Algorithm

연속된 부분 수열의 합

[프로그래머스] 연속된 부분 수열의 합

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.

입출력 예

sequence	k	result
[1, 2, 3, 4, 5]	7	[2, 3]
[1, 1, 1, 2, 3, 4, 5]	5	[6, 6]
[2, 2, 2, 2, 2]	6	[0, 2]

1차 시도 – 투 포인터 with 루프 두 개

- 전형적인 투 포인터 문제
- K보다 크면 left를 증가, 그렇지 않으면 right를 증가
- k와 같으면 수열 길이를 비교하여 정답을 업데이트
- 아이디어는 정답과 같으나, 시간초과 발생
- 현재 알고리즘에선 시간복잡도가 O(n^2)

```java
public int[] solution(int[] sequence, int k) {
    int[] answer = {0, 0};
    int gap = 1000000;
    int left = 0;
    int right = 0;
    while (right < sequence.length){
        int sum = 0;
        for (int i = left; i <= right; i++){
            sum += sequence[i];
        }
        if (sum == k){
            if (right - left < gap){
                answer[0] = left;
                answer[1] = right;
                gap = right - left;
            }
            right++;
        } else if(sum > k){
            left++;
        } else {
            right++;
        }
    }
    return answer;
}
```

다른 사람 풀이 참고

- 매 루프마다 left부터 right까지 모두 더하던 나의 코드와는 다르게

```
for (int i = left; i <= right; i++){
    sum += sequence[i];
}
```

  - 포인터가 움직일 때만 해당 인덱스에서 배열의 값을 가져와 sum에 값을 더하고
  - 판별식이 끝나면 sum에서 값을 빼는 형식으로 간다.
  - Sum이 루프 밖에서 선언됨

```
int N = sequence.length;
int left = 0, right = N;
int sum = 0;
for(int L = 0, R = 0; L < N; L++) {
    while(R < N && sum < k) {
        sum += sequence[R++];
    }

    if(sum == k) {
        int range = R - L - 1;
        if((right - left) > range) {
            left = L;
            right = R - 1;
        }
    }

    sum -= sequence[L];
}

int[] answer = {left, right};

return answer;
```

2차 시도 – 투 포인터

- 같은 투 포인터지만 내부 루프를 최대한 적게 사용하도록 수정
- While(sum > k)는 경우에 따라서 사용하지 않을 수 있다.
- Sum에 right의 배열 값을 더하는 것은 매 루프에 하나씩 일어남
- 시간 복잡도가 사실상 O(n)으로 줄어듦

```
class Solution {
    public int[] solution(int[] sequence, int k) {
        int[] answer = {0, 0};
        int gap = 1000000;
        int left = 0;
        int right = 0;
        int sum = 0;
        while (right < sequence.length){
            sum += sequence[right];
            while(sum > k) {
                sum -= sequence[left];
                left++;
            }
            if(sum == k) {
                if(gap > right - left) {
                    gap = right - left;
                    answer[0] = left;
                    answer[1] = right;
                }
            }
            right++;
        }
        return answer;
    }
```