FFT&NTT&FWT

FFT

- 用途：加速多项式乘法

  /卷积：$\sum_{i=0}^{2n-1}(\sum_{j=0}^{i} a_j * b_{i-j})x^i$

  $C_k = \sum_{i+j=k} A_i * B_j$

- 复杂度：$O(nlog_2n)$
- 注意事项：

  n 需补成2 的幂（n 必须超过a 和b 的最高指数之和）

- 模板：
  - 求A*B

```cpp
#include<bits/stdc++.h>
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i <
_##i; ++i)
using namespace std;
typedef double LD;
const int Maxn = 2e5 + 10; //数组开4倍
const LD PI = acos(-1);
struct C {
    LD r, i;    //equal a+bi
    C(LD r = 0, LD i = 0) : r(r), i(i) {}
};
C operator+(const C &a, const C &b) {
    return C(a.r + b.r, a.i + b.i);
}
C operator-(const C &a, const C &b) {
    return C(a.r - b.r, a.i - b.i);
}
C operator*(const C &a, const C &b) {
    return C(a.r * b.r - a.i * b.i, a.r * b.i + a.i * b.r);
}
void FFT(C x[], int n, int p) {
    for (int i = 0, t = 0; i < n; ++i) {
        if (i > t) swap(x[i], x[t]);
        for (int j = n >> 1; (t ^= j) < j; j >>= 1);
    }
    for (int h = 2; h <= n; h <<= 1) {
        C wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
        for (int i = 0; i < n; i += h) {
            C w(1, 0), u;
            for (int j = i, k = h >> 1; j < i + k; ++j) {
                u = x[j + k] * w;
                x[j + k] = x[j] - u;
                x[j] = x[j] + u;
                w = w * wn;
            }
        }
    }
    if (p == -1)    //IFFT
        FOR (i, 0, n)x[i].r /= n;
}
void conv(C a[], C b[], int n) {
    FFT(a, n, 1);
    FFT(b, n, 1);
    FOR (i, 0, n)a[i] = a[i] * b[i];
    FFT(a, n, -1);
}
char str1[Maxn / 2], str2[Maxn / 2];
C x1[Maxn], x2[Maxn];
int sum[Maxn];
int main() {
    while (cin >> str1 >> str2) {
        int len1 = strlen(str1);
        int len2 = strlen(str2);
        int len = 1;
        while (len < len1 * 2 || len < len2 * 2) {
            len <<= 1;
        }
        for (int i = 0; i < len1; ++i)
            x1[i] = C(str1[len1 - 1 - i] - '0', 0);
        for (int i = len1; i < len; ++i)
            x1[i] = C(0, 0);
        for (int i = 0; i < len2; ++i)
            x2[i] = C(str2[len2 - 1 - i] - '0', 0);
        for (int i = len2; i < len; ++i)
            x2[i] = C(0, 0);
        conv(x1, x2, len);
        for (int i = 0; i < len; i++) {
            sum[i] = (int) (x1[i].r + 0.5);
        }
        for (int i = 0; i < len; i++) {
            sum[i + 1] += sum[i] / 10;
            sum[i] %= 10;
        }
        len = len1 + len2 - 1;
        while (sum[len] <= 0 && len > 0) {
            len--;
        }
        for (int i = len; i >= 0; i--) {
            printf("%c", sum[i] + '0');
        }
        putchar('\n');
    }
    return 0;
}
```

- 给一个n次多项式F(x)和一个m次多项式G(x)，输出它们的卷积

```
#include<bits/stdc++.h>
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y);
i < _##i; ++i)
using namespace std;
typedef double LD;
const int Maxn = 4e6 + 10;
const LD PI = acos(-1);
struct C {    //数组开4倍
    LD r, i;    //equal a+bi
    C(LD r = 0, LD i = 0) : r(r), i(i) {}
};
C operator+(const C &a, const C &b) {
    return C(a.r + b.r, a.i + b.i);
}
C operator-(const C &a, const C &b) {
    return C(a.r - b.r, a.i - b.i);
}
C operator*(const C &a, const C &b) {
    return C(a.r * b.r - a.i * b.i, a.r * b.i + a.i * b.r);
}
void FFT(C x[], int n, int p) {
    for (int i = 0, t = 0; i < n; ++i) {
        if (i > t) swap(x[i], x[t]);
        for (int j = n >> 1; (t ^= j) < j; j >>= 1);
    }
    for (int h = 2; h <= n; h <<= 1) {
        C wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
        for (int i = 0; i < n; i += h) {
            C w(1, 0), u;
            for (int j = i, k = h >> 1; j < i + k; ++j) {
                u = x[j + k] * w;
                x[j + k] = x[j] - u;
                x[j] = x[j] + u;
                w = w * wn;
            }
        }
    }
    if (p == -1)    //IFFT
        FOR (i, 0, n)x[i].r /= n;
}
void conv(C a[], C b[], int n) {
    FFT(a, n, 1);
    FFT(b, n, 1);
    FOR (i, 0, n)a[i] = a[i] * b[i];
    FFT(a, n, -1);
}
int a[Maxn / 2], b[Maxn / 2];
C x1[Maxn], x2[Maxn];
int sum[Maxn];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    n++, m++;
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (int j = 0; j < m; j++)
        scanf("%d", &b[j]);
    int len = 1;
    while (len < n * 2 || len < m * 2) {
        len <<= 1;
    }
    for (int i = 0; i < n; ++i)
        x1[i] = C(a[n - 1 - i], 0);
    for (int i = n; i < len; ++i)
```

```
        x1[i] = C(0, 0);
    for (int i = 0; i < m; ++i)
        x2[i] = C(b[m - 1 - i], 0);
    for (int i = m; i < len; ++i)
        x2[i] = C(0, 0);
    conv(x1, x2, len);
    for (int i = 0; i < len; i++) {
        sum[i] = (int) (x1[i].r + 0.5);
    }
    for (int i = n + m - 2; i >= 0; i--)
        cout << sum[i] << " ";
    cout << endl;
    return 0;
}
```

NTT

- 用途：类似于FFT，拿原根来代替单位根，但是仅仅能够处理多项式系数都是整数的情况
- 模数有限制，NTT题的模数通常都是相同的998244353
- 其实这些模数的原根通常都是3
- 板子：

```
#include<bits/stdc++.h>
#define LL long long
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i <
_##i; ++i)
using namespace std;
const int N = 4e6 + 10;  //4倍
const int MOD=998244353; //模数
const int G=3;//原根
LL wn[N << 2], rev[N << 2];
LL bin(LL a,LL b,LL P){
    LL ret=1%P;
    while(b){
        if (b&1) ret=ret*a%P;
        a=a*a%P;
        b>>=1;
    }
    return ret;
}
int NTT_init(int n_) {
    int step = 0; int n = 1;
    for ( ; n < n_; n <<= 1) ++step;
    FOR (i, 1, n)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
    int g = bin(G, (MOD - 1) / n, MOD);
    wn[0] = 1;
    for (int i = 1; i <= n; ++i)
        wn[i] = wn[i - 1] * g % MOD;
    return n;
}
LL get_inv(LL n,LL P){
    return bin(n,P-2,P);
}
void NTT(LL a[], int n, int f) {
    FOR (i, 0, n) if (i < rev[i])
            std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            int t = n / (k << 1);
            FOR (j, 0, k) {
                LL w = f == 1 ? wn[t * j] : wn[n - t * j];
                LL x = a[i + j];
                LL y = a[i + j + k] * w % MOD;
                a[i + j] = (x + y) % MOD;
                a[i + j + k] = (x - y + MOD) % MOD;
            }
        }
    }
    if (f == -1) {
        LL ninv = get_inv(n, MOD);
        FOR (i, 0, n)
            a[i] = a[i] * ninv % MOD;
    }
}
void conv(LL a[], LL b[], int n) {
    NTT(a, n, 1);
    NTT(b, n, 1);
    FOR (i, 0, n)a[i] = a[i] * b[i];
    NTT(a, n, -1);
}
LL x1[N],x2[N];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    n++, m++;
    for (int i = 0; i < n; i++)
        scanf("%lld", &x1[i]);
    for (int j = 0; j < m; j++)
        scanf("%lld", &x2[j]);
    int lenn=NTT_init(n+m);
    conv(x1, x2, lenn);
    for (int i =0; i <=n+m-2; i++)
        cout << x1[i] << " ";
    cout << endl;
    return 0;
}
```

FWT

- 用途:

  $C_k = \sum_{i \& j=k} A_i B_j$

  $C_k = \sum_{i | j=k} A_i B_j$

  $C_k = \sum_{i\ xor\ j=k} A_i B_j$

- 复杂度$O(n*2^n)$ n为二进制位数
- FWT 完后需要先模一遍

```
template<typename T>
void fwt(LL a[], int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
            FOR (j, 0, d)
                f(a[i + j], a[i + j + d]);
}

void AND(LL& a, LL& b) { a += b; }
void OR(LL& a, LL& b) { b += a; }
void XOR (LL& a, LL& b) {
    LL x = a, y = b;
    a = (x + y) % MOD;
    b = (x - y + MOD) % MOD;
}
void rAND(LL& a, LL& b) { a -= b; }
void rOR(LL& a, LL& b) { b -= a; }
void rXOR(LL& a, LL& b) {
    static LL INV2 = (MOD + 1) / 2;
    LL x = a, y = b;
    a = (x + y) * INV2 % MOD;
    b = (x - y + MOD) * INV2 % MOD;
}
```

FWT 子集卷积

- S,T是集合，求C：$C_S = \sum_{S \subset T} A_T \times B_{S-T}$
- 设$A_{i,S} = [|S| = i] \times A_S$
- 令$C_i = \sum_{j<=i} A_j \mid B_{i-j}$
- 这样就把问题变成了or卷积
- 由于我们只关心满足$|T| = i$的 $C_{i,T}$，所以是不会多算的；
- 时间复杂度:$O(n^2 \times 2^n)$

```
//popcount-计算二进制里面有多少个1
//可直接使用__builtin_popcount()
a[popcount(x)][x] = A[x]
b[popcount(x)][x] = B[x]
fwt(a[i]) fwt(b[i])
c[i + j][x] += a[i][x] * b[j][x]
rfwt(c[i])
ans[x] = c[popcount(x)][x]
```