代码模板

zdq

SCUT

May 21, 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;
typedef pair<int,int> pii;
#define ff first
#define ss second
#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=1000007;
```

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;
}
```

对拍

```cpp
//in.txt
//AC.exe std.txt
//MY.exe my.txt

void init(){
    FILE*F=fopen("int.txt","w");

    //srand(time(0));
    //int a=(long long)rand()*rand()%1001;
    //fscanf(F,"%d",&a);fprintf(F,"%d\n",a);

    fclose(F);
}

int main(){
    init();
    while(1){
        system("AC.exe < in.txt > std.txt");

        system("MY.exe < in.txt > my.txt");

        if(system("fc std.txt my.txt")){
            puts("WA");
```

```
            return 0;
        }else puts("AC\n\n");

        init();
    }
}
```

数学

模乘模幂

- longlong 范围用 fpl

```
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

```
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

```
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 */
}
```

高次同余方程

BSGS

- 求最小非负整数 x 满足 `a^x=b(mod p)` 要求 `(a,p)=1` p 不必是质数
- 需要 `fp` 和 `map/unorded_map`
- 复杂度 O(sqrt(p))
- 如果哈希被卡
- – 直接改用 `map` 不过复杂度会多一个 log
- – 手动实现整数哈希 `unordered_map<int,int,custom_hash>h;`

```
inline int bsgs(int a, int b, int p){
    unordered_map<int, int> h;
    b %= p;
    int t = (int)sqrt(p) + 1;
    for (int j = 0, v = b; j < t; ++j, v = (LL)v * a % p) h[v] = j;
    a = fp(a, t, p);
    if (a == 0) return b == 0 ? 1 : -1;
    for (int i = 0, v = 1; i <= t; ++i, v = (LL)v * a % p){
        if (h.find(v) == h.end()) continue;
        int j = i * t - h[v];
        if (j >= 0) return j;
    }
    return -1;
}
```

线性筛

```
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

```
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
- –

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

- 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;
    }
}
```

类欧几里得算法

- 计算直线下整点数
- `f=∑[(ai+b)/c] g=∑i[(ai+b)/c] h=∑[(ai+b)/c]^2 i=0..n  a,b,n⊠N  c⊠N*`
- 复杂度 `log(MAX{a,c})`

```
struct dat{LL f, g, h;};
const LL i2 = 499122177, i3 = 332748118, M = 998244353; //预处理出模 M 意义下 2 和 3 的逆元
dat f(LL a, LL b, LL c, LL n){
    LL ac = a / c, bc = b / c;
    LL n2 = (n * (n + 1) % M) * i2 % M, n3 = n2 * (2ll * n + 1) % M * i3 % M;
    dat res = {
        (n2 * ac % M + (n + 1) * bc % M) % M,
        (ac * n3 % M + bc * n2 % M) % M,
        (ac * ac % M * n3 % M +
            bc * bc % M * (n + 1) % M + ac * bc % M * n2 % M * 2ll) % M};
    a %= c; b %= c; if (a == 0)return res;
    LL m = (a * n + b) / c;
    dat p = f(c, c - b - 1, a, m - 1);
    LL fc = (n * m % M - p.f + M) % M, gc = (n2 * m % M - i2 * (p.f + p.h) % M + M) % M;
```

```
    return{(res.f + fc) % M,    (res.g + gc) % M,
    (res.h + 2ll * (bc * fc % M + ac * gc % M) % M +
        n * m % M * m % M - 2ll * p.g - p.f + 3ll * M) % M};}
```

日期操作

用于跳转的常量

```
const LL year_1[2]={365, 366};
const LL year_400=1460097;
const LL m_day[13]={(LL)0x3f3f3f, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

辅助函数

```
inline bool isLeap(LL t){return (t % 400 == 0) ||((t % 4 == 0) && (t % 100));}
inline bool pick(LL a, LL b){return ((isLeap(a) && b <= 2) ||(isLeap(a + 1) && b > 2));}
inline LL dayThisMonth(LL y, LL m){return m_day[m] + isLeap(y) * (m == 2);}
```

日期和整数的一一对应

- LL 可以改成 int

```
struct MY_DATE{
    LL year, month, day;
    MY_DATE(LL y = 2021, LL m = 1, LL d = 1) : year(y), month(m), day(d){};
    LL p(MY_DATE op = {0, 0, 0}){//日期转换为整数
        LL y = year - op.year, m = month - op.month, d = day - op.day;
        if (m <= 2){ y--; m += 12;}
        return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307;
    }
    MY_DATE run(LL k){//当前日期过 k 天
        k += p();
        LL x = k + 1789995, n = 4 * x / 146097, i, j, d;
        x -= (146097 * n + 3) / 4;
        i = (4000 * (x + 1)) / 1461001;
        x -= 1461 * i / 4 - 31;
        j = 80 * x / 2447;
        d = x - 2447 * j / 80;
        x = j / 11;
        return MY_DATE(100 * (n - 49) + i + x, j + 2 - 12 * x, d);
    }
};
```

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); }
    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(); }
};
```

```cpp
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

```cpp
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

```cpp
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)

```cpp
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) {
        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

```
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)

```
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]);
            }
        }
    }
}
```

图论

存图

- 前向星
- 注意边数开够

```
int Head[N], Ver[N*2], Next[N*2], Ew[N*2], Gtot=1;
inline void graphinit(int n) {Gtot=1; for(int i=1; i<=n; ++i) Head[i]=0;}
inline void edge(int u, int v, int w=1) {Ver[++Gtot]=v; Next[Gtot]=Head[u]; Ew[Gtot]=w; Head[u]=Gtot;}
#define go(i,st,to) for (int i=Head[st], to=Ver[i]; i; i=Next[i], to=Ver[i])
```

最短路

Dijkstra

- 非负权图

```cpp
namespace DIJK{//适用非负权图 满足当前 dist 最小的点一定不会再被松弛
    typedef pair<long long,int> pii;
    long long dist[N];//存最短路长度
    bool vis[N];//记录每个点是否被从队列中取出  每个点只需第一次取出时扩展
    priority_queue<pii,vector<pii>,greater<pii> >pq;//维护当前 dist[] 最小值及对应下标 小根堆

    inline void dijk(int s,int n){//s 是源点 n 是点数
        while(pq.size())pq.pop();for(int i=1;i<=n;++i)dist[i]=INFLL,vis[i]=0;//所有变量初始化
        dist[s]=0;pq.push(make_pair(0,s));
        while(pq.size()){
            int now=pq.top().second;pq.pop();
            if(vis[now])continue;vis[now]=1;
            go(i,now,to){
                const long long relx(dist[now]+Ew[i]);
                if(dist[to]>relx){dist[to]=relx;pq.push(make_pair(dist[to],to));}//松弛
            }
        }
    }
}
```

LCA

- 倍增求 lca
- 数组开够

```cpp
namespace LCA_Log{
    int fa[N][22],dep[N];
    int t,now;
    void dfs(int x){
        dep[x]=dep[fa[x][0]]+1;
        go(i,x,to){
            if(dep[to])continue;
            fa[to][0]=x;for(int j=1;j<=t;++j)fa[to][j]=fa[fa[to][j-1]][j-1];
            dfs(to);
        }
    }

    //初始化接口
    inline void lcainit(int n,int rt){//记得初始化全部变量
        now=1;t=0;while(now<n)++t,now<<=1;
        for(int i=1;i<=n;++i)dep[i]=0,fa[i][0]=0;
        for(int i=1;i<=t;++i)fa[rt][i]=0;
        dfs(rt);
    }

    //求 lca 接口
    inline int lca(int u,int v){
        if(dep[u]>dep[v])swap(u,v);
        for(int i=t;~i;--i)if(dep[fa[v][i]]>=dep[u])v=fa[v][i];
        if(u==v)return u;
        for(int i=t;~i;--i)if(fa[u][i]!=fa[v][i])u=fa[u][i],v=fa[v][i];
        return fa[u][0];
    }
}
```

连通性

有向图强联通分量

- tarjan $O(n)$

```cpp
namespace SCC{
    int dfn[N],clk,low[N];
    bool ins[N];int sta[N],tot; //栈  存正在构建的强连通块
```

```
    vector<int>scc[N];int c[N],cnt;//cnt 为强联通块数 scc[i] 存放每个块内点 c[i] 为原图每个结点属于的块
    void dfs(int x){
        dfn[x]=low[x]=(++clk);//low[] 在这里初始化
        ins[x]=1;sta[++tot]=x;
        go(i,x,to){
            if(!dfn[to]){dfs(to);low[x]=min(low[x],low[to]);}//走树边
            else if(ins[to])low[x]=min(low[x],dfn[to]);//走返祖边
        }
        if(dfn[x]==low[x]){//该结点为块的代表元
            ++cnt;int u;
            do{u=sta[tot--];ins[u]=0;c[u]=cnt;scc[cnt].push_back(u);}while(x!=u);
        }
    }
    inline void tarjan(int n){//n 是点数
        for(int i=1;i<=cnt;++i)scc[i].clear();//清除上次的 scc 防止被卡 MLE
        for(int i=1;i<=n;++i)dfn[i]=ins[i]=0;tot=clk=cnt=0;//全部变量初始化
        for(int i=1;i<=n;++i)if(!dfn[i])dfs(i);
        for(int i=1;i<=n;++i)c[i]+=n;//此行 (可以省略) 便于原图上加点建新图 加新点前要初始化 Head[]=0
    }
}
```

数据结构

手写整数哈希

- 防止自带哈希被卡 T

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

堆式线段树

- 区间求和区间修改
- 空间估算
- 所有数组务必初始化

```
struct SegmentTree_Heap{
    #define TreeLen (N<<2)          //N
    #define lc(x)   ((x)<<1)
    #define rc(x)   ((x)<<1|1)
    #define sum(x)  (tr[x].sum)     //
    #define t(x)    (t[x])          //

    struct dat{
        LL sum;
        /* 这里写区间加法 */
        dat operator+(const dat&brother){
            dat result;
            result.sum=sum+brother.sum;
            return result;
        }
    }tr[TreeLen];
    LL t[TreeLen];  //lazy tag

    /* 单区间修改 */
    inline void change(const int&x,const int&l,const int&r,const LL&d){
        tr[x].sum=tr[x].sum+d*(r-l+1);
        t[x]=t[x]+d;
    }
```

```cpp

    inline void pushup(int x){tr[x]=tr[lc(x)]+tr[rc(x)];}

    inline void pushdown(int x,const int&l,const int&r,const int&mid){
        if(t(x)){//注意区间修改细节
            change(lc(x),l,mid,t(x));
            change(rc(x),mid+1,r,t(x));
            t(x)=0;
        }
    }

    void build(int x,int l,int r){
        t(x)=0;   // 记得初始化!!!
        if(l==r){
            sum(x)=0;
            return;
        }
        int mid=(l+r)>>1;
        build(lc(x),l,mid);
        build(rc(x),mid+1,r);
        pushup(x);
    }

    void add(int x,int l,int r,const int&L,const int&R,const LL&d){
        if(L<=l&&r<=R){
            change(x,l,r,d);
            return;
        }
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        if(L<=mid)add(lc(x),l,mid,L,R,d);
        if(R>mid)add(rc(x),mid+1,r,L,R,d);
        pushup(x);
    }

    LL ask(int x,int l,int r,const int&L,const int&R){
        if(L<=l&&r<=R)return sum(x);
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        LL res=0;
        if(L<=mid)res=(res+ask(lc(x),l,mid,L,R));
        if(mid<R)res=(res+ask(rc(x),mid+1,r,L,R));
        return res;
    }
};
```

小根堆

```cpp
namespace MyPQ{
    typedef int pqdat;    ////
    pqdat q[N];
    int tot;
    void up(int x){
        while(x>1)
        if(q[x]<q[x/2]){
            swap(q[x],q[x/2]);
            x/=2;
        }else return;
    }
    void down(int x){
        int ls=x*2;
        while(ls<=tot){
            if(ls<tot&&q[ls+1]<q[ls])++ls;
            if(q[ls]<q[x]){
                swap(q[x],q[ls]);x=ls;ls=x*2;
            }else return;
        }
    }
    void push(pqdat x){q[++tot]=x;up(tot);}
    pqdat top(){return q[1];}
    void pop(){if(!tot)return;q[1]=q[tot--];down(1);}
    void pop(int k){if(!tot)return;q[k]=q[tot--];up(k);down(k);}
}
```

Treap

```
#include<bits/stdc++.h>
using namespace std;
const int N=2000007,INF=(1ll<<30)+7;

//Treap 维护升序多重集
//支持操作: 数 <-> 排名 查询某数前驱后继
//操作数 x 可以不在集合中
//x 的排名: 集合中 <x 的数的个数 +1
//排名 x 的数: 集合中排名 <=x 的数中的最大数
//x 的前驱 比 x 小的最大数

struct treap{//所有点值不同 用副本数实现多重集
    int l,r;
    int v,w;//v 是数据 w 是维护堆的随机值
    int num,sz;//num 是该点副本数 sz 是该子树副本总数
}tr[N];int tot,rt;//tr[0] 始终全 0 使用范围 tr[1..n]
#define lc(x) tr[x].l
#define rc(x) tr[x].r
#define sz(x) tr[x].sz
#define num(x) tr[x].num
#define val(x) tr[x].v
#define wt(x) tr[x].w

inline int New(int x){
    val(++tot)=x; wt(tot)=rand();
    num(tot)=sz(tot)=1; return tot;
}

inline void upd(int p){sz(p)=sz(lc(p))+sz(rc(p))+num(p);}

inline void build(){//初始化 INF 和-INF 两个点
    srand(time(0));
    rt=1;tot=2;
    rc(1)=2;val(1)=-INF;wt(1)=rand();num(1)=1;sz(1)=2;
    val(2)=INF;wt(2)=rand();num(2)=1;sz(2)=1;
}

//调用时记得减一 askrk(rt,x)-1
int askrk(int p,int x){//当前子树中查询 x 的排名
    if(p==0)return 1;//说明某子树所有数均比 x 大
    if(x==val(p))return sz(lc(p))+1;
    return x<val(p)?askrk(lc(p),x):askrk(rc(p),x)+sz(lc(p))+num(p);
}

//调用时记得加一 kth(rt,++rank)
int kth(int p,int rk){//当前子树中查询排名 rk 的数
    if(p==0)return INF;//说明集合大小 <rk
    if(sz(lc(p))>=rk)return kth(lc(p),rk);
    rk-=sz(lc(p))+num(p);
    return (rk>0)?kth(rc(p),rk):val(p);
}

inline void zig(int &p){//与左子节点交换位置
    int q=lc(p);lc(p)=rc(q);rc(q)=p;
    upd(p);p=q;upd(p);
}

inline void zag(int &p){//与右子节点交换位置
    int q=rc(p);rc(p)=lc(q);lc(q)=p;
    upd(p);p=q;upd(p);
}

//insert(rt,x)
void insert(int &p,int x){//当前子树中插入 x
    if(p==0){p=New(x);return;}//x 首次插入
    if(x==val(p)){++num(p);++sz(p);return;}
    if(x<val(p)){
        insert(lc(p),x);
        if(wt(p)<wt(lc(p)))zig(p);//维护大根堆
```

```
    }else{
        insert(rc(p),x);
        if(wt(p)<wt(rc(p)))zag(p);//维护大根堆
    }
    upd(p);
}

//erase(rt,x)
void erase(int &p,int x){//当前子树中删除一个 x
    if(p==0)return;//已经无需删除
    if(val(p)==x){//如果找到了 x 的位置
        if(num(p)>1){//无需删点
            --num(p);--sz(p);return;//如果有多个 x 维护副本数即可
        }
        if(lc(p)||rc(p)){//该点不是叶子节点 则不断向下调整至叶子节点
            if(rc(p)==0||wt(lc(p))>wt(rc(p)))zig(p),erase(rc(p),x);//由于 rand() 的值域 & 大根堆的实现 故省略左子树为空的判断
            else zag(p),erase(lc(p),x);
            upd(p);
        }else p=0;//是叶子节点则直接删除
        return;
    }
    x<val(p)?erase(lc(p),x):erase(rc(p),x);upd(p);
}

int askpre(int x){
    int id=1;//-INF 若没有前驱则返回-INF
    //尝试自顶向下寻找 x 则 x 的前驱有两种情况
    //1) 未找到 x 或 x 没有左子树 则前驱在搜索路径上
    //2) 前驱是 x 的左子树中最大值 即 x 的左子树一直向右走
    int p=rt;
    while(p){
        if(x==val(p)){//找到 x
            if(lc(p)){p=lc(p);while(rc(p))p=rc(p);id=p;}
            break;
        }
        if(val(p)<x&&val(p)>val(id))id=p;//每经过一个点尝试更新前驱
        p=(val(p)>x?lc(p):rc(p));//找 x
    }
    return val(id);
}

int asknxt(int x){
    int id=2;//INF
    int p=rt;
    while(p){
        if(x==val(p)){
            if(rc(p)){p=rc(p);while(lc(p))p=lc(p);id=p;}
            break;
        }
        if(val(p)>x&&val(p)<val(id))id=p;
        p=(val(p)>x?lc(p):rc(p));
    }
    return val(id);
}
```

```
#include <bits/stdc++.h>
using namespace std;

struct MY_IO{
#define DEBUG 1///本地调试
#define MAXSIZE (1 << 20)
    inline bool isdigit(const char &x) { return x >= '0' && x <= '9'; }//字符集 看情况改
    inline bool blank(const char &c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }
#if DEBUG //
#else     //
    char buf[MAXSIZE + 3], *p1, *p2, pbuf[MAXSIZE + 3], *pp;
    MY_IO() : p1(buf), p2(buf), pp(pbuf) {}
    ~MY_IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
#endif    //

    inline char gc(){
#if DEBUG                 //
```

```cpp
        return getchar(); //
#else                     //
        if (p1 == p2)
            p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
        return p1 == p2 ? EOF : *p1++;
#endif                    //
    }

    inline void pc(const char &c){
#if DEBUG            //
        putchar(c); //
#else                //
        if (pp - pbuf == MAXSIZE)
            fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
#endif               //
    }

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

    inline int gets(char *s){
        char c = gc(); while (blank(c) && c != EOF) c = gc();
        if (c == EOF) return 0;
        int len = 0;
        while (!blank(c) && c != EOF) *s++ = c, c = gc(), ++len;
        *s = 0; return len;
    }

    inline void getc(char &c){for (c = gc(); blank(c) && c != EOF; c = gc());}

    /* 不能输出 (int)(-2^31)*/template<typename T>inline void write(T x){
        if (x < 0) x = -x, pc('-');
        static T sta[233];
        int top = 0;
        do{
            sta[top++] = x % 10, x /= 10;
        } while (x);
        while (top) pc(sta[--top] + '0');
    }

    template <typename T>inline void write(T x, const char &Lastchar){write(x); pc(Lastchar);}

    inline void puts(char *s){while ((*s) != 0)pc(*s++);}

    inline int getline(char *s){
        char c = gc();
        int len = 0;
        while (c != '\n' && c != EOF) *s++ = c, c = gc(), ++len;
        *s = 0; return len;
    }

    inline void putline(char *s){while ((*s) != 0)pc(*s++); pc('\n');}
}IO;
#define read IO.read
#define write IO.write
#define gc IO.gc
#define pc IO.pc
#define gets IO.gets
#define getc IO.getc
#define puts IO.puts
```

```
#define getl IO.getline
#define putl IO.putline
```