두 개 합 (TwoSum)
일일 온도 (Daily Temperature)
서브어레이 최대값 (Maximum Subarray)
그룹 아나그램 (Group Anagrams)
빗물 담기 (Trapping Rain Water)
누락 범위 (Missing Ranges)
나선형 매트릭스 (SpiralMatrix)

Array (시험에 나오는거 위주로)

1. 개념

1) Math.max(max, a) => max 구하기
2) Map+ Array => two sum 문제
3) array+ stack => Daily Temperature
4) sum = sum+ nums[] => sum += nums[]

두개 합(TwoSum)

설명

정수의 배열 nums와 정수 target이 주어집니다.
배열 nums에서 두 숫자의 값을 더하여 target값과 동일할 경우 두 숫자의 인덱스를 리턴합니다

각 입력에 정확히 하나의 솔루션 이 있다고 가정하며, 동일한 요소를 두 번 사용할 수 없습니다 .

Note) 시간복잡도O(n)으로 수행하세요

입출력

Input: nums = [2, 8, 11, 14], target= 16
Output: [1,4]
Because nums[0]+nums[3] = 2 + 14 = 16
return [1, 4].

Input: nums = [3,2,4], target = 6
Output: [2,3]

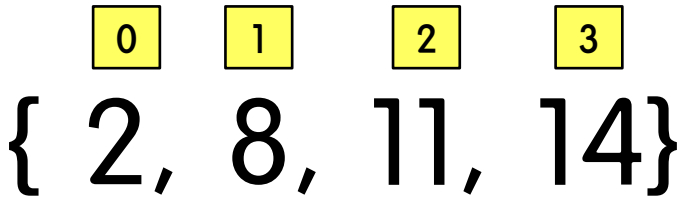
문제 Format

```
class Solution {
  public int[] solve(int[] nums, int target) { }
}
```

제한사항

$2 <= nums.length <= 10^4$
$-10^9 <= nums[i] <= 10^9$
$-10^9 <= target <= 10^9$

문제분석



1. for돌려서 target과 비교
2. 16-2=11
3. Map(숫자, 방번호)
4. 방번호만 리턴한다 int[]

문제분석

Array+Map

0 1 2 3

1 { 2, 8, 11, 14}

2 Int target =16

Key	Value
14 (16-2)	방번호 0
8 (16-8)	방번호 1
5 (16-11)	방번호 2

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(N)
   대상 : int[] nums
   이유 : for문 실행 0 -> len 까지

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(n)
   대상 : Map<Integer, Integer> map = new HashMap<>();
   이유 : for문 실행하면서 map.put

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

일일온도(Daily Temperature)

설명

일일 온도를 나타내는 int 배열(temperatures)이 주어집니다.
더 따뜻한 날씨의 날을 얻기 위해 해당날짜 이후에 기다려야하는 날짜의 수를 배열로 리턴하세요 .
더 따뜻한 날이 오지 않는다면 0을 리턴하세요.

입출력

Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]

Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]

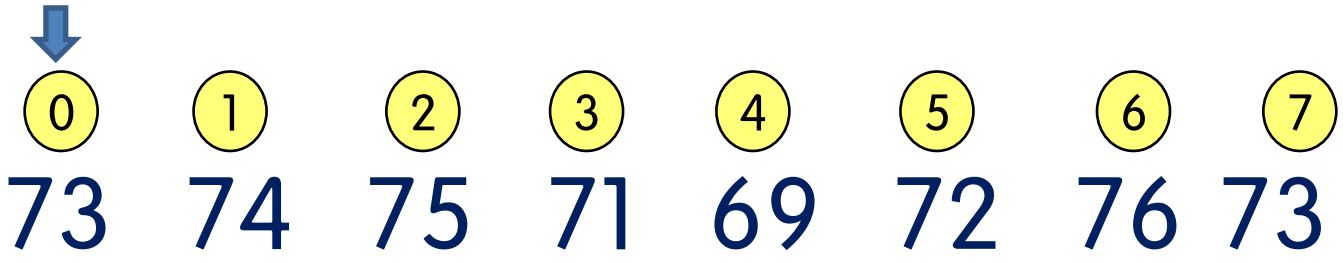
문제 Format

```
class Solution {
 public int[] solve(int[] nums) { }
}
```

제한사항

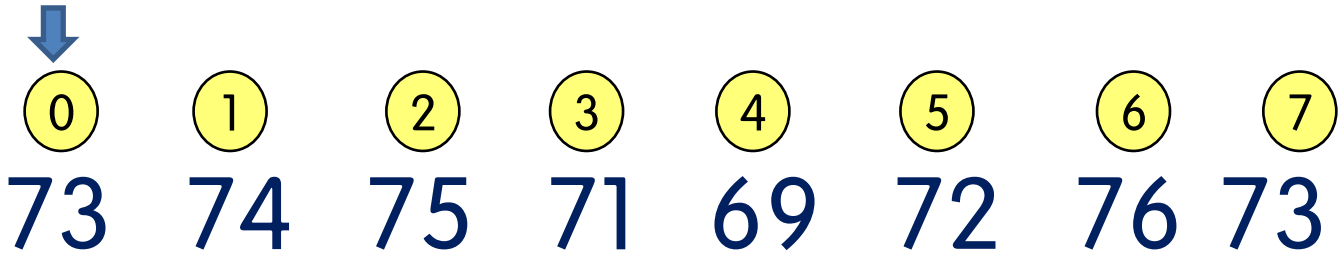
1 <= temperatures.length <= 10^5
30 <= temperatures[i] <= 100

문제분석



1. 73에서는 1일 후(74)
2. 74에서는 1일 후 (75)
3. 75에서는 4일 후(76)
4. 71에서는 2일 후(72)
5. 69에서는 1일 후(72)
6. 72에서는 1일 후(76)
7. 76, 73에서는 없음으로 0

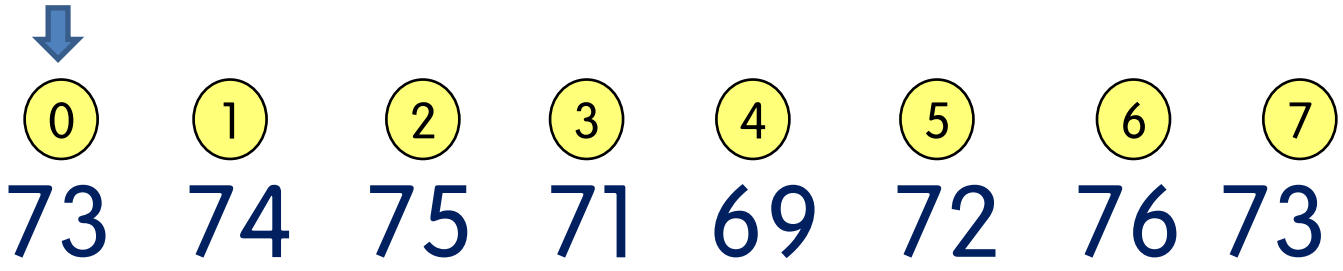
문제분석



1. for문을 돌린다
2. 73이 나온 상태에서 74를 만나야 되는 상황
3. 73과74를 비교해서 73<74이니까, 인덱스의차이 1을 결과값에 저장

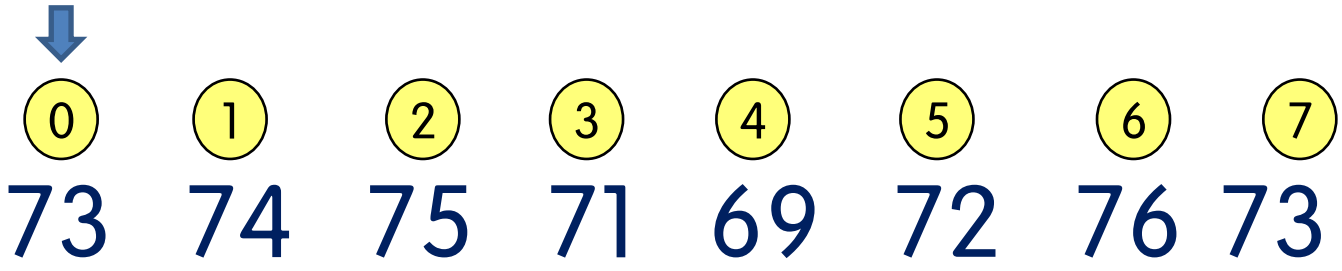
1. for문을 돌린다
2. 75가 나온 시점에, 71,69,72, 76를 만나야 되는 상황
3. 75과76를 비교해서 75<76이니까, 인덱스의차이 4 결과값에 저장

코딩화



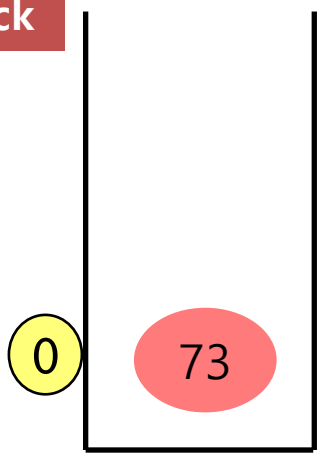
1. for문을 돌린다
2. 75가 나온 시점에, 71,69,72, 76를 만나야 되는 상황
   75보다 큰게 나올때까지 While문을 실행
   while문을 빠져나오는 시점, max-i가 됨

코딩화



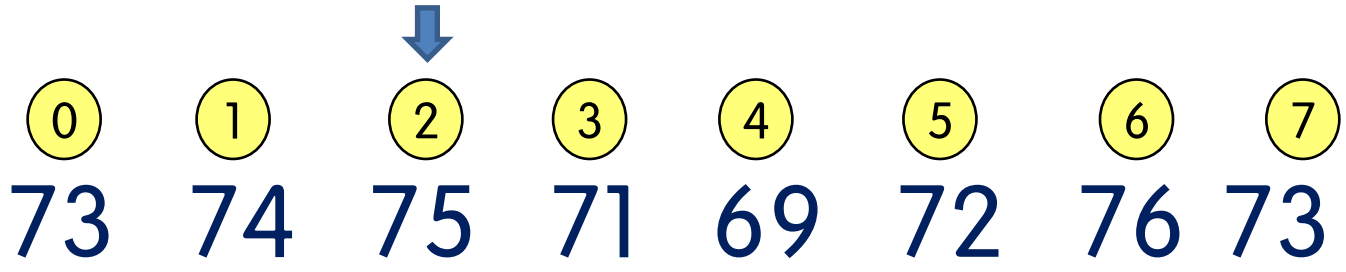
stack

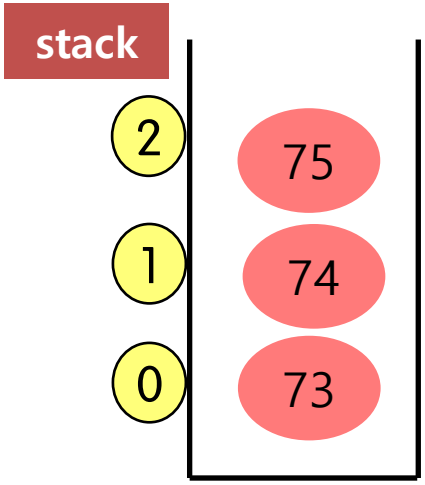
74



1. for문 돌리면서 0번방부터 스택에 저장하다가
2. 1번방을 만나서 비교 73<74 을 만나는 시점에
3. 인덱스를 이용(1-0) 1을 결과에 저장

코딩화





76

1. for문 돌리면서 0번방부터 스택에 저장하다가
2. 6번방을 만나서 비교 75<76 을 만나는 시점에
3. 인덱스를 이용(6-2) 1을 결과에 저장

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(N) 또는 Worst Case O(N^2)
   대상 : int[] temperatures
   이유 : for문 실행 0 -> len 까지, inner for 문 0+1 -> len-1 까지

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(n)
   대상 : int[] result = new int[len];
   이유 : for문 실행.

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

시간복잡도/공간복잡도 계산(stack 사용인경우)

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
    Time Complexity : O(N)
    대상 : int[] temperatures
    이유 : for문 실행 0 -> len 까지, while문에서는 스택에 있는값과 바로 찾을수도 있습니다.최악의 경우는 인덱스 끝까지 가는 경우도 발생하지만 이경우는 O(n+n)의 값 그래서 결론은 O(n)입니다.

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
    예) 프로그램을 실행 및 완료하는데 필요한 저장공간
    Space Complexity : O(n)
    대상 : int[] stack= new int[len];
    이유 : for문 실행.

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

SubArray 최대값(Maximum Subarray)

설명

정수 배열 nums가 주어지면 합계가 가장 큰 연속 하위 배열 (최소한 하나의 숫자 포함)을 찾아서 합계를 리턴합니다.

Note) 시간복잡도 O(n)으로 구하세요.

입출력

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1]

Input: nums = [5,4,-1,7,8]
Output: 23

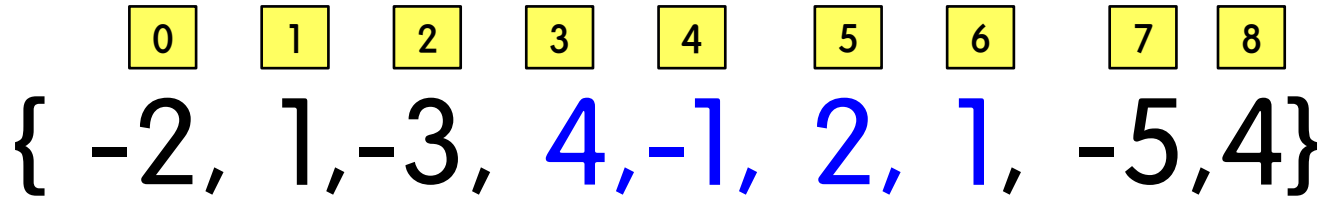
문제 Format

```
class Solution {
 public int solve(int[] nums) { }
}
```

제한사항

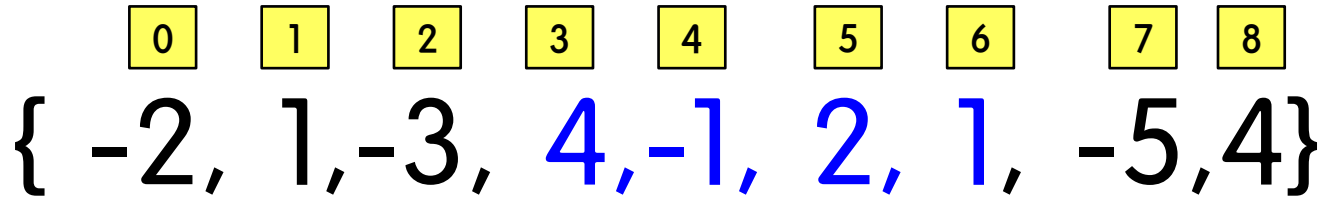
$1 <= nums.length <= 3 * 10^4$
$-10^5 <= nums[i] <= 10^5$

문제분석



1. 0번방: -2는 현재까지 제일 큰값
2. 1번방: -2+1 한값과, 1을 비교 => 그냥 1을 유지, 앞에서부터 Sum을 한것보다 크면 선택
3. 2번방: -3+1 한값과, -3을 비교
4. 3번방: 4,-1,2,1 sum을 한게 제일 크다, -5가 들어오는 순간 작아지므로
   전체를 누적한값을 가지고 비교한다

코딩화



1. 0번방: -2는 현재까지 제일 큰값
2. 1번방: -2+1 한값과, 1을 비교 => 그냥 1을 유지, 앞에서부터 Sum을 한것보다 크면 선택
3. 2번방: -3+1 한값과, -3을 비교
4. 3번방: 4,-1,2,1 sum을 한게 제일 크다, -5가 들어오는 순간 작아지므로
   전체를 누적한값을 가지고 비교한다

1. Max(처음부터 누적한값 , 현재값)
2. Max(1번에서 구한값, 전체 누적값)

```
1. curMax = Math.max(nums[i], curMax+nums[i]);
2. allMax = Math.max(allMax, curMax);
```

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(N)
   대상 : int[] nums
   이유 : for문 실행.

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(1)
   대상 : curMax, allMax
   이유 : 다른 저장공간 사용안함

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

그룹 아나그램(Group Anagrams)

설명

String 배열이 주어집니다. 주어진 String은 배열안에서 다른 String과 아나그램 관계입니다.
String 순서 상관없이 같은 아나그램을 리턴하세요

아나그램이란? 문자의 단어를 재배열하여 새로운 문자를 형성하는 것입니다.
즉, 같은 알파벳으로 구성된 단어끼리 묶어 출력하는 문제입니다

입출력

Input: strs =
["eat","tea","tan","ate","nat","bat"]
Output:
[["bat"],["nat","tan"],["ate","eat","tea"]]

Input: strs = [""]
Output: [[""]]

문제 Format

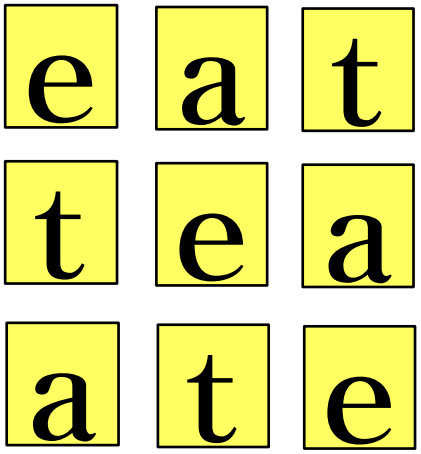
```
class Solution {
 public List<List<String>> solve(String[] strs) { }
}
```

제한사항

$1 <= strs.length <= 10^4$
$0 <= strs[i].length <= 100$
strs[i]은 영어소문자로만 구성되었습니다.

문제분석

Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]



문제분석

Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Key	Value
aet	["ate","eat","tea"]
ant	["nat","tan"]
abt	["bat"]

코딩화

Key	Value
aet	["ate","eat","tea"]
ant	["nat","tan"]
abt	["bat"]

1. 키값은 고유하게
for loop를 이용해서 한 개의 string을 뺀후 toCharArray를 이용해서 sort 후 키로 이용

2. Map을 이용해서 key, value로 넣는다.

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(NKlogK)
   대상 : String[] strs
   이유 : strs배열의 크기 n개, str의 length() K개 , 소팅실행 logk

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(NK)
   대상 : Map<String, List<String>> map = new HashMap<>();
   이유 : strs배열의 크기 n개, str의 length() K개

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

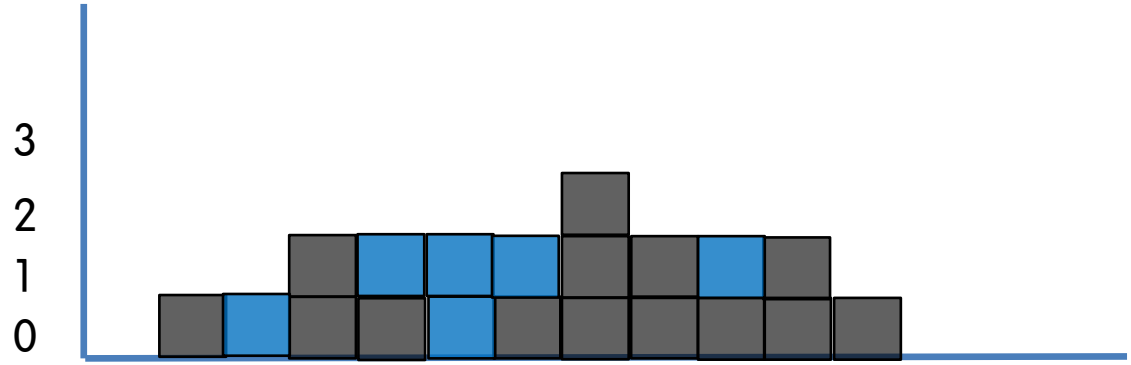
빗물 담기(Trapping Rain Water)

설명

elevation map(양의 정수)은 아래그림에서 검은 네모모양이며 너비가 1이다
비가 내린 후 가둘 수있는 물의 양을 계산합니다.

입출력

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: elevation map (검은부분) 표시 부분 [0,1,0,2,1,0,1,3,2,1,2,1].
빗물부분 (파란색 부분)은 6개 unit임



빗물 담기(Trapping Rain Water)

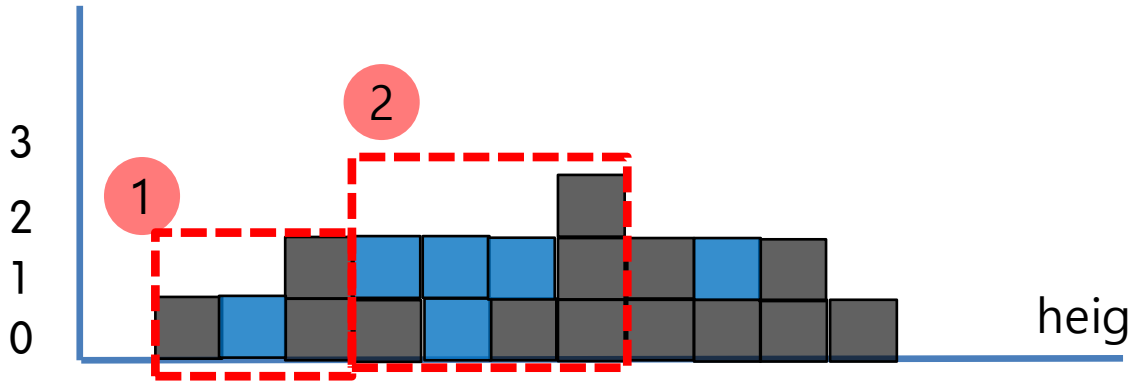
문제 Format

```
class Solution {
    public int solve(int[] height) {}
}
```

제한사항

n == height.length
0 <= n <= $3 * 10^4$
0 <= height[i] <= 10^5

문제분석

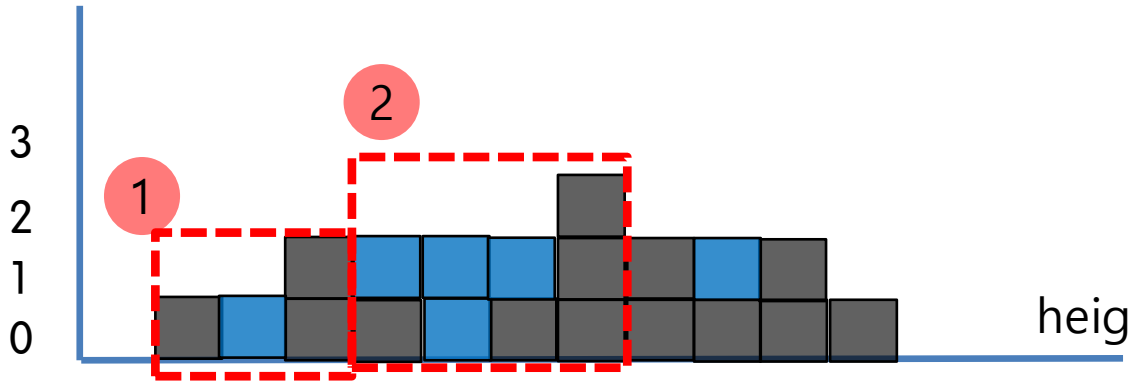


height = [0,1,0,2,1,0,1,3,2,1,2,1]

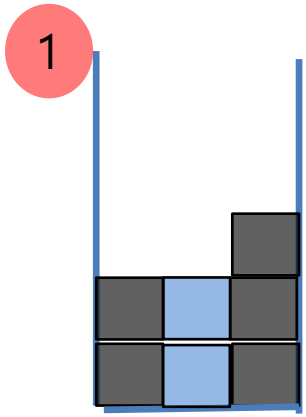
1. 물이 차는 영역 결정하는 부분
   물이 찬다는것은 왼쪽벽과 오른쪽 벽의 값의 차이가 존재해야 된다
   밑에 높이만큼 빼야 된다

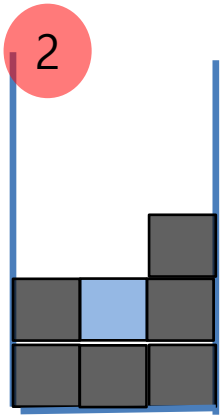
   왼쪽벽 1, 오른쪽벽 2일때 =1
   왼쪽벽 2, 오른쪽벽 3이라면 = 2
   left, right, height를 이용하여 값을 구할 수 있다

문제분석



height = [0,1,0,2,1,0,1,3,2,1,2,1]

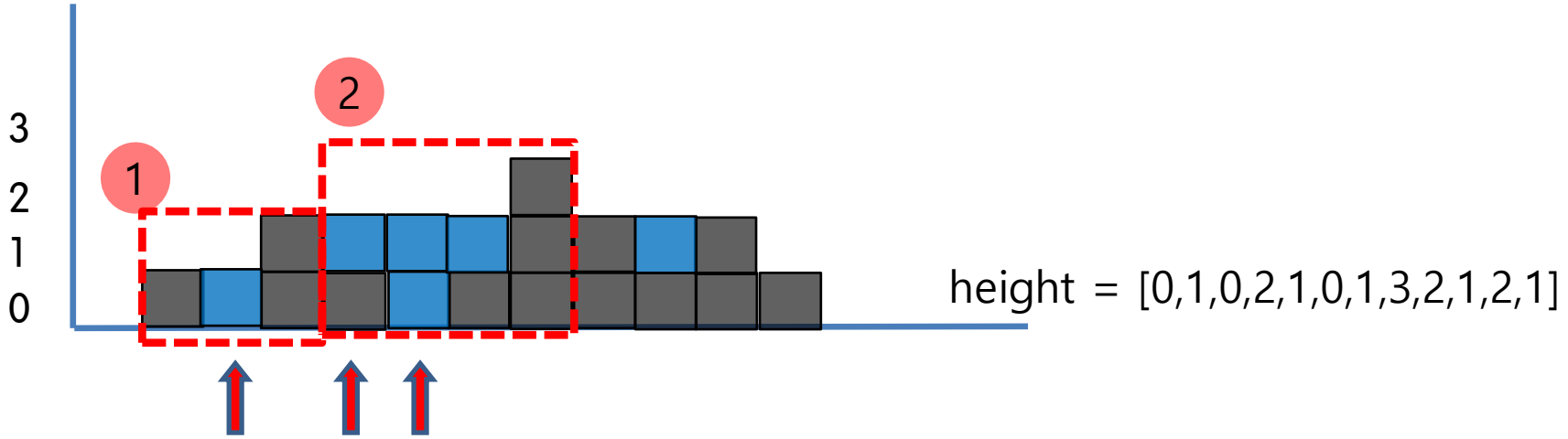




설명
1) 물을 부었을때 높이 = 2
왼쪽벽 2, 오른쪽벽 3
작은값 2

설명
2) 물을 부었을때 높이= 1
왼쪽벽 2, 오른쪽벽3
작은값 2 - 자체높이1=1

문제분석

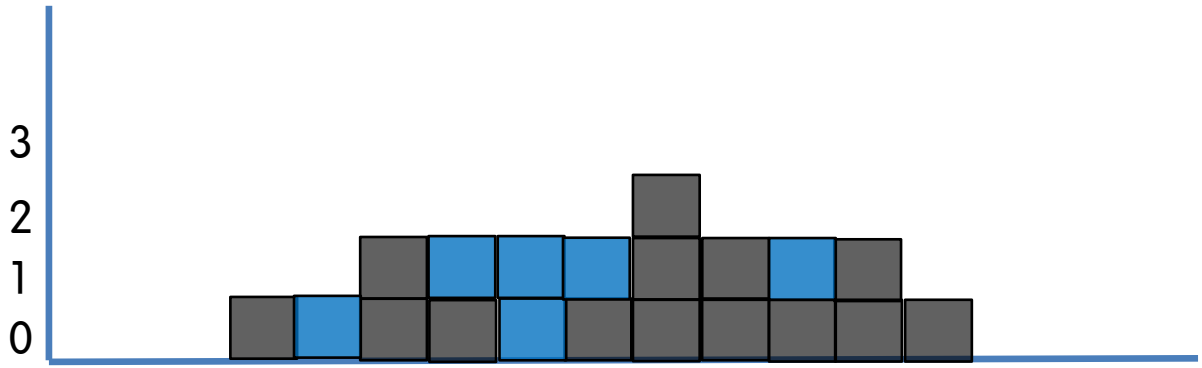


left==1, right==3, height==0
를 이용하여 값을 구할 수 있다
Math.min(1, 3)-0 =1

left==2, right==3, height==1
를 이용하여 값을 구할 수 있다
Math.min(2, 3)-1 =1

left==2, right==3, height==0
를 이용하여 값을 구할 수 있다
Math.min(2, 3)-0 =2

코딩화



1. int[ ] left: {0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3}
2. Int[ ] right: {3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 1}
3. Math.min : {0 1, 1, 2, 2, 2 2 3 2 2 ,2, 1}
4. Height : {0, 1, 0, 2, 1, 0, 1, 3, 2 ,1, 2, 1}
5. 0, 0 1 0, 1, 2, 1, 0, 0, 1, 0, 0

Math.min(left[i],right[i])-height[i];

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(N)
   대상 : int[] height
   이유 : height 배열의 크기 n개, for문 실행

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(N)
   대상 : int[] left , int[] right
   이유 : for문 실행

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

누락된 범위(Missing Ranges)

설명

모든 요소를 포함하는 범위[lower, upper]와 정렬된 고유한 정수 배열 nums와 주어집니다.
만약에 x라는 number가 범위[lower, upper]존재하고 nums배열에 없다면
누락된 값으로 간주합니다.
누락된 모든 숫자를 정확히 포함하는 가장 작은 정렬된 범위를 리턴합니다.
표현형식은 아래와 같습니다.

[a,b]목록의 각 범위 는 다음과 같이 출력되어야합니다.
"a->b" if a != b
"a" if a == b

입출력

Input: nums = [2,3,5,50,75], lower = 0, upper = 99
Output: [0->1, 4, 6->49, 51->74, 76->99]
Explanation:
전체 범위가 0-99 입니다.
nums[0]==2입니다. 0,1 빠졌으므로 0->1로표시

누락된 범위(Missing Ranges)

입출력

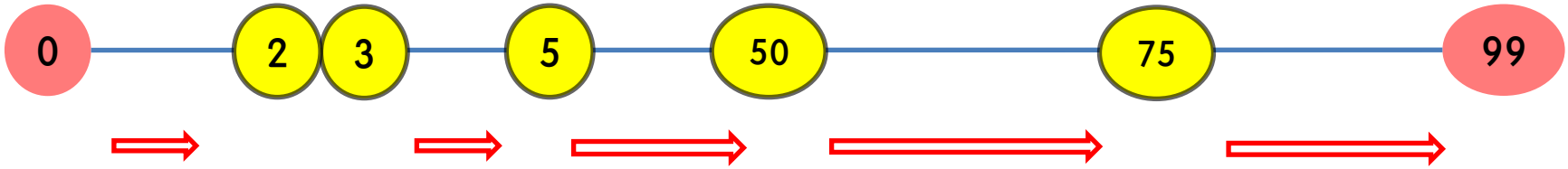
Input: nums = [], lower = 1, upper = 1
Output: ["1"]

제한사항

-10^9 <= lower <= upper <= 10^9
0 <= nums.length <= 100
lower <= nums[i] <= upper
All the values of nums are **unique**.

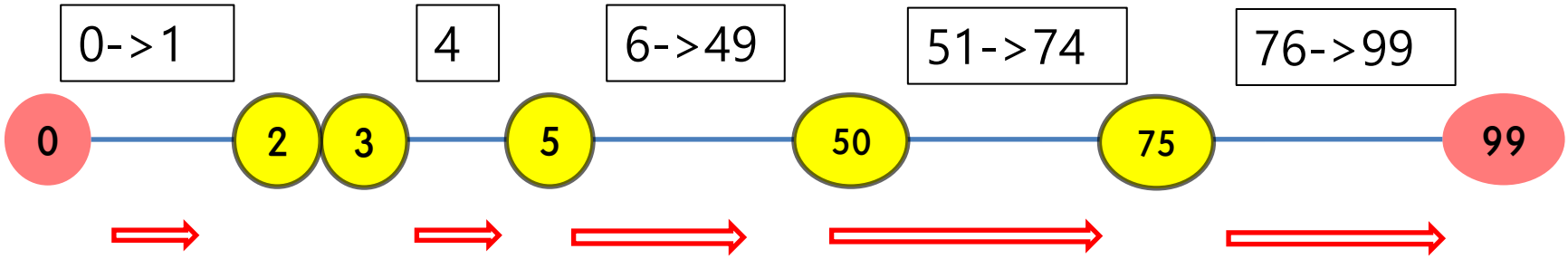
문제분석

Input: nums = [2,3,5,50,75], lower = 0, upper = 99
Output: [0->1, 4, 6->49, 51->74, 76->99]
Explanation:
전체 범위가 0-99 입니다.
nums[0]==2입니다. 0,1 빠졌으므로 0->1로표시



[0->1, 4, 6->49, 51->74, 76->99]

문제분석



1. 0->1
   lower< nums[0]

2. 3-5와 5-50
   nums[i]+1<nums[i+1]

3. 75-99
   nums[nums.length-1]<upper

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(N)
   대상 : int[] nums
   이유 : nums 배열의 크기 n개, for문 실행

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(N)
   대상 : List<String> result = new ArrayList<>();
   이유 : for문 실행

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

나선형매트릭스(SpiralMatrix)

설명

Given a matrix of m x n elements (m rows, n columns),
return all elements of the matrix in spiral order.

입출력

Input:
int[][] matrix = {
{ 1, 2, 3, 4},
{ 5, 6, 7, 8},
{ 9 ,10, 11, 12}
};
출력
[1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]

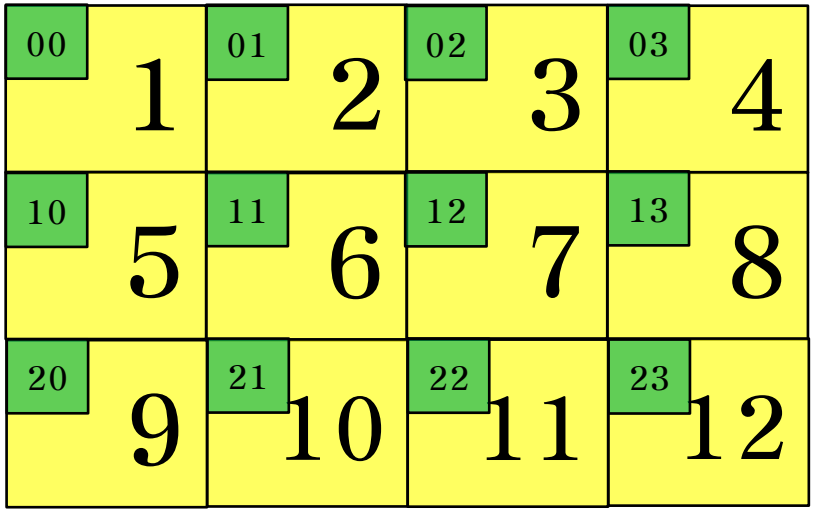
제한사항

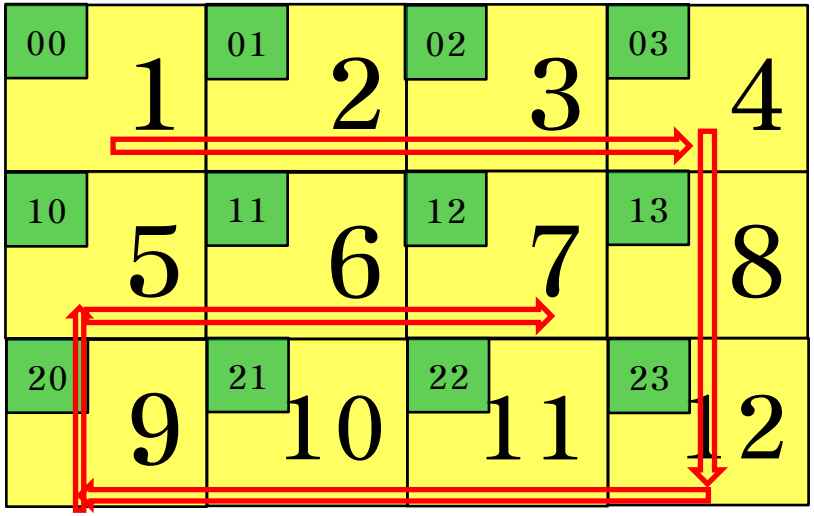
m == matrix.length
n == matrix[i].length
1 <= m, n <= 10
-100 <= matrix[i][j] <= 100

문제 Format

```
class Solution {
    public int solve(int[][] matrix) {
    }}
```

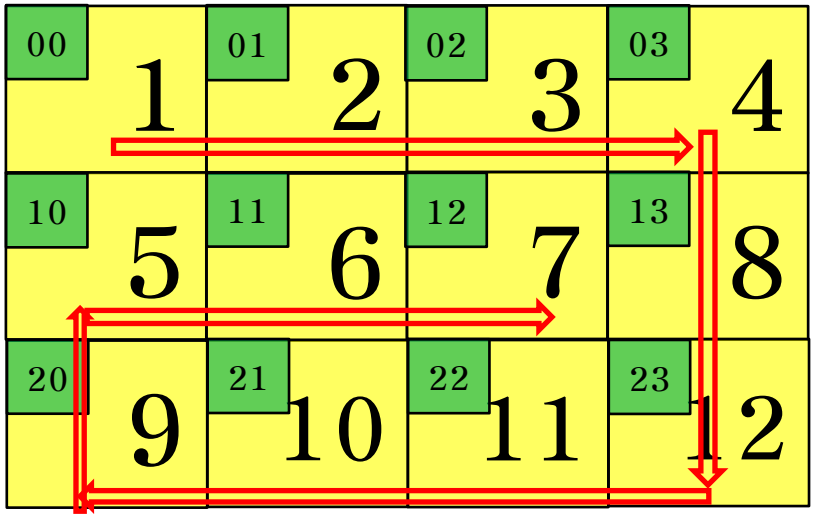
문제분석





1. 2차원 배열 좌표값 이해
2. 행(row), 열(column) 위치 파악
3. 상하좌후 위치 좌표값 변경

문제분석



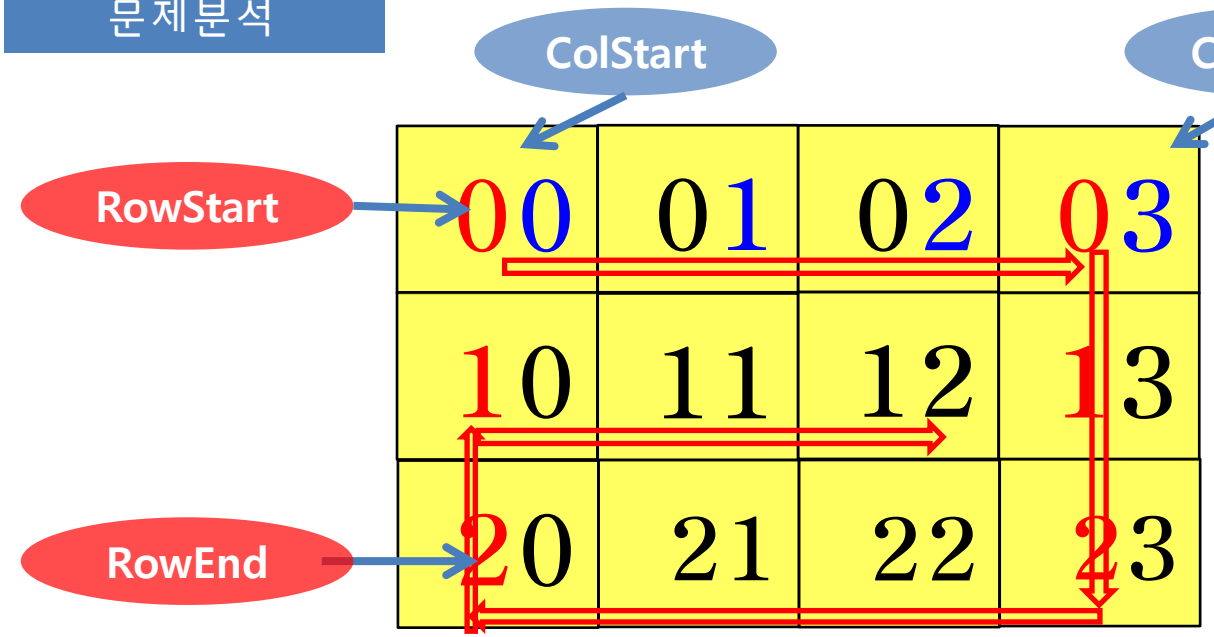
규칙 찾기

{ 00, 01, 02 , 03}, int rowStart = 0;
int rowEnd = 2 matrix.len

{ 10, 11, 12, 13},
int colStart = 0;
int colEnd = 3 matrix[0].l

{ 20, 21, 22, 23 }

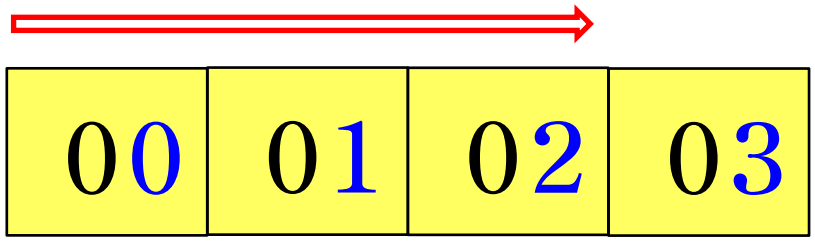
문제분석




```
int rowStart = 0;
int rowEnd   = 2
matrix.length-1;

int colStart  = 0;
int colEnd    = 3
matrix[0].length - 1;
```

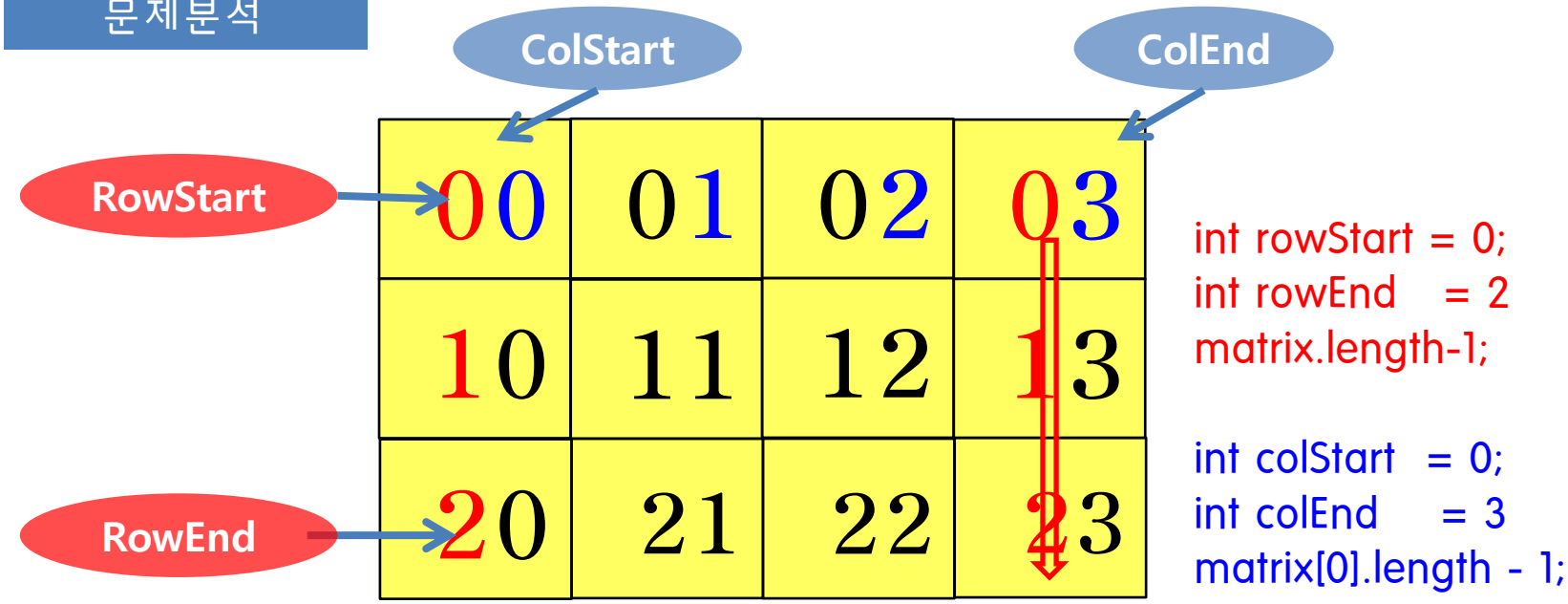
right



rowStart = 0는 그대로
int colStart = 0;
int colEnd   = 3
끝나고 rowStart ++

```
 //right
for(int i= colStart; i <= colEnd; i++ ) {
    result.add(matrix[rowStart][i]);
}
rowStart++;
```

문제분석




```
int rowStart = 0;
int rowEnd   = 2
matrix.length-1;

int colStart  = 0;
int colEnd    = 3
matrix[0].length - 1;
```

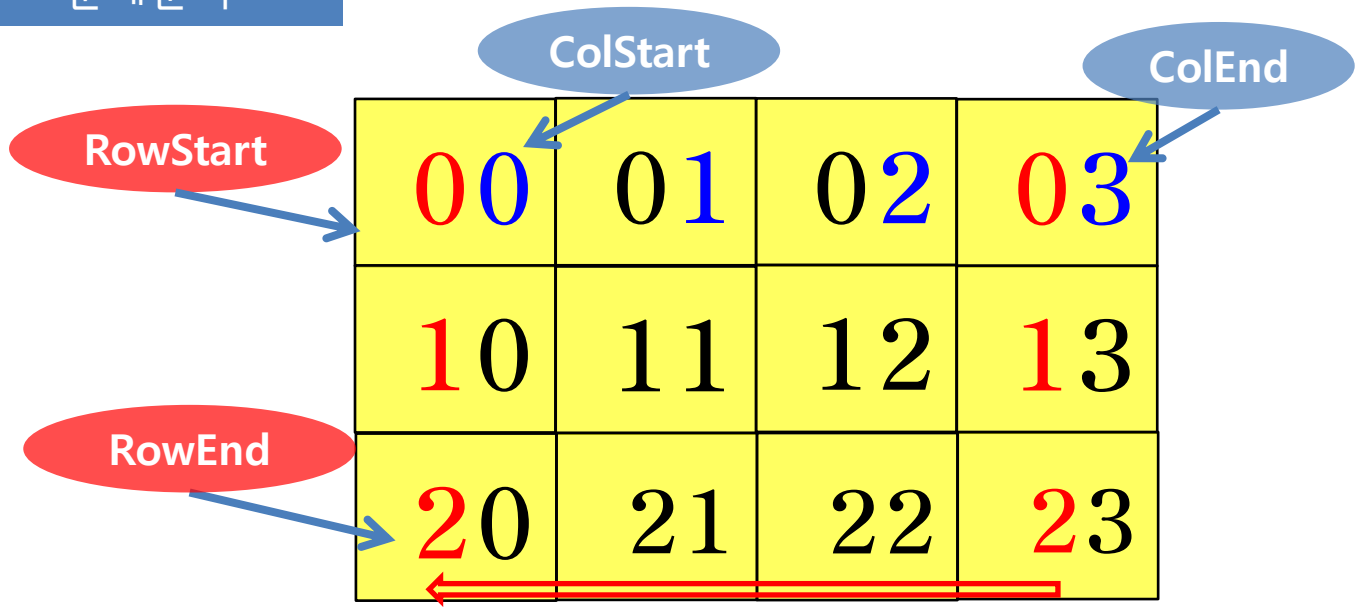
down

13
23

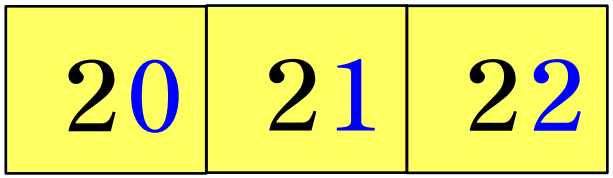
colEnd = 3는 그대로
int rowStart = 1
int rowStart = 2 으로 증가
끝나고 colEnd --

```
 //down
for(int i= rowStart; i <= rowEnd; i++ ) {
result.add(matrix[i][colEnd]);
}
colEnd--;
```

문제분석



left

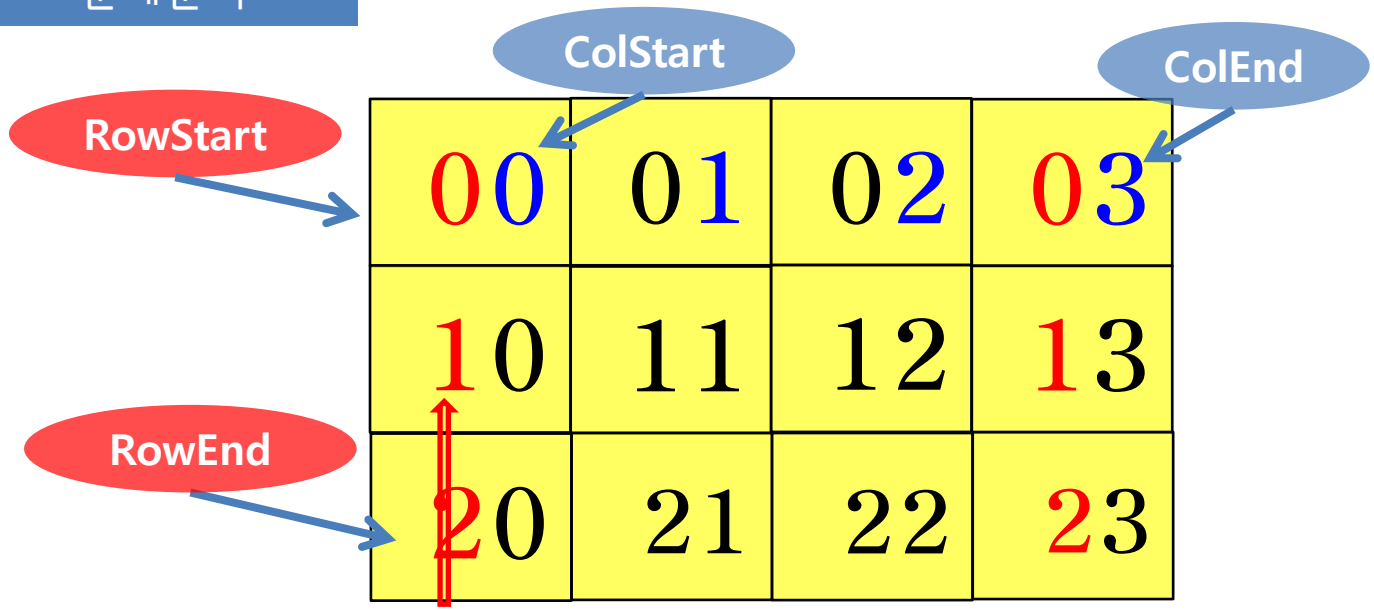


rowEnd = 2는 그대로
int colEnd = 2
int colStart = 0 감소

While문 안에서 **rowStart가 내부적으로 증가**

```
//left
if(rowStart <= rowEnd){
for(int i= colEnd; i >= colStart; i-- ) {
  result.add(matrix[rowEnd][i]);
  }
}
rowEnd--;
```

문제분석



While문 안에서 colStart가 내부적으로 증가

up

10

```
if (colStart <= colEnd) {
for (int i = rowEnd; i >= rowStart; i--) {
 result.add(matrix[i][colStart]);
 }
}
colStart++;
```

colStart = 0 그대로
int rowEnd = 1
int rowEnd = 0 감소
colStart ++