# 1.Array

*1. Two Sum*

*Given an array of integers, return indices of the two numbers such that they add up to a specific target.*

*You may assume that each input would have exactly one solution, and you may not use the same element twice.*

*Example:*

*Given nums = [2, 7, 11, 15], target = 9,Because nums[0] + nums[1] = 2 + 7 = 9,return [0, 1].*

public class Solution {

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

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

for (int j = i + 1; j < nums.length; j++) {

if (nums[j] == target - nums[i]) {

return new int[] { i, j };

}

}

}

throw new IllegalArgumentException("No two sum solution");

}

}

*26. Remove Duplicates from Sorted Array*

*Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.*

*Do not allocate extra space for another array, you must do this in place with constant memory.*

*For example,Given input array nums = [1,1,2],*

*Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.*

public class Solution {

public int removeDuplicates(int[] nums) {

int count=1;

if(nums.length<2)

return nums.length;

else

{

for(int i=0;i<nums.length-1;i++)

{

if(nums[i]!=nums[i+1])

{

nums[count]=nums[i+1];

count++;

}

}

}

//System.out.println(count);

//System.out.println(Arrays.toString(nums));

return count;

}

}

/\*不仅要返回非重复值的个数（count），还要让原数组的前count个数为非重复\*/

*27. Remove Element*

*Given an array and a value, remove all instances of that value in place and return the new length.*

*Do not allocate extra space for another array, you must do this in place with constant memory.*

*The order of elements can be changed. It doesn't matter what you leave beyond the new length.*

*Example:Given input array nums = [3,2,2,3], val = 3*

*Your function should return length = 2, with the first two elements of nums being 2.*

public class Solution {

public int removeElement(int[] nums, int val) {

int i = 0;

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

if (nums[j] != val) {

nums[i] = nums[j];

//System.out.print(nums[i]);

i++;

}

}

return i;

}

}

*35. Search Insert Position*

*Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.*

*You may assume no duplicates in the array.*

*Here are few examples.*

*[1,3,5,6], 5 → 2*

*[1,3,5,6], 2 → 1*

*[1,3,5,6], 7 → 4*

*[1,3,5,6], 0 → 0*

public class Solution {

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

int i=1;

if (target==0||target<=nums[0])

return 0;

if(target>nums[nums.length-1])

return nums.length;

for(;i<nums.length-1;i++)

{

if(nums[i]==target||(target>nums[i-1]&&target<nums[i]))

break;

}

return i;

}

}

*53. Maximum Subarray*

*Find the contiguous subarray within an array (containing at least one number) which has the largest sum.*

*For example, given the array [-2,1,-3,4,-1,2,1,-5,4],the contiguous subarray [4,-1,2,1] has the largest sum = 6.*

public class Solution {

public int maxSubArray(int[] nums) {

int maxSum=Integer.MIN\_VALUE;

int thisSum=0;

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

{

thisSum+=nums[i];

if(thisSum>maxSum)

maxSum=thisSum;

if(thisSum<0)

thisSum=0;

}

//System.out.println(maxSum);

return maxSum;

}

}

*66. Plus One*

*Given a non-negative integer represented as a non-empty array of digits, plus one to the integer.*

*You may assume the integer do not contain any leading zero, except the number 0 itself.*

*The digits are stored such that the most significant digit is at the head of the list.*

public class Solution {

public int[] plusOne(int[] digits) {

int flag=0;

if(digits.length==1&&digits[0]==9)//数组只有一个9的话，flag就不起作用

return new int[]{1,0};

digits[digits.length-1]++;

for(int i=digits.length-1;i>0;i--)

{

if(digits[i]==10)

{

flag=1;

digits[i-1]++;

digits[i]=0;

}

}

if(digits[0]==10&&flag==1){

digits[0]=0;

int []new\_dig=new int[digits.length+1];

new\_dig[0]=1;

for(int j=1;j<digits.length+1;j++){

new\_dig[j]=digits[j-1];

}

return new\_dig;

}

else {

return digits;

}

}

}

*88. Merge Sorted Array*

*Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.*

*Note:*

*You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.*

public class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

ArrayList<Integer> nums=new ArrayList<Integer>();

for(int i=0;i<m;i++)

{

nums.add(nums1[i]);

}

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

{

nums.add(nums2[i]);

}

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

{

nums1[i]=nums.get(i);

}

Arrays.sort(nums1);

}

}

*121. Best Time to Buy and Sell Stock*

*Say you have an array for which the ith element is the price of a given stock on day i.*

*If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.*

*Example 1:*

*Input: [7, 1, 5, 3, 6, 4]*

*Output: 5*

*max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)*

*Example 2:*

*Input: [7, 6, 4, 3, 1]*

*Output: 0*

*In this case, no transaction is done, i.e. max profit = 0.*

public class Solution {

public int maxProfit(int[] prices) {

int minprice = Integer.MAX\_VALUE;

int maxprofit = 0;

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

if (prices[i] < minprice)

minprice = prices[i];

else if (prices[i] - minprice > maxprofit)

maxprofit = prices[i] - minprice;

}

return maxprofit;

}

}

136. Single Number

Given an array of integers, every element appears twice except for one. Find that single one.

public class Solution {

public int singleNumber(int[] nums) {

int i;

int result=nums[0];

for(i=1;i<nums.length;i++){

result=result^nums[i];

}

//System.out.println(result);

return result;

}

}

*165. Compare Version Numbers*

*Compare two version numbers version1 and version2.*

*If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.*

*You may assume that the version strings are non-empty and contain only digits and the . character.*

*The . character does not represent a decimal point and is used to separate number sequences.*

*For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.*

*Here is an example of version numbers ordering:0.1 < 1.1 < 1.2 < 13.37*

public class Solution {

public int compareVersion(String version1, String version2) {

long a = 0, b =0;

int v1len = version1.length(), v2len = version2.length();

int i = 0, j = 0;

while (i < v1len || j < v2len) {

a = 0; b =0;

while (i < v1len && version1.charAt(i) != '.') {

a = a \* 10 + version1.charAt(i) - '0';

++i;

}

++i;

while (j < v2len && version2.charAt(j) != '.') {

b = b \* 10 + version2.charAt(j) - '0';

++j;

}

++j;

if (a > b) return 1;

if (a < b) return -1;

}

return 0;

}

}

*167. Two Sum II - Input array is sorted*

*Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.*

*The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.*

*You may assume that each input would have exactly one solution and you may not use the same element twice.*

*Input: numbers={2, 7, 11, 15}, target=9*

*Output: index1=1, index2=2*

public class Solution {

public int[] twoSum(int[] numbers, int target) {

int res[]=new int[2];

int left=0;

int right=numbers.length-1;

while(left<right)

{

if(numbers[left]+numbers[right]==target)

{

res[0]=left+1;

res[1]=right+1;

break;

}

if(numbers[left]+numbers[right]<target)

left++;

else

right--;

}

// System.out.println(Arrays.toString(res));

return res;

}

}

*169. Majority Element*

*Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.*

*You may assume that the array is non-empty and the majority element always exist in the array.*

public class Solution {

public int majorityElement(int[] nums) {

Arrays.sort(nums);

if(nums.length%2!=0)

return nums[nums.length/2];

int count1=0;

int count2=0;

if(nums.length%2==0){

for(int i=nums.length/2;i<nums.length-1;i++){

if(nums[i]==nums[i+1])

count1++;

}

for(int i=nums.length/2;i>1;i--){

if(nums[i]==nums[i-1])

count2++;

}

}

return count1>count2?nums[nums.length/2]:nums[nums.length/2-1];

}

}

*189. Rotate Array*

*Rotate an array of n elements to the right by k steps.*

*For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].*

public class Solution {

public void rotate(int[] nums, int k) {

if(k>nums.length)

k=k%(nums.length);

if(nums==null||k==0)

return;

int temp1,temp2,temp3,m,count=0;;

for(int i=0;i<nums.length/2;i++){

temp1=nums[i];

nums[i]=nums[nums.length-1-i];

nums[nums.length-1-i]=temp1;

// System.out.print(nums[i]);

}

for(int j=0;j<k/2;j++){

temp2=nums[j];

nums[j]=nums[k-1-j];

nums[k-1-j]=temp2;

// System.out.print(nums[j]);

}

for( m=k;m<(nums.length+k)/2;m++){

temp3=nums[m];

nums[m]=nums[nums.length-1-count];

nums[nums.length-1-count]=temp3;

//System.out.print(nums[m]);

count++;

}

}

}

*204. Count Primes(素数)*

*Count the number of prime numbers less than a non-negative number, n.*

public class Solution {

public int countPrimes(int n) {

boolean[] prime = new boolean[n];

Arrays.fill(prime, true);

for(int i = 2; i < n; i++){

if(prime[i]){

// 将i的2倍、3倍、4倍...都标记为非素数

for(int j = i \* 2; j < n; j = j + i){

prime[j] = false;

}

}

}

int count = 0;

for(int i = 2; i < n; i++){

if(prime[i]) count++;

}

return count;

}

}

*217. Contains Duplicate*

*Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.*

public class Solution {

public boolean containsDuplicate(int[] nums) {

Arrays.sort(nums);

int j=0;

boolean a = false;

while(j<nums.length-1){

if(nums[j]==nums[j+1]){

a=true;

break;

}

j++;

}

return a;

}

}

*219. Contains Duplicate II*

*Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the absolute difference between i and j is at most k.*

public class Solution {

public boolean containsNearbyDuplicate(int[] nums, int k) {

if(nums.length<2||nums.length>1000)

return false;

for(int i=0;i<nums.length-1;i++)

{

for(int j=i+1;j<nums.length;j++)

{

if(nums[i]==nums[j]&&j-i<=k)

return true;

}

}

return false;

}

}/\*beat99.91% 时间复杂度过高，纯粹骗过了测试用例\*/

public boolean containsNearbyDuplicate(int[] nums, int k) {

if(nums==null || nums.length<2) return false;

//key=int, val=index

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

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

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

int j = map.get(nums[i]);

if(i-j<=k) return true;

} else {

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

}

}

return false;

}

/\*正解还是应该用map来标记\*/

*242. Valid Anagram*

*Given two strings s and t, write a function to determine if t is an anagram of s.*

*For example,*

*s = "anagram", t = "nagaram", return true.*

*s = "rat", t = "car", return false.*

public class Solution {

public boolean isAnagram(String s, String t) {

char[] ArrS=s.toCharArray();

char[] ArrT=t.toCharArray();

Arrays.sort(ArrS);

Arrays.sort(ArrT);

return (String.valueOf(ArrS).equals(String.valueOf(ArrT)));

}

}

*268. Missing Number*

*Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.*

*For example,Given nums = [0, 1, 3] return 2.*

public class Solution {

public int missingNumber(int[] nums) {

Arrays.sort(nums);

int count=0;

while(count<nums.length)

{

if(nums[count]!=count)

break;

else

count++;

}

return count;

}

}

*283. Move Zeroes*

*Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.*

*For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].*

public class Solution {

public void moveZeroes(int[] nums) {

//只有当i指向为非0时，且ji不同时，j后移，将j处的数改成i处的数，

//并使i处的数字为0，ij同步后移，而当i指向为0时，只有i后移，j仍然指向0处

int i,j=0;

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

if(nums[i]!=0){

if(j!=i){

nums[j]=nums[i];

nums[i]=0;

}

j++;

}

}

}

}

*290. Word Pattern*

*Given a pattern and a string str, find if str follows the same pattern.*

*Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.*

*Examples:*

*pattern = "abba", str = "dog cat cat dog" should return true.*

*pattern = "abba", str = "dog cat cat fish" should return false.*

*pattern = "aaaa", str = "dog cat cat dog" should return false.*

*pattern = "abba", str = "dog dog dog dog" should return false.*

public class Solution {

public boolean wordPattern(String pattern, String str) {

Map<Character, String> map= new HashMap<Character, String>();

Set<String> set=new HashSet<String>();

String [] arrStrings=str.split(" ");

if(pattern.length()!=arrStrings.length)

return false;

if(pattern.length()==0||pattern.length()==1)

return true;

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

{

char a=pattern.charAt(i);

String b=arrStrings[i];

if(map.containsKey(a)){

if(!map.get(a).equals(b))

return false;

}

else{

if(set.contains(b))

return false;

else{

map.put(a, b);

set.add(b);

}

}

}

return true;

}

}

//与205一样

303. Range Sum Query – Immutable

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

Example:Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1

sumRange(2, 5) -> -1

sumRange(0, 5) -> -3

public class NumArray {

int[] sums;

public NumArray(int[] nums) {

if(nums==null){

sums = null;

}

else if (nums.length == 0) {

sums = new int[0];

}

else {

sums = new int[nums.length];

sums[0] = nums[0];

for (int i = 1; i < nums.length; i++) {

sums[i] = sums[i - 1] + nums[i];

}

}

}

public int sumRange(int i, int j) {

if (sums == null) {

return 0;

}

if (i >= sums.length || j >= sums.length || i > j) {

return 0;

}

else if (i == 0) {

return sums[j];

}

else {

return sums[j] - sums[i - 1];

}

}

}

// Your NumArray object will be instantiated and called as such:

// NumArray numArray = new NumArray(nums);

// numArray.sumRange(0, 1);

// numArray.sumRange(1, 2);

*349. Intersection of Two Arrays*

*Given two arrays, write a function to compute their intersection.*

*Example:Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].*

public class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

ArrayList<Integer> list=new ArrayList<Integer>();

int n1\_len=nums1.length;

int n2\_len=nums2.length;

for(int i=0;i<n1\_len;i++)

{

for(int j=0;j<n2\_len;j++)

{

if(nums1[i]==nums2[j]&&!list.contains(nums1[i])) //如果list不包含交叉值，才add，避免重复

list.add(nums1[i]);

}

}

int []nums=new int[list.size()];

for(int i=0;i<list.size();i++)

{

nums[i]=list.get(i);

}

return nums;

}

}

*396. Rotate Function*

*Given an array of integers A and let n to be its length.*

*Assume Bk to be an array obtained by rotating the array A k positions clock-wise, we define a "rotation function" F on A as follow:*

*F(k) = 0 \* Bk[0] + 1 \* Bk[1] + ... + (n-1) \* Bk[n-1].*

*Calculate the maximum value of F(0), F(1), ..., F(n-1).*

public class Solution {

public int maxRotateFunction(int[] A) {

if(A.length==0||A.length==1) //数组为空或长度为一，则返回0

return 0;

int[] m\_arr=new int[A.length]; //新建与原数组等长的数组，用来存放F(x)

int f=0;

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

{

for(int k1=0;k1<=(A.length-2)/2;k1++) //先翻转前n-1个数组成员

{

int temp=A[k1];

A[k1]=A[(A.length-2)-k1];

A[(A.length-2)-k1]=temp;

}

for(int k1=0;k1<=(A.length-1)/2;k1++) //再翻转所有数组成员来实现clock访问

{

int temp=A[k1];

A[k1]=A[(A.length-1)-k1];

A[(A.length-1)-k1]=temp;

}

for(int j=0;j<A.length;j++)

f=f+j\*A[j];

m\_arr[i]=f;

f=0;

}

Arrays.sort(m\_arr);

// System.out.println(m\_arr[m\_arr.length-1]);

return m\_arr[m\_arr.length-1];

}

}

*414. Third Maximum Number*

*Given a non-empty array of integers, return the third maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in O(n).*

*Example 1:*

*Input: [3, 2, 1]*

*Output: 1*

*Explanation: The third maximum is 1.*

*Example 2:*

*Input: [1, 2]*

*Output: 2*

*Explanation: The third maximum does not exist, so the maximum (2) is returned instead.*

*Example 3:*

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

*Output: 1*

*Explanation: Note that the third maximum here means the third maximum distinct number.*

*Both numbers with value 2 are both considered as second maximum.*

public class Solution {

public int thirdMax(int[] nums) {

long first, second, third;

first = second = third = Long.MIN\_VALUE;

for (int num : nums)

{

if (num == first || num == second || num == third) continue;

if (num > first)

{

third = second;

second = first;

first = num;

}

else if (num > second)

{

third = second;

second = num;

}

else if (num > third) third = num;

}

return (third == Long.MIN\_VALUE) ? (int)first : (int)third;

}

}

*453. Minimum Moves to Equal Array Elements*

*Given a non-empty integer array of size n, find the minimum number of moves required to make all array elements equal, where a move is incrementing n - 1 elements by 1.*

*Example: Input:[1,2,3] Output:3*

*Explanation:Only three moves are needed (remember each move increments two elements):*

*[1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]*

public class Solution {

public int minMoves(int[] nums) {

int min= Integer.MAX\_VALUE;

int sum = 0;

for(int n:nums){

//for each循环

min = Math.min(min, n);

sum+=n;

}

return sum-nums.length\*min;

}

}

*455. Assign Cookies*

*Example 1: Input: [1,2,3], [1,1] Output: 1*

*Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.*

*And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.*

*You need to output 1.*

*Example 2: Input: [1,2], [1,2,3]Output: 2*

*Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.*

*You have 3 cookies and their sizes are big enough to gratify all of the children,*

*You need to output 2.*

public class Solution {

public int findContentChildren(int[] g, int[] s) {

Arrays.sort(s);

Arrays.sort(g);

int i=0;

int j=0;

int count =0;

while(i<g.length&&j<s.length)//从小到大依次比较

{

if(g[i]<=s[j])

{

i++;

j++;

count++;

}

else

{

j++;

}

}

// System.out.println(count);

return count;

}

}

*463. Island Perimeter*

*You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water. Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells). The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.*

*Example:*

*[[0,1,0,0],*

*[1,1,1,0],*

*[0,1,0,0],*

*[1,1,0,0]]*

*Answer: 16*

*Explanation: The perimeter is the 16 yellow stripes in the image below:* 

public class Solution {

public int islandPerimeter(int[][] grid) {

if(grid == null || grid.length ==0 || grid[0].length == 0) return 0;

int count = 0;

int duplicate = 0;

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

for(int j =0 ;j<grid[0].length;j++){

if(grid[i][j] == 1){

count += 4;

if(i!= 0){

if(grid[i-1][j] == 1){

duplicate++;

}

}

if(i != grid.length-1){

if(grid[i+1][j] == 1){

duplicate++;

}

}

if(j != 0){

if(grid[i][j-1] == 1){

duplicate++;

}

}

if(j != grid[0].length-1){

if(grid[i][j+1] == 1){

duplicate++;

}

}

}

}

}

return count - duplicate;

}

}

*485. Max Consecutive Ones*

*Given a binary array, find the maximum number of consecutive 1s in this array.*

*Example 1:Input: [1,1,0,1,1,1] Output: 3*

*Explanation: The first two digits or the last three digits are consecutive 1s.*

*The maximum number of consecutive 1s is 3.*

public class Solution {

public int findMaxConsecutiveOnes(int[] nums) {

int answer=0;

int count =0;

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

{

if(i != nums.length && nums[i] == 1)

{

count++ ;

}

else{

if(count == 0)

{

continue;

}

answer=answer > count ? answer : count;

count = 0;

}

}

return answer;

}

}

*496. Next Greater Element I*

*You are given two arrays (without duplicates) nums1 and nums2 where nums1’s elements are subset of nums2. Find all the next greater numbers for nums1's elements in the corresponding places of nums2.*

*The Next Greater Number of a number x in nums1 is the first greater number to its right in nums2. If it does not exist, output -1 for this number.*

*Example 1:*

*Input: nums1 = [4,1,2], nums2 = [1,3,4,2].*

*Output: [-1,3,-1]*

*Explanation:*

*For number 4 in the first array, you cannot find the next greater number for it in the second array, so output -1.*

*For number 1 in the first array, the next greater number for it in the second array is 3.*

*For number 2 in the first array, there is no next greater number for it in the second array, so output -1.*

*Example 2:*

*Input: nums1 = [2,4], nums2 = [1,2,3,4].*

*Output: [3,-1]*

*Explanation:*

*For number 2 in the first array, the next greater number for it in the second array is 3.*

*For number 4 in the first array, there is no next greater number for it in the second array, so output -1.*

public class Solution {

public int[] nextGreaterElement(int[] findNums, int[] nums) {

int res[]=new int[findNums.length];

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

for(int i=0;i<nums.length;i++) //把主数组放到map(因为数组是不重复的)里标记好

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

for(int i=0;i<findNums.length;i++) //将子数组中的各元素对应到主数组中

{

int k=map.get(findNums[i]); //从主数组的特定位置(map中已经对应)开始遍历，节省开销

if(k==nums.length-1)

res[i]=-1;

for(int j=k;j<nums.length;j++)

{

if(nums[j]>findNums[i]){

res[i]=nums[j];

break;

}

else

res[i]=-1;

}

}

// System.out.println(Arrays.toString(res));

return res;

}

}

/beat 88%

*500. Keyboard Row*

*Given a List of words, return the words that can be typed using letters of alphabet on only one row's of American keyboard like the image below.*

*Example 1:*

*Input: ["Hello", "Alaska", "Dad", "Peace"]*

*Output: ["Alaska", "Dad"]*

public class Solution {

public String[] findWords(String[] words) {

String s1="qwertyuiopQWERTYUIOP";

String s2="asdfghjklASDFGHJKL";

String s3="zxcvbnmZXCVBNM";

ArrayList<String >list=new ArrayList<String>();

for(String w:words)

{

char first=w.charAt(0);//确定第一个字母属于哪个字符串

boolean flag=true;

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

{

if(s1.contains(first+"")&&!s1.contains(w.charAt(i)+""))//如果存在第一个字母属于s1，但又有s1不包含的字母，那他肯定不是

{

flag=false;

break;

}

else if(s2.contains(first+"")&&!s2.contains(w.charAt(i)+""))

{

flag=false;

break;

}

else if(s3.contains(first+"")&&!s3.contains(w.charAt(i)+""))

{

flag=false;

break;

}

}

if(flag==true)

list.add(w);

}

return list.toArray(new String[0]);

}

}

*152. Maximum Product Subarray*

*Find the contiguous subarray within an array (containing at least one number) which has the largest product.*

*For example, given the array [2,3,-2,4],*

*the contiguous subarray [2,3] has the largest product = 6.*

public class Solution {

public int maxProduct(int[] nums) {

/\*要保留一个到某一位来看的最大值和最小值。因为在数组中有负数的出现，所以到这一位为止的能得到的最大值，

可能是由之前的最大值和这个数相乘得到，也可能是最小值和这个数相乘得到的。\*/

if(nums == null || nums.length == 0)

return 0;

int max = nums[0];

int maxProduct = nums[0];

int minProduct = nums[0];

for(int i = 1; i < nums.length; i ++)

{

int prevMax = maxProduct;

int prevMin = minProduct;

maxProduct = Math.max(nums[i], Math.max(prevMax \* nums[i], prevMin \* nums[i]));

minProduct = Math.min(nums[i], Math.min(prevMin \* nums[i], prevMax \* nums[i]));

max = Math.max(max, maxProduct);

}

// System.out.println(max);

return max;

}

}

*419. Battleships in a Board*

*Given an 2D board, count how many battleships are in it. The battleships are represented with 'X's, empty slots are represented with '.'s. You may assume the following rules:*

*You receive a valid board, made of only battleships or empty slots.*

*Battleships can only be placed horizontally or vertically. In other words, they can only be made of the shape 1xN (1 row, N columns) or Nx1 (N rows, 1 column), where N can be of any size.*

*At least one horizontal or vertical cell separates between two battleships - there are no adjacent battleships.*

*Example:*

*X . . X*

*. . . X*

*. . . X*

*In the above board there are 2 battleships.*

*Invalid Example:*

*. . .X*

*XXXX*

*. . .X*

*This is an invalid board that you will not receive - as battleships will always have a cell separating between them.*

public class Solution {

public int countBattleships(char[][] board) {

int res = 0, height = board.length, width = board[0].length, i, j;

for (i = 0; i < height; i++)

for (j = 0; j < width; j++) {

if (board[i][j] == '.' || (i > 0 && board[i - 1][j] == 'X')

|| j > 0 && board[i][j - 1] == 'X')

continue;

res++;

}

return res;

}

}