总结

1. 前面速度挺快，就是困难的那个不会。。。
2. 困难题居然是暴力、、、以后遇到几何题，直接暴力、、、

1450. 在既定时间做作业的学生人数（简单）

1. 题目描述

   给你两个整数数组 `startTime`（开始时间）和 `endTime`（结束时间），并指定一个整数 `queryTime` 作为查询时间。

   已知，第 `i` 名学生在 `startTime[i]` 时开始写作业并于 `endTime[i]` 时完成作业。

   请返回在查询时间 `queryTime` 时正在做作业的学生人数。形式上，返回能够使 `queryTime` 处于区间 `[startTime[i], endTime[i]]`（含）的学生人数。

   示例 1：

   ```
   输入：startTime = [1,2,3], endTime = [3,2,7], queryTime = 4
   输出：1
   解释：一共有 3 名学生。
   第一名学生在时间 1 开始写作业，并于时间 3 完成作业，在时间 4 没有处于做作业的状态。
   第二名学生在时间 2 开始写作业，并于时间 2 完成作业，在时间 4 没有处于做作业的状态。
   第二名学生在时间 3 开始写作业，预计于时间 7 完成作业，这是是唯一一名在时间 4 时正在做作业的学生。
   ```

   示例 2：

   ```
   输入：startTime = [4], endTime = [4], queryTime = 4
   输出：1
   解释：在查询时间只有一名学生在做作业。
   ```

   示例 3：

   ```
   输入：startTime = [4], endTime = [4], queryTime = 5
   输出：0
   ```

   示例 4：

   ```
   输入：startTime = [1,1,1,1], endTime = [1,3,2,4], queryTime = 7
   输出：0
   ```

   示例 5：

   ```
   输入：startTime = [9,8,7,6,5,4,3,2,1], endTime = [10,10,10,10,10,10,10,10,10],
   queryTime = 5
   输出：5
   ```

提示：

- `startTime.length == endTime.length`
- `1 <= startTime.length <= 100`
- `1 <= startTime[i] <= endTime[i] <= 1000`
- `1 <= queryTime <= 1000`

2. 比赛实现

看数据量只有100，直接暴力

```
```

1451 重新排列句子中的单词（中等）

1. 题目描述

「句子」是一个用空格分隔单词的字符串。给你一个满足下述格式的句子 `text`：

- 句子的首字母大写
- `text` 中的每个单词都用单个空格分隔。

请你重新排列 `text` 中的单词，使所有单词按其长度的升序排列。如果两个单词的长度相同，则保留其在原句子中的相对顺序。

请同样按上述格式返回新的句子。

示例 1：

```
输入：text = "Leetcode is cool"
输出："Is cool leetcode"
解释：句子中共有 3 个单词，长度为 8 的 "Leetcode" ，长度为 2 的 "is" 以及长度为 4 的 "cool"
。
输出需要按单词的长度升序排列，新句子中的第一个单词首字母需要大写。
```

示例 2：

```
输入：text = "Keep calm and code on"
输出："On and keep calm code"
解释：输出的排序情况如下：
"On" 2 个字母。
"and" 3 个字母。
"keep" 4 个字母，因为存在长度相同的其他单词，所以它们之间需要保留在原句子中的相对顺序。
"calm" 4 个字母。
"code" 4 个字母。
```

示例 3：

```
输入：text = "To be or not to be"
输出："To be or to be not"
```

提示：

- `text` 以大写字母开头，然后包含若干小写字母以及单词间的单个空格。
- `1 <= text.length <= 10^5`

2. 比赛实现

stl中stable_sort可以实现稳定排序

```
class Solution {
public:
    void parse(string s, vector<string>& v){//把s中以单词分隔的各个单词解析到v中
        int r = 0;
        string cur = "";
        while(r < s.size()){
            if(s[r] == ' '){
                v.push_back(cur);
                cur = "";
            }
            else
                cur += s[r];
            r++;
        }
        v.push_back(cur);
    }
    static bool cmp(const string& a, const string& b){//按长度排序
        return a.size() < b.size();
    }
    string arrangeWords(string text) {
        if(text == "") return "";
        text[0] = text[0] - 'A' + 'a';//首字母变小写
        vector<string> v;
        parse(text, v);
        stable_sort(v.begin(), v.end(), cmp);//稳定排序
        string ans = "";
        v[0][0] = v[0][0] - 'a' + 'A';//首字母变大写
        for(int i = 0; i < v.size()-1; i++)
            ans += v[i] + ' ';
        ans += v.back();
        return ans;
    }
};
```

1452. 收藏清单（中等）

1. 题目描述

给你一个数组 `favoriteCompanies` ，其中 `favoriteCompanies[i]` 是第 `i` 名用户收藏的公司清单（**下标从 0 开始**）。

请找出不是其他任何人收藏的公司清单的子集的收藏清单，并返回该清单下标。下标需要按升序排列。

示例 1：

```
输入：favoriteCompanies = [["leetcode","google","facebook"],["google","microsoft"],
["google","facebook"],["google"],["amazon"]]
输出：[0,1,4]
解释：
favoriteCompanies[2]=["google","facebook"] 是 favoriteCompanies[0]=
["leetcode","google","facebook"] 的子集。
favoriteCompanies[3]=["google"] 是 favoriteCompanies[0]=
["leetcode","google","facebook"] 和 favoriteCompanies[1]=["google","microsoft"] 的子
集。
其余的收藏清单均不是其他任何人收藏的公司清单的子集，因此，答案为 [0,1,4] 。
```

示例 2：

```
输入：favoriteCompanies = [["leetcode","google","facebook"],["leetcode","amazon"],
["facebook","google"]]
输出：[0,1]
解释：favoriteCompanies[2]=["facebook","google"] 是 favoriteCompanies[0]=
["leetcode","google","facebook"] 的子集，因此，答案为 [0,1] 。
```

示例 3：

```
输入：favoriteCompanies = [["leetcode"],["google"],["facebook"],["amazon"]]
输出：[0,1,2,3]
```

提示：

- `1 <= favoriteCompanies.length <= 100`
- `1 <= favoriteCompanies[i].length <= 500`
- `1 <= favoriteCompanies[i][j].length <= 20`
- `favoriteCompanies[i]` 中的所有字符串 **各不相同** 。
- 用户收藏的公司清单也 **各不相同** ，也就是说，即便我们按字母顺序排序每个清单，`favoriteCompanies[i] != favoriteCompanies[j]` 仍然成立。
- 所有字符串仅包含小写英文字母。

2. 比赛实现

见注释吧还是

```
class Solution {
public:
    vector<int> peopleIndexes(vector<vector<string>>& favoriteCompanies) {
        int size = favoriteCompanies.size();
        if(size == 1) return {0};
        unordered_map<string, unordered_set<int>> m;//<单词，包含该单词的用户下标们>
        for(int i = 0; i < size; i++)//初始化m
            for(int j = 0; j < favoriteCompanies[i].size(); j++)
                m[favoriteCompanies[i][j]].insert(i);
        vector<int> ans;
        for(int i = 0; i < size; i++)//依次遍历各个用户
            int idx = favoriteCompanies[i].size() - 1;
            unordered_set<int> cur = m[favoriteCompanies[i][idx--]];//包含当前用户目
前遍历过的所有公司列表的其他用户下标，即可能使当前用户为其子集的用户们
```

```
            if(cur.size() == 1){//只有自己，说明不是别人的子集
                ans.push_back(i);
                continue;
            }
            while(idx >= 0 && cur.size() > 0){//看每一个公司
                unordered_set<int> tmp;
                for(auto it = cur.begin(); it != cur.end(); it++){
                    if(*it != i && m[favoriteCompanies[i][idx]].find(*it) !=
m[favoriteCompanies[i][idx]].end()){//去掉自己和不符合要求的
                        tmp.insert(*it);
                    }
                }
                cur = tmp;
                idx--;
            }
            if(cur.size() == 0)//不是别人的子集
                ans.push_back(i);
        }
        return ans;
    }
};
```

1453. 圆形靶内的最大飞镖数量（困难）

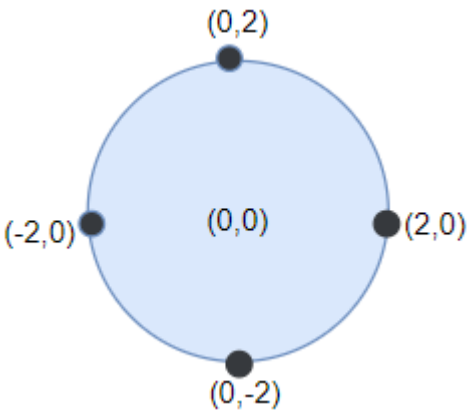
1. 题目描述

墙壁上挂着一个圆形的飞镖靶。现在请你蒙着眼睛向靶上投掷飞镖。

投掷到墙上的飞镖用二维平面上的点坐标数组表示。飞镖靶的半径为 `r` 。

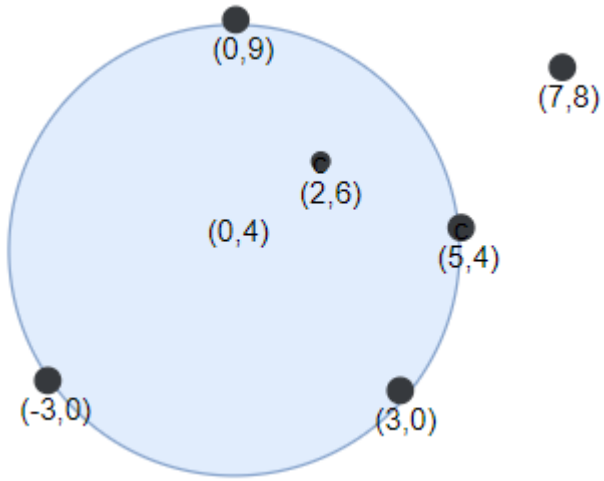
请返回能够落在 **任意** 半径为 `r` 的圆形靶内或靶上的最大飞镖数。

示例 1:




```
输入: points = [[-2,0],[2,0],[0,2],[0,-2]], r = 2
输出: 4
解释: 如果圆形的飞镖靶的圆心为 (0,0) ，半径为 2 ，所有的飞镖都落在靶上，此时落在靶上的飞镖数最大，
值为 4 。
```

示例 2:




```
输入：points = [[-3,0],[3,0],[2,6],[5,4],[0,9],[7,8]], r = 5
输出：5
解释：如果圆形的飞镖靶的圆心为 (0,4) ，半径为 5 ，则除了 (7,8) 之外的飞镖都落在靶上，此时落在靶上的飞镖数最大，值为 5 。
```

示例 3：

```
输入：points = [[-2,0],[2,0],[0,2],[0,-2]], r = 1
输出：1
```

示例 4：

```
输入：points = [[1,2],[3,5],[1,-1],[2,3],[4,1],[1,3]], r = 2
输出：4
```

提示：

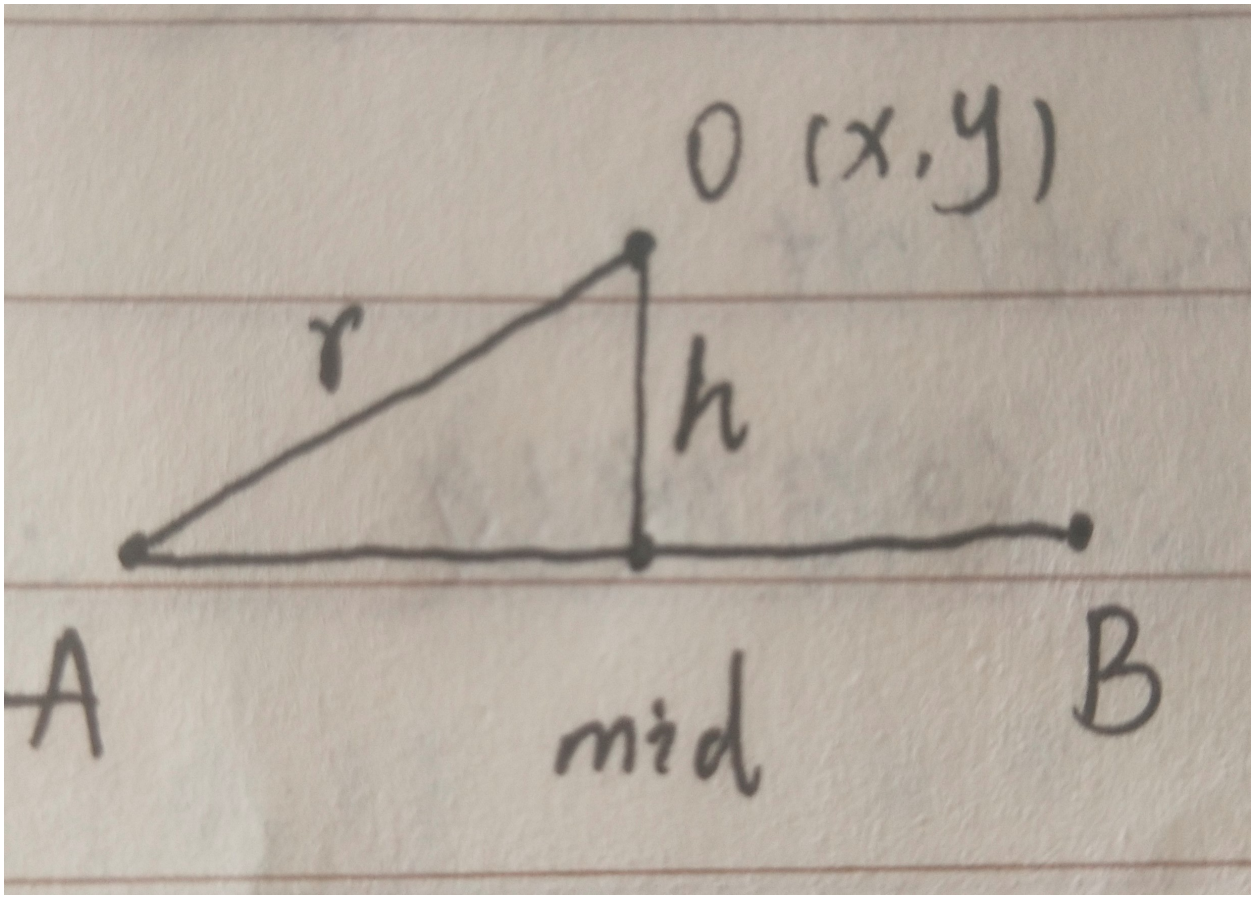
- `1 <= points.length <= 100`
- `points[i].length == 2`
- `-10^4 <= points[i][0], points[i][1] <= 10^4`
- `1 <= r <= 5000`

2. 正确解法

比赛时候还是想复杂了，只要两点一对暴力循环判断即可

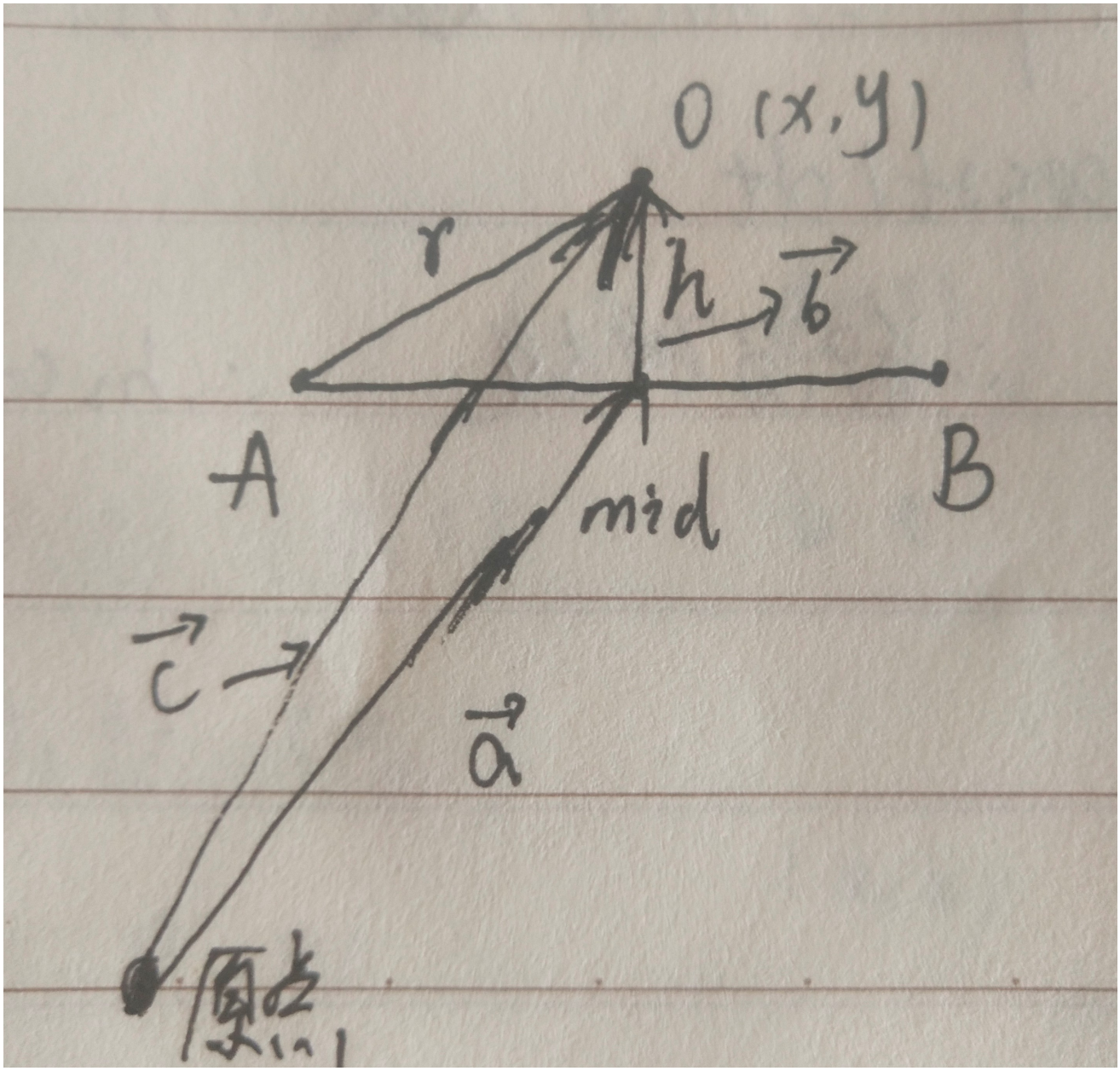
半径大于0，能覆盖的点数量至少为1. 如果圆能够覆盖两个以上的点，必定存在至少两个点在边界上的圆能够覆盖相同数量的点。
给定半径和两个不重合的点，能够确定两个圆心，枚举所有的圆心，找到最优解。

计算圆心 先给一张图：



给定A(x1,y1) B(x2,y2) 以及圆心r，首先就可以直接计算出垂线长度h和mid坐标(AB中点)以及AB长度d:

`d=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));` `h=sqrt(r*r-(d/2.0)*(d/2.0))` `mid=((x1+x2)/2.0,(y1+y2)/2.0)` 然后我们的目的是求O(x,y)，我们使用向量。 看这个图:



向量a+向量b=向量c 毫无疑问 向量a就是mid坐标，向量b就是AB垂线的单位方向向量乘以高度h，向量c就是O坐标

所以现在唯一的问题就在于如何计算AB垂线的方向向量 `向量AB=(x3,y3) 垂线的向量即为(-y3,x3)和(y3,-x3)` 点积为0

特殊情况，AB长度大于2*r (d>2r)，此时不存在圆心

```
struct point{
    double x,y;
    point(double i,double j):x(i),y(j){}
};

//算两点距离
double dist(double x1,double y1,double x2,double y2){
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
```

```
    //计算圆心
    point f(point& a,point& b,int r){
        //算中点
        point mid((a.x+b.x)/2.0,(a.y+b.y)/2.0);
        //AB距离的一半
        double d=dist(a.x,a.y,mid.x,mid.y);
        //计算h
        double h=sqrt(r*r-d*d);
        //计算垂线
        point ba(b.x-a.x,b.y-a.y);
        point hd(-ba.y,ba.x);
        double len=sqrt(hd.x*hd.x+hd.y*hd.y);
        hd.x/=len,hd.y/=len;
        hd.x*=h,hd.y*=h;
        return point(hd.x+mid.x,hd.y+mid.y);
    }

    class Solution {
    public:
        int numPoints(vector<vector<int>>& points, int r) {
            int n=points.size();
            int ans=0;
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    if(i==j){//一个点
                        int cnt=0;
                        for(int k=0;k<n;k++){
                            double tmp=dist(points[i][0],points[i][1],points[k]
[0],points[k][1]);
                            if(tmp<=r) cnt++;
                        }
                        ans=max(cnt,ans);
                    }else{//两个点
                        //通过长度判断有没有圆心
                        double d=dist(points[i][0],points[i][1],points[j][0],points[j]
[1]);
                        if(d/2>r) continue;

                        point a(points[i][0],points[i][1]),b(points[j][0],points[j][1]);
                        point res=f(a,b,r);
                        int cnt=0;
                        for(int k=0;k<n;k++){
                            double tmp=dist(res.x,res.y,points[k][0],points[k][1]);
                            if(tmp<=r) cnt++;
                        }
                        ans=max(cnt,ans);
                    }
                }
            }
            return ans;
        }
    };
```