Introduction

This Power Round is an exploration of information theory, a branch of applied mathematics that is relevant to electrical engineering and computer science. Information theory was developed initially to determine fundamental limits in the problems of data compression and reliable communication of data, but the applications of information theory extend into many other fields like neurobiology and cryptography.

A Brief Review of Probability Notation

We review some notation that will be used throughout the power round. Throughout this section, A and B are events.

Definition: $P(A)$ is the probability that event A happens.

Definition: $P(A \cap B)$ is the probability that both A and B happen.

Definition: $P(A \cup B)$ is the probability that at least one of A and B happens.

Definition: A **random variable** is a variable X that can take on a variety of different values with various probabilities. For example, we can define a random variable X that has value 1 with probability $\frac{1}{4}$ and value 0 with probability $\frac{3}{4}$. This is alternatively written as $P(X=1)=\frac{1}{4}$ and $P(X=0)=\frac{3}{4}$.

We will use $\sum_x h(x)$ to refer to the sum of $h(x)$ over all possible values of x. For example, using X as defined in the previous paragraph, we have that $\sum_x P(X=x) = P(X=0)+P(X=1) = \dfrac{3}{4}+\dfrac{1}{4} = 1$. Note that x can take on different values, but for this random variable, unless $x=0$ or $x=1$, then $P(X=x)=0$.

We will now give a few simple probability problems to warm you up for the power round.

1. (a) You flip a fair coin twice. Let A be the event "the first coin flip was heads" and B be the event "the second coin flip was heads."
   i. [**1**] Compute $P(A)$.
   ii. [**1**] Compute $P(B)$.
   iii. [**1**] Compute $P(A \cap B)$.
   iv. [**1**] Compute $P(A \cup B)$.

   (b) You flip a fair coin three times. Let X be a random variable which has value equal to the number of times the coin landed heads.
   i. [**1**] Compute $P(X=0)$.
   ii. [**1**] Compute $P(X=1)$.
   iii. [**1**] Compute $P(X=2)$.
   iv. [**1**] Compute $P(X=3)$.

   (c) You have two four-sided dice. Let A be the event "the sum of the two numbers rolled is even" and B be the event "the product of the two numbers rolled is even."
   i. [**2**] Compute $P(A)$.
   ii. [**2**] Compute $P(B)$.
   iii. [**2**] Compute $P(A \cap B)$.
   iv. [**2**] Compute $P(A \cup B)$.

Solution to Problem 1:

(a) i. The first coin is fair and therefore is heads with probability $\boxed{\dfrac{1}{2}}$.

ii. The second coin is also fair and therefore is heads with probability $\boxed{\frac{1}{2}}$.

iii. The two events are independent, and so therefore the probability that they both are heads is $\frac{1}{2} \cdot \frac{1}{2} = \boxed{\frac{1}{4}}$.

iv. The probability that both coin flips are tails is $\frac{1}{4}$ by the above logic, so therefore the probability that one of the coins is heads is $1 - \frac{1}{4} = \boxed{\frac{3}{4}}$.

(b) i. The coin will land tails with probability $\frac{1}{2}$, so the probability that the coin will land tails three times in a row is $\left(\frac{1}{2}\right)^3 = \boxed{\frac{1}{8}}$.

ii. There are three sequences of coin flips where the coin lands heads once - the coin is heads only on the first flip, the coin is heads only on the second flip, and the coin is heads only on the third flip. Each sequence happens with probability $\frac{1}{8}$, so the answer is $3 \cdot \frac{1}{8} = \boxed{\frac{3}{8}}$.

iii. The probability that the coin will be heads two times is equal to the probability that the coin will be tails two times, which is $\boxed{\frac{3}{8}}$ by the above.

iv. The probability that the coin will be heads three times is equal to the probability that the coin will be tails three times, which is $\boxed{\frac{1}{8}}$ by the above.

(c) i. If the sum of the two numbers is even, then the two numbers must have the same parity. Both numbers are odd with probability $\frac{1}{4}$ and both numbers are even with probability $\frac{1}{4}$, giving us an answer of $\frac{1}{4} + \frac{1}{4} = \boxed{\frac{1}{2}}$.

ii. If the product of the two numbers is even, then at least one of the two numbers must be even. Both numbers are odd with probability $\frac{1}{4}$, so therefore the answer is $1 - \frac{1}{4} = \boxed{\frac{3}{4}}$.

iii. If the sum and the product of the two numbers are both even, then both numbers must be even. This happens with probability $\boxed{\frac{1}{4}}$.

iv. The only situation where this is not the case is when the sum and the product are both odd. However, if the product of the two numbers is odd, then both numbers are odd, so their sum is even. Therefore, it is always the case that either the sum or the product is even, so the answer is $\boxed{1}$.

Entropy

Information theory is fundamentally concerned with entropy. We start by defining entropy and proving some basic results about entropy.

Definition: The **entropy** of a random variable X is $H(X) = -\sum_x P(X = x) \log_2 P(X = x)$. In the event where $P(X = x) = 0$, we define $P(X = x) \log_2 P(X = x) = 0$.

Example: Given a random variable X where $P(X = 0) = \frac{1}{3}$ and $P(X = 1) = \frac{2}{3}$, we have that $H(X) = -\left(\frac{1}{3} \log_2 \frac{1}{3} + \frac{2}{3} \log_2 \frac{2}{3}\right) = \log_2 3 - \frac{2}{3}$.

2. (a) **[2]** Given a fair coin, we flip the coin once and define a random variable X on the result of the coin flip. If the coin flip is heads, then $X = 1$. Otherwise, $X = 0$. Compute $H(X)$.

(b) **[3]** Given a fair coin, we flip the coin twice and define a random variable Y on the results of the coin flips. The random variable is defined as follows:

- If the first flip is heads and the second is heads, $Y = 0$.
- If the first flip is heads and the second is tails, $Y = 1$.
- If the first flip is tails and the second is heads, $Y = 2$.
- If the first flip is tails and the second is tails, $Y = 3$.

Compute $H(Y)$.

(c) **[3]** Given a fair coin, we flip the coin twice and define a random variable Z on the results of the coin flips. The value of Z is equal to the number of times the coin lands heads. Compute $H(Z)$.

(d) **[4]** Given a random variable X that can only take on the values 0 and 1, compute the maximum possible value of $H(X)$.

(e) **[3]** Prove that for all random variables X, $H(X) \geq 0$.

Solution to Problem 2:

(a) $H(X) = -2\left(\frac{1}{2}\log_2\frac{1}{2}\right) = 2 \cdot \frac{1}{2} = \boxed{1}$.

(b) $H(Y) = -4\left(\frac{1}{4}\log_2\frac{1}{4}\right) = 4 \cdot \frac{1}{2} = \boxed{2}$.

(c) $H(Z) = -2\left(\frac{1}{4}\log_2\frac{1}{4}\right) - \frac{1}{2}\log_2\frac{1}{2} = 1 + 0.5 = \boxed{1.5}$, or $\boxed{\frac{3}{2}}$.

(d) The maximum is realized when $P(X = 0) = P(X = 1) = \frac{1}{2}$, giving a maximum entropy of $\boxed{1}$.

(e) We claim that $p\log_2 p \leq 0$ for all $p \in [0, 1]$. We specify that this is true for $p = 0$. Note that $\log_2 p \leq 0$ for $p \in (0, 1]$, so therefore $p\log_2 p \leq 0$ for $p \in [0, 1]$. Therefore, $\sum_x P(X = x)\log_2 P(X = x) \leq 0$, so $-H(X) \leq 0$, implying $H(X) \geq 0$, as desired.

Coding Theory

Before we move on to the problem of data compression, we will briefly introduce coding theory.

Computers send messages to each other using only the digits 0 and 1. If we want to send a message through a computer, the computer needs a way to translate our message into zeros and ones. For the purposes of the power round, we assume that we will only be sending integers and we will only translate them into sequences of zeros and ones.

Definition: A **codeword** is a sequence of zeros and ones.

Definition: A **source code** C is a function that converts integers to codewords. We give an example of a source code C as follows:

x	$C(x)$
1	0
2	10
3	110
4	111

In the above source code, the integer 1 is converted to the codeword 0. The integer 4 is converted to the codeword 111. The integer 12 cannot be converted by the above source code.

Definition: An integer x is **encodable** by a source code C if x can be converted by C into a codeword.

Definition: The **size** of a source code C is the number of integers which are encodable by C. The size of a source code must be at least 1.

For the remainder of the power round, if a source code C is of size N, then the integers 1 through N must be encodable by C.

Therefore, the above source code has size 4 because the integers 1, 2, 3, and 4 are encodable by it.

Definition: The **length** of a codeword $C(x)$ is the number of zeros and ones needed to write out the codeword. When we want the length of the codeword corresponding to the integer x, we write $l(x)$. The length of a codeword must be at least 1.

Therefore, in the above source code, $l(1) = 1$, $l(2) = 2$, and $l(3) = l(4) = 3$.

Definition: Two source codes C_1 and C_2 are **distinct** if they are different sizes or if, for some integer x that is encodable by both C_1 and C_2, $C_1(x) \neq C_2(x)$.

Example: If $C_1(1) = \texttt{0}$ and $C_1(2) = \texttt{1}$, and $C_2(1) = \texttt{1}$ and $C_2(2) = \texttt{0}$, then C_1 and C_2 are distinct.

Definition: A source code C is **nonsingular** if, for every pair of encodable integers x and y, $C(x) = C(y)$ if and only if $x = y$. Note that the source code above is nonsingular.

3. (a) **[2]** Compute the maximum size of a nonsingular source code where every codeword has length exactly 3.

   (b) **[3]** Compute the maximum size of a nonsingular source code where every codeword has length at most 3.

   (c) **[4]** Compute the number of distinct nonsingular source codes of size 4 where every codeword has length exactly 2.

   (d) **[5]** Compute the number of distinct nonsingular source codes of any size, where every codeword has length exactly 2.

   Solution to Problem 3:

   (a) There are $2^3 = \boxed{8}$ codewords of length 3 made up of zeros and ones.

   (b) In general, we note that there are 2^n codewords of length n. This means that there are $2^1 + 2^2 + 2^3 = \boxed{14}$ distinct codewords we can have.

   (c) There are exactly 4 codewords of length 2. Therefore, the number of distinct source codes is equal to the number of ways we can permute the 4 codewords, which is $4! = \boxed{24}$.

   (d) We can compute this by doing casework on the size of the source code. There are 4 source codes with a size of 1. There are $4 \cdot 3 = 12$ source codes with a size of 2. There are $4 \cdot 3 \cdot 2 = 24$ source codes with a size of 3. There are $4 \cdot 3 \cdot 2 \cdot 1 = 24$ source codes with a size of 4. This gives us an answer of $4 + 12 + 24 + 24 = \boxed{64}$.

Now, we usually want to send fairly long messages, so we want to be able to extend source codes to let us send arbitrarily long messages.

Definition: The **extension** C^* of a source code C takes in a sequence of positive integers $x_1, \dots, x_n$, and returns the concatenation of the codewords $C(x_1), \dots, C(x_n)$. In the source code in the beginning of the section, $C^*(1, 2, 4) = \texttt{010111}$.

Definition: An extension is **nonsingular** if, for every pair of sequences s_1 and s_2 consisting of encodable integers, $C^*(s_1) = C^*(s_2)$ if and only if s_1 and s_2 are identical.

Definition: A source code is **uniquely decodable** if its extension is nonsingular.

4. (a) **[1]** Given a source code C where $C(1) = \texttt{00}$, $C(2) = \texttt{01}$, and $C(3) = \texttt{10}$, compute $C^*(1, 2, 3, 1)$.

   (b) **[1]** Given a source code C where $C(1) = \texttt{01}$, $C(2) = \texttt{10}$, and $C(3) = \texttt{00}$, compute the sequence of integers s such that $C^*(s) = \texttt{100001}$.

(c) [**2**] Given a source code C where $C(1) = \texttt{0}$, $C(2) = \texttt{00}$, and $C(3) = \texttt{000}$, determine all sequences of integers s where $C^*(s) = \texttt{000}$.

(d) [**2**] Construct a source code of size 2 which is nonsingular, but not uniquely decodable. The codewords must have length at most 3.

(e) [**3**] Compute the number of source codes of size 2 where the source code is uniquely decodable and every codeword has length at most 2.

(f) [**4**] Compute the number of source codes of size 3 where the source code is uniquely decodable and every codeword has length at most 2.

Solution to Problem 4:

(a) `00011000`.

(b) The sequence is $2, 3, 1$.

(c) The sequences are $(1, 1, 1)$, $(1, 2)$, $(2, 1)$, (3).

(d) $C(1) = \texttt{0}$, $C(2) = \texttt{00}$ is an example. If the source code can't be uniquely decodable, then both codewords must be either all zeros or all ones.

(e) There are $6 \cdot 6 = 36$ distinct source codes of size 2. Instead of counting all the ones which are uniquely decodable, we will determine the ones which are not. There are 6 source codes which are obviously not uniquely decodable - the ones where both 1 and 2 map to the same codeword. Furthermore, there are four other source codes which are not uniquely decodable. Note that if an integer maps to `0`, then the other one cannot map to `00`. Similarly, you cannot have the two integers mapping to `1` and `11`. All other mappings are acceptable. This gives us a count of $36 - 6 - 4 = \boxed{26}$.

(f) There are two cases to consider. In the first case, all codewords are of length 2. In this case, all triplets of codewords are acceptable, so this generates $4 \cdot 3 \cdot 2 = 24$ different codewords. In the second case, there is a single codeword of length 1. Assume without loss of generality that `0` is the codeword to consider. In this case, we cannot include `00` as a codeword, and we must include `11` as a codeword. This gives us 2 codewords to choose for the third one, which gives us $2 \cdot 3 \cdot 2 = 12$ source codes that contain the codeword `0`. By similar logic, there are 12 source codes that contain the codeword `1`, giving us a count of $24 + 12 + 12 = \boxed{48}$.

Definition: A source code is **instantaneous** if no codeword is a prefix of another codeword. Note that the source code at the beginning of the section is instantaneous.

5. (a) [**3**] Construct a source code that is **uniquely decodable** but not an instantaneous code. The source code must be of size at most 4, and no codeword can have length more than 3.

(b) [**5**] Prove that all instantaneous codes are uniquely decodable.

(c) [**7**] The Kraft inequality states that for any instantaneous source code C, $\sum_{x \in C} 2^{-l(x)} \leq 1$. Prove the Kraft inequality.

Solution to Problem 5:

(a) A complicated example is where $C(1) = \texttt{00}$, $C(2) = \texttt{10}$, $C(3) = \texttt{11}$, and $C(4) = \texttt{110}$. The simplest example is where $C(1) = \texttt{0}$ and $C(2) = \texttt{01}$. Checking that a source code is not instantaneous can be done via inspection, but checking that a source code is uniquely decodable quickly is best done with a computer.

(b) Assume for the sake of contradiction that there exists an instantaneous code that is not uniquely decodable. This means that the extension is not nonsingular, so there exists some concatenation of codewords that can be encoded using two separate sequences of integers. Consider these two separate integers, and look for the first index where these two integers do not match (this index must necessarily exist). Note that up until this point, the two sequences of codewords match exactly, but at this point, one sequence will generate a shorter codeword than the other, but this shorter codeword must therefore be a prefix of the longer codeword, contradiction.

(c) Consider a binary tree where the edges on the tree represent symbols of codewords. The leaves of the tree therefore correspond to codewords, and the path from the root of the tree to the leaf gives the codewords. Note furthermore that a node at level i has length $l(x) = i$.

Let L be the maximum length codeword, and imagine superimposing a complete binary tree on top of this with height L. The maximum number of leaves in this tree is therefore 2^L. Consider all actual codewords in the tree. Because the source code is instantaneous, any codeword at level $i \leq L$ blocks 2^{L-i} leaves from being valid codewords. Note that any leaf can be blocked out by at most one codeword, so therefore we have $\sum 2^{L-i} \leq 2^L$ as we sum across all codewords, or equivalently, $\sum 2^{-i} = \sum 2^{-l(x)} \leq 1$, as desired.

Data Compression

With coding theory out of the way, we can now move on to the problem of data compression.

Definition: The **expected length** of the source code C corresponding to random variable X is $L(C) = \sum_x P(X = x)l(x)$.

6. (a) **[2]** Given a random variable X where $P(X = 1) = \frac{1}{2}$, $P(x = 2) = \frac{1}{3}$, and $P(X = 3) = \frac{1}{6}$, compute the expected length of the following source code.

x	$C(x)$
1	`00`
2	`11`
3	`1010`

(b) **[3]** Given a random variable X that takes on the values 1, 2, 3, and 4 each with probability $\frac{1}{4}$, construct an instantaneous source code with minimal expected length.

(c) **[3]** Given a random variable X where $P(X = 1) = \frac{1}{2}$, $P(X = 2) = \frac{1}{4}$, and $P(X = 3) = \frac{1}{4}$, construct an instantaneous source code with minimal expected length.

Solution to Problem 6:

(a) The expected length is $2 \cdot \frac{1}{2} + 2 \cdot \frac{1}{3} + 4 \cdot \frac{1}{6} = \boxed{\frac{7}{3}}$.

(b) $C(1) =$ `00`, $C(2) =$ `01`, $C(3) =$ `10`, $C(4) =$ `11`. Any permutation of codewords will work as long as all four distinct codewords of length 2 appear.

(c) $C(1) =$ `0`, $C(2) =$ `10`, $C(3) =$ `11`. It is necessary for $l(1) = 1$ and $l(2) = l(3) = 2$.

We will now cover one specific type of codes, Shannon codes.

A Shannon code is constructed as follows: Given a random variable X that takes on all positive integers from 1 to m where $P(X = i) \geq P(X = i + 1)$ for $1 \leq i < m$, define

$f(i) = \sum_{k=1}^{i-1} P(X = k)$. $C(i)$ is the binary representation of $f(i)$, rounded off to l_i bits, where $l_i = \left\lceil \log_2 \frac{1}{P(X=i)} \right\rceil$. As an example, the Shannon code for a random variable where $P(X = 1) = \frac{1}{2}$, $P(X = 2) = \frac{1}{4}$, and $P(X = 3) = P(X = 4) = \frac{1}{8}$, the Shannon code generated is:

x	$C(x)$
1	`0`
2	`10`
3	`110`
4	`111`

More rigorously, we have that $f(1) = 0$ and $l_1 = 1$. $f(1)$ written in binary is `0.0`, so $C(1) =$ `0`. $f(2) = \frac{1}{2}$ and $l_2 = 2$. $f(2)$ written in binary is `0.10`, so $C(2) =$ `10`. $f(3) = \frac{3}{4}$ and $l_3 = 3$. $f(3)$ written in binary is `0.110`, so $C(3) =$ `110`. $f(4) = \frac{7}{8}$ and $l(3) = 3$. $f(4)$ written in binary is `0.111`, so $C(4) =$ `111`.

7. (a) **[3]** Construct the Shannon code corresponding to the random variable X where $P(X = 1) = P(X = 2) = P(X = 3) = \frac{1}{4}$ and $P(X = 4) = P(X = 5) = \frac{1}{8}$.

   (b) **[5]** Prove that all Shannon codes are instantaneous.

   (c) **[5]** If L_X is the expected length of the Shannon code for random variable X, prove that $H(X) \le L_X < H(X) + 1$.

Solution to Problem 7:

(a) $C(1) =$ `00`, $C(2) =$ `01`, $C(3) =$ `10`, $C(4) =$ `110`, $C(5) =$ `111`.

(b) Note that $\log_2 \frac{1}{P(X=i)} \le l_i$, so therefore $P(X = i) \ge 2^{-l_i}$. Therefore, $f(i) - f(j) \ge 2^{-l_j}$ for all $i > j$, so therefore the first l_j bits of $f(i)$ and $f(j)$ cannot be identical, implying that the code is instantaneous, as desired.

(c) Note that $\log_2 \frac{1}{P(X=i)} \le l_i < \log_2 \left(\frac{1}{P(X=i)} \right) + 1$, immediately implying $H(X) \le L_X < H(X) + 1$ when you sum over all p_i.

Gibbs' Inequality

There is a useful tool in information theory known as Gibbs' Inequality. We will use it to prove some interesting identities.

Theorem: Given nonnegative numbers $p_1, \dots, p_n$ and positive numbers $q_1, \dots, q_n$ such that $\sum_{i=1}^{n} p_i = 1$ and $\sum_{i=1}^{n} q_i \le 1$, then $\sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \ge 0$.

8. (a) **[5]** Prove Gibbs' Inequality. You may assume without proof that $\ln x \le x - 1$.

   (b) **[5]** Suppose an information source X has n possible outcomes $x_1, x_2, \dots, x_n$. Intuitively, our uncertainty about the actual value of X should be maximal when all the outcomes are equally likely, i.e. when

$$p(x_1) = p(x_2) = \cdots = p(x_n) = \frac{1}{n}.$$

   Prove that for any random variable X with n outcomes, $H(X) \le \log_2 n$ (with equality only in the case above).

Solution to Problem 8:

(a) We will prove that $-\sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \leq 0$. This sum is equivalent to $\sum_{i=1}^{n} p_i \log_2 \frac{q_i}{p_i}$, which is less than or equal to $\log_2 \sum_{i=1}^{n} p_i \cdot \frac{q_i}{p_i}$ by Jensen's inequality. This sum reduces to $\log_2 \sum_{i=1}^{n} q_i \leq \log_2 1 = 0$, as desired. Therefore, $\sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \geq 0$ as desired. Note that equality holds if and only if $p_i = q_i$ for $i = 1, \dots, n$.
Alternatively, using the hint that $\ln x \geq x - 1$, we first note that $\log_2 x = \frac{\ln x}{\ln 2}$, so therefore we can prove the given statement replacing $\log_2$ with ln and still have the proof be valid, since we only adjust the left-hand side of the inequality by a constant factor.
Therefore, we have that $-\sum_{i=1}^{n} p_i \ln \frac{q_i}{p_i} \geq \sum_{i=1}^{n} p_i \left(\frac{q_i}{p_i} - 1\right) = -\sum_{i=1}^{n} q_i + \sum_{i=1}^{n} p_i = 1 - \sum_{i=1}^{n} q_i \geq 0$. For equality, we need $\frac{q_i}{p_i} = 1$, so $p_i = q_i$ for all i.

(b) Let $q_1 = \cdots = q_n = \frac{1}{n}$. Therefore, from Gibbs' Inequality, we get that $\sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} = \sum_{i=1}^{n} p_i \left(\log_2 p_i - \log_2 \frac{1}{n}\right) \geq 0$. This reduces to $\sum_{i=1}^{n} p_i \log_2 p_i + \log_2 n \geq 0$, and so $H(X) \leq \log_2 n$ as desired. Note that when we set $p_1 = \cdots = p_n = \frac{1}{n}$, we have equality.

At this point, we are now able to state one of the most powerful theorems in information theory, Shannon's source coding theorem. We will be working with a slightly simpler variation of the source coding theorem, which is as follows:

Theorem: The minimum expected length of a uniquely decodable source code C corresponding to random variable X is $H(X)$.

Intuitively, if we retrieve a random value from X, we gain $H(X)$ bits of information. Shannon's source coding theorem says that we cannot compress those $H(X)$ bits of information any further. This is a fundamental limit on how much we can compress data - we can do no better than the natural entropy of the random variable.

We will ask you to prove that this theorem is true for instantaneous source codes. Proving Shannon's source coding theorem for uniquely decodable source codes is beyond the scope of this power round, but proving the theorem for instantaneous source codes is substantially simpler and can be done with Gibbs' inequality and the Kraft inequality, which you may use without proof.

9. (a) [**2**] Given a random variable X where $P(X = 1) = \frac{1}{2}$, $P(X = 2) = \frac{1}{4}$, and $P(X = 3) = P(X = 4) = \frac{1}{8}$, compute $H(X)$.

(b) [**3**] Given the random variable above, construct a corresponding instantaneous source code with minimum expected length.

(c) [**10**] Prove that, for a random variable X, it is impossible for an instantaneous source code C corresponding to random variable X to have an expected length less than $H(X)$.

Solution to Problem 9:

(a) $H(X) = -\left(\frac{1}{2}\log_2 \frac{1}{2} + \frac{1}{4}\log_2 \frac{1}{4} + \frac{2}{8}\log_2 \frac{1}{8}\right) = \frac{1}{2} + \frac{1}{2} + \frac{3}{4} = \boxed{\frac{7}{4}}$.

(b) Let $C(1) = \texttt{0}$, $C(2) = \texttt{10}$, $C(3) = \texttt{110}$, and $C(4) = \texttt{111}$. It must be the case that $l(i) = -\log_2 P(X = i)$.

(c) For a given code with n codewords, let $l(x_i)$ be the length of the i-th codeword and let $q_i = \frac{1}{2^{l(x_i)}}$. By Kraft's Inequality, $\sum_{i=1}^{n} q_i = \sum_{i=1}^{n} \frac{1}{2^{l(x_i)}} \leq 1$. Then the entropy is given by $H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$, so Gibbs' Inequality yields $H(X) \leq -\sum_{i=1}^{n} p_i \log_2 q_i = \sum_{i=1}^{n} p_i \cdot l(x_i)$. Thus, we see that the expected length of an outcome exceeds that of the entropy of the code $H(X)$.

Huffman Codes

Definition: A source code is **optimal** if it is instantaneous and no other instantaneous source code has a shorter expected length than the specified source code.

We will now give an algorithm that will generate optimal source codes. Consider a random variable X with the following probability distribution:

x	$P(X = x)$
1	0.18
2	0.19
3	0.20
4	0.21
5	0.22

$$\begin{tikzcd}
(1, 0.18) & (2, 0.19) & (3, 0.20) & (4, 0.21) & (5, 0.22)
\end{tikzcd}$$

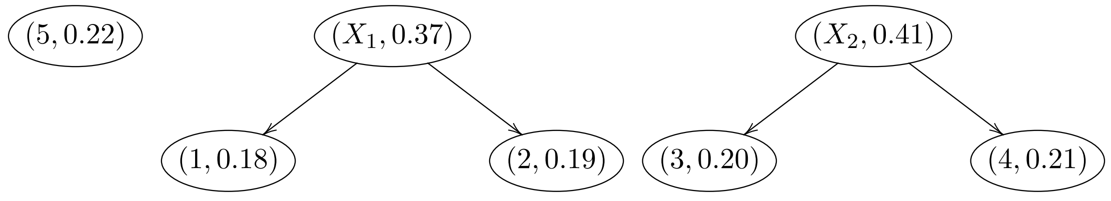
Here is how we will construct an optimal source code. Note that the values $x = 1$ and $x = 2$ are the values that occur least frequently. Therefore, intuitively, we expect that the codewords for $x = 1$ and $x = 2$ will be the longest code words. Furthermore, the two longest codewords should be the same length. If w_1 is the codeword for $x = 1$ and w_2 is the codeword for $x = 2$, then we can have w_1 and w_2 be identical except for the last digit, where w_1 ends with a `0` and w_2 ends with a `1`.

Because $x = 1$ and $x = 2$ will share a common prefix, constructing an optimal source code for the given random variable is the same as constructing an optimal source code for a slightly simpler random variable, where we delete the values associated with $x = 1$ and $x = 2$ and add another value that appears with probability $P(X = 1) + P(X = 2) = 0.37$.

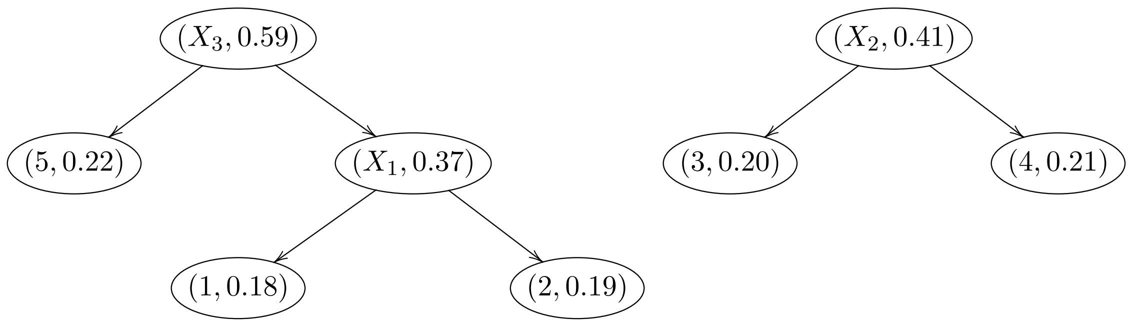
$$\begin{tikzcd}
(3, 0.20) & (4, 0.21) & (5, 0.22) & & (X_1, 0.37) \arrow[dl] \arrow[dr] & \\
 & & & (1, 0.18) & & (2, 0.19)
\end{tikzcd}$$

We use the letter X to notate values which do not map to original values, and will number each of these values for clarity. By convention, we have the value appearing less frequently on the left and the value appearing more frequently on the right.

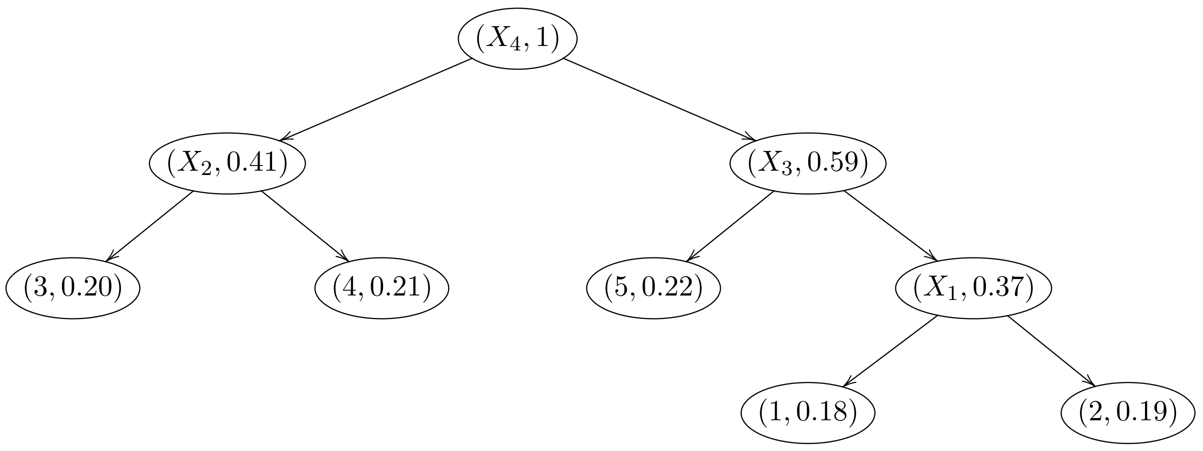
At this point, the values $x = 3$ and $x = 4$ are the least frequently occurring values, so we will combine those values as we did above.



We have only three values left now. $x = 5$ and $x = X_1$ are the least frequently occurring, so those values get combined.



We only have two values left, so we now combine them.



From this diagram, we now obtain the following source code:

x	$C(x)$
1	`110`
2	`111`
3	`00`
4	`01`
5	`10`

How did we construct this source code? Consider the value $x = 1$. From the top of the tree, we get to the leaf of the tree containing the value 1 by going right to X_3, right to X_1, and left to 1. In terms of directions, we went right, right, left. Replacing left with 0 and right with 1 gives us `110`. Repeating this process for the remaining values completes the source code as given above.

This algorithm generates a **Huffman code**, which is optimal.

10. (a) **[4]** Compute the Huffman code for a random variable with the following probability distribution:

x	$P(X = x)$
1	0.2
2	0.01
3	0.37
4	0.07
5	0.35

Recall that when you take two values and merge them, the value with the lower probability goes on the left.

(b) **[4]** Compute a probability distribution on a random variable X that takes on exactly five values that maximizes the expected length of the resulting Huffman code.

(c) **[4]** Prove that in a Huffman code for a random variable, if there exists some k such that $P(X = k) \geq \frac{1}{2}$, then $l(k) = 1$.

Solution to Problem 10:

(a) The correct answer with left-as-smaller convention is:

x	$C(x)$
1	`101`
2	`1000`
3	`0`
4	`1001`
5	`11`

In the event that they did right-as-smaller, all bits will end up being flipped. If they mixed conventions, the lengths will still be right but the bits will come out weirdly.

(b)

x	$P(X = x)$
1	$\frac{1}{2}$
2	$\frac{1}{4}$
3	$\frac{1}{8}$
4	$\frac{1}{16}$
5	$\frac{1}{16}$

(c) We claim that in the last step of the algorithm to generate a Huffman code, one of the two nodes that will be merged is the node corresponding to $x = k$. One of the two nodes that is being merged in must have probability at least $\frac{1}{2}$. If the node $x = k$ is not one of them, that implies that that node was merged in an earlier step, which is impossible because nodes are merged in increasing magnitude of probability. Therefore, the node corresponding to $x = k$ is merged in the last step, which means that it corresponds to a codeword of length 1, as desired.

We conclude this power round by asking you to prove that Huffman codes are optimal. This is a fairly involved proof, so we ask you to prove some facts that may be helpful in ultimately proving that Huffman codes are optimal.

11. (a) **[2]** Prove that in an optimal code of X, if $P(X = x) > P(X = y)$, then the codeword associated with x is no longer than the codeword associated with y.

(b) **[2]** Prove that in an optimal code of X, if l is the length of the longest codeword in X, then at least two codewords have length l.

(c) **[3]** Prove that in an optimal code of X, there are two longest codewords which differ only in the last bit.

(d) **[12]** Prove that Huffman codes are optimal.

Solution to Problem 11:

(a) Assume for the sake of contradiction that the given claim is not true, then swap the offending codewords to get a code with strictly smaller expected length, which contradicts optimality.

(b) Assume for the sake of contradiction that the given claim is not true. Therefore, there is a unique longest codeword. However, we can remove the last bit of that codeword and still maintain the instantaneous property, which contradicts optimality.

(c) Assume for the sake of contradiction that the given claim is not true. If all longest codewords differ by more than the last bit, then we can delete the last bit from one of the longest codewords and still maintain the instantaneous property, which contradicts optimality.

(d) We prove by induction that Huffman codes generate an optimal code.

The base case is when we have to encode exactly two symbols. In this case, our algorithm will output 0 and 1 as the codewords, which is obviously optimal.

We now assume that the Huffman code is optimal when encoding exactly $k-1$ symbols, and seek to prove that given this, the Huffman code is optimal when encoding exactly k symbols.

We assume without loss of generality that $P(X=i) \geq P(X=i+1)$ for $1 \leq i < k$. Let X' be the *Huffman reduction* of X, a random variable taking on $k-1$ values where $P(X=i) = P(X'=i)$ for $1 \leq i \leq k-2$ and $P(X'=k-1) = P(x=k-1) + P(x=k)$.

We will use *'s to annotate codes that are optimal.

Let $C^*_{k-1}(X')$ be the optimal code for X', and $C^*_k(X)$ be the canonical optimal code for X. From $C^*_{k-1}(X')$, we can construct a code $C_k(X)$ as follows - let $C_k(i) = C^*_{k-1}(i)$ for $1 \leq i \leq k-2$, $C_k(k-1) = C^*_{k-1}(k-1) + \texttt{0}$, and $C_k(k) = C^*_{k-1}(k-1) + \texttt{1}$. Note that the expected length of C is just $L(C^*_{k-1}(X')) + P(X=k-1) + P(X=k)$.

Now, from $C^*_k(X)$, we can construct a condensed code $C_{k-1}(X')$ by collapsing the two least likely symbols and taking their common prefix. Note that $L(C_{k-1}(X')) = L(C^*_k(X)) - P(X=k-1) + P(X=k)$. Note therefore that $L(C_k(X)) - L(C^*_k(X)) + L(C_{k-1}(X')) - L(C^*_{k-1}(X')) = 0$. However, note that $C^*_k(X)$ and $C^*_k(X')$ were optimal, so therefore this implies that our generated codes $C_k(X)$ and $C_{k-1}(X)$ are optimal because we have that $L(C_k(X)) - L(C^*_k(X)) \leq 0$ and $L(C_{k-1}(X')) - L(C^*_{k-1}(X')) \leq 0$ by optimality, completing the induction step. This proves that Huffman codes are optimal.