Median of two sorted Arrays

다시 풀어보기	☑
링크	https://leetcode.com/problems/median-of-two-sorted-arrays/
생성일	@2020년 12월 28일 오후 8:46
소요시간(분)	
유형	Binary Search
출처	Leet Code

문제

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

Follow up: The overall run time complexity should be `O(log (m+n))`.

문제 접근 방법

▼ **O(m+n) 풀이**

- nums1 배열과 nums2 배열의 원소를 앞에서 부터 하나씩 꺼낸다
- 각각의 원소의 크기를 비교하여 오름차순으로 새로운 배열 nums3에 저장한다
- 정렬된 배열 nums3에서 중앙값을 구한다.

▼ **O(log(m+n)) 풀이**

사용 알고리즘

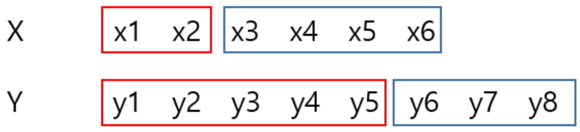
- Binary Search

아이디어

먼저 두 배열의 길이의 합이 짝수인 경우 먼저 살펴보자

X	x1	x2	x3	x4	x5	x6		
Y	y1	y2	y3	y4	y5	y6	y7	y8

위와 같이 오름차순으로 정렬되어 있는 두 배열이 주어졌을 때, 두 배열을 적당한 크기로 나눈다



이때 배열의 요소의 개수가 일치해야한다. 즉, 빨간색 박스를 씌운 요소들과 파란색 박스를 씌운 요소들의 개수가 서로 똑같도록 나누어 준다

각각 오름차순으로 정렬된 배열들이고 , x2와 y5는 좌측(빨간색 박스)에서 가장 큰 값일 것이다.

$$x2 \leq y6$$
$$y5 \leq x3$$

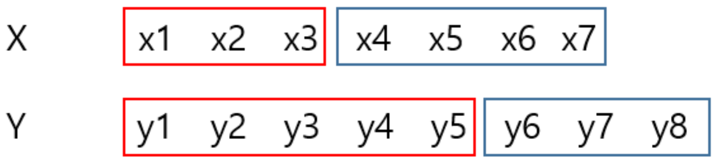
이때 위과 같은 가정을 하고 **두 조건 모두 참**이라면 **빨간박스 요소들은 파란박스 요소들보다 무조건 작거나 같을 것**이다.

그렇다면 중앙값은 x2, y6, y5, y3을 이용하여 $AVG(Max(x2, y5) + Min(x3, y6))$ 를 통해 구할 수 있다.

다시말해 빨간 박스에서 가장 큰값을 $Max(x2, y5)$ 연산으로 구하고 파란 박스에서 가장 작은 값을 $Min(x3, y6)$ 연산을 통해 구한 뒤

두 수의 평균을 구하면 그 값이 X배열과 Y배열의 중앙값이 된다

홀수의 경우는 더 간단하다



마찬가지로 배열을 적당한 크기로 나누었을 때 (홀수의 경우 작은 값들의 집합이 큰값들의 집합보다 하나 더 크다)

위의 조건을 만족한다면 중앙값은 $Max(x3, y5)$가 된다.

Binary Search(BS) 풀이

위 아이디어를 이해했다면 그래서 ***어떻게 배열을 적당한 크기로 나눌 것인가 ?*** 에 대한 궁금증이 생긴다.

배열을 적당한 크기로 나누는 방법은 **Binary Search(BS)를 사용하여 적정 경계값을 구할 것이다.**

BS를 시작하기 전 4가지의 변수와 한가지 간단한 공식에 대해서 유념해야한다.

- $start$: BS를 시작할 인덱스
- end : BS를 끝낼 인덱스
- $partitionX$: X배열을 작은 값들의 집합(빨간 박스)과 큰 값들의 집합(파란 박스)으로 나눌 경계값
- $partitionY$: Y배열을 나눌 경계값
- $partitionX + partitionY = (x + y + 1)/2$: x와 y는 X와 Y배열의 길이 (+1을 하는 이유는 홀수와 짝수 두 경우 모두 연산을 잘 수행하기 위함)

위와 같이 정의한 후 BS를 통해 적정 partion point를 정하고 우리가 정한 가정이 맞는지를 확인한다.

이때 가정이 맞지 않는 경우 start 또는 end 인덱스를 왼쪽이나 오른쪽으로 한칸씩 옮긴다.

또한, X 배열과 Y 배열 중 partion X를 가지는 배열은 두 배열 중 길이가 작은 배열로 선정한다

예제 1

배열의 총 요소의 개수가 홀수인 경우

$X = [1, 3, 8, 9, 15]$

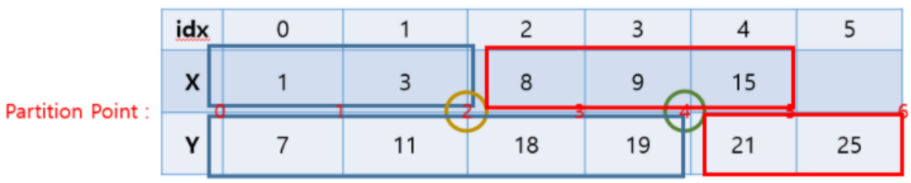
$Y = [7, 11, 18, 19, 21, 25]$

위와 같은 배열이 주어졌을 때 먼저 partitionX를 구한다.

partitionX는 $(start + end)/2$ 로 구하며 index이기 때문에 소수점은 버리고 정수형으로 취한다. (BS의 mid값과 대응)

그리고 partitionY를 구하는 공식에 대입하여 partitionY값 까지 구한다 ($partitionY = (x + y + 1)/2 - partitionX$)

그러면 배열은 다음과 같이 나눠진다.



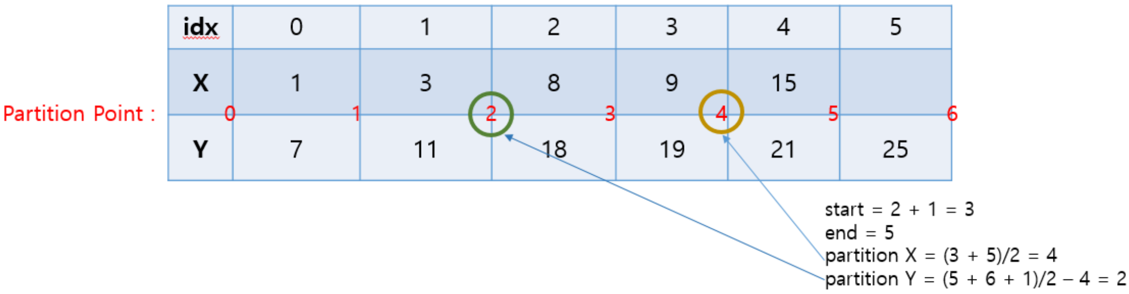
이제 아까세운 가정으로 비교를 해보면

X배열 좌측의 최대값은 3 이고 Y 배열 우측의 최소값은 21 이다. 3은 21보다 작으므로 **맞다**.

Y배열 좌측의 최대값은 19 이고 X 배열 우측의 최소값은 8 이다. 19는 8보다 작지 않으므로 **틀리다**

이 경우 (maxLeftY > minRightX) X 배열에서 partition X가 너무 좌측에 둔것이므로 우측으로 이동시켜 줘야한다.

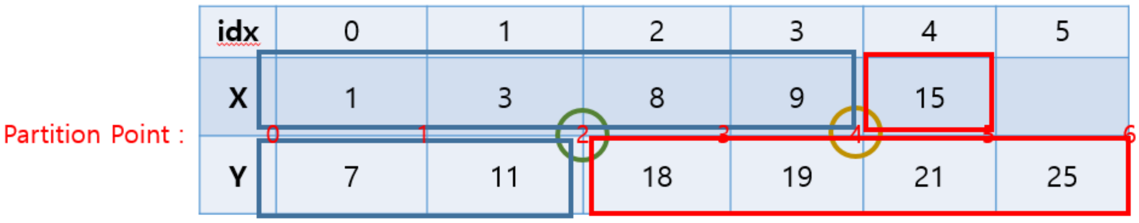
그러므로 start에 partitionX에 1을 더한 값을 start에 넣어주고 다시 검색을 시작한다



따라서, start = partitionX(2) +1가 되어 3이 되고 end는 변화가 없다

partitionX는 start와 end의 평균값으로 구하고 partitionY는 partitionX값을 토대로 구한다

그러면 아래와 같이 새로 배열이 나눠진다



다시 아까 세운 가정으로 비교를 해보면

X배열 좌측의 최대값은 9 이고 Y 배열 우측의 최소값은 18 이다. 9는 18보다 작으므로 **맞다**.

Y배열 좌측의 최대값은 11 이고 X 배열 우측의 최소값은 15 이다. 11은 15보다 작으므로 **맞다.**

이제 **적당하게 잘** 배열을 나눴으므로 중앙값을 구할 수 있다.

배열의 총 길이가 11로 홀수이므로 $Max(maxLeftX, maxLeftY)$로 정답은 11이 된다.

예제 2

다음은 배열의 총 요소의 개수가 짝수인 경우이고 처리가 까다로운 Edge case로 예제를 볼것이다

$$X = [23, 26, 31, 35]$$

$$Y = [3, 5, 7, 9, 11, 16]$$

두 배열을 합치면 총 10개이며 두 배열을 합치면 [3, 5, 7, 9, 11, 16, 23, 26, 31, 35] 순으로 정렬이 된다

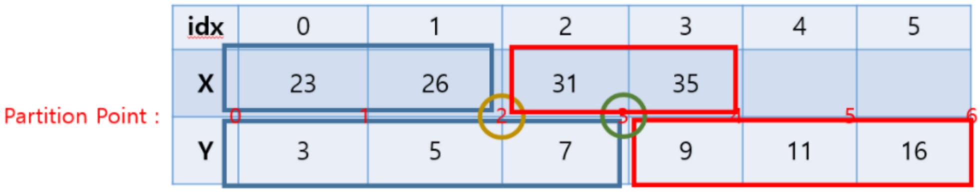
이 때의 중간값은 11과 16의 평균값인 13.5가 된다. 이를 Binary Search 를 이용하여 구해보자

X , Y 중 더 짧은 배열이 partitionX를 배정받을 배열이므로 여기선 X 배열을 이용한다

start = 0, end = 4

partitionX = (0 + 4) / 2 = 2

partitionY = (4 + 6 + 1) / 2 - partitionX = 3



X배열 좌측의 최대값은 26 이고 Y 배열 우측의 최소값은 9 이다. 26는 9보다 작지 않으므로 **틀리다**.

Y배열 좌측의 최대값은 7 이고 X 배열 우측의 최소값은 31 이다. 7은 31보다 작으므로 **맞다.**

이 경우 (maxLeftX > minRightY) X 배열에서 partitionX가 너무 우측에 둔것이므로 이를 좌측이로 이동시켜줘야 한다.

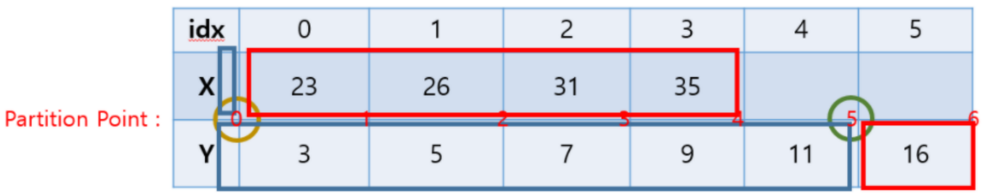
따라서 partitionX에서 -1한 값을 end 값에 대입해주어 좌측 구간으로 이동하도록 한다.

start = 0 , end = 2 - 1 = 1

partitionX = (0 + 1) / 2 = 0

partitionY = (4 + 6 + 1) / 2 - 0 = 5

계산한 값으로 다시 나눠보면 아래와 같다



이 경우 partitionX가 0이 되어 작은 값들의 집합 빈 집합이 되어버린다.

빈 집합이 될 경우 아까 세운 가정을 검증해보지 못하므로, 무한대 값을 하나 임의로 넣어준다

만약 좌측 배열이 빌 경우 - INFINITY , 우측 배열이 빌 경우 +INFINITY

이제 아까 세운 가정으로 비교를 해보면

X배열 좌측의 최대값은 -Infinity 이고 Y 배열 우측의 최소값은 16 이다. -Infinity는 16보다 작으므로 **맞다**.

Y배열 좌측의 최대값은 11 이고 X 배열 우측의 최소값은 23 이다. 11은 23보다 작으므로 **맞다.**

이제 올바르게 배열을 나눴으므로 중앙값을 구할 수 있다.

짝수의 경우 $(Max(-Infinity, 11) + Min(23, 16))/2 = 11 + 16 = 13.5$ 로 중앙값은 13.5가 된다.

소스 코드

```java
import java.util.*;

public class MedianOfTwoSortedArrays {
    static private final int INFINITY = Integer.MAX_VALUE;

    /**
     * Time complexity : O(log(m+n))
     * */
    public static double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int m = nums1.length;
        int n = nums2.length;

        // 길이가 더 작은 배열이 nums1이 될 수 있게 함
        if (m > n)
            return findMedianSortedArrays(nums2, nums1);

        double answer = 0.0;

        // 항상 m <= n
        int start = 0, end = m, halfLen = (m + n + 1) / 2;

        while (start <= end) {
            int partitionX = (start + end) / 2;
            int partitionY = halfLen - partitionX;

            int maxLeftX = (partitionX == 0) ? -INFINITY : nums1[partitionX - 1];
            int minRightX = (partitionX == m) ? INFINITY : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? -INFINITY : nums2[partitionY - 1];
            int minRightY = (partitionY == n) ? INFINITY : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((m + n) % 2 != 0) {
                    answer = Math.max(maxLeftX, maxLeftY);
                } else {
                    answer = (double)(Math.max(maxLeftX, maxLeftY) + Math.min(minRightX, minRightY)) / 2;
                }
                break;
            } else if (maxLeftX > minRightY) {
                end = partitionX - 1;
            } else {  //maxLeftY > minRightX
                start = partitionX + 1;
            }
        }
        return answer;
    }

    /**
     * Time complexity : O(m+n)
     * */
    public double Legacy__FindMedianSortedArrays(int[] nums1, int[] nums2) {
        List<Integer> nums3 = new ArrayList<>();
```

```
        int i = 0, j = 0;
        while (i < nums1.length && j < nums2.length) {
            if (nums1[i] <= nums2[j]) {
                nums3.add(nums1[i++]);
            } else {
                nums3.add(nums2[j++]);
            }
        }

        while (i < nums1.length)
            nums3.add(nums1[i++]);

        while (j < nums2.length)
            nums3.add(nums2[j++]);

        if (nums3.size() % 2 == 0) {
            return (double) (nums3.get(nums3.size() / 2) + nums3.get((nums3.size() / 2 - 1))) / 2;
        } else {
            return nums3.get(nums3.size() / 2);
        }
    }
}
```

마무리

시간 복잡도 O(n+m) 풀이는 쉽게 생각해낼 수 있는 풀이법이지만

O(log(m+n)) 풀이는 코드를 봐도 이해가 잘 되지 않았다.

특히 Binary Search에서 start나 end의 범위를 줄여야할 때, 해당 조건들이 확 와닿지 않는다..