# Algorithm Library

st1vdy

2021 年 10 月 20 日

# 目录

# 1 多项式

## 1.1 FFT - tourist

```
/* copy from tourist */
namespace FFT {
    typedef double dbl;

    struct num {
        dbl x, y;
        num() { x = y = 0; }
        num(dbl x, dbl y) : x(x), y(y) { }
    };

    inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
    inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
    inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
    inline num conj(num a) { return num(a.x, -a.y); }

    int base = 1;
    vector<num> roots = { {0, 0}, {1, 0} };
    vector<int> rev = { 0, 1 };

    const dbl PI = acosl(-1.0);

    void ensure_base(int nbase) {
        if (nbase <= base) {
            return;
        }
        rev.resize(1 << nbase);
        for (int i = 0; i < (1 << nbase); i++) {
            rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
        }
        roots.resize(1 << nbase);
        while (base < nbase) {
            dbl angle = 2 * PI / (1 << (base + 1));
            for (int i = 1 << (base - 1); i < (1 << base); i++) {
                roots[i << 1] = roots[i];
                dbl angle_i = angle * (2 * i + 1 - (1 << base));
                roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
            }
            base++;
        }
```

```
    }

    void fft(vector<num>& a, int n = -1) {
        if (n == -1) {
            n = a.size();
        }
        assert((n & (n - 1)) == 0);
        int zeros = __builtin_ctz(n);
        ensure_base(zeros);
        int shift = base - zeros;
        for (int i = 0; i < n; i++) {
            if (i < (rev[i] >> shift)) {
                swap(a[i], a[rev[i] >> shift]);
            }
        }
        for (int k = 1; k < n; k <<= 1) {
            for (int i = 0; i < n; i += 2 * k) {
                for (int j = 0; j < k; j++) {
                    num z = a[i + j + k] * roots[j + k];
                    a[i + j + k] = a[i + j] - z;
                    a[i + j] = a[i + j] + z;
                }
            }
        }
    }

    vector<num> fa, fb;

    vector<long long> multiply(vector<int>& a, vector<int>& b) {
        int need = a.size() + b.size() - 1;
        int nbase = 1;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if (sz > (int)fa.size()) {
            fa.resize(sz);
        }
        for (int i = 0; i < sz; i++) {
            int x = (i < (int)a.size() ? a[i] : 0);
            int y = (i < (int)b.size() ? b[i] : 0);
            fa[i] = num(x, y);
        }
        fft(fa, sz);
```

```
        num r(0, -0.25 / (sz >> 1));
        for (int i = 0; i <= (sz >> 1); i++) {
            int j = (sz - i) & (sz - 1);
            num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
            if (i != j) {
                fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
            }
            fa[i] = z;
        }
        for (int i = 0; i < (sz >> 1); i++) {
            num A0 = (fa[i] + fa[i + (sz >> 1)]) * num(0.5, 0);
            num A1 = (fa[i] - fa[i + (sz >> 1)]) * num(0.5, 0) * roots[(sz
                >> 1) + i];
            fa[i] = A0 + A1 * num(0, 1);
        }
        fft(fa, sz >> 1);
        vector<long long> res(need);
        for (int i = 0; i < need; i++) {
            if (i % 2 == 0) {
                res[i] = fa[i >> 1].x + 0.5;
            } else {
                res[i] = fa[i >> 1].y + 0.5;
            }
        }
        return res;
    }

    vector<long long> square(const vector<int>& a) {
        int need = a.size() + a.size() - 1;
        int nbase = 1;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if ((sz >> 1) > (int)fa.size()) {
            fa.resize(sz >> 1);
        }
        for (int i = 0; i < (sz >> 1); i++) {
            int x = (2 * i < (int)a.size() ? a[2 * i] : 0);
            int y = (2 * i + 1 < (int)a.size() ? a[2 * i + 1] : 0);
            fa[i] = num(x, y);
        }
        fft(fa, sz >> 1);
        num r(1.0 / (sz >> 1), 0.0);
```

```
        for (int i = 0; i <= (sz >> 2); i++) {
            int j = ((sz >> 1) - i) & ((sz >> 1) - 1);
            num fe = (fa[i] + conj(fa[j])) * num(0.5, 0);
            num fo = (fa[i] - conj(fa[j])) * num(0, -0.5);
            num aux = fe * fe + fo * fo * roots[(sz >> 1) + i] * roots[(sz >> 1) + i];
            num tmp = fe * fo;
            fa[i] = r * (conj(aux) + num(0, 2) * conj(tmp));
            fa[j] = r * (aux + num(0, 2) * tmp);
        }
        fft(fa, sz >> 1);
        vector<long long> res(need);
        for (int i = 0; i < need; i++) {
            if (i % 2 == 0) {
                res[i] = fa[i >> 1].x + 0.5;
            } else {
                res[i] = fa[i >> 1].y + 0.5;
            }
        }
        return res;
    }

    vector<int> multiply_mod(vector<int>& a, vector<int>& b, int m, int eq = 0) {
        int need = a.size() + b.size() - 1;
        int nbase = 0;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if (sz > (int)fa.size()) {
            fa.resize(sz);
        }
        for (int i = 0; i < (int)a.size(); i++) {
            int x = (a[i] % m + m) % m;
            fa[i] = num(x & ((1 << 15) - 1), x >> 15);
        }
        fill(fa.begin() + a.size(), fa.begin() + sz, num{ 0, 0 });
        fft(fa, sz);
        if (sz > (int) fb.size()) {
            fb.resize(sz);
        }
        if (eq) {
            copy(fa.begin(), fa.begin() + sz, fb.begin());
```

```
        } else {
            for (int i = 0; i < (int)b.size(); i++) {
                int x = (b[i] % m + m) % m;
                fb[i] = num(x & ((1 << 15) - 1), x >> 15);
            }
            fill(fb.begin() + b.size(), fb.begin() + sz, num{ 0, 0 });
            fft(fb, sz);
        }
        dbl ratio = 0.25 / sz;
        num r2(0, -1);
        num r3(ratio, 0);
        num r4(0, -ratio);
        num r5(0, 1);
        for (int i = 0; i <= (sz >> 1); i++) {
            int j = (sz - i) & (sz - 1);
            num a1 = (fa[i] + conj(fa[j]));
            num a2 = (fa[i] - conj(fa[j])) * r2;
            num b1 = (fb[i] + conj(fb[j])) * r3;
            num b2 = (fb[i] - conj(fb[j])) * r4;
            if (i != j) {
                num c1 = (fa[j] + conj(fa[i]));
                num c2 = (fa[j] - conj(fa[i])) * r2;
                num d1 = (fb[j] + conj(fb[i])) * r3;
                num d2 = (fb[j] - conj(fb[i])) * r4;
                fa[i] = c1 * d1 + c2 * d2 * r5;
                fb[i] = c1 * d2 + c2 * d1;
            }
            fa[j] = a1 * b1 + a2 * b2 * r5;
            fb[j] = a1 * b2 + a2 * b1;
        }
        fft(fa, sz);
        fft(fb, sz);
        vector<int> res(need);
        for (int i = 0; i < need; i++) {
            long long aa = fa[i].x + 0.5;
            long long bb = fb[i].x + 0.5;
            long long cc = fa[i].y + 0.5;
            res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
        }
        return res;
    }

    vector<int> square_mod(vector<int>& a, int m) {
```

```
        return multiply_mod(a, a, m, 1);
    }
};
```

### 1.2 形式幂级数

## 2 数论

### 2.1 简单的防爆模板

```
namespace SimpleMod {
    constexpr int MOD = (int)1e9 + 7;
    inline int norm(long long a) { return (a % MOD + MOD) % MOD; }
    inline int add(int a, int b) { return a + b >= MOD ? a + b - MOD : a + b
        ; }
    inline int sub(int a, int b) { return a - b < 0 ? a - b + MOD : a - b; }
    inline int mul(int a, int b) { return (int)((long long)a * b % MOD); }
    inline int powmod(int a, long long b) {
        int res = 1;
        while (b > 0) {
            if (b & 1) res = mul(res, a);
            a = mul(a, a);
            b >>= 1;
        }
        return res;
    }
    inline int inv(int a) {
        a %= MOD;
        if (a < 0) a += MOD;
        int b = MOD, u = 0, v = 1;
        while (a) {
            int t = b / a;
            b -= t * a; swap(a, b);
            u -= t * v; swap(u, v);
        }
        assert(b == 1);
        if (u < 0) u += MOD;
        return u;
    }
}
```

## 2.2 筛法

### 2.2.1 线性素数筛

```cpp
vector<bool> isPrime; // true 表示非素数  false 表示是素数
vector<int> prime; // 保存素数
int sieve(int n) {
    isPrime.resize(n + 1, false);
    isPrime[0] = isPrime[1] = true;
    for (int i = 2; i <= n; i++) {
        if (!isPrime[i]) prime.emplace_back(i);
        for (int j = 0; j < (int)prime.size() && prime[j] * i <= n; ++j) {
            isPrime[prime[j] * i] = true;
            if (!(i % prime[j])) break;
        }
    }
    return (int)prime.size();
}
```

### 2.3 欧拉定理

### 2.4 欧拉函数

### 2.5 中国剩余定理

#### 2.5.1 CRT

#### 2.5.2 EXCRT

### 2.6 BSGS

## 3 线性代数

### 3.1 高斯-约旦消元法

### 3.2 高斯消元法-bitset

### 3.3 线性基

### 3.4 矩阵树定理

### 3.5 LGV 引理

## 4 组合数学

### 4.1 组合数预处理

### 4.2 卢卡斯定理

### 4.3 小球盒子模型

### 4.4 斯特林数

## 5 博弈论

## 6 其他数学

### 6.1 蔡勒公式

## 7 图论

### 7.1 并查集

```
struct dsu {
private:
    // number of nodes
    int n;
    // root node: -1 * component size
    // otherwise: parent
    std::vector<int> pa;
```

```cpp
public:
    dsu(int n_ = 0) : n(n_), pa(n_, -1) {}
    // find node x's parent
    int find(int x) {
        return pa[x] < 0 ? x : pa[x] = find(pa[x]);
    }
    // merge node x and node y
    // if x and y had already in the same component, return false, otherwise
        return true
    // Implement (union by size) + (path compression)
    bool unite(int x, int y) {
        int xr = find(x), yr = find(y);
        if (xr != yr) {
            if (-pa[xr] < -pa[yr]) std::swap(xr, yr);
            pa[xr] += pa[yr];
            pa[yr] = xr; // y -> x
            return true;
        }
        return false;
    }
    // size of the connected component that contains the vertex x
    int size(int x) {
        return -pa[find(x)];
    }
};
```

### 7.2 最小树形图

### 7.3 最近公共祖先

### 7.4 最大流

### 7.5 全局最小割

### 7.6 二分图最大权匹配

### 7.7 一般图最大匹配

### 7.8 2-sat

### 7.9 最大团

## 8 字符串

### 8.1 KMP

### 8.2 Manacher

### 8.3 Trie

### 8.4 01-Trie

## 9 计算几何