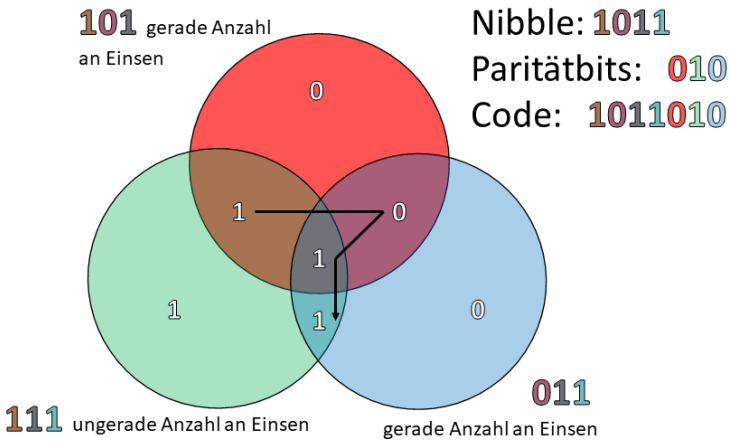
Klasse: _____

Datum: _____

Hamming-Code

Der Hamming-Code ist nach seinem Erfinder Richard Hamming benannt. Hamming arbeitete in den 1940er Jahren an einem damaligen Computer. Diese wurden noch nicht durch Maus und Tastatur oder gar über ein Touch-Display, sondern u.a. über Lochkarten bedient. Auf einer Lochkarte wurden Informationen gespeichert, indem an einer bestimmten Stelle entweder ein Loch oder eben kein Loch war. Loch bzw. kein Loch entspricht in modernen Computern 1 und 0, den sogenannten **Bits**.
Lochkarten waren sehr fehleranfällig, da das Papier durch die Benutzung geknickt werden konnte, oder unbeabsichtigt Löcher entstanden sind und der Computer eine Stelle für ein Loch hielt, die gar kein Loch sein sollte. Passierte so ein Fehler brachte dies die ganze Berechnung durcheinander, sodass der Fehler aufwändig von Hand korrigiert, oder die Berechnung neu gestartet werden musste.
Hamming suchte also eine Möglichkeit einen Code zu erstellen, der solche unvermeidlichen Fehler automatisch erkennen und korrigieren konnte. Die Lösung, sogenannte **Paritätsbits** zu verwenden, wird in ähnlicher Form in modernen Codierungen genutzt.

Wie funktioniert ein Hamming-Code?
Beim Hamming-Code werden die zu übertagenden Bits in 4-er Blöcke, sogenannte **Nibbles** aufgeteilt. Ein Nibble wird durch 3 Paritätsbits ergänzt und diese 7 Bits, anstelle der 4, übertragen.
Zur Bestimmung der Paritätsbits kannst du 3 Kreise aufmalen, wie auf dem Diagramm rechts. Die vier Bits des Nibbles werden dem Pfeil folgend in die Schnittbereiche der Kreise eingetragen. (Reihenfolge: oben links; oben rechts; Mitte; unten)



Bei den drei Bits, die sich im oberen Kreis befinden „101“ findest du eine gerade Anzahl an Einsen vor, deshalb ist das erste Paritätsbit **0**. Die drei Bits im linken Kreis „111“ haben eine ungerade Anzahl an Einsen, deshalb ist das zweite Paritätsbit **1**. Das letzte Paritätsbit ist **0**, denn unter den 3 Bits im rechten Kreis „011“ befinden sich 2, also eine gerade Anzahl an Einsen. Statt der 4 Bits „1001“ werden also 7 Bits „1011010“ versendet.

Wie du so nicht nur erkennen kannst, dass bei der Übertragung ein Fehler passiert ist, sondern diesen vielleicht sogar korrigieren kannst, erfährst du auf der nächsten Seite des Arbeitsblattes.

Aufgabe:

a. Ergänze die Bitfolgen um die drei Paritätsbits.
   1. 0110_____
   2. 0010_____
   3. 0111_____
b. Welches Bit fehlt hier, wenn die letzten drei Paritätsbits wie oben berechnet wurden?
   1. 0__110010
   2. 10__1101

Klasse: _____

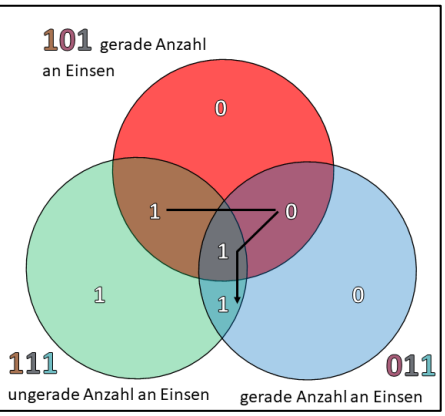
Datum: _____

Wie helfen uns die zusätzlichen Bits Fehler zu korrigieren?

Möchtest du prüfen, ob bei der Übertragung ein Fehler aufgetreten ist, ob also z.B. an einer Stelle eine 1 angekommen ist, wo allerdings eine 0 stehen sollte, kannst du genau wie zuvor vorgehen. Die ersten vier Bits enthalten die tatsächliche Information. Diese kannst du wieder in die Schnittflächen eintragen und die Paritätsbits errechnen. Nun vergleichst du die drei errechneten Paritätsbits mit den drei Übertragenen.

Je nachdem wie viele der errechneten Paritätsbits nicht mit den Übertragenen übereinstimmen, muss man Bits korrigieren.

übertragene Bitfolge:
1011011
Informations-Bits:
1011
Paritätsbits:
011
errechnete Paritätsbits:
010
tatsächliche Nachricht:
1011010



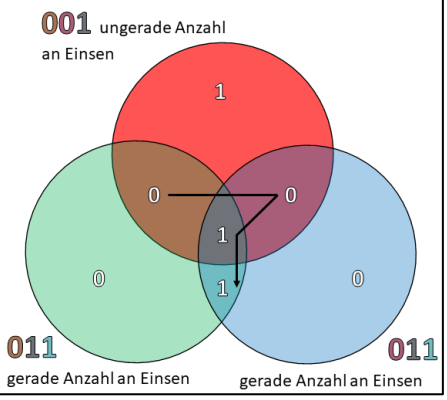
Eines der errechneten Paritätsbits stimmt nicht mit dem Übertragenen überein.

Dieses Bit ist falsch und wird ausgetauscht (0 gegen 1, oder 1 gegen 0).

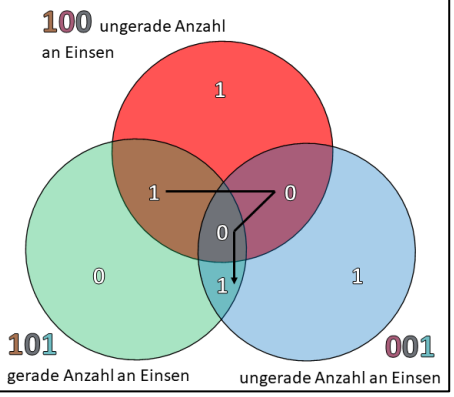
Zwei der errechneten Paritätsbits stimmen nicht mit den Übertragenen überein.

Das Informations-Bit, das sich in der Schnittfläche der beiden falschen Paritätsbits befindet ist falsch und wird ausgetauscht (0 gegen 1, oder 1 gegen 0).

übertragene Bitfolge:
0011010
Informations-Bits:
0001
Paritätsbits:
010
errechnete Paritätsbits:
100
tatsächliche Nachricht:
1011010



übertragene Bitfolge:
1001010
Informations-Bits:
1001
Paritätsbits:
010
errechnete Paritätsbits:
101
tatsächliche Nachricht:
1011010



Alle drei der errechneten Paritätsbits stimmen nicht mit den Übertragenen überein.

Das Informations-Bit in der Mitte ist falsch und wird ausgetauscht (0 gegen 1, oder 1 gegen 0).

Aufgabe:

Prüfe ob bei der Übertragung der 7 Bits ein Fehler aufgetreten ist und verbessere ihn, wenn möglich.

1. 0110 111
2. 1100 011
3. 1010 111
4. 0110 010