

Lecture 8

Information Redundancy - Part II

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
 - An analogy on error detection and correction codes
- Hamming codes
 - Basic intuition
 - Error detection
 - Error correction
 - Information rate
- Extended Hamming codes
- TODOs

Encoding and decoding

- Encoding: transforms data into code word



- Decoding: transforms code word back to data



- There is a difference in length between the codeword and data.
- This difference gives us the number of check bits which are required to make the encoding (for separable codes).

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happen, 000 becomes 001

Analogy

Dictionary: {accept, accent}

Distance = 1

Typed word: accept

A typo happens, accept becomes accent

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

Analogy

Dictionary: {accept, accent}

Distance = 1

Typed word: accept

A typo happens, accept becomes accent

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell there is an error in 001.

Analogy

Dictionary: {accept, accent}

Distance = 1

Typed word: accept

A typo happens, accept becomes accent

We cannot tell it is a typo.

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We can tell that 001 is an error (not a codeword in the code space)

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We can tell it is a typo.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell how to correct 001 (000 or 101).

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We cannot tell which word is mistyped.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell how to correct 001 (000 or 101).

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We cannot tell which word is mistyped.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 111}

$$C_d = 3$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We can correct 001 to 000 (closest).

Analogy

Dictionary: {except, exception}

Distance = 3

Spelling word: except

A typo happens, except becomes eccept

We can correct the typo.

Schedule for today

- Key concepts from last class
 - An analogy on error detection and correction codes
- **Hamming codes**
 - Basic intuition
 - Error detection
 - Error correction
 - Information rate
- Extended hamming codes

Hamming codes

Hamming code is a linear code that can detect and correct single-bit errors.

- Why? To detect and correct errors while sending less redundant data (than repetition codes).

Basic intuition: if we apply multiple parity checks to some carefully selected subsets of the data, we can pin down the location of any single-bit error.


A **linear code** is an error-correcting code for which any linear combination of codewords is also a codeword.

Logic behind Hamming codes

The overall idea is have the parity checks as a series of questions that help us detect and locate the single-bit error.

- Each question narrows down the search space
- Analogous to Sudoku
 - Rule for each row: comprise the numbers 1-9
 - Rule for each column: comprise the numbers 1-9
 - ...
- Hamming codes
 - Parity check for some specific groups of bits: odd parity
 - ...

(15, 11) Hamming code

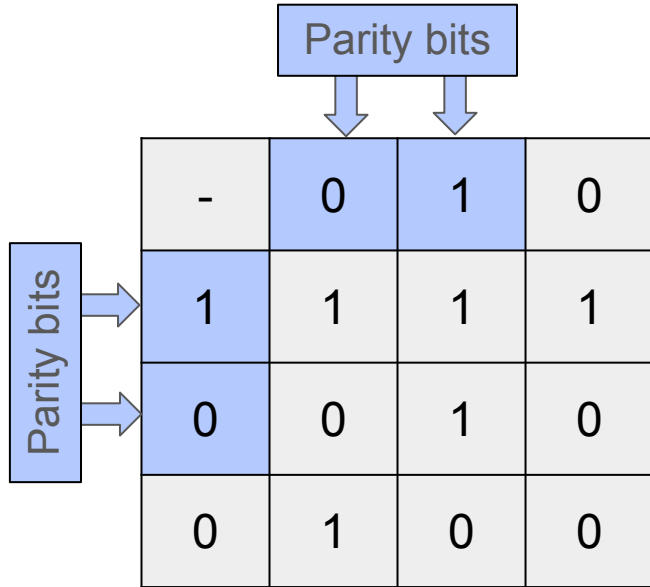
Not used			
			
-	0	1	0
1	1	1	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Definition of (15, 11)

- 15 bits to encode 11 bits of data
- The remaining 4 bits are parity bits
- The first position is not used

(15, 11) Hamming code



(15, 11) Hamming code

Definition of (15, 11)

- 15 bits to encode 11 bits of data
- The remaining 4 bits are parity bits
- The first position is not used

Position of the parity bits

- Set at position $\{2^n\}$, starting from $n = 0$
- Example
 - Parity bit #1: position 1
 - Parity bit #2: position 2
 - Parity bit #3: position 4
 - Parity bit #4: position 8

0	1	2	3
4	5	6	7
8	9	10	11
11	12	13	14

Positions of the bits

Error detection



-			0
	1	1	1
	0	1	0
0	1	0	0

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Step 1: calculate parity bit #1 at position 1

- We check the parity of the bits in the second and last columns.
- The bits {0 1 1 0 0 1 0} contain three “1”s.
- That is an odd number of “1”s.
- Therefore, we set the **first parity bit to 1** to make up for an even parity.

Error detection

-	1		0
	1	1	1
	0	1	0
0	1	0	0

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Step 2: calculate parity bit #2 at position 2

- We check the parity of the bits in the third and last columns.
- The bits {0 1 1 1 0 0 0} contain three “1”s.
- That is an odd number of “1”s.
- Therefore, we set the **second parity bit to 1** to make up for an even parity.

Error detection

-	1	1	0
	1	1	1
	0	1	0
0	1	0	0

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Step 3: calculate parity bit #3 at position 4

- We check the parity of the bits in the second and last rows.
- The bits {1 1 1 0 1 0 0} contain four “1”s.
- That is an even number of “1”s.
- Therefore, we set the **third parity bit to 0** as it is already an even parity

Error detection

-	1	1	0
0	1	1	1
	0	1	0
0	1	0	0

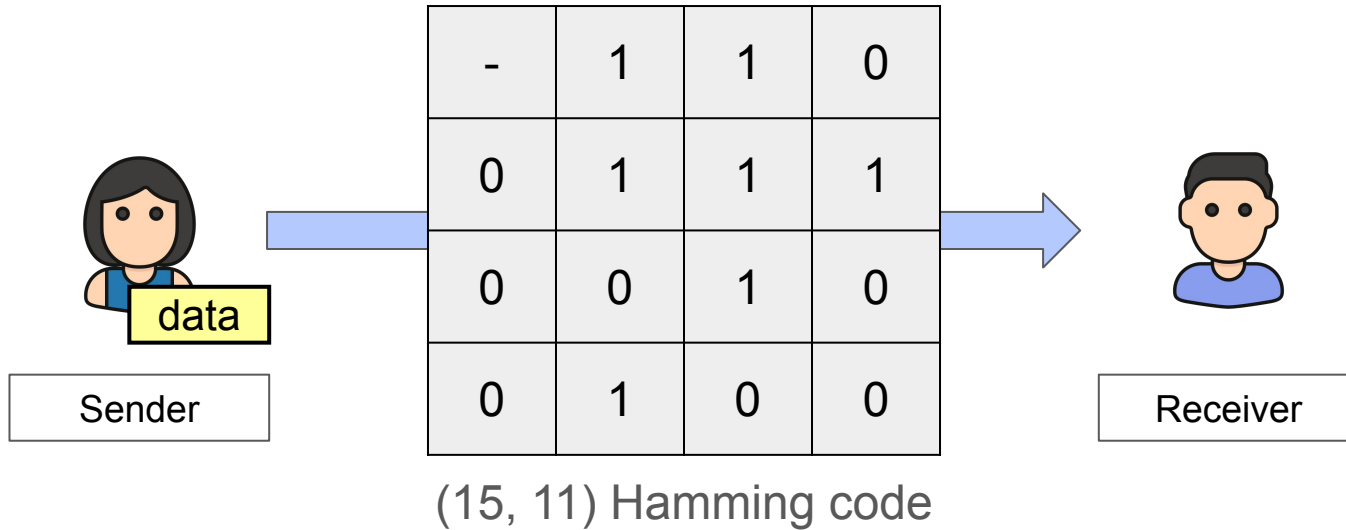
Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Step 3: calculate parity bit #4 at position 8

- We check the parity of the bits in the third and last rows.
- The bits {0 1 0 0 1 0 0} contain two “1”s.
- That is an even number of “1”s.
- Therefore, we set the **last parity bit to 0** as it is already an even parity

Error detection



Error detection



-	1	1	0
0	1	1	1
0	0	1	0
0	1	0	0

Parity check for parity bit at position 1

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: No error

The receiver repeat the same process.

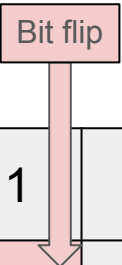
Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 1 1 1 0 0 0}, even parity, no error

Parity bit at position 4: {0 1 1 1 0 1 0 0}, even parity, no error

Parity bit at position 8: {0 0 1 0 0 1 0 0}, even parity, no error

Error detection



-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 0 1 1 0 0 0}, odd parity, **error**

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 0 1 1 0 0 0}, odd parity, **error**

Parity bit at position 3: {0 1 0 1 0 1 0 0}, odd parity, **error**

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 0 1 1 0 0 0}, odd parity, **error**

Parity bit at position 3: {0 1 0 1 0 1 0 0}, odd parity, **error**

Parity bit at position 8: {0 0 1 0 0 1 0 0}, even parity, no error

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error



-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Detection and correction by elimination

- First parity check: No error in the second and last columns.
- Second parity check: Error is somewhere in the last two columns.
- Intuition: the error must be in the third column.

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error



-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Detection and correction by elimination

- Third parity check: Error is somewhere in the second and last rows.
- Fourth parity check: No error in the third and last row.
- Intuition: Error must be at position 6.

Parity checks

The number of parity checks is based on the number of data bits.

- Every parity check reduces the possibility of an error occurring at a certain location by half.
- k bits = 2^n bits, requires n parity checks
- Examples:
 - E.g., 4 bits = 2^2 bits, 2 parity checks
 - E.g., 16 bits = 2^4 bits, 4 parity checks
 - E.g., 256 bits = 2^8 bits, 8 parity checks

Information rate

The information rate improves as we deal with larger block size.

- Bigger information rates produce stronger codes.
- Examples:
 - E.g., 16 bits, 11 data bits, 4 parity bits, information rate = $11/15 \approx 73\%$
 - E.g., 128 bits, 121 data bits, 6 parity bits, information rate = $121/128 \approx 95\%$
 - E.g., 256 bits, 247 data bits, 8 parity bits, information rate = $247/255 \approx 97\%$
 - E.g., 516 bits, 505 data bits, 10 parity bits, information rate = $505/515 \approx 98\%$
 - ...

The information is the ratio k/n , where

- k is the number of data bits
- n is the number of data bits + parity bits

Information rate

Data bits	Parity checks	Hamming codes
16	4	73%
128	6	95%
256	8	97%
516	10	98%
...
1,048,576	20	Nearly 100%

One million bits of data only requires 20 parity checks to locate the error.

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Parity bit at
position 1

Binary

0

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Parity bit at
position 1

Parity bit at
position 2

Binary

1	0
---	---

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 1

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 2

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 4

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 8

Binary

0	1	1	0
---	---	---	---



6


Decimal value

8	4	2	1
---	---	---	---

Decimal

What happens if there is an error on a parity bit?



				Bit flip
-	1	0	0	
0	1	1	1	
0	0	1	0	
0	1	0	0	

(15, 11) Hamming code
using even parity

-			

Parity
check
1

-			

Parity
check
2

-			

Parity
check
3

-			

Parity
check
4

Schedule for today

- Key concepts from last class
 - An analogy on error detection and correction codes
- Hamming codes
 - Basic intuition
 - Error detection
 - Error correction
 - Information rate
- Extended Hamming codes
- TODOs

Extended Hamming codes

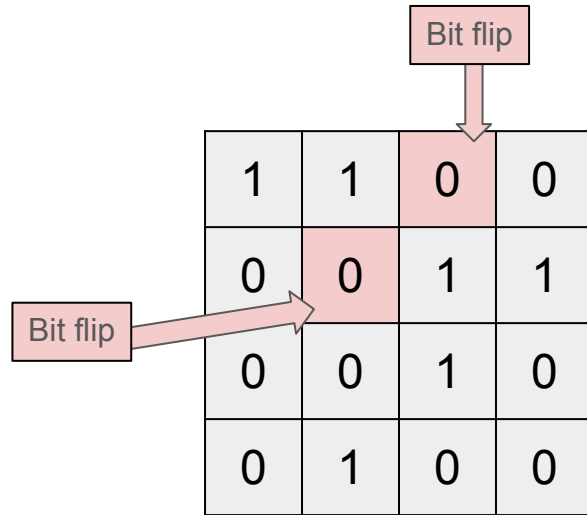
Extended Hamming code is a linear code that can detect and correct single-bit errors, and also detect double-bit errors.

- Uses an extra parity check for the whole block of bits
- For example, parity check on {1100 111 0010 0100}

1	1	1	0
0	1	1	1
0	0	1	0
0	1	0	0

Extended Hamming code (even parity)

Extended Hamming codes



Extended Hamming code
(even parity)

- There are six 1s in the whole block.
- That is an even number of 1s, the parity check at position 0 passes.
- However, the other parity checks (at position 1, 2 and 4) detect an error.
- Therefore, there are at least two errors.

TODOs

- For the deliverable, please don't forget to add your team and team member names (a title page).