Istnieje tylko 10 rodzajów ludzi:

ci, którzy rozumieją liczby binarne

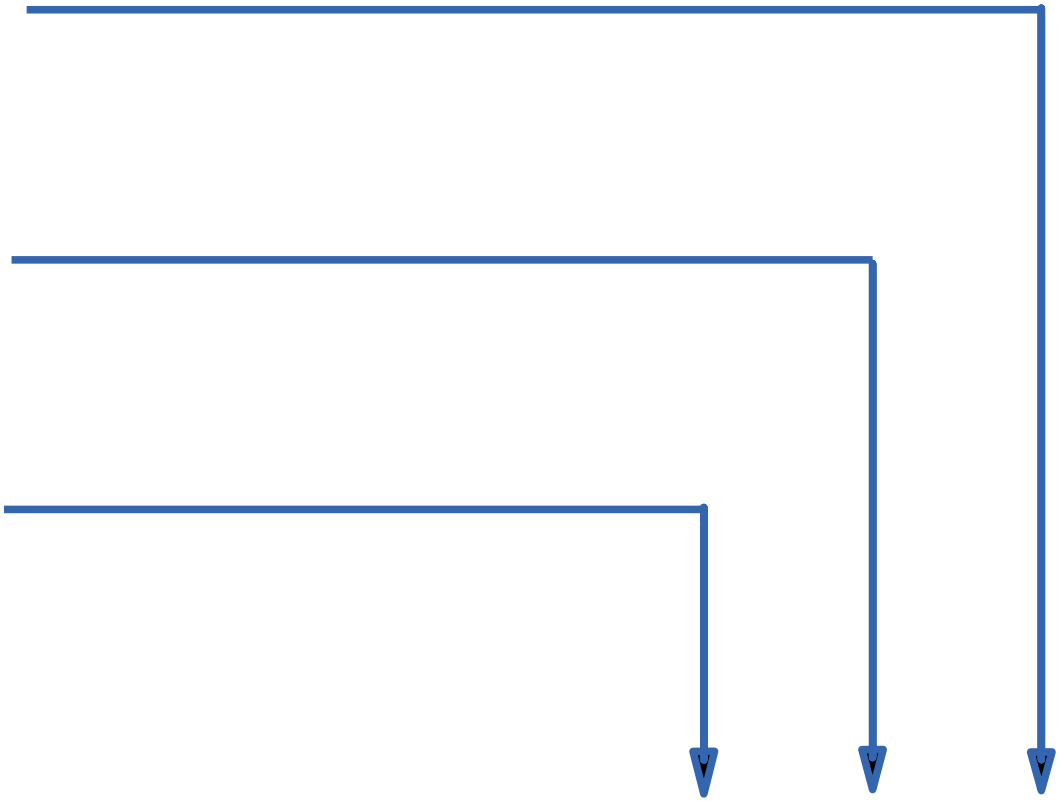
i ci, którzy nie rozumieją.

Zamiana liczby dziesiętnej na binarną

3 : 2 = (*1* • 2) + *1*

1 : 2 = (*0* • 2) + *1*

0 : 2 = (*0* • 2) + *0*



Legenda:

■ → ile dwójek zmieści się w **liczbie**

■ → ile brakuje do początkowej **liczby**

0 1 1

czyli: $\mathbf{3}_{(10)} = \mathbf{011}_{(2)}$

Zamiana liczby dziesiętnej na binarną

$7 : 2 = (3 \cdot 2) + 1$
$3 : 2 = (1 \cdot 2) + 1$ → **111**
$1 : 2 = (0 \cdot 2) + 1$

$6 : 2 = (3 \cdot 2) + 0$
$3 : 2 = (1 \cdot 2) + 1$ → **110**
$1 : 2 = (0 \cdot 2) + 1$

$5 : 2 = (2 \cdot 2) + 1$
$2 : 2 = (1 \cdot 2) + 0$ → **101**
$1 : 2 = (0 \cdot 2) + 1$

$4 : 2 = (2 \cdot 2) + 0$
$2 : 2 = (1 \cdot 2) + 0$ → **100**
$1 : 2 = (0 \cdot 2) + 1$

Zamiana liczby dziesiętnej na binarną

307 : 2 = (153 • 2) + 1

153 : 2 = (76 • 2) + 1

76 : 2 = (38 • 2) + 0

38 : 2 = (19 • 2) + 0

19 : 2 = (9 • 2) + 1

9 : 2 = (4 • 2) + 1

4 : 2 = (2 • 2) + 0

2 : 2 = (1 • 2) + 0

1 : 2 = (0 • 2) + 1

100110011

Zamiana liczby dziesiętnej $\Re$ na binarną

“Wyłapujemy” liczbę przed przecinkiem przy każdym mnożeniu przez 2.

0,5625 → 0
0,5625 • 2 = **1**,1250 → 1
0,1250 • 2 = **0**,2500 → 0
0,2500 • 2 = **0**,5000 → 0
0,5000 • 2 = **1**,0000 → 1

Wynik: $0{,}5625_{(10)} = 01001_{(2)}$

0,875 → 0
0,875 • 2 = **1**,75 → 1
0,75 • 2 = **1**,50 → 1
0,50 • 2 = **1**,00 → 1

Wynik: $0{,}875_{(10)} = 0111_{(2)}$

Dodawanie liczb binarnych

$0 + 0 = \mathbf{0}$

$1 + 0 = \mathbf{1}$

$0 + 1 = \mathbf{1}$

$1 + 1 = \mathbf{0}$
(1 dalej)

Przykład:

$3_{(10)} = 011_{(2)}$

$4_{(10)} = 100_{(2)}$

$$\begin{array}{r} 011 \\ +100 \\ \hline 111 \end{array}$$

Zamiana liczby binarnej na dziesiętną

a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**

$$(2^8 \bullet 1)+(2^7 \bullet 0)+(2^6 \bullet 0)+(2^5 \bullet 1)$$

$$+(2^4 \bullet 1)+(2^3 \bullet 0)+(2^2 \bullet 0)+(2^1 \bullet 1)+(2^0 \bullet 1)$$

$$256+0+0+32+16+0+0+2+1=\mathbf{307}$$

Problem

W jaki sposób przedstawić liczby całkowite

(a więc i ujemne) w systemie binarnym?

Ujemne liczby binarne

$$-7_{(10)} = -111_{(2)}?$$

Nie ma takiego znaku, jak „-” w liczbach binarnych. Jest tylko „1” i „0”!

A więc wprowadzamy dodatkowy „bit znaku” z **lewej** strony:

0 = +

1 = -

Ujemne liczby binarne

$$7_{(10)} = 0111_{(2)}$$

$$-7_{(10)} = 1111_{(2)}$$

Ujemne liczby binarne

Dla przykładowego systemu 3-bitowego, **nadmiarowa** liczba na początku będzie znakiem bitu.
Wady systemu ZM: mamy dwie liczby „zero”: 0 oraz -0. Nie działa także dodawanie liczb dodatnich do ujemnych.

Ujemne liczby binarne

U1 (*Uzupełnienie do jednego*)

$$\mathbf{0111}_{(ZM)} = \mathbf{1000}_{(U1)}$$

W tym systemie, zamieniamy wartościami każdą z liczb.

Ujemne liczby binarne

$-2_{(10)} = 1010_{(ZM)} = 0101_{(U1)}$
$3_{(10)} = 0011_{(ZM)} = 1100_{(U1)}$
Wyszło $2_{(10)}$, a powinno $1_{(10)}$. Winne dwa zera.

$$
\begin{array}{rl}
 & 0101_{(U1)} \\
+ & 1100_{(U1)} \\
\hline
 & 10001_{(U1)} \\
+ & \quad\;\; 1 \\
\hline
= & 0010_{(U1)}
\end{array}
$$

Ujemne liczby binarne

U2 (*Uzupełnienie do dwóch*)

$$\mathbf{0111}_{(ZM)} = \mathbf{1001}_{(U2)}$$

W tym systemie, zamieniamy wartościami każdą z liczb i dodajemy 1. Ostatnie przeniesienie ignorujemy (jeśli istnieje).

Przepełnienie zakresu

Jak wykryć przepełnienie zakresu liczbowego?

Jeśli przeniesienie **do** znaku bitu jest inne niż przeniesienie **ze** znaku bitu – mamy przepełnienie.

$$
\begin{array}{rr}
 & {\scriptstyle 1} \\
 & 0111 \\
+ & \\
 & 0011 \\
\hline
{\scriptstyle 0} & 1010
\end{array}
$$

Przepełnienie zakresu

W języku C++ przepełnienie zakresu przy sumowaniu

*liczb dodatnich może dać liczbę **ujemną**!*

System ósemkowy

Zamiana liczby dziesiętnej na ósemkową

$$307 : 8 = (38 \bullet 8) + \mathbf{3}$$
$$38 : 8 = (4 \bullet 8) + \mathbf{6}$$
$$4 : 8 = (0 \bullet 8) + \mathbf{4}$$

$$307_{(10)} = \mathbf{463}_{(8)}$$

Zamiana liczby ósemkowej na dziesiętną

$$463_{(8)} = (8^2 \bullet \mathbf{4}) + (8^1 \bullet \mathbf{6}) + (8^0 \bullet \mathbf{3}) =$$
$$256 + 48 + 3 = 307_{(10)}$$

System szesnastkowy (heksadeksymalny)

Symbole: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Zamiana liczby dziesiętnej na szesnastkową

$307 : 16 = (19 \bullet 16) + \mathbf{3}$

$19 : 16 = (1 \bullet 16) + \mathbf{3}$

$1 : 16 = (0 \bullet 16) + \mathbf{1}$

$307_{(10)} = \mathbf{133}_{(16)}$

Aby w praktyce łatwiej rozpoznać system szesnastkowy, zapisujemy liczbę w tej postaci:

0x133

System szesnastkowy (heksadeksymalny)

Inny przykład:

$255 : 16 = (15 \bullet 16) + \mathbf{F}$

$15 : 16 = (0 \bullet 16) + \mathbf{F}$

$255_{(10)} = \mathbf{FF}_{(16)}$

Zamiana liczby szesnastkowej na dziesiętną

$(16^1 \bullet 15) + (16^0 \bullet 15) = 240 + 15 = \mathbf{255}$

Kombinatoryka: Permutacja

(przestawianie elementów w zbiorze)

Zadanie 1: Złamać hasło 5-cyfrowe ze zbioru {1, 2, 3, 4, 5}, liczby mogą być użyte tylko raz.

Wzór:

$$\mathbf{P_n = n!}$$

5! = **120** możliwości

Kombinatoryka: Wariacja 1

(każdy element zbioru może być powtórzony wielokrotnie)

Zadanie 2: Złamać hasło 4-cyfrowe ze zbioru {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, liczby mogą powtarzać się.

Wzór:

$$W_{(ie,\ dh)} = ie^{dh}$$

(gdzie „ie” oznacza ilość elementów zbioru, a „dh” oznacza długość hasła)

$$10^4 = \mathbf{10000} \text{ możliwości}$$

Kombinatoryka: Wariacja 2

(bez powtórzeń)

Zadanie 3: Złamać hasło 4-cyfrowe ze zbioru {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, liczby nie powtarzają się w haśle.

Wzór:

$$W_{(ie,\ dh)} = ie! / (ie-dh)!$$

$$10! / (10-4)! = 3628800 / 720 = \mathbf{5040}\ \text{możliwości}$$