最大连续子序列（数组里面的子数组含有正数和负数 求子数组和的最大值）

void maxarray()

{

std::vector<int> vec = {3,-2,5,4,0,-6,43,9};

std::vector<int> dp(vec.size());

int max = INT\_MIN;

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

{

If (i == 0) dp[i] = vec[0];

else if (dp[i-1] >= 0) dp[i] = dp[i-1] + vec[i];

else dp[i] = vec[i];

max = std::max(max,dp[i]);

}

std::cout<<max<<”\n”;

}

二分查找：

Int binarysearch(int arr[], int len, int key)

{

Int left = 0, right = len -1;

while (left <= right)

{

int mid = (left + right) /2;

if (arr[mid] == key){

return mid;

}

else if (arr[mid] > key)

{

right = mid – 1;

}

else{

left = mid + 1;

}

}

return -1;

}

在有序旋转数组中找到最小值(可以有重复值)：[1,2,3,4,5,6,7]旋转为[4,5,6,7,1,2,3] 利用二分法 leetcode

int getMin(vector<int>& nums){

int left=0 , right = vec.size()-1, mid;

while (left < right){

mid = (left + right) / 2;

if (nums[mid] > nums[right]) left = mid+1;

else if (nums[mid] < nums[right]) right = mid;

else right = right -1; //为了防止数组中重复的元素

}

return nums[left];

}

有序旋转数组中寻找target leetcode 33

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

int len = nums.size();

if (len == 0) return -1;

int left = 0,right = len -1, mid ;

while (left <= right){

mid = (left + right) / 2;

if (nums[mid] == target) return mid;

if (nums[mid] > nums[left]) //左边升序{

if (target >= nums[left] && target <nums[mid]) //target在左边范围内{

right = mid -1; //在左边找

}

else{

left = mid + 1; //在右边找

}

}

else{ //右边升序

if (target >nums[mid] && target <= nums[right]) //target在右边范围内{

left = mid + 1; //在右边找

}

else{

right = mid – 1; //在左边找

}

}

}

return -1;

}

int sqrt(int n) //二分法求平方根 (Leetcode 69)

{

int left = 0, right = n;

while(left<=right){  
 int mid = (left+right) / 2;

long ret = mid \* mid;

if (ret == n) return mid;

else if (ret > n) right = mid -1;

else if (ret < n) left = mid + 1;

}

return right;

}

二分法求最大值，最小值

int a[9] = {1,9,5,8,6,3,2,4,7};

void maxmin(int i, int j, int& fmax, int& fmin){

int lmin, lmax, rmin, rmax;

if (i == j){

fmax = a[i];

fmin = a[i];

}

else if (j-i == 1){

fmax = a[j] > a[i] ? a[j] : a[i];

fmin = a[j] < a[i] ? a[j] : a[i];

}

else{

int mid = (i+j) / 2;

maxmin(i,mid,lmax,lmin); //获得左半部的最大值，最小值

maxmin(mid,j,rmax,rmin);//获得右半部的最大值，最小值

fmax = lmax > rmax ? lmax: rmax;//左半部的最大值和右半部的最大值比较

fmin = lmin < rmin ? lmin : rmin;//左半部的最小值和右半部的最小值比较

}

}

有序数组中一个数字出现的次数

int GetNumberOfK(vector<int>& data, int k){

int index = binarysearch(data, 0,data.size()-1,k);

if (index == -1) return 0;

int indexleft = index, indexright = index;

while (indexleft > 0 && data[indexleft-1] == k){

indexleft = binarysearch(data,0,indexleft-1,k);

}

while (indexright <data.size()-1 && data[indexright+1] == k){

indexright = binarysearch(data,indexright+1,data.size()-1,k);

}

return indexright – indexleft;

}

求局部最小：nums长度是1时，nums[0]是局部最小。Nums长度为N>1时，如果nums[0]<nums[1],那么nums[0]是局部最小；如果nums[N-1]<nums[N-2],那么nums[N-1]局部最小。另外nums[i-1]>nums[i]<nums[i+1],nums[i]是局部最小。已知nums中任意两个相邻的数都不相等，返回任意一个局部最小的位置即可。

int getLessIdx(vector<int>& nums){

int len = nums.size();

if (len == 0) return -1;//不存在

if (len == 1) return 0;

if (nums[0] < nums[1]) return 0;

if (nums[len-1] < nums[len-2]) return len-1;

int left = 1, right = len-2, mid = 0;

while (left < right){

mid = (left + right) / 2;

if (nums[mid] > nums[mid -1]) right = mid-1; //局部最小肯定在左边

else if (nums[mid] > nums[mid+1]) left = mid+1; //局部最小肯定在右边

else return mid;

}

return left;

}

最长上升子序列：dp

for (int i = 0;i<n;i++) dp[i] = 1;

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

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

if (vec[i] > vec[j]) dp[i] = max(dp[i], dp[j]+1);

}

}

二分优化：

vector<int>dp(nums.size()); dp存放当前遇到的元素形成的上升子序列

input:[0,8,4,12,2]->dp[0] ->dp[0,8] ->dp[0,4]->dp[0,4,12]->dp[0,2,12]

dp[0] = nums[0];

int right = 0; //右边界位置

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

if (nums[i] > dp[right]){

dp[++right] = nums[i]; //扩展右边位置

}

else{

int j = lower\_bound(dp,dp+right,nums[i]) - dp;//二分寻找插入的位置

dp[j] = nums[i]; //更新这个位置的元素

}

}

return right+1;

归并排序:空间复杂度O(n) 稳定 插入排序：稳定

void merge(int a[], int start, int mid, int end)

{

int \*tmp = (int \*)malloc((end-start+1)\*sizeof(int)); // tmp是汇总2个有序区的临时区域

int i = start; // 第1个有序区的索引

int j = mid + 1; // 第2个有序区的索引

int k = 0; // 临时区域的索引

while(i <= mid && j <= end)

{

if (a[i] <= a[j])

tmp[k++] = a[i++];

else

tmp[k++] = a[j++];

}

while(i <= mid)

tmp[k++] = a[i++];

while(j <= end)

tmp[k++] = a[j++];

// 将排序后的元素，全部都整合到数组a中。

for (i = 0; i < k; i++)

a[start + i] = tmp[i];

free(tmp);

}

void merge\_sort\_up2down(int a[], int start, int end)

{

if (start < end){

int mid = (end + start)/2;

merge\_sort\_up2down(a, start, mid); // 递归排序a[start...mid]

merge\_sort\_up2down(a, mid+1, end); // 递归排序a[mid+1...end]

merge(a, start, mid, end);

}

}

链表排序(使用归并排序的方式O(nlogn),mid位置通过求快慢指针求出链表的中间节点)

ListNode\* getmid(ListNode\* node){ //利用前后指针求中间节点

ListNode\* fast,\*slow;

slow = fast = node;

while (fast->next != nullptr && fast->next->next != nullptr){

fast = fast->next->next;

slow = slow->next;

}

fast = slow->next;

slow->next = nullptr; //截断链表

return fast;

}

ListNode\* merge(ListNode\* head1, ListNode\* head2){

ListNode\* dummy = new ListNode(-1),\*curnode = dummy;

while (!head1 || !head2){

if (head1->val < head2->val){curnode->next = head1; head1 = head1->next;}

else {curnode->next = head2; head2 = head2->next;}

curnode = curnode->next;

}

if (head1) curnode->next = head1;

if (head2) curnode->next = head2;

return dummy->next;

}

ListNode\* sortlist(ListNode\* head){

if (!head || !head->next) return head;

ListNode\* mid = getmid(head);

head = sortlist(head); //链表前半部分的头节点

mid = sortlist(mid); //链表后半部分的头结点

return merge(head,mid);  
}

快速排序(单链表):双指针,从头至p指针的节点都是<=key;从p->q之间的节点值都是>key

ListNode\* partition(ListNode\* pbeg, ListNode\* pend)

{

ListNode\* p = pbeg, \*q = p->next;

int key = p->val;

while (q!= pend)

{

if (q->val < key)

{

p = p->next; //找到一个位置的元素值是大于key的,用这个元素和q的元素(小于key)交换

swap(p->val, q->val);

}

q = q->next;

}

swap(p->val , pbeg->val);

return p; //最终p的位置就是分割点的位置

}

void quicksort(ListNode\* pbeg, ListNode\* pend)

{

if (pbeg != pend)

{

ListNode\* mid = partition(pbeg, pend);

quicksort(pbeg,mid);

quicksort(mid->next, pend);

}

}

数组中的逆序数(剑指offer)：归并排序修改 数组中两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。并对总数1000000007取模

long long merge(vector<int>& data, int beg, int mid, int end){

int i, j;

long long cnt = 0;

i = beg, j = mid+ 1;

vector<int>tmp;

while (i <= mid && j <= end){

if (data[i] <= data[j]) tmp.push\_back(data[i++]);

else { //data[i] > data[j];

cnt += (mid-i+1);

tmp.push\_back(data[j++]);

}

}

while (i <= mid) tmp.push\_back(data[i++]);

while (j<=end) tmp.push\_back(data[j++]);

cnt %= 100000007;

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

data[beg + i] = tmp[i]; //每次统计逆序数后，修改原数组，确保原数组中的数字是递增排序的。

}

return cnt;

}

int doInverse(vector<int>& data, int beg, int end){

if (beg < end){

int mid = (beg + end) /2;

long long left = doInverse(data,beg,mid) % 1000000007;

long long right = doInverse(data,mid+1,end) % 1000000007;

return (left + right + merge(data,beg,mid,end)) % 1000000007;

}

}

int InversePairs(vector<int>data){

return doInverse(data,0,data.size()-1);

}

int Partition(int arr[], int start, int end)

{

int i = start, j = end;

int tmp = arr[i];

while (i < j){

while (i < j && arr[j] >= tmp) j --;

if (i < j) {arr[i++] = arr[j];}

while (i < j && arr[i] <= tmp) i++;

if (i < j) {arr[j--] = arr[i];}

}

arr[i] = tmp;

return i;

}

寻找无序数组的中位数

int getmidnum(int arr[], int left, int right)

{

int mid = (left+right) / 2;

int div = Partition(arr,left,right);

while (div != mid)

{

if (div > mid) div = Partition(arr,left,div-1);

else div = Partition(arr,div+1,right);

}

return arr[div];  
}

void quicksort(int arr[], int left, int right)

{

if (left == right) return ;

int idx = Partition(arr,left,right);

quicksort(arr,left,idx-1);

quicksort(arr,idx+1,right);  
}

寻找无序数组中第K小的数(即partition位置为K-1)：利用Partition O(n)

int kthelem(vector<int>& nums, int low, int high, int k)

{

if (low >= high) return 0;

int pos = Partition(nums,low, high);

while (pos != k-1)

{

if (pos > k-1)

{

pos = Partition(nums, low, high-1);

}

else

{

pos = Partition(num,low+1,high);

}

}

return nums[pos];

}

快速排序 不稳定

include<stdio.h>

void quickSort(int a[],int left,int right)

{

int i=left;

int j=right;

int temp=a[left];

if(left>=right)

return;

while(i<j)

{

while(i<j&&a[j]>=temp) j--;

if(j>i)

a[i++]=a[j];//a[i]已经赋值给temp,所以直接将a[j]赋值给a[i],赋值完之后a[j],有空位

while(i<j&&a[i]<=temp) i++;

if(i<j)

a[j--]=a[i];

}

a[i]=temp;//把基准插入,此时i与j已经相等R[low..pivotpos-1].keys≤R[pivotpos].key≤

quickSort(a,left,i-1);/\*递归左边\*/

quickSort(a,i+1,right);/\*递归右边\*/

}

冒泡排序：稳定的

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

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

if (vec[j] > vec[j+1]){

swap(vec[j], vec[j+1]);

}

}

}

选择排序：每次选择最小的数放在最前位置 不稳定

int k;

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

k = i;

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

if (vec[k] > vec[j]){

k = j;

}

}

if (k != i) swap(vec[k], vec[i]);

}

堆排序：大顶堆: arr[i] >= arr[2i+1] && arr[i] >= arr[2i+2]; 小顶堆:arr[i] <= arr[2i+1] && arr[i] <= arr[2i+2]

思想：先建初始堆；再通过交换堆顶元素和堆的最末元素，调整堆。不稳定O(nlngn)；建堆的时间复杂度O(n)

//构造最大堆

void MaxHeapFixDown(int a[], int i, int n){ //拿当前节点a[i]和它的两个子节点a[i\*2+1],a[i\*2+2]比较大小

int j = 2 \* i +1;

int tmp = a[i];

while (j < n){

if (j+1 <n && a[j] < a[j+1]) j++;

if (tmp > a[j]) break;

else{

a[i] = a[j]; //把大的元素提上来

i = j; //把i位置元素降下去

j = 2 \* i+1;

}

}

a[i] = tmp;

}

void HeapSort(int a[], int n){

for (int i = n/2-1;i>=0;i--) MaxHeapFixDown(a,i,n); //建堆

for (int i = n-1;i>=0;i--){ swap(a[i],a[0]); MaxHeapFixDown(a,0,i);} //删最大元素，然后重新建堆

}

注意插入排序：稳定； 希尔排序： 不稳定；

求两个有序数组的交集，并集

K个一组翻转链表：1->2->3->4->5 当k=2时，返回2->1->4->3->5 利用栈保存k个元素，利用栈逆置元素

ListNode\* reverseKGroup(ListNode\* head, int k){

int cnt = 0;

ListNode\* node(head);

//查看能否组成k个一组

While (node){

cnt++;

if (cnt == k) break;

node = node->next;

}

if (cnt < k) return head; //无法组成k个一组，直接返回head

//把k个一组翻转

ListNode\* prev(nullptr), \*cur(head),\*tmp;

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

tmp = cur->next;

cur->next = prev;

prev = cur;

cur = tmp;

}

head->next = reverseKGroup(cur,k);

return prev;

}

priority\_queue本质是一个堆。priority\_queue<type,container,functional>；container为保存数据的容器(必须是用数组实现的容器，如vector,deque但不能是list,stl中默认是vector)。Functional为元素比较方式(默认比较方式是operator<,大根堆，即std::less<int>。大顶堆:队列中元素从大到小排列，用来求最小的K个。小顶堆std::greater<int>:队列中元素从小到大排列，用来求最大的K个)。

一个m阶B树：

根节点至少两个子节点；每个中间节点都包含k-1个元素和k个孩子，其中m/2 <=k<=m；所有叶子节点都在同一层。

一个m阶B+树：mysql索引使用B+树作为存储结构

有k个子树的中间节点包含有k个元素(B树中是k-1个元素)，每个元素不保存数据，只用来索引(B树中中间节点也保存数据)，所有数据都保存在叶子节点。树的所有叶节点构成一个有序链表，可以按照关键码排序的次序遍历全部记录。

红黑树是一种二叉查找树，每个节点红或黑。通过对任意一条从根到叶子的路径上各个节点着色的方式的限制，红黑树确保没有一条路径会比其他路径长出两倍。因此，红黑树是一种弱平衡二叉树，相对于要求严格的AVL树来说，它的旋转次数少，所以对于搜索，插入，删除操作较多的情况下，通常用红黑树。

性质:

每个节点非红即黑；根节点是黑色的；每个叶节点都是黑色的；如果一个节点是红色的，则它的子节点必须是黑色的；对于任意节点而言，其到叶子结点的每条路径上包含相同数目的黑色节点。

红黑树相比BST(二分查找树)和AVL(平衡二叉树)有什么优点：

红黑树牺牲了严格的高度平衡的优越条件为代价，只要求部分达到平衡要求，降低了对旋转的要求，从而提高性能。任何不平衡都会在三次旋转内解决。相比BST,红黑树能确保最长路径不大于两倍的最短路径,最坏情况下保证O(logN);二叉查找树最坏O(N)

红黑树相比于哈希表，在选择使用时有什么依据？查找速度，数据量，内存使用，可扩展性

Hash查找速度比map快，O(1);map查找速度是log(n)。若考虑效率，特别是元素达到一定数量级，考虑hash。若对内存使用严格,希望减少消耗内存,那么一定要小心；

hashmap在C++11中就是unordered\_map: 首先分配 一大片内存，形成许多桶。利用hash函数,对key进行映射到不同区域(桶)进行保存。

插入过程是:得到key;通过hash函数得到hash值;得到桶号(一般都为hash值对桶数求模);存放key和value在桶内

取值过程是:得到key;通过hash函数得到hash值;得到桶号(一般为hash值对桶数求模); 比较桶的内部元素是否与key相等,若都不相等,则没有找到。取出相等的记录的value

hashmap中直接地址用hash函数生成;解决冲突,用比较函数解决;由此可见,要实现hashtable，和用户相关的是: hash函数和比较函数。这两个参数 是在 使用 hashmap时需要制定的参数；哈希函数返回的是size\_t;map的比较函数，需要提供less函数。如果没有提供,缺省的是less<Key>,hashmap中要比较桶内的数据和key是否相等,因此需要重载operator==

管理桶：

无序容器在存储上组织为一组桶(拉链法),每个桶保存零个或多个元素。无序容器使用一个哈希函数将元素映射到桶(哈希函数求值，再%bucket\_num)。rehash:当存入到hashtable中的元素数量+1大于bucket\_num时，会进行容器扩容，选下一个质数(使用质数可以最大程度减少冲突)开辟桶数量，把原来桶数组中元素全部重新哈希到新的桶数组中

运行效率方面:unordered\_map最高，map效率较低，但稳定效率，有序排列

占用内存方面:map占用内存略低,unordered\_map利用了空间换时间,哈希表元素越少，哈希冲突可能性越少,效率接近于O(1),但是这样一来空间利用率也越小。随着哈希表元素增多，桶要扩容，保持哈希表的装填因子在0.7~0.8。

unordered\_map与map对比：

unordered\_map需要hash函数(等于函数),map需要比较函数(小于函数)；hash\_map采用hash表存储,map采用红黑树实现

判断链表是否有环(leetcode 141)：快慢指针

bool hasCycle(ListNode\* head){

ListNode\* slow(head), \*fast(head);

if (!head) return false;

while (fast && fast->next){

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}

return false;

}

判断两个链表是否相交：

链表相交后，后面的部分节点全部共用，可以用2个指针分别从这两个链表头部走到尾部，最后判断尾部指针的地址信息是否一样，若一样则代表链表相交；

找到相交的起点(快慢指针)：分别获得两链表的长度，求差值。长的链表先走这个差值步长，然后各走一步，指针相等就是相交的起点。

使用set；

环的入口处(leetcode 142)：

指针操作：先求出环的长度(快慢指针相遇后，慢指针再走一圈与快指针相遇算出环的长度)；两个指针从链表头往后走，一个指针先走环的长度后，两个指针再各自每次走一个距离，最终相遇处为环的入口处。

最长公共子序列(LCS):

dp[i][j] = 0 if(i==0 || j ==0) dp[i][j] = dp[i-1][j-1] +1 if(xi == yj) dp[i][j] = max(dp[i][j-1], dp[i-1][j]) if (xi != yj)

最长回文字串(leetcode 5):记录一下最长子串的起始位置

dp[i][j]:表示第i到第j字符间的长度(i <= j)

初始状态：dp[i][j] = 1 (i=0 : n-1. i==j)，dp[i][j]=0(i!=j)

状态转移方程：因为要计算dp[i][j],需要先计算dp[i+1][j-1],所以i从后往前推,j从前往后推 for (i = len-2;i>=0;i--){(for j = i+1;j<len;j++)}if (str[i] == str[j]) && (j-i == 1) dp[i][j] = 2;

if (str[i] == str[j] && j-i>1) && dp[i+1][j-1] != 0 那么dp[i][j] = dp[i+1][j-1] + 2; 若此时dp[i+1][j-1] = 0，那么dp[i][j] = 0; 若str[i] != str[j],那么dp[i][j] = 0;

字符串最短编辑距离：dp[i][j]表示str1[0..i]编辑成str2[0..j]的最小代价,编辑包括：插入一个字符，删除一个字符，替换一个字符。

dp[i][0] = i dp[0][j] = j;

dp[i][j] = dp[i-1][j-1] if (word1[i-1] == word2[j-1]) dp[i][j] = min(dp[i][j-1],dp[i-1][j],dp[i-1][j-1])+1

if (word1[i-1] != word2[j-1])

最长公共子串：

dp[i][j] = 0 if (i==0||j==0) dp[i][j] = dp[i-1][j-1] + 1 if (str1[i-1] == str2[j-1]) dp[i][j] = 0 if (str1[i-1] != str2[j-1])

0-1背包：i: 前i件物品 j:背包容量 dp[i][j]：表示前i件物品恰放入容量为j的背包可以获得的最大价值 dp[n+1][m+1] for (int i = 0;i<=n;i++) dp[i][0] = 0;

for (int i = 1;i<=m;i++) dp[0][i] = 0;(前0件物品)

dp[i][j] = dp[i-1][j] if (j<w[i-1]) dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i-1]] + v[i-1]); if (j>=w[i-1])

leetcode 416分割等和子集

完全背包(拼凑钱币 美团)：dp[i][j]: 表示前i种物品，拼凑j大小的面额，每件物品可以取无穷多次。初始化dp全为0且dp[i][0] = 1,

状态转移:dp[i][j] = max{dp[i-1][j-k\*data[i-1]] + k\*w[i-1]} , 0<=k\*data[i-1]<=j k=0,1,2….j/data[i-1]

多重背包：类似于完全背包，只是K的取值有限制，因为每件物品的数量有限制

状态转移：dp[i][j] = max{dp[i-1][j-k\*data[i-1]] + k\*w[i-1]} 0<=k<=n[i-1]

最近公共祖先节点(二叉树的后序遍历)： 由最近想到后序遍历

TreeNode\* ancestor(TreeNode\* T, TreeNode\* t1, TreeNode\* t2)

{

if (!T || T==t1 || T==t2) return T;

TreeNode\* left = ancestor(T->left, t1, t2);

TreeNode\* right = ancestor(T->right, t1, t2);

if (left && right) return T; //左子树中发现t1或t2，并且右子树中发现t2或t1;那么此时最小公共祖先就是T

return left ? left : right;

}

判断一个数是否2的倍数:若是2的倍数，一定是100000的形式，那么n-1=0111111

return n & (n-1) == 0 ? 1:0 ;

图的遍历：dfs

有N个节点的图的数据结构:vector<int>graph[N];通过graph[i].push\_back(j)来构造图，OJ时省内存

void dfs(vector<vector<int>>& graph,vector<bool>&visited, int pos,int sum, int n)

{

std::cout<<pos<<” ”;

sum++;

if (sum == n) return; //所有顶点都已经访问，退出

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

{

if (graph[pos][i] != 0 && !visited[i]) {

visited[i] = true;

dfs(graph,visited, I, sum, n);

//visited[i] = false;

}

}

}

最短路径:dfs min就是最短路径

void dfs(vector<vector<int>>&graph, vector<bool>&visited, int pos, int& min, int curlen, int target)

{

if (curlen > min) return;

if (pos == target -1){

if (curlen < min) min = curlen;

return;

}

for (int I = 0; i<target; i++)

{

if (graph[pos][i] != 0 && !visited[i])

{

visited[i] = true;

dfs(graph,visited, i, min,curlen+graph[pos][i], target);

visited[i] = false;

}

}  
}

最短路径算法floyd:

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

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

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

graph[i][j] = std::min(graph[i][k] + graph[k][j] , graph[i][j]);

}

}

}

STL求最大最小值

vector<int> vec= {2,3,-3,4,1,56,234,21};

auto it = std::max\_element(vec.begin(),vec.end()); //std::min\_element(beg, end);

cout<<"max: "<<\*it<<"\n";

求二叉树的叶子节点个数：

void leafcnt (TreeNode\* root){

if (root == nullptr) return 0;

int left = leafcnt(root->left);

int right = leafcnt(root->right);

if (!root->left && !root->right) return 1 + left + right;

else return left + right

}

求二叉树的高度:

void treedepth(TreeNode\* root){

if (root == nullptr) return 0;

int left = treedepth(root->left);

int right = treedepth(root->right);

return max(left,right) + 1;

}

判断平衡二叉树：使用到了求二叉树的高度

bool balancetree(TreeNode\* root){

if (root == nullptr) return true;

int left = treedepth(root->left);

int right = treedepth(root->right);

return (abs(left-right) <2 && balancetree(root->left) && balancetree(root->right));

}

或者后序遍历，遍历一遍即可

int helper(TreeNode\* root){

if (root == nullptr) return 0;

int left = helper(root->left); //返回左子树的高度；返回-1，说明已不是平衡二叉树

if (left == -1) return -1;

int right = helper(root->right);

if (right == -1) return -1;

return abs(left-right) >1? -1 : max(left,right)+1;

}

bool balancetree(TreeNode\* root){

return helper(root) != -1;

}

判断两个树是否相等:

bool issametree(TreeNode\* p, TreeNode\* q){

if (p == nullptr && q == nullptr) return true;

else if (p == nullptr || q == nullptr) return false;

return (p->val == q->val && issametree(p->left, q->left) && issametree(p->right, q->right));

}

void mirror(TreeNode\* root){//翻转二叉树LeetCode226

if (!root) return ;

TreeNode\* tmp = root->left;

root->left = mirror(root->right);

root->right = mirror(tmp);

}

判断 二叉树是否镜像(对称Leetcode 101)：

bool helper(TreeNode\* r1, TreeNode\* r2){

if (r1 && !r2) return false;

if (!r1 && r1) return false;

if (!r1 && !r2) return true;

return r1->val == r2->val && helper(r1->left,r2->right) && helper(r1->right,r2->left);

}

bool Issymmetry(TreeNode\* root){

if (!root) return true;

return helper(root->left, root->right);

}

给定三角形ABC和一点P(x,y,z),判断点P是否在ABC内，手写代码

根据面积法，如果P在三角形ABC内，那么S abp + S bcp + S acp = S abc； 根据三个点的坐标求出三角形面积(两个向量叉乘的模除以2)

struct Point{

float x;

float y;

};

给三个坐标点(x1,y1)(x2,y2)(x3,y3)，求三角形面积公式(海伦公式)s = sqrt(p\*(p-a)\*(p-b)\*(p-c)) a,b,c,为三角形三条边长 a = sqrt((x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2)) b = sqrt((x2-x3)\*(x2-x3) + (y2-y3)\*(y2-y3)) c = sqrt((x1-x3)\*(x1-x3) + (y1-y3)\*(y1-y3)) p = (a+b+c) / 2

给出4个顶点坐标，判断4个点是否能组成正方形？

使用unordered\_map <int(dist),int(次数)> 依次保存每两个不同点的距离，距离为0返回错误，距离只能是两种(正方形边长，正方形对角线边长)

求一个整数中1的位数：

int res = 0;

while (n != 0){

n = n & (n-1); //实质是抹掉最右边的1

res ++;

}

二叉树的序列化与反序列化

序列化:非空节点(数字#,),空节点(#,)

string Serialize(TreeNode\* root){

string str;

if (!root){

str.append("#,");

}

else{

str.append(to\_string(root->val)+"#,");

str.append(Serialize(root->left));

str.append(Serialize(root->right));

}

return str;

}

反序列化:把逗号前的字符串都split存入vector

TreeNode\* doDeserialize(vector<string>& vec,int& pos){

pos ++;

TreeNode\* root(nullptr);

if (vec[pos] != "#"){ //创建二叉树

root = new TreeNode(stol(vec[pos]));

root->left = doDeserialize(vec, pos);

root->right = doDeserialize(vec, pos);

}

return root;

}

TreeNode\* Deserialize(string data){

int pos = 0;

vector<string>vec;

string str, sub;

stringstream ss(data);

while (getline(ss,sub,’,’)){

vec.push\_back(sub);

}

pos = -1;

TreeNode\* root = doDeserialize(vec, pos);

return root;

}

合并两个有序链表(递归)：递归思路类似于创建二叉树

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

if (!l1) return l2;

if (!l2) return l1;

ListNode\* head = nullptr;

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

head = l1;

head->next = merge(l1->next,l2);

}

else{

head = l2;

head->next = merge(l1,l2->next);

}

return head;

}

复制带随机指针的链表(leetcode 138)：给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。

先复制next指针和value,创建N’节点(new新的节点，val同以前节点val一致);使用map<N,N’>保存对应关系,N’为复制的节点。(N和N,的val一致)再复制random指针。Clone->random = map[pHead->random];最终返回新创建节点的指针

给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中没有重复出现的数字：1->2->3->3->4->4->5 处理后为 1->2->5

采用递归

ListNode\* deleteDuplicates(ListNode\* head){

if (!head || !head->next) return head;

if (head->val == head->next->val){

while (!head->next && head->val == head->next->val){

head = head->next;

}

return deleteDuplicates(head->next);

}

head->next = deleteDuplicates(head->next);

return head;

}

Two Sum:

vector<int> twosum(vector<int>& nums, int target){//题设每种输入只有一个答案：因此可以确定nums中没有重复值

unordered\_map<int,int>hashmap; //<value,index> //通过hashmap查找，让两次循环变成一次循环

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

vector<int>res;

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

int v = target – nums[i];

if (hashmap.count(v) && hashmap[v] != i){

res.push\_back(i);

res.push\_back(hashmap[v]);

break;

}

}

return res;

}

三数之和: a + b + c =0 先排序然后双指针(leetcode 15 网易互娱，小红书等)

std::vector<std::vector<int>>res;

std::vector<int>tmp(3);

std::sort(nums.begin(),nums.end(),std::less<int>());

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

j = i+1;

k = len-1;

if (i>0 && nums[i] == nums[i-1]) continue; //防止重复

while (j<k){

if (nums[i] + nums[j] + nums[k] == 0){

tmp[0] = nums[i];

tmp[1] = nums[j];

tmp[2] = nums[k];

res.push\_back(tmp);

j++;k--;

while (nums[j] == nums[j-1]) j++; //防止重复

while (nums[k] == nums[k+1]) k--; //防止重复

}

else if (nums[i] + nums[j] + nums[k] > 0){

k--;

}

else if (nums[i] + nums[j] + nums[k] < 0){

j++;

}

}

}

return res;

}

树的子结构：判断二叉树b2是否是b1的子结构

bool Issame(BTNode\* b1, BTNode\* b2){

if (b2 == nullptr) return true;

if (b1 == nullptr) return false;

return b1->data == b2->data && Issame(b1->left,b2->left) && Issame(b1->right,b2->right);

}

bool IsSub(BTNode\* b1, BTNode\* b2){

bool bsame= false;

If (b1->data == b2->data) {

//search sub structure

bsame = Issame(b1,b2);

}

if (!bsame) bsame = IsSub(b1->left,b2);

if (!bsame) bsame = IsSub(b1->right,b2);

return bsame;

}

二叉树层次遍历：从上往下，逐层打印节点

vector<int> levelvisit(BTNode\* bt){

queue<BTNode\*>qu;

if (bt != nullptr) qu.push(bt);

vector<int>res, level;

int curlevelcnt = 1, nextlevelcnt = 0, visited = 0;

while (!qu.empty()){

if (curlevelcnt == visited){

curlevelcnt = nextlevelcnt;

nextlevelcnt = visited = 0;

for (auto& it : level) {cout<<it->val<<” ”;}

level.clear();

}

BTNode\* node = qu.front();

res.push\_back(node->data);

if (node->left) {qu.push(node->left); nextlevelcnt++;}

if (node->right) {qu.push(node->right);nextlevelcnt++;}

qu.pop();

level.push\_back(node); //为打印做准备

visited ++;

}

for (auto& it : level){cout<<it->val<<” ”;}

return res;

}

加密算法(单向加密，对称加密，非对称加密):

单向加密：又称为不可逆加密(可以由明文到密文，不能由密文到明文)，其密钥是由加密散列函数生成的。单向散列函数一般用于产生消息摘要，密钥加密等。

MD5(message digest algorithm 5): 单向散列算法，非可逆，相同的明文产生相同的密文

SHA(secure hash algorithm): 可以对任意长度的数据运算生成一个160位的数值。提供校验功能

算法特征: 输入一样，输出必然相同；定长输出，无论原始数据多大，结果大小都是相同的；不可逆，无法根据特征码还原原来的数据。

对称加密：

密钥(secret key)：一种参数，它是在明文转换为密文或将密文转换为明文的算法中输入的参数。密钥分为对称密钥和非对称密钥。

采用密钥系统的加密方法。同一个密钥可以同时用作信息的加密和解密。这种加密方法称为对称加密。

特点: 加密方和解密方使用同一个密钥；加密解密的速度比较快，适合数据比较长时使用；密钥传输的过程不安全，且容易被破解，密钥管理也比较麻烦。

优点:对称加密算法的优点时算法公开，计算量小，加密速度快，加密效率高。

缺点:对称加密算法的缺点是在数据传送前，发送方和接收方必须商定号密钥，然后使双方都能保存好密钥。其次如果一方的密钥被泄露，那么加密信息也就不安全了。另外每对用户每次使用对称加密算法时，都需要使用其他人不知道的唯一密钥，会使得收发双方密钥数量巨大，密钥管理称为负担。

非对称加密：

非对称密钥加密也称为公钥加密，由一对公钥和私钥组成。公钥是从私钥提取出来的。可以用公钥加密，再用私钥解密；当然也可以用私钥加密，用公钥解密。因此非对称加密的主要功能就是加密和数字签名。

特征: 密钥对：公钥和私钥；主要功能：加密和签名

常用非对称加密算法:RSA(既可以实现加密，又可以实现签名)

LRU缓存：最近最少使用；核心思想是：如果数据最近被访问过，那么将来被访问的几率也更高。get(int key) put(int key, int value)

实现：用一个std::list保存key, map<int,int> 保存放入list的(key,value)

class LRUCache{

public:

int \_capacity;

std::list<int> \_list;

map<int,int> \_m;

LRUCache(int capacity){

\_capacity = capacity;

}

int get(int key){

auto it = \_m.find(key);

if (it == \_m.end()){

return -1;

}

else{

\_list.remove(key);//更新优先级

\_list.push\_front(key);

return it->second;

}

}

void put(int key, int value){

int cap = \_m.size();

if (cap >= \_capacity){

auto it = \_m.find(key);

if (it == \_m.end()){

\_m.erase(\_list.back()); //删除最近最少使用的那个元素

\_list.pop\_back();

}

else{

\_list.remove(key);

}

\_m[key] = value;

\_list.push\_front(key);

}

else{

\_m[key] = value;

\_list.remove(key);

\_list.push\_front(key); //给予这个元素最高优先级

}

}

}

大数 n! 输入不超过1000的正整数n，输出n!=1\*2\*3\*……\*n的精确结果。  
样例输入：30

样例输出：265252859812191058636308480000000

int arr[5000], n =30;

memset(arr,0,sizeif(arr));

arr[0] = 1;

for (int i=2;i<=30;i++){

int jinwei = 0, s;

for (int j = 0; j<5000;j++){

s = arr[j] \* i + jinwei;

arr[j] = s % 10;

jinwei = s / 10;

}

}

int k;

for (k = 4999;k>=0;k--){

if (arr[k] != 0) break;

}

for (int i = k;i>=0;i--) cout<<arr[i];

全排列：

void helper(vector<vector<int>>& res, vector<int>& nums, vector<int>& tmp,

vector<bool>& visited){

if (tmp.size() == nums.size()){

res.push\_back(tmp);

return;

}

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

if (visited[i] == false){

visited[i] = true;

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

helper(res,nums,tmp,visited);

visited[i] = false;

tmp.pop\_back();

}

}

}

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

vector<vector<int>>res;

int len = nums.size();

vector<int>tmp;

vector<bool>visited(len);

helper(res,nums,tmp,visited);

return res;

}

利用stl解决全排列：next\_permutation(begin,end)//函数中每个元素必须完全不同(unique)

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

vector<vector<int>>res;

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

do{

for (auto it : nums){

cout<<it<<” ”;

}

cout<<”\n”;

}while(next\_permutation(nums.begin(),nume.end()));

}

并查集：

void Init(vector<int>& vec){

for (int i = 1;i<=vec.size();i++) vec[i] = i;

}

int dofind(vector<int>& vec, int v){

if (vec[v] == v) return v;

vec[v] = dofind(vec,vec[v]);

return vec[v];

}

void dounion(vector<int>& vec,int u, int v){

int fatheru = dofind(vec,u);

int fatherv = dofind(vec,v);

if (fatheru != fatherv){

vec[fatheru] = fatherv;

}

}

当要获得某个元素i的组号时: vec[i] = dofind(vec,i); paypal寻找关联用户

字典树(TrieTree):除根节点外，每个节点都包含一个字符；从根到某个节点，所有经过的字符组成的字符串即该节点对应的字符串；每个节点最多含有26个子节点。树的高度仅仅由最长单词的长度决定，不由文件篇幅决定。

优势是：利用字符串的公共前缀来节约存储空间，最大限度地减少无谓的字符串比较。

struct Node{

bool isWord; //判断当前字母是否为单词的最后一个字母标识

Node\* next[26];//26个分叉：对应26个字母

Node(){

isWord = false;

for (int i = 0;i < 26;i++){

next[i] = nullptr;

}

}

};

class DictTree{

public:

Node\* root; //字典树的根节点

DictTree(){

root = nullptr;

}

void Insert(string str){

if (root == nullptr) root = new Node;

Node\* head = root;

for (int I = 0;i< str.length();i++){

int num = str[i] – ‘a’;

if (head->next[num] == nullptr){

head->next[num] = new Node; //新增加一个有着26个分支的节点

}

head = head->next[num];

}

head->isWord = true;

}

bool Search (string str){

Node\* head = root;

for (int I = 0;i<str.length();i++){

int num = str[i] – ‘a’;

if (head->next[num] == nullptr){

return false;

}

else{

head = head->next[num];

}

}

return head->isWord;

}

};

rand\_shuffle(first,last)

洗牌算法：基本思想就是每次随机选出一个没有被选过的数放到队列中，如果随机出来的数已经被选过，那么继续随机直到遇到一个没有被选过的数放入到队列中。重复如此直到所有数被选择出来。

优化方法：当一个数被选之后没必要在下一次随机的时候考虑它。因此，每次只从可选的数的集合中进行随机，也就不用考虑是否会碰到已经选过的数了

在原始数组上对数字进行交互 (swap随机数和最末的那个数)，省去了额外的O(n)的空间。基本思想和之前类似，每次从未处理的数据中随机取出一个数字，然后把该数字放在数组的尾部，把原来尾部没有选出的数字和选出的数字交换。

void shuffle(int n){

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

swap(nums[i],nums[rand()%(i+1)]); //把选出来的放到尾部，同时把尾部没有选出来的和它交换，放到前面，可以作为下次备选对象

}

}

stl中关于二分查找的函数:lower\_bound, upper\_bound, binary\_search应用于有序空间(非递减序列。二分查找的前提)

binary\_search(arr[],arr[]+size,val) 在数组中以二分检索的方式查找，若在数组(要求数组元素非递减)中查找到val元素则返回真，否则返回假。

lower\_bound:查找第一个大于或等于某个元素的位置 lower\_bound(arr[], arr[]+size, val) 在first和last中的前闭后开区间进行二分查找，返回大于或等于val的第一个元素位置(注意是地址)。如果所有元素都小于val，则返回last的位置，且last的位置是越界的。

upper\_bound:查找第一个大于某个元素的位置 upper\_bound(arr[], arr[]+size, val),在前闭后开区间查找关键字的上界，返回大于val的第一个元素位置

已知rand7()可以产生1~7的7个数(均匀概率),利用rand7()产生rand10() 1~10(均匀概率)

rand7() = {1,2,3,4,5,6,7} rand7()-1 = {0,1,2,3,4,5,6} ; (rand7()-1) \* 7 = {0,7,14,21,28,35,42},那么

(rand()-1) \* 7 + rand7()-1 表示 0~49之间均匀分布的任何一个数，每个数的概率都是1/49；那么剔除41~49后剩下1~40也是均匀分布

int rand10(){

int x = 0;

do{

x = (rand7() -1) \* 7 + rand7()-1;

}while (x>40); //保留0~39的数

return x % 10 + 1;

}

给定rand5()随机生成1到5的函数，写出能随机生成整数1到7的函数 字节跳动

Int rand7(){

int x;

do{

x = (rand5()-1) \* 5 + rand5()-1;

}while (x>21);

return x % 7 + 1;

}

约瑟夫环:

int cycle(int n, int m){ //共n个人，数到m出列;从0开始计数

int cnt = n, step, idx;

vector<bool>visited(n);

idx = -1; step = 0;

while(cnt > 0){

idx ++;

if (idx == n) idx = 0;

if (visited[idx] == true) continue; //已经出队了

step ++;

if (step == m){

step = 0;

visited[idx] = true;

cnt --;

}

}

return idx; //返回最后出队的那个人的idx

}

1个n位的数，去掉其中的k位，使得剩下的n-k位数按顺序组成的数最小

从前往后遍历，每次访问一位，比较该位前边的数，如果比该位大，去掉。直到去掉K位或者到了最后。

保留2位小数

#include <iomanip>

cout<<setiosflags(ios::fixed)<<setprecision(2)<<sum<<”\n”;

bitset<32> bitvec(1U) //32位；低位为1，剩余位为0 bitvec.all();//如果所有位为1，返回true bitvec.count()返回其中1的个数 bitvec.any()只要有一个为1就返回true

提取bvitset的值：bitset.to\_ulong()

随机数库: #include <random>

default\_random\_engine e(time(0)); uniform\_int\_distribution<unsigned>u(0,9) cout<<u(e);

素数：2是素数

for (int i = 2;i<=sqrt(n);i++){

if (n % i == 0){

cout<<n<<” isn’t a prime”<<”\n”;

return false;

}

}

cout<<”is a prime\n”;

return true;

读入数字字符串，每个数字以逗号分: 简单C方法：scanf(“%d,%d,%d”,&x,&y,&z);

或者C++stringstream切割字符串方式

vector<string>vec;

string str, sub;

stringstream ss(data);

while (getline(ss,sub,’,’)){ //getline(ss,sub,’ ’)以空格切割

vec.push\_back(sub);

}

手写hashmap;

struct Node{ //拉链法

Int key, value;

Node\* next;

Node(int k, int v):key(k),value(v),next(nullptr){}

};

class hashmap

{

private: vector<Node\*> vec;

public:

hashmap(){ vec = vector<Node\*>(1e5,new Node(-1,-1));}

void put(int k, inv v){} // int idx = k % 1e5;

int get(int k){} // int idx = k % 1e5;

void remove(int k){}

};

海量数据如何取最大的K个？10亿个数中找出最大的10000个数(海量数据求top K问题)

先拿10000个数建小根堆，然后逐次遍历之后10亿-10000个元素，和堆顶(最小)数字进行比较，如果比堆顶数字大，则替换堆顶元素并重新调整最小堆。建堆时间O(mlogm)，算法时间复杂度O(nmlogm) (n:10亿,m：10000)

优化方法(思想：将大数据平分到多个小文件中，小文件中的数据去重(采用Hash(x) % 小文件个数)，对每个小文件中的数据建最小堆，选出最大的10000个)：先用hash法把10亿个数字去重，若重复率高，会减少很大的内存用量。把去重后的数据分组存放，分别放在1000个文件中，将每个文件数据建最小堆，在每个文件中找出最大的10000个数，合并到一起再找出最终结果。

布隆过滤器：

布隆过滤器时一种数据结构，特点是高效地插入和查询，可以用来告诉你某样东西一定不存在或者可能存在。相比于传统地list,set,map等数据结构，它更高效，占用空间更少，但缺点是其返回地结果是概率性地，而不是确切地。

Hashmap的问题：存储容量占比高，数据量一大，hashmap占的内存大小就很大了。

布隆过滤器数据结构：是一个bit向量或者说是bit数组。如果要映射一个值到布隆过滤器中，需要使用多个不同的哈希函数生成多个哈希值，并将每个生成的哈希值指向的bit位置成1。如果想查询某个值是否存在，对该值用哈希函数返回的位置，判断该位置上是否置为1，如果没有一个为1，可以确定的说这个值不存在。如果都为1，只能说可能存在。

使用场景：利用布隆过滤器减少磁盘IO或者网络请求。因为一旦一个值不存在的话，就没必要进行后续昂贵的查询请求。

只用2GB内存在20亿个整数中找到出现次数最多的数：有一个包含20亿个全是32位整数的大文件，在其中找到出现次数最多的数。

解决方法是把包含20亿个数的大文件用哈希函数分成16个小文件。根据哈希函数的性质，同一种数不可能被哈希到不同的小文件上，同时每个小文件中不同的数一定不会大于2亿中。对每一个小文件用哈希表统计其中每种数出现的次数，这样就得到了16个小文件中各自出现次数最多的数。接下来只要选出这16个小文件各自第一名出现的次数最多即可。

把一个大的集合通过哈希函数分配到多台机器中，或者分配到多个文件中，这种技巧是处理大数据面试题时最常用的技巧之一。

找到100亿个URL中重复的URL以及搜索词汇的topK问题:有一个包含100亿个URL的大文件，假设每个URL占用64B，找出所有重复的URL。

常规方法：把大文件通过哈希函数分配到多台机器上，或者通过哈希函数把大文件拆分成小文件。一直进行这种划分，直到划分的结果满足资源限制的要求。通过将每条URL通过哈希函数分配到若干机器或者拆分成若干小文件。然后每台机器分别统计分给自己的URL中是否有重复的URL同时哈希函数的性质决定了相同的URL不可能分给不同的机器(相同的URL一定在同一台机器上或者在同一个小文件中)。

找出每天最热门top100词汇的可行办法。topK问题：哈希函数分流，哈希表统计，小根堆做topK计算

首先通过哈希函数将大文件中的数据映射到一个个小文件中或者若干台机器中。当处理每个小文件时，用哈希表统计每种词及其词频，哈希表建立完成后，再遍历哈希表。通过使用大小为100的小根堆选出每一个小文件的top100。每个小文件都有自己词频的小根堆。对每个小文件的小根堆继续利用小根堆排序，得到最终的top100。

40亿个非负整数中找到没出现的数。

32位无符号整数的范围是0~4294967295,现在有一个正好包含40亿个无符号整数的文件

若用哈希表保存出现过的数，那么若40亿个数都不相同，则哈希表的记录数为40亿条，存一个32位整数需要4B，所以最差情况下需要40亿\*4B=160亿字节，大约要16GB的空间，不符合要求(最多可使用1GB的内存)。

哈希表占用很多空间，可以使用bitmap的方式来表示数出现的情况。申请一个424967295的bit类型的数组bitArr,bitArr上的每个位置只可以表示0或者1状态。8个bit为1B，所以长度为9294967295的bit类型的数组占用500MB空间。

遍历这40亿个无符号数，比如遇到7000，就把bitArr[7000]设置为1。遇到所有的数时，就把bitArr相应位置的值设置为1。

再次重新遍历bitArr，哪个位置上的值没被设置为1，哪个数就不在40亿个数中。

进阶问题：现在只有10MB的内存，要求找到其中一个没出现过的数。

将0~4294967295这个范围平均分为64个区间每个区间67108864个数。例如，第0个区间(0~67108863),….第63个区间(4227858432~4294967295)

第一次遍历时，先申请长度为64的整形数组countArr[0~63],countArr[i]用来统计区间i上的数有多少。遍历40亿个数，根据当前数是多少来决定哪一个区间上的计数增加。例如当前数是3422552090，3422552090/6108864=51，所以在第51区间上的计数增加，countArr[51]++。遍历完40亿个数之后，遍历countArr,必然会有某一个位置上的值(countArr[i])小于67108864，表示第i个区间上至少有一个数没有出现过。

一致性哈希算法的基本原理：

字节跳动

最大回文字串 leetocde 5

判断镜像二叉树

二叉树的最大路径和 leetcode 124 hard

下一个排列 leetcode 31 数组

中序非递归

数组不相邻元素之和的最大值(leetcode 打家劫舍)

加油站(leetcode 134)

打家劫舍 leetcode 198 [1,2,3,1]输出为4；相邻的数字不能拿dp[i] = max(dp[i-1],dp[i-2]+nums[i])

收到礼物最大值 (剑指offer，走格子，往下走，往右走)

找出数组中出现次数超过n/k的数,n为数组长度，k为给定的数 (剑指offer：找出一个数的出现次数大于数组中元素的一半) leetcode 求众数II

二叉树的左视图树(打印从左方向看到的二叉树，二叉树层次遍历：打印每一层的第一个节点。同理，二叉树的右视图)

最小堆

实现LRU

实现memcpy void\* memcpy(void\* dst, const void\* src, size\_t len) 内存区域重叠问题

memmove针对memcpy的内存覆盖问题，做了改进。

void\* my\_memcpy(void\* dst, const void\* src, size\_t n)

{

assert (dst == nullptr && src == nullptr && n <0) return nullptr;

char\* pdst= (char\*) dst;

char\* psrc = (char\*) src;

if (pdst > psrc && (psrc+ n)>pdst) //内存覆盖:pdst在psrc和psrc+n之间

{

psrc = psrc + n-1;

pdst = pdst + n -1;

while (n--) \*pdst-- = \*psrc--; //从后往前拷贝

}

else{

while(n--) \*pdst++ = \*psrc++; //一般情况:从前往后拷贝

}

return dst;

}

实现strcpy，内存覆盖

char\* strcpy(char\* dst, const char\* src)

{

assert(!dst && !src );

char\* pdst = dst;

while (\*src != ‘\0’) \*pdst++ = \*src++; //char\* my\_memcpy(char\* dst, const char\* src,int cnt)

\*pdst = ‘\0’; //因此strcpy后时，不用另外写str[len] = ‘\0’

return dst;

}

返回char\*,可以支持链式表达式。

股票买卖leetcode121(只能买卖一次)，122(可以买卖无限次)

Leetcode 862 双端队列

Leetcode 41缺失的第一个正数(重复出现)

字符串连续乱序字串匹配，例如"abcde" 中找 "cba" “dcb“之类的

leetcode 329矩阵中的最长递增路径dfs bfs 动态规划

旋转有序数组中找最小值(剑指offer leetcode)

给出中序和后序重建二叉树(剑指offer leetcode 106)

Leetcode200岛屿个数(微软)

Leetcode 695 岛屿的最大面积

Leetcode694不同岛屿的个数(给定基准坐标，将与基准坐标的差值存入set中,返回set.size())

Leetcode300最长上升子序列(动态规划二分)

Leetcode 48旋转图像(数组swap,reverse)

无序数组的中位数(partition)

Leetcode 287寻找重复数(快慢指针，相当于求环的入口)

层次遍历，奇数层逆置

单链表快排

最近公共祖先

Leetcode 863二叉树上距离目标节点为K的节点(保存每个节点对应的parent节点，左子树，右子树，父节点三个方向dfs遍历找节点)

leetcode 295数据流中的中位数(剑指offer 堆)：最大堆存放输入数据流中较小的一半，最小堆存放输入数据流中较大的一半，两个堆的top就是中位数

合并N个有序数组，每个数组长度为M(小根堆)

二叉树的最长路径(就是求二叉树的深度)

8G的URL文件，只有1G内存，找出重复的URL(大数据问题)

翻转字符串(剑指offer)

洗牌算法(给你一个1到n的序列，让你随机打乱，保证每个数出现在任意一个位置的概率相同，也就是说在n!个排列中，每一个排列出现的概率相同)

正则表达式匹配(leetcode 10)

bool isMatch(string s, string p){ // ‘.’匹配任意一个字符。’\*’匹配它之前字符的0次或多次

if (p.empty()) return s.empty();

bool first\_match = (!s.empty() && (s[0] == p[0] || p[0] == ‘.’));

if (p.length() >= 2 && p[1] == ‘\*’){

//ignore ‘\*’ 或者 使s往后移一个位置匹配

return isMatch(s,p.substr(2)) || (first\_match && isMatch(s.substr(1),p));

}

else{

return first\_match && isMatch(s.substr(1), p.substr(1));

}

}

判断回文链表(leetcode 234):slow,fast指针，将链表后半部分进栈，然后和链表前半部分做比较

两个线程交替打印1-100(condition\_variable unique\_lock<std::mutex>(lck))

LFU缓存(leetcode 460 社招)

寻找两个有序数组的中位数(leetcode 4hard)(开辟一个大数组有序存放两个小的有序数组，从大数组中找中位数O((m+n)/2))

由rand5()实现rand7()

K个一组翻转链表(Leetcode 25)利用递归k个一组先翻转，再head->next=reverseKGroup(…);

八皇后()

移掉K位数字(leetcode 402给定一个字符串表示的非负整数，移除这个数中的K位数字，使得剩下的数字最小)(最大)(贪心：从前往后找，每次访问一位，和该位之前的数字比较，如果之前的位比当前这位大，去掉之前的数位)反之使剩下的数最大(小红书，猿辅导)

string deleteKdigits(string num, int k){

bool flag ;

string::iterator it;

for (int i = k; i>0;i--){

flag = true;

for (it = num.begin(); it < num.end()-1; it++){

if (\*(it) > \*(it+1)) num.erase(it);

flag = false;

break;

}

if (flag){ //递增有序时，删除剩下可以删的

num.erase(num.end()-I, num.end());

break;

}

}

it = num.begin();

while (\*(it) == ‘0’) num.erase(it); //删除前置0

if (num.empty()) return “0”;

return num;

}

最长公共前缀(leetcode 14 循环依次比较string.substr())

接雨水(leetcode 42下雨后水能达到的最高位置，等于两边最大高度的较小值(相当于从当前位置分别向左，右两边去找最高的数值)减去当前高度的值，采用动态规划那么可以用两个数组把每个位置左右两侧最高的数值记录下来)

最大正方形(leetcode 221 在一个由0和1组成的二维矩阵内，找到只包含1的最大正方形，并返回其面积 百度 华为 网易互娱 小米)动态规划dp[i][j]：表示在(i,j)位置上最大正方形的边长，dp[i][j] = min(min(dp[i-1][j-1],dp[i-1][j]),dp[i][j-1]) + 1 求出最大的dp[i][j] \* dp[i][j]

实现哈希表(leetcode 706)

struct Node{

int nkey;

int nval;

Node\* next;

Node(int k, int v):nkey(k),nval(v),next(nullptr){};

};

vector<Node\*> \_vec(1000,Node(-1,-1)); //创建hashtable，1000个桶

void put(int k, int v){

int tmp = k % 1000;

Node\* p = \_vec[tmp]; //找到对应的桶

Node\* pre;

while (p){ //遍历这个桶

if (p->key == k) {p->val = v; return ;}

pre = p;

p = p->next;

}

Node\* node = new Node(k,v);

pre->next = node;

}

void get(int k){

Node\* p = \_vec[k % 1000];

while (p){

if (p->key == k) return p->val;

p = p->next;

}

return -1;

}

void remove(int k){

Node\* p = \_vec[k % 1000];

while (p){

if (p->key == k) {p->val = -1; return ;}

p = p->next;

}

}

拼多多：

寻找峰值(leetcode 162 寻找一个数比左边，右边大。二分查找)

两个链表相加(leetcode 2)

百度：

二叉树的高度(深度)

两个集合的交集，并集

单链表的反转

数组右移K位 (左移K位 k%len,右移K位(len-k) % len)

一个集合中只有两个元素出现了一次，其他都出现了两次，找出这两个元素。Leetcode 260

int sum = 0, a = 0 , b = 0;

for (int i = 0 ;i<len;i++) sum ^= nums[i]; // a ^ b = sum;

int flag = sum & (~(sum-1)); //a= 5 b = 3 sum = 110 sum-1 = 101 ~(sum-1) = 010 so flag = 010,使用flag 把数组元素分成两份

for (int I = 0;i<len;i++)

{

if (nums[i] & flag == 0) a ^= nums[i];

else b ^= nums[i];

}

//a,b就是出现了一次的元素

旋转数组找最小值

10亿个IP，取出频次出现最多的TOP10个IP

很大的数组中找第K大元素

手写shared\_ptr

一个数组，求和为M的数据对，要求返回所有结果

判断一个字符串是否是IP地址

数组长度为N，数据是1~N，判断是否有重复数据(时间O(n))剑指offer

海量数据，统计出字符串出现的个数(把海量数据hash到小文件中，小文件可以装入内存)

二叉搜索树转双向链表(剑指offer,中序遍历)

10亿个URL，每个URL大小小于56B，要求去重，内存4G(大数据问题)

无序数组求中位数

深信服

手写atoi(const char\* str)

统计一个文本出现最多的10个字符

腾讯：

两个有序链表的合并

有序链表删除重复结点

二叉树求根节点到叶子节点的路径和为sum的路径 leetcode 112

bool hasPathSum(TreeNode\* root, int sum){

helper(root, sum);

return res == true;

}

bool res = false;

void helper(TreeNode\* root, int sum){

if (!root) return;

if (!root->left && !root->right && sum-root->val==0){

res = true;

return ;

}

else{

helper(root->left,sum-root->val);

helper(root->right, sum-root->val);

}

}

或者

void helper(TreeNode\* root, int sum){

if (!root) return;

sum -= root->val;

if (!root->left && !root->right && sum==0){

res = true;

return ;

}

else{

helper(root->left,sum);

helper(root->right, sum);

}

sum += root->val;

}

求二叉排序树第三大结点(中序)

环形缓冲区 leetcode 622

int front,rear, size,capaciry

vector<int>vec; vec.resize(capacity);

enqueue: vec[rear++] = elem; rear = rear%capacity; size++;

outqueue: head = (head+1) % capacity; size--;

20个银币个金币

滴滴快车

二叉查找树的插入元素

如何保证线程安全

LRU实现

网易互娱：

实现shared\_ptr

环形队列 leetcode 622

猿辅导

链表第m到第n个反转(每隔K个leetcode)

一个n位数，现在可以删除其中任意k位，使得剩下的数最小(从前往后找，每次访问一位，和该位之前的数字比较，如果之前的位比当前这位大，去掉之前的数位)反之使剩下的数最大

二叉树中序遍历的后继节点(中序遍历改进)

360

快速排序

华为：

pow(double m,int n)剑指offer leetcode 50(注意当n 是负数)

Leetcode Top 100

1两数之和(hashtable)

2两数相加

3无重复字符的最长字串(哈希)

4寻找两个有序数组的中位数

5最长回文字串(dp)

6 Z字形变换(微软)

10正则表达式匹配(递归)

11盛最多水的容器(前后指针，逐渐向中心移动)

15三数之和(排序+前后指针)

17电话号码的字母组合(回溯)

19删除链表的倒数第N个节点

20有效的括号(栈)

21合并两个有序链表(递归)

22生成括号(左括号数目=右括号数目)

23合并K个排序链表(将多路排序归并为合并两个有序链表，分治思想 腾讯)

29两数相除(微软 给定两个整数，相除，要求不能用乘法，除法，mod。整数除法结果截去其小数部分)思路1：将除法转化为减法，循环相减。2：二分法迅速逼近结果

31下一个排列(从后往前找小的数，swap+排序)

32最长有效括号:找出包含最长有效括号的字串的长度(stack保存左括号的下标，记录有效括号的起始位置)

33搜索旋转排序数组

34在排序数组中查找元素的第一个和最后一个位置(二分)

39组合总数(经典回溯dfs)

42接雨水(前后指针)

44通配符匹配(微软 字符串动态规划)

46全排列(典型回溯，visited,与之前的回溯略有区别)

48旋转图像

49字母异位词分组

53最大子序和(动态规划)

55跳跃游戏(从后往前的dp)

56合并区间

57插入区间(微软 类似于合并区间)

62不同路径(动态规划：起始点在左上角，终点在右下角，每次只能向下或者向右，共有多少种方法，63不同路径II(矩阵中有障碍物)，174地下城游戏(反过来推))

64最小路径和(mXn的网格，每个网格上有对应的数字，找出一条从左上角到右下角的路径，使得路径上的数字总和最小。每次只能向下或向右)

70爬楼梯(爬n阶楼梯，可以爬1阶或2阶)

71简化路径(微软 栈 stringstream)

72编辑距离(dp)

75颜色分类(给定一个包含红，白，蓝，一共n个元素的数组，原地对他们进行排序，使得相同颜色的元素相邻，并按照红，白，蓝顺序排列，桶排序，计数排序)、

76最小覆盖字串(双指针)

78子集(经典回溯)

79单词搜索(dfs+visited)

84柱状图中最大的矩形(单调栈)

85最大矩形(给定一个仅包含0和1的二维矩阵，找出只包含1的最大矩形，返回面积。微软)动态规划；84，85相似

89格雷编码(微软)

94二叉树的中序遍历(非递归)

95不同的二叉搜索树II(微软：给定一个整数n,生成所有由1…n为节点所组成的二叉搜索树)递归建树

96不同的二叉搜索树

97交错字符串(微软 给定三个字符串s1,s2,s3，验证s3是否是由s1和s2交错组成的)

98验证二叉搜索树(最简单的方法，非递归中序遍历)

101对称二叉树(二叉树镜像)

102二叉树的层次遍历

104二叉树的最大深度

105从前序遍历和中序遍历序列构造二叉树

114二叉树展开为链表(二叉树先遍历保存入栈，出栈，尾插法)

121买卖股票的最佳时机(只能完成一次交易)

124二叉树中的最大路径和(后序遍历，一笔画的路径，路径不能重复走)

127 单词接龙(给定两个单词beg,end和一个字典,找到从beg到end的最短转换序列的长度)：BFS；优化：双向BFS

128最长连续序列(给定一个未排序的整数数组，找出最长连续序列的长度 O(n))

136只出现一次的数字(位运算leetcode 260剑指offer)

139单词拆分(动态规划dp[i],bool:表示字符串(0…..i)能否拆分;将字符串拆分成dp[j]和s.substr(j,i-j)两部分，如果这两部分都存在于字典中，那么dp[i]就是可以拆分的。)微软

140单词拆分II微软

141环形链表(判断是否有环)

142环形链表II(有环，返回环的第一个节点。无环，返回null)

143重排链表(微软)首先找到链表的后半部分(找链表的中间节点)，将链表分成两部分，将后半部分原地逆置，两部分节点分别使用。

146LRU缓存

148排序链表(O(nlogn)快排链表,partition时，前后指针p,q，0~p位置都<partition,p+1位置后>partition)

152乘积最大子序列(给定一个整数数组，找出一个序列中乘积最大的连续子序列。百度)

思路：动态规划。*f*max​(*i*)=max​{*fmax*(*i*−1)×*ai*​, *fmin*(i-1)x*ai*​, *ai*​} 依次遍历数组。使用三个变量：当前元素值，最大元素值，最小元素值，遍历时不断更新三个变量。最后返回最大变量(最大子序列。)

155最小栈(另外用一个栈，永远只进栈最小的元素，出栈最小的元素)

160相交链表(O(m+n))

164最大间距(微软：给定一个无序数组，在数组排序后，求出相邻元素之间最大的差值)桶排序+鸽笼原理

169求众数(给定一个大小为n的数组，找到其中的众数。众数是指在数组众出现次数大于n/2的元素。摩尔投票法)

171Excel表列序号(微软 进制转换)

187重复的DNA序列(微软)给定一个串S，在串S中查找目标字串，目标字串的长度为10，且在串S中出现次数超过1次。基于滑动窗口的hashset

198打家劫舍(数组相邻两个元素不能同时取的dp[0] = nums[0], dp[1] = max(nums[0],nums[1]), dp[i] = max(dp[i-1], dp[i-2] + nums[i])) (i>=2时)

200岛屿的个数(二维网格，’1’代表陆地，’0’代表水，计算岛屿的数量。把访问过的’1’置为’2’，防止重复访问)

206反转链表

207课程表(拓扑排序模板：构建邻接表和入度数组->入度为0入队列->广度遍历。如果能够遍历到所有顶点，说明没有环 MS)

208实现trie(前缀树)

214最短回文串(微软)：给定一个字符串S，通过在字符串前面添加字符将其转换为回文串。找出用这种方式转换的最短回文串。

215数组中的第k个最大元素(小根堆)

221最大正方形(最大矩形 微软：在一个由0和1组成的二维矩阵内，找到只包含1的最大正方形，返回其面积)

223矩形面积(微软 求两个矩形重叠后形成的总面积。每个矩形给出左下定点和右上定点坐标) 两个矩形的面积相加， 减去重叠的部分面积

226翻转二叉树

234回文链表(判断链表是否是回文链表)

236二叉树的最近公共祖先(后序遍历)

238除自身以外数组的乘积(MS)

239滑动窗口的最大值

240搜索二维矩阵II

279完全平方数

283移动零(给定一个数组，将所有0移动到数组的末尾，同时保持非零元素的相对顺序)

284顶端迭代器(微软。给定一个vector数组，实现next(),hasnext(),peek()。next()返回下一个，且迭代器指针后移，peek()返回下一个，迭代器指针不动)

287寻找重复数(排序后相邻的数相等，就是重复或者用set)

297二叉树的序列化和反序列化

300最长上升子序列

303区域和检索-数组不可变(微软：如果一个函数被调用1000次，其参数完全相同，怎么才能加快速度？ 缓存)

307区域和检索-数组可修改(微软 线段树：二叉树型数据结构。解决范围查找问题，在对数时间内从数组中找到最小值，最大值，总和，最大公约数，最小公倍数等)

309最佳买卖股票时期含冷冻期

312戳气球(dp)

316去除重复字母(微软 贪心+栈)

322零钱兑换(动态规划):dp[i] = min(dp[i],dp[i-coins(j)+1]);dp[i]表示需要兑换的钱。要兑换dp[i],首先要能兑换dp[i-conis[j]],dp[0] = 0表示兑换元有0中方法；初始dp[i] = INT\_MAX

334递增的三元子序列(微软)

337打家劫舍III

338比特位计数(栈)

341扁平化嵌套列表迭代器(微软)

347前K个高频元素(map，排序)

365水壶问题(微软：两个容量分别为x升和y升的水壶以及无限多的水，能否得到z升的水BFS)

386字典序排数(微软 :N叉树)

394字符串解码

406根据身高重建队列(打乱顺序的一群人站成一个队列。每个人由一个整数对(h,k)表示，h是该人的身高，K是排在这个人前面且身高大于等于h的人数，要求重建队列)

416分割等和子集(给定一个只包含正整数的非空数组。是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。0-1背包)

437路径总和III(给定二叉树，每个节点存放一个整数值。找出路径和等于给定数值的路径总数。路径不需要从根节点开始，也不需要在叶子节点结束)

438找到字符串中所有字母异位词(滑动窗口+哈希表)

448找到所有数组中消失的数字(一次遍历，不断swap)

457环形数组循环(微软双支针)：对于查找数组或者链表中有没有环的问题，多可以使用双支针(快慢指针)的思想 (1000\*len + nums[i] + i) % len 用1000是为了防止出现负数

461汉明距离(将两数分别右移(/2)，每一位%2作比较)

463岛屿的周长(微软 dfs)：岛的周长=从岛中的某一块到边界+到水中的和

494目标和(简单DFS)

503下一个更大元素II(微软 单调栈)

525连续数组(微软)：给定一个二进制数组，找到含有相同数量的0和1的最长连续子数组的长度。

538把二叉搜索树转换为累加树

541反转字符串II(微软)

543二叉树的直径(二叉树的直径长度是任意两个节点路径长度(不是节点个数，是节点个数减一)中的最大值。后序遍历，有点类似于二叉树的路径和)

560和为K的子数组(给定一个整数数组和一个整数k,找到和为k的连续的子数组的个数。用数组保存累加和)

572另一个树的子树(给定两棵树，判断其中一颗是否是另一个的子树 剑指offer)

582最短无序连续子数组(给定一个整数数组，需要寻找一个连续的子数组，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。你要找的子数组应是最短的，输出它的长度)(从前往后遍历，得到数组右边界；从后往前遍历，得到数组左边界。长度是右边界减去左边界)

617合并二叉树

621任务调度器(微软)：相同的任务之间必须有长度为n的冷却时间。用A-Z表示26中不同的任务，任务可以以任意顺序执行，并且每个任务在1个单位时间内执行完成，求完成所有任务所需的最短时间。

647回文字串(回文中心法，从中心往两边扩散，判断左右两边是否一致)

695岛屿的最大面积(微软)做法同之前岛屿数量基本一致

707设计链表(微软：单链表)

771宝石与石头

LC top 面试题

26.删除排序数组中的重复项(前后指针，后指针不断向后遍历，前指针发现有不重复的值，才往后移动，并且把之后一个位置赋值为不重复的值。有点类似单链表快排，也是前后指针)

27.移除元素(同上)

283.移动零(同上)

剑指offer：

不用加减乘除做加法：位运算。两数的和看作： 非进位的和(两数做异或运算) + 进位的和(两数与运算后<<1)，迭代直到进位的和为0