СПЕЦІАЛЬНІ РОЗДІЛИ ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №2

Багаторозрядна модулярна арифметика

1. Мета роботи

Отримання практичних навичок програмної реалізації багаторозрядної арифметики; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

2. Теоретичні відомості

2.1. Алгоритм Евкліда

Алгоритм Евкліда обчислює найбільший спільний дільник двох чисел $d = \gcd(a,b)$ шляхом ітеративної процедури, яка ґрунтується на такому факті: якщо $a \geq b$, то $\gcd(a,b) = \gcd(b, a-b)$. Звідси одразу випливає, що $\gcd(a,b) = \gcd(b, a \bmod b)$, і процедура обчислення НСД задається наступним чином.

Нехай $r_0 = a$, $r_1 = b$; обчислюємо послідовність (r_i) для $i \geq 2$ шляхом ділення з остачею:

$$
\begin{aligned}
& r_0 = r_1 q_1 + r_2, \\
& r_1 = r_2 q_2 + r_3, \\
& \dots \\
& r_{s-2} = r_{s-1} q_{s-1} + r_s; \\
& r_{s-1} = r_s q_s.
\end{aligned}
$$

Якщо на відповідному кроці виявилось, що $r_{s+1} = 0$, то $d = r_s$.

Складність алгоритму Евкліда є лінійною по відношенню до бітової довжини n аргументів. Дійсно, найгірший випадок для роботи алгоритму – коли всі $q_i = 1$. Цей випадок відповідає ситуації, коли a та b – два послідовні числа Фібоначчі. З формули Біне випливає, що число Фібоначчі асимптотично веде себе як $f_n \sim \phi^n$, де $\phi = \dfrac{\sqrt{5}+1}{2}$ – відношення «золотого перерізу», а тому алгоритм Евкліда виконає не більше ніж $\left| \log_\phi a \right| = O(\log a) = O(n)$ операцій.

Розширений алгоритм Евкліда обчислює дві додаткові послідовності (u_i) та (v_i) такі, що на кожному кроці виконується рівність $r_i = u_i a + v_i b$; зокрема, для найбільшого спільного дільника матимемо $d = r_s = u_s a + v_s b$. Ці послідовності також можна обчислити рекурентно за допомогою часток q_i:

$$u_0 = 1,\ u_1 = 0,\ u_{i+1} = u_{i-1} - q_i u_i;$$
$$v_0 = 0,\ v_1 = 1,\ v_{i+1} = v_{i-1} - q_i v_i.$$

Зауважимо, що хоча алгоритм Евкліда є лінійним за кількістю операцій, він використовує операції ділення з остачею, які самі по собі є важкими. Для запобігання цього був розроблений інший алгоритм, що має назву *бінарний алгоритм обчислення НСД* або *алгоритм Стейна*. Цей алгоритм базується на таких спостереженнях:

1) якщо і a, і b – парні, то $\gcd(a,b) = 2\gcd\left(\frac{a}{2}, \frac{b}{2}\right)$;
2) якщо a – парне, а b – непарне, то $\gcd(a,b) = \gcd\left(\frac{a}{2}, b\right)$;
3) якщо і a, і b – непарні, то $\gcd(a,b) = \gcd\left(\min\{a,b\}, |a-b|\right)$, причому різниця є парним числом.

На відміну від алгоритму Евкліда, бінарний алгоритм використовує лише віднімання та ділення на два, яке у двійкових архітектурах ефективно реалізується як бітовий зсув.

Отже, бінарний алгоритм можна подати у вигляді такої процедури.

```
d := 1;
while (a – парне) and (b парне) do:  // виокремлення загальної парної частини
      a := a / 2;
      b := b / 2;
      d := d * 2;

while (a – парне) do:
      a := a / 2;

while (b <> 0) do:
      while (b – парне) do:
            b := b / 2;
      (a, b) := (min{a, b}, abs(a – b))

d := d * a;
return d;
```

На кожному кроці одне з двох оброблюваних чисел скорочується на один біт (шляхом ділення на два), тому бінарний алгоритм виконає не більш ніж $2\log a$ кроків. Це більше, ніж в класичному алгоритмі Евкліда, але використання віднімання та зсувів замість ділення на практиці робить бінарний алгоритм суттєво швидшим.

2.2. Редукція за Барреттом

В багатьох асиметричних криптографічних алгоритмах потрібно виконувати обчислення за модулем деякого натурального числа. Прямолінійний підхід полягає в тому, щоб застосовувати ділення з остачею після кожної арифметичної операції; однак, як зазначалось, ділення є дуже складною операцією, тому були розроблені спеціальні алгоритми, які б дозволяли замінити ділення на інші, більш прості операції.

Перевага таких алгоритмів відчутна тоді, коли потрібно виконувати багато обчислень за одним й тим самим модулем. В цьому випадку за рахунок деяких передобчислень вдається

суттєво зекономити на обчисленнях в кожному конкретному випадку, що призводить до значного пришвидшення обчислень в цілому.

Розглянемо перший з таких методів – *алгоритм модулярної редукції Барретта*.

Отже, нехай дано багаторозрядні числа n та x у системі числення із основою β, причому довжина x вдвічі більша за n: $|n| = k$, $|x| = 2k$. Необхідно знайти частку q та остачу r від ділення x на n: $x = qn + r$, $0 \le r < n$.

Алгоритм Барретта передбачає «вгадування» частки q – точніше, її оцінку. Дійсно, можемо записати:

$$\frac{x}{n} = \frac{x}{\beta^{k-1}} \cdot \frac{\beta^{2k}}{n} \cdot \frac{1}{\beta^{k+1}},$$

$$q = \left\lfloor \frac{x}{n} \right\rfloor = \left\lfloor \frac{\frac{x}{\beta^{k-1}} \cdot \frac{\beta^{2k}}{n}}{\beta^{k+1}} \right\rfloor \ge \left\lfloor \frac{\left\lfloor \frac{x}{\beta^{k-1}} \right\rfloor \cdot \left\lfloor \frac{\beta^{2k}}{n} \right\rfloor}{\beta^{k+1}} \right\rfloor = \hat{q},$$

причому у виразі для $\hat{q}$ ділення на степені β насправді є лише відкиданням останніх цифр числа (тобто ніякого ділення там не відбувається), а множник $\mu = \left\lfloor \frac{\beta^{2k}}{n} \right\rfloor$ не залежить від x і тому може бути передобчислений. Барретт довів, що якщо $x \le n^2$, то знайдена таким чином частка $\hat{q}$ відрізняється від справжнього значення q не більш ніж на 2, причому в 90% випадків $\hat{q}$ та q взагалі збігаються.

Зауваження: для коректного обчислення μ необхідно правильно визначити степінь β^{2k}, яка має $2k+1$ цифру в записі (на одну більше, ніж в записі x); це може бути критичним при реалізації у моделі із фіксованою довжиною числа.

Алгоритм редукції за Барретом можна подати у вигляді такої процедури.

Процедура BarrettReduction (*x, n, μ, r*)

Вхід: багаторозрядні числа *x, n,* передобчислене значення $\mu = \left\lfloor \frac{\beta^{2k}}{n} \right\rfloor$.

Вихід: багаторозрядне число $r = x \bmod n$.

```
q := KillLastDigits(x, k-1);        // відкидання останніх k-1 цифри
q := q * μ;
q := KillLastDigits(q, k+1);
r := x – q * n;
while (r >= n) do:                  // Барретт гарантує, що цикл виконується
      r := r – n;                   //    не більше двох разів
return r;
```

Бачимо, що редукція за Барретом для обчислення остачі використовує не ділення, а множення (та декілька зсувів та віднімань), що значно пришвидшує обчислення за модулем.

Покажемо застосування редукції за Барретом на прикладі схеми Горнера. Саме в схемі Горнера під час піднесення до степеня потрібно виконувати багато операцій множення за одним модулем, що є необхідною передумовою для ефективного застосування редукції за Барретом.

Процедура LongModPowerBarrett (*A, B, N, C*)

Вхід: багаторозрядні числа *A, B, N*; *B* задане двійковим записом $B = b_{m-1}2^{m-1} + ... + b_1 2 + b_0$.

Вихід: багаторозрядне число $C = A^B \bmod N$.

```
C := 1;
μ := LongShiftDigitsToHigh(1, 2*k) / n;        // єдине ділення!
for i := 0 to m-1 do:
      if b[i] = 1 then:
            C := BarrettReduction(C * A, N, μ);
      A := BarrettReduction (A * A, N, μ);
return C
```

2.3. Редукція за Монтгомері

На відміну від метода Барретта, Монтгомері запропонував для обчислення лишків взагалі перейти у іншу арифметичну систему, в якій редукція сама по собі виконується значно швидше.

Нехай n – непарне число, а $R > n$ – число, взаємно просте із n (зазвичай для зручності обирають $R = 2^t$). *Лишком Монтгомері* числа x називають вираз $mont(x) = xR \bmod n$. *Функцією редукції Монтгомері* називають функцію $redc(x) = x \cdot R^{-1} \bmod n$, де R^{-1} – число, обернене до R за модулем n. Система лишків Монтгомері утворює арифметику із такими операціями:

$$mont(x \pm y) = mont(x) \pm mont(y),$$
$$mont(x \cdot y) = redc(mont(x) \cdot mont(y)),$$

і треба мати на увазі, що $redc(mont(x)) = x \bmod n$. Таким чином, довільний арифметичний алгоритм, який використовує лише додавання, віднімання та множення, може бути переписаний для системи лишків Монтгомері таким чином:

1) всі вхідні змінні x та константи замінюються на лишки Монтгомері $mont(x)$;

2) всі множення xy замінюються на $redc(xy)$;

3) всі вихідні змінні z замінюються на $redc(z)$.

Наприклад, схема Горнера у системі лишків Монтгомері буде виглядати так:

Процедура LongModPowerMontgomery (*A, B, N, C*)

Вхід: багаторозрядні числа *A, B, N*; *B* задане двійковим записом $B = b_{m-1}2^{m-1} + ... + b_1 2 + b_0$.

Вихід: багаторозрядне число $C = A^B \bmod N$.

```
A := mont(A);
C := mont(1);
for i := 0 to m-1 do:
      if b[i] = 1 then:
            C := redc(C * A);
      A := redc(A * A);
return redc(C)
```

Головною перевагою методу Монтгомері є обчислення функції $redc(x)$, яке може бути виконане суттєво швидше, ніж звичайне ділення з остачею. Операція ділення залишається лише при початкових обчисленнях $mont(x)$.

Покажемо схематично, як швидко обчислювати значення $redc(x)$.

1) За допомогою розширеного алгоритму Евкліда знаходяться такі числа R^{-1} та n', що $RR^{-1} - nn' = 1$ (зауважимо, що R^{-1} – це обернений до R за модулем n). Цей крок є передобчисленням.

2) Обчислити $u = x + (x \cdot n' \bmod R) \cdot n$. Оскільки внутрішнє множення береться за модулем $R = 2^t$, то для його обчислення достатньо взяти по t біт аргументів, що прискорює обчислення; також відмітимо, що операція $\bmod R$ – це просто одержання останніх t біт числа.

3) Обчислити $u = u / R$ (тобто у числа u відкидаються останні t біт). Якщо $u \geq n$, то $u = u - n$. Повернути u як $redc(x)$.

Цю процедуру також можна пришвидшити, якщо звести відповідні обчислення до окремих цифр результату. Детальніше про це наведено у конспекті лекцій.

3. Завдання до комп'ютерного практикуму

А) Доопрацювати бібліотеку для роботи з m-бітними цілими числами, створену на комп'ютерному практикумі №1, додавши до неї такі операції:

1) обчислення НСД та НСК двох чисел;
2) додавання чисел за модулем;
3) віднімання чисел за модулем;
4) множення чисел та піднесення чисел до квадрату за модулем;
5) піднесення числа до багаторозрядного степеня d по модулю n.

Модулярну арифметику рекомендовано реалізовувати на базі редукції Баррета, піднесення до степеня – на базі схеми Горнера. Мова програмування, семантика функцій та спосіб реалізації можуть обиратись довільним чином.

Окрім основного завдання, ви також можете виконати додаткове завдання згідно варіанту.

Таблиця 3.1 – Варіанти додаткових завдань.

Номер варіанту	**Додаткове завдання**
1	Реалізація операції піднесення до квадрату (звичайного та модулярного) більш ефективним методом, ніж звичайним множенням; порівняння ефективності.
2	Реалізація ділення класичним алгоритмом «у стовпчик» (див. [1], [2]), порівняння ефективності із алгоритмом «зсувай@віднімай»
3	Реалізація множення шляхом однократного застосування методу Карацуби, порівняння ефективності із звичайним множенням
4	Реалізація множення шляхом ітеративного застосування методу Карацуби, порівняння ефективності із звичайним множенням
5	Реалізація модулярного множення алгоритмом Блеклі (див. [5]), порівняння ефективності із звичайним модулярним множенням.
6	Реалізація модулярного піднесення до степеню віконним методом (4 біти), порівняння ефективності із методом Горнера
7	Реалізація модулярного піднесення до степеню віконними методами (2-8 біт), порівняння відносної ефективності цих методів між собою
8	Реалізація модулярних операцій на основі редукції Барретта та редукції Монтгомері, порівняння ефективності.

9	Обчислення НСД алгоритмом Евкліда та бінарним алгоритмом, порівняння їх ефективності; обчислення НСК
10	Розв'язування систем лінійних порівнянь за китайською теоремою про лишки; порівняння ефективності прямолінійного алгоритму та методу Гарнера.
11	Обчислення символів Якобі, оцінка трудомісткості алгоритму в залежності від довжини чисел; порівняння ефективності із обчисленням за критерієм Ойлера (для простих модулів).
12	Обчислення оберненого елементу за модулем; порівняння ефективності для обчислення за допомогою розширеного алгоритму Евкліда та за допомогою малої теореми Ферма (для простих модулів).
13	Реалізація тесту Соловея-Штрассена, оцінка швидкодії тесту
14	Реалізація тесту Міллера-Рабіна, оцінка швидкодії тесту
15	Обчислення звичайних квадратних (та, за бажанням кубічних) коренів.
16	Обчислення квадратних коренів за простим модулем.
17	Реалізація множення квадратних матриць; порівняння ефективності для множення прямолінійним методом та ітеративним застосуванням методу Штрассена.
18	Обчислення поліноміальних функцій за модулем (за допомогою схеми Горнера), оцінка швидкодії.

Б) Проконтролювати коректність реалізації алгоритмів; зокрема, для декількох багаторозрядних a, b, c, n перевірити тотожності:

1. $(a+b)\cdot c \equiv c\cdot(a+b) \equiv a\cdot c + b\cdot c \pmod n$;
2. $n\cdot a \equiv \underbrace{a+a+\ldots+a}_{n} \pmod m$, де n повинно бути не менш за 100;
3. $a^{\varphi(n)} = 1 \bmod n$ (за умови $\gcd(a,n)=1$).

Перевірити останню тотожність для простого n; перевірити для $n = 3^k$ та довільного $a \not\vdots 3$: $\varphi(3^k) = 3^k - 3^{k-1} = 2\cdot 3^{k-1}$, отже, має виконуватись $a^{\varphi(3^k)} = a^{2\cdot 3^{k-1}} = 1 \bmod 3^k$.

Продумати та реалізувати свої тести на коректність.

Для перевірки роботи операцій із простими числами можна використовувати заздалегідь відомі прості числа (наприклад, числа Мерсенна).

В) Обчислити середній час виконання реалізованих арифметичних операцій. Підрахувати кількість тактів процесора (або інших одиниць виміру часу) на кожну операцію. Результати подати у вигляді таблиць або діаграм.

Продемонструвати працюючу програму викладачеві!
(бажано компілювати на місці, щоб була можливість змінювати програму)

4. Оформлення звіту

Звіт оформлюється відповідно до стандартних правил оформлення наукових робіт. Звіт має містити:

1) мету, теоретичну частину та завдання, наведені вище, згідно варіанту;
2) стисло викладені теоретичні відомості по додатковому завданню;
3) бітові/символьні зображення тестових чисел та результатів операцій над ними;
4) бітові/символьні зображення результатів контролю за пунктом Б);
5) аналітичні відомості за пунктом В) завдання;
6) текст програми (у додатку)

5. Оцінювання лабораторної роботи

За виконання лабораторної роботи студент може одержати до 12 рейтингових балів; зокрема, оцінюються такі позиції:

– реалізація основного завдання – до 6-ти балів;
– реалізація додаткового завдання – до 4-х балів;
– оформлення звіту – 1 бал;
– своєчасне виконання основного завдання – 1 бал;
– несвоєчасне виконання роботи – (-1) бал за кожні два тижні пропуску.

6. Рекомендована література

1. *Д. Кнут*. Искусство программирования на ЭВМ. Т.2. – М.:Мир, 1976. –724 с.
2. *О.Н. Василенко.* Теоретико-числове алгоритмы в криптографии
3. *А.А. Болотов.* Алгоритмические основы современной криптологии
4. *А.В. Анісімов.* Алгоритмічна теорія великих чисел. Модулярна арифметика великих чисел. – К.: Академкнига, 2001.
5. *Çetin Kaya Koç*. High-Speed RSA Implementation.