Week 2 Discussion

Wednesday, 10/9/19

Reminders

PSA1 due Tuesday, October 15 11:59pm

Quiz 1 in-class today (October 9)

Today’s agenda

What is Hamming Code?

String vs StringBuilder

Debugging tips and tricks

Background review: bits

Bits: Smallest piece of information 1 or 0

5, as a 4 bit binary representation, is 0101

What is hamming encoding?

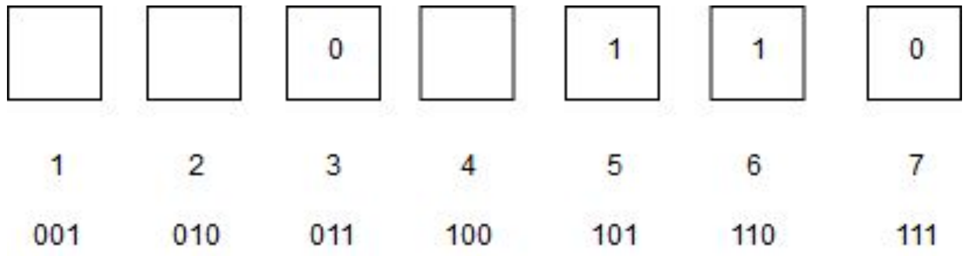
We are doing (7,4) hamming code

- Meaning using 7 bits to encode 4-bit info

Parity bit - Bit that is used as error flag

4 bits are message bits, 3 bits are parity bits

- 7 bits is formatted “ppmpmmm” (p: parity, m: message)



Why is it useful?

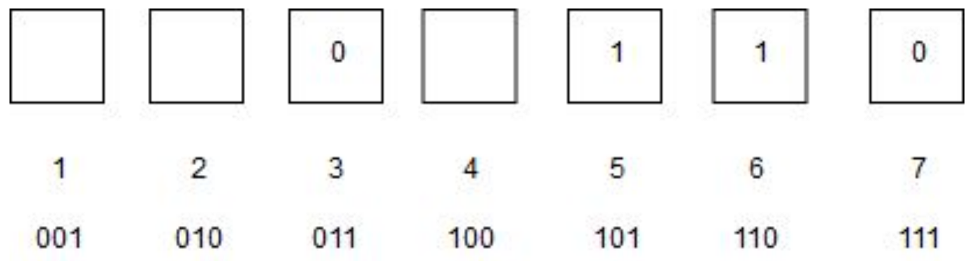
- When data is transmitted potential for errors to appear
- Allows us to detect and revert those 1-bit changes

PSA 1 parity bit

`parityHelper()` will return what goes in each parity bit position (1, 2, and 4)

Why(1, 2 and 4)?

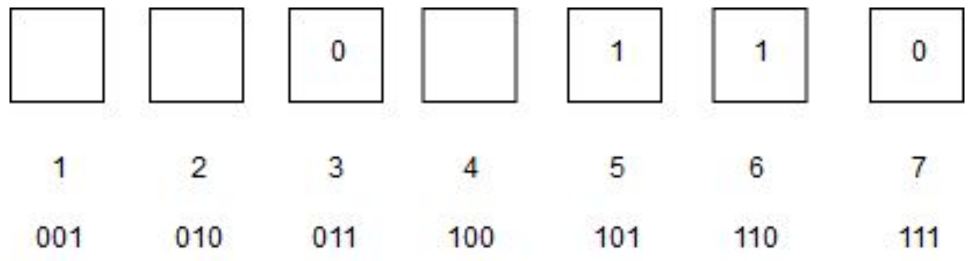
- All are power of 2 (2^0, 2^1, 2^2)
- Cross check all other indices.



PSA 1 parity bit

`parityHelper()` will return what goes in each parity bit position (1, 2, and 4)

Parity Bit	Bit Position to Count
1	3,5,7
2	3,6,7
4	5,6,7

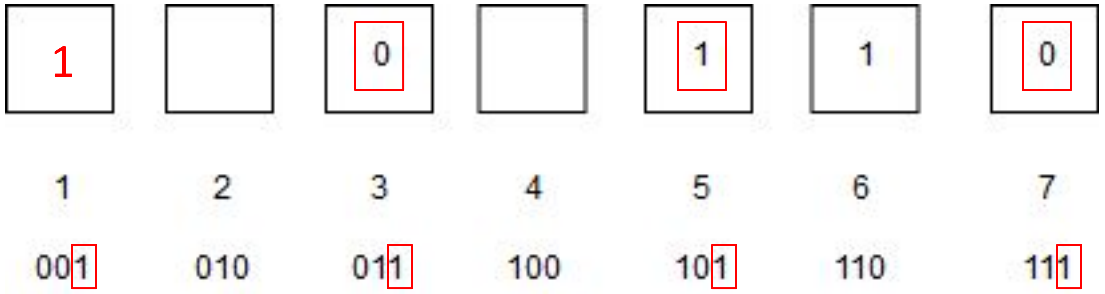


PSA 1 parity bit (1)

Check how many 1s are present in current sequence in positions 3, 5, and 7

If the amount is even, mark the parity bit as 0. Else, mark it as 1.

Parity Bit	Bit Position to Count
1	3,5,7
2	3,6,7
4	5,6,7



PSA 1 Hamming encoding

Given a 4 bit message

Store message bits in 7-bit sequence in positions 3, 5, 6, and 7

Save positions 1, 2, and 4 as parity bit positions

Ex. Store bit sequence 1010

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

PSA 1 Hamming decoding

Input: 1,0,1,1,0,1,1 -> extract 4-bit message as 1011

Parity Bit	Bit Position to Count
1	3,5,7
2	3,6,7
4	5,6,7

Check parity bits and track incorrect ones:

- Parity bit 1:
    - Should be 0 but is listed as 1, track 1
- Parity bit 2:
    - Should be 1 but is listed as 0, track 2
- Parity bit 4:
    - Should be 0 but is listed as 1, track 4

Add all incorrect parity bits: 1 + 2 + 4 = 7, bit 7 is incorrect so flip it

1,0,1,1,0,1,1 -> 1,0,1,1,0,1,**0** with correct 4-bit message as 1010

PSA 1 error detection

If one of the message bits is flipped by mistake, we can recover the original message sequence

Get the message sequence from output

Find the parity bits based off the message sequence (like before)

Track the bit positions of parity bits that are incorrect from output

Add positions to find bit position that needs to flipped in message sequence

If sum is 0, no need to change anything in sequence because it is correct

Worksheet

PSA 1 String vs StringBuilder

String

- Immutable, needs to be copied to new String to add characters to it
- Slow as a result

StringBuilder

- Allocates more space than just the amount of characters in the String
- Doesn't need to copy elements to new String everytime char is added to it
- Faster to append to it than appending to String

Debugging Tips!

- Print statements are your friend
  - Ex. System.out.println(“Code gets to this point”); or System.out.print(“Variable A: ” + A);
- Draw it out
  - Memory Models
  - Run through with examples on paper
- Run your program after every change; keep note of changes and their effects
- Check that you’re accounting for edge cases
  - Null, 0, very large/small inputs, “normal” inputs, etc.

JUnit Testing

- A way to unit test your methods!
- You will be provided with HammingTest.java
  - Follow the examples and add your own test cases
- To compile and run HammingTest.java, follow the commands at the top of the file

Assuming methodName() is a public static method in ClassName:

```
assertEquals( 10, ClassName.methodName() );
```

expected value

actual value