[CSE1017]

프로그래밍기초

Dynamic Programming

#07. 표채워풀기

김현하

한양대학교 ERICA 소프트웨어학부

2024년도 1학기

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

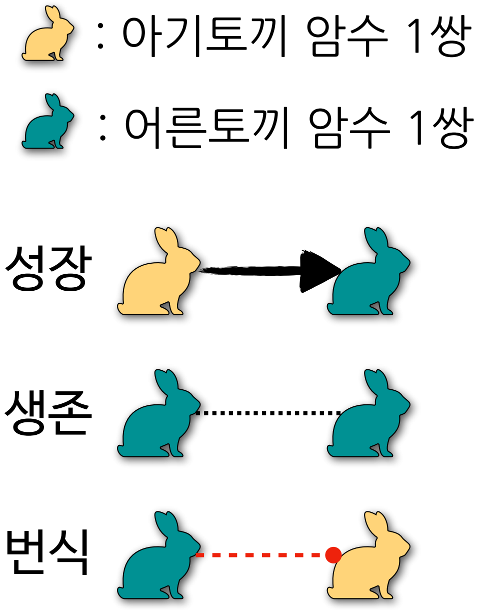
피보나치 수열

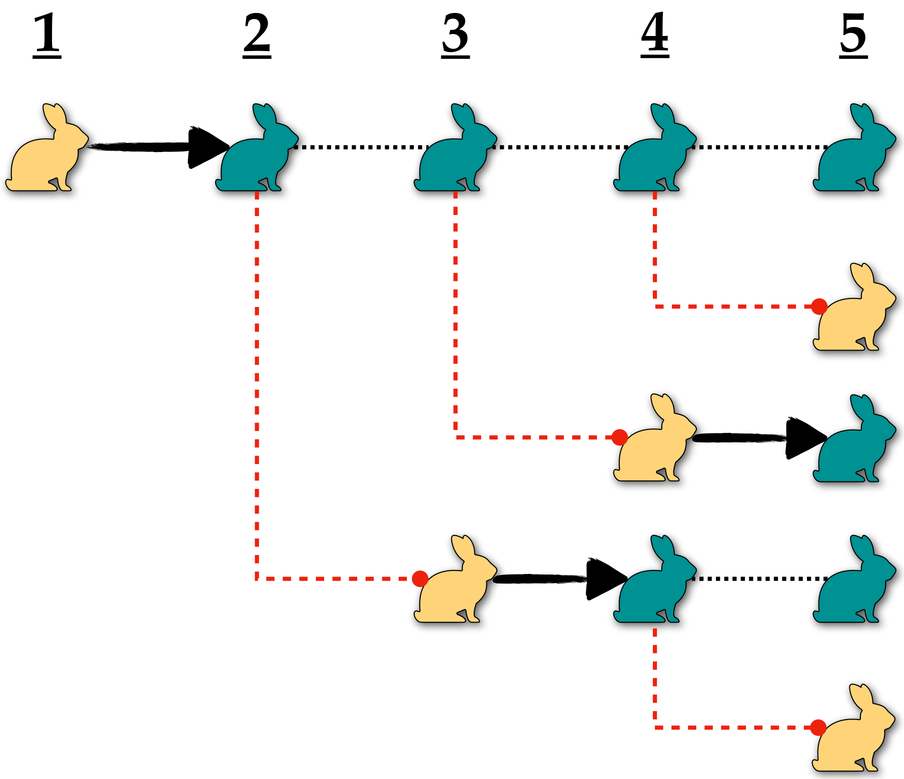
피보나치 수열

- 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 1쌍을 낳는다
  - 아기 토끼는 태어난지 1달이 지나면 어른 토끼가 되어 번식이 가능해진다
  - 토끼는 늙지도 않고 죽지도 않는다

그림출처 : https://ko.wikipedia.org/wiki/레오나르도_피보나치

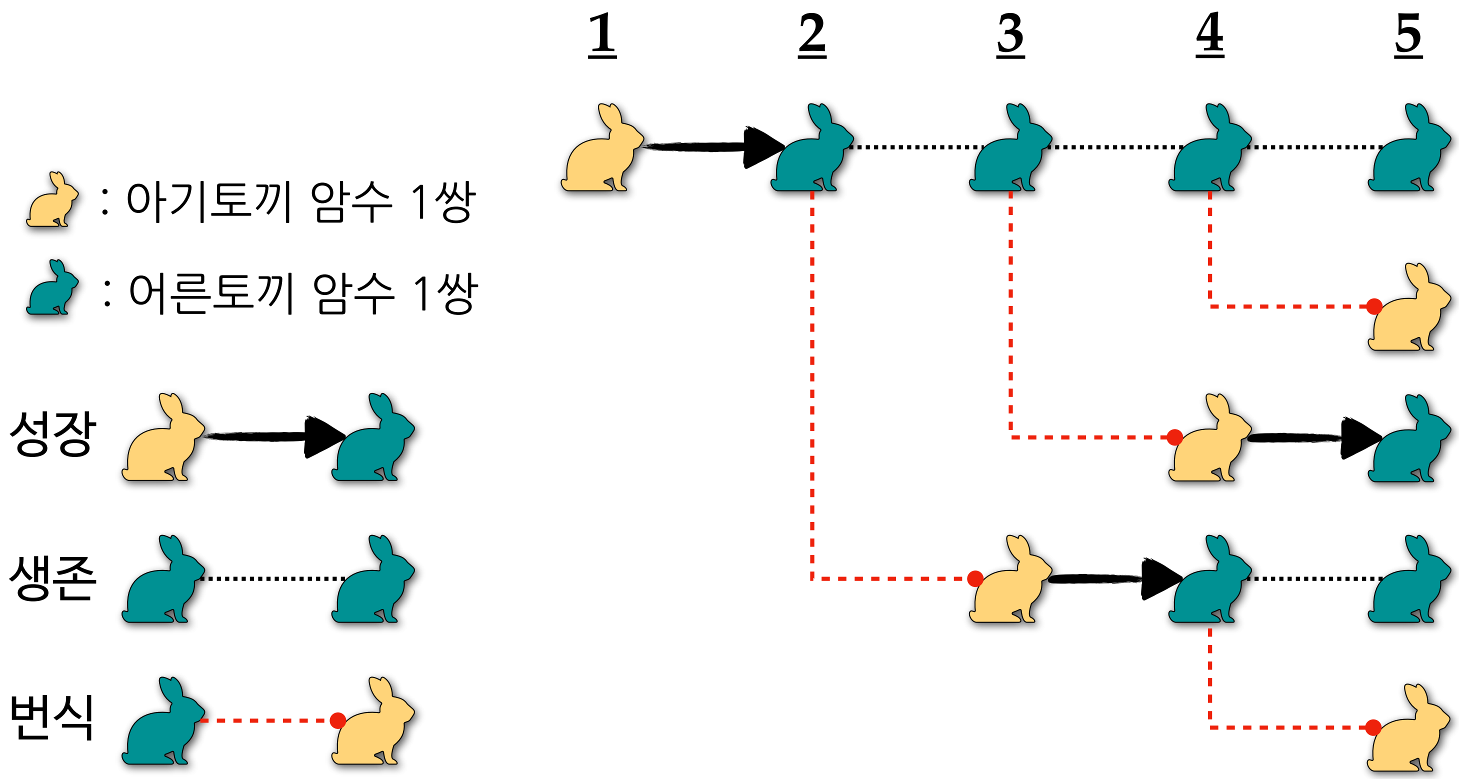
피보나치 수열





달 (month)	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 (sum)	0	1	1	2	3	5	8	13	21	34	55	89	144	...

피보나치 수열



달 (month)	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 (sum)	0	1	1	2	3	5	8	13	21	34	55	89	144	...

피보나치 수열

- 5년(60개월) 후에 토끼 가족의 규모는?

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

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

이달의 어른 토끼 쌍의 수 지난달의 전체 토끼 쌍의 수

생존 지난달의 어른 토끼 쌍의 수 $\texttt{rabby}_{i-1}$

성장 (이번달에 어른이 된) 지난달의 아이 토끼 쌍의 수 $\texttt{bunny}_{i-1}$

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

이달의 아이 토끼 쌍의 수 지지난달의 전체 토끼 쌍의 수

번식 지난달의 어른 토끼 쌍의 수 $\texttt{rabby}_{i-1}$

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

피보나치 수열의 재귀 알고리즘

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

```python
def fib(n):
    if n > 1:
        return fib(n - 1) + fib(n - 2)
    else:
        return n
```

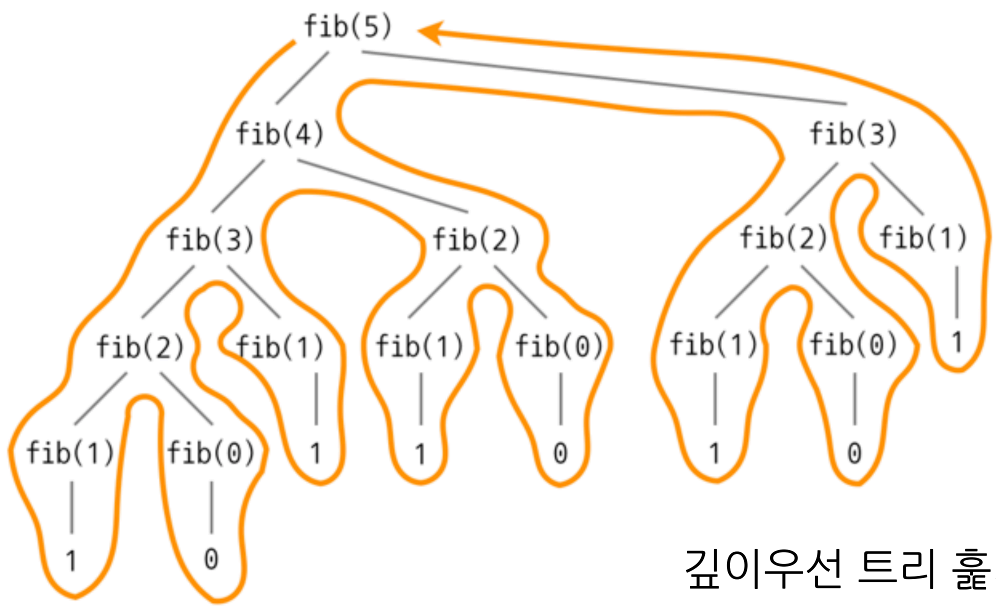
피보나치 수열의 재귀 알고리즘

```python
def fib(n):
    if n > 1:
        return fib(n - 1) + fib(n - 2)
    else:
        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

```python
def run_fib(n):
    from time import perf_counter
    start = perf_counter()
    answer = fib(n)
    finish = perf_counter()
    print("fib(", n, ") => ", answer, sep="")
    print(round(finish - start, 4), "seconds")
```



깊이우선 트리 훑기depth-first tree traversal

피보나치 수열의 표채워풀기 알고리즘

- 재귀 호출하는 대신, 목표로 하는 fib(n) 까지 필요한 fib(n-1), fib(n-2)를 계산해 올라감

```
fib(0) = 0
fib(1) = 1
fib(2) = fib(1) + fib(0) = 1 + 0 = 1
fib(3) = fib(2) + fib(1) = 1 + 1 = 2
fib(4) = fib(3) + fib(2) = 2 + 1 = 3
fib(5) = fib(4) + fib(3) = 3 + 2 = 5
fib(6) = fib(5) + fib(4) = 5 + 3 = 8
fib(7) = fib(6) + fib(5) = 8 + 5 = 13
fib(8) = fib(7) + fib(6) = 13 + 8 = 21
fib(9) = fib(8) + fib(7) = 21 + 13 = 34
fib(10) = fib(9) + 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 알고리즘while 루프ver.

- 재귀 호출하는 대신, 목표로 하는 `fib(n)` 까지 필요한 `fib(n-1)`, `fib(n-2)`를 계산해 올라감

```python
def fib(n):
    if n > 1:
        i = 2
        a, b = 0, 1
        while i <= n:
            a, b = b, a + b
            i += 1
        return b
    else:
        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 알고리즘for 루프ver.

- 재귀 호출하는 대신, 목표로 하는 fib(n) 까지 필요한 fib(n-1), fib(n-2)를 계산해 올라감

```python
def fib(n):
    if n > 1:
        a, b = 0, 1
        for _ in range(2, n+1):
            a, b = b, a + b
        return b
    else:
        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

```python
def fib(n):
    if n > 1:
        a, b = 0, 1
        for _ in range(2, n+1):
            a, b = b, a + b
        return b
    else:
        return n
```

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

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

조합

조합

- 조합combination $_nC_r$
  - n개의 자연수에서 순서에 상관없이 r개를 뽑는 가지 수

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

조합의 재귀 알고리즘

- 조합combination $_nC_r$
  - n개의 자연수에서 순서에 상관없이 r개를 뽑는 가지 수

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

```python
def comb(n, r):
    if r != 0 and r != n:
        return comb(n-1, r-1) + comb(n-1, r)
    else:
        return 1
```

time.perf_counter 로 시간 측정해보기

파스칼의 삼각형표채워풀기 알고리즘

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

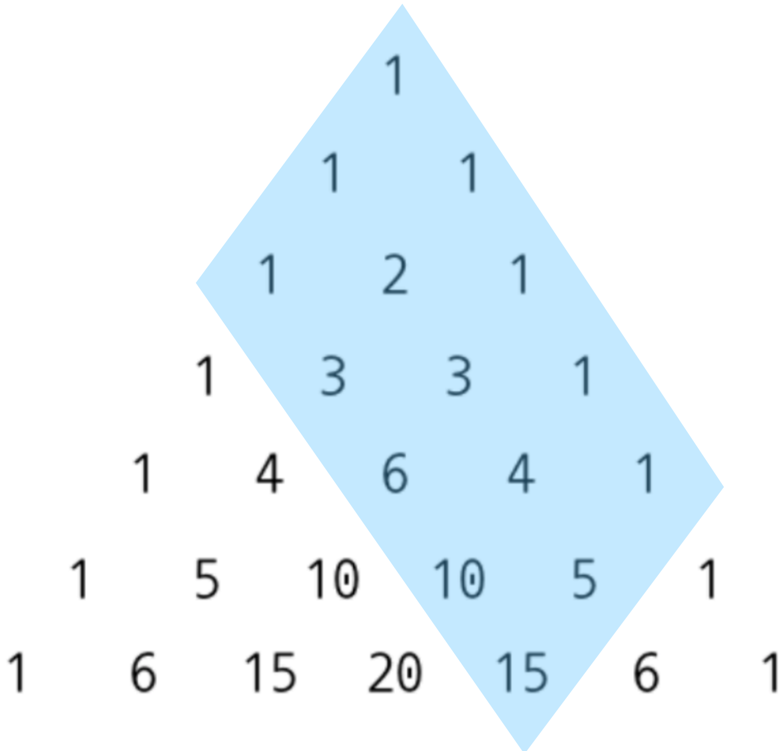
						1						
					1		1					
				1		2		1				
			1		3		3		1			
		1		4		6		4		1		
	1		5		10		10		5		1	
1		6		15		20		15		6		1

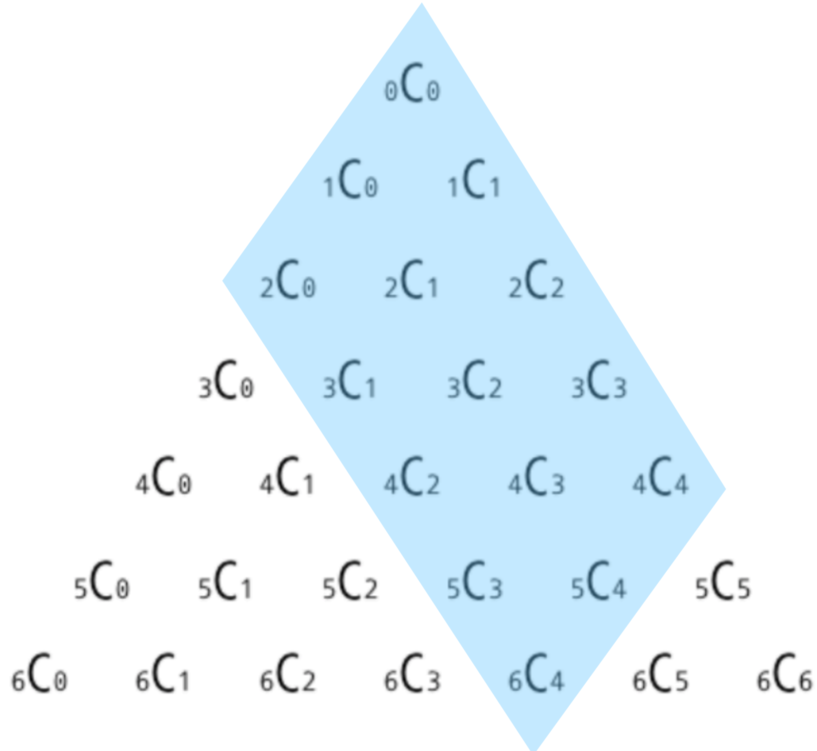
$$
\begin{array}{ccccccccccccc}
 & & & & & & {}_0C_0 & & & & & & \\
 & & & & & {}_1C_0 & & {}_1C_1 & & & & & \\
 & & & & {}_2C_0 & & {}_2C_1 & & {}_2C_2 & & & & \\
 & & & {}_3C_0 & & {}_3C_1 & & {}_3C_2 & & {}_3C_3 & & & \\
 & & {}_4C_0 & & {}_4C_1 & & {}_4C_2 & & {}_4C_3 & & {}_4C_4 & & \\
 & {}_5C_0 & & {}_5C_1 & & {}_5C_2 & & {}_5C_3 & & {}_5C_4 & & {}_5C_5 & \\
{}_6C_0 & & {}_6C_1 & & {}_6C_2 & & {}_6C_3 & & {}_6C_4 & & {}_6C_5 & & {}_6C_6
\end{array}
$$

그림출처 : https://ko.wikipedia.org/wiki/블레즈_파스칼

파스칼의 삼각형 표채워풀기 알고리즘

- ${}_6C_4$ 를 계산하기 위한 영역





파스칼의 삼각형 표채워풀기 알고리즘

- ${}_{6}C_{4}$ 를 계산하기 위한 영역

$$\begin{tikzcd}
1 & 1 \arrow[d] & 1 \arrow[d] & 1 \arrow[d] & 1 \arrow[d] \\
1 \arrow[r] & 2 \arrow[r] \arrow[d] & 3 \arrow[r] \arrow[d] & 4 \arrow[r] \arrow[d] & 5 \arrow[d] \\
1 \arrow[r] & 3 \arrow[r] & 6 \arrow[r] & 10 \arrow[r] & 15
\end{tikzcd}$$

$$\begin{tikzcd}
{}_{0}C_{0} & {}_{1}C_{1} \arrow[d] & {}_{2}C_{2} \arrow[d] & {}_{3}C_{3} \arrow[d] & {}_{4}C_{4} \arrow[d] \\
{}_{1}C_{0} \arrow[r] & {}_{2}C_{1} \arrow[r] \arrow[d] & {}_{3}C_{2} \arrow[r] \arrow[d] & {}_{4}C_{3} \arrow[r] \arrow[d] & {}_{5}C_{4} \arrow[d] \\
{}_{2}C_{0} \arrow[r] & {}_{3}C_{1} \arrow[r] & {}_{4}C_{2} \arrow[r] & {}_{5}C_{3} \arrow[r] & {}_{6}C_{4}
\end{tikzcd}$$

파스칼의 삼각형표채워풀기 알고리즘

- ${}_{n}C_{r}$ 를 계산하기 위한 영역

${}_{0}C_{0}$	${}_{1}C_{1}$	${}_{2}C_{2}$	${}_{3}C_{3}$	⋯	${}_{r-2}C_{r-2}$	${}_{r-1}C_{r-1}$	${}_{r}C_{r}$
${}_{1}C_{0}$	${}_{2}C_{1}$	${}_{3}C_{2}$	${}_{4}C_{3}$	⋯	${}_{r-1}C_{r-2}$	${}_{r}C_{r-1}$	${}_{r+1}C_{r}$
${}_{2}C_{0}$	${}_{3}C_{1}$	${}_{4}C_{2}$	${}_{5}C_{3}$	⋯	${}_{r}C_{r-2}$	${}_{r+1}C_{r-1}$	${}_{r+2}C_{r}$
${}_{3}C_{0}$	${}_{4}C_{1}$	${}_{5}C_{2}$	${}_{6}C_{3}$	⋯	${}_{r+1}C_{r-2}$	${}_{r+2}C_{r-1}$	${}_{r+3}C_{r}$
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯
${}_{n-r-2}C_{0}$	${}_{n-r-1}C_{1}$	${}_{n-r}C_{2}$	${}_{n-r+1}C_{3}$	⋯	${}_{n-4}C_{r-2}$	${}_{n-3}C_{r-1}$	${}_{n-2}C_{r}$
${}_{n-r-1}C_{0}$	${}_{n-r}C_{1}$	${}_{n-r+1}C_{2}$	${}_{n-r+2}C_{3}$	⋯	${}_{n-3}C_{r-2}$	${}_{n-2}C_{r-1}$	${}_{n-1}C_{r}$
${}_{n-r}C_{0}$	${}_{n-r+1}C_{1}$	${}_{n-r+2}C_{2}$	${}_{n-r+3}C_{3}$	⋯	${}_{n-2}C_{r-2}$	${}_{n-1}C_{r-1}$	${}_{n}C_{r}$

n−r+1

r+1

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]]
```

```
>>> 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 삼각형표채워풀기 알고리즘

- 행렬의 표현

$_nC_r$

```
                     r+1
matrix = [[1, 1, 1, 1, …, 1, 1, 1],
          [1],
          [1],                          n-r+1
          …,
          [1]]
```

- 리스트 축약list comprehension

```
>>> [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

$_nC_r$

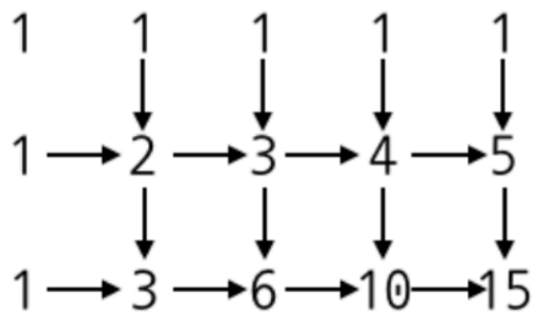
```
                 r+1
matrix = [[1, 1, 1, 1, ⋯, 1, 1, 1],
          [1],
          [1],                          n-r+1
          ⋯,
          [1]]
```

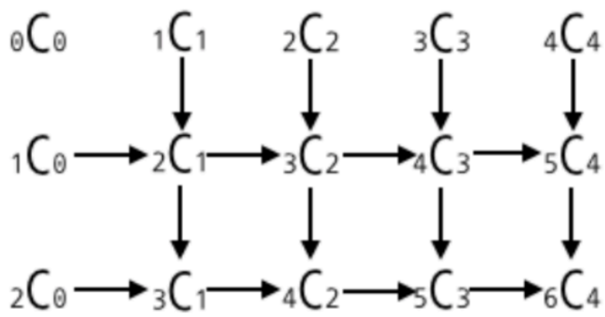
```python
def comb_pascal(n, r):
    row0 = [1 for _ in range(r+1)]
    matrix = [row0] + [[1] for _ in range(n-r)]
    for i in range(1, n - r + 1):
        for j in range(1, r + 1):
            newvalue = matrix[i][j-1] + matrix[i-1][j]
            matrix[i].append(newvalue)
    return matrix[n-r][r]
```

파스칼의 삼각형 표채워풀기 알고리즘

- 공간 절약

```python
def comb_pascal(n, r):
    vector = [1 for _ in range(r+1)]
    for _ in range(1, n - r + 1):
        for j in range(1, r + 1):
            vector[j] = matrix[j-1] + matrix[j]
    return vector[r]
```





1까지 줄이는 최소 스텝

1까지 줄이는 최소 스텝

- 세가지 스텝 중 하나를 선택해서 양수 n을 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 n을 1까지 줄여나가는 가장 짧은 경로의 길이를 구하시오.
- 예시 (n = 10)

$10 \xrightarrow{\textcircled{2}} 5 \xrightarrow{\textcircled{1}} 4 \xrightarrow{\textcircled{2}} 2 \xrightarrow{\textcircled{2}} 1$

$10 \xrightarrow{\textcircled{1}} 9 \xrightarrow{\textcircled{3}} 3 \xrightarrow{\textcircled{3}} 1$

$10 \xrightarrow{\textcircled{1}} 9 \xrightarrow{\textcircled{1}} 8 \xrightarrow{\textcircled{2}} 4 \xrightarrow{\textcircled{2}} 2 \xrightarrow{\textcircled{2}} 1$

1까지 줄이는 최소 스텝 :: 재귀

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

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

```python
def minsteps(n):
    if n > 1:
        steps = minsteps(n-1)
        if n % 2 == 0:
            steps = min(steps, minsteps(n // 2))
        if n % 3 == 0:
            steps = min(steps, minsteps(n // 3))
        return 1 + steps
    else:
        return 0
```

```python
def run_minsteps(n):
    from time import perf_counter
    start = perf_counter()
    answer = minsteps(n)
    finish = perf_counter()
    print("minsteps(", n, ") => ", answer, sep="")
    print(round(finish - start), "seconds")
```

1까지 줄이는 최소 스텝 :: 탐욕

- 탐욕 알고리즘greedy algorithm
  - 재귀 : 가능한 스텝을 모두 다 시도해보고 최소 스텝을 결정
  - 탐욕 : 당장 가장 좋아보이는 스텝을 바로 결정
    - 우선순위 : 3 → 2 → 1
    - 대체로 잘 작동하지만 그렇지 않은 경우도 있음

```python
def greedy_minsteps(n):
    if n > 1:
        if n % 3 == 0:
            return 1 + greedy_minsteps(n // 3)
        elif n % 2 == 0:
            return 1 + greedy_minsteps(n // 2)
        else:
            return 1 + greedy_minsteps(n - 1)
    else:
        return 0
```

```python
def run_minsteps(n):
    from time import perf_counter
    start = perf_counter()
    answer = greedy_minsteps(n)
    finish = perf_counter()
    print("minsteps(", n, ") => ", answer, sep="")
    print(round(finish - start), "seconds")
```

1까지 줄이는 최소 스텝 :: 탐욕

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

탐욕의 우선순위 : 3 → 2 → 1

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

1까지 줄이는 최소 스텝 :: 메모

- 메모해두기 알고리즘memoization algorithm
    - 1부터 n-1까지 계산결과를 저장해두고 이를 활용

```python
def memo_minsteps(n):
    memo = [0 for _ in range(n+1)] # n + 1개?, 가독성!
    def loop(n):
        if n > 1:
            if memo[n] == 0:
                steps = loop(n-1)
                if n % 2 == 0:
                    steps = min(steps, loop(n // 2))
                if n % 3 == 0:
                    steps = min(steps, loop(n // 3))
                memo[n] = steps + 1
            return memo[n]
        else:
            return 0
    return loop(n)
```

```python
def run_minsteps(n):
    from time import perf_counter
    start = perf_counter()
    answer = memo_minsteps(n)
    finish = perf_counter()
    print("minsteps(", n, ") => ", answer, sep="")
    print(round(finish - start), "seconds")
```

run_minsteps(1022) ?

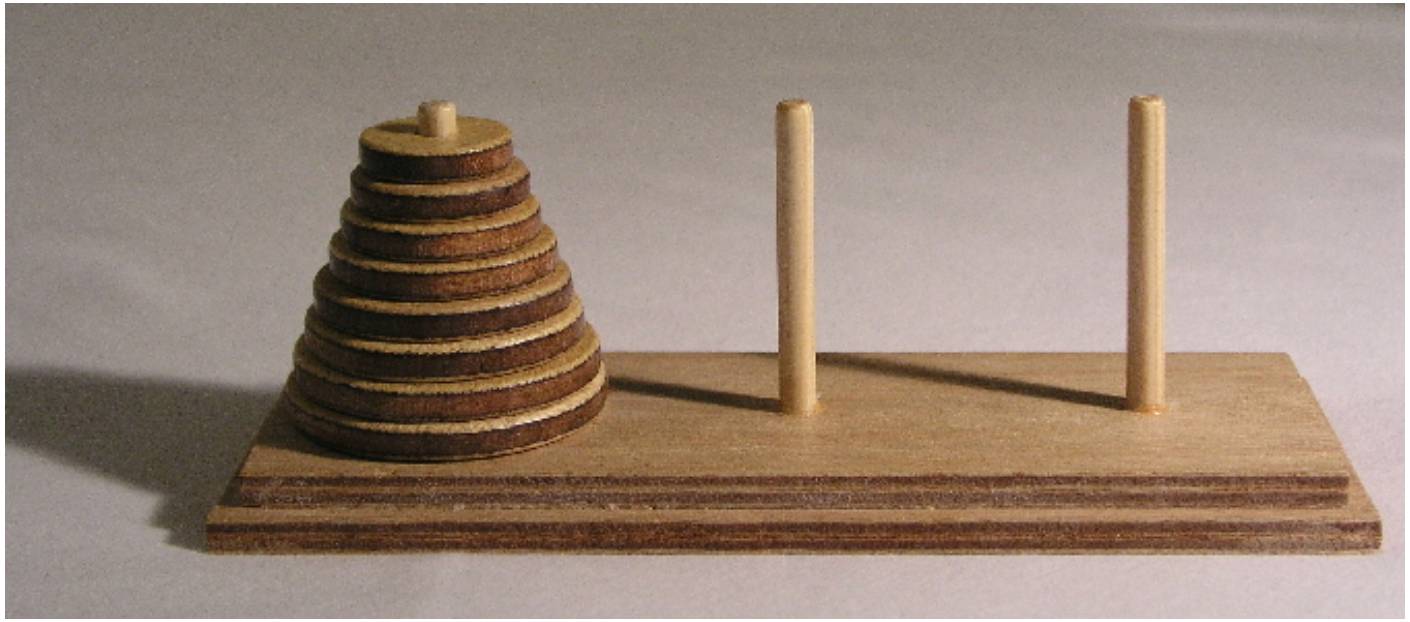
표채워풀기 알고리즘 구현 : 실습#7-3

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 문제많음
  - 대표주자 : 하노이의 탑

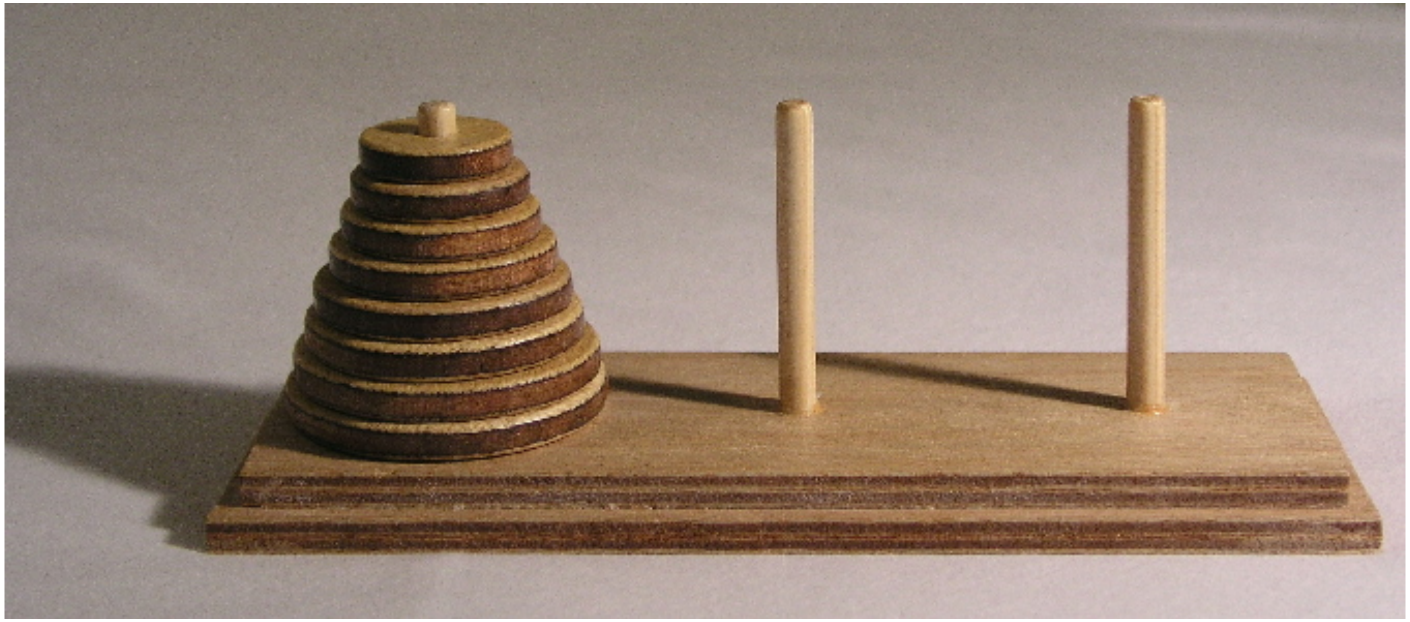
하노이의 탑



- 하노이의 탑Tower of Hanoi
  - 한 말뚝에 쌓여 있는 원반 n개를 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 탑Tower of Hanoi 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 올김

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

하노이의 탑

- 하노이의 탑Tower of Hanoi
  - [재귀] 출발말뚝 상위 n-1 개의 원반을 임시말뚝으로 옮김
  - 출발말뚝의 맨 아래에 남아 있는 가증 큰 원반을 도착말뚝으로 옮김
  - [재귀] 임시말뚝의 원반 n-1 개를 도착말뚝으로 올김

```python
def tower_of_hanoi(n, source, dest, temp):
    if n > 1:
        tower_of_hanoi(n-1, source, temp, dest)
        print(source, "에서", dest, "로 원반 하나를 이동")
        tower_of_hanoi(n-1, temp, dest, source)
    else:
        print(source, "에서", dest, "로 원반 하나를 이동")
```

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

하노이의 탑

전역변수global variable를 사용해서 원반 이동 횟수 세기

```python
def tower_of_hanoi(n, source, dest, temp):
    global count # 전역 변수 (global variable)
    if n > 1:
        tower_of_hanoi(n-1, source, temp, dest)
        count += 1
        tower_of_hanoi(n-1, temp, dest, source)
    else:
        count += 1

for n in [4,6,8,16,24,25,26,27,28]:
    count = 0
    from time import perf_counter
    start = perf_counter()
    tower_of_hanoi(n, "A", "C", "B")
    finish = perf_counter()
    print("원반", n, "개 :", count, "번 이동,", round(finish-start, 1), "초")
```