Fixed-to-Variable Source Coding

I-Hsiang Wang

Department of Electrical Engineering
National Taiwan University

ihwang@ntu.edu.tw

December 5, 2024

Redundancy in random data source

Recall: compression is possible since there is redundancy in the source.

One of the simplest ways to capture redundancy is to model the data source as a random process.

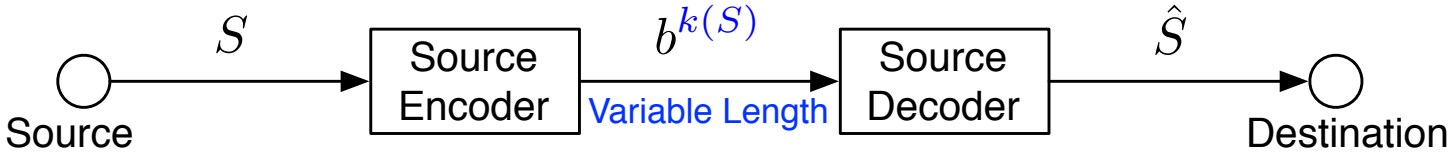
Redundancy comes from the fact that *different symbols in $\mathcal{S}$ take different probabilities to be drawn*.

Recall: with a random source model, there are two approaches one can take to demonstrate data compression:

- Allow variable codeword length for different symbols with different probabilities, rather than fixing it to be k.
- Allow (almost) lossless reconstruction rather than exact recovery.

Focus of this lecture: variable codeword length with exact reconstruction.

Fixed-to-variable source coding



Allow k to depend on the realization of the source, s^n.

Using variable codeword length is intuitive: for symbols with higher probability, we tend to use shorter codewords to represent it.

Note: the length n is not so important because concentration of probability is no longer required to achieve data compression.

It seems we can view the source data as a single random variable $S \sim \mathrm{P}_S$ taking values in a *countable* alphabet $\mathcal{S}$ and ask for the minimum possible expected length of the representation, $\mathrm{E}[k(S)]$.

Representing a single random symbol

Formulation:

1. An encoding function $\mathrm{enc} : \mathcal{S} \to \{0,1\}^*$, where $\{0,1\}^*$ denotes the collection of all finite-length bit sequences:

$$\{0,1\}^* := \{\emptyset, (0), (1), (0,0), (0,1), (1,0), (1,1), ...\}.$$

   Note: it is allowed to use "nothing" $\emptyset$ to represent a symbol.

2. A decoding function $\mathrm{dec} : \{0,1\}^* \to \mathcal{S}$. Under the exact recovery criterion, $\mathrm{dec}(\mathrm{enc}(s)) = s,\ \forall\, s \in \mathcal{S}$, equivalent to enc being *injective*.

   Note: given an injective encoder, decoder is straightforward. Hence, in this context, a source *code* usually refers to its encoding function.

3. The codeword length of a symbol under encoder enc is denoted by $k_{\mathrm{enc}}(s) :=$ length of $\mathrm{enc}(s)$. Usually, one aims to minimize the expected codeword length of the random source:

$$k^* := \min_{\mathrm{enc}:\mathcal{S}\to\{0,1\}^*} \mathsf{E}[k_{\mathrm{enc}}(S)].$$

Optimal representation

For representing a single symbol, it turns out we can easily construct an optimal source code. The idea is quite straightforward.

- Order letters in the alphabet $\mathcal{S}$ in descending probability:

$$\mathcal{S} = \{a_i \,|\, i \in \mathbb{N}\},\ \mathsf{P}_S(a_1) \geq \mathsf{P}_S(a_2) \geq ....$$

- Start from $i = 1$, assign bit sequence in ascending length:

$$\mathrm{enc}^*(a_1) = \emptyset,\ \mathrm{enc}^*(a_2) = (0),\ \mathrm{enc}^*(a_3) = (1), \mathrm{enc}^*(a_4) = (0,0), ...$$

  Intuition: the more likely a letter is, the shorter its codeword should be.

The codeword lengths of this code:

$$l_i^* \equiv k_{\mathrm{enc}^*}(a_i) = \lfloor \log_2 i \rfloor\ ,\ \forall\, i \in \mathbb{N}.$$

Note: the codeword length $L := k_{\mathrm{enc}}(S)$ can be viewed as a random variable.

Proposition 1 (Optimality)

The codeword length of enc^*, $L^* = k_{\mathrm{enc}^*}(S)$, *is stochastically dominated by the codeword length* $L := k_{\mathrm{enc}}(S)$ *of any injective* enc, *that is,*

$$\Pr\{L \le t\} \le \Pr\{L^* \le t\} \quad \forall\, t \in \mathbb{R}.$$

Note: stochastic dominance implies $\mathsf{E}[L^*] \le \mathsf{E}[L]$. This is left as exercise.

pf: Let's count the number of letters with codeword lengths $\le l,\ l = 0, 1, \ldots$:

$$|\{i \in \mathbb{N} \,|\, l_i \equiv k_{\mathrm{enc}}(a_i) \le l\}| \le \underbrace{2^0 + 2^1 + \ldots + 2^l}_{\text{\# of bit seq. with length} \le l} = |\{i \in \mathbb{N} \,|\, l_i^* \le l\}|\,,$$

where the last equality holds by our construction.

Since the probabilities of letters $a_1, a_2, \ldots$ are decreasing, the proof is complete by our construction. □

Minimum expected codeword length

The minimum expected codeword length can be computed as follows:

$$k^* = \mathsf{E}[L^*] = [\mathsf{P}_S(a_2) + \mathsf{P}_S(a_3)] + 2\,[\mathsf{P}_S(a_4) + \ldots + \mathsf{P}_S(a_7)] + \ldots.$$
$$= \sum_{l=1}^{\infty} \Pr\left\{I \geq 2^l\right\}, \quad \text{where we denote } S \equiv a_I.$$

The following shows that k^* is close to the entropy of the source, $\mathrm{H}(S)$.

Theorem 1 (Bounds on the Minimum Expected Codeword Length)

$$\mathrm{H}(S) - \log_2(e(\mathrm{H}(S)+1)) \leq k^* \leq \mathrm{H}(S).$$

Remark: suppose $S \equiv S^n$ consists of n i.i.d. symbols $S_i \overset{\text{i.i.d.}}{\sim} \mathsf{P}_S$. The above theorem tells us that the optimum compression rate $\mathrm{R}^* \equiv \frac{k^*}{n}$ satisfies

$$\mathrm{H}(\mathsf{P}_S) - \tfrac{\log n}{n} + O(\tfrac{1}{n}) \leq \mathrm{R}^* \leq \mathrm{H}(\mathsf{P}_S).$$

pf: For the upper bound, note that $k^* = \mathsf{E}[\lfloor \log_2 I \rfloor] \le \mathsf{E}[\log_2 I]$, where we identify $S \equiv a_I$. It remains to show that for $i = 1, 2, ...,$

$$\log_2 i \le \log_2 \frac{1}{\mathsf{P}_S(a_i)}, \text{ that is, } \mathsf{P}_S(a_i) \le \frac{1}{i},$$

which is true since the letters are labeled in descending order of probability.

For the lower bound, note that

$$\mathrm{H}(S) = \mathrm{H}(I) = \mathrm{H}(I, L^*) = \mathrm{H}(I|L^*) + \mathrm{H}(L^*)\,.$$

For the first term, note that $\mathrm{H}(I|L^* = l) \le l$ since given $L^* = l$, there are at most 2^l possible a_i's with that codeword length. Hence,

$$\mathrm{H}(I|L^*) \le \mathsf{E}[L^*] \equiv k^*.$$

For the second term, we use the following bound that relates the entropy of a random variable taking values in non-negative integers to its expected value:

$$\mathrm{H}(L^*) \leq \log_2(k^*+1) + k^* \log_2\left(1+\tfrac{1}{k^*}\right).$$

Since $k^* \leq \mathrm{H}(S)$ (by the upper bound) and $(1+\frac{1}{x})^x \leq e \;\forall\, x > 0$, we have

$$\mathrm{H}(L^*) \leq \log_2\left(e(k^*+1)\right) \leq \log_2\left(e(\mathrm{H}(S)+1)\right).$$

Combine the first and the second term, we reach

$$\mathrm{H}(S) \leq k^* + \log_2\left(e(\mathrm{H}(S)+1)\right).$$

The proof is complete by rearranging terms. □

Represent a sequence with a symbol-wise encoder

In practice, it is rare that one can do encoding *just for one symbol*.

Usually there is a sequence of symbols to compress and the length of the sequence may not be known *a priori*.

Hence, the single-symbol fixed-to-variable code developed above is usually applied repeatedly on the sequence of symbols.

Definition 1 (Extension of a code)

The (symbol-by-symbol) *extension* of a code $\mathrm{enc} : \mathcal{S} \to \{0,1\}^*$ is defined as

$$\mathrm{enc}^+ : \mathcal{S}^+ \to \{0,1\}^*,\ (s_1, ..., s_n) \mapsto (\mathrm{enc}(s_1), ..., \mathrm{enc}(s_n)),$$

$\forall\, n \in \mathbb{N}$ and $(s_1, ..., s_n) \in \mathcal{S}^+$, where $\mathcal{S}^+ := \bigcup_{n \geq 1} \mathcal{S}^n$.

But the decoder does not know where to separate the symbols.

Additional structural requirements are needed.

Definition 2 (Uniquely Decodable Codes)

A code is *uniquely decodable* if its *extension* is injective.

Note: a uniquely decodable code cannot use $\emptyset$ as its output.

We are going to show that for all uniquely decodable codes, its codeword lengths satisfy the following Kraft-McMilan inequality:

$$\sum_{s\in\mathcal{S}} 2^{-k_{\mathrm{enc}}(s)} \leq 1.$$

As a result, we can show that the optimum expected length (over all possible uniquely decodable codes) is close to the entropy of the random source:

$$\mathrm{H}(S) \leq k^*_{\text{u.d.}} \leq \mathrm{H}(S) + 1.$$

An optimal code with low implementation complexity, Huffman code, will also be introduced, with proof of its optimality.

Prefix-free implies uniquely decodable

Let us begin with a special kind of codes that are uniquely decodable and further enjoy the advantage that the decoder is able to recognize the end of a codeword immediately before the first bit of the next codeword comes in.

Definition 3 (Prefix-Free Codes)

A code is *prefix-free* if no codeword is the prefix of another codeword.

Example 1

For $\mathcal{S} = \{a_1, a_2, a_3, a_4\}$, consider the following code:

$$a_1 \mapsto 0,\ a_2 \mapsto 10,\ a_3 \mapsto 110,\ a_4 \mapsto 111.$$

1. Show that the code is prefix-free.
2. Decode the sequence 0110111100110.

Example 2 (Different kinds of codes)

For $\mathcal{S} = \{a_1, a_2, a_3, a_4\}$, consider the following codes:

Letter	Code 1	Code 2	Code 3	Code 4
a_1	0	10	0	0
a_2	010	00	10	01
a_3	01	11	110	011
a_4	10	110	111	111

Verify the following:

- Code 1 is injective but not uniquely decodable.
- Code 2 is uniquely decodable but not prefix-free.
- Code 3 is prefix-free.
- Code 4 is *suffix-free* and hence uniquely decodable.

Codeword lengths of a prefix-free code

The following theorem tells us about the necessary and sufficient condition on the codeword lengths of a prefix-free code.

Proposition 2 (Kraft Inequality)

For a prefix-free code

$$\mathrm{enc} : \mathcal{S} \equiv \{a_i \,|\, i \in \mathbb{N}\} \to \{0,1\}^*,$$

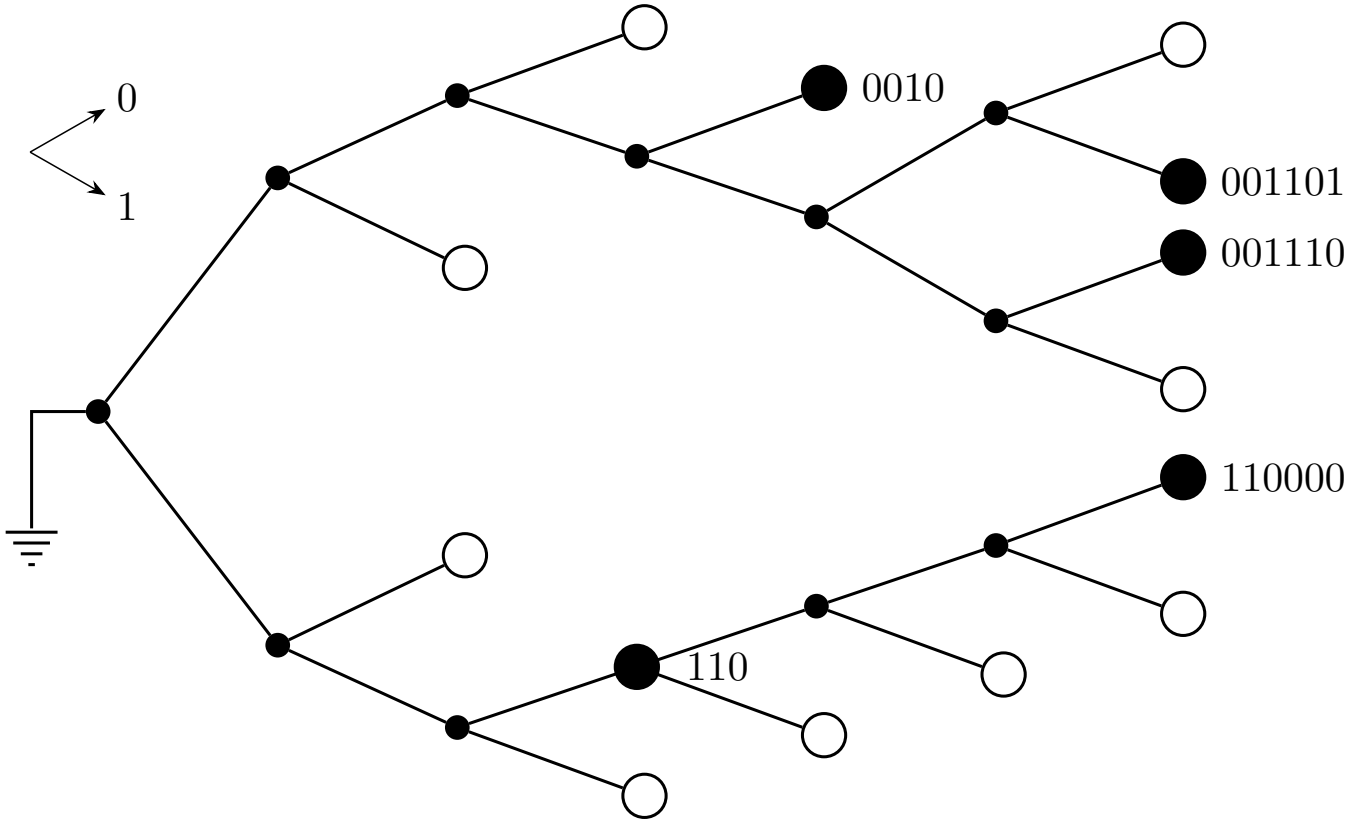
its codeword lengths $\{l_i \equiv k_{\mathrm{enc}}(a_i) \,|\, i \in \mathbb{N}\}$ *satisfy the following inequality:*

$$\sum_{i=1}^{\infty} 2^{-l_i} \leq 1. \tag{1}$$

Conversely, if a set of lengths $\{l_i \,|\, i \in \mathbb{N}\}$ *satisfy the Kraft Inequality* (1)*, there exists a prefix-free code with codeword lengths* $k_{\mathrm{enc}}(a_i) = l_i$*,* $\forall\, i \in \mathbb{N}$*.*

Code as a tree

To intuitively understand Kraft Inequality, let us adopt a useful data structure, *binary tree*, to represent a code.



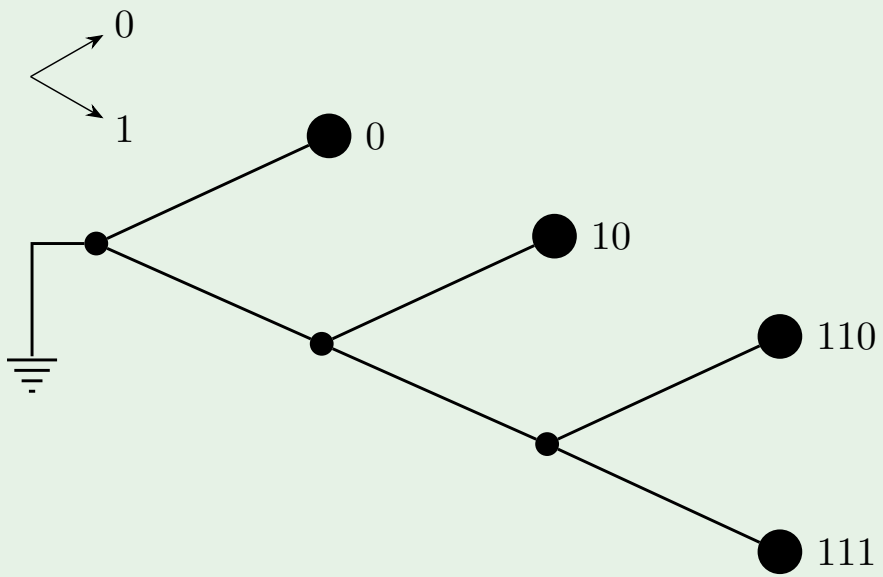
Stefan M. Moser, *Information Theory*, Version 6.5.

Fact 1

A code is prefix-free if and only if in its binary tree, all codewords are leaves.

Example 3

The binary tree of the prefix-free code in Example 1 is shown below.



Stefan M. Moser, *Information Theory*, Version 6.5.

Let $l_{\max}$ denote the largest *depth* (# of hops to the root) of the leaves in a tree.

In this last level, there are $2^{l_{\max}}$ leaves in an extended full tree. Now, trace each leaf in the last level back to a leaf in the original tree.

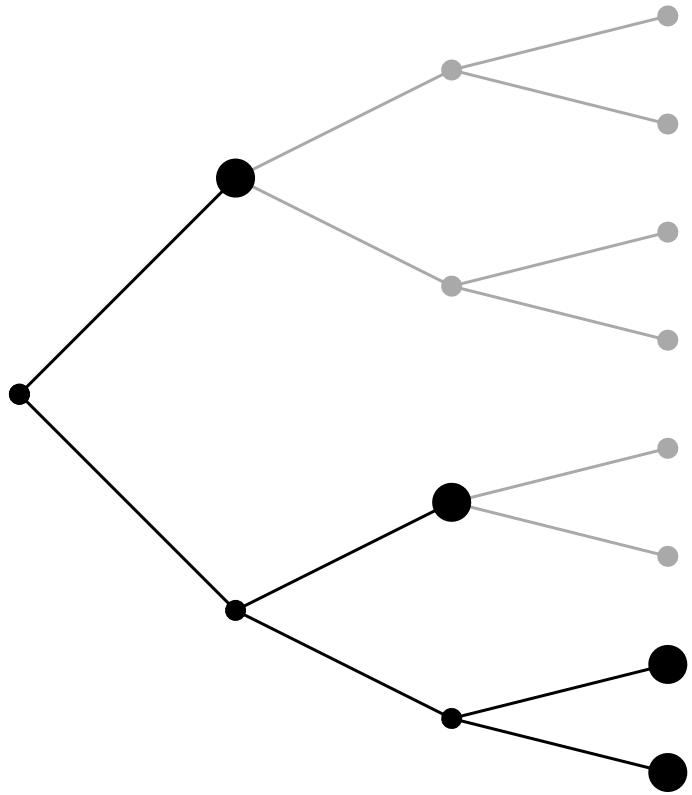
This node i with depth l_i has $2^{l_{\max}-l_i}$ leaf descendants in the extended tree.

By counting, we get

$$\sum_i 2^{l_{\max}-l_i} = 2^{l_{\max}} \implies \sum_i 2^{-l_i} = 1.$$

Since a prefix-free code may not use all leaves in the original tree, the above "$=$" is replaced by "$\leq$" for a prefix-free code.

This is Kraft Inequality, the forward direction of Proposition 2 for prefix-free codes with finite length.



For the converse direction of Proposition 2, let us re-order the lengths in ascending order: $1 \le l_1 \le l_2 \le .. \le l_m \equiv l_{\max}$.

Construct a full tree with l_m levels.

Step 1:

- Start with the top branch and locate the top node at level l_1.
- Put the first codeword there and eliminate it and all its descendants.

Step i $(2 \le i \le m)$: In the remaining tree:

- Move to the top branch at level l_{i-1} and locate the top node at level l_i.
- Put the i-th codeword there and eliminate it and all its descendants.

We leave it as an exercise for you to check that Step i $(2 \le i \le m)$ is always feasible if $l_1, ..., l_m$ satisfy Kraft Inequality.

Exercise 1

Show that if Step i $(2 \le i \le m)$ cannot be done, that is, there is no node at level l_{i-1} in the remaining tree, Kraft Inequality is violated, leading to contradiction.

Proof of Kraft Inequality

Here we give a formal proof of Proposition 2.

pf (Proposition 2): For the forward direction, we use a *probabilistic* argument. The idea is to show that the LHS of (1) is the probability of some event.

Given a code, we have a codebook (collection of all codewords). Consider a process of generating i.i.d. $\mathrm{Ber}(1/2)$ bits. It stops if the generated sequence falls into the codebook, and continues otherwise.

Due to the prefix-free condition, the events of the generated sequence matching a particular codeword are *disjoint*. Hence, the probability of termination of this process is just

$$\sum_{i=1}^{\infty} 2^{-l_i} \leq 1.$$

For the converse direction, let us re-order the lengths in ascending order: $1 \le l_1 \le l_2 \le \ldots$. The previous idea of assigning codewords on a tree can be rewritten as follows:

Let $r_i := \sum_{k=1}^{i-1} 2^{-l_k}$, for $i \ge 2$, and $r_1 = 0$. Note that $r_i \le 1$ for $i = 1, 2, \ldots$ since the lengths satisfy Kraft Inequality.

We use the first l_i bits in r_i's binary expansion after the decimal point as the i-th codeword in the codebook.

The remaining thing is to check the prefix-free condition. This can be proved by contradiction: suppose for some $j > i$, the i-th codeword is a prefix of the j-th codeword.

Then, since r_i and r_j agree on the first l_i bits in their binary expansions,

$$r_j - r_i < 2^{-l_i}.$$

By construction, $r_j - r_i = \sum_{k=i}^{j-1} 2^{-l_k} \ge 2^{-l_i}$, leading to contradiction. □

Codeword lengths of a uniquely decodable code

McMilan enhanced Kraft's inequality and showed that all uniquely decodable codes must satisfy (1) as well.

Theorem 2 (Kraft-McMilan Inequality)

For a uniquely decodable code $\mathrm{enc} : \mathcal{S} \equiv \{a_i \,|\, i \in \mathbb{N}\} \to \{0,1\}^*$*, its codeword lengths* $\{l_i \equiv k_{\mathrm{enc}}(a_i) \,|\, i \in \mathbb{N}\}$ *satisfy the following inequality:*

$$\sum_{i=1}^{\infty} 2^{-l_i} \leq 1. \tag{2}$$

Conversely, if a set of lengths $\{l_i \,|\, i \in \mathbb{N}\}$ *satisfy* (2)*, there exists a prefix-free (hence uniquely decodable) code with codeword lengths* $k_{\mathrm{enc}}(a_i) = l_i$*,* $\forall\, i \in \mathbb{N}$*.*

The proof involves generating functions, a tool in algebraic combinatorics. We skip it here. Check Chapter 5.5 of Cover and Thomas.

Remark: Hence, if the concern is only on the codeword lengths in designing uniquely decodable codes, there is no loss in restricting to prefix-free codes.

Since Kraft-McMilan Inequality completely characterizes the lengths of uniquely decodable codes, given a distribution of S, the random message to be compressed, there are two remaining questions to be answered:

1. What is the best set of codeword lengths, $\{l_1, l_2, ...\}$, that can minimize the expected codeword length? Given the probability of each letter, we need to specify the codeword length of each letter.
2. Once the codeword lengths are specified for all letters, is there a computationally simple algorithm that can come up with prefix-free codewords of those lengths?

It turns out that first question is not that easy to answer, since it is an integer programming (IP) problem, in general NP-complete (in $|\mathcal{S}|$).

We first give bounds on the optimum, $k^*_{\text{u.d.}}$, of this IP, showing that it is close to the entropy $\mathrm{H}(S)$ within 1 bit, that is, $\mathrm{H}(S) \le k^*_{\text{u.d.}} \le \mathrm{H}(S) + 1$.

Then, we give a nearly-linear-time (in $|\mathcal{S}|$) algorithm that solves this particular IP due to Huffman, also answering the above second question.

Bounds on the minimum expected codeword length

To find minimum expected codeword length of uniquely decodable codes, by Theorem 2, it suffices to solve the following optimization problem:

$$\begin{aligned} \min_{l_1,l_2,\ldots\in\mathbb{N}} \quad & \textstyle\sum_i p_i l_i \\ \text{subject to} \quad & \textstyle\sum_i 2^{-l_i} \le 1 \end{aligned} \tag{3}$$

Here we set $p_i \equiv \mathsf{P}_S(a_i)$, for $i \in \mathbb{N}$. Recall: $\mathcal{S} = \{a_i \,|\, i \in \mathbb{N}\}$.

This is unfortunately an integer programming (IP) problem due to the constraint that $l_1, l_2, \ldots$ must be integers.

We can try to *approximate* the value of $k^*_{\text{u.d.}}$, the result of the IP (3):

1. Relaxation: remove the integral constraint on $l_1, l_2, \ldots$ and solve (3) to get a lower bound on $k^*_{\text{u.d.}}$. Note: (3) is a convex program after relaxation.
2. Plug in a specific choice of $l_1, l_2, \ldots$ to get an upper bound on $k^*_{\text{u.d.}}$. This choice can be a rounding of the solution in the relaxed problem.

So, consider the relaxed problem

$$\begin{aligned} \min_{l_1, l_2, \ldots \in \mathbb{R}} \quad & \textstyle\sum_i p_i l_i \\ \text{subject to} \quad & \textstyle\sum_i 2^{-l_i} \leq 1 \end{aligned} \tag{4}$$

Check by yourself that this is a convex optimization problem. With the KKT condition, we can find the minimizer to this problem as follows (exercise):

$$l_i = -\log_2 p_i, \ i = 1, 2, \ldots \tag{5}$$

Plugging (5) back to (4), we get an lower bound on $k^*_{\text{u.d.}}$:

$$k^*_{\text{u.d.}} \geq \sum_i -p_i \log_2 p_i = \mathrm{H}(S) \, .$$

This relaxation *looks* informative: it says that the length of a codeword should be roughly equal to the logarithm of the inverse probability of the letter it corresponds to.

To get an upper bound on $k^*_{\text{u.d.}}$, let us round (5) as follows:

$$l_i = \lceil -\log_2 p_i \rceil \,, \; i = 1, 2, ... \tag{6}$$

Taking ceiling is because we do not want to violate the constraint $\sum_i 2^{-l_i} \leq 1$.

Plugging (6) back to (3), we get an upper bound on $k^*_{\text{u.d.}}$:

$$k^*_{\text{u.d.}} \leq \sum_i p_i \lceil -\log_2 p_i \rceil < \sum_i p_i (1 - \log_2 p_i) = \mathrm{H}(S) + 1.$$

The following theorem summarizes these bounds.

Theorem 3 (Bounds on the Minimum Expected Codeword Length)

$$\mathrm{H}(S) \leq k^*_{u.d.} < \mathrm{H}(S) + 1.$$

Exercise 2

Without using the relaxation idea, use Jensen's inequality to directly show that $\mathrm{H}(S) \leq \sum_i p_i l_i$ for any $l_1, l_2, ... \in \mathbb{R}$ satisfying $\sum_i 2^{-l_i} \leq 1$.

Towards an optimal uniquely decodable code

Finding an optimal (minimizing the expected codeword length) uniquely decodable code was left as an open problem in Shannon's 1948 paper.

Shannon tried to find prefix-free codes with codeword lengths satisfying (6). This kind of codes are called *Shannon-type codes*. Shannon himself gave the following simple procedure to construct such a code:

Shannon Code

1. Order the letters in descending order of probability.
2. Compute the partial sum of probability: $q_i = \sum_{j=1}^{i-1} p_j$ for $i \geq 2$ and $q_1 = 0$.
3. Use the first $l_i = \lceil -\log_2 p_i \rceil$ bits in q_i's binary expansion as the codeword for the i-th letter.

Exercise 3

Show that the above procedure gives a prefix-free code.

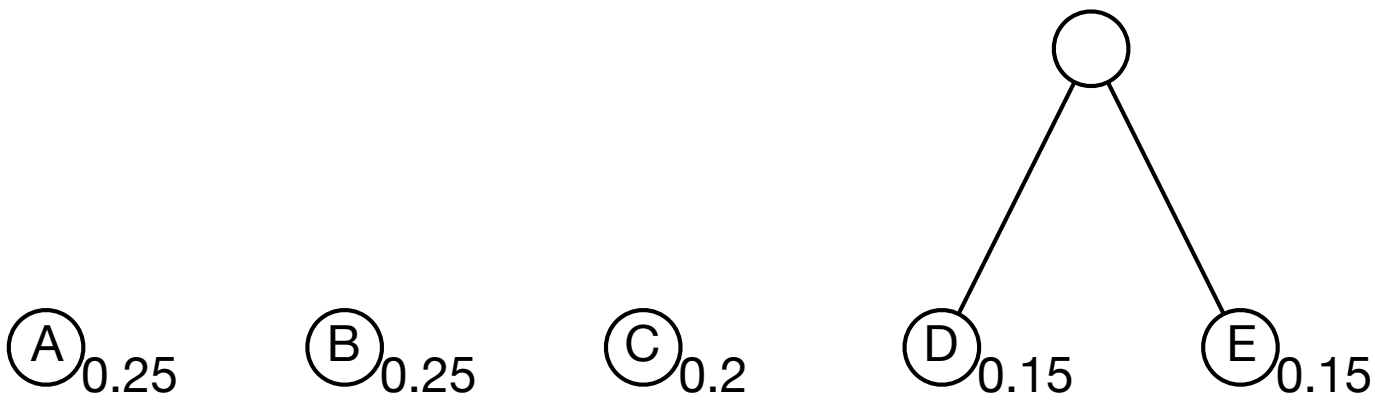
Huffman code

In 1951, D. Huffman attended R. Fano's course "Transmission of Information" at MIT. Fano gave the open problem to students as a topic of the term paper. Huffman solved this open problem beautifully for finite alphabets.

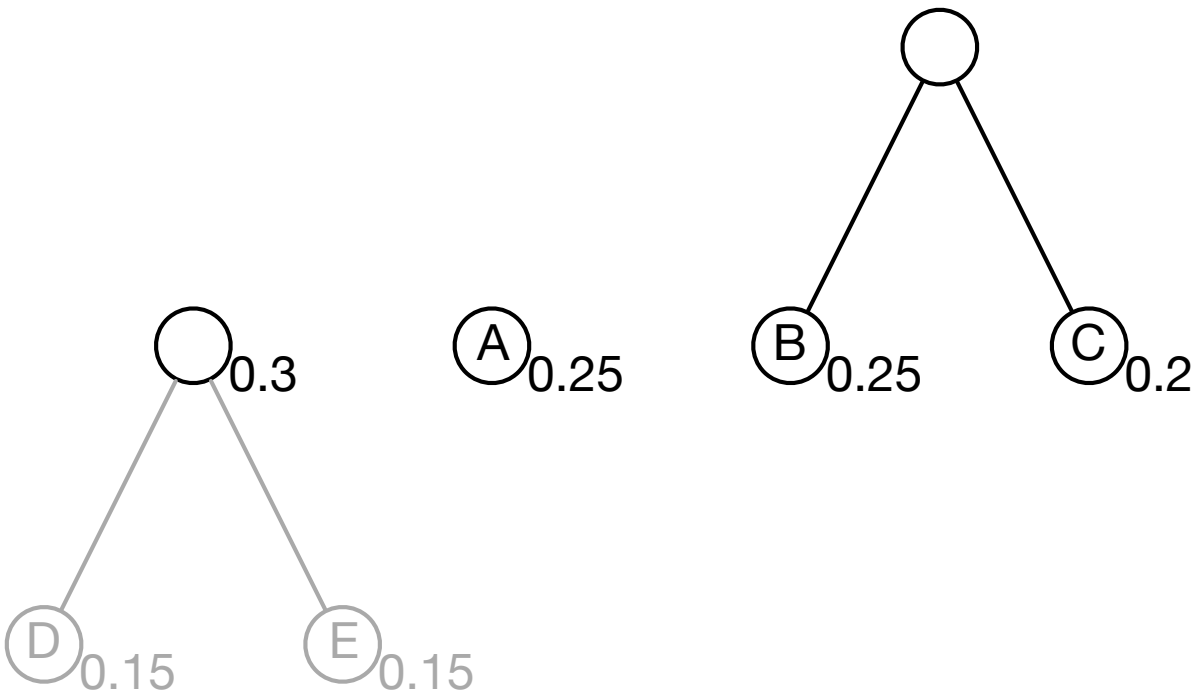
Huffman Code

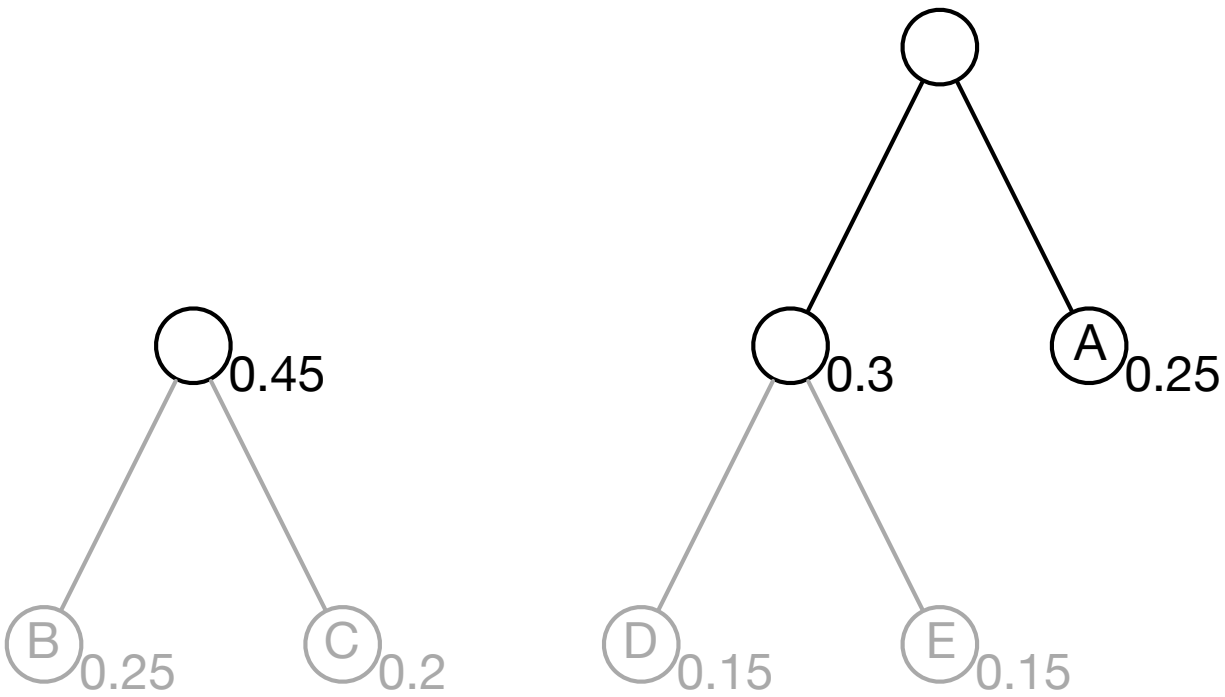
1. Construct an *ordered* list of $|\mathcal{S}|$ nodes, each of which corresponds to a letter. Associate the probability of each letter to its node.
2. Sort the list with descending probability of nodes.
3. Pick the two nodes with the least probabilities and connect them to a new node. Associate the sum of the two probabilities to this node.
4. Insert this new node into the list and remove the two nodes from the list.
5. Go back to Step 2 until there is only one node left in the list.

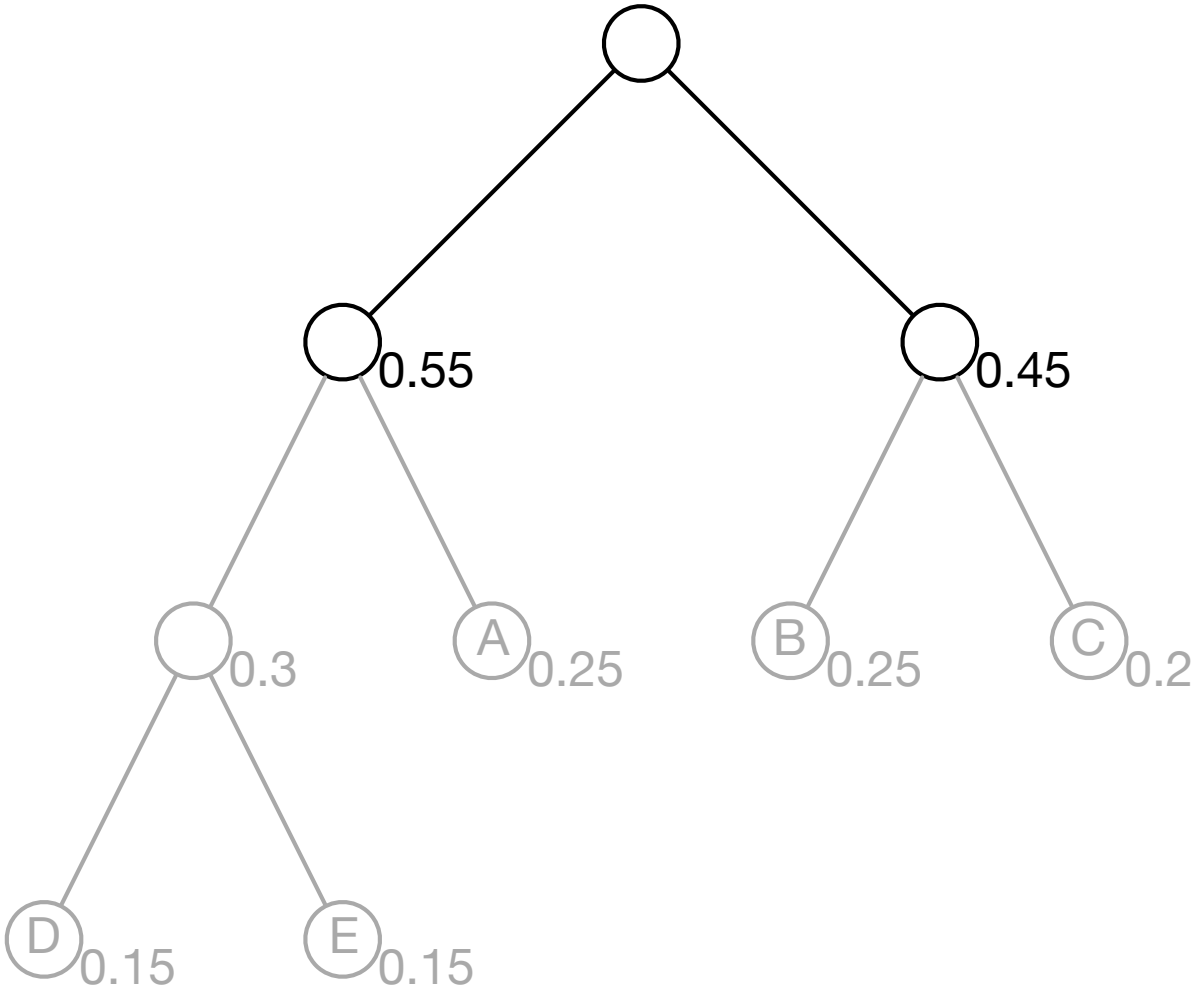
The constructed binary tree gives a prefix-free code.

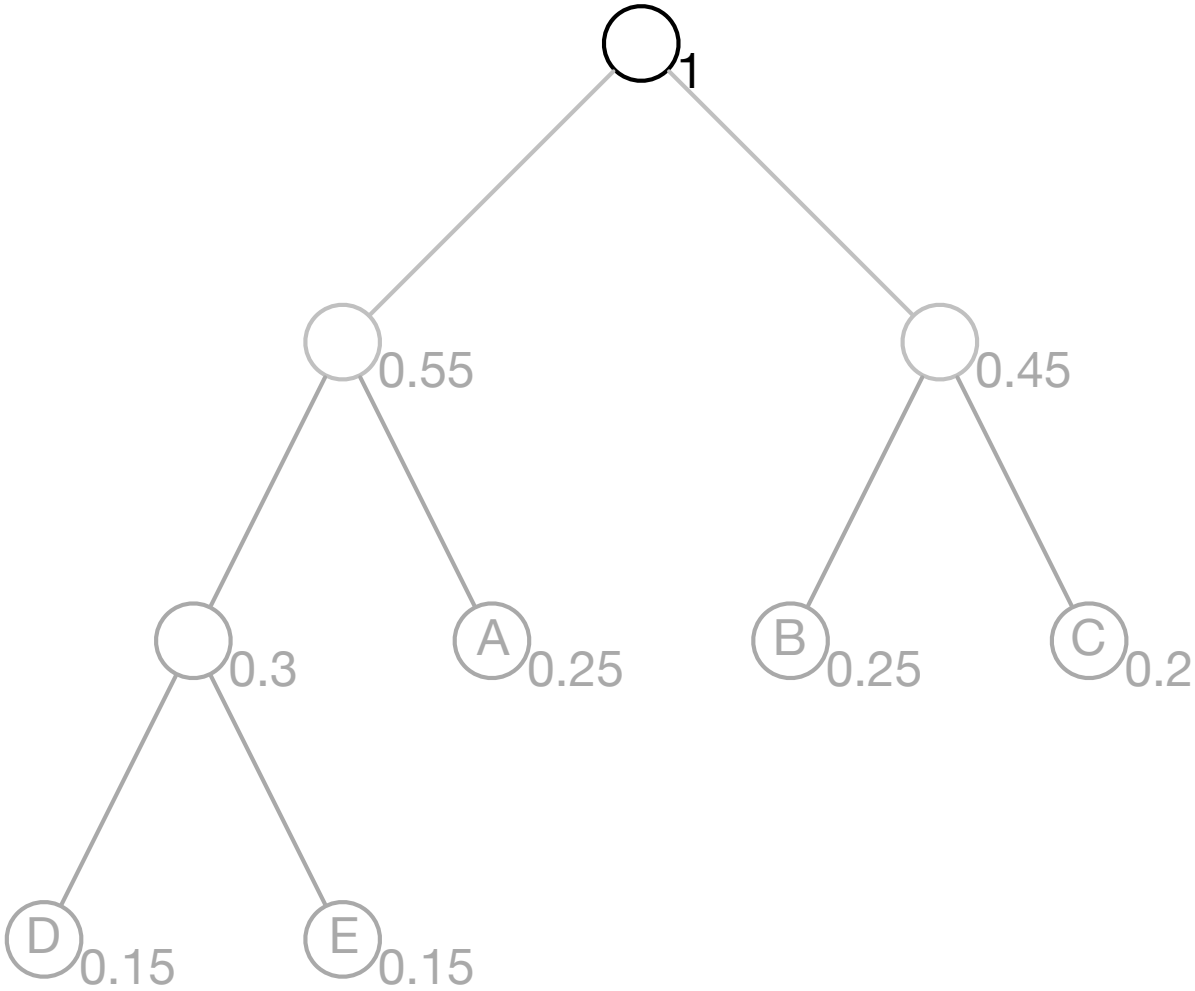


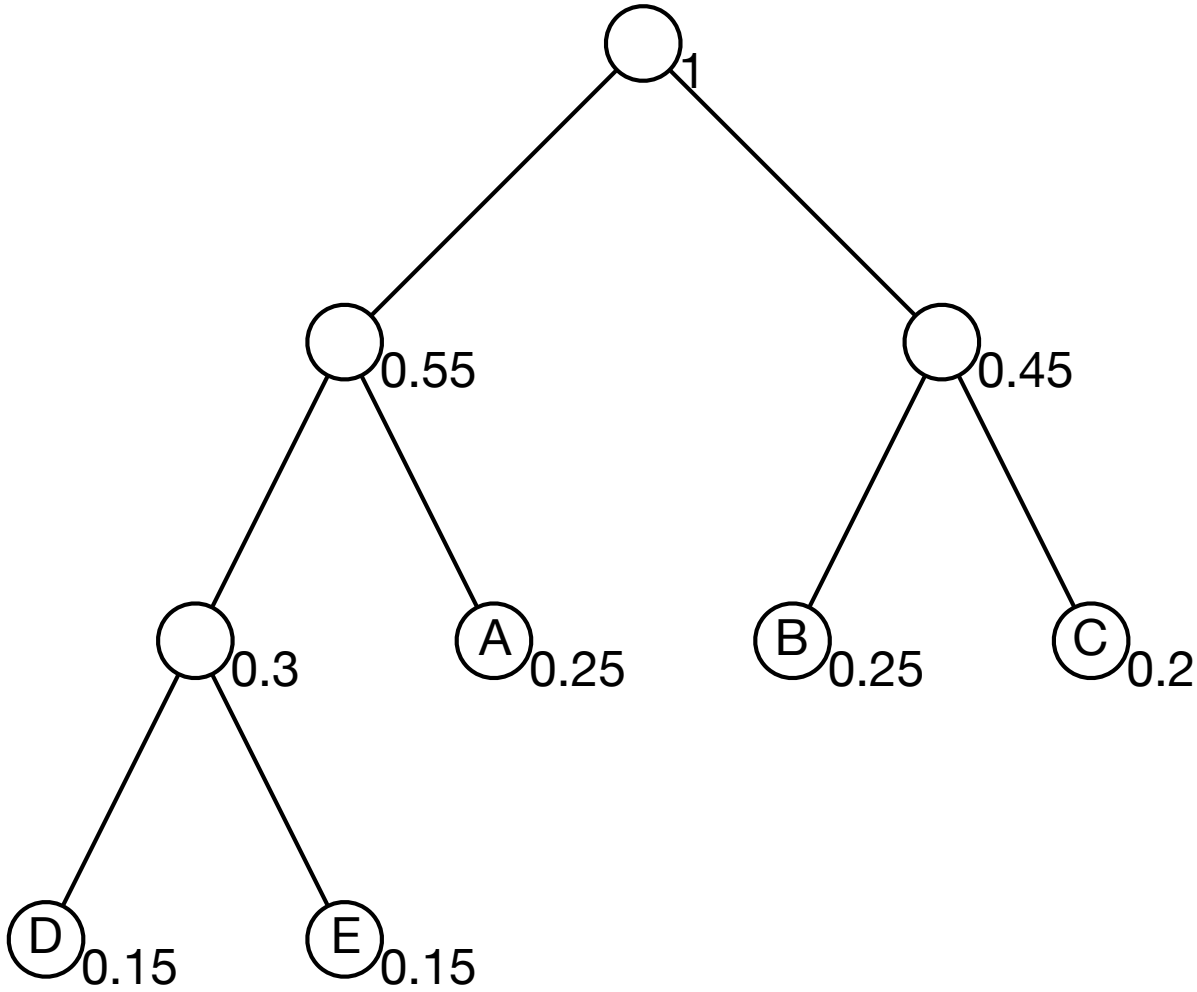
$\mathcal{S} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$. $p_1 = p_2 = 0.25$, $p_3 = 0.2$, $p_4 = p_5 = 0.15$.


0.3
A
0.25
B
0.25
C
0.2
D
0.15
E
0.15


0.45
B
0.25
C
0.2
0.3
A
0.25
D
0.15
E
0.15


0.55
0.45
0.3
A
0.25
B
0.25
C
0.2
D
0.15
E
0.15


1
0.55
0.45
0.3
A
0.25
B
0.25
C
0.2
D
0.15
E
0.15



A $\leftrightarrow 10$, B $\leftrightarrow 01$, C $\leftrightarrow 00$, D $\leftrightarrow 111$, E $\leftrightarrow 110$, E$[L] = 2.3$ bits.

Computation of the expected codeword length

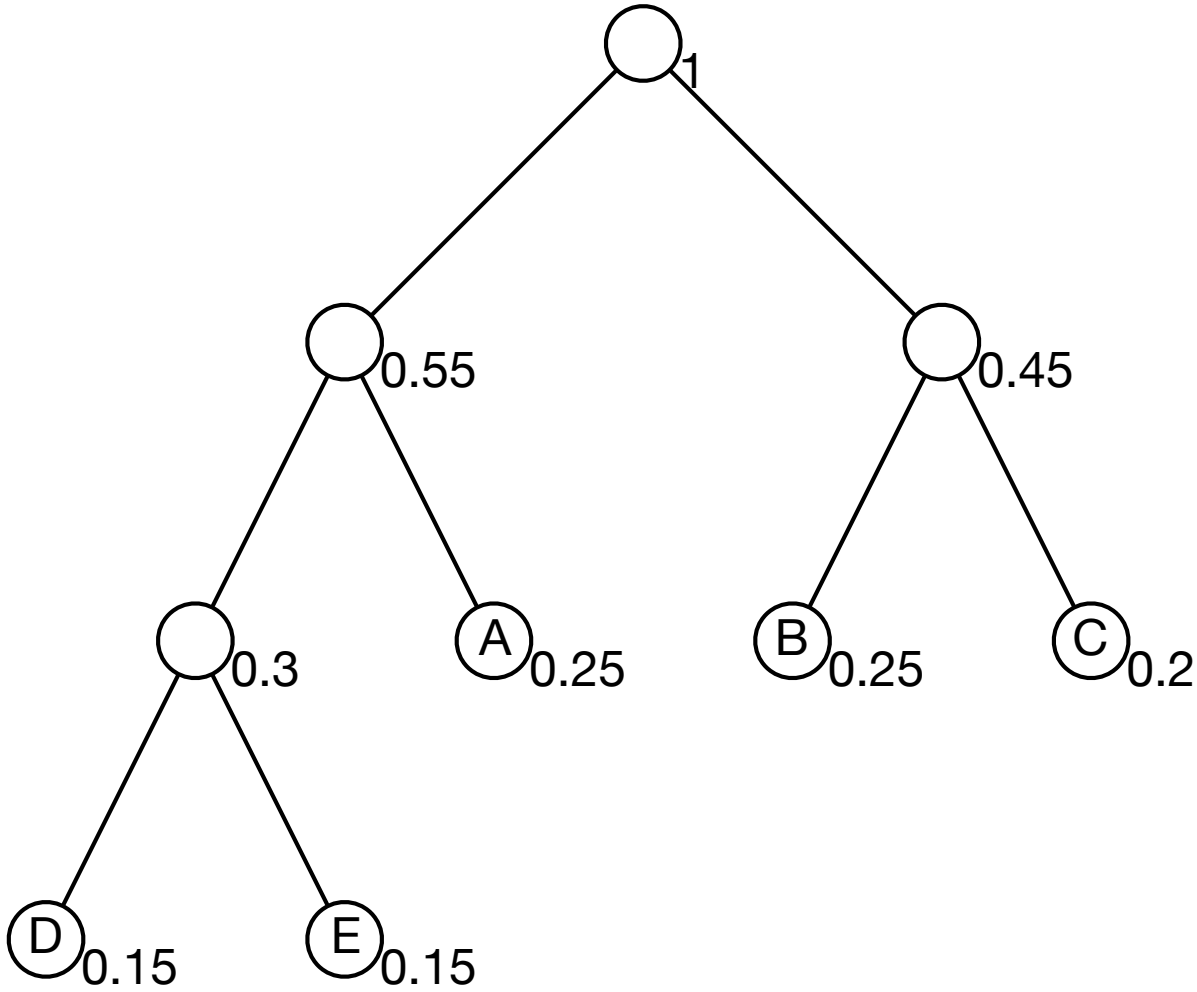
For a prefix-free code, the expected codeword length $\mathrm{E}[L]$ is equal to the expected *depth* of the codeword leaves in the code tree.

With probabilities assigned to the leaves, there is a simple bottom-up algorithm to compute the expected depth of the leaves in a binary tree.

1. Consider a binary tree with n leaves $\{\mathrm{leaf}_i \,|\, i = 1, ..., n\}$ and m non-leaf nodes $\{\mathrm{node}_j \,|\, j = 1, ..., m\}$.
2. Assign weights to all nodes in the tree from the bottom as follows:
   - Initially for $i = 1, ..., n$, leaf_i is associated with weight w_i.
   - For $j = 1, ..., m$, node_j is associated with the sum of weights of its children, denoted by u_j.
3. Let l_i denote the depth of leaf_i, $i = 1, ..., n$. Then,

$$\sum_{i=1}^{n} w_i l_i = \sum_{j=1}^{m} u_j.$$

   This is simple to prove – the two sides of the equality are just two different ways of summation.



$$\begin{aligned} \mathsf{E}[L] &= 3 \times 0.15 + 3 \times 0.15 + 2 \times 0.25 + 2 \times 0.25 + 2 \times 0.2 \\ &= 0.3 + 0.55 + 0.45 + 1 = 2.3 \end{aligned}$$

Properties of an optimal code

Recall that our goal is to solve the finite-dimensional version of the integer programming problem (3). Let us restate it as follows: ($d = |\mathcal{S}|$)

$$\begin{aligned} \min_{l_1,l_2,\ldots,l_d \in \mathbb{N}} \quad & \textstyle\sum_{i=1}^{d} p_i l_i \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{d} 2^{-l_i} \leq 1 \end{aligned} \tag{7}$$

Huffman's algorithm is *combinatorial* in nature: it outputs a code tree, equivalent to a prefix-free code, and hence the depths of the leaves automatically satisfy the constraint (Kraft Inequality) in (7). The remaining thing is to show that the depths $\{l_1, ..., l_d\}$ it found are optimal.

To do so, let us first see some nice properties of an optimal code.

Lemma 1

WLOG let $p_1 \geq p_2 \geq ... \geq p_d$. A necessary condition for $\{l_1^*, l_2^*, ..., l_d^*\}$ being the solution to (7) is that it is possible to swap codeword lengths of equiprobable letters so that the new codeword lengths $\{l_1, l_2, ..., l_d\}$ satisfy $l_1 \leq l_2 \leq ... \leq l_d$.

pf: Suppose $\exists$ an optimal code with some i, j such that $p_i > p_j$ and $l_i > l_j$. Then, one can swap the codewords for a_i and a_j and get a new code with expected codeword length

$$k = k^* - (p_i l_i + p_j l_j) + (p_i l_j + p_j l_i) = k^* + \underbrace{(p_j - p_i)}_{<0}\underbrace{(l_i - l_j)}_{>0} < k^*,$$

leading to contradiction. □

Next, we look at properties of the code tree of an optimal prefix-free code.

Lemma 2

All non-root nodes in the code tree of an optimal prefix-free code have siblings.

pf: Suppose a node in the code tree of a prefix-free code has no siblings. Then, we can move this node (and the sub-tree rooted at this node) one level up, replacing the parent of this node. The resulting new tree is still a valid code tree, with smaller expected depth of the leaves. □

Lemma 3

There exists an optimal prefix-free code such that the two least probable letters have codewords with the same length and only differ in the last bit, that is, $l_{d-1} = l_d$ and they are siblings in the code tree.

pf: Let us begin with an optimal prefix-free code. By Lemma 1, we can assume WLOG $l_1 \leq ... \leq l_{d-1} \leq l_d$. If a_{d-1} and a_d are siblings, we are done. Otherwise, there are two possibilities:

- $l_{d-1} = l_d$, but a_{d-1} and a_d are not siblings.

  In this case, by Lemma 2, a_d must have a sibling, say, a_i, $i < d-1$. This sibling has the same depth as l_d, that is, $l_i = l_{d-1} = l_d$, and hence one can swap the codewords for a_i and a_{d-1} without changing the expected codeword length. This gives us another optimal prefix-free code.
- $l_{d-1} < l_d$.

  In this case, by Lemma 2, there must be another letter a_i, $i < d-1$, which is the sibling of a_d, leading to contradiction since $l_i = l_d > l_{d-1}$.

□

Optimality of Huffman code

With Lemma 2 and 3, we conclude that the search of an optimal prefix-free code tree can be restricted to the following conditions:

- All nodes except the root have a sibling.
- Every leaf corresponds to a codeword of a letter.
- The two least probable letters corresponds to the two leaves in the code tree with the largest depth, and they are siblings.

Recall that for a weighted tree where leaves are assigned weights as the probabilities of the corresponding letters, that is, $w_i \leftarrow p_i$, $i = 1, ..., d$, we have

$$\mathsf{E}[L] = \sum_{i=1}^{d} p_i l_i = \sum_{j:\text{non-leaf node}} u_j.$$

These are keys to the proof of optimality of Huffman's greedy algorithm.

Theorem 4

Huffman code is optimal, that is, it finds an optimal solution to the minimum expected codeword length problem (7) *and gives a optimal prefix-free code.*

pf: We prove this by induction on d, the alphabet size. For $d = 2$, the algorithm outputs a tree of depth 1, the only feasible tree satisfying the conditions in the previous slide.

Suppose the algorithm is optimal for any $d \leq r$, $r \geq 2$, $r \in \mathbb{N}$.

For $d = r + 1$, the step of combining the two least probable codeword leaves together restricts the search of an optimal code tree to those satisfy the third condition in the previous slide.

Hence, we determine one non-leaf node in the code tree to be found. It remains to find the remaining non-leaf nodes to complete the whole weighted tree so that the sum of weights over the non-leaf nodes is minimized.

Obviously, this task is equivalent to finding the optimal code tree from the list of $d - 1 = r$ leaves after the combination step, which can be found by Huffman algorithm by the induction hypothesis.

The proof is complete by induction. □

Summary: Fixed-to-variable source coding

- Idea: use codewords of different length to uniquely represent a symbol, so that the expected codeword length over a given PMF is minimized.
- One-shot coding: encoding has to be injective, and
$$\mathrm{H}(S) - \log_2(\mathrm{e}(\mathrm{H}(S)+1)) \le k^* \le \mathrm{H}(S).$$
- Multi-shot coding with symbol-wise encoding: encoding has to be uniquely decodable, and
$$\mathrm{H}(S) \le k^*_{\text{u.d.}} < \mathrm{H}(S) + 1.$$
- Kraft-McMilan Inequality: a necessary and sufficient condition on codeword lengths of uniquely decodable codes. No loss to impose prefix-free constraints as far as codeword lengths are concerned.
- Huffman code: a greedy algorithm that finds an optimal prefix-free code, solving an integer programming problem with $\Theta(d \log d)$ and $\Theta(d)$ time complexity for unsorted and sorted PMFs respectively.