COMP20007 Design of Algorithms Semester 1 2016

Data Compression and Coding

We started with Google's certificate transparency log proofs

- https://www.certificate-transparency.org/log-proofs-work
- We looked particularly at how the Merkle Audit proofs work, and why you only need a proof of size O(log n) to prove your certificate is present, in a tree that holds n certificates.

How is data encoded in most computers?

- ASCII – American Standard Code for Information Interchange
- Unicode and the ISO/IEC 10646: UTF-8, UTF-16, UTF-32
- 8-bit ASCII (ISO-8859-1 (Latin 1))
- So the message

  to be or not to be
- would be coded as ASCII codes

  116 111 32 98 …
- which get mapped to the bit stream

  01110100 01101111 0100000 01100010 …

Cost in bits per symbol

- The data to be stored is called a message
- Each letter to be encoded is called a symbol
- Each bit pattern for each symbol is called a codeword
- A collection of codewords forms a code
- We talk about "sending a message" and "storing some data" interchangeably.
  Storing data is just like sending it to disk.
- The cost of a code can be measured in bits-per-symbol, bps, for a particular message: total bits divided by the number of symbols
- ASCII always has the cost of 8 bps

Wasted bits?

- 8-bit codewords are very convenient on modern computers because there are instructions for fast processing of bytes.
- For English text, ASCII always has the first bit as 0, so a code with 7 bps would lead to less bits being transmitted while still preserving the message
- What about a code with codewords that vary in length?

Unary code

- For storing integers, the unary code is simple

  To store *n*, output *n* 1-bits then a 0-bit

- For example

Number	Codeword
0	0
1	10
2	110
3	1110
4	11110
5	111110

Does the unary code give compression?

- What is the best compression we can get?
- If we know the frequency of occurrence of symbols in our message, then the best we can do is the Entropy of the probability distribution of the symbols
- Defined and proven by Shannon in 1948

  If a symbol occurs with probability p, the best possible encoding for that symbol is $-\log_2 p$

- The inverse of this is that a codeword of length x implies an expected probability of occurrence of 2^{-x}

Back to unary

- If we are using a unary code, what is our expected probability of occurrence for symbols?

Number	Codeword	Length	$2^{-length}$
0	0	1	0.5
1	10	2	0.25
2	110	3	0.125
3	1110	4	0.0625
4	11110	5	0.03125
5	111110	6	0.015625

- So if our message was half 0's, ¼ ones, etc, unary would be the best code (the lowest possible bps)

Static vs semi-static codes

- ASCII and unary are examples of a static code: it assigns the same codewords to symbols regardless of the message
- A semi-static code knows the probability of occurrence of each symbol in the message
- Shannon's entropy theorem:

  Given a message with n symbols each occurring p_i times, the best bps possible is

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

How to find the best semi-static code?

- Message = ABAABBCAAABC

  n=3 symbols {A, B, C}

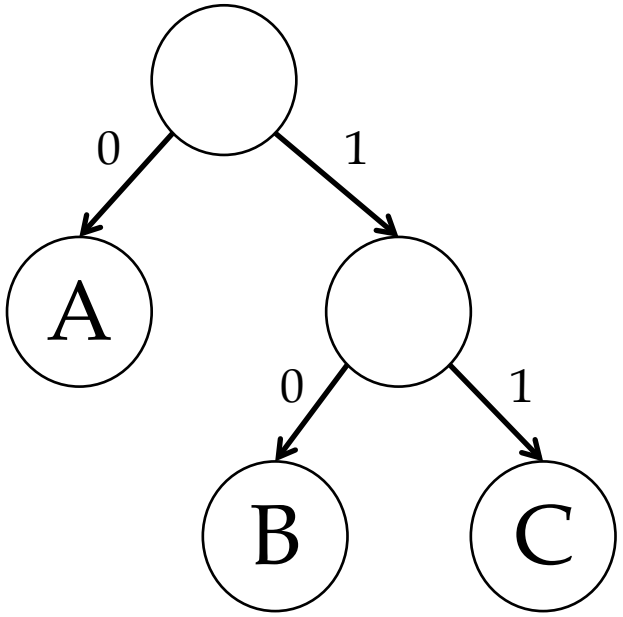
  p = {6/12, 4/12, 2/12}

  $-\log_2(p)$ = {1, 1.6, 2.6}

- We need a whole number of bits per codeword, and we need unique decipherability

Prefix-free codes

- A prefix-free code (or just prefix code) has no codeword as the prefix of any other
- Thus we can decode left to right one bit at a time
- Easy to visualise using a code tree
- Symbols are leaves
- Path from root to leaf defines the codeword
- Each path is unique and prefix-free



How to build a semi-static prefix code?

- Message = ABAABBCAAABC

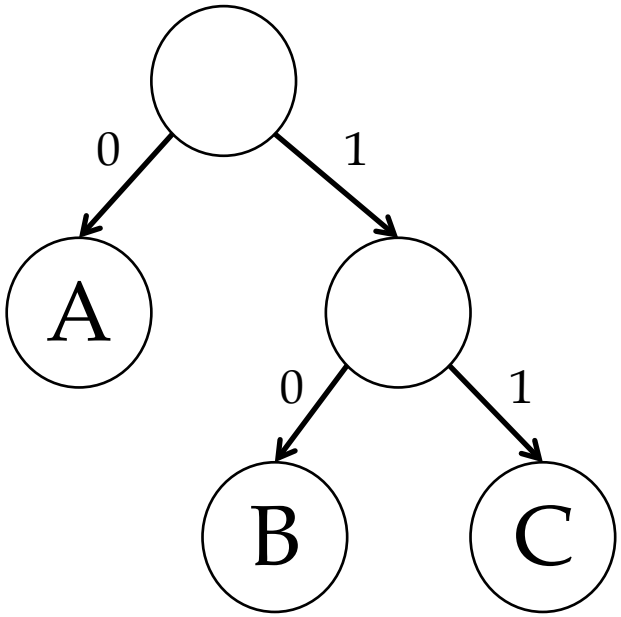
  n=3 symbols {A, B, C}

  p = {6/12, 4/12, 2/12}

  $-\log_2(p)$ = {1, 1.6, 2.6}

  code = {0, 10, 11}

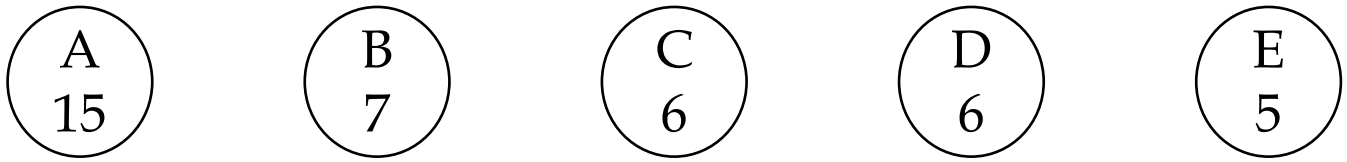
  bps = (6*1 + 4*2 + 2*2)/12 = 1.5



$$H(p) = -\frac{6}{12}\log_2\frac{6}{12} - \frac{4}{12}\log_2\frac{4}{12} - \frac{2}{12}\log_2\frac{2}{12} = 1.46$$

How find the best semi-static code?

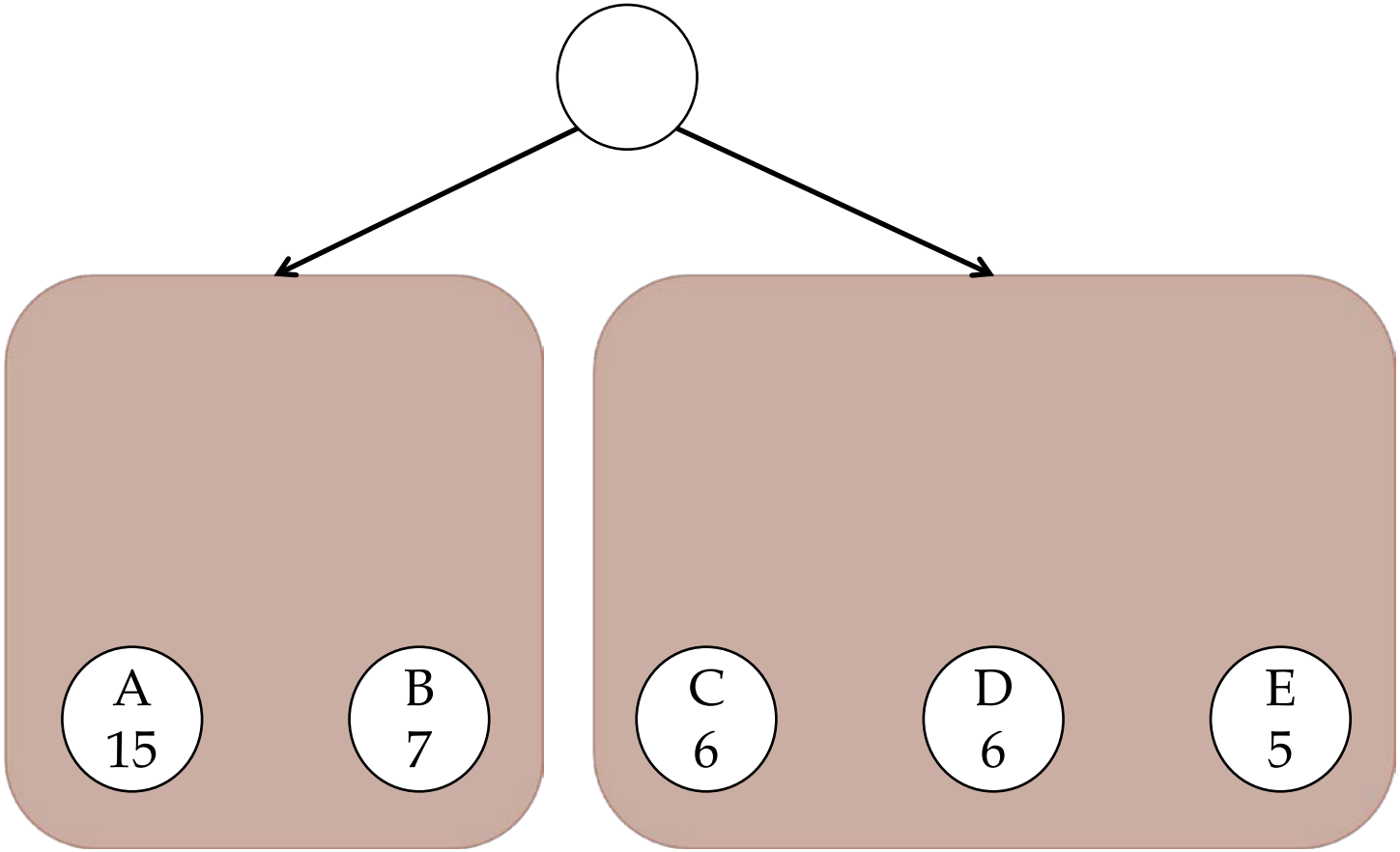
- $p = \{15, 7, 6, 6, 5\}/39$



Divide and Conquer

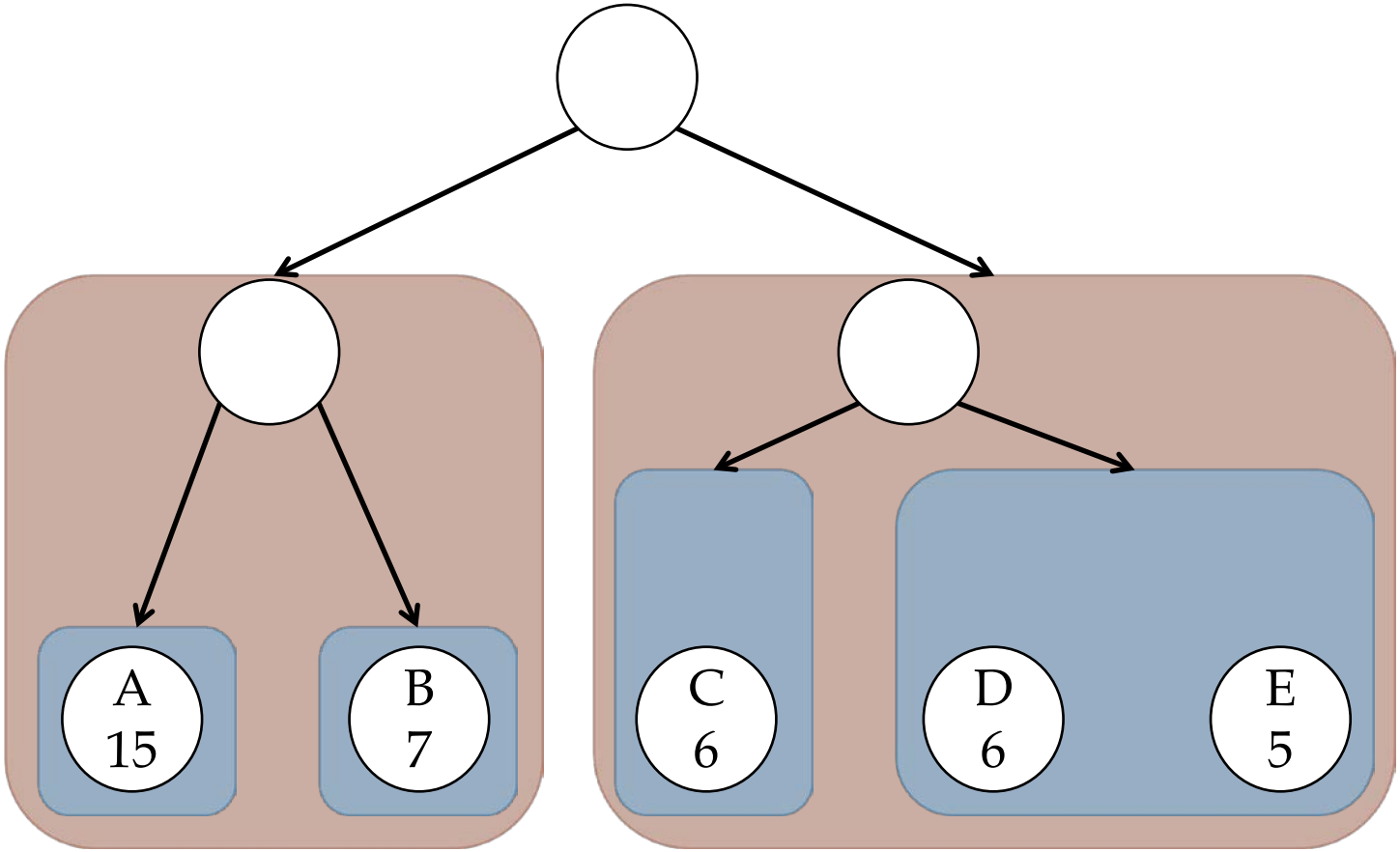
Divide and Conquer code making

- p = {15, 7, 6, 6, 5}/39



Divide and Conquer code making

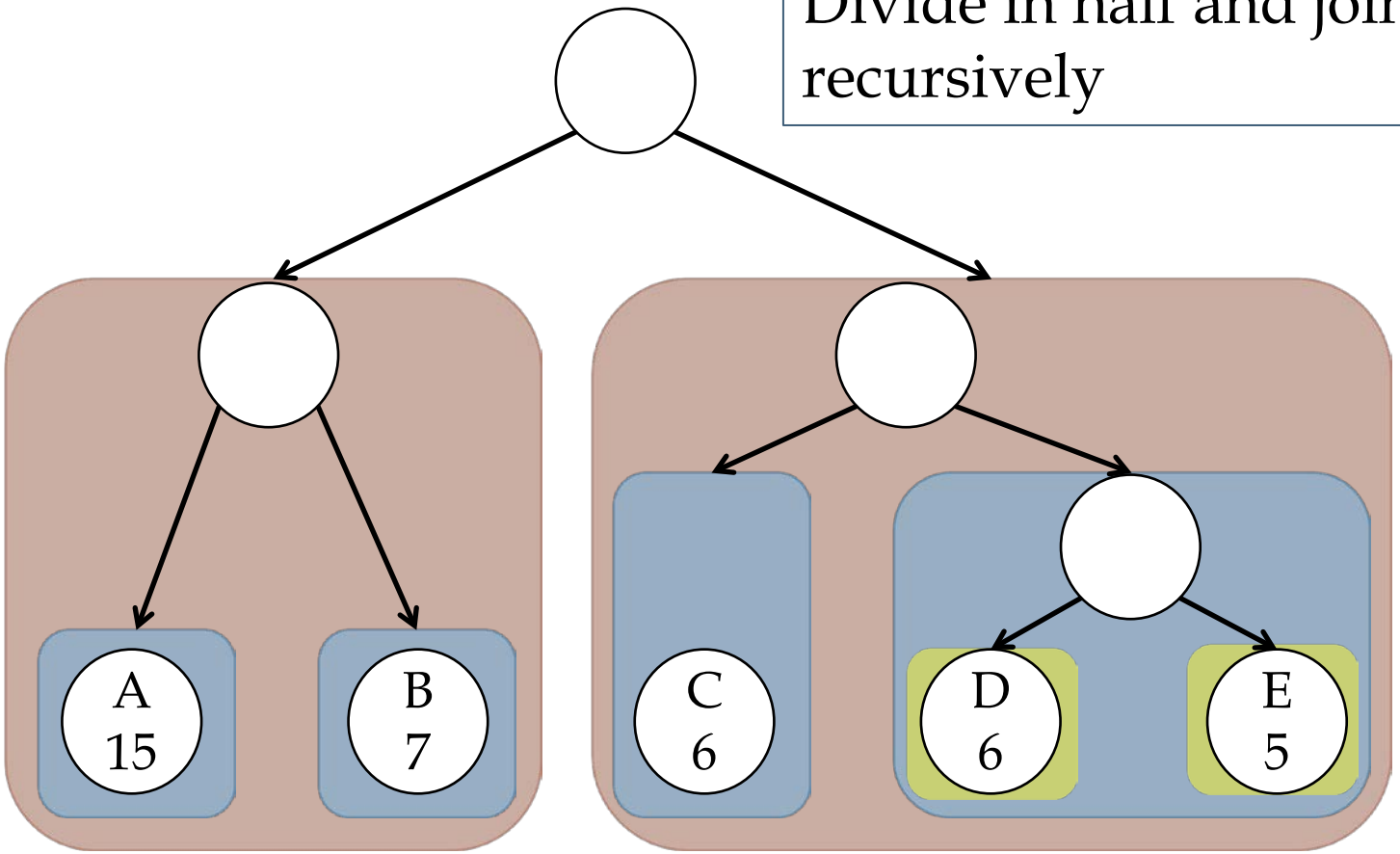
- $p = \{15, 7, 6, 6, 5\}/39$



Divide and Conquer code making

- p = {15, 7, 6, 6, 5}/39

Shannon-Fano algorithm
Divide in half and join recursively



bps = (30 + 14 + 12 + 18 + 15)/39 = 2.28

Bps summary

- p = {15, 7, 6, 6, 5}/39

Method	bps
Shannon-Fano	2.28
Huffman	2.23
Entropy	2.19

- Q: For Shannon-Fano, *if* you can always divide exactly evenly, what's the length of the resulting codeword (in terms of p)?
- Ans: - $\log_2 p$
- Huffman.. Next week with Alistair