2주차 재귀

BABO 4기 이론반

분석 23기 김윤주

CONTENTS

1. 1주차 과제 코드 리뷰

1주차 과제

1. **백준 1018** 체스판 다시 칠하기
2. **백준 18870** 좌표 압축
3. **프로그래머스** 연속된 부분 수열의 합

1주차 과제 – 백준 1018

문제

지민이는 자신의 저택에서 MN개의 단위 정사각형으로 나누어져 있는 M×N 크기의 보드를 찾았다. 어떤 정사각형은 검은색으로 칠해져 있고, 나머지는 흰색으로 칠해져 있다. 지민이는 이 보드를 잘라서 8×8 크기의 체스판으로 만들려고 한다.

체스판은 검은색과 흰색이 번갈아서 칠해져 있어야 한다. 구체적으로, 각 칸이 검은색과 흰색 중 하나로 색칠되어 있고, 변을 공유하는 두 개의 사각형은 다른 색으로 칠해져 있어야 한다. 따라서 이 정의를 따르면 체스판을 색칠하는 경우는 두 가지뿐이다. 하나는 맨 왼쪽 위 칸이 흰색인 경우, 하나는 검은색인 경우이다.

보드가 체스판처럼 칠해져 있다는 보장이 없어서, 지민이는 8×8 크기의 체스판으로 잘라낸 후에 몇 개의 정사각형을 다시 칠해야겠다고 생각했다. 당연히 8*8 크기는 아무데서나 골라도 된다. 지민이가 다시 칠해야 하는 정사각형의 최소 개수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 M이 주어진다. N과 M은 8보다 크거나 같고, 50보다 작거나 같은 자연수이다. 둘째 줄부터 N개의 줄에는 보드의 각 행의 상태가 주어진다. B는 검은색이며, W는 흰색이다.

출력

첫째 줄에 지민이가 다시 칠해야 하는 정사각형 개수의 최솟값을 출력한다.

1주차 과제 – 백준 18870

문제

수직선 위에 N개의 좌표 X_1, X_2, ..., X_N이 있다. 이 좌표에 좌표 압축을 적용하려고 한다.

X_i를 좌표 압축한 결과 X'_i의 값은 $X_i > X_j$를 만족하는 서로 다른 좌표 X_j의 개수와 같아야 한다.

X_1, X_2, ..., X_N에 좌표 압축을 적용한 결과 X'_1, X'_2, ..., X'_N를 출력해보자.

입력

첫째 줄에 N이 주어진다.

둘째 줄에는 공백 한 칸으로 구분된 X_1, X_2, ..., X_N이 주어진다.

출력

첫째 줄에 X'_1, X'_2, ..., X'_N을 공백 한 칸으로 구분해서 출력한다.

제한

- $1 \le N \le 1,000,000$
- $-10^9 \le X_i \le 10^9$

예제 입력 1 복사

```
5
2 4 -10 4 -9
```

예제 입력 2 복사

```
6
1000 999 1000 999 1000 999
```

예제 출력 1 복사

```
2 3 0 3 1
```

예제 출력 2 복사

```
1 0 1 0 1 0
```

1주차 과제 – 프로그래머스 연속된 부분 수열의 합

문제 설명

비내림차순으로 정렬된 수열이 주어질 때, 다음 조건을 만족하는 부분 수열을 찾으려고 합니다.

- 기존 수열에서 임의의 두 인덱스의 원소와 그 사이의 원소를 모두 포함하는 부분 수열이어야 합니다.
- 부분 수열의 합은 `k` 입니다.
- 합이 `k` 인 부분 수열이 여러 개인 경우 길이가 짧은 수열을 찾습니다.
- 길이가 짧은 수열이 여러 개인 경우 앞쪽(시작 인덱스가 작은)에 나오는 수열을 찾습니다.

수열을 나타내는 정수 배열 `sequence` 와 부분 수열의 합을 나타내는 정수 `k` 가 매개변수로 주어질 때, 위 조건을 만족하는 부분 수열의 시작 인덱스와 마지막 인덱스를 배열에 담아 return 하는 solution 함수를 완성해주세요. 이때 수열의 인덱스는 0부터 시작합니다.

제한사항

- 5 ≤ `sequence` 의 길이 ≤ 1,000,000
  - 1 ≤ `sequence` 의 원소 ≤ 1,000
  - `sequence` 는 비내림차순으로 정렬되어 있습니다.
- 5 ≤ `k` ≤ 1,000,000,000
  - `k` 는 항상 `sequence` 의 부분 수열로 만들 수 있는 값입니다.

2. 재귀

재귀 (Recursive Function)

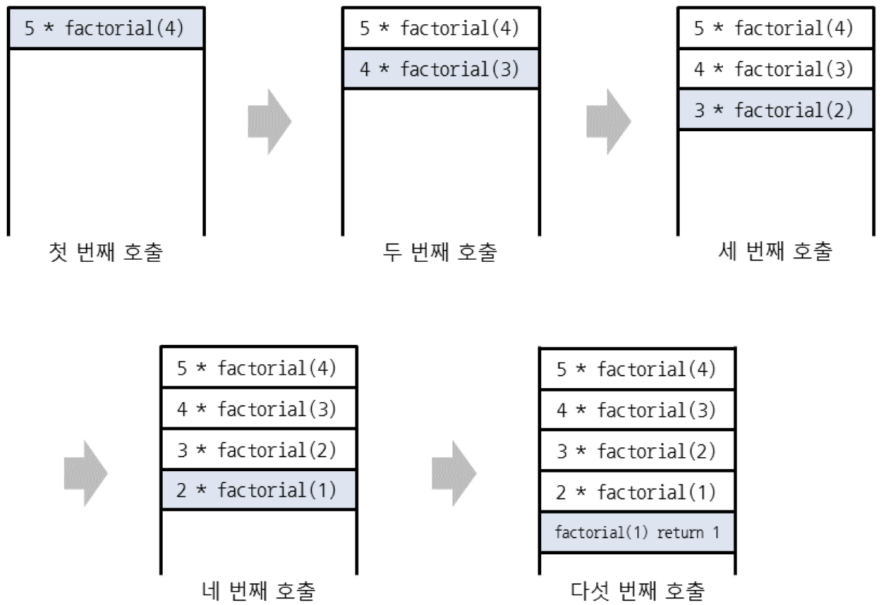
- 재귀함수란 어떤 함수에서 **자신을 다시 호출**하여 작업을 수행하는 방식의 함수를 의미
- 반복문을 사용하는 코드는 항상 재귀함수를 통해 구현하는 것이 가능하며 그 반대도 가능

- 코드가 간결하고 이해하기 쉬우며, 특정 유형의 문제(트리/그래프 탐색 등)에 매우 적합함
- 반복적인 스택의 사용으로 인해 메모리 및 속도에서 성능 저하가 발생할 수 있음

재귀 (Recursive Function)

- factorial(5) 를 호출했을 경우의 콜스택 변화

```
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n-1)
```



재귀 (Recursive Function)

재귀함수를 작성할 때 주의할 점

- 함수 내에서 다시 자신을 호출한 후 그 함수가 끝날 때까지 함수 호출 이후의 명령문이 수행되지 않기 때문에 종료조건이 꼭 포함이 되어야 함 (무한 루프 방지)

- 즉, 재귀 문제를 풀 때는 항상 **base case**와 **recursive case**로 경우를 나누어 주어야 함
- base case: 더 이상 자기 자신을 호출하지 않고, 직접 결과를 반환하는 조건

재귀 (Recursive Function)

앞서 작성한 factorial 에서 if(n <= 1) 부분이 빠진다면 어떻게 될까?

```
def factorial(n):
  return n * factorial(n - 1)
```

-> 메모리의 한계로 인해 무한대는 불가능하고 스택 오버플로우 문제 발생

-> 이렇게 발생하는 오버헤드를 방지하고자, **파이썬은 스택, 힙, 정적 메모리 제한**을 두고 있음

-> 기본적으로 재귀 깊이를 1000으로 제한

꼬리 재귀 (Tail Recursion)

- 함수 호출의 **결과**로 반환되는 값에 **함수 인자를 직접적으로 포함**시키는 방식으로 구현

- 재귀 호출 후 추가적인 연산이 필요하지 않다면 일반 함수를 호출하는 것처럼 시스템 콜 스택에 이것저것 저장하지 않고 선형적으로 구현할 수 있음

- 이렇게 구현된 함수는 컴파일러가 최적화를 수행하여 스택 메모리를 추가로 사용하지 않고, 반복문과 같이 빠르게 실행될 수 있음 (only 컴파일 언어)

꼬리 재귀 (Tail Recursion)

```python
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n-1)
```

```python
def factorial(n):
  if n <= 1:
    return total
  else:
  return factorial(n-1, n*total)
```

```python
def factorial(n, result=1):
    while n != 0:
        result = n * result
        n = n - 1
    return result
```

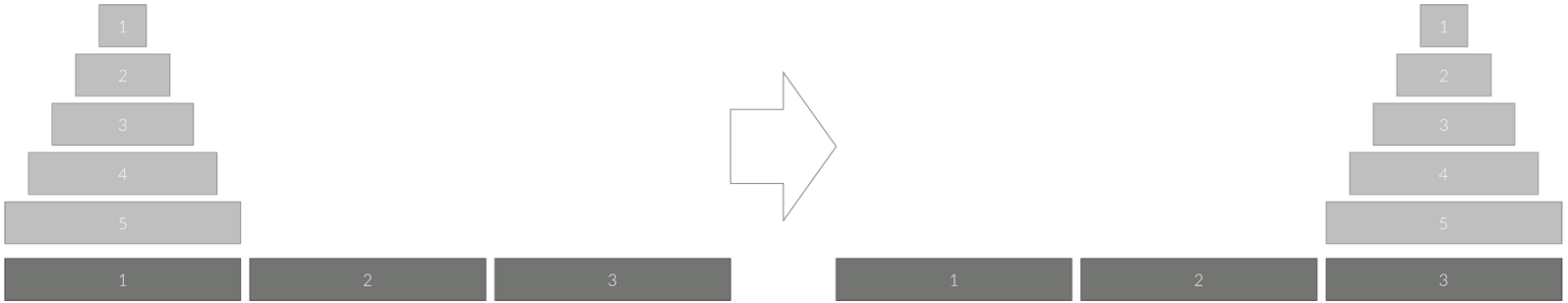
- **컴파일 언어**에서는 컴파일 시 꼬리 재귀 함수를 반복문으로 바꾸는 **성능 최적화**가 적용
- 하지만 파이썬은 **인터프리터 언어**라 자동적으로 최적화를 수행 X

=> **Tail Recursion Elimination**: 꼬리 재귀 조건을 만족한다면 실제로 함수를 호출하지 않고 반복문을 활용하도록 구현한 방식

재귀 관련 문제 - 백준 1914: 하노이탑

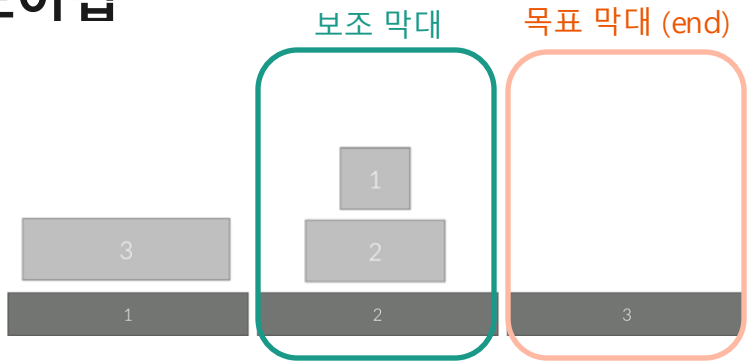
① 쌓아 놓은 원판은 항상 위의 것이 아래의 것보다 작아야 한다

② 원판은 위에서부터 꺼내, 1개씩 이동한다

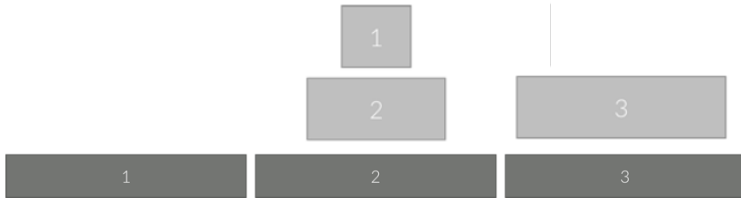


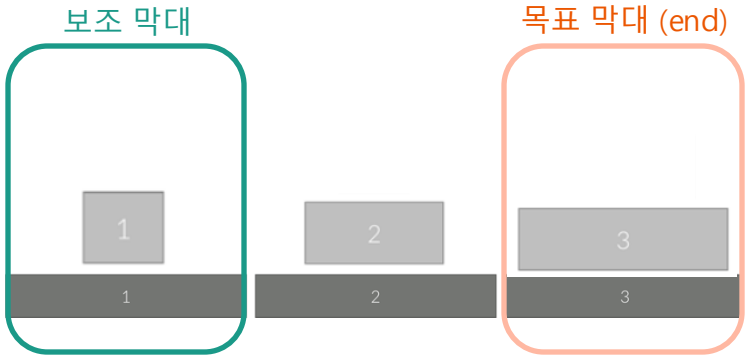
재귀 관련 문제 - 백준 1914: 하노이탑

n = 3



n = 2





재귀 관련 문제 - 백준 1914: 하노이탑

하노이 과정 일반화

① n이 2 이상이면, 보조 막대에 n-1 개를 옮기고(1단계) 맨 아래 원판을 목표 막대로 옮긴다(2단계)

② n이 1 이면, 목표 막대에 옮기고 끝낸다

```
# 입력
n = int(input())
print(2**n-1)  # 옮긴 횟수 K
hanoi_tower(n, 1, 3)
```

```
import sys
input = sys.stdin.readline

# 하노이의 탑 알고리즘
def hanoi_tower(n, start, end):
    # n이 1이 되면 종료
    if n == 1:
        print(start, end)
        return

    # 1단계, n-1개의 원판을 시작 막대에서 보조 막대로 옮김
    hanoi_tower(n-1, start, 6-start-end)

    # 2단계, 맨 아랫 원판을 시작 막대에서 목표 막대로 옮김
    print(start, end)

    # 다시 1, 2단계를 반복하기 위한 재귀호출
    hanoi_tower(n-1, 6-start-end, end)
```

과제 공지

1. 백준 4779번: 칸토어 집합 (S3)
2. 백준 1182번: 부분 수열의 합 (S2)
3. 백준 1759: 암호 만들기 (G5)