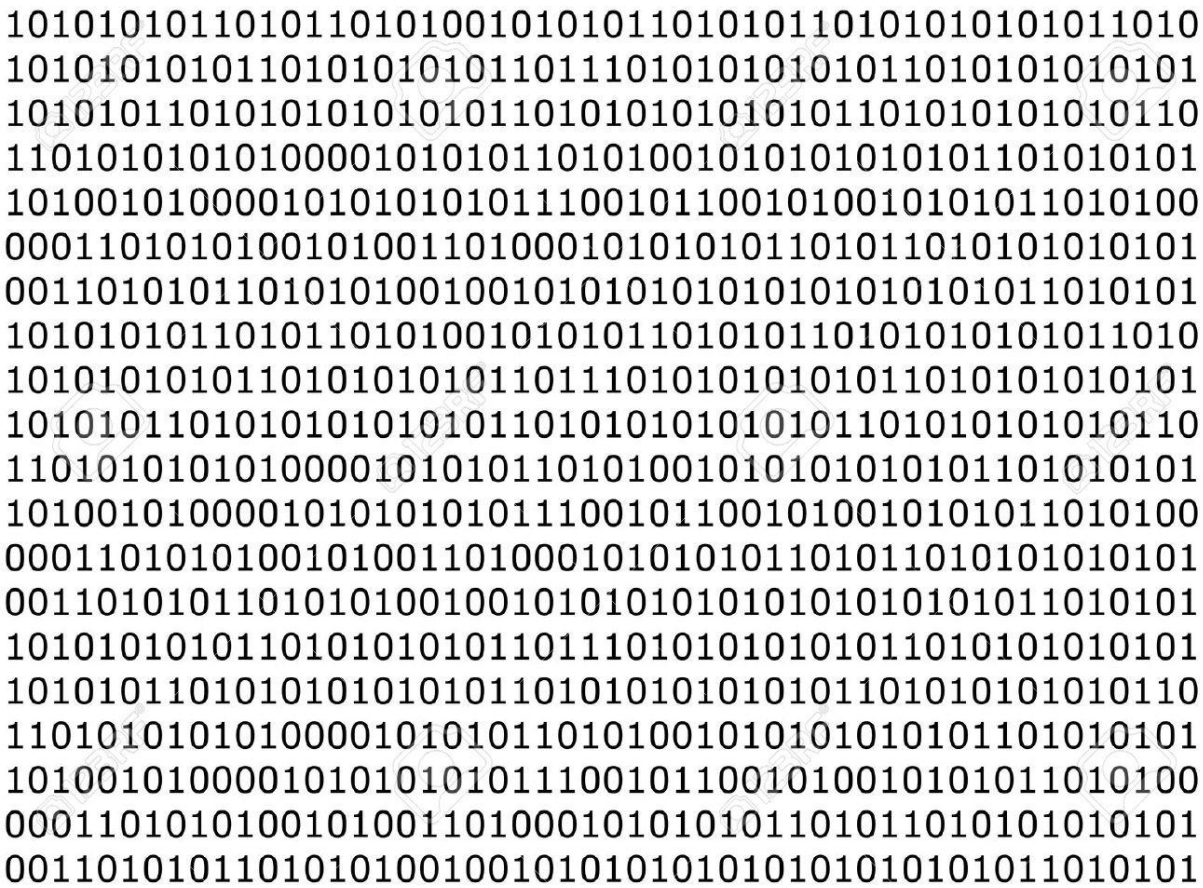
Profielwerkstuk


10101010110101101010010101011010101101010101011010
10101010101101010101011011101010101010110101010101010
10101011010101010101011010101010101011010101010101010110
11010101010100001010101101010010101010101011010101010101
10100101000010101010101110010110010100101010110101010100
00011010101001010011010001010101011010110101010101010101
00110101011010101001001010101010101010101010110101010101
10101010110101101010010101011010101101010101010110101010
10101010101101010101011011101010101010110101010101010101
10101011010101010101011010101010101011010101010101010110
11010101010100001010101101010010101010101011010101010101
10100101000010101010101110010110010100101010110101010100
00011010101001010011010001010101011010110101010101010101
00110101011010101001001010101010101010101010110101010101
10101010101101010101011011101010101010110101010101010101
10101011010101010101011010101010101011010101010101010110
11010101010100001010101101010010101010101011010101010101
10100101000010101010101110010110010100101010110101010100
00011010101001010011010001010101011010110101010101010101
00110101011010101001001010101010101010101010110101010101

Fout detecterende en verbeterende codes

Een compacte module over het onderwerp fouten detectie en verbetering

Gemaakt door Roy van Schaijk, Boris Kloeg en Willy Mackus

Inhoudsopgave

1. Introductie en coderen met cirkelschema's

Informatie wordt overgebracht door symbolen en tekens. In ons dagelijks leven gebruiken we het alfabet, dat uit 26 verschillende tekens bestaat en wij tellen in het tienstalligstelsel. Er zijn echter nog veel meer vormen van taal. Hieronder staat een lijst met verschillende voorbeelden.

• Piet eet vlees	{a, b, …, y, z}	'Ons' alfabet
• Παντα ρει	{α, β, …, ψ, ω}	Grieks alfabet
• 0123456789	{0, 1, …, 8, 9}	Tientallig stelsel (decimale talstelsel)
• 01000101110	{0, 1}	Tweetallig stelsel (binaire talstelsel)
• 0123456789ABCDEF	{0, 1, …, E, F}	Hexadecimaal talstelsel (16-tallig)
• . . . - - - . . .	{., -}	Morsecode
• 조선말		Koreaans

Vrijwel alle computers maken gebruik van het binaire talstelsel. Dit betekent dat alles wordt gecommuniceerd met enen en nullen. Over het algemeen wordt een combinatie van acht enen en nullen gebruikt om één letter van 'ons' alfabet weer te geven. Met andere woorden, één **byte** bevat 8 bits. De bits zijn de individuele symbolen (losse enen en nullen) en de byte is het codewoord dat dus bestaat uit een combinatie van acht enen en nullen. De letter 'a' wordt bijvoorbeeld gecodeerd als 01100001 en de letter 'b' als 01100010.
Wanneer je op de computer een tekst typt, zet de computer deze tekst om naar het binaire talstelsel. Het totale aantal symbolen zal dus acht keer zo groot worden.

Het komt regelmatig voor dat een 1 als een 0 wordt ontvangen, of dat een 0 als een 1 wordt ontvangen. Dit zijn fouten. Wanneer er geen foutdetecterend systeem is, zorgt elke incorrecte bit dus voor een incorrecte byte. Als er wel een controlesysteem is, maakt het niet uit of er een fout wordt gemaakt. In een code van 8 bits kan dan (maximaal) één fout worden opgespoord en verbeterd met behulp van het controlesysteem.

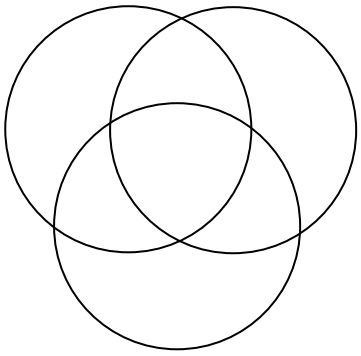
Het is bijna tijd om zelf een foutdetecterende en –verbeterende code te maken. Voor het gemak werken we in dit eerste hoofdstuk met boodschappen van 4 bits. Zo'n combinatie heet een **codewoord**. Een codewoord is dus een groepje enen en nullen dat voor één symbool/letter/teken codeert.

Een voorbeeld van zo'n foutdetecterende code is om het bericht te versturen met twee keer zoveel bits, waarbij elk bit wordt herhaald. Elke bit wordt dus twee keer achter elkaar verzonden. Bijvoorbeeld: 0100 wordt 00110000 en 1010 wordt 11001100.
Als de code 00110011 wordt ontvangen is de oorspronkelijke code dus 0101.
Stel dat één bit fout wordt ontvangen, ontstaat bijvoorbeeld de code 01110011. Het is nu duidelijk dat er een fout is gemaakt, omdat de code niet meer te herleiden is tot 0101. De code zou namelijk ook oorspronkelijk 1101 kunnen zijn, ervan uitgaande dat de eerste 0 eigenlijk een 1 is. Deze vorm van coderen is dus wel **detecterend** (je weet dat er een fout is), maar niet **verbeterend** (je kunt de code niet verbeteren).

Bedenk je eenzelfde soort code, maar wordt elke bit drie keer achterelkaar verstuurd, dan is de code wel verbeterend. 0101 wordt namelijk 000111000111. Is er een fout (bijvoorbeeld 010111000111), dan is duidelijk waar de fout zit én wat de oorspronkelijke boodschap is. Omdat er twee nullen zijn en maar één 1, is de 1 waarschijnlijk fout.
Dit is dus een voorbeeld van een foutdetecterende en –verbeterende code, maar dit vereist wel drie keer zoveel bits dan de oorspronkelijke boodschap. Dit is erg veel.

Nu gaan we zelf een 'eenvoudig' coderingssysteem maken. Het idee is dat aan elke code van vier bits, drie extra controle bits worden toegevoegd. De vier 'normale' bits noemen we **informatiebits**. Deze bevatten namelijk de boodschap. De drie 'extra' bits worden ook wel **pariteitsbits** genoemd. Dit zijn de bits die het mogelijk maken fouten te detecteren en te verbeteren. Verder is het belangrijk dat we ervan uitgaan dat er maximaal één fout wordt gemaakt in de gehele code.

De drie pariteitsbits worden gekozen aan de hand van de oorspronkelijke boodschap. Hiertoe zetten we de oorspronkelijke boodschap van vier bits een cirkelschema:

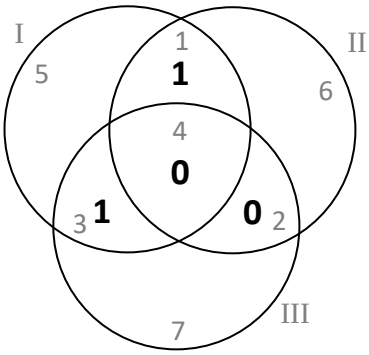


Figuur 1: Een cirkelschema

Op de volgende pagina staan twee voorbeeldopgaven die laten zien hoe het (de)coderen met behulp van cirkelschema's werkt.
Zoals je zult merken in de opgaven is deze manier van coderen fout detecterend en -verbeterend.

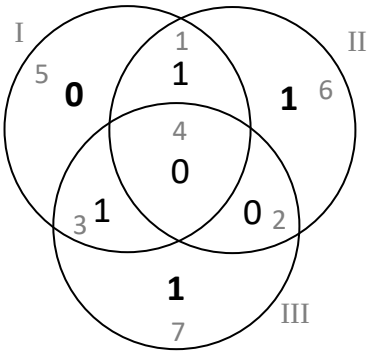
Voorbeeldopgave 1: Het codewoord is **1010**. Codeer met behulp van het cirkelschema.

- Plaats de eerste bit in vakje 1, de tweede in 2, etc.
  Zoals je ziet, zijn de vakjes genummerd 'met de klok mee'



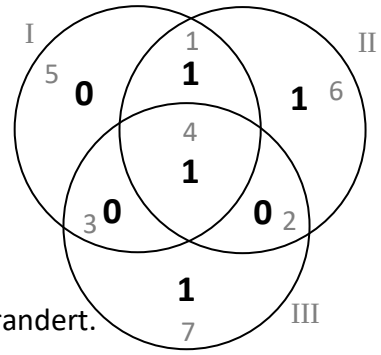
- Bepaal de overige drie bits met de regel:
  "In elke cirkel staat een even aantal enen"
  Je zet dus in elke cirkel (cirkel I, II en III) een 1 of een 0 om te voldoen aan de regel.

- Deze drie bits plaats je achter de oorspronkelijke code
  De nieuwe code is dus: 1010**011**



Voorbeeldopgave 2: Decodeerde de volgende (ontvangen) code: **1001011**.

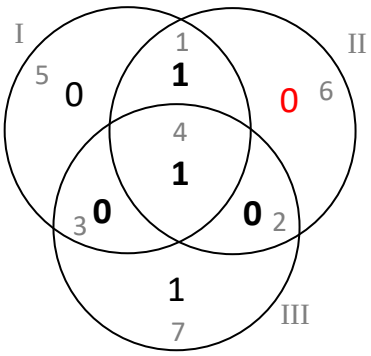
- Plaats de code in het cirkelschema



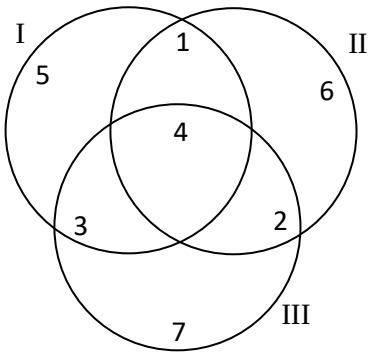
- Bepaal of het cirkelschema voldoet aan de regel:
  "In elke cirkel staan een even aantal enen"

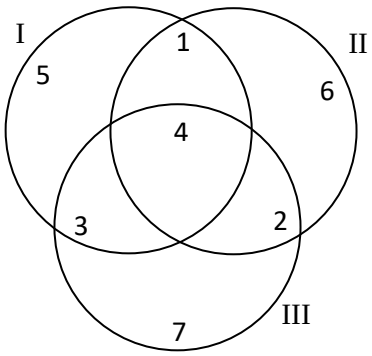
- Ga ervan uit dat er maximaal één fout gemaakt wordt
  en verbeter **één** bit (indien nodig) om te voldoen aan de regel.
  Het maakt niet uit of je één informatiebit of één pariteitsbit verandert.

De (verzonden) code was dus 1001001. (Fout pariteitsbit)
Het codewoord is dus **1001**



- Opgave 1: Codeer 0011 en 1011.





_____

_____

- Opgave 2: Decodeer 0101100, 1011000 en 1011001.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

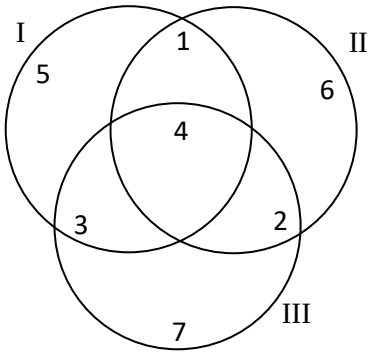
_____

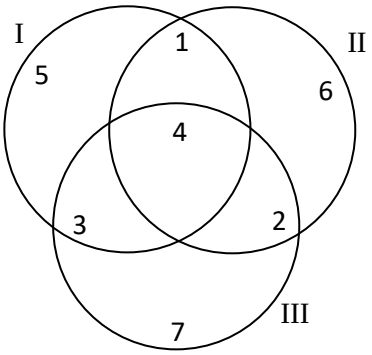
_____

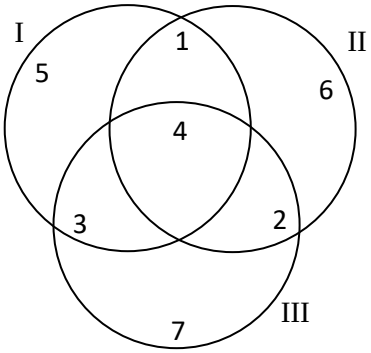
_____

_____

_____







2. Andere talstelsels

In Nederland en in een groot gedeelte van de wereld gebruiken we het 10-tallig stelsel. Dit betekent dat wij de cijfers 0, 1, 2, 3, 4, 5, 6, 7, 8 en 9 gebruiken. Het getal 1234 in het 10-tallig stelsel stelt eigenlijk het volgende voor: $1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$. Dit kunnen we opschrijven als $a_3a_2a_1a_0$ en dit geeft $a_3 \cdot 10^3 + a_1 \cdot 10^2 + a_1 \cdot 10^1 + a_0 \cdot 10^0$. Als je dit getal door 10 deelt, heb je dus als rest a_0 en het quotiënt is $a_3a_2a_1$. Bij het delen van $a_3a_2a_1$ door 10 is de rest dus a_1 en het quotiënt a_3a_2. Op deze manier kun je dus de losse cijfers op de plaatsen a_n bepalen. Dit is erg handig om getallen om te rekenen in andere talstelsels.

Maar wat is een ander talstelsel nu eigenlijk? We kijken als voorbeeld naar een octaal (8-tallig) stelsel. In dit stelsel gebruiken we alleen de cijfers 0, 1, 2, 3, 4, 5, 6 en 7. $7 + 1$ is in het octale stelsel dus 10. Om verwarring te voorkomen over welk talstelsel we het hebben, zullen we dit voortaan aangeven met een subscript. Dit is bijvoorbeeld 9_{dec}, wat dus 11_{oct} is. Het getal 1234_{oct} stelt dus eigenlijk $1 \cdot 8^3 + 2 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0$ voor. Om te weten wat 1234_{oct} in het decimale stelsel is, kun je dus simpelweg de som oplossen.

- Opgave 3**:** Schrijf 17534_{oct} en 267654_{oct} in de decimale schrijfwijze.

_____

_____

_____

_____

_____

Om de decimale schrijfwijze in de octale schrijfwijze om te zetten, moet er wel wat worden gerekend. Hiervoor kunnen we de manier gebruiken die in de uitleg bovenaan de pagina staat, namelijk door telkens de rest te bepalen bij het delen door *n* voor het gewenste *n*-tallig stelsel. In het geval van het voorbeeld bekijk je dus elke keer de rest bij delen door 8.

Voorbeeldopgave 3:

6839_{dec} octaal schrijven:

$6839 = 854 \cdot 8 + 7$	$a_0 = 7$
$854 = 106 \cdot 8 + 6$	$a_1 = 6$
$106 = 13 \cdot 8 + 2$	$a_2 = 2$
$13 = 1 \cdot 8 + 5$	$a_3 = 5$
$1 = 0 \cdot 8 + 1$	$a_4 = 1$

Het is dus 15267_{oct}

- Opgave 4**:** Schrijf 394521_{dec} en 921463_{dec} in de octale schrijfwijze.

_____

_____

_____

_____

_____

_____

_____

De regels die gelden voor het octale telsysteem gelden natuurlijk ook voor andere telsystemen. Zo wordt er tegenwoordig ook vaak gebruikt gemaakt van het hexadecimale (16-tallig) en binaire (2-tallig) stelsel. Om een stelsel te gebruiken met meer dan 10 cijfers moeten er natuurlijk wel symbolen zijn om dit te noteren. Hiervoor gebruiken we de symbolen A, B, C, D, E en F. Het hexadecimale stelsel bestaat dus uit 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Probeer hier maar eens mee te rekenen.

- Opgave 5: Schrijf $1\mathrm{AF6}_{\mathrm{hex}}$ in de decimale schrijfwijze.

_____

_____

_____

- Opgave 6: Schrijf 1000_{dec} in de hexadecimale schrijfwijze.

_____

_____

_____

_____

Het zou kunnen dat je de hexadecimale schijfwijze al ooit vaker hebt gezien. Deze schijfwijze wordt namelijk gebruikt bij **kleurcodes**. Een voorbeeld hiervan is #FFFFFF dat staat voor wit en #000000 staat voor zwart. Het hekje (#) is simpelweg een teken wat aangeeft dat het hier om een kleurcode gaat. Een kleurcode bestaat uit 3 gedeeltes van 2 hexadecimale cijfers: de eerste 2 cijfers staan voor de rode kleur, de middelste 2 cijfers voor het groene component en de laatste 2 cijfers voor het blauwe component. Deze 2 cijfers zijn dus samen maximaal $(15 \cdot 16^1) + (15 \cdot 16^0) = 255$, en dit staat voor de intensiteit van de kleur.

- Opgave 7: Als je kijkt naar de hexadecimale kleuren, wat zou de kleur #FF0000 zijn? En #0000FF? En welke kleur zou #FFFF00 voorstellen?

_____

_____

Tegenwoordig wordt er ook steeds meer de decimale notatie van de kleuren gebruikt. In deze notatie stelt rgb(0,0,0) zwart voor en wit wordt voorgesteld door, je raadt het al: rgb(255,255,255). Een computer moet dus omrekenen om beide methodes te kunnen gebruiken.

- Opgave 8: Help de computer een handje: zet het codewoord rgb(126,78,234) om in de hexadecimale variant.

  _____

  _____

  _____

  _____

- Opgave 9: Natuurlijk moet er ook ooit andersom gerekend worden. Zet het codewoord #AF1F33 om de decimale schrijfwijze.

  _____

  _____

  _____

Een stelsel waarmee tegenwoordig misschien meer wordt gerekend dan met ons decimale stelsel, is het binaire (2-tallige) stelsel. Elektronische en magnetische geheugens voor computers werken hiermee. Dit wordt gebruikt aangezien het simpel is om het verschil tussen “aan” en “uit” te meten. Omdat er maar 2 waardes zijn, zijn fouten makkelijk te verbeteren. Natuurlijk kunnen er nog steeds fouten in een codewoord verschijnen. Daarom krijg je verderop in dit boekje meer uitleg over andere manieren om te verbeteren. Je hebt geleerd dat 8 bits (binaire getallen) bytes zijn.

- Opgave 10: Uit hoeveel verschillende getallen kan een byte bestaan?

  _____

- Opgave 11: Schrijf 435_{dec} binair.

  _____

  _____

  _____

  _____

  Je kunt erg makkelijk zien of een binair getal deelbaar is door 2, 4, 8, 16, 32, 64, ect. (ofwel deelbaar is door 2^n) en wat dan de rest is.

- Opgave 12: Wat is de rest als je 11011010101110101001001110010100 11 deelt door 16?

  _____

  _____

3. Modulo rekenen

Het rekenen in andere talstelsels is erg handig voor het coderen met modulo rekenen.
Modulo rekenen wordt ook wel 'klokrekenen' genoemd. Bij 'klokrekenen' ga je ervan uit dat bijvoorbeeld 13 uur hetzelfde is als 1 uur en dat 26 uur hetzelfde is als 14 uur, dat weer hetzelfde is als 2 uur. Je maakt dus telkens stappen van 12 uur. Het eerst genoemde voorbeeld wordt genoteerd als: $13 \equiv 1(mod\ 12)$
Dit houdt dus in dat 13 gelijk is aan 1, rekenend modulo 12

De algemene notatie is: $a \equiv b(mod\ n)$, waarbij $0 \leq b < n$. De waarde van b is de kleinst mogelijke positieve waarde. Het verschil tussen a en b is dus een veelvoud van n. In formulevorm: $a - b = k \cdot n$. De waarde b bereken je dus met $a - k \cdot n = b$

In dit hoofdstuk gaan we leren modulo rekenen. In het volgende hoofdstuk gaan we hiermee coderen.

Voorbeeldopgave 4: Bepaal de waarde van b voor $160 \equiv b(mod\ 11)$.

- $0 \leq b < n$, dus in dit geval is $0 \leq b < 11$
- Haal het getal 11 net zo vaak van 160 af, totdat de uitkomst voldoet aan $0 \leq b < 11$.
  Je maakt dus stappen van 11:
  $160 - 11 = 149$
  $149 - 11 = 138$
  $138 - 11 = 127$
  …
  $28 - 11 = 17$
  $17 - 11 = 6$, voldoet aan $0 \leq b < 11$

  Oftewel het is de rest bij delen door 11

- Op deze manier is $160 \equiv \mathbf{6}(mod\ 11)$, dus b = 6

Als geldt dat a gelijk is aan b modulo n, geldt ook b is gelijk aan a modulo n.
$a \equiv b(mod\ n) \Leftrightarrow b \equiv a(mod\ n)$

De waarde van a kan ook negatief zijn. De rekenregels blijven echter hetzelfde. In plaats van stappen van n 'omlaag', maak je nu stappen van n 'omhoog'.

Voorbeeldopgave 5: Bereken: $18(mod\ 7)$, $-12(mod\ 7)$ en $-8^4(mod\ 9)$.

- $18 \equiv \mathbf{4}(mod\ 7)$, want $18 - x \cdot 7 = b$, waarbij b de restwaarde is $(0 \leq b < 7)$.
- $-12 \equiv \mathbf{2}(mod\ 7)$, want $-12 + x \cdot 7 = b$, waarbij b de restwaarde is $(0 \leq b < 7)$.
- $-8^4 \equiv -4096 \equiv \mathbf{8}(mod\ 9)$, want $-8^4 + x \cdot 9 = b$, met $(0 \leq b < 9)$.

- Opgave 13: Bereken: $27(mod\ 11),\ -23(mod\ 5)$ en $-8^7(mod\ 9)$.

_____

_____

_____

_____

_____

_____

_____

Twee andere rekenregels die erg handig zijn voor optellen en vermenigvuldigen:
Somregel: $\big(a\ (mod\ n\ ) +\ b(mod\ n)\big)\ mod\ n\ \Leftrightarrow\ (a+b)\ mod\ n$
Productregel: $\big(a\ (mod\ n\ ) \cdot b(mod\ n)\big)\ mod\ n\ \Leftrightarrow\ (a \cdot b)\ mod\ n$

De laatste opgave van opgave 12 was misschien een beetje gemeen. Met bovenstaande regels is die opgave veel sneller op te lossen.

Voorbeeldopgave 6: Bereken $-8^7(mod\ 9)$.

- Bedenk dat $-8^7 = -8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8$. Je kunt de productregel gebruiken.

- Dit houdt in dat elke $8 \equiv -1(mod\ 9)$
  $-8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \cdot 8 \equiv -(-1) \cdot (-1) \cdot (-1) \cdot (-1) \cdot (-1) \cdot (-1) \cdot (-1) \equiv 1(mod 9)$

- Dit schrijf je zo op:
  $-8^7 \equiv -(-1)^7 \equiv \mathbf{1}(mod\ 9)$

- Opgave 14: Bereken.

  - $6^{89}\ (mod\ 7)$
  - $13 \cdot 4^{22}\ (mod\ 15)$
  - $533.878.220.610\ (mod\ 8)$

_____

_____

_____

_____

_____

_____

_____

_____

Ook **delen** is mogelijk, maar niet met alle 'modulo's'. Bij modulo 13 blijkt het altijd mogelijk te zijn om te delen, dus daarom werken we nu alleen met modulo 13.

Wordt er gevraagd naar $3/6(mod\ 13)$, dan zoek je naar een getal, dat vermenigvuldigd met 6, het getal 3 oplevert. Dit kun je na gaan door bovenstaande formule om te schrijven:

$\frac{3}{6}(mod\ 13) \equiv b$
$b \equiv \frac{3}{6}(mod\ 13)$ *beide kanten vermenigvuldigen met 6*
$6 \cdot b \equiv 3\ (mod\ 13)$

b is dus bijvoorbeeld 7, want $6 \cdot 7 \equiv 42 \equiv 3\ (mod\ 13)$

<u>Voorbeeldopgave 7</u>: Bereken. $1/8\ (mod\ 13)$.

- $b \equiv \frac{1}{8}(mod\ 13)$ *beide kanten vermenigvuldigen met 8*
- $8 \cdot b \equiv 1(mod\ 13)$
- Hiertoe lossen we de volgende vergelijking op: $8 \cdot b = 1 + k \cdot 13$, waarbij k elk natuurlijk getal kan zijn. Dit kun je doen door 'trial-and-error' of door een tabel op te stellen:

k	0	1	2	3	4	5	6	7	8	9	10	11
$1 + k \cdot 13$	1	14	27	40	53	66	79	92	105	118	131	144
$8 \cdot b$	1	14	27	40	53	66	79	92	105	118	131	144
Is b een geheel getal?	nee	nee	nee	Ja b=**5**	nee	nee	nee	nee	nee	nee	nee	Ja b=18

- Hieruit volgt dat $b = 5$, maar ook dat $b = 18$ en $b = 31$.
  De algemene formule is $b = 5 + l \cdot 13$, omdat we rekenen met modulo 13.

- Dus $\frac{1}{8}(mod\ 13) \equiv \mathbf{5}$

- <u>Opgave 15</u>: Bereken: $5/3\ (mod\ 13)$.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

4. Coderen met behulp van modulo rekenen

Met behulp van modulo rekenen kun je een coderingssysteem maken. Hierbij werken we met modulo 13. Dit betekent dat we werken met het 13-tallig stelsel met de cijfers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (= A), 11 (= B), 12 (= C). In dit hoofdstuk maken we codewoorden met vijf **codecijfers** en twee **pariteitscijfers**. Een code heeft dus de volgende opbouw: [c_0 c_1 c_2 c_3 c_4 p_5 p_6], waarbij c_5 en c_6 de pariteitscijfers zijn. Een parititeitscijfer is in dit hoofdstuk een 'getal' van 0 t/m 12. Tip: schrijf bij de opgaven de getallen A, B en C om naar de decimale notatie, dus 10, 11 en 12, want dit rekent gemakkelijker.

De waarde van de pariteitscijfers wordt bepaald door de volgende twee formules:

$$\begin{cases} c_0 + c_1 + c_2 + c_3 + c_4 + p_5 + p_6 \equiv \mathbf{0} (mod\ 13) \\ c_1 + 2\cdot c_2 + 3\cdot c_3 + 4\cdot c_4 + 5\cdot p_5 + 6\cdot p_6 \equiv \mathbf{0} (mod\ 13) \end{cases}$$

Voorbeeldopgave 8: Codeer: [16B29]

- $c_0 + c_1 + c_2 + c_3 + c_4 + p_5 + p_6 \equiv 0 (mod\ 13)$
  $c_1 + 2\cdot c_2 + 3\cdot c_3 + 4\cdot c_4 + 5\cdot p_5 + 6\cdot p_6 \equiv 0 (mod\ 13)$

- Invullen levert:
  $01 + 06 + 11 + 02 + 09 + p_5 + p_6 \equiv 0 (mod\ 13)$
  $06 + 2\cdot 11 + 3\cdot 02 + 4\cdot 09 + 5\cdot p_5 + 6\cdot p_6 \equiv 0 (mod\ 13)$

- $29 + p_5 + p_6 \equiv 0 (mod\ 13)$ met $29 \equiv 3 (mod\ 13)$
  $70 + 5\cdot p_5 + 6\cdot p_6 \equiv 0 (mod\ 13)$ met $70 \equiv 5 (mod\ 13)$

- $3 + p_5 + p_6 \equiv 0 (mod\ 13)$
  $5 + 5\cdot p_5 + 6\cdot p_6 \equiv 0 (mod\ 13)$

- De eerste vergelijking omschrijven, levert: $p_5 \equiv -p_6 - 3\ (mod\ 13)$
  Dit vul je in, in de tweede vergelijking. Dit levert:
  $5 + 5\cdot(-p_6 - 3) + 6\cdot p_6 \equiv 0 (mod\ 13)$
  $5 - 5p_6 - 15 + 6\cdot p_6 \equiv 0 (mod\ 13)$
  $-10 + p_6 \equiv 0 (mod\ 13)$
  $p_6 \equiv \mathbf{10} (mod\ 13)$ ofwel p_6 = A$(mod\ 13)$

- Nu de waarde van p_6 bekend is, kun je de waarde van p_5 berekenen.
  $p_5 \equiv -p_6 - 3\ (mod\ 13)$ met $p_6 \equiv 10 (mod\ 13)$
  $p_5 \equiv -10 - 3\ (mod\ 13)$
  $p_5 \equiv \mathbf{0}\ (mod\ 13)$

- De gecodeerde code is dus [16B29**0A**]

- Opgave 16: Codeer: [C6253] en [37328].

Het coderen met behulp van modulo 13 is foutdetecterend en -verbeterend. Nu je weet hoe je een reeks van vijf cijfers kunt coderen, gaan we leren hoe je een reeks van zeven cijfer (dit is inclusief de twee pariteitscijfers) kunt decoderen. Hiervoor vul je alle zeven cijfers in, in de twee formules.

$$\begin{cases} c_0 + c_1 + c_2 + c_3 + c_4 + p_5 + p_6 \equiv \boldsymbol{x} (mod\ 13) \\ c_1 + 2 \cdot c_2 + 3 \cdot c_3 + 4 \cdot c_4 + 5 \cdot p_5 + 6 \cdot p_6 \equiv \boldsymbol{y} (mod\ 13) \end{cases}$$

Wanneer de waarden van x en y allebei gelijk zijn aan 0, dan is er geen fout gemaakt en is de code dus hoogstwaarschijnlijk correct, mits er niet meerdere fouten in de code zitten. Je zult echter merken dat dit niet altijd het geval is. Komt er op de plaats van de x het cijfer 9 te staan, dan betekent dit dat er één cijfer is dat 9 stappen 'te hoog' is.

Gebruik bij het decoderen de volgende set formules:

$$\begin{cases} c_0 + c_1 + c_2 + c_3 + c_4 + p_5 + p_6 \equiv \boldsymbol{e} (mod\ 13) \\ c_1 + 2 \cdot c_2 + 3 \cdot c_3 + 4 \cdot c_4 + 5 \cdot p_5 + 6 \cdot p_6 \equiv \boldsymbol{i} \cdot \boldsymbol{e} (mod\ 13) \end{cases}$$

Hierbij is e de grootte van de afwijking en is i de plaats van de afwijking. Zie het voorbeeld.

<u>Voorbeeldopgave 9</u>: Decodeer: [A592431]

- $\begin{cases} 10 + 05 + 09 + 02 + 04 + 03 + 01 \equiv \boldsymbol{e} (mod\ 13) \\ 05 + 2 \cdot 09 + 3 \cdot 02 + 4 \cdot 04 + 5 \cdot 03 + 6 \cdot 01 \equiv \boldsymbol{i} \cdot \boldsymbol{e} (mod\ 13) \end{cases}$

- $34 \equiv \boldsymbol{e} (mod\ 13)$, dus $e \equiv 8 (mod\ 13)$
  $66 \equiv \boldsymbol{i} \cdot \boldsymbol{e} (mod\ 13)$, dus $i \cdot e \equiv 1 (mod\ 13)$

- Substitueer de formules. Dit betekent dat $i \cdot 8 \equiv 1 (mod\ 13)$
  Deze vergelijking los je op. Dit doe je het hetzelfde als in het vorige hoofdstuk.

- Je zult vinden dat $i = 5$
  Dus $e = \mathbf{8}$ en $i = \mathbf{5}$

- Op plaats C$\boldsymbol{i}$ is het ontvangen cijfer, $\boldsymbol{e}$ stappen 'te groot'.
  Met andere woorden: op plaats C_5 is het cijfer **8** stappen 'te groot'.
  [10, 05, 09, 02, 04, <u>03</u>, 01] → C_5 = 03, dus $3 - 8 = -5 \equiv 8 (mod\ 13)$

- De verzonden boodschap was dus: [A5924**8**1]
  Het codewoord blijft in dit voorbeeld onveranderd: [A5924]

- Opgave 17: Decodeer: [9272CA4] en [A347182].

5. Hamming-code

Een veelgebruikte manier om tekst om te zetten naar een rij van bits en omgekeerd is door gebruik te maken van het ASCII (American Standard Code for Information Interchange)-protocol, waarvan je hieronder een vereenvoudigde versie vindt. Met behulp van deze tabel worden letters en symbolen omgezet naar een rij van telkens 7 bits. Zo kunnen we het woord “code” bijvoorbeeld omzetten naar 1100011110111111001001100101. Anderzijds leert de tabel ons dat de bits 1000011110111111011001100001 voor het woord “Cola” staan. (Let op hoofdletter of kleine letter!). Je verdeelt de code dus in stukken van 7 bits, en dan ga je kijken in het “woordenboek” hieronder. We gaan dit later gebruiken.

symbool	bits	symbool	bits	symbool	bits
a	1100001	A	1000001	0	0110000
b	1100010	B	1000010	1	0110001
c	1100011	C	1000011	2	0110010
d	1100100	D	1000100	3	0110011
e	1100101	E	1000101	4	0110100
f	1100110	F	1000110	5	0110101
g	1100111	G	1000111	6	0110110
h	1101000	H	1001000	7	0110111
i	1101001	I	1001001	8	0111000
j	1101010	J	1001010	9	0111001
k	1101011	K	1001011	(spatie)	0100000
l	1101100	L	1001100	.	0101111
m	1101101	M	1001101	,	0101100
n	1101110	N	1001110	:	0111010
o	1101111	O	1001111	;	0111011
p	1110000	P	1010000	?	0111111
q	1110001	Q	1010001	!	0100001
r	1110010	R	1010010	(	0101000
s	1110011	S	1010011	)	0101001
t	1110100	T	1010100	[	1011011
u	1110101	U	1010101	]	1011101
v	1110110	V	1010110	*	0101010
w	1110111	W	1010111	/	0101111
x	1111000	X	1011000	+	0101011
y	1111001	Y	1011001	–	0101101
z	1111010	Z	1011010	=	0111101

Los van het ASCII-protocol wordt de Hamming-code veel gebruikt als foutdetectie en correctie, doordat hij simpel is, maar ook effectief en efficiënt. De **Hamming-code** is maar een van de vele vormen van een foutdetecterende en -verbeterende code. Het is wel de meest gebruikte.

Bij gebruik van de Hamming-code worden er pariteitsbits toegevoegd aan een oorspronkelijke code en bij de meest gebruikte vorm van de Hamming-code kan er 1 fout worden gedetecteerd en gecorrigeerd. Richard Hamming heeft deze manier van coderen in 1950 bedacht. Je hebt in hoofdstuk 1 met een cirkelschema gewerkt. Dit was al een vorm van de Hamming-code, en direct ook de meest gebruikte. Aan elke 4 informatiebits worden 3 pariteitsbits toegevoegd, hierdoor wordt het ook wel de (7,4)-Hamming-code genoemd.

De meest gebruikte toepassing voor Hamming-codes is tegenwoordig in zogenaamd ECC (Error Correcting Code) geheugen. Dit wordt veelal in servers gebruikt waarbij er geen fout mag ontstaan in een berekening. Vaak wordt er een afgesproken lengte aan de Hamming-code gegeven. Dit wordt gedaan omdat het irreëel is dat er in een lange code van bijvoorbeeld 100 karakters maar 1 fout is. In plaats daarvan wordt er dan gezegd dat je de (14,10)-Hamming-code gebruikt. Je splitst daartoe de code op in stukjes van elk 10 karakters. In elk stukje kan dan één fout opgespoord en verbeterd worden.

Om een Hamming-code te maken heb je een boodschap nodig. De Hamming-code codeert namelijk de oorspronkelijke code met behulp van pariteitsbits. Als voorbeeld zullen we een boodschap van 4 bits kiezen, bijvoorbeeld 1101.

Om hier de pariteitsbits van te berekenen, zet je een pariteitsbit op elke positie van een macht van 2 (2^n). Dus op plaats 1, 2 en 4 komt in dit geval een pariteitsbit te staan. Op de overige plaatsen komen de codebits te staan. De code ziet er dan zo uit: $p_1 p_2 1 p_4 101$. Er zijn dus 3 pariteit bits toegevoegd.

Om de pariteitsbit op plaats **1** te berekenen, neem je de waarde op plaats 1 (nog niet gedefinieerd), dan sla **1** plaats over en pak dan weer de waarde, etc. De opgetelde waarde van deze bits moet even zijn. Voor pariteitsbit 1 is het dus $p_1 + 1 + 1 + 1$. De waarden van p_1 moet dus 1 zijn.

Voor de tweede pariteitsbit pak je de waardes van de eerste **2** plaatsen (vanaf de **2**de plaats dus), sla je er **2** over, en pak je weer **2** waardes. Ook dit moet even worden. Dit is dus $p_2 + 1 + 0 + 1$, de waarden van p_2 is dus 0.

Voor de derde pariteit bit (op plaats **4**) geldt dat je de waardes van de eerste **4** plaatsen (vanaf de pariteit bit), sla je er **4** over, en pak je weer **4** waardes. Dit moet ook weer even worden. Als je het einde van de code bereikt stop je. Dit is dus $p_4 + 1 + 0 + 1$. Hiervan is de waarde ook weer 0. De Hamming-code is dus 1010101. Zie ook de tabel hieronder.

	p1	p2	d1	p4	d2	d3	d4
p1	✗		✗		✗		✗
p2		✗	✗			✗	✗
p4				✗	✗	✗	✗

Dus p1 + d1 + d2 + d4
Dus p2 + d1 + d3 + d4
Dus p4 + d2 + d3 + d4

Regel: sommen moeten even zijn

Om een fout te kunnen detecteren en corrigeren zullen we een foute code pakken: 1000101. Om deze code te checken ga je na of alle pariteitsbits kloppen. Voor p_1: **1**0**0**0**1**0**1**. p_1 klopt dus **niet**. Voor p_2: 1**00**01**01**. p_2 klopt dus ook niet. Voor p_4: 100**0101**. p_4 klopt dus wel. Nu komt de 'truc': je pakt de foute pariteitsbits en je telt de positienummers bij elkaar op. In dit geval dus $p_1 + p_2 = 3$. Plaats 3 is dus fout en moet dus een 1 zijn. Hieronder staat als handigheidje de tabel tot en met 15 data bits. NB: als er maar 1 pariteitsbit fout is, dan zit de fout in de pariteitsbit zelf. Verder worden bij het coderen van bits nullen toegevoegd als de deling niet uitkomt, bijvoorbeeld de boodschap 10 wordt 1000. Achter een code kun je dus net zoveel nullen plakken als je wilt, om het rekenen eventueel makkelijker te maken.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	p1	**p2**	**d1**	**p4**	**d2**	**d3**	**d4**	**p8**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**p16**	**d12**	**d13**	**d14**	**d15**
p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗	
p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗	
p4				✗	✗	✗	✗					✗	✗	✗	✗					✗
p8								✗	✗	✗	✗	✗	✗	✗	✗					
p16																✗	✗	✗	✗	✗

<u>Voorbeeldopgave 10</u>: We hebben de code 10010010111. Codeer deze.

Er moeten op plaats 2^n pariteitsbits, dus op plaats 1, 2, 4 en 8. De codebits worden op de overgebleven plekken geplaatst. We krijgen dus $p_1p_2 1 p_4 001 p_8 0010111$. Bereken nu de pariteitsbits:

$p_1 + 1 + 0 + 1 + 0 + 1 + 1 + 1 \equiv even.$ p_1 is dus 1 *om te voldoen aan de regel.*

$p_2 + 1 + 0 + 1 + 0 + 1 + 1 + 1 \equiv even.$ p_2 is dus 1 *om te voldoen aan de regel.*

$p_4 + 0 + 0 + 1 + 0 + 1 + 1 + 1 \equiv even.$ p_4 is dus 0 *om te voldoen aan de regel.*

$p_8 + 0 + 0 + 1 + 0 + 1 + 1 + 1 \equiv even.$ p_8 is dus 0 *om te voldoen aan de regel.*

De code is dus $\mathbf{11}1\mathbf{0}001\mathbf{0}0010111$

- <u>Opgave 18:</u> Maak van de boodschap 100001 een Hamming-code. Doe dit ook voor 1011110001101.

_____

_____

_____

_____

_____

_____

- Opgave 19: Controleer het codewoord 0110010111101 en verbeter. Doe dit ook voor 101000100010111.

- Opgave 20: Codeer het woord *Wiskunde* volgens de ASCII-tabel en gebruik hierbij de (7,4)-Hamming-code door de informatiebits te verdelen in groepjes van vier en hier telkens drie pariteitsbits aan toe te voegen.

- Opgave 21: Decodeer het volgende woord:
  0011011100000110010110010101100111011100000
  De (7,4)-Hamming-code is gebruikt, net als in de vorige opgave.

6. Antwoorden

Opgave 1:
0011 → 0011010
1011 → 1011100

Opgave 2:
0101100 → 0101
1011000 → 1011
1011001 → 1001

Opgave 3:
8028_{dec}
94124_{dec}

Opgave 4:
1402431_{oct}
3407567_{oct}

Opgave 5:
6902_{dec}

Opgave 6:
$3E8_{hex}$

Opgave 7:
Rood, blauw, geel

Opgave 8:
$\#7E4EEA$

Opgave 9:
rgb(175,31,51)

Opgave 10:
$2^8 = 256$

Opgave 11:
110110011_{bin}

Opgave 12:
De rest is 3

Opgave 13:
$27 \equiv \mathbf{5}(mod 11)$ Dus: 5
$-23 \equiv \mathbf{2}(mod 5)$ Dus: 2
$-8^7 = \mathbf{1}(mod 9)$ Dus: 1

Opgave 14:
$6^{89} \equiv \mathbf{6}\ (mod\ 7)$ Dus: 6
$13 \cdot 4^{22} \equiv \mathbf{13}(mod 15)$ Dus: 13
$533.878.220.610 \equiv \mathbf{2}(mod 8)$ Dus: 2

Opgave 15:
$5/3(mod 13) \equiv \mathbf{6}$ Dus: 6

Opgave 16:
[C6253CC]
[37328C4]

Opgave 17:
[9202C]
[A3475]

Opgave 18:
100001: 1010000101

1011110001101: 011001101100011101

Opgave 19:
d_{10} is fout: $011001011\mathbf{0}101$
p_2 is fout: $1\mathbf{0}1000100010111$

Opgave 20:
10110101011010101101010011111010001001011001100110011101011001101100110110
01101011100100101

Opgave 21:
Hoi