데이터 구조 입문

김종현

05

재귀 알고리즘

05-1 재귀 알고리즘의 기본

재귀 알아보기

- **재귀**recursion
  - 어떠한 이벤트에서 자기 자신을 포함하고 다시 자기 자신을 사용하여 정의되는 경우
  - 무한히 존재하는 자연수를 재귀적 정의recursive definition를 사용하여 두 문장으로 정의

    - 1은 자연수
    - 어떤 자연수의 바로 다음 수도 자연수

  - 재귀를 효과적으로 사용하면 이러한 정의 뿐만 아니라 프로그램을 간결하고 효율성 좋게 작성할 수 있음

05-1 재귀 알고리즘의 기본

팩토리얼 알아보기 – (1)

- **팩토리얼**factorial
  - 양의 정수의 곱을 구하는 문제
  - 양의 정수를 순서대로 곱한다는 의미로 순차 곱셈이라고도 함
  - 재귀를 사용하는 대표적인 예
  - 양의 정수 n의 팩토리얼(n!)의 재귀적 정의

    - 0! = 1
    - n > 0이면 n! = n × (n - 1)!

- **실습 5-1**
  - 양의 정수 n의 팩토리얼을 구하는 프로그램

Do it! 실습 5-1

• 완성 파일 chap05/factorial.py

```
01: # 양의 정수 n의 팩토리얼 구하기
02:
03: def factorial(n: int) -> int:
04:     """양의 정수 n의 팩토리얼값을 재귀적으로 구함"""
05:     if n > 0:
06:         return n * factorial(n - 1)
07:     else:
08:         return 1
09:
10: if __name__ == '__main__':
11:     n = int(input('출력할 팩토리얼값을 입력하세요.: '))
12:     print(f'{n}의 팩토리얼은 {factorial(n)}입니다.')
```

▶ 실행 결과

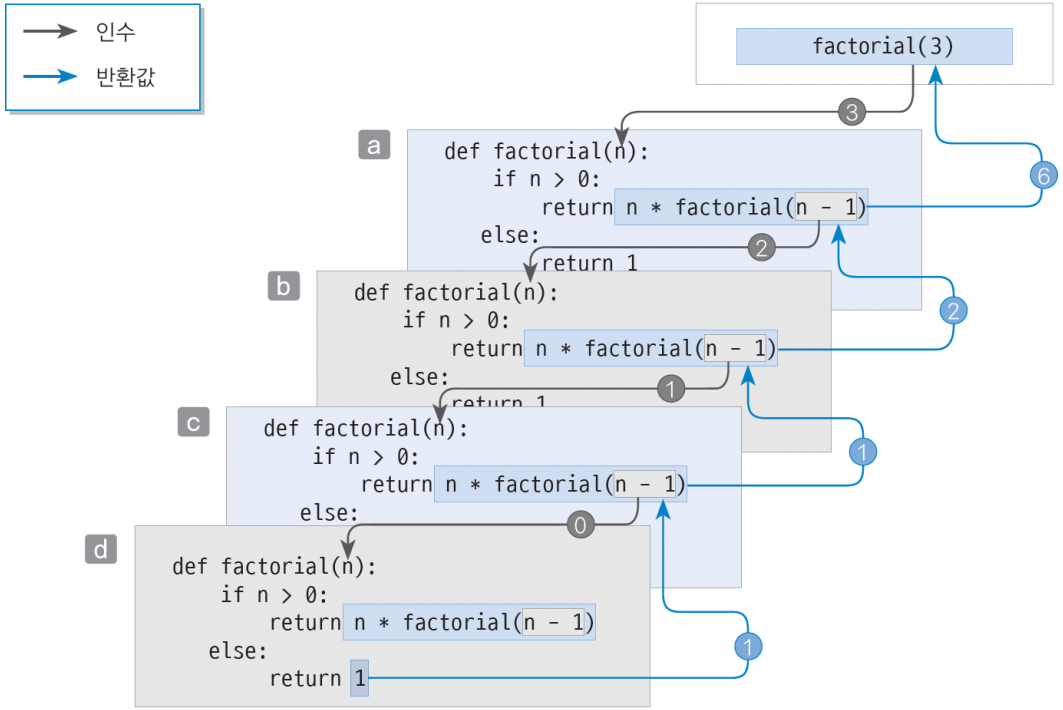
```
출력할 팩토리얼값을 입력하세요.: 3
3의 팩토리얼은 6입니다.
```

05-1 재귀 알고리즘의 기본

팩토리얼 알아보기 – (2)

- **재귀 호출recursive call**
  - 자기 자신과 똑같은 함수를 호출
    - a : factorial(3)을 실행하면 factorial( ) 함수가 호출됨
      - 매개변수 n에 3을 전달받아 3 * factorial(2) 값 반환
      - 값을 구하기 위해 인수로 2를 전달하여 factorial(2) 호출
    - b : 매개변수 n에 2를 전달받아
      2 * factorial(1)을 실행하기 위해 factorial(1) 호출
    - c : 매개변수 n에 1을 전달받아
      1 * factorial(0)을 실행하기 위해 함수 factorial(0) 호출
    - d : 매개변수 n에 0을 전달받아 1을 반환
      - 처음으로 return 문이 실행되고 반환값 1을 로 보냄
    - c : 반환된 값 1을 전달받아 1 * factorial(0), 즉 (1 * 1) 반환
    - b : 반환된 값 1을 전달받아 2 * factorial(1), 즉 (2 * 1) 반환
    - a : 반환된 값 2를 전달받아 3 * factorial(2), 즉 (3 * 2) 반환

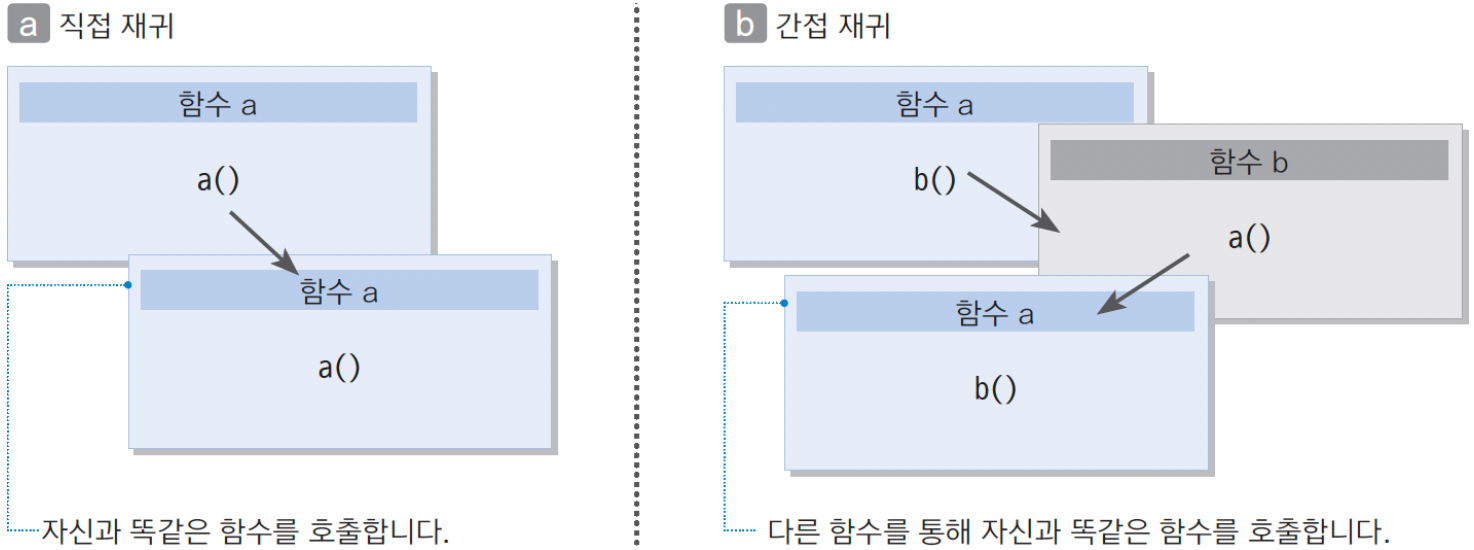
    ➔ 최종 3의 팩토리얼값인 6을 얻을 수 있음



05-1 재귀 알고리즘의 기본

팩토리얼 알아보기 – (3)

- **직접direct 재귀**
  - 자신과 똑같은 함수를 호출하는 방식
- **간접indirect 재귀**
  - a( ) 함수가 b( ) 함수를 호출하고 다시 b( ) 함수가 a( ) 함수를 호출하는 구조

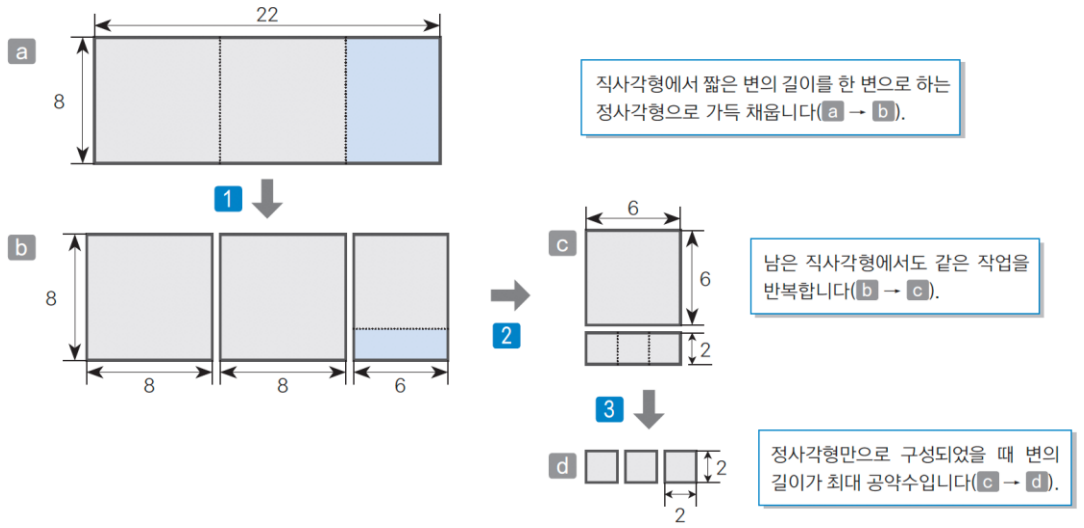


05-1 재귀 알고리즘의 기본

유클리드 호제법 알아보기 – (1)

- **두 정숫값의 최대 공약수GCD 재귀로 구하기**
  - 2개의 정숫값을 직사각형 두 변의 길이라고 할 때

Q. 직사각형 안을 정사각형 여러 개로 가득 채워 나간다. 이렇게 만들 수 있는 정사각형 가운데 가장 작은 정사각형의 변의 길이를 구하라.



  - 2개의 정숫값을 직사각형 두 변의 길이라고 할 때
    - 1 : 22 × 8 크기의 직사각형에서 짧은 변의 길이인 8을 한 변으로 하는 정사각형으로 나눔
      → 8 × 8 크기의 정사각형 2개가 만들어지고
      8 × 6 크기의 직사각형이 남음
    - 2 : 남은 8 × 6 크기의 직사각형에서도 같은 과정 수행
      → 6 × 6 크기의 정사각형 1개가 만들어지고
      6 × 2 크기의 직사각형이 남음
    - 3 : 남은 6 × 2 크기의 직사각형에서도 같은 과정 수행
      → 2 × 2 크기의 정사각형 3개가 만들어짐
    - ➔ 최종 정사각형 변의 길이인 2가 8과 22의 최대 공약수!

05-1 재귀 알고리즘의 기본

유클리드 호제법 알아보기 – (2)

- **유클리드 호제법**Euclidean agorithm
  - 두 정수 x와 y의 최대 공약수를 gcd(x, y)라 표기
  - x = az와 y = bz를 만족하는 정수 a, b와 최대의 정수 z가 존재할 때 z는 gcd(x, y)
  - 최대 공약수 구하기

    - y가 0이면 … x
    - y가 0이 아니면 … gcd(y, x % y)

- **실습 5-2**
  - 유클리드 호제법으로 두 정숫값의 최대 공약수를 구하는 프로그램

Do it! 실습 5-2

• 완성 파일 chap05/gcd.py

```
01: # 유클리드 호제법으로 최대 공약수 구하기
02:
03: def gcd(x: int, y: int) -> int:
04:     """정숫값 x와 y의 최대 공약수를 반환"""
05:     if y == 0:
06:         return x
07:     else:
08:         return gcd(y, x % y)
09:
10: if __name__ == '__main__':
11:     print('두 정숫값의 최대 공약수를 구합니다.')
12:     x = int(input('첫 번째 정숫값을 입력하세요.: '))
13:     y = int(input('두 번째 정숫값을 입력하세요.: '))
14:
15:     print(f'두 정숫값의 최대 공약수는 {gcd(x, y)}입니다.')
```

▶ 실행 결과

```
두 정숫값의 최대 공약수를 구합니다.
첫 번째 정숫값을 입력하세요.: 22
두 번째 정숫값을 입력하세요.: 8
두 정숫값의 최대 공약수는 2입니다.
```

05-2 재귀 알고리즘 분석

재귀 알고리즘의 2가지 분석 방법 – (1)

- **실습 5-3**
  - recur( )라는 재귀 함수 프로그램
  - recur( ) 함수는 함수 안에서 재귀 호출을 2번 실행
  - 순수한genuinely 재귀: 재귀 호출을 여러 번 실행하는 함수
  - 하향식top-down 분석
    - 가장 상위 함수 호출부터 시작하여 계단식으로 자세히 조사해 나가는 분석 방법
    - recur(4)의 실행 과정

      1. recur(3) 실행
      2. 4 출력
      3. recur(2) 실행

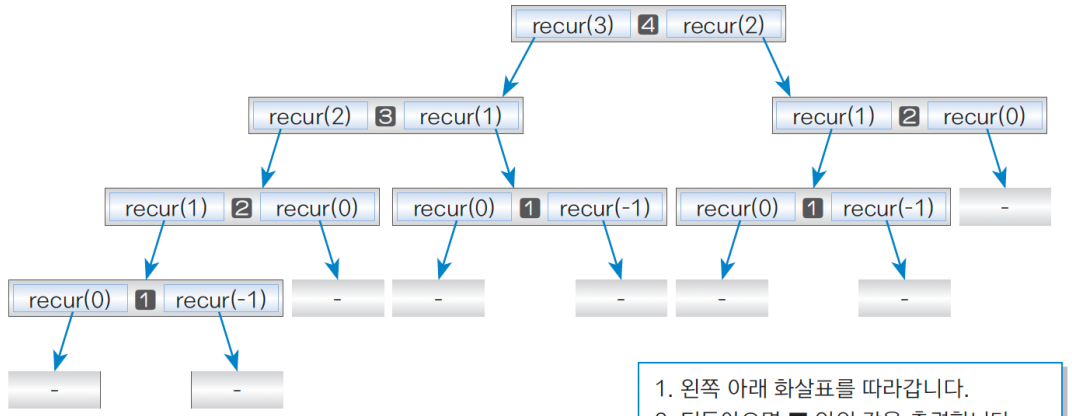
    - 같은 함수를 여러 번 호출할 수 있어, 효율성 ↓

Do it! 실습 5-3 • 완성 파일 chap05/recur1.py

```
01: # 순수한 재귀 함수 구현하기
02:
03: def recur(n: int) -> int:
04:     """순수한 재귀 함수 recur의 구현"""
05:     if n > 0:
06:         recur(n - 1)
07:         print(n)
08:         recur(n - 2)
09:
10: x = int(input('정숫값을 입력하세요.: '))
11:
12: recur(x)
```

실행 결과

```
정숫값을 입력하세요.: 4
1
2
3
1
4
1
2
```



05-2 재귀 알고리즘 분석

재귀 알고리즘의 2가지 분석 방법 – (2)

- **실습 5-3**
  - 상향식bottom-up 분석
    - 하향식 분석과는 반대로 아래쪽부터 쌓아 올리며 분석하는 방법
    - recur(1)의 실행 과정

      1. recur(0) 실행
      2. 1 출력
      3. recur(-1) 실행

    - recur(2)의 실행 과정

      1. recur(1) 실행
      2. 2 출력
      3. recur(0) 실행

```
recur(-1)  :  아무것도 하지 않음
recur(0)   :  아무것도 하지 않음
-------------------------------------------------------------------------------
recur(1)   :  recur(0)  1  recur(-1)  →  1
recur(2)   :  recur(1)  2  recur(0)   →  1 2
recur(3)   :  recur(2)  3  recur(1)   →  1 2 3 1
recur(4)   :  recur(3)  4  recur(2)   →  1 2 3 1 4 1 2
```

05-2 재귀 알고리즘 분석

재귀 알고리즘의 비재귀적 표현 – (1)

- **꼬리 재귀 제거하기**
  - recur( ) 함수의 맨 끝에서 재귀 호출하는 꼬리 재귀
    recur(n - 2) 함수의 의미는 '인수로 n-2의 값을 전달하고 recur( ) 함수를 호출하는 것'
  - 다음 동작으로 변경 가능

> n의 값을 n– 2로 업데이트하고 함수의 시작 지점으로 돌아감

- **실습 5-4**
  - 함수의 맨 끝에서 실행된 재귀 호출인 꼬리 재귀tail recursion 제거한 프로그램

Do it! 실습 5-4

• 완성 파일 chap05/recur1a.py

```
01: # 비재귀적으로 재귀 함수 구현하기(꼬리 재귀를 제거)
02:
03: def recur(n: int) -> int:
04:     """꼬리 재귀를 제거한 recur( ) 함수"""
05:     while n > 0:
06:         recur(n - 1)
07:         print(n)
08:         n = n - 2
(… 생략 …)
```

05-2 재귀 알고리즘 분석

재귀 알고리즘의 비재귀적 표현 – (2)

- **재귀 제거하기**
  - n값을 출력하기 전에 recur(n – 1)을 실행해야 하기 때문에, 맨 앞에서 재귀 호출하는 recur(n – 1) 함수 제거는 까다로움
  - 현재의 n값을 임시로 저장해야함
    → 04장에서 다룬 스택stack으로 해결 가능

- **실습 5-5**
  - 스택을 사용하여 비재귀적으로 구현한 recur( ) 함수

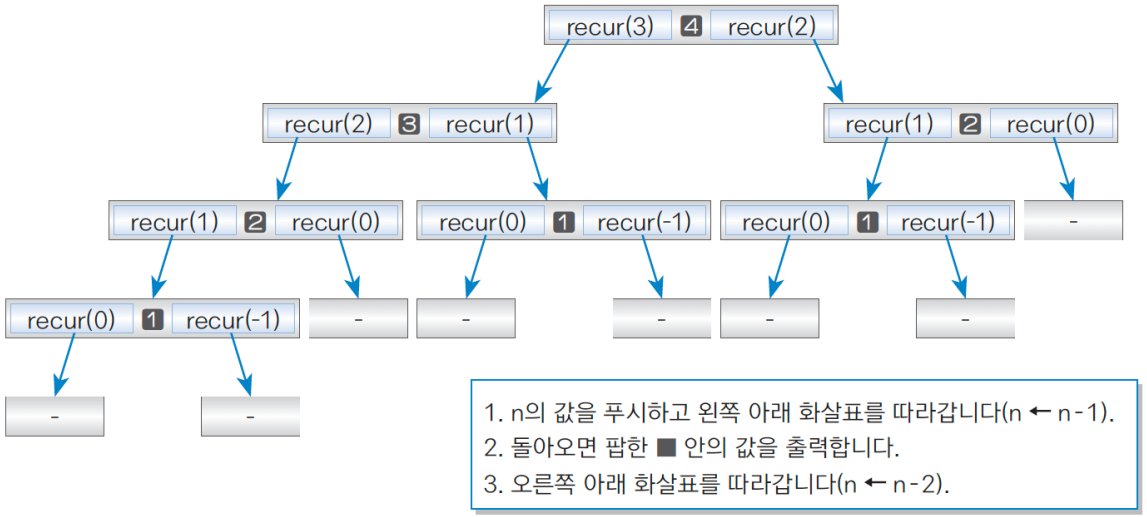
Do it! 실습 5-5 • 완성 파일 chap05/recur1b.py

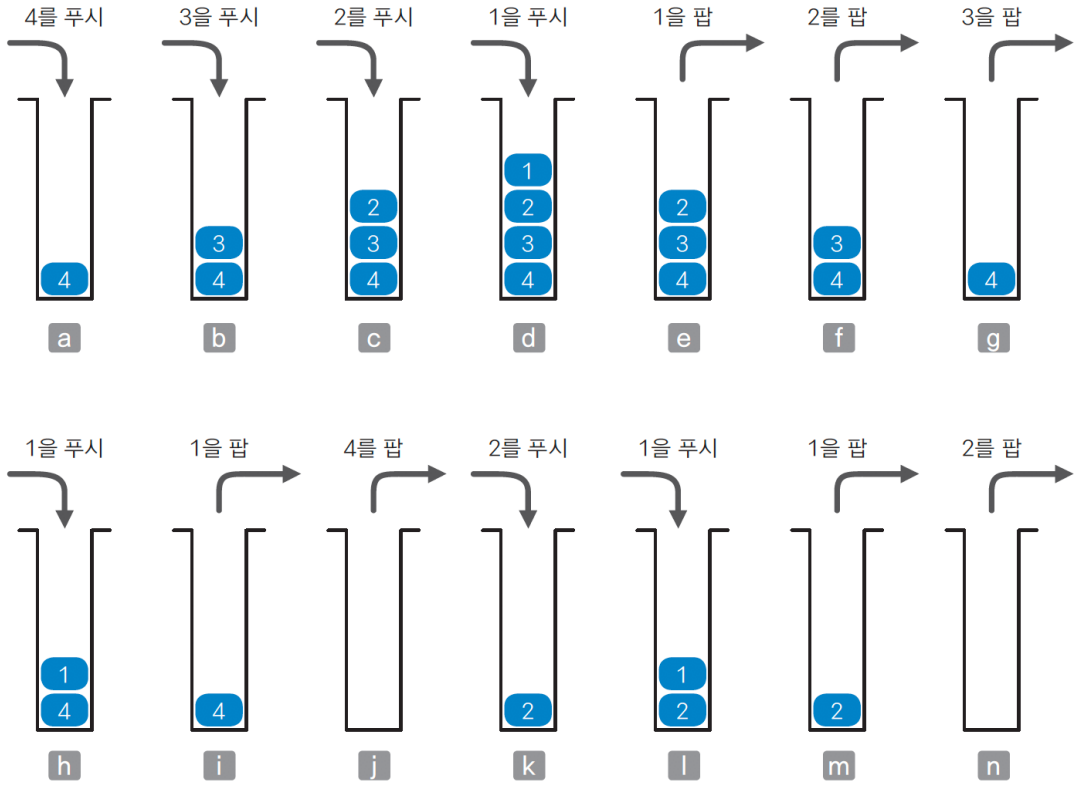
```
01: # 스택으로 재귀 함수 구현하기(재귀를 제거)
02:
03: from stack import Stack          # stack.py의 Stack 클래스를 임포트
04:
05: def recur(n: int) -> int:
06:     """재귀를 제거한 recur( ) 함수"""
07:     s = Stack(n)
08:
09:     while True:
10:         if n > 0:
11:             s.push(n)            # n값을 푸시          1
12:             n = n - 1                                  2
13:             continue                                   3
14:         if not s.is_empty():     # 스택이 비어 있지 않으면
15:             n = s.pop()          # 저장한 값을 n에 팝   4
16:             print(n)                                   5
17:             n = n - 2                                  6
18:             continue                                   7
19:         break
20:
21: x = int(input('정숫값을 입력하세요.: '))
22:
23: recur(x)
```

05-2 재귀 알고리즘 분석

재귀 알고리즘의 비재귀적 표현 – (3)

- **실습 5-5의 recur( ) 함수 실행에 따른 스택의 변화**

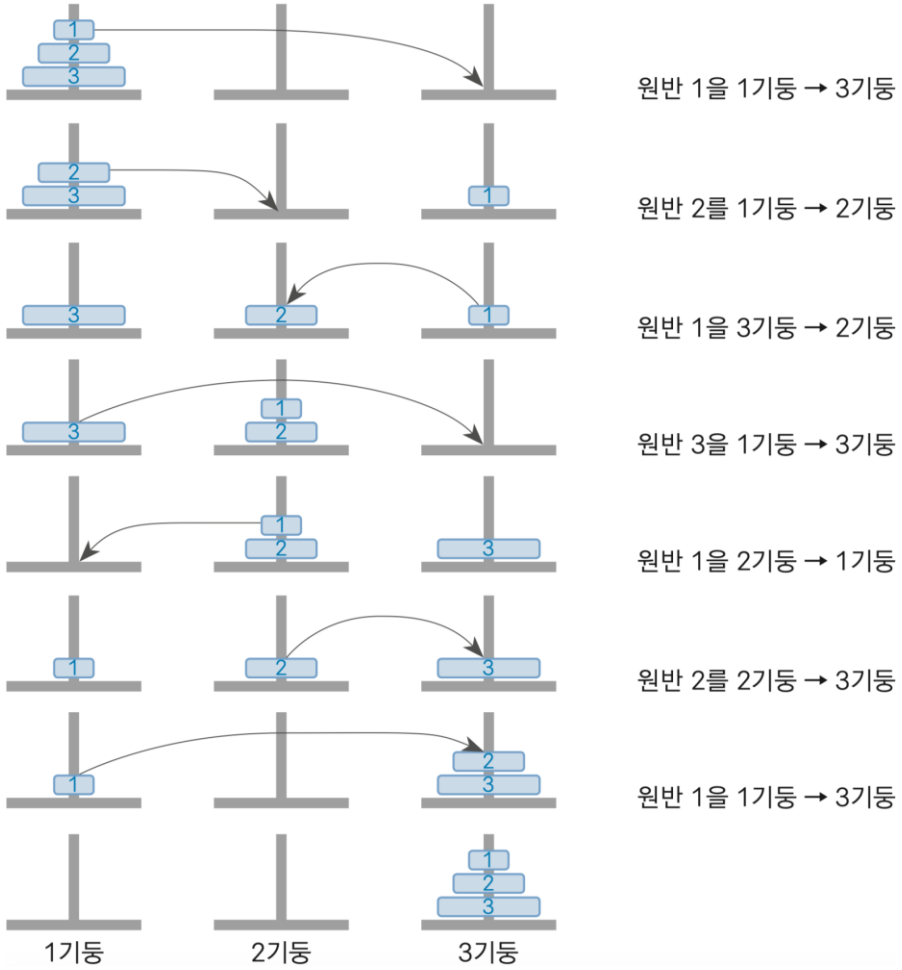




05-3 하노이의 탑

하노이의 탑 알아보기 – (1)

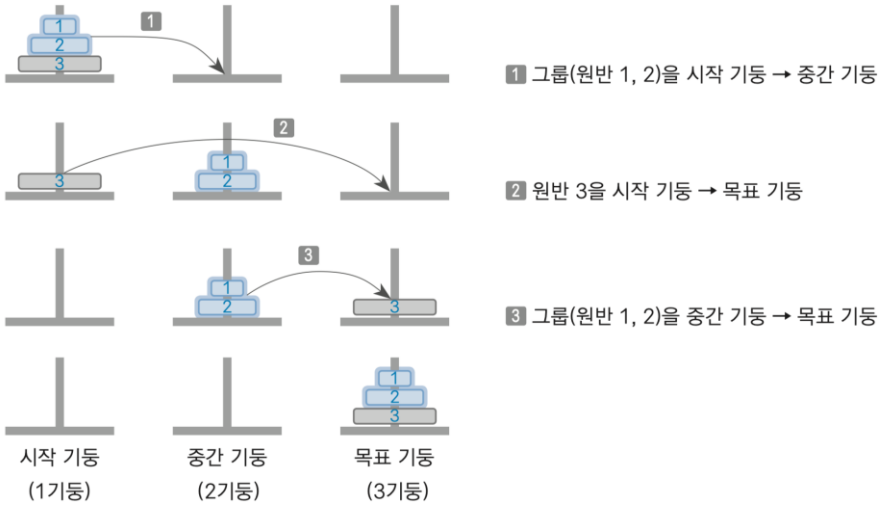
- **하노이의 탑**towers of Hanoi
  - 작은 원반이 위에, 큰 원반이 아래에 위치하는 규칙을 지키면서 기둥 3개를 이용해서 원반을 옮기는 문제
  - 크기가 모두 다른 원반이 첫 번째 기둥에 쌓여 있는 상태로 시작
  - 작은 원반은 위에, 큰 원반은 아래에 쌓여 있음
  - 원반은 1개씩 옮길 수 있으며 큰 원반은 작은 원반 위에 쌓을 수 없음



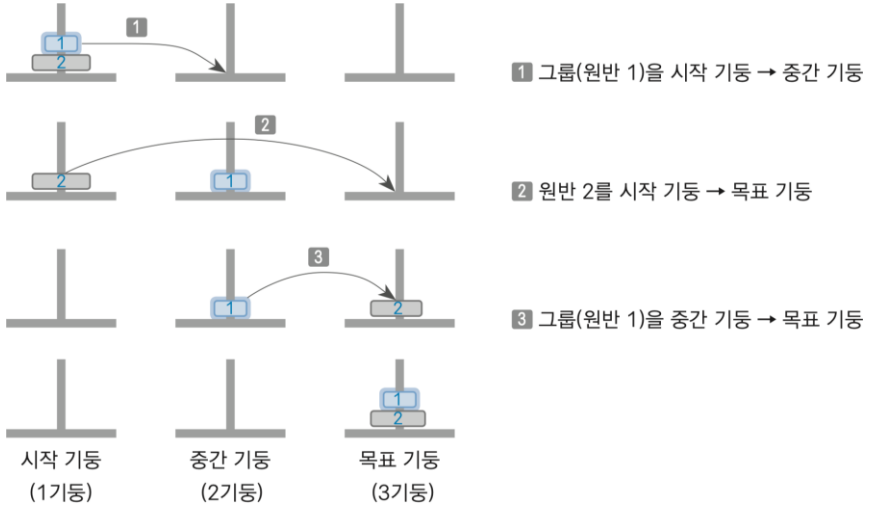
05-3 하노이의 탑

하노이의 탑 알아보기 – (2)

- **하노이의 탑**towers of Hanoi
  - a 원반이 3개일 때
    - 원반 1과 원반 2를 그룹으로 묶으면,
      가장 먼저 이 그룹을 중간 기둥으로 옮긴 후에
      가장 큰 원반 3을 목표 기둥으로 옮김
    - 총 3 단계로 이동 가능



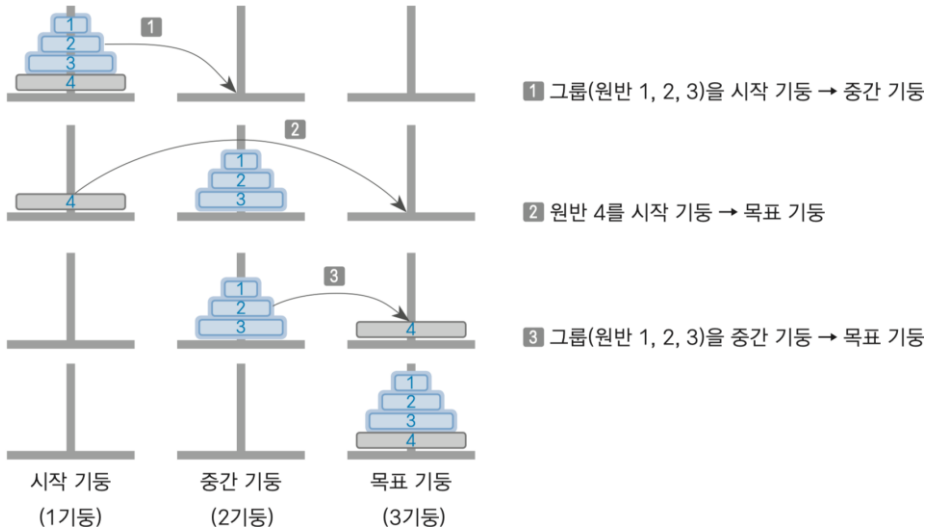
  - b 원반이 2개일 때
    - 원반 1과 원반 2를 묶은 그룹을 풀어서 옮기는 단계를
      어떻게 구현할까?
    - 원반 1을 그룹으로 본다면,
      a 와 똑같이 3단계로 이동 가능



05-3 하노이의 탑

하노이의 탑 알아보기 – (3)

- **하노이의 탑**towers of Hanoi
  - C 원반이 4개일 때
    - 원반 1, 원반 2, 원반 3을 그룹으로 묶으면 3단계로 이동 가능
    - 원반 3개를 묶었던 그룹을 풀어서 이동
    - 원반이 n개인 하노이의 탑 문제 해결 가능!



1 그룹(원반 1, 2, 3)을 시작 기둥 → 중간 기둥

2 원반 4를 시작 기둥 → 목표 기둥

3 그룹(원반 1, 2, 3)을 중간 기둥 → 목표 기둥

05-3 하노이의 탑

하노이의 탑 알아보기 – (4)

- **실습 5-6**
  - 하노이의 탑을 구현하는 프로그램
  - move( ) 함수의 매개변수 no: 옮겨야 할 원반의 개수
  - x: 시작 기둥 번호
  - y: 목표 기둥 번호

Do it! 실습 5-6

• 완성 파일 chap05/hanoi.py

```
01: # 하노이의 탑 구현하기
02:
03: def move(no: int, x: int, y: int) -> None:
04:     """원반 no개를 x기둥에서 y기둥으로 옮김"""
05:     if no > 1:
06:         move(no - 1, x, 6 - x - y)
07:
08:     print(f'원반 [{no}]을(를) {x}기둥에서 {y}기둥으로 옮깁니다.')
09:
10:     if no > 1:
11:         move(no - 1, 6 - x - y, y)
12:
13: print('하노이의 탑을 구현합니다.')
14: n = int(input('원반의 개수를 입력하세요.: '))
15:
16: move(n, 1, 3)  # 1기둥에 쌓인 원반 n개를 3기둥으로 옮김
```

실행 결과

```
하노이의 탑을 구현합니다.
원반의 개수를 입력하세요.: 3
원반 [1]을 1기둥에서 3기둥으로 옮깁니다.
원반 [2]를 1기둥에서 2기둥으로 옮깁니다.
원반 [1]을 3기둥에서 2기둥으로 옮깁니다.
원반 [3]을 1기둥에서 3기둥으로 옮깁니다.
원반 [1]을 2기둥에서 1기둥으로 옮깁니다.
원반 [2]를 2기둥에서 3기둥으로 옮깁니다.
원반 [1]을 1기둥에서 3기둥으로 옮깁니다.
```

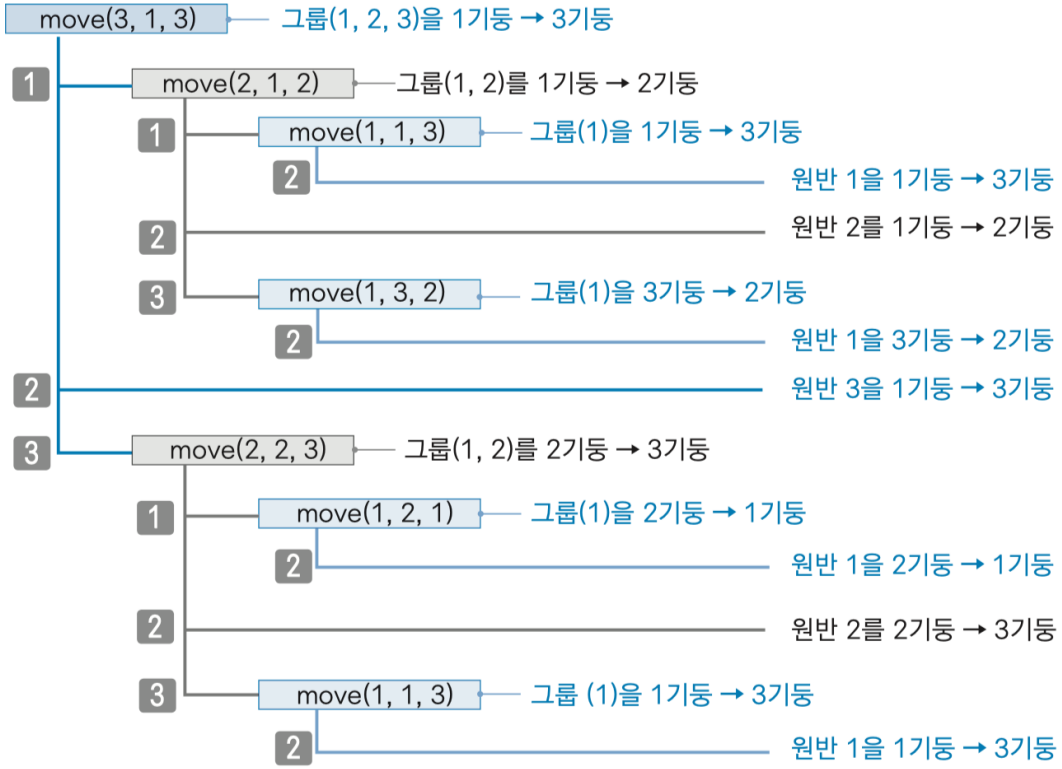
05-3 하노이의 탑

하노이의 탑 알아보기 – (5)

- **실습 5-6**
  - move( ) 함수의 동작

1 바닥에 있는 원반을 제외한 그룹(원반[1]~원반[no – 1]) 1기둥 → 2기둥
2 바닥에 있는 원반 [no]를 1기둥 → 3기둥
3 바닥에 있는 원반을 제외한 그룹(원반[1]~원반[no – 1]) 2기둥 → 3기둥

  - 1, 3 과정은 재귀 호출로 구현



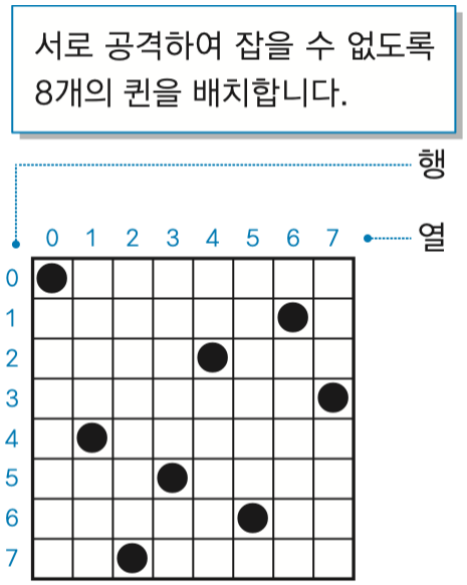
05-4 8퀸 문제

8퀸 문제 알아보기 – (1)

- **8퀸 문제**8-Queen problem
  - 재귀 알고리즘을 설명할 때 자주 나오는 예제
  - 19세기의 수학자 카를 F. 가우스Carl F. Gauss가 오답을 발표한 것으로 유명

> 8개의 퀸이 서로 공격하여 잡을 수 없도록 8 × 8 체스판에 배치하세요

  - 총 92가지 해결 방법



05-4 8퀸 문제

퀸 배치하기

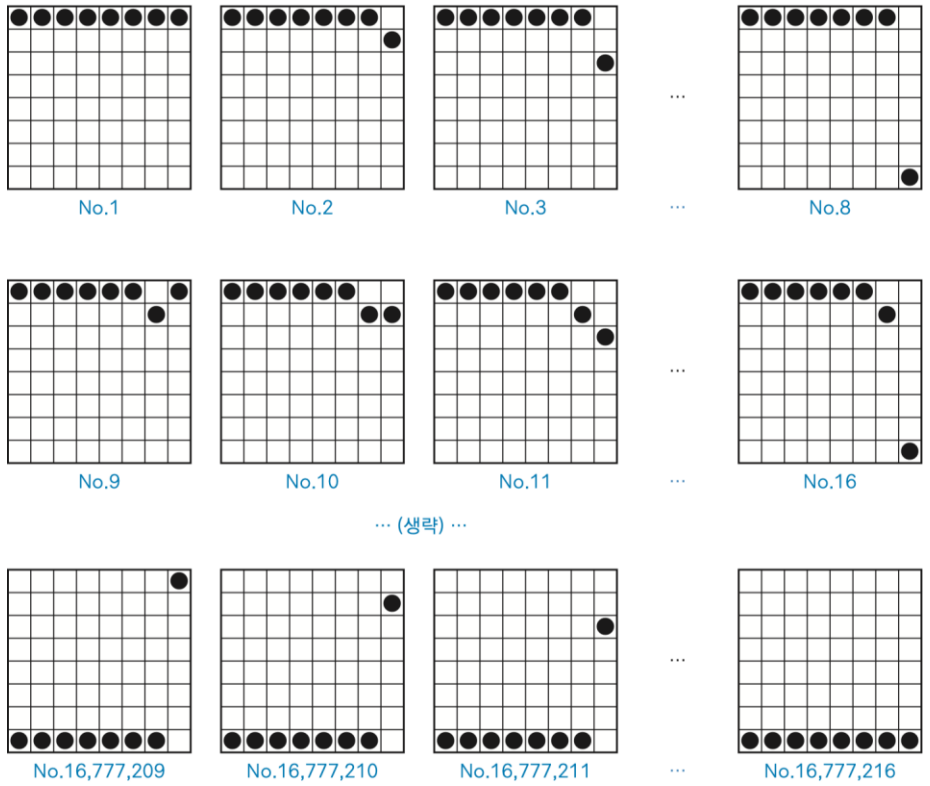
- **퀸 8개의 배치 조합**
  - 체스판 64칸(8 × 8)
  - 첫 번째 퀸을 배치할 때는 64칸 중 한 곳 선택
  - 두 번째 퀸을 배치할 때는 63칸 중 한 곳 선택
  - 이렇게 8번째까지 퀸을 배치하면

$$64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57 = 178{,}462{,}987{,}637{,}760$$

  - 규칙 1

규칙 1 각 열에 퀸을 1개만 배치

$$8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 = 16{,}777{,}216$$



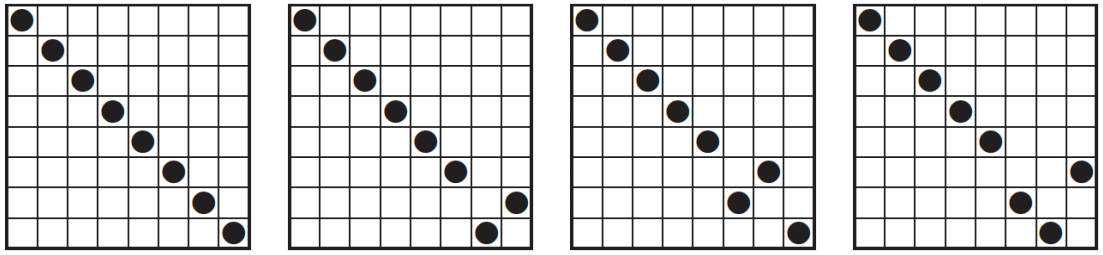
05-4 8퀸 문제

퀸 배치하기

- **퀸 8개의 배치 조합**
  - 규칙 2

> **규칙 2** 각 행에 퀸을 1개만 배치

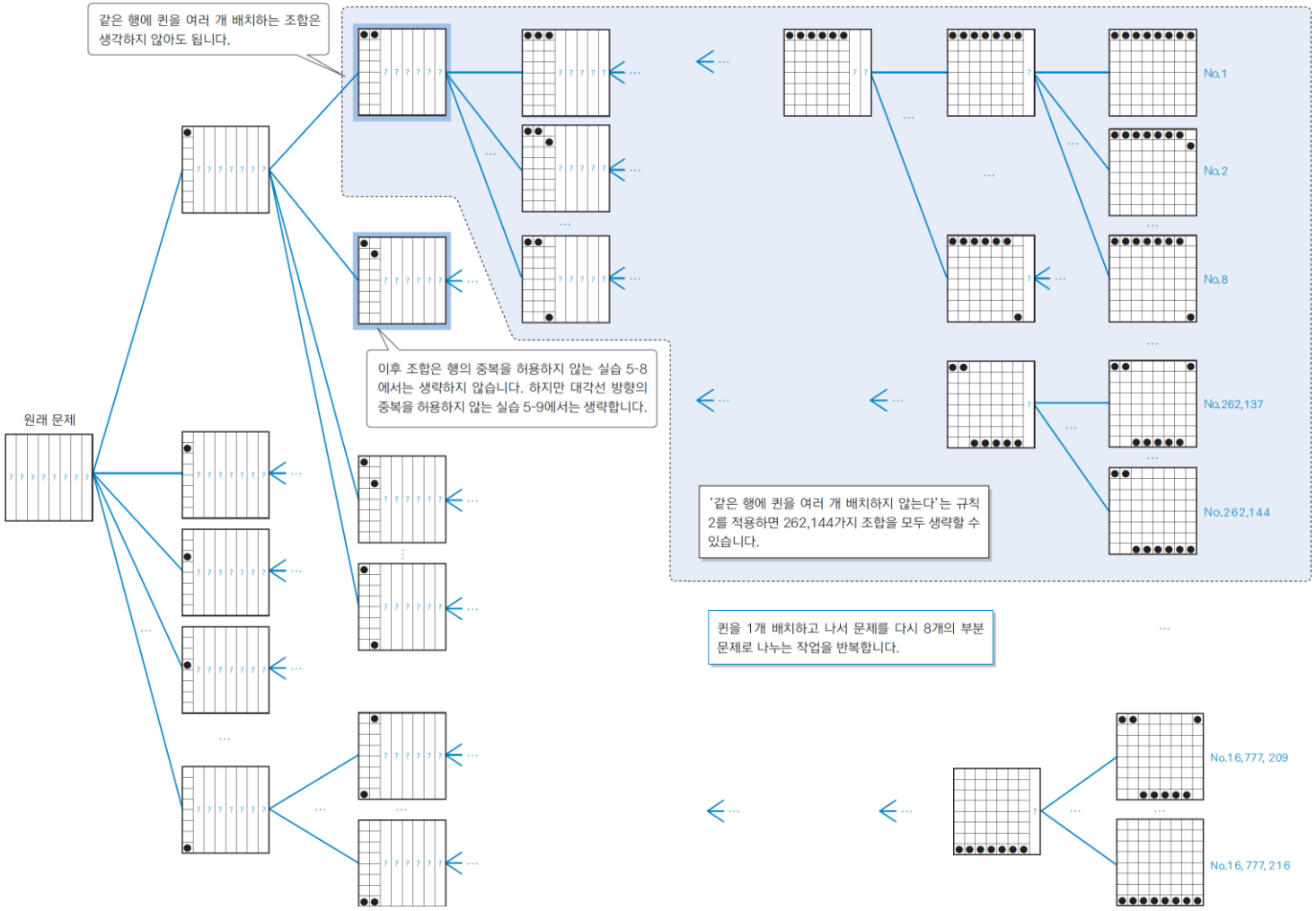
  - 규칙 1과 규칙 2를 만족하는 예시



05-4 8퀸 문제

퀸 배치하기

- **퀸 8개의 배치 조합**
  - **분기branching 작업**
    - 차례대로 가지가 뻗어 나가듯이 배치 조합을 열거하는 방법
  - **분할 정복법divide and conquer**
    - 하노이의 탑이나 8퀸 문제처럼 큰 문제를 작은 문제로 분할하고 작은 문제 풀이법을 결합하여 전체 풀이법을 얻는 방법

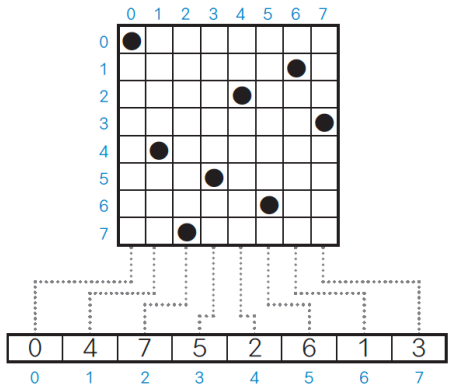


05-4 8퀸 문제

8퀸 문제 알아보기

- **실습 5-7**
  - 모든 조합을 나열하는 프로그램
  - 배열 pos는 퀸의 배치를 나타냄
  - i열에 배치한 퀸의 위치가 j행이면, pos[i] = j

i열에 배치한 퀸의 위치가 j행이면 pos[i]의 값을 j로 합니다.



Do it! 실습 5-7

• 완성 파일 chap05/8queen_b.py

```
01: # 각 열에 퀸을 1개 배치하는 조합을 재귀적으로 나열하기
02:
03: pos = [0] * 8              # 각 열에서 퀸의 위치를 출력
04:
05: def put() -> None:
06:     """각 열에 배치한 퀸의 위치를 출력"""
07:     for i in range(8):
08:         print(f'{pos[i]:2}', end='')
09:     print()
10:
11: def set(i: int) -> None:
12:     """i열에 퀸을 배치"""
13:     for j in range(8):
14:         pos[i] = j          # 퀸을 j행에 배치
15:         if i == 7 :         # 모든 열에 퀸 배치를 종료
16:             put()
17:         else:
18:             set(i + 1)      # 다음 열에 퀸을 배치
19:
20: set(0)  # 0열에 퀸을 배치
```

실행 결과

```
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 3
 0 0 0 0 0 0 0 4
 0 0 0 0 0 0 0 5
 0 0 0 0 0 0 0 6
 0 0 0 0 0 0 0 7
 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 1 2
 0 0 0 0 0 0 1 3
 0 0 0 0 0 0 1 4
 0 0 0 0 0 0 1 5
 0 0 0 0 0 0 1 6
(… 생략 …)
 7 7 7 7 7 7 7 6
 7 7 7 7 7 7 7 7
```

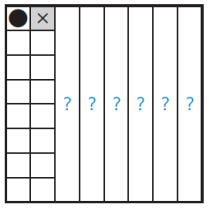
05-4 8퀸 문제

한정 작업과 분기 한정법 – (1)

- **실습 5-8**
  - 각 행에 퀸을 1개만 배치하는 규칙 1를 적용한 프로그램
  - flag 배열
    - 같은 행에 중복하여 퀸을 배치하지 않기 위한 표시
    - j행에 퀸을 배치하면 flag[j] = True, 배치하지 않으면 False

0행 0열에 퀸을 배치한 상태에서 1열에 퀸을 배치하는 방법 검토합니다.

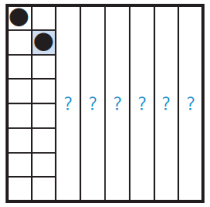
a



0	True
1	False
2	False
3	False
4	False
5	False
6	False
7	False

0행에는 이미 퀸을 배치했으므로 이 조합은 생각할 필요가 없습니다.

b



0	True
1	False
2	False
3	False
4	False
5	False
6	False
7	False

1행에는 퀸을 아직 배치하지 않았으므로 퀸을 배치합니다.

※ 2~7행도 마찬가지로 배치합니다.

Do it! 실습 5-8

• 완성 파일 chap05/8queen_bb.py

```
01: # 행과 열에 퀸을 1개 배치하는 조합을 재귀적으로 나열하기
02:
03: pos = [0] * 8                  # 각 열에서 퀸의 위치
04: flag = [False] * 8             # 각 행에 퀸을 배치했는지 체크
05:
06: def put() -> None:
07:     """각 열에 배치한 퀸의 위치를 출력"""
08:     for i in range(8):
09:         print(f'{pos[i]:2}', end='')
10:     print()
11:
12: def set(i: int) -> None:
13:     """i열의 알맞은 위치에 퀸을 배치"""
14:     for j in range(8):
15:         if not flag[j]:            # j행에 퀸을 배치하지 않았으면
16:             pos[i] = j             # 퀸을 j행에 배치
17:             if i == 7:             # 모든 열에 퀸 배치를 완료
18:                 put()
19:             else:
20:                 flag[j] = True
21:                 set(i + 1)     # 다음 열에 퀸을 배치
22:                 flag[j] = False
23:
24: set(0)                         # 0열에 퀸을 배치
```

▶ 실행 결과

```
 0 1 2 3 4 5 6 7
 0 1 2 3 4 5 7 6
 0 1 2 3 4 6 5 7
 0 1 2 3 4 6 7 5
 0 1 2 3 4 7 5 6
 0 1 2 3 4 7 6 5
 0 1 2 3 5 4 6 7
 0 1 2 3 5 4 7 6
 0 1 2 3 5 6 4 7
 0 1 2 3 5 6 7 4
 0 1 2 3 5 7 4 6
 0 1 2 3 5 7 6 4
 0 1 2 3 6 4 5 7
 0 1 2 3 6 4 7 5
 0 1 2 3 6 5 4 7
 0 1 2 3 6 5 7 4
 0 1 2 3 6 7 4 5
 0 1 2 3 6 7 5 4
 0 1 2 3 7 4 5 6
 0 1 2 3 7 4 6 5
(… 생략 …)
 7 6 5 4 3 2 1 0
```

05-4 8퀸 문제

한정 작업과 분기 한정법 – (2)

- **실습 5-8**
  - set( ) 함수에서는 퀸을 아직 배치하지 않은 행(flag[j]가 False인 행)에만 퀸을 배치할 수 있음
  - **한정bounding 작업**
    - 필요하지 않은 분기를 없애서 불필요한 조합을 열거하지 않는 방법
  - **분기 한정법branching and bounding method**
    - 분기 작업과 한정 작업을 조합하여 문제를 풀이하는 방법

Do it! 실습 5-8

• 완성 파일 chap05/8queen_bb.py

```
01: # 행과 열에 퀸을 1개 배치하는 조합을 재귀적으로 나열하기
02:
03: pos = [0] * 8                  # 각 열에서 퀸의 위치
04: flag = [False] * 8             # 각 행에 퀸을 배치했는지 체크
05:
06: def put() -> None:
07:     """각 열에 배치한 퀸의 위치를 출력"""
08:     for i in range(8):
09:         print(f'{pos[i]:2}', end='')
10:     print()
11:
12: def set(i: int) -> None:
13:     """i열의 알맞은 위치에 퀸을 배치"""
14:     for j in range(8):
15:         if not flag[j]:          # j행에 퀸을 배치하지 않았으면
16:             pos[i] = j           # 퀸을 j행에 배치
17:             if i == 7:           # 모든 열에 퀸 배치를 완료
18:                 put()
19:             else:
20:                 flag[j] = True
21:                 set(i + 1)       # 다음 열에 퀸을 배치
22:                 flag[j] = False
23:
24: set(0)                          # 0열에 퀸을 배치
```

▶ 실행 결과

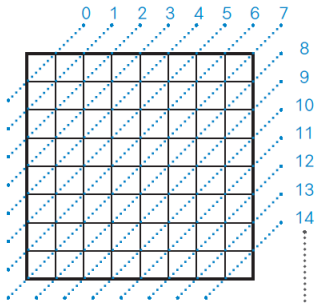
```
 0 1 2 3 4 5 6 7
 0 1 2 3 4 5 7 6
 0 1 2 3 4 6 5 7
 0 1 2 3 4 6 7 5
 0 1 2 3 4 7 5 6
 0 1 2 3 4 7 6 5
 0 1 2 3 5 4 6 7
 0 1 2 3 5 4 7 6
 0 1 2 3 5 6 4 7
 0 1 2 3 5 6 7 4
 0 1 2 3 5 7 4 6
 0 1 2 3 5 7 6 4
 0 1 2 3 6 4 5 7
 0 1 2 3 6 4 7 5
 0 1 2 3 6 5 4 7
 0 1 2 3 6 5 7 4
 0 1 2 3 6 7 4 5
 0 1 2 3 6 7 5 4
 0 1 2 3 7 4 5 6
 0 1 2 3 7 4 6 5
(… 생략 …)
 7 6 5 4 3 2 1 0
```

05-4 8퀸 문제

8퀸 문제 해결 프로그램 만들기

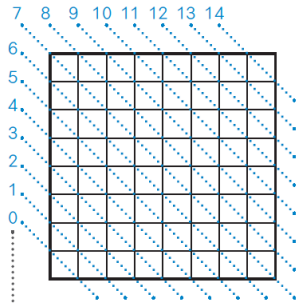
- **실습 5-9**
  - 체스에서 퀸은 대각선 방향으로 이동할 수 있으므로 어떤 대각선에서 보더라도 퀸을 1개만 배치하는 한정 작업을 추가로 적용해야 함
  - 8퀸의 대각선 배치 문제까지 고려하여 해결한 프로그램
  - flag_b, flag_c 배열
    - 양쪽 대각선 방향에 퀸을 배치했는지 검토하는 배열

a flag_b를 검토하는 라인



j행 i열의 값은 i + j로 구할 수 있습니다.

b flag_c를 검토하는 라인



j행 i열의 값은 i - j +7로 구할 수 있습니다.

Do it! 실습 5-9

• 완성 파일 chap05/8queen.py

```
01: # 8퀸 문제 알고리즘 구현하기
02:
03: pos = [0] * 8
04: flag_a = [False] * 8
05: flag_b = [False] * 15
06: flag_c = [False] * 15
07:
08: def put() -> None:
09:     """각 열에 배치한 퀸의 위치를 출력"""
10:     for i in range(8):
11:         print(f'{pos[i]:2}', end='')
12:     print()
13:
14: def set(i: int) -> None:
15:     """i열의 알맞은 위치에 퀸을 배치"""
16:     for j in range(8):
17:         if(     not flag_a[j]             # j행에 퀸이 배치되지 않았다면
18:             and not flag_b[i + j]         # 대각선 방향(／)으로 퀸이 배치되지 않았다면
19:             and not flag_c[i - j + 7]):   # 대각선 방향(＼)으로 퀸이 배치되지 않았다면
20:             pos[i] = j                    # 퀸을 j행에 배치
21:             if i == 7:                    # 모든 열에 퀸을 배치 완료
22:                 put()
23:             else:
24:                 flag_a[j] = flag_b[i + j] = flag_c[i - j + 7] = True
25:                 set(i + 1)                # 다음 열에 퀸을 배치
26:                 flag_a[j] = flag_b[i + j] = flag_c[i - j + 7] = False
27:
28: set(0)          # 0열에 퀸을 배치
```

실행 결과

```
 0 4 7 5 2 6 1 3
 0 5 7 2 6 3 1 4
 0 6 3 5 7 1 4 2
 0 6 4 7 1 3 5 2
 1 3 5 7 2 0 6 4
 1 4 6 0 2 7 5 3
 1 4 6 3 0 7 5 2
(… 생략 …)
 7 2 0 5 1 4 6 3
 7 3 0 2 5 1 6 4
```