public class Solution {

/\*\*

\* @param source:

\* @param target:

\* @return: return the index

\*/

public int strStr(String source, String target) {

// Write your code here

final int BASE = 1000000;

if(source == null || target == null){

return -1;

}

int n = target.length();

if(n == 0) {

return 0;

}

int pow = 1;

for(int i = 0; i < n; i++){

pow = (pow \* 31) % BASE;

}

int targetCode = 0;

for(int i = 0; i < target.length(); i++){

targetCode = (targetCode \* 31 + target.charAt(i)) % BASE;

}

int hashCode = 0;

for(int i = 0; i < source.length(); i++){

hashCode = (hashCode \* 31 + source.charAt(i)) % BASE;

if(i < n - 1) {

continue;

}

if(i >= n){

hashCode = hashCode - (source.charAt(i - n) \* pow) % BASE;

if(hashCode < 0) {

hashCode += BASE;

}

}

if(hashCode == targetCode) {

if(source.substring(i - n + 1, i + 1).equals(target)) {

return i - n + 1;

}

}

}

return -1;

}

}

public class Solution {

public void sortIntegers2(int[] A) {

// write your code here

int start = 0;

int end = A.length - 1;

quickSort(A, start, end);

}

public void quickSort(int[] A, int start, int end) {

if(start >= end) return;

int pivot = A[start + (end - start) / 2];

int left = start;

int right = end;

while(left <= right) {

while(left <= right && A[left] < pivot) {

left++;

}

while(left <= right && A[right] > pivot) {

right--;

}

if(left <= right){

int temp = A[left];

A[left] = A[right];

A[right] = temp;

left++;

right--;

}

}

quickSort(A, start, right);

quickSort(A, left, end);

}

}

class Solution:

def twoSum(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: List[int]

"""

if not nums or len(nums) < 2:

return []

nums = [(num, i) for i, num in enumerate(nums)]

nums = sorted(nums, key=lambda x: x[0])

start, end = 0, len(nums) - 1

while start < end:

sum = nums[start][0] + nums[end][0]

if sum > target:

end -= 1

elif sum < target:

start += 1

else:

return [nums[start][1] , nums[end][1]]

class Solution:

def twoSum7(self, nums, target):

# Write your code here

nums = [(num, i) for i, num in enumerate(nums)]

target = abs(target)

n, indexs = len(nums), []

nums = sorted(nums, key=lambda x: x[0])

j = 0

for i in xrange(n):

if i == j:

j += 1

while j < n and nums[j][0] - nums[i][0] < target:

j += 1

if j < n and nums[j][0] - nums[i][0] == target:

indexs = [nums[i][1] + 1, nums[j][1] + 1]

if indexs[0] > indexs[1]:

indexs[0], indexs[1] = indexs[1], indexs[0]

return indexs

class Solution:

def twoSum7(self, A, target):

NOT\_FOUND = [-1, -1]

if not A or len(A) < 2:

return NOT\_FOUND

target = abs(target)

n = len(A)

A = [(A[i], i) for i in range(n)]

A.sort()

left = 0

for right in range(1, n):

while left + 1 < right and A[right][0] - A[left][0] > target:

left += 1

if A[right][0] - A[left][0] == target:

return sorted([

A[left][1] + 1,

A[right][1] + 1

])

return NOT\_FOUND

class Solution:

def twoSum7(self, nums, target):

hashmap = dict()

for i, num in enumerate(nums):

if num in hashmap:

return [hashmap[num] + 1, i + 1]

hashmap[num + target] = i

hashmap[num - target] = i

public class Solution {

public int[] twoSum7(int[] nums, int target) {

if (nums == null || nums.length < 2) {

return new int[]{};

}

Map<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < nums.length; ++i) {

if (map.containsKey(nums[i] + target)) {

return new int[]{map.get(nums[i] + target) + 1, i + 1};

} else if (map.containsKey(nums[i] - target)) {

return new int[]{map.get(nums[i] - target) + 1, i + 1};

} else {

map.put(nums[i], i);

}

}

return new int[]{};

}

}

Sort color

**def** **sortColors**(**self**, nums):

i = j = 0

**for** k **in** xrange(len(nums)):

v = nums[k]

nums[k] = 2

**if** v < 2:

nums[j] = 1

j += 1

**if** v == 0:

nums[i] = 0

i += 1

### 3Sum

def threeSum(self, numbers):

# write your code here

res = []

if not numbers or len(numbers) < 3:

return res

numbers.sort()

n = len(numbers)

for i in range(0, n - 2):

start, end = i + 1, n - 1

if i and numbers[i] == numbers[i - 1]:

continue

sum = 0 - numbers[i]

while start < end:

if numbers[start] + numbers[end] < sum:

start += 1

elif numbers[start] + numbers[end] > sum:

end -= 1

else:

res.append([numbers[i], numbers[start], numbers[end]])

start += 1

end -= 1

while start < end and numbers[start] == numbers[start - 1]:

start += 1

while start < end and numbers[end] == numbers[end + 1]:

end -= 1

return res

range sum query 2D - Immutable

private int[][] dp;

public NumMatrix(int[][] matrix) {

if( matrix == null || matrix.length == 0 || matrix[0].length == 0 ){return; }

int m = matrix.length;

int n = matrix[0].length;

dp = new int[m + 1][n + 1];

for(int i = 1; i <= m; i++){

for(int j = 1; j <= n; j++){

dp[i][j] = dp[i - 1][j] + dp[i][j - 1] -dp[i - 1][j - 1] + matrix[i - 1][j - 1] ; }}

}

public int sumRegion(int row1, int col1, int row2, int col2) {

int iMin = Math.min(row1, row2);

int iMax = Math.max(row1, row2);

int jMin = Math.min(col1, col2);

int jMax = Math.max(col1, col2);

return dp[iMax + 1][jMax + 1] - dp[iMax + 1][jMin] - dp[iMin][jMax + 1] + dp[iMin][jMin]; }

class Solution:

"""

@param: numCourses: a total of n courses

@param: prerequisites: a list of prerequisite pairs

@return: true if can finish all courses or false

"""

def canFinish(self, numCourses, prerequisites):

# write your code here

indegree = [0] \* numCourses

neighbor = {i: [] for i in range(numCourses)}

for i, j in prerequisites:

neighbor[j].append(i)

indegree[i] += 1

queue, count = [], 0

for index in range(numCourses):

if indegree[index] == 0:

queue.append(index)

while len(queue) > 0:

course = queue.pop(0)

count += 1

for c in neighbor[course]:

indegree[c] -= 1

if indegree[c] == 0:

queue.append(c)

return count == numCourses