고급 소프트웨어 실습(CSE4152)

8 주차

Two Pointers
Sliding Window

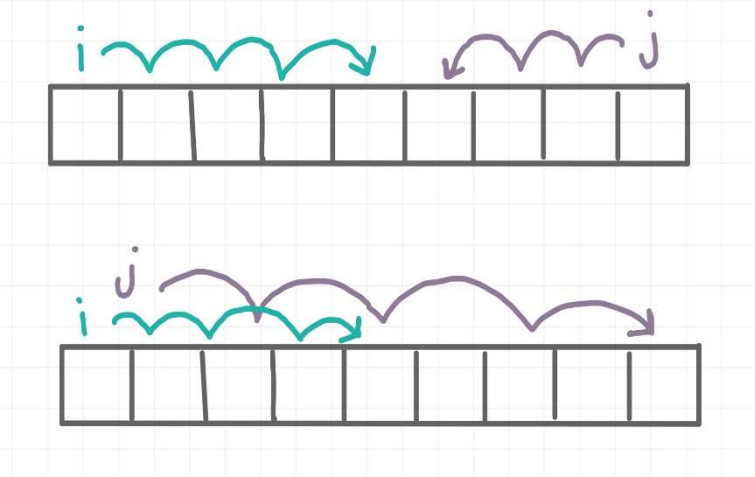
목차

실습 환경

- Google Colaboratory (Colab)
- 브라우저 내에서 Jupyter Notebook 기반의 Python 스크립트를 작성하고 실행 가능
- GPU 무료 제공
- https://colab.research.google.com/?hl=ko

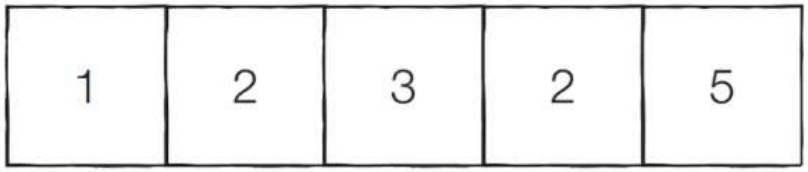
Two Pointer

- 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)

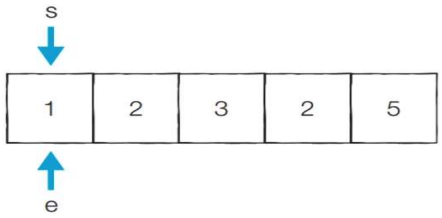


Two Pointer 동작 원리

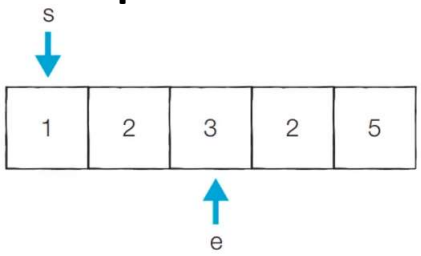
- 합이 5가 되는 부분 연속 수열 찾기



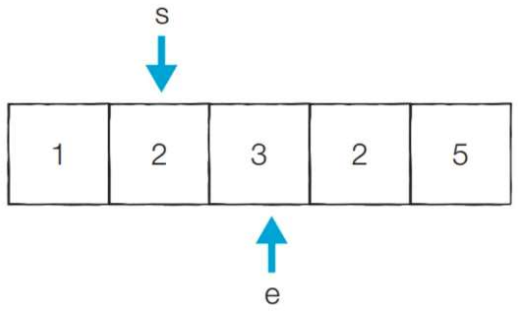
*** Step1**



*** 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, 맨앞 포인터는 +1, 맨 끝 포인터는 -1씩 이동
- 3. 중지 조건

  원하는 요소를 찾고 나서도 포인터를 이동시키면 안 되므로, 중지 조건 필수

Two Pointer 예시1

- Reversed Array

  주어진 배열을 뒤집기

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

```
def reverseArray(array):
    # 1.start, end를 배열의 양쪽 끝 포인터로 초기화
    start, end = 0, len(array)-1

    # 3. 종지 조건
    while start< end:
        # reverse array elements
        array[start], array[end] = array[end], array[start]

        # 2.포인터 이동
        start += 1
        end -= 1
```

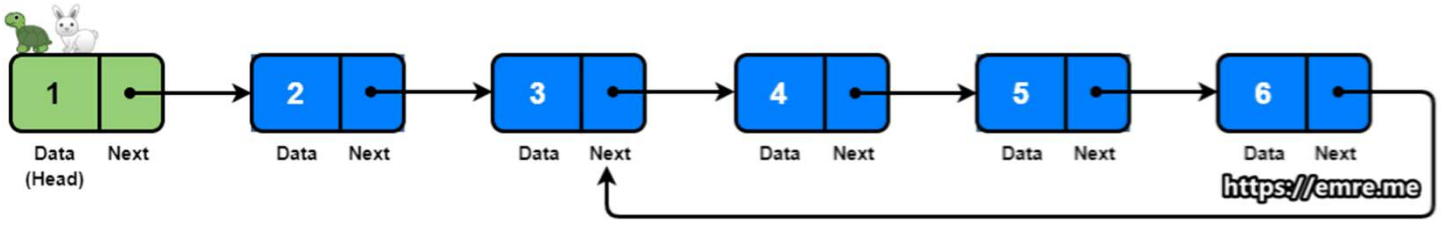
Fast & Slow runner

- 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.

```python
class ListNode:
    def __init__(self, val):
        self.val = val
        self.next = None


class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        slow, fast = head, head
        while fast is not None and fast.next is not None:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                return True

        return False
```

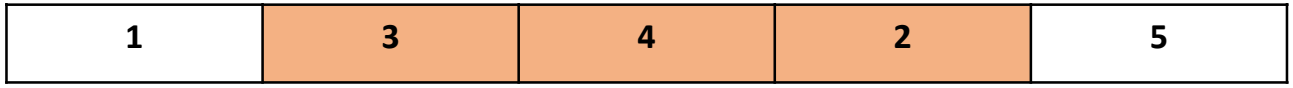


Sliding Window

- 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

```
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))
```

Two Pointer 실습 1

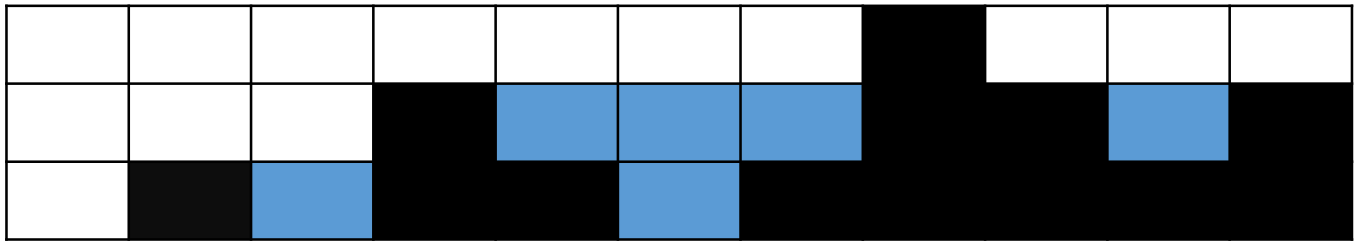
- 아래 배열의 숫자(오름차순 정렬)들 중 2개를 골라 더하였을 때 target 값을 만들 수 있는 숫자와 해당 숫자의 인덱스를 출력하시오
- [2, 7 , 11, 15], target : 9

  - 단순 반복문으로 구현했을 때의 경우

```
def sum_function(nums, target):
    for i in range(len(nums)):
        for j in range(i+1, len(nums)):
            if nums[i] + nums[j] == target:
                return [nums[i], nums[j]], [i,j]
```

Two Pointer 실습 2

- 주어진 배열 내 각 원소는 벽 높이를 뜻하고, 높이를 입력 받았을 때 벽 사이 사이에 얼마나 많은 물이 쌓일지 계산하세요. (아래 그림 참고)
- [0, 1 , 0, 2, 1, 0, 1, 3,2, 1, 2]
- Answer : 6

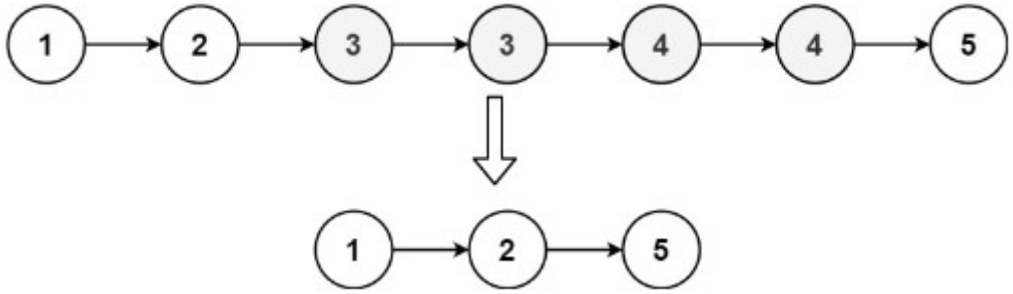


Two Pointer 실습 3

- 연결리스트(오름차순 정렬)가 아래와 같이 존재할 때, 중복된 원소의 노드를 삭제한 결과값을 출력하시오.

Input : head = [1,2,3,3,4,4,5]

Answer : [1,2,5]



Two Pointer 실습 4

- 배열이 아래와 같이 존재할 때, 중복된 원소가 최대 2개 까지만 가지는 결과값을 출력하시오. (중복된 원소는 인접해 있음)

Input : [1, 1, 2, 2, 2, 3, 3, 3]

Answer : [1, 1, 2, 2, 3, 3]

Sliding Window 실습 1

- 애너그램(다른 단어나 구문의 글자를 다시 배열해서 만든 단어나 구) 찾기
- 문자열 s1과 s2가 주어졌을 때 s2가 s1의 애너그램이 되는 시작 인덱스 값들을 반환

  s1 = “cbaebabacd”, s2 = “abc”

  answer = [0, 6]

  ex) bac, cba는 abc의 애너그램 중 일부이다.

Sliding Window 실습 2

- 문제 : 주어진 문자열에서 가장 긴 부분 문자열 찾기 (반복 글자 제외)

  Sliding Window 개념을 이용해서 풀 것

- Str : “pwwkew”
- Answer : Answer : 3(“wke”)

```python
def function(self, s: str) -> int:

    sub_str_list=[]
    ret=0
    for cursor in range(len(s)):
        count_dict={s[cursor]:1}
        for move in range(cursor+1,len(s)):
            if count_dict.get(s[move],False):
                sub_str_list.append(''.join(count_dict.keys()))
                break
            else:
                count_dict[s[move]]=1

        sub_str_list.append(''.join(count_dict.keys()))



    if len(sub_str_list) !=0:
        ret=len(max(sub_str_list,key=lambda x: len(x)))

    return ret
```

*슬라이딩 윈도우를 쓰지 않고 풀었을 때

Sliding Window 실습 3

- Moving average
- 숫자 리스트와 윈도우 크기가 주어졌을 때, 윈도우를 이동시키며 평균값을 계산
- 효율적인 평균 값 계산을 위하여, 윈도우 이동시 제외되는 수와 추가되는 수를 이용하여 현재 평균값을 업데이트
- 아래와 같은 Sub function 구현

  new_avg = update_avg(cur_avg, old, new)

- ex)

  Input = [1,2,3,2,4,5,4,6], 5

  output = 2.4 3.2 3.6 4.2

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.