

3. 최대공약수(gcd)

최대공약수(gcd)를 구하는 방법은 여러 가지가 있을 수 있다.

소인수분해를 이용한 gcd 구하기	
gcd(4, 2) = ?	gcd(30, 24) = ?
$\begin{array}{r} 2 \) \ 4, \ 2 \\ \underline{2, \ 1} \end{array}$	$\begin{array}{r} 2 \) \ 30, \ 24 \\ \underline{3 \) \ 15, \ 12} \\ \quad \underline{5, \ 4} \end{array}$
gcd(4, 2) = 2	gcd(30, 24) = 2 × 3 = 6

↓
↓ 다르게 분석하면(정수 연산에서 나머지(%)를 이용)
↓

gcd(a, b)에서 a mod b = ?	a % b = 0이면, gcd(a, b) = b이다.	a % b = 0이 아니면, gcd(a, b) = gcd(b, a % b)이다.
	<ul style="list-style-type: none"> • gcd(a, b) = gcd(4, 2) • 4 % 2 = 0이므로, • gcd(4, 2) = 2이다. • 결론 : gcd(2, 0) = 2 	<ul style="list-style-type: none"> • gcd(a, b) = gcd(30, 24) • 30 % 24 = 6이다. • 30 % 24가 0이 아니므로 • gcd(30, 24) = gcd(24, 30 % 24)이다.

- 이를 유클리드 호제법이라 한다
- gcd(a, b) = gcd(b, a%b), a>b
- 어떤 수와 0의 최대공약수는 자기 자신이다.
- 즉, gcd(n, 0) = n이다.

// 소인수분해(prime factorization, integer factorization)

- 소인수분해는 1보다 큰 자연수를 소인수(소수인 인수)들만의 곱으로 나타내는 것
- 또는 합성수를 소수의 곱으로 나타내는 것



예제

두 수 96과 60의 최대공약수(gcd)는?

소인수분해에 의한 gcd 구하기	유클리드 호제법에 의한 gcd 구하기
$\begin{array}{r} 2 \) \ 96, \ 60 \\ 2 \) \ 48, \ 30 \\ 3 \) \ 24, \ 15 \\ \quad 8, \ 5 \end{array}$	<ul style="list-style-type: none"> • $\text{gcd}(96, 60) = ?$ <li style="text-align: center;">↓ • $\text{gcd}(60, 96\%60) = \text{gcd}(60, 36)$ • $\text{gcd}(36, 60\%36) = \text{gcd}(36, 24)$ • $\text{gcd}(24, 36\%24) = \text{gcd}(24, 12)$ • $\text{gcd}(12, 24\%12) = \text{gcd}(12, 0) = 12$
$\text{gcd}(96, 60) = 2 \times 2 \times 3 = 12$	$\text{gcd}(12, 24\%12) = \text{gcd}(12, 0) = 12$

- ① 먼저, 최대공약수를 구하는 방법은 여러 가지가 있을 수 있다.
- ② 소인수분해에 의한 방법과 유클리드 호제법에 의한 방법 중 어느 것이 더 효과적인가?
 - 결론은, 유클리드 호제법으로 최대공약수를 구하는 것이 훨씬 빠르다.
 - 사람은 소인수분해 방식으로 최대공약수를 구하는 것이 더 쉬울 수 있다.(????)
- ③ 계산량 이론에서 최대공약수를 구하는 것은 "쉬운 문제"로서 알려져 있다.
 - 하지만, 소인수분해는 "어려운 문제"로 평가되고 있다.
 - 최대공약수를 구할 때, 소인수분해 방식을 이용할 필요가 없다는 것이다.

// 유클리드 호제법

- 유클리드 호제법은 2개의 자연수에 대한 **최대공약수(gcd)**를 구하는 알고리즘이다.
- 유클리드 호제법(互除法)은 두 수를 **나누어** 나머지가 0이 되는 원리를 이용한다.
- **호제법**은 두 수가 서로(互) 상대방 수를 나누어(除) 원하는 수를 얻는 알고리즘을 나타낸다.
- 기원전300년경에 발표되었다.(알려진 가장 오래된알고리즘)
- 유클리드는 고대 그리스 수학자이다.
- **최대공약수**를 구하는 여러 알고리즘 중에 가장 유명한 것은 **유클리드 호제법**이다.

● 유클리드 호제법에 의한 최대공약수(gcd) 구하기 연습

두 정수 a, b의 최대공약수를 구하는 유클리드 호제법의 기본 원리는 다음과 같다.

$$\text{gcd}(a, b) = \text{gcd}(b, r)$$

- r은 큰 수(a)에서 작은 수(b)를 나눈 나머지이다. ($r = a \bmod b$)
- 작은 수(b)와 나머지 r의 최대공약수가 곧 두 정수의 최대공약수이다.



연습

유클리드 호제법으로 두 정수 a, b에 대한 최대공약수 구하기(a>b)

a = 36, b = 10	a = 60, b = 36	a = 7, b = 4	a = 96, b = 61
gcd(a, b)	gcd(a, b)	gcd(a, b)	gcd(a, b)
= gcd(36, 10)	= gcd(60, 36)	= gcd(7, 4)	= gcd(96, 61)
= gcd(10, 6)	= gcd(36, 24)	= gcd(4, 3)	= gcd(61, 35)
= gcd(6, 4)	= gcd(24, 12)	= gcd(3, 1)	= gcd(35, 26)
= gcd(4, 2)	= gcd(12, 0)	= gcd(1, 0)	= gcd(26, 9)
= gcd(2, 0)	= 12	= 1	= gcd(9, 8)
= 2			= gcd(8, 1)
			= gcd(1, 0)
			= 1

- $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$, $a > b$
- 어떤 수와 0의 최대공약수는 자기 자신이다.
- 즉, $\text{gcd}(n, 0) = n$ 이다.

// 두 정수가 서로소 관계이면, gcd는 1이다.

- $\text{gcd}(a, b) = \text{gcd}(7, 4) = 1$
- $\text{gcd}(a, b) = \text{gcd}(96, 61) = 1$

- $\text{gcd}(a, b) = \text{gcd}(60, 36) = 12 \rightarrow 60$ 과 36 은 서로소가 아니다.

// 다음은 유클리드 호제법에 의한 최대공약수(gcd)를 구하는 알고리즘이다.

	C	Python
순환함수 (재귀함수)	<pre>int gcd(int a , int b) { if (b==0) // 완료조건 return a; else return gcd(b, a%b); }</pre>	<pre>def gcd(a, b): if b==0: #완료조건 return a #나머지==0 else: return gcd(b, a%b) #나머지!=0 print (gcd(24, 30)) #출력 : 6</pre>
	<pre>int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }</pre>	
반복함수	<pre>int gcd(int a, int b){ int r; // 나머지 while(b){ // b!=0과 같음 r = a % b; a = b; b = r; } return a; }</pre>	<pre>def gcd(a, b): while b != 0: #나머지!=0 r = a % b a = b b = r return a #나머지가 0일때 print (gcd(24, 30)) #출력 : 6</pre>

// 다음은 뱀셈에 의한 최대공약수(gcd)를 구하는 알고리즘이다.

	C	Python
반복함수	<pre>int main() { int a = 120, b = 45; while(a != b) { if(a > b) a = a - b; else b = b - a; } printf("%d", a); // 출력 : 15 }</pre>	<pre>a = 120 b = 45 while a != b: if a > b: a = a - b else: b = b - a print (a) # 출력 : 15</pre>

- 유클리드 호제법의 시간복잡도 : $O(\log n)$
- 참고로, 더 정확한 유클리드 호제법의 시간복잡도를 구하는 방식도 있다.
- 피보나치 수열을 이용하여 다소 복잡한 방식이다.