POLITECHNIKA GDAŃSKA

Modulo matematyczne a informatyczne

Reszta euklidesowa, reszta po dzieleniu całkowitym i źródła rozbieżności między językami programowania

Piotr Lange

16 kwietnia 2026

Cel prezentacji

Celem prezentacji jest rozdzielenie dwóch pojęć, które w języku potocznym i w dokumentacji języków programowania często bywają mieszane:

1. **modulo w sensie matematycznym**, związane z kongruencją i resztą euklidesową,
2. **operator % w językach programowania**, który bardzo często oznacza jedynie *pewną resztę po dzieleniu całkowitym*.

Przewodnie pytanie brzmi: dlaczego dla liczb ujemnych ten sam zapis może dawać różne wyniki w matematyce, w C i w Pythonie?

Plan

1. Matematyczny punkt wyjścia
2. Skąd biorą się różne wyniki?
3. C a Python
4. Dlaczego projektanci języków wybrali różne reguły?
5. Konsekwencje dla algorytmiki i implementacji

Kongruencja i klasy reszt

Dla ustalonego modułu $m \in \mathbb{Z}^+$ definiujemy relację kongruencji:

$$a \equiv b \pmod{m} \iff m \mid (a - b).$$

Interpretacja

Liczby a i b są równoważne modulo m, gdy mają tę samą resztę euklidesową przy dzieleniu przez m.

Przykład

Dla $m = 5$ liczby

$$-7,\ -2,\ 3,\ 8,\ 13$$

należą do tej samej klasy reszt, ponieważ każda z nich jest kongruentna do 3 modulo 5.

W matematyce dyskretnej pracujemy więc najczęściej nie z dowolnym przedstawicielem klasy, ale z jej **kanonicznym reprezentantem** z przedziału $\{0, 1, \ldots, m-1\}$.

Tw. o dzieleniu euklidesowym

Twierdzenie (Dzielenie euklidesowe)

Dla dowolnych liczb całkowitych $a \in \mathbb{Z}$ oraz $b \in \mathbb{Z} \setminus \{0\}$ istnieje dokładnie jedna para liczb całkowitych q, r taka, że

$$a = bq + r$$

oraz

$$0 \leq r < |b|.$$

Wniosek

Liczba r jest **resztą euklidesową**. Jest ona zawsze nieujemna i jednoznacznie wyznaczona.

Uwaga terminologiczna

W matematyce przez zapis $a \bmod m$ dla $m > 0$ najczęściej rozumie się właśnie tę resztę r.

Co mamy dla liczb ujemnych?

W matematyce znak *dzielnej* lub *dzielnika* nie zmienia definicji reszty euklidesowej: nadal wymagamy, aby

$$0 \leq r < |b|.$$

Przykład 1: ujemna dzielna

$$-7 = 5 \cdot (-2) + 3$$

Zatem

$$-7 \bmod 5 = 3.$$

Przykład 2: ujemny dzielnik

$$7 = (-5) \cdot (-1) + 2$$

Zatem

$$7 \bmod (-5) = 2.$$

Kluczowa obserwacja

Matematycznie znak modułu nie wnosi zmian: kongruencje modulo m i modulo $-m$ są identyczne.

Jedna tożsamość, wiele reszt

W każdym rozsądnym formalizmie chcemy zachować zależność

$$a = bq + r.$$

Problem polega na tym, że dla ustalonych a i b istnieje **wiele** par (q, r) spełniających tę równość.

Przykład: $a = -7$, $b = 5$

Mamy jednocześnie:

$$-7 = 5 \cdot (-2) + 3 \qquad \text{oraz} \qquad -7 = 5 \cdot (-1) + (-2).$$

Obie równości są poprawne algebraicznie. Aby reszta była jednoznaczna, trzeba z góry ustalić **regułę wyboru ilorazu** q. Od tego wyboru zależy wartość r.

Dwa sposoby wyboru

Wariant A: dzielenie euklidesowe

Dobieramy q tak, aby reszta spełniała

$$0 \leq r < |b|.$$

To prowadzi do matematycznej reszty euklidesowej.

Wariant B: dzielenie całkowite zależne od języka

Najpierw definiujemy, czym jest iloraz całkowity q:

- obcięcie do zera,
- zaokrąglenie w dół (*floor*),
- rzadziej inne konwencje.

Potem definiujemy resztę wzorem

$$r = a - bq.$$

Uwaga

Różne języki programowania nie kłócą się o algebrę; one po prostu **inaczej definiują iloraz całkowity**. To automatycznie daje inne reszty.

Uwaga 2

Operator dzielenia całkowitego w Pythonie to //, a w C / o ile oba argumenty są typu całkowitego.

C: iloraz obcinany do zera

W językach z rodziny C (do której należy m.in. C++) przyjmuje się dla liczb całkowitych iloraz obcinany do zera. Dla $a = -7$ i $b = 5$ mamy

$$-7/5 = -1.4 \quad \Longrightarrow \quad q = -1.$$

Zatem $r = -7 - 5 \cdot (-1) = -2$.

Konsekwencja w C

$$-7\%5 = -2.$$

```
-7 / 5  == -1
-7 % 5  == -2
 7 % -5 ==  2
-7 % -5 == -2
```

Charakterystyczna własność

W tej konwencji reszta ma zwykle ten sam znak co **dzielna**.

Python: iloraz równy $\lfloor a/b \rfloor$

W Pythonie operator // oznacza dzielenie całkowite przez **zaokrąglenie w dół**. Dla $a = -7$ i $b = 5$ mamy

$$-7/5 = -1.4 \quad \Longrightarrow \quad q = \lfloor -1.4 \rfloor = -2.$$

Stąd $r = -7 - 5 \cdot (-2) = 3$.

Konsekwencja w Pythonie

$$-7 \% 5 = 3.$$

```
-7 // 5 == -2
-7 %  5 ==  3
 7 % -5 == -3
-7 % -5 == -2
```

Charakterystyczna własność

W Pythonie reszta ma ten sam znak co **dzielnik**.

Porównanie w jednej tabeli

Działanie	Matematyka euklidesowa	C	Python
$7 \bmod 5$	2	2	2
$-7 \bmod 5$	3	-2	3
$7 \bmod (-5)$	2	2	-3
$-7 \bmod (-5)$	3	-2	-2

Jak czytać tę tabelę?

- Matematyka wybiera resztę kanoniczną z przedziału $[0, |b|)$.
- C wybiera iloraz przez obcięcie do zera.
- Python wybiera iloraz przez *floor*, czyli największą liczbę całkowitą nieprzekraczającą $\frac{a}{b}$.

Dlaczego C daje takie wyniki?

1. **Bliskość sprzętu i języków systemowych.** Historycznie języki z rodziny C były projektowane tak, aby arytmetyka całkowita była możliwie bliska zachowaniu maszyn i kompilatorów.
2. **Prosta reguła dla ilorazu.** Obcięcie do zera jest intuicyjnie związane z „ucięciem części ułamkowej”.
3. **Spójność z równaniem** $a = bq + r$. Gdy raz ustalimy $q = \mathrm{trunc}(a/b)$, reszta musi być równa $a - bq$ i wtedy dla liczb ujemnych staje się ujemna.

Efekt uboczny

Operator % w C *nie jest* matematycznym modulo w sensie teorii liczb, lecz resztą skojarzoną z dzieleniem obciętym do zera.

A w Pythonie?

1. **Powiązanie z funkcją podłogi.** Python buduje arytmetykę całkowitą wokół zależności
$$a = b \cdot (a//b) + (a\%b).$$
2. **Dobre własności dla dodatniego modułu.** Dla $m > 0$ otrzymujemy zawsze
$$0 \leq a\%m < m,$$
czyli dokładnie to, czego oczekujemy od reszty euklidesowej.
3. **Wygoda praktyczna.** Takie zachowanie jest bardzo wygodne w cyklicznym indeksowaniu, teorii grafów, algorytmach na klasach reszt i implementacjach struktur periodycznych.

Przykład praktyczny

$$(-1)\%5 = 4,$$

więc przejście „o jeden w lewo po cyklu długości 5” nie wymaga dodatkowej normalizacji wyniku.

Źródło rozbieżności

Matematyka

Najpierw wybieramy **własność reszty**: ma być kanonicznym reprezentantem klasy kongruencji, zwykle z zakresu $0, 1, \ldots, m-1$.

Języki programowania

Najpierw wybieramy **semantykę dzielenia całkowitego**, a dopiero potem definiujemy resztę przez wzór

$$r = a - bq.$$

Główny wniosek

Różne wyniki nie świadczą o tym, że któreś podejście jest „błędne”. Oznaczają tylko, że zapis a % b bywa niejednoznaczny: raz oznacza resztę euklidesową, a raz resztę z konkretnej implementacji dzielenia całkowitego.

Na co trzeba uważać?

1. **Kryptografia, teoria liczb, haszowanie modularne.** Zwykle potrzebujemy reszty euklidesowej dla dodatniego modułu.
2. **Portowanie kodu między językami.** Ten sam zapis `a % m` może zmieniać wynik dla liczb ujemnych.
3. **Indeksowanie cykliczne.** W Pythonie często działa „od razu”, w C trzeba wynik jawnie znormalizować.

Przykładowa normalizacja w językach typu C dla $m > 0$

$$((a\%m) + m)\%m.$$

Przykład

W C dla $a = -7$, $m = 5$ mamy najpierw $a\%m = -2$, a po normalizacji:

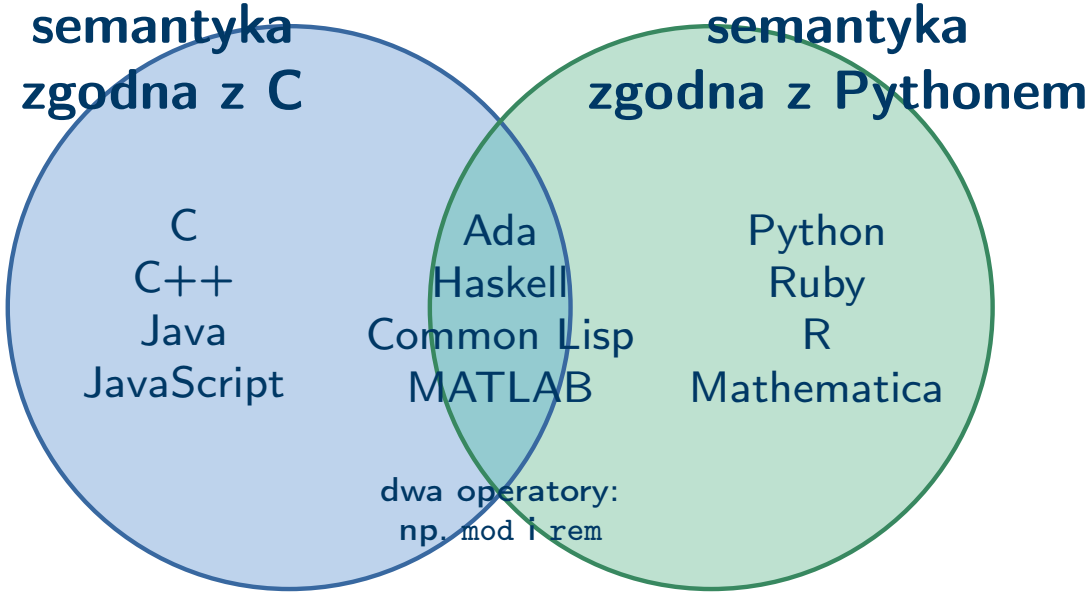
$$((-2) + 5)\%5 = 3.$$

Podsumowanie

1. W matematyce dyskretnej **modulo** jest związane z klasami reszt i resztą euklidesową.
2. Dla $b \neq 0$ reszta euklidesowa r jest określona warunkiem

$$a = bq + r, \qquad 0 \leq r < |b|.$$

3. W programowaniu operator % nie musi oznaczać tej samej operacji pomiędzy językami.



Źródła

Wykorzystane materiały

- A. Szepietowski, *Matematyka dyskretna*.
- W. Guzicki, *Algorytmiczna teoria liczb*.
- J. Topp, *Elementy teorii liczb*.
- Python Software Foundation, *The Python Language Reference*.
- cppreference.com: *C reference*.
- R Core Team, *The R Manuals*.
- MathWorks, *MATLAB Documentation*.
- Wolfram Research, *Wolfram Language Documentation*.

POLITECHNIKA
GDAŃSKA