1. 给你一个整数数组 nums 。如果任一值在数组中出现 至少两次 ，返回 true ；如果数组中每个元素互不相同，返回 false 。（map）

*bool containsDuplicate(int\* nums, int numsSize){*

*int a[1000][100]={0};*

*int b[1000]={0};*

*int i,j;*

*for(i=0;i<numsSize;i++)*

*{*

*int x=abs(nums[i])%1000;*

*for(j=0;j<b[x];j++)*

*{*

*if(a[x][j]==nums[i])*

*{*

*return true;*

*}*

*}*

*b[x]++;*

*a[x][b[x]-1]=nums[i];*

*}*

*return false;*

*}*

*func containsDuplicate(nums []int) bool {*

*m := make(map[int]int)*

*for \_,value:=range nums{*

*m[value] ++*

*if m[value]>1{*

*return true*

*}*

*}*

*return false*

*}*

1. 给你一个整数数组 nums ，请你找出一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。子数组 是数组中的一个连续部分。（比较当前数组值）

*int maxSubArray(int\* nums, int numsSize){*

*int sum,i,j,temp,ack;*

*sum = nums[0];*

*ack = nums[0];*

*for(i=1;i<numsSize;i++){*

*if(sum+nums[i]>nums[i]){*

*sum = sum +nums[i];*

*}else{*

*sum=nums[i];*

*}*

*if(ack<sum){*

*ack = sum;*

*}*

*}*

*return ack;*

*}*

*func maxSubArray(nums []int) int {*

*sum := nums[0]*

*ack := nums[0]*

*for i:=1;i<len(nums);i++{*

*if(sum+nums[i]>nums[i]){*

*sum = sum + nums[i]*

*}else{*

*sum = nums[i]*

*}*

*if(ack<sum){*

*ack = sum*

*}*

*}*

*return ack*

*}*

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出 和为目标值 target 的那 两个 整数，并返回它们的数组下标。（map）

*int\* twoSum(int\* nums, int numsSize, int target, int\* returnSize){*

*int \*p = malloc(sizeof(int) \* 2);*

*\*returnSize = 2;*

*for(int i=0;i<numsSize-1;i++){*

*int m = target - nums[i];*

*for(int j = numsSize-1;j>0;j--){*

*if(nums[j]==m && i!=j){*

*p[0]=i;*

*p[1]=j;*

*return p;*

*}*

*}*

*}*

*return NULL;*

*}*

*func twoSum(nums []int, target int) []int {*

*m := make(map[int]int,len(nums))*

*for i:=0;i<len(nums);i++{*

*m[nums[i]]=i*

*}*

*for j:=0;j<len(nums);j++{*

*if n,ok:=m[target-nums[j]];ok{*

*if(j !=n ){*

*return []int{j,n}*

*}*

*}*

*}*

*return []int{}*

*}*

1. 给你两个按 非递减顺序 排列的整数数组 nums1 和 nums2，另有两个整数 m 和 n ，分别表示 nums1 和 nums2 中的元素数目。（从后往前）

*void merge(int\* nums1, int nums1Size, int m, int\* nums2, int nums2Size, int n){*

*int i,j,k;*

*k = n+m-1;*

*for(i=m-1,j=n-1;i>=0||j>=0;){*

*if (i == -1) {*

*nums1[k--] = nums2[j--];*

*} else if (j == -1) {*

*nums1[k--] = nums1[i--];*

*} else if (nums1[i]>nums2[j]){*

*nums1[k--] = nums1[i--];*

*}else{*

*nums1[k--]=nums2[j--];*

*}*

*}*

*}*

*func merge(nums1 []int, m int, nums2 []int, n int)  {*

*i,j,k := m-1,n-1,n+m-1*

*for;i>=0||j>=0;k--{*

*if i==-1{*

*nums1[k]=nums2[j]*

*j--*

*}else if j==-1{*

*nums1[k]=nums1[i]*

*i--*

*}else if nums1[i]>nums2[j] {*

*nums1[k]=nums1[i]*

*i--*

*}else{*

*nums1[k]=nums2[j]*

*j--*

*}*

*}*

*}*

1. 给你两个整数数组 nums1 和 nums2 ，请你以数组形式返回两数组的交集。返回结果中每个元素出现的次数，应与元素在两个数组中都出现的次数一致（如果出现次数不一致，则考虑取较小值）。可以不考虑输出结果的顺序 （1先排序再比较，2 map）

*int cmp(int \*a,int \*b){*

*return \*a - \*b;*

*}*

*int\* intersect(int\* nums1, int nums1Size, int\* nums2, int nums2Size, int\* returnSize){*

*qsort(nums1,nums1Size,sizeof(int),cmp);*

*qsort(nums2,nums2Size,sizeof(int),cmp);*

*int\* mr = (int\*)malloc(sizeof(int) \* fmin(nums1Size, nums2Size));*

*int i,j;*

*\*returnSize = 0;*

*for (i=0,j=0;i<nums1Size&&j<nums2Size;){*

*if (nums1[i]<nums2[j]){*

*i++;*

*}else if( nums1[i]>nums2[j]){*

*j++;*

*}else{*

*mr[(\*returnSize)++]=nums1[i];*

*i++;*

*j++;*

*}*

*}*

*return mr;*

*}*

*func intersect(nums1 []int, nums2 []int) []int {*

*// sort.Ints(nums1)*

*// sort.Ints(nums2)*

*// mr :=[]int{}*

*// len1,len2 := len(nums1),len(nums2)*

*// for i,j:=0,0;i<len1&&j<len2;{*

*//     if nums1[i]<nums2[j]{*

*//         i++*

*//     }else if nums1[i]>nums2[j]{*

*//         j++*

*//     }else{*

*//         mr = append(mr,nums1[i])*

*//         i++*

*//         j++*

*//     }*

*// }*

*// return mr*

*mr :=[]int{}*

*m1 := make(map[int]int)*

*for \_,value:=range nums1{*

*m1[value]++*

*}*

*for \_,value1:=range nums2{*

*if m1[value1]>0{*

*mr = append(mr,value1)*

*m1[value1]--*

*}*

*}*

*return mr*

*}*

1. 给定一个数组 prices ，它的第 i 个元素 prices[i] 表示一支给定股票第 i 天的价格。你只能选择 某一天 买入这只股票，并选择在 未来的某一个不同的日子 卖出该股票。设计一个算法来计算你所能获取的最大利润。返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回 0。（贪心，动规）

*// int maxProfit(int\* prices, int pricesSize){*

*//     int i,j;*

*//     int ack = 0;*

*//     for(i=0;i<pricesSize-1;i++){*

*//         for(j=pricesSize-1;j>i;j--){*

*//             if((prices[j]-prices[i])>ack){*

*//                 ack = prices[j]-prices[i];*

*//             }*

*//         }*

*//     }*

*//     return ack;*

*// }//暴力超时*

*int maxProfit(int\* prices, int pricesSize){*

*int i ;*

*int min=prices[0];*

*int ack=0;*

*for (i=0;i<pricesSize;i++){*

*if(min>prices[i]){*

*min=prices[i];*

*}*

*if(ack<prices[i]-min){*

*ack = prices[i]-min;*

*}*

*}*

*return ack;*

*}//贪心*

*int maxProfit(int\* prices, int pricesSize)*

*{*

*int dp[pricesSize][2];*

*int i;*

*dp[0][0] = 0;*

*dp[0][1] = -prices[0];*

*for(i = 1; i < pricesSize; i++ )*

*{*

*//第i天没有股票，有两种情况 昨天没有买 今天保持  昨天买了今天 卖出了*

*dp[i][0] = fmax(dp[i-1][0],dp[i-1][1] + prices[i]);*

*//第i天持有股票，有两种情况 昨天没有买 今天保持  昨天没买了  今天买*

*dp[i][1] = fmax(dp[i-1][1],-prices[i]);*

*}*

*return dp[pricesSize-1][0];*

*}//动规*

*int maxProfit(int\* prices, int pricesSize) {*

*int minprice = prices[0];*

*int dp[pricesSize];*

*int i;*

*dp[0] = 0;*

*for (i = 1; i < pricesSize; i++){*

*minprice = fmin(minprice, prices[i]);*

*dp[i] = fmax(dp[i - 1], prices[i] - minprice);*

*}*

*return dp[i-1];*

*}//动规*

*func maxProfit(prices []int) int {*

*i,min:=0,prices[0]*

*ack := 0*

*for i=0;i<len(prices);i++{*

*if min >prices[i]{*

*min = prices[i]*

*}*

*if ack< prices[i]-min{*

*ack = prices[i]-min*

*}*

*}*

*return ack*

*}*

1. 在 MATLAB 中，有一个非常有用的函数 reshape ，它可以将一个 m x n 矩阵重塑为另一个大小不同（r x c）的新矩阵，但保留其原始数据。给你一个由二维数组 mat 表示的 m x n 矩阵，以及两个正整数 r 和 c ，分别表示想要的重构的矩阵的行数和列数。重构后的矩阵需要将原始矩阵的所有元素以相同的 行遍历顺序 填充。如果具有给定参数的 reshape 操作是可行且合理的，则输出新的重塑矩阵；否则，输出原始矩阵。

*int\*\* matrixReshape(int\*\* mat, int matSize, int\* matColSize, int r, int c, int\* returnSize, int\*\* returnColumnSizes){*

*int m = matSize;         //行*

*int n = \*matColSize;     //列*

*if (m \* n != r \* c) {*

*\*returnSize = matSize;*

*\*\*returnColumnSizes = \*matColSize;*

*return mat;*

*}*

*\*returnSize = r;*

*\*returnColumnSizes = malloc(sizeof(int) \* r);*

*int\*\* ans = malloc(sizeof(int\*) \* r);//开辟指针数组，指向r行数组*

*//初始化*

*for (int i = 0; i < r; i++) {*

*(\*returnColumnSizes)[i] = c;//该数组用来标记，每一行的列数*

*ans[i] = malloc(sizeof(int) \* c);//每一行指针指向的数组空间大小*

*}*

*for (int x = 0; x < m \* n; ++x) {*

*ans[x / c][x % c] = mat[x / n][x % n];*

*}*

*return ans;*

*}*

*func matrixReshape(mat [][]int, r int, c int) [][]int {*

*m,n:=len(mat),len(mat[0])*

*if m\*n!=r\*c{*

*return mat*

*}*

*a := make([][]int,r)*

*for i:=range a{*

*a[i] = make([]int,c)*

*}*

*for k:=0;k<r\*c;k++{*

*a[k/c][k%c]=mat[k/n][k%n]*

*}*

*return a*

*}*

1. 给定一个非负整数 numRows，生成「杨辉三角」的前 numRows 行。在「杨辉三角」中，每个数是它左上方和右上方的数的和。

*int\*\* generate(int numRows, int\* returnSize, int\*\* returnColumnSizes){*

*int \*\*a = malloc(sizeof(int \*) \* numRows);*

*\* returnSize = numRows;*

*\*returnColumnSizes = malloc(sizeof(int )\*numRows);*

*int i,j;*

*for (i=0;i<numRows;i++){*

*a[i]=malloc(sizeof(int )\*(i+1));*

*(\*returnColumnSizes)[i] = i + 1;//由于读取也需要读取列数所以列数也需要赋值*

*for (j=0;j<=i;j++){*

*if(j==0||j==i){*

*a[i][j]=1;*

*}else{*

*a[i][j]=a[i-1][j-1]+a[i-1][j];*

*}*

*}*

*}*

*return a;*

*}*

*func generate(numRows int) [][]int {*

*a:=make([][]int,numRows)*

*for m:=0;m<numRows;m++{*

*a[m]=make([]int,m+1)*

*}*

*for i:=0;i<numRows;i++{*

*for j:=0;j<=i;j++{*

*if(j==0||j==i){*

*a[i][j]=1*

*}else{*

*a[i][j]=a[i-1][j-1]+a[i-1][j]*

*}*

*}*

*}*

*return a*

*}*

1. 请你判断一个 9 x 9 的数独是否有效。只需要 根据以下规则 ，验证已经填入的数字是否有效即可。

*bool isValidSudoku(char\*\* board, int boardSize, int\* boardColSize){*

*int yz1[9][9]={0};*

*int yz2[9][9]={0};*

*int yz3[3][3][9]={0};*

*int temp,n;*

*for (int i =0;i<9;i++){*

*for(int j=0;j<9;j++){*

*if(board[i][j]=='.'){*

*continue;*

*}*

*temp = (board[i][j]-'0');*

*n=i/3\*3+j/3;*

*yz1[i][temp-1]++;*

*yz2[j][temp-1]++;*

*yz3[i/3][j/3][temp-1]++;*

*if(yz1[i][temp-1]>1||yz2[j][temp-1]>1||yz3[i/3][j/3][temp-1]>1){*

*return false;*

*}*

*}*

*}*

*return true;*

*}*

*func isValidSudoku(board [][]byte) bool {*

*yz1,yz2,yz3 := make(map[int]bool,9),make(map[int]bool,9),make(map[int]bool,9)*

*for i:=0;i<9;i++{*

*for j:=0;j<9;j++{*

*if board[i][j] == '.' {*

*continue*

*}else{*

*tmp := int( board[i][j]- '0')*

*n := (i/3)\*3 + j/3*

*a, b, c := i\*10+tmp, j\*10+tmp, n\*10+tmp*

*if yz1[a] || yz2[b] || yz3[c] {*

*return false*

*}*

*yz1[a] = true*

*yz2[b] = true*

*yz3[c] = true*

*}*

*}*

*}*

*return true*

*}*

1. 给定一个 m x n 的矩阵，如果一个元素为 0 ，则将其所在行和列的所有元素都设为 0 。请使用 原地 算法。

*void setZeroes(int\*\* matrix, int matrixSize, int\* matrixColSize){*

*int \*save1=malloc(sizeof(int)\*matrixSize);*

*int \*save2=malloc(sizeof(int)\*(\* matrixColSize));*

*int i,j;*

*for (i=0;i<matrixSize;i++){*

*for (j=0;j<\* matrixColSize;j++){*

*if (matrix[i][j] == 0) {*

*save1[i]=1;*

*save2[j]=1;*

*}*

*}*

*}*

*for (i=0;i<matrixSize;i++){*

*for (j=0;j<\* matrixColSize;j++){*

*if (save1[i]==1||save2[j]==1){*

*matrix[i][j]=0;*

*}*

*}*

*}*

*}*

*func setZeroes(matrix [][]int)  {*

*m,n:= len(matrix),len(matrix[0])*

*save1,save2 := make([]bool,m),make([]bool,n)*

*for i:=0;i<m;i++{*

*for j:=0;j<n;j++{*

*if matrix[i][j] == 0 {*

*save1[i],save2[j]=true,true*

*}*

*}*

*}*

*for i:=0;i<m;i++{*

*for j:=0;j<n;j++{*

*if save1[i]==true||save2[j]==true{*

*matrix[i][j]=0*

*}*

*}*

*}*

*}*

1. 给定一个字符串 s ，找到 它的第一个不重复的字符，并返回它的索引 。如果不存在，则返回 -1 。

*int firstUniqChar(char \* s){ (sizeof计算了字符串指针的长度，应该用strlen，sizeof可以用类型做参数，strlen只能用char\*做参数，且必须是以''\0''结尾的。)*

*int ms[26]={0};*

*int n = strlen(s);*

*for(int i=0;i<n;i++){*

*ms[s[i]-'a']++;*

*}*

*for(int j=0;j<n;j++){*

*if(ms[s[j]-'a']==1){*

*return j;*

*}*

*}*

*return -1;*

*}*

*func firstUniqChar(s string) int {*

*ms := make([]int,26)*

*for \_,value:= range s{*

*value = value-'a'*

*ms[value]++*

*}*

*for k,value2:=range s{*

*if ms[value2-'a']==1{*

*return k*

*}*

*}*

*return -1*

*}*

1. 给你两个字符串：ransomNote 和 magazine ，判断 ransomNote 能不能由 magazine 里面的字符构成。如果可以，返回 true ；否则返回 false 。magazine 中的每个字符只能在 ransomNote 中使用一次。

*bool canConstruct(char \* ransomNote, char \* magazine){*

*int ms[26] = {0};*

*int m = strlen(ransomNote);*

*int n = strlen(magazine);*

*for(int i=0;i<n;i++){*

*ms[magazine[i]-'a']++;*

*}*

*for(int i=0;i<m;i++){*

*ms[ransomNote[i]-'a']--;*

*if(ms[ransomNote[i]-'a']==-1){*

*return false;*

*}*

*}*

*return true;*

*}*

*func canConstruct(ransomNote string, magazine string) bool {*

*mr1:=make(map[int]int,26)*

*m:=len(magazine)*

*for i:=0;i<m;i++{*

*mr1[int(magazine[i]-'a')]++*

*}*

*n:=len(ransomNote)*

*for j:=0;j<n;j++{*

*mr1[int(ransomNote[j]-'a')]--*

*if mr1[int(ransomNote[j]-'a')]==-1{*

*return false*

*}*

*}*

*return true*

*}*

1. 给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。

注意：若 s 和 t 中每个字符出现的次数都相同，则称 s 和 t 互为字母异位词。

*func isAnagram(s string, t string) bool {*

*ms :=make([]int,26)*

*m := len(s)*

*n := len(t)*

*if m!=n{*

*return false*

*}*

*for i:=0;i<n;i++ {*

*ms[t[i]-'a']++*

*ms[s[i]-'a']--*

*}*

*for i:=0;i<26;i++{*

*if ms[i]!=0{*

*return false*

*}*

*}*

*return true*

*}*

*bool isAnagram(char \* s, char \* t){*

*int ms[26] = {0};*

*int m = strlen(s);*

*int n = strlen(t);*

*if(m!=n)*

*return false;*

*int i;*

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

*ms[t[i]-'a']++;*

*ms[s[i]-'a']--;*

*}*

*for(i=0;i<26;i++){*

*if(ms[i]!=0)*

*return false;*

*}*

*return true;*

*}*

1. 将两个升序链表合并为一个新的 升序 链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

*struct ListNode {*

*int val;*

*struct ListNode \*next;*

*};*

*struct ListNode\* mergeTwoLists(struct ListNode\* list1, struct ListNode\* list2){*

*if (list1==NULL){*

*return list2;*

*}*

*if (list2==NULL){*

*return list1;*

*}*

*struct ListNode \*list3 ;*

*if (list1->val <=list2->val){*

*list3=list1;*

*list1=list1->next;*

*}else{*

*list3=list2;*

*list2=list2->next;*

*}*

*struct ListNode \* tmp =list3;*

*for (;list1!=NULL||list2!=NULL;){*

*if (list1==NULL&&list2!=NULL){*

*list3->next=list2;*

*return tmp;*

*}*

*if (list2==NULL&&list1!=NULL){*

*list3->next=list1;*

*return tmp;*

*}*

*if (list1->val <=list2->val){*

*list3->next=list1;*

*list1=list1->next;*

*}else{*

*list3->next=list2;*

*list2=list2->next;*

*}*

*list3=list3->next;*

*}*

*return tmp;*

*}*

*func mergeTwoLists(list1 \*ListNode, list2 \*ListNode) \*ListNode {*

*if list1==nil{*

*return list2*

*}*

*if list2==nil{*

*return list1*

*}*

*list3 :=new(ListNode)*

*if list1.Val <=list2.Val{*

*list3.Next=list1*

*list1=list1.Next*

*}else{*

*list3.Next=list2*

*list2=list2.Next*

*}*

*tmp :=list3.Next*

*list3 = tmp*

*for ;list1!=nil||list2!=nil;{*

*if list1==nil&&list2!=nil{*

*list3.Next=list2*

*return tmp*

*}*

*if list2==nil&&list1!=nil{*

*list3.Next=list1*

*return tmp*

*}*

*if list1.Val <=list2.Val{*

*list3.Next=list1*

*list1=list1.Next*

*list3=list3.Next*

*}else{*

*list3.Next=list2*

*list2=list2.Next*

*list3=list3.Next*

*}*

*}*

*return tmp*

*}*

1. 环形链表

*bool hasCycle(struct ListNode \*head) {*

*struct ListNode \*p1;*

*struct ListNode \*p2;*

*p1=p2=head;*

*while(p1!=NULL&&p2!=NULL){*

*if(p1->next==NULL||p2->next==NULL)*

*return false;*

*p1=p1->next->next;*

*p2=p2->next;*

*if (p1==p2){*

*return true;*

*}*

*}*

*return false;*

*}*

*func hasCycle(head \*ListNode) bool {*

*var p1,p2 \*ListNode*

*p1,p2=head,head*

*for ;p1!=nil&&p2!=nil;{*

*if(p1.Next==nil||p2.Next==nil){*

*return false*

*}*

*p1=p1.Next.Next*

*p2=p2.Next*

*if p1==p2{*

*return true*

*}*

*}*

*return false*

*}*

1. 给你一个链表的头节点 head 和一个整数 val ，请你删除链表中所有满足 Node.val == val 的节点，并返回 新的头节点

*struct ListNode\* removeElements(struct ListNode\* head, int val){*

*struct ListNode \*first;*

*while(head!=NULL&&head->val==val){*

*head=head->next;*

*}*

*if(head==NULL){*

*return head;*

*}*

*first=head;*

*while(first->next != NULL){*

*if(first->next->val==val){*

*first->next=first->next->next;*

*}else {*

*first=first->next;*

*}*

*}*

*return head;*

*}*

*func removeElements(head \*ListNode, val int) \*ListNode {*

*var first \*ListNode*

*first = head*

*for ;head!=nil&&head.Val==val;{*

*head=head.Next*

*}*

*if head==nil{*

*return head*

*}*

*for ;first.Next != nil;{*

*if first.Next.Val==val{*

*first.Next=first.Next.Next*

*}else{*

*first=first.Next*

*}*

*}*

*return head*

*}*

1. 给你单链表的头节点 head ，请你反转链表，并返回反转后的链表。

*struct ListNode\* reverseList(struct ListNode\* head){*

*struct ListNode\* ntmp,\*pre,\*uu;*

*uu = head;*

*pre= NULL;*

*while(uu!=NULL){*

*ntmp=uu->next;*

*uu->next=pre;*

*pre=uu;*

*uu = ntmp;*

*}*

*return pre;*

*}*

*func reverseList(head \*ListNode) \*ListNode {*

*var pre,cur,ntmp \*ListNode*

*pre = nil*

*cur = head*

*for ;cur!=nil;{*

*ntmp = cur.Next*

*cur.Next = pre*

*pre = cur*

*cur = ntmp*

*}*

*return pre*

*}*

1. 给定一个已排序的链表的头 head ， 删除所有重复的元素，使每个元素只出现一次 。返回 已排序的链表 。

*struct ListNode\* deleteDuplicates(struct ListNode\* head){*

*struct ListNode\* pre=head;*

*int tmp;*

*while(pre!=NULL&&pre->next!=NULL){*

*tmp = pre->next->val;*

*if(pre->val==tmp){*

*pre->next=pre->next->next;*

*}else{*

*pre=pre->next;*

*}*

*}*

*return head;*

*}*

*func deleteDuplicates(head \*ListNode) \*ListNode {*

*pre := head*

*tmp := 0*

*for ;pre!=nil&&pre.Next!=nil;{*

*tmp = pre.Next.Val*

*if pre.Val==tmp{*

*pre.Next=pre.Next.Next*

*}else{*

*pre=pre.Next*

*}*

*}*

*return head*

*}*

1. (使用栈)给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。有效字符串需满足：左括号必须用相同类型的右括号闭合。左括号必须以正确的顺序闭合。

char pairs(char a) {

*if (a == '}') return '{';*

*if (a == ']') return '[';*

*if (a == ')') return '(';*

*return 0;*

*}*

*bool isValid(char\* s) {*

*int n = strlen(s);*

*if (n % 2 == 1) {*

*return false;*

*}*

*int stk[n + 1], top = 0;*

*for (int i = 0; i < n; i++) {*

*char ch = pairs(s[i]);*

*if (ch) {*

*if (top == 0 || stk[top - 1] != ch) {*

*return false;*

*}*

*top--;*

*} else {*

*stk[top++] = s[i];*

*}*

*}*

*return top == 0;*

*}*

*func isValid(s string) bool {*

*n := len(s)*

*if n % 2 == 1 {*

*return false*

*}*

*pairs := map[byte]byte{*

*')': '(',*

*']': '[',*

*'}': '{',*

*}*

*stack := []byte{}*

*for i := 0; i < n; i++ {*

*if pairs[s[i]] > 0 {*

*if len(stack) == 0 || stack[len(stack)-1] != pairs[s[i]] {*

*return false*

*}*

*stack = stack[:len(stack)-1]*

*} else {*

*stack = append(stack, s[i])*

*}*

*}*

*return len(stack) == 0*

*}*

1. 给你一棵二叉树的根节点 root ，翻转这棵二叉树，并返回其根节点。

*void invert(struct TreeNode\* root){*

*struct TreeNode \*tmp;*

*if(root != NULL){*

*tmp = root->left;*

*root->left = root->right;*

*root->right = tmp;*

*}else{*

*return;*

*}*

*invert(root->left);*

*invert(root->right);*

*}*

*struct TreeNode\* invertTree(struct TreeNode\* root){*

*struct TreeNode \*tmp;*

*tmp = root;*

*invert(root);*

*return tmp;*

*}*

*func invert(root \*TreeNode){*

*var tmp \*TreeNode*

*if root != nil{*

*tmp = root.Left*

*root.Left = root.Right*

*root.Right = tmp*

*}else{*

*return*

*}*

*invert(root.Left)*

*invert(root.Right)*

*}*

*func invertTree(root \*TreeNode) \*TreeNode {*

*var tmp \*TreeNode*

*tmp = root*

*invert(root)*

*return tmp*

*}*

1. 给你二叉树的根节点 root 和一个表示目标和的整数 targetSum 。判断该树中是否存在 根节点到叶子节点 的路径，这条路径上所有节点值相加等于目标和 targetSum 。如果存在，返回 true ；否则，返回 false 。

*bool hasPathSum(struct TreeNode\* root, int targetSum){*

*if(root==NULL){*

*return false;*

*}else{*

*if(root->left==NULL&&root->right==NULL){*

*return targetSum==root->val;*

*}*

*return hasPathSum(root->left,targetSum-root->val)||hasPathSum(root->right,targetSum-root->val);*

*}*

*}*

*/\*\**

*\* Definition for a binary tree node.*

*\* type TreeNode struct {*

*\*     Val int*

*\*     Left \*TreeNode*

*\*     Right \*TreeNode*

*\* }*

*\*/*

*func hasPathSum(root \*TreeNode, targetSum int) bool {*

*if root == nil{*

*return false*

*}else if root.Left==nil&&root.Right==nil{*

*return targetSum == root.Val*

*}*

*return hasPathSum(root.Left,targetSum-root.Val)||hasPathSum(root.Right,targetSum-root.Val)*

*}*

1. 给定一个二叉树，找出其最大深度。二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

*int max(struct TreeNode\* root,int n){*

*if(root==NULL){*

*return n;*

*}*

*return fmax(max(root->right,n),max(root->left,n))+1;*

*}*

*int maxDepth(struct TreeNode\* root){*

*if(root==NULL){*

*return 0;*

*}*

*int n=0;*

*return max(root,n);*

*}*

*func maxDepth(root \*TreeNode) int {*

*if root==nil{*

*return 0*

*}*

*return fmax(maxDepth(root.Left),maxDepth(root.Right))+1*

*}*

*func fmax(a,b int) int{*

*if a>b{*

*return a*

*}*

*return b*

*}*

1. 给你一个二叉树的根节点 root ， 检查它是否轴对称。

*bool cmp(struct TreeNode\* left,struct TreeNode\* right){*

*if(left==NULL&&right==NULL)*

*return true;*

*if(left == NULL||right==NULL||left->val!=right->val)*

*return false;*

*return cmp(left->left,right->right) && cmp(left->right,right->left);*

*}*

*bool isSymmetric(struct TreeNode\* root){*

*if(root==NULL){*

*return true;*

*}*

*return cmp(root->left,root->right);*

*}*

*func cmp(left,right \*TreeNode)bool{*

*if left==nil&&right==nil{*

*return true*

*}*

*if left == nil||right==nil||left.Val!=right.Val{*

*return false*

*}*

*return cmp(left.Left,right.Right) && cmp(left.Right,right.Left)*

*}*

*func isSymmetric(root \*TreeNode) bool {*

*if root==nil{*

*return true*

*}*

*return cmp(root.Right,root.Left)*

*}*

1. 给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

*int singleNumber(int\* nums, int numsSize){*

*int result =0;*

*for(int i=0;i<numsSize;i++){*

*result ^= nums[i];*

*}*

*return result;*

*}*

*func singleNumber(nums []int) int {*

*result:=0*

*for i,\_:=range nums{*

*result ^=nums[i]*

*}*

*return result*

*}*

1. 给定一个大小为 n 的数组 nums ，返回其中的多数元素。多数元素是指在数组中出现次数 大于 ⌊ n/2 ⌋ 的元素。你可以假设数组是非空的，并且给定的数组总是存在多数元素。

*int cmp(int\*a,int\*b){*

*return \*a-\*b;*

*}*

*int majorityElement(int\* nums, int numsSize){*

*qsort(nums,numsSize,sizeof(int),cmp);*

*return nums[numsSize/2];*

*}*

*func majorityElement(nums []int) int {*

*mr:=make(map[int]int)*

*for \_,value := range nums{*

*mr[value]++*

*if mr[value]>len(nums)/2{*

*return value*

*}*

*}*

*return 0*

*}*