**LeetCode刷题**

## Hot 100

#### 两数之和

https://leetcode.cn/problems/two-sum/书签：[力扣（LeetCode）官网 - 全球极客挚爱的技术成长平台](https://leetcode.cn/problems/two-sum/)

|  |  |
| --- | --- |
|  | 左图中提到，每种输入只会对应一种答案，那么最简单的就是穷举的思想：设立两层for循环，对数组内的元素两两相加，当和与target相等时，则break结束循环输出  进阶版的两数之和是时间复杂度小于的算法，思想如下：  通过设置map，key为数组值，value为索引，记录某值是否存在于数组  然后遍历数组，判断差值是否存在于map且索引不同于当前索引，如果存在这样的索引，那么就插入输出的vector中 |

//O(n^2)  
vector<int> twoSum(vector<int>& nums, int target) {  
 vector<int>temp;  
 for (size\_t i = 0; i < nums.size(); i++){  
 bool success=false;  
 for (size\_t j = i+1; j < nums.size(); j++){  
 if (nums[i]+nums[j]==target){  
 temp.push\_back(i);  
 temp.push\_back(j);  
 bool success=true;  
 break;  
 }  
 }  
 if (success){  
 break;  
 }  
 }  
 return temp;  
}  
  
//小于O(n^2)  
vector<int> twoSum2(vector<int>& nums, int target) {  
 vector<int>temp,\_nums;  
 map<int,int>isExit;  
 for (size\_t i = 0; i < nums.size(); i++){  
 isExit.insert(make\_pair(nums[i],i));  
 }  
   
 for (size\_t i = nums.size()-1; i >=0; i--){  
 if (isExit.find(target-nums[i])!=isExit.end()&&isExit[target-nums[i]]!=i){  
 temp.push\_back(i);  
 temp.push\_back(isExit[target-nums[i]]);  
 break;  
 }  
 }  
 return temp;  
}

C++

#### 两数相加

https://leetcode.cn/problems/add-two-numbers/书签：[力扣（LeetCode）官网 - 全球极客挚爱的技术成长平台](https://leetcode.cn/problems/add-two-numbers/)

|  |  |
| --- | --- |
|  |  |

题目要求是：两个链表逆序存储两个数，将这两个数相加，然后再按逆序存储在链表中

刚开始我的思路是先将两个链表中的数转换为整数，再运算然后再将结果转换为列表存储。这个思路看起来是不错的，但是**忽略了链表中的节点数范围是[1,100]，即数最大是，远远超出了int甚至unsigned long long的表示范围**，因此出现执行时错误的提示

之后的思路即是不转换，直接对链表对应节点上的值相加，然后大于10进位，小于10取模

这种方法在链尾多存储了一个节点，用来表示进位，当这个节点值为0时，需要删除这个节点。**为了便于删除，采用了双指针技术p和q**

/\*\*  
 \* Definition for singly-linked list.  
 \* struct ListNode {  
 \* int val;  
 \* ListNode \*next;  
 \* ListNode() : val(0), next(nullptr) {}  
 \* ListNode(int x) : val(x), next(nullptr) {}  
 \* ListNode(int x, ListNode \*next) : val(x), next(next) {}  
 \* };  
 \*/  
struct ListNode {  
 int val;  
 ListNode \*next;  
 ListNode() : val(0), next(nullptr) {}  
 ListNode(int x) : val(x), next(nullptr) {}  
 ListNode(int x, ListNode \*next) : val(x), next(next) {}  
};  
class Solution {  
public:  
 ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {  
 //节点范围在1~100之间  
 ListNode \*L=new ListNode();  
 ListNode \*p=L,\*q;  
 while(l1!=nullptr&&l2!=nullptr){  
 int val=l1->val+l2->val+p->val;  
 p->val=val%10;  
 ListNode \*temp=new ListNode(val/10);  
 temp->next=p->next;  
 p->next=temp;  
 q=p;  
 p=p->next;  
 l1=l1->next;  
 l2=l2->next;  
 }  
 if(l1!=nullptr){  
 while(l1!=nullptr){  
 int val=l1->val+p->val;  
 p->val=val%10;  
 ListNode \*temp=new ListNode(val/10);  
 temp->next=p->next;  
 p->next=temp;  
 q=p;  
 p=p->next;  
 l1=l1->next;  
 }  
 }  
 if(l2!=nullptr){  
 while(l2!=nullptr){  
 int val=l2->val+p->val;  
 p->val=val%10;  
 ListNode \*temp=new ListNode(val/10);  
 temp->next=p->next;  
 p->next=temp;  
 q=p;  
 p=p->next;  
 l2=l2->next;  
 }  
 }  
 if(p->next==nullptr&&p->val==0){  
 q->next=nullptr;  
 delete p;  
 }  
 return L;  
 }  
};

C++

#### 无重复字符的最长子串

https://leetcode.cn/problems/longest-substring-without-repeating-characters/书签：[力扣（LeetCode）官网 - 全球极客挚爱的技术成长平台](https://leetcode.cn/problems/longest-substring-without-repeating-characters/)

|  |  |
| --- | --- |
|  | 在给定字符串中查找**最长的不含重复字符**的子串  起初的做法是一次循环遍历，比如“abcabcbb”这样，在找到[3]是'a'与之前重复后，就继续从[3]找。这样的方法一个是忽略了"vdvf"这样的情况，一个是""等情况处理不了  最后的做法是**两层循环，寻找每一个字符起始的最长不含重复字符的子串长度。**  共性问题是：最后找到拼接最后一个字符时，没有更新max，需要内层循环完以后再更新max——可以合并，**去掉内层break之前的max更新，内层结束后统一更新** |

int lengthOfLongestSubstring(string s) {  
 int max=0;  
 for (int i = 0; i < s.length(); i++){  
 int cur=1;  
 string temp=s.substr(i,1);  
 for (int j = i+1; j < s.length(); j++){  
 if (temp.find\_first\_of(s[j])!=-1){  
 break;  
 }else{  
 cur++;  
 temp=s.substr(i,j-i+1);  
 }  
 }  
 if (max<cur){//最后一个字符的拼接  
 max=cur;  
 }  
 }  
 return max;  
}

C++

#### 寻找两个正序数组的中位数



这个题比较简单，示例即给出了解法：先合并两个vector，然后排序。对于偶数size，则取中间的两个；对于奇数size，则取最中间的即可

代码中我用到的是algorithm库中的sort函数，复杂度为

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {  
 vector<int>temp=nums1;  
 for (int i = 0; i < nums2.size(); i++) {  
 temp.push\_back(nums2[i]);  
 }  
 sort(temp.begin(),temp.end());//O((m+n)log(m+n))  
 if (temp.size()%2){  
 return temp[temp.size()/2];  
 }else{  
 return (temp[temp.size()/2]+temp[temp.size()/2-1])/2.0;  
 }  
}

C++

还有种做法是，遍历数组，双指针合并，边合并边排序，复杂度为

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {  
 vector<int>temp;  
 int j=0,k=0;  
 for (; j < nums1.size()&&k<nums2.size();) {  
 if(nums1[j]<nums2[k]){  
 temp.push\_back(nums1[j]);  
 ++j;  
 }else{  
 temp.push\_back(nums2[k]);  
 ++k;  
 }  
 }  
 if (j<nums1.size()){  
 for (int i = j; i < nums1.size(); i++){  
 temp.push\_back(nums1[i]);  
 }  
 }  
 if (k<nums2.size()){  
 for (int i = k; i < nums2.size(); i++){  
 temp.push\_back(nums2[i]);  
 }  
 }  
 if (temp.size()%2){  
 return temp[temp.size()/2];  
 }else{  
 return (temp[temp.size()/2]+temp[temp.size()/2-1])/2.0;  
 }   
   
}

C++

#### 最长回文子串

|  |  |
| --- | --- |
|  | 大致的思路就是暴力求解，列出所有的子串，然后一一判断是不是回文串（子函数判断）  为了减小一些时间复杂度，可以利用一个变量表示当前最长的回文字符子串，当裁剪的字符串长度小于该变量时，就不执行裁剪→可以直接通过循环遍量设置  外层i：0→size-cur；内层:cur+i到size |

C++暴力超时，代码如下，接下来试试C

从最简单无限制的三层循环，到根据cur，来控制不裁剪比cur小的字符子串→均超时

class Solution {  
public:  
 bool isPalindrome(string s,int size){  
 bool sign=true;  
 for (int i = 0; i < size/2; i++){  
 if (s[i]!=s[size-1-i]){  
 sign=false;  
 break;  
 }  
 }  
 return sign;  
 }  
 string longestPalindrome(string s) {  
 string res="";  
 res.append(1,s[0]);  
 int cur=1,size=s.length();  
 for (int i = 0; i < size-cur; i++) {//cur长的最后几个字符就不遍历了  
 for (int j = cur+i; j < size; j++) {  
 string temp=s.substr(i,j-i+1);  
 if (isPalindrome(temp,j-i+1)){  
 res=temp;  
 cur=j-i+1;  
 }   
 }  
 }  
 return res;  
 }  
};

C++

C是可以的，主要是**要注意'\0'的处理，char\*要比字符串长度多1**；然后**对于长数组，放在全局声明区域声明，放在函数内部**，会导致栈内存超出。

* 1. **数组如果大小较大(级别)，则需要定义到主函数的外面，否则会使程序异常退出**

原因是：函数内部申请的局部变量来自于系统栈，允许申请的空间较小；  
 函数外部申请的全局变量来自静态存储区，允许申请的空间较大。

char res[1001];  
char temp[1001];  
  
int isPalindrome(char \*s){  
 int size=strlen(s);  
 int sign=1;  
 for (int i = 0; i < size/2; i++){  
 if (s[i]!=s[size-1-i]){  
 sign=0;  
 break;  
 }  
 }  
 return sign;  
}  
  
char \* longestPalindrome(char \* s){  
 int size=strlen(s);  
 memset(res,'\0',size+1);  
 res[0]=s[0];  
 int cur=1;  
 for (int i = 0; i < size-cur; i++){  
 for (int j = cur+i; j<size; j++){  
 strncpy(temp,s+i,j-i+1);  
 temp[j-i+1]='\0';  
 if(isPalindrome(temp)){  
 strncpy(res,temp,j-i+1);  
 cur=j-i+1;  
 }  
 }  
   
 }  
 return res;  
}

C++

#### N字形变换

https://leetcode.cn/problems/zigzag-conversion/description/书签：[力扣（LeetCode）官网 - 全球极客挚爱的技术成长平台](https://leetcode.cn/problems/zigzag-conversion/description/)

|  |  |
| --- | --- |
| 这个题比较简单，找到规律填充数组即可 |  |

char res[1000][1000]={{0}};  
 string convert(string s, int numRows) {  
 string temp=s;  
 if (numRows!=1){  
 int one\_size=2\*numRows-2,one\_cow=numRows-1;  
 int group=s.size()%one\_size==0?s.size()/one\_size:s.size()/one\_size+1;  
 int row=numRows,col=group\*one\_cow;  
 int i=1;  
 while (i<=s.size()){  
 int cycle=(i-1)/one\_size;  
 for (int j = 0; j < numRows&&i<=s.size(); j++) {  
 res[j][cycle\*one\_cow]=s[i-1];  
 ++i;  
 }  
 for (int j = numRows-2; j >=1&&i<=s.size(); j--) {  
 res[j][(i-(cycle\*one\_size)-numRows)+cycle\*one\_cow]=s[i-1];  
 ++i;  
 }  
 }  
 i=1;  
 for (int j = 0; j < numRows; j++) {  
 for (int k = 0; k < col; k++){  
 if (res[j][k]!=0){  
 temp[i-1]=res[j][k];  
 ++i;  
 if (i>s.size()){  
 break;  
 }  
 }  
 }  
 if (i>s.size()){  
 break;  
 }  
 }  
 }  
 return temp;  
 }

C++

#### 整数反转

|  |  |
| --- | --- |
|  | 利用stringstream类来实现字符串和数字的转换，从而实现倒序  对正整数和负数要分别处理  正整数：倒序后开头的0，当数值不是0时要省去。然后省去以后的字符串要先和INTMAX转换的字符串比较→长度大于等于且字符串大于则认为越界，最后再将字符串转整数输出  负整数：保留负号，对剩下的值按照正整数的处理方式处理，最后的字符串和-拼接合并，再和INTMIN比较→比较规则同上  先比较再转换是因为直接转换会溢出 |

int reverse(int x) {  
 stringstream ss;  
 ss<<x<<endl;  
 string s;  
 ss>>s;  
 int z;  
 if (x<0){  
 string temp=s.substr(1,s.size()-1);  
 ::reverse(temp.begin(),temp.end());  
 int i=0;  
 while (temp[i]=='0'){  
 ++i;  
 }  
 int k=i,j;  
 for (j = 1; j <=temp.size()-k; j++) {  
 s[j]=temp[i];  
 ++i;  
 }  
 s[j]='\0';  
 string temp1;  
 ss<<INT32\_MIN<<endl;  
 ss>>temp1;  
 if (s.length()>=temp1.length()&&s>temp1){  
 return 0;  
 }  
 ss<<s<<endl;  
 ss>>z;  
 }else{  
 ::reverse(s.begin(),s.end());  
 int i=0;  
 while (s[i]=='0'&&x!=0){  
 ++i;  
 }  
 string temp=s.substr(i,s.size()-i);  
 string temp1;  
 ss<<INT32\_MAX<<endl;  
 ss>>temp1;  
 if (temp.length()>=temp1.length()&&temp>temp1){  
 return 0;  
 }  
 ss<<temp<<endl;  
 ss>>z;  
 }  
 return z;  
}

C++

## 分类刷题