УДК 511.3

ОПТИМІЗАЦІЇ АЛГОРИТМУ КВАДРАТИЧНОГО РЕШЕТА QS

О.І. Ізюмченко, Л. В. Ізюмченко

Розглянуто алгоритм квадратичного решета факторизації чисел, отримано оптимізації методу QS та розроблена програмна реалізація для оптимізацій алгоритму квадратичного решета.

The quadratic sieve algorithm of integer factorization was examined, the QS method optimization was received and software implementation for quadratic sieve algorithm optimizations were developed.

Питання захисту інформації є однією з перспективних галузей прикладної математики. На сьогоднішній день існує величезна кількість криптографічних алгоритмів, що відрізняються як своїми загальними характеристиками, так і принципами, на яких базується їх робота. Не всі вони є однаково надійними – серед них є навіть такі, що оформлені як стандарти та при цьому майже не забезпечують реального захисту. Насправді створення надійного криптографічного алгоритму є досить важкою задачею. Крім того, надійність є відносна річ: багато з раніше розроблених алгоритмів, які вважалися надійними, тепер або ненадійні, або їх надійність викликає великий сумнів. Тому при розробці криптографічного алгоритму необхідно враховувати тенденції розвитку комп'ютерної техніки, а також інші фактори, що потенційно можуть знизити його стійкість в майбутньому.

Стійкість багатьох сучасних криптосистем базується на складності задачі факторизації. Можна виділити два основних типи алгоритмів розкладу на множники – спеціалізовані й універсальні. Спеціалізовані алгоритми розкладу на множники використовують спеціальні особливості числа n, що факторизується. Один з найбільш потужних спеціалізованих алгоритмів розкладу на множники – метод еліптичних кривих (режим виправлення помилок), що був винайдений в 1985 році Х. Ленстром молодшим. Час виконання цього методу залежить від розміру головних множників n, і відповідно алгоритм має тенденцію знаходити спочатку маленькі множники. До розвитку RSA системи шифрування, найкращим універсальним алгоритмом розкладу на множники був алгоритм ланцюгового поділу.

Найкращі універсальні алгоритми розкладу на множники, що використовуються сьогодні – квадратичне решето (QS) та решето цифрового поля (NFS). Експерименти довели, що NFS є дійсно добрим алгоритмом для розкладу на множники чисел, що мають принаймні 120 десяткових цифр (400 бітів). Квадратичне решето було розроблене Карлом Померанцом у 1984 р. Алгоритм квадратичного решета (англ. quadratic sieve, QS) – це алгоритм факторизації цілих чисел і, на практиці, другий за швидкістю з відомих (після загального решета цифрового поля) і помітно простіший, ніж решето цифрового поля. Це метод факторизації загального призначення, тобто час його виконання залежить лише від розміру цілого числа на вході, а не на особливостях його структури. Дослідження присвячене саме алгоритму QS квадратичного решета, створенню його оптимізацій та розробці програмної реалізації для алгоритму квадратичного решета.

Теоретичними основами задачі факторизації є теорія конгруенцій, у т.ч. конгруенцій 2-го степеня, застосування символів Лежандра і Якобі; імовірнісні та детерміновані тести простоти чисел, добування квадратного кореня з квадратичного лишка, зокрема, алгоритм Шенкса-Тонеллі. У роботі оцінено складність виконання факторизації методом квадратичного решета та порівняно з складністю алгоритму факторизації пробними діленнями; розглянуті існуючі та розроблені нові оптимізації для алгоритму квадратичного решета; розглянуті варіанти паралельного виконання алгоритму, у тому числі було спроектовано та реалізовано програмний засіб на основі алгоритму QS для факторизації чисел. Практична реалізація виконана мовою `Java`.

Вибір мови програмування є важливою частиною роботи в реалізації певної задачі. Ознакою правильного вибору мови програмування може бути використання для розробки програмної реалізації існуючих бібліотек цієї мови. Важливою частиною розробки програмної реалізації алгоритму QS є довга арифметика. Початковим набором мов програмування для написання програми, що реалізує алгоритм квадратичного решета, були `C++`, `Java` і `Python`. `C++` потенціально найбільш швидка мова з них, але в стандартних бібліотеках мови

C++ немає реалізованої довгої арифметики, тоді як в Java і Python вона є. Мова Python значно повільніша, ніж Java та C++. З цих міркувань мовою для написання програми було обрано Java.

Ідея методу Ферма. Нехай $n = p \cdot q -$ відоме ціле число, що є добутком двох невідомих простих чисел p і q, які потрібно знайти. Більшість сучасних методів факторизації ґрунтується на ідеї, запропонованій ще П'єром Ферма, що полягає у пошуку пар натуральних чисел A і B таких, що виконується співвідношення

$$A^2 - B^2 = n. \qquad (1)$$

Алгоритм Ферма має наступний вигляд:

1. Обчислимо цілу частину від кореня квадратного із n: $m = [\sqrt{n}]$.

2. Для $x = 1, 2, ...$ будемо обчислювати значення

$$q(x) = (m + x)^2 - n, \qquad (2)$$

до тих пір, доки наступне значення $q(x)$ не виявиться рівним повному квадрату.

3. Нехай $q(x)$ є повним квадратом числа B: $q(x) = B^2$. Визначимо $A = m + x$, звідки з рівності $A^2 - n = B^2$ знайдемо $n = A^2 - B^2$, $n = (A - B)(A + B)$ і шукані дільники p і q обчислюються, як $p = A + B$ і $q = A - B$.

Ідея методу Моріса Крейтчика. У 20-х роках XX сторіччя Моріс Крейтчик, узагальнюючи ідею Ферма, запропонував замість пар чисел, які задовольняють рівнянню $A^2 - B^2 = n$, шукати пари чисел, що задовольняють більш загальному рівнянню

$$A^2 \equiv B^2 \pmod{n}. \qquad (3)$$

Крейтчик помітив, що багато із чисел $q(x)$ в (2) розкладаються в добуток невеликих простих чисел, тобто є гладкими за сучасною термінологією. Розглянемо приклад із [1, с. 115], зауважимо, що у [1] при наборі допущено помилки, тому розв'язання наводимо заново:

Нехай число $n = 2041$ треба розкласти. Найближчим до n числом, що є повним квадратом, є $m = 45$, $m^2 = 2025$. Розглянемо послідовність пар чисел

$\{x; q(x)\}$, де $q(x)$ обчислюються за формулами (2), а x приймає близькі до нуля значення: {(–2; –192), (–1; –105), (0; –16), (1; 75), (2; 168), (3; 263), (4; 360), (5; 459), (6; 560)}.

Можна помітити, що другі координати деяких із цих пар розкладаються у добуток невеликих простих чисел: $-192 = -2^6 \cdot 3$, $-105 = -3 \cdot 5 \cdot 7$, $-16 = -2^4$, $75 = 3 \cdot 5^2$, $168 = 2^3 \cdot 3 \cdot 7$, $360 = 2^3 \cdot 3^2 \cdot 5$, $560 = 2^4 \cdot 5 \cdot 7$.

Виберемо множину невеликих простих чисел і назвемо її факторною базою FB = {2; 3; 5; 7}. Будь-яке ціле число, що може бути розкладене в добуток множників з факторної бази, назвемо гладким відносно цієї факторної бази. Таким чином, наведені вище числа є гладкими. Коли факторна база вибрана, будь-яке гладке число можна зобразити вектором показників степенів $\bar{v} = (r_1, r_2, ..., r_k)$, де k – розмір факторної бази. Наприклад, числу 168 відповідає вектор $\bar{v} = (3, 1, 0, 1)$.

Далі Крейтчик помітив, що можна перемножати деякі гладкі числа так, щоб отримувати повні квадрати:

$75 \cdot 168 \cdot 360 \cdot 560 = 2^{10} \cdot 3^4 \cdot 5^4 \cdot 7^2 = 50400^2$, $(-192) \cdot (-16) \cdot 75 = 2^{10} \cdot 3^2 \cdot 5^2 = 480^2$.

Відмітимо, що при множенні чисел, їх вектори показників степенів додаються. Для того, щоб добуток гладких чисел був повним квадратом, необхідно підібрати таку комбінацію співмножників, щоб сума векторів показників мала тільки парні координати, наприклад, добуткові $75 \cdot 168 \cdot 360 \cdot 560$ відповідає сума векторів

(0; 1; 2; 0) + (3; 1; 0; 1) + (3; 2; 1; 0) + (4; 0; 1; 1) = (10; 4; 4; 2).

Замість цілочисельних координат векторів степенів і їхніх сум достатньо розглядати їхні остачі по модулю 2, тобто виконувати усі обчислення у полі $F_2 = \{0;1\}$, тоді добуток елементів є повним квадратом тоді і тільки тоді, коли вектор-сума по модулю 2 усіх векторів-показників є нульовим вектором. Множина усіх можливих векторів розмірності k над полем $F_2 = \{0;1\}$ утворює лінійний простір L_k розмірності k, а тому будь-яка множина векторів, що містить більше, ніж k елементів, є лінійно залежною, а тому існує нетривіальна

лінійна комбінація таких векторів, що дорівнює нульовому вектору. Коефіцієнти цієї лінійної комбінації можна знайти, розв'язуючи систему лінійних рівнянь, складених з коефіцієнтів степенів, взятих за модулем 2.

Оскільки коефіцієнти цієї лінійної комбінації рівні або 0, або 1, то вона являє собою просто суму деякої підмножини векторів. Звідси випливає той факт, що для знаходження нетривіальної підмножини гладких чисел, добуток яких є повний квадрат, достатньо мати $k+1$ гладке число, де k – розмір факторної бази.

Коли набір гладких пар, що містить не менш ніж $k+1$ елементів, знайдений, то розв'язання рівняння (1) можна знайти, розв'язуючи відповідну систему лінійних рівнянь, наприклад, за допомогою методу виключення невідомих Гаусса.

Ідея методу QS. Першим кроком алгоритму квадратичного решета є визначення факторної бази. Для оптимальної швидкодії межа для факторної бази обирається в залежності від числа, яке потрібно факторизувати, і обчислюється за формулою: $e^{\sqrt{\frac{\sqrt{2}}{4}\cdot\ln(n)\cdot\ln(\ln(n))}}$, де n – число, яке треба факторизувати. Далі, для n шукаються такі x, для яких значення полінома $Q(x)=(x+m)^2-n$, де $m=\lfloor\sqrt{n}\rfloor$, є гладкими за основою цієї бази. Для швидкого знаходження таких значень використовується ідея Померанца для квадратичного решета [1]:

Ідея Померанца полягала в тому, що якщо для простого числа $p\in FB$ знайдено аргумент x такий, що $q(x)\equiv 0(\mathrm{mod}\,p)$, то на p будуть ділитися всі елементи $q(y)$, де y відрізняється від x на аргумент, кратний p, тобто $y=x+k\cdot p$, $k\in Z$. А тому, якщо для даного p знайдено корінь x рівняння $q(x)\equiv 0(\mathrm{mod}\,p)$, то для усіх y, $y\equiv x(\mathrm{mod}\,p)$ буде також виконуватися умова $q(y)\equiv 0(\mathrm{mod}\,p)$.

Алгоритм Померанца працює наступним чином:

1. Вибирається деякий числовий інтервал $[-L;L]$, що називається

інтервалом просіювання;

2. У масив цілих чисел $W[-L..L]$ заносяться значення поліному $q(x)$ для $x \in [-L; L]$;

3. Для кожного числа p із факторної бази FB шукають числа $0 \le x < p$ такі, що виконується конгруенція

$$q(x) \equiv 0 (\mathrm{mod}\, p) \qquad (4)$$

Відмітимо, що ця конгруенція матиме або два розв'язки, або жодного.

4. Для кожного розв'язку x рівняння (4) в циклі продивляються числа виду $x_k = x + k \cdot p$ з інтервалу $[-L; L]$, $k \in Z$, і виконується поділ елементів $W[x_k]$ на p.

Процедура повторюється для усіх чисел p з факторної бази і їхніх степенів $p^k < B$, обмежених зверху границею факторної бази FB.

У результаті виконання процедури просіювання деякі елементи $W[x]$ масива W стануть рівними ± 1. Відповідні пари $(x; q(x))$ є гладкими. Для надійного знаходження нетривіального розв'язку рівняння (3) необхідно, щоб число знайдених гладких пар перевищувало, щонайменше, на 1 розмір факторної бази. Тоді із коефіцієнтів розкладу гладких чисел за елементами факторної бази можна скласти невизначену систему лінійних рівнянь, яка матиме нетривіальний розв'язок.

Відмітимо, що наявність нетривіального розв'язку гарантує нам знаходження пари натуральних чисел (A, B), що задовольняють (3). Проте не всі пари такого роду дають шуканий розв'язок, а тому для надійності отримання нетривіальних дільників n кількість гладких чисел повинна бути більшою за розмірність факторної бази на число $k > 1$, наприклад, на $k = 3$ або $k = 4$.

Додатковою перевагою методу Померанца є можливість розпаралелювання обчислень і запуску алгоритму для одночасного просіювання на декількох комп'ютерах. Метод Померанца отримав назву квадратичного решета. Використовуючи цей метод, у 1994 році Аткінс, Граф, Лейланд і Ленстра зуміли розкласти 129-значне число, запропоноване засновниками RSA.

Процедура просіювання є найбільш затратною по часу частиною алгоритму квадратичного решета. У ході її виконання шукаються пари чисел (A,B), що задовольняють

$$A^2 \equiv B \pmod{n}. \qquad (5)$$

Розмір факторної бази є одним з ключевих параметрів алгоритму просіювання, що визначає його ефективність. Якщо її розмір є надто малим, то гладкі числа будуть попадатися дуже рідко або не будуть попадатися взагалі. При недостатньому розмірі факторної бази приходиться вибирати великий радіус L інтервалу просіювання, що спричиняє збільшення загального часу обчислень.

Якщо ж розмір факторної бази надто великий, то вимагається пошук великої кількості гладких чисел, що також збільшує загальний час обчислень. Відзначимо, що для розкладу 129-значного числа засновників RSA була використана факторна база, що налічувала 524338 простих чисел.

Позначимо розмір факторної бази через k. Для розкладу числа n необхідно відшукати k' гладких пар (A,B), $k' \geq k+2$. Після цього формується система лінійних рівнянь розмірності $k \times k'$ з коефіцієнтами із поля $F_2 = \{0;1\}$ і нульовим стовпцем вільних членів. Система не є визначеною і тому має нетривіальний розв'язок, що представляє множину M перших аргументів гладких пар $(x, q(x))$. Тоді пара (A,B), що задовольняє рівняння (3), знаходиться як

$$A = \prod_{x \in M} (x+m) \pmod{n}, \qquad B = \prod_{x \in M} q(x) \pmod{n} \qquad (6)$$

Тепер дільники p вихідного числа n можна знайти, обчислюючи НСД $(n, A \pm B)$. У деяких випадках обидва знайдених дільники виявляються тривіальними (рівними 1 або n), тоді потрібно шукати інший розв'язок системи і іншу пару (A,B).

Оптимізації методу QS.

Для підвищення швидкодії та зменшенні використовуваної пам'яті було застосовано наступний метод перевірки гладкості полінома: замість значення полінома для аргументу x зберігається значення його натурального логарифма з

деякою точністю (в програмній реалізації `Java` використовується звичайне число з плаваючою точкою подвійної точності `double`), замість операції ділення на число з факторної бази використовується віднімання логарифма відповідного числа; для того щоб визначити, чи відбувся повний розклад значення полінома $Q(x)$ в факторній базі, перевіряється приблизна рівність нулю поточного значення логарифма полінома, оскільки логарифм рівний нулю, коли його аргумент рівний одиниці, що і означає те, що значення полінома є гладким відносно факторної бази. Ця оптимізація допомагає зберігти значну частину пам'яті через значну різницю в кількості пам'яті, яку займає число та його логарифм з константною точністю.

Модифікований метод Гаусса. Для кожного знайденого гладкого у факторній базі значення полінома записується вектор його степенів розкладу у факторній базі. Після цього шукаються нетривіальні лінійні комбінації векторів, що рівні нуль-вектору. Цю задачу було розв'язано деякою модифікацією методу Гаусса. В основу даної модифікації було покладено метод пошуку оберненої матриці за допомогою дописування до неї справа одиничної матриці. З векторів степенів розкладу у факторній базі значень полінома, як із рядків, складається матриця. Далі до неї справа дописується одинична квадратна матриця, яку надалі називатимемо допоміжною частиною матриці, а початкову частину матриці – основною. Далі послідовно для кожного рядка матриці виконуються наступні дії:

1) Якщо в основній частині матриці існує ненульовий елемент, то використовуючи його, методом елементарних перетворень над рядками, отримуються нулі в стовпчику з ненульовим елементом у всіх інших рядках матриці.

2) Якщо в основній частині матриці всі елементи нульові, то маємо нетривіальну лінійну комбінацію векторів, що рівна нулю за модулем два. Коефіцієнти цієї комбінації записані в допоміжній частині матриці.

Метод масок. Нескладними математичними міркуваннями отримаємо факт, що лінійна комбінація нетривіальних лінійних комбінацій є нетривіальною

лінійною комбінацією. Після виконання модифікованого методу Гаусса отримано деяку кількість нетривіальних лінійних комбінацій. Множину цих комбінацій будемо називати базовими лінійними комбінаціями. Для знаходження усіх нетривіальних лінійних комбінацій, що являють собою лінійні комбінації базових нетривіальних лінійних комбінацій, можна застосувати метод масок.

Метод масок – це комп'ютерний алгоритм, типовим застосуванням якого є знаходження всіх підмножин заданої множини. Для реалізації методу масок потрібно знати кількість елементів у множині. Нехай у множині k елементів. Щоб згенерувати всі підмножини заданої множини, для кожного числа від 0 до $2^k - 1$, знаходиться їх бінарне представлення. Кожному бінарному представленню числа однозначно відповідає одна з підмножин множини з k елементами. Для того, щоб по бінарному представленню числа знайти відповідну йому множину, використовується наступне правило:

1) Якщо i-й розряд бінарного представлення числа рівний 0, то i-й елемент множини не входить у цю підмножину.

2) Якщо i-й розряд представлення числа рівний 1, то i-й елемент множини входить у цю підмножину.

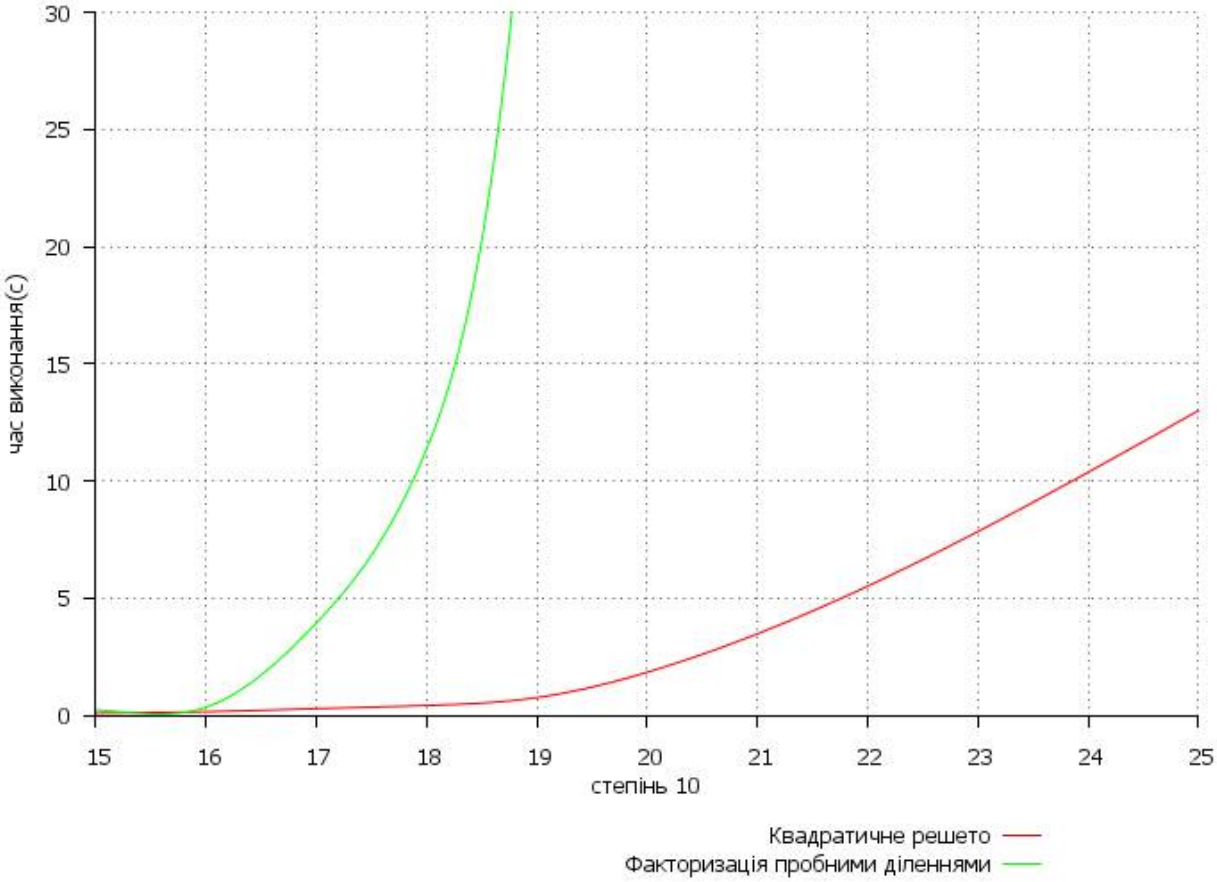
Нескладно довести, що цей метод згенерує всі підмножини заданої множини, і що жодна з підмножин не буде згенерована більше одного разу.

Для того, щоб застосувати метод масок для знаходження всіх нетривіальних лінійних комбінацій, маючи базові нетривіальні лінійні комбінації, треба для кожного числа від 1 до $2^l - 1$ знайти аналогічним вищеописаному способом індекси базових нетривіальних лінійних комбінацій та додати їх (де l – кількість базових нетривіальних комбінацій). На відміну від класичного методу масок, число 0 не потрібно розглядати, так як воно відповідає тривіальній лінійній кобмінації, яка не цікавить нас в даній задачі. Далі застосовується універсальний алгоритм QS.

Для оцінювання ефективності реалізації алгоритму квадратичного решета проведено її порівняння з програмною реалізацією алгоритму факторизації

пробними діленнями. Тестувались програми на 2-х ядерному комп'ютері з частотою процесора 1.3 Ghz та оперативною пам'яттю 3 Gb.

Наступні дані містять інформацію по часу виконання вищезгаданих програмних реалізацій алгоритмів факторизації (порядок числа; час виконання у с за методом квадратичного решета; час у с пробними діленнями): 10^{15}, 0.096, 0.240; 10^{16}, 0.163, 0.352; 10^{17}, 0.292, 3.944; 10^{18}, 0.428, 11.36; 10^{19}, 0.766, 42.17; 10^{20}, 1.847, 136.87; 10^{25}, 13.01, –; 10^{30}, 129.144, –.



Як видно з даних, квадратичне решето працює швидше, ніж факторизація пробними діленнями навіть для досить невеликих чисел. Крім того, з графіка видно, що при збільшенні порядку числа швидкість алгоритму квадратичного решета змінюється досить плавно, на відміну від алгоритму факторизації пробними діленнями. З цього можна зробити висновок, що програмна реалізація є достатньо якісною.

ПОСИЛАННЯ

[1] Ишмухаметов Ш.Т. *Методы факторизации натуральных чисел: учебное пособие.* – Казань: Казанский университет, 2011. – 190 с.