나는 평소에 유튜브를 정말 많이 보는 편인데, 최근에 수학 영상 두 개를 재미있게 봤다.
여기서 영감을 얻어 글을 써본다.
이쪽 분야에 전문적인 사람이 쓰는 글이 아니니, 테트레이션에 대해 모르는 사람이 대강 개념 익히는 정도로 읽으면 좋을 듯.

https://youtu.be/PlnTKK2w9og

https://youtu.be/1ooGWvwdcno

내가 생각하는 프로그래밍의 핵심 중 하나는 반복이다.
반복되는 모습에서 찾아낸 규칙성을 간결한 코드로 정의하는 것이 무엇보다 중요하다.
수학도 반복 참 좋아한다.

1. (덧셈) $a + n = a + \underbrace{1 + \cdots + 1}_{n번}$

2. (곱셈) $a \times n = \underbrace{a + \cdots + a}_{n번}$

3. (거듭제곱) $a^n = \underbrace{a \times \cdots \times a}_{n번}$

사진 출처

덧셈도 일종의 반복이다.
x에 n을 더한다는 말은, x에 1을 n 번 반복해서 더한다는 말로 나타낼 수 있다.

곱셈도 반복이다.
x에 n을 곱한다는 말은, x를 n 번 반복해서 더한다는 말로 나타낼 수 있다.

거듭제곱도 반복이다.
x를 n 제곱한다는 말은, x를 n 번 반복해서 곱한다는 말로 나타낼 수 있다.

그다음도 있을까?
내가 정규교육과정에서 배웠던 수학에서의 연산 반복은 거듭제곱까지였다.
그다음 단계도 있다는 걸 위 두 영상을 보고 나서야 처음 알았다.
거듭제곱의 다음 단계가 바로 테트레이션이다.

1. 덧셈

   $a + n = a + \underbrace{1 + 1 + \cdots + 1}_{n}$

   n번 a에 1이 더해졌다.

2. 곱셈

   $a \times n = \underbrace{a + a + \cdots + a}_{n}$

   n번 a가 덧셈으로 결합했다.

3. 거듭제곱

   $a^n = \underbrace{a \times a \times \cdots \times a}_{n}$

   n번 a가 곱셈으로 결합했다.

4. 테트레이션

   ${}^n a = \underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{n}$

   n번 a가 오른쪽에서 왼쪽으로 결합했다.

사진 출처

영상에 따르면 덧셈은 1차 연산이라고도 불리는데, 곱셈은 덧셈의 반복이므로 2차 연산, 거듭제곱은 덧셈의 반복이므로 3차 연산이라고 부른다.
테트레이션은 거듭제곱의 반복이므로 4차 연산이라고 부른다.
4를 의미하는 'tetra'와 반복을 의미하는 'iteration'이 합쳐져 tetration(테트레이션)이라는 이름이 붙여졌다.

테트레이션은 거듭제곱의 반복이다.
x, n을 테트레이션한다는 말은, x를 n 번 반복해서 거듭제곱한다는 말로 나타낼 수 있다.
n 개의 x를 밑과 지수 형태로 쌓아 올리는 것이다.

n 차 연산을 n-1 차 연산의 반복이라고 하면, 테트레이션 다음 차 연산도 추론할 수 있다.
테트레이션의 반복을 펜테이션 또는 5차 연산이라 부르고, 펜테이션의 반복을 헥세이션 또는 6차 연산이라 부른다.
이 반복되는 n 차 연산을 하이퍼 연산, hyper operation이라고 부르는데, 제타위키에 깔끔하게 정리된 문서(링크)가 있어 가져와봤다.

2. 정의

a에 대한 n차 하이퍼 연산을 $H_n(a,b)$라고 할 때,

- $H_n(a,b) = \begin{cases} b+1 & (n=0) \\ a & (n=1, b=0) \\ 0 & (n=2, b=0) \\ 1 & (n \geq 3, b=0) \\ H_{n-1}(a, H_n(a, b-1)) & \text{otherwise} \end{cases}$
  - 위에서 a는 밑, b는 지수(또는 하이퍼 지수), n은 계수라고 한다.
  - 각각의 n에 대해, n=4일 때 테트레이션(tetration), n=5일 때 펜테이션(pentation), ..., (n을 뜻하는 그리스어 접두사)-tion부른다.

여기에는 0차 연산도 정의되어 있다.
'succession', 우리 말로는 '다음 수'라고 부르는 듯.
말 그대로 b에 1을 더한 b의 다음 수를 구하는 연산이라고 보면 된다.

1차 연산을 0차 연산의 반복이라고 본다면 앞서 표현한 것과 다르게 이런 식으로도 표현할 수 있겠다.
x에 n을 더한다는 말은, x의 n 번째 다음 수를 구한다는 말과 같다는 식으로.

3. 표기법

- 표기법이 매우 다양함

이름	$H_n(a,b)$에 대응되는 표기	조건	비고
커누스 윗화살표 표기법	$a \uparrow^{n-2} b$	$n \geq 3$	커누스가 사용함
굿스틴 표기법	$G(n,a,b)$		Goodstein이 사용함
초기 아커만 함수	$\phi(a,b,n-1)\ (1 \leq n \leq 3)$ $\phi(a,b-1,n-1)\ (n \geq 4)$		빌헬름 아커만이 사용함
아커만 함수	$A(n,b-3)+3$	$a=2$	
상자 표기법	$a \boxed{n} b$		Rubtsov와 Romerio가 사용함
위첨자 표기법	$a^{(n)}b$		로버트 무나포가 사용함
아래첨자 표기법	$a_{(n)}b$		
연산자 표기법	$aO_{n-1}b$	$n \geq 1$	John Donner와 Alfred Tarski가 사용함
대괄호 표기법	$a[n]b$		인터넷상에서 ASCII의 편의성에 의해 자주 사용됨
Conway 연쇄 화살표 표기	$a \to b \to (n-2)$	$n \geq 3$	John Horton Conway가 사용함
Bowers' Exploding Array Function	$\{a,b,n,1\}$	$n \geq 1$	조나단 바워스가 사용함

하이퍼 연산의 표기법에는 여러 가지가 있다고 하는데, 커누스 윗화살표 표기법으로 표기된 자료가 많았다.

4. 예시

- $a \uparrow^{n-2} b = H_n(a,b)$의 연산 과정과 결과는 아래와 같다.

n	연산 과정	결과	비고
$n=0$	$1+\underbrace{1+1+\cdots+1}_{b\text{개}}$	$b+1$	
$n=1$	$a+\underbrace{1+1+\cdots+1}_{b\text{개}}$	$a+b$	덧셈
$n=2$	$\underbrace{a+a+\cdots+a}_{b\text{개}}$	$a\times b$	곱셈
$n=3$	$\underbrace{a\times a\times\cdots\times a}_{b\text{개}}$	a^b	거듭제곱
$n=4$	$\underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{b\text{개}}$	$a \uparrow\uparrow b$	테트레이션(tetration)
$n=5$	$\underbrace{a \uparrow\uparrow a \uparrow\uparrow a \uparrow\uparrow \cdots \uparrow\uparrow a}_{b\text{개}}$	$a \uparrow\uparrow\uparrow b$	펜테이션(pentation)
$n=6$	$\underbrace{a \uparrow\uparrow\uparrow a \uparrow\uparrow\uparrow a \uparrow\uparrow\uparrow \cdots \uparrow\uparrow\uparrow a}_{b\text{개}}$	$a \uparrow^4 b$	헥세이션(hexation)
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	$\underbrace{a \uparrow^{n-3} a \uparrow^{n-3} a \uparrow^{n-3} \cdots \uparrow^{n-3} a}_{b\text{개}}$	$a \uparrow^{n-2} b$	(n을 의미하는 그리스어 접두사)+'-tion'

커누스 윗화살표 표기법으로 연산 결과를 표기하면 n = 3인 거듭제곱 연산부터 a와 b 사이에 화살표가 하나씩 늘어난다고 보면 된다.
화살표가 너무 많아지면 표기하기 곤란해서인지 화살표의 개수는 지수 형태로도 표기할 수 있다.

계산해 보면 알겠지만 테트레이션 연산부터는 밑이나 지수가 아주 조금만 커져도 결괏값이 무지막지하게 커진다.
나는 거듭제곱도 나름 큰 수를 표현하기에 충분하다고 생각했는데, 그보다도 몇 단계 위의 연산이니...
파이썬 소스 코드로 작성해서 실행해도 대부분 결과를 구하기가 어려운데, 그렇게 큰 숫자를 이렇게 간단한 형태로 표기할 수 있다는 것이 놀라울 따름이다.
수학적 미가 무엇인지 조금은 알 것 같은 느낌.

이제 테트레이션이 무엇인지 조금은 알았으니, 파이썬으로 테트레이션 계산 프로그램을 구현해 보기로 했다.
사실 이미 하이퍼 연산 파이썬 라이브러리는 존재하지만(Hyperoperators 깃허브 링크), 그보다는 조금 더 귀여운(?) 수준으로 구현해 보았다.
'Learning Scientific Programming with Python'이라는 사이트에 적절한 테트레이션 솔루션이 있어, 해당 솔루션에 코드를 덧붙여봤다.

Tetration

Solution P2.7.8

Hide Solution

The code below presents both recursive and non-recursive functions for calculating the tetration. ${}^{3}5$ has 2185 digits; ${}^{5}2$ has 19729 digits.

```
import math

def tet(x, n):
    """ Tetration, ^nx, by loop over decreasing exponent counter. """
    if n == 0:
        return 1
    p = x
    while n > 1:
        p = x**p
        n -= 1
    return p

def tet2(x, n):
    """ Tetration, ^nx, by recursion. """
    if n == 0:
        return 1
    return x**tet2(x, n-1)

x, n = 5, 3
t = tet2(x,n)
print('tet({:d}, {:d}) = {:d}'.format(x, n, t))
ndigits = int(math.log10(t)) + 1
print('(a number with {:d} digits)'.format(ndigits))
```

사진 출처

구현한 코드는 아래와 같다.

```python
# Solution P3.10.0
import math

def suc(x, n):
    """ Succession, n+1 """
    return n + 1

def add(x, n):
    """ Addition, x+n """
    a = x
    while n > 0:
        a = suc(None, a)
        n -= 1
    return a

def mul(x, n):
    """ Multiplication, x*n """
    if n == 0:
        return 0
    a = x
    while n > 1:
        a = add(a, x)
        n -= 1
    return a

def pow(x, n):
    """ Exponentiation, x**n or x^n or x↑n"""
    if n == 0:
        return 1
    a = x
    while n > 1:
        a = mul(a, x)
        n -= 1
    return a

def tet(x, n):
    """ Tetration, x^^n or x↑↑n or ^nx, by loop over decreasing exponent counter. """
    if n == 0:
        return 1
    if n == -1:
        return 0
    p = x
    while n > 1:
        p = x**p # p = pow(x, p)
        n -= 1
    return p
def tet2(x, n):
    """ Tetration, x^^n or x↑↑n or ^nx, by recursion. """
    if n == 0:
        return 1
    if n == -1:
        return 0
    return x**tet2(x, n-1) # return pow(x, tet2(x, n-1))

def pen(x, n):
    """ Pentation, x^^^n or x↑↑↑n """
    p = x
    while n > 1:
        p = tet(x, p)
        n -= 1
```

```
        return p

    def hex(x, n):
        """ Hexation, x^^^^n or x↑↑↑↑n """
        p = x
        while n > 1:
            p = pen(x, p)
            n -= 1
        return p

    # x, n은 0 또는 양의 정수
    x, n = 5, 3
    # x, n = 2, 5

    # 덧셈
    t = add(x, n)
    print('add({:d}, {:d}) = {:d}'.format(x, n, t))

    # 곱셈
    t = mul(x, n)
    print('mul({:d}, {:d}) = {:d}'.format(x, n, t))

    # 거듭제곱
    t = pow(x, n)
    print('pow({:d}, {:d}) = {:d}'.format(x, n, t))

    # 테트레이션
    t = tet(x, n) # t = tet2(x, n)
    print('tet({:d}, {:d}) = {:d}'.format(x, n, t))
    ndigits = int(math.log10(t)) + 1
    print('(a number with {:d} digits)'.format(ndigits))

    """ 출력
    add(5, 3) = 8
    mul(5, 3) = 15
    pow(5, 3) = 125
    tet(5, 3) = 19110125979454775203564045597039645991980810489900943371395127892465205302423
    (a number with 2185 digits)
    """
```

5차, 6차도 함수 형태로 구현은 해뒀는데, 아마 정상 동작하지는 않을 것이다.
차수가 늘어날수록 함수 호출 횟수가 무지막지하게 늘어나다 보니 당장 테트레이션부터 pow를 못 쓴다.
5차, 6차에서의 성질도 반영되지 않았으니, 코드는 참고용으로만 활용하는 것을 추천한다.

이래저래 재미있는 공부가 되었다.
끝!