

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:

```
1  #include<algorithm>
2  #include<cstdio>
3  #include<cmath>
4
5  using namespace std;
6
7  const double eps = 1e-8;
8  const double PI = acos(-1);
9  const double INF = 1e16;
10 inline int sgn(double x){
11     if(fabs(x) < eps) return 0;
12     else if(x > 0) return 1;
13     else return -1;
14 }
15 inline int cmp(double x, double y){
16     if(fabs(x-y) < eps) return 0;
17     else if(x > y) return 1;
18     else return -1;
19 }
20
21 struct Vector{
22     double x, y;
23     Vector(double x = 0, double y = 0):x(x), y(y){}
24     void read(){ scanf("%lf%lf", &x, &y); }
25 };
26 typedef Vector Point;
27 Vector operator + (Vector A, Vector B){ return Vector{A.x + B.x, A.y + B.y}; }
28 Vector operator - (Vector A, Vector B){ return Vector{A.x - B.x, A.y - B.y}; }
29 double operator * (Vector A, Vector B){ return A.x * B.x + A.y * B.y; } // dot product
30 double Length(Vector A){ return sqrt(A * A); }
31 struct Line{
32     Point p; Vector v;
33 };
34 struct Circle{
35     Point p;
36     double r;
37     bool operator < (const Circle &C) const{ return r > C.r; }
38 };
39
40 // ----- //
41
42 const int N = 1005;
43
44 int n;
45 Circle c[N];
46 bool b[N];
47 double ans;
48
49 struct Segment{
50     double l, r;
51     bool operator < (const Segment &A) const{ return l < A.l; }
52 }a[N], seg[N];
53
54 double f(double x){
55     int id = 0;
56     for(int i = 1; i <= n; i++){
57         double d = c[i].r * c[i].r - (c[i].p.x - x) * (c[i].p.x - x);
58         if(sgn(d) <= 0) continue;
59         d = sqrt(d);
60         a[++id] = (Segment){c[i].p.y - d, c[i].p.y + d};
```

```

61     }
62     sort(a+1, a+id+1);
63     double res = 0, pre = -1e9;
64     for(int i = 1; i <= id; i++){
65         if(a[i].l > pre)    res += a[i].r - a[i].l, pre = a[i].r;
66         else if(a[i].r > pre)    res += a[i].r - pre, pre = a[i].r;
67     }
68     return res;
69 }
70 double simpson(double l, double r){
71     double mid = (l + r) / 2;
72     return (f(l) + 4 * f(mid) + f(r)) * (r - l) / 6;
73 }
74 double solve(double l, double r, double _eps){
75     double mid = (l + r) / 2;
76     double Il = simpson(l, mid), Ir = simpson(mid, r), I = simpson(l, r);
77     if(fabs(Il + Ir - I) <= 15 * _eps)    return Il + Ir + (Il + Ir - I) / 15;
78     return solve(l, mid, _eps / 2) + solve(mid, r, _eps / 2);
79 }
80
81 int main(){
82     scanf("%d", &n);
83     for(int i = 1; i <= n; i++)
84         scanf("%lf%lf%lf", &c[i].p.x, &c[i].p.y, &c[i].r);
85
86     sort(c+1, c+n+1);
87     int cid = 0;
88     for(int i = 1; i <= n; i++){
89         if(sgn(c[i].r) == 0)    continue;
90         bool in = false;
91         for(int j = 1; j <= cid; j++){
92             if(cmp(Length(c[j].p - c[i].p), c[j].r - c[i].r) <= 0){
93                 in = true;
94                 break;
95             }
96         }
97         if(!in) c[++cid] = c[i];
98     }
99     n = cid; cid = 0;
100
101     for(int i = 1; i <= n; i++)
102         for(int j = 1; j < i; j++)
103             if(cmp(Length(c[i].p - c[j].p), c[i].r + c[j].r) < 0)
104                 b[i] = b[j] = 1;
105     for(int i = 1; i <= n; i++)
106         if(!b[i])    ans += PI * c[i].r * c[i].r;
107     else    c[++cid] = c[i];
108     n = cid;
109
110     int id = 0;
111     for(int i = 1; i <= n; i++)
112         seg[++id] = (Segment){c[i].p.x - c[i].r, c[i].p.x + c[i].r};
113     sort(seg+1, seg+id+1);
114     double pre = -1e9;
115     for(int i = 1; i <= id; i++){
116         if(seg[i].l > pre)    ans += solve(seg[i].l, seg[i].r, 1e-5), pre = seg[i].r;
117         else if(seg[i].r > pre)    ans += solve(pre, seg[i].r, 1e-5), pre = seg[i].r;
118     }
119     printf("%.3f\n", ans);
120     return 0;
121 }

```