leetcode大赛

176.第二高的薪水（SQL）
```
SELECT
    IFNULL(
      (SELECT DISTINCT Salary
       FROM Employee
       ORDER BY Salary DESC
        LIMIT 1 OFFSET 1),
    NULL) AS SecondHighestSalary
```

182. 查找重复的电子邮箱（SQL）
```
Select Email from Person group by Email Having count(Email)>1
```

185. 部门工资前三高的所有员工（SQL）
```
select Department,Employee,Salary
from (
    select b.Name as Department,a.Name as Employee,a.Salary,
    dense_rank()over(partition by b.Name order by a.Salary desc) ladder
    from Employee as a join Department as b
    on a.DepartmentId = b.Id) M
    where ladder<=3
```

184. 部门工资最高的员工（SQL）
```
SELECT
    Department.name AS 'Department',
    Employee.name AS 'Employee',
    Salary
FROM
    Employee
        JOIN
    Department ON Employee.DepartmentId = Department.Id
WHERE
    (Employee.DepartmentId , Salary) IN
    (   SELECT
            DepartmentId, MAX(Salary)
        FROM
            Employee
        GROUP BY DepartmentId
        )
;
```

1. 两数之和（**golang**）
```
func twoSum(nums []int, target int) []int {
	m:=[]int{}
	s :=len(nums)
	for index:=0;index<s;index++ {
		f := target - nums[index]
		for i,v := range nums{
			if f==v && i!=index {
				m = append(m, index,i)
				return m
			}
		}
	}
	return m
}
```

7. 整数反转（js）

```
var reverse = function(x) {
    var abs = false
    if(x<0) {x = -x;abs = true}
    x = x.toString()
    t = x.split("")
    t = t.reverse();
    x = t.join("")
    x = Number(x)
    if(x>Math.pow(2,31)-1)
    {
        return 0
    }
    if(abs)
    {
        x = 0-x
    }
    return x
};
```

9. 回文数（js）

```
var isPalindrome = function(x) {
    if(x<0)
    {
        return false
    }
    x = x.toString()
    var t = x.split("")
    t.reverse()
    var xx = t.join("")
    if (-1!=xx.indexOf(x) && xx.length == x.length){
        return true
    }
    return false
};
```

20. 有效的括号（js）

```
var isValid = function(s) {
    var tsize = 0
    var size = s.length
    while(tsize!=size)
    {
        size = tsize
        s = s.replace(/\(\)|\[\]|\{\}/g,"");
        tsize = s.length
    }
    return s.length==0
};
```

26. 删除排序数组中的重复项（C++）

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int l = nums.size();
        if( l<=0) return 0;
        int i = 0;
        for(auto k = 0 ; k<l;k++){
            if(nums[i]!=nums[k])
            {
                i++ ;
                nums[i] = nums[k]  ;
            }
        }
        return i+1;
    }
};
```

27. 移除元素（js）

```
var removeElement = function(nums, val) {
    var i = nums.length-1;
    while(i>=0)
    {
        if(val == nums[i])
        {
            nums.splice(i,1)
        }
        i--;

    }
    return nums.length

};
```

459. 重复的子字符串（js）

```
var repeatedSubstringPattern = function(s) {

    var t =s+s
    t = t.slice(1,t.length)
    var pindex = t.indexOf(s)
    if(pindex==s.length-1)
    {
        return false
    }
    return true
};
```

665. 非递减数列（js）

```
var checkPossibility = function(nums) {
    var size = nums.length
    if(size<1) return false
    if(size==2) return true
    var count = 0
    for(var i=0;i<size-1;i++){
        if(!(nums[i] <= nums[i + 1]))
        {
            if(0==i) //自己向后同步
            {
                nums[i] = nums[i+1]
            }
            else if(i>0 && (nums[i] > nums[i-1]) && (nums[i-1] <= nums[i+1])){ //自己向前同步，再探索一
次
                nums[i] = nums[i-1]
                i--
            }
            else if(i>0 && (nums[i] >= nums[i+1])){//只能后向自己同步
                nums[i+1] = nums[i] //解决
            }
            else{
                return false //解决不了
            }
            count++
        }
        if(count>=2){
            return false
        }
    }
    return count<2
};
```

876. 链表的中间结点（C++）

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* pmid = head;
        ListNode* pnext = head;
        int count = 0;
        while(pnext)
        {
            pnext = pnext->next;
            count++;
            if(0==count%2)
            {pmid = pmid->next;}
        }
        return pmid;
    }
};
```

1207. 独一无二的出现次数（golang）

```
func uniqueOccurrences(arr []int) bool {
            s:=make([]int,2000)
            for _,v := range arr {
                        v = v + 1000
                        if(0==s[v]){
                                    s[v] = 1
                        } else {
                                    s[v] = s[v]+1
                        }
            }
            sort.Ints(s)
            for i:=0;i<len(s)-1;i++ {
                        if s[i] ==s[i+1] && s[i]!=0 {
                                    return false
                        }
            }
            return true
}
```

1431. 拥有最多糖果的孩子（golang）

```
func kidsWithCandies(candies []int, extraCandies int) []bool {
            news :=make([]int,len(candies))
            copy(news,candies)
            sort.Ints(news)
            max := news[len(news)-1]
            r := []bool{}
            for _,v:= range candies {
                    sum := v + extraCandies
                    r = append(r, sum>=max) ;
            }
            return r
}
```

1486. 数组异或操作（C++）

```
class Solution {
public:
    int xorOperation(int n, int start) {
        int r = start;
        int i = 1;
        while(n-1>0)
        {
            r = r ^ (start + i * 2);
            n--,i++;
        }
        return r;
    }
};
```

892. 三维形体的表面积（C++）

```
class Solution {
public:
    int surfaceArea(vector<vector<int>>& grid) {
        int dr[]{0, 1, 0, -1};
        int dc[]{1, 0, -1, 0};

        int N = grid.size();
        int ans = 0;

        for (int r = 0; r < N; ++r)
            for (int c = 0; c < N; ++c)
                if (grid[r][c] > 0) {
                    ans += 2;
                    for (int k = 0; k < 4; ++k) {
                        int nr = r + dr[k];
                        int nc = c + dc[k];
                        int nv = 0;
                        if (0 <= nr && nr < N && 0 <= nc && nc < N)
                            nv = grid[nr][nc];

                        ans += max(grid[r][c] - nv, 0);
                    }
                }

        return ans;
    }
};
```

606. 根据二叉树创建字符串（C++）

```
class Solution {
public:
    string tree2str(TreeNode* t) {
        if (nullptr == t)
        {
            return "";
        }
        const string s = std::to_string(t->val);
        const string left = tree2str(t->left);
        const string right = tree2str(t->right);

        if (nullptr == t->left && nullptr == t->right)
        {
            return s;
        }

        if (nullptr == t->right)
        {
            return s + "(" + left + ")";
        }

        return s + "(" + left + ")" + "(" + right + ")";
    }
};
```

268. 缺失数字（C++）

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        long long sum = 0;
        int n = nums.size();
        for(int i = 0; i < n; i++)
        {
            sum += i;
            sum -= nums[i];
        }
        sum += n;
        return sum;
    }
};
```

2. 两数相加（golang）

```
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
            add:=0
            r := list.New()
            for ;l1!=nil || l2!=nil || add!=0;  {
                        s:=0
                        if nil!=l1 {
                                    s = s+l1.Val
                                    l1=l1.Next
                        }
                        if nil!=l2{
                                    s = s+l2.Val
                                    l2=l2.Next
                        }
                        s = s+ add;
                        add = 0
                        if s>=10{
                                    s=s-10
                                    add = 1
                        }
                        r.PushBack(s)
            }
            o := list.New()
            for ; r.Len()>0;{
                        var x ListNode
```

```
                    x.Val=r.Back().Value.(int)
                    r.Remove(r.Back())
                    x.Next=nil
                    if(nil!=o.Front()){
                            a :=o.Front().Value.(ListNode)
                            x.Next = &a
                    }
                    o.PushFront(x)
        }
        rr:= o.Front().Value.(ListNode)
        return &rr
}
```

16. 最接近的三数之和（C++）

```
class Solution {
public:

    int threeSumClosest(vector<int>& nums, int target) {
        map<int,int> ky;
        int min = nums[0]+nums[1]+nums[2];
        sort(nums.begin(),nums.end());
        for(auto i=0;i<nums.size();i++)
        {
            for(auto x=i+1;x<nums.size();x++)
            {
                for(auto y=0;y<nums.size();y++)
                {
                    if(i!=x && x!=y && i!=y)
                    {
                        int v = nums[i]+nums[x]+nums[y];
                        int k = abs(v-target);
                        if(k < abs(min-target))
                        {
                            min = v;
                        }
                        else if(k < abs(min-target) && min>target )
                        {
                             min = v;
                             continue;
                        }
                        if(0==k)
                        {
                            return min;
                        }

                    }
                }
            }
        }
        return min;

    }
};
```

29. 两数相除（js）

```
var divide = function(dividend, divisor) {
    const max = Math.pow(2,31)-1
    abs = dividend < 0
    abs = abs ^ (divisor < 0)
    dividend= Math.abs(dividend)
    divisor= Math.abs(divisor)
    //建立一个减法快除表
    sub = new Array()
    p = 0
    sub[p] = divisor
    do{
        r = sub[sub.length-1] + sub[sub.length-1]
        sub[++p] = r
    }
    while(r<max && r<dividend)
    //用快除表去除，然后记录总数
    i = 0
    while(dividend>=divisor)
    {
        for(p=sub.length+1;p>=0;)
        {
            if(dividend>=sub[p])
            {
                dividend = dividend - sub[p]
                i = i+Math.pow(2,p)
            }
            else{
                p--
            }
        }
    }
    if(abs)
    {
        i = 0-i
    }
    if(i>max)
    {i=max}
    return i
};
```

63. 不同路径 II（js）

```
var count = 0
var Grid = []


var uniquePathsWithObstacles = function(obstacleGrid) {


    Grid = JSON.parse(JSON.stringify(obstacleGrid))
    for(var y=0;y<Grid.length;y++)
    {
        for(var x=0;x<Grid[y].length;x++)
        {
            Grid[y][x] = 0
        }
    }
    if(undefined===obstacleGrid[0][0] || 1 ===obstacleGrid[0][0])
    {return 0}
    else{
        Grid[0][0] = 1
    }
    for(var y=0;y<Grid.length;y++)
    {
        for(var x=0;x<Grid[y].length;x++)
        {
            if(1!=obstacleGrid[y][x])
            {
                if(y>0)
                {
                    Grid[y][x] += Grid[y-1][x]
                }
                if(x>0)
                {
                    Grid[y][x] += Grid[y][x-1]
                }
            }
        }
    }
    var r = Grid[obstacleGrid.length-1][Grid[obstacleGrid.length-1].length-1]
    return r
};
```

131. 分割回文串（js）

```
var partition = function(s) {
    var all = new Array()
    for(var i=1;i<=s.length;i++)
    {
        var sub = s.slice(0,i);
        if(sub.length>0 && sub == sub.split("").reverse().join(""))
        {
            ssub = s.slice(i,s.length)
            if(ssub.length>0)
            {
                var rr = partition(ssub)
                rr.forEach((el)=>{
                    var r = new Array()
                    r.push(sub)
                    if (el instanceof Array) {
                        el.forEach((el1)=>{
                            r.push(el1)
                        })
                    }
                    else {r.push(el)}
                    all.push(r)
                })
            }
            else{
                var r = new Array()
                r.push(sub)
                all.push(r)
            }

        }
    }
    return all
};
```

133. 克隆图（C++）

```
class Solution {
public:
    Node* cloneGraph(Node* node) {
        if(NULL!=node){
            unordered_map<int,Node*>::iterator it = set.find(node->val);
            if(it!=set.end())
            {
                return (*it).second;
            }
            Node* Head=new Node(node->val);
            set[Head->val]=Head;
            vector<Node*> cp = vector<Node*>();
            for(vector<Node*>::iterator it=node->neighbors.begin();it!=node->neighbors.end();it++)
            {
                Node* p = cloneGraph(*it);
                cp.push_back(p);
            }
            Head->neighbors = cp;
            return Head;
        }
        return NULL;
    }
    unordered_map<int,Node*> set;
};
```

139. 单词拆分（C++）

```
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
        vector<bool> dp(s.size()+1, false);
            unordered_set<string> m(wordDict.begin(), wordDict.end());
            dp[0] = true;
            //获取最长字符串长度
            int maxWordLength = 0;
            for (int i = 0; i < wordDict.size(); ++i){
                maxWordLength = std::max(maxWordLength, (int)wordDict[i].size());
            }
            for (int i = 1; i <= s.size(); ++i){
                for (int j = std::max(i-maxWordLength, 0); j < i; ++j){
                    if (dp[j] && m.find(s.substr(j, i-j)) != m.end()){
                        dp[i] = true;
                        break;
                    }
                }
            }
            return dp[s.size()];
    }
};
```

209. 长度最小的子数组（golang）

```
func minSubArrayLen(s int, nums []int) int {
            const UINT_MAX = int(^uint(0) >> 1)
            ss:=len(nums)
            minlens:=UINT_MAX
            sum :=0
            leftstep :=0
            rightstep :=0
            for ; leftstep <ss; leftstep++ {
                        sum += nums[leftstep]
                        for ; rightstep <ss ;{
                                    if sum>= s {
                                                lens := rightstep - leftstep + 1
                                                if (lens <= minlens) {
                                                            minlens = lens
                                                }
                        sum -= nums[leftstep]
                        leftstep++
                    } else {
                        rightstep++
                        if rightstep < ss {

                            sum += nums[rightstep]
                        }
                    }
                }
            }
    if UINT_MAX==minlens {minlens=0}
    return minlens
}
```

215. 数组中的第K个最大元素（js）

```
var findKthLargest = function(nums, k) {
  r = 0
  nums.sort((a,b)=>b-a)
  nums.forEach((el,i)=>{
    if(i+1==k)
    {
      r = el
    }
  })
  return r
};
```

215. 数组中的第K个最大元素（golang）

```
func findKthLargest(nums []int, k int) int {
    sort.Sort(sort.Reverse(sort.IntSlice(nums)))
    for i,v := range nums{
        if(i+1==k) {
            return v
        }
    }
    return 0
}
```

287. 寻找重复数（golang）

```
func findDuplicate(nums []int) int {
  slow, fast := 0, 0
  for slow, fast = nums[slow], nums[nums[fast]]; slow != fast; slow, fast = nums[slow],
nums[nums[fast]] { }
  slow = 0
  for slow != fast {
    slow = nums[slow]
    fast = nums[fast]
  }
  return slow
}
```

287. 寻找重复数（js）

```
var findDuplicate = function(nums) {
  let slow = 0, fast = 0;
  do {
    slow = nums[slow];
    fast = nums[nums[fast]];
  } while (slow != fast);
  slow = 0;
  while (slow != fast) {
        slow = nums[slow];
        fast = nums[fast];
    }
    return slow;
};
```

378. 有序矩阵中第K小的元素（js）

```
var kthSmallest = function(matrix, k) {

    var t = new Array()
    for(var i = 0; i< matrix.length;i++){
        t = t.concat(matrix[i])
    }
    t.sort((a,b)=>{return a-b})
    return t[k-1];

};
```

718. 最长重复子数组（js）

```
var findLength = function(A, B) {
    var max=0
    if(A.length==0) {return max}
    size = A.length
    var ll= 0 ,rr=ll+1;
    var X = A.slice(ll,rr)
    while(X.length<=size || X.length<0)
    {
        var a = X.toString()
        var b = B.toString()
        a = ","+a+","
        b = ","+b+","
        if(-1!=b.indexOf(a)){
            var t = true
            X.forEach(element => {
                t = t && B.includes(element)
            });
            if(t) {
                max = max>X.length?max:X.length
                if(rr<size){rr++}else{
                    break;
                }
            }
            else{
                ll++
                if(rr==ll && rr<size)(rr++)
            }
        }
        else{
            ll++
            if(rr==ll && rr<size)(rr++)
        }
        if(ll==rr){
            break
        }
        X = A.slice(ll,rr)
    }
    return max
};
```

18. 四数之和（C++）

```
class Solution{
        public:
        vector<vector<int>> fourSum(vector<int>& nums, int target) {
    sort(nums.begin(),nums.end());
    vector<vector<int> > res;
    if(nums.size()<4)
    return res;
    int a,b,c,d,_size=nums.size();
    for(a=0;a<=_size-4;a++){
        if(a>0&&nums[a]==nums[a-1]) continue;      //确保nums[a] 改变了
        for(b=a+1;b<=_size-3;b++){
                if(b>a+1&&nums[b]==nums[b-1])continue;   //确保nums[b] 改变了
                c=b+1,d=_size-1;
                while(c<d){
                        if(nums[a]+nums[b]+nums[c]+nums[d]<target)
                            c++;
                        else if(nums[a]+nums[b]+nums[c]+nums[d]>target)
                            d--;
                        else{
                                res.push_back({nums[a],nums[b],nums[c],nums[d]});
                                while(c<d&&nums[c+1]==nums[c])      //确保nums[c] 改变了
                                    c++;
                                while(c<d&&nums[d-1]==nums[d])      //确保nums[d] 改变了
                                    d--;
                                c++;
                                d--;
                                        }
                                }
                        }
                }
                return res;
  }
};
```

31. 下一个排列（C++）

```
class Solution {
public:
  void nextPermutation(vector<int>& nums) {
    int low=-1,high=nums.size()-1;
    for(int i=0;i<nums.size()-1;++i)
    {
      if(nums[i] < nums[i+1])
      {
        low = i;
      }
    }
    for(int i=low;i<nums.size();++i)
    {
      if(nums[i] > nums[low])
      {
        high = i;
      }
    }
    if(low>=0 && high<nums.size())
      swap(nums[low],nums[high]);
    low++;high=nums.size()-1;
    while(low < high)
```

```
        {
            swap(nums[low],nums[high]);
            low++;high--;
        }
    }
};
```

36. 有效的数独（C++）

```
class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        vector<int> wow(9,0);
        int mux1;
        int mux2;
        int mux3;
        int box_index;

        for(int i=0;i<9;i++){
            for(int j=0;j<9;j++){
                if(board[i][j] == '.'){
                    continue;
                }
                mux1 = 0x01 << (board[i][j] - '1');
                mux2 = 0x01 << 9 << (board[i][j] - '1');
                mux3 = 0x01 << 9 << 9 << (board[i][j] - '1');
                box_index = (i/3) * 3 + j/3;
                if((wow[i]&mux1) != mux1 && (wow[j]&mux2) != mux2 && (wow[box_index]&mux3) !=
mux3){
                    wow[i] = wow[i]|mux1;
                    wow[j] = wow[j]|mux2;
                    wow[box_index] = wow[box_index]|mux3;
                }
                else{
                    return false;
                }
            }
        }
        return true;
    }
};
```

32. 最长有效括号（js）

```
var longestValidParentheses = function(s) {
    var tsize = 0
    var size = s.length
    while(tsize!=size)
    {
        size = tsize
        var ll = s.match(/\([\s]*\)/)
        var str = ""
        for (var i = 0; ll!=null && i < ll[0].length-1; i++) {
            str += " ";
        }
        s = s.replace(/\([\s]*\)/,str);
        tsize = s.length
    }
    //return s.length==0
    var max = 0
    var lll = 0
    var rrr = lll
    while (lll<s.length-rrr)
    {
        for(rrr = 0;lll+rrr<s.length;)
        {
            if(" "==s[lll+rrr])
            {
                max = max>(rrr+1)?max:(rrr+1)
                rrr++
            }
            else
            {
                lll = lll+rrr+1
                break
            }
        }
    }
    return max*2

}
```

41. 缺失的第一个正数（golang）

```
func firstMissingPositive(nums []int) int {
        i:=1
        find:=false
        for find==false {
                find = true
                for index := 0; index < len(nums);index++ {
                        v := nums[index]
                        if(i>=v){
                                if v>0{find=false};
                                nums = append(nums[:index], nums[index+1:]...)
                                index--
                        }
                }
                if find {
                        return i
                }
                i++
        }
        return i
}
```

4. 寻找两个正序数组的中位数（C++）

```
class Solution {
public:
    int getKthElement(const vector<int>& nums1, const vector<int>& nums2, int k) {
        /* 主要思路：要找到第 k (k>1) 小的元素，那么就取 pivot1 = nums1[k/2-1] 和 pivot2 = 
nums2[k/2-1] 进行比较
         * 这里的 "/" 表示整除
         * nums1 中小于等于 pivot1 的元素有 nums1[0 .. k/2-2] 共计 k/2-1 个
         * nums2 中小于等于 pivot2 的元素有 nums2[0 .. k/2-2] 共计 k/2-1 个
         * 取 pivot = min(pivot1, pivot2)，两个数组中小于等于 pivot 的元素共计不会超过 (k/2-1) + (k/
2-1) <= k-2 个
         * 这样 pivot 本身最大也只能是第 k-1 小的元素
         * 如果 pivot = pivot1，那么 nums1[0 .. k/2-1] 都不可能是第 k 小的元素。把这些元素全部 "删
除"，剩下的作为新的 nums1 数组
```

```
         * 如果 pivot = pivot2，那么 nums2[0 .. k/2-1] 都不可能是第 k 小的元素。把这些元素全部 "删除"，剩下的作为新的 nums2 数组
         * 由于我们 "删除" 了一些元素（这些元素都比第 k 小的元素要小），因此需要修改 k 的值，减去删除的数的个数
         */

        int m = nums1.size();
        int n = nums2.size();
        int index1 = 0, index2 = 0;

        while (true) {
            // 边界情况
            if (index1 == m) {
                return nums2[index2 + k - 1];
            }
            if (index2 == n) {
                return nums1[index1 + k - 1];
            }
            if (k == 1) {
                return min(nums1[index1], nums2[index2]);
            }

            // 正常情况
            int newIndex1 = min(index1 + k / 2 - 1, m - 1);
            int newIndex2 = min(index2 + k / 2 - 1, n - 1);
            int pivot1 = nums1[newIndex1];
            int pivot2 = nums2[newIndex2];
            if (pivot1 <= pivot2) {
                k -= newIndex1 - index1 + 1;
                index1 = newIndex1 + 1;
            }
            else {
                k -= newIndex2 - index2 + 1;
                index2 = newIndex2 + 1;
            }
        }
    }

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int totalLength = nums1.size() + nums2.size();
        if (totalLength % 2 == 1) {
            return getKthElement(nums1, nums2, (totalLength + 1) / 2);
        }
        else {
            return (getKthElement(nums1, nums2, totalLength / 2) + getKthElement(nums1, nums2, totalLength / 2 + 1)) / 2.0;
        }
    }
};
```

25. K 个一组翻转链表（js）

```
const myReverse = (head, tail) => {
    let prev = tail.next;
    let p = head;
    while (prev !== tail) {
        const nex = p.next;
        p.next = prev;
        prev = p;
        p = nex;
    }
    return [tail, head];
}
var reverseKGroup = function(head, k) {
    const hair = new ListNode(0);
    hair.next = head;
    let pre = hair;

    while (head) {
        let tail = pre;
        // 查看剩余部分长度是否大于等于 k
        for (let i = 0; i < k; ++i) {
            tail = tail.next;
            if (!tail) {
                return hair.next;
            }
        }
        const nex = tail.next;
        [head, tail] = myReverse(head, tail);
        // 把子链表重新接回原链表
        pre.next = head;
        tail.next = nex;
        pre = tail;
        head = tail.next;
    }
    return hair.next;
};
```

37. 解数独（golang）

```
func solveSudoku(board [][]byte) {
    var line, column [9][9]bool
    var block [3][3][9]bool
    var spaces [][2]int

    for i, row := range board {
        for j, b := range row {
            if b == '.' {
                spaces = append(spaces, [2]int{i, j})
            } else {
                digit := b - '1'
                line[i][digit] = true
                column[j][digit] = true
                block[i/3][j/3][digit] = true
            }
        }
    }

    var dfs func(int) bool
    dfs = func(pos int) bool {
        if pos == len(spaces) {
            return true
        }
        i, j := spaces[pos][0], spaces[pos][1]
        for digit := byte(0); digit < 9; digit++ {
            if !line[i][digit] && !column[j][digit] && !block[i/3][j/3][digit] {
                line[i][digit] = true
                column[j][digit] = true
                block[i/3][j/3][digit] = true
                board[i][j] = digit + '1'
                if dfs(pos + 1) {
                    return true
                }
                line[i][digit] = false
                column[j][digit] = false
                block[i/3][j/3][digit] = false
            }
        }
        return false
    }
    dfs(0)
}
```

42. 接雨水（C++）

```
class Solution{
public:
int trap(vector<int>& height) {
    int n = height.size();
    int ans = 0;
    for (int i = 1; i < n - 1; i++) {
        int l_max = 0, r_max = 0;
        // 找右边最高的柱子
        for (int j = i; j < n; j++)
            r_max = max(r_max, height[j]);
        // 找左边最高的柱子
        for (int j = i; j >= 0; j--)
            l_max = max(l_max, height[j]);
        // 如果自己就是最高的话,
        // l_max == r_max == height[i]
        ans += min(l_max, r_max) - height[i];
    }
    return ans;
}
};
```