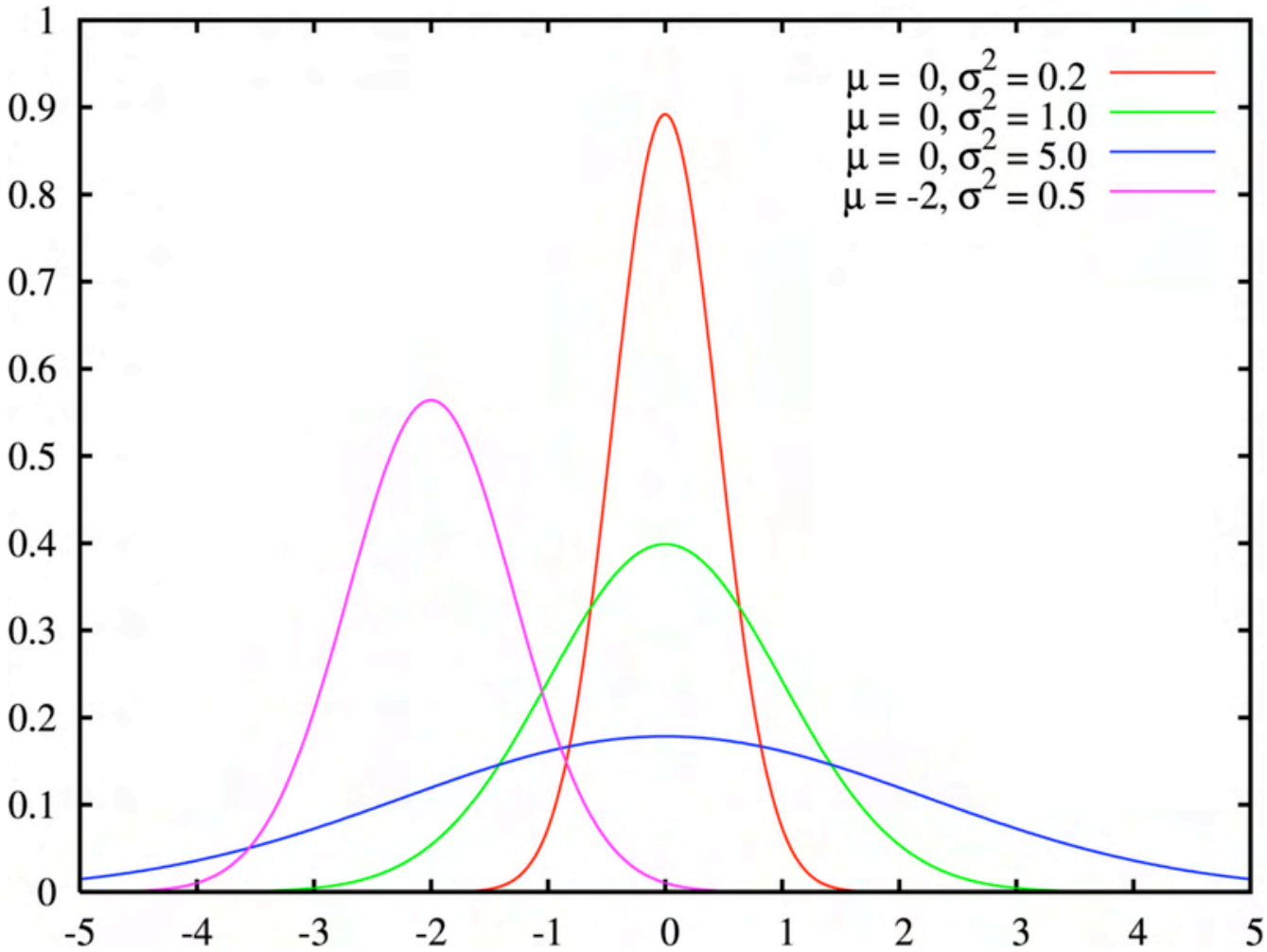
Probability distribution

X has a *normal distribution* N(μ,σ²) if

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Central limit theorem: when measuring values of r.v.X, after a sufficiently large number *n* of iterations, the average of measured values has distribution close to N(EX,Var X/*n*).

Normal distribution



Problem formulation

Source - random variable X with density f(x)

- common assumption: f is symmetric about 0

Determine

- number of intervals (*levels*)
- interval endpoints b_0 ,..., b_M (*decision boundaries*)
- representative y_i for each interval (i=0,...,M)
  $Q(x) = y_i \quad \Leftrightarrow \quad b_{i-1} < x \leq b_i$

To minimize the quantization error (*distortion*) σ^2

$$\sigma^2 = \int_{-\infty}^{\infty} \left(x - Q(x)\right)^2 f(x)dx = \sum_{i=1}^{M} \int_{b_{i-1}}^{b_i} \left(x - y_i\right)^2 f(x)dx$$

Problem formulation

Fixed length codewords $\Rightarrow$ M levels

Input

- given M levels
- and density $f(x)$ = probability distribution of the source X

Problem

- find $\{b_i\}$ and $\{y_i\}$
- to minimize σ^2

Uniform quantization

Uniform probability distribution of X in $\langle -X_{max}, X_{max} \rangle$

All intervals of equal length Δ

Quantizers of two types

- origin is endpoint / midpoint of the interval

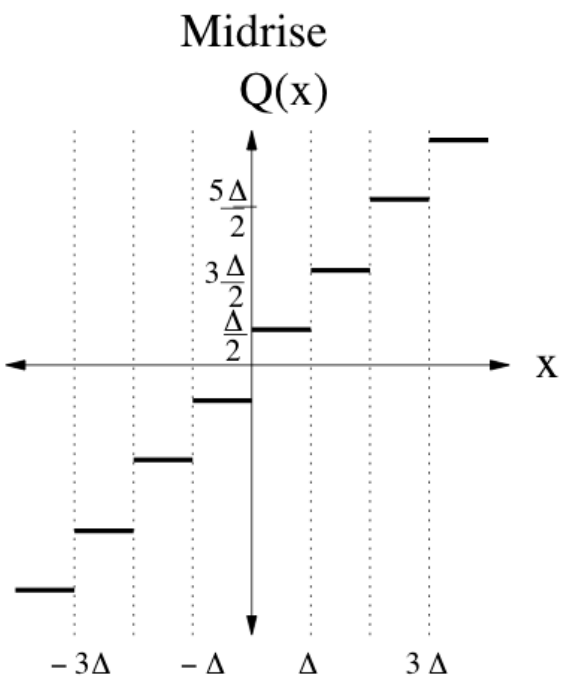
Midrise quantizer

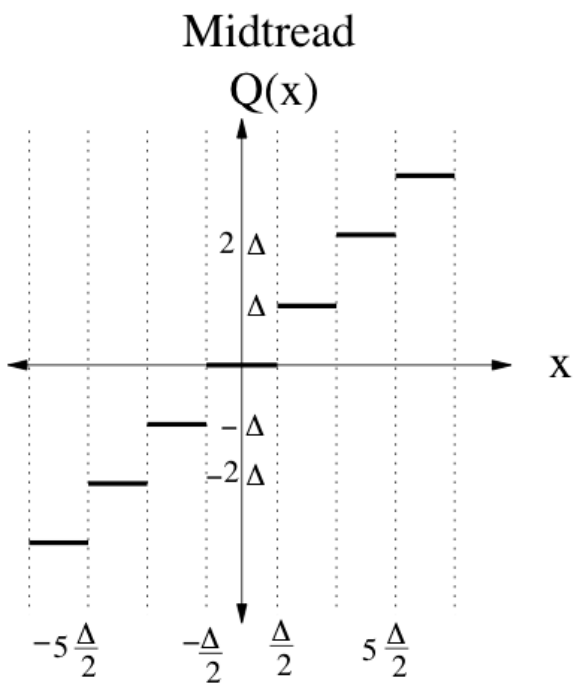
- M is even and $b_{M/2} = 0$

Midtread quantizer

- M is odd and $b_{(M-1)/2} = -\Delta/2$
- $Q(x) = 0$ when $x \in [-\Delta/2, \Delta/2]$

Midrise / midtread quantizer





Uniform quantizer – example

☀ **Example:** image compression

b bits/pixel $\Rightarrow X \in \langle 0, 2^b\text{-}1 \rangle$

$b = 8$, choose $M = 2^3$

$\{b_i\} = \{0,32,64,96,128,160,192,224,255\}$

$\{y_i\} = \{16,48,80,112,144,176,208,240\}$

Uniform quantizer – general case

Range of X need not be bounded

Probability distribution need not be uniform

Adaptive quantization

Forward adaptive quantization (off-line)

- split the input into blocks
- analyze each block and set the parameters accordingly
- parameters are transmitted as a *side information*

☀ **Example**: image compression

- 3bit forward adaptive quantizer
- block size 8×8
- max & min value for each block
- side info = 16/ (8×8)=0.25 bits / pixel

Adaptive quantization

Backward adaptive quantization (on-line)

- based on quantized input
- no side info necessary

Jayant quantizer (1973)

k^{th} interval $\rightarrow$ multiplier M_k

- inner intervals $\rightarrow$ $M_k < 1$
- outer intervals $\rightarrow$ $M_k > 1$

Δ_n − step size in n^{th} iteration

$I(n)$ − interval used in the nth iteration

$\Delta_n = M_{I(n-1)} \times \Delta_{n-1}$

Example: 3bit Jayant quantizer

Multipliers

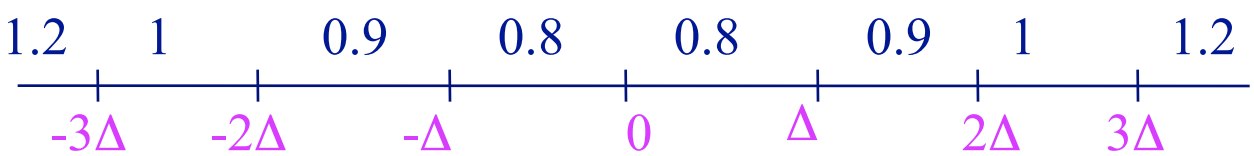
$M_0 = M_4 = 0.8$

$M_1 = M_5 = 0.9$

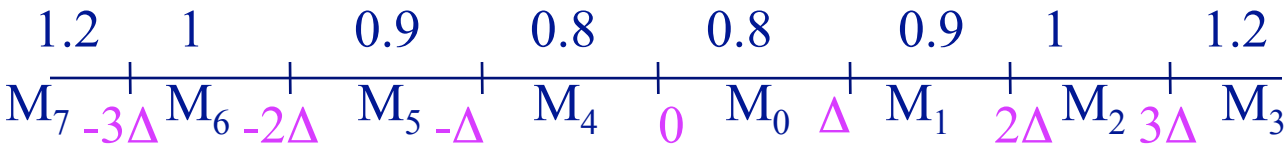
$M_2 = M_6 = 1$

$M_3 = M_7 = 1.2$

Initial value: $\Delta_0 = 0.5$

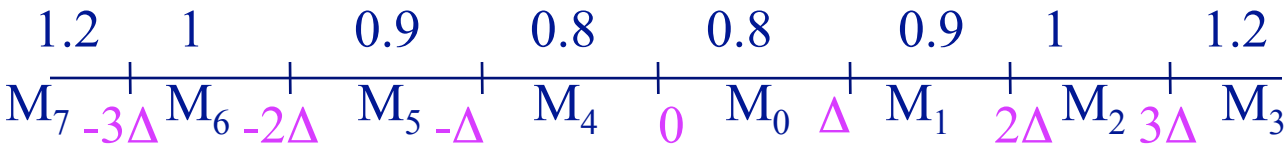


3bit Jayant quantizer



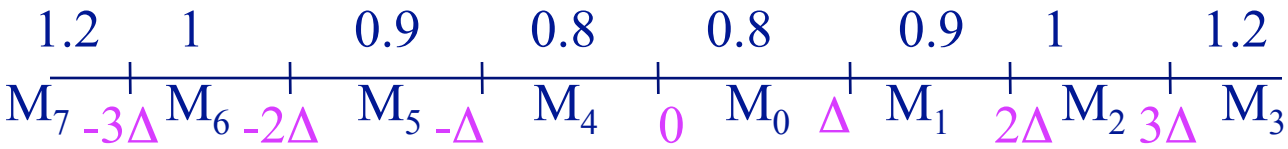
n	Δ_n	input	code	update
0	0.5	0.1	0	$\Delta_1 = M_0 \times \Delta_0$

3bit Jayant quantizer



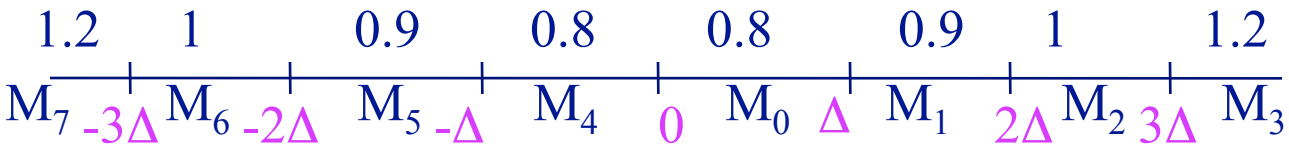
n	Δ_n	input	code	update
0	0.5	0.1	0	$\Delta_1 = M_0 \times \Delta_0$
1	0.4	-0.2	4	$\Delta_2 = M_4 \times \Delta_1$

3bit Jayant quantizer



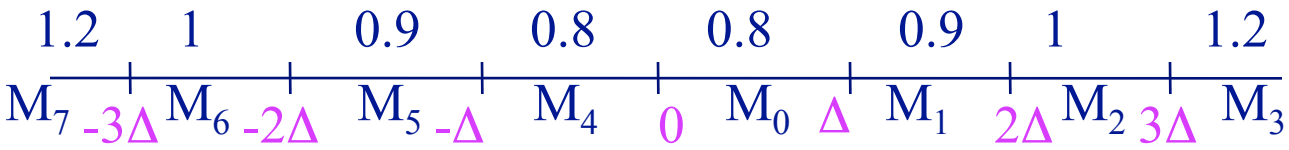
n	Δ_n	input	code	update
0	0.5	0.1	0	$\Delta_1 = M_0 \times \Delta_0$
1	0.4	-0.2	4	$\Delta_2 = M_4 \times \Delta_1$
2	0.32	0.2	0	$\Delta_3 = M_0 \times \Delta_2$

3bit Jayant quantizer



n	Δ_n	input	code	update
0	0.5	0.1	0	$\Delta_1 = M_0 \times \Delta_0$
1	0.4	-0.2	4	$\Delta_2 = M_4 \times \Delta_1$
2	0.32	0.2	0	$\Delta_3 = M_0 \times \Delta_2$
3	0.256	0.1	0	$\Delta_4 = M_0 \times \Delta_3$
4	0.2048	-0.3	5	$\Delta_5 = M_5 \times \Delta_4$
5	0.1843	0.1	0	$\Delta_6 = M_0 \times \Delta_5$
6	0.1475	0.2	1	$\Delta_7 = M_1 \times \Delta_6$
7	0.1328	0.5	3	$\Delta_8 = M_3 \times \Delta_7$

3bit Jayant quantizer



n	Δ_n	input	code	update	output	error
0	0.5	0.1	0	$\Delta_1 = M_0 \times \Delta_0$	0.25	0.15
1	0.4	-0.2	4	$\Delta_2 = M_4 \times \Delta_1$	-0.2	0.0
2	0.32	0.2	0	$\Delta_3 = M_0 \times \Delta_2$	0.16	0.04
3	0.256	0.1	0	$\Delta_4 = M_0 \times \Delta_3$	0.128	0.028
4	0.2048	-0.3	5	$\Delta_5 = M_5 \times \Delta_4$	-0.3072	-0.0072
5	0.1843	0.1	0	$\Delta_6 = M_0 \times \Delta_5$	0.0922	-0.0078
6	0.1475	0.2	1	$\Delta_7 = M_1 \times \Delta_6$	0.2212	0.0212
7	0.1328	0.5	3	$\Delta_8 = M_3 \times \Delta_7$	0.4646	-0.0354

Nonuniform quantization

$$\sigma^2 = \int_{-\infty}^{\infty} \left(x - Q(x)\right)^2 f(x)dx = \sum_{i=1}^{M} \int_{b_{i-1}}^{b_i} \left(x - y_i\right)^2 f(x)dx$$

$$y_j = \frac{\int_{b_{j-1}}^{b_j} xf(x)dx}{\int_{b_{j-1}}^{b_j} f(x)dx} \qquad b_j = \frac{y_{j+1} + y_j}{2}$$

Lloyd - Max algorithm

Lloyd - Max algorithm

<u>Goal</u>: M-level symmetric quantizer

Determine $\{b_1, b_2, \ldots, b_{M/2-1}\}$, $\{y_1, y_2, \ldots, y_{M/2}\}$

- $\{b_{-1}, b_{-2}, \ldots, b_{-(M/2-1)}\}$, $\{y_{-1}, y_{-2}, \ldots, y_{-M/2}\}$ symmetric
- $b_0 = 0$

Choose initial approximation y_1, compute b_1 from the equation

$$y_1 = \frac{\int_{b_0}^{b_1} x f(x)\,dx}{\int_{b_0}^{b_1} f(x)\,dx}$$

$y_2 = 2\,b_1 - y_1$

Similarly b_2, y_3, ...

Lloyd - Max algorithm

Computation depends on the initial estimate y_1 !

Correction: We know $b_{M/2}$ = max input value

$$y^*_{\frac{M}{2}} = \frac{\int_{b_{\frac{M}{2}-1}}^{b_{\frac{M}{2}}} x f(x)dx}{\int_{b_{\frac{M}{2}-1}}^{b_{\frac{M}{2}}} f(x)dx}$$

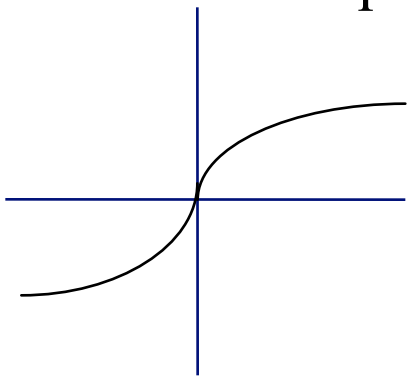
if $\left| y^*_{M/2} - y_{M/2} \right| < \varepsilon$ **then** stop

else correct the estimate of y_1 in the direction indicated by the sign of the difference and repeat the whole procedure

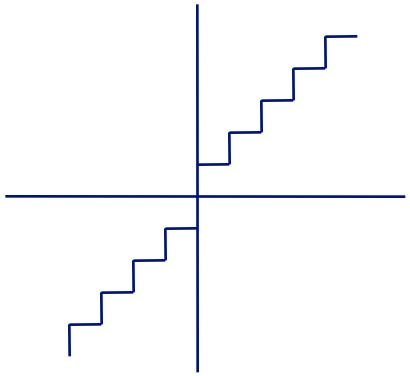
Nonuniform quantization: Companding

Companding (compressing + expanding)

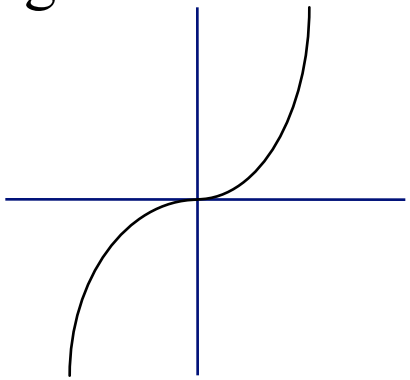
- compressor
  - » stretch the high probability regions close to the origin
  - » compress the regions in which the input occurs with low probability
- uniform quantization
- expander
  - » transform quantized values to the original scale



compressor



uniform q.



expander

μ-law

μ-law

- telephony, USA, Japan
- μ=225

$$c(x) = x_{\max} \frac{\ln(1 + \mu \frac{|x|}{x_{\max}})}{\ln(1+\mu)} \mathrm{sgn}(x)$$

$$c^{-1}(x) = \frac{x_{\max}}{\mu} [(1+\mu)^{\frac{|x|}{x_{\max}} - 1}] \mathrm{sgn}(x)$$

A-law

A-law

- telephony, Europe
- A = 87.6

$$c(x) = \begin{cases} \frac{A|x|}{1+\ln A}\operatorname{sgn}(x) & 0 \le \frac{|x|}{x_{\max}} \le \frac{1}{A} \\ x_{\max}\frac{1+\ln\frac{A|x|}{x_{\max}}}{1+\ln A}\operatorname{sgn}(x) & \frac{1}{A} \le \frac{|x|}{x_{\max}} \le 1 \end{cases}$$

$$c^{-1}(x) = \begin{cases} \frac{|x|}{A}(1+\ln A) & 0 \le \frac{|x|}{x_{\max}} \le \frac{1}{1+\ln A} \\ \frac{x_{\max}}{A}\exp\left[\frac{|x|}{x_{\max}}(1+\ln A)-1\right] & \frac{1}{1+\ln A} \le \frac{|x|}{x_{\max}} \le 1 \end{cases}$$

G.711

ITU-T standard for audio companding, primarily used in telephony

sampling frequency 8kHz, sample length

- 14b (μ-law)
- 13b (A-law)

Tranformation μ (A) → approximation

Uniform quantization

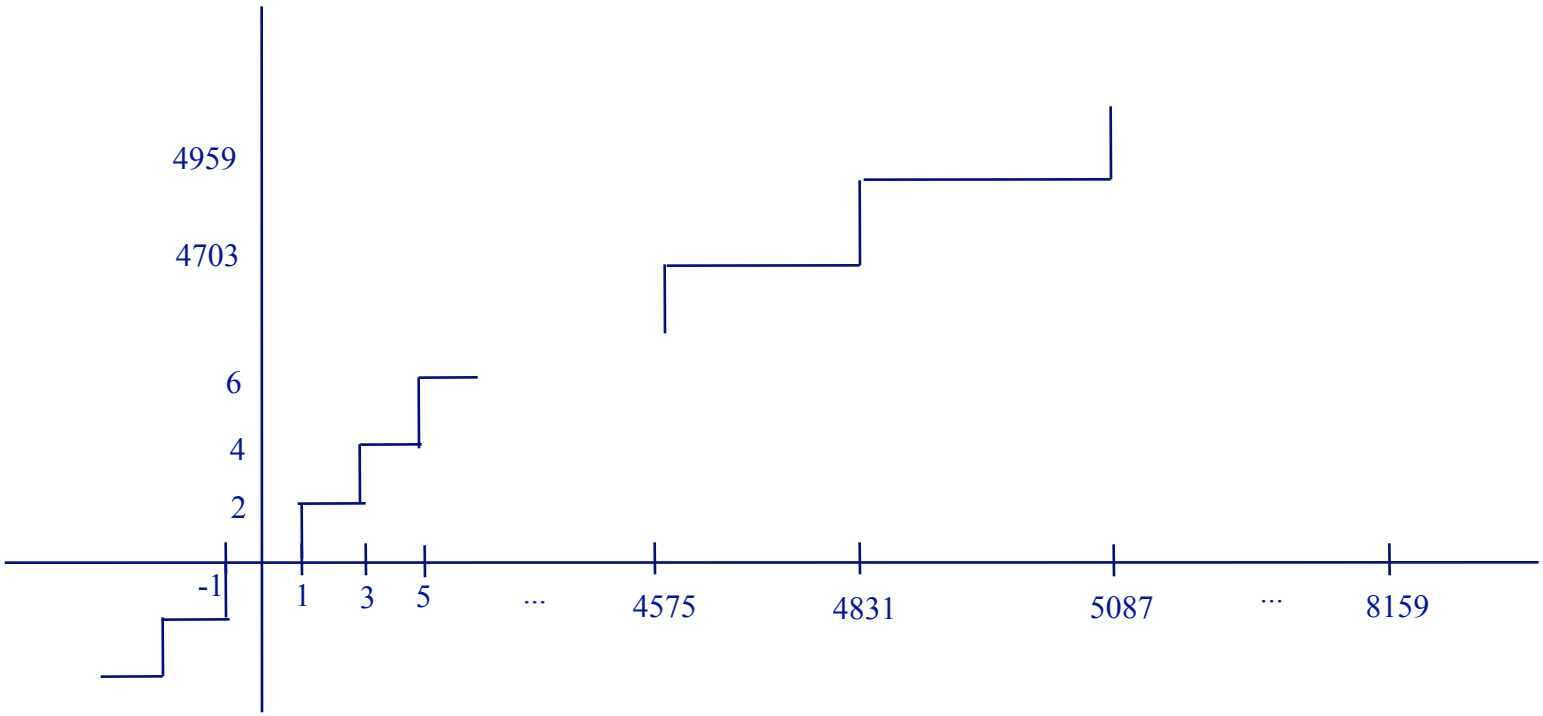
- μ-law: midtread quantizer (zero interval)
- A-law: midrise quantizer (zero endpoint)

G.711

Encoder output

- 8b samples, i.e. 64 Kbps
- 7b samples (56 Kbps) omit one LSB
- 6b samples (48 Kbps) omit two LSB

μ-law as a nonuniform quantizer

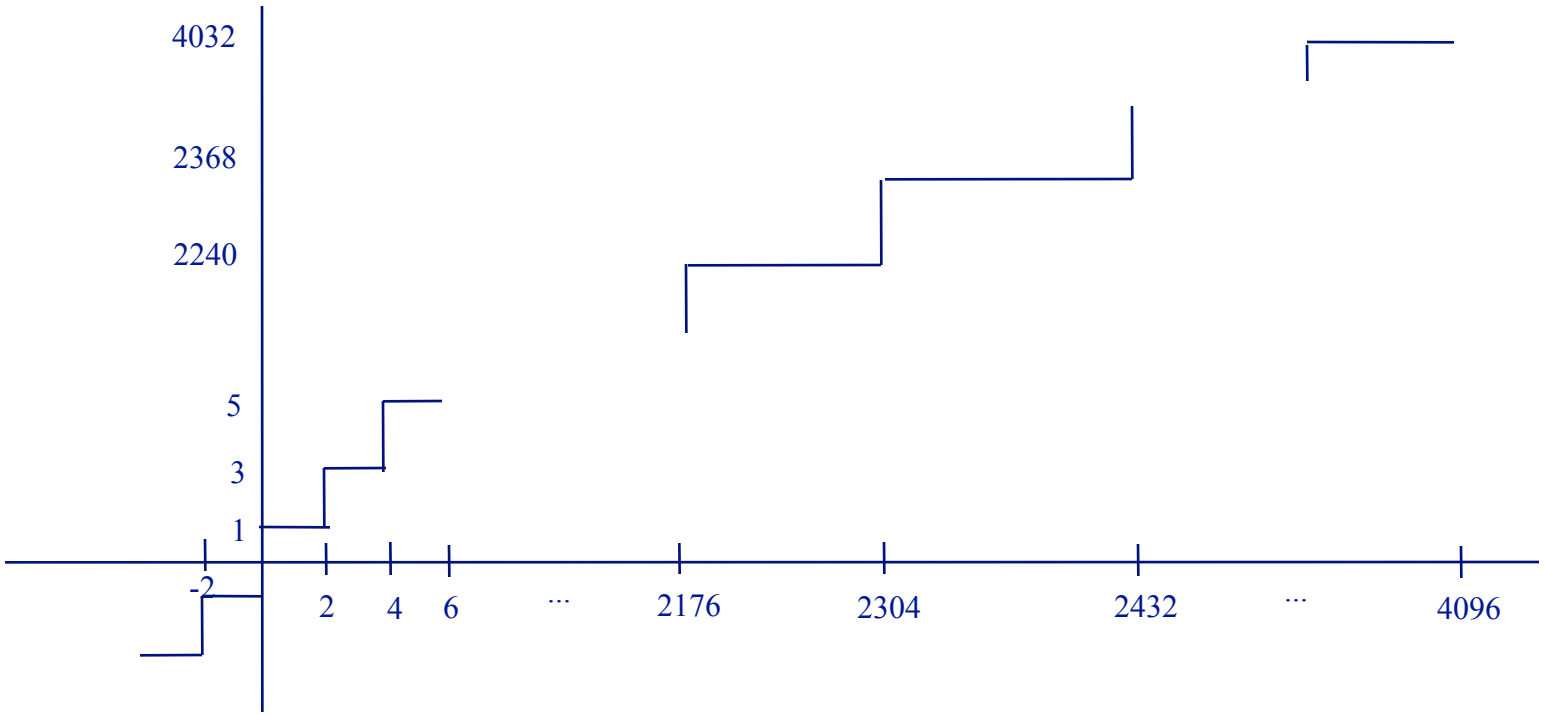


14b samples ⇒ ⟨-8192, 8191⟩, approximation only ⟨-8160, 8159⟩

255 intervals symmetric around 0, which is the midpoint of the middle interval

reconstruction levels = interval midpoints

A-law as a nonuniform quantizer



13b samples $\Rightarrow \langle -4096, 4095 \rangle$

256 intervals symmetric around 0, which is the endpoint

reconstruction levels = interval midpoints

μ-law approximation used in G.711

$x = x_{12} \ldots x_1 x_0$:= input without sign

$x := x + 32$

$pos := \max\{i \mid x_i = 1,\ 5 \le i \le 12\}$

output := sign bit | 3bit code of *pos - 5* |

$x_{pos-1}\ x_{pos-2}\ x_{pos-3}\ x_{pos-4}$

output := ~ *output*

Biased Input Values													Compressed Code Word						
													Chord			Step			
bit: 12	11	10	9	8	7	6	5	4	3	2	1	0	bit: 6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	a	b	c	d	x	0	0	0	a	b	c	d
0	0	0	0	0	0	1	a	b	c	d	x	x	0	0	1	a	b	c	d
0	0	0	0	0	1	a	b	c	d	x	x	x	0	1	0	a	b	c	d
0	0	0	0	1	a	b	c	d	x	x	x	x	0	1	1	a	b	c	d
0	0	0	1	a	b	c	d	x	x	x	x	x	1	0	0	a	b	c	d
0	0	1	a	b	c	d	x	x	x	x	x	x	1	0	1	a	b	c	d
0	1	a	b	c	d	x	x	x	x	x	x	x	1	1	0	a	b	c	d
1	a	b	c	d	x	x	x	x	x	x	x	x	1	1	1	a	b	c	d

Comparison / application

μ-law

- zero interval ⇒ noise reduction in blocks of silence
- higher dynamic range

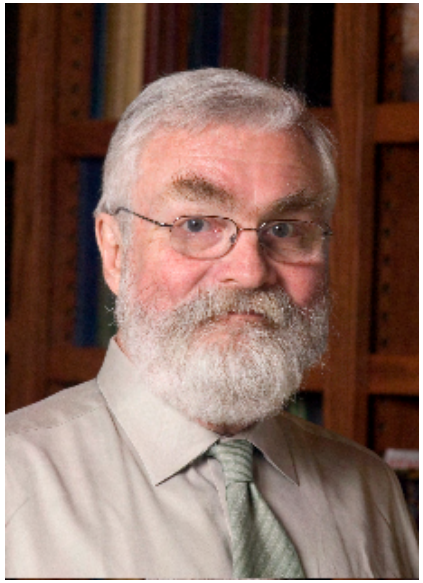
A-law:

- lower relative distortion of small values

<u>Application</u>: VoIP (Internet telephony)

Data Compression Algorithms

Vector Quantization



Robert M. Gray

Quantization

{large set} → {smaller set}

Quantization = 2 mappings

- encoding
- decoding

Quantizer

- scalar
- vector

Group input samples → vectors

- L consecutive samples of speech
- block of $L = n \times m$ pixels of an image (usually n=m)
- → vector of dimension L

Vector quantization – idea

Codebook of the vector quantizer

- consists of code-vectors selected to represent
- the vectors generated from the source
- code-vector → binary index

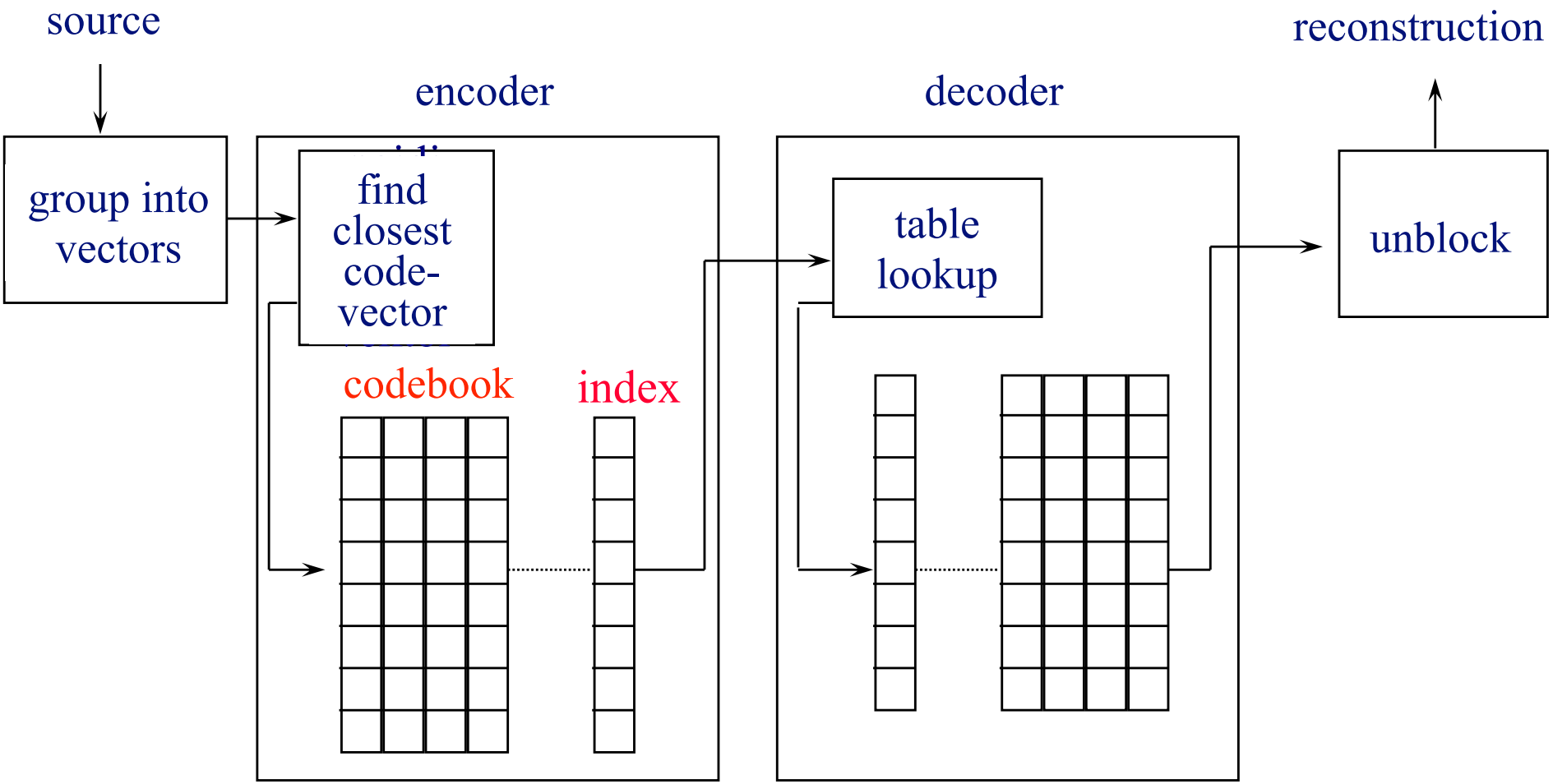
Encoder

- find the code-vector
- closest to the source vector
- transmit/store the binary index of the code-vector

Decoder

- retrieve the code-vector given its binary index
- code-vector components = quantized source

Vector quantization procedure



Preliminaries

Compression performance → rate [bits / sample]

Codebook size K (K-level vector quantizer)

- $\lceil \log_2 K \rceil$ bits to specify the index of a code-vector
- L-dimensional source vector
- rate $= \dfrac{\lceil \log_2 K \rceil}{L}$ bits / sample

How to measure the distance of vectors?

- codebook $C = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K\}$

Source vector $\mathbf{x}$ is closest to the code-vector $\mathbf{y}_j$ if

- $d(\mathbf{x}, \mathbf{y}_j) \leq d(\mathbf{x}, \mathbf{y}_i)$ for all $i = 1, \ldots, K$
- $d((x_1, \ldots x_L), (y_1, \ldots y_L)) = (x_1 - y_1)^2 + \ldots + (x_L - y_L)^2$

≈ mean squared error

Codebook design

Automatic procedure

- to locate where the source vectors are clustered

Pattern recognition : k-means algorithm

- large set of source vectors → training set
- initial set of k representative patterns
- assign each vector of the training set
- to the closest representative pattern
- update the representative pattern by computing
- the centroid of the training set vectors assigned to it

Output: k groups of vectors clustered around representatives

LBG algorithm

Yoseph Linde, Andrés Buzo, Robert M. Gray (1980)

⓪ Initialization

- training set $T = \{\mathbf{x}_i\}_{i=1,\ldots,N}$
- initial codebook $C = \{\mathbf{y}_i^{(0)}\}_{i=1,\ldots,K}$
- $D^{(0)} = 0$ # distortion
- select ε # threshold
- $s = 1$ # step

① Compute quantization regions

- **for** i **in range**$(1, K+1)$ **:** # $1 \leq i \leq K$

$$V_i^{(s)} = \{\mathbf{x} \in T \mid \mathrm{d}(\mathbf{x}, \mathbf{y}_i) \leq \mathrm{d}(\mathbf{x}, \mathbf{y}_j) \text{ for all } j \neq i\}$$

- each $\mathbf{x}$ is assigned to exactly one region
- # assume that $V_i^{(s)} \neq \emptyset$ for all i

LBG algorithm

② Compute the average distortion

- $D^{(s)} = \frac{1}{N} \sum_{i=1}^{K} \sum_{\mathbf{x} \in V_i^{(s)}} d(\mathbf{x}, \mathbf{y}_i^{(s)})$

③ Check the threshold

- **if** $\frac{|D^{(s)} - D^{(s-1)}|}{D^{(s)}} < \varepsilon$ **: return** C

④ Update the codebook

- s += 1
- $C = \{\mathbf{y}_i^{(s)}\}_{i=1,\ldots,K}$
- where $\mathbf{y}_i^{(s)}$ = average over all vectors of $V_i^{(s)}$

⑤ **Go to** step ①

LBG – example : initialization

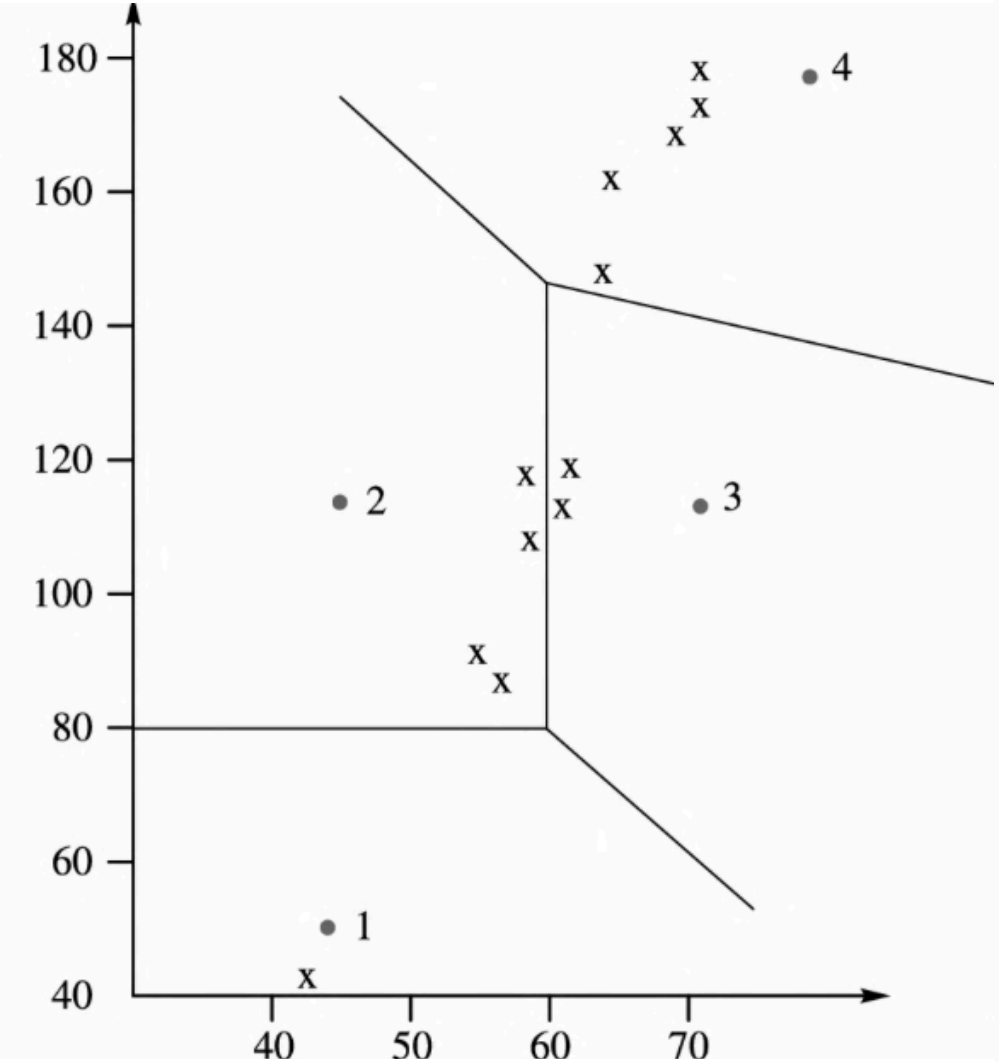
$T =$

weight	height
72	180
65	120
59	119
64	150
65	162
57	88
72	175
44	41
62	114
60	110
56	91
70	172

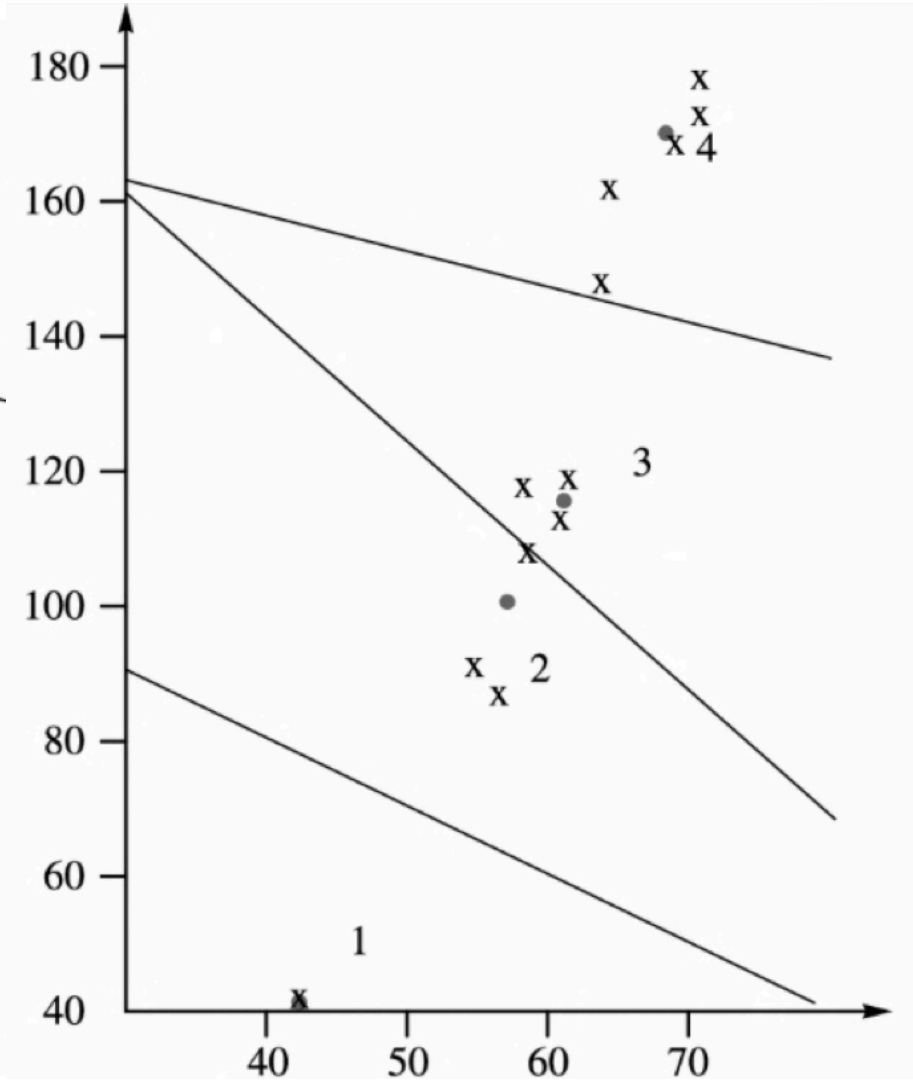
$C =$

weight	height
45	50
75	117
45	117
80	180

LBG – example : step $s = 1, 2$

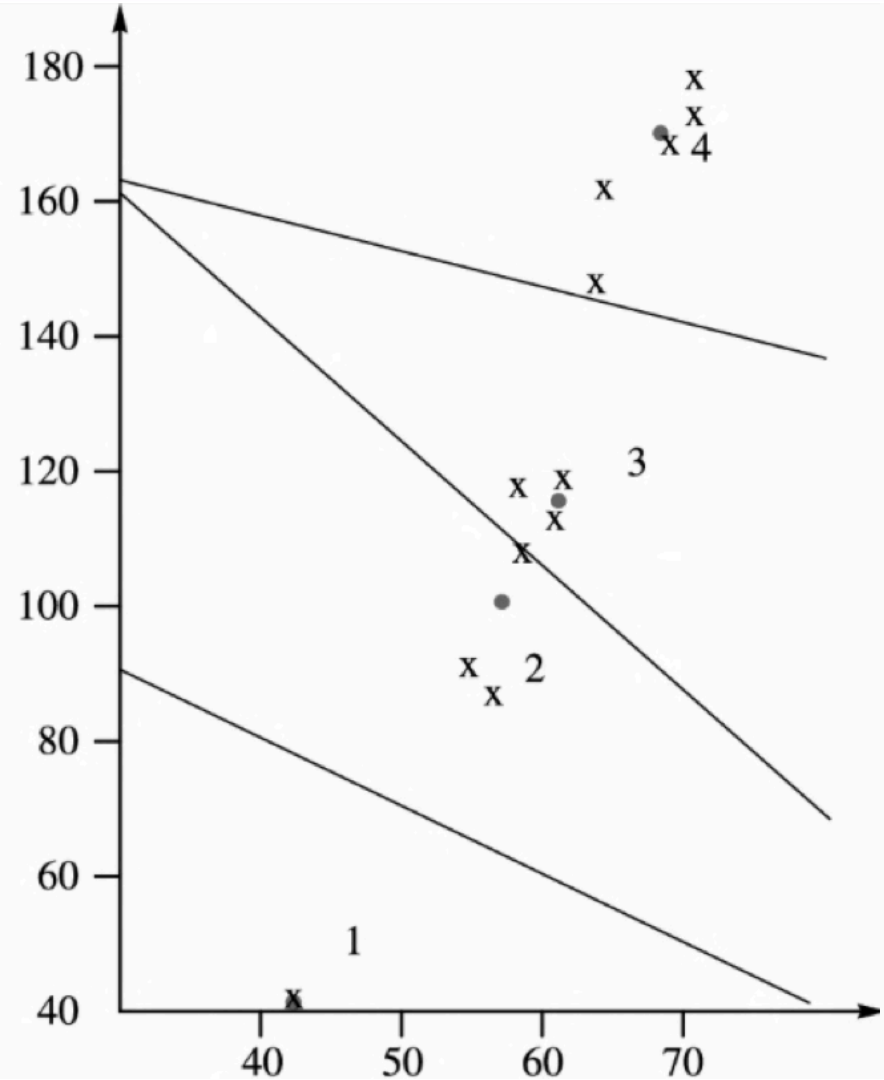


$D^{(1)} = 387.25$

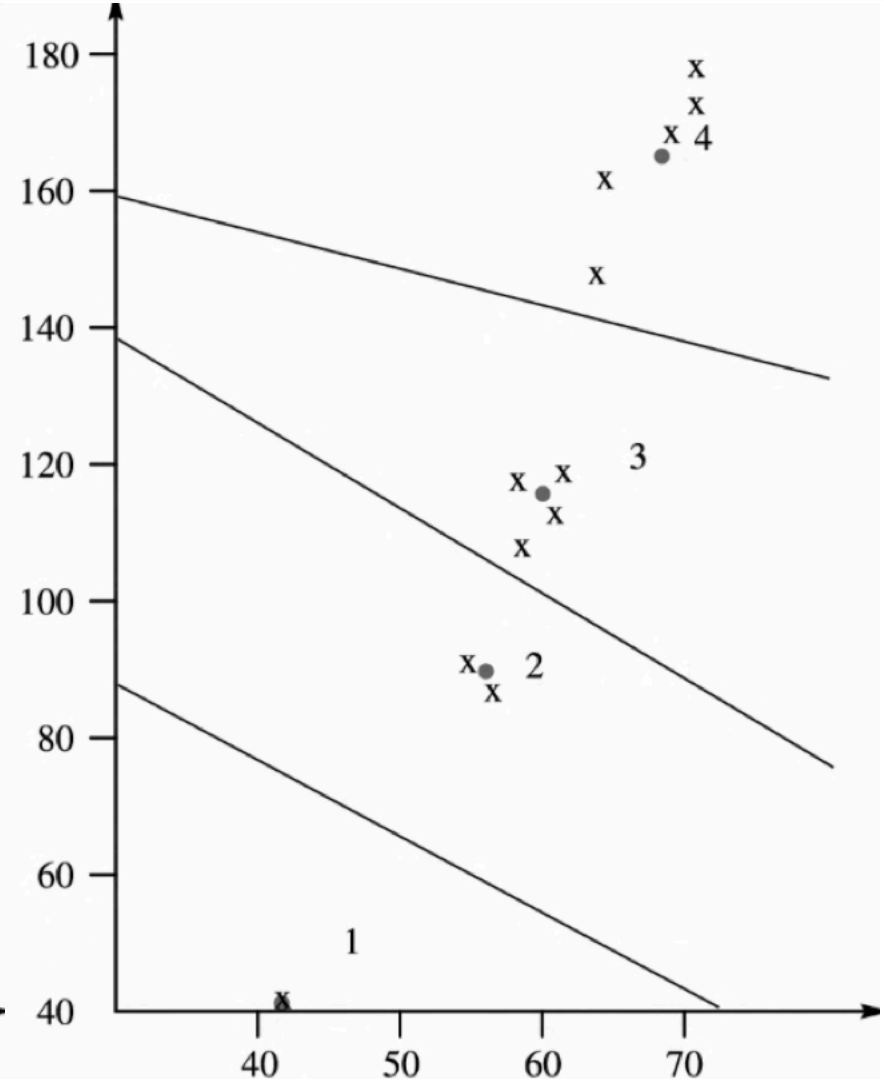


$D^{(2)} = 89$

LBG – example : step $s = 2, 3$



$D^{(2)} = 89$



$D^{(3)} = 60.17$

LBG Algorithm – initialization

Splitting technique (Lindo, Buzo, Gray)

- given training set T, set $\mathbf{v}$ ={average of T}
- # now C ={$\mathbf{v}$}
- randomly generate a perturbation vector $\boldsymbol{e}$
- LBG({$\mathbf{v}$, $\mathbf{v}$+$\mathbf{e}$}) → {$\mathbf{x}$, $\mathbf{y}$}
- # now C ={$\mathbf{x}$, $\mathbf{y}$}
- repeat until the desired number of levels is reached
- **if** desired $|C| \neq$ power of 2 **:** at the last step
  - » perturb only as many code-vectors as necessary
  - » those with the largest # of training set vectors
  - » or with the largest distortion

Example – splitting technique

$T =$

weight	height
72	180
65	120
59	119
64	150
65	162
57	88
72	175
44	41
62	114
60	110
56	91
70	172

$C =$

weight	height
62	127

fixed perturbation vector
$\boldsymbol{e} = (10, 10)$

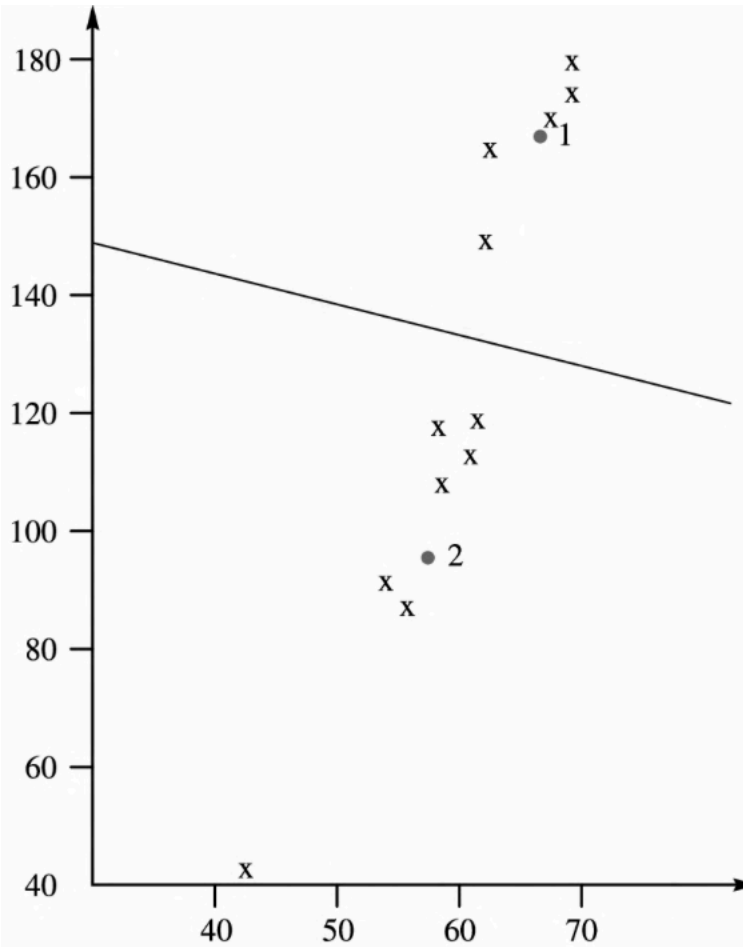
initial 2-level

weight	height
62	127
72	137

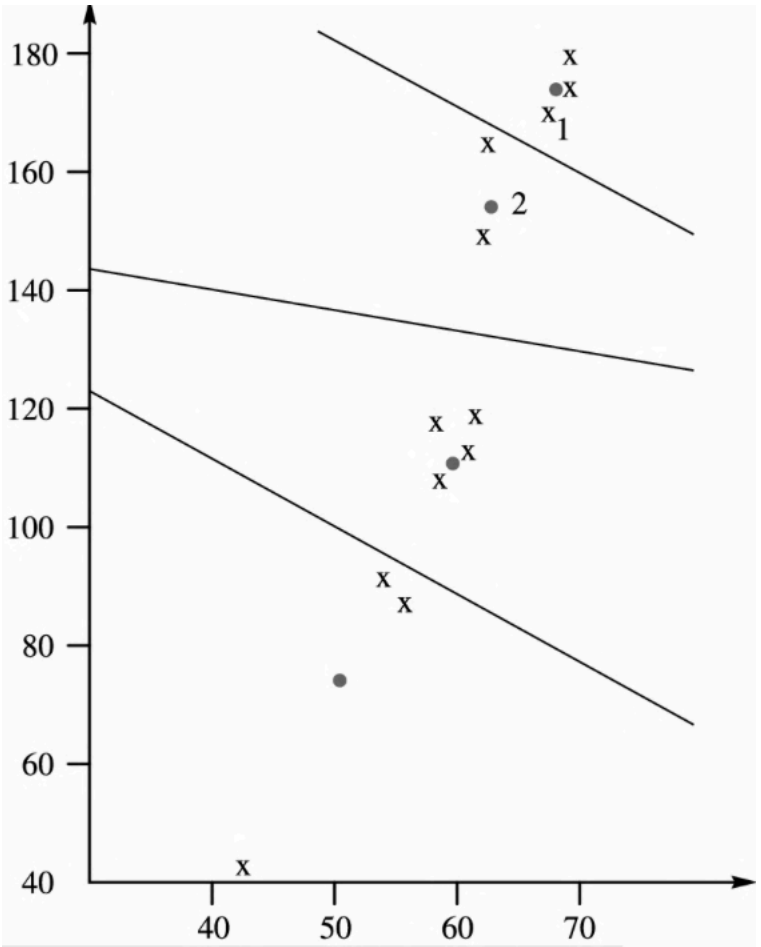
final 2-level

weight	height
58	98
69	168

Example – splitting technique



$D = 468.58$



$D = 156.17$

LBG Algorithm – initialization

Random init (Hilbert)

- initialize C randomly several times
- select the alternative with lowest distortion

Pairwise nearest neighbor – PNN (Equitz, 1989)

- $C = T$ # $|T|$-level quantizer
- in each step, replace two code-vectors by their mean
- goal: smallest increase in distortion
- combining code-vectors $\mathbf{y}_i$ and $\mathbf{y}_j$ increases distortion by

$$(n_i \cdot n_j / (n_i + n_j))\ \mathrm{d}(\mathbf{y}_i, \mathbf{y}_j)$$

- n_i = number of training set vectors in the cluster represented by $\mathbf{y}_i$

Example – PNN

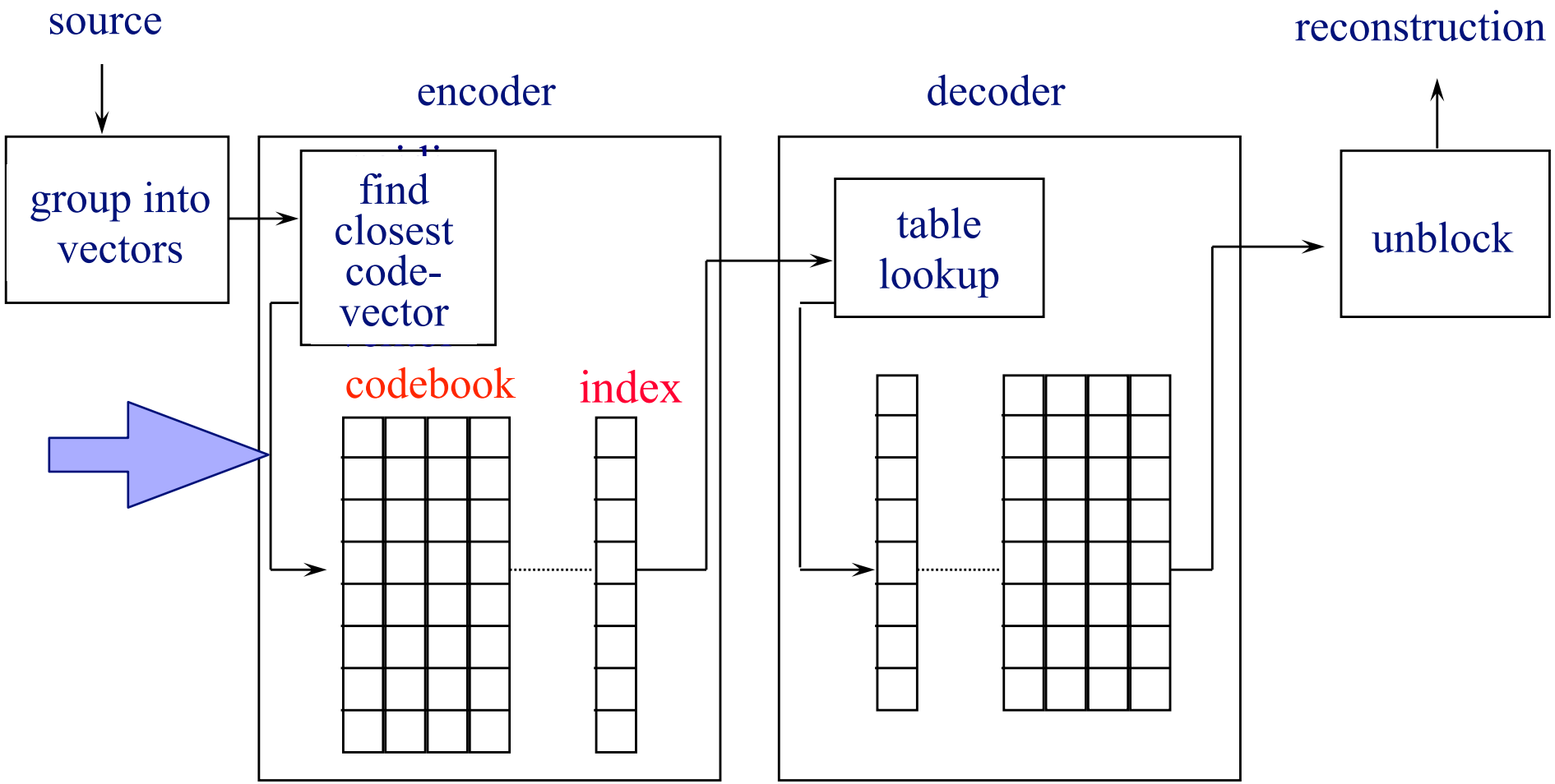
72,180	72,180	71,176 (3)	71,176 (3)	69,168 (5)	69,168 (5)
72,175	71,174 (2)				
65,120	65,120	65,120	65,120	65,120	62,116 (4)
44,41	44,41	44,41	44,41	44,41	44,41
59,119	61,117 (2)	61,117 (2)	60,114 (3)	60,114 (3)	
62,114					
64,150	65,156 (2)	65,156 (2)	65,156 (2)		
60,110	60,110	60,110			
65,162					
56,91	57,90 (2)	57,90 (2)	57,90 (2)	57,90 (2)	57,90 (2)
57,88					
70,172					

Empty cell problem

What if $V_i^{(s)} = \emptyset$?

- replace $\mathbf{y}_i$ in C with a new code-vector
- chosen randomly from the cluster with
  - » the highest population of training vectors
  - » or the highest associated distortion

Codebook searching problem



Tree-structured vector quantization

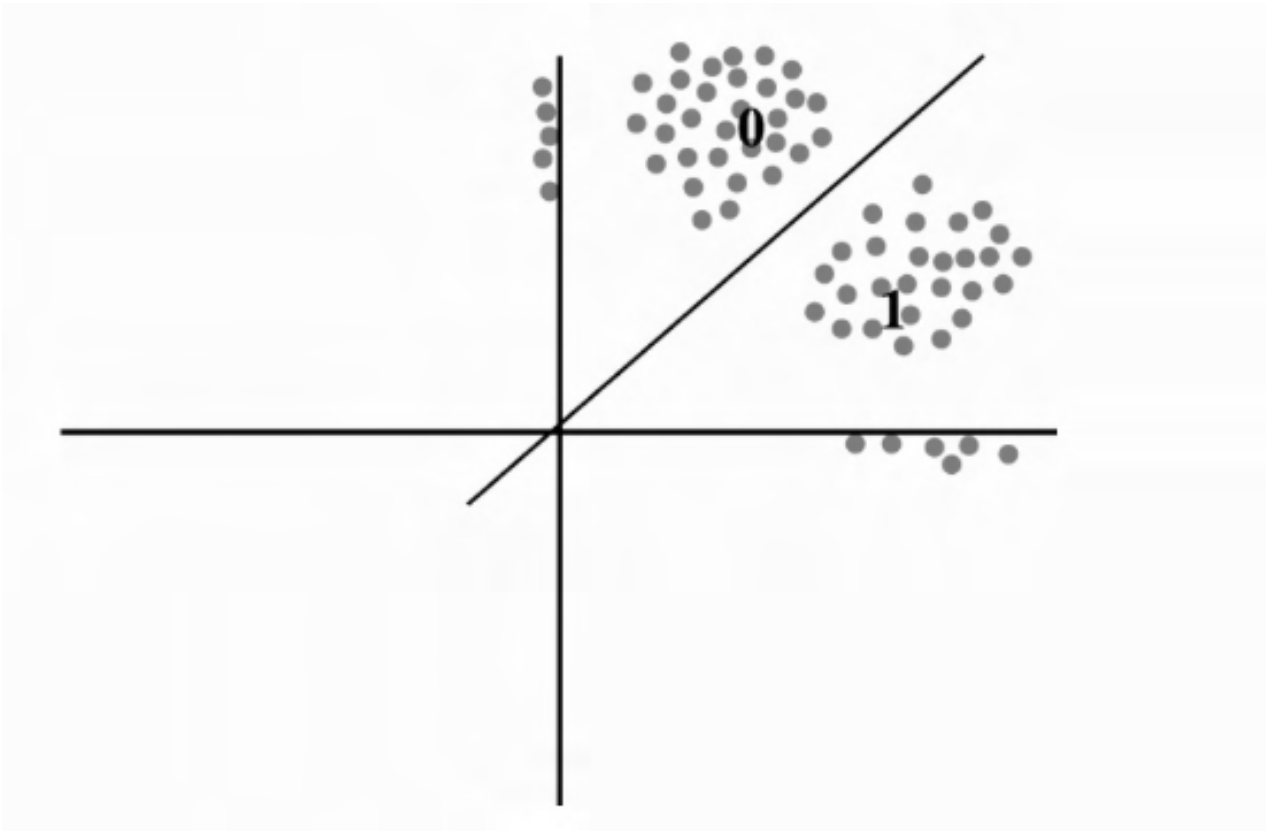
C = group0 ∪ group1

- assign to each group test vectors: vector0, vector1
- $\mathbf{y} \in$ group0: d($\mathbf{y}$,vector0) < d($\mathbf{y}$,vector1)
- $\mathbf{y} \in$ group1: d($\mathbf{y}$,vector1) < d($\mathbf{y}$,vector0)

Given a source vector $\mathbf{x}$

- first compare $\mathbf{x}$ with vector0 and vector1
- then compare $\mathbf{x}$ to the code-vectors from the group
- associated with the test vector closer to $\mathbf{x}$
- if both groups are about the same size
- we save about one half of the comparisons

Example – TSVQ



Tree-structured vector quantization

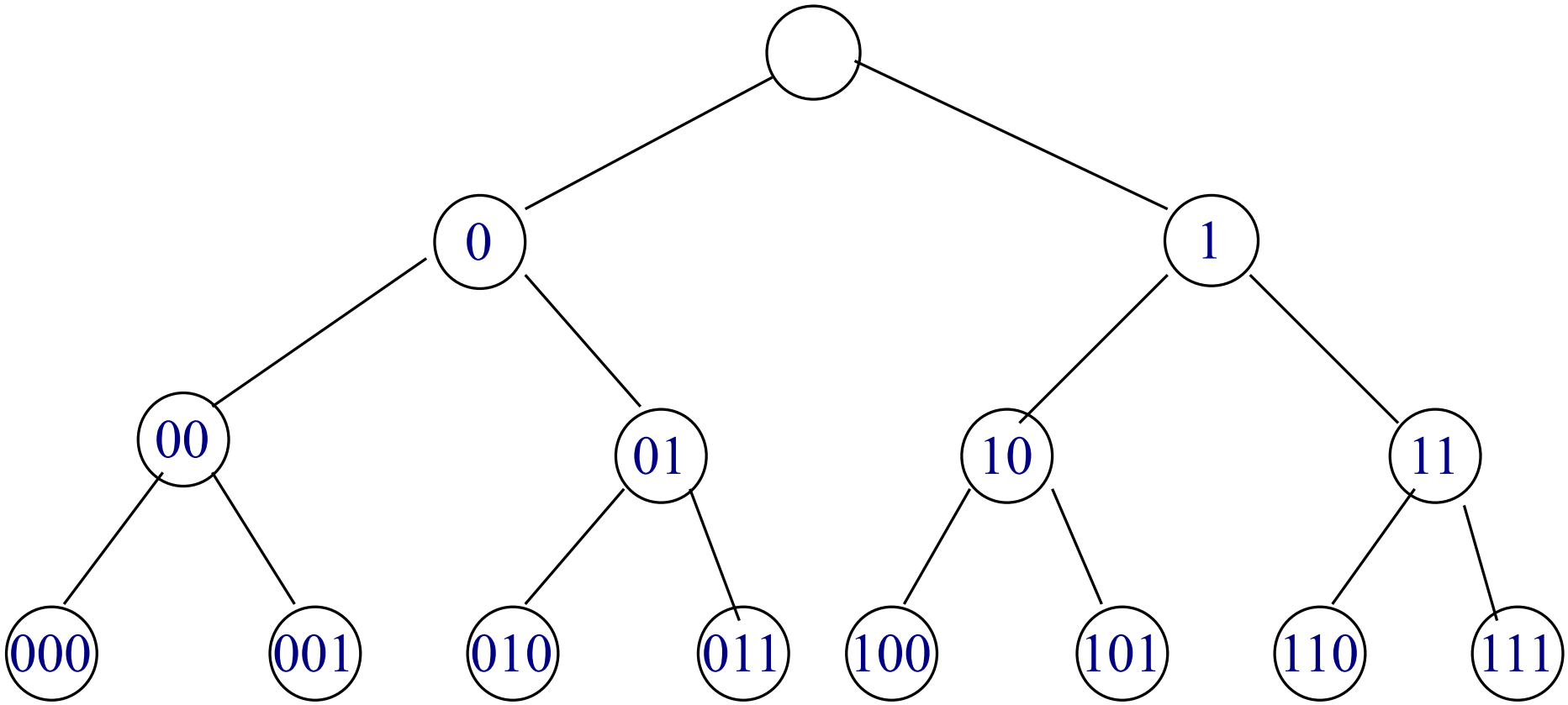
Continue in the same fashion

- group0 = group00 ∪ group01
- select test vectors vector00, vector01
- group1 = group10 ∪ group11
- select test vectors vector10, vector11

Continue this procedure

- successively dividing each group into two
- until the last set of groups consists of single points

TSVQ – decision tree



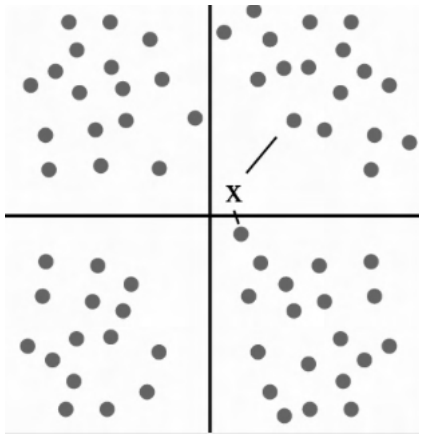
TSVQ – properties

✔ Pros

- number of comparisons : $K \rightarrow 2 \cdot \log_2 K$
- progress down → build a binary string

✖ Cons

- increased storage requirement
  - » codebook + test vectors
- possible increase in distortion



TSVQ – design

Idea: imitate the splitting technique

- given training set T, set $\mathbf{v}$ ={average of T}
- randomly generate perturbation vector $\boldsymbol{e}$
- LBG({$\mathbf{v}$, $\mathbf{v}+\boldsymbol{e}$}) → vector0, vector1
- # vector0, vector1 → test vectors
- split C = group0 ∪ group1 accordingly
- generate $\boldsymbol{e}$ # now we deviate from the splitting t.
- LBG({vector0, vector0+$\boldsymbol{e}$}, group0) → vector00, vector01
- split group0 = group00 ∪ group01 accordingly
- similarly use vector1 to determine vector10, vector11 as well as group10, group11
- continue until the required # of levels obtained

Pruned TSVQ

Idea: improve the rate of a TS codebook

- by pruning = removing carefully selected subgroups
- reduces codebook size

✖ May increase the distortion

- remove the subgroups → best tradeoff rate / distortion

Pruning

- decreases average rate (codeword length) ΔR
- increases average distortion ΔD
- prune the subtree with min $|\Delta D / \Delta R|$

☞ Binary codewords of variable length

- codewords ↔ paths in the decision tree
- → prefix code ✔

Lattice vector quantizers

Idea: regular arrangement of code-vectors in space

- quantization regions should tile the output space of the source (squares vs. circles)

Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_L$ be L independent L-dimensional vectors. Then the set

$$\{\sum_{i=1}^{L} u_i \mathbf{v}_i \mid u_i \in \mathbf{Z}\}$$

is called a lattice.

Lattice vector quantizer

- codebook = subset of lattice points

Example

- $\mathbf{v}_1 = (0,1)$, $\mathbf{v}_2 = (1,0)$
- integer lattice

Lattice vector quantizers

$$\{\sum_{i=1}^{L} u_i \mathbf{v}_i \mid u_i \in \mathbf{Z}\}$$

Basis vectors $\mathbf{v}_i \rightarrow$ generator matrix G

☞ Each lattice point $\mathbf{x} = \mathbf{u}\,G$

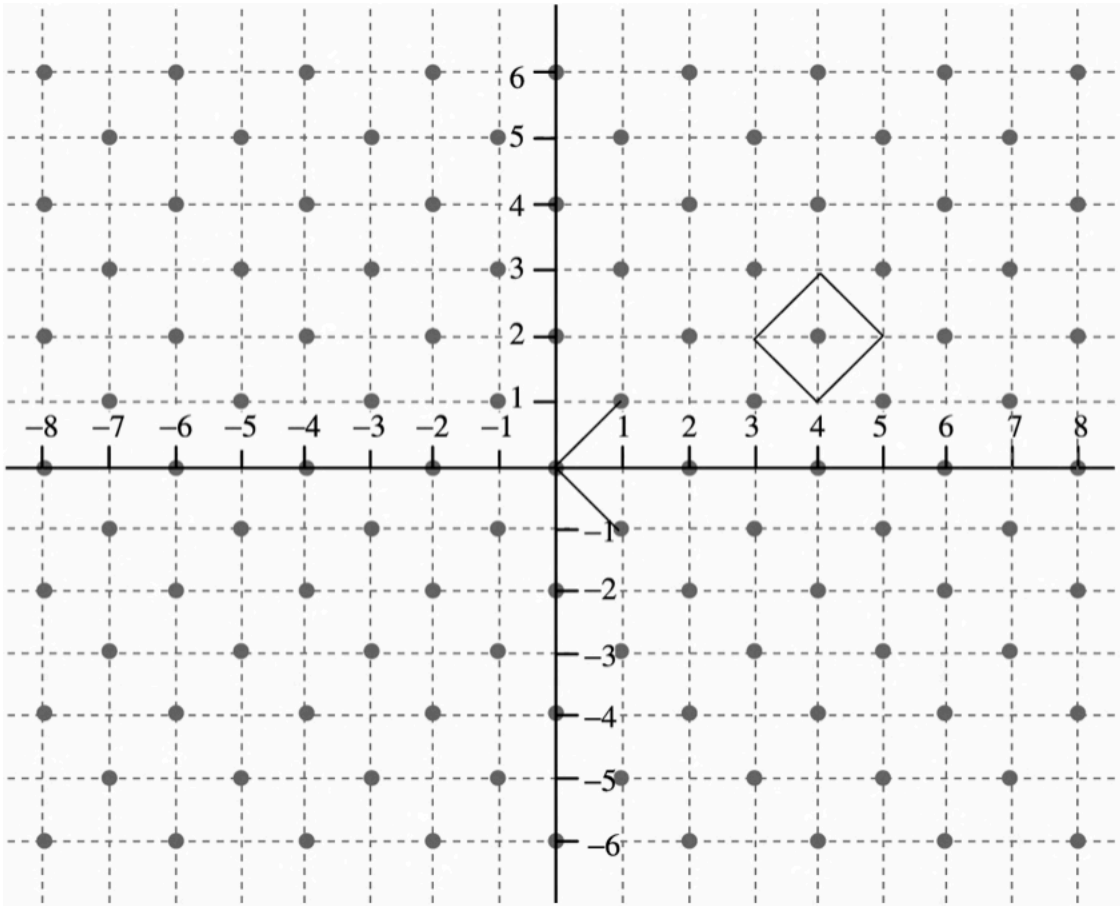
Row of G are independent $\rightarrow$ G^{-1} exists

☞ Given a lattice point $\mathbf{x}$, we can recover $\mathbf{u}$

$\mathbf{u} = \mathbf{x}\,G^{-1}$

Lattice vector quantizers – examples

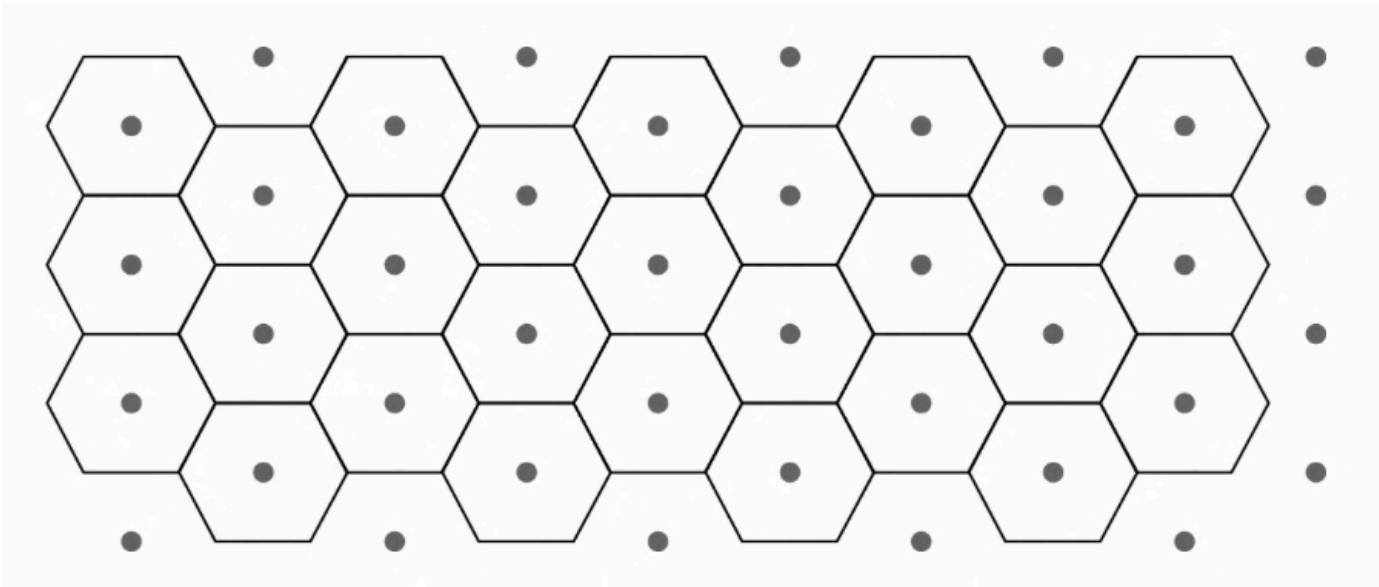
$\mathbf{v}_1=(0,2)$ $\mathbf{v}_2=(1,1)$



D_2 lattice

Lattice vector quantizers – examples

$\mathbf{v}_1=(1,0)$ $\mathbf{v}_2=(-1/2,\sqrt{3}/2)$



A_2 lattice

G.719

ITU-T standard for wideband audio coding

- 20 Hz - 20 kHz bandwidth
- sampling frequency 48kHz

Modified DCT

Vector quantizer based on D_8 lattice

$$G = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad G^{-1} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Classified vector quantization

Idea: divide source vectors into classes with different properties

- separate vector quantizers for different classes

Example: image compression

- edges / nonedge regions
- separate vector quantizers for each class

Encoding

- source vector tested for edges
- check the variance of pixels
  - » large variance ⇒ presence of an edge
- vector classified → corresponding codebook used
- transmit : codebook label + vector label

Vector quantization – applications

Audio

- CELP
- Ogg Vorbis
- G.719
- TwinVQ

Video

- QuickTime (Apple)
- VQA (Westwood Studios)

Data Compression Algorithms

Differential encoding

C. Chapin Cutler
(1914 - 2002)

☀ Idea

Speech & images

- high sample-to-sample correlation

Samples $\{x_n\}$ do not change too much $\rightarrow$

- sequence $\{d_n = x_n - x_{n-1}\}$
- has narrower range and variance than $\{x_n\}$

☞ Encode $\{d_n\}$ instead of $\{x_n\}$

Example

$\{x_n\} =$ 6.2, 9.7, 13.2, 5.9, 8, 7.4, 4.2, 1.8

$\{d_n\} =$ 6.2, 3.5, 3.5, -7.3, 2.1, -0.6, -3.2, -2.4

7-level quantizer with output -6, -4, -2, 0, 2, 4, 6

$\{\hat{d}_n\} =$	6	4	4	-6	2	0	-4	-2
$\{\hat{x}_n\} =$	6	10	14	8	10	10	6	4
$\{x_n - \hat{x}_n\} =$	0.2	-0.3	-0.8	-2.1	-2	-2.6	-1.8	-2.2

$$\hat{d}_n = d_n + q_n \text{ then } \hat{x}_n = x_n + \sum_{i=1}^{n} q_i$$

☞ Solution: $d_n = x_n - \hat{x}_{n-1}$, then $\hat{x}_n = x_n + q_n$

Basic algorithm

We need d_n as small as possible

Instead of $\hat{x}_{n-1}$ use a more general function f for prediction

$$p_n = f(\hat{x}_{n-1}, \hat{x}_{n-2}, \ldots, \hat{x}_0)$$

$$d_n = x_n - p_n$$

DPCM

C. Chapin Cutler, Bell Labs (1952)

Minimize

$$\sigma^2 = \mathrm{E}\left[(x_n - p_n)^2\right]$$

Linear predictor

$$p_n = \sum_{i=1}^{N} a_i \hat{x}_{n-i}$$

Assumption of „sufficiently subtle“ quantization

$$p_n = f(x_{n-1}, x_{n-2}, \ldots, x_0)$$

Problem: Find a_j which minimize σ^2

Set the derivative σ^2 w.r.t $a_j = 0$

Wiener-Hopf equations

$$R_{xx}(k) = E[x_n\, x_{n+k}]$$

$$RA = P$$

$$R = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) & R_{xx}(2) & \cdots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-2) \\ R_{xx}(2) & R_{xx}(1) & R_{xx}(0) & \cdots & R_{xx}(N-3) \\ \vdots & \vdots & \vdots & & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & R_{xx}(N-3) & \cdots & R_{xx}(0) \end{bmatrix}$$

$$A = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \qquad P = \begin{bmatrix} R_{xx}(1) \\ R_{xx}(2) \\ \vdots \\ R_{xx}(N) \end{bmatrix}$$

Estimate autocorrelation coefficients

Estimate $R_{xx}(k)$, M input values

$$R_{xx}(k) = \frac{1}{M-k} \sum_{i=1}^{M-k} x_i x_{i+k}$$

Assumption: stationary process

ADPCM

DPCM-APF (forward adaptive prediction)

Estimate → blocks

- speech - 16ms
- images - 8×8

Autocorrelation coefficients for each block

ADPCM

DPCM-APB (backward adaptive prediction)

1st order predictor

$$d_n^2 = (x_n - a_1 \hat{x}_{n-1})^2$$

$$a_1^{(n+1)} = a_1^{(n)} + \alpha \hat{d}_n \hat{x}_{n-1}$$

Nth order predictor

$$a_j^{(n+1)} = a_j^{(n)} + \alpha \hat{d}_n \hat{x}_{n-j}$$

LMS (least mean squared) algorithm

Matrix form $$A^{(n+1)} = A^{(n)} + \alpha \hat{d}_n \hat{X}_{n-1}$$

$$\hat{X}_n = \begin{bmatrix} \hat{x}_n \\ \hat{x}_{n-1} \\ \vdots \\ \hat{x}_{n-N+1} \end{bmatrix}$$

G.726

ITU-T standard for speech compression based on ADPCM

Sampling frequency 8kHz, resolution 8b (64kb/s)

Encoder output

- 5,4,3,2b/sample
- 40,32,24,16kb/s
- most popular 32 kb/s → compression ratio 2:1

Predictor

$$p_k = \sum_{i=1}^{2} a_i^{(k-1)} \hat{x}_{k-i} + \sum_{i=1}^{6} b_i^{(k-1)} \hat{d}_{k-i}$$

a_i, b_i update - simplified LMS algorithm

Backward adaptive quantizer (modified Jayant)

- $2^{b/s}$-1 levels, $b/s \in \{3,4,5\}$ (midtread)
- 4 levels for 2 b/s (midrise)

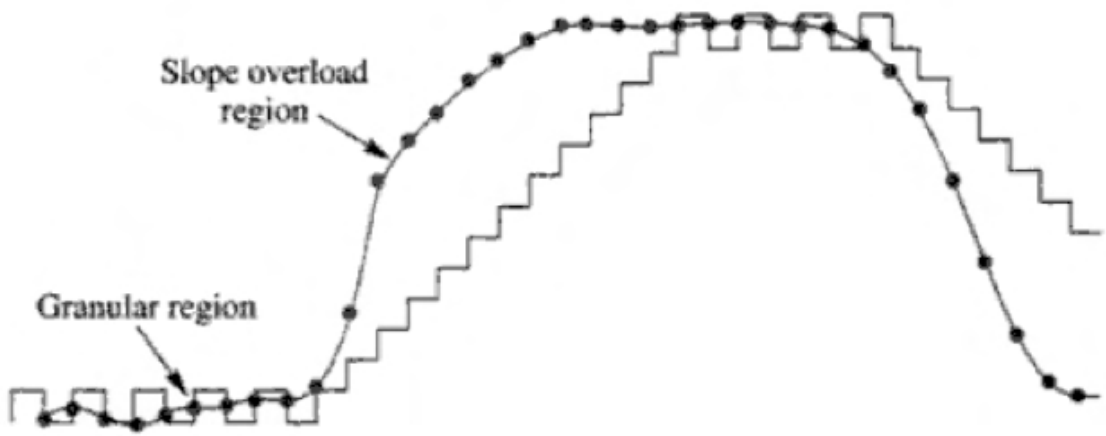
Delta modulation

Speech coding

DPCM with 2-level quantizer $(+\Delta, -\Delta)$

$f_s / 2f_{max} \in \langle 1,100 \rangle$

Delta modulation with fixed Δ



Problems

- granular region: constant input output alternates up / down by Δ
- slope overload region: input rises / falls fast reconstructed output cannot keep up

Solution

- adapt Δ to the characteristics of the input

Constant factor adaptive delta modulation (CFDM)

Objective

- $\Delta \uparrow$ in overload regions
- $\Delta \downarrow$ in granular regions

How to identify the regions?

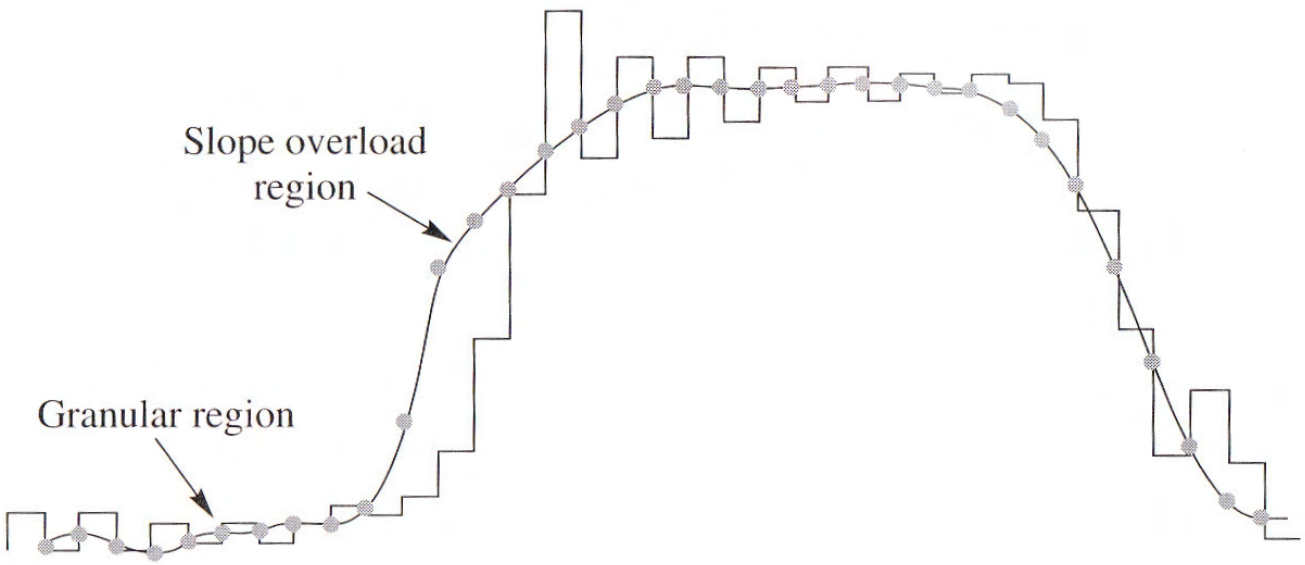
- simple solution: use a history of 1 sample

$$s_n = \begin{cases} 1 & \hat{d}_n > 0 \\ -1 & \hat{d}_n < 0 \end{cases}$$

$$\Delta_n = \begin{cases} M_1 \Delta_{n-1} & s_n = s_{n-1} \\ M_2 \Delta_{n-1} & s_n \neq s_{n-1} \end{cases}$$

$$2 > M_1 = \frac{1}{M_2} > 1$$

CFDM: input / output



CFDM with 2-sample memory

R.Steele (1975)

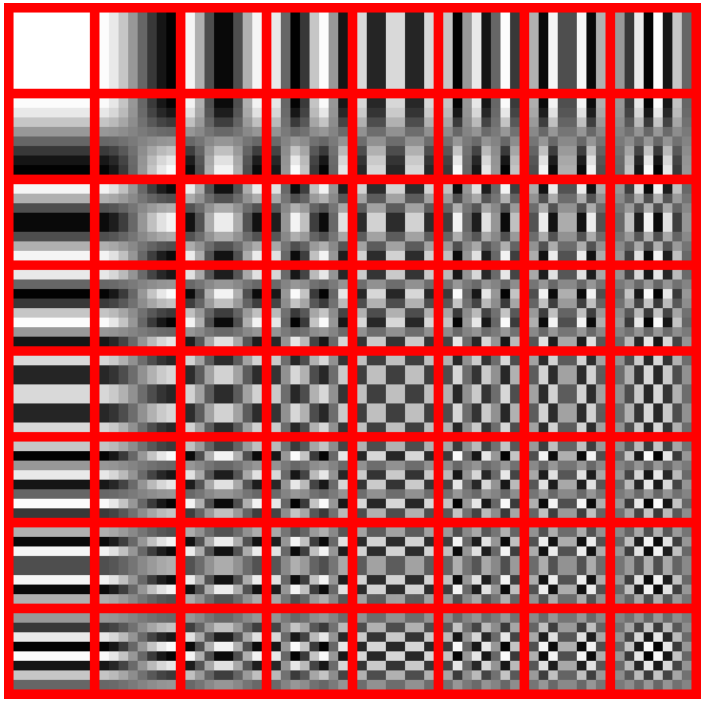
$$s_n \neq s_{n-1} = s_{n-2} \qquad M_1 = 0.4$$
$$s_n \neq s_{n-1} \neq s_{n-2} \qquad M_2 = 0.9$$
$$s_n = s_{n-1} \neq s_{n-2} \qquad M_3 = 1.5$$
$$s_n = s_{n-1} = s_{n-2} \qquad M_4 = 2.0$$

Application: space shuttle $\leftrightarrow$ ground terminal

- CDFM with 7-sample memory

Data Compression Algorithms

Transform coding



Discrete Cosine Transform

Scheme

$\{x_n\} \rightarrow$ blocks of size N

Transform each block

Quantization

Coding

Transforms

$\{x_n\} \rightarrow \{\theta_n\}$

Linear transform

$$\theta_n = \sum_{i=0}^{N-1} a_{n,i} x_i \qquad \theta = \mathbf{A}x \qquad x = \mathbf{A}^{-1}\theta$$

Images → 2-dimensional transform

Separable transform

$$\theta_{k,l} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_{i,j,k,l} x_{i,j}$$

- first transform in one direction (rows)
- then repeat the operation along the other direction (columns)

$$\theta_{k,l} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_{k,i} a_{l,j} x_{i,j} \qquad \Theta = \mathrm{AXA}^{\mathrm{T}} \qquad \mathrm{X} = \mathrm{A}^{-1}\Theta\left(\mathrm{A}^{-1}\right)^{\mathrm{T}}$$

Orthogonal transform

Ortogonal matrix $A^{-1} = A^T$

If A is orthogonal, then $X = A^T \Theta A$

Moreover $$\sum_{i=0}^{N-1} \theta_i^{\,2} = \sum_{i=0}^{N-1} x_i^{\,2}$$

Gain G_T of a transform T

$$G_T = \frac{\frac{1}{N}\sum_{i=0}^{N-1} \sigma_i^{\,2}}{\left(\prod_{i=0}^{N-1} \sigma_i^{\,2}\right)^{\frac{1}{N}}}$$

σ_i^2 = variance of coefficient θ_i

Example

$$A = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$(10,10) \rightarrow (10\sqrt{2},0)$$
$$(3,1) \rightarrow (2\sqrt{2},\sqrt{2})$$
$$(3,-1) \rightarrow (\sqrt{2},2\sqrt{2})$$

Karhunen-Loéve transform

Minimizes the geometric mean of the variance of the transform coefficients

Transform matrix consists of eigenvectors of the autocorrelation matrix R

- $R = (r_{ij})$, $r_{ij} = E[x_n x_{n+(i-j)}]$

✎ Problem

- non-stationary input $\Rightarrow$
- matrix R changes with time $\Rightarrow$
- KLT needs to be recomputed

Discrete cosine transform

$A=(a_{i,j})$

$$a_{i,j} = \begin{cases} \sqrt{\frac{1}{8}} \cos \frac{(2j+1)i\pi}{16} & i=0,\ j=0,\dots,7 \\ \sqrt{\frac{1}{4}} \cos \frac{(2j+1)i\pi}{16} & i=1,\dots,7,\ j=0,\dots,7 \end{cases}$$

N=8

- X input block 8×8
- θ output block 8×8

$$\theta_{k,l} = \sum_{i=0}^{7} \sum_{j=0}^{7} a_{k,i} a_{l,j} x_{i,j} \qquad \Theta = \mathrm{AXA}^{\mathrm{T}} \qquad \mathrm{X} = \mathrm{A}^{-1} \Theta \left(\mathrm{A}^{-1}\right)^{\mathrm{T}}$$

G_{DCT} close to the optimum value

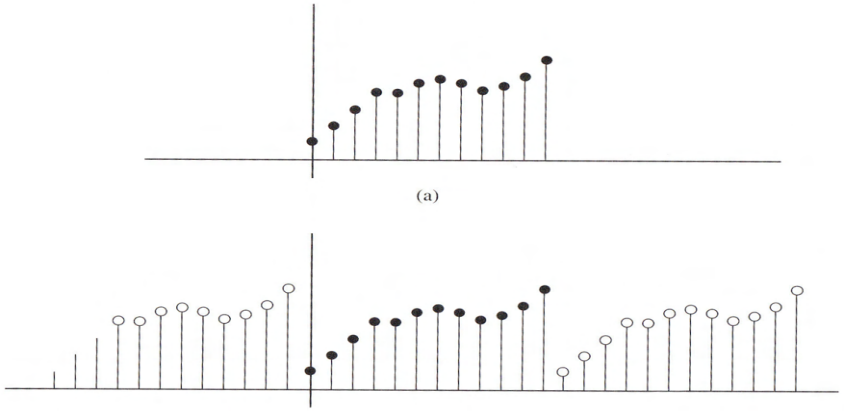
- for Markov sources
- with high correlation coefficient

$$\rho = \frac{\mathrm{E}[x_n x_{n+1}]}{\mathrm{E}[x_n^2]}$$

DCT – why better than DFT?

Relationship to the discrete Fourier transform DFT

- FT requires a periodic signal

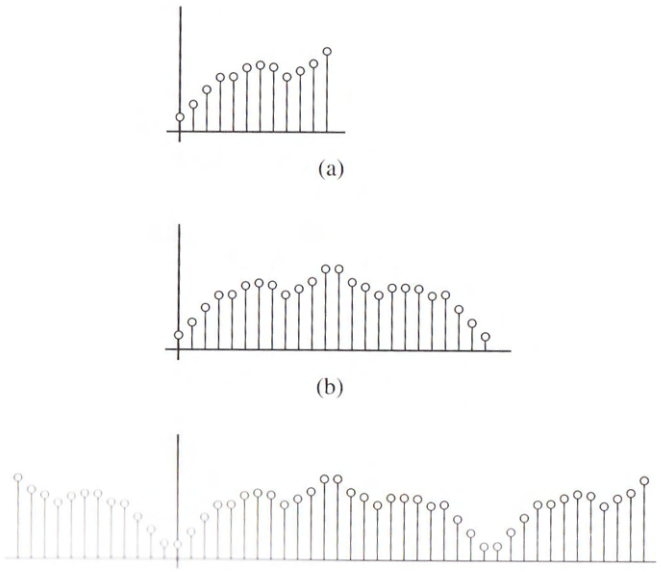


- periodic extension of non-periodic signal by DFT
- $\longrightarrow$ introduces sharp discontinuities
- $\longrightarrow$ non-zero high-frequency coefficients

DCT - why better than DFT?

DCT may be obtained from DFT

- "mirror" the original non-periodic signal
- N-point sequence → 2N-point sequence
- apply DFT

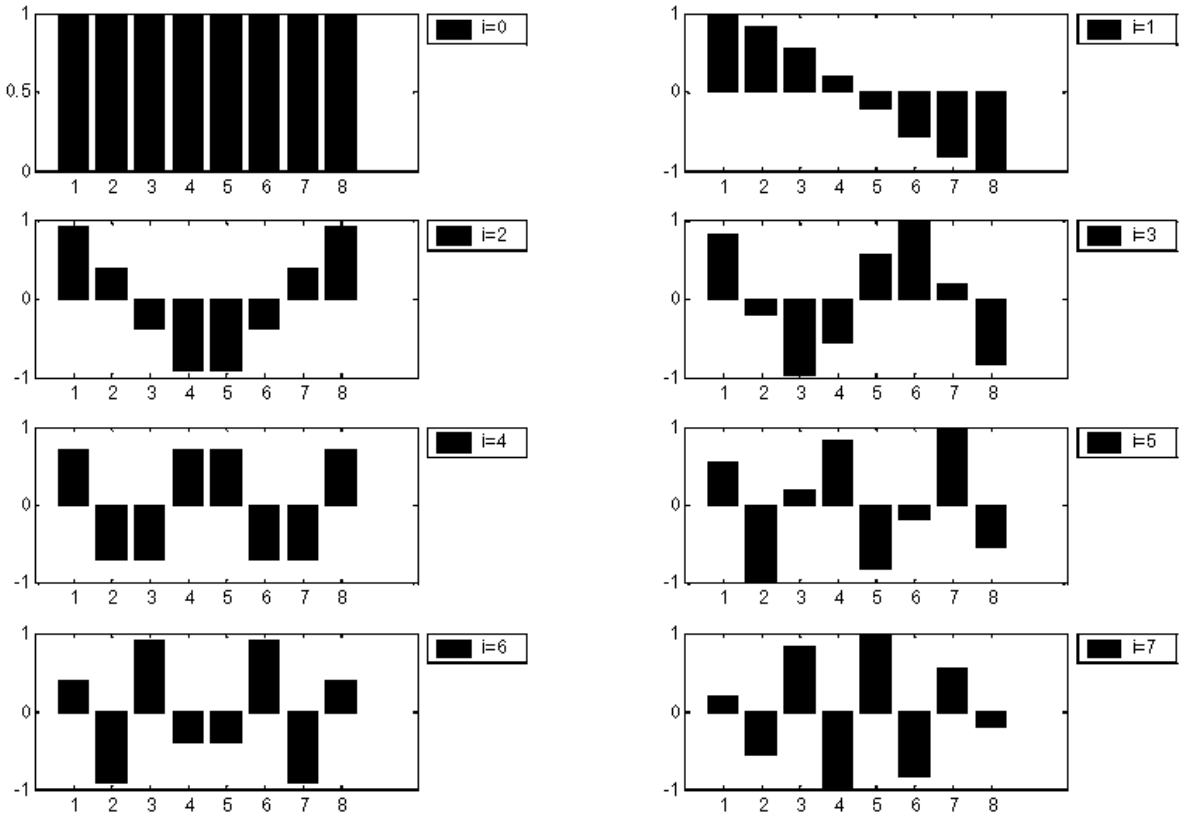


- take the first N points → output
- DCT = „cosine component" of DFT

DCT – properties

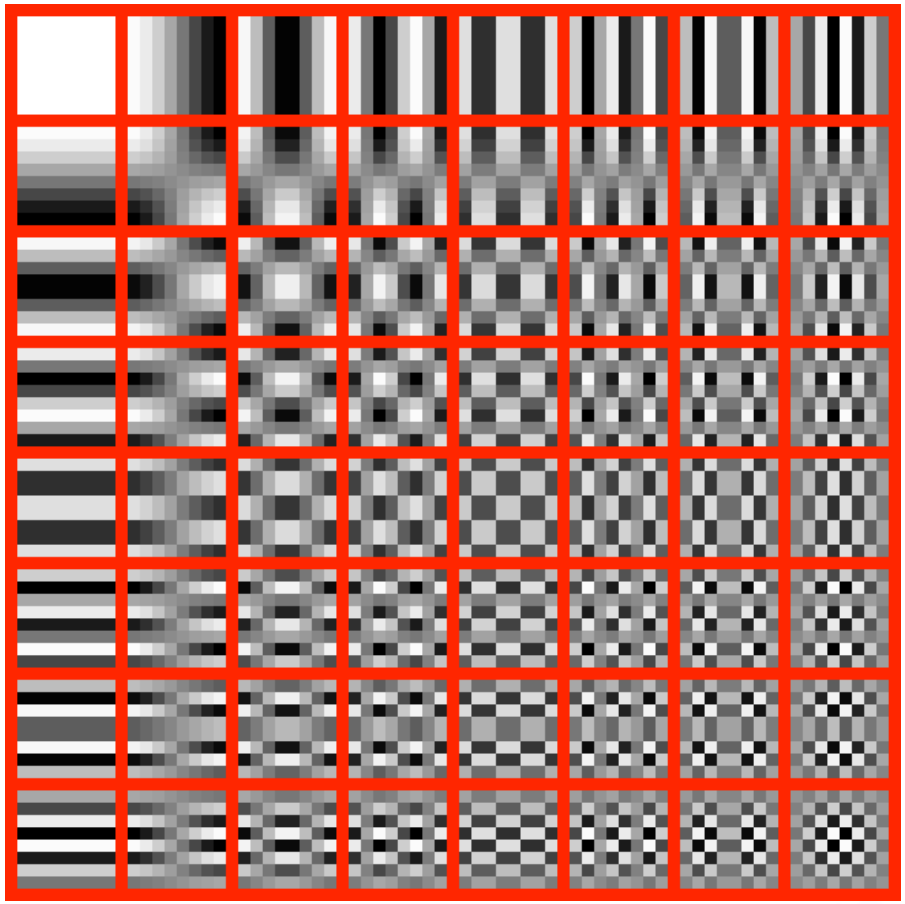
Transform matrix

- 1st row = constant vector
- following rows = vectors with higher variation



DCT: increasing variation $\theta_{1,1} \rightarrow \theta_{N\text{-}1,N\text{-}1}$

$$\theta_{k,l} = \sum_{i=0}^{7} \sum_{j=0}^{7} a_{k,i} a_{l,j} x_{i,j}$$



Discrete sine transform

$\mathbf{A}=(a_{i,j})$

$$a_{i,j} = \left\{ \sqrt{\frac{2}{N+1}} \sin \frac{\pi(i+1)(j+1)}{N+1} \right. \quad i,j = 0,\ldots,N-1$$

- G_{DST} close to the optimum value
  - for Markov sources
  - with low correlation coefficient

$$\rho = \frac{E\left[x_n x_{n+1}\right]}{E\left[x_n^{\ 2}\right]}$$

Hadamard matrices

Hadamard matrix H of order N

- square matrix of order N
- such that $HH^T = NI$
- I is the identity matrix of order N

$$H_1 = [1] \quad H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Discrete Walsh-Hadamard transform

DWHT transform matrix is a rearrangement of H_N

- multiply H_N by normalizing factor $1/\sqrt{N}$
- reorder the rows by increasing number of sign changes

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \dashrightarrow \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

✔ Pro: speed

✘ Con: $G_{DWHT} << G_{DCT}$

Quantization of transform coefficients

Zonal sampling

- allocation of bits to each coefficient
- coefficients with higher expected variance
  $\longrightarrow$ more bits

8	7	5	3	1	1	0	0
7	5	3	2	1	0	0	0
4	3	2	1	1	0	0	0
3	3	2	1	1	0	0	0
2	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization of transform coefficients

Zonal sampling

- allocation of bits to each coefficient
- coefficients with higher expected variance
  → more bits

✔ Pro: simplicity

✘ Con: bit allocations based on average values

- local variations need not be reconstructed properly
- example: sharp edge on a plain background

Quantization and coding

Threshold coding

- which coefficients kept / discarded
- **not** decided a priori

Pratt:

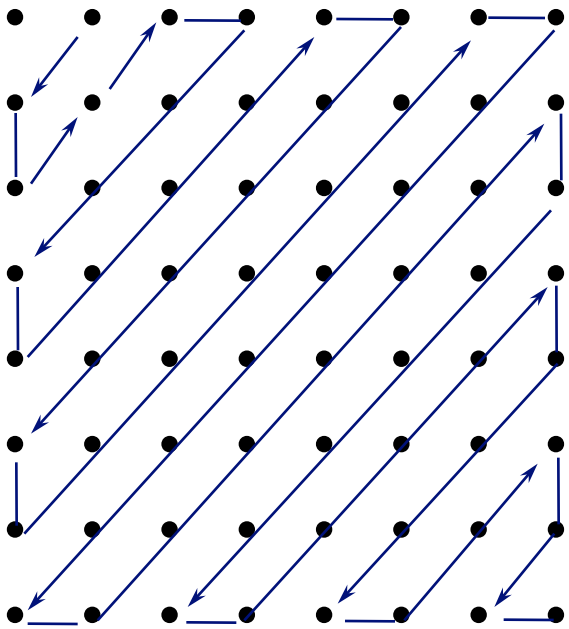
if x_n is 1st in the block **then** encode x_n

else if $x_n > \mathit{threshold}$ **then** encode x_n

output # coefficients
since the last value > *threshold*

Quantization and coding

Chen & Pratt: *zigzag scan*



Tail of the scan should consist of zeros

- special *end-of-block* symbol EOB

Application: JPEG

Joint Photographic Experts Group

- 1986: founded
- 1992: standard
- 1994: ISO 10918-1

JPEG standard specifies a *codec*

- lossy / lossless compression
- progressive transmission scheme

File format

- JPEG/JFIF (File Interchange Format)
  - » minimal format version
- JPEG/Exif (Exchangeable image file format)
  - » digital photography

Use

- JPEG: digitized images
- GIF/PNG: synthetic images

JPEG compression scheme

Color space transform

- RGB $\rightarrow$ YC_BC_R
- human eye more sensitive to details
  - » in Y (luminance) component
  - » rather than chroma C_B (blue) a C_R (red) components

JPEG compression scheme

Reduction of colors (*chroma subsampling*)

- color components only, Y does not change
- 4:4:4
- 4:2:2
  - » 2h1v: average neighboring pairs
  - » saves 33% space
- 4:2:0
  - » 2h2h: average neighboring quadruples
  - » saves 50% space

JPEG compression scheme

Color image: apply to each component

Split into blocks

- level shift $x := x - 2^{P-1}$
- split into blocks of 8×8 pixels
- duplicate the last row / column if necessary

DCT of each block

JPEG compression scheme

DCT coefficients quantization

- uniform quantizer for each coefficient
- each coefficient divided by the corresponding table value + rounded
  - » standard recommends tables for luminance / chroma components
  - » utilities (Photoshop, GIMP,…) allow user to specify quality $Q \in \{1,...,100\}$
    - quantization table values (correspond to e.g. Q=75) are adjusted accordingly
  - » tables stored in the compressed file

Coding

- zigzag matrix scan
- Huffman code / arithmetic coding

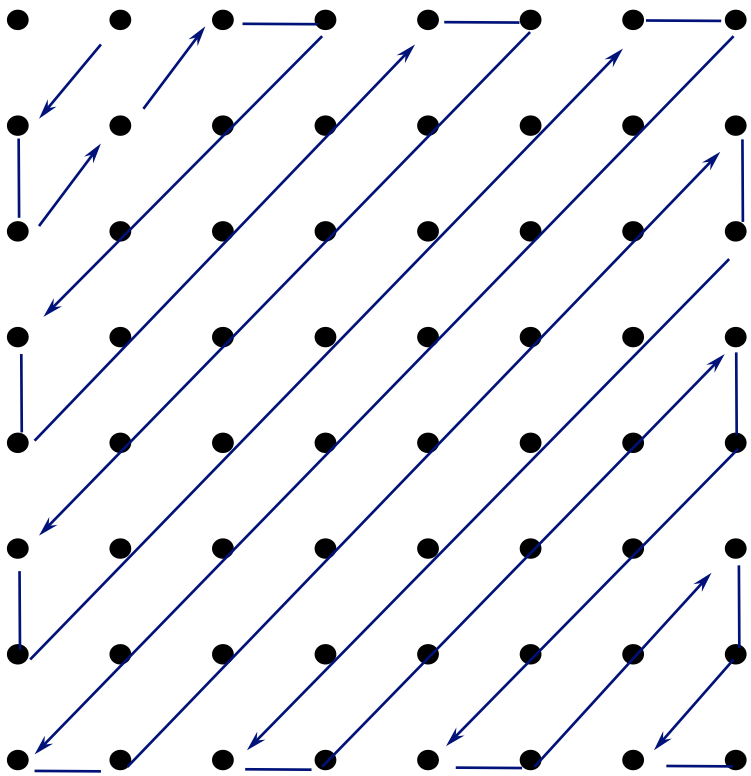
JPEG – quantization

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Quantization $\theta_{ij} \rightarrow l_{ij} \quad l_{ij} = \left\lfloor \frac{\theta_{ij}}{Q_{ij}} + 0.5 \right\rfloor$

Reconstruction $l_{ij} \rightarrow \theta_{ij} \;\; \theta_{ij} = l_{ij} Q_{ij}$

Zigzag matrix scan



Example: level shift, DCT

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} \quad \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Example: quantization,zigzag scan

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$-26, -3, 0, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1,$
$1, -1, 2, 0, 0, 0, 0, 0, -1, -1,$ EOB

JPEG – DC coefficient encoding

θ_{11} = constant multiple of the average value in the block

- DC coefficient / AC coefficient

Instead of θ_{11} encode the difference between neighboring blocks

Split the range of values into categories

- let θ_{11} fall into category n

code(θ_{11}) = Huffman code of n & n bits

- which identify the value of θ_{11} within the nth category

JPEG – coding

0			0			
1			-1	1		
2		-3	-2	2	3	
3	-7	...	-4	4	...	7
4	-15	...	-8	8	...	15
5	-31	...	-16	16	...	31
6	-63	...	-32	32	...	63
...						
15	-32767	...	-16384	16384	...	32768
16			32768			

JPEG – AC coefficients

Let θ_{ij} fall into category n

Let z be # zero coefficients since the last non-zero, $z \in \langle 0,15 \rangle$

code(θ_{ij}) = Huffman code (z/n) & n bits identifying the coefficient within the nth category

Special symbols

- EOB *end-of-block*
- ZRL *more than 15 zero coefficients*

Extended JPEG: arithmetic coding

JPEG – use

Digitized images

Realistic scenes

Subtle variations in tone and color

✖ Disadvantage

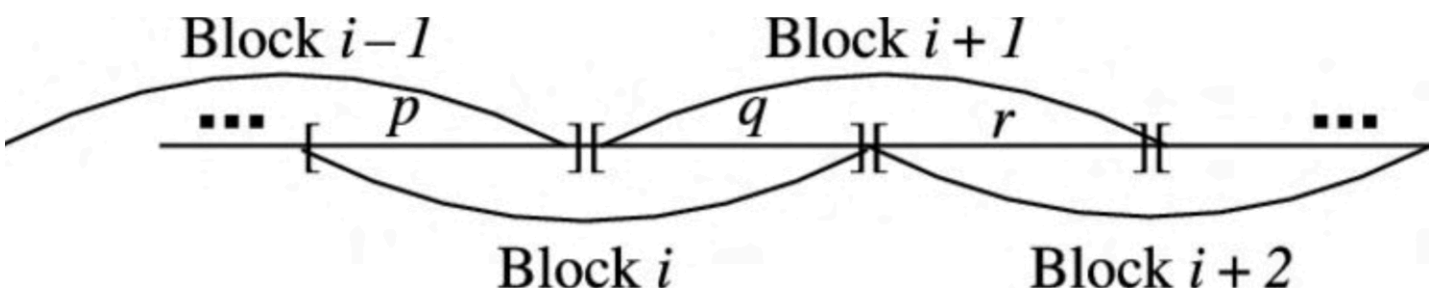
- artificial division into blocks
- → coding artifacts at the block edges
- → tile effect

Overlapped transforms

Block-based transform → overlapping blocks

Modified DCT (MDCT)

- 50% overlap
- each block overlaps half
- of the previous / next block



Modified DCT

✎ Problem

- 2x coefficients as samples
- reduce the number of coefficients
- (sub) sampling at less than Nyquist frequency
- → aliasing

☀ Solution

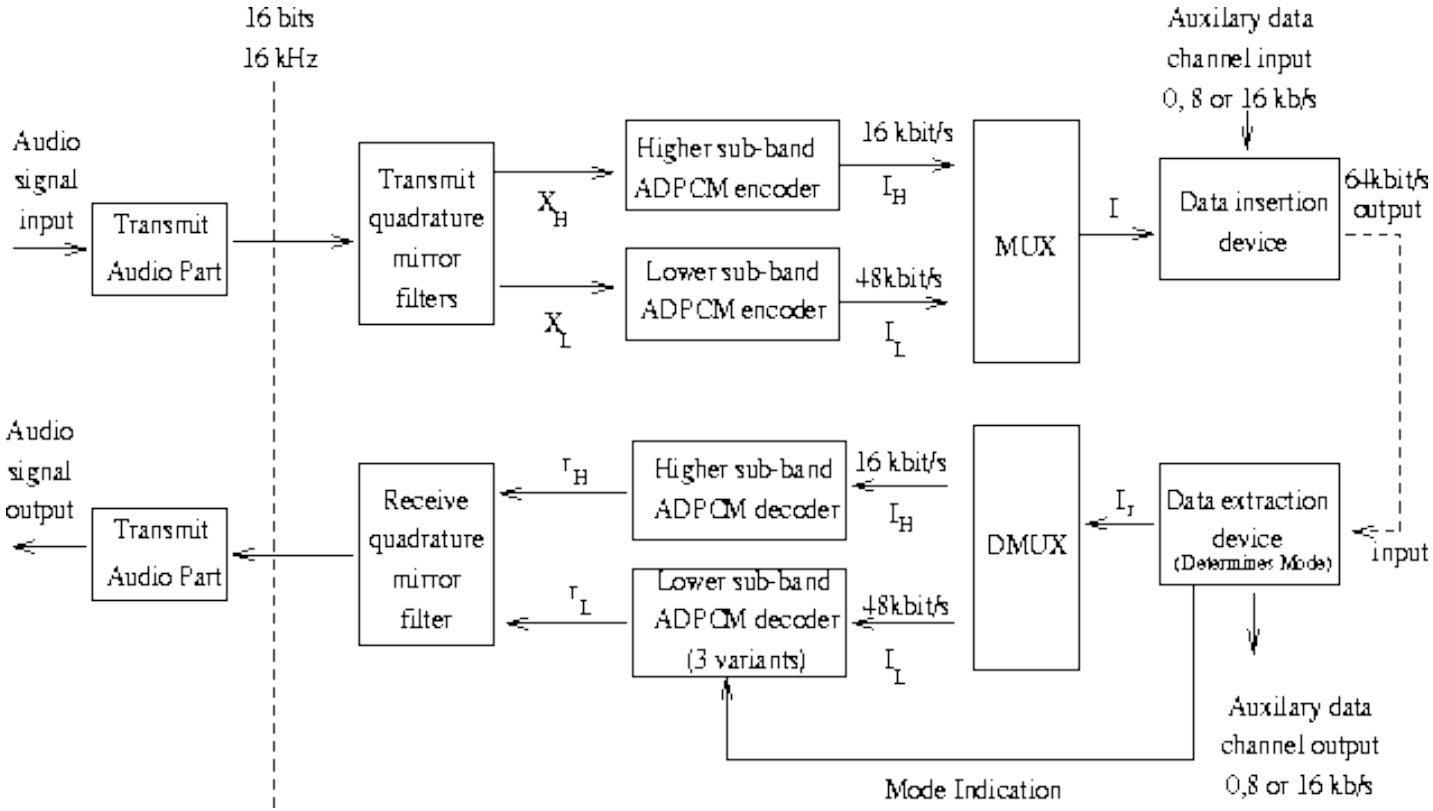
- construct the lapped transforms so that
- aliasing in consecutive blocks
- cancels each other out

Application: audio compression

- mp3, AAC, Ogg Vorbis

Data Compression Algorithms

Subband coding

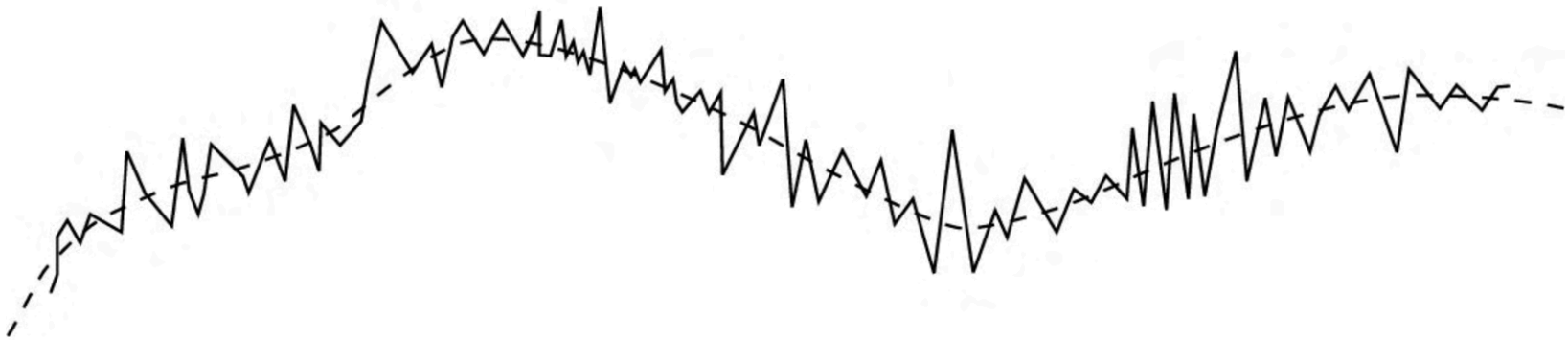


Example

$\{x_n\}$	=	10	14	10	12	14	8	14	12	10	8	10	12
$\{x_n - x_{n-1}\}$	=	10	4	-4	2	2	-6	6	-2	-2	-2	2	2

- $\in \langle -6,6 \rangle$
- uniform M-level quantizer
- $\Delta = 12/M$, max error $6/M$

✎ Counterexample



✖ Rapid sample-to-sample variation

✔ Long-term trend varies slowly

☀ Idea

- smooth out the rapid variation
- by averaging
- in a sliding window

Example

$y_n = (x_n + x_{n-1})/2$

$\{y_n\} = 10\ 12\ 12\ 11\ 13\ 11\ 11\ 13\ 11\ 10\ 9\ 11$

$\{y_n - y_{n-1}\} = 10\ 2\ 0\ -1\ 2\ -2\ 0\ 2\ -2\ -1\ -1\ 2$

- $\in \langle -2,2 \rangle$
- $\Delta = 4/M$, max error $2/M$

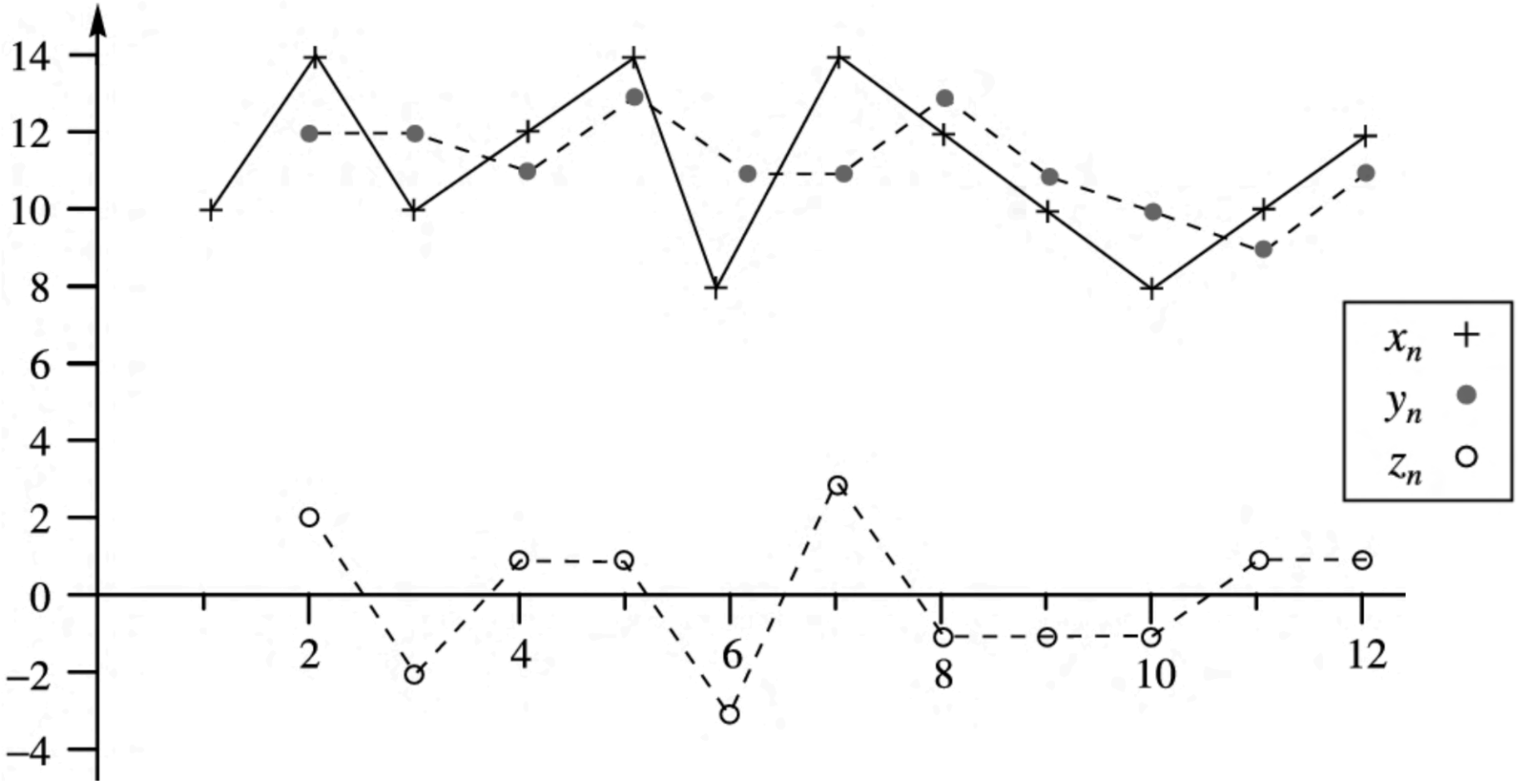
$z_n = x_n - y_n = (x_n - x_{n-1})/2$

$\{z_n\} = 0\ 2\ -2\ 1\ 1\ -3\ 3\ -1\ -1\ -1\ 1\ 1$

- $\in \langle -3,3 \rangle$
- $\Delta = 6/M$, max error $3/M$

$2/M + 3/M = 5/M < 6/M$

Example



Example

Problem

- max error 5/M
- **but** twice as many samples!

Solution

- $\{y_n\} \rightarrow \{y_{2n}\}$
- $\{z_n\} \rightarrow \{z_{2n}\}$

Reconstruction of $\{x_n\}$:

$$y_{2n} = (x_{2n} + x_{2n-1})/2$$
$$z_{2n} = (x_{2n} - x_{2n-1})/2$$
$$y_{2n} + z_{2n} = x_{2n}$$
$$y_{2n} - z_{2n} = x_{2n-1}$$

Instead of $\{y_n - y_{n-1}\}$ we need $\{y_{2n} - y_{2n-2}\}$

- the range of $y_{2n} - y_{2n-2}$ is still 4
- y_{2n} & z_{2n} still incurs less distortion than x_n

Filters

Filter isolates certain frequency components of a signal

Example: coffee filter

- blocks coarse particles
- allows the finer-grained components to pass through

Our filters: selectively let through/block any range of frequencies

- low-pass filter: lets through components below a cutoff frequency f_0
- high-pass filter: blocks all frequency components below f_0
- band-pass filter: lets through components between f_1 and f_2

Digital filters

- operate on a sequence $\{x_n\}$ of numbers
- usually samples of a continuous signal

Filters

Digital filter with

- input $\{x_n\}$
- output $\{y_n\}$
- coefficients $\{a_i\}$ a $\{b_i\}$

$$y_n = \sum_{i=0}^{N} a_i x_{n-i} + \sum_{i=1}^{M} b_i y_{n-i}$$

FIR finite impulse response

- $b_i = 0$ for all i
- impulse response dies out after N samples
- N – number of taps

IIR infinite impulse response

Example

$a_0 = 1.25$ $a_1 = 0.5$

Impulse response sequence: $\{h_n\} = \{1.25, 0.5, 0, \ldots\}$

$a_0 = 1$ $b_1 = 0.9$

Impulse response: $\{h_n\} = \{1, 0.9, 0.81, \ldots\}$

Convolution

Impulse response $\{h_k\}_{k=0}^{M}$ completely specifies the filter:

$$y_n = \sum_{k=0}^{M} h_k x_{n-k}$$

M is finite for FIR filters, infinite for IIR filters

✎ Example

$\{x_n\}$ = 10 14 10 12 14 8 14 12 10 8 10 12

$y_n = (x_n + x_{n-1})/2$

$\{y_n\}$ = 10 12 12 11 13 11 11 13 11 10 9 11

☞ averaging / low-pass filter

- two-tap FIR filter
- impulse response $\{h_n\} = \{0.5, 0.5, 0, \ldots\}$

$z_n = x_n - y_n = (x_n - x_{n-1})/2$

$\{z_n\}$ = 0 2 -2 1 1 -3 3 -1 -1 -1 1 1

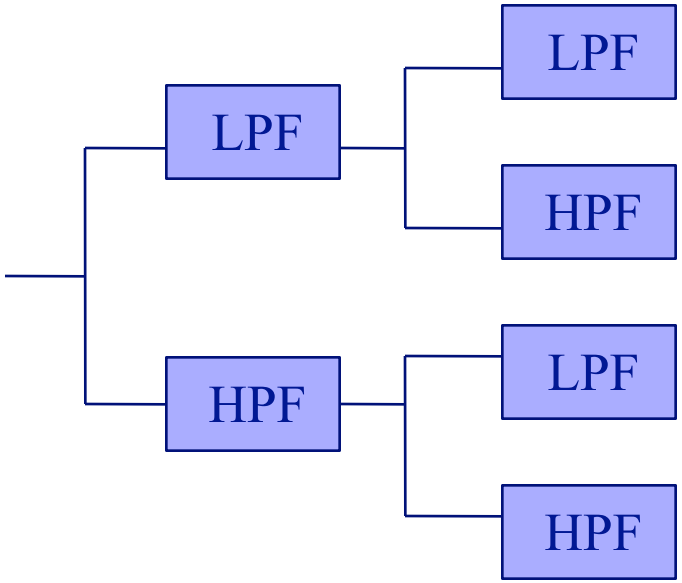
☞ difference / high-pass filter

- impulse response $\{h_n\} = \{0.5, -0.5, 0, \ldots\}$

Filters used in subband coding

Filter bank

- a cascade of stages
- each stage: low-pass & high-pass filters



Filters used in subband coding

QMF - quadrature mirror filter

- impulse response of the low-pass filter $\{h_n\}$
- high-pass impulse response $\{(-1)^n h_{N-1-n}\}$

✎ Example

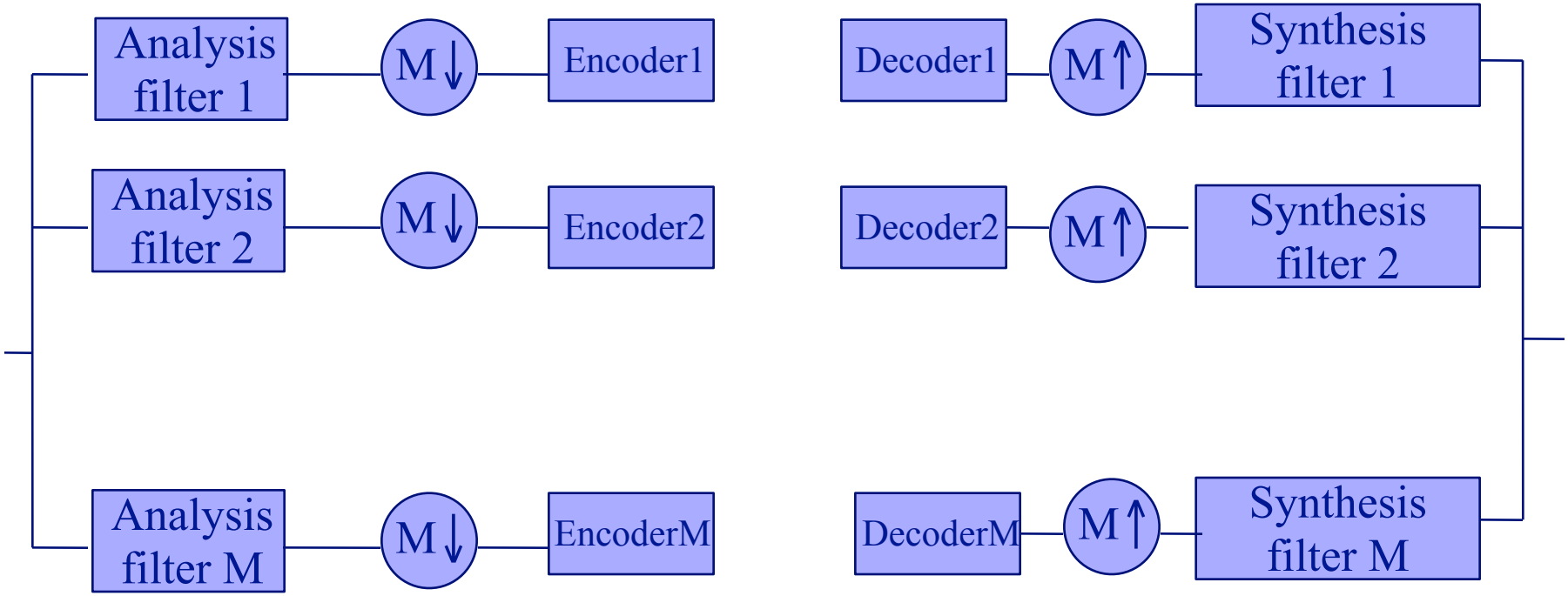
8-tap Johnston QMF filter

h_0, h_7	0.00938725
h_1,h_6	0.06942827
h_2,h_5	-0.07065183
h_3,h_4	0.48998080

Properties

- fewer taps → lower efficiency in decomposition
- # taps → # operations to generate the output

Basic subband coding algorithm



Basic algorithm – description

1. Analysis
   - Filter 1,...,Filter M cover the frequency range of the input
   - Nyquist rule: $f_s \geq 2f_{max}$
   - generalized Nyquist rule: $\langle f_1, f_2 \rangle$, $f_s \geq 2(f_2 - f_1)$
2. Downsampling (decimation) M↓
   - bandwidth of filter output=1/M ∗ bandwidth of filter input
   - keep each Mth sample
3. Quantization + coding
   - allocation of bits between subbands
   - ADPCM, PCM, vector quantization

Basic algorithm – description

4. Decoding
5. Upsampling M↑
   - insert appropriate number of 0's
6. Synthesis
   - bank of reconstruction filters

Summary: 3 major components

- analysis & synthesis filters
- bit allocation scheme
- encoding scheme

Bit allocation

A heuristic for bit allocation

Assumption: M subbands of equal bandwidth

$\sigma^2_{y_k}$ - k^{th} subband variance

Estimate $\sigma^2_{y_k}$ for each k

$R_k = 0$ for each k, total # of bits R_b

while $R_b > 0$ **do** $\sigma^2_{y_i} = \max\{\sigma^2_{y_k}\}$

R_i ++

$\sigma^2_{y_i} := \sigma^2_{y_i} /2$

R_b--

Application – G.722

ITU recommendation G.722 for speech coding

- objective: high-quality speech at 64 (56,48) kb/s
- antialiasing filter with cutoff frequency 7kHz
- sampling frequency 16kHz
- 14b uniform quantizer

Filtering: a bank of 2 QMF filters

- 24 tap FIR filter
- low-pass 0-4kHz, high-pass remaining frequencies

Downsampling by a factor of 2

Application – G.722

Encoding: ADPCM

- low-pass: 6b/sample
  - » possibility of dropping 1 / 2 least significant bits
- high-pass: 2b/sample

ADPCM: adaptation and prediction

$$\hat{x}_{n-1}, \hat{x}_{n-2}, \hat{d}_{n-1}, \ldots, \hat{d}_{n-6}$$

Quantization

- a variation of Jayant algorithm

At the receiver

- ADPCM decoder
- upsampling: insert 0 after each sample
- reconstruction filters (identical to decomposition filters)

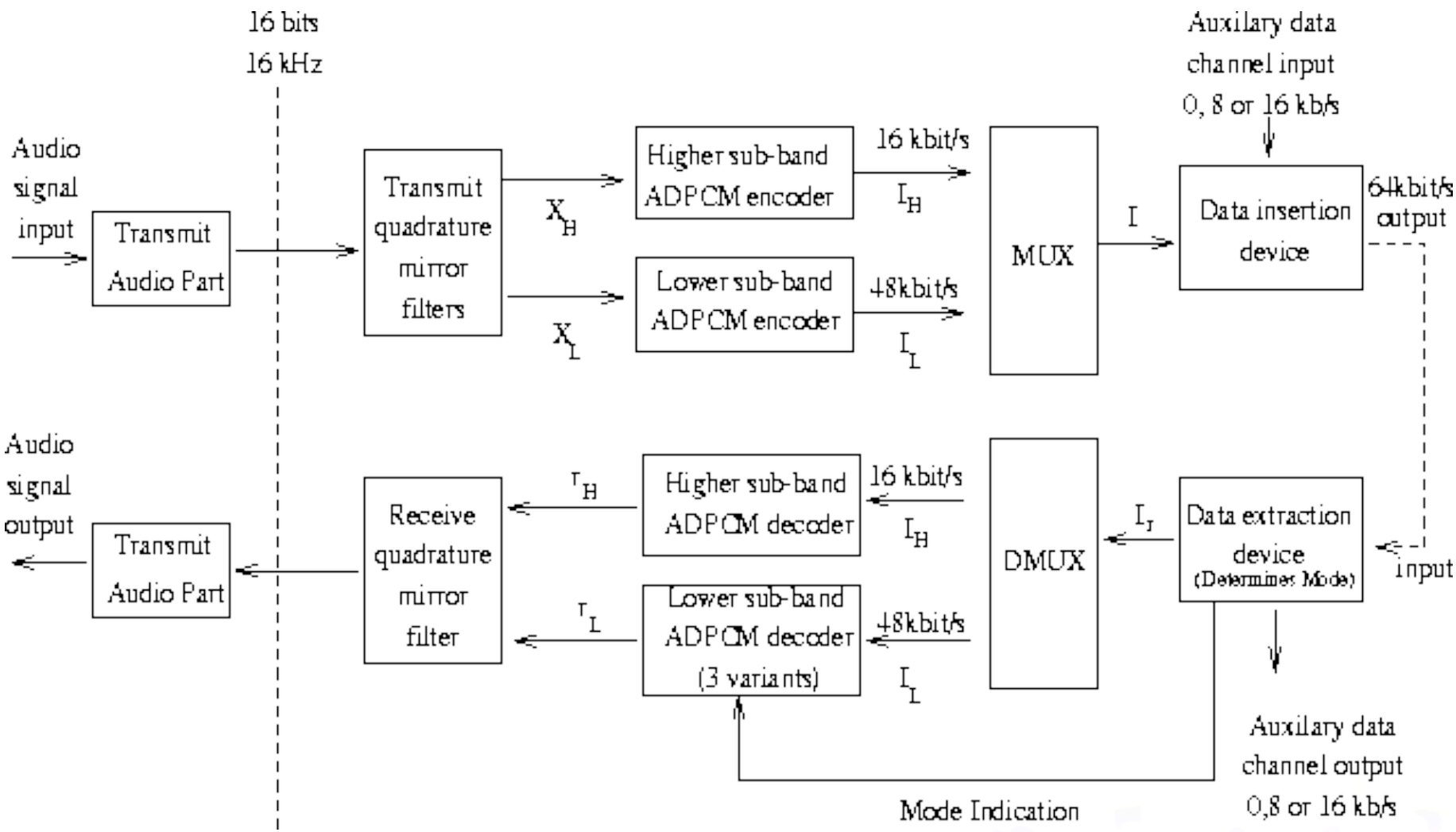
Coefficients of low-pass filters

h_0, h_{23}	3.66211×10^{-4}
h_1, h_{22}	-1.34277×10^{-3}
h_2, h_{21}	-1.34277×10^{-3}
h_3, h_{20}	6.46973×10^{-3}
h_4, h_{19}	1.46484×10^{-3}
h_5, h_{18}	-1.90430×10^{-2}
h_6, h_{17}	3.90625×10^{-3}
h_7, h_{16}	4.41895×10^{-2}
h_8, h_{15}	-2.56348×10^{-2}
h_9, h_{14}	-9.82666×10^{-2}
h_{10}, h_{13}	1.16089×10^{-1}
h_{11}, h_{12}	4.73145×10^{-1}

$$h_{HP,n} = (-1)^n h_{NF,n}$$

reconstruction filters are identical

Block scheme of G.722



Application – MPEG audio

Moving Picture Experts Group

- ISO/IEC working group (ISO/IEC JTC1/SC29 WG11)
- to develop standard for digital audio a video encoding
- 1988 (Ottawa)
- cca 350 members

Standard describes

- bitstream format and decoder, not an encoder

MPEG-1 (1993)

- lossy compression of sound & video
- up to 4095×4095, 30 frames/s, 1.5Mb/s, YUV color space
- standard for Video CD
- Layer 3 (.mp3) format for audio compression

Application – MPEG audio

Layer 1-3

- 64kbps (layer 3)
- ≈ 128kbps (layer 2)
- ≈ 192kbps (layer 1)

Sampling frequency

- MPEG-1: 32kHz, 44.1kHz, 48kHz
- MPEG-2: 16kHz, 22.05kHz, 24kHz, 32kHz, 44.1kHz, 48kHz

MPEG audio

Subband coding

- 1-2: a bank of QMF filters
  - » 32 bands, each with a bandwith of $f_s/64$
- 3: QMF + modified DCT (18 coefficients)
  - » $= 32 \cdot 18 = 576$ subbands

Quantization

- uniform (1-2)
- nonuniform (3) with a variable number of bits

MPEG audio – quantization

Bit allocation to subbands

- psychoacustic model of human perception
- discard info that cannot be perceived

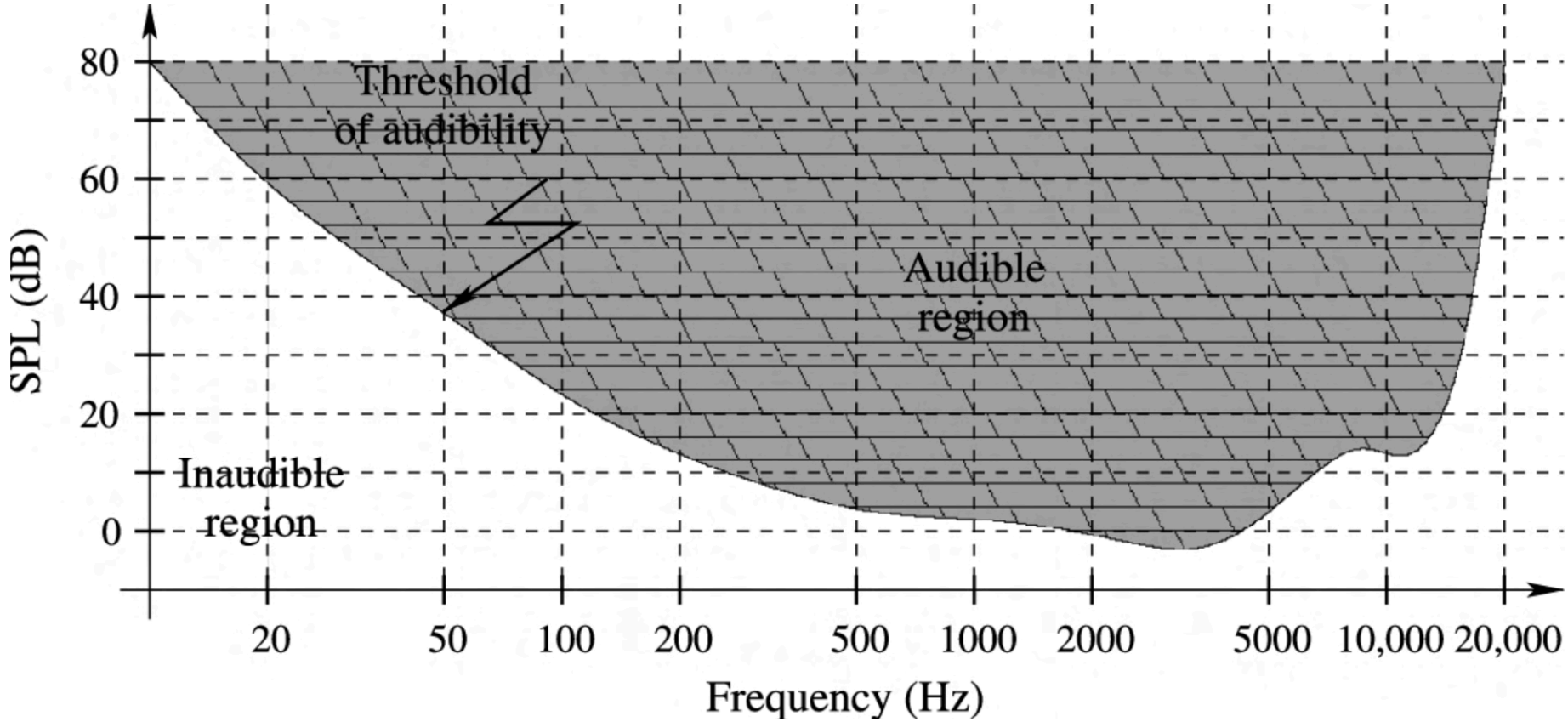
☀ Idea: hearing is frequency dependent

- sounds perceived as equally loud
- at different frequencies
- may have different sound pressure levels (SPL)

Threshold-of-hearing curve

- SPL curve
- boundary of audible / inaudible sounds
- at different frequencies

Audibility threshold



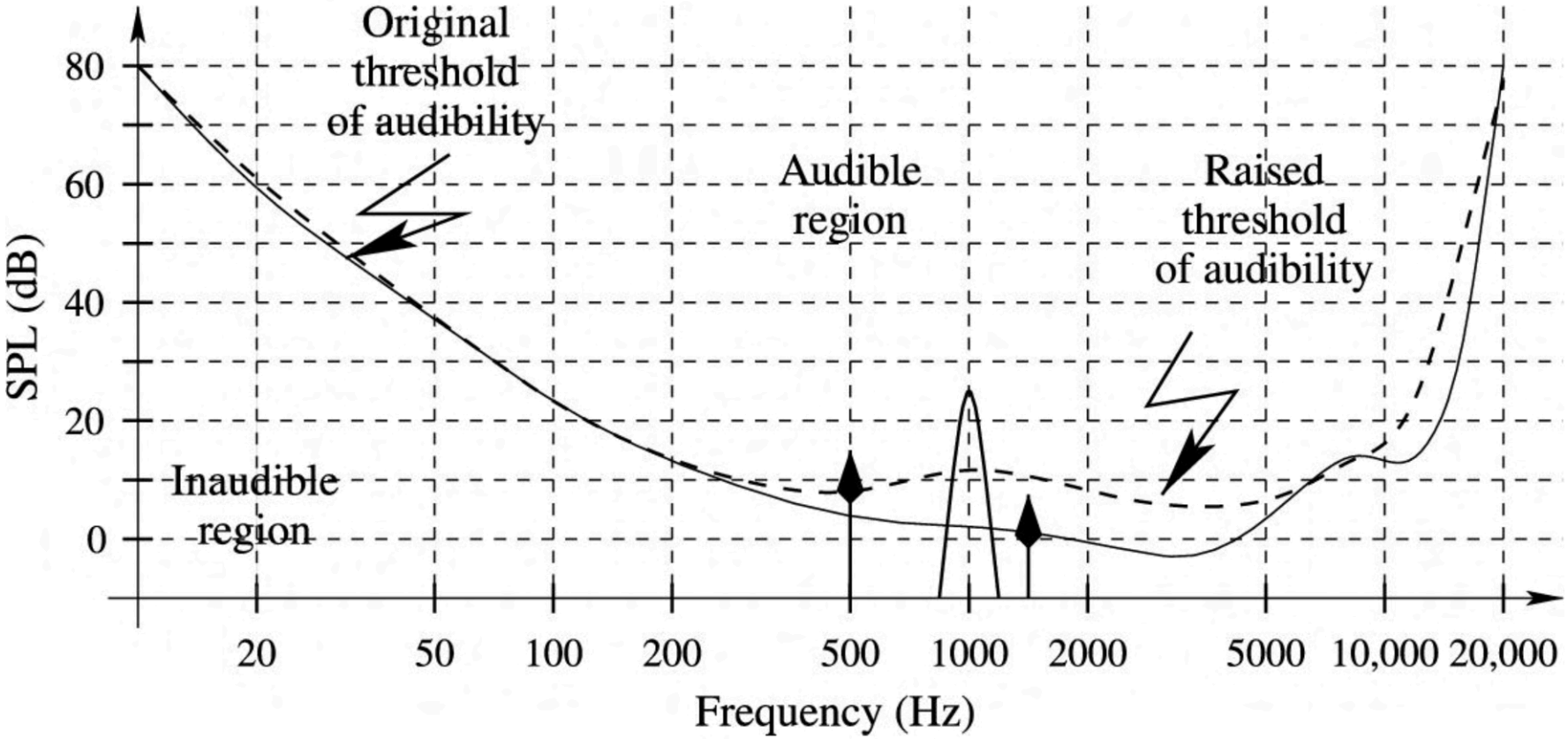
Sounds below the threshold - not perceived by humans

Psychoacustic model

Spectral masking

- signals below a particular amplitude at a particular frequency are not audible
- threshold of audibility rises when multiple sounds impinge on human ear

Spectral masking

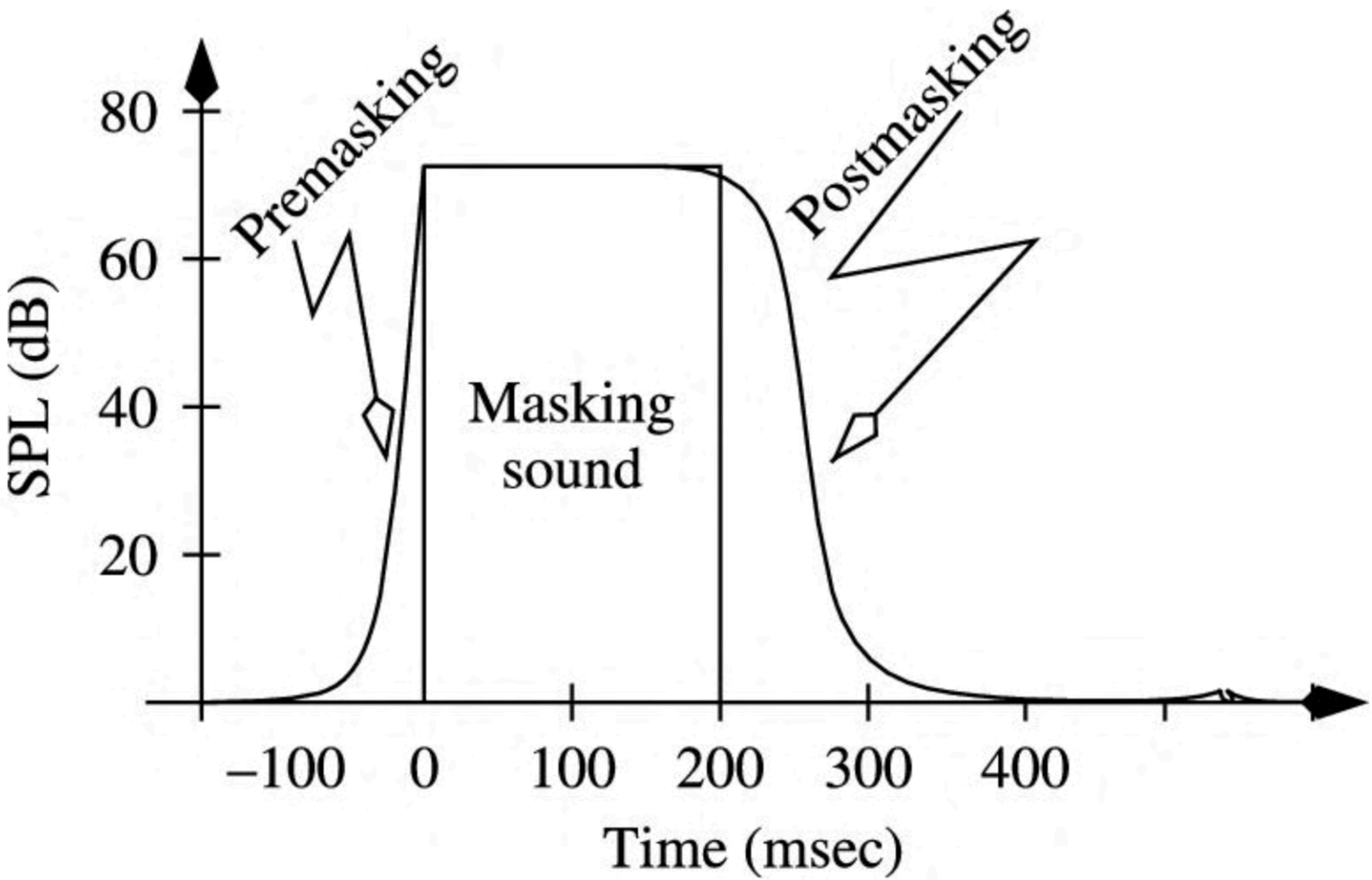


Psychoacustic model

Temporal masking

- sound raises the audibility threshold for a brief interval preceding and following the sound
- sounds occurring in an interval around the masking sound can be masked as well

Temporal masking



MPEG audio – coding

Masking: for each frame

Encoding

- 1-2: fixed codeword length
- 3: Huffman coding
  - » a buffer to ensure fixed transmission rate

Image compression

Dependencies in two dimensions

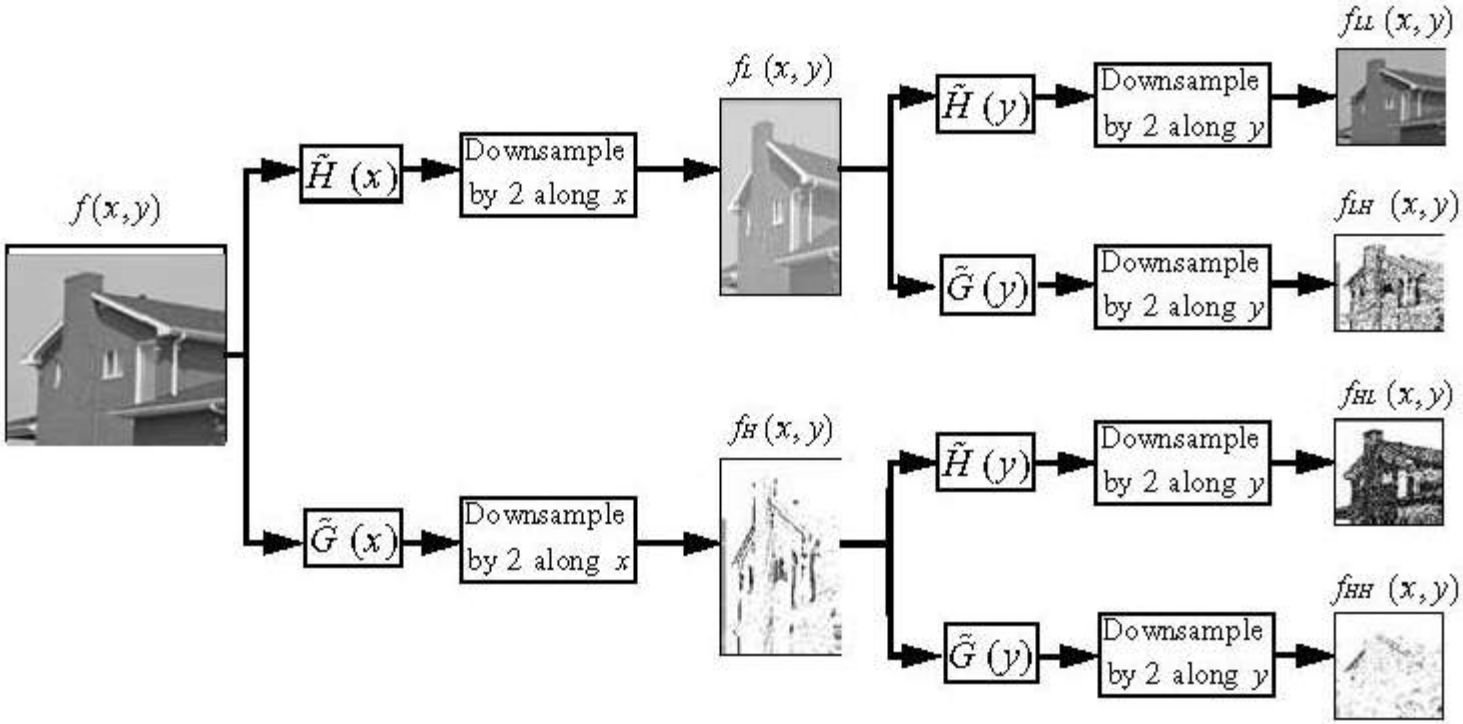
- 2dim filter → separation into two 1dim filters

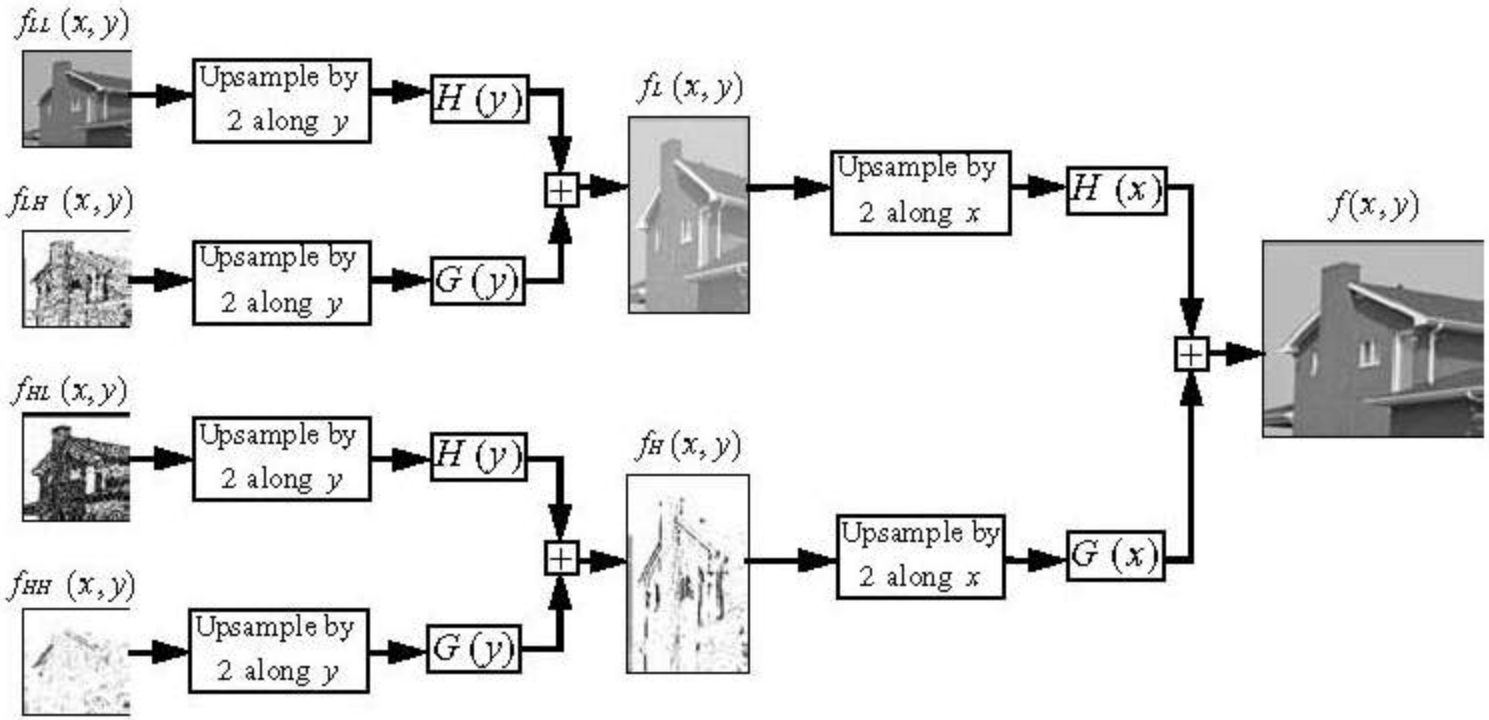
N×N image

- filter each row separately using a HP & LP filters
- downsampling by a factor of 2 → 2 images N×N/2
- filter each column of the two subimages
- downsampling by a factor of 2 → 4 images N/2×N /2

STOP

- or continue the decomp with one or more subimages
- results in 7, 10, 13, or 16 subimages
- usually only 1 or 2 subimages further decomposed
  - » many pixels in high-frequency subimages close to 0


$f(x,y)$
$\tilde{H}(x)$
$\tilde{G}(x)$
Downsample by 2 along x
Downsample by 2 along x
$f_L(x,y)$
$f_H(x,y)$
$\tilde{H}(y)$
$\tilde{G}(y)$
$\tilde{H}(y)$
$\tilde{G}(y)$
Downsample by 2 along y
Downsample by 2 along y
Downsample by 2 along y
Downsample by 2 along y
$f_{LL}(x,y)$
$f_{LH}(x,y)$
$f_{HL}(x,y)$
$f_{HH}(x,y)$


$f_{LL}(x,y)$
Upsample by 2 along y
$H(y)$
$f_{LH}(x,y)$
Upsample by 2 along y
$G(y)$
$f_{L}(x,y)$
Upsample by 2 along x
$H(x)$
$f(x,y)$
$f_{HL}(x,y)$
Upsample by 2 along y
$H(y)$
$f_{HH}(x,y)$
Upsample by 2 along y
$G(y)$
$f_{H}(x,y)$
Upsample by 2 along x
$G(x)$

Application – JPEG 2000

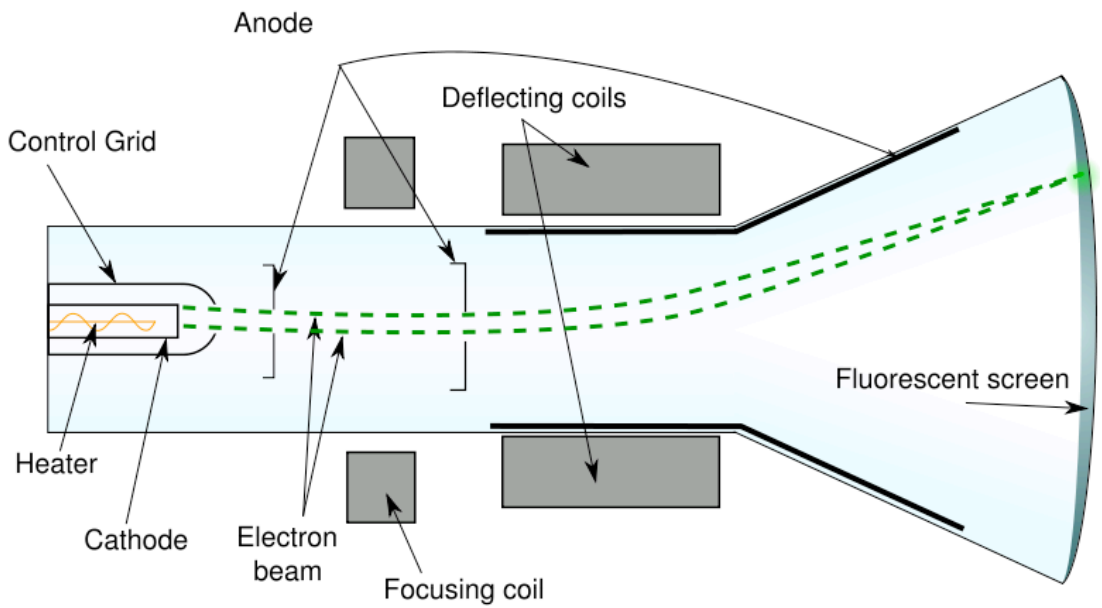
Wavelet transform

Coefficients correspond to details for various resolutions

- fairly precise quantization possible
  - » FBI fingerprint image compression standard
  - » medical images compression
- data optimization for progressive transmission over a network
- a possibility to define regions of interest for higher-quality encoding

Data Compression Algorithms

Video Compression



Introduction

Video ≈ a time sequence of images

Video × image compression

- change in the average intensity of the pixels
  - » negligible after a decompression of a still image
  - » quite annoying in a motion video sequence
- poor reproduction of sharp edges
  - » serious problem in the compression of still images
  - » negligible in a video sequence

Introduction

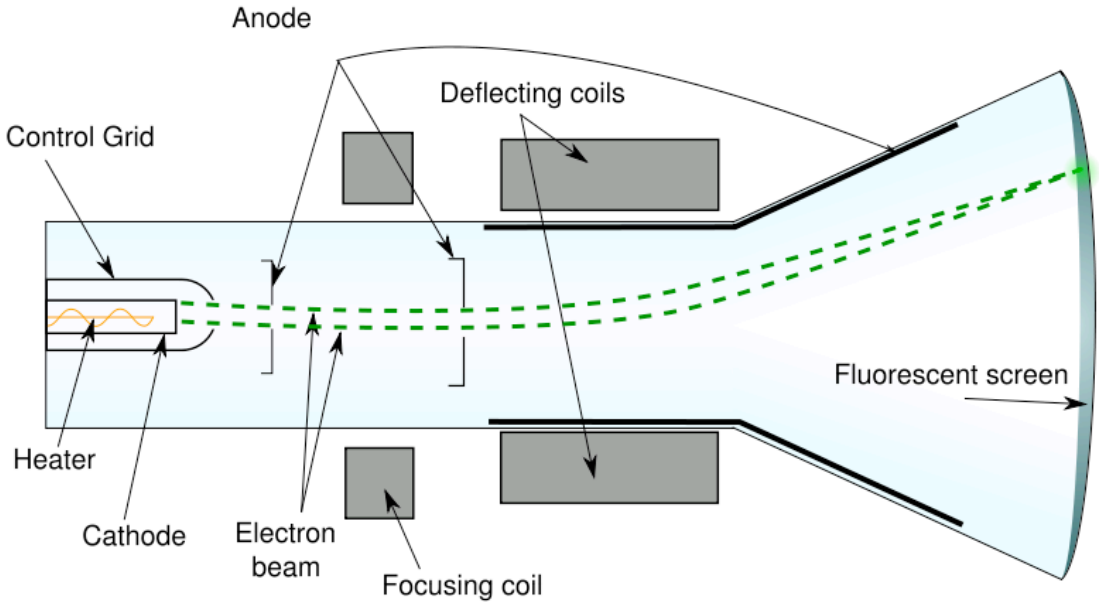
Classification of algorithms by application field

- both-ways communication $\Rightarrow$ symmetric compression / decompression
- one-way communication $\Rightarrow$ higher tolerance for the complexity of compression

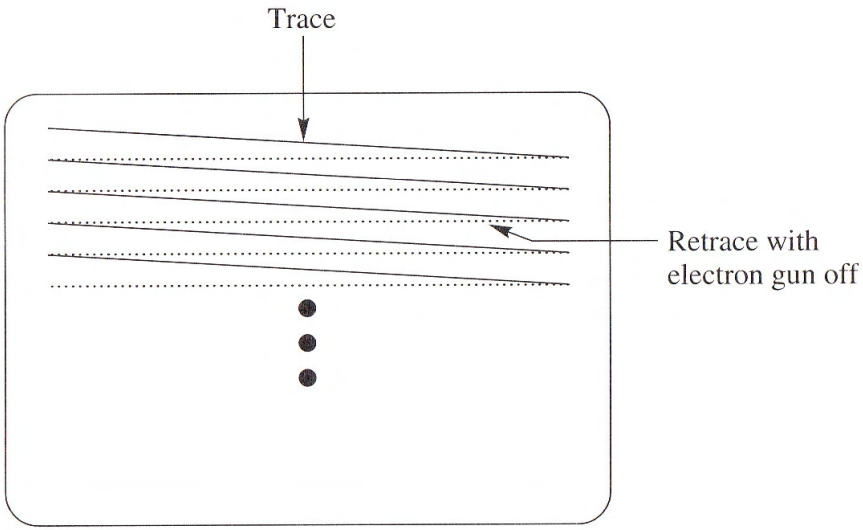
Video signal representation

CRT – Cathode Ray Tube

- electron gun emits a stream of electrons
- front surface coated with a phosphor compound
- converts the kinetic energy of the electrons to light



Video signal representation



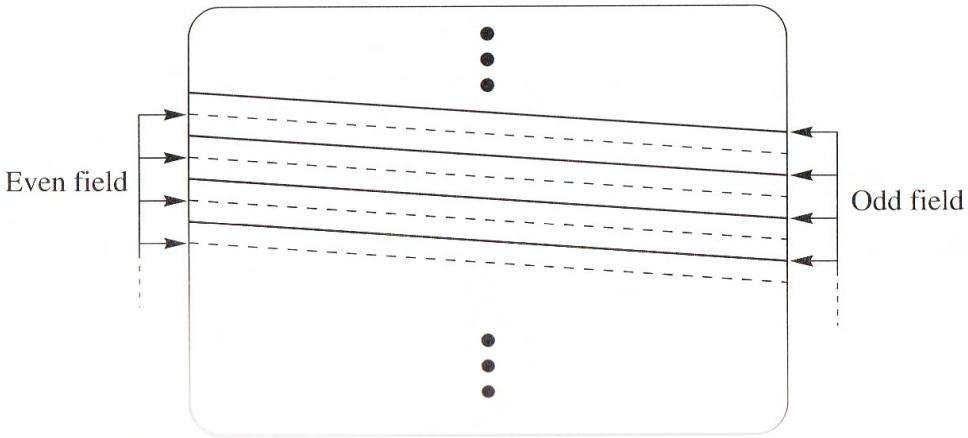
Refresh rate fps (frames per second)

- min 15fps
- 16fps → 24fps (35mm film)
- doubling 48fps (72fps)
- PAL,SECAM 25fps
- USA 30fps (1930), 29.97fps (1953)

Analog TV (USA)

NTSC (National Television Standards Committee)

- 525 rows, but only 483 visible (vertical blanking)
- aspect ratio (width:height) 4:3 $\Rightarrow$
  line size $4/3 * 483 = 644$
- 59.94fps, interlacing (2× 262.5 rows)



Analog TV (Europe)

PAL (Phase Alternating Line, Germany)

- 625 rows (about 576 visible)
- aspect ratio 4:3
- 50fps, interlacing 2:1

SECAM (Séquential Couleur avec Mémoire, France)

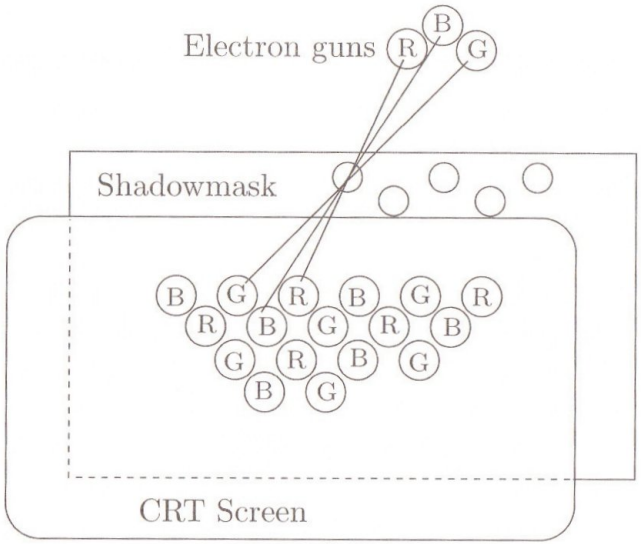
Various aspect ratios (film / TV)

- NTSC,PAL,SECAM 1.33
- 16mm, 35mm film 1.33
- HDTV 1.78
- Widescreen film 1.85
- 70mm film 2.10
- Cinemascope 2.35

Representation of colors

Color TV

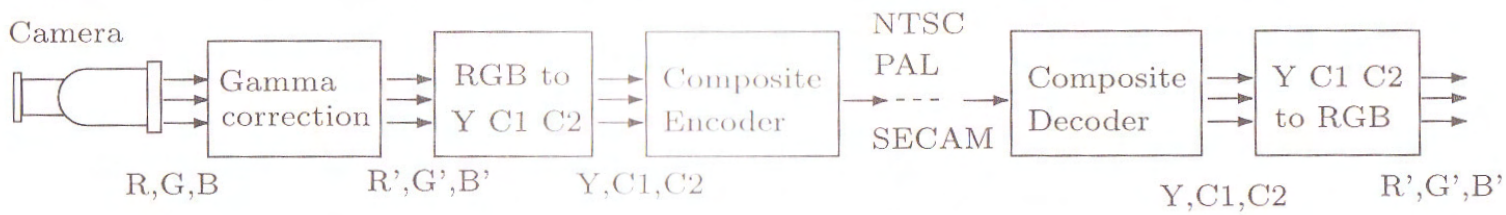
- 3 separate electron beams → R, G, B



Representation of colors

Composite signal

- 1953: backward compatibility with b/w TV
- Y (luminance), C_B,C_R (chrominance)
  - » Y used by b/w TV



Composite / component video

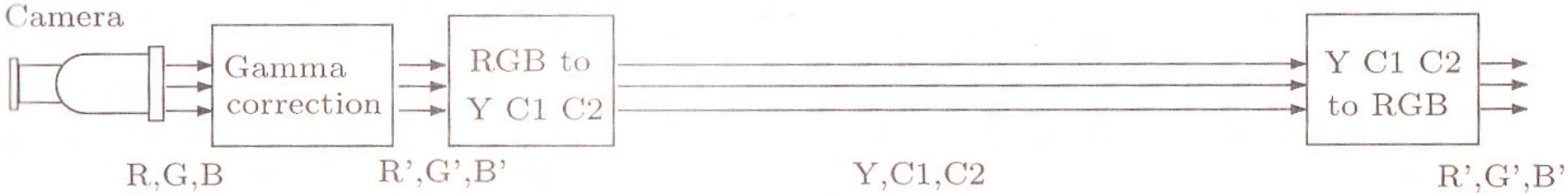
NTSC: YIQ

- $Y = 0.299\ R' + 0.587\ G' + 0.114\ B'$
- $I = 0.596\ R' - 0.274\ G' - 0.322\ B'$
- $Q = 0.211\ R' - 0.523\ G' + 0.311\ B'$
- R',G',B' := R,G,B after gamma correction

PAL: YUV SECAM: YD_rD_b

Single channel for all 3 components

Component video: 3 channels for separate transmission



Digital video

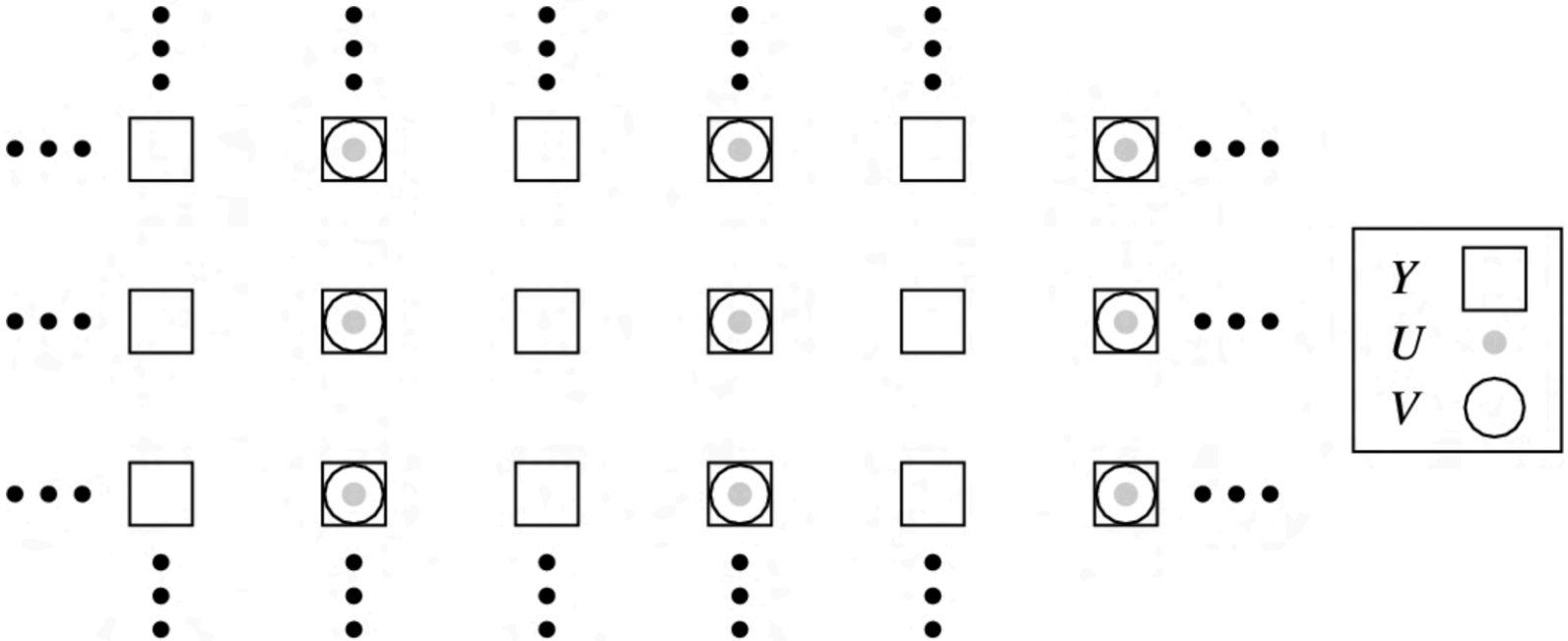
Early attempts: composite video sampling

- NTSC x PAL problem
- 70s: components sampling $\Rightarrow$ international standard

ITU – R recommendation 601 (CCIR 601)

- digitization of interlaced analog videosignal
- color space YC_bC_r
- sampling frequency 3.375MHz
  - » for each component an integral multiple (up to 4x)
  - » 4:4:4 $\Rightarrow$ 13.5MHz each
  - » 4:2:2 $\Rightarrow$ 13.5MHz luminance, 6.75MHz chrominance
    - Y: 720 samples/row , C: 360 samples/row

Rec. 601 sampling format 4:2:2



ITU – R 601

Normalize YC_bC_r samples

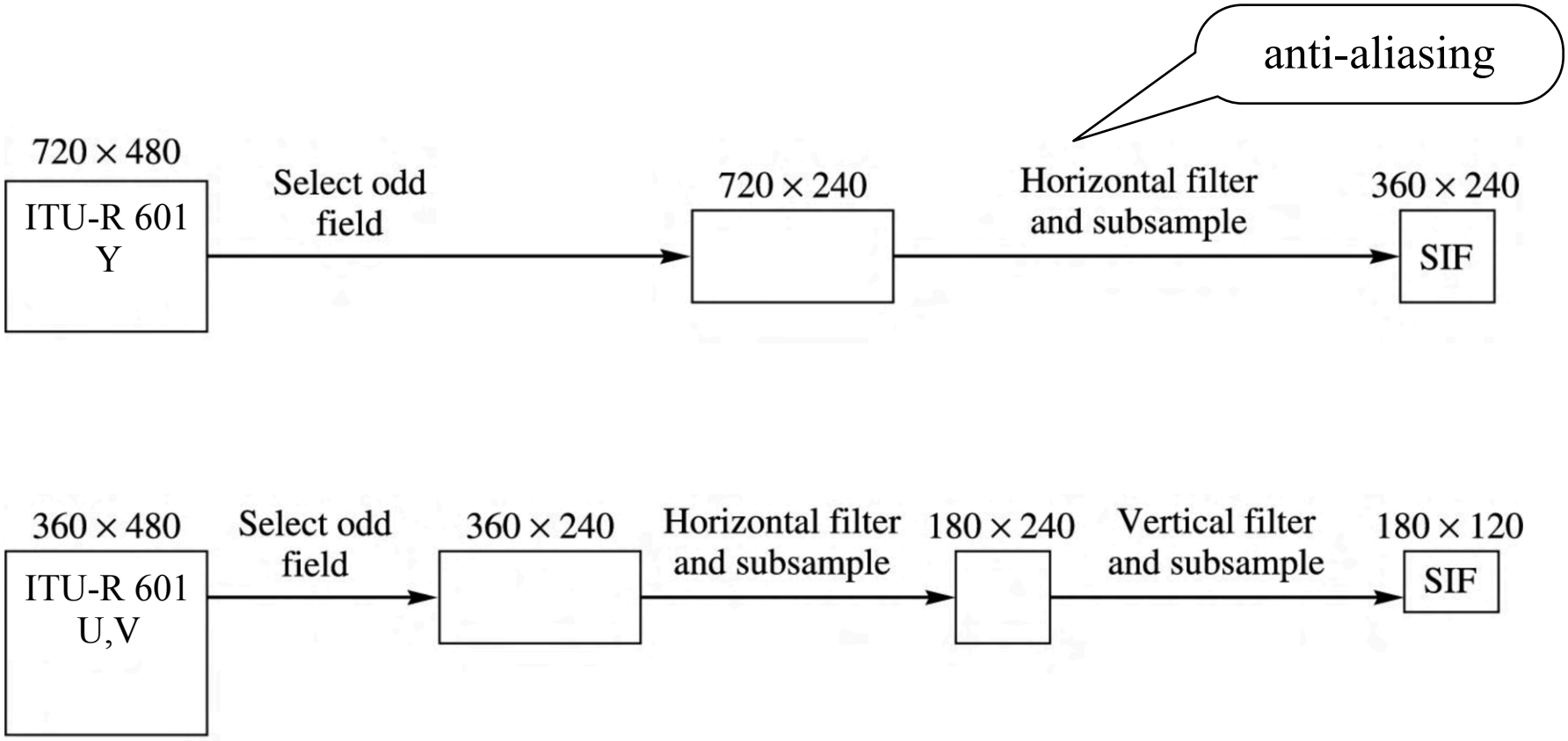
- $Y_s \in \langle 0,1 \rangle$, $C_{b_s}, C_{r_s} \in \langle -1/2, 1/2 \rangle$

Convert to 8b strings

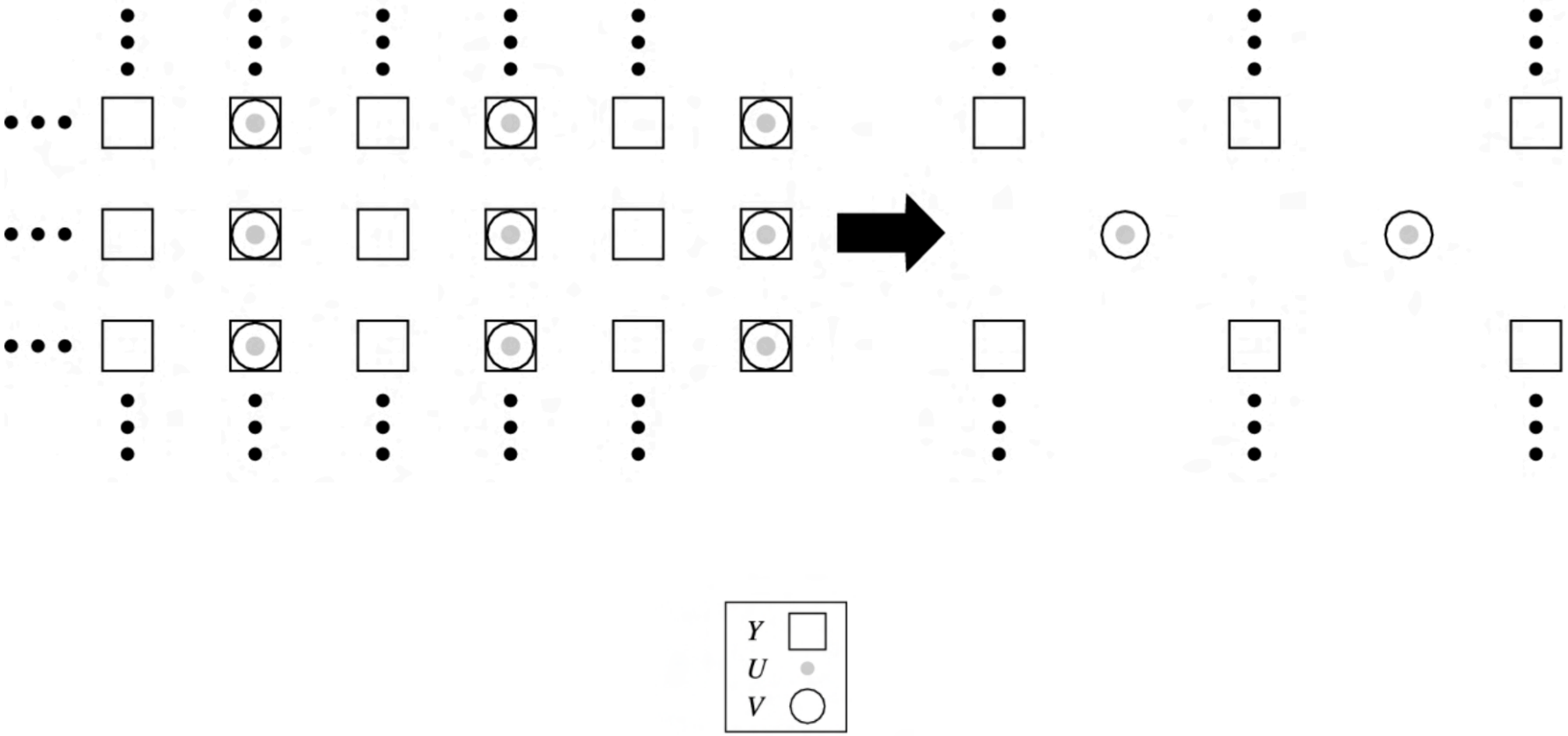
- $Y = 219Y_s + 16$
- $U = 224C_{b_s} + 128$
- $V = 224C_{r_s} + 128$

MPEG – 1

MPEG-SIF format: based on ITU-R 601 4:2:2



ITU-R 601 → MPEG-SIF



H.261

ITU-R H.261

- videotelephony, videoconferencing

Format CIF (Common Interchange Format)

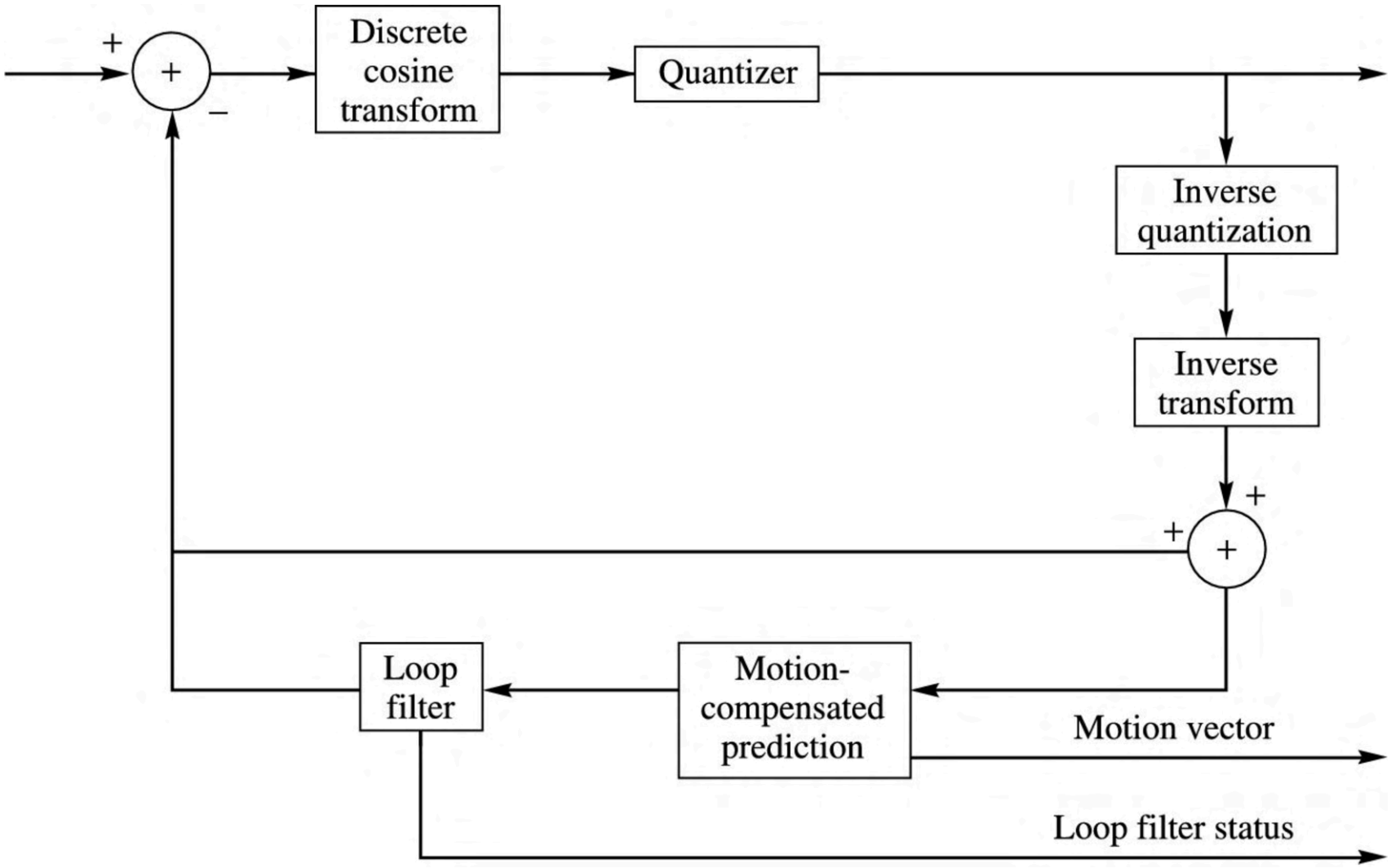
- Y: 288×352
- UV: 144×176
- QCIF (Quarter) half in both directions

H.261

☀ Idea

- frame → 8 × 8 blocks
- prediction + differencing
  - » prediction by the previous frame
  - » or O → if non-existent or too different
- DCT
- quantization
- entropy coder
  - » with variable codeword length

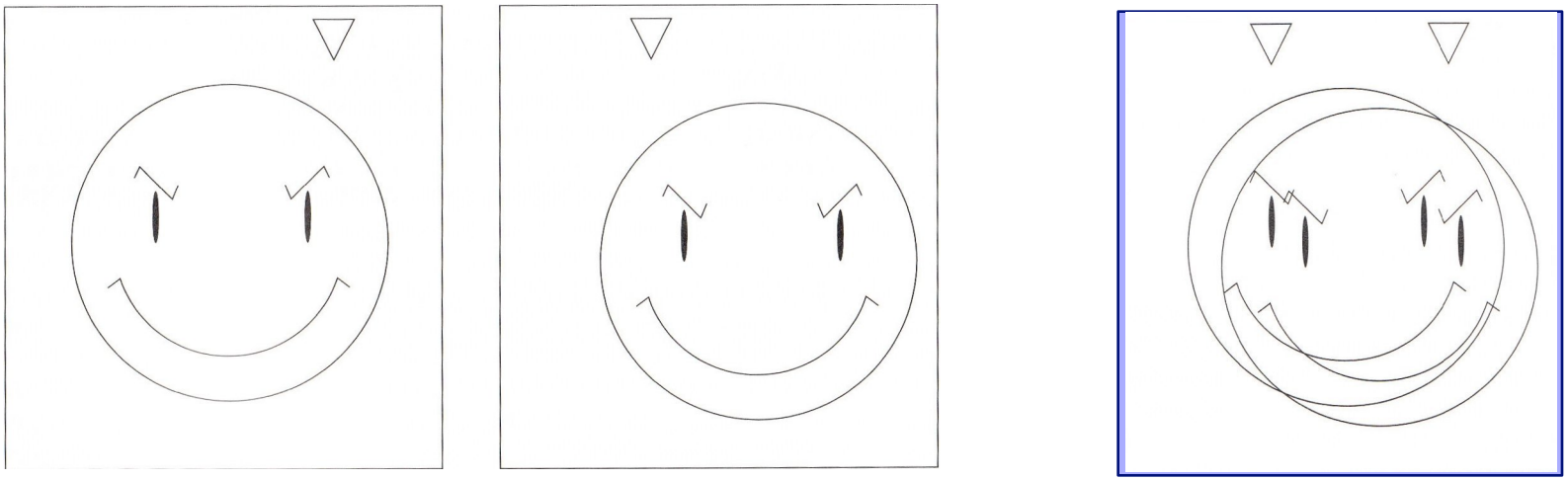
H.261 scheme


+
+
−
Discrete cosine transform
Quantizer
Inverse quantization
Inverse transform
+
+
+
Loop filter
Motion-compensated prediction
Motion vector
Loop filter status

Motion compensation

Differential encoding

- prediction by the same position on the previous frame insufficient
- motion vector

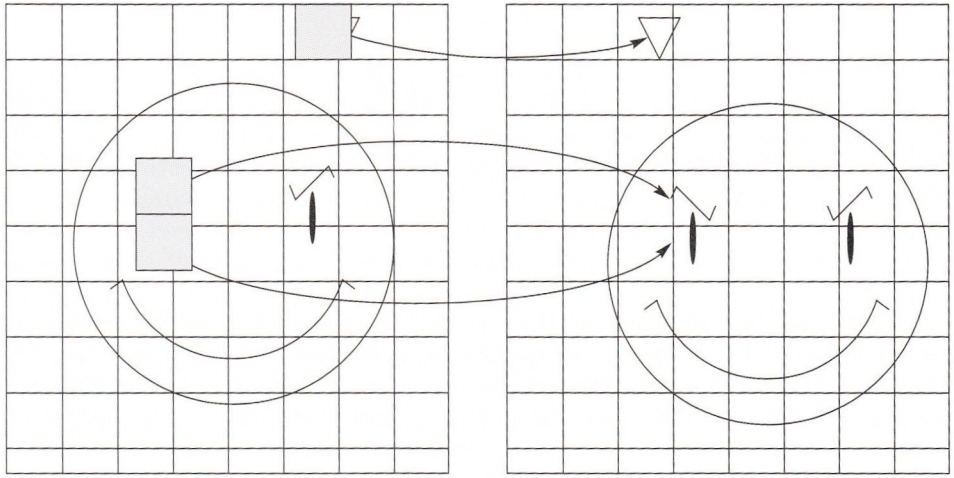


Block-based motion compensation

frame → square blocks

```
for each block B of the frame being encoded :
    find the block P of the previous frame
    with min d(B,P), d ∈{d1,σ2}
    if d(B,P) > ε : encode B
                        // no prediction used
    else : motion-vector = upper-left-corner(B) -
                           upper-left-corner(P)
           encode motion-vector and B - P
```

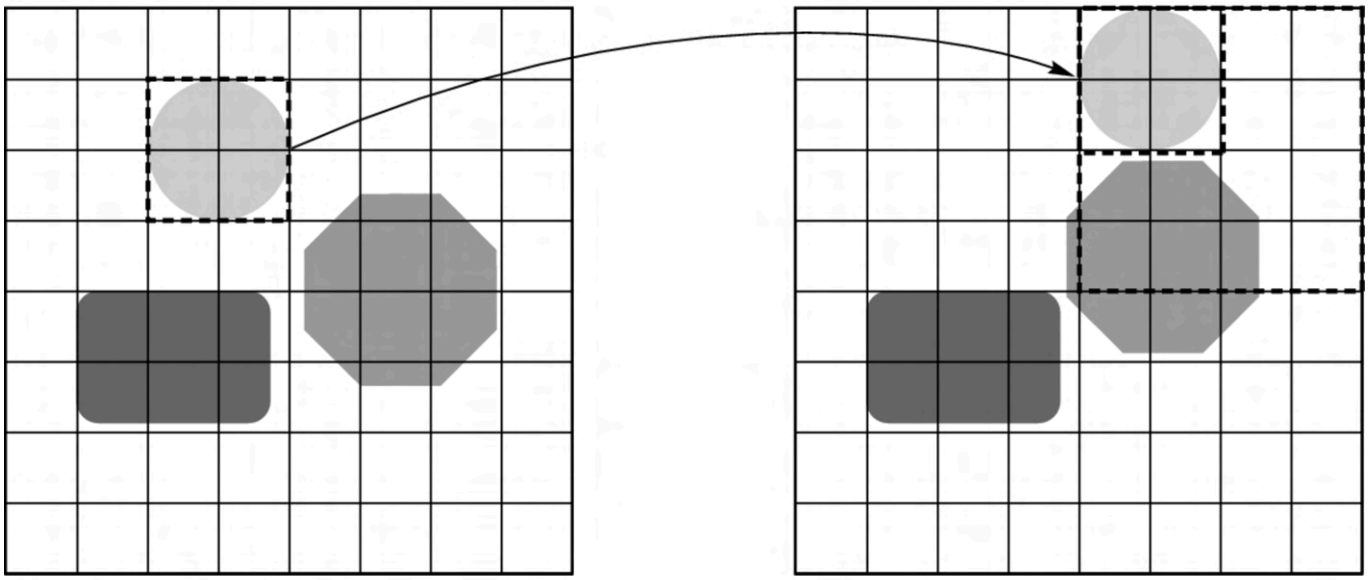
Block-based motion compensation



Motion compensation: reducing computation

Increase block size

- ✔ fewer blocks
- ✖ more operations per comparison
- ✖ objects moving in different directions



Motion compensation: reducing computation

Reduce the search space

- ✔ smaller # comparisons
- fewer candidates
- ✖ higher probability of missing a match

H.261: motion compensation

- Solution H.261
  - macroblocks
    - » Y: 16 × 16 (4 blocks 8 × 8)
    - » UV: 8 × 8
  - motion-compensated prediction on macroblock level
    - » search for the best match on the previous frame
    - » for Y component only
    - » search space: < ±15 in both directions
    - » motion vector for UV: (motion vector for Y) × ½
      - (-3,10) → (-1,5)

The loop filter

Sharp edges in the block used for prediction

- higher variation in differences
- high values of HF coefficients after DCT
- worse compression ratio

Smoothing filter

- 2dimensional separable: rows, columns
- coefficients (1/4, ½, 1/4)
- block boundaries remain unchanged

Example

110	218	116	112
108	210	110	114
110	218	210	112
112	108	110	116

110	165	140	112
108	159	135	114
110	188	187	112
112	109	111	116

110	165	140	112
108	167	148	113
110	161	154	113
112	109	111	116

Transform

DCT of 8×8 blocks

- block of differences (*inter* mode)
- original block (*intra* mode)

Coder simulates the decoder

- compressed frames are reconstructed
- stored in a frame store

Quantization

High variation of coefficients to be quantized

- large DC coefficient of intra block
- little frame-to-frame motion $\Rightarrow$
- $\Rightarrow$ small coefficients of inter block

32 different quantizers

- intra DC coefficient: uniform midrise quantizer with step size 8
- others: uniform midtread with step size $\in \langle 2,62 \rangle$
- smaller step size $\Rightarrow$ more nonzero coefficients likely

Quantization

Identification of the selected quantizer

- macroblock header
- GOB (group of blocks): 3 rows of 11 macroblocks
  - » 5bit identifier in GOB header

Coding

Zigzag scan through the block of quantized coeff.

- ≈ JPEG
- (# of preceding 0's, current coefficient)
- 20 most common → single variable length codeword
- other → fixed length 20b
  - » escape (6b), length of run of 0's (6b), coefficient (8b)

CBP (coded block pattern)

- in macroblock header - to avoid zero blocks
- which blocks (4 luminance, 2 chrominance) contain nonzero coefficients
- 64 pattern numbers → variable length code
  - » $CBP = 32P_1 + 16P_2 + 8P_3 + 4P_4 + 2P_5 + P_6$

Rate control

Application: videophone, videoconferencing

- only minimal coding delay (< 150ms)

Transmission buffer

- keeps the output rate of the encoder fixed
- buffer may become empty / overflow
- if coder sends data with lower / higher rate than necessary for the transmission
- buffer may request a change

Request a higher/lower transmission rate

- coder selects quantizer with larger / smaller step size
  - » larger step → more zero coefficients
- last resort: drop certain frames

MPEG

Moving Picture Experts Group

- ISO/IEC (ISO/IEC JTC1/SC29 WG11) working group
- to develop standards for digital audio and video encoding
- 1988 (Ottawa)
- cca 350 members

Standard describes

- bitstream format and decoder
- not encoder

MPEG

MPEG-1 (1993)

- lossy compression of sound & video
- up to 4095×4095, 30 frames/s, 1.5Mb/s
- YUV color space
- standard for Video CD
- Layer 3 (.mp3) format for audio compression

MPEG-2 (1994)

- up to 15Mb/s
- TV broadcast (DVB, cable TV, HDTV), DVD

MPEG-3

- originally intended for HDTV
- development stopped, merged with MPEG-2

MPEG-1

Same basis as H.261

- 8×8 blocks
- inter/intra
- motion-compensated prediction on macroblock level
- DCT
- quantization and coding
- rate control using a transmission buffer

Differences

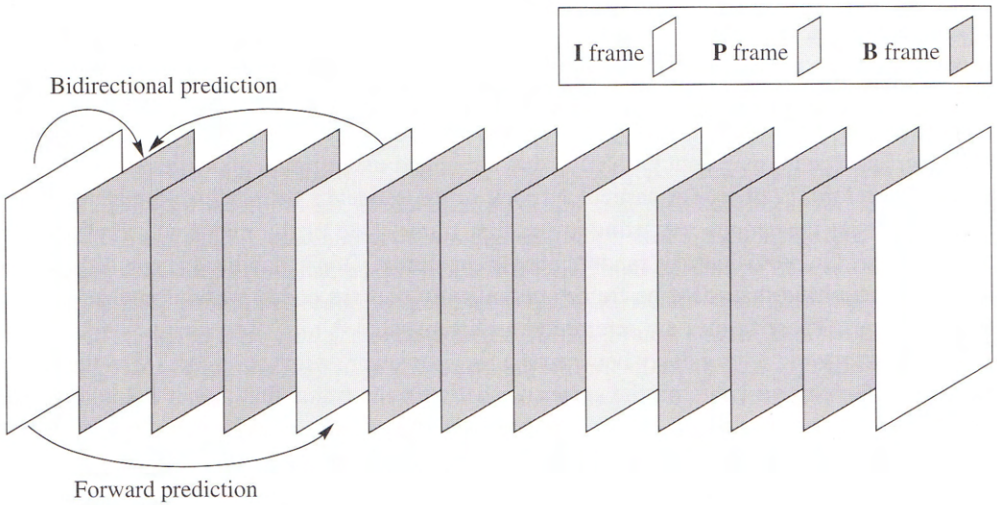
- different applications
- MPEG-1: digital storage & retrieval

Frame types

Frames

- I-frame (intraframe) current frame
  - » coded with no reference to other frames
- P-frame (predictive frame) difference between current and most recent I or P-frame
- B-frame (bidirectionally predictive) difference between two closest I/P-frames (most recent & closest future fram
- e.g. I:P:B ≈ 2:5:12

anchor frames



Groups of pictures

Group of Pictures (GOP)

- smallest random access unit in the video sequence
- a repeating sequence of (typically) 10-30 frames
- DVD (MPEG-2) restrictions
  - » PAL DVD ≤15 frames / GOP
  - » NTSC DVD ≤18 frames / GOP

GOP

At least one I-frame

Start

- I-frame
- B-frames which use only the following I-frame for prediction

Properties

- replay at original speed even on a slow hardware
- B, P, or I-frames may be skipped
  to keep the original speed
- forward, jump forward/ back

GOP

☀ Example

I B B P B B P B B P B B I

Display order

- in which the sequence is displayed to the user
- I_1 B_2 B_3 P_4 B_5 B_6 P_7 B_8 B_9 P_{10} B_{11} B_{12} I_{13}

Bitstream order

- the processing order
- I_1 P_4 B_2 B_3 P_7 B_5 ...

Coding

Motion compensation

- standard does not specify a particular method
- search space size should grow with the distance between frames

DCT & quantization ≈ JPEG

- different quantization tables for different frames

Rate control

- sequence level: B-frames first
- individual frames
  - » increase quantizer step size
  - » omit HF coefficients

Coding

CPB (constrained parameter bitsteam)

- recommended values of parameters
- horizontal size $\leq$ 768, vertical $\leq$ 576
- $\leq$ 396 macroblocks / frame (25 fps)
- $\leq$ 330 macroblocks / frame (30 fps)
- 352x288 (25fps), 352x240 (30fps) -> 1-1.5Mb/s

Properties

- VHS quality (low-moderate motion)
- worse than VHS (high-motion)
- interlacing not considered

analog video tape cassettes

MPEG-2

Idea: app. - independent standard (toolkit approach)

Profiles

- algorithms to be used
- simple, main, snr-scalable, spatially scalable, high
- simple: no B-frames
- main: see MPEG-1

Possibility to use several *layers*

- snr-scalable, spatially scalable, high
- basic layer: encodes low-quality videosignal
- enhancement layers: differential encoding to improve quality

Example

DCT coefficients

29.75 6.1 -6.03 1.93 -2.0 1.23 -0.95 2.11

Quantizer with step size 4

- same step for all coefficients (for simplicity)
- $l_{ij} = \lfloor \theta_{ij}/4 + 0.5 \rfloor$
- $\hat{\theta}_{ij} = l_{ij} \cdot 4$

Reconstructed coefficients

28 8 -8 0 -4 0 0 4

Difference *original - reconstructed* (quantizer error)

1.75 -1.9 1.97 1.93 1.99 1.23 -0.95 -1.89

Example

Enhancement layer

- quantize differences
- quantizer with step size 2

Enhancement layer values after reconstruction

2 -2 2 2 2 2 0 -2

After summation with basic layer values

30 6 -6 2 -2 2 0 2

Quantizer error

-0.25 0.1 -0.03 -0.07 -0.01 -0.77 -0.95 0.11

MPEG-2

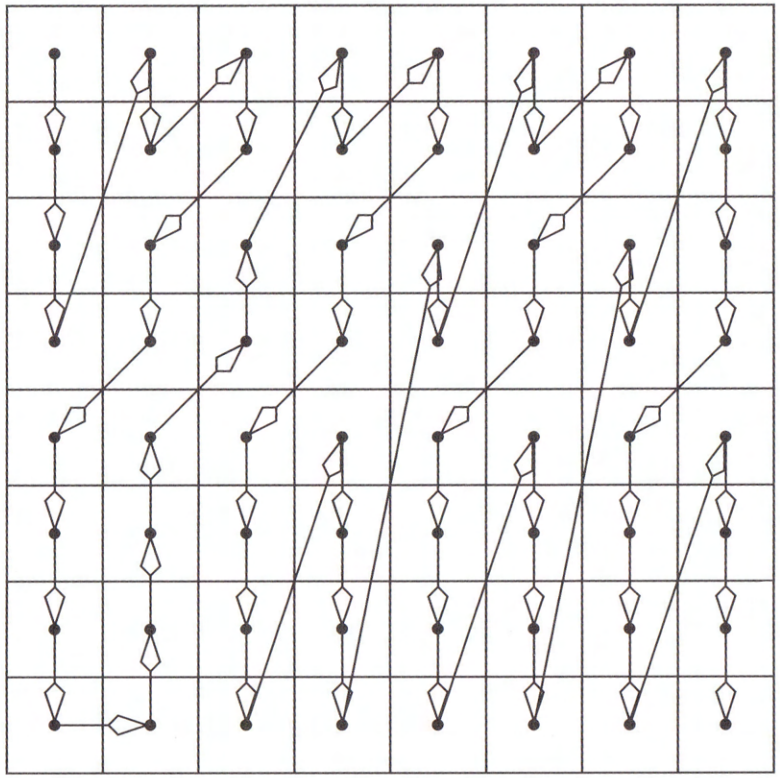
Levels

- low (352 x 240)
- main (720 x 480)
- high 1440 (1440 x 1152)
- high (1920 x 1080)
- all levels 30 fps

Interlacing : field-based alternatives to I,P,B

- P-frames: may be replaced by 2 P-fields
- B-frames: may be replaced by 2 B-fields
- I-frames: 2 I-fields or I and P-fields
  - » predict the bottom field by the top field

MPEG-2



MPEG-2 : new prediction modes

New motion-compensated prediction modes

P-frames : field predictions

- based on 1 or 2 most recently decoded fields
- 1st field in a frame
  - » based on 2 fields from the previous frame
- 2nd field in in a frame
  - » based on the 2nd field of the previous frame
  - » and 1st field of the current
- which field used - stored

MPEG

MPEG-4 (2001)

- objective: encode video at low bitrates
- various pieces of information in several data streams
  - » object oriented approach
    - audio, video a VRML objects
  - » 3D scene description using virtual cameras
  - » description and animation of facial expression (talking-head)
  - » phonemes description for an automated text-to-speech translation
  - » DRM (Digital Rights Management)

MPEG

MPEG-4 (2001)

- allows to use arbitrary codecs
  - » MPEG-1-2 encoding specified in the standard
  - » more efficient compression/ lower compatibility
    - codecs must be installed
  - » MPEG-4 Part 10 (H.264): HDTV, Blu-ray

MPEG-M (2010-13)

- Part 2
  - » High Efficiency Video Coding (HEVC)
  - » = ITU-T H.265
  - » used in DVB-T2

H.264 improvements – examples

Macroblock

- may be further divided
- sub-macroblocks 8x4, 4x8, 4x4
- motion-compensated prediction
- tracking finer details

Transform

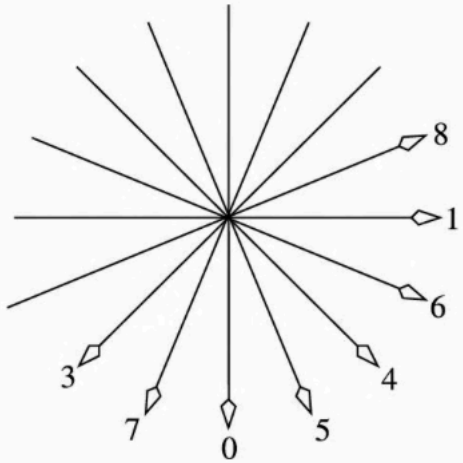
- 4 x 4 DCT-like transform

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & 2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

H.264 improvements – examples

Intra prediction

- 9 prediction modes for 4x4 blocks



Q	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

- mode 2 (DC) : average of the top & left boundary

H.264 improvements – examples

Encoding

- option 1
- exponential Golomb code
- = binary code for x+1
- preceded by unary code of its length
- + variable length code (CAVLC)
- option 2
- adaptive binary arithmetic coding

H.265 improvements – examples

Block structure

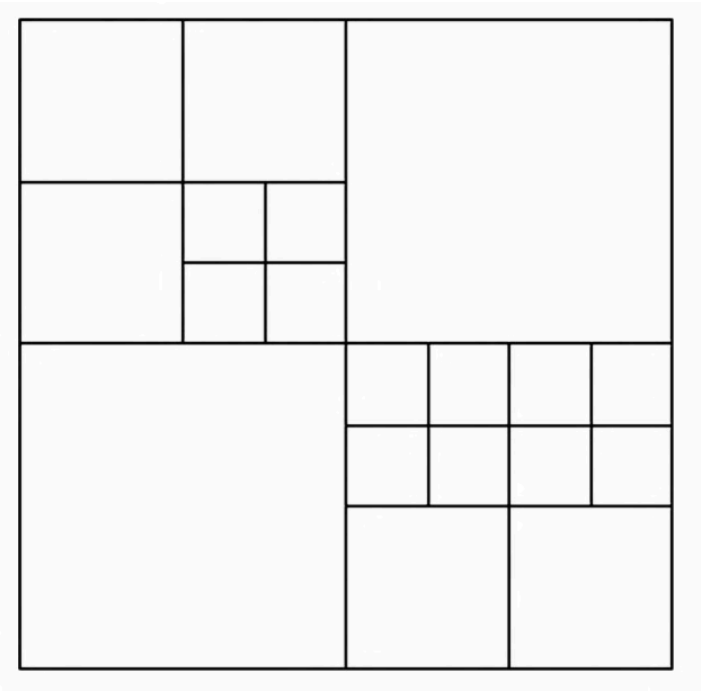
- frame partitioned Coded Tree Units
- one Luma Coded Tree Block (CTB)
- size $2N$x$2N$, $N \in \{8,16,32\}$, fixed for the entire video
- two Chroma CTB, size NxN
- CTB may partitioned into Coded Blocks (CB)
- using quadtree structure

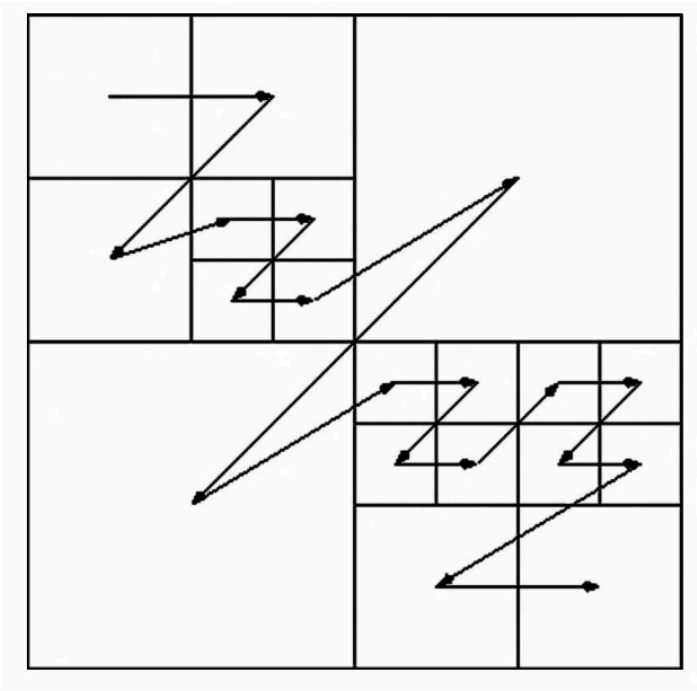
H.265 improvements – examples

Quadtree partitioning

- divide into 4 square (sub)blocks
- (sub)block has unsatisfactory reconstruction
- divide it further into 4 (subsub)blocks
- may be repeated

H.265 – block structure





CB within CTB scanned using Z-scan

H.265 – block structure

Pros

- regions with little activity - larger CB - bitrate saving
- regions with a lot of activity - small CB
- more accuracy
- each CB may be coded independently
- using inter / intra prediction