Aegleseeker

题目描述

记 $I_n=\{1,2,\dots,n\}$ ，则 I_n^3 成为 $\mathbf{R}_n^3$ 的第一卦限中各维坐标均不超过 n 的整点构成的集合。

给定 n 个 I_n^3 中**互不相同**的整点$P_1,P_2,\dots,P_n$ ，试在I_n^3 中找到与 $P_1,P_2,\dots,P_n$ **均不重合**的整点A ，使得对于任意$1\le i<j\le n$ 有 A,P_i,P_j 三点不共线。

若满足条件的 A 存在，则输出其坐标。否则输出 -1 。

输入

本题测试点包含多组数据

第一行，一个正整数 T $(1\le T\le 100)$ ，表示数据组数。

对于每组数据：

第一行，一个正整数 n $(1\le n\le 1000)$ ，表示整点个数。

接下来 n 行，每行三个正整数 x_i,y_i,z_i $(1\le x_i\le n, 1\le y_i\le n, 1\le z_i\le n)$ ，表示给定的整点 P_i 。

对于每个测试点，保证有 $\sum n\le 5000$ 。

输出

对于每组数据：

若满足条件的 A 存在，则输出三个正整数 x_A,y_A,z_A ，表示 A 的坐标。否则输出一个整数 -1 。

若有多个 A 满足条件，输出任意一个即可。

分析

考虑两个朴素的做法：找出给定的 P_i 两两组成的所有直线，在给定的空间域中标记被它们占据的点，再重新枚举每一个点直到找到符合题意的点；枚举每一个点，对于这些点枚举所有直线判断是否相交。

本题数据范围较大，n 的大小高达 1000 。朴素想法的时间复杂度为 $O(n^3)$ ，不可接受。

考虑优化。

对于算法一，每条直线的贡献是不可忽略的，可以认为没有很大优化空间，所以考虑算法二的优化。

枚举所有直线的 $O(n^2)$ 贡献是不可避免的，但是枚举点的 $O(n)$ 贡献还有优化空间，具体体现为我们可以随机选取空间中 m 个点进行检验，这样复杂度降为 $O(mn^2)$ ，可以将 n 的数量级降低为 m 的数量级。

当线的覆盖很稀疏时，符合条件的点的数目很大，在空间中取到符合题意的点的概率很可观；

当线的覆盖很稠密时，对于每个检查的点，被检测到不符合题意的开销远小于稀疏的情况。

在 n^3 的空间内，n^2 级别的线能够覆盖的点的级别即使达到 90% ，当我们把 m 的值设置到 10^2 级别，出错的概率在 10^{-5} 数量级，在可接受范围之内。

在这样的高容错随机化优化下，如果仍然不能抽取的符合题意的点，可以认为不存在这样的点。

标准算法代码

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>

using namespace std;

const int N = 1005;

int n;
int coord[N][3];
int out[3];

int gcd(int a, int b)
{
    if (a == 0 || b == 0)
        return a + b;
    if (a % b == 0)
        return b;
    return gcd(b, a % b);
}

int check()
{
    for (int d = 0; d < 3; d++)
        if (out[d] < 1 || out[d] > n)
            return -1;
    for (int i = 1; i <= n; i++)
    {
        int cnt = 0;
        for (int d = 0; d < 3; d++)
            if (out[d] == coord[i][d])
                cnt++;
        if (cnt == 3)
            return -1;
    }
    for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
        {
            int px = coord[j][0] - coord[i][0];
            int py = coord[j][1] - coord[i][1];
            int pz = coord[j][2] - coord[i][2];
            int qx = out[0] - coord[i][0];
            int qy = out[1] - coord[i][1];
```

```
            int qz = out[2] - coord[i][2];
            int gp = gcd(gcd(abs(px), abs(py)), abs(pz));
            int gq = gcd(gcd(abs(qx), abs(qy)), abs(qz));
            px /= gp, py /= gp, pz /= gp;
            qx /= gq, qy /= gq, qz /= gq;
            if (px == qx && py == qy && pz == qz)
                return -1;
            if (px == -qx && py == -qy && pz == -qz)
                return -1;
        }
    return 0;
}

int rand(int l, int r)
{
    return rand() % (r - l + 1) + l;
}

void solve()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        for (int d = 0; d < 3; d++)
            scanf("%d", &coord[i][d]);
    for (int i = 1; i <= 100; i++)
    {
        for (int d = 0; d < 3; d++)
            out[d] = rand(1, n);
        if (check())
            continue;
        printf("%d %d %d\n", out[0], out[1], out[2]);
        return;
    }
    printf("-1\n");
}

int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
        solve();
    return 0;
}
```