CS630 Graduate Algorithms

November 5, 2024

by Dora Erdos and Jeffrey Considine

- Compression

Compression

“The process of encoding information using fewer bits than the original representation”

Also has connections to randomized algorithms, random data, and complexity…

Lossless, Not Lossy Compression

Must be recovering original data exactly, not an approximation.

- Lossy compression introduces tradeoffs with hard to measure qualities.

Example: Run-Length Encoding

Some pieces of data have easy short descriptions.

- aaaaaaaaaaaaaaaaaaaa = a x 20
- abababababababababab = ab x 10
- aaaaaaaaaaccccccccccc = a x 10 + c x 10

Example: Run-Length Encoding

Some pieces of data have easy short descriptions.

- aaaaaaaaaaaaaaaaaaaa = a x 20
- abababababababababab = ab x 10
- aaaaaaaaaacccccccccc = a x 10 + c x 10

This basic idea is called run length encoding.

- Pretty easy case to compress.
- Often used as a subroutine in other compression algorithms.
  - First transform data to increase the number of runs.
  - Then use run length encoding.

Will be looking at somewhat harder encodings today.

Kolmogorov complexity

What is the shortest program that can output a given string s?

Kolmogorov complexity

What is the shortest program that can output a given string s?

The Kolmogorov complexity of string s, $C(s)$, is the length of the shortest program that outputs s.

- Choice of computer is just an additive constant difference.
- So is inclusion/exclusion of standard decompression code.
- However, $C(s)$ is uncomputable.
  - It is impossible to write a program outputting $C(s)$ that is guaranteed to halt.

Kolmogorov complexity

What is the shortest program that can output a given string s?

The Kolmogorov complexity of string s, $C(s)$, is the length of the shortest program that outputs s.

- Choice of computer is just an additive constant difference.
- So is inclusion/exclusion of standard decompression code.
- However, $C(s)$ is uncomputable.
  - It is impossible to write a program outputting $C(s)$ that is guaranteed to halt.

Computable time-bounded variations exist, but they take exponential time to find.

- We want fast compression too.

Entropy (physics)

The entropy S of a physical system is

$$S = -k_B \sum_{s_i} p(s_i) \ln p(s_i)$$

where

- k_B is the Boltzmann constant,
- each s_i refers to a different possible “micro state” of the physical system,
- and $p(s_i)$ is the probability of that micro state.

Entropy (information theory)

The information theory version of entropy was introduced by Claude Shannon while writing about limits communicating with a limited communication channel.

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

where $p(x)$ denotes the probability of the random variable X taking on value x.

Entropy - Physics vs Information Theory

Physics:

$$S = -k_B \sum_{s_i} p(s_i) \ln p(s_i)$$

Information theory:

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

Pretty similar except that $-k_B$ and the unspecified logarithm base…

Entropy (information theory)

Where does this definition come from?

An alternate definition of entropy -

$H(X) = E(I(X))$ where $I(X)$ represents the information content of X.

For a particular probability p, $I(p)$ is intended to measure how much information we get or learn from observing a value x with probability p.

Entropy (information theory)

For a particular probability p, $I(p)$ is intended to measure how much information we get or learn from observing a value x with probability p.

Requirements for $I(p)$:

- $I(p)$ decreases monotonically with p.
  - High probability ~ boring ~ low information.
- $I(1) = 0$.
  - 100% = no uncertainty = no information from observing
- $I(p_1 \cdot p_2) = I(p_1) + I(p_2)$
  - Independent events happen together add the information.

Entropy (information theory)

For a particular probability p, $I(p)$ is intended to measure how much information we get or learn from observing a value x with probability p.

Requirements for $I(p)$:

- $I(p)$ decreases monotonically with p.
  - High probability ~ boring ~ low information.
- $I(1) = 0$.
  - 100% = no uncertainty = no information from observing
- $I(p_1 \cdot p_2) = I(p_1) + I(p_2)$
  - Independent events happen together add the information.

These constraints are sufficient to force

$$I(p) = -\log_b p$$

for some $b > 1$

Entropy (information theory)

These constraints are sufficient to force

$$I(p) = -\log_b p$$

for some $b > 1$, so

$$H(X) = E(I(X)) = -\sum_{x \in X} p(x) \log_b p(x)$$

and that unspecified b gives us a scaling constant.

We call $H(X)$ the binary entropy if $b = 2$.

Shannon's Source Coding Theorem

If you are compressing a stream of independent and identically distributed random variables,

- then it is impossible for the average number of bits per symbol to be better than the binary entropy,
- but you can get arbitrarily close.

So the binary entropy is our idealized target for compression.

Prefix Codes

A prefix code is an encoding mapping “prefixes” to data values.

For example,

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

Example from CLR.

Prefix Codes

A prefix code is an encoding mapping “prefixes” to data values.

For example,

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

Properties of prefix codes

- None of these encodings is a prefix of another encoding.
- Encoding = concatenation of data value encodings
- Decoding = decode only prefix at beginning of encoding, then remove

Example from CLR.

Prefix Codes

A prefix code is an encoding mapping “prefixes” to data values.

For example,

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

110001001101111 = 1100 0 100 1101 111 = faced

Example from CLR.

Aside: Large Language Model Tokenization Looks Similar But 🤦

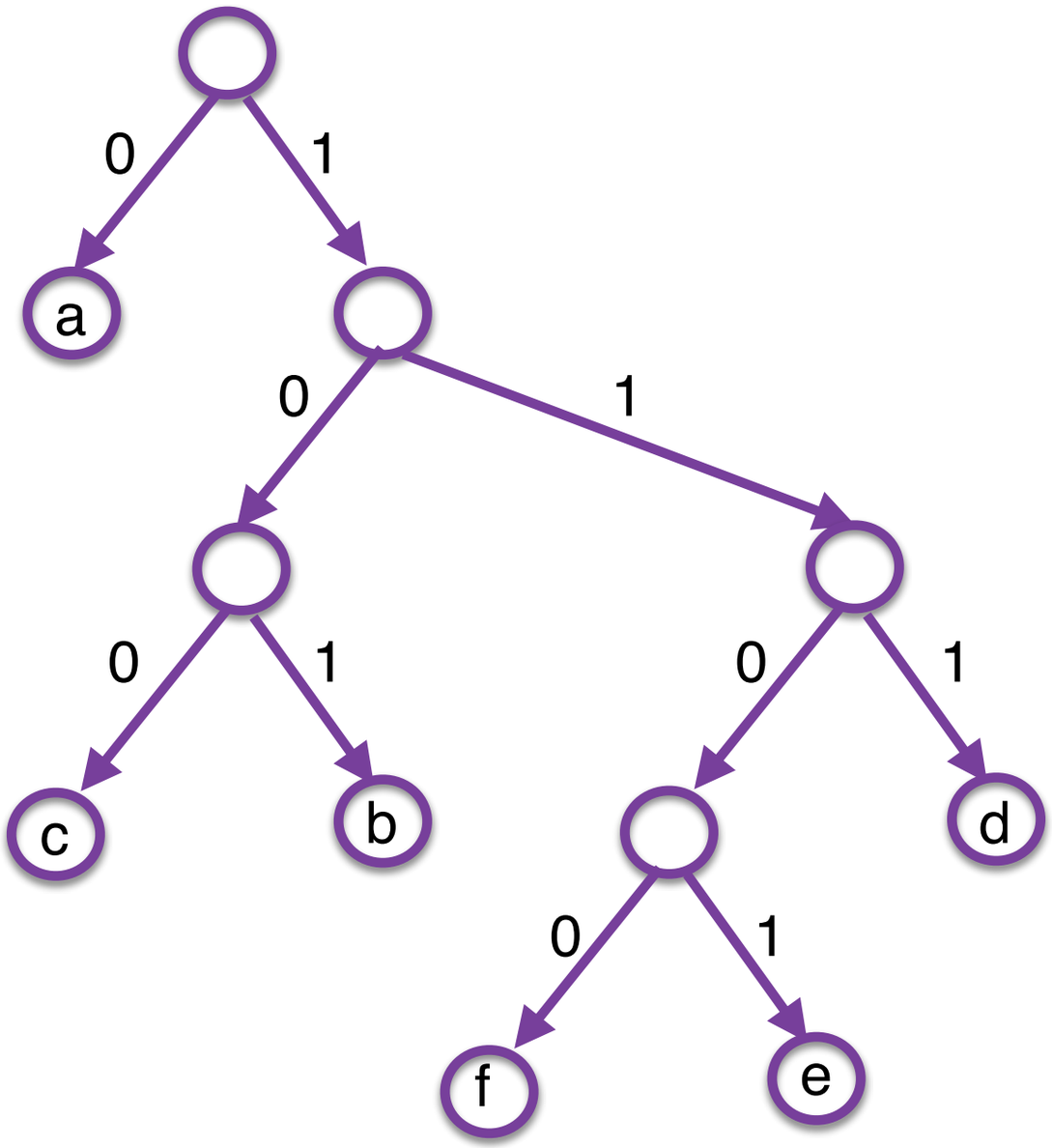
Large language models use a similar scheme to break text into tokens.

- But tokens may be prefixes of other tokens.
- Token identification greedily picks longest prefix matching a token.
  - Prefix codes pick unique prefix found.

Prefix Codes as Trees

Given this prefix code,

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



If we navigate from the root reading bits from the encoding, when we reach a leaf, the leaf's value is the newly decoded value.

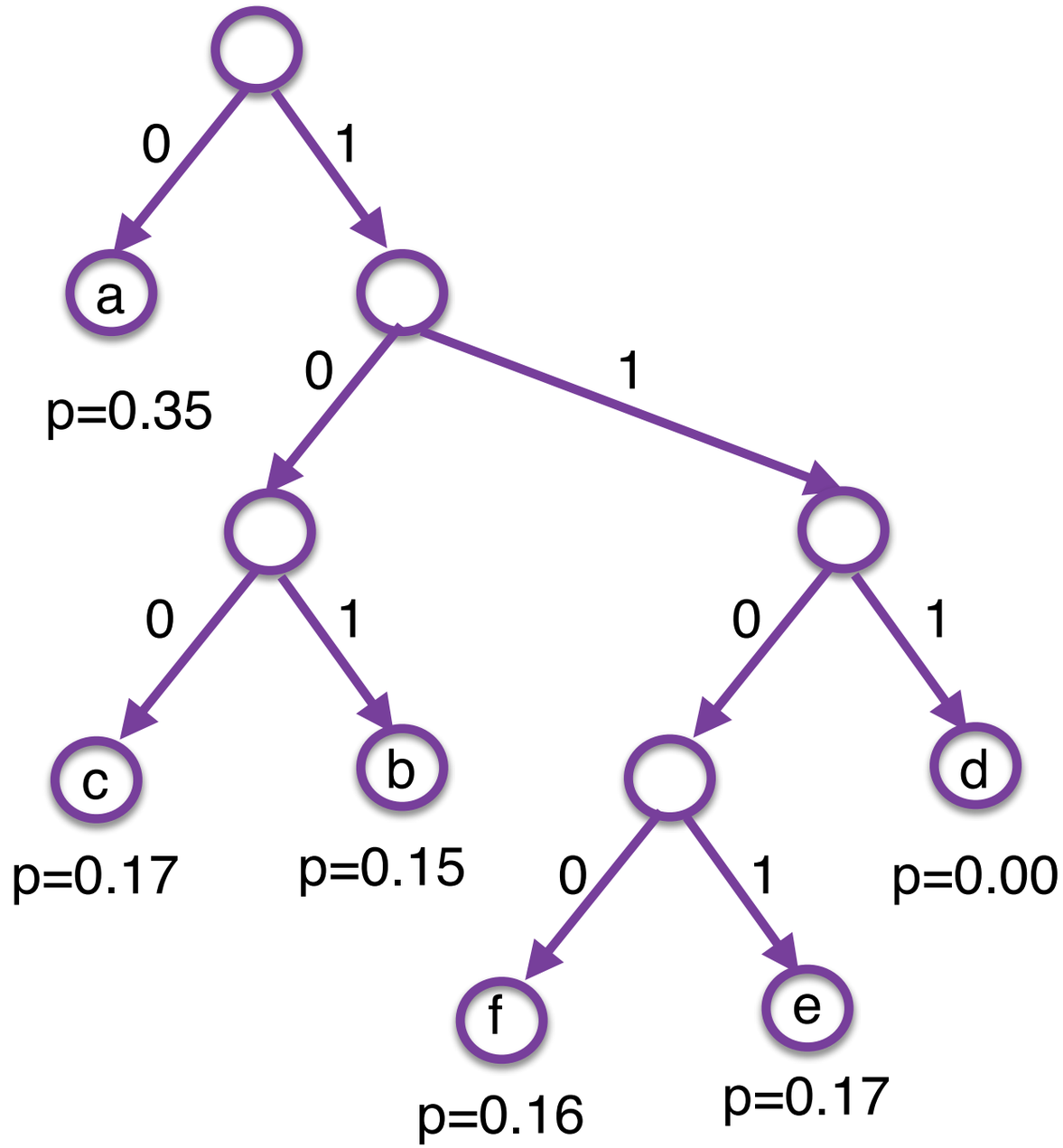
Note that the values are all at the leaves of this tree.

- This is from the prefix property and avoids ambiguity.

Example from CLR.

Prefix Codes for Compression

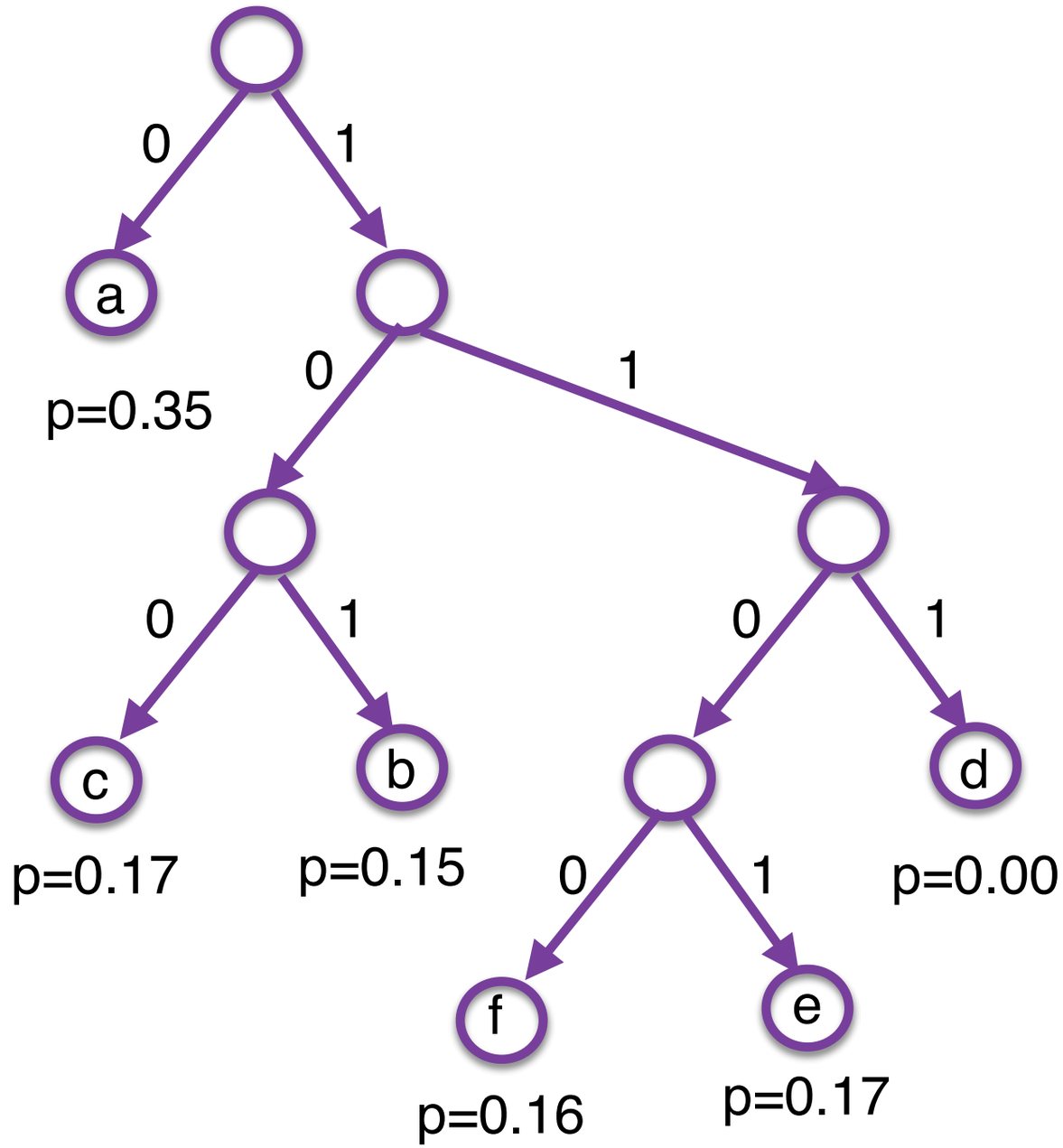
An optimal prefix code minimizes the average path length.



Prefix Codes for Compression

An optimal prefix code minimizes the average path length.

- We need probabilities for each value to reason about the average path length.



Prefix Codes for Compression

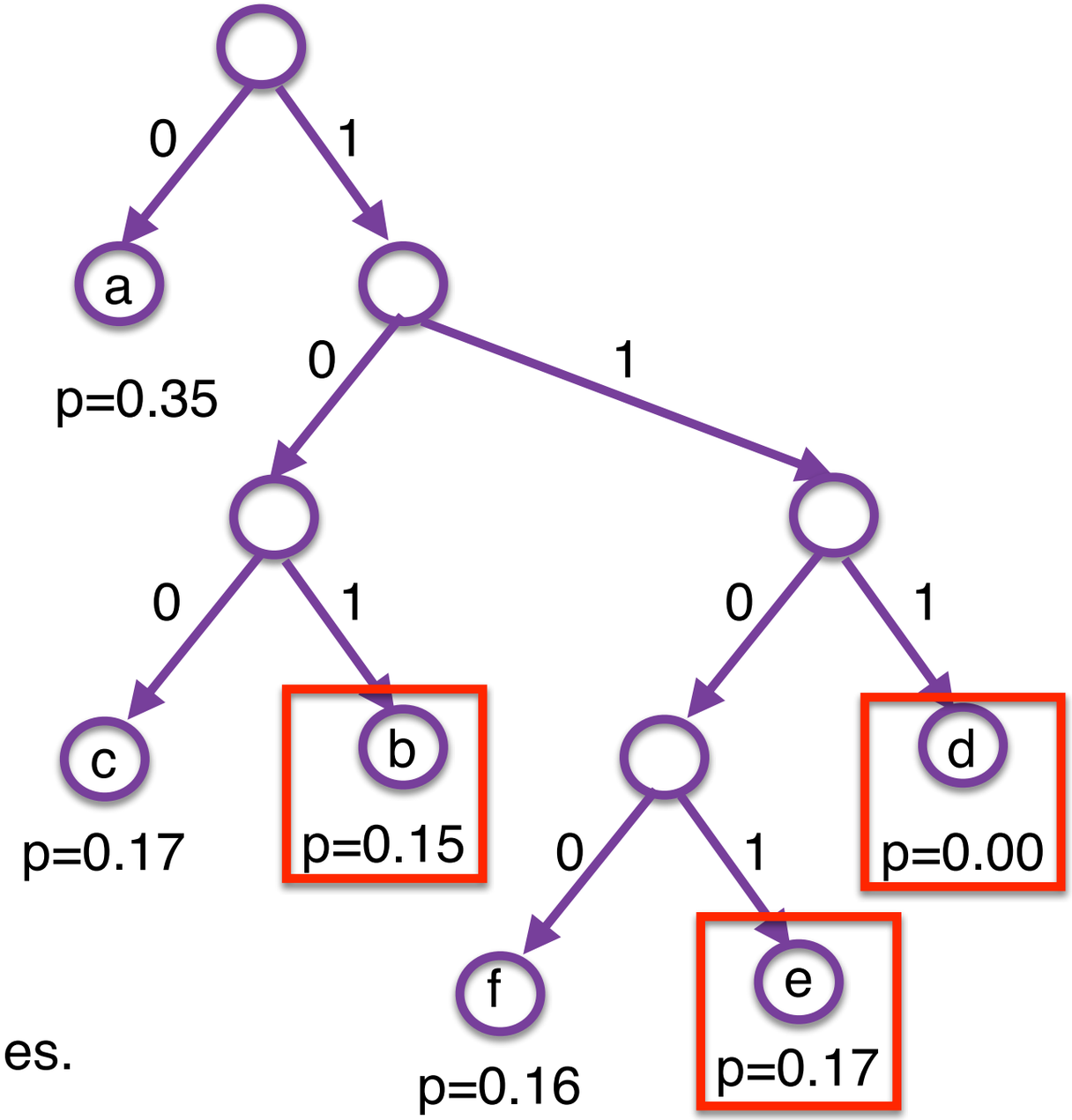
An optimal prefix code minimizes the average path length.

- We need probabilities for each value to reason about the average path length.

Greedy property

- Prefer shorter path lengths for higher probability values.
- Swap prefixes when shorter path has lower probability.

- Also get rid of probability zero values.
  - Don't just push them to the bottom.



Shannon-Fano Coding (specifically Fano's)

Shannon-Fano coding refers to a couple different methods by Shannon and Fano.

- Both trying to find good prefix codes.
- Names commingled by historical accident.
- Presenting Fano's one now.

Given sorted list of probabilities, repeatedly split in half as evenly as possible.

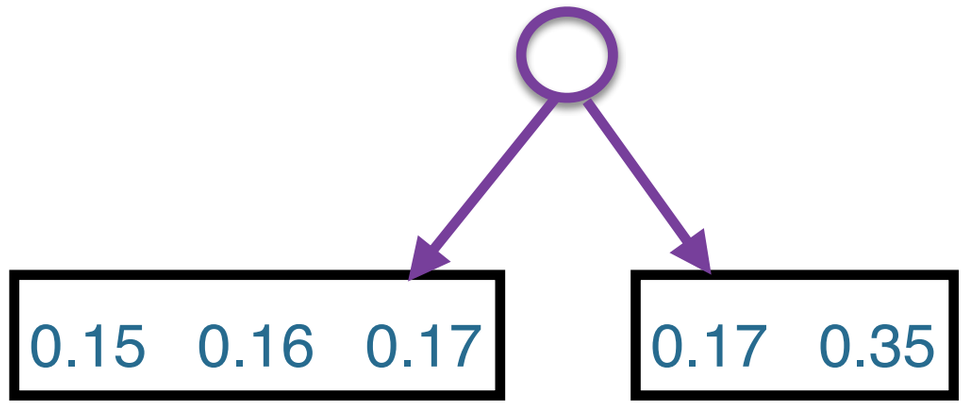
0.15	0.16	0.17	0.17	0.35

Shannon-Fano Coding (specifically Fano's)

Shannon-Fano coding refers to a couple different methods by Shannon and Fano.

- Both trying to find good prefix codes.
- Names commingled by historical accident.
- Presenting Fano's one now.

Given sorted list of probabilities, repeatedly split in half as evenly as possible.

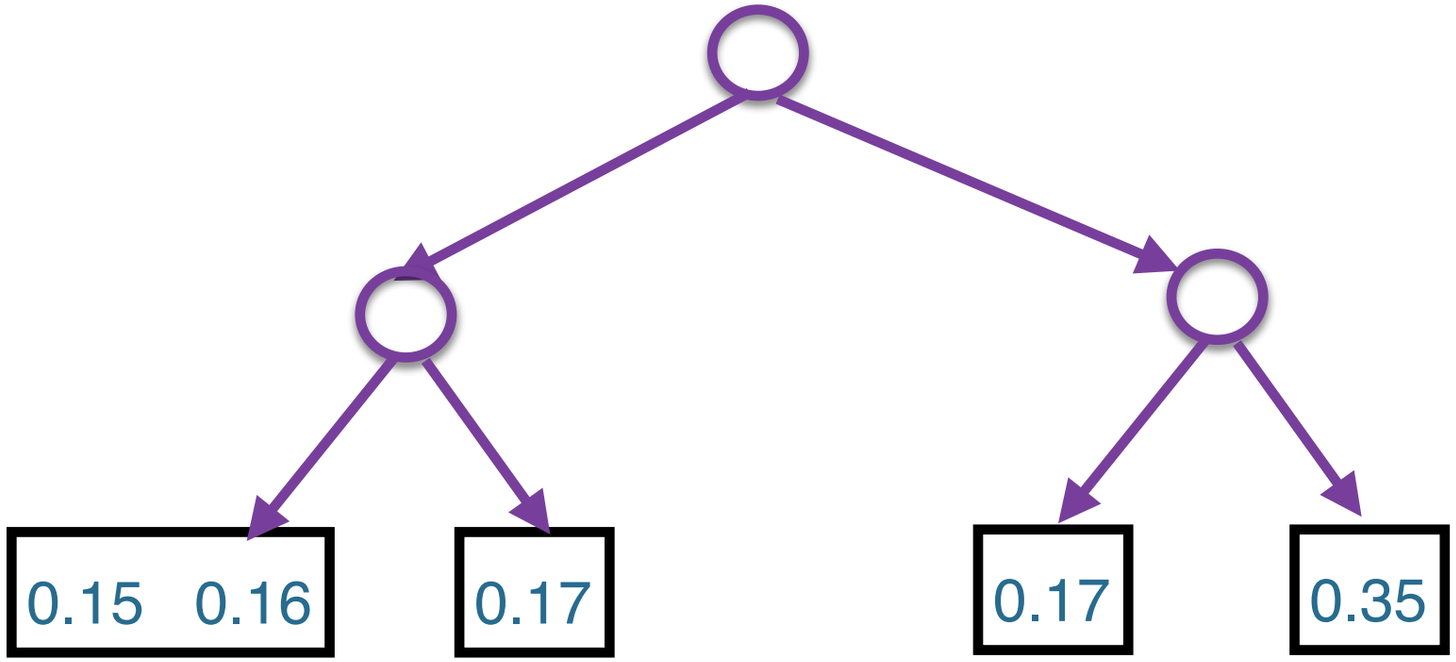


Shannon-Fano Coding (specifically Fano's)

Shannon-Fano coding refers to a couple different methods by Shannon and Fano.

- Both trying to find good prefix codes.
- Names commingled by historical accident.
- Presenting Fano's one now.

Given sorted list of probabilities, repeatedly split in half as evenly as possible.

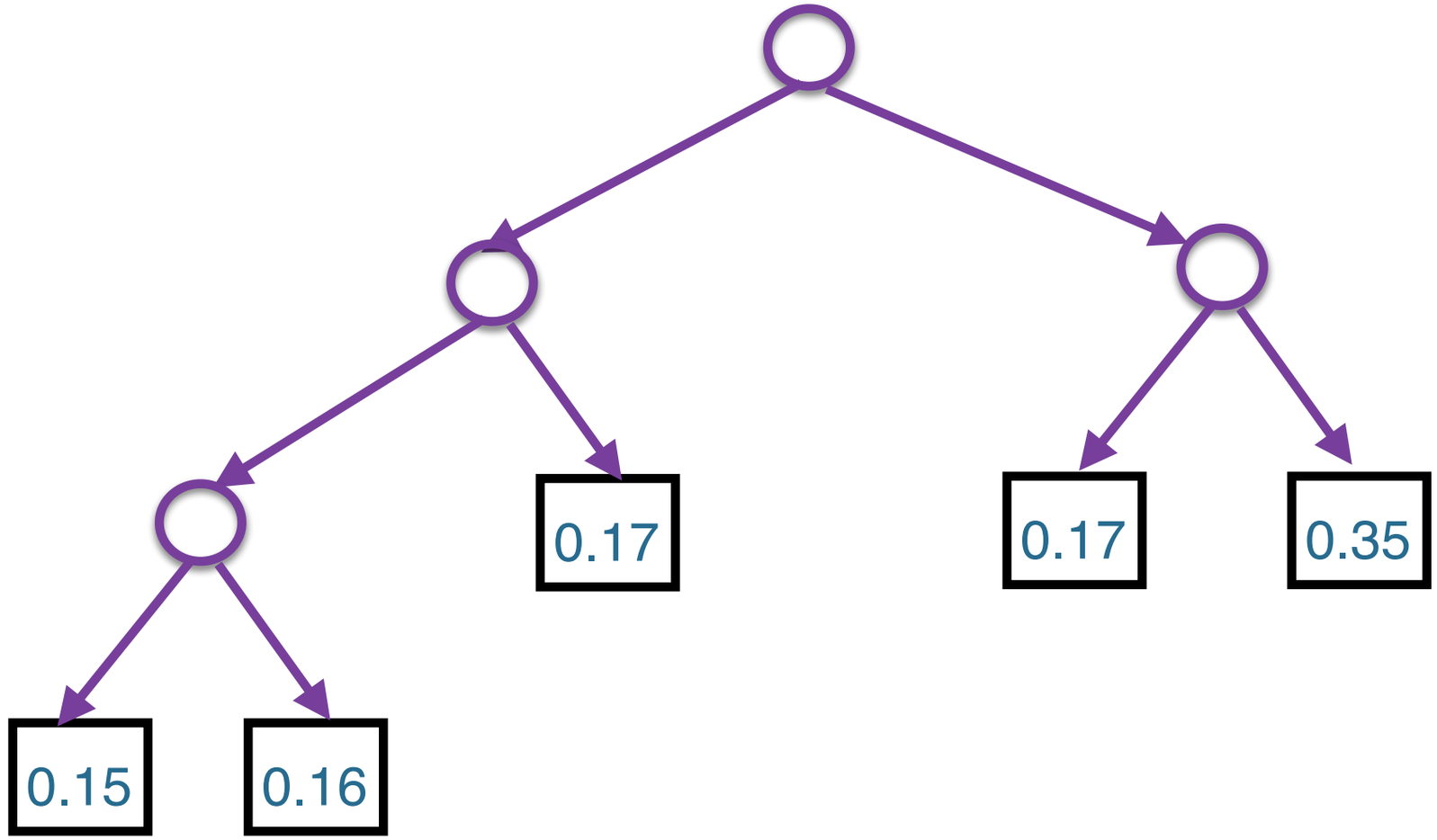


Shannon-Fano Coding (specifically Fano's)

Shannon-Fano coding refers to a couple different methods by Shannon and Fano.

- Both trying to find good prefix codes.
- Names commingled by historical accident.
- Presenting Fano's 1949 algorithm now.

Given sorted list of probabilities, repeatedly split in half as evenly as possible.



Average path length = 0.15*3+0.16*3+0.17*2+0.17*2+0.35*2 = 2.31

Huffman Coding

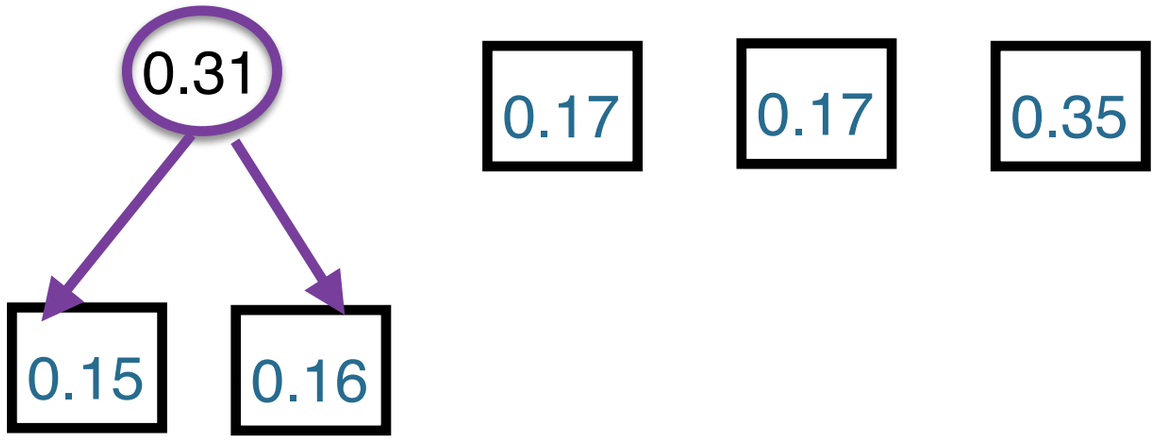
Huffman devised a greedy algorithm that he proved optimal for prefix codes.

Idea: repeatedly merge the lowest probability nodes.

Huffman Coding

Huffman devised a greedy algorithm that he proved optimal for prefix codes.

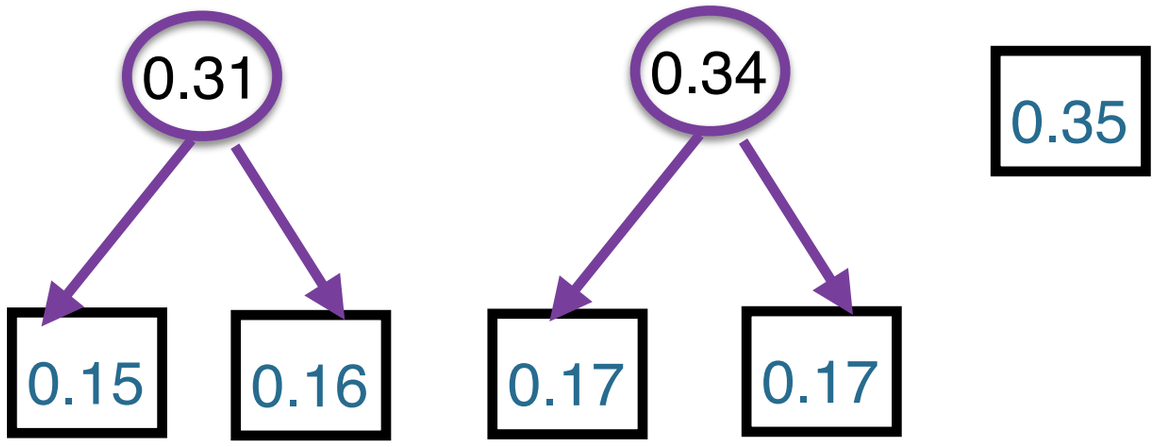
Idea: repeatedly merge the lowest probability nodes.



Huffman Coding

Huffman devised a greedy algorithm that he proved optimal for prefix codes.

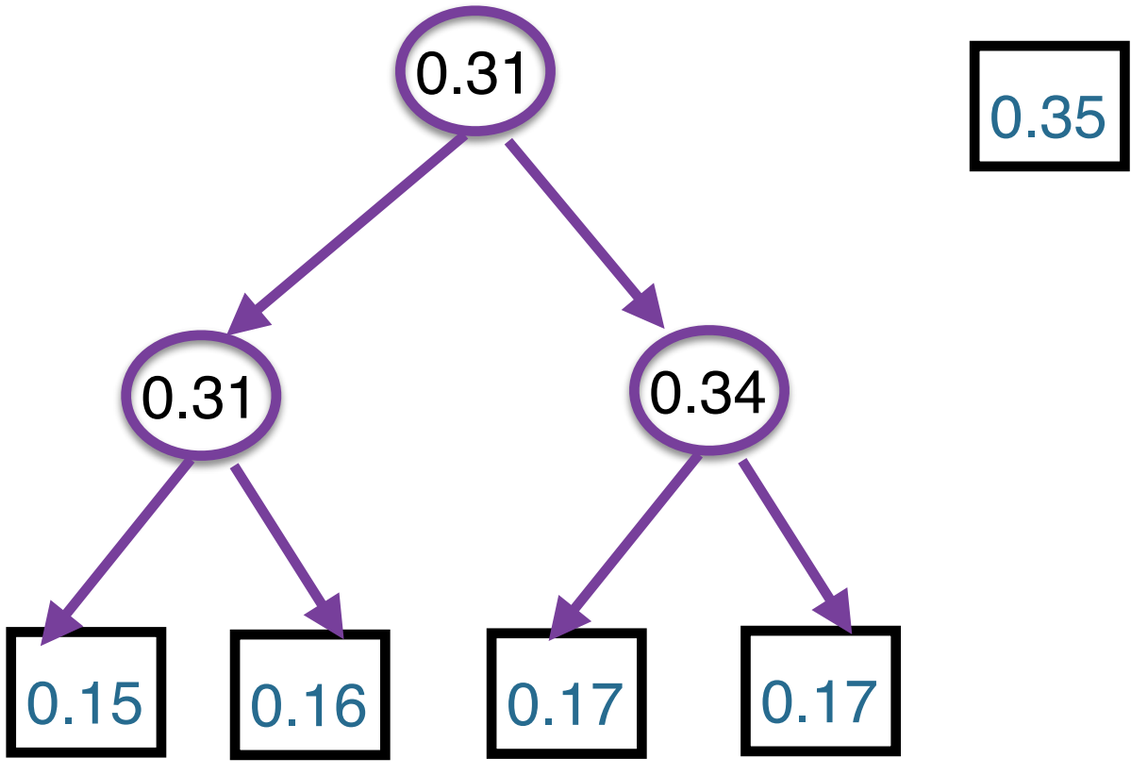
Idea: repeatedly merge the lowest probability nodes.



Huffman Coding

Huffman devised a greedy algorithm that he proved optimal for prefix codes.

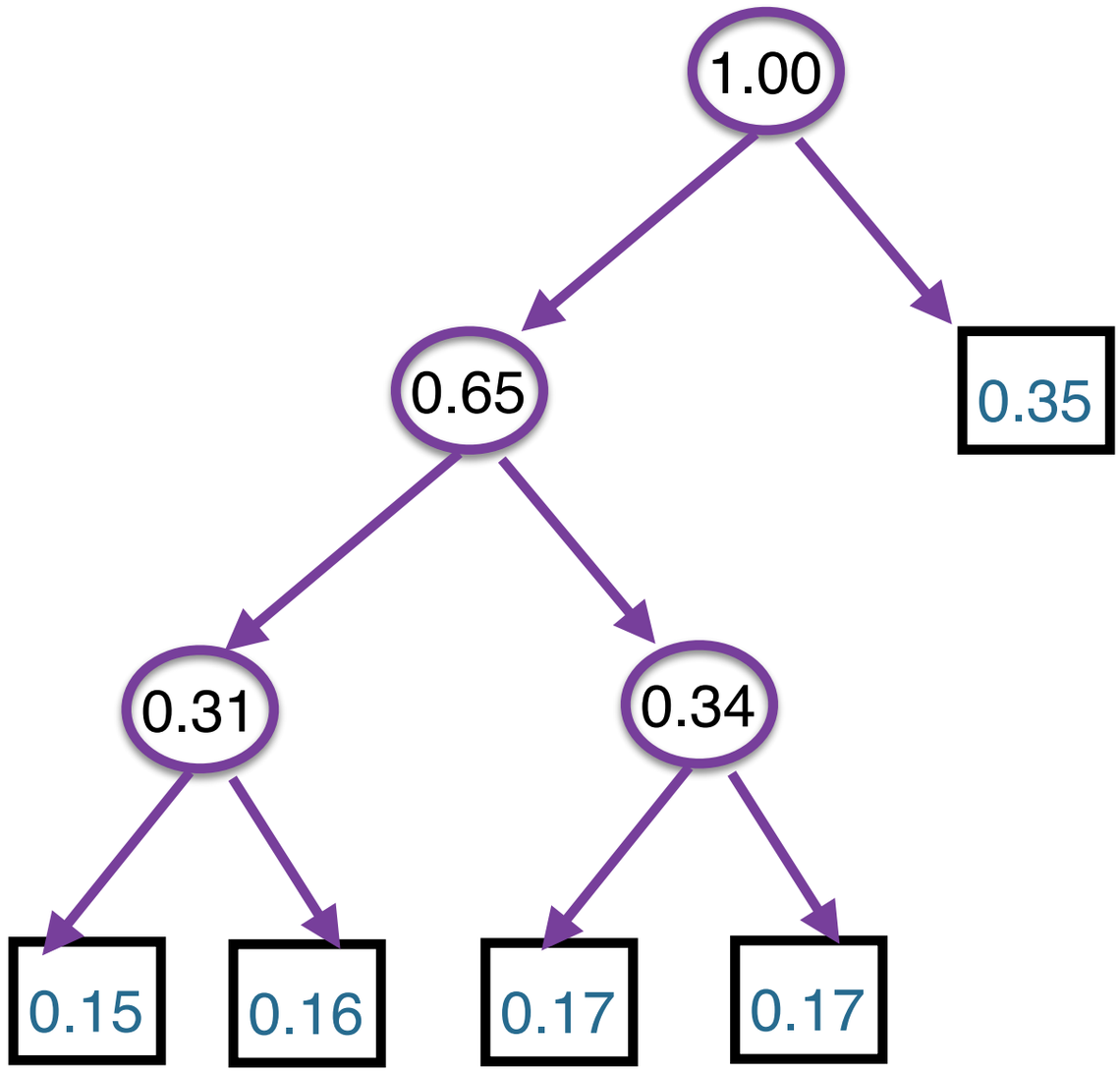
Idea: repeatedly merge the lowest probability nodes.



Huffman Coding

Huffman devised a greedy algorithm that he proved optimal for prefix codes.

Idea: repeatedly merge the lowest probability nodes.

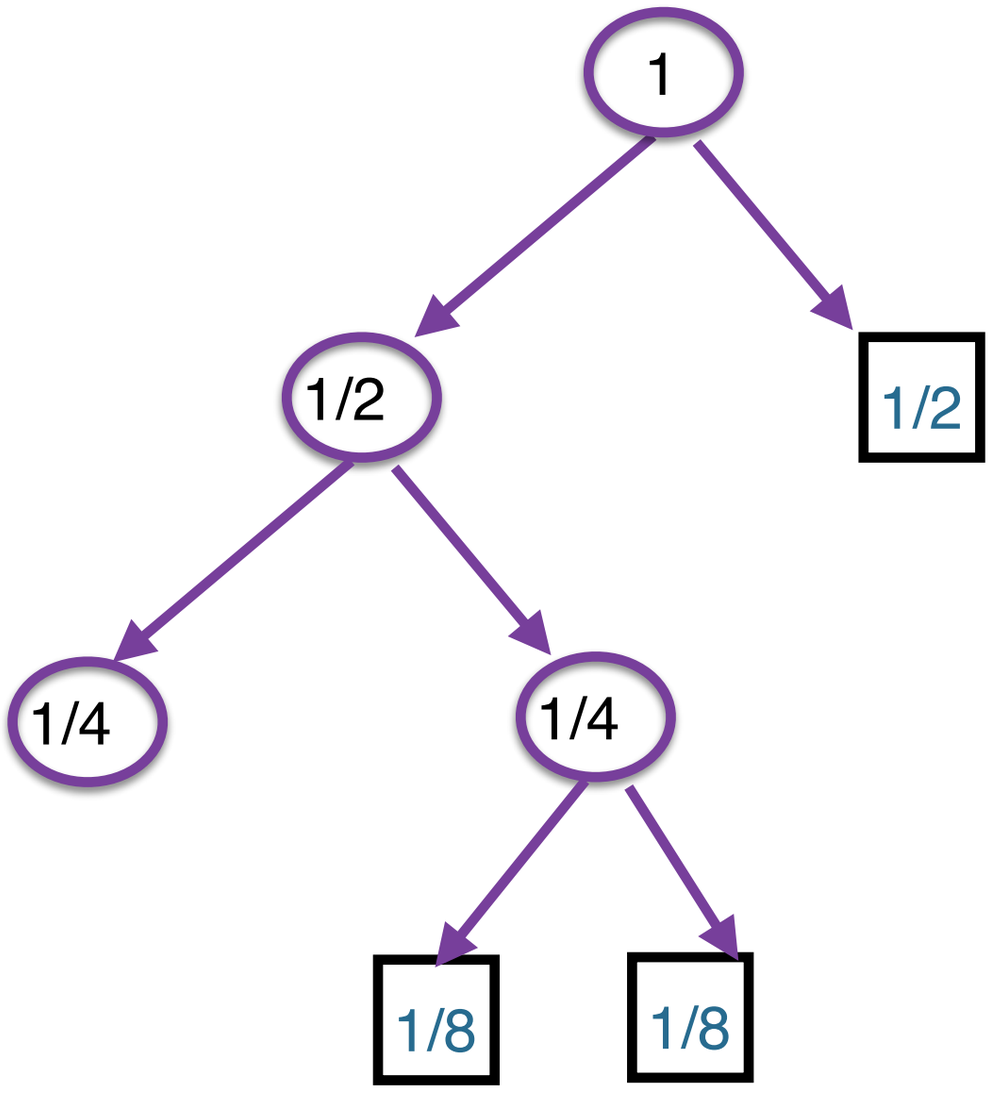


Average = 0.15*3+0.16*3+0.17*3+0.17*3+0.35*1 = 2.3 (vs Fano's 2.31)

Huffman Coding and Powers of 1/2

Observation:

- If all probabilities are powers of $1/2$,
  - Then values with probability $1/2^k$ are at depth k.



TopHat

Which of the following are true?

If all values have probabilities of the form $1/2^k$, then a Huffman code has average code length $\sum_{x \in X} 1/2^k \ k$ where 1/2^k is the probability of the current x.

1. If all values have probabilities of the form $1/2^k$, then a Huffman code has average code length $-\sum_{x \in X} p(x) \log_2 p(x)$.

2. If all values have probabilities of the form $1/2^k$, then a Huffman code always has the optimal average code length.

3. A Huffman code always has the optimal average code length.

Huffman Coding vs Binary Entropy

If all probabilities are powers of 1/2, then Huffman Coding matches the entropy bound.

But most distributions are not just powers of 1/2.

Optimality of Huffman Coding

Claim: For any probability distribution of at least two values, there always exists an optimal prefix code where the two lowest probability values are at the bottom level of the tree.

Optimality of Huffman Coding

Claim: For any probability distribution of at least two values, there always exists an optimal prefix code where the two lowest probability values are at the bottom level of the tree.

Proof:

- There will always be at least two leaf nodes at the bottom of the tree.
- If one of the two lowest probability values is not at the bottom level,
  - There must be at least one value with higher probability at the bottom level.
  - Swap those values to improve the average code length.

Optimality of Huffman Coding

Claim: For any probability distribution of at least two values, there always exists an optimal prefix code where the two lowest probability values are **siblings** at the bottom level of the tree.

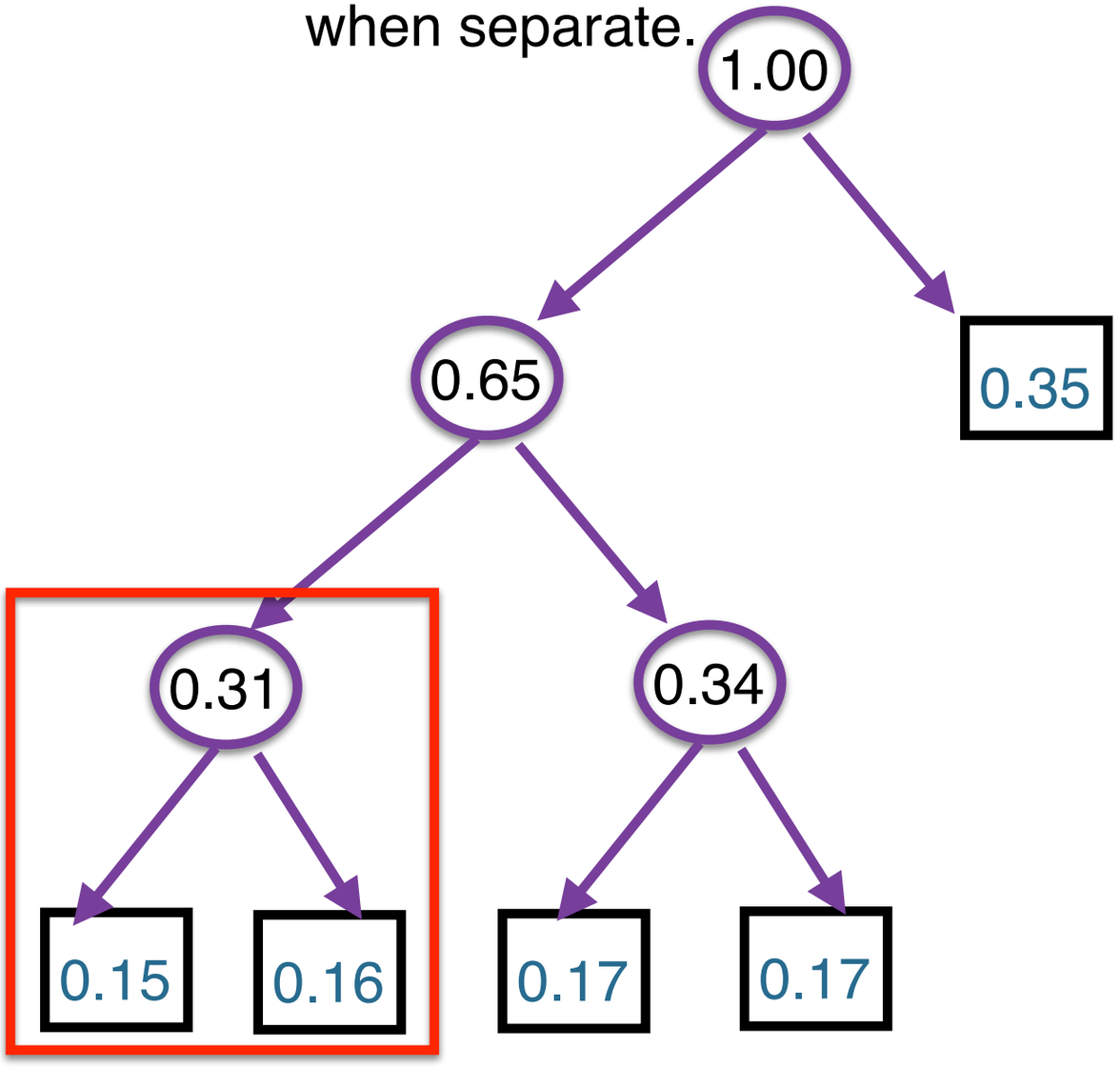
Proof:

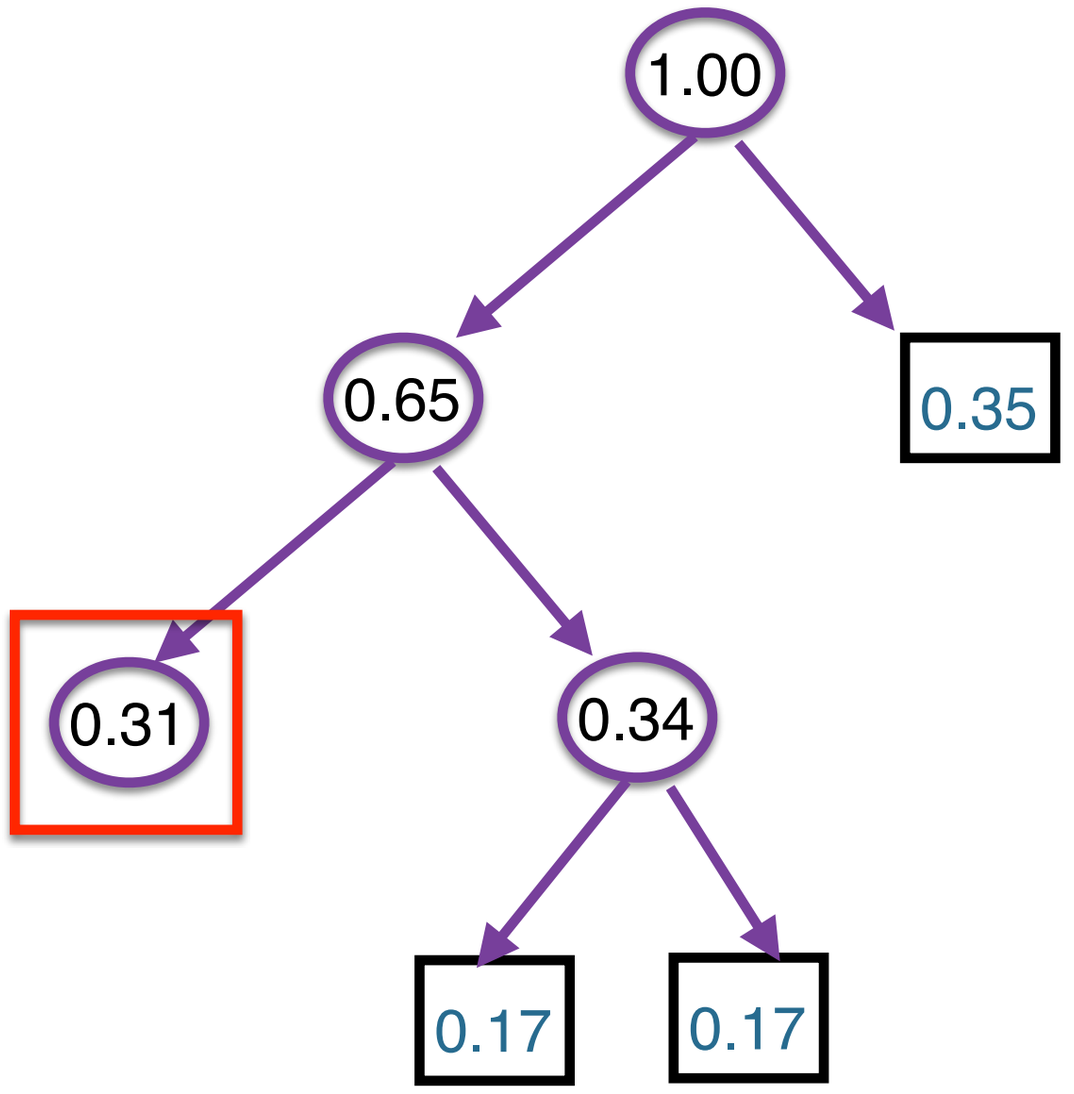
- Same as before, plus…
- There must exist sibling leaves at the bottom level of the tree.
- Swapping values at the same level does not change the average code length.
- So, pick a pair of sibling leaves and swap the two lowest probability values there.

Optimality of Huffman Coding

Sketch of the remainder:

- Since we know that there is an optimal prefix code with the two lowest probability values (p_1 and p_2), we can "stick them together" and treat them as one combined value.
  - Average code length with combined values will be a fixed $p_1 + p_2$ lower than when separate.





Optimality of Huffman Coding

Theorem: Huffman codes are always optimal prefix codes.

Optimality of Huffman Coding

Theorem: Huffman codes are always optimal prefix codes.

Theorem: Huffman codes are always optimal codes if each value is represented by a whole number of bits.

Optimality of Huffman Coding

Theorem: Huffman codes are always optimal prefix codes.

Theorem: Huffman codes are always optimal codes if each value is represented by a whole number of bits.

Implicit assumption from Shannon's Source Coding Theorem:

- The values being encoded here are independent and identically distributed.
- Correlations between values will affect compression rates.

Construction Time for Huffman Codes

How long does it take to construct a Huffman code?

Construction Time for Huffman Codes

How long does it take to construct a Huffman code?

- $O(n \log n)$ using a priority queue to pick the lowest weight nodes.

Construction Time for Huffman Codes

How long does it take to construct a Huffman code?

- $O(n \log n)$ using a priority queue to pick the lowest weight nodes.
- $O(n)$ if the values are pre-sorted.

Construction Time for Huffman Codes

How long does it take to construct a Huffman code?

- $O(n \log n)$ using a priority queue to pick the lowest weight nodes.

- $O(n)$ if the values are pre-sorted.
  - Two queues. One for sorted leaf nodes. One for merged nodes.
  - Merged node queue always gets inserts in sorted order.
  - So lowest weight node is always at beginning of one of them.

Construction Time for Huffman Codes

Q1:

Q2:

Construction Time for Huffman Codes

Q1: 0.17 0.17 0.35

Q2: 0.31=0.15+0.16

Construction Time for Huffman Codes

Q1: 0.35

Q2: 0.31=0.15+0.16 0.34=0.17+0.17

Construction Time for Huffman Codes

Q1: 0.35

Q2: 0.65=(0.15+0.16)+(0.17+0.17)

Construction Time for Huffman Codes

Q1:

Q2: 1.0=0.35+((0.15+0.16)+(0.17+0.17))

Encoding Speed for Huffman Codes

How fast is encoding each value?

Encoding Speed for Huffman Codes

How fast is encoding each value?

- Dictionary lookup.
- Write encoding.

Encoding Speed for Huffman Codes

How fast is encoding each value?

- Dictionary lookup - $O(1)$?
- Write encoding - $O(\text{encoded length})$

Encoding Speed for Huffman Codes

How fast is encoding each value?

- Dictionary lookup - $O(1)$?
- Write encoding - $O(\text{encoded length})$

Total time: $O(\text{total encode length})$

Arithmetic Encoding

Huffman codes are always optimal codes if each value is represented by a whole number of bits.

Next time we will talk about arithmetic encoding which allows us to use fractional bits to more generally match the entropy bound…