Nanjing University

ACM-ICPC Codebook 2

Number Theory
Linear Algebra
Combinatorics

August 1, 2017

Contents

1 Number Theory

1.1 Modulo operations

1.1.1 Modular exponentiation (fast power-mod)

Calculate $b^e \bmod m$.

Time complexity: $O(\log e)$

```
LL powmod(LL b, LL e, LL m){
    LL r = 1;
    while (e){
        if (e & 1) r = r * b % m;
        b = b * b % m;
        e >>= 1;
    }
    return r;
}
```

1.1.2 Mathematical modulo operation

The result has the same sign as divisor.

```
inline LL mathmod(LL a, LL b){
    return (a % b + b) % b;
}
```

1.2 Extended Euclidian algorithm

Solve $ax + by = g = \gcd(a, b)$ w.r.t. x, y.

If (x_0, y_0) is an integer solution of $ax + by = g = \gcd(x, y)$, then every integer solution of it can be written as $(x_0 + kb', y_0 - ka')$, where $a' = a/g$, $b' = b/g$, and k is arbitrary integer.

⚠ x and y must be positive.

Usage:

`exgcd(a, b, g, x, y)`	Find a special solution to $ax+by = g = \gcd(a, b)$.

Time complexity: $O(\log \min\{a, b\})$

```
void exgcd(LL a, LL b, LL &g, LL &x, LL &y){
    if (!b) g = a, x = 1, y = 0;
    else exgcd(b, a % b, g, y, x), y -= x * (a / b);
}
```

1.2.1 Modular multiplicative inverse

An integer a has modular multiplicative inverse w.r.t. the modulus m, iff $\gcd(a, m) = 1$. Assume the inverse is x, then

$$ax \equiv 1 \mod m.$$

Call `exgcd(a, m, g, x, y)`, if $g = 1$, $x + km$ is the modular multiplicative inverse of a w.r.t. the modulus m.

```
inline LL minv(LL a, LL m){
    LL g, x, y;
    exgcd(a, m, g, x, y);
    return (x % m + m) % m;
}
```

Or, by Fermat's little theorem ($a^{p-1} \equiv 1 \mod p$), when $m = p$ is a prime, the multiplicative inverse can also be written as $a^{-1} = \left(a^{p-2} \bmod p\right)$.

1.3 Primality test (Miller-Rabin)

Test whether n is a prime.

The array `a[]` (excluding sentinel, e.g. `LLONG_MAX`) should be

`{2}`	when $n < 2,047$.
`{2, 7, 61}`	when $n < 4,759,123,141$ (2^{32}).
`{2, 3, 5, 7, 11}`	when $n < 2.1 \times 10^{12}$.
`{2, 325, 9375, 28178, 450775, 9780504, 1795265022}`	when $n < 2^{64}$.

⚠ When n exceeds the range of `int`, the mul-mod and pow-mod operations should be rewritten.

Requirement:
1.1.1 Modular exponentiation (fast power-mod)

Time complexity: $O(\log n)$

```
bool test(LL n){
    if (n < 3) return n==2;
```

```
    // ! The array a[] should be modified if the range of x changes.
    const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
    LL r = 0, d = n-1, x;
    while (~d & 1) d >>= 1, r++;
    for (int i=0; a[i] < n; i++){
        x = powmod(a[i], d, n);
        if (x == 1 || x == n-1) goto next;
        rep (i, r) {
            x = (x * x) % n;
            if (x == n-1) goto next;
        }
        return false;
next:;
    }
    return true;
}
```

1.4 Sieve of Eratosthenes

Usage:

`sieve()`	Generate the table.
`p[i]`	True if i is **not** a prime; otherwise false.

Time complexity: Approximately linear.

```
const int MAXX = 1e7+5;
bool p[MAXX];

void sieve(){
    p[0] = p[1] = 1;
    for (int i = 2; i*i < MAXX; i++) if (!p[i])
        for (int j = i*i; j < MAXX; j+=i) p[j] = true;
}
```

1.5 Number theoretic transform

⚠ The size of the sequence must be some power of 2.
⚠ When performing convolution, the size of the sequence should be doubled. To compute k, one may call `32-__builtin_clz(a+b-1)`, where a and b are the lengths of two sequences.

Usage:

`NTT(k)`	Initialize the structure with maximum sequence length 2^k.
`ntt(a)`	Perform number theoretic transform on sequence a.
`intt(a)`	Perform inverse number theoretic transform on sequence a.
`conv(a, b)`	Convolve sequence a with b.

Time complexity: $O(n \log n)$.

```
const int NMAX = 1<<21;
// 998244353 = 7*17*2^23+1, G = 3
const int P = 1004535809, G = 3; // = 479*2^21+1

LL rev[NMAX];
LL omega[NMAX], oinv[NMAX];
struct NTT{
    int g, g_inv; // g: g_n = G^((P-1)/n)
    int K, N;

    LL powmod(LL b, LL e){
        LL r = 1;
        while (e){
            if (e&1) r = r * b % P;
            b = b * b % P;
            e >>= 1;
        }
        return r;
    }

    NTT(int k){
        K = k; N = 1 << k;
        g = powmod(G, (P-1)/N);
        g_inv = powmod(g, N-1);
        omega[0] = oinv[0] = 1;
        rep (i, N){
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
            if (i){
                omega[i] = omega[i-1] * g % P;
                oinv[i] = oinv[i-1] * g_inv % P;
            }
        }
    }

    void _ntt(LL* a, LL* w){
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int l = 2; l <= N; l *= 2){
            int m = l/2;
            for (LL* p = a; p != a + N; p += l)
                rep (k, m){
                    LL t = w[N/l*k] * p[k+m] % P;
```

```
                p[k+m] = (p[k] - t + P) % P;
                p[k] = (p[k] + t) % P;
            }
        }
    }

    void ntt(LL* a){_ntt(a, omega);}
    void intt(LL* a){
        LL inv = powmod(N, P-2);
        _ntt(a, oinv);
        rep (i, N) a[i] = a[i] * inv % P;
    }

    void conv(LL* a, LL* b){
        ntt(a); ntt(b);
        rep (i, N) a[i] = a[i] * b[i] % P;
        intt(a);
    }
};
```

2 Linear Algebra

2.1 Modular exponentiation of matrices

Calculate $b^e \bmod modular$, where b is a matrix. The modulus is element-wise.

Usage:

`n`	Order of matrices.
`modular`	The divisor in modulo operations.
`m_powmod(b, e)`	Calculate $b^e \bmod modular$. The result is stored in `r`.

Time complexity: $O(n^3 \log e)$

```
const int MAXN = 105;
const LL modular = 1000000007;
int n; // order of matrices

struct matrix{
    LL m[MAXN][MAXN];

    void operator *=(matrix& a){
        static LL t[MAXN][MAXN];
        Rep (i, n){
            Rep (j, n){
```

```
                t[i][j] = 0;
                Rep (k, n){
                    t[i][j] += (m[i][k] * a.m[k][j]) % modular;
                    t[i][j] %= modular;
                }
            }
        }
        memcpy(m, t, sizeof(t));
    }
};

matrix r;
void m_powmod(matrix& b, LL e){
    memset(r.m, sizeof(r.m), 0);
    Rep(i, n)
        r.m[i][i] = 1;
    while (e){
        if (e & 1) r *= b;
        b *= b;
        e >>= 1;
    }
}
```

3 Combinatorics

3.1 Möbius inversion

Möbius function:

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } {p_i}^{a_i} \mid n \text{ where } a_i > 0 \\ (-1)^r & \text{if } n \text{ is the product of } r \text{ distinct primes} \end{cases}$$

If $S_f(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d) S_f(n/d)$.

3.2 Pólya enumeration theorem