Applications

Updated 10/14/2024

Part 3: Practice
L11: Compression (1/4)

Javed Aslam, Wolfgang Gatterbauer

cs7840 Foundations and Applications of Information Theory (fa24)

https://northeastern-datalab.github.io/cs7840/fa24/

10/9/2024

Last time

- Finish basics of Information theory

Today

- Start compression as an application

Next time

- Continue

randomness

entropy

compressibility $\longleftrightarrow$ predictability

Compression Taxonomy:

① Fixed source alphabet of bounded size

ⓐ assume we know underlying distribution $\vec{p}$; must code symbol-by-symbol

- Shannon Code $E[L] < H(x)+1$
- Fano Code $E[L] \leq H(x)+1-p_{min}$
- Huffman code optimal but $\geq H(x)$

* Shannon-Fano-Elias $H[x]+1 \leq E[L] < H[x]+2$

* Arithmetic Coding approaches $H(x)$ in limit
  - block coding "for free"
    - no exponential blow up

↑ today

- Sender & receiver must agree on codebook (or it must be sent)
- coding dist. $\vec{q}$ must match actual distribution $\vec{p}$, else suffer $D(\vec{p}\,\|\,\vec{q})$ loss
- in general, only approaches entropy limit with block coding
  - code book grows exponentially in size

(b) Dictionary-based coding

- Lempel-Ziv

- don't need underlying dist. $\vec{p}$
- no codebook to agree on or send
- approaches entropy limit, even for Markov sources

(2) Compression for streams

- audio, video, images, fax, etc.

- Run-length encoding & others

(3) Transforms

- make data easier to encode

- Delta encoding
- Burrows-Wheeler

$$\begin{array}{ccccccc} 2 & 4 & 6 & 8 & 7 & 9 & 6 \\ & 2 & 2 & 2 & -1 & 2 & -3 \end{array}$$

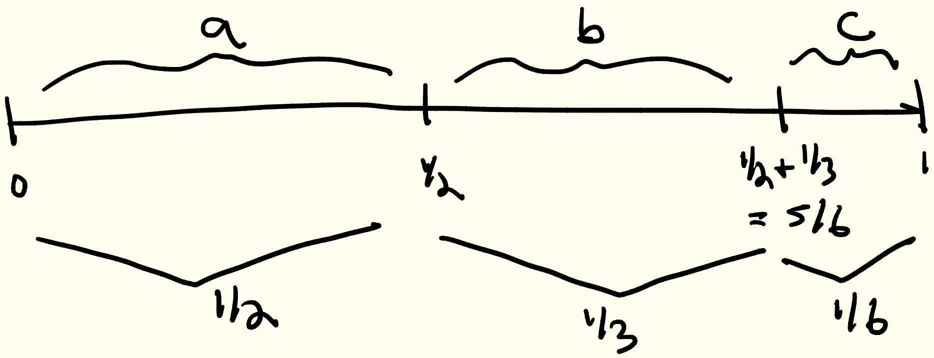
(4) Universal codes for integers of unbounded size

- Unary
- Elias Gamma code
- Elias Omega code
- Fibonacci code

Shannon - Fano - Elias

- Let $\mathcal{X} = \{a, b, c\}$ $\vec{p} = (1/2, 1/3, 1/6)$

- Consider intervals in range $[0, 1]$



① Represent each symbol by a non-overlapping interval in $[0, 1)$ whose width is its probability

$a \to [0, 1/2)$

$b \to [1/2, 5/6)$

$c \to [5/6, 1)$

② Represent each interval by its midpoint in binary

	midpoints	binary
$a \to [0, 1/2)$	$\frac{0+1/2}{2} = 1/4$	$.01$
$b \to [1/2, 5/6)$	$\frac{1/2+5/6}{2} = 2/3$	$.1010101\overline{10}$
$c \to [5/6, 1)$	$\frac{5/6+1}{2} = 11/12$	$.11101\overline{10}$

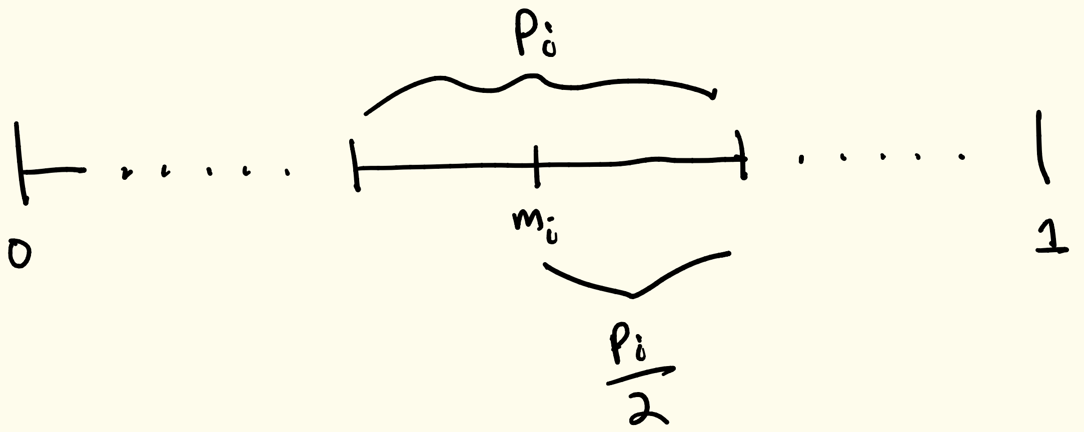
- binary expansion can be of infinite length
- consider truncating to ℓ bits
  - let $\lfloor m \rfloor_\ell$ be the truncation of m to ℓ bits
  - such a truncation implicitly corresponds to an interval of all numbers that could truncate to that same value, e.g., $\lfloor .1010101\overline{10} \rfloor_3 = .101$, but all of $[.101, .1011111\overline{1})$ truncate to same
  - to ensure unique decodability & efficiency truncate to largest implicit interval that lies entirely within original interval

Note: $m - \lfloor m \rfloor_\ell < 2^{-\ell}$

③ For midpoint in interval of width p_i,
truncate to $l_i = \lceil \lg 1/p_i \rceil + 1$ bits

$\Rightarrow$ implicit truncation interval width

$$2^{-l_i} = 2^{-(\lceil \lg 1/p_i \rceil + 1)} = \frac{2^{-\lceil \lg 1/p_i \rceil}}{2} < \frac{2^{-\lg 1/p_i}}{2} = \frac{p_i}{2}$$



$\Rightarrow$ implicit truncation intervals lie entirely within original interval

$\Rightarrow$ implicit truncation intervals do not overlap

$\Rightarrow$ unique decodability

In our example:

- a use $\lceil \lg 1/(1/2) \rceil + 1 = \lceil \lg 2 \rceil + 1 = 2$ bits

  midpoint $0.01000\overline{0}$

  $\boxed{a \to 01}$

- b use $\lceil \lg 1/(1/3) \rceil + 1 = \lceil \lg 3 \rceil + 1 = 3$ bits

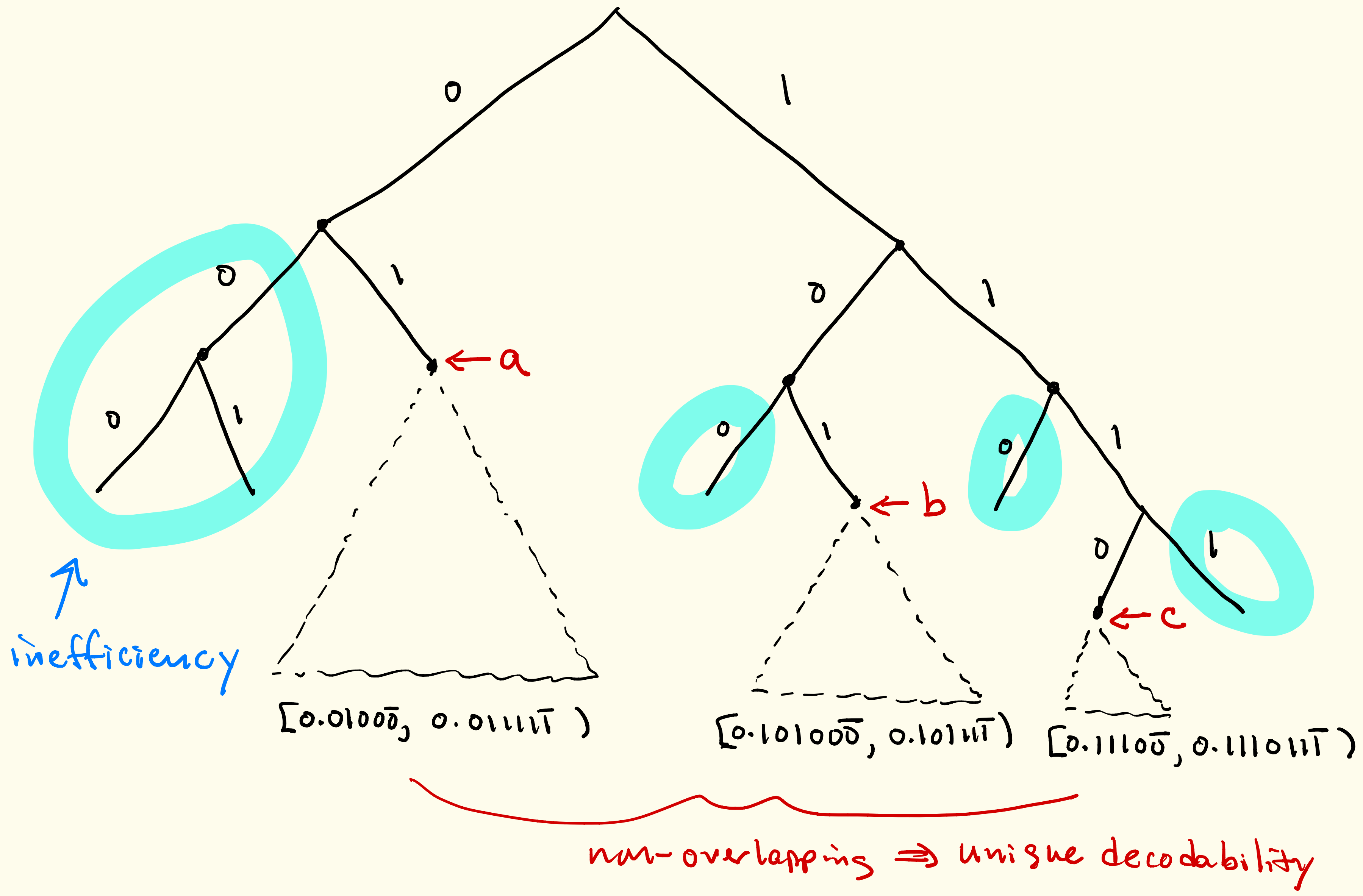
  midpoint $0.1010\overline{10}$

  $\boxed{b \to 101}$

- c use $\lceil \lg 1/(1/6) \rceil + 1 = \lceil \lg 6 \rceil + 1 = 4$ bits

  midpoint $0.111010\overline{10}$

  $\boxed{c \to 1110}$


0
1
0
1
0
1
0
1
0
1
0
1
0
1
← a
← b
← c
inefficiency
[0.01000, 0.01111)
[0.101000, 0.10111)
[0.11100, 0.111011)
non-overlapping ⇒ unique decodability

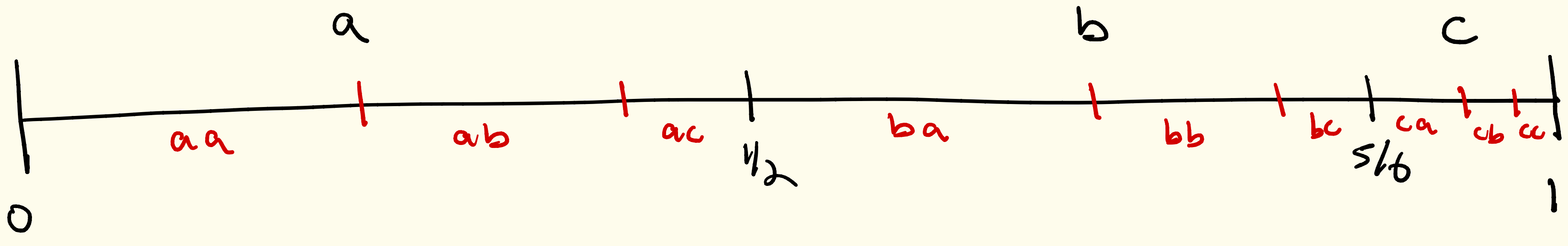
For $l_i = \lceil \lg \frac{1}{p_i} \rceil + 1$:

- $$\begin{aligned} E[L] = \sum_i p_i \cdot l_i &= \sum_i p_i \left(\lceil \lg 1/p_i \rceil + 1 \right) \\ &= \left(\sum_i p_i \cdot \lceil \lg 1/p_i \rceil \right) + 1 \\ &< \left(\sum_i p_i \cdot \left(\lg 1/p_i + 1 \right) \right) + 1 \\ &= \left(\sum_i p_i \lg 1/p_i \right) + 2 \\ &= H(X) + 2 \end{aligned}$$

- $$\begin{aligned} E[L] = \sum_i p_i \cdot l_i &= \sum_i p_i \left(\lceil \lg 1/p_i \rceil + 1 \right) \\ &\geq \sum_i p_i \left(\lg 1/p_i + 1 \right) \\ &= H(X) + 1 \end{aligned}$$

$$\Rightarrow \quad H(X) + 1 \leq E[L] < H(X) + 2$$

Arithmetic Encoding: Apply Shannon-Fano-Elias idea recursively



- easy to keep track of smaller-and-smaller intervals as symbols to encode are processed
- encode final interval as in Shannon-Fano-Elias
- can output bits as you go, e.g., once interval is entirely contained in either $[0, 1/2)$ or $[1/2, 1)$, then can output first 0 or 1
- easy to decode iteratively as well
- effectively get block coding "for free"

Efficiency:

- Use $\lceil \lg 1/w \rceil + 1$ bits if final interval is of width w
- Final interval width is $p(x_1) \cdot p(x_2) \cdot p(x_3) \cdots p(x_n) = p(x_1 x_2 \ldots x_n)$
- By AEP, for large n, with high probability,

$$p(x_1 x_2 \ldots x_n) \sim 2^{-nH(X)}$$

- total bits $\sim \lceil \lg 1/2^{-nH(X)} \rceil + 1 < nH(X) + 2$
- bits per encoded character on average

$$\sim 1/n \, (nH(X) + 2) = H(X) + 2/n$$

→ approaches entropy limit

⟹ practical, efficient, and commonly used

Updated 10/17/2024

Part 3: Practice
L12: Compression (2/4)

Javed Aslam, Wolfgang Gatterbauer

cs7840 Foundations and Applications of Information Theory (fa24)

https://northeastern-datalab.github.io/cs7840/fa24/

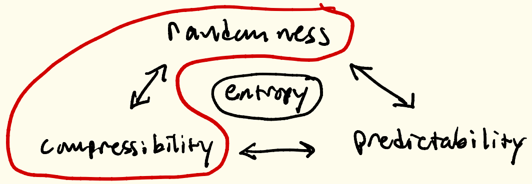
10/16/2024

Last time

- Compression as an application of Information theory
  - taxonomy
  - Shannon-Fano-Elias
  - Arithmetic Encoding

Today

- Final thoughts on Arithmetic Encoding
- Connections between compression & randomness



- Encoding arbitrary integers
  - unary
  - Elias Gamma
  - Elias Delta
  - Elias Omega
  - Fibonacci

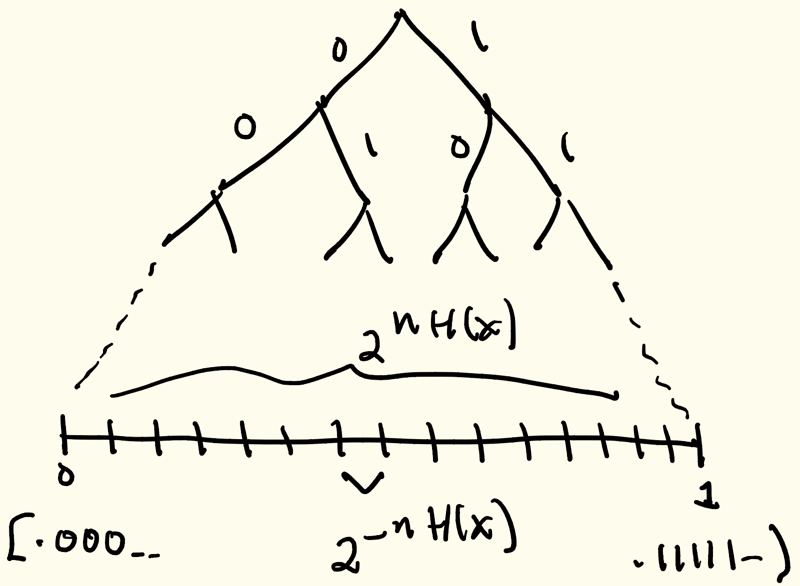
  + some fun facts
- Lempel-Ziv
  - idea & method

Next time

- Lempel-Ziv analysis

Think about what Arith. Enc. <u>outputs</u>, the actual binary string

- by AEP, only see typical sequences, w.h.p.
- $P(x_1 x_2 \dots x_n) \sim 2^{-nH(X)}$ ← width of encoding interval
- $\sim 2^{nH(X)}$ typical sequences ← # encoding intervals



- each interval ~ equally likely
- they are all equally spaced since they are non-overlapping and must fill the range [0,1)
- What is output of Arith. Enc.?

  $\Rightarrow$ random bit strings of length $\sim H(x)$

$\therefore$ From a source whose entropy is $H(x)$,

we can produce $\sim H(x)$ random bits per character

on average via compression

$\Rightarrow$ 1 We can generate ~ fair bits from a

rand var. X via compression.

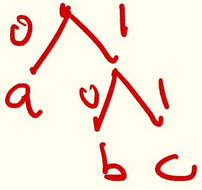
Now suppose we want to generate a r.v. X from fair bits, e.g.,

$$X = \begin{cases} a & \text{w/ prob } 1/2 \\ b & \text{w/ prob } 1/4 \\ c & \text{w/ prob } 1/4 \end{cases}$$

How? Obvious

- Given a stream of fair bits $z_1 z_2 \ldots$
  - if bit is 0, output a
  - if 10, output b
  - if 11, output c
  - repeat

$0 \to a$
$10 \to b$
$11 \to c$

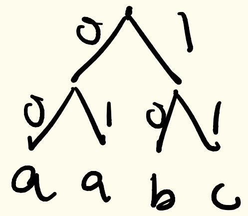


- How efficient?

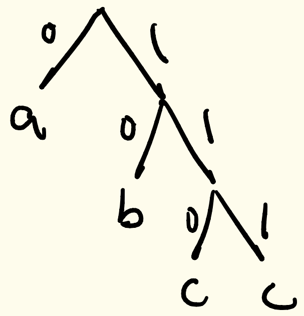
$$\tfrac{1}{2} \cdot 1 + \tfrac{1}{4} \cdot 2 + \tfrac{1}{4} \cdot 2 = 1.5 \text{ bits per character on average}$$
$$= H(X)$$

- But not all such trees are optimal

00 → a
01 → a
10 → b
11 → b



2 bits per char.

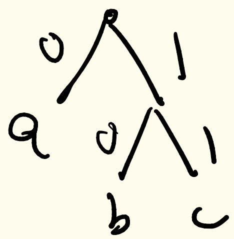


0 → a
10 → b
110 → c
111 → c

1.75 bpc

- How to do constructively?
  - if X is dyadic (inverse powers of 2), just use obvious compression tree for X (e.g. from Huffman)

$p(a) = 1/2$
$p(b) = 1/4$
$p(c) = 1/4$

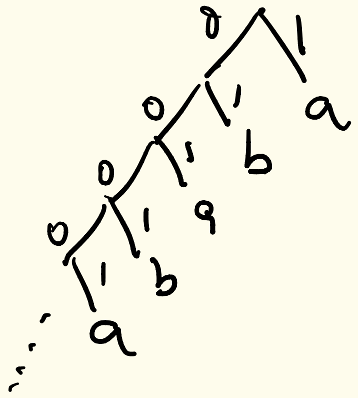


- If X is non-dyadic, consider binary representation of source probabilities

$$X = \begin{cases} a & \text{w/ prob } 2/3 \\ b & \text{w/ prob } 1/3 \end{cases}$$

$$\frac{2}{3} = 0.10101\overline{10} = \underset{1/2}{.1} + \underset{1/8}{.001} + \underset{1/32}{.00001}$$

$$\frac{1}{3} = 0.01010\overline{01} = \underset{1/4}{.01} + \underset{1/16}{.0001} + \underset{1/64}{.000001}$$



- Can show that using a binary expansion tree

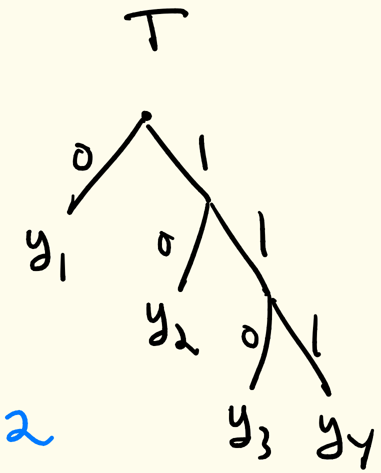
$$H(X) \le E\{T\} \le H(X) + 2$$

- How to reason about efficiency of any such construction

1. Tree should be complete
   - every node is either a leaf or has two descendants
2. Prob of leaf at depth k is 2^{-k}
3. Many leaves may be labelled w/ same output symbol
   - sum of prob. must equal desired ~~total~~ prob.
4. Expected # fair bits using tree T, $E[T]$, is the expected depth of tree.

Analysis

- Let $\mathcal{Y} = \{y_1 y_2 \ldots y_n\}$ be set of leaves
- Let $k(y)$ be depth of y e.g. $k(y_2) = 2$
- Let Y be a r.v. with dist $p(y) = 2^{-k(y)}$
- Let X be r.v. of interest to be generated, which is a <u>mapping</u> from Y



$p(y_1) = 1/2$

$p(y_2) = 1/4$

$p(y_3) = 1/8$

$p(y_4) = 1/8$

$y_1 \to x_1 = a$

$y_2 \to x_2 = b$

$y_3 \to x_3 = c$

$y_4 \to x_3 = c$

Thm: $E\{T\} = H(Y)$

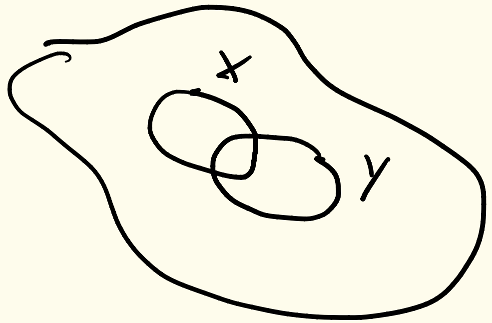
Pf: $$H(Y) = -\sum_{y \in \mathcal{Y}} p(y) \lg p(y)$$

$$= -\sum_{y \in \mathcal{Y}} 2^{-k(y)} \lg 2^{-k(y)}$$

$$= \sum_{y \in \mathcal{Y}} k(y) \cdot 2^{-k(y)}$$

$$= E\{T\} \quad \checkmark$$

thm $H(Y) \ge H(X)$



pf

because X is a deterministic mapping from Y

$$\overset{0}{\cancel{H(X|Y)}} + H(Y) = H(X,Y) = H(X) + H(Y|X)$$

$$\Rightarrow H(Y) = H(X) + \underbrace{H(Y|X)}_{\ge 0}$$

$$\Rightarrow H(Y) \ge H(X)$$

- only achieves equality if 1-to-1 mapping X to Y
- otherwise, strictly positive
  - inefficiency

$\therefore$ # fair bits on average to generate X

$$= E\{T\} = H(Y) \geq H(X)$$

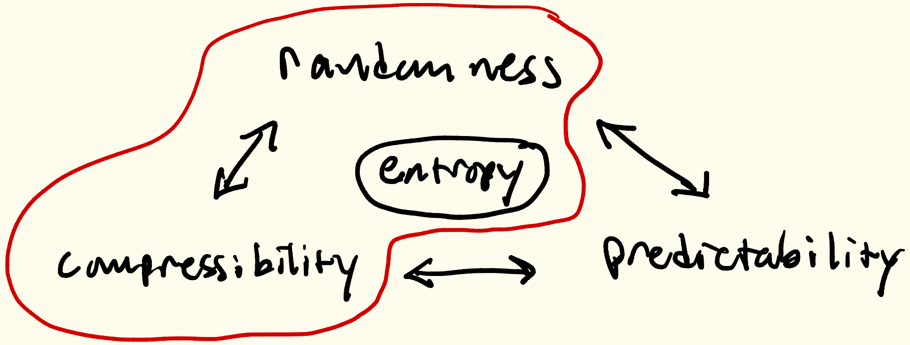
$\Rightarrow$ need at least entropy $H(X)$ fair bits, on average, to generate r.v. X

Upshot:

1. Can generate fair bits from r.v. X via compression

2. Can generate r.v. X from fair bits via compression-like trees

3. Entropy dictates limits of efficiency



Updated 10/22/2024

Part 3: Practice
L13: Compression (3/4)
[Encoding arbitrary integers, Lempel-Ziv]

Javed Aslam, Wolfgang Gatterbauer

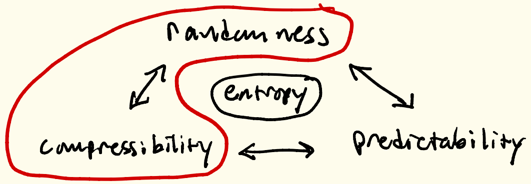
cs7840 Foundations and Applications of Information Theory (fa24)

https://northeastern-datalab.github.io/cs7840/fa24/

10/21/2024

Last time

- Final thoughts on Arithmetic Encoding
- Connections between compression & randomness
  - fair bits from r.v. via compression
  - r.v. from fair bits via compression-like trees to "decode" a fair bit stream



Today

- Encoding arbitrary integers
  - unary
  - Elias Gamma
  - Elias Delta
  - Elias Omega
  - Fibonacci

  + some fun facts
- Lempel - Ziv

Next time

- Finish L-Z or next topics (WG)

Universal codes for strictly positive integers $\{1, 2, 3, \ldots\}$

- Why?
  - Encoding an integer n requires $\lfloor \lg n \rfloor + 1$ bits

    $13: \quad \lfloor \lg 13 \rfloor + 1 = 3 + 1 = 4$ bits

    - $\lfloor \lg 13 \rfloor = 3$ is highest power of 2 in 13
    - need bits for $2^0\, 2^1\, 2^2 \ldots 2^{\lfloor \lg 13 \rfloor} = \lfloor \lg 13 \rfloor + 1$ bits

    $$\begin{array}{rcccc} & 2^3 & 2^2 & 2^1 & 2^0 \\ & 8 & 4 & 2 & 1 \\ 13 = & 1 & 1 & 0 & 1 \end{array}$$

  - if the range of n is known, you can fix the number of bits needed: $\lfloor \lg n_{max} \rfloor + 1$
  - If the range of n is not known, need a different strategy: universal codes

- what about zero? what about negative integers?

- create mappings:

$$\begin{array}{ll}
0 \to 1 & 0 \to 1 \\
1 \to 2 & 1 \to 2 \\
2 \to 3 & -1 \to 3 \\
\vdots & 2 \to 4 \\
n \to n+1 & -2 \to 5 \\
 & 3 \to 6 \\
 & \vdots
\end{array}$$

$$n \to \begin{cases} 2n & \text{if } n \geq 0 \\ 2|n|+1 & \text{if } n < 0 \end{cases}$$

- thus, w.l.o.g., can restrict to strictly positive integers

① Unary

To encode n, use n zeroes followed by a trailing one

$5 \rightarrow 000001$

Requires $n+1$ bits

② Elias Gamma

- let $N = \lfloor \lg n \rfloor$ be the highest power of 2 in n
- separate the binary encoding of n into its highest power of 2 followed by the remaining N bits
- encode N in unary
- append remaining N bits of n

13: $N = \lfloor \lg 13 \rfloor = 3$

$$
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
8 & 4 & 2 & 1 \\
1 & 1 & 0 & 1
\end{array}
$$

$$
\underbrace{\overbrace{000}^{\#\text{ bits past initial } 1}\,1}_{N \text{ in unary}}\;\overbrace{\underbrace{101}_{\text{trailing } N \text{ bits}}}^{n \text{ in binary}}
$$

- requires $2\lfloor \lg n \rfloor + 1$ bits
  $\sim 2 \lg n$
- unary part seems wasteful...

③ Elias Delta

- Separate the binary encoding of n into its highest power of 2 followed by the remaining N bits
- Encode $N+1$ w/ Elias Gamma
- Append final N bits of n

$13: \quad N = \lfloor \lg 13 \rfloor = 3 \qquad N+1 = 4 \quad \text{Elias Gamma} \quad \underbrace{00}_{2\text{ bits}}\underbrace{100}_{4}$

$$\begin{array}{cccc} 2^3 & 2^2 & 2^1 & 2^0 \\ 8 & 4 & 2 & 1 \\ 1 & 1 & 0 & 1 \end{array} \rightarrow \underbrace{00100}_{\text{Elias Gamma}}\ \underbrace{101}_{\text{remaining bits}}$$

$$\underbrace{\lfloor \lg n \rfloor}_{\text{remaining bits}} + \underbrace{2 \lfloor \lg (\overbrace{\lfloor \lg n \rfloor + 1}^{N+1}) \rfloor + 1}_{\text{Elias Gamma}} \sim \lg n + 2 \lg \lg n$$

- why not apply recursively to get something like $\lg n + \lg \lg n + \lg \lg \lg n + \ldots$

(4) Elias Omega

output = 0
while $n > 1$
  output = dec2bin(n) . output
  $n = \lfloor \lg n \rfloor$

dec2bin(n) : binary representation of decimal n

a.b : a appended w/ b

13 : $\underset{3}{11}$ $\underset{13}{1101}$ $\underset{\text{trailing zero}}{0}$

$$1 + \underbrace{(\lfloor \lg n \rfloor + 1) + (\lfloor \lg \lfloor \lg n \rfloor \rfloor + 1) + \cdots}_{\lg^* n \text{ times}} \quad \text{bits}$$

n	$\lg^* n$
2	1
$2^2 = 4$	2
$2^4 = 16$	3
$2^{16} = 65536$	4
$2^{65536} \sim 10^{19728}$	5

Note: There are only $\sim 10^{80}$ atoms in the entire observable universe!
$\Rightarrow \lg^* n \leq 5$ for all practical purposes

Lempel-Ziv

Basic idea:

- parse string into unique substrings
- break unique substrings into previously seen substring + suffix bit
- encode reference to previously seen substring + suffix bit
- lots of variants, depending on how one encodes references
  - absolute index (2-pass with fixed # bits or 1-pass w/ Elias)
  - relative index (1-pass)
  - sliding windows to collect index distributions and then Huffman code
  - etc.

Worked example w/ back references:

① 1 0 1 1 0 1 0 1 1 1 0 1

② 1 | 0 | 11 | 01 | 011 | 10 | 1

substring #	1	2	3	4	5	6	7
③	$(\varepsilon,1)$	$(\varepsilon,0)$	$(1,1)$	$(0,1)$	$(01,1)$	$(1,0)$	$(\varepsilon,1)$
④	$(0,1)$	$(0,0)$	$(2,1)$	$(2,1)$	$(1,1)$	$(5,0)$	$(0,1)$

substring #	1	2	3	4	5	6	7
⑤	$(\_,1)$	$(0,0)$	$(10,1)$	$(10,1)$	$(001,1)$	$(101,0)$	$(000,1)$

$\Rightarrow$ 1 00 101 101 0011 1010 0001

Note:
- every unique substring is a prefix + suffix bit
- prefix may be empty, ε
- determine back references for prefixes; use zero for ε
- use $\lceil \lg k \rceil$ bits to encode back references for substring k why?

Yes, longer than input string, but input was short

Example: My initials in ASCII

j → 0110 1010

a → 0110 0001

a → 0110 0001

⇒ 0110 1010 0110 0001 0110 0001

You try ...

Example: My initials in ASCII

j → 01101010

a → 0110 0001

a → 0110 0001

⇒ 0110 1010 0110 0001 0110 0001

0 | 1 | 10 | 101 | 00 | 11 | 000 | 01 | 011 | 0000 | 1

(0,0) (0,1) (1,0) (1,1) (4,0) (4,1) (2,0) (7,1) (1,1) (3,0) (0,1)

(–,0) (0,1) (01,0) (01,1) (100,0) (100,1) (010,0) (111,1) (0001,1) (0011,0) (0000,1)

0010 1001 1100 0100 1010 0111 1000 1100 1100 0001

Again, longer, but... when processing long string will get long substrings that will be represented compactly with short binary encodings of references. (AEP argument)

Lempel-Ziv analysis setup:

- We will assume an i.i.d. source of bits, with encoding in bits
  - works for Markov sources over arbitrary alphabets
- If a string of length n is parsed into c unique substrings, we will assume that references are encoded w/ $\lg c$ bits
  - as if using absolute indexes
  - worse than method described (relative indexes)
  - actual implementations do even better w/ sliding windows to capture distributions over (relative) indexes

Analysis highlights & steps:

① Let $c = c(n)$ be # distinct substrings in parsing of string of length n. Then $\boxed{c \leq \frac{n}{(1-\varepsilon_n)\lg n}}$ where $\varepsilon_n \to 0$ as $n \to \infty$

Pf: counting argument

② Let $\vec{x} = x_1 x_2 \dots x_n$ be string of length n

Let $\vec{y} = y_1 y_2 \dots y_c$ be any parsing of that string into c substrings

$\forall \ell$, let c_ℓ be # substrings of length ℓ $\sum_\ell c_\ell = c$ & $\sum_\ell \ell \cdot c_\ell = n$

Then $\boxed{\sum_\ell c_\ell \lg c_\ell \leq -\lg p(\vec{x})}$

③ $\boxed{c \lg c \leq \sum_\ell c_\ell \lg c_\ell + c\left[\lg\left(\frac{n}{c}+1\right) + \lg e\right]}$

④ $\boxed{c(\lg c + 1) \leq -\lg p(\vec{x}) + c\left[\lg\left(\frac{n}{c}+1\right) + \lg e + 1\right] = -\lg p(\vec{x}) + O\left(\frac{n \lg \lg n}{\lg n}\right)}$

⑤ $\boxed{\lim_{n \to \infty} \frac{c(\lg c + 1)}{n} \leq H(x) \text{ with prob } 1}$ $c \lg c$ to $\sum_\ell c_\ell \lg c_\ell$ to $-\lg p(\vec{x})$ to $H(x)$

② Let $\vec{x} = x_1 x_2 \cdots x_n$ be string of length n

Let $\vec{y} = y_1 y_2 \cdots y_c$ be any parsing of that string into c substrings

$\forall \ell$, let c_ℓ be # substrings of length ℓ $\sum_\ell c_\ell = c$ & $\sum_\ell \ell \cdot c_\ell = n$

Then $\boxed{\sum_\ell c_\ell \lg c_\ell \le -\lg p(\vec{x})}$

Pf: $\lg p(\vec{x}) = \lg p(\vec{y})$ i.i.d: $p(\vec{z}) = p(z_1)p(z_2)\cdots p(z_k) = \prod_{i=1}^{k} p(z_i)$

$$= \lg p(y_1 y_2 \cdots y_c)$$

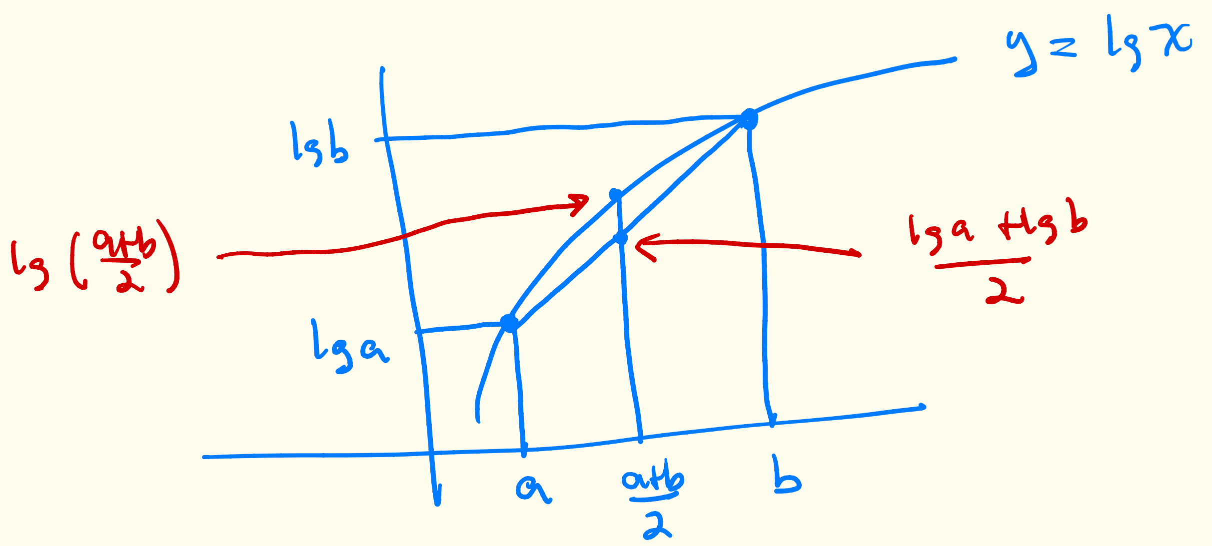
$$= \lg\left(\prod_{i=1}^{c} p(y_i)\right)$$

$$= \sum_{i=1}^{c} \lg p(y_i)$$

$$= \sum_\ell \sum_{i:|y_i|=\ell} \lg p(y_i)$$ reordered sum

$$= \sum_\ell c_\ell \underbrace{\left[\frac{1}{c_\ell} \sum_{i:|y_i|=\ell} \lg p(y_i)\right]}_{\text{avg. of logs } \le \text{ log of avg. by Jensen's Inequality}}$$

Jensen's Inequality :



$\Longrightarrow$ avg. of logs $\leq$ log of average

$$= \sum_{\ell} c_\ell \left[\frac{1}{c_\ell} \sum_{i : |y_i| = \ell} \lg p(y_i) \right]$$

$$\leq \sum_{\ell} c_\ell \lg \left[\frac{1}{c_\ell} \cdot \sum_{i : |y_i| = \ell} p(y_i) \right]$$

$$\leq \sum_{\ell} c_\ell \lg \frac{1}{c_\ell}$$

$$= - \sum_{\ell} c_\ell \lg c_\ell$$

So... $\lg p(\vec{x}) \leq - \sum_{\ell} c_\ell \lg c_\ell$

$$\Longleftrightarrow \boxed{\sum_{\ell} c_\ell \lg c_\ell \leq -\lg p(\vec{x})}$$

. but y_i are distinct, so

$$\sum_{i : |y_i| = \ell} p(y_i) \leq 1$$

③ $\boxed{c \lg c \le \sum_\ell c_\ell \lg c_\ell + c\left[\lg\left(\frac{n}{c}+1\right) + \lg e\right]}$

Pf: $\sum_\ell c_\ell \lg c_\ell = c \cdot \sum_\ell \frac{c_\ell}{c} \lg\left(\frac{c_\ell}{c} \cdot c\right)$

$= c \cdot \sum_\ell \pi_\ell \lg(\pi_\ell \cdot c)$

$= c \cdot \sum_\ell \pi_\ell (\lg c + \lg \pi_\ell)$

$= c \lg c \cdot \sum_\ell \pi_\ell + c \sum_\ell \pi_\ell \lg \pi_\ell$

$= c \lg c - c\, H(\vec{\pi})$

... we need to bound $H(\vec{\pi})$

- let $\pi_\ell = \frac{c_\ell}{c}$
- distribution over substring lengths
- $\sum_\ell \pi_\ell = 1$

- but $n = \sum_\ell \ell \cdot c_\ell$

$= c \sum_\ell \ell \cdot \pi_\ell$

$\Leftrightarrow E[\vec{\pi}] = n/c$

- c substrings covering n bits

$\Rightarrow$ avg. substring length is n/c

- What is maximum $H(\vec{\pi})$ subject to constraint $E[\vec{\pi}] = n/c$?

- Max. ent. distributions, solved via constrained optimization
  - will cover later when discuss max. ent. method

- Here, can show that

$$H(\vec{\pi}) \leq \lg\left(\frac{n}{c}+1\right) + \lg e$$

- entropy can't be too high if mean is small

So,
$$\sum_{\ell} c_\ell \lg c_\ell = c \lg c - c\, H(\vec{\pi})$$
$$\geq c \lg c - c\left[\lg\left(\frac{n}{c}+1\right) + \lg e\right]$$

$$\Leftrightarrow \quad \boxed{c \lg c \leq \sum_{\ell} c_\ell \lg c_\ell + c\left[\lg\left(\frac{n}{c}+1\right) + \lg e\right]}$$

(4) Lempel-Ziv # bits to encode is $c(\lg c + 1)$

substrings

bits to encode reference

suffix bit

$$c(\lg c+1) \leq \sum_{\ell} c_\ell \lg c_\ell + c\left[\lg\left(\tfrac{n}{c}+1\right) + \lg e + 1\right]$$

$$\leq -\lg p(\vec{x}) + c\left[\lg\left(\tfrac{n}{c}+1\right) + \lg e + 1\right]$$

$$= -\lg p(\vec{x}) + O\left(\frac{n \lg\lg n}{\lg n}\right) \quad \text{since } c \leq \frac{n}{(1-\varepsilon_n)\lg n}$$

$$\Rightarrow \boxed{c(\lg c+1) \leq -\lg p(\vec{x}) + O\left(\frac{n \lg\lg n}{\lg n}\right)}$$

⑤ Compression rate in limit

$$\lim_{n\to\infty} \frac{c(\lg c+1)}{n} \le \lim_{n\to\infty}\left[\overbrace{-\frac{1}{n}\lg p(\vec{x})}^{H(x) \text{ by AEP}} + \overbrace{O\left(\frac{\lg\lg n}{\lg n}\right)}^{0}\right]$$

$$= H(x) \quad \checkmark$$

Updated 10/25/2024

Part 3: Practice
L14: Compression (4/4)
[Fibonacci Encodings]

Javed Aslam, Wolfgang Gatterbauer

cs7840 Foundations and Applications of Information Theory (fa24)

https://northeastern-datalab.github.io/cs7840/fa24/

10/23/2024

Last time

- Encoding arbitrary integers
  - Unary
  - Elias Gamma
  - Elias Delta
  - Elias Omega
- Lempel - Ziv

Today

- Fibonacci Encoding
  + some fun facts

~~~~~~

- Decision Trees (W 6)

Next time

- Continue applications w/ Wolfgang
~~~~~~

Fibonacci Encoding: A universal code for integers using a Fibonacci representation

First, let's consider arbitrary integer representations

- Assume we have place values

$$\ldots\ p_4 > p_3 > p_2 > p_1 = 1$$

- Greedy algorithm to produce standard representation:

To encode n:

1. Find largest place value $p_i \le n$
2. Find largest integer constant c_i s.t. $c_i \cdot p_i \le n$
3. Put c_i in position i
4. $n \leftarrow n - c_i \cdot p_i$; repeat
5. Put zeroes in unused place value positions

Example: Binary Representation

- Place values are powers of 2

	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...	64	32	16	8	4	2	1
		1	1	1	1	0	0

$$
\begin{array}{r}
n = 60 \\
-32 \\ \hline
28 \\
-16 \\ \hline
12 \\
-8 \\ \hline
4 \\
-4 \\ \hline
0
\end{array}
$$

Example: Ternary Representation

- place values are powers of 3

	3^4	3^3	3^2	3^1	3^0
...	81	27	9	3	1
		2	0	2	0

$n = 60$

$-2 \times 27 = 54$

6

$-2 \times 3 = 6$

0

* Note, however, that **nothing** in our setup or the greedy algorithm that produces a standard representation **requires** that place values be powers of some base.

- Consider using the Fibonacci sequence as our place values:

$$F(1) = 1$$
$$F(2) = 1$$
$$\forall n > 2 \quad F(n) = F(n-1) + F(n-2)$$

... 89 55 34 21 13 8 5 3 2 1 1

we won't use the extra initial 1

Example: Fibonacci Representation

$n = 60$

$- 55$

5

$- 5$

0

89	55	34	21	13	8	5	3	2	1
	1	0	0	0	0	1	0	0	0

Claims:

1. Representation contains only 0 & 1

why? $p_{i+1} = p_i + p_{i-1}$

$< 2 p_i$ since $p_i > p_{i-1}$

$\Rightarrow$ If needed 2 or more p_i, greedy would pick a p_{i+1} instead

2. Never have consecutive 1s

why? If $\begin{matrix} p_{i+2} & p_{i+1} & p_i \\ 0 & 1 & 1 \end{matrix}$ greedy would choose $\begin{matrix} p_{i+2} & p_{i+1} & p_i \\ 1 & 0 & 0 \end{matrix}$ instead

∴ Fibonacci representation uses only bits (0 & 1) and never has consecutive 1s

⇒ Can use consecutive 1s as a form of <u>punctuation</u> in a universal <u>encoding</u> of integers.

<u>Fibonacci Encoding</u>:

① Create Fibonacci representation of n

② write down least significant bit to most significant bit (LSB → MSB)

Note: MSB is <u>always</u> a 1

③ tack on a trailing 1, which creates a pair of consecutive 1s, which serves as our punctuation

Example: Fibonacci Encoding

$n=60$ · Fibonacci representation: 100001000

· Fibonacci encoding:

LSB → MSB trailing 1

$\overbrace{0001000011}$ 1

punctuation

· Easy to encode and decode

Properties:

① Efficiency $F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$ where

$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$

$\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0.618$

ϕ = "Golden Ratio"

- Since $|\hat{\phi}| < 1$, $\hat{\phi}^n \to 0$ as $n \to \infty$

$\Rightarrow$ for large n, $F_n \approx \frac{\phi^n}{\sqrt{5}}$

$\Rightarrow$ $F_n / F_{n-1} \approx \phi$

$\Rightarrow$ Fibonacci sequence grows (about) exponentially w/ base ϕ

$\Rightarrow$ length of Fibonacci representation/encoding is about

$$\log_\phi n = \frac{\lg n}{\lg \phi} \approx 1.44 \cdot \lg n$$

→ 44% longer than a binary representation

→ but, universal + nice properties (robustness)

② Robustness: bit errors have limited impact

- consider a sequence of integers encoded in Fibonacci
- consider a bit error $\begin{cases} 0 \to 1 \\ 1 \to 0 \end{cases}$

Cases: ⓐ If bit error does not introduce or remove a "11", then error only impacts a single encoded number - everything else OK

ⓑ If bit error introduces a new "11", then corrupt one encoded number and introduce a new encoded number - everything else OK

ⓒ If bit error removes a "11", collapse two encoded integers to one

- everything else ok

And now for some fun... We can use Fibonacci representations to convert from miles to kilometers and inches to centimeters without multiplication.

⟹ To do so, we will leverage the fact that the ratio between consecutive Fibonacci numbers is $\approx \phi = 1.618\ldots$

Ratios:

144	1.6179	89	1.6182	55	1.6176	34	1.619	21	1.615	13	1.625

13	1.625	8	1.6	5	1.66	3	1.5	2	1

⟹ In a Fibonacci representation...

① shifting left effectively multiplies by $\approx \phi = 1.618\ldots$

② shifting right effectively divides by $\approx \phi = 1.618\ldots$

① Miles (or mph) $\Longleftrightarrow$ Kilometers (or Kph)

- There are 1.609 km/mile, very close to $\phi = 1.618$
- To approximately multiply or divide by 1.609, just represent in Fibonacci and shift left or right

Ratios: 1.6179 1.6182 1.6176 1.619 1.615 1.625 1.6 1.66 1.5

144 89 55 34 21 13 8 5 3 2 1

$$60 \text{ mph} = 55+5 \xrightarrow[\text{left}]{\text{shift}} 89+8 = 97 \text{ Kph}$$ (actual: 96.56)

$$100 \text{ Kph} = 89+8+3 \xrightarrow[\text{right}]{\text{shift}} 55+5+2 = 62 \text{ mph}$$ (actual: 62.137)

Aslam, Gatterbauer. Foundations and Applications of Information Theory: https://northeastern-datalab.github.io/cs7840/

② Inches $\Longleftrightarrow$ Centimeters

- There are 2.54 cm/in, close to $\phi^2 = 2.618$
- To approximately <u>multiply</u> or <u>divide</u> by 2.54, just represent in Fibonacci and shift <u>left</u> or <u>right</u> <u>2 places</u>

Ratios: 1.6179 1.6182 1.6176 1.619 1.615 1.625 1.6 1.66 1.5

144 89 55 34 21 13 8 5 3 2 1

$$12 \text{ inches} = 8 + 3 + 1 \xrightarrow[\text{left twice}]{\text{shift}} 21 + 8 + 3 = 32 \text{ cm} \quad (\text{actual: } 30.48)$$

$$100 \text{ cm} = 89 + 8 + 3 \xrightarrow[\text{right twice}]{\text{shift}} 34 + 3 + 1 = 38 \text{ in} \quad (\text{actual: } 39.37)$$