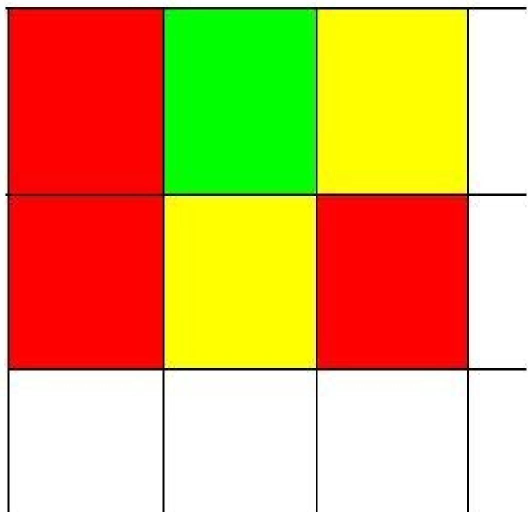
Correcting Bad Data Using Parity Bits



The first few pixels in a large image

Data is sent as a string of '1's and '0's which are then converted into useful numbers by computer programs. A common application is in digital imaging. Each pixel is represented as a 'data word' and the image is recovered by relating the value of the data word to an intensity or a particular color. In the sample image to the left, red is represented by the data word '10110011', green is represented by '11100101' and yellow by the word '00111000', so the first three pixels would be transmitted as the 'three word' string '101100111110010100111000'. But what if one of those 1-s or 0-s was accidentally reversed? You would get a garbled string and an error in the color used in a particular pixel.

Since the beginning of the Computer Era, engineers have anticipated this problem by adding a 'parity bit' to each data word. The bit is '1' if there are an even number of 1's in the word, and' 0' if there is an odd number. In the data word for red '1011001**1**' the last '**1**' to the right is the parity bit.

When data is produced in space, it is protected by parity bits, which alert the scientists that a particular data word may have been corrupted by a cosmic ray accidentally altering one of the data bits in the word. For example, Data Word A '1110001**1**' is valid but Data Word B '1111001**1**' is not. There are five '1's but instead of the parity bit being '0' ('1110001**0**'), it is '1' which means Data Word B had one extra '1' added somewhere. One way to recover the good data is to simply re-transmit data words several times and fill-in the bad data words with the good words from one of the other transmissions. For example:

Corrupted data string:	1011110**0**	101101**0**	1010101**1**	0011001**1**	1011101**0**
Good data string:	1011110**0**	100101**0**	1010101**1**	1011001**1**	1011101**0**

The second and fourth words have been corrupted, but because the string was re-transmitted twice, we were able to 'flag' the bad word and replace it with a good word with the correct parity bit. Cosmic rays often cause bad data in hundreds of data words in each picture, but because pictures are re-transmitted two or three times, the bad data can be eliminated and a corrected image created.

Problem: Below are two data strings that have been corrupted by cosmic ray glitches. Look through the data (a process called parsing) and use the right-most parity bit to identify all the bad data. Create a valid data string that has been 'de-glitched'.

String 1:	10111010	11110101	10111100	11001011	00101101
	01010000	01111010	10001100	00110111	00100110
	01111000	11001101	10110111	11011010	11100001
	10001010	10001111	01110011	10010011	11001011

String 2:	10111010	01110101	10111100	11011011	10101101
	01011010	01111010	10001000	10110111	00100110
	11011000	11001101	10110101	11011010	11110001
	10001010	10011111	01110011	10010001	11001011

Answer Key:

Problem: Below are two data strings that have been corrupted by cosmic ray glitches. Look through the data (a process called parsing) and use the right-most parity bit to identify all the bad data. Create a valid data string that has been ‘de-glitched’.

The highlighted data words are the corrupted ones.

String 1:	10111010	**11110101**	10111100	11001011	**00101101**
	01010000	01111010	10001100	00110111	00100110
	01111000	11001101	**10110111**	11011010	**11100001**
	10001010	10001111	01110011	**10010011**	11001011

String 2:	10111010	01110101	10111100	**11011011**	10101101
	01011010	01111010	**10001000**	**10110111**	**10100110**
	11011000	11001101	10110101	11011010	11110001
	10001010	**10011111**	01110011	10010001	**01001011**

In the first string, 11110101 has a parity bit of ‘1’ but it has an odd number of ‘1’ so its parity should have been ‘0’ if it were a valid word. Looking at the second string, we see that the word that appears at this location in the grid is ‘01110101’ which has the correct parity bit. We can see that a glitch has changed the first ‘1’ in String 2 to a ‘0’ in the incorrect String 1.

By replacing the highlighted, corrupted data words with the uncorrupted values in the other string, we get the following de-glitched data words:

Corrected:	10111010	01110101	10111100	11001011	10101101
	01011010	01111010	10001100	00110111	00100110
	11011000	11001101	10110101	11011010	11110001
	10001010	10001111	01110011	10010001	11001011

The odd word is the first word in the third row. The first transmission says that it is ‘0111100**0**’ and the second transmission says it is ‘1101100**0**’ Both wrong words have a parity of ‘1’ which means there is an even number of ‘1’ in the first seven places in the data word. But the received parity bit says ‘**0**’ which means there was supposed to be an odd number of ‘1’s in the correct word. Examining these two words, we see that the first three digits are ‘011’ and ‘110’ so it looks like the first and third digits have been altered. Unfortunately, we can’t tell what the correct string should have been. Because the rest of the word ‘11000’ has an even parity, all we can tell about the first three digits is that they had an odd number of ‘1’s so that the total parity of the complete word is ‘0’ . This means the correct digits could have been ‘100’, ‘010’, ‘111’, or ‘111’, but we can’t tell which of the three is the right one. That means that this data word remains damaged and can’t be de-glitched even after the second transmission of the data strings.