增加

fantasy

目录

1 圆的面积并

```
/*
 * 圆的面积并模板  v1.0
 */
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;

const double pi = acos(-1);

const double eps = 1e-8;
int dcmp(double x) {
	if(fabs(x) < eps) return 0;
	else return x < 0 ? -1 : 1;
}

struct Point {
	double x, y;
	Point(double x=0, double y=0):x(x),y(y) { }
};

typedef Point Vector;
```

```
Vector operator + (const Vector& A, const Vector& B) { return Vector(A.x+B.x, A.y+B.y); }
Vector operator - (const Point& A, const Point& B) { return Vector(A.x-B.x, A.y-B.y); }
Vector operator * (const Vector& A, double p) { return Vector(A.x*p, A.y*p); }
bool operator == (const Point& a, const Point &b) { return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0; }
double Dot(const Vector& A, const Vector& B) { return A.x*B.x + A.y*B.y; }
double Length(const Vector& A) { return sqrt(Dot(A, A)); }
double Angle(const Vector& A, const Vector& B) { return acos(Dot(A, B) / Length(A) / Length(B)); }
double Cross(const Vector& A, const Vector& B) { return A.x*B.y - A.y*B.x; }
double angle(Vector v) { return atan2(v.y, v.x); }

struct Circle {
	Point c;
	double r;
	Circle(Point c = Point(), double r = 0):c(c),r(r) {}
	Point point(double a) {
		return Point(c.x + cos(a)*r, c.y + sin(a)*r);
	}
};
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
	double d = Length(C1.c - C2.c);
	if(dcmp(d) == 0) {
		if(dcmp(C1.r - C2.r) == 0) return -1; // 重合，无穷多交点
		return 0;
	}
	if(dcmp(C1.r + C2.r - d) < 0) return 0;
	if(dcmp(fabs(C1.r-C2.r) - d) > 0) return 0;

	double a = angle(C2.c - C1.c);
	double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*C1.r*d));
	Point p1 = C1.point(a-da), p2 = C1.point(a+da);

	sol.push_back(p1);
	if(p1 == p2) return 1;
	sol.push_back(p2);
	return 2;
}

namespace Circle_Union{
	const int maxn = 1005;

	int n;
	Circle c[maxn];
	int vis[maxn];
	double ans[maxn];
	struct Node{
		double ang;
		int kind;
		Node(double ang = 0, int kind = 0):ang(ang), kind(kind){}
		friend bool operator < (const Node &a, const Node &b){
			if(a.ang == b.ang) return a.kind > b.kind;
			return a.ang < b.ang;
		}
	};
	void work(int x){
		for(int i = 1; i <= n; i++) if( i != x ){
			if(Length(c[i].c - c[x].c) + c[x].r <= c[i].r) {
				vis[x]++;
			}
		}
		vector <Node> vec; vec.clear();
		vector <Point> sol;
```

```
		for(int i = 1; i <= n; i++) if( i != x ){
			sol.clear();
			getCircleCircleIntersection(c[x], c[i], sol);
			if(sol.size() < 2) continue;
			if(angle(sol[1] - c[x].c) < angle(sol[0] - c[x].c)){
				vec.push_back(Node(pi, -1));
				vec.push_back(Node(-pi, 1));
			}
			vec.push_back(Node(angle(sol[0] - c[x].c), 1));
			vec.push_back(Node(angle(sol[1] - c[x].c), -1));
		}
		vec.push_back(Node(-pi, 1));
		vec.push_back(Node(pi, -1));
		sort(vec.begin(), vec.end());
		Node last = vec[0]; int cnt = 0;
		for(int i = 0; i < vec.size(); i++){
			ans[cnt + vis[x]] += 0.5 * Cross(c[x].point(last.ang) - Point(0,0), c[x].point(vec[i].ang) - Point(0,0));
			double del = vec[i].ang - last.ang;
			ans[cnt + vis[x]] += 0.5 * c[x].r * c[x].r * (del - sin(del));
			last = vec[i];
			cnt += vec[i].kind;
		}
	}
	void init(){
		memset(vis, 0, sizeof(vis));
		memset(ans, 0, sizeof(ans));
	}
	void WORK(){ for(int i = 1; i <= n; i++) work(i); }
};

//useage
//调用init()
//输入n个圆c[1...n]
//调用WORK()
//ans[i] 表示被覆盖i次及以上的面积。
```

2 点双连通分量

```
const int maxn = 20005;
const int maxm = 100005;
int n, m;
struct Data{
	int from, to, next, id;
}data[maxm * 2];
int head[maxn];
int tot;
void insert(int a, int b, int id){
	tot++;
	data[tot].to = b;
	data[tot].from = a;
	data[tot].next = head[a];
	data[tot].id = id;
	head[a] = tot;
}
int pre[maxn], lowpt[maxn], time_flag;
stack <int> st;
void Vbcc(int v, int u){
```

```
        pre[v] = lowpt[v] = ++ time_flag;
        for(int k = head[v]; k; k = data[k].next){
                int w = data[k].to;
                if(!pre[w]){
                        st.push(k);
                        Vbcc(w, v);
                        lowpt[v] = min(lowpt[v], lowpt[w]);
                        if(lowpt[w] >= pre[v]){
                                while(!st.empty() && pre[data[st.top()].from] >= pre[w]){
                                        // do what you want to do here
                                        st.pop();
                                }
                                // do what you want to do here
                                st.pop();
                        }
                } else if(pre[w] < pre[v] && w != u){
                        st.push(k);
                        lowpt[v] = min(lowpt[v], pre[w]);
                }
        }
}
```

3 边双连通分量

```
const int maxn = 20005;
const int maxm = 100005;
int n, m;
struct Data{
        int from, to, next;
}data[maxm * 2];
int head[maxn];
int tot;
void insert(int a, int b){
        tot++;
        data[tot].to = b;
        data[tot].from = a;
        data[tot].next = head[a];
        head[a] = tot;
}

int pre[maxn], lowpt[maxn], time_flag;
stack <int> st;
int ins[maxn];
void Ebcc(int v, int u){
        pre[v] = lowpt[v] = ++time_flag;
        st.push(v); ins[v] = 1;
        for(int k = head[v]; k; k = data[k].next){
                int w = data[k].to;
                if(!pre[w]){
                        Ebcc(w, v);
                        lowpt[v] = min(lowpt[v], lowpt[w]);
                } else if(pre[w] < pre[v] && ins[w] && w != u){
                        lowpt[v] = min(lowpt[v], pre[w]);
                }
        }
        if(lowpt[v] == pre[v]){
                while(!st.empty() && st.top() != v){
                        //do what you want to do here
```

```
            ins[st.top()] = 0;
            st.pop();
        }
        //do what you want to do here
        ins[st.top()] = 0;
        st.pop();
    }
}
```

4 FFT

```
/*
 * FFT 模板v1.0, 注意调用时, 先需要保证 n = 2^k, 并且搞好初始值。
 * 调用DFT()进行FFT、调用DFT_h()进行逆运算（插值）。
 * A数组: [0,n)
 */
#include <iostream>
#include <cstring>
#include <cstdio>
#include <complex>
#include <cmath>
using namespace std;
namespace FFT{
    const double pi = acos(-1);
    int inv_flag = 1;
    complex<double> omega(int n){
        double u = 2 * pi/ n * inv_flag;
        return complex<double>(cos(u),sin(u));
    }
    inline void BitReverseCopy(complex<double> A[], int n, int k){
        for(int i = 0; i < n; i++){
            int t = 0;
            for(int j = 0; j < k; j++) if(i & (1<<j)){ // 二进制位[0,k)
                t |= 1 << (k - j - 1);
            }
            if(t > i) swap(A[i],A[t]);
        }
    }
    inline void DFT(complex<double> A[], int &n){  // 这个算法会修改n, 不过这样正好
        int k = 0; while((1LL<<k) < n) k++;
        for(int i = n, ed = 1LL << k; i < ed; i++) A[i] = 0;
        n = 1LL << k;

        BitReverseCopy(A,n,k);
        complex<double> wm,w,t,u;
        for(int s = 1; s <= k; s++){
            int m = 1 << s;
            wm = omega(m);
            for(int k = 0; k < n; k += m){
                w = 1;
                for(int j = 0, md = m/2; j < md; j++){
                    t = w * A[k+j+md], u = A[k+j];
                    A[k+j] = u + t, A[k+j+md] = u - t;
                    w *= wm;
                }
            }
        }
    }
```

```
        inline void DFT_h(complex<double> A[], int n){
                inv_flag = -1;
                DFT(A,n);
                inv_flag = 1;
                for(int i = 0; i < n; i++) A[i] /= n;
        }
}
```

5 Millar Rabin and Rollard Rho

```
/*
 *  Millar Rabin &  Pollard Rho 模板 v1.0
 */
#include <cstring>
#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>
#include <ctime>
using namespace std;
typedef long long LL;
LL GCD, LCM;
inline LL multiply_mod(LL a, LL b, LL mod){
        LL ans = 0;
        while(b){
                if(b & 1){
                        ans += a; ans %= mod;
                }
                a += a; a %= mod; b >>= 1;
        }
        return ans;
}
inline LL pow_mod(LL x, LL n, LL mod){
        x %= mod;
        LL ans = 1;
        while(n){
                if(n & 1){
                        ans = multiply_mod(ans, x, mod); ans %= mod;
                }
                x = multiply_mod(x, x, mod); x %= mod; n >>= 1;
        }
        return ans;
}
inline LL add_mod(LL a, LL b, LL mod){
        return (a + b) % mod;
}
LL gcd(LL a, LL b){
        if(b == 0) return a;
        return gcd(b, a%b);
}

int prime[10000005];
int not_prime[10000005];
int cnt_prime;
const int PRIME_LIM = 10000000;
inline void init_prime(){
        not_prime[1] = 1;
        for(int i = 2; i <= PRIME_LIM; i++){
```

```cpp
        if(!not_prime[i]) prime[++cnt_prime] = i;
        for(int k = 1; k <= cnt_prime; k++){
            if(prime[k] * i > PRIME_LIM) break;
            not_prime[prime[k] * i] = 1;
            if(i % prime[k] == 0) break;
        }
    }
}
inline bool test(LL n, LL b){
    LL m = n - 1;
    LL counter = 0;
    while(~m & 1){
        m >>= 1;
        counter++;
    }
    LL ret = pow_mod(b, m, n);
    if(ret == 1 || ret == n - 1){
        return true;
    }
    counter --;
    while(counter >= 0){
        ret = multiply_mod(ret, ret, n);
        if(ret == n - 1){
            return true;
        }
        counter --;
    }
    return false;
}
const int BASE[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43};
inline bool is_prime(LL n){
    if(n < 2) return false;
    if(n < 4) return true;
    if((n & 1) == 0) return false;
    if(n == 3215031751LL){
        return false;
    }
    for(int i = 0; i < 14 && BASE[i] < n; ++i){
        if(!test(n, BASE[i])){
            return false;
        }
    }
    return true;
}
LL pollard_rho(LL n, LL seed){
    LL x, y, head = 1, tail = 2;
    x = y = rand() % (n - 1) + 1;
    while(1){
        x = multiply_mod(x, x, n);
        x = add_mod(x, seed, n);
        if(x == y){
            return n;
        }
        LL d = gcd(abs(x - y), n);
        if(1 < d && d < n){
            return d;
        }
        head ++;
        if(head == tail){
            y = x;
            tail <<= 1;
        }
```

```
	}
}
LL divisors[1005], mul[1005];
int tot;
void factorize(LL n){
	if(n > 1){
		if(is_prime(n)){
			divisors[++tot] = n;
		} else {
			cerr << n << endl;
			LL d = n;
			while(d >= n){
				d = pollard_rho(n, rand() % (n - 1) + 1);
			}
			factorize(n / d);
			factorize(d);
		}
	}
}
void Factorize(LL n){
	tot = 0;
	for(int i = 1; i <= cnt_prime && prime[i] <= n; i++){
		while(n % prime[i] == 0){
			divisors[++tot] = prime[i];
			n /= prime[i];
		}
	}
	factorize(n);
}

//注意 以上 模板要使用小素数先判断之后才能继续使用。
```

6 模方程

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <stack>
#include <vector>
using namespace std;
typedef long long LL;
namespace ModEquation{
void exgcd(LL a, LL b, LL &x, LL &y){ // calc: ax + by == gcd(a, b)
	if(b == 0){ x = 1, y = 0; return; }
	exgcd(b, a % b, y, x); y -= a / b * x;
}

LL gcd(LL a, LL b){ // clac GCD( a, b )
	return b == 0 ? a : gcd(b, a % b);
}
bool mod_equation(LL a, LL &x, LL b, LL m) { // calc: ax == b ( mod m ) -> x % m (最小正整数解)
	b = ((b % m) + m) % m;
	LL g = gcd(a, m);
	if(b % g != 0) return false;
	LL y; exgcd(a, m, x, y);
	LL md = m/g;
	x = ((x % md) + md) % md;
	x %= m;
```

```cpp
        x = x * (b/g % m) % m;
        return true;
}
bool mod_equation(LL a, vector <LL> &ans, LL b, LL m){ // 可以算出模域内所有可能的解
        ans.clear(); LL x;
        b = ((b % m) + m) % m;
        LL g = gcd(a, m);
        if(b % g != 0) return false;
        LL y; exgcd(a, m, x, y);
        LL md = m/g;
        x = ((x % md) + md) % md;
        x %= m; x = x * (b/g % m) % m;
        for(int i = 0; i < g; i++){
                ans.push_back(x);
                x = (x + md) % m;
        }
        return true;
}

// return the minmum non-negetive x


/*
        clac:
        / x == b1 (mod m1)
        | x == b2 (mod m2)
        |  ...
        \ x == bn (mod mn)
        return the minmum non-negetive x
*/

struct Equation{
        LL b, m;
        Equation(LL b = 0, LL m = 0):b(b), m(m){}
};
stack <Equation> st;
LL mod_equation_vector(){
        while(st.size() >= 2){
                Equation e1 = st.top(); st.pop();
                Equation e2 = st.top(); st.pop();
                LL g = gcd(e1.m, e2.m);
                LL k1;
                if((e2.b - e1.b) % g == 0 && mod_equation(e1.m/g, k1, (e2.b - e1.b)/g, e2.m/g)){
                        LL m = e1.m / g * e2.m;
                        k1 = ((k1 % m) + m)%m;
                        LL b = (k1 * e1.m) % m + e1.b;
                        b = ((b % m) + m) % m;
                        st.push(Equation(b, m));
                } else {
                        return -1;
                }
        }
        LL x; Equation e = st.top();
        mod_equation(1, x, e.b, e.m);
        return x;
}
};
```

7 NTT

```cpp
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
typedef long long LL;
namespace NTT{
	const LL P = 998244353;// == 2^23 * 7 * 17 + 1
	const LL G = 3; // G^Phi(P) == 1 (mod P)
	LL g = 3;

	inline LL pow_mod(LL x, LL n, LL mod){
		LL ans = 1;
		while(n){
			if(n & 1){ ans *= x; ans %= mod; }
			x *= x; x %= mod; n >>= 1;
		}
		return ans;
	}
	inline LL inv(LL x){
		return pow_mod(x, P-2, P);
	}
	inline LL omega(int n){
		LL u = (P-1) / n;
		return pow_mod(g, u, P);
	}
	inline void BitReverseCopy(LL A[], int n, int k){
		for(int i = 0; i < n; i++){
			int t = 0;
			for(int j = 0; j < k; j++) if(i & (1<<j)) // 二进制位[0,k)
				t |= 1 << (k - j - 1);
			if(t > i) swap(A[i],A[t]);
		}
	}
	inline void DFT(LL A[], int &n){
		int k = 0; while((1LL<<k) < n) k++;
		for(int i = n, ed = 1LL << k; i < ed; i++) A[i] = 0;
		n = 1LL << k;
		BitReverseCopy(A,n,k);
		LL wm,w,t,u;
		for(int s = 1; s <= k; s++){
			int m = 1 << s;
			wm = omega(m);
			for(int k = 0; k < n; k += m){
				w = 1;
				for(int j = 0, md = m/2; j < md; j++){
					t = w * A[k+j+md] % P, u = A[k+j];
					A[k+j] = u + t, A[k+j+md] = u - t + P;
					A[k+j] -= A[k+j] >= P ? P : 0;
					A[k+j+md] -= A[k+j+md] >= P ? P : 0;
					w = (w * wm) % P;
				}
			}
		}
	}
	inline void DFT_h(LL A[], int n){
		g = inv(G);
		DFT(A,n);
```

```
                g = G;
                LL invN = inv(n);
                for(int i = 0; i < n; i++) A[i] = A[i] * invN % P;
        }
}
// 使用DFT进行NT变换
// 使用DFT_h进行逆变换
// 因为模数的关系，n不能超过1e6
```

8 原根

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <unordered_map>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;
typedef long long LL;
namespace  PrimitiveRoot{
        LL pow_mod(LL x, LL n, LL mod){
                LL ans = 1;
                while(n){
                        if(n & 1){ ans *= x; ans %= mod; }
                        x *= x; x %= mod; n >>= 1;
                }
                return ans;
        }
        LL Phi(LL n){ // 计算n的欧拉函数
                LL ans = n;
                for(int i = 2; i * i <= n; i++) if(n % i == 0) {
                        ans = ans / i * (i-1);
                        while(n % i == 0) n /= i;
                }
                if(n > 0) ans = ans / n * (n-1);
                return ans;
        }
        void exgcd(LL a, LL b, LL &x, LL &y){ // calc: ax + by == gcd(a, b)
                if(b == 0){ x = 1, y = 0; return; }
                exgcd(b, a % b, y, x); y -= a / b * x;
        }

        LL gcd(LL a, LL b){ // clac GCD( a, b )
                return b == 0 ? a : gcd(b, a % b);
        }
        bool mod_equation(LL a, LL &x, LL b, LL m) { // calc: ax == b ( mod m ) -> x % m
                b = ((b % m) + m) % m;
                LL g = gcd(a, m);
                if(b % g != 0) return false;
                LL y; exgcd(a, m, x, y);
                LL md = m/g;
                x = ((x % md) + md) % md;
                x %= m;
                x = x * (b/g % m) % m;
                return true;
        }
        bool mod_equation(LL a, vector <LL> &ans, LL b, LL m){ //计算模方程所有可能的解
```

```
        ans.clear(); LL x;
        b = ((b % m) + m) % m;
        LL g = gcd(a, m);
        if(b % g != 0) return false;
        LL y; exgcd(a, m, x, y);
        LL md = m/g;
        x = ((x % md) + md) % md;
        x %= m; x = x * (b/g % m) % m;
        for(int i = 0; i < g; i++){
            ans.push_back(x);
            x = (x + md) % m;
        }
        return true;
    }

    LL root, phi, P;

    LL inv(LL a){
        LL x; mod_equation(a, x, 1, P);
        return x;
    }
    void set_mod(LL P_) { // P = 2, 4, p^a, 2*p^a (其中p是奇素数)
        P = P_;
    }
    vector <LL> factor;
    LL get_primitive_root(){ //get x^{Phi(P)} == 1 (mod P) -> x是P的原根 O(玄)
        phi = Phi(P); // P = 2, 4, p^a, 2*p^a (其中p是奇素数)
        factor.clear(); int tmp = phi;
        for(int i = 2; i * i <= phi; i++) if(tmp % i == 0){
            factor.push_back(i);
            while(tmp % i == 0) tmp /= i;
        }
        if(tmp > 1) factor.push_back(tmp);
        for(int i = 1; i <= P; i++){
            bool flag = pow_mod(i, phi, P) == 1;
            if(!flag) continue;
            for(int j = 0; j < factor.size(); j++){
                if(pow_mod(i, phi/factor[j], P) == 1){
                    flag = 0; break;
                }
            }
            if(flag) return root = i;
        }
        return -1;
    }
    unordered_map <LL, LL> mp;
    void D_log_prework(){ // 大步小步法的预处理
        mp.clear(); LL q = sqrt(phi) + 1;
        for(int i = 0; i * q <= phi - 1; i++)
            mp[pow_mod(root, i * q, P)] = i * q;
    }
    LL D_log(LL x){ // 使用大步小步法计算x模P的离散对数
        x %= P;
        LL q = sqrt(phi) + 1;
        LL inv_root = inv(root);
        for(int i = 0; i < q; i++){
            LL right = pow_mod(inv_root, i, P) * x % P;
            if(mp.count(right)){
                return mp[right] + i;
            }
        }
        return -1;
```

```
    }
    bool n_times_remainder(vector <LL> x, LL K, LL A, LL P){ // clac: x^K == A (mod P) 的所有的x, 且按照升序
        x.clear();
        if(A == 0) { x.push_back(0); return true; } // log(0), 特判
        else if(P == 2) { x.push_back(A); return true; } // 底为1，特判
        else {
            set_mod(P);
            get_primitive_root();
            D_log_prework();
            LL b = D_log(A);
            if(b == -1){
                return false;
            }

            mod_equation(K, x, b, phi);
            for(int i = 0; i < x.size(); i++){
                x[i] = pow_mod(root, x[i], P);
            }
            sort(x.begin(), x.end());
            return true;
        }
    }
};

//useage:
//要使用n_times_remainder的话，直接调用即可。
//
//只需要求P的原根时
//set_mod(P); ans = get_primitive_root();
//
//只需要求A关于P的离散对数时
//set_mod(P); get_primitive_root();
//D_log_prework(); ans = D_log(A);
```