快速数论变换

简介

数论变换（number-theoretic transform, NTT）是离散傅里叶变换（DFT）在数论基础上的实现；**快速数论变换**（fast number-theoretic transform, FNTT）是 快速傅里叶变换（FFT）在数论基础上的实现。

数论变换 是一种计算卷积（convolution）的快速算法。最常用算法就包括了前文提到的快速傅里叶变换。然而快速傅立叶变换具有一些实现上的缺点，举例来说，资料向量必须乘上复数系数的矩阵加以处理，而且每个复数系数的实部和虚部是一个正弦及余弦函数，因此大部分的系数都是浮点数，也就是说，必须做复数而且是浮点数的运算，因此计算量会比较大，而且浮点数运算产生的误差会比较大。

NTT 解决的是多项式乘法带模数的情况，可以说有些受模数的限制，数也比较大。目前最常见的模数是 998244353。

前置知识

学习数论变换需要前置知识：离散傅里叶变换、生成子群、原根、离散对数。相关知识可以在对应页面中学习，此处不再赘述。

定义

数论变换

在数学中，NTT 是关于任意 环 上的离散傅立叶变换（DFT）。在有限域的情况下，通常称为数论变换（NTT）。

数论变换（NTT）是通过将离散傅立叶变换化为 $F=\mathbb{Z}/p$，整数模质数 p。这是一个 **有限域**，只要 n 可除 $p-1$，就存在本原 n 次方根，所以我们有 $p=\xi n+1$ 对于 正整数 ξ。具体来说，对于质数 $p=qn+1,(n=2^m)$，原根 g 满足 $g^{qn}\equiv 1 \pmod p$, 将 $g_n=g^q \pmod p$ 看做 ω_n 的等价，则其满足相似的性质，比如 $g_n^n\equiv 1 \pmod p, g_n^{n/2}\equiv -1 \pmod p$。

因为这里涉及到数论变化，所以 N（为了区分 FFT 中的 n，我们把这里的 n 称为 N）可以比 FFT 中的 n 大，但是只要把 $\frac{qN}{n}$ 看做这里的 q 就行了，能够避免大小问题。

常见的有：

$$p=167772161=5\times 2^{25}+1, g=3$$

$$p = 469762049 = 7 \times 2^{26} + 1, g = 3$$

$$p = 754974721 = 3^2 \times 5 \times 2^{24} + 1, g = 11$$

$$p = 998244353 = 7 \times 17 \times 2^{23} + 1, g = 3$$

$$p = 1004535809 = 479 \times 2^{21} + 1, g = 3$$

就是 g^{qn} 的等价 $\mathrm{e}^{2\pi i n}$。

迭代到长度 l 时 $g_l = g^{\frac{p-1}{l}}$，或者 $\omega_n = g_l = g_N^{\frac{N}{l}} = g_N^{\frac{p-1}{l}}$。

快速数论变换

快速数论变换（FNTT）是数论变换（NTT）增加分治操作之后的快速算法。

快速数论变换使用的分治办法，与快速傅里叶变换使用的分治办法完全一致。这意味着，只需在快速傅里叶变换的代码基础上进行简单修改，即可得到快速数论变换的代码。

在算法竞赛中常提到的 NTT 一词，往往实际指的是快速数论变换，一般默认「数论变换」是指「快速数论变换」。

这样简写的逻辑与快速傅里叶变换相似。事实上，「快速傅里叶变换」（FFT）一词指的是「快速离散傅里叶变换」（FDFT），但由于「快速」只能作用于离散，甚至是本原单位根阶数为 2 的幂的特殊情形，不能作用于连续，因此「离散」一词被省略掉，FDFT 变为 FFT，即 FFT 永远指的是特殊的离散情形。

数论变换或快速数论变换是在取模意义下进行的操作，不存在连续的情形，永远是离散的，自然也无需提到离散一词。

在算法领域，不进行提速的操作是无意义的。在快速傅里叶变换中介绍 DFT 一词，是因为 DFT 在信号处理、图像处理领域也有其他的具体应用，同时 DFT 也是 FFT 的原理或前置知识。

在不引起混淆的情形下，常用 NTT 来代指 FNTT。为了不引起下文进一步介绍的混淆，下文的 NTT 与 FNTT 两个词进行了分离。

DFT、FFT、NTT、FNTT 的具体关系是：

- 在 DFT 与 NTT 的基础上，增加分治操作，得到 FFT 与 FNTT。分治操作的办法与原理，可以参见快速傅里叶变换一文。
- 在 DFT 与 FFT 的基础上，将复数加法与复数乘法替换为模 p 意义下的加法和乘法，一般大小限制在 0 到 $p-1$ 之间；将本原单位根改为模 p 意义下的相同阶数的本原单位根，阶数为 2 的幂，即可得到 NTT 与 FNTT。

由于替换的运算只涉及加法和乘法，因此 DFT、FFT、NTT、FNTT 拥有相同的原理，均在满足加法与乘法的环上进行，无需域上满足除法运算的更加严格的条件。

事实上，只要拥有原根，即群论中的生成元，该模数下的 NTT 或 FNTT 即可进行。考虑到模数为 1、2 和 4 的情形太小，不具有实际意义，对于奇素数 p 和正整数 α，只要给出模数为 p^α 和 $2p^\alpha$ 的原根 g，采用同样的办法，则 NTT 或 FNTT 仍然可以进行。

模板

下面是一个大数相乘的模板，参考来源。

参考代码

```cpp
#include <algorithm>
#include <bitset>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <string>
#include <vector>
using namespace std;

int read() {
  int x = 0, f = 1;
  char ch = getchar();
  while (ch < '0' || ch > '9') {
    if (ch == '-') f = -1;
    ch = getchar();
  }
  while (ch <= '9' && ch >= '0') {
    x = 10 * x + ch - '0';
    ch = getchar();
  }
  return x * f;
}

void print(int x) {
  if (x < 0) putchar('-'), x = -x;
  if (x >= 10) print(x / 10);
  putchar(x % 10 + '0');
}

constexpr int N = 300100, P = 998244353;

int qpow(int x, int y) {
  int res(1);
  while (y) {
    if (y & 1) res = 1ll * res * x % P;
    x = 1ll * x * x % P;
    y >>= 1;
  }
  return res;
}

int r[N];
```

```cpp

void ntt(int *x, int lim, int opt) {
  int i, j, k, m, gn, g, tmp;
  for (i = 0; i < lim; ++i)
    if (r[i] < i) swap(x[i], x[r[i]]);
  for (m = 2; m <= lim; m <<= 1) {
    k = m >> 1;
    gn = qpow(3, (P - 1) / m);
    for (i = 0; i < lim; i += m) {
      g = 1;
      for (j = 0; j < k; ++j, g = 1ll * g * gn % P) {
        tmp = 1ll * x[i + j + k] * g % P;
        x[i + j + k] = (x[i + j] - tmp + P) % P;
        x[i + j] = (x[i + j] + tmp) % P;
      }
    }
  }
  if (opt == -1) {
    reverse(x + 1, x + lim);
    int inv = qpow(lim, P - 2);
    for (i = 0; i < lim; ++i) x[i] = 1ll * x[i] * inv % P;
  }
}

int A[N], B[N], C[N];

char a[N], b[N];

int main() {
  int i, lim(1), n;
  scanf("%s", a);
  n = strlen(a);
  for (i = 0; i < n; ++i) A[i] = a[n - i - 1] - '0';
  while (lim < (n << 1)) lim <<= 1;
  scanf("%s", b);
  n = strlen(b);
  for (i = 0; i < n; ++i) B[i] = b[n - i - 1] - '0';
  while (lim < (n << 1)) lim <<= 1;
  for (i = 0; i < lim; ++i) r[i] = (i & 1) * (lim >> 1) + (r[i
>> 1] >> 1);
  ntt(A, lim, 1);
  ntt(B, lim, 1);
  for (i = 0; i < lim; ++i) C[i] = 1ll * A[i] * B[i] % P;
  ntt(C, lim, -1);
  int len(0);
  for (i = 0; i < lim; ++i) {
    if (C[i] >= 10) len = i + 1, C[i + 1] += C[i] / 10, C[i] %=
10;
    if (C[i]) len = max(len, i);
  }
  while (C[len] >= 10) C[len + 1] += C[len] / 10, C[len] %= 10,
len++;
```

```
    for (i = len; ~i; --i) putchar(C[i] + '0');
    puts("");
    return 0;
  }
```

参考资料与拓展阅读

1. FWT（快速沃尔什变换）零基础详解 qaq（ACM/OI）
2. FFT（快速傅里叶变换）0 基础详解！附 NTT（ACM/OI）
3. Number-theoretic transform(NTT) - Wikipedia
4. Tutorial on FFT/NTT—The tough made simple. (Part 1)

本页面最近更新：2025/8/18 22:40:35，更新历史

发现错误？想一起完善？ 在 GitHub 上编辑此页！

本页面贡献者：Ir1d, Tiphereth-A, Enter-tainer, Xeonacid, ChungZH, isdanni, ouuan, shuzhouliu, XuYueming520, Yukimaikoriya, 383494, billchenchina, c-forrest, CCXXXI, GeZiyue, Great-designer, henryrabbit, killcerr, ksyx, O-Omega, ranwen, Saisyc, sshwy, tigerruanyifan, TrisolarisHD, YifanRuan