Cosc 1p03 Assignment 1

(Due date Jan 29th 16:00 est, Late date Feb 1st 16:00 est)

Error Detection and Correction

For the purpose of clarity, the educative section of this assignment will use natural numbers to identify rows and columns. When we get to the assignment section, zero based indexing will be used.

In data communications a simple but effective scheme was developed to detect errors in a bit stream. This was called parity checking. With parity checking a bit stream, often 8 bits was converted into a 9 bit stream. The 9th bit was the parity bit. Two schemes were developed, even parity and odd parity. With even parity the sum of the set bits (1's) in the stream must be an even number. With odd parity the sum of the set bits must be odd. For this assignment we will only be using even parity.

E.g.

0	0	1	1	1	0	1	0	0

The 9 bit stream above, uses even parity. There are 4 - 1's set in the data portion of the stream (black bits). To maintain even parity, the 9th bit would be set to 0. If there were an odd number of bits set in the data portion, then the parity bit would be set.

The bit stream would then be transmitted.

On the receiver end, knowing that even parity was used, the receiver would count the number of set bits, and if an even number was observed would conclude that the bit stream arrived without error. The astute would recognize that if a bit did flip during transmission then the receiver could only determine that the stream had an error but would not know which bit flipped, and thus could not correct the error. To make matters worse, if two bits flipped, the stream would pass the error check, hence a false negative. So, something better is needed.

Parity Matrix

Consider now that a bit stream is 64 bits in length. To generate the parity for this stream we pack the stream into an 8x8 matrix. Using the above concept of generating even parity, a 9th row and 9th column is added to the matrix, where each element in the 9th row represents the parity of the 8 rows of that column. The same is done for the parity column. See E.g. below.

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

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

The original 64 bit data stream is on the left. The parity matrix is on the right. Those entries in red are the parity bits for the columns and the rows. The bit in green is the parity of the red parity bits.

Once the bit stream is transmitted, the receiver will check the matrix by testing each row and column for parity, hence each row and column should have an even number of 1's.

What happens if there is an error. Let's assume that the bit in blue is a flipped bit. This means that a row and column would fail the parity check and that the intersection of the row and column would identify the flipped bit, allowing it to be corrected, thus the receiver would accept the data packet. This will work if only 1 error occurs. If more than 1 bit flipped hence 2 errors, then multiple rows and columns would fail the parity check, and thus the receiver would reject the data packet. If a bit flipped in a parity row or column (Red bits) then the green bit would show the parity error, allowing for these type of errors to be detected and corrected.

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

The Assignment

Create a 9x9 integer matrix, randomly populate rows and columns 0 to 7 so that each cell of the matrix represents a bit in the form of an integer, where the chance of it being a 1 is 40%.

Calculate the parity for each row and column. These will be row and column 8. Calculate the parity bit for the parity row and column, the green bit.

Print the matrix.

Assume that the matrix has been transmitted, where each bit has a 1.2% chance of flipping. Simulate this process by enumerating the matrix and flipping each bit based on a 1.2% chance of doing so. This will be the error rate of transmission.

Print the matrix.

Using the process described above, determine if the matrix contains a single bit error or is error free. If the matrix contains a correctable error, you are to identify the error and correct it.

Print the matrix with a note “Packet Accepted, no errors found”.

or

Print the matrix with a note “Packet Accepted, Error corrected at (x,y)”.

If the matrix contains an uncorrectable error:

Print the matrix with a note “Packet Rejected, too many errors found”.

To properly test your program, run it a total of 3 times with data size 4, 6 and 8; producing the output as described. Be sure that your program is flexible in that it can self-configure to any size matrix. That is, do not hard code any indices. Above a 9x9 matrix (data size 8) was used for example purposes, thus tempting one to identify the parity row as row 8. This should be coded as M.length-1, allowing the algorithm can use any size matrix and still work.

Friendly advice

Here are a few pointers to help you with this assignment. Although the assignment seems trivial, there are a few challenging sections.

1. Start early. This is not 1P02, and starting the night before is a certain recipe for disaster. You will need time to debug and refine your solution.
2. Printing a matrix is a common task, write a method to do this. Pass the matrix as a parameter to the method, and ensure the method uses the M.length attributes to determine row and column length.
3. Write a method to generate the row parity.
4. Write a method to generate the column parity. Pass the matrix as a parameter.
5. Write a method to generate the row, column parity bit.
6. Write a method to verify a matrix has uncorrectable errors. This should be easy and should precede a check for a correctable error.
7. Write a method to verify the matrix and correct any error, see educative section of assignment for details.
8. Did I mention start early!

Good Luck