문제해결프로그래밍실습
(CSE4152)

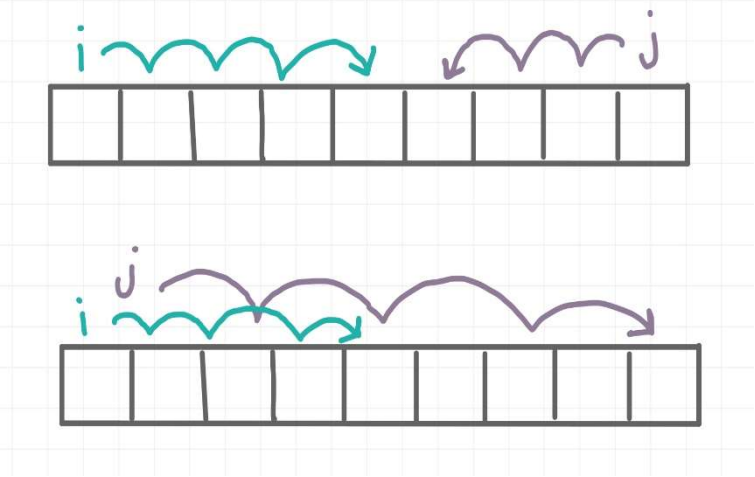
14 주차

Two Pointers
&
Sliding Window

목차

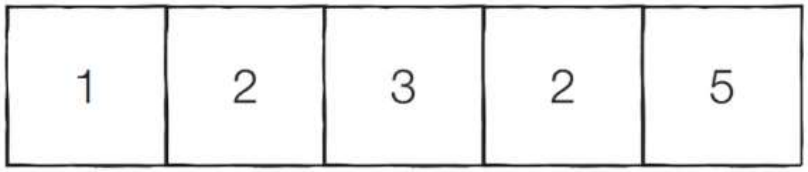
Two Pointer

- 배열 및 리스트에 순차적으로 접근해야 할 때 **두 개의 점의 위치를 기록하면서 처리**하는 알고리즘 (시간 복잡도 O(N))
- 원하는 값을 찾는데 있어서 반복문을 통한 탐색 알고리즘에 비해서 메모리 및 시간 복잡도에 있어서 효율적임
- 두 가지 방식으로 접근할 수 있는데, 시작과 끝점에서 포인터를 시작하는 방법과 빠른 포인터와 느린포인터의 개념으로 해결하는 방법이 있다. (같은 방향으로)

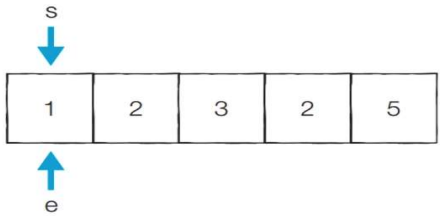


Two Pointer 동작 원리

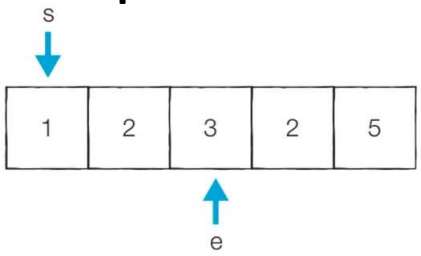
- 합이 5가 되는 부분 연속 수열 찾기



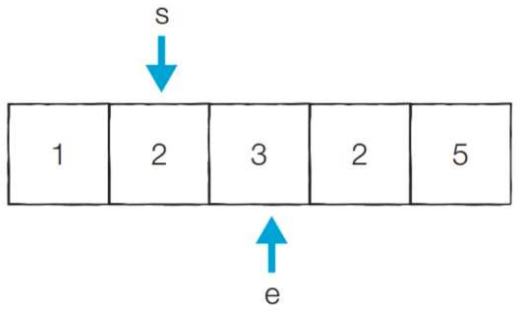
*** Step1**



*** Step2**



*** Step3**



1. 시작 포인터, 끝 포인터가 첫 인덱스 가리키게 초기화.
2. 현재 부분 합이 5와 같다면 Count ++
3. 5보다 작다면, 끝 포인터를 1 증가
4. 5보다 크다면 시작 포인터를 1 증가
5. 모든 경우 확인할 때까지 위 과정(2~4) 반복

Two Pointer

- Two Pointer를 이용할 경우 문제 접근 순서
- 1. Pointer 초기화

  문제 정의에 따라서 두 개의 포인터를 어디서 시작할지 정한다

  예) 맨 앞에서 두 개 시작, 양 쪽 끝에서 시작
- 2. Pointer 이동

  포인터 위치 초기화 후, 어떻게 각각의 포인터를 이동할 지 정한다.

  예) 양쪽 끝에서 시작한 경우, 맨 앞 포인터는 +1, 맨 끝 포인터는 -1씩 이동
- 3. 중지 조건

  모든 검색이 끝나는 조건. 예를 들면 s와 e가 모두 마지막 원소에 위치.

Two Pointer 예시1

- Reversed Array

  주어진 배열을 뒤집기

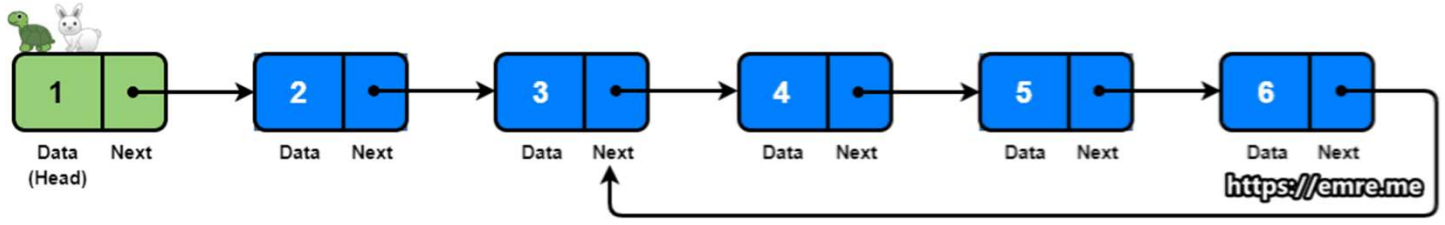
  array : [10, 20, 30, 40, 50] || answer : [50, 40, 30, 20, 10]

```
void reverseArray(vector<int>& array) {
    // 1. start, end를 배열의 양쪽 끝 포인터로 초기화
    int start = 0, end = array.size() - 1;

    // 3. 중지 조건
    while (start < end) {

        swap(array[start], array[end]);

        // 2. 포인터 이동
        start++;
        end--;
    }
}
```

Fast & Slow runner

- Linked list를 순회할 때 2개의 포인터를 동시에 사용하는 방법
- 각 포인터를 Fast runner, Slow runner라고 하며, 일반적으로 Fast runner는 2칸, Slow runner는 1칸 씩 이동하게 된다.

```cpp
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int value) : val(value), next(nullptr) {}
};

class Solution {
public:
    bool hasCycle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;
        while (fast != nullptr && fast->next != nullptr) {
            fast = fast->next->next;
            slow = slow->next;
            if (slow == fast) {
                return true;
            }
        }
        return false;
    }
};
```

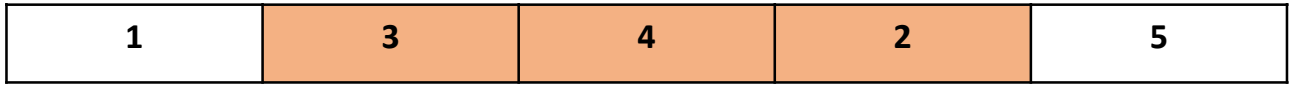


Sliding Window

- 정의 : 고정 사이즈의 윈도우를 이동시키면서 윈도우 내에 있는 데이터를 이용해서 문제를 해결 (시간 복잡도 O(N))
- Two pointer와 다르게 정렬 여부에 큰 상관이 없음.
- Two pointer는 정렬된 경우에서만 적용되는 경우가 있음.

Step 1 (window size : 3)

Step 2



Step 3

Sliding Window 예시

- 정의 : 고정 사이즈의 윈도우를 이동시키면서 윈도우 내에 있는 데이터를 이용해서 문제를 해결
- 문제 : 주어진 배열과 윈도우 사이즈를 사용해서 슬라이딩 윈도우를 했을 때 윈도우마다 최대값을 출력하기
- Array : [1, 3, -1, -3, 5, 3, 6, 7], window size : 3
- Ex) [1, 3, -1]의 경우에는 최대값 3을 출력

Sliding Window 예시

- 문제 : 주어진 배열과 윈도우 사이즈를 사용해서 슬라이딩 윈도우를 했을 때 최대값을 가지게 되는 윈도우 구하기
- Array : [1, 3, -1, -3, 5, 3, 6, 7], window size : 3

```python
def sliding_windows(arr, k):
    if not arr:
        return arr

    r = []
    for i in range(len(arr) - k + 1):
        r.append(max(arr[i:i+k]))

    return r

arr = [1, 3, -1, -3, 5, 3, 6, 7]
k = 3

print(sliding_windows(arr, k))
```

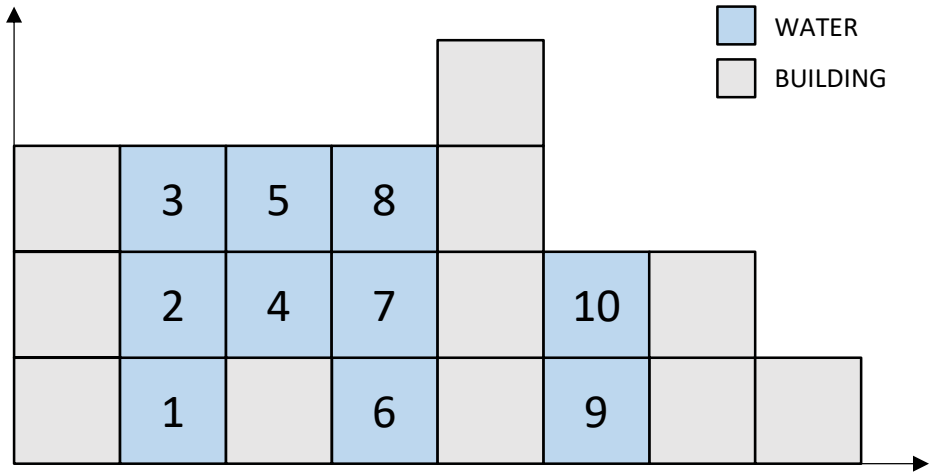
```cpp
vector<int> sliding_windows(vector<int>& arr, int k) {
    if (arr.empty() || k <= 0) {
        return {};
    }
    vector<int> result;
    deque<int> dq;
    for (int i = 0; i < arr.size(); i++) {
        if (!dq.empty() && dq.front() < i - k + 1) {
            dq.pop_front();
        }
        while (!dq.empty() && arr[dq.back()] < arr[i]) {
            dq.pop_back();
        }
        dq.push_back(i);
        if (i >= k - 1) {
            result.push_back(arr[dq.front()]);
        }
    }

    return result;
}
```

보고서 10 과제

- 앞에서 다루지 않은 문제 중에 Two pointers와 Sliding window를 사용했을 때 더 효율적으로 풀 수 있는 문제를 각각 하나씩 들고, 어떤 면에서 효율적인지 알고리즘을 설명하시오.

코딩 과제 10 – Rainwater Trapping

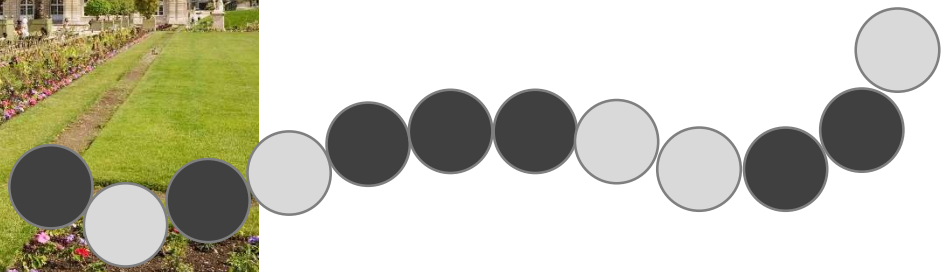
- 고도 지도는 너비가 1, 높이가 1인 블록으로 나타내어졌다.
- 물은 양쪽에 쌓여진 블록들 사이에 고인다.
- 저장된 빗물의 양은 0보다 크거나 같은 정수로 나타내어진다.
- 모든 블록 사이에서 저장된 물의 양의 총합을 구하여라.



INPUT	OUTPUT
2 8 3 0 1 0 4 0 2 1 5 3 0 2 0 4	10 7

코딩 과제 10 – Pebbles

- 정원에 검은색 혹은 흰색 돌들이 일렬로 늘어서 있다.
- 나무를 심고 싶은데, 나무와 어울리는 돌은 검은색 돌이라서, 흰색 돌 몇 개를 검은색으로 고쳐서 분위기를 맞추고 싶다.
- 그러나 여분의 검은색 돌이 K개밖에 없다.



코딩 과제 10 – Pebbles

- 나무를 심을 자리를 만들려면 가장 길게 일렬로 늘어선 검은색 돌의 개수를 최대화해야 한다.
- 최대 K개의 흰색 돌을 검은색으로 바꿔서 최대화해 보고, 바꿔야 하는 돌의 위치를 구하라.

INPUT
BWBWBBBWWBBW

OUTPUT
3 : **3개**의 돌을 바꾸면 됨
4 8 9 : **4번째, 8번째, 9번째** 돌을 바꾸면 됨