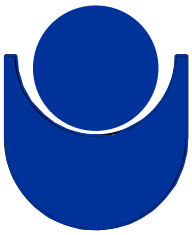

НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ

Департамент Информатика

Проект „*Модулна аритметика*“ към курс CSCB109 Програмиране

гл. ас. д-р Марияна Райкова mariana_sokolova@yahoo.com
гл. ас. д-р Стоян Боев stoyan@nbu.bg

Въведение

Математиката като наука се дели на два основни дяла - чиста и приложна математика. Частта от математиката, разглеждаща целите числа и техните свойства, се нарича теория на числата и до началото на 20 век се е приемала като „най-чистата“ област от чистата математика. С откриването на компютъра и развитието на техниките за криптиране и декриптиране на информация теорията на числата добива все по-голяма актуалност и се отнася все повече към приложната математика.

Ако е дадено едно цяло число m, то съществуват безбройно много цели числа, които дават един и същ остатък при деление с m. Например числата 2, 7, 12, -3 и -8 дават един и същ остатък 2 при деление с 5, а множеството от всички нечетни числа $\{\ldots, -5, -3, -1, 1, 3, 5, \ldots\}$ се състои от всички цели числа, които дават остатък 1 при деление с две. Често пъти в практиката се интересуваме не от самото число, а именно от остатъка, който то дава при деление с дадено цяло число m. Така например, ако сега е 14.00 часа, то след 50 часа ще бъде 16.00 часа, т.е. интересува ни остатъка при деление с 24, колкото часа е едно денонощие. Ако в джоба имаме 3 лева и 60 стотинки и ни върнат 1 и 50 стотинки, то казваме, че разполагаме с 5 лева и 10 стотинки, а не с 4 лева и 110 стотинки. Цифрите, с които изписваме числата в двоична, троична или десетична бройна система са именно остатъците при деление с 2, 3 и 10 съответно.

Постановка

Изготвянето на проекта предполага реализация на C++ на редица функции, свързани с понятията *остатъчен пръстен* и *остатъчно поле*.

Остатъчен пръстен $\mathbb{Z}_n$

Множеството $\mathbb{Z}_n = \{0, 1, \ldots, n\}$ от остатъци *по модул* n (т.е. при деление с n) с дефинирани операции *събиране* $(+)$, *изваждане* $(-)$ и *умножение* $(\times)$ по модул n наричаме *остатъчен пръстен*.

Пример. Ако разгледаме множеството от остатъците по модул 7 (т.е. при деление на 7)

$$\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\},$$

то резултатът от операциите$+$, $-$ и $\times$ винаги ще бъде остатък при деление на 7, т.е. елемент от $\mathbb{Z}_7$:

$3+4 \equiv_7 0$	$2-5 \equiv_7 4$	$2\times 6 \equiv_7 1$
$2+6 \equiv_7 1$	$1-3 \equiv_7 5$	$4\times 4 \equiv_7 2$
$4+5 \equiv_7 2$	$4-5 \equiv_7 6$	$5\times 5 \equiv_7 4$

Задача 1. Да се дефинира C++ функция, която запълва масив с елементите на остатъчния пръстен $\mathbb{Z}_n$.

Задача 2. Да се дефинира C++ функция, която изпълнява операция **събиране** на елементи от остатъчния пръстен $\mathbb{Z}_n$.

Задача 3. Да се дефинира C++ функция, която изпълнява операция **изваждане** на елементи от остатъчния пръстен $\mathbb{Z}_n$.

Задача 4. Да се дефинира C++ функция, която изпълнява операция **умножение** на елементи от остатъчния пръстен $\mathbb{Z}_n$.

Деление в $\mathbb{Z}_n$

В остатъчния пръстен $\mathbb{Z}_n = \{0, 1, \ldots, n\}$ операцията *деление* $(/)$ се дефинира посредством операцията умножение $(\times)$ с реципрочния елемент на делителя. С други думи,

$$a : b = a \times (1/b),$$

където $1/b$ е *реципрочния елемент* на b от пръстена $\mathbb{Z}_n$, т.е. числото, което умножено с b дава 1. Важно е да се отбележи, че това не е възможно, ако реципрочен елемент не съществува. Понякога реципрочния елемент на b се нарича *обратен* и се бележи с b^{-1}.

Пример. В остатъчния пръстен $\mathbb{Z}_7$ имаме

$1/0$	$1/1$	$1/2$	$1/3$	$1/4$	$1/5$	$1/6$
–	1	4	5	2	3	6

и следователно

$$2 : 3 \equiv_7 2 \times (1/3) \equiv_7 2 \times 5 \equiv_7 3$$

$$5 : 6 \equiv_7 5 \times (1/6) \equiv_7 5 \times 6 \equiv_7 2$$

$$3 : 4 \equiv_7 3 \times (1/4) \equiv_7 3 \times 2 \equiv_7 6.$$

Тук не можем да делим само на 0. В същото време, в остатъчния пръстен $\mathbb{Z}_6$ имаме

$1/0$	$1/1$	$1/2$	$1/3$	$1/4$	$1/5$	$1/6$
–	1	–	–	–	5	–

и следователно не можем да делим на 0, 2, 3, 4 и 6.

Забележка. Ако m и n са взаимно прости естествени числа, т.е. $(m, n) = 1$, то от тъжеството на Безу знаем, че съществуват цели числа s и t, такива че $sm + tn = 1$. Тогава s се оказва реципрочният елемент на m в $\mathbb{Z}_n$. В същото време, ако m и n не са взаимно прости естествени числа, то m няма обратен елемент в $\mathbb{Z}_n$.

Пример. Нека $m = 3$, $n = 7$. От тъждеството на Безу следва, че $5m - 2n = 5.3 - 2.7 = 1$ и следователно $5 \times 3 \equiv_7 1$, т.е. $1/3 \equiv_7 5$. Ако обаче $n = 6$, то понеже 3 и 6 не са взаимно прости числа, то 3 няма реципрочен елемент в $\mathbb{Z}_6$.

Задача 5. Да се дефинира C++ функция, която намира двойките реципрочни елементи в $\mathbb{Z}_n$. Намерените елементи да се запишат в двумерен масив.

Задача 6. Да се дефинира C++ функция, която намира **реципрочен (обратен)** на зададен от потребителя елемент от множеството $\mathbb{Z}_n$, ако такъв съществува. Използвайте образувания двумерен масив в горния пример.

Задача 7. Да се дефинира C++ функция, която намира **реципрочен (обратен)** на зададен от потребителя елемент от множеството $\mathbb{Z}_n$, ако такъв съществува. Използвайте тъждеството на Безу и обобщения алгоритъм на Евклид. Сравнете бързодействието с предходното решение.

Задача 8. Да се дефинира C++ функция, която изпълнява операция **деление** на елементи от остатъчния пръстен $\mathbb{Z}_n$ и връща резултат -1, ако не е допустимо.

Степенуване в $\mathbb{Z}_n$

Едни от най-ефективните алгоритми в криптографията изискват повдигане на висока степен на елементи от пръстена $\mathbb{Z}_n$. С други думи, ако $a \in \mathbb{Z}_n$, а m е голямо естествено число, то целта е да пресметнем a^m за оптимално време. Има два подхода:

1) Намираме най-малкото естествено число k, за което $a^k \equiv_n 1$ и пресмятаме

$$a^m \equiv_n a^{m \bmod k}.$$

Пример. Нека $n = 7$, $a = 3$, $m = 100$. Пресмятаме в $\mathbb{Z}_7$ последователно

$$\begin{array}{l} 3^2 = 3.3^1 \equiv_7 2 \\ 3^3 = 3.3^2 \equiv_7 3.2 \equiv_7 6 \\ 3^4 = 3.3^3 \equiv_7 3.6 \equiv_7 4 \\ 3^5 = 3.3^4 \equiv_7 3.4 \equiv_7 5 \\ 3^6 = 3.3^5 \equiv_7 3.5 \equiv_7 1 \end{array}$$

и достигаме до извода, че $k = 6$. Тогава

$$3^{100} \equiv_7 3^{100 \bmod 6} \equiv_7 3^4 \equiv_7 4.$$

2) Представяме m в двоична бройна система и пресмятаме последователно $a, a^2, a^{2^2}, a^{2^3}, \ldots, a^{2^k}$, където k е старшият значещ разряд в двоичното представяне на m.

Пример. Нека $n = 7$, $a = 3$, $m = 100$. Тогава $100_{10} = 64 + 32 + 4 = (1100100)_2$ и последователно пресмятаме:

$$\begin{array}{l} 3^2 = (3^1)^2 \equiv_7 2 \\ 3^4 = (3^2)^2 \equiv_7 2^2 \equiv_7 4 \\ 3^8 = (3^4)^2 \equiv_7 4^2 \equiv_7 2 \\ 3^{16} = (3^8)^2 \equiv_7 2^2 \equiv_7 4 \\ 3^{32} = (3^{16})^2 \equiv_7 4^2 \equiv_7 2 \\ 3^{64} = (3^{32})^2 \equiv_7 2^2 \equiv_7 4 \end{array}.$$

Тогава

$$3^{100} = 3^{64+32+4} \equiv_7 4.2.4 \equiv_7 4.$$

Задача 9. Да се дефинира C++ функция, която извършва **бързо степенуване** на база на въведени остатъчен пръстен, основа и степен от потребителя, използвайки първия подход.

Задача 10. Да се дефинира C++ функция, която извършва **бързо степенуване** на база на въведени остатъчен пръстен, основа и степен от потребителя, използвайки втория подход. Сравнете бързодействието с предходното решение.

Логаритмуване в $\mathbb{Z}_n$

Елементът $g \in \mathbb{Z}_n$ наричаме *примитивен корен* по модул n, ако множеството $G = \{g, g^2, g^3, \ldots\}$ съдържа всички числа $b \in \mathbb{Z}_n$, които са взаимно прости с n. Ако означим с $\varphi(n)$ броят на числата ненадминаващи n и взаимно прости с n, то $G = \{g, g^2, \ldots, g^{\varphi(n)} = 1\}$.

Пример. В остатъчния пръстен $\mathbb{Z}_9$ елементът $g = 2$ е примитивен корен, защото

$$g^1 \equiv_9 2,\ g^2 \equiv_9 4,\ g^3 \equiv_9 8,\ g^4 \equiv_9 7,\ g^5 \equiv_9 5,\ g^6 \equiv_9 1$$

и $G = \{1, 2, 4, 5, 7, 8\}$ е множеството от всички елементи на $\mathbb{Z}_9$, взаимно прости с 9. Но лесно може да се провери, че например в $\mathbb{Z}_8$ или $\mathbb{Z}_{12}$ няма примитивен корен.

Ако g е примитивен корен в $\mathbb{Z}_n$, то от горната дефиниция следва, че сравнението

$$g^x \equiv_n b$$

има единствено решение $x \in \{1, 2, \ldots, \varphi(n)\}$ за произволно $b \in \mathbb{Z}_n$ и b взаимно просто с n. Това решение наричаме *дискретен логаритъм* от b при основа g и означаваме с $dlog_g b$. По презюмпция $dlog_g 1 = 0$, а не $\varphi(n)$.

Пример. Ако $n = 7$, то $\varphi(n) = 6$ и $g = 3$ и 5 са всички примитивни корени в $\mathbb{Z}_7$, защото

g	g^2	g^3	g^4	g^5	g^6
1	1	1	1	1	1
2	4	1	2	4	1
3	**2**	**6**	**4**	**5**	**1**
4	2	1	4	2	1
5	**4**	**6**	**2**	**3**	**1**
6	1	6	1	6	1

Тогава $dlog_3 4 = 4$, $dlog_3 6 = 3$, $dlog_5 3 = 5$, $dlog_5 1 = 0$.

Задача 11. Да се дефинира C++ функция, която **проверява** дали дадено число е примитивен корен в $\mathbb{Z}_n$.

Задача 12. Да се дефинира C++ функция, която **намира всички** примитивни корени в $\mathbb{Z}_n$.

Задача 13. Да се дефинира C++ функция, която изчислява **дискретен логаритъм** в $\mathbb{Z}_n$ при дадени основа и елемент от $\mathbb{Z}_n$ и връща -1, ако това не е допустимо.

Остатъчно поле $\mathbb{F}_n$

Ако n е просто число, то пръстенът $\mathbb{Z}_n$ наричаме *остатъчно поле* и бележим с $\mathbb{F}_n$. Едно от най-важните свойства на $\mathbb{F}_n$ е, че всеки елемент различен от 0 има реципрочен (обратен). С други думи, $\mathbb{F}_n$ е затворено относно операциите събиране, изваждане, умножение и степенуване по подобие на множеството на рационалните числа $\mathbb{Q}_n$, което обаче е безкрайно. Също така се оказва, че във $\mathbb{F}_n$ винаги има поне един примитивен корен, т.е. $\mathbb{F}_n$ е затворено и относно операцията логораритмуване при основа този корен.

Задача 14. Да се дефинира C++ функция, която проверява дали даден остатъчен пръстен е остатъчно поле.

Задача 15.* Да се създаде десктоп приложение за *модулен калкулатор*, който дава възможност за извършване на модулна аритметика в зададено остатъчно поле $\mathbb{F}_n$ с включенни операции - събиране, изваждане, умножение, деление, степенуване и логаритмуване.