Number Theoretic Algorithms

최대공약수와 유한 체

April 16, 2020

1 시작

아무 것도 믿지 않고서는 아무 것도 할 수 없습니다.

우리는 일반적인 논리계를 믿습니다. 우리는 가장 기초적인 것들, 예를 들어 정수의 덧셈, 뺄셈, 곱셈이 가능하고 그 값이 정수라는 사실을 믿습니다. 우리는 정수의 덧셈, 뺄셈, 곱셈을 하는 방법을 알고 있습니다.

우리는 정수 a와 영이 아닌 정수 b가 존재하여

$$a = bq + r, \quad 0 \leq r < |b|$$

를 만족시키는 정수 q와 r이 유일하게 존재한다는 것을 믿습니다. 이 사실은 우리가 이야기를 시작할 수 있게 해 주는 가장 큰 원동력이 됩니다. 우리는 또한 $|a|$와 $|b|$의 길이가 상수일 때, q와 r을 상수 시간에 찾는 방법을 알고 있습니다.

대수학(추상대수) 과목, 더 나아가 집합론 과목을 들으면, 위 사실들을 증명하게 됩니다. 그때에는 기초적인 공리계조차 부정하여 무엇을 할 수 있는지 살펴봅니다. 이 스터디의 목적은 그것이 아니므로, 우리는 고등학교 때까지 배운 "수학"을 모두 믿기로 합니다.

2 자릿수 기준으로 사고하기

여러분은 이미 큰 수에 대해 다룰 때에는 단순한 연산도 상수 시간으로 취급해서는 안 된다는 것을 알고 계실 겁니다. 예를 들어 큰 수 $A + B$를 직접 계산해 보셨다거나 하는 것입니다. 여기서는 $A_1, \cdots, A_n$의 n개의 수를 더하는 알고리즘을 구상해 봄으로써 이것이 얼마나 중요한 문제인지를 다시 확인해 봅시다. 각 수의 길이의 합을 L이라 하면 입력의 길이는 L에 비례하므로, 이것보다 시간이 더 줄어들 수는 없습니다.

가장 단순하게 생각해 볼 수 있는 알고리즘은 앞에서 순서대로 큰 수를 더하듯이 더하는 것입니다. 그러면 $\mathcal{O}(Ln)$ 시간이 걸리게 됩니다: 큰 수 $A + B$가 자릿수가 큰 쪽에 비례하여 연산되므로, 예를 들어 큰 L에 대해 $10^{L-n}, (-1), 1, \cdots$의 입력이 들어오게 되면 모든 덧셈 연산에 L에 비례하는 시간이 들어가게 됩니다.

그렇다면, 자연스레 '자릿수 순서대로 정렬하면 되지 않을까?' 하는 생각을 할 수 있습니다. 이 방법에 대한 시간 복잡도를 계산해 보도록 합시다.

- 먼저, 자릿수를 명시적으로 계산해야 하는데, 2진법에서 일반적으로 아는 1씩 더하는 방법을 쓰는 경우 자릿수의 자릿수 $\mathcal{O}(\log\log L)$에 비례하는 계산 시간이 들어갑니다. 이것은 입력의 크기마다 모두 더해 주어야 하므로, 일단 $\mathcal{O}(L\log\log L)$ 시간이 들고 시작합니다.

- 이제 정렬을 해야 하는데, 비교 기반 정렬 알고리즘을 쓰면 비교 횟수를 $\mathcal{O}(n\log n)$으로 만들 수 있다는 사실은 익히 알고 계실 것입니다. 하지만 이제 **한 번의 비교**가 $\mathcal{O}(\log\log L)$ 시간이 걸리기 때문에, 비교에만 $\mathcal{O}(n\log n\log\log L)$ 시간이 걸립니다.

    - radix sort 등의 비-비교 기반 정렬 알고리즘을 쓰면 상황이 좀 나아질까요? 각 step마다, 비트를 모두 훑는 것은 $\mathcal{O}(n)$ 시간에 할 수 있습니다만, 길이 $\mathcal{O}(\log n)$인 index n개를 복사해야 하므로 $\mathcal{O}(n\log n)$ 시간이 들어가게 됩니다. step의 횟수는 $\mathcal{O}(\log\log L)$에 비례할 것이므로, 전체 시간복잡도는 비교 기반 정렬 알고리즘과 마찬가지로 $\mathcal{O}(n\log n\log\log L)$ 시간이 들어갑니다.

- 마지막으로, 정렬한 순서대로 더하게 됩니다. 이 과정은, 다행히도, L에 비례하는 시간이 소요됩니다.

따라서 이 방법으로 계산하면, $\mathcal{O}((L+n\log n)\log\log L)$ 시간이 걸린다는 것을 알 수 있습니다. 1번, 2번 과정에서 자릿수의 크기마저도 짐작할 수 없기 때문에 생긴 항인 $\log\log L$ 항에 주목해서 위 논증을 다시 읽어보시기 바랍니다.

3 최대공약수

모두 영인 것은 아닌 두 정수 a와 b에 대해 g가 a와 b를 모두 나누어떨어뜨릴 때, 그러한 g 중 가장 큰 것을 a와 b의 최대공약수라 하고, $\gcd(a,b)$로 씁니다.

a와 b에 대해 다음 소인수분해를 생각합니다:

$$a = p_1^{a_1}p_2^{a_2}\cdots p_n^{a_n}$$
$$b = p_1^{b_1}p_2^{b_2}\cdots p_n^{b_n}$$

이때 a_i 혹은 b_i들은 음이 아닌 정수이며, 0도 허용합니다. 그러면

$$\gcd(a,b) = p_1^{c_1}p_2^{c_2}\cdots p_n^{c_n}, \qquad c_i := \min(a_i,b_i)$$

인 것이 잘 알려져 있습니다. 그러나 이 방법은 소인수분해를 필요로 하기 때문에, 시간이 상당히 많이 걸리는 방법입니다.

일반성을 잃지 않고 $a > b > 0$이라고 합시다. 그러면 $a = bq + r, \quad 0 \le r < b$인 정수 q와 r이 존재합니다. 이때 $\gcd(a,b) = \gcd(b,r)$인 것이 알려져 있습니다.

증명은 다음과 같이 합니다. $\gcd(a,b) =: g$, $\gcd(b,r) =: g'$이라 하면, r이 g로 나누어떨어짐을 보여 $g \le g'$을 보입니다. 또 a가 g'으로 나누어떨어짐을 보여 $g' \le g$를 보입니다. 따라서

$g = g'$입니다.

나눗셈을 이용해서 계산하면, 최대공약수를 구하는 데 시간이 얼마나 걸릴까요? 이것을 나눗셈 횟수를 통해 확인해 봅시다. b보다 큰 정수 a에 대해 $\gcd(a, b)$를 구하는 데 걸리는 최대 나눗셈 횟수를 T_b라 합니다. 가장 간단히 생각할 수 있는 것은, $r < b$이므로 $T_b \leq 1 + \max_{0 \leq i < b} T_i$입니다. 그러나 $r > b/2$일 때 다음 단계를 고려하면 $\gcd(b, r) = \gcd(r, b - r)$이 되고, $b - r < b/2$이므로

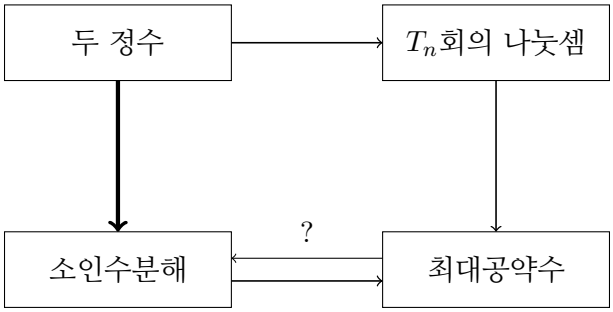
$$T_b \leq \max\left(1 + \max_{0 \leq i \leq b/2} T_i, 2 + \max_{0 \leq i < b/2} T_i\right) \leq 2 + \max_{0 \leq i \leq b/2} T_i$$

입니다. 이제 수학적 귀납법 등을 사용하여

$$T_n \leq 2 \log_2 n + 1 = \log_{\sqrt{2}} n + 1$$

를 증명할 수 있습니다.

우리가 지금 떠올려야 하는 diagram은 다음과 같습니다:



그럼 반대로 정수를 받았을 때, 어떤 정수를 잘 생성해서 최대공약수를 구하면 소인수분해 시간을 줄일 수 있지 않을까요?[1] 이를 위해서 빠른 나눗셈 방법이 필요합니다! 따라서, 큰 수의 가감승제 및 계산, 소수 여부의 판단 및 소수의 분포를 구하는 방법과 더불어 소인수분해를 빠르게 하는 방법을 이 스터디의 앞부분에서 다루게 됩니다.

4 체

가감승제가 자유로운 집합을 체라 합니다.

가감승제라 함은 더하기, 빼기, 곱하기, 나누기를 일컫습니다. 자유롭다 함은 다음을 일컫습니다:

- 더하기, 곱하기를 한 연산의 결과가 집합 안에 있습니다.
- 더하기끼리 혹은 곱하기끼리는 연산[2]이나 피연산자[3]의 순서에 상관없이 값이 같습니다.
- 더하기와 곱하기 사이에 분배 법칙이 성립합니다.

[1] 이런 식으로 사고하도록 연습하십시오!
[2] 결합 법칙.
[3] 교환 법칙.

- 집합 내에 0과 1이 존재하며, 서로 다릅니다.[4]
- 어떤 수 a를 가져와도 $(-a)$와 a^{-1}가 집합 내에 존재합니다. 단 0^{-1}은 없고 이것은 예외로 합니다.

예를 들어, 유리수는 체입니다. 실수와 복소수도 체가 됩니다.

5 유한 체

어떤 체 F가 주어졌을 때, F의 원소의 개수 $|F|$가 1 이하일 수는 없습니다. 체의 정의에 따라 $0 \in F$, $1 \in F$이고 $0 \neq 1$이어야 하기 때문입니다.

체 F에는 0, 1의 두 원소가 주어지고, 반드시 다음 연산표를 만족해야 합니다.

$+$	0	1
0	0	1
1	1	x

$\times$	0	1
0	y	0
1	0	1

y는 $0 \cdot 0 = (0+0) \cdot 0 = 0 \cdot 0 + 0 \cdot 0$에 의해 반드시 0이어야 합니다. x가 1이라면, $1+1 = 0+1$, 즉 $0 = 1$이라고 주장하는 셈이기 때문에 안 됩니다. 그런데 만약 $x \neq 1$이면, 한번 $x = 0$으로 놓아 봅시다.

계산하면 할수록 $x = y = 0$으로 놓았을 때 체의 모든 성질이 만족되는 것 같습니다. 이 집합은 실제로 체가 되며, 이 체는 컴퓨터들이 뛰노는 체입니다. 이 체의 이름은 $\mathbb{F}_2$입니다.

이제 x를 0과 1과 모두 다른 수로 생각하고, 연산표를 확장해서 2개보다 많은 수의 원소를 가지는 체를 만들어 봅시다.

$+$	0	1	x
0	0	1	x
1	1	x	y
x	x	y	z

$\times$	0	1	x
0	0	0	0
1	0	1	x
x	0	x	w

$0 \cdot x$가 0인 것은 위에서와 비슷한 이유입니다.

이번에 우리가 정해야 하는 것은 y, z, w의 세 변수입니다. 이 변수들을 0, 1 혹은 x로 두어 체의 모든 정의들을 만족하도록 시도해 봅시다! 이렇게 하여 원소의 개수가 3개인 체를 구성할 수 있습니다. 이 체의 이름은 $\mathbb{F}_3$입니다.

비슷한 방법으로 원소의 개수가 4개인 체, 즉 $\mathbb{F}_4$를 만들 수 있습니까? 못 만들 것은 없지만, 이제 y, z, w 중 어떤 것을 포함시킬까를 궁리해야 합니다. 또 채워야 하는 칸의 수도, 가능한 경우도 많아집니다. 따라서, 유한 체를 구성하는 일반적인 방법이 필요해 보입니다.

유한 체는 직관적인 더하기 및 곱하기가 성립하고, 이를 기반으로 다항식을 만들 수 있는 등의 이유로 유한 체를 기반으로 한 수많은 아름다운 정리 및 알고리즘이 존재합니다. 이들은 이 스터디의 뒷부분에서 다루게 됩니다.

[4] 0은 덧셈의 항등원, 1은 곱셈의 항등원이라는 특징이 있어야 합니다.

6 문제

1. n개의 수를 더하는 알고리즘의 1번 과정은 사실 $\mathcal{O}(L)$임을 확인하세요.

2. (Atomic Complexity Analysis) 이 문제는 평소에 우리가 자잘한 부분에 대해 얼마나 많은 믿음을 가지고 있는지를 재미있게 깨닫게 하기 위한 문제입니다.

   (a) 모든 프로그램은 메모리를 활용합니다. 현재 알려진 물리 법칙으로는, 메모리의 크기를 V라 할 때, 한 번의 메모리 접근에 빨라야 $\mathcal{O}(V^{1/3})$의 시간이 걸림을 납득하세요.

   (b) 만일 메모리를 원형의 고리처럼 배열해서 뱅글뱅글 돌릴 수 있다면, 머지 소트의 경우 익히 알고 있는 $\mathcal{O}(n \log n)$의 시간 복잡도를 유지할 수 있지만, 힙 소트의 경우 이 메모리 배열로는 $\mathcal{O}(n^2)$ 복잡도임을 납득하세요.

3. $a > b > 0$인 두 정수 a, b에 대해 $a = bq + r$인 정수 q와 r이 존재하여 $0 \leq r < b$를 만족할 때, $\gcd(a, b)$가 r을 나누고, $\gcd(b, r)$이 a를 나눔을 보이세요.

4. (a) 수학적 귀납법을 이용해서 $T_n \leq 2 \log_2 n + 1 = \log_{\sqrt{2}} n + 1$을 보이세요.

   (b) 일반적으로 $T_n \leq \log_u n + \mathcal{O}(1)$이 성립하는 실수 u들을 생각할 때, $u = \sqrt{2}$는 tightest bound가 아닙니다. 우리의 결론이 $u = \sqrt{2}$로 지어진 데는

   $$\gcd(a, b) = \gcd(b, r) = \gcd(b, b - r)$$

   에서 나눗셈을 두 번 거쳤음에도 $b - r$이 b의 반이 되었다고 생각한 것에 있습니다. 이제, 경우를 잘 나누어, $b - r < b/u^2$이 되도록 하는 실수 $u > 1$의 범위를 구하세요.[5] 이 범위에서 u를 골라 $\sqrt{2}$보다 크게 할 수 있습니까?

   (c) (b)에서 구한 실수 중 가장 큰 것(상한)을 φ라 합시다. $T_n \leq \log_\varphi n + 1$이 성립함을 보이고, 이 bound는 tight함을 보이세요. 즉 아무리 큰 N이 주어져도 $T_n = \lfloor \log_\varphi n + 1 \rfloor$를 만족하는 $n > N$이 존재함을 보이면 됩니다. (Hint: Fibonacci Sequence를 이용하세요.)

5. (크기가 소수인 체 구성) 소수 p를 고정합니다. $\mathbb{F}_p$를, p 미만의 음이 아닌 정수들에, 덧셈과 곱셈을 각각 정수의 덧셈과 곱셈을 시행한 뒤 p로 나눈 나머지가 되도록 연산을 정의한 "체"라 하겠습니다. 우리는 이미 $\mathbb{F}_2$가 존재하고 심지어 유일하다는 것까지 알고 있으므로, 편의상 $p > 2$로 두겠습니다. 지금 우리가 자신 있게 $\mathbb{F}_p$를 체라고 말하지 못하는 이유는 순전히 곱셈의 역원 때문입니다.

   (a) $\mathbb{F}_p$에는 2의 곱셈에 대한 역원이 있음을 보이세요.

   (b) $3 \leq a < p$에 대해, $ax = k$인 $x \in \mathbb{F}_p$와 $0 < k < a$가 존재함을 보이세요. Hint.[6] 이 경우 xk^{-1}가 a의 곱셈에 대한 역원이 됩니다.

[5] 이때 실제로 b를 r로 나눈 나머지가 $b - r$이 됨도 확인해 보셔야 합니다!

[6] 이를 위해 $p = aq + r$ $(0 < r < a)$인 정수 q와 r이 존재함을 이용합니다. 의도한 대로 푸셨다면 $k = n - r$입니다.

(c) p가 합성수일 때 이 논리를 적용하지 못하는 이유는 무엇입니까? a에 조건을 더 붙여, 문제가 되는 부분을 고친 명제를 증명하세요. n에 대해서 "$\mathbb{F}_n$"의 원소 중 몇 개가 곱셈에 대한 역원을 가집니까? (소수이면? 합성수이면?) 정수 n에 대해 이 개수를 $\varphi(n)$이라고 표기합니다. 곱셈에 대한 역원을 가지는 원소들만 모은 대수적 구조를 $\mathbb{Z}_n^\times$로 씁니다.

(d) 2019년의 1주차 스터디에서 똑같은 사실을 전혀 다른 방법으로 증명했습니다. 이 증명과 2019년의 증명의 차이점은 무엇입니까? 어떤 증명이 더 좋은 증명이라고는 할 수 없지만, 여러분은 이 증명을 더 선호했으면 좋겠습니다.

6. (a) 원소의 개수가 3001개인 체가 존재함을 설명하세요.

(b) 다음 연산표는 원소의 개수가 8개인 체를 표현합니다. 이 연산표가 정말로 체가 됨을 증명하세요. 결합/분배 법칙, 덧셈의 역원과 곱셈의 역원을 확인하면 됩니다.[7]

$+$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

$\times$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

(c) 원소의 개수가 6개인 체 F_6가 존재합니까? F_6를 구성하려고 노력해 보고, "존재하지 않을 것 같다"는 확신을 가지세요.

7. 영이 아닌 $\mathbb{F}_p$의 원소 a를 고정하고, $\mathbb{F}_p$에서 함수 $f : \mathbb{F}_p \to \mathbb{F}_p$가 정의역의 모든 x에 대해 $f(x) = ax$를 만족한다 합시다.

(a) f가 bijection임을 보이세요. 이를 위해서 $f(x) = f(y)$이면 $x = y$임을 보이고 (injective), 임의의 $\mathbb{F}_p$의 원소 y에 대해 $f(x) = y$를 만족하는 x가 있음을 보이면 됩니다(surjective).

(b) (Fermat) 위 사실과 $f(0) = 0$에서부터 $a^{p-1} = 1$을 유도하세요. 따라서 임의의 영이 아닌 $\mathbb{F}_p$의 원소 a에 대해, a의 $(p-1)$제곱을 p로 나눈 나머지는 1입니다. (Hint: $f(1), \cdots, f(p-1)$에는 1부터 $p-1$까지가 모두 한 번씩 나타나므로, 곱하면 $(p-1)!$과 같습니다.)

(c) a^{-1}를 어떻게 빨리 계산할 수 있습니까?

8. (Euler) 위 문제의 논증에서 더하기를 찾아볼 수 없습니다. 따라서 $\mathbb{Z}_n^\times$에 대해서도 똑같은 논증을 적용할 수 있습니다. 임의의 $a \in \mathbb{Z}_n^\times$에 대해서, $a^{\varphi(n)} = 1$임을 증명하세요.

[7] 교환 법칙 및 덧셈의 항등원과 곱셈의 항등원은 눈으로도 확인할 수 있어서 뺐습니다. 노가다를 하려면 $3 \cdot 8^3 + 2 \cdot 8^2 = 1664$가지를 전부 확인하셔야 합니다. 코딩을 통해 확인하세요.