



## 유클리드 알고리즘과 연분수 자동 조판

### Euclidean Algorithm and Typesetting Continued Fractions

노나메 Nona Mé

한글 텍 사용자 그룹 KTUG

**KEYWORDS** expl3,  $\LaTeX$  programming, continued fraction, Euclidean algorithm, GCD

**ABSTRACT** 최대공약수를 계산하는 유클리드 알고리즘을 Expl3로 구현하고, 이 알고리즘을 연분수 전개에 적용할 수 있음을 보인다. 유클리드 알고리즘을 수작업으로 계산하는 소위 ‘호제법’ 계산식을 자동으로 표현하는 방법을 제시한다.

#### 1 유클리드 호제법

유클리드 호제법(互除法)<sup>1</sup> 또는 유클리드 알고리즘이란 유클리드의 《원론》 제7권에서 다루어지는 것으로 최대공약수를 구하는 알고리즘이다. 인류 최초의 알고리즘이라고 알려져 있다.<sup>2</sup>

《원론》의 명제를 제시하면 다음과 같다 [10].

**명제 1** 서로 다른 두 수가 있다. 작은 쪽을 큰 쪽으로부터 차례로 제거해갔을 때 마지막에 하나의 단위(unit)<sup>3</sup>만을 남겨둔다면 두 수는 서로소(relatively prime)이다.

**명제 2** 두 수가 서로소가 아니면 두 수로부터 최대인 공통 인수(= 최대공약수)를 구할 수 있다.  
[따름정리] 만약 어떤 수가 두 수의 인수라면, 그 수는 최대인 공통 인수의 인수가 된다.

**명제 3** 세 수가 서로소가 아니라면 최대인 공통 인수를 찾을 수 있다.

명제 2의 증명 과정이 오늘날 “유클리드 호제법”이라고 부르는 것이다. 이것을 한국어판 위키백과에서 다음과 같이 설명하고 있다 [9].

2개의 자연수(또는 정식)  $a$ ,  $b$ 에 대해서  $a$ 를  $b$ 로 나눈 나머지를  $r$ 이라 하면(단,  $a > b$ ),  $a$ 와  $b$ 의 최대공약수는  $b$ 와  $r$ 의 최대공약수와 같다. 이 성질에 따라,  $b$ 를  $r$ 로 나눈 나머지  $r'$ 를 구하고, 다시  $r$ 을  $r'$ 로 나눈 나머지를 구하는 과정을 반복하여 나머지가 0이 되었을 때 나누는 수가  $a$ 와  $b$ 의 최대공약수이다.

<sup>1</sup> 호제법이라는 말은, 서로(互) 나누어(除) 가서 원하는 결과를 얻는 방법이라는 뜻이다.

<sup>2</sup> 현행 중학교 교육과정에서는 최대공약수를 소인수분해에 기초한 방법으로 다루기 때문에 유클리드 호제법은 포함하고 있지 않다. 실용적으로 번거로운 방법이기 때문에 별도로 가르칠 필요가 없어서이겠지만 알고리즘 등을 다룰 때에는 알아두는 편이 도움이 된다.

<sup>3</sup> ‘단위’(unit)는 《원론》 7권에서 새롭게 등장하는 정의이다. “단위라 함은 각 사물(도형)에서 1이라 할 수 있는 것이다.”

달리 말하면,  $a > b$ 인 두 수  $a$ 와  $b$ 의 최대공약수를  $\gcd(a, b)$ 라고 쓰기로 하고,  $a$ 를  $b$ 로 나눈 나머지를  $a \bmod b$ 와 같이 쓰기로 할 때,

$$\gcd(a, b) = \gcd(b, (a \bmod b)) \quad (1)$$

라는 것이다. 이 관계를 재귀적으로 적용하여 마지막에 나누는 수가 최대공약수이고, 그 마지막에 나누는 수가 1이면 두 수는 서로소이다.

### 1.1 초등적 증명

식 (1)을 증명한다.  $a, b \in \mathbb{Z}$ 이고  $a > b$ 이면  $a = bq + r$ 인 정수  $q$ 와  $r$ 이 존재한다.  $a = bq + r$ 에서  $\gcd(a, b) = \gcd(b, r)$ 임을 증명하겠다.

증명.  $\gcd(a, b) = d$ 라 하면  $a = d\alpha$ ,  $b = d\beta$ 이고  $\alpha$ 와  $\beta$ 는 서로소이다.

$a = bq + r$ 에서

$$\begin{aligned} d\alpha &= d\beta q + r \\ &= d(\beta q + \rho) \end{aligned}$$

이므로  $d$ 는  $r$ 의 인수이기도 해야 한다 ( $r = d\rho$ ).

이제  $\beta$ 와  $\rho$ 의 최대공약수  $d' > 1$ 가 있어,  $\beta = d'\beta'$ ,  $\rho = d'\rho'$ 이라 하면,  $a = bq + r$ 로부터

$$\begin{aligned} a &= bq + r \\ d\alpha &= d\beta q + d\rho \\ &= dd'\beta'q + dd'\rho' \\ &= dd'(\beta'q + \rho') \\ \alpha &= d'(\beta'q + \rho') \end{aligned}$$

$\alpha = d'(\beta'q + \rho')$ 이고  $\beta = d'\beta'$ 이므로 둘은 공약수  $d'$ 을 가진다. 이것은  $\alpha$ 와  $\beta$ 가 서로소라는 가정에 모순이다. 그러므로  $\gcd(\beta, \rho) = d' > 1$ 이 존재할 수 없고  $\gcd(\beta, \rho) = 1$ 이어야 한다. 이로부터  $\gcd(b, r) = d$ . □

## 2 최대공약수 알고리즘의 `ex13` 구현

이 글의 목적이  $\text{\LaTeX}$  프로그래밍에 관한 것이기 때문에 우리는 두 자연수에 대한 최대공약수만을 다룰 것이다.

다음은 유클리드 알고리즘을 python 코드로 표현한 것이다.

```
def gcd(a,b):
    if a<b:
        a,b=b,a
    while b != 0:
        t = a%b
```

```

a,b=b,t
return a

```

첫 if문은 큰 수에서 작은 수를 나누도록 하는 것인데,  $a = bq + r$ 에서  $q$ 가 0의 값을 가질 수 있다고 하면 계산 과정이 한 번 증가하지만 제거할 수 있다. 이를 제거하면  $a = 3$ ,  $b = 7$ 일 때, 첫 번째 계산이  $3 = 7 \times 0 + 3$ 이고 그 다음에  $7 = 3 \times 2 + 1$  계산을 진행하게 되는데 이 한 번의 계산을 줄이려고 처음부터 큰 수를 앞세우게 한 것이다.

Expl3로 번역하면 다음과 같이 된다 [8, p. 35].

```

\int_zero_new:N \l_a_int
\int_zero_new:N \l_b_int
\int_zero_new:N \l_t_int
\NewDocumentCommand \euclideangcd { m m }
{
  \int_set:Nn \l_a_int { \int_max:nn { #1 } { #2 } }
  \int_set:Nn \l_b_int { \int_min:nn { #1 } { #2 } }
  \int_while_do:nn { \l_b_int != 0 }
  {
    \int_set:Nn \l_t_int
      { \int_mod:nn { \l_a_int } { \l_b_int } }
    \int_set_eq:NN \l_a_int \l_b_int
    \int_set_eq:NN \l_b_int \l_t_int
  }
  \int_use:N \l_a_int
}

```

120과 64의 최대공약수를 구해보자.

```
\euclideangcd{120}{64}
```

8

### 3 연분수 계산

방금 작성한 `\euclideangcd` 함수에서 이어지는 나눗셈의 몫(quotient)을 한 군데 모아보자. expl3의 자료 구조의 하나인 `clist`에 차례로 넣어두었다가 마지막에 이를 표현(use)하게 하였다.

```

\clist_new:N \l_quot_clist
\NewDocumentCommand \euccontfrac { m m }
{
  % a, b는 분자, 분모이므로 크기에 따라 순서를 바꾸지 않는다.

```

```

\int_set:Nn \l_a_int { #1 }
\int_set:Nn \l_b_int { #2 }
\clist_clear:N \l_quot_clist
\int_while_do:nn { \l_b_int != 0 }
{
  \int_set:Nn \l_t_int
    { \int_mod:nn { \l_a_int } { \l_b_int } }
  \clist_put_right:Nx \l_quot_clist
    { \int_div_truncate:nn { \l_a_int } { \l_b_int } }
  \int_set_eq:NN \l_a_int \l_b_int
  \int_set_eq:NN \l_b_int \l_t_int
}
% 얻어진 clist의 출력
/\clist_use:Nn \l_quot_clist {;~}/
}

```

두 가지 예를 들어보겠다.

<pre> \eucontfrac{45}{16}\l \eucontfrac{9}{17} </pre>	<pre> /2; 1; 4; 3/ /0; 1; 1; 8/ </pre>
---	--

$/2; 1; 4; 3/$ 이 의미하는 것은 다음과 같다.

$$\frac{45}{16} = 2 + \frac{1}{1 + \frac{1}{4 + \frac{1}{3}}}$$

그 다음 결과인  $/0; 1; 1; 8/$ 이 의미하는 것은 다음과 같다.

$$\frac{9}{17} = \frac{1}{1 + \frac{1}{1 + \frac{1}{8}}}$$

이와 같이 유클리드 알고리즘은 분수로 표시되는 유리수의 연분수 전개에 사용할 수 있다 [4, 11]. 분자와 분모에 대하여 유클리드 알고리즘을 반복 적용하면서 각 단계의 몫을 차례로 얻어 나열하면 연분수가 된다.

연분수를  $\text{T}_{\text{E}}\text{X}$ 으로 하여금 계산하게 하고 이를 자동 조판하는 문제는 남수진의 논문 [4]에 잘 설명되어 있다. 이 논문에서 제시하는 코드를  $\text{expl3}$ 로 번역한 [7]도 참고할 수 있는데, 이 글은 남수진의 논문이 plain  $\text{T}_{\text{E}}\text{X}$ 으로 제시한 내용을 거의 그대로  $\text{expl3}$ 로 재현하고 있으므로 plain  $\text{T}_{\text{E}}\text{X}$ 과  $\text{expl3}$ 의 차이를 한눈에 알 수 있는 좋은 참고자료이다. 이 논의들은 KTUG 게시물인 [6]에 요약되어 있다.

## 4 유클리드 호제법의 수계산 조판

유클리드 호제법으로 두 수의 최대공약수를 구하는 과정을 수계산할 때, 다음과 같이 하는 수가 있다.

	4165	6035	1
		4165	
2	4165	1870	
	3740		
	425	1870	4
		1700	
2	425	170	
	340		
	85	170	2
		170	
		0	

이를 더 간략히

2	4165	6035	1
	3740	4165	
2	425	1870	4
	340	1700	
	85	170	2
		170	
		0	

이같이 표현하는 경우도 있으나, 연분수와의 관계를 문제삼으려는 우리 목적에 두 번째 그림은 적절하지 못하다.

이 산법을  $\text{\LaTeX}$ 으로 조판하는 방법을 구현해본다. 단지 결과의 수만을 얻는 것과 달리 이 문제의 해결을 위해서는 각 단계별로 도식적 틀 안에 이를 식자해야 하는 문제가 있다.

noname는 [5]에서 자연수의 진법을 변환하기 위한 연속 나눗셈을 시각적으로 디스플레이하는 방법을 제시한 바가 있다. 같은 아이디어를 적용하여 이를 시도해보았다.

### 4.1 coffin의 아이디어

이런 유형의 문제는 몇 단계로 계산이 종료될지 사전에 알 수 없다. 따라서 미리 일정 공간을 확보하거나 선을 그어놓고 시작할 수 없다. 그래서 하나의 나눗셈이 이루어지는 한 단계를 `expl3`에서 제공하는 자료형인 `coffin`<sup>4</sup>에 넣고 단계별로 이 `coffin`들을 결합(join)해 나가다가, 마지막 단계에서 그 때까지 모인 `coffin` 덩어리를 한꺼번에 출력하는 방법을 쓰고 있다. 우리는 `expl3`의 `coffin` 함수만을 사용하였지만 실용적인 목적으로 이를 응용하려면 `xcoffins` 패키지를 참고하는 것이 편하다 [2]. 선을 표현하기 위하여 각 `coffin` 안에 일부만 선이 그어지는 `fbox`, 즉 `efbox`를 저장하였다 [3].

한 단계의 처리 과정은 다음과 같다.  $a$ 와  $b$ 는 인자로 주어진다.

<sup>4</sup>`coffin` 자료형은 결합을 위한 손잡이가 있는 `box`라고 생각하면 된다. [1,2]를 참고.

- (1)  $a > b$ 이면 `\lefty_draw:nn`를, 그렇지 않으면 `\righty_draw:nn`를 호출한다. 이 둘은 몫과 나머지를 배치하는 위치만 다르고 처리 과정은 본질적으로 동일하다. 이후는 `\lefty_draw:nn`만을 설명한다.
- (2)  $a$ 와  $b$ 를 `coffin`에 넣고 이 둘을 결합한다. 그러면 다음과 같은 `coffin`이 생성된다 (아직 식자는 이루어지지 않음). 이 `coffin`을  $A$ 라 하자.

$a$	$b$
-----	-----

- (3) 몫을 구하는 것은 `\int_div_truncate:nn`로 계산한다. 구한 몫( $q$ )을 `coffin`에 넣은 다음  $A$ 에 왼쪽에서 결합(join)<sup>5</sup>한다.

$q$	$a$	$b$
-----	-----	-----

- (4) 몫과 나누는 수를 곱한 값을  $a$ 의 아랫쪽 위치에서 `coffin`으로 결합한다.

$q$	$a$	$b$
	$bq$	

- (5) 나머지를 `\int_mod:nn`로 구한다. 만약 나머지가 1이거나 0이면 나머지를 넣은 `coffin`을  $A$ 의 아래에 붙이고 이를 출력한다. 재귀호출에 있어 종료 조건을 충족한 경우이다.

$q$	$a$	$b$
	$bq$	
	$r$	

- (6) 만약 1이나 0이 아니면  $b$ 와 나머지  $r$ 을 인자로 주어(즉  $a = b, b = r$ ) 이 함수를 재귀호출한다. 현재 과정의  $r$ 은 다음 번 호출 때에 나누는 수로 그려질 것이다.<sup>6</sup> 현재 결합되어 있는 `coffin`  $A$ 는 다음 번 절차에서 다시 사용되어 마지막의 종료조건을 충족하면 모두 출력(typeset)한다.

실제 함수의 구현은 필요한 선을 그리거나 그리지 않는 절차를 포함하고 “마지막에 나눈 수”가 최대 공약수임을 나타내기 위해 색을 칠하는 등, 조금 더 복잡하지만 기본적인 아이디어는 이러하다.

## 4.2 재귀호출

앞서 GCD를 구하는 함수는 `while` 문을 사용하여 구현하였다. 반면, 여기서는 각 단계별로 한 단계의 나눗셈을 하나의 함수로 표현하고 이를 종료조건이 만족할 때까지 재귀호출하는 방법을 이용하였다. 앞서 보인 한 단계 처리 함수를 나머지가 1 또는 0이 될 때까지 반복 실행하는 것이다.

```
\cs_new:Npn \draw_euc_alg:nn #1 #2
{
  ....

  \int_compare:nTF { \l_r_int == 0 }
  {
```

<sup>5</sup> `coffin`의 결합(join) 또는 부착(attach) 위치나 방법은 매우 자유롭다. 그러나 이 글이 다루는 문제의 범위에서는 모든 `coffin`을 잇대어 붙이면 되기 때문에 결합점(handle)을 사각형의 꼭짓점으로만 한정해도 괜찮았다.

<sup>6</sup> 실제 구현에서는  $(r, b)$  순으로 인자를 주어 호출한다. 그 이유는 이 그림에서  $b$ 의 위치가 고정되어 있어야 이어내려 쓴 것임을 나타낼 수 있기 때문이다. 그리고 다음 번에는 몫이 그려지는 위치가 건너편이 될 것이다.



%% 얻어진 *clist*의 출력

```
\int_zero:N \l_tmpa_int
[
  \clist_map_inline:Nn \l_quot_clist
  {
    \int_incr:N \l_tmpa_int
    \int_compare:nTF { \l_tmpa_int < 2 }
    {
      ##1;~
    }
    {
      ##1
      \int_compare:nT
      { \l_tmpa_int < \clist_count:N \l_quot_clist }
      { ,~}
    }
  }
]
```

`\euccontfrac{23}{9}`

[2; 1, 1, 4]

수계산 산법에서 연분수 리스트 표기의 각 항은 오른쪽, 왼쪽 날개에 표기되는 몫을 순차적으로 읽어서 얻을 수 있다. 단, 우리의 다이어그램에서 서로소인 두 수는 나머지가 1이 되는 순간 계산을 끝내므로 이를 한 단계 더 진행시키기 위해 `\exttozeroes`를 선언하고, `\eucgcd{23}{9}`를 명령한다.

2	23	9	
	18		
	5	9	1
		5	
1	5	4	
	4		
	1	4	4
		4	
		0	

이 도식에서 왼쪽 날개에 있는 숫자부터 차례로 읽으면 [2; 1, 1, 4]를 얻는다. 이것은 `\euccontfrac`으로 얻은 결과와 같다.

$$\frac{23}{9} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{4}}}$$



이를 얻는 계산 과정을 차례로 나열해보면,

$$\frac{23}{9} = 2 + \frac{5}{9} = 2 + \frac{1}{\frac{9}{5}} = 2 + \frac{1}{1 + \frac{4}{5}} = 2 + \frac{1}{1 + \frac{1}{\frac{5}{4}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{4}}}$$

$$\begin{array}{l} 23 \div 9 = 2 \dots 5 \\ 9 \div 5 = 1 \dots 4 \\ 5 \div 4 = 1 \dots 1 \\ 4 \div 1 = 4 \end{array}$$

와 같이 된 것이다.

만약 분모가 분자보다 큰 수일 때에는,

3	17	5
	15	
1	3	2
	2	
	1	2
	2	
	0	

이런 모양이 되는데, 이 때는 처음 숫자가 오른쪽 날개부터 있으니까, 왼쪽 날개에 0이 있다고 생각하고 읽어서  $3/17 = [0; 5, 1, 2]$ 를 얻을 수 있다. 연분수의 리스트 표현형의 맨처음에 0을 추가하면 원래 수의 역수가 된다.  $17/3 = [5; 1, 2]$ 인 것이다.

## 5 결론

L<sup>A</sup>T<sub>E</sub>X3의 핵심 기능인 `expl3`는 이제 L<sup>A</sup>T<sub>E</sub>X 자체의 표준적인 언어로 자리를 잡았다. 다양한 자료형(data type)과 인터페이스 함수들의 도움으로 이전에 비하면 훨씬 수월하게 문제를 해결할 수 있는 가능성이 확장되었다.

이 글이 다루는 문제 자체가 정수에 관련된 것이라서 `int` 자료형이 대폭 사용되었고, 이를 시각적으로 표현하기 위해 `coffin` 자료형을 응용하였다. 이와 더불어 `while` 반복문이나 재귀호출과 같은 방법으로 반복 작업을 쉽게 구현할 수 있음을 보였다.

## 참고 문헌

- [1] The L<sup>A</sup>T<sub>E</sub>X Project, *The L<sup>A</sup>T<sub>E</sub>X3 Interfaces*, 2021-02-18. T<sub>E</sub>X Live Documentation. <https://texdoc.org/serve/interface3/0>
- [2] The L<sup>A</sup>T<sub>E</sub>X Project, *The xcoffins package: Design level coffins*, 2021-02-18. T<sub>E</sub>X Live Documentation. <https://texdoc.org/serve/xcoffins/0>.
- [3] José Romildo Malaquias, “The efbbox package,” <https://texdoc.org/serve/efbbox/0>.

- [4] 남수진, 〈연분수 자동 조판〉, *The Asian Journal of T<sub>E</sub>X*, Vol. 1, No. 1, 2007. pp. 57–65. <http://ajt.ktug.org/2007/0101sjnam.pdf>
- [5] noname, 〈진법 변환 나누기〉, KTUG 게시물, 2018/11/6, [http://www.ktug.org/xe/index.php?mid=KTUG\\_open\\_board&document\\_srl=232524](http://www.ktug.org/xe/index.php?mid=KTUG_open_board&document_srl=232524).
- [6] noname, 〈연분수 조판 revisited〉, KTUG 게시물, 2018/09/25, [http://www.ktug.org/xe/index.php?document\\_srl=231984&mid=KTUG\\_open\\_board](http://www.ktug.org/xe/index.php?document_srl=231984&mid=KTUG_open_board).
- [7] 이엑스피엘쓰리스타디그룹, 〈남수진의 ‘연분수 자동 조판’에 정의된 `\fraction` 매크로〉, [6]의 첨부파일.
- [8] 이엑스피엘쓰리스타디그룹, 《예제로 배우는 Expl3》, <http://wiki.ktug.org/wiki/wiki.php/expl3%20Study%20Group>.
- [9] “유클리드 호제법,” 《한국어 위키백과》, [https://ko.wikipedia.org/wiki/%EC%9C%A0%ED%81%B4%EB%A6%AC%EB%93%9C\\_%ED%98%B8%EC%A0%9C%EB%B2%95](https://ko.wikipedia.org/wiki/%EC%9C%A0%ED%81%B4%EB%A6%AC%EB%93%9C_%ED%98%B8%EC%A0%9C%EB%B2%95)
- [10] 유클리드, 《원론》. 웹페이지. *Euclid’s Element: Book VII*, <https://mathcs.clarku.edu/~djoyce/elements/bookVII/bookVII.html>
- [11] “Continued Fraction,” Wolfram Math World, <https://mathworld.wolfram.com/ContinuedFraction.html>.

## A 부록

수계산 조판의 소스 코드.

```
\usepackage{xcolor}
\usepackage{efbox}
\efboxsetup{hidealllines}

\ExplSyntaxOn
\coffin_new:N \l_output_coffin
\int_new:N \l_r_int
\bool_new:N \l_exttzero_bool

\NewDocumentCommand \exttzeroyes {}
{
  \bool_set_true:N \l_exttzero_bool
}
\NewDocumentCommand \exttzerono {}
{
  \bool_set_false:N \l_exttzero_bool
}

\NewDocumentCommand \SetCof { o 0{2.5em} o m m }
{
  \IfValueT { #1 }
  {
    \clist_set:Nn \l_tmpa_clist { #1 }
    \tl_clear:N \l_tmpa_tl
    \clist_map_inline:Nn \l_tmpa_clist
    {
      \str_case:nn { ##1 }
      {
        { l } { \tl_put_right:Nn \l_tmpa_tl { leftline = true, } }
        { r } { \tl_put_right:Nn \l_tmpa_tl { rightline = true, } }
        { t } { \tl_put_right:Nn \l_tmpa_tl { topline = true, } }
        { b } { \tl_put_right:Nn \l_tmpa_tl { bottomline = true, } }
      }
    }
  }

  \IfValueT { #3 }
  {
    \tl_put_right:Nn \l_tmpa_tl { linecolor = #3 }
  }

  \coffin_if_exist:cF { l_#4_coffin }
  {
    \coffin_new:c { l_#4_coffin }
  }

  \hcoffin_set:cn { l_#4_coffin }
  {
```

```

\exp_last_unbraced:NNo \efbox [\l_tmpa_tl]
{ \makebox [ #2 ] [ c ] { #5 \rule[-3pt]{0pt}{10pt} } }
}
}

\NewDocumentCommand \PrintCof { m }
{
\coffin_typeset:cnnnn { \l_#1_coffin }
{ t } { l } { 0pt } { 0pt }
}

\NewDocumentCommand \JoinCof { mmmm }
{
\coffin_join:cnnnnnn { \l_#1_coffin } { #3 } { #4 }
{ \l_#2_coffin } { #5 } { #6 } { 0pt } { 0pt }
}

\NewDocumentCommand \eucgcd { m m }
{
\coffin_clear:N \l_output_coffin
\int_zero:N \l_idxcnt_int
\draw_euc_alg:nn { #1 } { #2 }
}

\int_new:N \l_idxcnt_int

\cs_new:Npn \draw_euc_alg:nn #1 #2
{
\int_incr:N \l_idxcnt_int

\int_compare:nTF { #1 > #2 }
{
\lefty_draw:nn { #1 } { #2 }
}
{
\righty_draw:nn { #1 } { #2 }
}
}

\cs_new:Npn \lefty_draw:nn #1 #2
{
\int_set:Nn \l_tmpa_int { \int_div_truncate:nn { #1 } { #2 } }
\int_set:Nn \l_tmpb_int { \l_tmpa_int * #2 }
\int_set:Nn \l_r_int { \int_mod:nn { #1 } { #2 } }

\int_compare:nTF { \l_idxcnt_int > 1 }
{
\SetCof[l,t][2.5em]{A}{#1}
\int_compare:nTF { \l_r_int == 0 }
{
\SetCof[r,t][2.5em]{B}{\color{red}\bfseries #2 }
}
}
}

```

```

    {
        \SetCof[r,t][2.5em]{B}{#2}
    }
}
{
    \SetCof[l][2.5em]{A}{#1}
    \int_compare:nTF { \l_r_int == 0 }
    {
        \SetCof[r][2.5em]{B}{ \color{red}\bfseries #2 }
    }
    {
        \SetCof[r][2.5em]{B}{#2}
    }
}
\SetCof[r][1em][white]{q}{\int_use:N \l_tmpa_int }
\SetCof[l][2.5em]{qa}{ \int_eval:n { #2 * \l_tmpa_int } }

\SetCof[r][2.5em]{empty}{}
\coffin_attach:NnnNnnnn \l_B_coffin
    { r } { b } \l_empty_coffin
    { r } { t } { 0pt } { 0pt }

\JoinCof ABrtlt
\JoinCof A{qa}lbt
\JoinCof Aqltrt

\bool_if:NTF \l_exttozero_bool
{
    \int_compare:nTF { \l_r_int == 0 }
}
{
    \int_case:nnTF { \l_r_int }
    {
        { 0 } { }
        { 1 } { }
    }
}
{
    \SetCof[l,t][2.5em]{R}{ \bfseries \int_use:N \l_r_int }
    \SetCof[r,t][2.5em]{empty}{}
    \SetCof[l][1em][white]{blank}{}
    \JoinCof R{empty}rtlt
    \JoinCof R{blank}ltrt
    \JoinCof ARLbtl

    \int_compare:nTF { \l_idxcnt_int == 1 }
    {
        \JoinCof{output}{A}tltl
    }
    {
        \JoinCof{output}{A}bltl
    }
}

```

```

\PrintCof{output}
}
{
\int_compare:nTF { \l_idxcnt_int == 1 }
{
\JoinCof{output}{A}tltl
}
{
\JoinCof{output}{A}bltl
}

\exp_args:Nxx \draw_euc_alg:nn { \int_use:N \l_r_int } { #2 }
}
}

\cs_new:Npn \righty_draw:nn #1 #2
{
\int_set:Nn \l_tmpa_int { \int_div_truncate:nn { #2 } { #1 } }
\int_set:Nn \l_tmpb_int { \l_tmpa_int * #1 }
\int_set:Nn \l_r_int { \int_mod:nn { #2 } { #1 } }
\tl_set:Nx \l_r_tl { \int_use:N \l_r_int }
\tl_set:Nx \l_a_tl { \int_use:N \l_tmpa_int }
\tl_set:Nx \l_b_tl { \int_use:N \l_tmpb_int }

\int_compare:nTF { \l_idxcnt_int > 1 }
{
\SetCof[r,t][2.5em]{A}{#2}
\int_compare:nTF { \l_r_int == 0 }
{
\SetCof[l,t][2.5em]{B}{ \color{red}\bfseries #1 }
}
{
\SetCof[l,t][2.5em]{B}{#1}
}
}
{
\SetCof[r][2.5em]{A}{#2}
\int_compare:nTF { \l_r_int == 0 }
{
\SetCof[l][2.5em]{B}{ \color{red}\bfseries #1 }
}
{
\SetCof[l][2.5em]{B}{#1}
}
}

\SetCof[l][1em][white]{q}{ \l_a_tl }
\SetCof[r][2.5em]{qa}{ \int_eval:n { #1 * \l_tmpa_int } }

\SetCof[l][1em][white]{blank}{ }

\JoinCof ABltrt

```

```

\SetCof[l][2.5em]{empty}{}
\coffin_attach:NnnNnnnn \l_A_coffin
  { l } { b } \l_empty_coffin
  { l } { t } { 0pt } { 0pt }

\JoinCof A{qa}rbrt
\JoinCof Aqrtlt
\JoinCof A{blank}ltrt

\bool_if:NTF \l_exttzero_bool
{
  \int_compare:nTF { \l_r_int == 0 }
}
{
  \int_case:nnTF { \l_r_int }
  {
    { 0 } { }
    { 1 } { }
  }
}
{
  \SetCof[r,t][2.5em]{R}{ \bfseries \l_r_tl }
  \SetCof[l,t][2.5em]{empty}{}
  \JoinCof R{empty}ltrt
  \JoinCof R{blank}rtlt
  \JoinCof ARrbrt

  \int_compare:nTF { \l_idxcnt_int == 1 }
  {
    \JoinCof{output}Atl tl
  }
  {
    \JoinCof{output}Abl tl
  }

  \PrintCof{output}
}
{
  \int_compare:nTF { \l_idxcnt_int == 1 }
  {
    \JoinCof{output}Atl tl
  }
  {
    \JoinCof{output}Abl tl
  }

  \exp_args:Nxx \draw_euc_alg:nn { #1 } { \int_use:N \l_r_int }
}
}
\ExplSyntaxOff

```