# CS 212 PA #6

*Note: This assignment has been adapted from Stanford’s list of Nifty Assignments.*

## Introduction

In this assignment, you will write a program that encoded and decodes *Hamming Codes*. Hamming codes are a very clever way of ensuring data integrity during data transmission. [Wikipedia has a fantastic article](https://en.wikipedia.org/wiki/Hamming(7,4)) on Hamming Codes, which I recommend that you read. Much of the background content in this writeup has been pulled from that article.

## The Problem

Transmitting data is a risky endeavor. If just one bit gets mangled, the integrity of the entire package may be compromised. To address this issue, Dr. Richard Hamming invented a system that can correct a single-bit error per byte and detect two-bit errors per byte. Here’s how it works:

Each 4-bit sequence is complimented with 3 additional *parity* bits (7 bytes total). Note that in some cases in our program, we will expand this into a fully byte by adding an additional 8th dummy bit for the sake of simplicity. Note that an expanded version of our Hamming Code actually uses the 8th bit for an additional integrity check, but we will always just leave it zeroed out. Below is a table that provides a "byte" mapping (sans dummy bit):

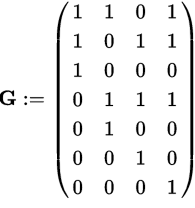
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit # | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Transmitted bit |  |  |  |  |  |  |  |
|  | Yes | No | Yes | No | Yes | No | Yes |
|  | No | Yes | Yes | No | No | Yes | Yes |
|  | No | No | No | Yes | Yes | Yes | Yes |

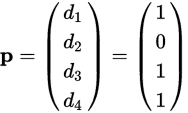
This table provides a meaning to each bit (1-7) as well as indicating which parity bits are responsible for maintaining data integrity. For example, the integrity of data at bit 3 () is managed by parity bits 1 and 2. As you can see, each data bit is monitored by exactly two parity bits. By having two checks per data bit, the system guarantees that it can recover from a single failure. Here's another image that provides a graphical representation of the above table:

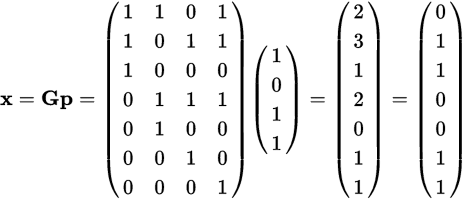
## 

## Encoding Data

Encoding data into the Hamming Code format requires a bit of matrix multiplication. To encode a 4-bit sequence, we take the sequence as a 4-element array, conceptualized as a 4x1 vector and multiply it by the following *code generator matrix*:

For example, assume that we have the 4-bit sequence {1, 0, 1, 1}, this translates into the vector P:

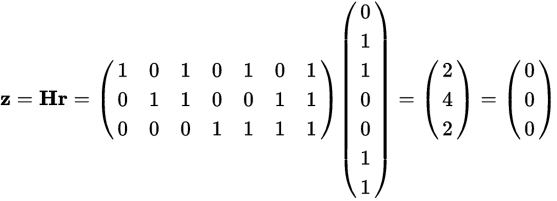
To encode our data, we multiply G by P, which yields X. Because we are working at the bit level, our result X is modded (%) by 2:



Thus, when encoded using Hamming Codes, the sequence {1, 0, 1, 1} is converted into the sequence {0, 1, 1, 0, 0, 1, 1}.

## Decoding Data

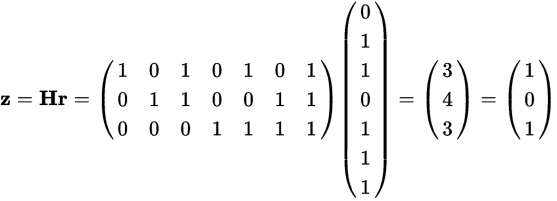
To decode a Hamming Code, we multiply the received code (R) by the *parity check matrix* (H), again modding by 2 to keep the results in binary format. This process yields the 3x1 *parity vector* (Z):



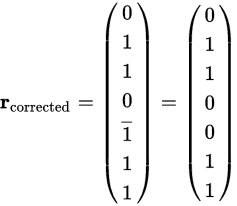
If any element in Z is non-zero, then the algorithm has detected bad data. Note that the Hamming Code algorithm that we use cannot determine *how many* data flips have occurred. Therefore, depending on application, we would either throw the data away and ask for a resend from the client or we would try to correct the error. Note that our correction will only be valid when **exactly** one error has occurred. If two errors have occurred, the attempt to correct will result in garbage data.

### Checking for Parity

To correct for an error, we simply rotate Z so that it becomes a 1x3 vector. Then, treating that 1x3 vector as a binary number, we identify and correct the invalid row. For example, assume that our decode yields the following matrix Z:



Rotating Z yields {1, 0, 1}. Treating this as a binary sequence yields the number 5. Therefore, row 5 of our received matrix has been compromised. To correct, we negate the 5th entry of R:



Having corrected for errors, we can then the table on the first page as a guide to extract the four original data bits.

## Expected Functionality

Your program should be able to both encode and decode files. Included in this repository is a completed file that encodes and decodes files using the Hamming Code algorithm that you may test your solution against.

## Header Comment, and Formatting

1. Be sure to modify the file header comment at the top of your program to indicate your name, student ID, completion time, and the names of any individuals that you collaborated with on the assignment.
2. Remember to follow the basic coding style guide. For a list of basic rules, [see my website](http://adamcarter.com/teaching/cpts121/style) or examine my example files from previous assignments and labs.

## Reflection Essay

In addition to the programming tasks listed above, your submission must include an essay that reflects on your experiences with this homework. This essay must be at least 350 words long. Note that the focus of this paper should be on your reflection, ***not*** on structure (e.g. introductory paragraph, conclusion, etc.). The essay is graded on content (i.e. it shows deep though) rather than syntax (e.g. spelling) and structure. Below are some prompts that can be used to get you thinking. Feel free to use these or to make up your own.

* Describe a particular struggle that you overcame when working on this programming assignment.
* Conversely, describe an issue with your assignment that you were unable to resolve.
* Provide advice to a future student on how he or she might succeed on this assignment.
* Describe the most fun aspect of the assignment.
* Describe the most challenging aspect of the assignment.
* Describe the most difficult aspect of the assignment to understand.
* Provide any suggestions for improving the assignment in the future.

## Deliverables

You must upload your assignment through Canvas no later than midnight on Sunday, May 12, 2019.

## Grading Criteria

Your assignment will be judged by the following criteria:

### Reflection essay (10pts)

* Your reflection meets the minimum requirements as specified earlier in this document.

### Encoding (40pts)

* Your program can encode a text file using the Hamming Code algorithm

### Decoding Tier 1 (15pts)

* Your program can decode a previously encoded file **WITHOUT** error checking

### Decoding Tier 2 (35pts)

* Your program can decode a previously encoded file **WITH** error checking

### Decoding Tier 3 (10pts)

* Upon detecting an error, your program notifies the user that the file may be corrupted and asks the user if they'd like to attempt to decode with the acknowledgement that the decode may be incorrect.