# 链表

## 35，翻转链表

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: n

\* @return: The new head of reversed linked list.

\*/

ListNode \* reverse(ListNode \* head) {

// write your code here

if(head==NULL)

{

return NULL;

}

ListNode\*cur,\*pre,\*next;

pre=head;

cur=pre->next;

while(cur)

{

next=cur->next;

cur->next=pre;

pre=cur;

cur=next;

}

head->next=NULL;

head=pre;

return head;

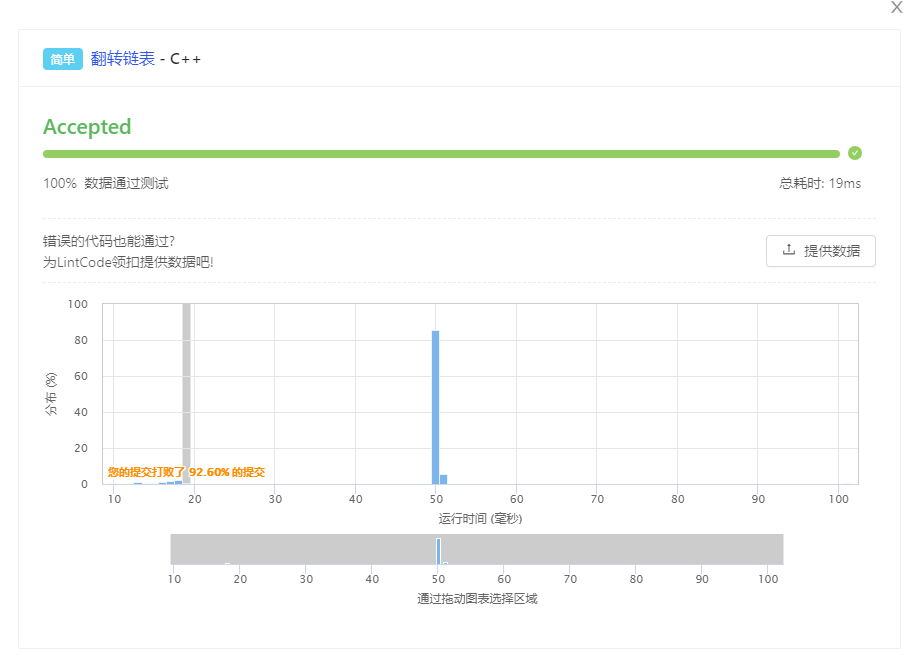
}

};

**2）思路 ：**

遍历链表中所有的节点，依次将各节点从链表中抽出，插到头结点和第一个数据节点之间，直到最后一个节点插完

**3）测试通过截图：**



## 112.删除链表中重复的元素

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: head is the head of the linked list

\* @return: head of linked list

\*/

ListNode \* deleteDuplicates(ListNode \* head) {

// write your code here

if(head==NULL)

return NULL;

ListNode \*cur=head;

ListNode \*pre=head;

while(cur!=NULL)

{

if(pre->val==cur->val)

{

pre->next=cur->next;

cur=pre->next;

}

else{

pre=cur;

cur=pre->next;

}

}

return head;

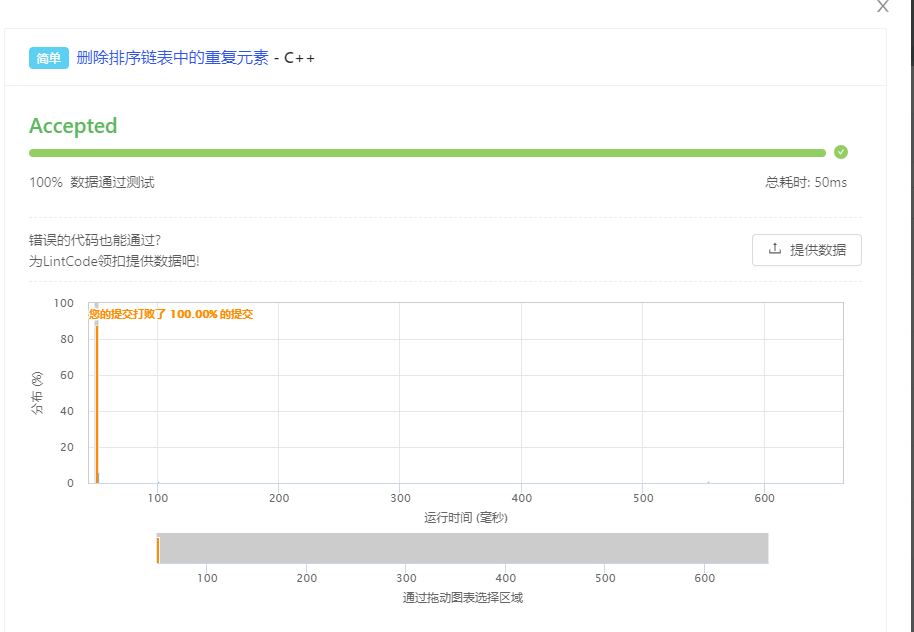
}

};

**2）思路：**

设置两个指针，一个指向当前的节点，一个指向后面的节点，如果是相等的节点，当前节点不动，后面的节点向后移动；遇到不相等的节点，当前节点指向后面的节点，后面的节点再指向下一个。这样循环，直到链表结束。

**3）测试通过截图：**



## 165,.合并两个排序链表

**1）代码：**

class Solution {

public:

/\*\*

\* @param l1: ListNode l1 is the head of the linked list

\* @param l2: ListNode l2 is the head of the linked list

\* @return: ListNode head of linked list

\*/

ListNode \* mergeTwoLists(ListNode \* l1, ListNode \* l2) {

// write your code here

if(l2==NULL)

{

return l1;

}

else if(l1==NULL)

{

return l2;

}

if(l1->val<l2->val)

{

l1->next=mergeTwoLists(l1->next,l2);

return l1;

}

else{

l2->next=mergeTwoLists(l1,l2->next);

return l2;

}

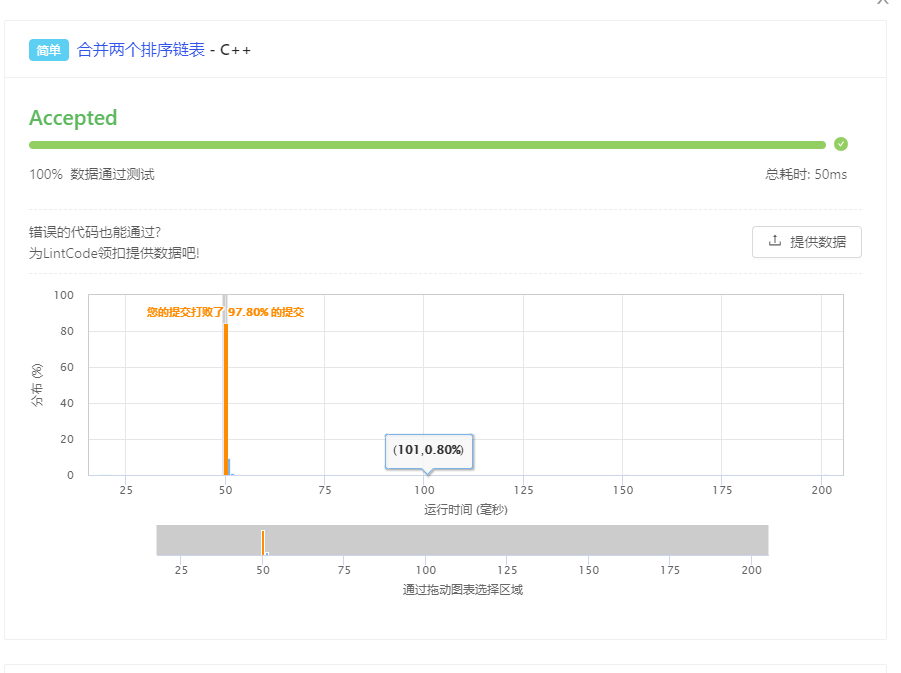
}

};

**2）思路：**

先考虑l1和l2中其中一个为NULL的情况，直接返回另一个，对于l1和l2都不为NULL的情况，先合并两个列表，然后排序

**3）测试通过截图：**



## 166.链表倒数第n个节点

**1）代码：**

class Solution {

public:

/\*

\* @param head: The first node of linked list.

\* @param n: An integer

\* @return: Nth to last node of a singly linked list.

\*/

ListNode \* nthToLast(ListNode \* head, int n) {

// write your code here

if (head == NULL || n == 0)

{

return NULL;

}

ListNode\* pAhead = head;

ListNode\* pBehind = NULL;

for (int i = 0; i < n-1; i++)

{

if (pAhead->next != NULL)

{

pAhead = pAhead->next;

}

else

{

return NULL;

}

}

pBehind = head;

while(pAhead->next != NULL)

{

pAhead = pAhead->next;

pBehind = pBehind->next;

}

return pBehind;

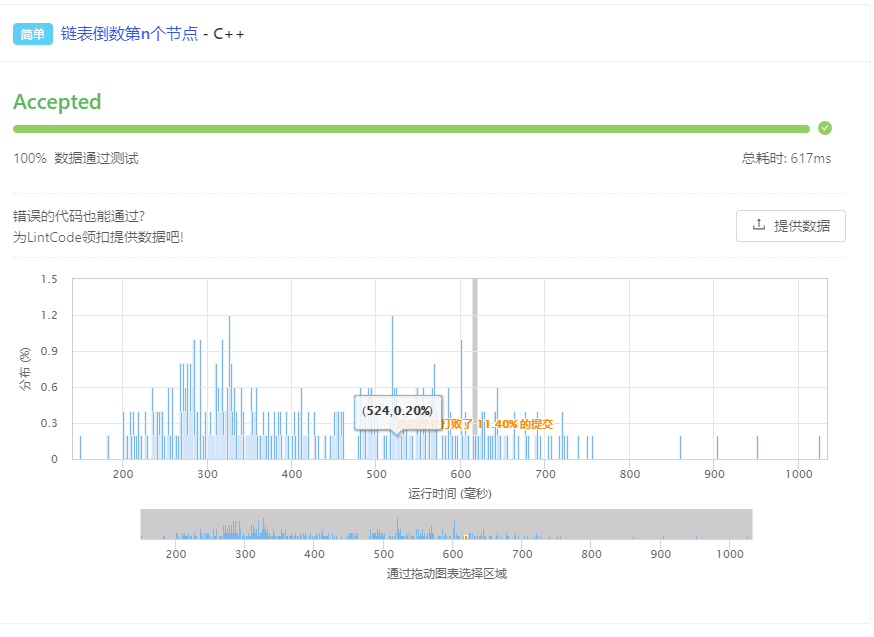
}

};

**2）思路：**

定义两个指针，第一个指针从链表的头指针开始遍历向前走n-1；第二个指针保持不动，从第n步开始，第二个指针也开始遍历，两个指针差距n-1个距离，当第一个指针走到了尾巴节点，第二个指针正好在倒数第n个节点

**3）测试通过截图：**



## 167.链表求和

**1）代码：**

class Solution {

public:

/\*\*

\* @param l1: the first list

\* @param l2: the second list

\* @return: the sum list of l1 and l2

\*/

ListNode \* addLists(ListNode \* l1, ListNode \* l2) {

// write your code here

ListNode \*head = new ListNode(-1), \*p1 = l1, \*p2 = l2, \*p = head;

int sum = 0, carr = 0;

while (p1 || p2 || carr)

{

sum = 0;

if(p1)

{

sum += (p1->val);

p1 = p1->next;

}

if(p2)

{

sum += (p2->val);

p2 = p2->next;

}

sum += carr;

ListNode \*t = new ListNode(sum % 10);

carr = sum / 10;

p->next = t;

p = p->next;

}

return head->next;

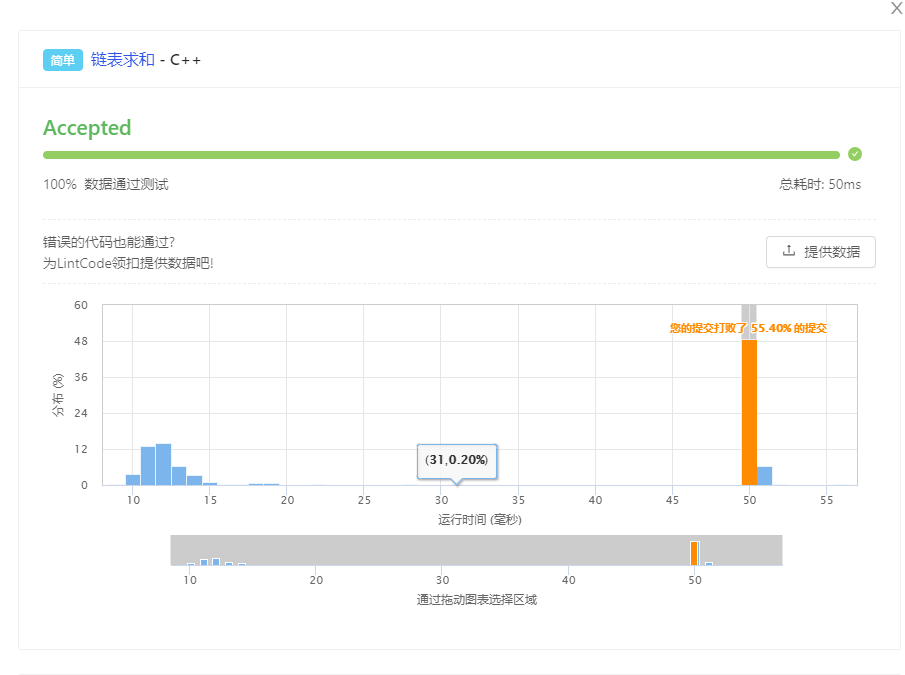
}

};

**2）思路：**

先对应位求和，加上上一次的进位，得到当前位，记录进位，当位数大的数遍历完且进位为0的时候停止。

**3）测试通过截图：**



## 173.链表插入排序

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: The first node of linked list.

\* @return: The head of linked list.

\*/

ListNode \* insertionSortList(ListNode \* head) {

// write your code here

if(head==NULL)

return NULL;

ListNode \*first=new ListNode(0);

while(head!=NULL)

{

ListNode \*p=first;

while(p->next!=NULL&&p->next->val<head->val)

{

p=p->next;

}

ListNode \*tmp=head->next;

head->next=p->next;

p->next=head;

head=tmp;

}

return first->next;

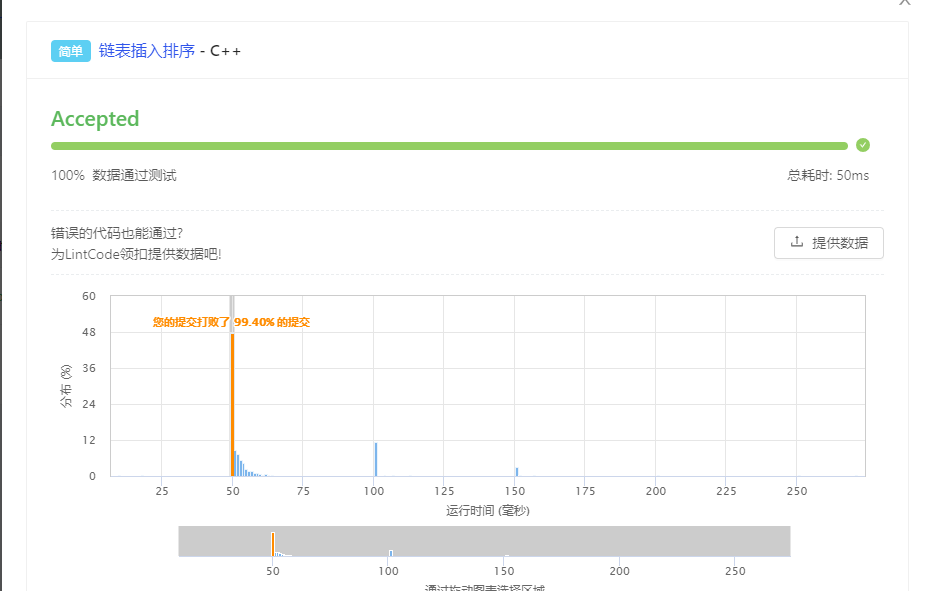
}

};

**2）思路：**

冒泡排序

**3）测试通过截图：**



## 174.删除链表中的倒数第n个节点

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: The first node of linked list.

\* @param n: An integer

\* @return: The head of linked list.

\*/

ListNode \* removeNthFromEnd(ListNode \* head, int n) {

// write your code here

ListNode \*p1 , \*p2, \*pre;

if ( head == NULL || n <= 0 )

{

return NULL;

}

p1 = head;

for ( int i = 0; i < n-1 ; i++ )

{

if ( p1 -> next != NULL )

{

p1 = p1 -> next;

}

else

{

return NULL;

}

}

p2 = head;

while ( p1 ->next != NULL )

{

p1 = p1 ->next;

pre = p2;

p2 = p2 ->next;

}

if(p2 == head)

{

head = head->next;

}

else

{

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

}

return head;

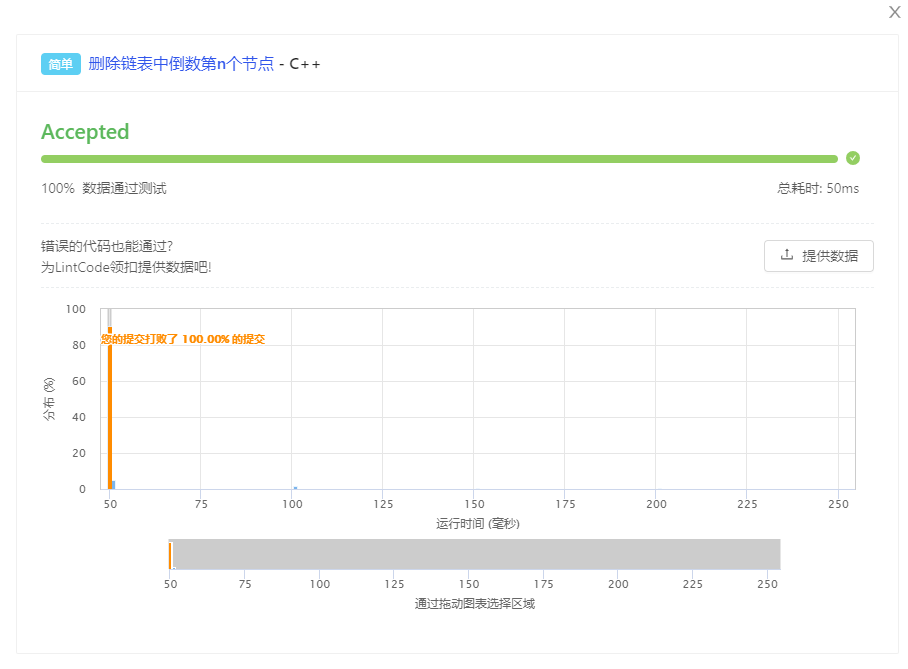
}

};

**2）思路：**

如果链表为空，或者输入的数字非正，返回空，第一个指针开始遍历，保证输入的n值小于链表长度，否则返回空，然后第二个指针开始遍历

**3）测试通过截图：**



## 451.两两交换链表中的节点

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: a ListNode

\* @return: a ListNode

\*/

ListNode \* swapPairs(ListNode \* head) {

// write your code here

ListNode \*p;

p=head;

if(head==NULL)

return NULL;

while(p!=NULL&&p->next!=NULL)

{

int x;

x=p->val;

p->val=p->next->val;

p->next->val=x;

p=p->next->next;

}

return head;

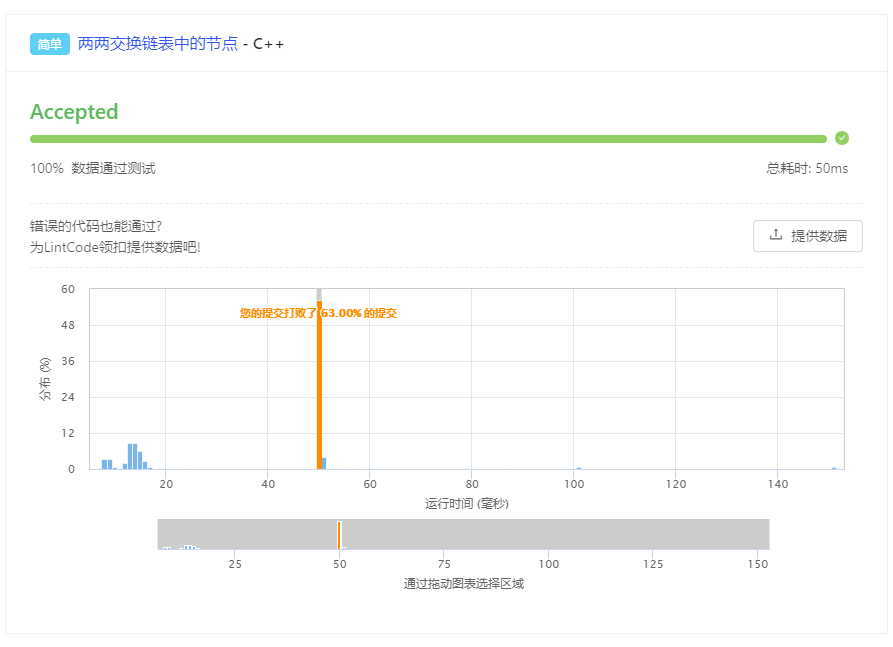
}

};

**2）思路：**

在进行循环是要一次移动两个指针，直接交换链表中节点的值。

**3）测试通过截图：**



## 452.删除链表中的元素

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: a ListNode

\* @param val: An integer

\* @return: a ListNode

\*/

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

// write your code here

if(head==NULL)

return NULL;

ListNode \*cur=head;

ListNode \*tmp=NULL;

while(cur->next)

{

if(cur->next->val==val)

{

tmp=cur->next;

cur->next=tmp->next;

free(tmp);

}

else

{

cur=cur->next;

}

}

if(head->val==val)

{

tmp=head->next;

free(head);

head=tmp;

}

return head;

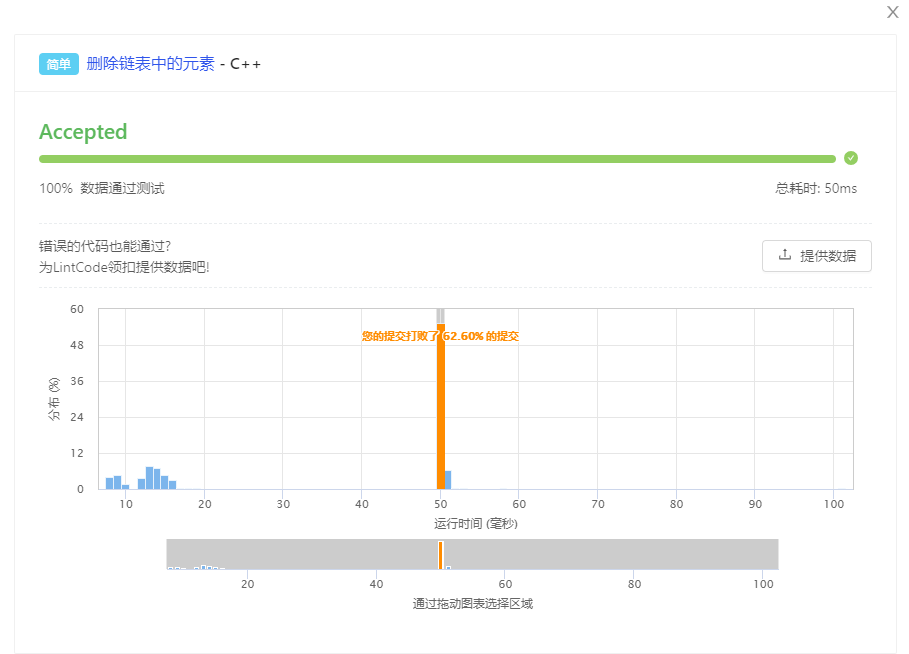
}

};

**2）思路：**

直接遍历，删除与val相等的节点

**3）测试通过截图：**



**756.两数相乘**

**1）代码：**

class Solution {

public:

/\*\*

\* @param l1: the first list

\* @param l2: the second list

\* @return: the product list of l1 and l2

\*/

long long multiplyLists(ListNode \* l1, ListNode \* l2) {

// write your code here

long long sum1=0;

long long sum2=0;

while(l1!=NULL)

{

sum1+=l1->val;

l1=l1->next;

if(l1!=NULL) sum1\*=10;

}

while(l2!=NULL)

{

sum2+=l2->val;

l2=l2->next;

if(l2!=NULL) sum2\*=10;

}

return sum1\*sum2;

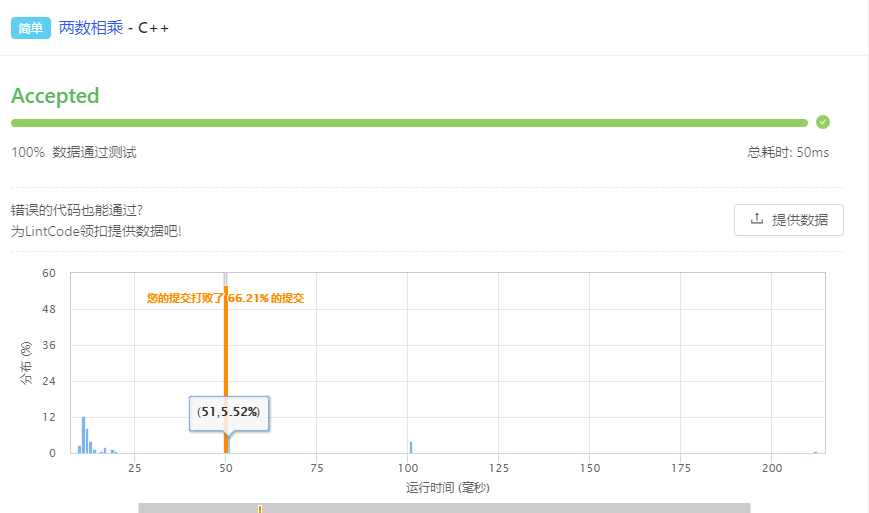
}

};

**2）思路：**

求出两个链表表示的数，然后直接相乘。

**3）测试通过截图：**



## 822.相反的顺序存储

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: the given linked list

\* @return: the array that store the values in reverse order

\*/

vector<int> reverseStore(ListNode \* head) {

// write your code here

vector<int> res;

stack<int> stk;

while(head != NULL)

{

stk.push(head->val);

head = head->next;

}

while(!stk.empty())

{

res.push\_back(stk.top());

stk.pop();

}

return res;

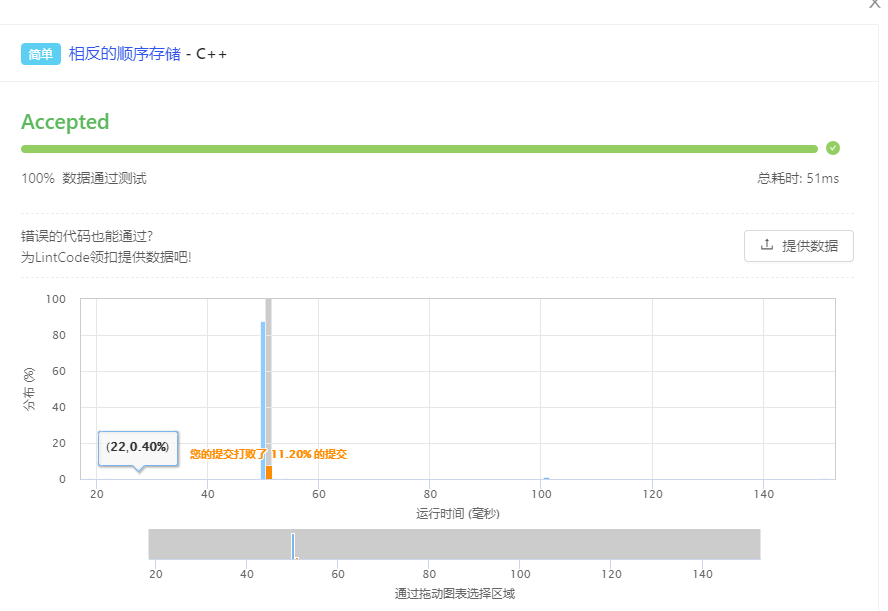
}

};

**2）思路：**

将list元素依次压入栈中，然后将栈中元素依次推出。

**3）测试通过截图：**



## 1609.链表的中间节点

**1）代码：**

class Solution {

public:

/\*\*

\* @param head: the head node

\* @return: the middle node

\*/

ListNode \* middleNode(ListNode \* head) {

// write your code here.

ListNode \*p=new ListNode(0);

p->next=head;

ListNode\*fast=p;

ListNode\*slow=p;

while(fast)

{

fast=fast->next;

if(fast) fast=fast->next;

slow=slow->next;

}

return slow;

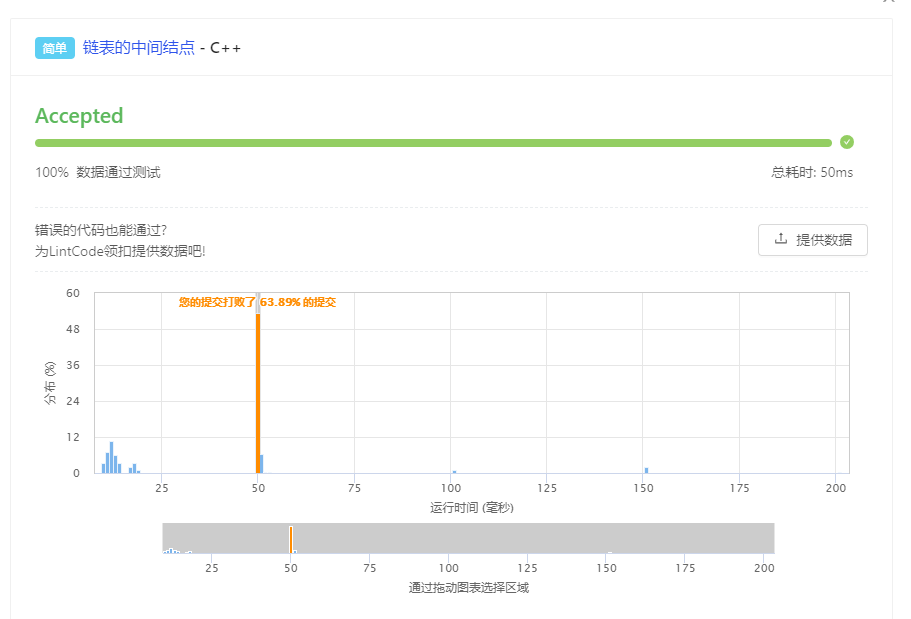
}

};

**2）思路：**

给定一个带有头结点head的非空单链表，返回链表的中间结点。如果有两个中间结点，则返回第二个中间结点。

**3）测试通过截图：**



# 数组

## 14.二分查找

**1）代码：**

class Solution {

public:

/\*\*

\* @param nums: The integer array.

\* @param target: Target to find.

\* @return: The first position of target. Position starts from 0.

\*/

int binarySearch(vector<int> &nums, int target) {

// write your code here

int low = 0;

int high = nums.size();

int mid;

int pos = -1;

while(low <= high)

{

mid = (low + high) / 2;

if (nums[mid] == target) {

pos = mid;

}

if (nums[mid] >= target) {

high = mid - 1;

} else {

low = mid + 1;

}

}

return pos;

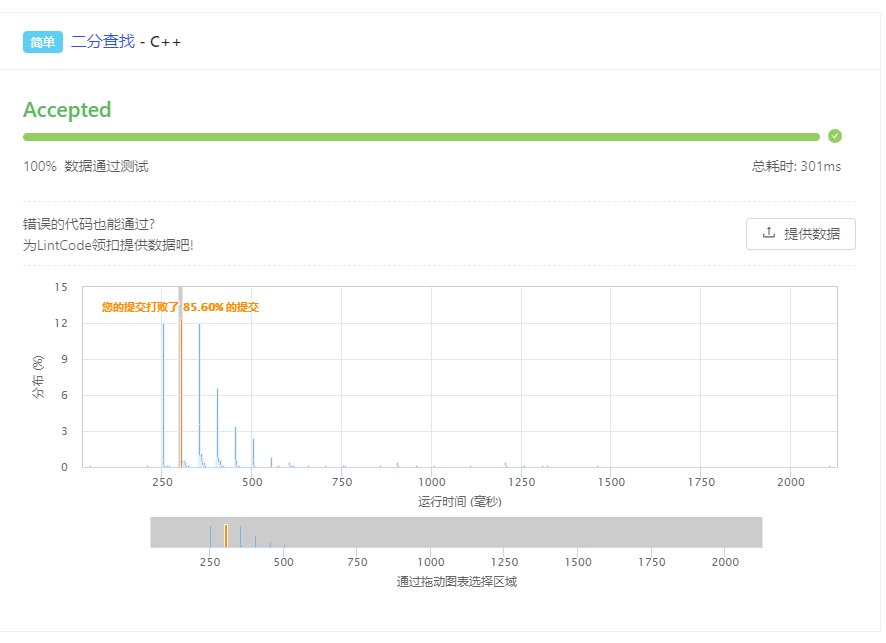
}

};

**2）思路：**

二分法

**3）测试通过截图：**



## 39.恢复旋转排序数组

**1）代码：**

class Solution {

public:

/\*\*

\* @param nums: An integer array

\* @return: nothing

\*/

void recoverRotatedSortedArray(vector<int> &nums) {

// write your code here

int min=nums[0];

for(int i=0;i<nums.size();i++){

if(min>nums[i]){

min=nums[i];

}

}

for(int i=0;nums[i]>min;){

int temp=nums[0];

for(int j=0;j<nums.size()-1;j++){

nums[j]=nums[j+1];

}

nums[nums.size()-1]=temp;

}

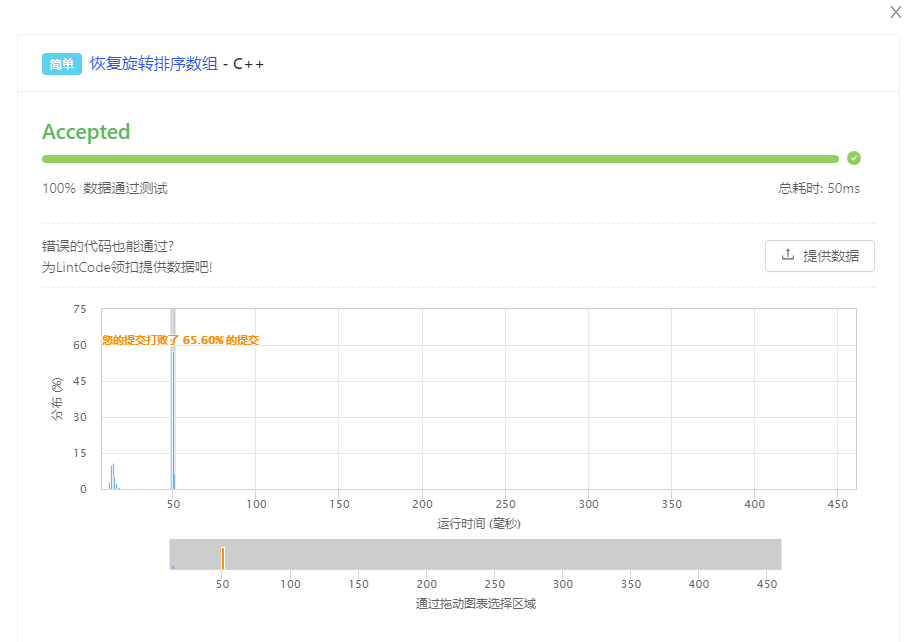
}

};

**2）思路：**

先遍历找到数组中最小的数，然后再将数组的第一个拿出来，将后面的元素依次往前排一位，再将第一个放到数组的后面，直到最小的一个排到第一个位置，

**3）测试通过截图：**



## 56.两数之和

**1）代码：**

class Solution {

public:

/\*\*

\* @param numbers: An array of Integer

\* @param target: target = numbers[index1] + numbers[index2]

\* @return: [index1 + 1, index2 + 1] (index1 < index2)

\*/

vector<int> twoSum(vector<int> &numbers, int target) {

// write your code here

vector<int> ans;

for(int i=0;i<numbers.size();i++){

for(int j=i+1;j<numbers.size();j++){

if(numbers[i]+numbers[j]==target){

ans.push\_back(i);

ans.push\_back(j);

return ans;

}

}

}

return ans;

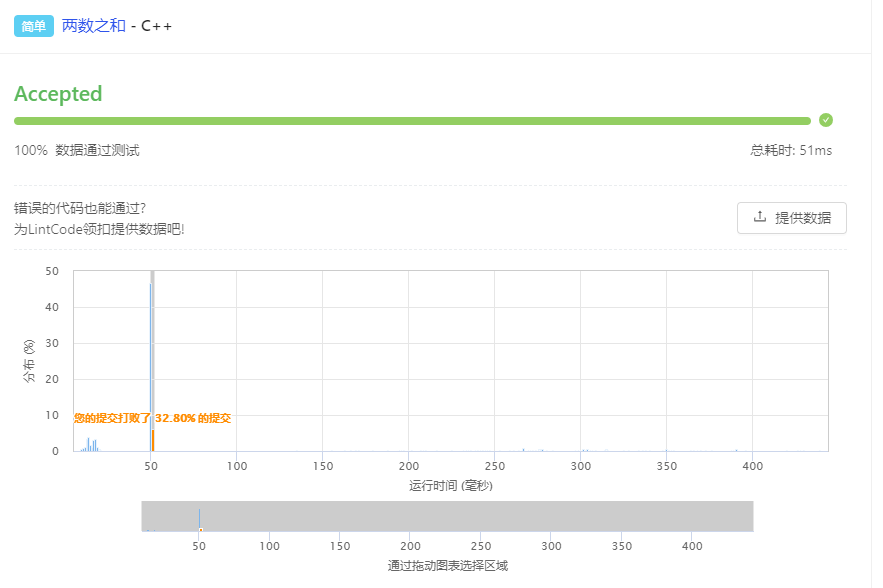
}

};

**2）思路：**

遍历每个元素X,并查找是否存在一个值与target-x相等的目标元素

**3）测试通过截图：**



## 60.搜索插入位置

**1）代码：**

class Solution {

public:

/\*\*

\* @param A: an integer sorted array

\* @param target: an integer to be inserted

\* @return: An integer

\*/

int searchInsert(vector<int> &A, int target) {

// write your code here

int i, j;

for(i = 0; i < A.size(); i++)

{

if(A[i] == target)

{

return i;

}

else if(A[i] > target)

{

break;

}

}

A.push\_back(target);

for(j = A.size(); j >= i; j--)

{

A[j] = A[j - 1];

if(j == i)

{

A[j] = target;

}

}

return i;

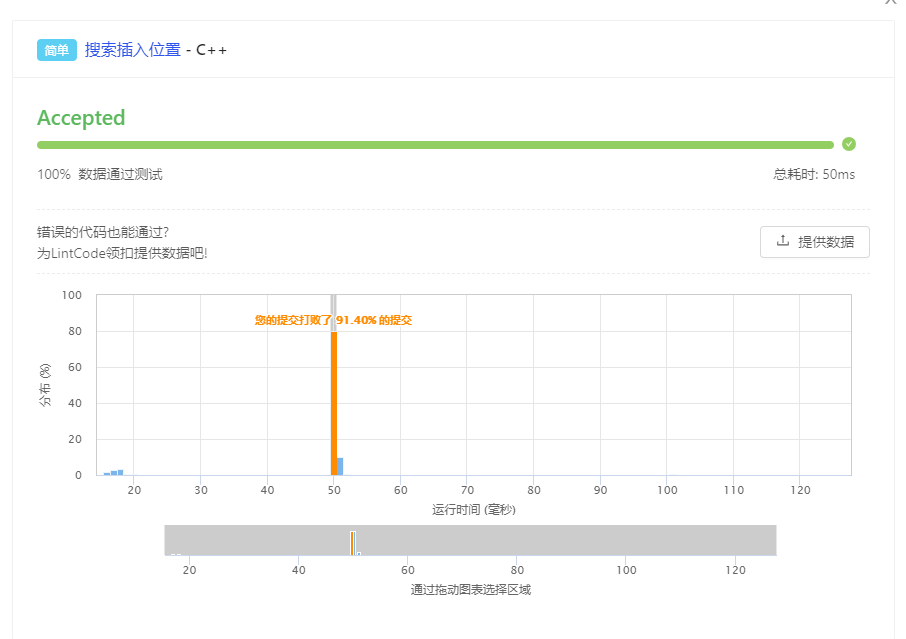
}

};

**2）思路：**

遍历每个元素

**3）测试通过截图：**



## 64.合并排序数组

**1）代码：**

class Solution {

public:

/\*

\* @param A: sorted integer array A which has m elements, but size of A is m+n

\* @param m: An integer

\* @param B: sorted integer array B which has n elements

\* @param n: An integer

\* @return: nothing

\*/

void mergeSortedArray(int A[], int m, int B[], int n) {

// write your code here

int sum=m;

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

if(m==0){

for(int j=0;j<n;j++){A[j]=B[j];}

}

for(int j=0;j<sum;j++){

if(B[i]<=A[j]){

for(int k=sum;k>j;k--){

A[k]=A[k-1];

}

A[j]=B[i];

sum++;

break;

}

if(B[i]>A[j]){

if(j==sum-1){

A[j+1]=B[i];

sum++;

break;

}

}

}

}

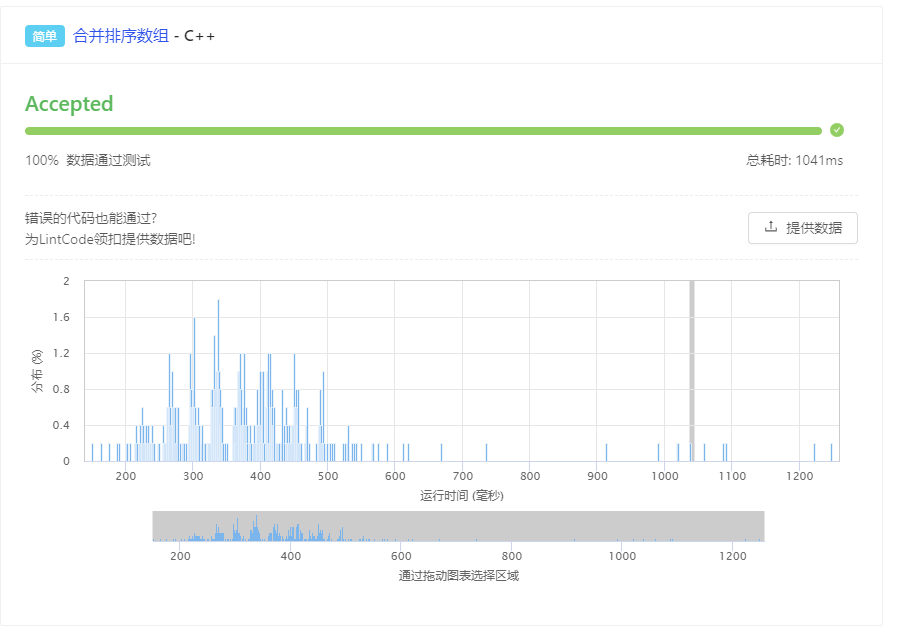
}

};

**2）思路：**

比较两个数组中的元素，把不相同的元素添加到A数组中

**3）测试通过截图：**



## 80.中位数

**1）代码：**

class Solution {

public:

/\*\*

\* @param nums: A list of integers

\* @return: An integer denotes the middle number of the array

\*/

int median(vector<int> &nums) {

// write your code here

sort(nums.begin(),nums.end());

int i=nums.size();

if(i%2==0) return nums[(i/2)-1];

if(i%2!=0) return nums[(i+1)/2-1];

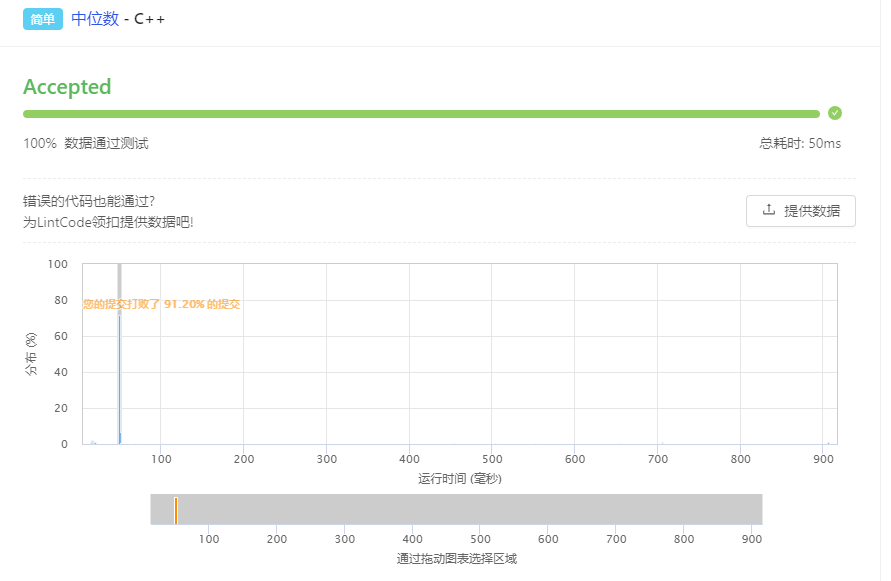
}

};

**2）思路：**

先进行排序，然后返回中间的值即可

**3）测试通过截图：**



## 100.删除排序数组中的重复数字

**1）代码：**

class Solution {

public:

/\*

\* @param nums: An ineger array

\* @return: An integer

\*/

int removeDuplicates(vector<int> &nums) {

// write your code here

int num = nums.size();

if(num == 0) return 0;

int i = 0;

for(int j = 1; j < num; j++)

{

if(nums[i] != nums[j])

{

i++;

nums[i] = nums[j];

}

}

nums.resize(i + 1);

return nums.size();

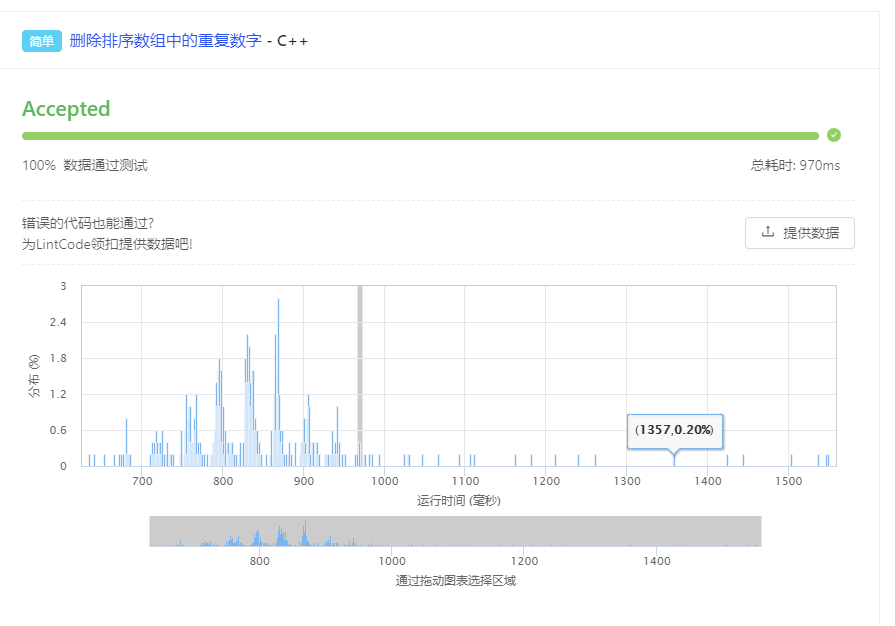
}

};

**2）思路：**

使用i和j遍历整个数组，I用来表示出现了重复数字的位置，j用来向后遍历，直到找不到重复的数字，再复制到i所在的位置的后面。

**3）测试通过截图：**



## 101.删除排序数组中的重复数字

**1）代码：**

class Solution {

public:

/\*\*

\* @param A: a list of integers

\* @return : return an integer

\*/

int removeDuplicates(vector<int> &nums) {

// write your code here

if(nums.empty())

{

return 0;

}

int n = nums.size(), k=0, times=1;

for(int i=1; i<n; ++i)

{

if(nums[i] != nums[k])

{

nums[++k] = nums[i];

times = 1;

}else if(nums[i] == nums[k]){

if(times >= 2)

{

continue;

}else{

nums[++k] = nums[i];

++times;

}

}

}

nums.resize(k+1);

return k+1;

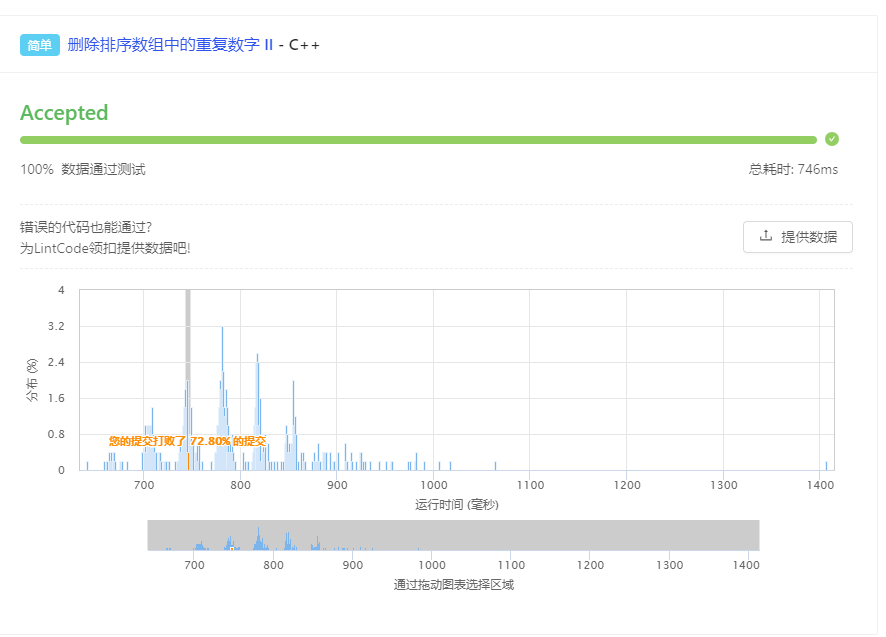
}

};

**2）思路：**

跟上题思路相同，只需要增加一个记录元素出现次数的变量即可，限制最多出现两次。

**3）测试通过截图：**



## 156，合并区间

**1）代码：**

class Solution {

public:

/\*\*

\* @param intervals: interval list.

\* @return: A new interval list.

\*/

static bool cmp(const Interval &s1,const Interval &s2)

{

return s1.start<s2.start;

}

vector<Interval> merge(vector<Interval> &intervals) {

vector<Interval> res;

if(intervals.empty())

return res;

sort(intervals.begin(),intervals.end(),cmp);

int i=0;

int sz=intervals.size();

while(i<sz-1)

{

if(intervals[i].start<=intervals[i+1].start&&intervals[i].end>=intervals[i+1].end)

{

intervals.erase(intervals.begin()+i+1);

sz--;

}

else if(intervals[i].start<=intervals[i+1].start&&intervals[i].end>=intervals[i+1].start)

{

intervals[i].end=intervals[i+1].end;

intervals.erase(intervals.begin()+i+1);

sz--;

}

else

i++;

}

return intervals;

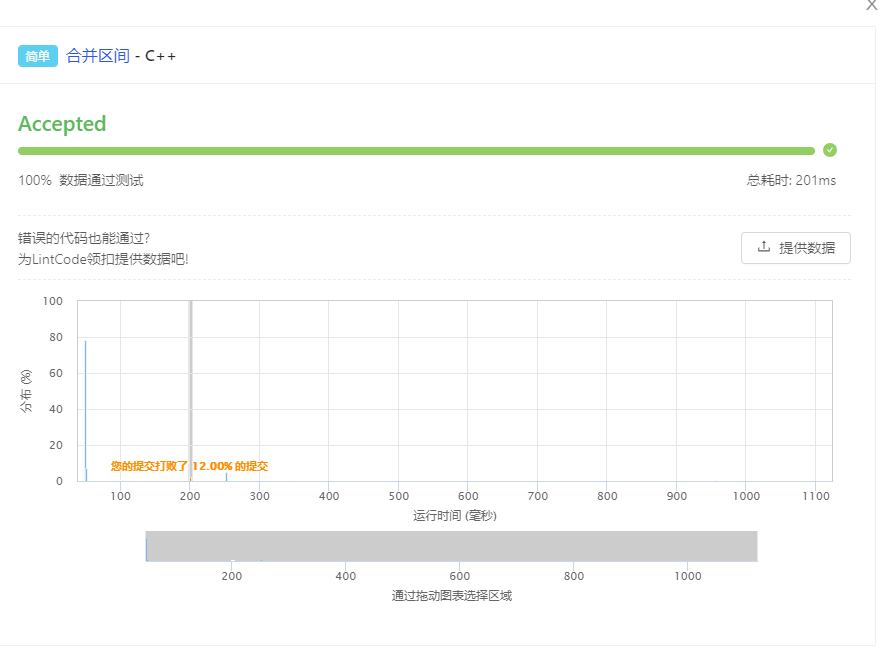
}

};

**2）思路：**

先排序，如果两者有交叉，就删除后面的节点，没有重叠就处理下一个区间

**3）测试通过截图：**



## 172，删除元素

**1）代码：**

class Solution {

public:

/\*

\* @param A: A list of integers

\* @param elem: An integer

\* @return: The new length after remove

\*/

int removeElement(vector<int> &A, int elem) {

// write your code here

int num=0;

for (int i = 0; i<A.size(); i++) {

if (A[i] != elem) {

A[num++] = A[i];

}

}

return num;

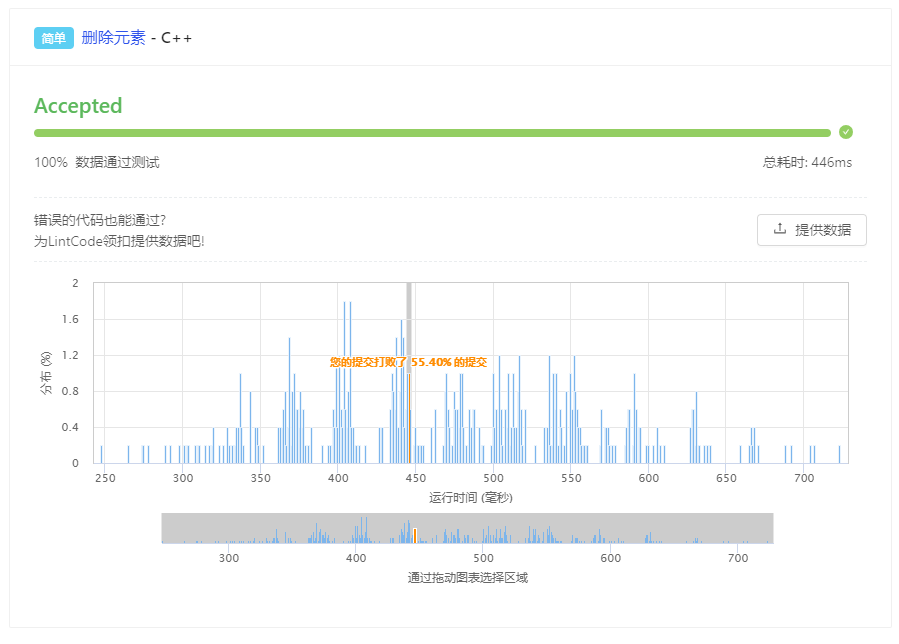
}

};

**2）思路：**

遍历

**3）测试通过截图：**



## 280.最近的城市

**1）代码：**

class Solution {

public:

/\*\*

\* @param x: an array of integers, the x coordinates of each city[i]

\* @param y: an array of integers, the y coordinates of each city[i]

\* @param c: an array of strings that represent the names of each city[i]

\* @param q: an array of strings that represent the names of query locations

\* @return: the closest city for each query

\*/

vector<string> NearestNeighbor(vector<int> &x, vector<int> &y, vector<string> &c, vector<string> &q) {

// write your code here

int cx, cy;

vector <string> near;

for(int queryIndex = 0; queryIndex < q.size(); queryIndex++)

{

string city = "NONE";

int minDistance = 0;

int findIndex = 0;

for(int i = 0; i < c.size(); i++)

{

if(c[i] == q[queryIndex])

{

cx = x[i];

cy = y[i];

findIndex = i;

break;

}

}

for(int i = 0; i < c.size(); i++)

{

if(i == findIndex)

{

continue;

}

if(x[i] == cx)

{

if(abs(cy - y[i]) < minDistance || minDistance == 0)

{

minDistance = abs(cy - y[i]);

city = c[i];

}

if(abs(cy - y[i]) == minDistance && c[i] < city)

{

minDistance = abs(cy - y[i]);

city = c[i];

}

}

if(y[i] == cy)

{

if(abs(cx - x[i]) < minDistance || minDistance == 0)

{

minDistance = abs(cx - x[i]);

city = c[i];

}

if(abs(cx - x[i]) == minDistance && c[i] < city)

{

midstance = abs(cx - x[i]);

city = c[i];

}

}

}

near.push\_back(city);

}

return near;

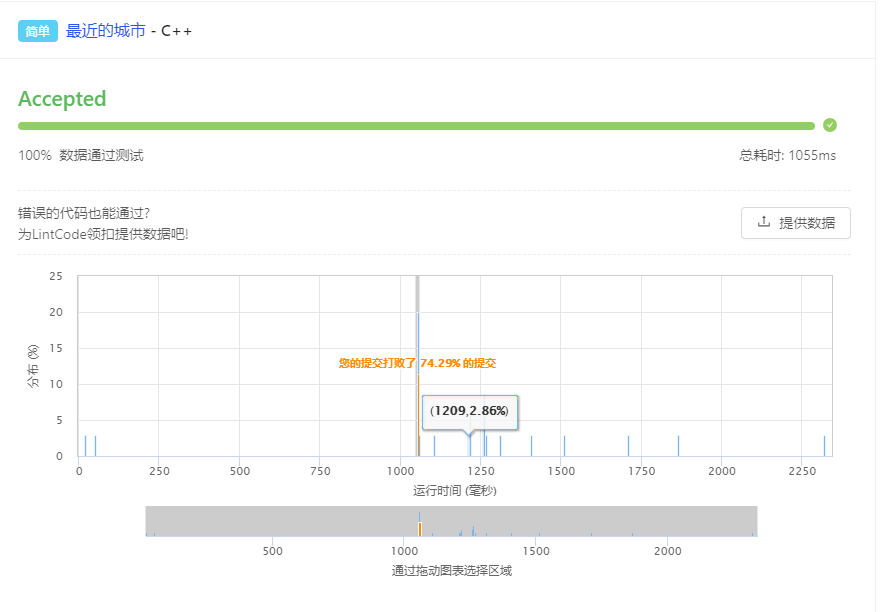
}

};

**2）思路：**

遍历比较

**3）测试通过截图：**



## 407.加一

**1）代码：**

class Solution {

public:

/\*\*

\* @param digits: a number represented as an array of digits

\* @return: the result

\*/

vector<int> plusOne(vector<int> &digits) {

// write your code here

vector<int> res;

int tem = 1, n = digits.size();

for (int i = n - 1; i >= 0; --i) {

int sum = digits[i] + tem;

res.insert(res.begin(), sum % 10);

tem = sum / 10;

}

if (tem == 1) res.insert(res.begin(), 1);

return res;

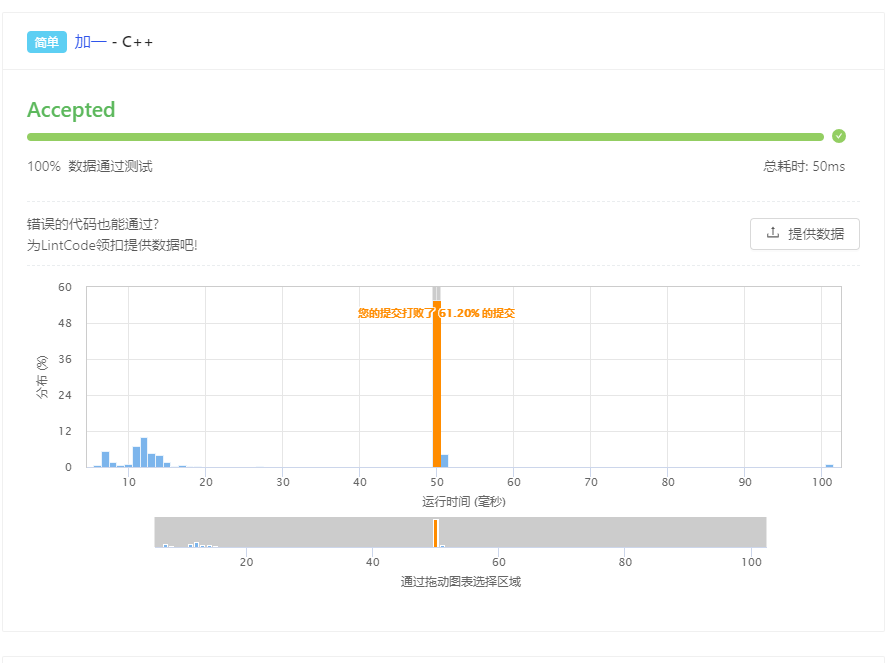
}

};

**2）思路：**

求出数组代表的数值，然后加一

**3）测试通过截图：**



## 1157.最短无序连续子数组

**1）代码：**

class Solution {

public:

/\*\*

\* @param nums: an array

\* @return: the shortest subarray's length

\*/

int findUnsortedSubarray(vector<int> &nums) {

// Write your code here

vector<int> tmp=nums;

sort(tmp.begin(),tmp.end());

int min=nums.size();

int max=0;

for(int i=0;i<nums.size();i++)

{

if(nums[i]!=tmp[i]) {min=i;break;}

}

for (int i = nums.size()-1; i >= 0; i--) {

if(nums[i]!=tmp[i]) {max=i;break;}

}

if(max==0) return 0;

else return (max-min+1);

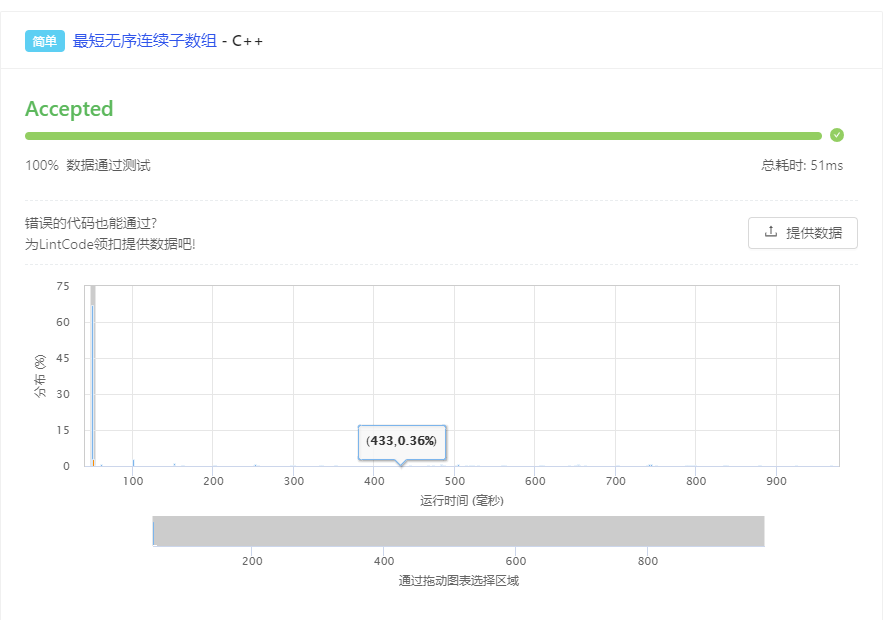
}

};

**2）思路：**

将整个数组排序，用排序后的序列与原序列的首尾进行一一对比，除去首尾一一对应相等的元素之后剩下的序列即为最短的无序子序列。

**3）测试通过截图：**



## 1250.第三大的数

**1）代码：**

class Solution {

public:

/\*\*

\* @param nums: the array

\* @return: the third maximum number in this array

\*/

int thirdMax(vector<int> &nums) {

// Write your code here.

if(nums.size()==0) return 0;

else if(nums.size()==1) return nums[0];

else if(nums.size()==2) return nums[0]>nums[1]?nums[0]:nums[1];

sort(nums.begin(),nums.end());

int cnt=1;

int num=nums[nums.size()-1];

for (int i = nums.size()-1; i >= 0; i--) {

if(nums[i]!=num) {cnt++;num=nums[i];}

if(cnt==3) return nums[i];

}

return nums[nums.size()-1];

}

};

**2）思路：**

先考虑0,1,2个元素的特殊情况，然后将数组排序，返回第三大的元素

**3）测试通过截图：**



## 1794.重复计数

**1）代码：**

class Solution {

public:

/\*\*

\* @param nums: a integer array

\* @return: return an integer denoting the number of non-unique(duplicate) values

\*/

vector<int> CountDuplicates(vector<int> &nums) {

// write your code here

unordered\_map<int, int> v;

vector<int> res;

for(int i = 0; i < nums.size(); ++ i) {

int count = v[nums[i]];

if(count == 1) {

res.push\_back(nums[i]);

}

++ v[nums[i]];

}

return res;

}

};

**2）思路：**

遍历

**3）测试通过截图：**

