个人常用代码库

zqy1018

2020 年 9 月 22 日

目录

1 数据结构

1.1 非旋转 Treap

```
struct Tr {
    int siz, v, prio, lch, rch;
};
Tr tr[400005];
int S, root;
void maintain(int x){
    // 更新 x 的子树大小
    tr[x].siz = 1 + tr[tr[x].lch].siz + tr[tr[x].rch].siz;
}
void init_env(){
    // 初始化相关变量
    S = 0; root = 0; tr[0].siz = 0;
    srand(time(NULL));
}
int tree_new(int k){
    // 分配一个新节点
    ++S;
    tr[S].siz = 1, tr[S].v = k,
    tr[S].prio = rand(),
    tr[S].lch = tr[S].rch = 0;
    return S;
}
void Split(int now, int k, int &x, int &y){
    if (!now) x = y = 0;
    else {
        if (tr[now].v <= k){
            x = now, Split(tr[now].rch, k, tr[now].rch, y);
        }else {
            y = now, Split(tr[now].lch, k, x, tr[now].lch);
        }
        maintain(now);
    }
}
void Split_K(int now, int k, int &x, int &y){
    if (!now) x = y = 0;
    else {
        if (k > tr[tr[now].lch].siz){
            x = now, Split_K(tr[now].rch, k - tr[tr[now].lch].siz - 1, tr[now].rch, y);
        }else {
            y = now, Split_K(tr[now].lch, k, x, tr[now].lch);
        }
        maintain(now);
    }
}
int Merge(int x, int y){
    if (!x || !y) return x + y;
    if (tr[x].prio < tr[y].prio){
```

```
        // y 所有节点的值 > x，故接到 x 的右子树上
        tr[x].rch = Merge(tr[x].rch, y);
        maintain(x);
        return x;
    }else{
        // x 所有节点的值 < y，故接到 y 的左子树上
        tr[y].lch = Merge(x, tr[y].lch);
        maintain(y);
        return y;
    }
}
```

2 数学

2.1 快速乘

```
// 有 __int128 请用 __int128 !!! 不要冒险!!!
inline ll fstmul(ll a, ll b, ll M){
    a %= M, b %= M;
    return (a * b - (ll)((long double)a / M * b) * M + M) % M;
}
```

2.2 扩展 Euclid 及应用

```
// ax+by=gcd(a, b)
ll extgcd(ll a, ll b, ll &x, ll &y){
    ll d;
    if (b == 0){
        d = a, x = 1, y = 0;
    } else {
        d = extgcd(b, a % b, y, x);
        y -= x * (a / b);
    }
    return d;
}
// find minimal non-negative x so that ax+by=c
ll solve(ll a, ll b, ll c){
    ll x, y;
    ll d = extgcd(a, b, x, y);
    if (c % d != 0) return NO_SOLUTION;
    x *= (c / d), y *= (c / d);
    ll t = b / d;
    x = (x % t + t) % t;
    return x;
}
```

2.3 线性同余不等式

给定 $0 \le l \le r < m, d < m$，找 $l \le dx \bmod m \le r$ 的最小非负整数解。详见 POJ 3530。

```
ll modinq(ll m, ll d, ll l, ll r){
    if (r < l) return NO_SOLUTION;
    if (l == 0) return 0;
    if (d == 0) return NO_SOLUTION;
    if ((r / d) * d >= l) return (l - 1) / d + 1;
    ll res = modinq(d, m % d, (d - r % d) % d, (d - l % d) % d);
    return res == NO_SOLUTION ? NO_SOLUTION : (m * res + l - 1) / d + 1;
}
```

2.4 求逆元

```
// inv[i] = 1ll * (p - p / i) * inv[p % i] % p;
// all different a_i ? first find the inverse of their product, then trace back!
```

2.5 值域固定的快速 gcd

```
// O(N)-O(1)
const int N = 1000000, SQRTN = 1000;
int minp[N + 5] = {0}, prime[(N + 5) / 2], tot = 0;
int xa[N + 5], xb[N + 5], xc[N + 5];
int gcd_table[SQRTN + 5][SQRTN + 5];
void build(){
    // combine linear sieve with the preprocessing
    minp[1] = xa[1] = xb[1] = xc[1] = 1;
    for (int i = 2; i <= N; ++i){
        if (!minp[i])
            prime[++tot] = i, minp[i] = i;
        for (int j = 1; j <= tot; ++j){
            if (1ll * i * prime[j] > N || prime[j] > minp[i]) break;
            minp[i * prime[j]] = prime[j];
        }
        int bf = i / minp[i], xxa = xa[bf] * minp[i];
        int xxb = xb[bf], xxc = xc[bf];
        if (xxa >= xxc)
                xa[i] = xxb, xb[i] = xxc, xc[i] = xxa;
        else {
            xc[i] = xxc;
            if (xxa >= xxb) xa[i] = xxb, xb[i] = xxa;
            else xa[i] = xxa, xb[i] = xxb;
        }
    }
    // build the map
    for (int i = 1; i <= SQRTN; ++i)
        gcd_table[i][0] = gcd_table[0][i] = i;
    for (int i = 1; i <= SQRTN; ++i)
        for (int j = 1; j <= i; ++j)
            gcd_table[i][j] = gcd_table[j][i] = gcd_table[i % j][j];
}
int query(int x, int y){
```

```
    int res = 1, tmp;
    if (1ll * xc[x] * xc[x] > x && y % xc[x] == 0)
        y /= xc[x], res *= xc[x];
    else if (1ll * xc[x] * xc[x] <= x){
        tmp = gcd_table[xc[x]][y % xc[x]];
        res *= tmp, y /= tmp;
    }
    tmp = gcd_table[xa[x]][y % xa[x]];
    res *= tmp, y /= tmp;
    tmp = gcd_table[xb[x]][y % xb[x]];
    res *= tmp, y /= tmp;
    return res;
}
```

2.6 CRT 及其扩展

普通 CRT：对于 n 个形如 $x \equiv a_i \pmod{m_i}$ 的线性方程，要求 m_i 间两两互质。设 $M = \prod m_i, M_i = \frac{M}{m_i}, b_i M_i \equiv 1 \pmod{m_i}$，那么 $x = \sum a_i b_i M_i$ 是唯一解。

扩展 CRT：考虑每次合并两个线性方程：$x \equiv b_1 \pmod{m_1}, ax \equiv b_2 \pmod{m_2}$。写 $x = b_1 + tm_1$，带入得 $m_1 at \equiv b_2 - ab_1 \pmod{m_2}$。解出 t，如 $t \equiv c \pmod{m_2}$，则 $t = c + dm_2$，带入，从而 $x \equiv b_1 + cm_1 \pmod{\operatorname{lcm}(m_1, m_2)}$。

```
ll extCRT(ll m[], ll b[], int n){
    ll M = m[1], ans = b[1], x, y;
    for (int i = 2; i <= n; ++i){
        ll d = extgcd(M, m[i], x, y), a_ = (b[i] - ans) % m[i];
        if (a_ < 0) a_ += m[i];
        if (a_ % d) return NO_SOLUTION;
        x = fstmul(x, a_ / d, m[i] / d);
        if (x < 0) x += m[i] / d;
        ll MM = M;
        M /= d, M *= m[i];
        ans = (ans + fstmul(x, MM, M)) % M;
        if (ans < 0) ans += M;
    }
    return ans;
}
```

2.7 组合数取模

普通 Lucas：p 为质数时，等价于把 n, m 在 p 进制下分解后求组合数再乘起来。

$$\binom{n}{m} \equiv \binom{\lfloor \frac{n}{p} \rfloor}{\lfloor \frac{m}{p} \rfloor} \binom{n \bmod p}{m \bmod p} \pmod{p}$$

扩展 Lucas：p 不为质数时将其分解成 p_i^k，对每一个分别求解然后用 CRT 合并。分开处理，对阶乘中 p 的倍数计算 p 的幂次，对其他部分找循环节。

```
// first 记录的是阶乘模 P 的值，second 记录该阶乘中 p 的指数
pair<ll, ll> mod_fac(ll n, ll p, ll bigp){
    if (n == 0) return make_pair(1, 0);
    pair<ll, ll> res = mod_fac(n / p, p, bigp);
```

```cpp
    ll fir = res.first * poww(fac[bigp], n / bigp, bigp) % bigp;
    fir = fir * fac[n % bigp] % bigp;
    return make_pair(fir, res.second + n / p);
}
ll extLucas(ll n, ll m, ll p, ll k, ll bigp){
    ll cnt = 1;
    fac[0] = 1;
    for (ll i = 1; i <= bigp; ++i){
        if (cnt == p) cnt = 1, fac[i] = fac[i - 1];
        else cnt++, fac[i] = fac[i - 1] * i % bigp;
    }
    pair<ll, ll> fz = mod_fac(n, p, bigp);
    pair<ll, ll> fm1 = mod_fac(m, p, bigp);
    pair<ll, ll> fm2 = mod_fac(n - m, p, bigp);
    if (fz.second - fm1.second - fm2.second >= k) return 0;
    ll ps = poww(p, fz.second - fm1.second - fm2.second, bigp);
    ps = ps * fz.first % bigp;
    ps = ps * inv(fm1.first, bigp) % bigp;
    ps = ps * inv(fm2.first, bigp) % bigp;
    return ps;
}
// 求 C(n, m) mod p^k 时，调用 extLucas(n, m, p, k, p^k)
```

2.8 高斯消元

```cpp
db a[maxn][maxn], x[maxn];
int main()
{
    int rank = 0;

    for (int i = 1, now = 1; i <= n && now <= m; ++now)
    {
        int tmp = i;
        for (int j = i + 1; j <= n; ++j)
            if (fabs(a[j][now]) > fabs(a[tmp][now]))tmp = j;
        for (int k = now; k <= m; ++k)
            std::swap(a[i][k], a[tmp][k]);
        if (fabs(a[i][now]) < eps) continue;

        for (int j = i + 1; j <= n; ++j)
        {
            db tmp = a[j][now] / a[i][now];
            for (int k = now; k <= m; ++k)
                a[j][k] -= tmp * a[i][k];
        }
        ++i; ++rank;
    }

    if (rank == n)
    {
        x[n] = a[n][n + 1] / a[n][n];
```

```
        for (int i = n - 1; i; --i)
        {
            for (int j = i + 1; j <= n; ++j)
                a[i][n + 1] -= x[j] * a[i][j];
            x[i] = a[i][n + 1] / a[i][i];
        }
    }
    else puts("Infinite Solution!");
    return 0;
}
```

2.9 大数分解

```
class Factorization{
    typedef long long ll;
    const ll prime[12];
    vector<ll> res;
    // gcd, poww, fstmul 自己写
    bool witness(ll a, ll n, ll t, ll u){
        ll x = poww(a, u, n);
        for (ll i = 0; i < t; ++i){
            ll xx = fstmul(x, x, n);
            if (xx == 1 && x != 1 && x != n - 1)
                return true;
            x = xx;
        }
        if (x != 1) return true;
        return false;
    }
    bool miller_rabin(ll n){
        for (int i = 0; i < 12; ++i)
            if (n % prime[i] == 0){
                if (n == prime[i]) return true;
                return false;
            }
        ll t = 0, u = n - 1;
        while (!(u & 1ll)) ++t, u >>= 1;
        for (int i = 0; i < 12; ++i)
            if (witness(prime[i], n, t, u))
                return false;
        return true;
    }
    ll rho(ll n, ll c){
        ll x = rand();
        ll y = x, d = 1, q = 1;
        for (int k = 2; d == 1; k <<= 1, y = x, q = 1){
            for (int i = 0; i < k; ++i){
                x = fstmul(x, x, n) + c;
                if (x >= n) x -= n;
                ll Abs = (x > y) ? x - y : y - x;
                q = fstmul(q, Abs, n);
```

```
                if (!(i & 127)){
                    d = gcd(q, n);
                    if (d > 1) return d;
                }
            }
            d = gcd(q, n);
        }
        return d;
    }
    ll Pollard(ll n){
        for (int i = 0; i < 12; ++i)
            if (n % prime[i] == 0)
                return prime[i];
        ll d = n;
        while (d == n)
            d = rho(n, rand() % (n - 1) + 1);
        return d;
    }
    void findD(ll n){
        if (miller_rabin(n)){
            res.push_back(n);
            return ;
        } else {
            ll d = Pollard(n);
            findD(d), findD(n / d);
        }
    }
public:
    Factorization(): prime{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}{}
    vector<ll> get(ll n, bool need_unique){
        res.clear();
        srand(time(NULL));
        findD(n);
        sort(res.begin(), res.end());
        if (need_unique){
            int n = unique(res.begin(), res.end()) - res.begin();
            while ((int)res.size() > n)
                res.pop_back();
        }
        return res;
    }
};
```

2.10 博弈论

2.11 数学知识

2.11.1 循环矩阵乘法

定义：方阵，下一行为上一行的右循环移位。关于加法、乘法封闭。乘法规则：

$$c_k = \sum_{(i+j) \bmod n = k} a_i b_j$$

可以只用第一行表示。用行向量乘方便，$aB = c \iff AB = C$。

2.11.2 组合学定理

- (Ramsey) $\forall p \geq r(m,n), K_p \to K_m, K_n$.
- (Dilworth) 最小链覆盖的链数等于最长反链长度，反之亦然。

3 杂项

3.1 Zeller 公式

输入年月日，告诉是周几，返回 1 就是星期一，7 就是星期天。

```
int weekd(int y, int m, int d){
    int yc = y / 100, yy = y % 100;
    if (m <= 2) m += 12;
    int t = 7 + (yy + (yy >> 2) + (yc >> 2) - (yc << 1) + (26 * (m + 1) / 10) + d - 1) % 7;
    return (t > 7 ? t - 7: t);
}
```

3.2 自适应 Simpson 积分

```
double F(double x);
double calc(double l, double r){
    return (F(l) + 4 * F((l + r) / 2.) + F(r)) * (r - l) / 6.;
}
double asr(double l, double r, double A){
    double mid = (l + r) / 2.;
    double sl = calc(l, mid), sr = calc(mid, r);
    if (fabs(sl + sr - A) / 15. < eps) return sl + sr + (sl + sr - A) / 15.;
    return asr(l, mid, sl) + asr(mid, r, sr);
}
double asr(double l, double r){ // call this
    return asr(l, r, calc(l, r));
}
```

3.3 位运算

```
// 枚举 S 的子集
for (int T = S; T > 0; T = (T - 1) & S) ;
// 格雷码: i ^ (i >> 1)
```

```
__builtin_popcount(unsigned int x) // x 中 1 的个数
__builtin_ctz(unsigned int x) // x 末尾 0 的个数
__builtin_clz(unsigned int x) // x 前导 0 的个数
__builtin_ffs(unsigned int x) // log2(lowbit(x)) + 1
// 三进制向量运算，x0 记录 x 为 1 的位，x1 记录 x 为 2 的位
res_0 = (x1 & y1) | (~(x1 | y1) & (x0 ^ y0));
res_1 = (x0 & y0) | (~(x0 | y0) & (x1 ^ y1)); // 不进位加
dot(x, y) = (2 * __builtin_popcount((x0 & y1) | (x1 & y0)) + __builtin_popcount((x0 & y0) | (x1 & y1))) % 3 // 内积
```

3.4 Python 读到 EOF

```python
import sys
while True:
    line = sys.stdin.readline()
    if not line:
        break
    # do something
```