

[COM1002]
프로그래밍1

Dynamic Programming

#07. 표채워풀기

김현하

한양대학교 ERICA 소프트웨어학부

2021.11.2.

2021년도 2학기

목차

- 피보나치 수열
 - 재귀 알고리즘, 동적계획 알고리즘
- 조합
 - 재귀 알고리즘, 파스칼의 삼각형
- 1까지 줄이는 최소 스텝
 - 재귀 알고리즘, 탐욕 알고리즘, 메모해두기 알고리즘, 동적계획 알고리즘
- 하노이의 탑

피보나치 수열

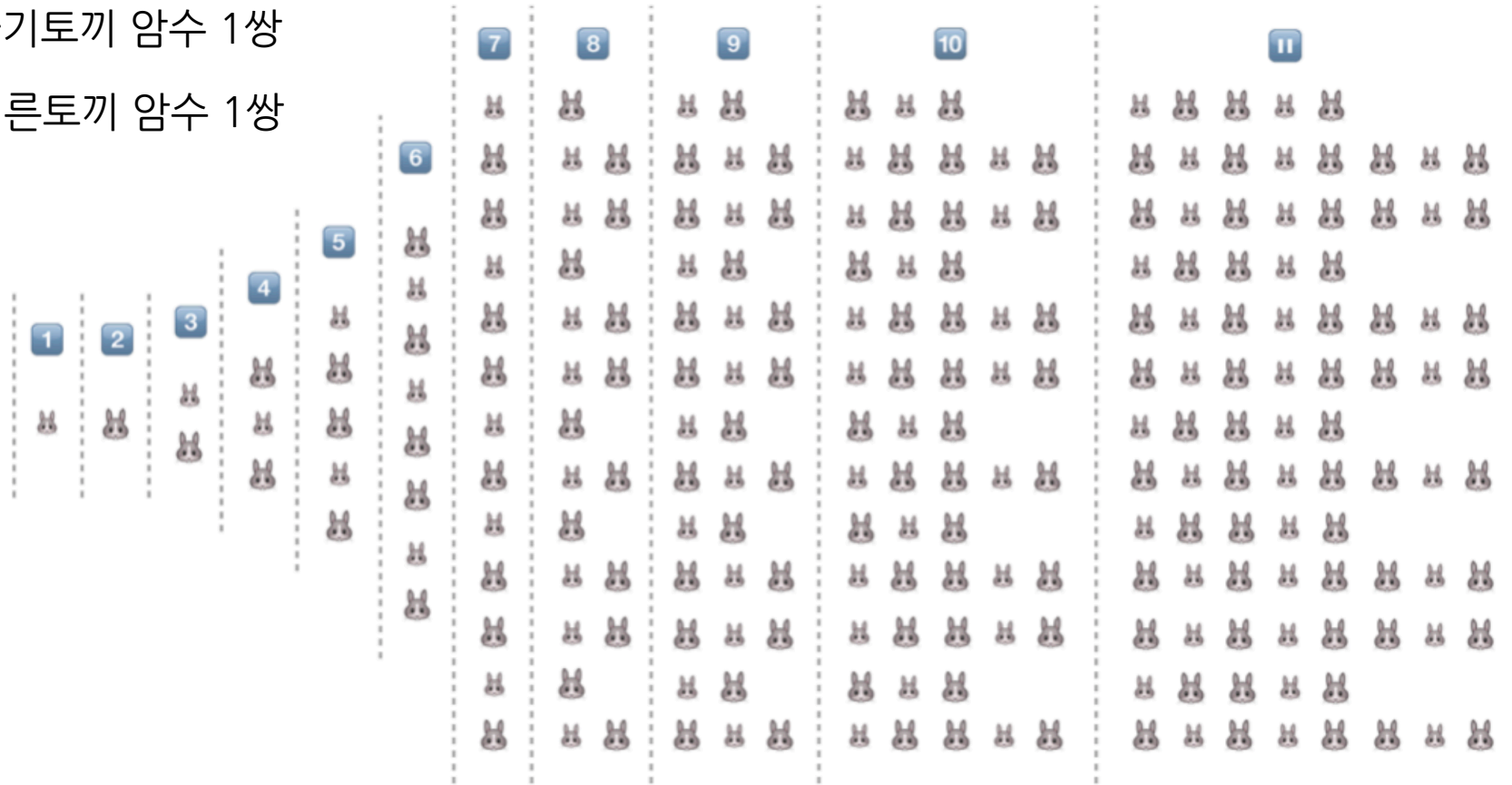
피보나치 수열

- 이탈리아의 수학자 레오나르도 피보나치(Leonardo Fibonacci 1170~1250년)가 1202년 제시한 수열
- 달나라에 간 토끼의 번식 규칙
 - 어른 토끼 암수 1쌍은 매달 아기 토끼 암수 1쌍씩을 낳는다
 - 아기 토끼는 태어난 지 1달이 지나면 어른 토끼가 되어 번식이 가능해진다
 - 토끼는 늙지도 않고 죽지도 않는다

피보나치 수열

🐰 : 아기토끼 암수 1쌍

🐰 : 어른토끼 암수 1쌍



달	0	1	2	3	4	5	6	7	8	9	10	11	12	...
아기 토끼	0	1	0	1	1	2	3	5	8	13	21	34	55	...
어른 토끼	0	0	1	1	2	3	5	8	13	21	34	55	89	...
합	0	1	1	2	3	5	8	13	21	34	55	89	144	...

피보나치 수열

- 5년(60개월) 후에 토끼 가족의 규모는?

$$\text{fib}_i = \underbrace{\text{rabby}_i}_{\text{어른 토끼 암수}} + \underbrace{\text{bunny}_i}_{\text{아기 토끼 암수}}$$

$$\text{rabby}_i = \text{fib}_{i-1}$$

이달의 어른 토끼 쌍의 수

지난달의 전체 토끼 쌍의 수

지난달의 어른 토끼 쌍의 수

rabby_{i-1}

(이번달에 어른이 된) 지난달의 아이 토끼 쌍의 수

bunny_{i-1}

$$\text{bunny}_i = \text{fib}_{i-2}$$

이달의 아이 토끼 쌍의 수

지지난달의 전체 토끼 쌍의 수

지난달의 어른 토끼 쌍의 수

rabby_{i-1}

$$\text{fib}_i = \text{fib}_{i-1} + \text{fib}_{i-2}$$

피보나치 수열의 재귀 알고리즘

$$fib(n) = \begin{cases} fib(n-1) + fib(n-2) & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 \end{cases}$$

```
1  def fib(n):  
2      if n > 1:  
3          return fib(n - 1) + fib(n - 2)  
4      else:  
5          return n
```

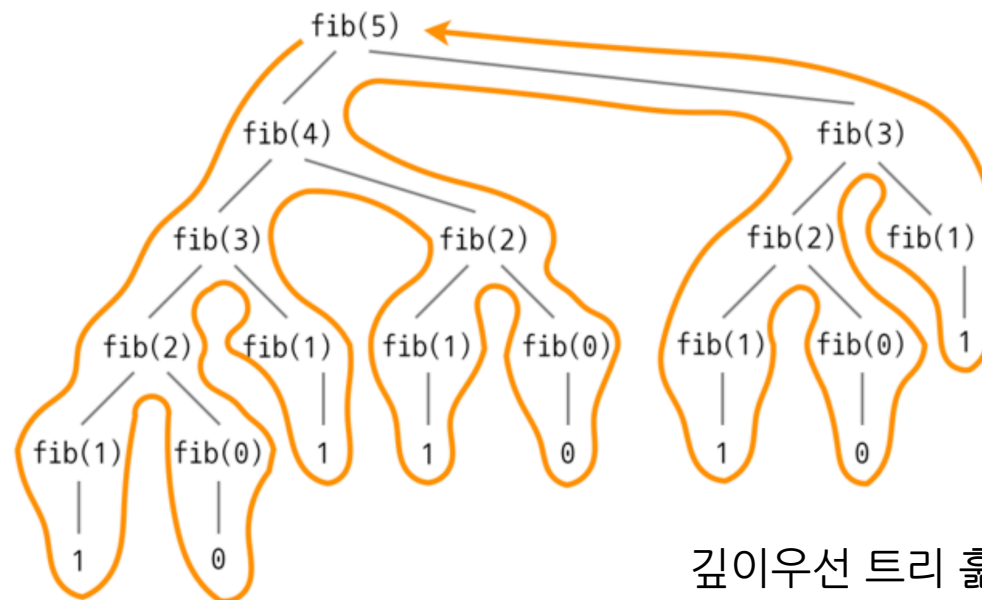
피보나치 수열의 재귀 알고리즘

```
1 def fib(n):  
2     if n > 1:  
3         return fib(n - 1) + fib(n - 2)  
4     else:  
5         return n
```

```
fib(5)  
⇒ fib(4) + fib(3)  
⇒ (fib(3) + fib(2)) + fib(3)  
⇒ ((fib(2) + fib(1)) + fib(2)) + fib(3)  
⇒ (((fib(1) + fib(0)) + fib(1)) + fib(2)) + fib(3)  
⇒ (((1 + fib(0)) + fib(1)) + fib(2)) + fib(3)  
⇒ (((1 + 0) + fib(1)) + fib(2)) + fib(3)  
⇒ ((1 + fib(1)) + fib(2)) + fib(3)  
⇒ ((1 + 1) + fib(2)) + fib(3)  
⇒ (2 + fib(2)) + fib(3)  
⇒ (2 + (fib(1) + fib(0))) + fib(3)  
⇒ (2 + (1 + fib(0))) + fib(3)  
⇒ (2 + (1 + 0)) + fib(3)  
⇒ (2 + 1) + fib(3)  
⇒ 3 + fib(3)  
⇒ 3 + (fib(2) + fib(1))  
⇒ 3 + ((fib(1) + fib(0)) + fib(1))  
⇒ 3 + ((1 + fib(0)) + fib(1))  
⇒ 3 + ((1 + 0) + fib(1))  
⇒ 3 + (1 + fib(1))  
⇒ 3 + (1 + 1)  
⇒ 3 + 2  
⇒ 5
```


피보나치 수열의 재귀 알고리즘

```
1 def run_fib(n):  
2     from time import perf_counter  
3     start = perf_counter()  
4     answer = fib(n)  
5     finish = perf_counter()  
6     print("fib(", n, ") ⇒ ", answer, sep="")  
7     print(round(finish - start, 4), "seconds")
```



깊이우선 트리 훑기 | depth-first tree traversal

피보나치 수열의 표채워풀기 알고리즘

- 재귀 호출하는 대신, 목표로 하는 $\text{fib}(n)$ 까지 필요한 $\text{fib}(n-1)$, $\text{fib}(n-2)$ 를 계산해 올라감

$$\text{fib}(0) = 0$$

$$\text{fib}(1) = 1$$

$$\text{fib}(2) = \text{fib}(1) + \text{fib}(0) = 1 + 0 = 1$$

$$\text{fib}(3) = \text{fib}(2) + \text{fib}(1) = 1 + 1 = 2$$

$$\text{fib}(4) = \text{fib}(3) + \text{fib}(2) = 2 + 1 = 3$$

$$\text{fib}(5) = \text{fib}(4) + \text{fib}(3) = 3 + 2 = 5$$

$$\text{fib}(6) = \text{fib}(5) + \text{fib}(4) = 5 + 3 = 8$$

$$\text{fib}(7) = \text{fib}(6) + \text{fib}(5) = 8 + 5 = 13$$

$$\text{fib}(8) = \text{fib}(7) + \text{fib}(6) = 13 + 8 = 21$$

$$\text{fib}(9) = \text{fib}(8) + \text{fib}(7) = 21 + 13 = 34$$

$$\text{fib}(10) = \text{fib}(9) + \text{fib}(8) = 34 + 21 = 55$$

n	0	1	2	3	4	5	6	7	8	9	10	11	12	...
fib(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	...

피보나치 수열의 표채워풀기 알고리즘 **while** 루프 ver.

- 재귀 호출하는 대신, 목표로 하는 $\text{fib}(n)$ 까지 필요한 $\text{fib}(n-1)$, $\text{fib}(n-2)$ 를 계산해 올라감

```
1  def fib(n):  
2      if n > 1:  
3          i = 2  
4          a, b = 0, 1  
5          while i <= n:  
6              a, b = b, a + b  
7              i += 1  
8          return b  
9      else:  
10         return n
```

chapter 7, run_fib (pp 332, code 7-2.py)

n	0	1	2	3	4	5	6	7	8	9	10	11	12	...
fib(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	...

피보나치 수열의 동적계획 알고리즘 **for** 루프 ver.

- 재귀 호출하는 대신, 목표로 하는 $\text{fib}(n)$ 까지 필요한 $\text{fib}(n-1)$, $\text{fib}(n-2)$ 를 계산해 올라감

```

1  def fib(n):
2      if n > 1:
3          a, b = 0, 1
4          for _ in range(2, n+1):
5              a, b = b, a + b
6          return b
7      else:
8          return n

```

n	0	1	2	3	4	5	6	7	8	9	10	11	12	...
fib(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	...

피보나치 수열

```
1  def fib(n):  
2      if n > 1:  
3          a, b = 0, 1  
4          for _ in range(2, n+1):  
5              a, b = b, a + b  
6          return b  
7      else:  
8          return n
```

```
1  def fibseq(n):  
2      fibs = [0, 1]  
3      for i in range(2, n+1):  
4          fibs.append(fibs[i-1] + fibs[i-2])  
5      return fibs
```

```
1  def fib(n):  
2      return fibseq(n)[-1]
```

조합

조합

- 조합combination ${}_nC_r$
 - n 개의 자연수에서 순서에 상관없이 r 개를 뽑는 가지 수

$${}_nC_r = \begin{cases} {}_{n-1}C_{r-1} + {}_{n-1}C_r & \text{if } r \neq 0 \text{ and } r \neq n \\ 1 & \text{if } r = 0 \\ 1 & \text{if } r = n \end{cases}$$

조합의 재귀 알고리즘

- 조합combination ${}_nC_r$

- n 개의 자연수에서 순서에 상관없이 r 개를 뽑는 가지 수

$${}_nC_r = \begin{cases} {}_{n-1}C_{r-1} + {}_{n-1}C_r & \text{if } r \neq 0 \text{ and } r \neq n \\ 1 & \text{if } r = 0 \\ 1 & \text{if } r = n \end{cases}$$

```
1  def comb(n, r):
2      if r != 0 and r != n:
3          return comb(n-1,r-1) + comb(n-1,r)
4      else:
5          return 1
```

time.perf_counter 로 시간 측정해보기

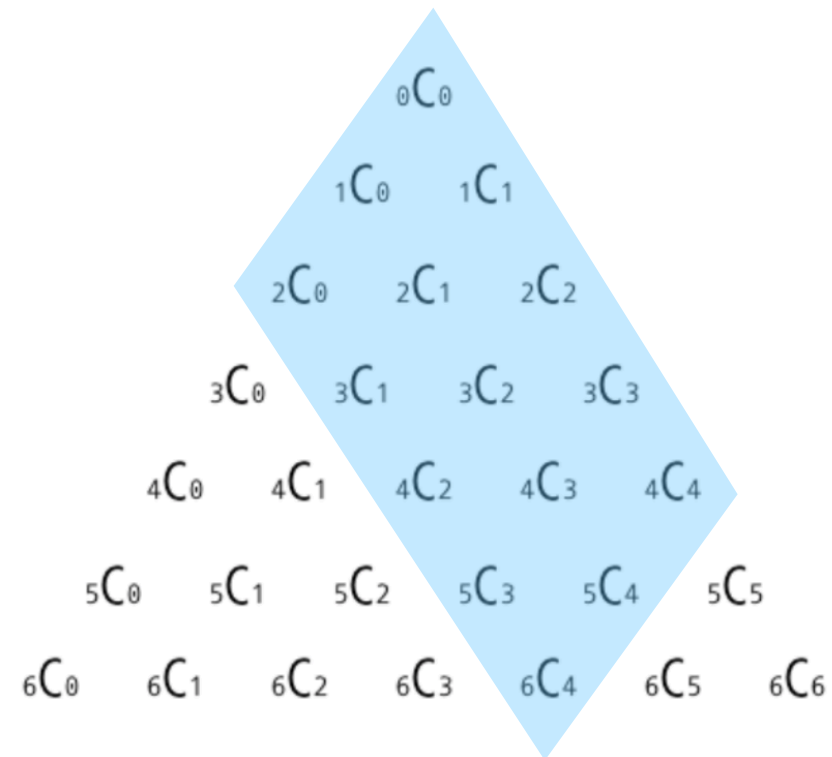
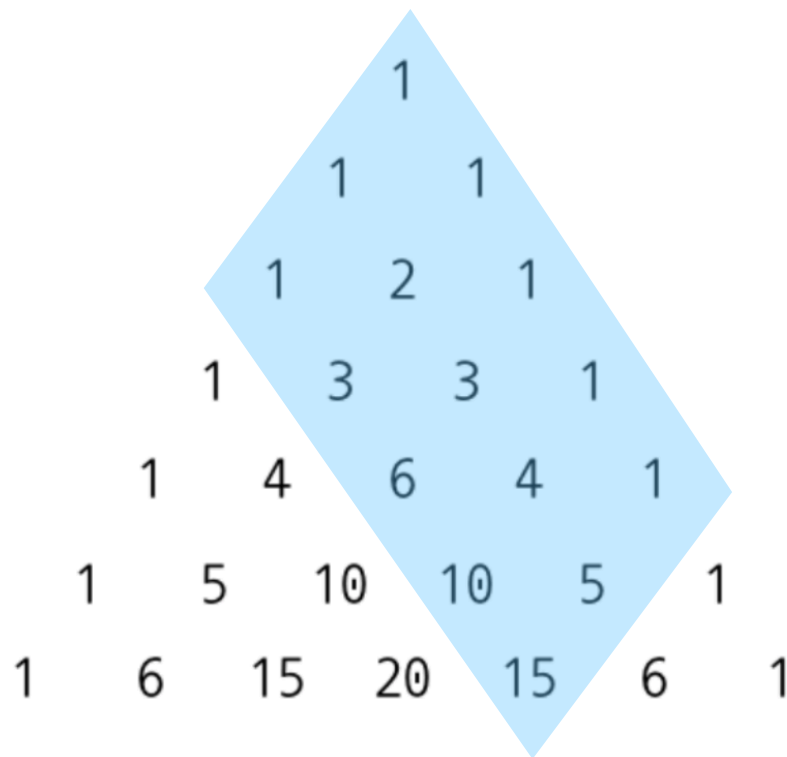
파스칼의 삼각형표채워풀기 알고리즘

- 프랑스 수학자 블레즈 파스칼(Blaise Pascal, 1623~1662)이 고안한 삼각형

[illegible]

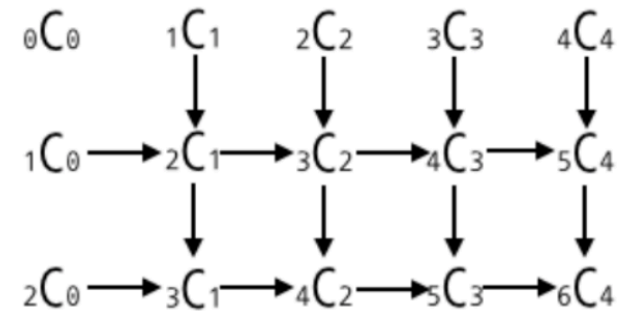
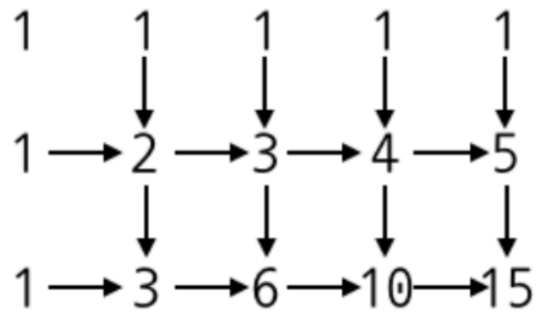
파스칼의 삼각형 표채워풀기 알고리즘

- 6C_4 를 계산하기 위한 영역



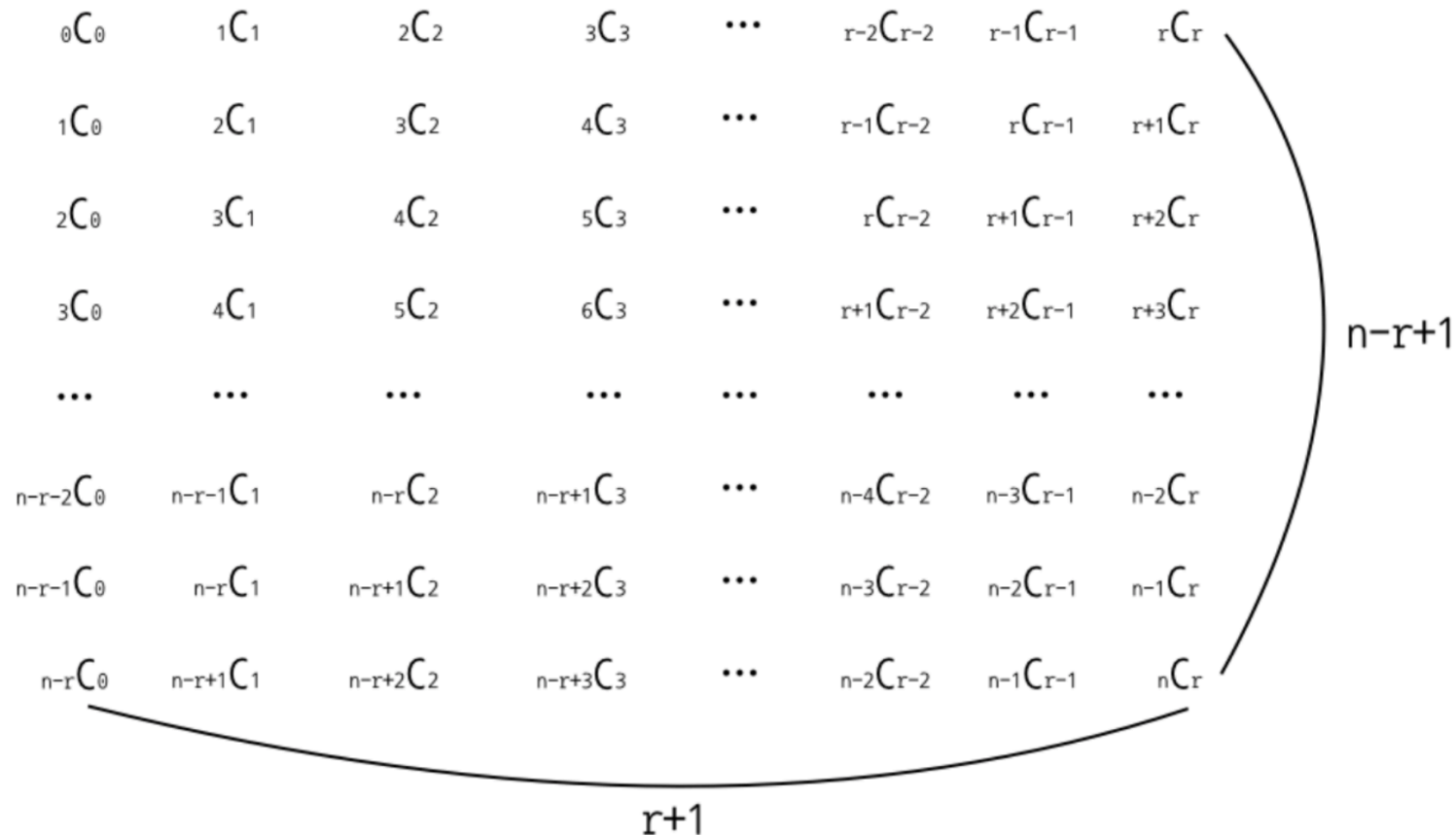
파스칼의 삼각형 표채워풀기 알고리즘

- 6C_4 를 계산하기 위한 영역



파스칼의 삼각형 표채워풀기 알고리즘

- nC_r 를 계산하기 위한 영역



파스칼의 삼각형표채워풀기 알고리즘

- 행렬의 표현

 ${}_6C_4$

1	1	1	1	1
1	2	3	4	5
1	3	6	10	15

```
[[1, 1, 1, 1, 1],
 [1, 2, 3, 4, 5],
 [1, 3, 6, 10, 15]]
```

```
[[1, 1, 1, 1, 1], [1], [1]]
⇒ [[1, 1, 1, 1, 1], [1, 2], [1]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3], [1]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3, 4], [1]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 3]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 3, 6]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 3, 6, 10]]
⇒ [[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 3, 6, 10, 15]]
```

파스칼의 삼각형 표채워풀기 알고리즘

- 행렬의 표현

 6C_4

1	1	1	1	1
1	2	3	4	5
1	3	6	10	15

```
[[1, 1, 1, 1, 1],
 [1, 2, 3, 4, 5],
 [1, 3, 6, 10, 15]]
```

- [\[Python 인터프리터\]](#)

- `matrix = [[1,2,3,4], [5,6,7,8], [9,10,11,12]]`
- `matrix[1][2]`
- `matrix[1][2] = -7`
- `matrix[1] = [55,66,77]`
- `matrix[0] = 1234`

파스칼의 삼각형 표채워풀기 알고리즘

- 행렬의 표현

 nC_r

matrix = [[1, 1, 1, 1, ..., 1, 1, 1],
 [1],
 [1],
 ...,
 [1]]

r+1

n-r+1

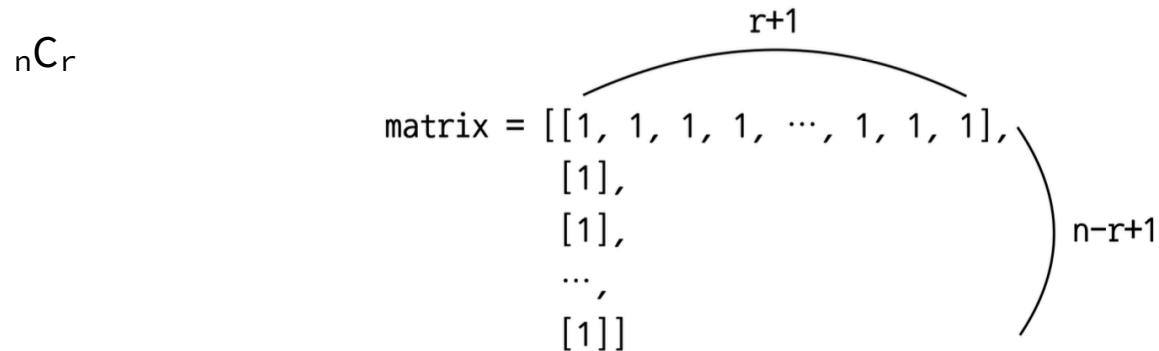
- 리스트 축약list comprehension

- [[Python 인터프리터](#)]

- `[i for i in range(10)]`
- `[i for i in range(10) if i % 2 == 0]`
- `row0 = [1 for _ in range(r+1)]`
- `[[1] for _ in range(r+1)]`
- `matrix = [row0] + [[1] for _ in range(n-r)]`

파스칼의 삼각형 표채워풀기 알고리즘

- 행렬의 표현



```

1 def comb_pascal(n, r):
2     row0 = [1 for _ in range(r+1)]
3     matrix = [row0] + [[1] for _ in range(n-r)]
4     for i in range(1, n - r + 1):
5         for j in range(1, r + 1):
6             newvalue = matrix[i][j-1] + matrix[i-1][j]
7             matrix[i].append(newvalue)
8     return matrix[n-r][r]
```

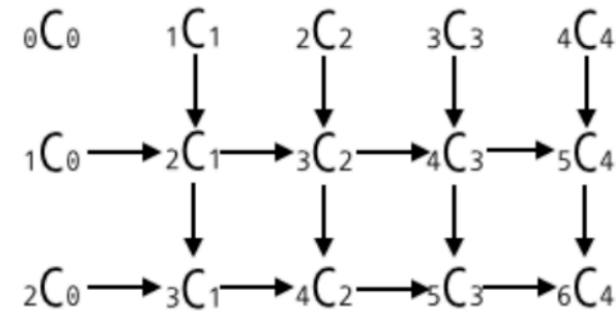
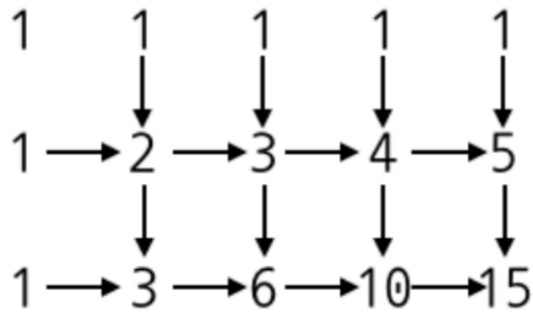

파스칼의 삼각형 표채워풀기 알고리즘

- 공간 절약

```

1  def comb_pascal(n, r):
2      vector = [1 for _ in range(r+1)]
3      for _ in range(1, n - r + 1):
4          for j in range(1, r + 1):
5              vector[j] = matrix[j-1] + matrix[j]
6      return vector[r]

```



1까지 줄이는 최소 스텝

1까지 줄이는 최소 스텝

- 세가지 스텝 중 하나를 선택해서 양수 n 을 1까지 줄이기 (minimum step to one)
 - 스텝 (1) : 1을 뺀다.
 - 스텝 (2) : 2로 나누어지면, 2로 나눈다.
 - 스텝 (3) : 3으로 나누어지면, 3으로 나눈다.
- 제약사항
 - 스텝 (1)은 1보다 큰 모든 양수에 적용할 수 있다.
 - 스텝 (2)는 2의 배수, 스텝 (3)은 3의 배수에만 적용할 수 있다.
- 목표
 - 임의의 양수 n 을 1까지 줄여나가는 가장 짧은 경로의 길이를 구하시오.
- 예시 ($n = 10$)

② ① ② ②
 $10 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 1$

① ③ ③
 $10 \rightarrow 9 \rightarrow 3 \rightarrow 1$

① ① ② ② ②
 $10 \rightarrow 9 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

1까지 줄이는 최소 스텝 :: 재귀

- 세가지 스텝 중 하나를 선택해서 양수 n 을 1까지 줄이기 (minimum step to one)
 - 스텝 (1) : 1을 뺀다.
 - 스텝 (2) : 2로 나누어지면, 2로 나눈다. (n 이 2의 배수인 경우)
 - 스텝 (3) : 3으로 나누어지면, 3으로 나눈다. (n 이 3의 배수인 경우)

$$\text{minsteps}(n) = \begin{cases} 1 + \text{minimum}\{\text{minsteps}(n-1), \text{minsteps}(n/2), \text{minsteps}(n/3)\} & \text{if } n > 1 \\ 0 & \text{if } n = 1 \end{cases}$$

```

1  def minsteps(n):
2      if n > 1:
3          steps = minsteps(n-1)
4          if n % 2 == 0:
5              steps = min(steps, minsteps(n // 2))
6          if n % 3 == 0:
7              steps = min(steps, minsteps(n // 3))
8          return 1 + steps
9      else:
10         return 0

```

```

1  def run_minsteps(n):
2      from time import perf_counter
3      start = perf_counter()
4      answer = minsteps(n)
5      finish = perf_counter()
6      print("minsteps(", n, ") ⇒ ", answer, sep="")
7      print(round(finish - start), "seconds")

```

1까지 줄이는 최소 스텝 :: 탐욕

- 탐욕 알고리즘 greedy algorithm
 - 재귀 : 가능한 스텝을 모두 다 시도해보고 최소 스텝을 결정
 - 탐욕 : 당장 가장 좋아보이는 스텝을 바로 결정
 - 우선순위 : $3 \rightarrow 2 \rightarrow 1$
 - 대체로 잘 작동하지만 그렇지 않은 경우도 있음

```
1 def greedy_minsteps(n):
2     if n > 1:
3         if n % 3 == 0:
4             return 1 + greedy_minsteps(n // 3)
5         elif n % 2 == 0:
6             return 1 + greedy_minsteps(n // 2)
7         else:
8             return greedy_minsteps(n-1)
9     else:
10        return 0
```

```
1 def run_minsteps(n):
2     from time import perf_counter
3     start = perf_counter()
4     answer = greedy_minsteps(n)
5     finish = perf_counter()
6     print("minsteps(", n, ") ⇒ ", answer, sep="")
7     print(round(finish - start), "seconds")
```

1까지 줄이는 최소 스텝 :: 탐욕

- 세가지 스텝 중 하나를 선택해서 양수 n 을 1까지 줄이기 (minimum step to one)
 - 스텝 (1) : 1을 뺀다.
 - 스텝 (2) : 2로 나누어지면, 2로 나눈다. (n 이 2의 배수인 경우)
 - 스텝 (3) : 3으로 나누어지면, 3으로 나눈다. (n 이 3의 배수인 경우)

탐욕의 우선순위 : 3 \rightarrow 2 \rightarrow 1

함수 호출	답	줄이기 절차
minsteps(3)	1	• 3 \rightarrow 1 [탐욕] [정답]
minsteps(4)	2	• 4 \rightarrow 2 \rightarrow 1 [탐욕] [정답]
minsteps(7)	3	• 7 \rightarrow 6 \rightarrow 2 \rightarrow 1 [탐욕] [정답]
minsteps(10)	3	• 10 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 1 [탐욕] • 10 \rightarrow 9 \rightarrow 3 \rightarrow 1 [정답]
minsteps(23)	6	• 23 \rightarrow 22 \rightarrow 11 \rightarrow 10 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 1 [탐욕] • 23 \rightarrow 22 \rightarrow 21 \rightarrow 7 \rightarrow 6 \rightarrow 2 \rightarrow 1 [정답]
minsteps(237)	8	• 237 \rightarrow 79 \rightarrow 78 \rightarrow 26 \rightarrow 13 \rightarrow 12 \rightarrow 4 \rightarrow 2 \rightarrow 1 [탐욕] [정답]
minsteps(317)	10	• 317 \rightarrow 316 \rightarrow 158 \rightarrow 79 \rightarrow 78 \rightarrow 39 \rightarrow 13 \rightarrow 12 \rightarrow 4 \rightarrow 2 \rightarrow 1 [탐욕] [정답]
minsteps(514)	8	• 514 \rightarrow 257 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 [탐욕] • 514 \rightarrow 513 \rightarrow 171 \rightarrow 57 \rightarrow 19 \rightarrow 18 \rightarrow 6 \rightarrow 2 \rightarrow 1 [정답]

1까지 줄이는 최소 스텝 :: 메모

- 메모해두기 알고리즘 memoization algorithm
 - 1부터 n-1까지 계산결과를 저장해두고 이를 활용

```

1  def memo_minsteps(n):
2      memo = [0 for _ in range(n+1)] # n + 1개?, 가독성!
3      def loop(n):
4          if n > 1:
5              if memo[n] == 0:
6                  steps = loop(n-1)
7                  if n % 2 == 0:
8                      steps = min(steps, loop(n // 2))
9                  if n % 3 == 0:
10                     steps = min(steps, loop(n // 3))
11                     memo[n] = steps + 1
12             return memo[n]
13         else:
14             return 0
15     return loop(n)

```

```

1  def run_minsteps(n):
2      from time import perf_counter
3      start = perf_counter()
4      answer = memo_minsteps(n)
5      finish = perf_counter()
6      print("minsteps(", n, ") => ", answer, sep="")
7      print(round(finish - start), "seconds")

```

run_minsteps(1022) ?

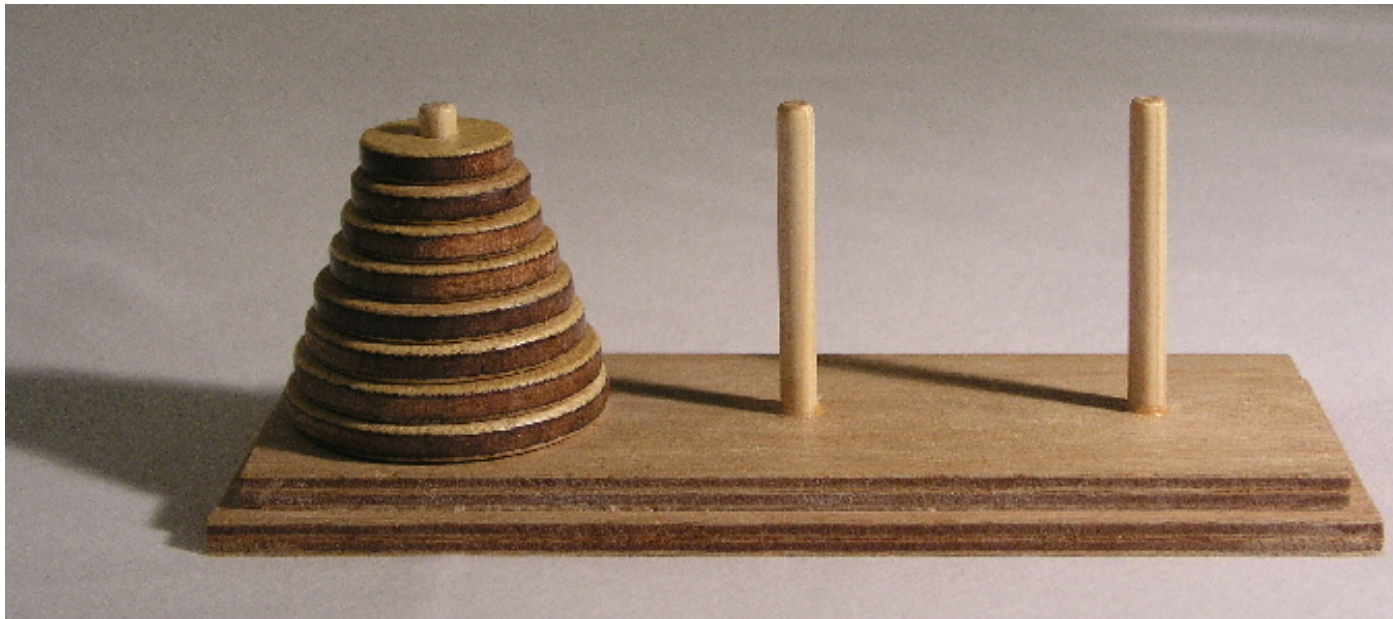
표채워풀기 알고리즘 구현 : 실습#7-3

하노이의 탑

하노이의 탑

- [피보나치 수열], [조합], [1까지 줄이는 최소 스텝]은 효율적으로 풀 수 있는 동적계획 알고리즘이 존재하는 '트리 재귀 문제'
- 동적계획 알고리즘이 없는(정확히는 아직 찾지를 못한) 트리 재귀 알고리즘 문제 많음
 - 대표주자 : 하노이의 탑 Tower of Hanoi

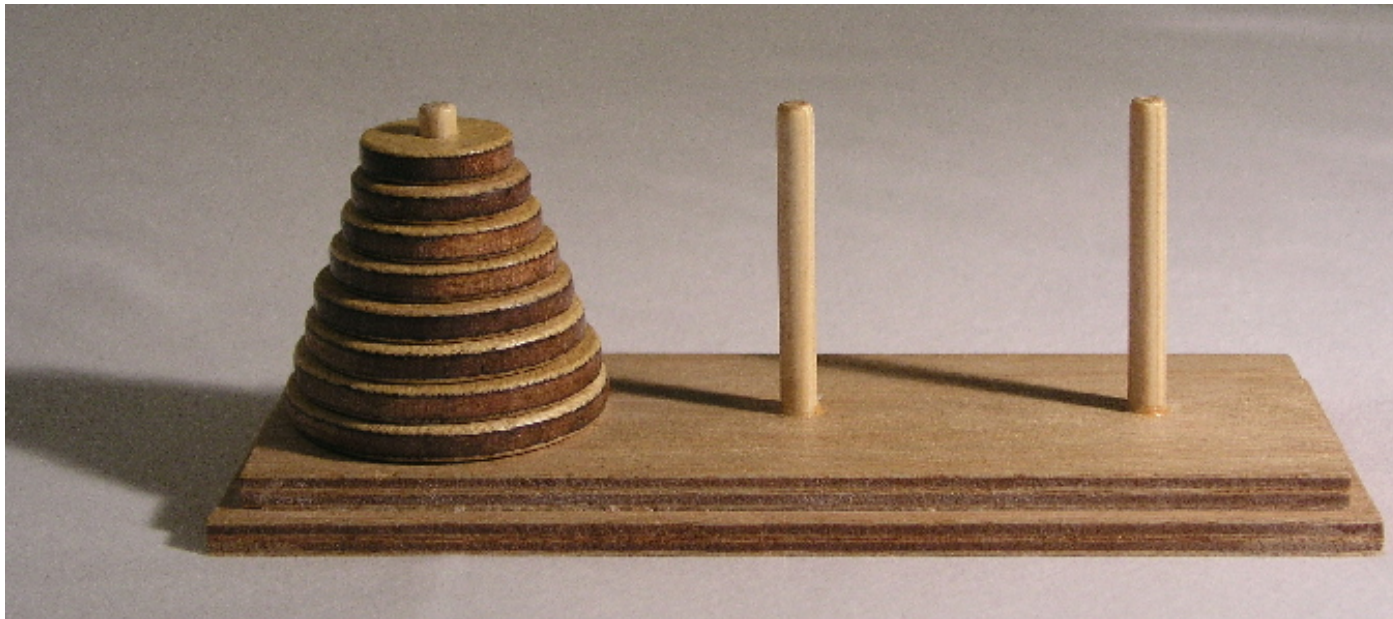
하노이의 탑



- 한 말뚝에 쌓여 있는 원반 n 개를 모두 다른 말뚝으로 옮겨 쌓는 문제
- 한번에 하나의 원반만 옮길 수 있음
- 큰 원반이 작은 원반의 위에 올라갈 수 없음
- 원반은 항상 말뚝에 꽂혀 있어야 함 (옮기는 과정을 제외하고, 말뚝 이외의 다른 곳에 둘 수 없음)

그림 출처 : https://en.wikipedia.org/wiki/Tower_of_Hanoi

하노이의 탑



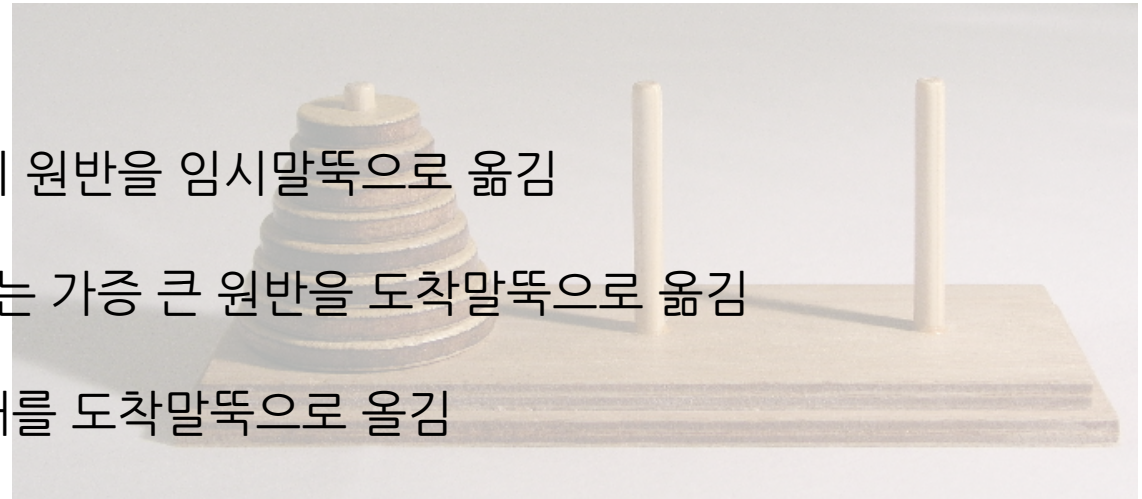
- 하노이의 탑: 재귀 알고리즘
 - [재귀] 출발말뚝 상위 $n-1$ 개의 원반을 임시말뚝으로 옮김
 - 출발말뚝의 맨 아래에 남아 있는 가장 큰 원반을 도착말뚝으로 옮김
 - [재귀] 임시말뚝의 원반 $n-1$ 개를 도착말뚝으로 옮김

그림 출처 : https://en.wikipedia.org/wiki/Tower_of_Hanoi

하노이의 탑

- 하노이의 탑: 재귀 알고리즘

- [재귀] 출발말뚝 상위 $n-1$ 개의 원반을 임시말뚝으로 옮김
- 출발말뚝의 맨 아래에 남아 있는 가장 큰 원반을 도착말뚝으로 옮김
- [재귀] 임시말뚝의 원반 $n-1$ 개를 도착말뚝으로 옮김



```
1  def tower_of_hanoi(n, source, dest, temp):
2      if n > 1:
3          tower_of_hanoi(n-1, source, temp, dest)
4          print(source, "에서", dest, "로 원반 하나를 이동")
5          tower_of_hanoi(n-1, temp, dest, source)
6      else:
7          print(source, "에서", dest, "로 원반 하나를 이동")
```

그림 출처 : https://en.wikipedia.org/wiki/Tower_of_Hanoi

하노이의 탑

전역변수 global variable를 사용해서 원반 이동 횟수 세기

```
1  def tower_of_hanoi(n, source, dest, temp):
2      global count # 전역 변수 (global variable)
3      if n > 1:
4          tower_of_hanoi(n-1, source, temp, dest)
5          count += 1
6          tower_of_hanoi(n-1, temp, dest, source)
7      else:
8          count += 1
9
10 for n in [4,6,8,16,24,25,26,27,28]:
11     count = 0
12     from time import perf_counter
13     start = perf_counter()
14     tower_of_hanoi(n, "A", "C", "B")
15     finish = perf_counter()
16     print("원반", n, "개 :", count, "번 이동,", round(finish-start, 1), "초")
```