Standard Code Library

Your TeamName

Your School

March 26, 2021

Contents

开始

宏定义

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef __int128 LLL;
typedef unsigned u32;
typedef unsigned long long u64;
typedef long double LD;
#define il inline
#define pln putchar('\n')
#define For(i,a,b)  for(int i=(a),(i##i)=(b);i<=(i##i);++i)
#define Rep(i,n)    for(int i=0,(i##i)=(n);i<(i##i);++i)
#define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)
const int M=1000000007,INF=0x3f3f3f3f;
const long long INFLL=0x3f3f3f3f3f3f3f3fLL;
const int N=1000010;
```

快读

```cpp
ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
```

```cpp
template <typename T>
inline bool read(T &x) {
    x = 0; char c = getchar(); int f = 1;
    while (!isdigit(c) && (c != '-') && (c != EOF)) c = getchar();
    if (c == EOF) return 0;
    if (c == '-') f = -1, c = getchar();
    while (isdigit(c)) { x = x * 10 + (c & 15); c = getchar();}
    x *= f; return 1;
}

template <typename T, typename... Args>
inline bool read(T &x, Args &...args) {
    bool res = 1;
    res &= read(x);
    res &= read(args...);
    return res;
}
```

数学

模乘模幂

- longlong 范围用 fpl

```cpp
inline LL mul(LL a, LL b, LL p) {
    LL res = a * b - ((LL)((LD)a * b / p) * p);
    return res < 0 ? res + p : (res < p ? res : res - p);
}

inline LL fp(LL a, LL b, LL Mod) {
    LL res = (Mod != 1);
    for (; b; b >>= 1, a = a * a % Mod)
        if (b & 1)
            res = res * a % Mod;
    return res;
}

inline LL fpl(LL a, LL b, LL Mod) {
    LL res = (Mod != 1);
    for (; b; b >>= 1, a = mul(a, a, Mod))
        if (b & 1)
            res = mul(res, a, Mod);
    return res;
}
```

GCD

```cpp
template <typename T>
inline T gcd(T a, T b) {
    while (b){
        T t = b;
        b = a % b;
        a = t;
    }
    return a;
}

template <typename T>
inline T lcm(T a, T b) { return a / gcd(a, b) * b; }

template <typename T>
T exgcd(T a, T b, T &x, T &y) {
    if (!b) {
        x = 1;
        y = 0;
        return a;
    }
    T res = exgcd(b, a % b, x, y);
    T t = x;
    x = y;
    y = t - a / b * y;
    return res;
}
```

CRT

- 需要 GCD 64 位模乘
- 用来合并同余方程
- 返回最小正数解或最小非负解无解返回-1

```cpp
inline LL Crt(LL a1, LL a2, LL mod1, LL mod2) {
    LL u, v;
    LL g = exgcd(mod1, mod2, u, v);
    if ((a2 - a1) % g) return -1;
    LL m12 = abs(lcm(mod1, mod2));
    LL res = (mul(mod1, mul(u, ((a2 - a1) / g), m12), m12) + a1) % m12;
    return res <= 0 ? res + m12 : res; /* 求最小正数解还是非负解 */
}
```

线性筛

```cpp
struct primenumberlist{
#define MAXN (100000000)
    int cnt, pri[10000000];
    bool np[MAXN + 10];
    primenumberlist(){
        np[1] = 1; cnt = 0;
        for (int i = 2; i <= MAXN; ++i) {
            if (!np[i]) pri[++cnt] = i;
            for (int j = 1; j <= cnt; ++j) {
                LL t = pri[j] * i;
                if (t > MAXN) break;
                np[t] = 1;
                if (!(i % pri[j])) break;
            }
        }
    }
} prime;
```

φ 单点欧拉函数

```cpp
template <typename T>
inline T phi(T x) {
    T res = x;
```

```
    for (T i = 2; i * i <= x; ++i)
        if ((x % i) == 0) {
            res = res / i * (i - 1);
            while ((x % i) == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);
    return res;
}
```

Miller-Rabin 素性测试

- $n <= 10^{18}$
- 需要 64 位模乘 64 位模幂

```
inline bool MR(LL x, LL n, int t) {
    LL las = x;
    for (int i = 1; i <= t; ++i) {
        x = mul(x, x, n);
        if (x == 1 && las != 1 && las != (n - 1)) return 0;
        las = x;
    }
    return x == 1;
}

inline bool isPrime(LL n) {
    if (n == 46856248255981ll || n < 2) return 0;
    if (n == 2 || n == 3 || n == 7 || n == 61 || n == 24251) return 1;
    LL d = n - 1;
    int t = 0;
    while ((d & 1) == 0) d >>= 1, ++t;
    return MR(fpl(2, d, n), n, t) && MR(fpl(61, d, n), n, t);
}
```

Pollard-Rho 分解质因数

- 需要 64 位模乘 gcd
- 求 n 的一个大于 1 的因子可能返回 n 本身
- 调用 PR() 前务必判断 n 的素性检查 $n > 1$

```
mt19937 mt(time(0)); //随机化
inline LL PR(LL n) {
    LL x = uniform_int_distribution<LL>(0, n - 1)(mt), s, t, c = uniform_int_distribution<LL>(1, n - 1)(mt); //随机化
    for (int gol = 1; 1; gol <<= 1, s = t, x = 1) {
        for (int stp = 1; stp <= gol; ++stp) {
            t = (mul(t, t, n) + c) % n;
            x = mul(x, abs(s - t), n);
            if ((stp & 127) == 0) {
                LL d = gcd(x, n);
                if (d > 1) return d;
            }
        }
        LL d = gcd(x, n);
        if (d > 1) return d;
    }
}
```

组合数

- 数较小模数为较大质数求逆元
-    – 如果模数固定可以 O(n) 预处理阶乘的逆元
- 数较大模数为较小质数用 $Lucas$ 定理
-    –

$$C_n^m \equiv C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n \bmod p}^{m \bmod p} (mod\ p)$$

- 数较大模数较小用 $exLucas$ 定理求 $C_n^m mod\ P$

exLucas

- 需要模乘 CRT
- O(P log P)
- 不要求 P 为质数

```
namespace EXLUCAS {
    inline LL idxp(LL n, LL p) {
        LL nn = n;
        while (n > 0) nn -= (n % p), n /= p;
        return nn / (p - 1);
    }

    LL facp(LL n, LL p, LL pk) {
        if (n == 0) return 1;
        LL res = 1;
        if (n >= pk) {
            LL t = n / pk, k = 1, els = n - t * pk;
            for (LL i = 1; i <= els; ++i) if (i % p) k = k * i % pk;
            res = k;
            for (LL i = els + 1; i < pk; ++i) if (i % p) k = k * i % pk;
            res = res * fp(k, n / pk, pk) % pk;
        }
        else for (LL i = 1; i <= n; ++i) if (i % p) res = res * i % pk;
        return res * facp(n / p, p, pk) % pk;
    }

    inline LL exlucas(LL n, LL m, LL p, LL pk, LL k) {
        LL a = facp(n, p, pk) * fp(facp(n - m, p, pk) * facp(m, p, pk) % pk, pk / p * (p - 1) - 1, pk) % pk;
        LL b = idxp(n, p) - idxp(m, p) - idxp(n - m, p);
        if (b >= k) return 0;
        while (b--) a *= p;
        return a % pk;
    }

    /* 接口 */ inline LL exlucas(LL n, LL m, LL p) {
        LL a = 0, b = 1;
        for (LL i = 2; i * i <= p; ++i) {
            if (p % i) continue;
            LL t = 0, pk = 1;
            while (p % i == 0) ++t, p /= i, pk *= i;
            a = Crt(a, exlucas(n, m, i, pk, t), b, pk);
            b *= pk;
        }
        return (p > 1) ? Crt(a, exlucas(n, m, p, p, 1), b, p) : a;
    }
}
```

二维计算几何

- Point 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 1..n

点向量基本运算

```
template <typename T>
struct Point {
    T x, y;
    Point() {}
    Point(T u, T v) : x(u), y(v) {}
    Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
    Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
    Point operator*(const T &a) const { return Point(x * a, y * a); }
```

```cpp
    T operator*(const Point &a) const { return x * a.x + y * a.y; }
    T operator%(const Point &a) const { return x * a.y - y * a.x; }
    double len() const { return hypot(x, y); }
    double operator^(const Point &a) const { return (a - (*this)).len(); }
    double angle() const { return atan2(y, x); }
    bool id() const { return y < 0 || (y == 0 && x < 0); }
    bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
};
typedef Point<double> point;

#define sqr(x) ((x) * (x))
const point O(0, 0);
const double PI(acos(-1.0)), EPS(1e-8);
inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }
```

位置关系

```cpp
inline bool in_same_seg(point p, point a, point b) {
    if (fabs(mul(p, a, b)) < EPS) {
        if (a.x > b.x) swap(a, b);
        return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
    } else return 0;
}

inline bool is_right(point st, point ed, point a) {
    return ((ed - st) % (a - st)) < 0;
}

inline point intersection(point s1, point t1, point s2, point t2) {
    return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
}

inline bool parallel(point a, point b, point c, point d) {
    return dcmp((b - a) % (d - c), 0);
}

inline double point2line(point p, point s, point t) {
    return fabs(mul(p, s, t) / (t - s).len());
}

inline double point2seg(point p, point s, point t) {
    return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
}
```

多边形

求多边形面积

```cpp
inline double area(int n, point s[]) {
    double res = 0;
    s[n + 1] = s[1];
    for (int i = 1; i <= n; ++i)
        res += s[i] % s[i + 1];
    return fabs(res / 2);
}
```

判断点是否在多边形内

- 特判边上的点
- 使用了 `a[1]...a[n+1]` 的数组

```cpp
inline bool in_the_area(point p, int n, point area[]) {
    bool ans = 0; double x;
    area[n + 1] = area[1];
    for (int i = 1; i <= n; ++i) {
        point p1 = area[i], p2 = area[i + 1];
        if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
```

```
        if (p1.y == p2.y) continue;
        if (p.y < min(p1.y, p2.y)) continue;
        if (p.y >= max(p1.y, p2.y)) continue;
        ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
    }
    return ans;
}
```

凸包

- *Andrew* 算法
- $O(n \log n)$
- 可以应对凸包退化成直线/单点的情况但后续旋转卡壳时应注意特判
- 注意是否应该统计凸包边上的点

```
inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }

inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
    sort(p + 1, p + 1 + n, pcmp1);
    int m = 0;
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    int k = m;
    for (int i = n - 1; i >= 1; --i) {
        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    return m - (n > 1); //返回凸包有多少个点
}
```

凸包直径·平面最远点对

- 旋转卡壳算法
- $O(n)$
- 凸包的边上只能有端点，否则不满足严格单峰
- 凸包不能退化成直线，调用前务必检查 `n>=3`
- 使用了 `a[1]...a[n+1]` 的数组

```
inline double Rotating_Caliper(int n, point a[]) {
    a[n + 1] = a[1];
    double ans = 0;
    int j = 2;
    for (int i = 1; i <= n; ++i) {
        while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
        ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
    }
    return ans;
}
```

平面最近点对

- 分治 + 归并
- $O(n \log n)$

```
namespace find_the_closest_pair_of_points {
    const int N = 200010; //maxn
    inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
    inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }

    point a[N], b[N];
    double ans;
    inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }

    void find(int l, int r) {
```

```cpp
        if (l == r) return;
        if (l + 1 == r) {
            if (a[l] > a[r]) swap(a[l], a[r]);
            upd(a[l], a[r]); return;
        }
        int mid = (l + r) >> 1;
        double mx = (a[mid + 1].x + a[mid].x) / 2;
        find(l, mid); find(mid + 1, r);
        int i = l, j = mid + 1;
        for (int k = l; k <= r; ++k) b[k] = a[((j > r) || (i <= mid && a[j] > a[i])) ? (i++) : (j++)];
        for (int k = l; k <= r; ++k) a[k] = b[k];
        int tot = 0;
        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);
            b[++tot] = a[k];
        }
    }

    //接口
    inline double solve(int n, point ipt[]){
        ans = 0x3f3f3f3f3f3f3f3fll; //max distance
        for (int i = 1; i <= n; ++i) a[i] = ipt[i];
        sort(a + 1, a + 1 + n, cmp1);
        find(1, n);
        return ans;
    }
}
```

圆

三点垂心

```cpp
inline point geto(point p1, point p2, point p3) {
    double a = p2.x - p1.x;
    double b = p2.y - p1.y;
    double c = p3.x - p2.x;
    double d = p3.y - p2.y;
    double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
    double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
    return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
}
```

最小覆盖圆

- 随机增量 O(n)

```cpp
inline void min_circlefill(point &o, double &r, int n, point a[]) {
    mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
    o = a[1];
    r = 0;
    for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
        o = a[i];
        r = 0;
        for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
            o = (a[i] + a[j]) * 0.5;
            r = (a[i] ^ a[j]) * 0.5;
            for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
                o = geto(a[i], a[j], a[k]);
                r = (o ^ a[i]);
            }
        }
    }
}
```