【题解】2020 牛客 NOIP 赛前集训营-提高组（第五场）

写在前面

作为一个不是特别专业的出题人，之前只在牛客上面出过挑战赛和练习赛，因此出 NOIP 提高组的经验属实欠缺。在出这场题之前一直在思考到底是要出一些超出考纲比较多的题目还是老老实实按照往年 NOIP 的难度来出这套题。

思前想后还是决定去参考往年的 NOIP 难度来出，然后发现提高组难度并不高，所以这套题也是出的有点迷迷糊糊。尤其 T3 在比赛过程中我意识到这道题实在是过于简单，如果把 T4 换到 T3 然后再加一道防 AK 的题目可能这套题看上去会更好，可惜错误的预计了模拟赛的难度和选手的实力，导致这场题的难度变得偏低了。

Problem A. 三元组计数

本来是想出一道数论的套路题，利用整除分块的性质来进行计数，写std的时候发现有特别暴力的写法，考虑到要有一道签到题让大家都能参与比赛，因此决定直接降低难度送温暖。

这题只需要通过调和级数的性质来暴力枚举a和b再来算c的个数即可，复杂度是$O(nlogn)$。

参考代码

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#include<cmath>
using namespace std;
typedef long long ll;
const int maxn = 1e6;
int n;
ll ans;
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        for (int j = 2 * i; j <= n; j += i)
            ans += n / j - 1;
    printf("%lld\n", ans);
    return 0;
}
```

Problem B. K 匹配

这个题是纯粹为了考察大家的字符串知识点，但是$NOIP$的字符串知识点只有$hash$，KMP，$manacher$之类的算法，因此就选择了KMP作为考察点。

首先这个题是我思考的另外一个题的弱化版，我们先来考虑这个题目，给定一个串S和一个长度为k的串T，如果对于S的某个子串，我们假设这个子串的范围是$[l,r]$，设为 S_l,r。如果S_l,r中存在一个长度为k的子串和T相等，那么我们就认为子串S_l,r和串T满足k匹配，现在要问串S有多少个子串和T满足k匹配。

首先暴力的做法就是直接枚举串S的左右端点，然后判这个子串S_l,r是否包含了串T。复杂度是$O(n^2k)$。

现在考虑稍微优化的做法，枚举一个左端点$\boldsymbol{l}$，然后我们需要求满足条件的右端点个数，很显然我们只需要找到第一个满足条件的右端点r并且保证S串的区间$[l,r]$包含串T就行了，那么r以及r之后的每个右端点都是满足条件的，找右端点

我们可以直接对串S做一次哈希，然后枚举着找，这样复杂度是$O(n^2)$的。

考虑到上述枚举左端点，找右端点的做法，很自然的发现可以直接用KMP找出所有满足条件的左端点，然后再去枚举就能直接查询下一个符合条件的位置。注意到串长是 $1e7$，而满足条件的端点是递增的，因此可以用一个指针来标记当前匹配到的位置，然后每次往后移动即可。复杂度是$O(n)$的。

参考代码

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#include<cmath>
#include<vector>
#include<string>

using namespace std;

typedef long long ll;

const int maxn = 1e7;

int Next[maxn + 5], n, k;
char S[maxn + 5], T[maxn + 5];
vector<int> lid;
ll res;

void get_next(char* s) {
    memset(Next, 0, sizeof(Next));
    int pre = 0;
    for (int suf = 1; suf < k; ++suf) {
        while (pre && s[pre] != s[suf])pre = Next[pre];
        pre += (s[pre] == s[suf]), Next[suf + 1] = pre;
    }
}

void KMP(char* t, char* s) {
    int index = 0;
    for (int i = 0; i < n; ++i) {
        while (index && s[index] != t[i]) index = Next[index];
```

```
        index += (s[index] == t[i]);
        if (index == k) {
            lid.push_back(i - k + 2);
            index = Next[index];
        }
    }
}

int main()
{
    scanf("%d %d", &n, &k);
    scanf("%s", S);
    scanf("%s", T);
    get_next(T);
    KMP(S, T);
    if (lid.empty()) {
        printf("0\n");
        return 0;
    }
    for (int i = 1, j = 0; i <= n; ++i) {
        if (lid[j] < i && j + 1 < lid.size())j += 1;
        int r = lid[j] + k - 1;
        if (i > lid[j])break;
        res += n - r + 1;
    }
    printf("%lld\n", res);
    return 0;
```

}考虑一个比较有趣的情况，当串 TT 的长度并不固定为 kk 的时候这个题该怎么做。

Problem C. 经典字符串问题

这个题算是比较失败的产物，本意是一个在$trie$上乱搞的题目，写std的时候发现了简单做法，于是这题变成了相当弱的一道题。

暴力做法就是直接把$[l,r]$区间的所有字符串拿出来$sort$一遍再输出第k个即可，复杂度是 $O(nqlogn)$。

考虑到实际上我们可以对每个串直接排序，因此我们一开始就能拿到每个串的

$rank$，因此这个题只需要对最后的$rank$去建主席树，然后就变成了求区间第k大的简单题，复杂度是 $O(nlogn)$。

参考代码

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
#include<string>

using namespace std;

const int maxn = 1e5;
const int maxm = 20;

int a[maxn + 5], rk[maxn + 5], id[maxn + 5], n, q;
string str[maxn + 5];

struct TreeNode {
    int l = 0, r = 0, sum = 0;
}arr[maxm * maxn + 5];
int tot, T[maxn + 5];

int calval(const string& s) {
    int ans = 0;
    for (int i = 0; i < s.length(); ++i)ans = ans * 10 + s[i] - '0';
    return ans;
}

void build(int& now, int l, int r) {
    if (!now)now = ++tot;
    int mid = l + r >> 1;
    if (l < r) {
        build(arr[now].l, l, mid);
        build(arr[now].r, mid + 1, r);
    }
}

void modify(int& now, int pre, int l, int r, int pos) {
    if (!now)now = ++tot;
    arr[now] = arr[pre];
```

```
    arr[now].sum += 1;
    if (l < r) {
        int mid = l + r >> 1;
        if (mid >= pos) {
            arr[now].l = 0;
            modify(arr[now].l, arr[pre].l, l, mid, pos);
        }
        else {
            arr[now].r = 0;
            modify(arr[now].r, arr[pre].r, mid + 1, r, pos);
        }
    }
}

int query(int Lid, int Rid, int l, int r, int k) {
    if (l >= r)return l;
    int mid = l + r >> 1;
    int sum = arr[arr[Rid].l].sum - arr[arr[Lid].l].sum;
    if (sum >= k)return query(arr[Lid].l, arr[Rid].l, l, mid, k);
    else return query(arr[Lid].r, arr[Rid].r, mid + 1, r, k - sum);
}

int main()
{
    scanf("%d %d", &n, &q);
    for (int i = 1; i <= n; ++i)scanf("%d", &a[i]);
    for (int i = 1; i <= n; ++i)str[i] = to_string(a[i]);
    sort(str + 1, str + 1 + n);
    for (int i = 1; i <= n; ++i) {
        int val = calval(str[i]);
        rk[i] = val;
        id[val] = i;
    }
    for (int i = 1; i <= n; ++i)a[i] = id[a[i]];
    build(T[0], 1, n);
    for (int i = 1; i <= n; ++i)modify(T[i], T[i - 1], 1, n, a[i]);
    for (int i = 1, l, r, k; i <= q; ++i) {
        scanf("%d %d %d", &l, &r, &k);
        if (r - l + 1 < k)printf("-1\n");
        else printf("%d\n", rk[query(T[l - 1], T[r], 1, n, k)]);
    }
    return 0;
```

Problem D. 圆与圆的距离不能一概而论

出这个题的起因是在一次训练中遇到了用圆的扫描线来求这种不存在相交关系的圆的森林。

比较暴力的做法就是直接对每个圆求哪些圆包含了这个圆，再顺势建出森林的关系，甚至可以直接建树后暴力找 lca判段答案。

因为找lca的做法太多了，所以我在数据方面没有考虑去卡树的高度，也是太懒了，其实卡树高度的数据更好构造。

所以一开始我对这题的定位就在于圆的扫描线算法，我们对读进来的圆设置两条边界线分别是$x-r$和$x+r$，那么遇到左端点就是要把这个圆加进去，而遇到右端点则是把这个圆踢出去。现在我们处理出所有的事件，用一条竖直线从左到右扫描所有的事件，处理每个圆“刚接触扫描线”和“刚离开扫描线”两个事件点就行了。当遇到“刚接触扫描线”的事件时，我们设这个事件属于圆x，查询左端点上方的第一个圆弧是下半圆还是上半圆。如果没有发现圆弧，说明没有任何其他圆是包含圆x的，因此圆x属于一棵树的祖先。如果找到了圆弧，我们设第一个找到的圆弧是y，如果是上半圆说明当前圆包含在那个半圆所代表的圆中，这里我们就可以找到一对父子关系，令$fa[x]=y$，如果是下半圆则说明这两个圆是兄弟关系，那么可以发现y的父亲一定是x的父亲，我们可以设$fa[x]=fa[y]$。弧的排序关系用与当前扫描线的交点的纵坐标表示就行了，因为这些圆不存在相交，因此移动扫描线对于已经扫过的弧的相对关系是没有影响的。

后面就是用倍增或者树链剖分去找两个节点的lca，复杂度是$O(nlogn+qlogn)$。

参考代码

```
#include<iostream>
#include<algorithm>
```

```cpp
#include<cstdio>
#include<cmath>
#include<vector>
#include<cstring>
#include<set>

using namespace std;

const double eps = 1e-6;
const int maxn = 1e5;
int nowx;

namespace UnionFindSet {
    int ufs[maxn + 5];
    void init(int n) {
        for (int i = 1; i <= n; ++i)ufs[i] = i;
    }
    int find(int x) {
        return x == ufs[x] ? x : ufs[x] = find(ufs[x]);
    }
    void unite(int x, int y) {
        x = find(x), y = find(y);
        if (x != y)ufs[x] = y;
    }
}

namespace TreeChainSplitting {

    vector<int> g[maxn + 5];

    int parent[maxn + 5], son[maxn + 5], sz[maxn + 5], deep[maxn + 5];
    int top[maxn + 5], id[maxn + 5];
    int tot;

    void init() {
        memset(son, -1, sizeof(son));
        memset(deep, 0, sizeof(deep));
        tot = 0;
    }

    void dfs1(int u, int father) {
        deep[u] = deep[father] + 1, sz[u] = 1, parent[u] = father;
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i];
```

```
            if (v == father)continue;
            dfs1(v, u);
            sz[u] += sz[v];
            if (son[u] == -1 || sz[v] > sz[son[u]])son[u] = v;
        }
    }

    void dfs2(int u, int root) {
        top[u] = root, id[u] = ++tot;
        if (son[u] == -1)return;
        dfs2(son[u], root);
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i];
            if (v != son[u] && v != parent[u])dfs2(v, v);
        }
    }

    int lca(int x, int y) {
        while (top[x] != top[y]) {
            if (deep[top[x]] < deep[top[y]])swap(x, y);
            x = parent[top[x]];
        }
        return deep[x] < deep[y] ? x : y;
    }
}

struct circle {
    circle(int x = 0, int y = 0, int r = 0) :x(x), y(y), r(r) {}
    int x, y, r;
}c[maxn + 5];

struct event{
    int x, type, id;
    event(int x = 0, int type = 0, int ss = 0) : x(x), type(type),
id(ss) {}
}eve[2 * maxn + 5];

bool operator < (const event&a, const event &b) {
    if (a.x == b.x) {
        if (a.type == b.type)return a.id < b.id;
        else return a.type > b.type;
    }
    else return a.x < b.x;
}
```

```
struct node {
    node(int ss = 0, int type = 0) :id(ss), type(type) {}
    int id, type;
};

double get_pos(const node &p) {
    int type = p.type, ss = p.id;
    if (type == 1) return (double)c[ss].y + sqrt((double)c[ss].r *
c[ss].r - (double)(c[ss].x - nowx) * (c[ss].x - nowx));
    if (type == -1)return (double)c[ss].y - sqrt((double)c[ss].r *
c[ss].r - (double)(c[ss].x - nowx) * (c[ss].x - nowx));
}

bool operator < (const node &a, const node &b) {
    double A = get_pos(a), B = get_pos(b);
    return  A > B || fabs(A - B) < eps && a.type > b.type;
}

bool operator == (const node &a, const node &b) { return a.id == b.id
&& a.type == b.type; }

set<node> s;
int fa[maxn + 5], n, q, cnt;

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d %d %d", &c[i].x, &c[i].y, &c[i].r);
        eve[++cnt] = event(c[i].x - c[i].r, 1, i);
        eve[++cnt] = event(c[i].x + c[i].r, -1, i);
    }
    sort(eve + 1, eve + 1 + cnt);
    for (int i = 1; i <= cnt; ++i) {
        int id = eve[i].id;
        nowx = eve[i].x;
        if (eve[i].type == 1) {
            auto up = s.lower_bound(node(id, 1));
            auto down = s.upper_bound(node(id, 1));
            if (up == s.begin() || down == s.end()) {
                fa[id] = 0;
                s.insert(node(id, 1));
                s.insert(node(id, -1));
                continue;
```

```
            }
            --up;
            if (up->id == down->id)fa[id] = up->id;
            else if (fa[up->id] != fa[down->id]) {
                if (fa[up->id] == down->id)    fa[id] = down->id;
                else if (fa[down->id] == up->id)    fa[id] = up->id;
            }
            else fa[id] = fa[up->id];
            s.insert(node(id, 1));
            s.insert(node(id, -1));
        }
        else {
            s.erase(node(id, 1));
            s.erase(node(id, -1));
        }
    }
    UnionFindSet::init(n);
    TreeChainSplitting::init();
    for (int i = 1; i <= n; ++i) {
        if (fa[i] == 0)continue;
        int u = fa[i], v = i;
        TreeChainSplitting::g[u].push_back(v);
        TreeChainSplitting::g[v].push_back(u);
        UnionFindSet::unite(v, u);
    }
    for (int i = 1; i <= n; ++i) {
        if (UnionFindSet::ufs[i] == i) {
            TreeChainSplitting::dfs1(i, i);
            TreeChainSplitting::dfs2(i, i);
        }
    }
    scanf("%d", &q);
    for (int i = 1, u, v; i <= q; ++i) {
        scanf("%d %d", &u, &v);
        int fu = UnionFindSet::find(u), fv = UnionFindSet::find(v);
        if (fu != fv) {
            int d1 = TreeChainSplitting::deep[u], d2 =
TreeChainSplitting::deep[v];
            int res = d1 + d2 - 2;
            printf("%d\n", res);
        }
        else {
            int lca = TreeChainSplitting::lca(u, v);
```

```
            int du = TreeChainSplitting::deep[u], dv =
TreeChainSplitting::deep[v], dlca = TreeChainSplitting::deep[lca];
            int res = du + dv - 2 * dlca - 2;
            if (u == lca || v == lca) res += 1;
            printf("%d\n", res);
        }
    }
    return 0;
}
```