## 1154. 一年中的第几天（简单）

1. 题目描述

   给你一个按 `YYYY-MM-DD` 格式表示日期的字符串 `date`，请你计算并返回该日期是当年的第几天。

   通常情况下，我们认为 1 月 1 日是每年的第 1 天，1 月 2 日是每年的第 2 天，依此类推。每个月的天数与现行公元纪年法（格里高利历）一致。

   **示例 1：**

   ```
   输入：date = "2019-01-09"
   输出：9
   ```

   **示例 2：**

   ```
   输入：date = "2019-02-10"
   输出：41
   ```

   **示例 3：**

   ```
   输入：date = "2003-03-01"
   输出：60
   ```

   **示例 4：**

   ```
   输入：date = "2004-03-01"
   输出：61
   ```

   **提示：**

   - `date.length == 10`
   - `date[4] == date[7] == '-'`，其他的 `date[i]` 都是数字。
   - `date` 表示的范围从 1900 年 1 月 1 日至 2019 年 12 月 31 日。

2. 比赛实现

```
class Solution {
public:
    //去掉2月的前i月天数和
    vector<int> days_in_months = {0, 31, 31, 62, 92, 123, 153, 184, 215, 245, 276,
306, 337};
    bool isRunYear(int n){//判断闰年
        if((n%4 == 0 && n%100 != 0) || n %400 == 0)
            return true;
        else
            return false;
    }
    int dayOfYear(string date) {
```

```
        int month = (date[5]-'0')*10 + date[6]-'0';
        int day = (date[8]-'0')*10 + date[9]-'0';
        int ans = days_in_months[month-1] + day;
        if(month > 2){
            int year = stoi(date.substr(0, 4));
            if(isRunYear(year))
                ans += 29;
            else
                ans += 28;
        }
        return ans;
    }
};
```

## 1155. 掷色子的N种方法（中等）

1. 题目描述

这里有 `d` 个一样的骰子，每个骰子上都有 `f` 个面，分别标号为 `1, 2, ..., f`。

我们约定：掷骰子的得到总点数为各骰子面朝上的数字的总和。

如果需要掷出的总点数为 `target`，请你计算出有多少种不同的组合情况（所有的组合情况总共有 `f^d` 种），**模 10^9 + 7** 后返回。

**示例 1：**

```
输入：d = 1, f = 6, target = 3
输出：1
```

**示例 2：**

```
输入：d = 2, f = 6, target = 7
输出：6
```

**示例 3：**

```
输入：d = 2, f = 5, target = 10
输出：1
```

**示例 4：**

```
输入：d = 1, f = 2, target = 3
输出：0
```

**示例 5：**

```
输入：d = 30, f = 30, target = 500
输出：222616187
```

**提示：**

- `1 <= d, f <= 30`
- `1 <= target <= 1000`

2. 比赛实现

本题说半天组合，其实还是个排列问题（所有的骰子看作不一样的），一开始想用DFS，结果总是超时，后来发现可以用动态规划，所以啊，所有方法都想想试试没准就有答案了WTF

此外，本题又是一个类似背包的问题，需要多注意啊！！！

`dp[i][j]`表示用第0-i个筛子投掷出j的所有的情况，则转移方程为`dp[i][j] = dp[i-1][j-1]+...+dp[i-1][j-f]`，即第i个骰子分别掷出1 ~ f时，需要前i-1个骰子共分别掷出j-1 ~ j-f

**此外，要注意细节，不要越界**

```
class Solution {
public:
    int MOD = 1e9 + 7;
    int numRollsToTarget(int d, int f, int target) {
        int bottom = d;//可以掷出的最小值
        int top = d*f;//可以掷出的最大值
        if(target < bottom || target > top)
            return 0;
        if(d == 1 || target == bottom || target == top)
            return 1;
        vector<vector<int>> dp(d, vector<int>(target+1, 0));
        for(int i = 1; i <= min(f, target); i++)//注意取min
            dp[0][i] = 1;
        for(int i = 1; i < d; i++){
            for(int j = i+1; j <= min(target, (i+1)*f); j++)//注意取min
                for(int k = 1; k <= min(f, j); k++)//注意取min
                    dp[i][j] = (dp[i][j] + dp[i-1][j-k]) % MOD;
        return dp[d-1][target];
    }
};
```

## 1156. 单字符重复子串的最大长度（中等）

1. 题目描述

如果字符串中的所有字符都相同，那么这个字符串是单字符重复的字符串。

给你一个字符串 `text`，你只能交换其中两个字符一次或者什么都不做，然后得到一些单字符重复的子串。返回其中最长的子串的长度。

**示例 1：**

```
输入：text = "ababa"
输出：3
```

**示例 2：**

```
输入：text = "aaabaaa"
输出：6
```

**示例 3：**

```
输入：text = "aaabbaaa"
输出：4
```

**示例 4：**

```
输入：text = "aaaaa"
输出：5
```

**示例 5：**

```
输入：text = "abcdef"
输出：1
```

**提示：**

- `1 <= text.length <= 20000`
- `text` 仅由小写英文字母组成。

2. 简单实现

模拟的时候本来想用动态规划来着，规划半天浪费了很多时间，后来想到了本解法，但是没及时写完

```
class Solution {
public:
    int maxRepOpt1(string text) {
        int len = text.size();
        vector<int> cnt(26, 0);//记录各个字符在text中出现的次数
        for(int i = 0; i < len; i++)
            cnt[text[i]-'a']++;
        //从左向右挨个字符遍历
        char cur_c = text[0];//当前单字符
        int cur_l = 1;//当前单字符串长度
        int idx = 1;//当前遍历到的字符索引
        int ans = -1;//答案
        while(idx < len){
            while(idx < len && text[idx] == cur_c){
                idx++;
                cur_l++;
            }
            if(cur_l < cnt[cur_c-'a']){//可以把某个位置的cur_c交换到idx位置，使单字符串得以延长
                cur_l++;
                int tmp = idx + 1;//继续向后延长单字符串
                while(tmp < len && text[tmp] == cur_c){
                    cur_l++;
                    tmp++;
                }
```

```
                }
                //当aaaba这种情况时，前"交换"的a会在继续延长时再次计数，所以要取min，达到去重的作用
                cur_l = min(cur_l, cnt[cur_c-'a']);
                ans = max(ans, cur_l);
                if(idx < len){//继续记录下一段单字符串
                    cur_c = text[idx];
                    cur_l = 1;
                    idx++;
                }
            }
            return ans;
        }
    };
```

## 1157. 子数组中占绝大多数的元素（困难）

1. 题目描述

   实现一个 `MajorityChecker` 的类，它应该具有下述几个 API：

   - `MajorityChecker(int[] arr)` 会用给定的数组 `arr` 来构造一个 `MajorityChecker` 的实例。
   - ```
     int query(int left, int right, int threshold)
     ```

     有这么几个参数：

     - `0 <= left <= right < arr.length` 表示数组 `arr` 的子数组的长度。
     - `2 * threshold > right - left + 1`，也就是说阈值 `threshold` 始终比子序列长度的一半还要大。

   每次查询 `query(...)` 会返回在 `arr[left], arr[left+1], ..., arr[right]` 中至少出现阈值次数 `threshold` 的元素，如果不存在这样的元素，就返回 `-1`。

   **示例：**

   ```
   MajorityChecker majorityChecker = new MajorityChecker([1,1,2,2,1,1]);
   majorityChecker.query(0,5,4); // 返回 1
   majorityChecker.query(0,3,3); // 返回 -1
   majorityChecker.query(2,3,2); // 返回 2
   ```

   **提示：**

   - `1 <= arr.length <= 20000`
   - `1 <= arr[i] <= 20000`
   - 对于每次查询，`0 <= left <= right < len(arr)`
   - 对于每次查询，`2 * threshold > right - left + 1`
   - 查询次数最多为 `10000`

2. 解法一——分块
   ```

针对不同的询问区间长度，使用两种不同的方法。取一个分界值 $s$。

如果区间长度 $\leq s$，直接暴力即可。时间复杂度为 $O(s)$。

如果区间长度 $> s$，则绝对众数出现次数 $> \frac{s}{2}$，因此可能的答案个数 $\leq \frac{2n}{s}$（出现次数 $> \frac{s}{2}$ 的不同数字个数 $\leq \frac{2n}{s}$）。

统计每个前缀内这些数各自出现了多少次。询问时枚举每一个可能的数，对两个前缀和求差即可得到这个数在区间内的出现次数。时间复杂度为 $O(\frac{2n}{s})$。

取 $s = \sqrt{2n}$（考虑到常数，实践中不一定最优），两种方法时间复杂度均为 $O(\sqrt{2n}) = O(\sqrt{n})$。因此总时间复杂度为 $O((n+q)\sqrt{n})$。可以通过本题。

题解中所谓的可能的答案是指，所有可能作为区间长度>s的查询的返回值的个数，这个个数一定小于等于 `n/(s/2)`

不管是暴力算法还是前缀法我都想到了，但是只用单一的任何一个解法都不能通过，反而是这种分块的解法将两者结合起来可以，也是很神奇的

```
class MajorityChecker {
    int n,N,s,a[20005],b[205][20005],d[205];
    map<int,int> m;
public:
    MajorityChecker(vector<int>& arr) {
        n=arr.size();
        N=0;
        for(int i=0;i<n;i++)
            m[ a[i]=arr[i] ]++;
        s=sqrt(n*2);
        for(auto i:m)
            if(i.second > s>>1){//可能作为区间长度>s的查询的答案
                b[++N][0]=0;
                d[N]=i.first;//记录第N个可能值
                for(int j=0;j<n;j++)
                    b[N][j+1] = b[N][j]+(a[j]==d[N]);//第N个可能值在前j个数中出现的次数
            }
    }

    int query(int left, int right, int threshold) {
        int i,j,k;
        if(right-left<=s){//暴力解法
            j=k=0;
            //假设存在绝对众数，找到它
            for(i=left;i<=right;i++)
                if(a[i]==j)k++;
                else if(k)k--;
                else {
                    j=a[i];
                    k=1;
                }
            //检测是不是绝对众数
            for(i=left,k=0;i<=right;i++)
                if(a[i]==j) k++;
            if(k<threshold)j=-1;
```

```
                return j;
            }
            for(i=1;i<=N;i++)
                if(b[i][right+1]-b[i][left] >= threshold)
                    return d[i];
            return -1;
        }
    };
```

3. 解法二——二分查找

# 算法描述

因为元素值范围为 1 - 20000，所以我们可以用20000个vector来存储每个值出现的位置。在query的时候，对每个元素依次使用二分查找，找到区间内的元素个数，与阈值比较。

# 复杂度分析

令数组长度为n：
查找时间复杂度：O（nlogn）
空间复杂度：O（n）

```
class MajorityChecker {
public:
    MajorityChecker(vector<int>& arr) {
        for (int i = 0; i < arr.size(); i++)
            pos[arr[i]].push_back(i);
    }
    int query(int left, int right, int threshold) {
        for (int i = 0; i < 20001; i++) {
            if (pos[i].size() < threshold) continue;
            auto u = upper_bound(pos[i].begin(), pos[i].end(), right);
            auto l = lower_bound(pos[i].begin(), pos[i].end(), left);
            int num = u - l;
            if (num >= threshold)
                return i;
        }
        return -1;
    }
private:
    vector<int> pos[20001];
};
```

4. 解法三——线段树

# 算法描述

解法2是对解法1的优化，不需要遍历所有的值，用线段树二分查找找到区间内最多的元素。
要使用线段树，需要元线段可累加，对这道题目，可以使用摩尔投票法对元线段进行累加。
令元素的值为val，元素的个数为count，
（val1, count1） + （val2, count2） =
（val1, count1+count2） cond val1 == val2
（val1, count1-count2） cond val1 != val2 && count1 > count2
（val2, count2-count1） cond val1 != val2 && count2 > count1
这样建立完树后，就可以使用线段树的查找操作，直接找到区间内的众数的值。

# 复杂度分析

令数组长度为n：
查找时间复杂度：O（logn）
空间复杂度：O（4 * n）

```
class MajorityChecker {
public:
    struct TreeNode {
        int val;
        int count;
        TreeNode operator+ (const TreeNode& other) const {
            TreeNode n;
            if (val == other.val) {
                n.val = val;
                n.count = count + other.count;
                return n;
            } else if (count >= other.count) {
                n.val = val;
                n.count = count - other.count;
            } else {
                n.val = other.val;
                n.count = other.count - count;
            }
            return n;
        }
    };

    MajorityChecker(vector<int>& arr) {
        size = arr.size();

        for (int i = 0; i < size; i++)
            pos[arr[i]].push_back(i);

        for (int i = 0; i < 60004; i++) {
```

```
            tree[i].val = 0;
            tree[i].count = -1;
        }

        buildTree(1, 0, size-1, arr);
    }

    void buildTree(int pos, int left, int right, vector<int> &arr) {
        if (left == right) {
            tree[pos].val = arr[left];
            tree[pos].count = 1;
            return;
        }

        int mid = left + (right - left) / 2;
        buildTree(pos*2, left, mid, arr);
        buildTree(pos*2+1, mid+1, right, arr);
        tree[pos] = tree[pos*2] + tree[pos*2+1];
    }

    TreeNode findTree(int pos, int left, int right, int fleft, int fright)
    {
        if (left == fleft && right == fright)
            return tree[pos];

        int mid = left + (right - left) / 2;

        if (fright <= mid)
            return findTree(pos*2, left, mid, fleft, fright);

        if (fleft > mid)
            return findTree(pos*2+1, mid+1, right, fleft, fright);
        return findTree(pos*2, left, mid, fleft, mid) +  findTree(pos*2+1, mid+1,
right, mid+1, fright);
    }

    int query(int left, int right, int threshold) {
        TreeNode node = findTree(1, 0, size-1, left, right);
        int val = node.val;

        auto u = upper_bound(pos[val].begin(), pos[val].end(), right);
        auto l = lower_bound(pos[val].begin(), pos[val].end(), left);
        int num = u - l;
        if (num >= threshold)
            return val;

        return -1;
    }

private:
    int size;
    vector<int> pos[20001];
    TreeNode tree[65536];
```

```
};
```