Number Theoretic Algorithms

큰 수의 소인수분해

May 22, 2019

1 큰 그림

큰 수의 소인수분해를 더 빠르게 만드는 기법은 전부[1] 지난 주에 다루었던 $x^2 = y^2$의 비자명해를 찾는 것으로, 그 방법으로 특히 $x^2 = y$인 B-smooth y를 $\pi(B)$개에 근접하게 찾는 방법을 이용하여 시간을 줄이는 방법을 이용합니다.

소인수분해 알고리즘은 크게 두 가지 부류로 나뉘는데,

- 특수 목적 소인수분해 알고리즘; 이 알고리즘은 수행 시간이 n의 크기 이외의 것에 의존하는 알고리즘들을 일컫습니다. 예를 들어 Pollard's ρ algorithm은 수행 시간이 가장 작은 소인수의 크기에 의존합니다. Special Number Field Sieve는 아예 한 술 더 떠서, 충분히 작은 m, b, c에 대해 $n = m^b \pm c$의 꼴일 것을 강제합니다.
- 일반 목적 소인수분해 알고리즘; 이 알고리즘은 수행 시간이 n의 크기에만 의존합니다. 예를 들어 여기서 다룰 Dixon's Factorization Algorithm, Quadratic Sieve, 그리고 다루지는 않지만 General Number Field Sieve가 여기에 속합니다.

여기서의 주 목표인 Quadratic Sieve는 현재 알려진 실용적인[2] 일반 목적 소인수분해 알고리즘 중에서 두 번째로 빠릅니다.

또, General Number Field Sieve는 Special Number Field Sieve의 확장인데, 이 확장 과정에 시간이 많이 소모되어 실제로 구현할 경우 십진수로 100자리 정도까지는 Quadratic Sieve가 가장 빠르다고 합니다.

2 Dixon's Factorization Algorithm

기본적으로 적당한 B에 대해 $x^2 = y$인 B-smooth y를 $k := \pi(B) + \mathcal{O}(1)$개 찾습니다. 그런데 Dixon's Factorization에서는 x를 **랜덤으로** 골라서 찾습니다. 가장 단순한 형태의 알고리즘이라고 할 수 있습니다. 이 알고리즘의 수행 시간은 B를 얼마나 크게 잡느냐에 따라 결정됩니다.

$x^2 \mod n$이 균등하게 분포되어 있다고 가정합니다. 그러면 n 이하의 모든 자연수 중에서 B-smooth인 수의 개수는 대략 $\Psi(n, B) = nu^{-u}$ where $u = \log n / \log B$임이 알려져

[1] 확실한 속도 향상이 있는 알고리즘 전부.

[2] 양자컴퓨터 위에서 돌아가는 Shor의 알고리즘을 제외한다는 의미로 받아들이시면 되겠습니다. Shor의 알고리즘은 양자컴퓨터에서 사용할 수 있는 $\mathcal{O}(\log^2 n)$ FFT에 기반을 두고 있습니다.

있고, 이 사실을 사용할 수 있습니다. 즉, B개를 찾기 위해서는 평균적으로 Bu^u번의 시도가 필요합니다. 이 값을 (WolframAlpha 등을 사용해) 최소화하는 B를 찾으면

$$B = \exp\left(\frac{1}{\sqrt{2}}\sqrt{\log n}\sqrt{W(2e^2 \log n)}\right)$$

으로 구해집니다. W의 값과 다른 기본 연산들의 값을 빠르게 구할 수 있으므로, 컴퓨터가 실제로 B를 찾을 수 있습니다.

B의 크기를 대강 따져 보면,

$$\begin{aligned}\log B &= \frac{1}{\sqrt{2}}\sqrt{\log n}\sqrt{W(2e^2 \log n)} \\ &\approx \frac{1}{\sqrt{2}}\sqrt{\log n}\sqrt{\log 2e^2 + \log\log n - \log\log\log n} \\ &\approx \frac{1}{\sqrt{2}}\sqrt{\log n \log\log n}\end{aligned}$$

이므로 n이 100자리쯤일 때는 B가 feasible한 크기가 된다는 사실을 알 수 있습니다.

이 알고리즘은 Quadratic Sieve로 가기 위한 발판이기 때문에 소개하기도 했지만, 이 알고리즘 자체에서도 의미를 찾을 수 있습니다: 최초의 병렬화 가능한 소인수분해 알고리즘입니다.

3 Quadratic Sieve

여기까지 와서 우리가 소인수분해의 방법론에 대해 크게 오해할 수 있는 것이 하나 있는데, 그것은 $x^2 = y$를 찾을 때 y가 반드시 x^2을 n으로 나눌 나머지일 필요는 없다는 점입니다. 단순히 $x^2 - n$의 계산 결과이기만 해도 좋습니다. 우리는 y가 작으면 y의 소인수도 당연히 작다는 사실을 이용하기 위해 여태까지 n으로 나눠 오고 있었습니다.

대신에 그냥 $x \approx \sqrt{n}$ 근처의 값만을 보는 것은 어떨까요? 어차피 우리는 $\sqrt{n}$이 충분히 큰 경우만 볼 것이므로, $\sqrt{n}$과 $\sqrt{2n}$의 차이는 크고, 따라서 $\sqrt{n}$보다 크면서 그 근처에 있는 값들만으로 충분히 B-smooth y인 $x^2 = y$를 찾을 수 있지 않을까! Quadratic Sieve는 이 발상을 조금 더 활용합니다.

(x^2-n)이 B-smooth인지를 확인하기 위해서, 무식하게는 B 이하의 소수 p에 대해 $(x^2-n) \bmod p$를 모두 계산하는 과정이 필요해지게 됩니다. 우리는 이 값이 0이기를 원합니다. 그러면 $x^2 = n \mod p$를 만족해야 하고, Tonelli-Shanks를 이용해 이러한 x가 법 p에 대해 어떤 관계를 만족해야 하는지 알 수 있습니다. 따라서 p로 나누는 대상을 $2/p$배로 줄일 수 있습니다. 또 만일 n이 법 p에 대한 이차잉여가 아니라면, p를 나누는 대상에서 완전히 제거해 버릴 수도 있습니다!

즉, 일정 구역을 나눠서 p만큼 껑충껑충 뛰며 여러 개의 수를 한꺼번에 확인할 수 있습니다. 이 방법이 에라토스테네스의 체와 매우 유사하기 때문에[3] 알고리즘에 **이차 체**라는 이름이 붙어 있습니다.

[3]한 구간에 대해서 따진다면, 에라토스테네스의 체와 시간 복잡도마저 같습니다.

4 최적화 및 병렬화

여러 가지 최적화 방법이 나왔지만 몇 개만 소개합니다.

4.1 Find and Merge

이 최적화는 기본적으로, 상당히 오랜 시간이 걸릴 수 있는 마지막 단계인 행렬의 nonzero kernel을 찾는 최적화입니다. 아이디어는 새로 들어온 bit vector에 대해 곧바로 Gaussian elimination을 시행하는 것입니다.

새로운 bit vector v가 들어올 때마다, bit에 대응하는 가장 큰 소수를 찾아 그 소수와 짝을 짓습니다. 만일 이미 소수 p와 bit vector v_p가 짝지어져 있다면, $v \oplus v_p$는 대응하는 가장 큰 소수가 p보다는 작을 것입니다. 따라서 p에 짝지어진 bit vector를 v_p와 v 중 bit의 개수가 적은 쪽으로 update하고, $v \oplus v_p$를 가지고 비슷한 행동을 반복합니다. 이렇게 해서 v에 bit이 하나도 없으면, 그 v는 우리가 원하는 해일 가능성이 상당히 높습니다.

n-bit일 때 (sparsity의 이득을 누려) 시간 복잡도로 $\mathcal{O}(n^2 \log n)$을 기대하나, 최악의 경우 시간 복잡도는 여전히 그대로입니다. 그러나 이 과정은 만일, 운이 아주 좋아 bit vector를 예상치보다 훨씬 적게 가지고도 이로부터 dependence를 만들어낼 수 있는 경우 엄청난 성능 향상을 발휘합니다. 기대되는 공간 복잡도는, 각 bit vector 중 작은 쪽을 고르므로 여전히 $\mathcal{O}(n \log n)$으로 그대로입니다.

이 방법은 Block Lanczos Algorithm이나 Block Wiedemann Algorithm 등의 훨씬 빠름이 보장되는(즉, $\mathcal{O}(n^2 \log n/32)$ 시간 복잡도와 $\mathcal{O}(n \log n)$ 공간 복잡도가 **보장되는**) 알고리즘들이 알려지면서 사용이 급격하게 줄었습니다.

4.2 B^2 Library

이 최적화는 그렇게 효과적인 최적화는 아니나, 속도 향상은 확실히 있는 방법입니다.

$(x^2 - n)$들의 배열에서, p에 대해 순회하면서 나누어떨어지는 소수를 모두 나눈 다음, 남은 것이 1이어야 B-smooth라고 말할 수 있습니다. 그리고 만일 이 수가 1이 아니면, 거기 남아 있는 수 R_x는 B 이하의 소수로 모두 나누어떨어지지 않습니다. 만일 $R_x < B^2$이라면, R_x는 소수입니다.

서로 다른 x, y에 대해 $R_x = R_y$라면, 두 bit vector를 서로 합성해서 B-smooth인 bit vector 하나를 만들어낼 수 있습니다. 따라서 이 방법의 유일한 단점 – **실제로 B^2 bound까지 저장해놓을 수는 없다** – 을 알 수 있습니다. 작은 경우에 대해 B^2 bound까지 전부 저장해 보면, 실제로 거의 B^2을 bound로 삼는 것과 다름없는 속도 향상을 보여 줍니다. 그리고 보통 B가 굉장히 커지면 B 자체로도 이미 너무 크기에 이 방법은 사용할 수 없게 됩니다.

4.3 Multiple Polynomials

QS의 유일하다고 할 수 있는 병렬화입니다.

$f(x) = x^2 - n$ 이외에도, $f(x) = (Px+Q)^2 - n = P(Px^2 + 2Qx + R)$ 꼴의 polynomial을 많이 사용하는 방법입니다. $R = \frac{Q^2-n}{P}$이고, P를 제곱수로, R을 정수로 두어 $Px^2 + 2Qx + R$

가 B-smooth인지만 확인하는 방법을 주로 씁니다.

2개까지는 큰 속도 향상이 없습니다: 1개일 때 나누어떨어지지 않는 p를 확정적으로 제거했지만, 2개 이상일 때는 그럴 수 없기 때문입니다. 1개인 경우 소수가 제거되는 비율을 생각해 보면 대강 1/2쯤 되는 것을 알 수 있습니다. 3개 이상부터 어마어마하게 빠른 속도 향상을 보이는데, 예를 들어 RSA-129를 QS로 소인수분해하는 데 1600개 이상의 polynomial이 사용되었다고 합니다.

◇

전반부의 꽃인 Quadratic Sieve를 다루었습니다. 여기까지 재미있으셨나요?

수 하나만으로 훨씬 다양하게 소개하고 싶은 알고리즘들이 많이 있었습니다. 그러나 수는 수 자체만으로 독립적으로 존재하지 않습니다. 수의 표현 속에 숨은 다양한 성질은 다양한 대수학적 구조 – 예를 들어 체, 다항식 환, 아이디얼 등 – 와 모두 연결되어 있습니다. 그렇기에 소개하지 못한 알고리즘, 혹은 소개했더라도 증명하지 않고 미흡하게 넘어간 알고리즘이 너무 많습니다.

후반부에서는 다항식과 그에 관련된 알고리즘, 그리고 다항식에서도 마찬가지로 소인수분해하는 알고리즘을 알아봅니다. 그러나 환 혹은 구체적으로는 다항식 환을 다루면서, 앞에서 다루었던 수많은 다른 알고리즘들에 눈이 뜨이게 되시리라고 믿습니다.

5 문제

1. Dixon's Factorization Algorithm에서, k는 얼마나 커야 할까요?

   (a) 전체 개수가 k일 때까지 찾을 확률을 곱 기호를 사용하여 정확히 표현하세요.

   (b) $k \approx \pi(B)$라고 가정하고, (a)를 근사하세요.

   (c) $\pi(B) = 100$일 때 위 근삿값과 곱을 실제로 계산한 값을 모두 구해, 아래 표와 비슷한 결과를 얻으세요. 근사치는 꽤 큰 k에 대해서도 정확하며, k가 $\pi(B)$에 꽤 가까워져도 확률은 매우 작습니다. 이 결과를 바탕으로, block Lanczos algorithm이 나왔을 때, QS에서는 Find and Merge 최적화를 미련 없이 버릴 수 있었습니다.

k	정확한 값	추정한 값
50	$8.881784197001 \cdot 10^{-16}$	$8.881784197001 \cdot 10^{-16}$
60	$9.094947017729 \cdot 10^{-13}$	$9.094947017729 \cdot 10^{-13}$
70	$9.313225746155 \cdot 10^{-10}$	$9.313225746155 \cdot 10^{-10}$
80	$9.536740132043 \cdot 10^{-7}$	$9.536738616589 \cdot 10^{-7}$
90	$9.762446529070 \cdot 10^{-4}$	$9.760858180243 \cdot 10^{-4}$

2. (Lanczos algorithm) Block Lanczos algorithm은 Lanczos algorithm에 두 가지 변형:

   - $F = \mathbb{R}$ 혹은 $F = \mathbb{C}$에서 $F = F_{p^n}$으로
   - $F = F_2 = \mathbb{Z}_2$인 경우 시간 복잡도를 $\mathcal{O}(n^2 \log n)$에서 $\mathcal{O}((n^2 \log n)/N)$으로

   을 가한 형태입니다. Technical change이므로 궁금하신 분들은 다음[4]을 읽어 보시기 바랍니다. 여기서는 Lanczos algorithm을 알아봅니다. $F = \mathbb{R}$ 혹은 $F = \mathbb{C}$입니다.

   (a) (Gram-Schmidt orthogonalization process) vector space V에 bilinear form $B : V \times V \to F$가 주어져 있다고 합시다. 일차독립인 아무 $\{v_1, \cdots, v_n\}$에 대해

   $$w_i = v_i - \sum_{j=1}^{i-1} \frac{B(v_j, v_i)}{B(v_j, v_j)} v_j$$

   를 모든 i에 대해 정의**할 수 있다면,** $\{w_1, \cdots, w_n\}$은 둘씩 B-수직임을 증명하세요.

   (b) 주어진 선형사상 A가 nontrivial kernel을 가질 때 그 (0 아닌) 원소를 아무 것이나 하나 구하고자 합니다. 이를 위해 아무 $0 \neq w_0 \in V$를 잡고, A에 대해

   $$w_i = A^{\mathbf{t}} A w_{i-1} - \sum_{j=0}^{i-1} \frac{\langle w_j, A^{\mathbf{t}} A w_{i-1} \rangle_{A^{\mathbf{t}} A}}{\langle w_j, w_j \rangle_{A^{\mathbf{t}} A}} w_j$$

   ($\langle a, b \rangle_L := a^{\mathbf{t}} L b$)로 벡터열 $\{v_i\}$을 만듭니다. w_i에 대한 식을 간단히 하세요. 세 개의 항만이 주어져야 합니다.

   (c) 만일 $\langle w_i, w_i \rangle_{A^{\mathbf{t}} A} = 0$이면, 이 과정을 더 이상 진행할 수 없습니다. 이때 w_i가 A의 nontrivial kernel에 속함을 보이세요. 우리의 경우에서 A는 최대 $n \log n$개의 element를 가진 행렬이고, iteration 횟수는 최대 n번이므로, (왜 그렇습니까?) 시간 복잡도는 $\mathcal{O}(n^2 \log n)$이 됩니다.

   (d) F를 finite field로 바꾸려 하는 경우, (c)의 증명에 문제를 겪게 됩니다. 무엇입니까? (Hint: I의 positive definiteness.)

3. (Properties of QS)

   (a) n이 아주 크면, $f(x) = x^2 - n$이 $x \approx \sqrt{n}$ 근처에서 선형 증가함을 보이고, 자연수 m에 대해 $f(m+1) - f(m)$을 구체적으로 계산하세요. 따라서 많이 계산해도 수가 확 커질 가능성을 고려하지 않을 수 있습니다.

   (b) (a)를 이용하면, single polynomial QS의 전체 step에 큰 수에 대한 연산이 **큰 수의 자릿수에 비례하는 횟수로** 사용되게 할 수 있음을 보이세요.[5]

[4] Montgomery, Peter L. "A block Lanczos algorithm for finding dependencies over **GF**(2)." *International Conference on the Theory and Applications of Cryptographic Techniques*. 1995. **GF**(2)는 F_2 i.e. $\langle \mathbb{Z}_2, +, \cdot \rangle$를 일컫습니다.

[5] 가감승제의 경우 수의 크기가 비슷해야 큰 수 연산으로 칩니다; 큰 수에 아주 작은 수를 연산하는 것은 대단히 빠르게 할 수 있으므로, 큰 수 연산으로 치지 않습니다.

4. (RSA) Rivest-Shamir-Adleman 암호는 소인수분해의 어려움에 기반하고 있는 암호입니다. 이 암호는 다음과 같은 방법을 통해 암호 통신을 진행합니다.

   i. **수신자는** 두 소수 p, q를 곱한 자연수 $n = pq$을 구하고, 이와 서로소인 수 e를 아무거나 하나 잡는다.

   ii. 수신자가 $de = 1 \mod (p-1)(q-1)$인 d를 구하고, (n, e)를 **송신자에게** 보낸다. d는 공개하지 않는다.

   iii. 송신자가 보내고 싶은 메세지를 m이라 할 경우 $m^e \mod n$을 수신자에게 보낸다.

   iv. 수신자는 받은 정보 i에 대해 $i^d \mod n$을 계산하여, $i^d = m^{ed} = m \mod n$을 구할 수 있다.

   효과적인 소인수분해 방법이 나오면 RSA는 무용지물이 됩니다.[6] 실버가 키파에게 $n = 1031582059362490128566030159178040513943163 7$과 $e = 65537$을 주었습니다. 이를 이용하여 키파가 실버에게 정보를 전달하고자 합니다. 흉악한 민티는 키파와 실버의 모든 통신을 도청했습니다.

   (a) Quadratic Sieve를 사용하여 n을 소인수분해하세요.[7]

   (b) 키파가 실버에게 암호화된 정보

   $$i = 488638311873847726943562015168522005484550 0$$

   을 전달했습니다. (a)를 바탕으로 d를 계산해, m을 알아내세요.

   (c) m이 AES-ECB 암호화의 128-bit 키로 이용되었다고 합니다. 128-bit m을 8-bit 씩 끊어 문자열로 만든 뒤, 이곳[8]에서 키파가 실버에게 보낸 문자열을 알아내세요. 통신 내용은 다음과 같습니다.

   ```
   8796B84A 132D9AC9 1CC7ACD6 D096B5FF
   BD61C2E5 075BD834 4C595963 723BF917
   7A2D4E5D 5EA46627 7D28CA5D BCF1EDCC
   A06B341C 66F1FE57 4EA54B30 B90CB6EB
   3474A5FE 0DF573B9 91D947B7 0DEEFEDA
   40472116 B2343BEC 85D1206B 82180C6E
   241EE26C 21A7999C DC67A638 51C6FE16
   ```

[6] 그러나 효과적인 소인수분해 방법이 나온다고 해서 현대의 모든 암호가 무용지물이 되지는 않습니다: Diffie-Hellman 키 교환 등의 소인수분해보다 훨씬 어려운 방법에 기반한 정보 교환 알고리즘이 있고, (비록 초기에 정보를 약간 알아야 하지만) AES 등의 소인수분해의 어려움에 전혀 의존하지 않는 암호도 있습니다.

[7] WolframAlpha에 넣어도 소인수분해가 되기는 합니다. 이는 WolframAlpha도 Quadratic Sieve를 사용하기 때문입니다. 그러니 제발 그러지 말아주세요... 제 컴퓨터(single core)에서는 `pypy3`로 639초가 걸렸습니다.

[8] `https://www.devglan.com/online-tools/aes-encryption-decryption`. 사이트에 버그가 있어서, 결과 문자열의 base64 utf-8 decode는 `https://www.base64decode.org/` 등의 다른 사이트에서 하시기 바랍니다.