力扣刷题笔记

力扣刷题笔记
- 最长回文子串
  - 动态规划
  - 扩展中心法
  - 马拉车算法
- 双指针
  - 两数之和 II - 输入有序数组
- 树
  - 二叉树前序中序构造二叉树（I）
  - 二叉树的中序遍历（I）
- 寻找重复数（环表操作）
  - 快慢指针（链表判圈法、龟兔赛跑法）
- 栈
  - 394 字符串编码
  - 柱状图中最大的矩形
- 图
  - 顺时针打印矩阵
  - 搜索二维矩阵（图、分治）
- 动态规划
  - 买股票的最佳时机
  - 最大子序和
  - 最长上升子序列
    - 我的解法——O(n^3)
    - 长度记录法——O(n^2)
- 贪心算法
  - 跳跃游戏 I
  - 跳跃游戏 II
  - 加油站
- 摩尔投票法
  - 多数元素

> I标志代表迭代表示，R标志代表递归

最长回文子串

链接

动态规划

我的动态规划——O(n^2)：

```
int len = s.length();
            if(len<=1)return s;
            vector<vector<pair<int,int>>> grp(len, vector<pair<int,int>>(len,pair<int,int>(0,0)));
```

```
            for (int i = 0; i < len; ++i) {
                grp[i][i] = make_pair(i,i);
                if(i+1<len&&s[i]==s[i+1])grp[i][i+1]=make_pair(i,i+1);
            }
            for (int gap = 2; gap < len; ++gap) {
                for (int i = 0; i + gap< len; ++i) {
                    int j = i + gap;
                    pair<int,int> p = grp[i+1][j-1];
                    if(s[i]==s[j]&&i+1==p.first&&j-1==p.second)
                        grp[i][j] = make_pair(i,j);
                    else {
                        pair<int,int> a = grp[i][j-1], b = grp[i+1][j];
                        int da=a.second-a.first,db=b.second-b.first,dc=p.second-p.first;
                        grp[i][j] = p;
                        if(da>dc&&da>=db)grp[i][j] = make_pair(a.first,a.second);
                        else if(db>dc&&db>da)grp[i][j] = make_pair(b.first,b.second);
                    }
                }
            }
            return s.substr(grp[0][len-1].first,grp[0][len-1].second - grp[0][len-1].first + 1);
```

别人的动态规划：

```
string longestPalindrome(string s) {
        int n = s.size();
        vector<vector<int>> dp(n, vector<int>(n));
        string ans;
        for (int l = 0; l < n; ++l) {
            for (int i = 0; i + l < n; ++i) {
                int j = i + l;
                if (l == 0) {
                    dp[i][j] = 1;
                }
                else if (l == 1) {
                    dp[i][j] = (s[i] == s[j]);
                }
                else {
                    dp[i][j] = (s[i] == s[j] && dp[i + 1][j - 1]);
                }
                if (dp[i][j] && l + 1 > ans.size()) {
                    ans = s.substr(i, l + 1);
                }
            }
        }
        return ans;
    }
```

执行结果： 通过 显示详情 >

执行用时： 464 ms ，在所有 C++ 提交中击败了 28.15% 的用户

内存消耗： 192.3 MB ，在所有 C++ 提交中击败了 5.95% 的用户

修正版：

```
string longestPalindrome(string s) {
```

```
        int len = s.length();
        if(len<=1)return s;
        vector<vector<int>> grp(len, vector<int>(len));
        string ans = "";
        for (int gap = 0; gap < len; ++gap) {
              for (int i = 0; i + gap< len; ++i) {
                    int j = i + gap;
                    if(gap==0||(gap==1&&s[i]==s[j])||(s[i]==s[j]&&grp[i+1][j-1]))grp[i][j]=1;
                    if(grp[i][j]&&gap + 1>ans.length())ans=s.substr(i,gap+1);
              }
        }
        return ans;
    }
```

grp使用int型

执行结果： 通过 显示详情 >

执行用时： 476 ms ，在所有 C++ 提交中击败了 26.41% 的用户

内存消耗： 192.5 MB ，在所有 C++ 提交中击败了 5.95% 的用户

grp使用bool型：

执行结果： 通过 显示详情 >

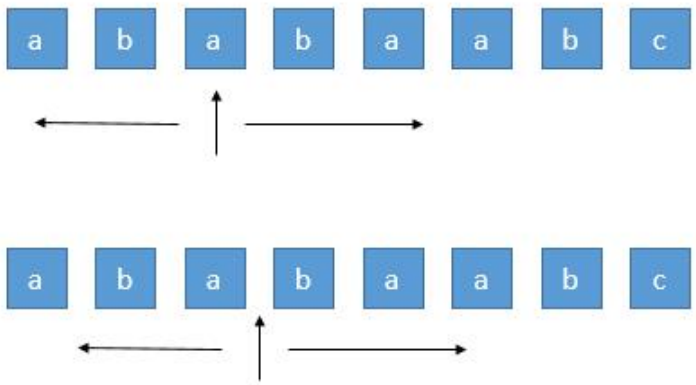
执行用时： 980 ms ，在所有 C++ 提交中击败了 9.86% 的用户

内存消耗： 24.6 MB ，在所有 C++ 提交中击败了 58.33% 的用户

扩展中心法

> 时间复杂度：O(n²) ,空间复杂度：O(1)

由于回文字符串是对称的，所以我们可以选取字符串的中心开始，可以是一个字符开始或者是两个字符开始。



马拉车算法

> 时间复杂度：O(n) ,空间复杂度：O(n)

双指针

两数之和 II - 输入有序数组

两数之和 II - 输入有序数组

使用缩减空间法

```
vector<int> twoSum(vector<int>& numbers, int target) {
      int i = 0, j = numbers.size() - 1;
      while(1){
            if(numbers[i]+numbers[j]==target)return{i+1,j+1};
            if(numbers[i]+numbers[j]<target)i++;
            else j--;
      }
}
```

树

二叉树前序中序构造二叉树（I）

题解

二叉树的中序遍历（I）

代码一：

```
void midTraverseL(Node *t) {
      stack<Node *> s;
      s.push(t);
      while (!s.empty()) {
            Node *tmp = s.top();
            s.pop();
            if (tmp->right)s.push(tmp->right);
            if (tmp->left || tmp->right) {
                  Node *t = new Node;
                  t->c = tmp->c;
                  s.push(t);
            } else cout << tmp->c;
            if (tmp->left)s.push(tmp->left);
      }
}
```

代码二：

```
public class Solution {
      public List < Integer > inorderTraversal(TreeNode root) {
            List < Integer > res = new ArrayList < > ();
```

```
            Stack < TreeNode > stack = new Stack < > ();
            TreeNode curr = root;
            while (curr != null || !stack.isEmpty()) {
                  while (curr != null) {
                        stack.push(curr);
                        curr = curr.left;
                  }
                  curr = stack.pop();
                  res.add(curr.val);
                  curr = curr.right;
            }
            return res;
      }
}
```

寻找重复数（环表操作）

快慢指针（链表判圈法、龟兔赛跑法）

栈

394 字符串编码

地址

> 给定一个经过编码的字符串，返回它解码后的字符串。
>
> 编码规则为: `k[encoded_string]`，表示其中方括号内部的 encoded_string 正好重复 k 次。注意 k 保证为正整数。
>
> 你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。
>
> 此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 `3a` 或 `2[4]` 的输入。

考点：栈

算法复杂度：O(n)

以前犯的错误：在向栈中保存数据的时候，什么都想往栈中保存，其实没必要，只要保留对应的界定符"[" "]"即可，由界定符的index就可以确定两者之间的字符串。

```
string decodeString(string s) {
      string str=s;
      stack<int> left;
      int i=0;
      while(i<str.length()){
            if(str[i]=='[')left.push(i);
            else if(str[i]==']'){
                  string tail = str.substr(i+1);
                  int index = left.top() - 1,j=1,sum = 0;
```

```
                while(index>=0&&isdigit(str[index])){
                      sum = sum + (str[index]-'0') * j;
                      j *= 10;
                      index--;
                }
                string tmp, dup = str.substr(left.top()+1,i-left.top()-1);
                for (int k = 0; k < sum; ++k) tmp+=dup;
                if(index>=0)tmp = str.substr(0,index+1).append(tmp);
                i = tmp.length() - 1;
                str= tmp.append(tail);
                left.pop();
            }
             i++;

     }
      return str;
}
```

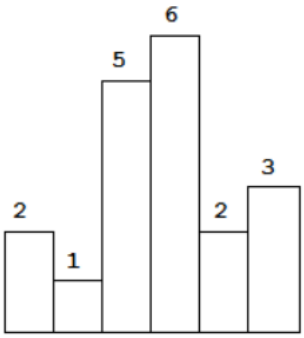
递归方法：

```
string analysis(string s, int& index) {
      string res;
      int num = 0;
      string temp;
      while (index < s.size()) {
            if (s[index] >= '0' && s[index] <= '9') {
                  num = 10 * num + s[index] - '0';
            }
            else if (s[index] == '[') {
                  temp = analysis(s, ++index);//碰到'[',开始递归
                  while(num-->0) res += temp;
                  num = 0;//num清零
            }
            else if (s[index] == ']') break;//碰到']',结束递归
            else res += s[index];
            index++;
     }
      return res;
}
```

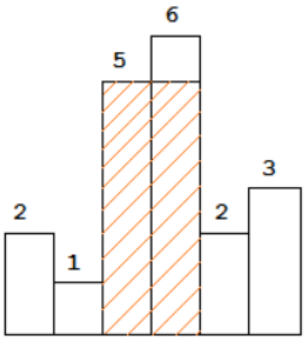
柱状图中最大的矩形

柱状图中最大的矩形

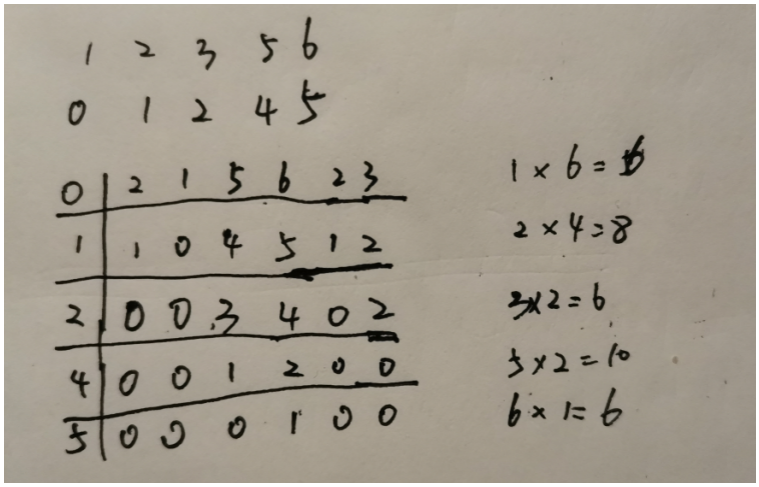
求在该柱状图中，能够勾勒出来的矩形的最大面积。



以上是柱状图的示例，其中每个柱子的宽度为 1，给定的高度为 [2, 1, 5, 6, 2, 3]。



我的解法，时间复杂度O(n^2)，空间复杂度O(n)——应付几千个数据没问题，几万个就会超时。




```
int largestRectangleArea(vector<int>& heights) {
      if(heights.size()==0)return 0;
      vector<int> stage = heights;
      sort(stage.begin(),stage.end());// 排序,
      int i=1,j=1,last=stage[0],n=heights.size();
      if(stage[0]>0)stage[0]--;
      while(j<n){
            stage[i]=stage[j]-1<0?0:stage[j]-1; //去重+减一
            if(stage[j]!=last){
                  last=stage[j];
                  i++;
                  j++;
            }else j++;
     }
      vector<int> c=heights;
      int maxi = -1;
```

```
        for(int k=0;k<i;k++){
            int gap = k==0?stage[0]:stage[k]-stage[k-1],coh=0,max_coh=0;    //coh 连续>0的个数
            for(int index=0;index<n;index++){
                c[index]=c[index]-gap<0?0:c[index]-gap;     //看哪个连续的最多
                if(c[index]==0)coh=0;
                else coh++;
                max_coh=coh>max_coh?coh:max_coh;    //更新最多连续的次数
            }
            maxi=(stage[k]+1)*max_coh>maxi?(stage[k]+1)*max_coh:maxi;   //面积公式 (k+1)*coh
        }
        return maxi;
    }
```

图

顺时针打印矩阵

链接

体会：说实话，看见这个标签“简单”，本以为没什么问题，做了30分钟才做出来。心态就炸了，开始着急，结果越慌越忙，越忙越慌，反而做不好了。首先需要在纸上模拟一遍，不要靠脑子空想，真当你是无敌的吗？

关于这个题：刚开始的出错原因是类似与“先斩后奏”这种操作，先输出，在移动指针，导致指针越界之后就无法进行下一次移动，在脑子中的流程的指针是不越界的，这两者想的和实际做出来的反而相互矛盾。

改进：对于这种题，一个首先设定一个“探测”指针，判断是否越界，如果越界就不执行，不越界就访问即可。

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
            if(matrix.size()==0 || matrix[0].size()==0)return {};
            int i = 0,j = 0;
            int op[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};//记录移动步数
            int dir = 0; // 0 1 2 3 记录方向 右 下 左 上
            int rows = matrix.size(), cols = matrix[0].size(), sum = rows*cols;
            vector<vector<bool>> visited(rows,vector<bool>(cols,false));
            vector<int> ret;
            for(int cnt=0;cnt<sum;cnt++){
                visited[i][j] = 1;
                ret.push_back(matrix[i][j]);
                int ni = i + op[dir][0],nj = j + op[dir][1];
                if(ni<0 || ni>=rows || nj<0 || nj>=cols || visited[ni][nj]){
                    //越界或者被访问过就转向
                    dir = (dir + 1) % 4;
                }
                i = i + op[dir][0],j = j + op[dir][1];
            }
            return ret;
    }
```

搜索二维矩阵（图、分治）

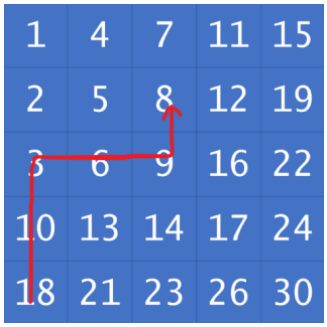
搜索二维矩阵 II

分治法：比较之后分为两个矩阵依次解决——我的方法$O(nlgn)$


```
bool search(vector<vector<int>>& matrix, int target,int si,int sj,int ei,int ej){
      int row = matrix.size(),col = matrix[0].size();
      if(si>row||ei>row||si>ei||sj>ej)return false;
      int midx=(si+ei)/2,midy=(sj+ej)/2;
      if(matrix[midx][midy]==target)return true;
      else{
            if(si==ei&&sj==ej)return false;
            if(matrix[midx][midy]>target){
                  return search(matrix,target,si,sj,midx-1,ej)||search(matrix,target,midx,sj,ei,midy-1);
            }else {
                  return search(matrix,target,si,midy+1,ei,ej)||search(matrix,target,midx+1,sj,ei,midy);
            }
      }
}

bool searchMatrix(vector<vector<int>>& matrix, int target) {
      int row = matrix.size(),col = matrix[0].size();
      return search(matrix,target,0,0,row-1,col-1);
}
```

别人的方法，路径搜索$O(m+n)$：



```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
      int row = matrix.size();
      if(row==0)return false;
      int col = matrix[0].size();
      int x = row-1,y=0;
      while(x>=0&&y<col){
            int tmp = matrix[x][y];
            if(tmp==target)return true;
            if(tmp<target)y++;
            else x--;
      }
      return false;
}
```

动态规划

买股票的最佳时机

买股票的最佳时机

思路：

经典O(n^3)。先gap在推到公式

$$m[i,j]=max\{m[i,mid],m[mid,j],num[j]-num[i]\}$$

代码：

```
int maxProfit(vector<int>& prices) {
      //添加记录数据
      int len = prices.size();
      if(len<=1)return 0;
      int **arr = new int*[len];
      for (int i = 0; i < len; ++i) {
            arr[i]=new int[len];      //arr[i][j] 表示从第i天到第j天的可以获取的最大利润(j>i)
            if(i<len-1)arr[i][i+1]=prices[i+1]-prices[i];//顺便初始化
      }
      //经典O(n^3)
      int m = 0;
      for (int gap = 1; gap < len; ++gap) {
            for (int i = 0; i + gap< len; ++i) {
                  int j = i + gap, bet = prices[j] - prices[i];
                  for (int k = i + 1; k < j; ++k) {
                        int better = max(arr[i][k], arr[k][j]);
                        bet = better > bet ? better : bet;
                  }
                  arr[i][j]=bet;
                  m=bet>m?bet:m;
            }
      }
      return m;
}
```

思路二：——O(n)

选对一个好的公式是很重要的：

$$gap\{i,j\} = max\{A_{i+1},..,A_j\} - A_i$$

代码:

```
int maxProfit(vector<int>& prices) {
      int len = prices.size();
      if(len<=1)return 0;
      int gap = 0, big = 0;
      big=prices[len-1];
      for (int i = len - 2; i >= 0; --i) {
            if(prices[i]>big)big=prices[i];
            if(big-prices[i]>gap)gap=big-prices[i];
     }
      return gap;
}
```

最大子序和

最大子序和

关键：不要上来就无脑O(n^2) O(n^3)，多深入了解一下有没有更优的递推式。

```
int maxSubArray(vector<int>& nums) {
      int p = 0,m=nums[0];
      for(int i:nums){
      p=p+i>i?p+i:i;
      if(p>m)m=p;
     }
      return m;
}
```

最长上升子序列

我的解法——O(n^3)

思想：

1. 首先初始化一个记录数组record，record[i,j]表示从第i到第j个的最长上升子序列。record防止子问题的重复计算。并且初始化record[i,i].
2. 设置gap由小到大解决子问题，先解决m[i ,i+gap] (gap={1,2,3,....,len-1})。
3. 对于未求解的子问题，再根据已求解子问题计算。m[i,j]=m[i,mid]+m[mid,j]

不足：

复杂度太高，不能AC，deprecated

代码：

```
int lengthOfLIS(vector<int>& nums) {
            int len = nums.size();
            if(len<=1)return len;
            vector<vector<vector<int>>> record;
         //初始化过程
            for (int i = 0; i < len; ++i) {
                  vector<vector<int>> a;
                  for (int j = 0; j < len; ++j) {
```

```
                        vector<int> b;
                        a.push_back(b);
                  }
                   a[i].push_back(nums[i]);
                   record.push_back(a);
            }
             //开始规划
             int max_len = -1;
             for (int gap = 1; gap < len; ++gap) {
                   for (int i = 0; i + gap< len; ++i) {
                         int j = i + gap;
                         int size = record[i][j].size();
                         for (int mid = i + 1; mid <= j; ++mid) {
                               //m[i+mid-1] + m[mid, j]
                               vector<int> a = record[i][mid-1], b = record[mid][j];
                               if((b.size()==0||a.size()==0||b[0]>a[a.size()-1]) && a.size()+b.size() >
size){
                                     a.insert(a.end(),b.begin(),b.end());
                                     record[i][j] = a;
                               }else if(a.size()>record[i][j].size()||b.size()>record[i][j].size())
                                     a.size()>b.size()?record[i][j] = a:record[i][j] = b;
                         }
                         if((int)record[i][j].size()>max_len)max_len = record[i][j].size();
                   }
             }
              return (int)record[0][len-1].size();
       }
```

长度记录法——O(n^2)

思想：

设置dp[i]为serial[0:i]最长上升子序列的长度，dp[i]=max{dp[j]}+1，其中0≤j<i

如果

代码精简

不足：

不记录上升的序列

代码：

```
int lengthOfLIS(vector<int>& nums) {
       int len = nums.size(), *dp = new int[len];
       if(len<=1)return len;
       int fmax = 1;
       for (int i = 0; i < len; ++i) {
             dp[i] = 1;
             for (int j = 0; j < i; ++j)
                   if(nums[j]<nums[i]&&dp[j]>=dp[i])
                         dp[i] = dp[j]+1;
             if(dp[i]>fmax)fmax=dp[i];
       }
        return fmax;
}
```

贪心算法

跳跃游戏 I

跳跃游戏

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

```
输入: [2,3,1,1,4]
输出: true
输入: [3,2,1,0,4]
输出: false
```

思想:

```
bool canJump(vector<int>& nums) {
      int m = 0,n=nums.size();
      for(int i=0;i<n-1;i++){
            if(i<=m){
                  m=max(i+nums[i],m);
                  if(m>=n-1)return true;
            }
     }
      return m>=n-1;
}
```

跳跃游戏 II

跳跃游戏 II

```
int jump(vector<int>& nums) {
      int m = 0, n = nums.size(),e = min(nums[0],n-1),step=0;
      if(n==1)return 0;
      for(int i=0;i<n;i++){
            m=max(m,i+nums[i]);
            if(i==e){
                  step++;
                  e = min(m,n-1);
            }
     }
      return step;
}
```

加油站

加油站

```
int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
      int n = gas.size(), start = -1, sum_gas = 0;
      int i = 0,j=2*n+1;
      while(j--){
```

```
            i%=n;
            if(i==start)return start;
            if(sum_gas+gas[i]>=cost[i]){
                  if(start==-1)start = i;
                  sum_gas+=gas[i]-cost[i];
            }else{
                  start = -1;
                  sum_gas = 0;
            }
            i++;
      }
      return -1;
}
```

时间复杂度O(2*n)

任务 ： 将算法复杂度降低到O(n)

摩尔投票法

多数元素

多数元素

```
int majorityElement(vector<int>& nums) {
      int can = nums[0], cnt = 1;
      for(int i=1;i<nums.size();i++){
            if(nums[i]==can)cnt++;
            else{
                  if(cnt==0){
                        cnt++;
                        can=nums[i];
                  }else cnt--;
            }
      }
      return can;
}
```