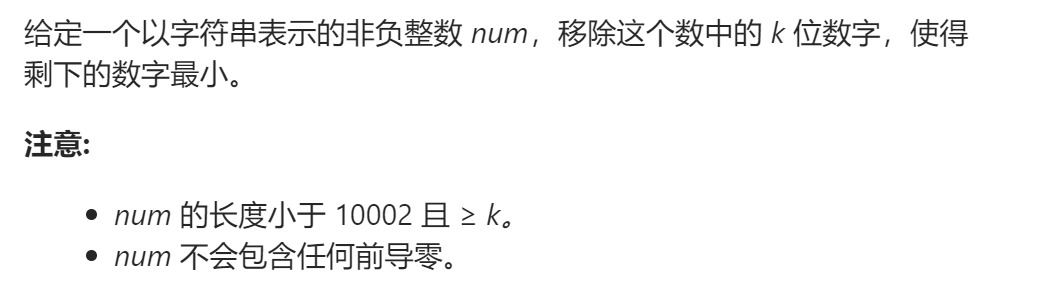
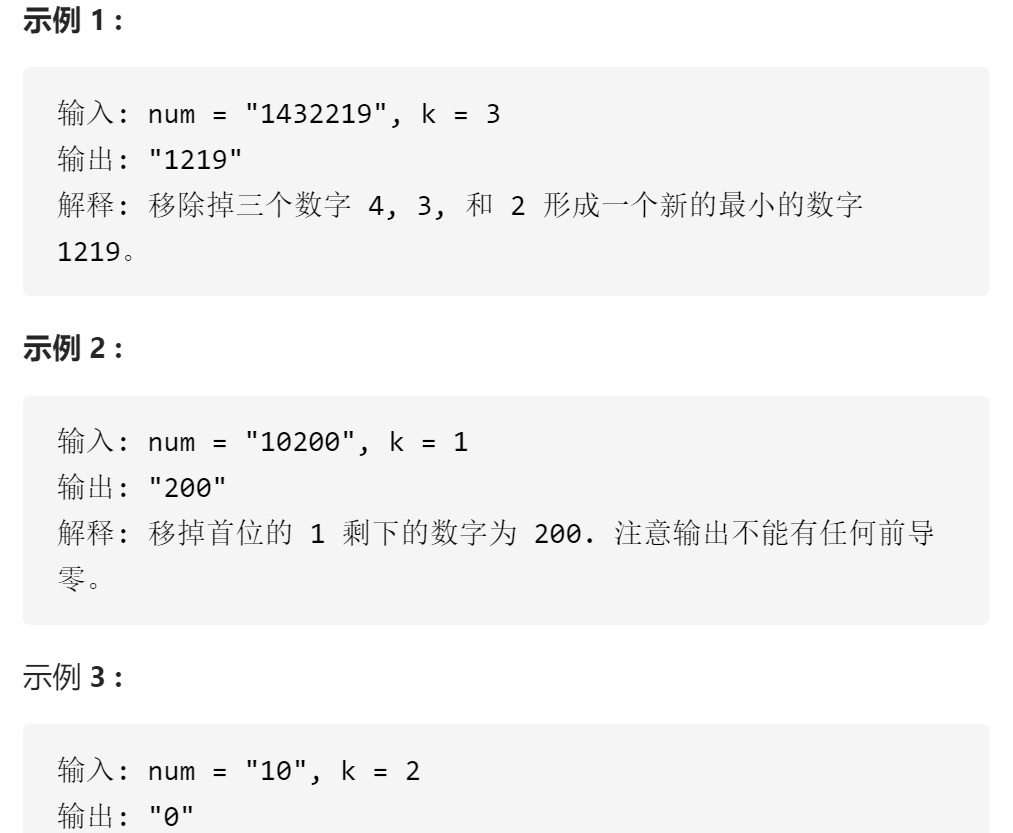
# [402. 移掉K位数字]【贪心算法】【代码功底】【用对数据结构-堆栈】

## 题目描述：





## 题目分析：【贪心算法】

1. 对于a(i)a(i-1)a(i-2)a(i-3)...a0;对于a(i)(i>0)若a(i)>=a(i-1),则a(i)可以删除；且k--;
2. 如果遍历了一遍之后,k>0；说明剩下的数是递增的；则从前向后删除；再取剩下的数；
3. 选正确的数据结构，不然很多细节方面不好处理

## 双指针法：错误的数据结构使得代码漏洞百出

class Solution {  
public:  
 string removeKdigits(string num, int k) {  
 //贪心算法  
 string res;  
 int n=num.size();  
 vector<bool>path(n,false);//用于判断改数是否取了  
 vector<bool>path2(n,false);//用于判断是否删除  
 //贪心算法对代码功底的要求比较高  
 //特殊情况处理  
 //双指针法  
 int front=0,back=0;  
 while(k&&front<n&&back<n){  
 if(front==back)front++;  
 else if(num[front]>num[back]){  
 front++;  
 //back++;  
 if(!path[back]){  
 res.push\_back(num[back]);  
 path[back]=true;//该数已经取了  
 }  
 back=front-1;  
 }else if(num[front]<=num[back]){  
 //等于的情况的特殊处理  
 if(num[front]==num[back]&&front+1<n&&num[front+1]>num[front]){  
 front++;  
 if(!path[back]){  
 res.push\_back(num[back]);  
 path[back]=true;//该数已经取了  
 }  
 back=front-1;  
 continue;  
 }  
  
 path[back]=true;//表示该数已删  
 path2[back]=true;  
  
 //此处一直有错误  
 while(back>0&&path[back])back--;  
 if(path[back])back=front;  
 k--;  
 }/\*else if(num[front]==num[back]){  
 front++;  
 back=front-1;  
 }\*/  
 }  
  
 if(k){  
 int back[2];  
 back[0]=-1;  
 back[1]=INT\_MIN;  
  
 res.clear();  
 for(int i=n-1;i>=0;i--){  
 if(k){  
 if(!path2[i]&&num[i]<=back[1]){  
 k--;  
 if(back[0]!=-1){  
 path2[back[0]]=true;  
 back[1]=num[i];  
 back[0]=i;  
 }  
 //path2[back[0]]=true;  
 }else if(!path2[i]&&num[i]>back[1]){  
 back[1]=num[i];  
 back[0]=i;  
 }  
 }else break;  
 }  
  
 if(k>0){  
 path2[0]=true;  
 k--;  
 }  
 if(k>0){  
 for(int i=n-1;i>=0;i--){  
 if(!path2[i]&&k){  
 k--;  
 path2[i]=true;  
 }  
 }  
 }  
  
 for(int i=0;i<n;i++){  
 if(!path2[i]){  
 res.push\_back(num[i]);  
 }  
 }  
 }else{  
 for(int i=front;i<n;i++){  
 res.push\_back(num[i]);  
 }  
 }  
  
 //去除前导0  
 int num0=0,i=0;  
 while(res[i++]=='0'){  
 num0++;  
 }  
 res.erase(res.begin(),res.begin()+num0);  
 //为空的情况处理  
 if(res.size()==0)res.push\_back('0');  
 return res;  
 }  
};  
  
  
//后面大数据的示例无法通过  
//优化  
class Solution {  
public:  
 string removeKdigits(string num, int k) {  
 //贪心算法  
 int n=num.size();  
 //vector向量也可以实现栈的功能  
 string stk;  
 int i=0;  
 while(k&&i<n){  
 if(stk.empty())  
 stk.push\_back(num[i]);  
 else if(stk.back()>num[i]){  
 while(stk.back()>num[i]&&k){  
 stk.pop\_back();  
 k--;  
 //stk.push\_back(num[i]);  
 }  
 stk.push\_back(num[i]);  
 }else if(stk.back()<=num[i]){  
 stk.push\_back(num[i]);  
 }  
 i++;  
 }  
  
 while(k){  
 stk.pop\_back();  
 k--;  
 }  
  
 //消除前导0  
 while(i<n){  
 stk.push\_back(num[i]);  
 i++;  
 }  
 string res;  
 int j=0;  
 while(stk[j]=='0')j++;  
 for(;j<stk.size();j++){  
 res.push\_back(stk[j]);  
 }  
  
 return res==""?"0":res;  
 }  
};

## 用Vector模板实现栈+可变长数组

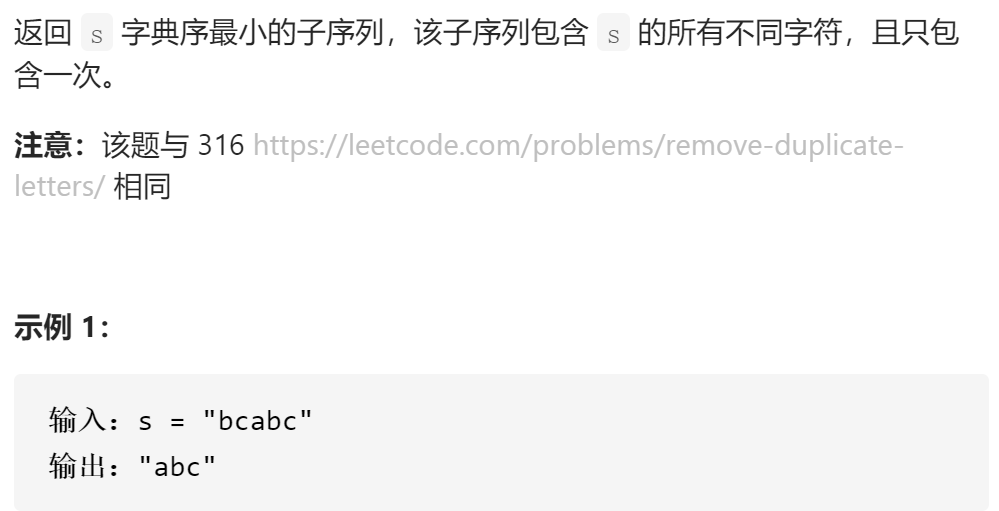
class Solution {  
public:  
 string removeKdigits(string num, int k) {  
 vector<char> stk;  
 for (auto& digit: num) {  
 while (stk.size() > 0 && stk.back() > digit && k) {  
 stk.pop\_back();  
 k -= 1;  
 }  
 stk.push\_back(digit);  
 }  
  
 for (; k > 0; --k) {  
 stk.pop\_back();  
 }  
  
 string ans = "";  
 bool isLeadingZero = true;  
 for (auto& digit: stk) {  
 if (isLeadingZero && digit == '0') {  
 continue;  
 }  
 isLeadingZero = false;  
 ans += digit;  
 }  
 return ans == "" ? "0" : ans;  
 }  
};  
  
作者：LeetCode-Solution  
链接：https://leetcode-cn.com/problems/remove-k-digits/solution/yi-diao-kwei-shu-zi-by-leetcode-solution/  
来源：力扣（LeetCode）  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 总结反思:

1. 贪心算法的题一般好想，但是代码的实现并不是很简单
2. 选择正确的数据结构
3. 养成进行代码优化的习惯

# [1081. 不同字符的最小子序列]【贪心算法】【单调栈】

## 题目描述：



## 题目分析：

1. 此题与316题类似；贪心算法+单调栈
2. 不同点：该题增加了约束条件
3. 为了判断单调栈中某字符是否存在，引入exist数组
4. 为了判断后续字符中是否还存在该字符，引入了nums数组以统计后续字符中所剩字符种类和数量
5. 需要维护exist和nums数组