자료구조

자료구조는 데이터를 효율적으로 저장 , 접근 , 수정하기 위한 그릇

코딩 테스트에서는 각문제에 주어진 입력 데이터의 형태와 사용해야 하는 알고리즘에 따라 적절한 자료구조를 선정해 사용하는 것이 매우 중요.

자료구조 관련 문제 유형

1. 배열과 리스트
2. 구간 합
3. 투 포인터
4. 슬라이딩 윈도우
5. 스택과 큐

자료구조 관련 유형별 문제 예시

출처: 백준 온라인 저지

#01. 배열과 리스트

> 매우 기초적인 문제 입니다.

주소	문제	제한시간
https://www.acmicpc.net/problem/11720	숫자의 합 구하기	1초
https://www.acmicpc.net/problem/1546	평균 구하기	2초

#02. 구간 합

주소	문제	제한시간
https://www.acmicpc.net/problem/11659	구간 합 구하기 1	0.5초
https://www.acmicpc.net/problem/11660	구간 합 구하기 2	1초
https://www.acmicpc.net/problem/10986	나머지 합 구하기	1초

#03. 투 포인터

투 포인터(Two Pointers) 기법은 배열이나 리스트와 같은 선형 자료구조에서 두 개의 포인터를 사용하여 문제를 해결하는 알고리즘 기법

1. 부분 배열의 합: 특정 조건을 만족하는 부분 배열을 찾는 문제.
2. 정렬된 배열에서 두 수의 합: 두 수의 합이 특정 값을 만족하는 경우를 찾는 문제.
3. 중복된 요소 제거: 배열에서 중복된 요소를 제거하는 문제.

기본 아이디어

- 두 개의 포인터를 사용하여 배열의 시작과 끝, 또는 특정 위치에서 시작하여 조건을 만족할 때까지 이동
- 포인터의 이동: 조건에 따라 한 포인터를 이동시키거나 두 포인터를 동시에 이동

예제

정렬된 배열 arr와 정수 target이 주어졌을 때, 두 수의 합이 target이 되는 배열의 두 인덱스를 찾으시오.

```
def find_two_sum(arr, target):
    left = 0
    right = len(arr) - 1

    while left < right:
        sum = arr[left] + arr[right]
        if sum == target:
            return left, right
        elif sum < target:
            left += 1
        else:
            right -= 1
    return None

# 예제 사용
arr = [1, 2, 3, 4, 6]
target = 6
result = find_two_sum(arr, target)

if result:
    print(f"Indices: {result[0]}, {result[1]}")
else:
    print("No two sum solution")
```

1. 포인터 초기화: left 포인터는 배열의 시작(0)에서, right 포인터는 배열의 끝(len(arr) - 1)에서 시작
2. 반복문: left 포인터가 right 포인터보다 작을 때까지 반복
3. 합 계산: 현재 left와 right 포인터가 가리키는 요소의 합을 계산
4. 조건 검사:
    - 합이 target과 같으면 두 포인터의 인덱스를 반환
    - 합이 target보다 작으면 left 포인터를 오른쪽으로 이동
    - 합이 target보다 크면 right 포인터를 왼쪽으로 이동
5. 반환: 반복문이 종료될 때까지 조건을 만족하는 두 수를 찾지 못하면 None을 반환

```
public class TwoPointersExample {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 6};
        int target = 6;
        int[] result = findTwoSum(arr, target);

        if (result != null) {
            System.out.println("Indices: " + result[0] + ", " + result[1]);
        } else {
```

```
            System.out.println("No two sum solution");
        }
    }

    public static int[] findTwoSum(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left < right) {
            int sum = arr[left] + arr[right];
            if (sum == target) {
                return new int[]{left, right};
            } else if (sum < target) {
                left++;
            } else {
                right--;
            }
        }
        return null;
    }
}
```

주소	문제	제한시간
https://www.acmicpc.net/problem/2018	연속된 자연수의 합 구하기	2초
https://www.acmicpc.net/problem/1940	주몽의 명령	2초
https://www.acmicpc.net/problem/1253	좋은 수 구하기	2초

#04. 슬라이딩 윈도우

슬라이딩 윈도우(Sliding Window) 기법은 배열이나 리스트와 같은 선형 자료구조에서 일정한 크기의 윈도우(부분 배열)를 이동시키면서 문제를 해결하는 알고리즘 기법

1. 부분 배열의 최대/최소값 찾기: 일정한 크기의 부분 배열에서 최대값이나 최소값을 찾는 문제.
2. 부분 배열의 합: 일정한 크기의 부분 배열의 합을 구하는 문제.
3. 특정 조건을 만족하는 부분 배열 찾기: 예를 들어, 부분 배열의 요소들이 특정 조건을 만족하는 경우를 찾는 문제.

기본 아이디어

- 윈도우 크기 설정: 문제에 따라 고정된 크기의 윈도우를 설정합니다.
- 윈도우 이동: 윈도우를 배열의 시작부터 끝까지 한 요소씩 이동시키면서 필요한 계산을 수행
- 효율적인 계산: 이전 윈도우의 결과를 이용하여 다음 윈도우의 결과를 효율적으로 계산

예제

배열 arr와 윈도우 크기 k가 주어졌을 때, 각 윈도우에서 최대값을 구하시오.

```python
def max_sliding_window(arr, k):
    if not arr or k == 0:
        return []

    result = []
    window = []

    for i in range(len(arr)):
        # 윈도우 범위를 벗어난 인덱스를 제거
        if window and window[0] <= i - k:
            window.pop(0)

        # 현재 요소보다 작은 인덱스를 제거
        while window and arr[window[-1]] < arr[i]:
            window.pop()

        # 현재 인덱스를 리스트에 추가
        window.append(i)

        # 윈도우의 첫 번째 요소가 최대값
        if i >= k - 1:
            result.append(arr[window[0]])

    return result

# 예제 사용
arr = [1, 3, -1, -3, 5, 3, 6, 7]
k = 3
print(max_sliding_window(arr, k))  # 출력: [3, 3, 5, 5, 6, 7]
```

1. 리스트 사용: window 리스트를 사용하여 현재 윈도우 내의 인덱스를 관리
2. 윈도우 범위 관리: 현재 인덱스가 윈도우 범위를 벗어나면 리스트의 첫 번째 요소를 제거
3. 현재 요소와 비교: 현재 요소보다 작은 인덱스를 리스트에서 제거하여 최대값을 유지
4. 결과 저장: 윈도우의 첫 번째 요소가 최대값이므로 이를 결과 리스트에 추가

```java
import java.util.ArrayList;
import java.util.List;

public class SlidingWindow {
    public static void main(String[] args) {
        int[] arr = {1, 3, -1, -3, 5, 3, 6, 7};
        int k = 3;
        List<Integer> result = maxSlidingWindow(arr, k);
        System.out.println(result);  // 출력: [3, 3, 5, 5, 6, 7]
    }

    public static List<Integer> maxSlidingWindow(int[] arr, int k) {
        List<Integer> result = new ArrayList<>();
        List<Integer> window = new ArrayList<>();
```

```
        for (int i = 0; i < arr.length; i++) {
            // 윈도우 범위를 벗어난 인덱스를 제거
            if (!window.isEmpty() && window.get(0) <= i - k) {
                window.remove(0);
            }

            // 현재 요소보다 작은 인덱스를 제거
            while (!window.isEmpty() && arr[window.get(window.size() - 1)] <
arr[i]) {
                window.remove(window.size() - 1);
            }

            // 현재 인덱스를 리스트에 추가
            window.add(i);

            // 윈도우의 첫 번째 요소가 최대값
            if (i >= k - 1) {
                result.add(arr[window.get(0)]);
            }
        }

        return result;
    }
}
```

주소	문제	제한시간
https://www.acmicpc.net/problem/12891	DNA 비밀주소	2초
https://www.acmicpc.net/problem/11003	최소값 찾기 1	2.4초

#05. 스택과 큐

주소	문제	제한시간
https://www.acmicpc.net/problem/1874	스택으로 수열 만들기	2초
https://www.acmicpc.net/problem/17298	오큰수 구하기	1초
https://www.acmicpc.net/problem/2164	카드 게임	2초
https://www.acmicpc.net/problem/11286	절대값 힙 구현하기	2초