1. Two Sum

```
/* 1 Two Sum 2017-11-28

给一个数组和一个数字，要求在数组里找出相加等于目标数字的两个数
输出它们的序号

Example:
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].

分析：暴力破解

*/

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */

int* twoSum(int* nums, int numsSize, int target) {
    int num_one, num_two;
    int* ans;
    ans = (int*)malloc(2 * sizeof(int));
    for(num_one = 0; num_one < numsSize; num_one++)
        for(num_two = num_one + 1; num_two < numsSize; num_two++)
        {
            if(nums[num_one] + nums[num_two] == target)
            {
                *ans = num_one;
                *(ans + 1) = num_two;
            }
        }
    return ans;
}
```

```
// java暴力解法
class Solution {
    public int[] twoSum(int[] nums, int target) {
        for(int i = 0; i < nums.length; i++)
        {
            for(int j = i + 1; j < nums.length; j++)
            {
                if(nums[j] + nums[i] == target)
                    return new int[] {i, j};
            }
        }
        throw new IllegalArgumentException("No two sum solution");
    }
}
```

```
/* java hash解法 时间复杂度为O(n)

利用补(target - nums[i]target-nums[i]) 和 hash表 减少查找时间
*/

class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for(int i = 0; i < nums.length; i++)//建立hash表
        {
            map.put(nums[i], i);
        }
        for(int i = 0; i < nums.length; i++)
        {
            int complement = target - nums[i];//得出补数

            //减少了查找的时间，就只用查询补数是否存在和是否为自己
            if(map.containsKey(complement) && map.get(complement) != i)
                return new int[] {i, map.get(complement)};
        }
        throw new IllegalArgumentException("No two sum solution");
    }
}
```

```
/* java hash解法优化 时间复杂度为O(n)

减少建立hash表的时间
*/

class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for(int i = 0; i < nums.length; i++)
        {
            int complement = target - nums[i];//得出补数

            //减少了查找的时间
            if(map.containsKey(complement))
                return new int[] {i, map.get(complement)};

            map.put(nums[i], i);//补充hash表
        }
        throw new IllegalArgumentException("No two sum solution");
    }
}
```

2. Add Two Numbers

```
/* LeetCode 2. Add Two Numbers 2017-12-5

题意：
两个链表里存储着两个数字，链表每一个点存一位数字，按低位存到高位
要求输出一个相同格式的链表，数字是这两链表数字的和

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
Explanation: 342 + 465 = 807.

分析：
因为链表没有长度限制，所以不能想着把它们化出来相加先得到sum

*/
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* ans = new ListNode(-1);
        ListNode* cnt = ans;
        int first = 1, temp2 = 0;
        while(l1 != NULL && l2 != NULL)
        {
            int temp = l1->val + l2->val + temp2;
            temp2 = 0;
            if(temp > 9)
            {
                temp2 = temp / 10;
                temp %= 10;
            }
            cnt->next = new ListNode(temp);
            cnt = cnt->next;
            l1 = l1->next;
            l2 = l2->next;
        }
        while(l1 != NULL)
        {
            int temp = l1->val + temp2;
            temp2 = 0;
            if(temp > 9)
            {
                temp2 = temp / 10;
                temp %= 10;
            }
            cnt->next = new ListNode(temp);
            cnt = cnt->next;
            l1 = l1->next;
        }
        while(l2 != NULL)
        {
            int temp = l2->val + temp2;
            temp2 = 0;
            if(temp > 9)
            {
                temp2 = temp / 10;
                temp %= 10;
            }
            cnt->next = new ListNode(temp);
            cnt = cnt->next;
            l2 = l2->next;
        }
        if(temp2 != 0) cnt->next = new ListNode(temp2);
        return ans->next;
    }
};
```

```
//java解法 时间复杂度不变为O(max(m, n))
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummyHead = new ListNode(0);
        ListNode p = l1, q = l2, curr = dummyHead;
        int carry = 0;
        while(p != null || q != null)
        {
            int x = (p != null) ? p.val : 0;
            int y = (q != null) ? q.val : 0;
            int sum = carry + x + y;
            carry = sum / 10;
            curr.next = new ListNode(sum % 10);
            curr = curr.next;
            if(p != null) p = p.next;
            if(q != null) q = q.next;
        }
        if(carry > 0)
        {
            curr.next = new ListNode(carry);
        }
        return dummyHead.next;
    }
}
```

``

3. Longest Substring Without Repeating Characters

```
/* LeetCode 3. Longest Substring Without Repeating Characters 2017-12-5

题意：
再一个字符串里找出一个连续的没有相同字符的子字符串

下面是大佬的解答

*/

/**
 * Solution (DP, O(n)):
 *
 * Assume L[i] = s[m...i], denotes the longest substring without repeating
 * characters that ends up at s[i], and we keep a hashmap for every
 * characters between m ... i, while storing <character, index> in the
 * hashmap.
 * We know that each character will appear only once.
 * Then to find s[i+1]:
 * 1) if s[i+1] does not appear in hashmap
 *    we can just add s[i+1] to hash map. and L[i+1] = s[m...i+1]
 * 2) if s[i+1] exists in hashmap, and the hashmap value (the index) is k
 *    let m = max(m, k), then L[i+1] = s[m...i+1], we also need to update
 *    entry in hashmap to mark the latest occurency of s[i+1].
 *
 * Since we scan the string for only once, and the 'm' will also move from
 * beginning to end for at most once. Overall complexity is O(n).
 *
 * If characters are all in ASCII, we could use array to mimic hashmap.
 */


class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        // for ASCII char sequence, use this as a hashmap
        vector<int> charIndex(256, -1);//因为是英文所以用ASCII特点创建数组来作为hashmap
         int longest = 0, m = 0;

        for (int i = 0; i < s.length(); i++) {
            m = max(charIndex[s[i]] + 1, m);    // automatically takes care of -1 case
            //如果字符未出现的情况则longest-0;
            //如果字符已经出现则longest-(出现字符的位置+1)
            charIndex[s[i]] = i;//更新字母在字符串里的位置
            longest = max(longest, i - m + 1);
        }

        return longest;
    }
};
```

6. ZigZag Conversion

```
// 以z型输入一串字符串，求它原本的顺序
class Solution {
public:
    string convert(string s, int numRows) {
        if(numRows == 1) return s;

        //计算经过多少个字符后将回到第一行
        int n = 2 * numRows - 2;
        string ans[numRows];

        for(int i = 0; i < s.size(); i++)
        {
            int lineNum = i % n;
            if(lineNum < numRows)
                ans[lineNum] += s[i];
            else
                ans[2*numRows - lineNum - 2] += s[i];

        }

        string a;
        for(int i = 0; i < numRows; i++)
            a += ans[i];

        return a;
    }
};
```

7. Reverse Integer

```
// Runtime: 43 ms
// Reverse Integer with an environment which could only hold integers within the 32-bit signed integer range
// 在一个只有32-bit signed integer range的系统里翻转一个数字
class Solution {
public:
    int reverse(int x) {

        //negative
        if(x == INT_MIN) return 0;
        if(x < 0) return -reverse(-x);

        int rx = 0;
        while(x != 0)
        {
            //check overflow
            if(rx > INT_MAX / 10) return 0;

            rx = rx*10 + x%10;
            x /= 10;
        }

        return rx;
    }
};
```

8. String to Integer (atoi)

```
class Solution {
public:
    int myAtoi(string str) {
        int i;
        for(i = 0; i < str.size(); i++)
        {
            if(str[i] == ' ')
                continue;
            /*else if(str[i] == '-')
                break;
            else if(str[i] == '+')
                break;
            else
                continue;*/
        }

        if(i == str.size()) return 0;

        bool judge = true;
        if(str[i] == '-')
        {
            judge = false;
            i++;
        }
        else if(str[i] == '+')
            i++;
        else if(str[i] < '0' || str[i] > '9')
            return 0;

        int ans = 0;
        for(;i < str.size(); i++)
        {
            if(str[i] >= '0' && str[i] <= '9')
            {
                if((ans*10 + (str[i] - '0')) > INT_MAX || (ans*10 + (str[i] - '0')) < 0)
                {
                    if(judge)
                        return INT_MAX;
                    else
                        return INT_MIN;
                }
                else
                    ans = ans*10 + (str[i] - '0');
            }
            else
                break;
        }

        if(judge)
            return ans;
        else
            return -ans;
    }
};
```

13. Roman to Integer

```
// 将罗马数字转换成数字
// about unordered_map  https://blog.csdn.net/u012530451/article/details/53228098
class Solution {
public:
    int romanToInt(string s) {
        unordered_map<char, int> T = {
            {'I', 1}, {'V', 5}, {'X', 10}, {'L', 50}, {'C', 100}, {'D', 500}, {'M', 1000}
        };

        int ans = T[s.back()];
        for(int i = s.size() - 2; i >= 0; i--)
        {
            if(T[s[i]] < T[s[i+1]])
                ans -= T[s[i]];
            else
                ans += T[s[i]];
        }
        return ans;
    }
};
```

20. Valid Parentheses

```
//考察符号匹配
class Solution {
public:
    bool isValid(string s) {
        typedef string::size_type str_s;

        stack<char> Stack;

        for(str_s i = 0; i < s.size(); i++)
        {
            if(s[i] == '(' || s[i] == '{' || s[i] == '[')
                Stack.push(s[i]);

            if(Stack.empty()) return false;
            else
            {
                if(s[i] == ')')
                {
                    if(Stack.top() == '(') Stack.pop();
                    else return false;
                }
                else if(s[i] == '}')
                {
                    if(Stack.top() == '{') Stack.pop();
                    else return false;
                }
                else if(s[i] == ']')
                {
                    if(Stack.top() == '[') Stack.pop();
                    else return false;
                }
            }
        }
        if(Stack.empty())
            return true;
        else
            return false;
    }
};
```

561. Array Partition I

```
/* 2017-11-28

给定一个2n整数的数组，你的任务是把这些整数分成n对整数，例如（a 1，b 1），（a 2，b 2），...，（a n，b n）所有我从1到n 的min（a i，b i）之和尽可能大

Example 1:
Input: [1,4,3,2]
Output: 4
Explanation: n is 2, and the maximum sum of pairs is 4 = min(1, 2) + min(3, 4).

Note:
n is a positive integer, which is in the range of [1, 10000].
All the integers in the array will be in the range of [-10000, 10000].

分析：
对输入的2n数组进行排序，从小到达两两取min，最后的sum将最大

*/

int arrayPairSum(int* nums, int numsSize) {
    int n = numsSize / 2;
    int* group = (int*)malloc(sizeof(int) * n);//创建一个n数组
    int count, temp;
    int ans;
    int com(const void *a, const void *b){ return *(int *)a - *(int *)b; }//从小到大

    qsort(nums, numsSize, sizeof(int), com);
    for(count = 0, temp = 0; count < numsSize; count = count + 2, temp++)
    {
        if(nums[count] < nums[count+1]) group[temp] = nums[count];
        else group[temp] = nums[count+1];
    }

    for(count = 0, ans = 0; count < n; count++)
    {
        ans = ans + group[count];
    }
    return ans;
}
```

724. Find Pivot Index

```
/* 2017-11-26

求一个数列里某一个数：它的左边所有数的和=右边所有数的和
并输出此数所在的位置

Input:
nums = [1, 7, 3, 6, 5, 6]
Output: 3
Explanation:
The sum of the numbers to the left of index 3 (nums[3] = 6) is equal to the sum of numbers to the right of index 3.
Also, 3 is the first index where this occurs.

Input:
nums = [1, 2, 3]
Output: -1
Explanation:
There is no index that satisfies the conditions in the problem statement.

The length of nums will be in the range [0, 10000].
Each element nums[i] will be an integer in the range [-1000, 1000].

分析：
可以暴力破解，
用一个数组s存储从左边开始数的累和，d为右边开始的累和
最后遍历s和d，当s[i] = d[i]时，输出i-1

*/

class Solution {
public:
    int pivotIndex(vector<int>& nums) {
        const int n = nums.size();
        vector<int> s(n + 3), d(n + 3);
        for (int i = 1; i <= n; ++i) s[i] = s[i - 1] + nums[i - 1];
        for (int i = n; i >= 1; --i) d[i] = d[i + 1] + nums[i - 1];
        for (int i = 1; i <= n; ++i) if (s[i] == d[i]) return i - 1;
        return -1;
    }
};
```

725. Split Linked List in Parts

```
/* 2017-11-28

题意：
给一个链表和一个k，要让链表尽量k等分（任两部分不能相差超过1），过多的可以为null
各部分输出要按原来顺序，而且前部分一定大于或等于后面的部分

Input:
root = [1, 2, 3], k = 5
Output: [[1],[2],[3],[],[]]
Explanation:
The input and each element of the output are ListNodes, not arrays.
For example, the input root has root.val = 1, root.next.val = 2, \root.next.next.val = 3, and root.next.next.next = null.
The first element output[0] has output[0].val = 1, output[0].next = null.
The last element output[4] is null, but it's string representation as a ListNode is [].

Input:
root = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 3
Output: [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]
Explanation:
The input has been split into consecutive parts with size difference at most 1, and earlier parts are a larger size than the la
ter parts.

分析：
简单题，注意一下两个情况
1-当k大于链表时
2-链表不能被k整除时

*/
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */

class Solution {
    int len(ListNode *x) {//得到链表的长度
        int ret = 0;
        while (x) x = x->next, ret++;
        return ret;
    }
public:
    vector<ListNode*> splitListToParts(ListNode* root, int k) {
        int l = len(root);
        vector<ListNode*> ret;
        for (int i = 0; i < k; ++i) {
            if (root == nullptr) {//nullptr是C++ 11的空指针
                ret.push_back(nullptr);
                continue;
            }
            int t = (l / k) + (i < (l % k));//用(i < (l % k))把不整分的部分加到前面中，每次加1
            ret.push_back(root);
            for (int i = 0; i < t - 1; ++i) root = root->next;
            ListNode *tmp = root->next;
            root->next = nullptr;
            root = tmp;
        }
        return ret;
    }
};
```

728. Self Dividing Numbers

```
/* LeetCode Weekly Contest 59 第一题
 * problem 728 Self Dividing Numbers

 * 给定一个范围：left <= n <= right
 * 求在这范围里每一个能被自己每一位数字整除的整数
 * 如：128 % 1 == 0, 128 % 2 == 0, and 128 % 8 == 0 所以128符合要求
 * 要排除有除数为零时的整数

Input:
left = 1, right = 22
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]

 * 关于vetor的资料：
 * http://www.cnblogs.com/mr-wid/archive/2013/01/22/2871105.html
 * http://blog.csdn.net/hancunai0017/article/details/7032383
 */

//他人解法
class Solution {
public:
    vector<int> selfDividingNumbers(int left, int right) {
        vector<int> a;
        for (int i = left; i <= right; ++i) {
            string s = to_string(i);//把读入的数字转变为字符串
            bool ok = true;
            for (char c : s) {//遍历字符串的字符
                int x = c - '0';
                if (x == 0 || i % x) {
                    ok = false;
                    break;
                }
            }
            if (ok) a.push_back(i);//把合格的数字压入数组
        }
        return a;
    }
};

//我的解法
class Solution {
public:
    vector<int> selfDividingNumbers(int left, int right) {
        int cnt = 0;
        vector<int> ans(10010, 0);
        for(int i = left; i <= right; i++)
        {
            if(i == 10000)
            {
                ans[cnt] = 10000;cnt++;
                break;
            }
            else if(i >= 1000)
            {
                int a = i % 10;
                int b = (i % 100 - a) / 10;
                int c = (i % 1000 - b - a) / 100;
                int d = (i - c - b - a) / 1000;
                if(a!=0&&i%a==0 && b!=0&&i%b==0 && c!=0&&i%c==0 && d!=0&&i%d==0)
                {
                    ans[cnt] = i;
                    cnt++;
                }
            }
            else if(i >= 100)
            {
                int a = i % 10;
                int b = (i % 100 - a) / 10;
                int c = (i % 1000 - b - a) / 100;
                if(a!=0&&i%a==0 && b!=0&&i%b==0 && c!=0&&i%c==0)
                {
                    ans[cnt] = i;
                    cnt++;
                }
            }
            else if(i >= 10)
            {
                int a = i % 10;
                int b = (i % 100 - a) / 10;
                if(a!=0&&i%a==0 && b!=0&&i%b==0)
                {
                    ans[cnt] = i;
                    cnt++;
                }
            }
            else
            {
```

```
                ans[cnt] = i;
                cnt++;
            }
        }
        vector<int> answ(cnt,0);
        for(int i = 0; i <= cnt-1; i++) answ[i] = ans[i];
        return answ;
    }
};
```

729. My Calendar I

```
/* LeetCode Weekly Contest 59 第二题
 * Probelm 729 My Calendar I

 * 每次都给定一个范围：start <= x < end
 * 然后设定一个函数
 * 如果给定的范围都不曾给过，就进行记录，并返回ture；如果已经给过，就返回false

MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true
Explanation:
The first event can be booked.  The second can't because time 15 is already booked by another event.
The third event can be booked, as the first event takes every time less than 20, but not including 20.

 * 关于pair的资料：
 * http://blog.csdn.net/edward_wong/article/details/39480527
 * https://www.quora.com/How-can-I-use-pair-int-int-v-in-C++-language
 *
 * 关于map的资料：
 * https://www.cnblogs.com/hailexuexi/archive/2012/04/10/2440209.html
 */

//他人解法
using PII = pair<int, int>;
#define F first
#define S second
class MyCalendar {
public:
    map<PII, int> m;
    MyCalendar() {

    }

    bool book(int start, int end) {
        m[{end, -1}]++;//注意这里的巧妙设计
        m[{start, 1}]++;
        int s = 0;
        for (const auto &it : m) {
            s += it.S * it.F.S;
            if (s >= 2) {//如果在一段区域内出现了第二个start，s就会>=2
                m[{end, -1}]--;
                m[{start, 1}]--;
                return false;
            }
        }
        return true;
    }
};

/**
 * Your MyCalendarTwo object will be instantiated and called as such:
 * MyCalendarTwo obj = new MyCalendarTwo();
 * bool param_1 = obj.book(start,end);
 */
```

730. Count Different Palindromic Subsequences

```
/* LeetCode Weekly Contest 59 第三题
 * problem 730 Count Different Palindromic Subsequences

 * 给一个字符串，求出它所有非空子串是回文字符串的个数（相同的字串不重复计算
 * 求出的数目取 mod 10^9 + 7

Input:
S = 'bccb'
Output: 6
Explanation:
The 6 different non-empty palindromic subsequences are 'b', 'c', 'bb', 'cc', 'bcb', 'bccb'.
Note that 'bcb' is counted only once, even though it occurs twice.

Input:
S = 'abcdabcdabcdabcdabcdabcdabcdabcddcbadcbadcbadcbadcbadcbadcbadcba'
Output: 104860361
Explanation:
There are 3104860382 different non-empty palindromic subsequences, which is 104860361 modulo 10^9 + 7.

The length of S will be in the range [1, 1000].
Each character S[i] will be in the set {'a', 'b', 'c', 'd'}.
 */

//他人解法
#define REP(i, n) for (int i = 0; i < int(n); ++i)
using LL = long long;

const LL mod = 1000000007;
char s[1010];
int nxt[1010][26];
int lst[1010][26];

LL dp[1010][1010];

LL f(int l, int r) {
    LL &ret = dp[l][r];
    if (ret >= 0)
      return ret;
    if (l > r) return ret = 1;
    ret = 1;
    REP(i, 26) {
      int a = nxt[l][i];
      int b = lst[r][i];
      if (a < b) ret += f(a + 1, b - 1);
      if (a <= r) ret += 1;
    }
    ret %= mod;
    return ret;
}


int solve(string S) {
    strcpy(s + 1, S.c_str());//c_str()返回一个临时指针
    int n = strlen(s + 1);
    REP(i, 26) nxt[n + 1][i] = n + 1;//初始化第n+1个数组，数字为27（超出26
    for (int i = n; i >= 1; --i) {
      REP(j, 26) nxt[i][j] = nxt[i + 1][j];//将后面的数组都初始化为n+1数组
      nxt[i][s[i] - 'a'] = i;//将字符串转变为数字，nxt[字符串里第i个字母][此字母在字母表中的位置]=i
    }
    REP(i, 26) lst[0][i] = 0;//初始化空出来的第一个数组
    for (int i = 1; i <= n; ++i) {
      REP(j, 26) lst[i][j] = lst[i - 1][j];//将初始化向后赋值
      lst[i][s[i] - 'a'] = i;//同nxt
    }
    memset(dp, -1, sizeof dp);
    LL ans = f(1, n);
    ans = ans == 0 ? mod - 1 : ans - 1;
    return ans;
}



class Solution {
public:
    int countPalindromicSubsequences(string S) {
        return solve(S);
    }
};
```

731. My Calendar II

```
/* LeetCode Weekly Contest 59 第四题
 * Probelm 731 My Calendar II

 * 每次都给定一个范围：start <= x < end
 * 然后设定一个函数
 * 如果给定的范围都不曾给超过三次（包括第三次）过，就进行记录，并返回ture；如果已经给过，就返回false

MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(50, 60); // returns true
MyCalendar.book(10, 40); // returns true
MyCalendar.book(5, 15); // returns false
MyCalendar.book(5, 10); // returns true
MyCalendar.book(25, 55); // returns true
Explanation:
The first two events can be booked.  The third event can be double booked.
The fourth event (5, 15) can't be booked, because it would result in a triple booking.
The fifth event (5, 10) can be booked, as it does not use time 10 which is already double booked.
The sixth event (25, 55) can be booked, as the time in [25, 40) will be double booked with the third event;
the time [40, 50) will be single booked, and the time [50, 55) will be double booked with the second event.

The number of calls to MyCalendar.book per test case will be at most 1000.
In calls to MyCalendar.book(start, end), start and end are integers in the range [0, 10^9].

 */

//他人解法
using PII = pair<int, int>;
#define F first
#define S second
class MyCalendarTwo {
public:
    map<PII, int> m;
    MyCalendarTwo() {

    }

    bool book(int start, int end) {
        m[{end, -1}]++;
        m[{start, 1}]++;
        int s = 0;
        for (const auto &it : m) {
            s += it.S * it.F.S;
            if (s >= 3) {
                m[{end, -1}]--;
                m[{start, 1}]--;
                return false;
            }
        }
        return true;
    }
};
```

732. My Calendar III

```
/* Probelm 732 My Calendar III

 * 每次都给定一个范围：start <= x < end
 * 然后设定一个函数，返回重复预订的次数

Example 1:

MyCalendarThree();
MyCalendarThree.book(10, 20); // returns 1
MyCalendarThree.book(50, 60); // returns 1
MyCalendarThree.book(10, 40); // returns 2
MyCalendarThree.book(5, 15); // returns 3
MyCalendarThree.book(5, 10); // returns 3
MyCalendarThree.book(25, 55); // returns 3

Explanation:
The first two events can be booked and are disjoint, so the maximum K-booking is a 1-booking.
The third event [10, 40) intersects the first event, and the maximum K-booking is a 2-booking.
The remaining events cause the maximum K-booking to be only a 3-booking.
Note that the last event locally causes a 2-booking, but the answer is still 3 because
eg. [10, 20), [10, 40), and [5, 15) are still triple booked.
Note:

The number of calls to MyCalendarThree.book per test case will be at most 400.
In calls to MyCalendarThree.book(start, end), start and end are integers in the range [0, 10^9].

 */

using PII = pair<int, int>;
#define F first
#define S second
class MyCalendarThree {
public:
    map<PII, int> m;
    MyCalendarThree() {

    }

    int book(int start, int end) {
        m[{end, -1}]++;
        m[{start, 1}]++;
        int s = 0, maxn = 0;
        for (const auto &it : m) {
            s += it.S * it.F.S;
            if (maxn <= s) {
                maxn = s;
            }
        }
        return maxn;
    }
};
```

733. Flood Fill

```
/* LeetCode Weekly Contest 60 第一题
 * problem 733 Flood Fill

给定一个二维平面，二维数组内容都是0到65535的数组
再给一个起始点，给一个染色数字，让起始点与所有与起始点同数字且与起始点在四个方向连成一片的点都变成染色数字

Input:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2

Output: [[2,2,2],[2,2,0],[2,0,1]]

Explanation:
From the center of the image (with position (sr, sc) = (1, 1)), all pixels connected
by a path of the same color as the starting pixel are colored with the new color.
Note the bottom corner is not colored 2, because it is not 4-directionally connected
to the starting pixel.

1, 1, 1   2, 2, 2
1, 1, 0 ->2, 2, 0
1, 0, 1   2, 0, 1

The length of image and image[0] will be in the range [1, 50].

The given starting pixel will satisfy 0 <= sr < image.length and 0 <= sc < image[0].length.

The value of each color in image[i][j] and newColor will be an integer in [0, 65535].

解法：
可以用队列和标记法来实现广度优先搜索来解决

 */

class Solution {
public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {
        int n = image.size(), m = image[0].size();//n为行数，m为列数
        vector<vector<int> > res = image;//复制出一个相同的地图
        queue<pair<int, int> > que;//创建一个队列，成员是pair<int, int>
        vector<vector<bool> > vis(n, vector<bool>(m));//创建一个bool类型的二维数组，这二维数组有n行，vector<bool>(m)表示构造一个无名且含m个0的vector<bool>对象
        que.push(make_pair(sr, sc));//在队列中压入起始点
        vis[sr][sc] = true;//把起始点的内容转换并标记
        res[sr][sc] = newColor;
        int dx[] = {0, 1, 0, -1};//前进的四个方向
        int dy[] = {-1, 0, 1, 0};
        //对于队列中的每个点依照先进先出的原则，对每个出队的点寻找它四个方向的点，如果有与出队点颜色颜色相同且未被标记的点，就改变其颜色且标记，并入队，直到队列非空为止
        while (!que.empty()) {
            int x = que.front().first, y = que.front().second;
            que.pop();
            for (int dir = 0; dir < 4; dir++) {
                int nx = x + dx[dir], ny = y + dy[dir];
                if (0 <= nx && nx < n && 0 <= ny && ny < m && !vis[nx][ny] && image[x][y] == image[nx][ny]) {
                    res[nx][ny] = newColor;
                    vis[nx][ny] = true;
                    que.push(make_pair(nx, ny));
                }
            }
        }
        return res;
    }
};
```

734. Sentence Similarity

```
/* LeetCode Weekly Contest 60 第二题
 * problem 734 Sentence Similarity

给两个字符串和一个相似单词列表，判断俩字符串是否相似

For example, "great acting skills" and "fine drama talent" are similar, if the similar word pairs are pairs = [["great", "fin
e"], ["acting","drama"], ["skills","talent"]].

这种相似没有传递性，而且每个单词与它自己总是相似的

Note:

The length of words1 and words2 will not exceed 1000.
The length of pairs will not exceed 2000.
The length of each pairs[i] will be 2.
The length of each words[i] and pairs[i][j] will be in the range [1, 20].

 * 关于map的资料：
 * https://www.cnblogs.com/hailexuexi/archive/2012/04/10/2440209.html
 * 关于set的资料：
 * https://www.cnblogs.com/wonderKK/archive/2012/04/10/2441379.html

解法：
灵活运用STL进行解答
 */

class Solution {
public:
    bool areSentencesSimilar(vector<string>& words1, vector<string>& words2, vector<pair<string, string>> pairs) {
        map<string, set<string> > dict;//用map建立一一对应的关系，set建立一个不重复的字符串有序数组
        for (auto item : pairs) {//遍历相似单词对应表
            dict[item.first].insert(item.second);//建立第一个单词和第二个单词的对应关系
            dict[item.second].insert(item.first);
        }
        if (words1.size() != words2.size()) {//如果两字符串的单词数量不同则必然不相似
            return false;
        }
        //如果有单词不相等且由word1查找到表尾部也未能找到对应的相似单词，则返回false
        for (int i = 0; i < words1.size(); i++) {
            if (words1[i].compare(words2[i]) != 0 && dict[words1[i]].find(words2[i]) == dict[words1[i]].end()) {
                return false;
            }
        }
        return true;
    }
};
```

735. Asteroid Collision

```
/* LeetCode Weekly Contest 60 第三题
 * problem 735 Asteroid Collision

输入一个数组，正数代表向右移动，负数代表向左一定
两数相撞小的消失（绝对值比较），如果两数相等则都消失，输出最后的数组

Example 1:

Input:
asteroids = [5, 10, -5]

Output: [5, 10]

Explanation:
The 10 and -5 collide resulting in 10.  The 5 and 10 never collide.

Example 2:

Input:
asteroids = [8, -8]

Output: []

Explanation:
The 8 and -8 collide exploding each other.

Example 3:
Input:
asteroids = [10, 2, -5]

Output: [10]

Explanation:
The 2 and -5 collide resulting in -5.  The 10 and -5 collide resulting in 10.

Example 4:

Input:
asteroids = [-2, -1, 1, 2]

Output: [-2, -1, 1, 2]

Explanation:
The -2 and -1 are moving left, while the 1 and 2 are moving right.
Asteroids moving the same direction never meet, so no asteroids will meet each other.
Note:

The length of asteroids will be at most 10000.
Each asteroid will be a non-zero integer in the range [-1000, 1000]..

 * 关于deque的资料：
 * https://www.cnblogs.com/scandy-yuan/archive/2013/01/09/2853603.html

解法：
应用deque灵活解

 */

class Solution {
public:
    vector<int> asteroidCollision(vector<int>& asteroids) {
        deque<int> que;//创建
        vector<int> &a = asteroids;
        int n = a.size();
        vector<int> res;
        for (int i = 0; i < n; i++) {//遍历每个数
            if (a[i] < 0) {//数小于0时
                bool exist = true;
                while (!que.empty()) {//deque非空
                    int val = que.back() + a[i];
                    //判断最后压入的正数是否大于此负数
                    if (val < 0) {//如果整数小就删除这个正数
                        que.pop_back();
                    } else if (val == 0) {//相等时
                        que.pop_back();
                        exist = false;
                        break;
                    } else {//正数大时直接无视此负数
                        exist = false;
                        break;
                    }
                }
                if (exist) {//当负数大于所有整数时压入负数
                    res.push_back(a[i]);
                }
            } else {//数为正数时就压入deque
```

```
                que.push_back(a[i]);
            }
        }
        while (!que.empty()) {
            res.push_back(que.front());
            que.pop_front();
        }
        return res;
    }
};
```

738. Monotone Increasing Digits

```
/* LeetCode Weekly Contest 61
 * problem 738. Monotone Increasing Digits

题意：
给一个数n，求出比n小的能让各个位的数字从小到大的最大的数

Example 1:
Input: N = 10
Output: 9

Example 2:
Input: N = 1234
Output: 1234

Example 3:
Input: N = 332
Output: 299

Note: N is an integer in the range [0, 10^9].

*/

class Solution {
public:
    int monotoneIncreasingDigits(int N) {
        vector<int> a;
        for (int t = N; t; t /= 10) a.push_back(t % 10);//从高位到低位压入n的每一位数
        int m = a.size();
        int ret = 0, pre = 0;//ret为返回的结果数字，pre记录目前我们已经取得的数字的最大值
        for (int i = 0; i < m; ++i)//从高位到低位检测n的每一个数
        {
            for (int k = 9; k >= pre; --k)//对于每一位，检测从pre到9那个数字合适
            {
                int cur = ret;//把目前已经取得的数字赋值给cur
                for (int j = i; j < m; ++j)//目前距离最低位还有多少个数字就进行多少次循环
                {
                    cur = cur * 10 + k;//后面的值都先取一样值
                }

                if (cur <= N)//如果得出的值小于N
                {
                    ret = ret * 10 + k;//将值加入ret
                    pre = k;//更新pre
                    break;
                }
            }
        }
        return ret;
    }
};
```

739. Daily Temperatures

```
/* LeetCode Weekly Contest 61 第一题
 * problem 739. Daily Temperatures

题意：
给一个数组t，求下一个比t[i]大的数距离i多少位，如果不存在这为0，输出一个距离数组

Note: The length of temperatures will be in the range [1, 30000]. Each temperature will be an integer in the range [30, 100].

分析：
可以从后向前记录更新一个best数组来解

*/
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        int n = temperatures.size();
        vector<int> ret(n);//记录结果的数组
        vector<int> best(101, -1);//记录已经检测的数值的位置
        for (int i = n - 1; i >= 0; --i)//从后往前检测，这样相同数值的数的位置在best数组会不断更新
        {
            ret[i] = 1 << 30;
            for (int j = temperatures[i] + 1; j <= 100; ++j)//检测所有比在数组i位的值大的值
            {
                if (best[j] >= 0)//如果此值存在于已检测的数中
                {
                    ret[i] = min(ret[i], best[j] - i);//则减去i取得相对距离，并取距离最短的
                }
            }
            if (ret[i] == 1 << 30) ret[i] = 0;//如果不存在以检测的数比i的值大，则为0
            best[temperatures[i]] = i;//更新best数组
        }
        return ret;
    }
};
```

740. Delete and Earn

```
/* LeetCode Weekly Contest 61
 * problem 740. Delete and Earn

题意：
给一个数组，如果我们选取一个数num[i]则我们将删除这个数与每一个值为num[i] + 1 或 num[i] - 1 的数并得到num[i]积分
求怎么获得最大积分

Example 1:
Input: nums = [3, 4, 2]
Output: 6

Explanation:
Delete 4 to earn 4 points, consequently 3 is also deleted.
Then, delete 2 to earn 2 points. 6 total points are earned.

Example 2:
Input: nums = [2, 2, 3, 3, 3, 4]
Output: 9

Explanation:
Delete 3 to earn 3 points, deleting both 2's and the 4.
Then, delete 3 again to earn 3 points, and 3 again to earn 3 points.
9 total points are earned.

Note:
The length of nums is at most 20000.
Each element nums[i] is an integer in the range [1, 10000].

分析：
我们可以很容易的用dp求解
对于一个数值i，我们要么不取，那么我们会取值i-1，这样我们会删除数值i，则：dp[i] = dp[i - 1];
如果取数值i，则我们不取i-1：cur = i * cnt[i] + (i - 2 >= 0 ? dp[i - 2] : 0);

*/

class Solution {
public:
    int deleteAndEarn(vector<int>& nums) {
        const int n = 10001;
        vector<int> cnt(n);
        for (auto& x : nums)//记录在每个值的数的数目
        {
            cnt[x]++;
        }
        vector<int> dp(n);
        dp[0] = 0;
        for (int i = 1; i < n; ++i)//遍历每个值，dp求解
        {
            dp[i] = dp[i - 1];//取值i-1情况
            int cur = i * cnt[i] + (i - 2 >= 0 ? dp[i - 2] : 0);//取值i情况
            dp[i] = max(dp[i], cur);
        }
        return dp[n - 1];
    }
};
```