圆的面积并

Union of Circles

Idea：圆的面积并用扫描线比较难处理，下采用自适应 Simpson 积分法。将 $x=x_0$ 处圆覆盖的线段长度作为函数值 $f(x_0)$，这个函数值可以 $O(n\lg n)$ 求得。对这个函数 $f(x)$ 在整个区间内作 Simpson 积分即可。但是当圆的分布较为稀疏时，函数 $f(x)$ 可能有大量的 0 点，在自适应 Simpson 积分时可能不会递归求解下去。所以我们选取一系列有圆覆盖的区连续间分段积分。

一些优化：小圆在大圆内部先删去；单独一个没有与其他圆有交的圆先算出答案并删去。

Code：

```
#include<algorithm>
#include<cstdio>
#include<cmath>

using namespace std;

const double eps = 1e-8;
const double PI = acos(-1);
const double INF = 1e16;
inline int sgn(double x){
    if(fabs(x) < eps)   return 0;
    else if(x > 0)  return 1;
    else    return -1;
}
inline int cmp(double x, double y){
    if(fabs(x-y) < eps) return 0;
    else if(x > y)  return 1;
    else    return -1;
}

struct Vector{
    double x, y;
    Vector(double x = 0, double y = 0):x(x), y(y){}
    void read(){ scanf("%lf%lf", &x, &y); }
};
typedef Vector Point;
Vector operator + (Vector A, Vector B){ return Vector{A.x + B.x, A.y + B.y}; }
Vector operator - (Vector A, Vector B){ return Vector{A.x - B.x, A.y - B.y}; }
double operator * (Vector A, Vector B){ return A.x * B.x + A.y * B.y; } // dot product
double Length(Vector A){ return sqrt(A * A); }
struct Line{
    Point p; Vector v;
};
struct Circle{
    Point p;
    double r;
    bool operator < (const Circle &C) const{ return r > C.r; }
};

// ------------------------------------------------------------------------------- //

const int N = 1005;

int n;
Circle c[N];
bool b[N];
double ans;

```

```
struct Segment{
    double l, r;
    bool operator < (const Segment &A) const{ return l < A.l; }
}a[N], seg[N];

double f(double x){
    int id = 0;
    for(int i = 1; i <= n; i++){
        double d = c[i].r * c[i].r - (c[i].p.x - x) * (c[i].p.x - x);
        if(sgn(d) <= 0) continue;
        d = sqrt(d);
        a[++id] = (Segment){c[i].p.y - d, c[i].p.y + d};
    }
    sort(a+1, a+id+1);
    double res = 0, pre = -1e9;
    for(int i = 1; i <= id; i++){
        if(a[i].l > pre)    res += a[i].r - a[i].l, pre = a[i].r;
        else if(a[i].r > pre)   res += a[i].r - pre, pre = a[i].r;
    }
    return res;
}
double simpson(double l, double r){
    double mid = (l + r) / 2;
    return (f(l) + 4 * f(mid) + f(r)) * (r - l) / 6;
}
double solve(double l, double r, double _eps){
    double mid = (l + r) / 2;
    double Il = simpson(l, mid), Ir = simpson(mid, r), I = simpson(l, r);
    if(fabs(Il + Ir - I) <= 15 * _eps)   return Il + Ir + (Il + Ir - I) / 15;
    return solve(l, mid, _eps / 2) + solve(mid, r, _eps / 2);
}

int main(){
    scanf("%d", &n);
    for(int i = 1; i <= n; i++)
        scanf("%lf%lf%lf", &c[i].p.x, &c[i].p.y, &c[i].r);

    sort(c+1, c+n+1);
    int cid = 0;
    for(int i = 1; i <= n; i++){
        if(sgn(c[i].r) == 0)    continue;
        bool in = false;
        for(int j = 1; j <= cid; j++){
            if(cmp(Length(c[j].p - c[i].p), c[j].r - c[i].r) <= 0){
                in = true;
                break;
            }
        }
        if(!in) c[++cid] = c[i];
    }
    n = cid; cid = 0;

    for(int i = 1; i <= n; i++)
        for(int j = 1; j < i; j++)
            if(cmp(Length(c[i].p - c[j].p), c[i].r + c[j].r) < 0)
                b[i] = b[j] = 1;
    for(int i = 1; i <= n; i++)
        if(!b[i])   ans += PI * c[i].r * c[i].r;
        else    c[++cid] = c[i];
    n = cid;

    int id = 0;
    for(int i = 1; i <= n; i++)
        seg[++id] = (Segment){c[i].p.x - c[i].r, c[i].p.x + c[i].r};
    sort(seg+1, seg+id+1);
    double pre = -1e9;
    for(int i = 1; i <= id; i++){
        if(seg[i].l > pre)  ans += solve(seg[i].l, seg[i].r, 1e-5), pre = seg[i].r;
        else if(seg[i].r > pre) ans += solve(pre, seg[i].r, 1e-5), pre = seg[i].r;
    }
    printf("%.3f\n", ans);
    return 0;
}
```