NANJING UNIVERSITY

ACM-ICPC Codebook 2

Number Theory
Linear Algebra
Combinatorics

June 10, 2017

Contents

1 Number Theory

1.1 Modulo operations

1.1.1 Modular exponentiation (fast power-mod)

Calculate $b^e \bmod m$.

⚠ Cannot be performed on `long long`, unless use 1.1.3 Modular multiplication on long long .

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;
}
```

1.1.3 Modular multiplication on long long

Calculate $ab \bmod m$, where a, b, m are `long long` integers.

⚠ a, b, m must be non-negative.

Time complexity: $O(\log b)$

```
LL mulmod(LL a, LL b, LL m){
    LL r = 0;
    a %= m; b %= m;
    while(b) {
        if(b & 1) r += a, r %= m;
```

```
        b >>= 1;
        if(a < m - a)
            a <<= 1;
        else
            a -= (m - a);
    }
    return r;
}
```

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 + x) % m;
}
```

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;
        for (int i = 0; i < r; i++) {
            x = (x * x) % n;
            if (x == n-1) goto next;
        }
        return false;
next:;
    }
    return true;
}
```

1.4 Sieve of Eratosthenes

1.5 Chinese remainder theorem

1.6 Quadratic residue

1.6.1 Legendre symbol

For non-negative integer a and **odd** prime p, the Legendre symbol is defined as

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \mid p \\ 1 & \text{if } a \nmid p \text{ and } a \text{ is a quadratic residue modulo } p \\ p-1 & \text{if } a \nmid p \text{ and } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

Call `powmod(a, (p-1)/2, p)` to calculate Legendre symbol.

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];
        for (int i=0; i<n; i++){
            for (int j=0; j<n; j++){
                t[i][j] = 0;
                for (int k=0; k<n; k++){
                    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);
    for (int i=0; i<n; i++)
        r.m[i][i] = 1;
    while (e){
        if (e & 1) r *= b;
        b *= b;
        e >>= 1;
    }
}
```

3 Combinatorics

3.1 Pólya enumeration theorem