力扣周赛总结

力扣第259场周赛：

- 力扣前三道题一般得拿下；一道简单，两道中等；
- 力扣最后一题用于学习；如果有思路有时间，则尝试去写，如果没有则竞赛之后去学习。

第259场周赛反思：

- 前两道题比较顺利的做了出来，但是第三道题却卡住了；
- 第三道题其实不难；一是审题出错，题目要求是正方形，但是我一开始把其当作长方形来处理了；然后就是代码出现了**bug**，本来是所有的都要判断的一个关系愣是被我写成了**if else**形式，还死活查不出来。

总结：

1. 增加代码书写的严谨性
2. 下次目标是前三题**AC**

力扣第260场周赛：

排名	用户名	得分	完成时间	题目1 (3)	题目2 (4)	题目3 (5)	题目4 (6)
797 / 3653	旭	7	0:53:06	0:08:49	0:48:06 1		

- 第一道题比较顺利的**AC**了，申请题意即可。
- 第二道题需要仔细的分析一下：第二道题用时**40**分钟，有点长了。

  第二道如果想明白了，也有一点贪心的意思；

  数据结构用前缀和；

  算法：第一个机器人只能选择一个位置进行一个下行，有n种可能的下行结果；第二个机器人为取得最大点，只能选择一直走第一条道或者一直走第二条道，所以其只能选择两者之中的最大值；利用前缀和可以在O（n)时间内解决这些问题。
- 第三题几乎是要做出来了，但是**left**和**right**的方向反了，最后死活不知道是为什么！！！

  此题其实也没有什么特殊的，就是模拟；数据结构上会用到双指针；遍历查找所有竖直方向和左右方向；检查符合**word.size()**大小的区间内能否装下单词即可。

```
class Solution {
```

```
public:
    bool placeWordInCrossword(vector<vector<char>>& board, string word)
{
        //模拟
        int size=word.size();
        int m=board.size(),n=board[0].size();
        for(int i=0;i<m;i++){
            //双指针
            int left=0,right=0;
            while(right<n&&left<n){
                while(left<n&&board[i][left]=='#')++left;
                right=left;
                while(right<n&&board[i][right]!='#')++right;
                if(right-left==size){

                    int flag=true,k=0;
                    for(int j=left;j<right;j++){
                        if(board[i][j]==word[k]||board[i][j]==' ')++k;
                        else{
                            flag=false;
                            break;
                        }
                    }
                    if(flag)return true;

                    flag=true,k=size-1;
                    for(int j=left;j<right;j++){
                        if(board[i][j]==' '||board[i][j]==word[k])--k;
                        else{
                            flag=false;
                            break;
                        }
                    }

                    if(flag)return true;
                }

                left=right;

            }
        }

        for(int j=0;j<n;j++){
            //双指针
            int left=0,right=0;
            while(left<m&&right<m){
                while(left<m&&board[left][j]=='#')++left;
                right=left;
                while(right<m&&board[right][j]!='#')++right;
```

```
                if(right-left==size){
                    bool flag=true;
                    int k=0;
                    for(int i=right;i<left;i++){//此处left和right的方向有问
题
                        if(board[i][j]==' '||board[i][j]==word[k])++k;
                        else{
                            flag=false;
                            break;
                        }
                    }
                    if(flag)return true;
                    flag=true;
                    k=size-1;
                    for(int i=right;i<left;i++){//此处right和乐left的方向有
问题
                        if(board[i][j]==' '||board[i][j]==word[k])--k;
                        else{
                            flag=false;
                            break;
                        }
                    }
                    if(flag)return true;

                }
                left=right;
            }

        }
        return false;
    }
};
```

总结反思：

1. 一定要审题清晰，分析严密，不要去想当然
2. 此次竞赛的第二题有一个坑，就是数据大小超出了**INT_MAX**；此时数据的最大值为**long long Int**型的最大值**0x7fffffffffffff;**
3. 此次竞赛第二题的另一个不应该就是，对于二维数组的横纵轴没有搞清楚；对于二位数组**grid;**

   grid.size()表示的是高，即纵轴；

   grid[0].size()表示的是宽，即横轴；

- 加强代码力；此次的第三题，难度其实不是很大，但是翘起代码来还是会有点棘手。代码越长，越容易犯小错误。
- **left**和**right**一定要分清楚大小啊；以后建议可以用**begin**，**end**或者**small**和**big**来代替**left** 和**right**

力扣第261场周赛

第三题：

5892. 石子游戏 IX

难度 中等 10

Alice 和 Bob 再次设计了一款新的石子游戏。现有一行 n 个石子，每个石子都有一个关联的数字表示它的价值。给你一个整数数组 `stones` ，其中 `stones[i]` 是第 `i` 个石子的价值。

Alice 和 Bob 轮流进行自己的回合，**Alice** 先手。每一回合，玩家需要从 `stones` 中移除任一石子。

- 如果玩家移除石子后，导致 **所有已移除石子** 的价值 **总和** 可以被 3 整除，那么该玩家就 **输掉游戏** 。
- 如果不满足上一条，且移除后没有任何剩余的石子，那么 Bob 将会直接获胜（即便是在 Alice 的回合）。

假设两位玩家均采用 **最佳** 决策。如果 Alice 获胜，返回 `true` ；如果 Bob 获胜，返回 `false` 。

题目分析：这是一道博弈题

- 先手应该能够决定取胜与否
- 抓住问题的本质：将数转换为与3的余数
- 加强逻辑性的分析

```
class Solution {
public:
    bool firstTest(map<int,int>count,vector<int>stones){
        //序列1 12121212的检测
        if(count[1]==0)return false;
        count[1]-=1;
        int length=min(count[1],count[2])*2+count[0]+1;
        //如果count[1]>count[2],序列后面还可以添加1
        if(count[1]>count[2])length++;
        if(length%2==1&&length<stones.size())return true;
        else return false;
    }
    bool secondTest(map<int,int>count,vector<int>stones){
        //序列2 21212121的检测
        if(count[2]==0)return false;
```

```
        count[2]-=1;
        int length=min(count[1],count[2])*2+count[0]+1;
        //如果count[2]>count[1]，序列后面还可以添加2
        if(count[2]>count[1])length++;
        if(length%2==1&&length<stones.size())return true;
        else return false;
    }
    bool stoneGameIX(vector<int>& stones) {
        map<int,int>count;
        for(int i=0;i<stones.size();i++){
            if(count.count(stones[i]%3)==0)count[stones[i]%3]=1;
            else count[stones[i]%3]+=1;
        }
        return firstTest(count,stones)||secondTest(count,stones);
    }
};
```

第四题：（字符串处理-包含条件的最小字符序列处理）

5893. 含特定字母的最小子序列

难度 困难 2

给你一个字符串 `s` ，一个整数 `k` ，一个字母 `letter` 以及另一个整数 `repetition` 。

返回 `s` 中长度为 `k` 且 **字典序最小** 的子序列，该子序列同时应满足字母 `letter` 出现 **至少** `repetition` 次。生成的测试用例满足 `letter` 在 `s` 中出现 **至少** `repetition` 次。

子序列 是由原字符串删除一些（或不删除）字符且不改变剩余字符顺序得到的剩余字符串。

字符串 `a` 字典序比字符串 `b` 小的定义为：在 `a` 和 `b` 出现不同字符的第一个位置上，字符串 `a` 的字符在字母表中的顺序早于字符串 `b` 的字符。

题目分析：

- 最小子序列；含一个“最”字，且可以比较——数据结构：单调栈
- 单调栈本质上就是有条件的一个贪心
- 本题的条件限制：字符长度、指定重复字符的个数
- 单调栈处理的时候：主要时处理入栈和出栈时的条件关系

```
class Solution {
public:
```

```
    string smallestSubsequence(string s, int k, char letter, int
repetition) {
        //单调栈处理最小序列类问题
        //本质上是贪心
        //根据问题的描述，对出栈和入栈加以约束
        int n=s.size();
        //求某个下标i(包括i)之后的letter字符的数量
        vector<int>buf(n+1,0);
        for(int i=n-1;i>=0;i--)buf[i]=buf[i+1]+(s[i]==letter);
        stack<int>stk;
        int num=0;//记录特定字符的数量
        for(int i=0;i<n;i++){
            //弹出的条件-非空、更好的选择、长度冗余、重复冗余
            while(!stk.empty()&&stk.top()>s[i]&&(int)stk.size()+n-1-
i>=k){
                int temp=num;
                if(letter==stk.top())--temp;
                if(temp+buf[i]<repetition)break;
                stk.pop();
                num=temp;
            }
            //入栈的条件--栈长度有要求、留足空间for要求重复的字符
            if(stk.size()<k&&(s[i]==letter||k-
(int)stk.size()>repetition-num)){
                if(s[i]==letter)++num;
                stk.push(s[i]);
            }
        }
        string ans;
        while(!stk.empty())ans.push_back(stk.top()),stk.pop();
        reverse(ans.begin(),ans.end());
        return ans;
    }
};
```

类似题目：

1081. 不同字符的最小子序列

难度 中等 114

返回 `s` 字典序最小的子序列，该子序列包含 `s` 的所有不同字符，且只包含一次。

注意：该题与 316 https://leetcode.com/problems/remove-duplicate-letters/ 相同

周赛总结：

- 模拟分析、抓住问题发本质、代码处理时要严谨。

排名	用户名	得分	完成时间	题目1 (3)	题目2 (4)	题目3 (5)	题目4 (6)
873 / 3367	旭	7	0:22:31	0:10:15	0:22:31		

力扣第262场周赛：美团+力扣、遭遇滑铁卢

排名	用户名	得分	完成时间	题目1 (3)	题目2 (4)	题目3 (5)	题目4 (6)
1782 / 4260	旭	3	0:06:58	0:06:58			

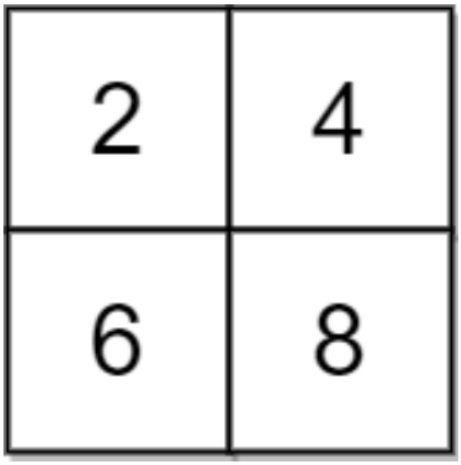
题目二：5895. 获取单值网格的最小操作数

给你一个大小为 `m x n` 的二维整数网格 `grid` 和一个整数 `x` 。每一次操作，你可以对 `grid` 中的任一元素 **加** `x` 或 **减** `x` 。

单值网格 是全部元素都相等的网格。

返回使网格化为单值网格所需的 **最小** 操作数。如果不能，返回 `-1` 。

示例 1：



题目分析：

- 错在想当然：不加证明的以为平均值左右可以取得最小操作
- 前面的思路大体是对的：
    1. 将二维数组转化为一维数组
    2. 可行性分析：所有的数对**x**的余数必须得相等
    3. 将增减量化为**1**；将所有数除以**x**
- 归纳法证明最小操作数：

代码分析：

```
//代码
class Solution {
public:
    int minOperations(vector<vector<int>>& grid, int x) {
        vector<int>datas;
        //可行性分析
        int base=grid[0][0]%x,temp=0;
        int m=grid.size(),n=grid[0].size();
        long long int sum=0,len=m*n;
```

```
        if(len==1)return 0;
        for(int i=0;i<m;i++){
            for(int j=0;j<n;j++){
                if(grid[i][j]%x!=base)return -1;
                temp=grid[i][j]/x;
                datas.push_back(temp);
                sum+=temp;
            }
        }
        //排序
        sort(datas.begin(),datas.end());
        //法二
        long long int res=0,begin=0,end=len-1;
        while(begin<end){
            res+=datas[end]-datas[begin];
            ++begin;
            --end;
        }
        return res;
    }
};
```

题目三：5896. 股票价格波动

给你一支股票价格的数据流。数据流中每一条记录包含一个 **时间戳** 和该时间点股票对应的 **价格** 。

不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的。如果两个有相同时间戳的记录出现在数据流中，前一条记录视为错误记录，后出现的记录 **更正** 前一条错误的记录。

请你设计一个算法，实现：

- **更新** 股票在某一时间戳的股票价格，如果有之前同一时间戳的价格，这一操作将 **更正** 之前的错误价格。
- 找到当前记录里 **最新股票价格** 。**最新股票价格** 定义为时间戳最晚的股票价格。
- 找到当前记录里股票的 **最高价格** 。
- 找到当前记录里股票的 **最低价格** 。

请你实现 `StockPrice` 类：

- `StockPrice()` 初始化对象，当前无股票价格记录。
- `void update(int timestamp, int price)` 在时间点 `timestamp` 更新股票价格为 `price` 。
- `int current()` 返回股票 **最新价格** 。
- `int maximum()` 返回股票 **最高价格** 。
- `int minimum()` 返回股票 **最低价格** 。

题目分析：

- 这道题比较容易报超时
- 题目比较长，但理解起来其实并没有这么难；最新股价这个数据比较好维护；
- 主要是更新数据后：最高股价和最低股价的维护比较费时，涉及数据的增删；
- 多种数据结构结合以弥补各数据结构之间的性能差异：通过两个优先队列来维护数据的最大值和最小值；通过两个哈希分别来记录时间戳的存在性和价格数据的个数

代码分析：

起初用二分查找来增删，但是效果不好；增删的代价依然很高，虽然查找比较容易。后来用哈希来增删，一定程度上是用空间换时间

```
class StockPrice {
private:
    //多种数据结构的结合以相互弥补各自性能的差异
    //优先队列用于排序
    //哈希用于存在性的验证（查找和删除）
    priority_queue<int,vector<int>>bigque;
    priority_queue<int,vector<int>,greater<int>>samllque;
    pair<int,int>newest;
    unordered_map<int,int>mymap;//价格-个数
    unordered_map<int,int>map;//最新:时间-价格

public:
    StockPrice() {
        newest={0,0};
    }

    void update(int timestamp, int price) {
        //最新价格的更新
        if(timestamp>=newest.first){
            newest={timestamp,price};
        }
        //价格入队列
        samllque.push(price);
        bigque.push(price);
```

```
        //哈希变化--相同价格的个数变化
        if(map.find(timestamp)!=map.end()){
            //timestamp存在
            --mymap[map[timestamp]];
        }
        //哈希变化-时间-价格
        map[timestamp]=price;
        ++mymap[price];
    }

    int current() {
        return newest.second;
    }

    int maximum() {
        while(!bigque.empty()){
            if(mymap[bigque.top()]){
                return bigque.top();
            }
            bigque.pop();
        }
        return -1;
    }

    int minimum() {
        while(!samllque.empty()){
            if(mymap[samllque.top()]){
                return samllque.top();
            }
            samllque.pop();
        }
        return -1;
    }
};

/**
 * Your StockPrice object will be instantiated and called as such:
 * StockPrice* obj = new StockPrice();
 * obj->update(timestamp,price);
 * int param_2 = obj->current();
 * int param_3 = obj->maximum();
 * int param_4 = obj->minimum();
 */
```

题目四：5897. 将数组分成两个数组并最小化数组和的差

给你一个长度为 `2 * n` 的整数数组。你需要将 `nums` 分成 **两个** 长度为 `n` 的数组，分别求出两个数组的和，并 **最小化** 两个数组和之 **差的绝对值** 。 `nums` 中每个元素都需要放入两个数组之一。

请你返回 **最小** 的数组和之差。

题目分析：

- 起初对于这道题：我又不加证明的想当然的认为，排序后求相邻对数之间的差值，对差值又进行相同的操作，显然，没这么简单
- 这道题的数据很少，**n<=15**；需要暴力枚举所有可能进行判断，但是**2**的**30**次方有点大，超时了；

```
class Solution {
public:
    int minimumDifference(vector<int>& nums) {
        //n<15
        //暴力枚举
        int n=nums.size();
        int res=INT_MAX;
        for(int i=(1<<n/2)-1;i<(1<<n);i++){
            //i中含1的个数
            int num1=0;
            int sum=0;
            for(int j=0;j<n;j++){
                if((i>>j)&1==1)++num1;
            }
            if(num1!=n/2)continue;
            for(int j=0;j<n;j++){
                if((i>>j)&1==1){
                    sum+=nums[j];
                }else sum-=nums[j];
            }
            res=min(res,abs(sum));
        }
        return res;
    }
};
```

- 折半搜索+暴力枚举；两个**2**的**15**次方之间进行匹配搜索（超时）

```
class Solution {
private:
    vector<vector<int>>begin,end;
```

```
    void _enum(vector<int>&nums,int L,int R,vector<vector<int>>&tp){
        int n=R-L;
        for(int i=0;i<(1<<n);i++){
            int num1=0,sum=0;
            for(int j=0;j<n;j++){
                if((i>>j)&1==1){
                    sum+=nums[L+j];
                    ++num1;
                }else sum-=nums[L+j];
            }
            tp[num1].push_back(sum);
        }
    }
public:
    int minimumDifference(vector<int>& nums) {
        //n<15
        //暴力枚举
        //常规暴力枚举超出时间限制
        //分成两半(折半搜索)+暴力枚举
        int n=nums.size();
        begin.resize(n/2+1);
        end.resize(n/2+1);
        _enum(nums,0,n/2,begin);
        _enum(nums,n/2,n,end);

        //枚举
        int ans=INT_MAX;
        for(int i=0;i<=n/2;i++){
            for(int j=(int)begin[i].size()-1;j>=0;j--){
                for(int k=(int)end[n/2-i].size()-1;k>=0;k--){
                    ans=min(ans,abs(begin[i][j]+end[n/2-i][k]));
                }
            }
        }
        return ans;

    }
};
```

- 剪枝优化：对相关数组进行排序；两个数组之间进行枚举时，只要第一个大于**0**的值

```
class Solution {
private:
    vector<vector<int>>begin,end;
    void _enum(vector<int>&nums,int L,int R,vector<vector<int>>&tp){
        int n=R-L;
        for(int i=0;i<(1<<n);i++){
            int num1=0,sum=0;
            for(int j=0;j<n;j++){
```

```
                if((i>>j)&1==1){
                    sum+=nums[L+j];
                    ++num1;
                }else sum-=nums[L+j];
            }
            tp[num1].push_back(sum);
        }
        //排序以剪枝条
        for(int i=0;i<tp.size();i++){
            sort(tp[i].begin(),tp[i].end());
        }
    }
public:
    int minimumDifference(vector<int>& nums) {
        //n<15
        //暴力枚举
        //常规暴力枚举超出时间限制
        //分成两半(折半搜索)+暴力枚举
        int n=nums.size();
        begin.resize(n/2+1);
        end.resize(n/2+1);
        _enum(nums,0,n/2,begin);
        _enum(nums,n/2,n,end);

        //枚举
        int ans=INT_MAX;
        for(int i=0;i<=n/2;i++){
            for(int j=(int)begin[i].size()-1,k=0;j>=0;j--){
            //搜索减去枝条
            //只取大于0处的正数
            //ans=min(ans,abs(begin[i][j]+end[n/2-i][k]));
                while(k<end[n/2-i].size()&&begin[i][j]+end[n/2-i][k]
<0)++k;
                if(k<end[n/2-i].size())ans=min(ans,abs(begin[i]
[j]+end[n/2-i][k]));

            }
        }
        return ans;

    }
};
```

周赛总结:

- 不要想当然；所有的算法都要尽可能的有依据；不要被以前所做过的题目所诱导。
- 充分结合不同的数据结构之间的特点，数据结构的优化也能很大程度上优化时间（即用空间换时间）
- 当数据很小时，一般可以试一下暴力枚举和剪枝搜索

力扣第63场双周赛总结

- 前两道题都还比较简单
- 第三道题：**5888.** 网络空闲的时刻
- 思路很清晰：**dijkstra**模板+简单数学推导
- 但是在写这道题的时候，我发现我对图论中经典算法的熟练度还有欠缺
- ```
  class Solution {
  public:
      int networkBecomesIdle(vector<vector<int>>& aedges, vector<int>&
  patience) {
          //求每个结点的最短到主结点的最短路径
          int n=patience.size();//结点数
          vector<vector<int>>edges(n);
          for(int i=0;i<aedges.size();i++){
              edges[aedges[i][0]].push_back(aedges[i][1]);
              edges[aedges[i][1]].push_back(aedges[i][0]);
          }
          vector<int>path(n,INT_MAX);
          vector<bool>visited(n,false);
          path[0]=0;visited[0]=true;
          int tempnode=0,num=1;

          //默认从大到小排序priority_queue<pair<int,int>>myque;
          priority_queue<pair<int,
  int>,vector<pair<int,int>>,greater<pair<int,int>> >myque;
          unordered_set<int>myset;
          while(num<n){
              visited[tempnode]=true;//表示已经访问
              for(auto&edge:edges[tempnode]){
                  if(visited[edge])continue;
                  path[edge]=min(path[edge],path[tempnode]+1);
                  if(myset.find(edge)==myset.end())
                  myque.push({path[edge],edge});
                  myset.insert(edge);
              }
              ++num;


              tempnode=myque.top().second;
              myque.pop();
          }
          //找延时的最大值
          //return path[0];
          int ans=0;
          for(int i=1;i<n;i++){
              ans=max(ans,1+2*path[i]+patience[i]*
  (2*path[i]/patience[i]-(2*path[i]%patience[i]==0?1:0)));
  ```
  ```

```
            }
            return ans;



        }
    };
```

5888. 网络空闲的时刻

难度 中等 0 收藏 分享 切换为英文 接收动态 反馈

给你一个有 `n` 个服务器的计算机网络，服务器编号为 `0` 到 `n - 1` 。同时给你一个二维整数数组 `edges` ，其中 `edges[i] = [u_i, v_i]` 表示服务器 `u_i` 和 `v_i` 之间有一条信息线路，在 **一秒** 内它们之间可以传输 **任意** 数目的信息。再给你一个长度为 `n` 且下标从 **0** 开始的整数数组 `patience` 。

题目保证所有服务器都是 **相通** 的，也就是说一个信息从任意服务器出发，都可以通过这些信息线路直接或间接地到达任何其他服务器。

编号为 `0` 的服务器是 **主** 服务器，其他服务器为 **数据** 服务器。每个数据服务器都要向主服务器发送信息，并等待回复。信息在服务器之间按 **最优** 线路传输，也就是说每个信息都会以 **最少时间** 到达主服务器。主服务器会处理 **所有** 新到达的信息并 **立即** 按照每条信息来时的路线 **反方向** 发送回复信息。

在 `0` 秒的开始，所有数据服务器都会发送各自需要处理的信息。从第 `1` 秒开始，**每** 一秒最 **开始** 时，每个数据服务器都会检查它是否收到了主服务器的回复信息（包括新发出信息的回复信息）：

- 如果还没收到任何回复信息，那么该服务器会周期性 **重发** 信息。数据服务器 `i` 每 `patience[i]` 秒都会重发一条信息，也就是说，数据服务器 `i` 在上一次发送信息给主服务器后的 `patience[i]` 秒 **后** 会重发一条信息给主服务器。
- 否则，该数据服务器 **不会重发** 信息。

当没有任何信息在线路上传输或者到达某服务器时，该计算机网络变为 **空闲** 状态。

请返回计算机网络变为 **空闲** 状态的 **最早秒数** 。

力扣第263场周赛：字节跳动

排名	用户名	得分	完成时间	题目1 (3)	题目2 (4)	题目3 (5)	题目4 (6)
1076 / 4571	旭	12	0:43:01	0:07:29	0:38:01 1	0:28:08	

感觉这次周赛的前三题并不是很难；第三题与上次周赛的最后一题有点相似，暴力枚举就可以出来！！！

这周的周赛和双周赛都卡在图上了，这让我很伤，下一周的目标就是强化图一章相关的数据结构和算法！！！

题目描述：5905. 到达目的地的第二短时间

城市用一个 **双向连通** 图表示，图中有 `n` 个节点，从 `1` 到 `n` 编号（包含 `1` 和 `n`）。图中的边用一个二维整数数组 `edges` 表示，其中每个 `edges[i] = [$u_i$, $v_i$]` 表示一条节点 `$u_i$` 和节点 `$v_i$` 之间的双向连通边。每组节点对由 **最多一条** 边连通，顶点不存在连接到自身的边。穿过任意一条边的时间是 `time` 分钟。

每个节点都有一个交通信号灯，每 `change` 分钟改变一次，从绿色变成红色，再由红色变成绿色，循环往复。所有信号灯都 **同时** 改变。你可以在 **任何时候** 进入某个节点，但是 **只能** 在节点 **信号灯是绿色时** 才能离开。如果信号灯是 **绿色** ，你 **不能** 在节点等待，必须离开。

第二小的值 是 **严格大于** 最小值的所有值中最小的值。

- 例如，`[2, 3, 4]` 中第二小的值是 `3` ，而 `[2, 2, 4]` 中第二小的值是 `4` 。

给你 `n`、`edges`、`time` 和 `change` ，返回从节点 `1` 到节点 `n` 需要的 **第二短时间** 。

注意：

- 你可以 **任意次** 穿过任意顶点，**包括** `1` 和 `n` 。
- 你可以假设在 **启程时** ，所有信号灯刚刚变成 **绿色** 。

提示：

- `2 <= n <= $10^4$`
- `n - 1 <= edges.length <= min(2 * $10^4$, n * (n - 1) / 2)`
- `edges[i].length == 2`
- `1 <= $u_i$, $v_i$ <= n`
- `$u_i$ != $v_i$`
- 不含重复边
- 每个节点都可以从其他节点直接或者间接到达
- `1 <= time, change <= $10^3$`

题目分析：（最短路径算法+松弛）

方法一：BFS + 二次松弛

注意到$time$和$change$都是全局统一的，我们到达一个点的时间，实际上是由经过的边数唯一确定的。

现在假设到达点n最短需要经过k条边，则我们可以知道，经过$k+2$条边一定可以到达n——只需要随意去一个相邻点再回头即可。题目要求第二短时间，这就说明我们只需要考虑是否能经过$k+1$条边到达点n即可。

解决方法是，在BFS时允许对每个点进行两次松弛，也即，在第一次到达这个点（假设经过的距离为d），以及第一次经过距离$d+1$到达这个点时，都对这个点进行松弛。

最后，如果我们可以经过$k+1$条边到达点n，最短时间就是行动$k+1$次的总时间；否则，最短时间就是行动$k+2$次的总时间。因为每个点最多入队两次，所以时间复杂度和标准的BFS一样，还是线性的。

- 时间复杂度$\mathcal{O}(V+E)$。
- 空间复杂度$\mathcal{O}(V+E)$。

- 现在题目做到这个份上：感觉大部分的题目首先是理解题意进行分析，然后抓住问题的本质和关键，选择恰当的数据结构以及算法来解决；以前总感觉看到题目没办法分析，下不了手，现在感觉可以下手，但是数据结构和算法部分不是很熟练，老是出**bug.**
- 周赛和双周赛就暴露出，我在图论这部分的算法和数据结构不是很熟，需要进行一个强化
    1. 这个问题所有路径的长度都相同，求最短路径可以用广度优先搜索或者是深度优先搜索
    2. 第二长的路径本质上就是求是否存在最短路径**+1**的长度的路径；应为最短长度路径**+2**长度的路径是一定存在的
    3. 解决这道题的关键就是松弛操作：通过点的两次入队来判断到该点的路径是否存在最短路径**+1**的路径

```
class Solution {
public:
    int secondMinimum(int n, vector<vector<int>>& edges, int time, int
change) {
        //求最短路径
        vector<vector<int>>e(n+1);
        vector<int>path(n+1,INT_MAX);
        vector<bool>visited(n+1,false);
        path[1]=0;
        visited[1]=true;
        //构建邻接表
        for(auto&v:edges){
            e[v[0]].push_back(v[1]);
            e[v[1]].push_back(v[0]);
        }
```

```
        //由于每条边的长度相同-广度优先搜索
        queue<int>que;
        que.push(1);
        int tag=1;
        while(!que.empty()){
            tag=que.front();
            que.pop();
            for(auto v:e[tag]){
               //松弛操作
               if(path[tag]+1<path[v]){
                   path[v]=path[tag]+1;
                   que.push(v);//一次入队
               }else if(path[tag]==path[v]||
(visited[tag]&&path[v]>path[tag])){
                   //1:两点的最短距离相等，则到该点的距离可以加一
                   //2：tag点可以松弛，v点在tag点之后，符合路径要求
                   if(!visited[v]){
                       que.push(v);//二次入队
                   }
                   visited[v]=true;//表示已经二次入队了
               }

            }
        }
        int len=path[n]+2;
        if(visited[n])len-=1;
        int wait=0,sum=0;//等待的时间
        for(int i=0;i<len;i++){
            if(sum/change%2==1){
                wait+=change-sum%change;
                sum+=change-sum%change;
            }
            sum+=time;
        }
        return time*len+wait;


    }
};
```

周赛总结：

本周出**bug**的地方：

1. 混淆变量：如何改正这一点：平时写代码的时候尽量的去多谢注释
2. 优先队列的使用
    - 默认是从大到小排序
    - 若要修改为从小到大排序，要在后面加上 **vector<类型>,greater<类型>**

双线方针：

- 通过做困难题来锤炼自己分析问题的能力
- 通过特定的数据结构及其相关算法的强化来强化自己的代码能力

力扣第264场周赛：四题的水平最后却只AC了一题！！！

第一题：5906. 句子中的有效单词数

条件判断比较多。尽管做题时已经是比较的严谨了，但是最后还是忽略了诸如Q-!的情况：即每个连接符的左右必须是字母！！！

5906. 句子中的有效单词数

句子仅由小写字母（`'a'` 到 `'z'`）、数字（`'0'` 到 `'9'`）、连字符（`'-'`）、标点符号（`'!'`、`'.'` 和 `','`）以及空格（`' '`）组成。每个句子可以根据空格分解成 **一个或者多个 token**，这些 token 之间由一个或者多个空格 `' '` 分隔。

如果一个 token 同时满足下述条件，则认为这个 token 是一个有效单词：

- 仅由小写字母、连字符和/或标点（不含数字）。
- **至多一个** 连字符 `'-'`。如果存在，连字符两侧应当都存在小写字母（`"a-b"` 是一个有效单词，但 `"-ab"` 和 `"ab-"` 不是有效单词）。
- **至多一个** 标点符号。如果存在，标点符号应当位于 token 的 **末尾**。

这里给出几个有效单词的例子：`"a-b."`、`"afad"`、`"ba-c"`、`"a!"` 和 `"!"`。

给你一个字符串 `sentence`，请你找出并返回 `sentence` 中 **有效单词的数目**。

```
class Solution {
private:
    bool isvalid(string&t){
        //判断是否是有效单词
        int n=t.size();
        if(n==0)return false;
```

```
        //开头和末尾不能有连字符
        if(t[0]=='-'||t[n-1]=='-')return false;
        //统计连字符个数-统计标点符号个数-判断是否有数字存在
        int numl=0,numb=0;
        for(int i=0;i<n;i++){
            if(t[i]>='0'&&t[i]<='9')return false;//存在数字，返回false
            if(t[i]=='-'){
                //此处要判断连接符的左右必须为小写字母
                if(i>0&&i<n-1&&(t[i-1]<'a'||t[i-1]>'z'||t[i+1]
<'a'||t[i+1]>'z'))
                    return false;
                ++numl;
            }
            else if(t[i]<'a'||t[i]>'z')++numb;
        }
        if(numl>1)return false;//连字符至多有一个
        if(numb){
            //存在标点符号
            if(numb>1)return false;
            if(t[n-1]!='!'&&t[n-1]!=','&&t[n-1]!='.')return false;
        }
        return true;
    }
public:
    int countValidWords(string s) {
        //以空格为单位对单词进行分隔
        int n=s.size();
        string temp;
        int res=0,i=0;
        while(i<n){
            temp.clear();
            while(i<n&&s[i]!=' '){
                temp.push_back(s[i]);
                ++i;
            }
            //存在多个空格在一起的可能--排除空格
            while(i<n&&s[i]==' ')++i;

            if(isvalid(temp))++res;
        }
        return res;
    }
};
```

第二题：直接暴力AC

第三题：5908. 统计最高分的节点数目

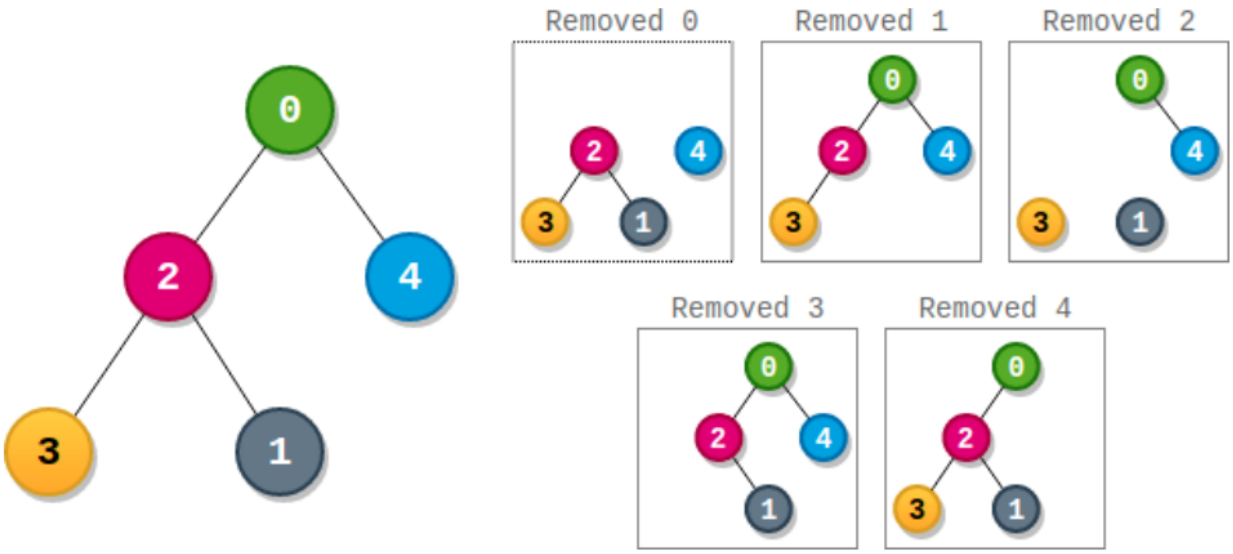
该题的思路也比较简单：直接构建二叉树+dfs(一开始超时了，我忽略了每个右父节点的最终父节点都是根结点这个条件)

给你一棵根节点为 `0` 的 **二叉树** ，它总共有 `n` 个节点，节点编号为 `0` 到 `n - 1` 。同时给你一个下标从 **0** 开始的整数数组 `parents` 表示这棵树，其中 `parents[i]` 是节点 `i` 的父节点。由于节点 `0` 是根，所以 `parents[0] == -1` 。

一个子树的 **大小** 为这个子树内节点的数目。每个节点都有一个与之关联的 **分数** 。求出某个节点分数的方法是，将这个节点和与它相连的边全部 **删除** ，剩余部分是若干个 **非空** 子树，这个节点的 **分数** 为所有这些子树 **大小的乘积** 。

请你返回有 **最高得分** 节点的 **数目** 。

示例 1:



输入：parents = [-1,2,0,2,0]
输出：3
解释：
- 节点 0 的分数为：3 * 1 = 3
- 节点 1 的分数为：4 = 4
- 节点 2 的分数为：1 * 1 * 2 = 2
- 节点 3 的分数为：4 = 4
- 节点 4 的分数为：4 = 4
最高得分为 4 ，有三个节点得分为 4 （分别是节点 1，3 和 4 ）。

```
class Solution {
private:
    //首先要构建一颗子树
    struct node{
        int id;//结点编号
        long long int score;//结点分数
        node*left;
        node*right;//左右节点
        node*parent;//父节点
        long long int size;//以该结点为根结点的子树的大小
        int tag;//tag为0时表示左，为1时表示右
        node(int i){parent=right=left=nullptr;id=i;score=size=tag=0;}
    };
    vector<node*>nodes;
    int dfs(node*root){
        if(root==nullptr)return 0;
        int sum=0;
        sum+=dfs(root->left);
        sum+=dfs(root->right);
        root->size=sum+1;
        return sum+1;
```

```
    }
    long long int getscore(node*root){
        //求每个结点的分数
        long long int res=1;
        if(root->parent){
            //当存在父节点
            //求终极父结点--即根结点
            //node*temp=root;
            //while(temp->parent)temp=temp->parent;
            res*=nodes[0]->size - root->size;
        }
        if(root->left)res*=root->left->size;
        if(root->right)res*=root->right->size;
        root->score=res;
        return res;
    }
    //由于超时-这次采用dfs来求分数
public:
    int countHighestScoreNodes(vector<int>& parents) {
        int n=parents.size();
        //vector<node*>nodes(n);
        nodes.resize(n);
        for(int i=0;i<n;i++){
            nodes[i]=new node(i);
        }
        for(int i=1;i<n;i++){
            int p=parents[i];
            nodes[i]->parent=nodes[p];//父节点
            if(nodes[p]->tag==0){
                nodes[p]->left=nodes[i];
                ++nodes[p]->tag;
            }else nodes[p]->right=nodes[i];
        }
        //搜索二叉树，以确定每棵子树的大小
        nodes[0]->size=dfs(nodes[0]);
        //求每个子节点的分数，并且确定最大分数
        long long int max_score=0;
        for(int i=0;i<n;i++){
            max_score=max(getscore(nodes[i]),max_score);
        }
        int ans=0;
        for(int i=0;i<n;i++){
            if(nodes[i]->score==max_score)++ans;
        }
        //return nodes[1]->score;
        return ans;

    }
};
```

第四题：图中求关键路径——5909. 并行课程 III

这道题属于是图中的一个应用——求图中的关键路径：利用队列进行拓扑排序+状态转移

每个结点的完成时间为其所有前趋结点完成的最晚时间+完成该结点课程所需要的时间

```
class Solution {
public:
    int minimumTime(int n, vector<vector<int>>& relations, vector<int>&
time) {
        //构建有向图
        //每个结点的开始时间为其前趋结点中的最晚完成时间
        //每个结点的结束时间为：最晚完成时间+完成该课程所需的时间
        //int n=time.size();
        vector<vector<int>>edges(n+1);
        vector<int>indegree(n+1,0);//每个结点的入度
        for(auto&v:relations){
            edges[v[0]].push_back(v[1]);
            ++indegree[v[1]];
        }
        vector<int>endtime(n+1,0);//表示的没门课程的最早完成时间
        queue<int>que;
        //入度为0的结点入队--从1开始
        int res=0;
        for(int i=1;i<=n;i++){
            if(indegree[i]==0){
                que.push(i);
                endtime[i]=time[i-1];
            }
        }
        while(!que.empty()){
            int t=que.front();
            que.pop();
            for(auto u:edges[t]){
                endtime[u]=max(endtime[u],endtime[t]+time[u-1]);
                --indegree[u];
                if(indegree[u]==0)que.push(u);
            }
        }
        for(int i=1;i<=n;i++){
            res=max(res,endtime[i]);
        }
        return res;
    }
};
```

周赛总结：

1. 周赛的时候允许开一个**vs studio**用于调试程序中的**bug,**比如此次第一题难度其实不大，但是由于判断的条件比较多，很容易漏掉条件或者判断条件重复，并且很难找到**bug**，只有结合具体的错误的实例才能够比较容易的找到**bug.**
2. 建议：如果一个题目的输入和输出比较的好实现，建议在实在查不出错的时候使用**vs**来调试
3. 此次第三题左后由于时间问题超时也没时间取思考为什么超时---总的来说还是反映出我做题的速度和代码力上还有待提升
4. 大三下还是不以刷力扣题为主，主要还是查漏补缺以及通过做困难题来锻炼自己的思路
5. 寒假的时候可以针对自己做中等题的速度和正确率方面进行一个较为系统性的训练

力扣第265场周赛：

又是两题收尾！！！

排名	用户名	得分	完成时间	题目1 (3)	题目2 (4)	题目3 (5)	题目4 (6)
1454 / 4181	旭	7	0:19:23	0:04:41	0:19:23		

5916. 转化数字的最小运算数

题目描述：

给你一个下标从 **0** 开始的整数数组 `nums` ，该数组由 **互不相同** 的数字组成。另给你两个整数 `start` 和 `goal` 。

整数 `x` 的值最开始设为 `start` ，你打算执行一些运算使 `x` 转化为 `goal` 。你可以对数字 `x` 重复执行下述运算：

如果 `0 <= x <= 1000` ，那么，对于数组中的任一下标 `i` （`0 <= i < nums.length`），可以将 `x` 设为下述任一值：

- `x + nums[i]`
- `x - nums[i]`
- `x ^ nums[i]` （按位异或 XOR）

注意，你可以按任意顺序使用每个 `nums[i]` 任意次。使 `x` 越过 `0 <= x <= 1000` 范围的运算同样可以生效，但该该运算执行后将不能执行其他运算。

返回将 `x = start` 转化为 `goal` 的最小操作数；如果无法完成转化，则返回 `-1` 。

提示：

- `1 <= nums.length <= 1000`
- -10^9 `<= nums[i], goal <=` 10^9
- `0 <= start <= 1000`
- `start != goal`
- `nums` 中的所有整数互不相同

题目分析：

1. 数据**1000**，暴力行不通，**dfs-pass;**求最小操作--加减和异或--是否是有什么数学性质--动态规划？？？
2. 说实话我被这大的数据给蒙住了，思路一下子就被带偏了；
3. 求最小操作数的最佳的暴力法应该是广度优先搜索，广度优先搜索一定能找到最佳的答案，但是搜索面比较广，但是此处可以有一处剪枝-->数据范围限定在**0--1000**以内。
4. 广度优先的实现：通过队列

```
class Solution {
public:
    int minimumOperations(vector<int>& nums, int start, int goal) {
        //bfs--深度优先搜索
        /*
```

起初看到这道题目的时候，看到数据是1000,我在潜意识中就排除了暴力的解法，其中就包括深度优先。

但是我没有意识到的是，尽管深度优先和广度优先都是暴力的搜索，但是对于这种求最小操作数的问题，用广度优先可以覆盖所有情况，并且党找到一条路径时，一定是最近的，避免了很多的盲目搜索

```
        */
        queue<int>que;
        que.push(start);
        int res=0;
        //剪枝-避免重复搜索
        //如果0-1000中的某个数已经出现过，则不再搜索
        vector<bool>exist(1005,false);
        exist[start]=true;
        while(!que.empty()){
            //一圈一圈的扩展
            int n=que.size();
            for(int i=0;i<n;i++){
                int cur=que.front();
                que.pop();
                for(auto t:nums){
                    int a=cur+t;
                    int b=cur-t;
                    int c=cur^t;
                    if(a==goal||b==goal||c==goal)return res+1;
                    if(a<=1000&&a>=0&&!exist[a]){
                        que.push(a);
                        exist[a]=true;
                        }
                    if(b<=1000&&b>=0&&!exist[b]){
                        que.push(b);
                        exist[b]=true;
                    }
                    if(c<=1000&&c>=0&&!exist[c]){
                        que.push(c);
                        exist[c]=true;
                    }
                }
            }
            ++res;
        }
        return -1;
    }
};
```

题目总结：

- 对于常规的题目的分析，首先想到的肯定是暴力的思维！而对于暴力方法，常见的就是暴力枚举、深度优先、广度优先。
- 暴力枚举需要列举所有的情况，适用于数据较小，且所有情况容易列举的情况下使用。
- 深度优先一般解决的是存在性的问题----即是否存在这样一个符合条件的答案；深度优先对于解决存在性的问题的效率比较高，但前提是深度优先必须要有一个终止的机制，否则会陷入无限递归。
- 广度优先适宜于解决最优问题，诸如最小，最优之类的。如果存在答案，那么广度优先可以比较高效的找到这最优的答案；
- 深度优先和广度优先都可以通过剪枝等方法来提高效率

5917. 同源字符串检测（比较难的动态规划）

力扣第266场周赛（重大滑铁卢-一题未做出来）

反思：

1. 一是平时做题习惯不好，分析问题，代码书写，时间把控都比较随意！！！导致在比赛时漏洞百出！！！
2. 二是最近的刷题的方针有点不对头，总是执着于困难题目，殊不知现在简单题和中等题都没有较高的把握拿下！！！
3. 三是，比赛当天晚上没有睡好觉，加上又比较冷！！！身体状态有点差！！！
4. 这次周赛的题还是比较灵活的，审题大意！！

2062. 统计字符串中的元音子字符串

> 2062. 统计字符串中的元音子字符串
>
> 难度 简单 2
>
> **子字符串** 是字符串中的一个连续（非空）的字符序列。
>
> **元音子字符串** 是 **仅** 由元音（`'a'`、`'e'`、`'i'`、`'o'` 和 `'u'`）组成的一个子字符串，且必须包含 **全部五种** 元音。
>
> 给你一个字符串 `word` ，统计并返回 `word` 中 **元音子字符串的数目** 。

提示：

- `1 <= word.length <= 100`
- `word` 仅由小写英文字母组成

示例 3：

```
输入：word = "cuaieuouac"
输出：7
解释：下面列出 word 中的元音子字符串（斜体加粗部分）：
- "c*uaieuo*uac"
- "c*uaieuou*ac"
- "c*uaieuoua*c"
- "cu*aieuo*uac"
- "cu*aieuou*ac"
- "cu*aieuoua*c"
- "cua*ieuoua*c"
```

分析

1. 仔细审题会发现，这道题处理起来还是有点棘手
2. 双指针的把控可能会有点不熟练；
3. 建议对于数据**<100**可以尝试一下暴力迅速通过

2063. 所有子字符串中的元音

2063. 所有子字符串中的元音

难度 中等 5

给你一个字符串 `word` ，返回 `word` 的所有子字符串中 **元音的总数** ，元音是指 `'a'` 、 `'e'` 、 `'i'` 、 `'o'` 和 `'u'` 。

子字符串 是字符串中一个连续（非空）的字符序列。

注意： 由于对 `word` 长度的限制比较宽松，答案可能超过有符号 32 位整数的范围。计算时需当心。

示例 2：

```
输入：word = "abc"
输出：3
解释：
所有子字符串是："a"、"ab"、"abc"、"b"、"bc" 和 "c" 。
- "a"、"ab" 和 "abc" 每个都有 1 个元音
- "b"、"bc" 和 "c" 每个都有 0 个元音
因此，元音总数 = 1 + 1 + 1 + 0 + 0 + 0 = 3 。
```

提示：

- `1 <= word.length <= 10^5`
- `word` 由小写英文字母组成

分析：

1. 一定要注意数据量，此时的数据量很大，这道题用暴力搜索一定会超时

组合数学 计数 Go

遍历 $word$，若 $word[i]$ 是元音，我们考察它能出现在多少个子字符串中。

设 $word$ 的长度为 n。子字符串 $word[l..r]$ 若要包含 $word[i]$，则必须满足

- $0 \le l \le i$
- $i \le r \le n-1$

这样的 l 有 $i+1$ 个，r 有 $n-i$ 个，因此有 $(i+1)(n-i)$ 个子字符串，所以 $word[i]$ 在所有子字符串中一共出现了 $(i+1)(n-i)$ 次。

累加所有出现次数即为答案。

```
class Solution {
public:
    long long countVowels(string word) {
        long long int n=word.size();
        long long int res=0;
        for(long long int i=0;i<n;i++){

 if(word[i]=='a'||word[i]=='e'||word[i]=='i'||word[i]=='o'||word[i]=='u'
){
                res+=(i+1)*(n-i);
            }
        }
        return res;
    }
};
```

2064. 分配给商店的最多商品的最小值

2064. 分配给商店的最多商品的最小值

难度 中等 10

给你一个整数 `n` ，表示有 `n` 间零售商店。总共有 `m` 种产品，每种产品的数目用一个下标从 **0** 开始的整数数组 `quantities` 表示，其中 `quantities[i]` 表示第 `i` 种商品的数目。

你需要将 **所有商品** 分配到零售商店，并遵守这些规则：

- 一间商店 **至多** 只能有 **一种商品** ，但一间商店拥有的商品数目可以为 **任意** 件。
- 分配后，每间商店都会被分配一定数目的商品（可能为 `0` 件）。用 `x` 表示所有商店中分配商品数目的最大值，你希望 `x` 越小越好。也就是说，你想 **最小化** 分配给任意商店商品数目的 **最大值** 。

请你返回最小的可能的 `x` 。

示例 2：

```
输入：n = 7, quantities = [15,10,10]
输出：5
解释：一种最优方案为：
- 15 件种类为 0 的商品被分配到前 3 间商店，分配数目为：5，5，5 。
- 10 件种类为 1 的商品被分配到接下来 2 间商店，数目为：5，5 。
- 10 件种类为 2 的商品被分配到最后 2 间商店，数目为：5，5 。
分配给所有商店的最大商品数目为 max(5, 5, 5, 5, 5, 5, 5) = 5 。
```

提示：

- `m == quantities.length`
- `1 <= m <= n <= 10^5`
- `1 <= quantities[i] <= 10^5`

分析：

1. 注意数据量，这道题的数据量依然很大
2. 通过暴力的方发依次列举所有可能会超时
3. 这里有一个特殊的方法，即二分搜索答案

二分法
二分选取一个mid作为答案，检查mid是否合法。
由于商品必须分配完成不能剩余 且 每个商店只能分配一种商品。
则为每一个商店都分配mid件商品，有三种情况：
令

$$cnt = \sum_{k=0}^{n} quantities_k / mid$$

代表 对每个商店都分配 mid 商品可以满足多少个商店

$$f(k) = \begin{cases} 1, quantities_k \bmod mid \neq 0 \\ 0, quantities_k \bmod mid = 0 \end{cases}, mod = \sum_{k=0}^{n} f(k)$$

代表 对cnt个商店都分配 mid 个商品后，还剩余 mod 种商品没有分配完

- $cnt > n$ 说明足够对每个商店分配 mid 商品，但是一定存在剩余，不合法。
- $cnt = n$ 说明足够对每个商店分配 mid 商品，但必须满足没有剩余 即 $mod = 0$
- $cnt < n$ 说明不能对每个商店都分配 mid 商品，但剩余的商品种类必须满足未能分配的商店即 $mod \leq n - cnt$。我们可以把所有剩下的商品分配到剩余的商店中，若仍有商店剩余就不分配商品，则每种剩余的商品数量一定小于mid。

总和所有条件，满足

$$cnt + mod \leq n$$

```
class Solution {
public:
    int minimizedMaximum(int n, vector<int>& quantities) {
        //二分枚举答案
        int
begin=1,end=*max_element(quantities.begin(),quantities.end());
        while(begin<=end){
            int mid=(begin+end)>>1;
```

```
            int cut=0;
            int mod=0;
            for(auto q:quantities){
                if(q%mid!=0)++mod;
                cut+=q/mid;
            }
            if(cut+mod>n){
                begin=mid+1;
            }else end=mid-1;
        }
        return begin;
    }
};
```

5921. 最大化一张图中的路径价值

5921. 最大化一张图中的路径价值

难度 困难 7

给你一张 **无向** 图，图中有 `n` 个节点，节点编号从 `0` 到 `n - 1` （**都包括**）。同时给你一个下标从 **0** 开始的整数数组 `values` ，其中 `values[i]` 是第 `i` 个节点的 **价值** 。同时给你一个下标从 **0** 开始的二维整数数组 `edges` ，其中 `edges[j] = [uj, vj, timej]` 表示节点 `uj` 和 `vj` 之间有一条需要 `timej` 秒才能通过的无向边。最后，给你一个整数 `maxTime` 。

合法路径 指的是图中任意一条从节点 `0` 开始，最终回到节点 `0` ，且花费的总时间 **不超过** `maxTime` 秒的一条路径。你可以访问一个节点任意次。一条合法路径的 **价值** 定义为路径中 **不同节点** 的价值 **之和** （每个节点的价值 **至多** 算入价值总和中一次）。

请你返回一条合法路径的 **最大** 价值。

注意： 每个节点 **至多** 有 **四条** 边与之相连。

提示：

- `n == values.length`
- `1 <= n <= 1000`
- `0 <= values[i] <= 10^8`
- `0 <= edges.length <= 2000`
- `edges[j].length == 3`
- `0 <= uj < vj <= n - 1`
- `10 <= timej, maxTime <= 100`
- `[uj, vj]` 所有节点对 **互不相同** 。
- 每个节点 **至多有四条** 边。
- 图可能不连通。

分析：

1. 这道题，同样要注意数据，发现**maxtime<100;**每个节点至多有四条边
2. 可以回溯爆索

```
class Solution {
private:
    int maxTime;
    int maxValue,current_value,current_time;
    vector<vector<pair<int,int>>>Edge;
    vector<int>values,vis;
    void dfs(int u){
        if(!vis[u]){
            current_value+=values[u];
        }
        vis[u]++;
        if(u==0){
            maxValue=max(maxValue,current_value);
        }
        for(auto [v,time]:Edge[u]){
            if(current_time+time<=maxTime){
                current_time+=time;
                dfs(v);
                current_time-=time;
            }
        }
        //回溯
        vis[u]--;
        if(!vis[u]){
            current_value-=values[u];
        }
    }
```

```
public:
    int maximalPathQuality(vector<int>& values, vector<vector<int>>&
edges, int maxTime) {
        //建图+爆搜
        //初始化
        this->maxTime=maxTime;
        this->values=values;
        int n=values.size();
        Edge=vector<vector<pair<int,int>>>(n);
        vis=vector<int>(n);
        //return n;
        //建图
        for(auto&e:edges){
            Edge[e[0]].emplace_back(e[1],e[2]);
            Edge[e[1]].emplace_back(e[0],e[2]);
        }
        maxValue=current_value=current_time=0;
        dfs(0);
        return maxValue;
    }
};
```

反思总结：

1. 后续方针调整：力扣每日推荐（困难）||力扣每日推荐（简单||中等）+中等题一道
2. 时间限制，简单**20min,**中等**30min,**困呐**1h;**
3. 认真审题--->细致分析--->代码简洁有力
4. 复盘：思维优化-代码优化-学会复杂度分析