

int ceil(int val){

        if(val%3==0){

            return val/3;

        }

        return val/3+1;

    }

    long long maxKelements(vector<int>& nums, int k) {

        //类型 贪心 局部最优 每次都取当前可选的最大的数

        // 优先队列 的运用 每次队头都是当前所有元素中最大的原始

        priority\_queue<int> Q(nums.begin(),nums.end());

        long long sum=0;

       while(k){

           int t=Q.top();

           Q.pop();

           sum+=t;

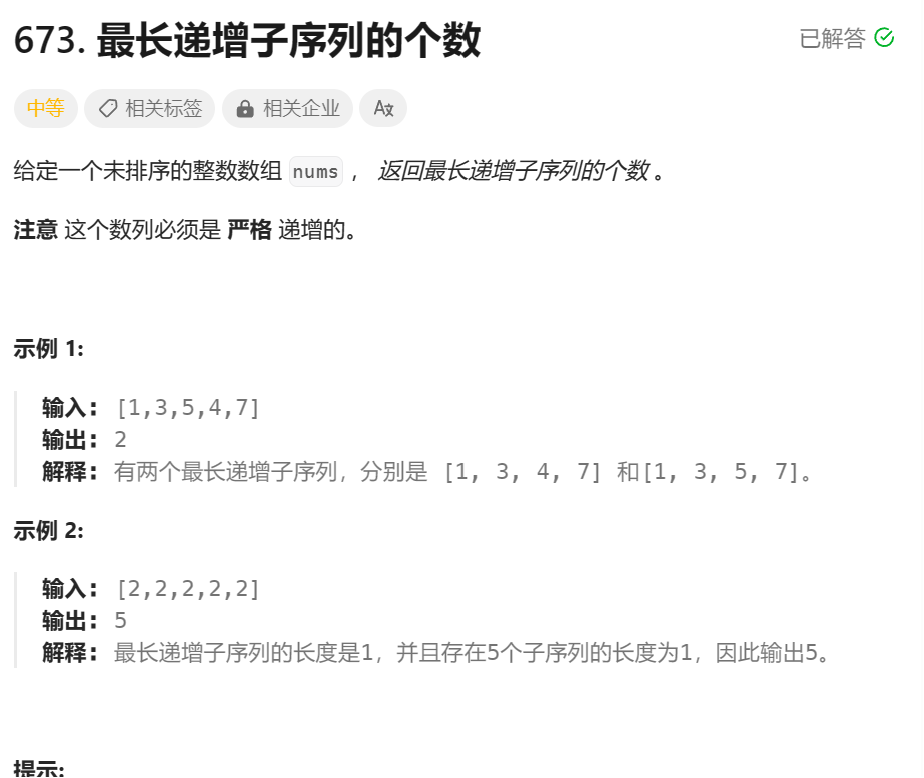
           Q.push(ceil(t));

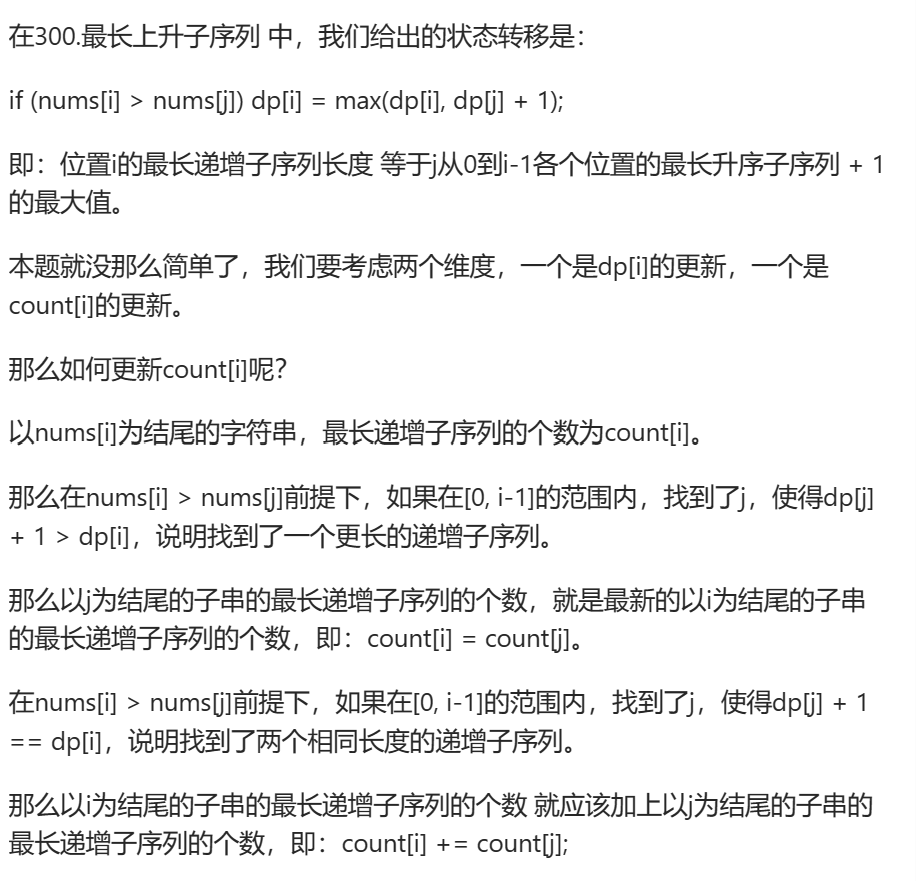
           k--;

       }

        return sum;

}





双dp : 1.和字符串有关

1. 在经典 类型的dp字符串问题上又加一层 问题

3.多个维度先确定一个维度先，再推导另外的维度

int findNumberOfLIS(vector<int>& nums) {

      //双维度dp

      int n=nums.size();

      vector<int> dp(n,1);

      vector<int> count(n,1);//以nums[i]为结尾的字符串，最长递增子序列的个数为count[i]

      int max\_s=1;

      int ans=0;

      for(int i=1;i<nums.size();i++){

          for(int j=0;j<i;j++){

              if(nums[j]<nums[i]){

                  if(dp[j]+1>dp[i]){

                      count[i]=count[j];

                  }

                  else if(dp[j]+1==dp[i]){//当前dp[i] 存储最长递增子序列长度不包含以其前面 是nums[j]的情况 因为其前一个字符在 是从 nums[0]\_nums[j]情况

                  count[i]+=count[j];

                  }

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

              }

              max\_s=max(dp[i],max\_s);

          }

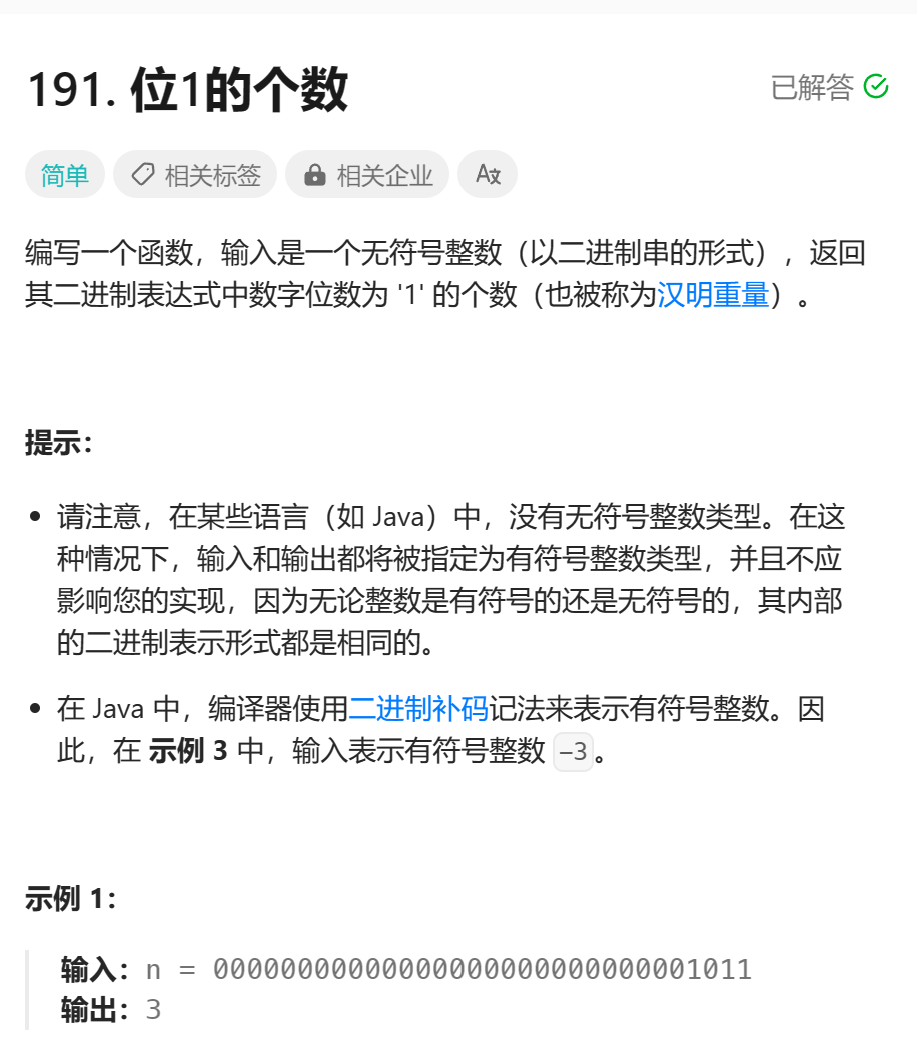
      }

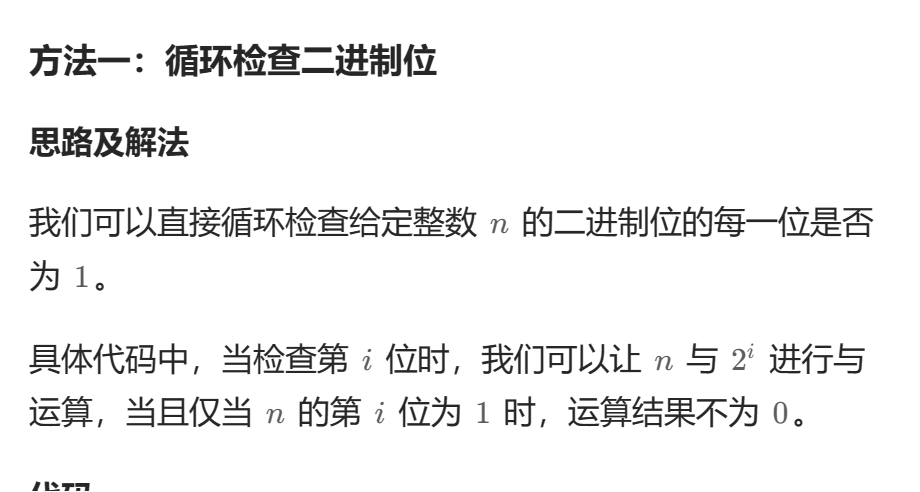
      for(int i=0;i<nums.size();i++){

          if(dp[i]==max\_s) ans+=count[i];

      }

      return ans;





class Solution {

public:

    int hammingWeight(uint32\_t n) {

        int ret = 0;

        for (int i = 0; i < 32; i++) {

            if (n & (1 << i)) {// 检查每一位是否位1

                ret++;

            }

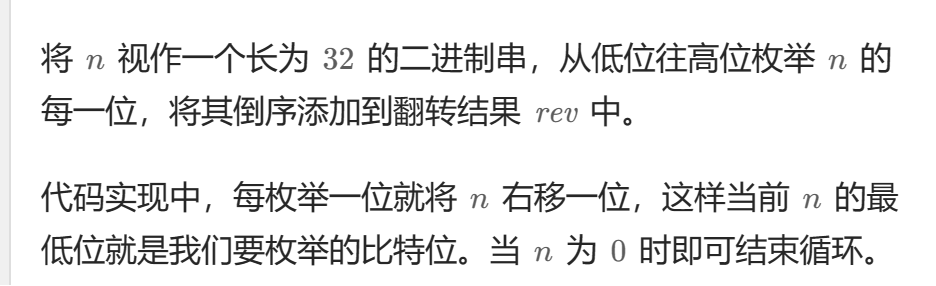
        }

        return ret;

    }

};





uint32\_t reverseBits(uint32\_t n) {

          uint32\_t ans=0;

          for(int i=0;i<32&&n>0;i++){//利用位运算来移位

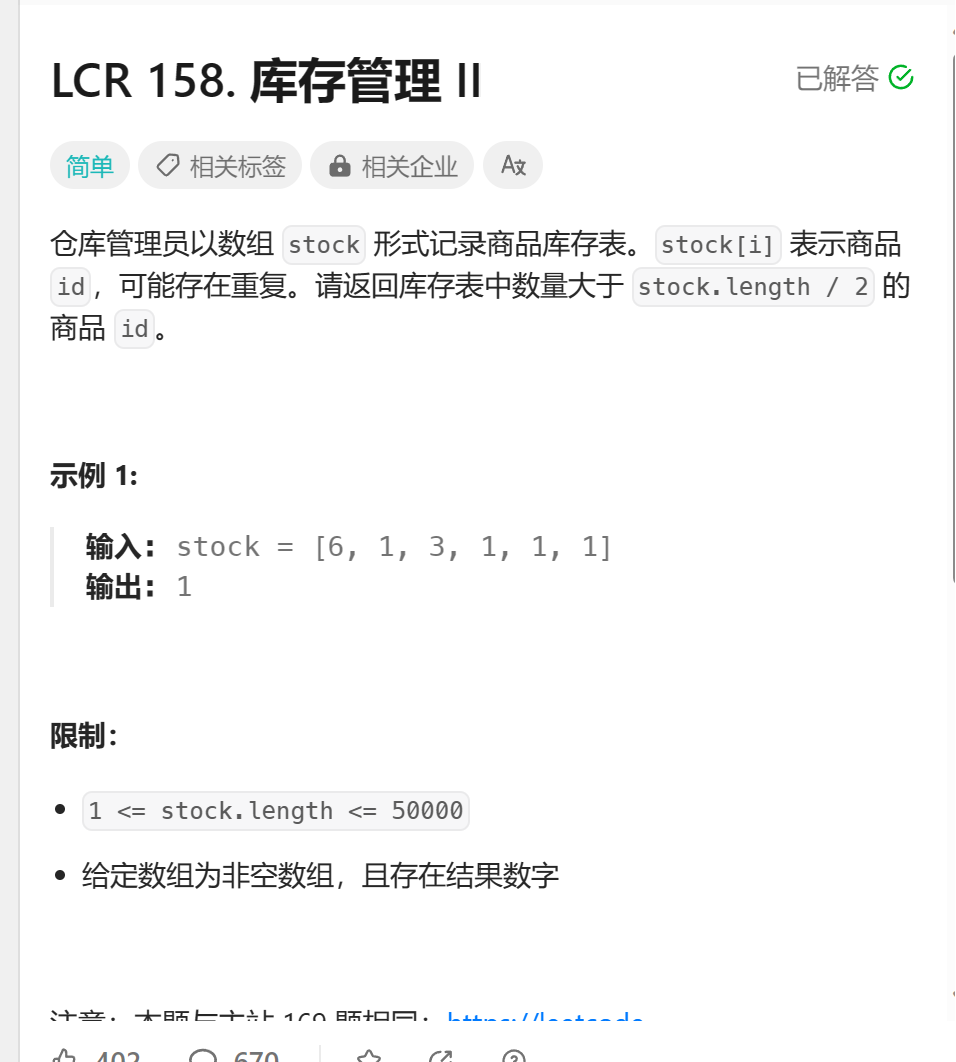
              ans|=(n&1)<<(31-i);//从低位往高位枚举 nnn 的每一位，将其倒序添加到翻转结果 rev中。

              n>>=1;//代码实现中，每枚举一位就将 n 右移一位，这样当前 n 的最低位就是我们要枚举的比特位。当 n 为 000 时即可结束循环。

          }

          return ans;

    }



拓展知识点:摩尔投票法



 int inventoryManagement(vector<int>& stock) {

        if(stock.size()<=2){

            return stock[0];

        }

        int num=stock[0];

        int vote=1;

        //因为众数的票数>n/2 所以使用摩尔投票法  最后 众数的票肯定>=1;

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

            if(vote==0){//重新选人

              num=stock[i];

              vote=1;//给新选择的人 开始有一票

            }

            else{

                if(num!=stock[i]){//新的一票和原本的选出来的人不相同，原来的人的票数-1

                 vote--;

                }

                else{

                    vote++;

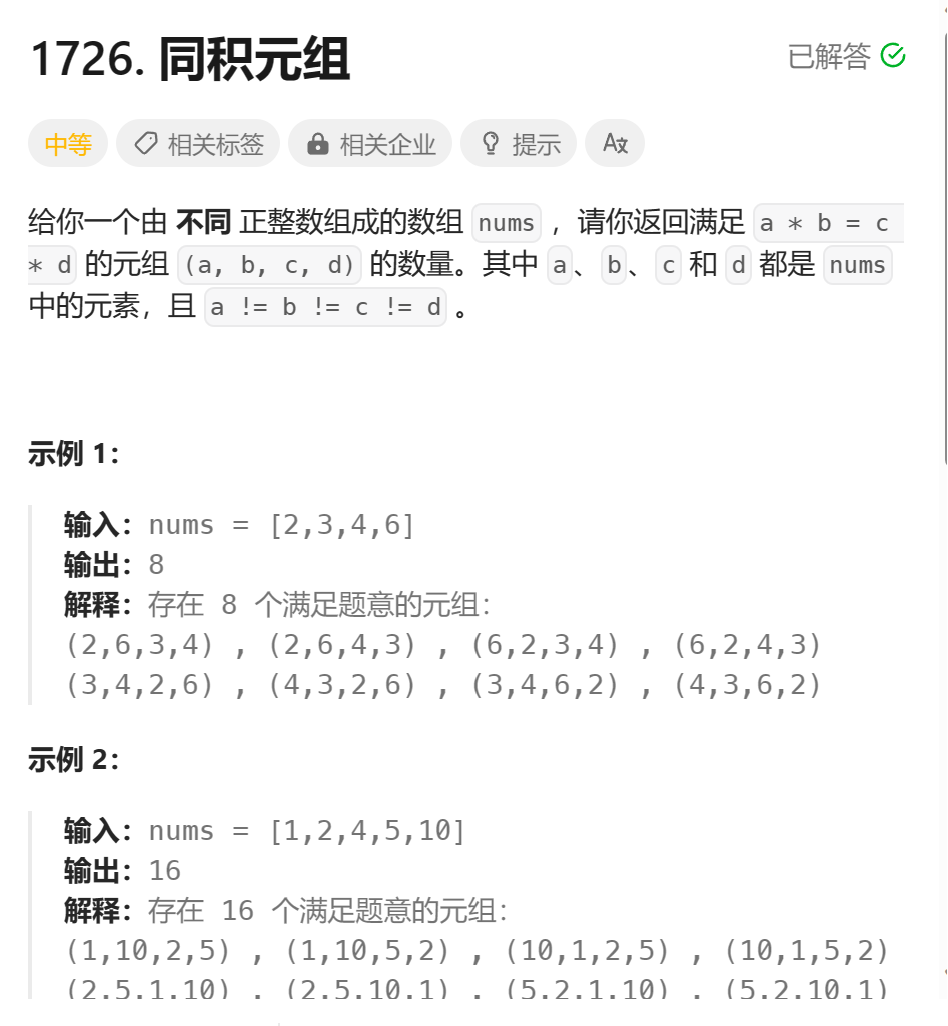
                }

            }

        }

  return num;

    }



具有枚举性质的算法：1.回溯 2.dp 3.hash 枚举

 int tupleSameProduct(vector<int>& nums) {

        //当需要枚举的时候->先思考回溯算法

        //当回溯算法无论怎么样都回溯不出来的时候

        //不要犹豫马上换思路 向着具有枚举性质的算法思考

        //具有映射关系的枚举-->使用哈希表

       unordered\_map<int,int> hash;

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

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

               hash[nums[i]\*nums[j]]++;//hash表创建时默认val 值为0;

           }

       }

       int sum=0;

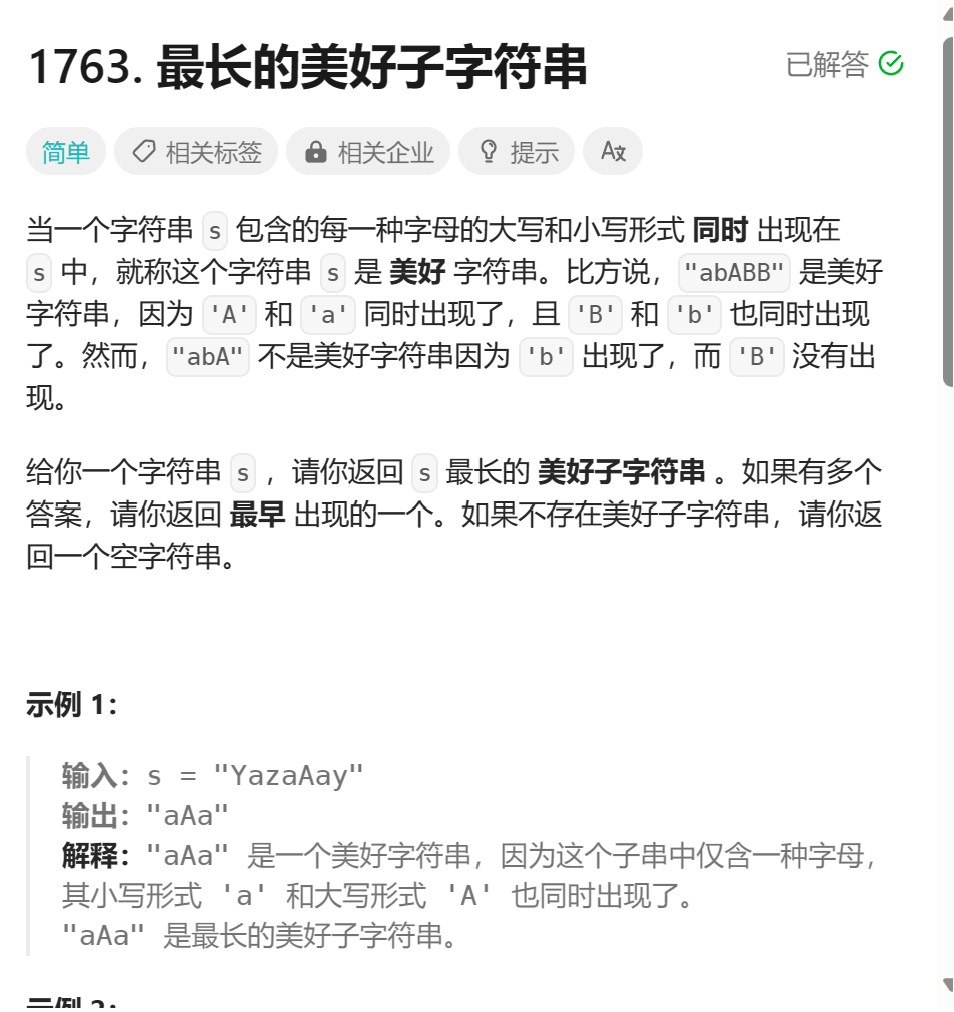
       for(auto &[ first,second]:hash){

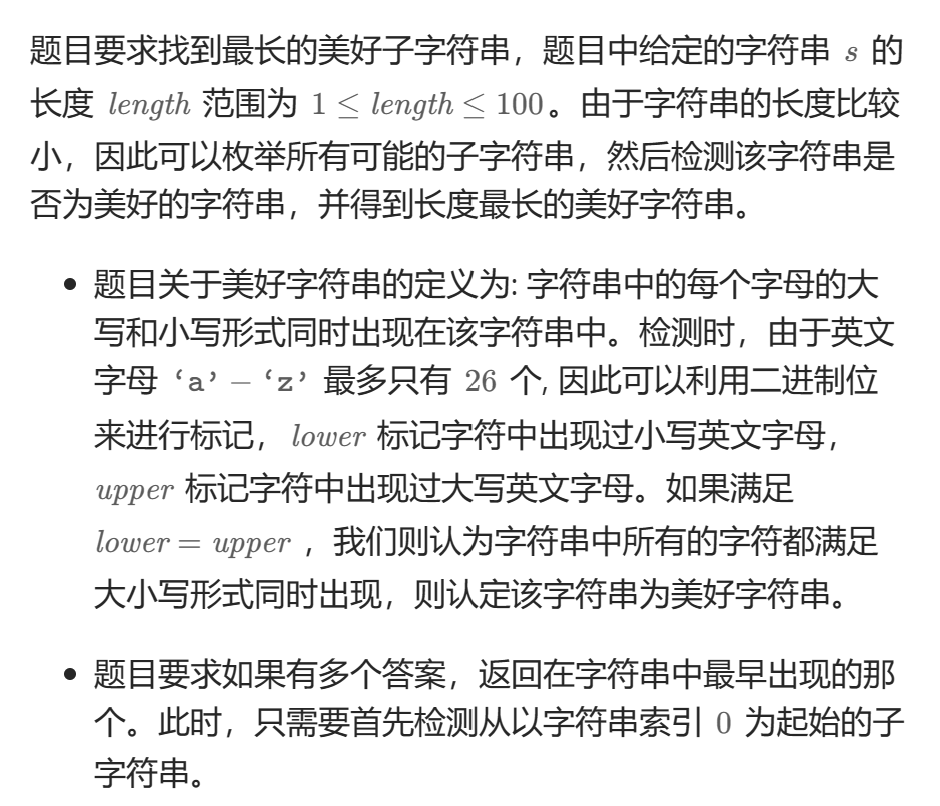
           sum+=second\*(second-1)\*4;

       }

       return sum;

    }





string longestNiceSubstring(string s) {

    //因为字母只有26个 所以可以巧妙的通过位运算来 记录出先过的 字母

    int maxpos=0;

    int maxlen=0;

    for(int i=0;i<s.size();i++){

        int lower=0;

        int height=0;

        for(int j=i;j<s.size();j++){

             if(islower(s[j])){

                 lower|= 1<<(s[j]-'a');//巧妙的通过位运算来 记录出先过的 小写字母

             }

             else{

                 height|=1<<(s[j]-'A');//巧妙的通过位运算来 记录出先过的 大写字母

             }

             if(lower==height&&j-i+1>maxlen){

                 maxpos=i;

                 maxlen=j-i+1;

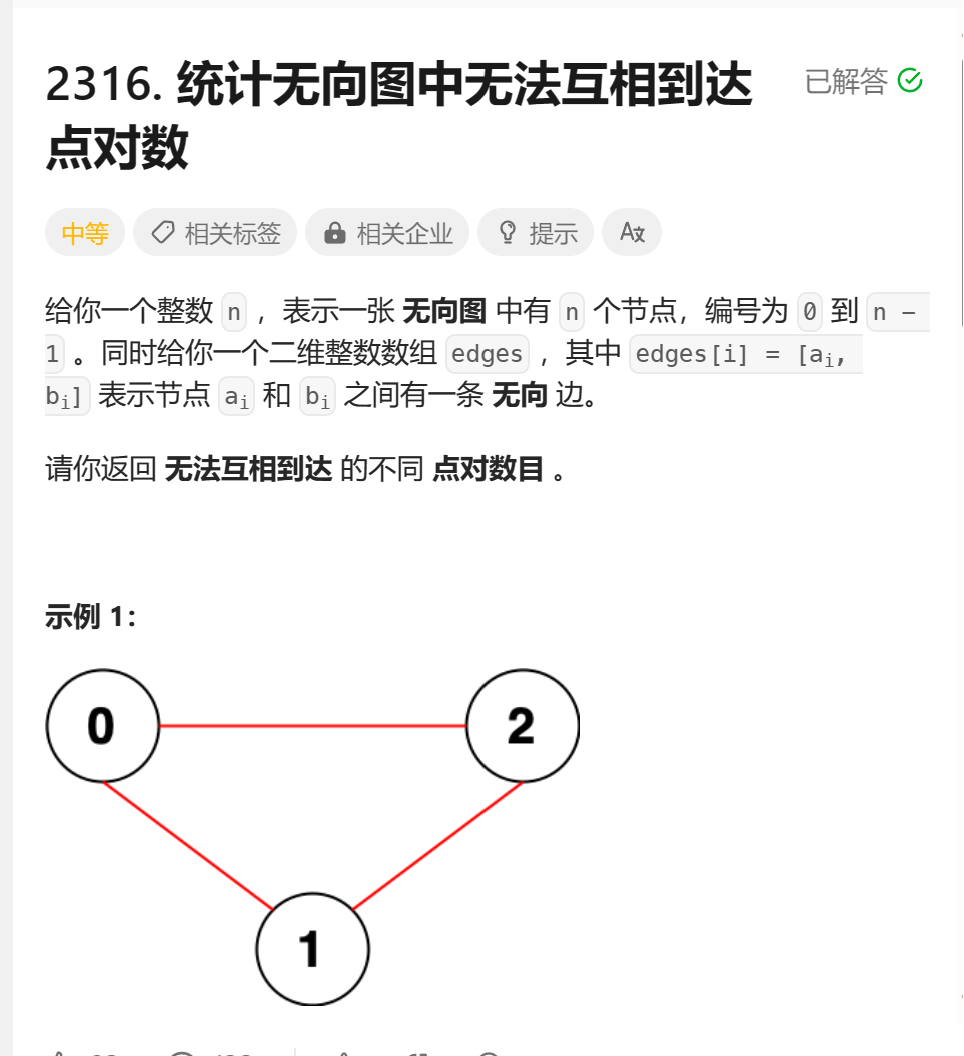
             }

        }

    }

    return s.substr(maxpos,maxlen);

}



void dfs( vector<vector<int>> &near\_grap,vector<bool>&visit,long long&count,int start){

        for(int i=0;i<near\_grap[start].size();i++){

              if(visit[near\_grap[start][i]]==true){

                  continue;

              }

              visit[near\_grap[start][i]]=true;

              count++;

              dfs(near\_grap,visit,count,near\_grap[start][i]);

          }

          return ;

    }

    long long countPairs(int n, vector<vector<int>>& edges) {

        //dfs 和 bfs 需要邻接接表或者 邻接矩阵辅助

        vector<vector<int>> near\_grap(n);//创建邻接表

        for(auto i:edges){

            near\_grap[i[0]].push\_back(i[1]);

            near\_grap[i[1]].push\_back(i[0]);

        }

         vector<bool>visit(n,false);

         long long ans=0, s=0;

         for(int i=0;i<n;i++){

             long long count=0;//用来记录 同一个联通分量中节点的个数 代码模板

      if(visit[i]!=true){

          visit[i]=true;

            count++;

           dfs(near\_grap,visit,count,i);

        }

             ans+=s\*count;

             s+=count;

         }

         return ans;

    }

ti

不看题解按照 dp五部曲独立完成

   int countVowelStrings(int n) {

      vector<char> help={'a','e','i','o','u'};

      if(n==1) return 5;

      vector<vector<int>> dp1(n+1,vector<int>(5,0));//先思考 dp数组的含义

      vector<int> dp2(n+1,0);

      for(int i=0;i<5;i++){

          dp1[1][i]=1;//初始化

          dp2[1]+=dp1[1][i];

      }

      for(int i=2;i<n+1;i++){

          for(int j=0;j<5;j++){

              for(int k=0;k<=j;k++){

                  dp1[i][j]+=dp1[i-1][j-k]; //思考递推公式

              }

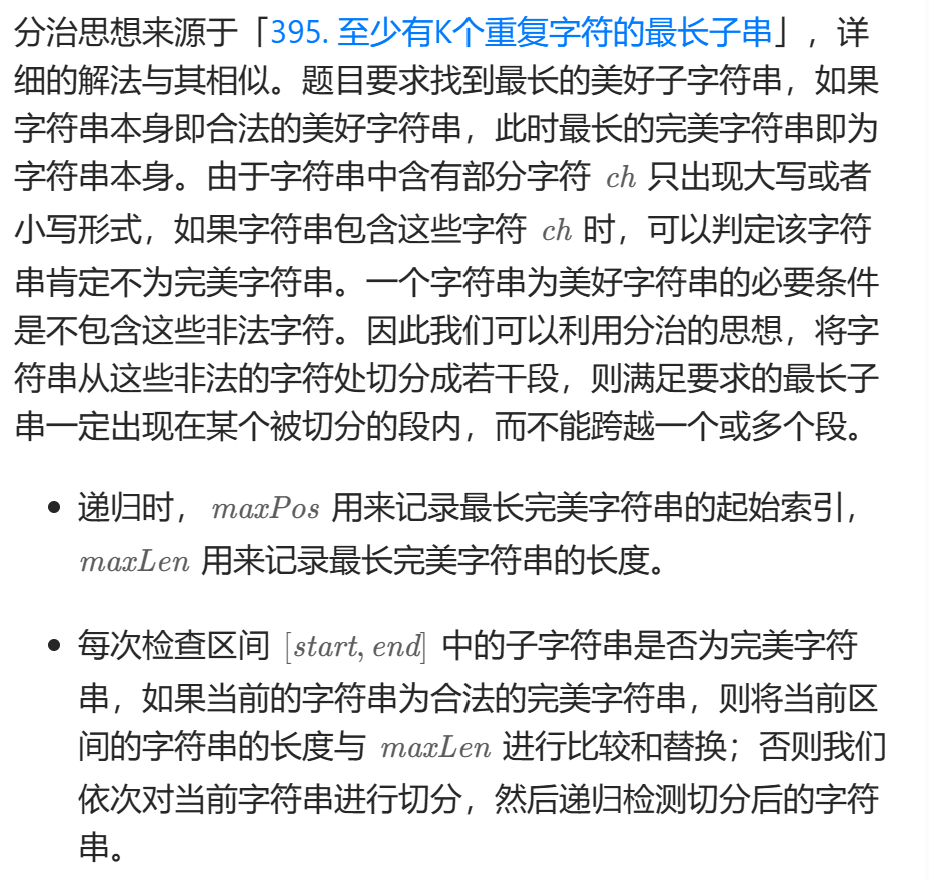
              dp2[i]+=dp1[i][j];

          }

      }

      return dp2[n];

}



 int maxpos=0;

     int maxlen=0;

     void dfs(string s,int start,int end){//写递归函数时要先考虑返回的特殊情况

     if(start>end){

         return;

     }

     int lower=0;

     int height=0;

     for(int i=start;i<=end;i++){

         if(islower(s[i])){

             lower|=1<<(s[i]-'a');

         }

         else{

             height|=1<<(s[i]-'A');

         }

     }

     if(lower==height){

         if(end-start+1>maxlen){

             maxpos=start;

             maxlen=end-start+1;

         }

            return;

     }

     int vivid=lower&height;

     int pos=start;

     while(pos<=end){//分治操作 将 不合群的字符分割出去

         start=pos;//更新切割的起始点

         while(pos<=end&&vivid& (1<<(tolower(s[pos])-'a'))){//判断是否是不合群的字符位置

             pos++;

         }

         dfs(s,start,pos-1);

         pos++;

     }

     return ;

     }

    string longestNiceSubstring(string s) {

    dfs(s,0,s.size()-1);

    return s.substr(maxpos,maxlen);

    }



//用hash 来记忆 单词

    unordered\_map<string,int> hash;

    WordsFrequency(vector<string>& book) {

        for(auto i:book){

            hash[i]++;

        }

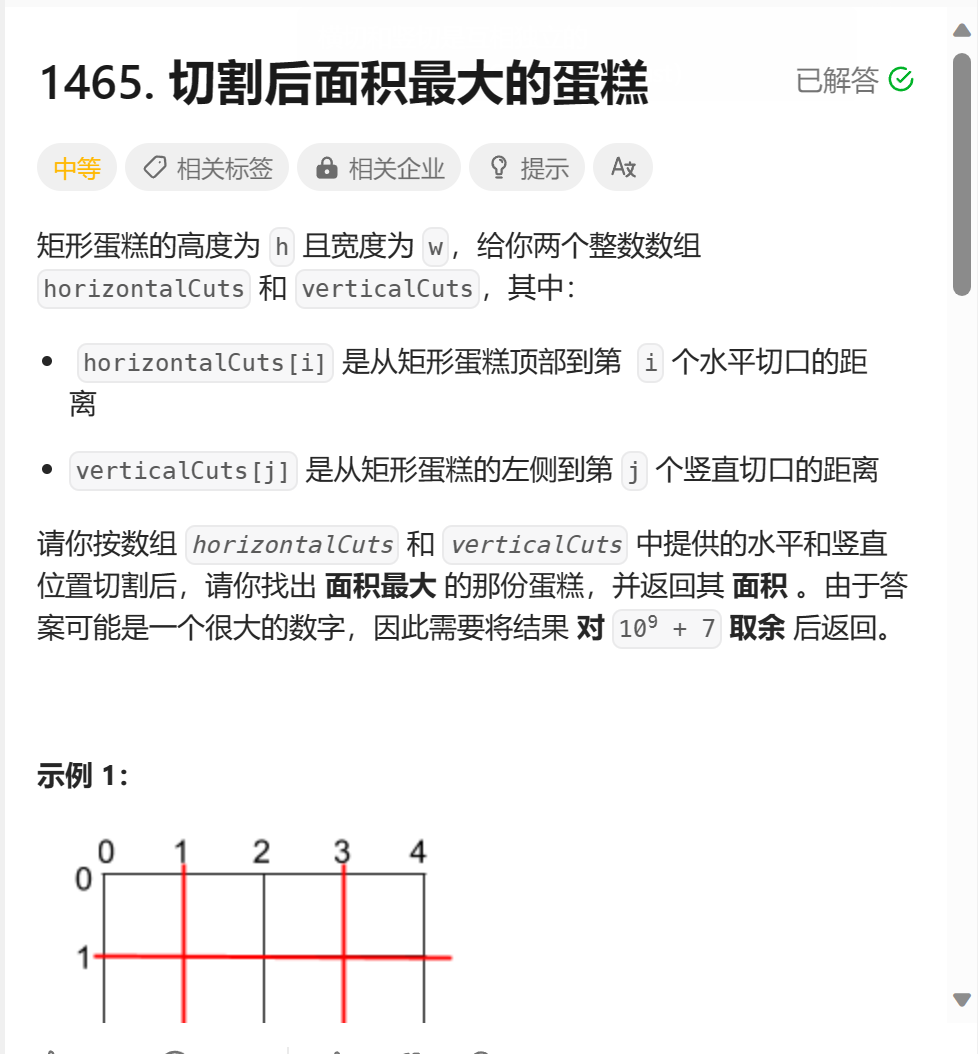
    }

    int get(string word) {

        return hash[word];

    }

};



先思考题目本质上问的是什么。

再思考是什么题型。

考虑答案由哪些变量得到。

如果涉及多个维度则先定下一个维度的答案，再思考另外维度的答案

 // s=x\*y 因为x与y相互独立 所以分别找x与y的最大·值

    long long get\_max(vector<int> &cuts,int limit){

        int n=cuts.size();

        int ans=max(cuts[0],limit-cuts.back());

        for(int i=1;i<n;i++){

            ans=max(ans,cuts[i]-cuts[i-1]);

        }

        return ans;

    }

    int maxArea(int h, int w, vector<int>& horizontalCuts, vector<int>& verticalCuts) {

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

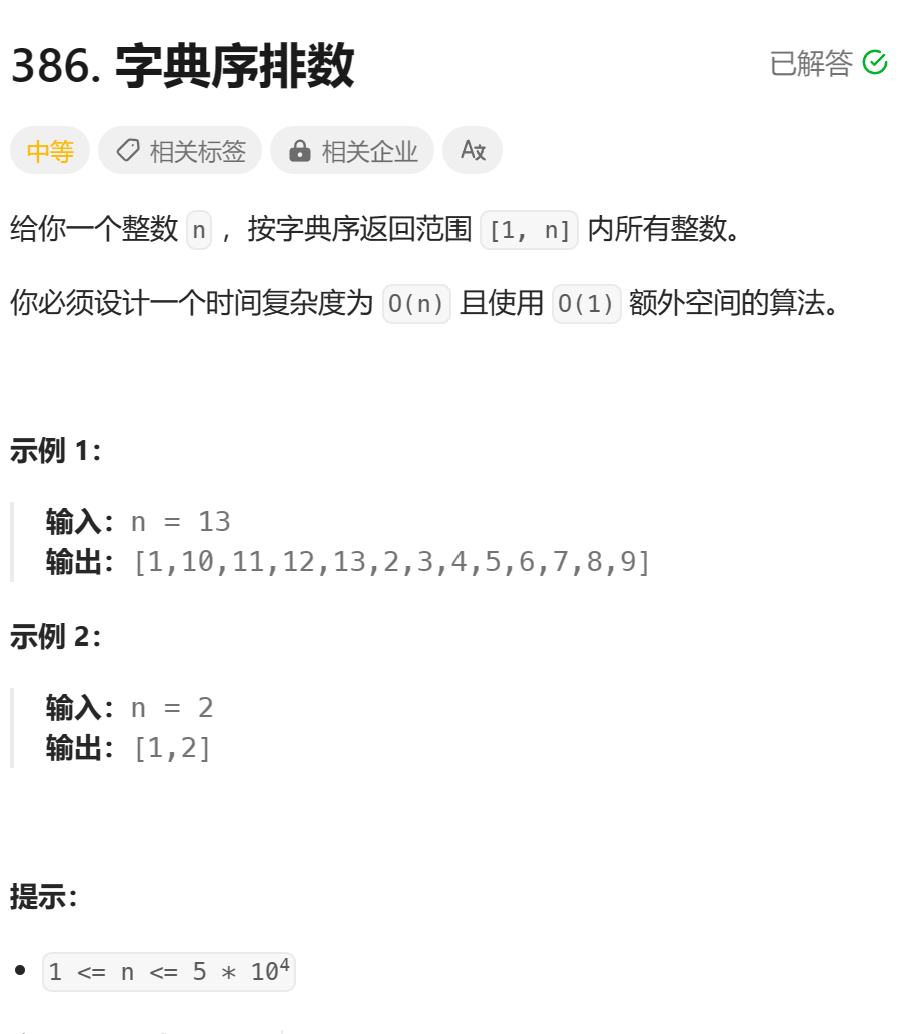
        sort(verticalCuts.begin(),verticalCuts.end());

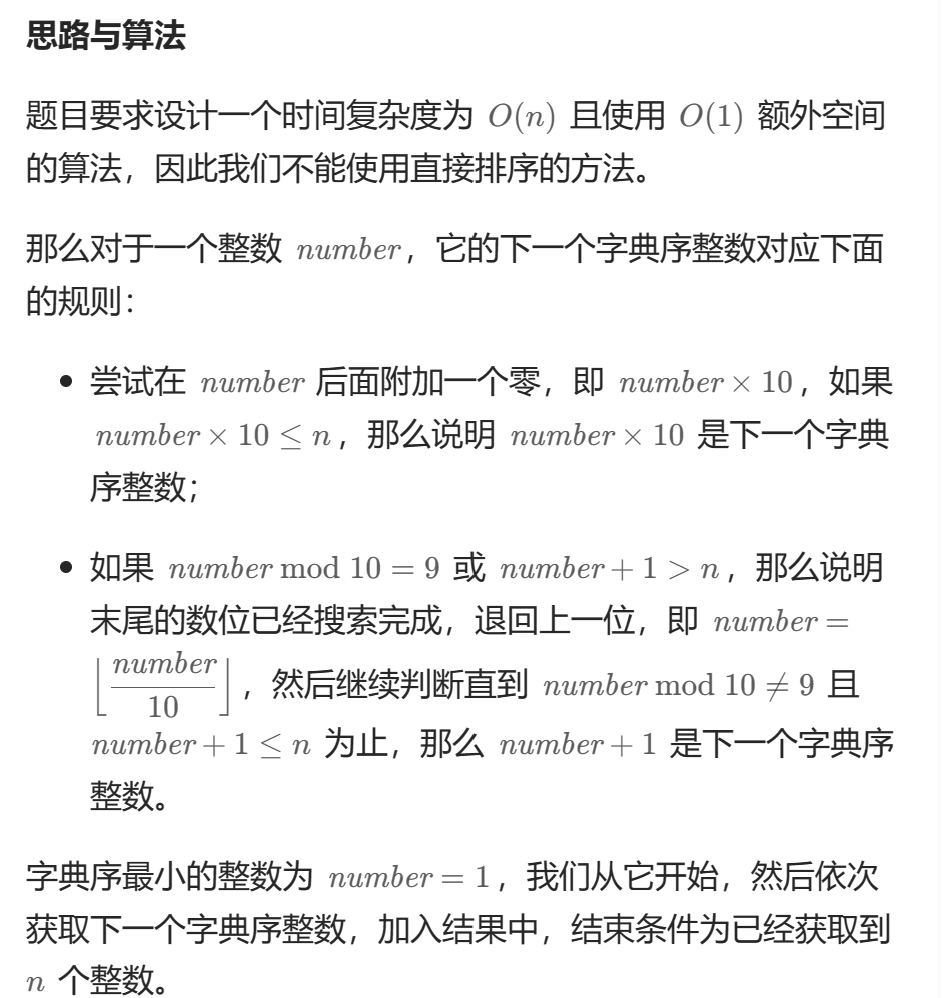
        long long x=get\_max(horizontalCuts,h);

        long long y=get\_max(verticalCuts,w);

        return x\*y%1000000007;

    }





 vector<int> lexicalOrder(int n) {

    //迭代法实现dfs 模板

    vector<int> ans(n);

    int num=1;

    for(int i=0;i<n;i++){

        ans[i]=num;

        if(num\*10<=n){

            num\*=10;

        }

        else{

            while(num%10==9||num+1>n){

                num/=10;

            }

            num++;

        }

    }

    return ans;

    }