КОМП'ЮТЕРНИЙ ПРАКТИКУМ №1
З КУРСУ
ТЕОРЕТИКО-ЧИСЛОВІ АЛГОРИТМИ В КРИПТОЛОГІЇ

Пошук канонічного розкладу великого числа, використовуючи відомі методи факторизації

1 Мета роботи

Практичне ознайомлення з різними методами факторизації чисел, реалізація цих методів і їх порівняння. Виділення переваг, недоліків та особливостей застосування алгоритмів факторизації. Застосування комбінації алгоритмів факторизації для пошуку канонічного розкладу заданого числа.

2 Необхідні теоретичні відомості

Факторизація цілого числа — це пошук нетривіального дільника (одного, не обов'язково простого) заданого числа. Якщо велике складене число розкладено на добуток менших цілих чисел, кожне з яких є простим числом, то такий розклад називається *канонічним розкладом складеного числа*. Таким чином, завданням задачі пошуку канонічного розкладу натурального числа є пошук всіх його простих дільників. Доведено, що задача пошуку канонічного розкладу числа та задача факторизації є поліноміально еквівалентними за часовою складністю. Ці задачі вважаються складними, тобто на сьогодні не існує ефективного алгоритму їх розв'язання в загальному випадку. Це й обумовлює використання цих задач в будові багатьох криптографічних примітивів, наприклад, криптосистеми RSA.

Однак існує багато алгоритмів факторизації, що мають експоненційну або субекспоненційну оцінку складності, або, наприклад є ефективними для деяких часткових випадків. В цьому комп'ютерному практикумі ми на практиці ознайомимося з деякими з них, а саме з методом пробних ділень, ρ-методом Полларда, алгоритмом Брілхарта-Моррісона (CFRAC) та алгоритмом Померанця (квадратичного сита), та спробуємо поєднати ці алгоритми для пошуку канонічного розкладу великого складеного числа.

Для розв'язання задачі пошуку канонічного розкладу числа необхідними є тести на простоту для перевірки чи є частка від ділення числа на простий дільник простим числом, чи необхідно продовжити розклад.

2.1 Тести перевірки натуральних чисел на простоту

Для побудови, формального опису та оцінки коректності та ефективності тестів на простоту використовуються так звані псевдопрості числа — складені числа, що зберігають ті чи інші властивості простих.

Означення 1. Непарне число p називається *псевдопростим за основою* $a \in \mathbb{N}$, якщо $\gcd(a,p) = 1$ і $a^{p-1} \equiv 1(\mod p)$, де $\gcd(a,b)$ (від англ. *greatest common divisor*) — найбільший спільний дільник чисел a та b.

Означення 2. Непарне число p називається *псевдопростим числом Ойлера за основою* $a \in \mathbb{N}$, якщо $\gcd(a,p) = 1$ та $(\frac{a}{p}) \equiv a^{\frac{p-1}{2}} (\mod p)$, де $(\frac{a}{p})$ — символ Якобі.

Означення 3. Нехай p — непарне число, $p-1 = d \cdot 2^s$, d — непарне. Число p називається *сильним псевдопростим числом за основою* $a \in \mathbb{N}$, якщо $a^d \equiv 1(\mod p)$ або $a^{d \cdot 2^r} \equiv -1(\mod p)$ при якомусь $r : 0 \leq r < s$.

2.1.1 Імовірнісний тест Соловея-Штрассена

Перевіримо число $p \in \mathbb{N}$. Оберемо деяке число $k \in \mathbb{N}$.

0. Встановлюємо лічильник 0.
1. Вибираємо випадкове число $x \in \mathbb{N}$ з інтервалу $1 < x < p$ (незалежне від раніше обраних x, якщо такі були). За допомогою алгоритму Евкліда знаходимо $\gcd(x,p)$. Якщо $\gcd(x,p) = 1$, то переходимо до кроку 2, якщо $\gcd(x,p) > 1$, то p — складене число, алгоритм завершує свою роботу.
2. Перевіряємо, чи є p псевдопростим Ойлера за основою x. Якщо p виявилось не псевдопростим за основою x, то p — складене число, алгоритм завершує свою роботу; інакше лічильник збільшується на 1.
3. Якщо лічильник менший за k, то переходимо до кроку 1, інакше алгоритм завершує роботу.

У результаті число p буде перевірено тестом Соловея-Штрассена при k різних основах. Якщо p виявиться не псевдопростим хоча б за однією основою, то p складене. Якщо p псевдопросте за всіма цими основами, то вважаємо p простим числом.

Тест Соловея-Штрассена є одностороннім, тобто в ньому відсутні помилки першого роду: якщо p — просте число, то тест завжди покаже, що воно просте. Однак, якщо p — число складене, то для кожної можливої основи тест із імовірністю $\frac{1}{2}$ поверне помилкову відповідь (повідомлення, що p — просте). Використання k різних випадкових основ зменшує помилку тесту до 2^{-k}.

2.1.2 Імовірнісний тест Міллера-Рабіна

Перевіримо число $p \in \mathbb{N}$. Оберемо деяке число $k \in \mathbb{N}$.

0. Знаходимо розклад $p-1 = d \cdot 2^s$ та встановлюємо лічильник 0.
1. Вибираємо випадкове число $x \in \mathbb{N}$ з інтервалу $1 < x < p$ (незалежне від раніше обраних x, якщо такі були). За допомогою алгоритму Евкліда знаходимо $\gcd(x,p)$. Якщо $\gcd(x,p) = 1$, то переходимо до кроку 2, якщо $\gcd(x,p) > 1$, то p — складене число, алгоритм завершує свою роботу.
2. Перевіряємо, чи є p сильно псевдопростим за основою x:
   (а) Якщо $x^d \bmod p = \pm 1$, то p є сильно псевдопростим за основою x.
   (б) Інакше для всіх $r = \overline{1, s-1}$ виконати такі дії:
      i. Обчислити $x_r = x^{d \cdot 2^r} \bmod p$ (тобто $x_r = x_{r-1}^2 \bmod p$).

ii. Якщо $x_r = -1$, то p є сильно псевдопростим за основою x, якщо $x_r = 1$, то p не є сильно псевдопростим за основою x, інакше перейти до наступного значення r.

(в) Якщо за кроки 2а та 2б для усіх r не було встановлено, що p є сильно псевдопростим за основою x, то p не є сильно псевдопростим за основою x.

3. Якщо p виявилось не сильно псевдопростим за основою x, то p — складене число, алгоритм завершує свою роботу; інакше лічильник збільшується на 1.

4. Якщо лічильник менший за k, то переходимо до кроку 1, інакше алгоритм завершує роботу.

У результаті число p буде перевірено тестом Міллера-Рабіна при k різних основах. Якщо p виявиться не сильно псевдопростим хоча б за однією основою, то p складене. Якщо p сильно псевдопросте за всіма цими основами, то вважаємо p простим числом.

Тест Міллера-Рабіна є одностороннім, тобто в ньому відсутні помилки першого роду: якщо p — просте число, то тест завжди поверне, що воно просте. Однак, якщо p — число складене, то для кожної можливої основи тест із імовірністю $\frac{1}{4}$ поверне помилкову відповідь (повідомлення, що p — просте). Використання k різних випадкових основ зменшує помилку тесту до 4^{-k}.

2.2 Метод пробних ділень

Найпростішим рішенням для виконання факторизації цілого числа є *метод пробних ділень*. Він полягає у перевірці на ділення без остачі числа n почергово на набір дільників з множини простих чисел 2, 3, 5, 7, 9, 11, ..., $[\sqrt{n}]$. На практиці метод пробних ділень використовується як для перевірки числа на простоту (первинний етап), так і для пошуку найменших простих дільників числа в задачі факторизації.

Для використання методу пробних ділень для довільних малих дільників використовують так звану *ознаку подільності Паскаля*. Нехай натуральне число n записане у системі числення з основою B:

$$n = \overline{a_t a_{t-1} ... a_1 a_0} = a_t B^t + a_{t-1} B^{t-1} + ... + a_1 B + a_0,$$

де $0 \leq a_i < B - 1$. Тоді для довільного (малого) m, кандидата в дільники, виконується рівність:

$$n \equiv \sum_{i=0}^{t} a_i r_i (\mathrm{mod}\ m),$$

де $r_i = B^i \bmod m$. Сума в правій частині не перевищує величини tmB, що є значно меншим за оригінальне N. Послідовність r_i не залежить від N та може бути обчислена заздалегідь за рекурентною формулою $r_0 = 1$, $r_{i+1} = r_i \cdot B \bmod m$. Більш того, послідовність r_i є періодичною, що спрощує обчислення.

2.3 ρ-метод Полларда

Алгоритм для факторизації цілих чисел, що був запропонований Джоном Поллардом в 1974 році є найбільш ефективним для пошуку малих дільників числа. Складність алгоритму оцінюється як $O(n^{\frac{1}{4}})$.

Суть ρ-методу Полларда полягає в побудові числової послідовності, елементи якої, починаючи з деякого номера, утворюють цикл. Приклад такого циклу зображено на рисунку 1.

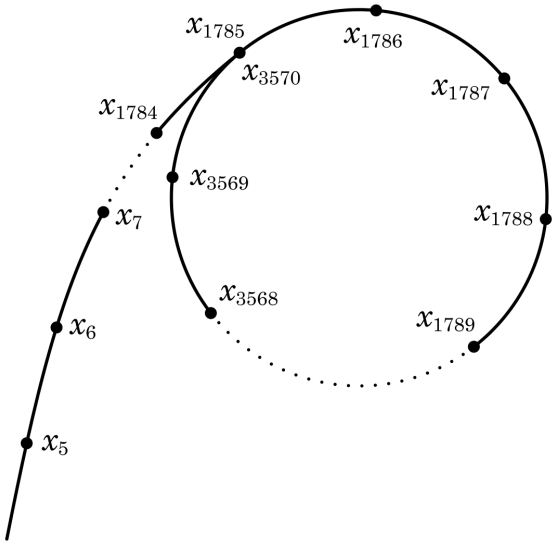

Рис. 1: Приклад зациклення числової послідовності у ρ-методі Полларда

ρ-метод Полларда є алгоритмом типу Las-Vegas, що означає, що він не завжди видає якийсь результат, але якщо видає результат, то він точно правильний.

Вхідними даними для алгоритму є n — складене число, яке треба факторизувати, і деяка функція $f : S \to S$, де S — скінченна множина. Зазвичай використовується функція $f(x) = (x^2 + 1) \bmod n$, але зустрічається і використання інших функцій. Виходом алгоритму є або нетривіальний дільник числа n, або повідомлення, що дільник не знайдено.

Для початку роботи алгоритму потрібно вибрати початкове значення $x_0 \in \mathbb{Z}_n$, зазвичай беруть $x_0 = 2$. Далі обчислюється послідовність x_i за таким правилом $x_i = f(x_{i-1})$, тобто $x_1 = f(x_0)$, $x_2 = f(x_1) = f(f(x_0))$ і т.д. Отримана послідовність x_i є скінченною, бо S — скінченна множина, отже, починаючи з якогось номеру, послідовність зациклиться. Далі, для кожного x_k розглядаються різниці $(x_k - x_j)$, де $j < k$, і обраховуються $d = \gcd(x_k - x_j, n)$. Якщо знаходиться таке $d \neq 1$, воно і є нетривіальним дільником n, інакше алгоритм повертає повідомлення, що дільника не знайдено. У випадку, якщо дільника не знайдено, потрібно знову запустити алгоритм з інакшим початковим значенням x_0.

Для зменшення кількості ітерацій, Робертом Флойдом була запропонована модифікація алгоритму. Алгоритм починає роботу з пари (x_0, x_0) та ітеративно обраховує пари (x_1, x_2), (x_2, x_4), (x_3, x_6), ... В кожній ітерації функція $f(x)$ застосовується один раз до першого елементу пари і двічі до другого елементу пари. Потім розглядається різниця елементів обчислених пар. Таким чином, модифікація Флойда ρ-методу Полларда має вигляд:

1. $x_0 = 2, y_0 = 2, d = 1$
2. $x_i = f(x_{i-1})$,
   $y_i = f(f(y_{i-1}))$,
   $d = \gcd(x_i - y_i, n)$.
3. Якщо $x = y$, зі заданими початковими значеннями алгоритм не знайшов дільника числа n. Повернутися на крок 1 та поміняти початкові значення.
4. Якщо $d \neq 1$, алгоритм закінчує роботу зі знайденим дільником d, інакше повернутися на крок 2.

2.4 Метод Брілхарта-Моррісона

Основна ідея алгоритму Брілхарта-Моррісона полягає в заміні випадкового (як в алгоритмі Діксона) вибору чисел, квадрат яких за модулем перевіряється на гладкість по факторній базі, на повністю детермінований пошук. Цей вибір чисел тут здійснюється за допомогою ланцюгових дробів числа $\sqrt{n}$, де n — число, що треба факторизувати.

Алгоритм

1. Побудувати факторну базу $\mathcal{B} = \{-1,\ p_1,\ p_2,\ ...,\ p_k\}$, де $p_i, i = \overline{1,k}$ — прості числа, такі що символ Лежандра $(\frac{n}{p_i}) = 1$ і $1 < p_i < L^a$, де $L = \exp((\log n \log\log n)^{1/2})$, a — деяка константа. В нашому випадку візьмемо $a = 1/\sqrt{2}$.

2. За допомогою розкладення числа $\sqrt{n}$ в ланцюговий дріб $[a_0, a_1, ..., a_k, a_{k+1}]$, знайти $k+1$ $\mathcal{B}$-гладких чисел $b_i^2 \bmod n$, де b_i обчислюється з a_i за допомогою рекурентної формули:

$$b_i = a_i \cdot b_{i-1} + b_{i-2}, \quad \text{причому } b_{-1} = 1,\ b_{-2} = 0.$$

   Розкласти $\sqrt{n}$ в ланцюговий дріб $[a_0, a_1, ..., a_k, a_{k+1}]$ можна за допомогою таких співвідношень:

$$v_0 = 1 \qquad v_{i+1} = \frac{n - u_i^2}{v_i}$$

$$\alpha_0 = \sqrt{n} \qquad \alpha_{i+1} = \frac{\sqrt{n} + u_i}{v_{i+1}}$$

$$a_0 = [\alpha_0] \qquad a_{i+1} = [\alpha_i]$$

$$u_0 = a_0 \qquad u_{i+1} = a_{i+1}v_{i+1} - u_i$$

3. Нехай $\overrightarrow{s_i} = (\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,k})$ — вектор степенів розкладу числа b_i^2 по факторній базі $\mathcal{B}$. Далі потрібно розв'язати систему рівнянь

$$x_o \cdot \overrightarrow{s_0} + x_1 \cdot \overrightarrow{s_1} + ... + x_k \cdot \overrightarrow{s_k} + x_{k+1} \cdot \overrightarrow{s_{k+1}} = 0,\ x_i \in \{0, 1\},$$

   та знайти значення $x_0, x_1, ..., x_{k+1}$, не всі рівні нулю. Такий розв'язок існує, оскільки кількість невідомих (k) менше кількості рівнянь ($k+1$). Розв'язати систему лінійний рівнянь можна будь-яким відомим способом.

4. Знайти

$$X = \prod_{i=0}^{k+1} b_i^{x_i},$$

$$Y = \prod_{j=0}^{k} p_j^{\frac{\sum_{i=0}^{k+1} x_i \cdot \alpha_{i,j}}{2}}$$

5. Далі перевіряємо умову $1 < \gcd(X \pm Y, n) < n$. У випадку успіху, $\gcd(X \pm Y, n)$ і є шуканим дільником числа n. Якщо дільника не знайдемо, повертаємось на крок 1 та збільшуємо значення константи a.

2.5 Метод Померанця

Відмінність методу Померанця від алгоритму Брілхарта-Морісона полягає у способі формування множини чисел, квадрат яких потенційно є $\mathcal{B}$-числами. Основна ідея в тому, щоб відкидати частину чисел, які з великою ймовірністю не є $\mathcal{B}$-числами, до застосування методу пробних ділень до цих чисел. Такий підхід реалізовується етапом *просіювання*.

Більш детально алгоритм Померанця розглядається на лекційних заняттях.

3 Порядок виконання роботи і методичні вказівки

-1. Ознайомитись з порядком виконання комп'ютерного практикуму та відповідними вимогами до виконання роботи.

0. Уважно прочитати необхідні теоретичні відомості до комп'ютерного практикуму.

1. Написати програми, що реалізовують такі алгоритми:

   (а) один з тестів на простоту (Міллера-Рабіна або Соловея-Штрассена);

   (б) метод пробних ділень;

   (в) ρ-метод Полларда;

   (г) метод Брілхарта-Моррісона.

2. Створити та реалізувати алгоритм для пошуку канонічного розкладу числа, що використовує (всі) вищезгадані алгоритми:

   (а) Вхід: n. Перевірити чи вхідне число n є простим. Якщо число є простим, то додати n до результату і завершити роботу, якщо число є складеним, перейти до наступного кроку.

   (б) Вхід: n. Застосувати метод пробних ділень для пошуку дільників вхідного числа. Пробні ділення робити числами, що не перевищують 47. Якщо дільник a знайдено, записати його у вихідний результат і повернутися до кроку 2а з вхідним значенням n/a. Якщо дільник не знайдено перейти до кроку 2г.

   (в) Вхід: n. Перевірити чи вхідне число є простим. Якщо число є простим, то додати n до результату і завершити роботу, якщо число є складеним, перейти до наступного кроку.

   (г) Вхід: n. Застосувати ρ-метод Полларда. Якщо дільник a знайдено, записати його у вихідний результат і повернутися на один крок назад з числом n/a. Якщо дільник не знайдено перейти до кроку 2е.

   (д) Вхід: n. Перевірити чи вхідне число є простим. Якщо число є простим, то додати n до результату і завершити роботу, якщо число є складеним, перейти до наступного кроку.

   (е) Вхід: n. Застосувати метод Брілхарта-Моррісона. Якщо дільник a знайдено, записати його у вихідний результат і повернутися на один крок назад з числом n/a. Якщо дільник не знайдено завершити роботу з повідомленням "я не можу знайти канонічний розклад числа :(".

3. Застосувати алгоритм, створений на попередньому кроці, для пошуку канонічного розкладу числа відповідного варіанту.

Варіант	Число
1	49347803087
2	40314824977
3	56872139357
4	25169050345
5	17807879769
6	37007068943
7	36478753357
8	44729396957
9	62728529929
10	20057299527

4. Застосувати реалізовані алгоритми факторизації (ρ-метод Полларда та метод Брілхарта-Моррісона) почергово до всіх наступних чисел, знайти один дільник цього числа, заміряти час роботи кожного алгоритму на кожному числі та порівняти результати. Числа:

- 9172639163
- 8627969789
- 8937716743
- 278874899
- 99400891
- 116381389
- 4252083239
- 6633776623
- 227349247
- 3568572617

5. Оформити звіт до комп'ютерного практикуму.

Додаткове завдання #1**:** Замість методу Брілхарта-Моррісона реалізувати метод Померанця і використовувати його. Бонус за це завдання +3 бали до рейтингу.

Додаткове завдання #2**:** Реалізувати розв'язання системи лінійних рівнянь одним з відомих методів, наприклад, за допомогою алгоритму Ланцоша, Відемана або Гауса. У такому випадку, звіт має містити уточнення про використання цього методу, опис алгоритму та заміри часу його роботи. Бонус за це завдання +3 бали до рейтингу.

4 Оформлення звіту та порядок захисту комп'ютерного практикуму

Звіт про виконання комп'ютерного практикуму оформлюється згідно зі стандартними правилами оформлення наукових робіт. Рекомендується виконувати звіти за допомогою системи набору і верстки LaTeX, причому дозволяється використовувати розмір шрифту 12pt та одинарний міжрядковий інтервал.

Звіт обов'язково має містити:

- мету комп'ютерного практикуму;
- постановку задачі та варіант завдання;
- хід роботи, опис труднощів, що виникали під час виконання завдання, та шляхи їх подолання;
- результати дослідження, зокрема результат канонічного розкладу числа відповідного варіанту (можна кількох варіантів);
- результати продуктивності роботи програмної реалізації алгоритмів (заміри часу роботи на числах різної довжини, як для алгоритму перевірки на простоту, так і для алгоритмів факторизації);
- оцінку максимального порядку числа, що піддається розкладу створеною програмною реалізацією за час, що не дорівнює часу життя всесвіту;
- висновки до роботи (зокрема аналіз ефективності реалізованих алгоритмів, рекомендації щодо вибору алгоритму(-ів) факторизації для застосування в криптографічних цілях тощо).

Тексти коду програми дозволяється не включати у звіт. Тексти всіх програм здаються викладачу в електронному вигляді для перевірки на плагіат. До захисту практичної частини комп'ютерного практикуму допускаються студенти, які оформили звіт та надали його для перевірки викладачу. Для зарахування комп'ютерного практикуму студенту необхідно виконати захист практичної частини роботи та тексти програм даного студента повинні пройти перевірку на плагіат.

Оцінювання комп'ютерного практикуму

Можлива кількість рейтингових балів 9

- Програмна реалізація 7
- Звіт 2

Додаткове завдання #1 ..... +3 бали до рейтингу

Додаткове завдання #2 ..... +3 бали до рейтингу

Несвоєчасне виконання роботи ..... -1 бал за кожне заняття пропуску

Академічний плагіат до 10% ..... 0 балів за комп'ютерний практикум, -10 балів до рейтингу

Академічний плагіат більше 10% ..... анулювання балів за семестр та недопуск до заліку