香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Information Theory

Isaac Chan {`snowysecret`}

2025-06-28

Table of Contents

Today's Lecture

We will focus on a few **introductory** ideas and concepts in information theory today, (fortunately) without delving too much into the mathematical details. (Feel free to read the reference materials at the end of this slide deck if you are interested in the mathematics behind it.)

We will also consider examples of information theory analysis applied to OI tasks (since learning more in OI is why we are here today!)

What is Information Theory?

"The mathematical study of the quantification, storage, and communication of information."

Two fundamental questions arise:

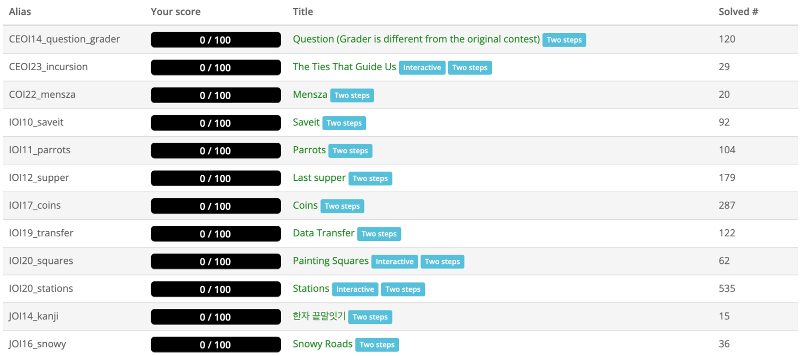
1. How much information is contained in a signal/data/message?
2. What are the limits to information transfer over a channel that is subject to noisy perturbations (i.e. a *noisy channel*)?

Why do we care?

- Binge-watching a Netflix series on Wi-Fi
- Video calling your friend
- Backing up files to cloud storage
- Scanning a QR code to pay for coffee
- Unlocking an iPhone with Face ID

Why do we care?

Information Theory is closely related to *communication tasks* in OI.



Alias	Your score	Title	Solved #
CEOI14_question_grader	0 / 100	Question (Grader is different from the original contest) Two steps	120
CEOI23_incursion	0 / 100	The Ties That Guide Us Interactive Two steps	29
COI22_mensza	0 / 100	Mensza Two steps	20
IOI10_saveit	0 / 100	Saveit Two steps	92
IOI11_parrots	0 / 100	Parrots Two steps	104
IOI12_supper	0 / 100	Last supper Two steps	179
IOI17_coins	0 / 100	Coins Two steps	287
IOI19_transfer	0 / 100	Data Transfer Two steps	122
IOI20_squares	0 / 100	Painting Squares Interactive Two steps	62
IOI20_stations	0 / 100	Stations Interactive Two steps	535
JOI14_kanji	0 / 100	한자 끝말잇기 Two steps	15
JOI16_snowy	0 / 100	Snowy Roads Two steps	36

Revision of Logarithms

You should have encountered logarithms in OI many times, but just as a refresher:

Definition

For $a, c > 0$, $\log_a c = b \iff a^b = c$

In OI (and in today's lecture) we will mainly use $a = 2$. Some examples:

- $\log_2 1 = 0, 2^0 = 1$
- $\log_2 16 = 4, 2^4 = 16$
- $\log_2 6 \approx 2.585, 2^{2.585} \approx 6$
- $\log_2 0.5 = -1, 2^{-1} = \frac{1}{2} = 0.5$

Discrete Random Variables

Before we begin our study of information, it is helpful to understand the notion of a **discrete random variable**.

Definition

A **discrete random variable** is a variable that can take one or more discrete values, each associated with a certain probability.

Everyday examples:

- **Coin toss**: $X \in \{\text{Heads}, \text{Tails}\}$, with $\Pr\{X = \text{Heads}\} = 0.5$.
- **Die roll**: $Y \in \{1, 2, 3, 4, 5, 6\}$, each with probability $1/6$.
- **Daily weather**: $W \in \{\text{Sunny}, \text{Cloudy}, \text{Rainy}\}$, where $\Pr(W = \text{Sunny}) = 1/4, \Pr(W = \text{Cloudy}) = 1/2, \Pr(W = \text{Rainy}) = 1/4$.

What is Information?

- Information is anything that has the power *to inform*—to reduce uncertainty about the outcome of a random event.
- In information theory, we quantify that reduction. The smaller the remaining uncertainty, the more *information* we must have received. (We will formally see how this is done later.)
- Information is measured in **bits**. Informally, one bit answers a yes/no question (or is a true/false statement).

Measuring Information in Bits

- Let's say we have a secret number X from 1 to 200 inclusive, where X can be each number with equal probability.
- Here, the *space of possibilities* for the discrete random variable X is $\{1, 2, \ldots, 200\}$.

Measuring Information in Bits

- Let's say we have a secret number X from 1 to 200 inclusive, where X can be each number with equal probability.
- Here, the *space of possibilities* for the discrete random variable X is $\{1, 2, \ldots, 200\}$.
- Now, let's introduce an observation: "The secret number X is at most 100."
- This observation cuts the space of possibilities in **half**: now it is equal to $\{1, 2, \ldots, 100\}$.
- We say that this observation holds **one bit of information**, since it cuts the space of possibilities by a factor of 2.

Measuring Information in Bits

- What if our observation was instead: ”The secret number X is at most 50.”
- This observation cuts the space of possibilities by 75%: now it is equal to $\{1, 2, \ldots, 50\}$.
- We say that this observation holds **two bits of information**, since it cuts the space of possibilities by a factor of 4.
- Continuing the pattern, what if our observation was instead: ”The secret number X is at most 25.”? (Hint: This observation cuts the space of possibilities by (approximately) a factor of 8.)

Measuring Information in Bits

Observation	$X \leq 100$	$X \leq 50$	$X \leq 25$
Nr. of bits of info, I	1	2	3
Probability of observation, p	$1/2$	$1/4$	$1/8$

- The observation ”The secret number X is at most 25” is a lot more ”valuable” in a sense that it conveys the most number of bits of information (and intuitively, it reduces the search space by a lot).
- However, it may be obvious that high information content observations also come with low probability.
- We can try to find a relationship between number of bits of information, I, and the probability of observation, p. It turns out that this relationship is $I = -\log_2(p)$.

Surprisal

Define the **surprisal** (or self-information), $S(x)$, with respect to an outcome x:

Definition

$S(x) = -\log_2 P(x)$, where $P(x)$ is the probability of outcome x.

- **Coin toss**: $P(x_1) = 0.5 \;\Rightarrow\; S(x_1) = 1$.
- **Die roll**: $P(x_2) = 1/6 \;\Rightarrow\; S(x_2) \approx 2.58$.
- **Rarer event**: A lottery win with $P(x_3) = 10^{-7}$ has $S(x_3) \approx 23.3$ (a huge surprise!).

We can see that, the lower the probability of an outcome, *(1) the higher the surprisal* and *(2) the more information is conveyed*. (From now on, we may use phrases such as ”high surprisal” and ”containing many bits of information” interchangeably.)

A Harder Example: Wordle

1. A hidden 5–letter English word.
2. You get 6 guesses.
3. After each guess every letter is marked:

🟩	correct letter in the correct place
🟨	correct letter, wrong place
⬜	letter not in the word

A Harder Example: Wordle

To make sure everyone understands the rules of Wordle (and as an excuse to have some fun), let's play this custom Wordle game!

https://mywordle.strivemath.com/?word=zssxr

Rules:

1. No yelling out the answer, no collaboration with other participants.
2. The word is not too easy for non-OIers but should be easy for everyone here :)
3. Jot down or try to recall what techniques you used to try guessing the word using as few guesses as possible.

A Harder Example: Wordle

The answer is DEBUG!

What ”heuristics” or strategies did you use to minimize the number of guesses you made?

https://www.menti.com/alk8mifhhwxm

A Harder Example: Wordle

The answer is DEBUG!

What ”heuristics” or strategies did you use to minimize the number of guesses you made?

https://www.menti.com/alk8mifhhwxm

- As many distinct letters as possible in the first few guesses?
- Prioritize vowels over consonants?
- Go for a risky guess that narrows down the search space greatly if some meaningful result is given?
- Aim to maximize the information we get out of each guess, e.g. we don’t really want to guess obscure words that might give `NNNNN`.

Wordle: Guess an Anagram of `S, A, B, R, E`

Suppose we want to guess a secret word that is guaranteed to be an *anagram* of the letters `S, A, B, R, E`.

Under this restriction there are 6 equally likely candidates: `SABRE, SABER, BASER, BARES, BEARS, BRAES`.

Wordle: Guess an Anagram of `S, A, B, R, E`

Suppose we guess the word `SABRE`. Because every letter must be present, the feedback we get can only be *green*(G) or *yellow*(Y).
Here are the five possible patterns, the probabilities of seeing them, and the information each conveys:

Pattern	Remaining words	p	Information, $I = -\log_2 p$
GGGGG	SABRE	$1/6$	2.58
GGGYY	SABER	$1/6$	2.58
YYYGY	BEARS	$1/6$	2.58
YYYYY	BRAES	$1/6$	2.58
YGYYY	BASER, BARES	$2/6$	1.58

Wordle: Guess an Anagram of S, A, B, R, E

If the feedback after SABRE is YGYYY, the candidate list is BASER, BARES.

- **Next guess:** play BASER.
- **Two possible outcomes:**
  - GGGGG: Word is BASER. Game over.
  - GGYGY: Word is BARES. Game over.

Thus, even in the worst case we finish in 2 guesses.

Analysis

Pattern	Remaining words	p	Information, $I = -\log_2 p$	# of guesses
GGGGG	SABRE	$1/6$	2.58	1
GGGYY	SABER	$1/6$	2.58	1
YYYGY	BEARS	$1/6$	2.58	1
YYYYY	BRAES	$1/6$	2.58	1
YGYYY	BASER, BARES	$2/6$	1.58	2

Takeaways:

1. Outcomes that have higher surprisal / carry more bits of information solve the puzzle using fewer number of guesses!
2. Surprising / informative outcomes are unlikely to appear; less surprising / informative outcomes are more likely to appear.

So why do we care about surprisal?

- Intuition: Measuring ”surprisal” gives us an indication on the number of guesses to solve a guessing game.
- The larger the surprisal, the more we learn in one shot.

Optimal Solver for Wordle

- Let’s say we want to build an optimal solver for Wordle; here, by ”optimal” we mean ”minimizing the expected number of guesses”.
- You may note that throughout the game, we constantly need to make the following decision: ”Given the information from previous guesses (possibly none), find the *best* next guess.”
- Here, ”best” would mean a choice that yields the minimum expected number of guesses, as that is our final objective.

Optimal Solver for Wordle

- How do we find the next guess that yields the minimum expected number of guesses until we find the secret word?
- We have learnt that ”surprising results” (results with high surprisal), i.e. results conveying many bits of information, would result in fewer guesses needed to find the secret word.
- Maybe we want to find a guess with the highest surprisal?

Optimal Solver for Wordle

- How do we find the next guess that yields the minimum expected number of guesses until we find the secret word?
- We have learnt that ”surprising results” (results with high surprisal), i.e. results conveying many bits of information, would result in fewer guesses needed to find the secret word.
- Maybe we want to find a guess with the highest surprisal? **... Not quite.** Note that surprisal is not defined *per se*, but rather based on an outcome. In this case, we **first** submit a guess, then the surprisal of the **feedback** (sequence of 5 colours) is calculated with respect to this guess.
- What can we do instead?

Optimal Solver for Wordle

- Beforehand you can talk about the expected surprisal (over all possible feedback) instead - the expected surprisal for a guess is well-defined!
- We can see that finding the guess with maximum expected surprisal is equivalent to finding the next ”best” guess for minimizing expected number of total guesses.
- In general, the concept of ”expected surprisal” is instrumental in information theory. We refer to this as **entropy**.

Table of Contents

Entropy: The Math

Definition

For a discrete random variable X with probability mass function $P(x)$, its **entropy** is defined as $H(X) = -\sum_{x \in \mathcal{X}} P(x) \log_2 P(x)$, where $\mathcal{X}$ denotes the set of possible values X can take.

- $H(X) = \mathbb{E}[S(X)]$ —the *expected surprisal*.
  - Proof: $H(X) = -\sum_{x \in \mathcal{X}} P(x) \log_2 P(x) = \sum_{x \in \mathcal{X}} P(x) S(x) = \mathbb{E}[S(X)]$
- We can alternatively write $H(X) = -\mathbb{E}[\log_2 P(x)]$.

Entropy: The Intuition (arguably more important)

1. **Entropy as the Expectation of Information Gain** when receiving a message X. (This follows directly from the equation $H(X) = \mathbb{E}[S(X)]$.)
2. **Entropy as the Average Uncertainty** of a random variable: *roughly* how many yes/no questions you still need, on average, to learn the value of X.

Entropy: Numerical Example

- Consider a coin with Pr(heads) = p, Pr(tails) = $1 - p$.
- Calculate the entropy when $p = 0.5$, $p = 0.9$, and $p = 1$ respectively.

Entropy: Numerical Example

- Consider a coin with Pr(heads) = p, Pr(tails) = $1 - p$.
- Calculate the entropy when $p = 0.5$, $p = 0.9$, and $p = 1$ respectively.
- $p = 0.5$: $H(0.5) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = -(0.5 \cdot (-1) + 0.5 \cdot (-1)) = 1$.
- $p = 0.9$: $H(0.9) = -(0.9 \log_2 0.9 + 0.1 \log_2 0.1) \approx 0.47$.
- $p = 1$: $H(1) = -(1 \log_2 1 + 0 \log_2 0) = 0$ (here, we use the convention that $0 \log_2 0 = 0$).

Entropy: Numerical Example

- Interpretation of the result: Given knowledge about p, we want to store the results of a sequence of n independent coin flips. Then it is intuitive that we will need $H(0.5) \times n = 1 \times n = n$ bits to store the results of all coin flips when $p = 0.5$;
- whereas since we know that most results will be heads, for $p = 0.9$ we only need $\approx 0.47n$ bits to store the results;
- and we do not need any bits at all for $p = 1$, we know they are all heads.

Entropy: Applied to Wordle

- Now that we are equipped with knowledge about entropy, back to our Wordle game.
- Recall where we left off: we want to follow a strategy that maximizes the expected surprisal; that is, one that maximizes the **expected** number of bits of information from a particular guess.
- A “high quality” guess is one whose entropy is maximized.

One-Step Entropy Maximizing Algorithm

1. Initialize candidate list with the list of all 5-letter English words.
2. Score every legal guess by its entropy.
3. Play the guess with the highest score.
4. Prune the candidate list using the actual colour feedback.
5. Repeat (2) to (4) until only one word remains.

One-Step Entropy Maximizing Algorithm

# Options / Uncertainty
12972 Opt / 13.66 Bits

word	relative prob.
aback	▬▬▬▬
abase	▬▬▬▬
abate	▬▬▬▬
abbey	▬▬▬▬
abbot	▬▬▬▬

⋮

Top picks	**E[info.]**
tares	**6.19**
lares	6.15
rales	6.11
rates	6.10
teras	6.08

One-Step Entropy Maximizing Algorithm

# Options / Uncertainty

12972 Opt / 13.66 Bits

761 Opt / 9.57 Bits

T	A	R	E	S

4.09 Bits

Top picks	**E[info.]**
doily	**5.46**
oldie	5.41
colin	5.41
noily	5.39
login	5.37

word	relative prob.
bebop	▬
becke	▬
bedim	▬
bedye	▬
beech	▬

⋮

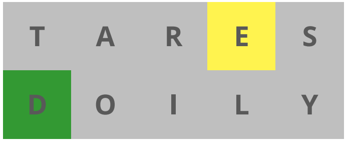
One-Step Entropy Maximizing Algorithm

# Options / Uncertainty

12972 Opt / 13.66 Bits

761 Opt / 9.57 Bits

7 Opt / 2.81 Bits



6.76 Bits

Top picks	E[info.]
begun	2.81
bough	2.81
bound	2.81
budge	2.81
bugle	2.81

word	relative prob.
debud	▬
debug	▬
deeve	▬
degum	▬
dench	▬
deuce	▬
dunce	▬

One-Step Entropy Maximizing Algorithm

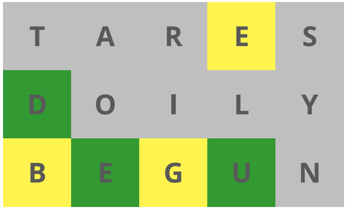
# Options / Uncertainty

12972 Opt / 13.66 Bits

761 Opt / 9.57 Bits

7 Opt / 2.81 Bits

1 Opt / 0 Bits



2.81 Bits

Top picks	E[info.]
debug	0.00

word	relative prob.
debug	▬

One-Step Entropy Maximizing Algorithm

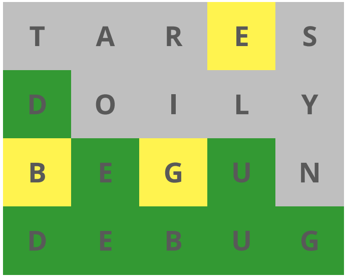
# Options / Uncertainty

12972 Opt / 13.66 Bits

761 Opt / 9.57 Bits

7 Opt / 2.81 Bits

1 Opt / 0 Bits



0.00 Bits

Solved in 4 guesses!

Review: How Entropy is Used Here

In the Wordle example, we can see that entropy can be used for:

1. Calculating the expected information gain from a given guess
2. Quantifying the current game state's remaining uncertainty

Wordle: Nuances

- What we have looked at is a simplified version of Wordle; in the actual game, some words (common English words) have higher probability of appearing than other words.
- Hence we can no longer assume that every word in the ”dictionary” (list of possible secret words) has equal probability of being the secret word.
- This could affect the uncertainty calculations: For example, under weighted probabilities, if we have two words left but one word is 99 times more likely to appear as the secret word compared to the second word, then we are relatively certain about the answer!

Wordle: Further Adjustments

Of course, we can modify our algorithm based on our specific needs:

1. Incorporate probability of each word occurring as the secret word into the calculations (see previous slide).
2. Look 2 steps ahead (maximize entropy sum of 2 steps).
3. Look k steps ahead.
4. Instead of aiming for minimizing the expected number of guesses, minimize the chance of needing $\geq L$ guesses (maybe $L = 5$ for the case of Wordle).
5. Maybe there is a (non-linear) score assigned to each number of guesses (e.g. 150 points if guessed in 1 try, 100 points if guessed in 3, 90 points if guessed in 4, ...), we might want to maximize the expected score instead.

Table of Contents

What is Data Compression?

- Data Compression: The process of encoding information in **fewer bits** than its original representation.
- There are two main types:
  - Lossless compression, which preserves all original data
  - Lossy compression, which discards some information
- We will focus on lossless compression today.

Terminology in Data Compression

- An **alphabet**, Σ, is a set of *symbols* we want to store or send, e.g. the 26 Latin letters.

Terminology in Data Compression

- An **alphabet**, Σ, is a set of *symbols* we want to store or send, e.g. the 26 Latin letters.
- A **code alphabet** is a set of symbols we actually save after compression. For the entirety of this lecture, we will assume the code alphabet is $\{0, 1\}$ (i.e. we save data as a sequence of bits).

Terminology in Data Compression

- An **alphabet**, Σ, is a set of *symbols* we want to store or send, e.g. the 26 Latin letters.
- A **code alphabet** is a set of symbols we actually save after compression. For the entirety of this lecture, we will assume the code alphabet is $\{0, 1\}$ (i.e. we save data as a sequence of bits).
- A **symbol code** enc is a **function** (or more simply, a rule), that assigns every symbol in the alphabet Σ a pattern of bits (we write $enc : \Sigma \longrightarrow \{0, 1\}^*$).

Terminology in Data Compression

- An **alphabet**, Σ, is a set of *symbols* we want to store or send, e.g. the 26 Latin letters.
- A **code alphabet** is a set of symbols we actually save after compression. For the entirety of this lecture, we will assume the code alphabet is $\{0, 1\}$ (i.e. we save data as a sequence of bits).
- A **symbol code** enc is a **function** (or more simply, a rule), that assigns every symbol in the alphabet Σ a pattern of bits (we write $enc : \Sigma \longrightarrow \{0, 1\}^*$).
- **Codeword**, $enc(a)$, is the particular bit pattern for a single symbol $a \in \Sigma$ (e.g. $enc(\mathtt{E}) = 110$).

Terminology in Data Compression (cont.)

Example:

Character	Codeword
A	0110
B	00
C	111
D	001100
E	110

Facts:

- Alphabet, $\Sigma = \{A, B, C, D, E\}$
- Codeword of A is $enc(A) = 0110$.
- Codeword of B is $enc(B) = 00$.
- The function enc is termed the *symbol code*.

Encoding a String

- A string x is a finite sequence of symbols in the alphabet (we write $x \in \Sigma^*$), e.g. `HKOI` for a 26-letter alphabet.
- For a string $x = x_1 x_2 \dots x_n$ we just glue the codewords together. The encoded message of x is defined as:

$$enc^*(x_1 x_2 \dots x_n) = enc(x_1)\, enc(x_2)\ \dots\ enc(x_n).$$

- For example, $enc^*(\texttt{ACE}) =$ `"0110"+"111"+"110"` $=$ `"0110111110"`

The Symbol Code

- The symbol code defines a rule that assigns each character a codeword; it is not hard to see that this rule must be ”clever” in some sense, at least we can’t just assign each character to the same codeword!
- Encoding is important, but so is decoding - that is the ultimate goal of data compression!
- General goals:
  1. Correctness (injective / uniquely decodable - more on that in a minute).
  2. Minimize *expected* length of encoded message (does this remind you of communication tasks in OI?).
  3. Sometimes ease of decoding.

The Symbol Code

We say that a symbol code is...

- *Unambiguous*, if each **symbol** (e.g. A, B, C, D, E) gets its own codeword (one-to-one).
- *Uniquely decodable*, if after concatenation, there is still only one sensible way to split a sequence of bits back into a string.
- A *prefix code*, if no codeword is a prefix of another. As soon as the decoder finishes reading a codeword, it instantly knows it can start the next one.

Clearly,

$$\{\text{prefix codes}\} \subset \{\text{uniquely decodable codes}\} \subset \{\text{unambigiuous codes}\}$$

The Symbol Code: Example 1

Character	Codeword
A	00
B	01
C	10
D	11

1. Is this an unambiguous code?
2. Is this a uniquely decodable code?
3. Is this a prefix code?

The Symbol Code: Example 1

Character	Codeword
A	00
B	01
C	10
D	11

1. Is this an unambiguous code? **Yes.**
2. Is this a uniquely decodable code? **Yes.**
3. Is this a prefix code? **Yes.**

The Symbol Code: Example 2

Character	Codeword
A	0
B	10
C	01

1. Is this an unambiguous code?
2. Is this a uniquely decodable code?
3. Is this a prefix code?

The Symbol Code: Example 2

Character	Codeword
`A`	`0`
`B`	`10`
`C`	`01`

1. Is this an unambiguous code? **Yes.**
2. Is this a uniquely decodable code? **No.** (Consider `"010" = "0" + "10" = "01" + "0"`: two interpretations.)
3. Is this a prefix code? **No.**

The Symbol Code: Example 3

Character	Codeword
A	0110
B	00
C	111
D	001100
E	110

1. Is this an unambiguous code?
2. Is this a uniquely decodable code?
3. Is this a prefix code?

The Symbol Code: Example 3

Character	Codeword
A	0110
B	00
C	111
D	001100
E	110

1. Is this an unambiguous code? **Yes, by inspection.**
2. Is this a uniquely decodable code?
3. Is this a prefix code?

The Symbol Code: Example 3

Character	Codeword
A	0110
B	00
C	111
D	001100
E	110

1. Is this an unambiguous code? **Yes, by inspection.**
2. Is this a uniquely decodable code?
3. Is this a prefix code? **No, consider B and D.**

The Symbol Code: Example 3

Character	Codeword
A	0110
B	00
C	111
D	001100
E	110

1. Is this an unambiguous code? **Yes, by inspection.**
2. Is this a uniquely decodable code? **???**
3. Is this a prefix code? **No, consider B and D.**

Sardinas–Patterson Algorithm

- The Sardinas–Patterson algorithm decides whether a given finite symbol code is **uniquely decodable**. (Recall: uniquely decodable $\Rightarrow$ every encoded string has *exactly one* possible interpretation.)
- Idea: If a real ambiguity exists, then some encoded message can be built in **two different** ways.
- We try to grow two partial reconstructions w_1 and w_2 (assume $|w_1| < |w_2|$) step-by-step by appending a complete codeword onto w_1.
- At the same time, maintain a **dangerous suffix** for each ongoing partial reconstruction w_1 and w_2 (assume $|w_1| \leq |w_2|$) - what string must be appended to the end of w_1 to make it equal to w_2.
- Iteratively update the dangerous suffix set. If the suffix set contains ε then the symbol code is not uniquely decodable.

Sardinas-Patterson Algorithm

Formally, let $\mathcal{C}$ be the symbol code we want to test and write S_k for a set of dangerous suffixes after k steps.

1. **Step 1.** Compute
$$S_1 = \{v \mid \exists u \in \mathcal{C} \text{ such that } uv \in \mathcal{C}\}$$
2. **General step.** For $k \geq 1$ define
$$S_{k+1} = \{v \mid \exists u \in S_k,\ x \in \mathcal{C} \text{ with } x = u\,v \text{ or } u = x\,v\}.$$

Stopping rules:

1. If $\varepsilon \in S_k$ for some k then the symbol code is **not** uniquely decodable.
2. If some S_k is **empty** or coincides with a previous $S_{k'}$ (with $k' < k$), then the symbol code is uniquely decodable.

Sardinas-Patterson Algorithm

Consider a familiar example from previous slides:

Character	Codeword
A	0110
B	00
C	111
D	001100
E	110

For $\mathcal{C} = \{0110, 00, 111, 001100, 110\}$, is this symbol code uniquely decodable?

Sardinas–Patterson Algorithm: Worked Example

Consider the code $\mathcal{C} = \{0110,\ 00,\ 111,\ 001100,\ 110\}$.

1. Suffixes left over when one codeword is the *prefix* of another: $001100 = 00\,\underline{1100}$. Hence $S_1 = \{1100\}$.
2. For each $u \in S_1$ and each $x \in \mathcal{C}$ create new suffixes: $u = 1100,\ x = 110$ gives $u = x\,\underline{0}$. Thus $S_2 = \{0\}$.
3. Again combine $u = 0$ (the only element in S_2) with every codeword:

$$\begin{aligned} x &= 0110 &\Rightarrow\quad x &= u\,\underline{110}\ (\rightarrow 110) \\ x &= 00 &\Rightarrow\quad x &= u\,\underline{0}\ (\rightarrow 0) \\ x &= 001100 &\Rightarrow\quad x &= u\,\underline{01100}\ (\rightarrow 01100) \end{aligned}$$

   Hence $S_3 = \{110,\ 0,\ 01100\}$.
4. For each $u \in S_3$ and each $x \in \mathcal{C}$ create new suffixes: in particular, $u = 110$ and $x = 110$ gives $u = x\underline{\varepsilon}$. Hence $\varepsilon \in S_4$, we fall into Case 1 and $\mathcal{C} = \{0110,\ 00,\ 111,\ 001100,\ 110\}$ is not uniquely decodable.

*Dangerous suffix: grey

Step 1
001100
00$\,$1100

Step 2
001100
00$\underline{110}$0

Step 3
001100110
00110$\underline{0110}$

Step 4
001100$\underline{110}$
001100110

Sardinas-Patterson Algorithm: Conclusion

So, to conclude how the algorithm works:

- Intuitively think of the algorithm as trying to construct the same message in two different ways.
- In step 1 of the algorithm we look for pairs of distinct characters (c_1, c_2) where one of their codewords is a prefix of the other's codeword. Initialize the partial reconstructions $w_1 = enc(c_1), w_2 = enc(c_2)$.
- Then we try to make the two messages equal by appending codewords to the shorter message, while maintaining the property that ”the shorter of the two resulting messages is a prefix of the longer of them”.
- If we can ever make $w_1 = w_2$, the code is not uniquely decodable since the encoded message w_1 can be decoded in two distinct ways.

Let’s solve M0413 - Ambiguous Code together!

Abridged Statement

Given a symbol code (”encoding scheme”), determine whether it is uniquely decodable. If it is, then report it.
If it is not, find the shortest encoded message that is ambiguous (i.e. has more than one interpretation).

- It is worth mentioning that the problem title is slightly misleading.
- Here we really mean finding the shortest ambiguous *message*; whether a (symbol) code is ambiguous or not only depends on the codeword of each distinct symbol (as we have previously seen).

M0413 - Ambiguous Code

- We will use Sardinas-Patterson Algorithm (how surprising!).

M0413 - Ambiguous Code

- We will use Sardinas-Patterson Algorithm (how surprising!).
- But how do we find *any* encoded message that is ambiguous?
  Recall that in Sardinas-Patterson, we only keep track of a set of dangerous *suffixes* $suf_1, suf_2, \ldots, suf_k$. But we can actually store some more information along with each dangerous suffix suf_i, such as a single shortest valid encoded message w_{i1}, such that $w_{i2} = w_{i1} + suf_i$ is also a valid encoded message.

M0413 - Ambiguous Code

- We will use Sardinas-Patterson Algorithm (how surprising!).
- But how do we find *any* encoded message that is ambiguous? Recall that in Sardinas-Patterson, we only keep track of a set of dangerous *suffixes* $suf_1, suf_2, \ldots, suf_k$. But we can actually store some more information along with each dangerous suffix suf_i, such as a single shortest valid encoded message w_{i1}, such that $w_{i2} = w_{i1} + suf_i$ is also a valid encoded message.
- But how do we find the shortest ambiguous message? We can establish an order of processing the pairs (suf_i, w_{i1}) that are currently in the dangerous set. For example, at each iteration we process the pair that minimizes $|suf_i| + |w_{i1}|$, i.e. the length of the longer of the two encoded messages, with common prefix $|w_{i1}|$ and dangerous suffix suf_i.

M0413 - Ambiguous Code

- We will use Sardinas-Patterson Algorithm (how surprising!).
- But how do we find *any* encoded message that is ambiguous?
  Recall that in Sardinas-Patterson, we only keep track of a set of dangerous *suffixes* $suf_1, suf_2, \ldots, suf_k$. But we can actually store some more information along with each dangerous suffix suf_i, such as a single shortest valid encoded message w_{i1}, such that $w_{i2} = w_{i1} + suf_i$ is also a valid encoded message.
- But how do we find the shortest ambiguous message? We can establish an order of processing the pairs (suf_i, w_{i1}) that are currently in the dangerous set. For example, at each iteration we process the pair that minimizes $|suf_i| + |w_{i1}|$, i.e. the length of the longer of the two encoded messages, with common prefix $|w_{i1}|$ and dangerous suffix suf_i.
- This way the first ambiguous message we find will also be the shortest.

M0413 - Ambiguous Code

Example implementation here

Sardinas-Patterson Algorithm: Comments

- The algorithm runs in polynomial time. By using a suffix tree, the time for the algorithm can be bounded by $O(nk)$, where n is the total length of the codewords and k is the number of codewords.
- If you are interested, refer to the paper in the references section. We will not cover the complexity analysis here.
- Ambiguous codes and prefix codes do not require Sardinas-Patterson Algorithm to determine unique decodability. Specifically, ambiguous codes must be not uniquely decodable, whilst prefix codes must be uniquely decodable.

Review: General Goals of Data Compression

- Recall that **correctness** and **minimizing expected length of the encoded message** are two of the main goals of data compression.
- We will now discuss the expected length of the encoded message,
- Key question: Given that we can design the symbol code however we like, how can we minimize the expected length of the encoded message?

Expected Length of Encoded Message

- For a code $C = \{c_1, \ldots, c_n\}$ with codeword lengths $\ell_i = |c_i|$ and symbol probabilities p_i, the expected length is

$$\mathbb{E}[L] = \sum_{i=1}^{n} p_i \ell_i \quad \text{bits.}$$

- For symbols that appear more often, their codeword lengths contribute more to the expected encoded message length.
- We want to design a **prefix** code that minimizes $\mathbb{E}[L]$.
- Recall that a prefix code is one in which no codeword is a prefix of another. It gives advantages such as unambiguous decoding and low decoding cost.

Huffman Coding: High–Level Idea

- Huffman Coding gives a prefix code which minimizes $\mathbb{E}[L]$.
- Idea: Build the code *tree* bottom–up by repeatedly combining the two least–probable symbols.
- Each combination creates an internal node whose probability is the sum of its children.
- When the tree is complete, assign 0 to every left edge and 1 to every right edge.
- The label of a leaf is then the codeword for that symbol.

Huffman Coding: Step-by-Step Algorithm

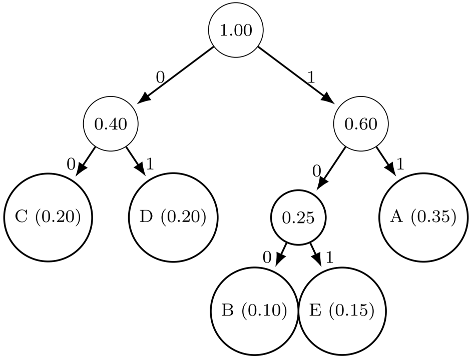
1. **Initial list.** Start with a multiset of leaf nodes, one per symbol, weighted by probability.
2. **Merge.** Repeat until one node remains:
    1. Remove the two nodes with the smallest probabilities p_{min_1} and p_{min_2}.
    2. Create a new parent node whose weight is their sum $p' = p_{\text{min}_1} + p_{\text{min}_2}$.
    3. Insert this node back into the multiset.
3. **Label edges.** Assign $0/1$ to the edges on each level (conventionally left=0, right=1).
4. **Read off codewords.** Concatenate the edge labels from root to leaf.

Complexity: $\mathcal{O}(n \log n)$ with a binary heap.

Worked Example

Symbol	Probability
A	0.35
B	0.10
C	0.20
D	0.20
E	0.15

1. Merge B and E ($0.10 + 0.15 = 0.25$).
2. Merge C and D ($0.20 + 0.20 = 0.40$).
3. Merge (B,E) with A ($0.25 + 0.35 = 0.60$).
4. Merge (C,D) with (A,B,E) ($0.40 + 0.60 = 1.00$).



Resulting code:

A	11
B	100
C	00
D	01
E	101

Worked Example

Analysis:

A	11
B	100
C	00
D	01
E	101

- In the above example,
  $\mathbb{E}[L] = 0.35 \cdot 2 + 0.10 \cdot 3 + 0.20 \cdot 2 + 0.20 \cdot 2 + 0.15 \cdot 3 = 2.25$ bits/symbol.

Huffman Coding: Takeaways

- Expected length $\mathbb{E}[L]$ is the key quantity we want to minimize.
- Huffman coding achieves the minimum $\mathbb{E}[L]$ among all prefix codes.

Table of Contents

Noiseless vs. noisy channels

- A **noiseless** channel is ideal: every transmitted symbol arrives exactly as sent. We don't really need any redundant information, since all the transmitted information is correct anyway.
- A **noisy** channel, by contrast, occasionally flips or loses symbols. Any **single** error could scramble the entire message (e.g. the grader can change at most $1/4$ of the bits Alice sends to Bob...)

Motivation: the noisy-channel problem

- It refers to the challenge of transmitting information accurately when the communication channel introduces errors or noise
- *Error-correction coding*: Add carefully designed redundancy so that the original data can still be recovered (with high probability), even when the channel introduces errors.
- A **block code** is specified by two integers (n, k). It maps a block of k bits to an n-bit codeword. The $n - k$ **redundancy** bits are chosen so that error correction is possible.

A Trivial Example: $(n, 1)$ Block Code (Repetition Code)

- We can use n bits to transmit 1 bit.
- Simply repeat the bit n times, so assuming n is large enough and the probability of error is small enough, we can simply use the majority bit as the answer.
- Of course, this is rather space-inefficient.

Hamming Codes

Hamming codes can correct a **single** bit error.

Binary Hamming code

A binary Hamming code is usually written (n, k) where:

$$n = 2^r - 1, \qquad r + k = n$$

- r is the number of parity bits ("redundant data").
- If we index the positions $1, 2, \ldots, n$, then we should place parity bits at powers of 2 positions, i.e. $1, 2, 4, 8, \ldots$.
- The remaining slots hold the k data bits.

Put simply, The (n, k) Hamming code uses n bits in total to hold k data bits; if a single data bit is corrupted, the error can be detected and corrected.

Hamming codes: Encoding, Decoding

Encoding

For each parity bit p_j (indexed starting from 1, at position 2^{j-1}) compute the XOR of all positions whose binary index has the j-th least-significant bit set.

Decoding

On reception, recompute the parity checks to obtain the **syndrome** s (an r-bit number). If $s = 0$ no error occurred; otherwise the integer value of s equals the index of the corrupted bit, which can then be flipped to repair the codeword.

$(7,4)$ Hamming Code: Worked Example

Assume we want to send the data word `1011`.

Step 1: Reserve Positions & Drop the Data Bits

Position	1	2	3	4	5	6	7
Contents	p_1	p_2	d_1	p_3	d_2	d_3	d_4
Bit			**1**		**0**	**1**	**1**

- Positions that are powers of two (1, 2, 4) hold parity bits.
- Fill the remaining slots left-to-right with the data word `1011`.

$(7,4)$ Hamming Code: Worked Example

Assume we want to send the data word `1011`.
Step 2: Compute Each Parity Bit (even parity)

Parity	Positions it checks	# of 1's	Value of p_i
p_1	1, 3, 5, 7	2 (even)	0
p_2	2, 3, 6, 7	3 (odd)	1
p_3	4, 5, 6, 7	2 (even)	0

$$\boxed{p_1 = 0,\ p_2 = 1,\ p_3 = 0}$$

$(7, 4)$ Hamming Code: Worked Example

Assume we want to send the data word `1011`.
Step 3: Assemble the 7-bit Codeword

Position	1	2	3	4	5	6	7
Bit	0	1	1	0	0	1	1

Codeword $= \boxed{0110011}$

$(7,4)$ Hamming Code: Worked Example

Assume we want to send the data word `1011`.

Step 4: Quick Check —Using the Syndrome

- On reception, recompute the three parities.
- The three-bit *syndrome* gives the binary index of a single errored bit (e.g. syndrome $101_2 = 5$ means "flip position 5").
- If all parity checks are zero, the codeword is assumed error-free.

$(7,4)$ Hamming Code: Worked Example

Assume we want to send the data word `1011`.

Step 4: Quick Check —Using the Syndrome

Example

Let's say the received word is 0110111.

Parity	Positions it checks	# of 1's	Value of p_i
p_1	1, 3, 5, 7	3 (odd)	1
p_2	2, 3, 6, 7	4 (even)	0
p_3	4, 5, 6, 7	3 (odd)	1

$$p_1 = 1,\ p_2 = 0,\ p_3 = 1$$

Syndrome = $(p_3p_2p_1)_2 = 101_2 = 5$. Hence, position 5 is the unique error bit.
We then recover the codeword to be 0110011, so the correct data is 1011.

Error Correction in Communication Tasks

Review of Communication Tasks (taken from CAST (II) (2025)):

- Generally, you have to write two subprograms (or two modes) to collaboratively solve a problem.
- Judging flow:
  A: source input $\rightarrow$ program mode A $\rightarrow$ output A
  B: input based on output A $\rightarrow$ program mode B $\rightarrow$ final output

The means of communication are limited in some way.

- Limited in amount of data sent
- Data may be reordered (e.g. IOI 2011 - Parrots)
- Data may be corrupted (e.g. IOI 2019 Practice - Data Transfer)
- Data must be in some specified form (e.g. M2332 - Collaborative Sudoku)

Error Correction in Communication Tasks

- We will focus on a specific type of communication task: the Encoding-Decoding type, which consists of two parts
- ”Encoding” by the first pass, and ”decoding” by the second pass.
- It seems to be simple in theory, but problem setters get very creative when adding restrictions that make simple things hard.
- Communication tasks frequently hide a ”noisy channel” twist. You usually need to guarantee that certain information from the first pass can be recovered in the second pass despite the jury trying to ”mess with” your data.
- We will look at some examples.

IOI 2019 Practice - Data Transfer

This has been discussed in CAST (II) (2025), we will not go into too much detail today.

Abridged Statement

Alice has N bits of data, she should attach K bits on the end and send to Bob.
At most one bit of the $N + K$ bits might be corrupted.
Bob needs to recover the original N bits.
$N = 255, K \leq 9$ for full marks.

It should be obvious that this task is actually testing you to come up with some Hamming Code!

IOI 2019 Practice - Data Transfer

- Solution 1: Inspired by the Hamming Code we see earlier, use 8 parity bits $p_1, p_2, \ldots, p_8$, where

$$p_1 = data_1 \oplus data_3 \oplus data_5 \oplus \ldots \oplus data_{255}$$

$$p_2 = data_2 \oplus data_3 \oplus data_6 \oplus data_7 \oplus \ldots \oplus data_{255}$$

and so on.

IOI 2019 Practice - Data Transfer

- Solution 1: Inspired by the Hamming Code we see earlier, use 8 parity bits $p_1, p_2, \ldots, p_8$, where

$$p_1 = data_1 \oplus data_3 \oplus data_5 \oplus \ldots \oplus data_{255}$$

$$p_2 = data_2 \oplus data_3 \oplus data_6 \oplus data_7 \oplus \ldots \oplus data_{255}$$

  and so on.
- In the second pass, check if parity bit equals the actual XOR sum of those data bits. If XOR-sum of data bits == parity bit: NO corruption (note: at most 1-bit of corruption in this task). Else: we can detect the corruption with the signature!

IOI 2019 Practice - Data Transfer

- Solution 1: Inspired by the Hamming Code we see earlier, use 8 parity bits $p_1, p_2, \ldots, p_8$, where

$$p_1 = data_1 \oplus data_3 \oplus data_5 \oplus \ldots \oplus data_{255}$$

$$p_2 = data_2 \oplus data_3 \oplus data_6 \oplus data_7 \oplus \ldots \oplus data_{255}$$

  and so on.
- In the second pass, check if parity bit equals the actual XOR sum of those data bits. If XOR-sum of data bits == parity bit: NO corruption (note: at most 1-bit of corruption in this task). Else: we can detect the corruption with the signature!
- Hmm... but what if the parity bits themselves are corrupted?

IOI 2019 Practice - Data Transfer

- Solution 2: Add 1 more bit denoting the parity of $p_1 + p_2 + \ldots + p_8$. (9 bits used in total)

IOI 2019 Practice - Data Transfer

- Solution 2: Add 1 more bit denoting the parity of $p_1 + p_2 + \ldots + p_8$. (9 bits used in total)
- If parity of sum of parity bits $\neq$ that final bit: Corruption with one of the 9 extra bits, data is not corrupted

IOI 2019 Practice - Data Transfer

- Solution 2: Add 1 more bit denoting the parity of $p_1 + p_2 + \ldots + p_8$. (9 bits used in total)
- If parity of sum of parity bits $\neq$ that final bit: Corruption with one of the 9 extra bits, data is not corrupted
- Else: the 9 extra bits are not corrupted, we can detect the corruption with the signature!

JOI Spring Camp 2017 - Broken Device

- Anna wants to send a 60-bit integer X to Bruno. She has a device that can send a sequence of 150 numbers that are either 0 or 1.
- The twist is that K positions of the device $(K \leq 40)$ are **broken** and can only send 0.
- Anna knows X and the K broken positions. Bruno receives the sequence Anna sent, but he does not know the broken positions.
- Bruno must decode the value of X. Partial score based on the maximum K (number of broken positions) that Anna and Bruno can handle.
- Note: Not strictly ”error correction” in the same sense as before, but in this problem we would want to *make use of errors* as part of our information. We will shortly see how.

Solution 1: $K = 0$

- You need to know how to solve this case in order to solve the full problem.
- The idea is very simple: send the binary encoding of X. We will use the binary encoding in all other subtasks, but you need to keep this idea in mind.
- We use the first 60 positions to send the encoding and leave the other 90 unused.

Solution 2: $K = 1$

- Because there is a broken place, we might not be able to send X using the 60 first positions. However, it will not be difficult to overcome that.
- Let's introduce some extra data.
- The key idea is to send a signal meaning that this is the place the binary representation starts. That is, the first number 1 of the sequence will mean "The following 60 numbers are the binary encoding of X".
- In order to use this idea, we need to be sure that there will be 61 adjacent positions that are not broken. Because $K = 1$, we know that this is true.

Solution 3: $K = 15$

- Split the 150 positions into 75 buckets of size 2.
- Our encoding will use every bucket to send at most one bit. Notice that sometimes we may not send any information because of the broken places, but that is okay.
- In some cases, we are actually making use of the broken bits to transmit information!
- Encoding table per bucket:

Bits in bucket	Encoded bit
00	$\emptyset$
01	$\emptyset$
10	0
11	1

Solution 4: Intermediary K

- Previously, only the patterns 10 and 11 encode any meaning at all.
- The easiest improvement is to give meaning to pattern 01 to transmit extra info (e.g. repeat last bit).
- Randomisation tricks to avoid worst case of the algorithm.

Solution 5: $K = 40$

- It turns out that we can split the 150 positions into 50 buckets of size 3 instead.
- We want to encode up to 2 bits for each bucket.
- Even if $K = 40$ buckets lose 1 bit each, we still have $(50 - 40) \cdot 2 + 40 \cdot 1 = 60$ bits.
- Encoding table per bucket (needs some clever thinking):

Bits in bucket	Encoded bit
000	$\emptyset$
001	1
010	0
011	10
100	00
101	01
110	1
111	11

Takeaways for Communication Tasks

- This problem is very much an error-correction coding problem, albeit not the classic Hamming code problem.
- Being able to design ways to recover data in face of errors, introduce redundancy to transmit information accurately, or even making use of errors as part of information, might give you a large competitive edge in communication tasks.

Table of Contents

The Activity

- You will be given a single OI problem.
- Work in teams of 5 to 6.
- Write down your solution on paper, and briefly justify its correctness.
- Time limit: 30 minutes. Submit your solutions to me after that.
- 1-2 teams with the highest points (depending on correctness, number of queries) receive cards (number TBC).

The Problem: Coin Weighing

Statement

There are N coins. Exactly one of them is defective - it is slightly heavier or slightly lighter than the remaining coins. You have a balance scale. Every time you can pick some coins to place on both sides of the balance scale, then you will know which side is heavier or lighter (or two sides have the same weight). Identify which coin is the defective coin, and whether it is heavier or lighter than the remaining coins. **Minimize the expected number of times you use the balance scale** (don't worry much about constants, e.g. +1 vs +c is the same).

Hints

- Color the coins that might be lighter red, the coins that might be heavier blue.

Hints

- Color the coins that might be lighter red, the coins that might be heavier blue.
- Say there are x red coins and x blue coins. Can you reduce the search space to $x/3$ red and $x/3$ blue using 1 query only? (Hint: It is possible)

Problem List

- M0413 Ambiguous Code
- N1521 荷馬史詩
- IOI 2011 - Parrots
- IOI 2017 Practice - Coins
- IOI 2019 Practice - Data Transfer
- JOI Spring Camp 2017 - Broken Device
- JOI Spring Camp 2018 - Airline Route Map
- JOI Spring Camp 2023 - The Last Battle
- 洛谷 P9477 猜數
- 洛谷 P8079 猜詞

Acknowledgements

- `kevinxiehk` for proofreading this slide deck and providing many resources.
- `mtyeung1` for the Group Activity idea.

References

There are lots of resources for information theory! Plus if you want to learn more of the mathematics behind today's content, these links can be useful :)

- Constructive Algorithms & Special Tasks (II) (2025) https://assets.hkoi.org/training2025/cast-ii.pdf
- Course pages: Information Theory (2024-25) | University of Cambridge | https://www.cl.cam.ac.uk/teaching/2425/InfoTheory/
- B8.4 Information Theory Notes (2024-25) | University of Oxford | https://courses.maths.ox.ac.uk/pluginfile.php/105214/mod_resource/content/12/notes.pdf
- Solving wordle using information theory –https://www.youtube.com/watch?v=v68zYyaEmEA
- But what are Hamming codes? The origin of error correction –https://www.youtube.com/watch?v=X8jsijhllIA
- https://www.ruanyifeng.com/blog/2019/08/information-theory.html
- Huffman Codes: An Information Theory Perspective –https://www.youtube.com/watch?v=B3y0RsVCyrw
- Information Theory | Ivan Carvalho https://ivaniscoding.github.io/categories/information-theory/
- https://www.cl.cam.ac.uk/teaching/2425/ForModLang/slides/all_slides.pdf
- Formal Models of Language - Language as Information Notes https://www.cl.cam.ac.uk/teaching/2425/ForModLang/notes/Language_as_information_notes.pdf
- https://ieeexplore.ieee.org/document/1056535/