NANJING UNIVERSITY

ACM-ICPC Codebook 2

Number Theory
Linear Algebra
Combinatorics

June 18, 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: $O(\log e)$

```
1  LL powmod(LL b, LL e, LL m){
2      LL r = 1;
3      while (e){
4          if (e & 1) r = r * b % m;
5          b = b * b % m;
6          e >>= 1;
7      }
8      return r;
9  }
```

1.1.2 Mathematical modulo operation

The result has the same sign as divisor.

```
1  inline LL mathmod(LL a, LL b){
2      return (a % b + b) % b;
3  }
```

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: $O(\log b)$

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

```
6          b >>= 1;
7          if(a < m - a)
8              a <<= 1;
9          else
10             a -= (m - a);
11     }
12     return r;
13 }
```

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

```
1 void exgcd(LL a, LL b, LL &g, LL &x, LL &y){
2     if (!b) g = a, x = 1, y = 0;
3     else exgcd(b, a % b, g, y, x), y -= x * (a / b);
4 }
```

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.

```
1 inline LL minv(LL a, LL m){
2     LL g, x, y;
3     exgcd(a, m, g, x, y);
4     return (x % m + m) % m;
5 }
```

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: $O(\log n)$

```
1   bool test(LL n){
2       if (n < 3) return n==2;
3       // ! The array a[] should be modified if the range of x changes.
4       const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
5       LL r = 0, d = n-1, x;
6       while (~d & 1) d >>= 1, r++;
7       for (int i=0; a[i] < n; i++){
8           x = powmod(a[i], d, n);
9           if (x == 1 || x == n-1) goto next;
10          rep (i, r) {
11              x = (x * x) % n;
12              if (x == n-1) goto next;
13          }
14          return false;
15  next:;
16      }
17      return true;
18  }
```

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

```
1   const int MAXN = 105;
2   const LL modular = 1000000007;
3   int n; // order of matrices
4
5   struct matrix{
6       LL m[MAXN][MAXN];
7
8       void operator *=(matrix& a){
9           static LL t[MAXN][MAXN];
10          Rep (i, n){
11              Rep (j, n){
12                  t[i][j] = 0;
13                  Rep (k, n){
14                      t[i][j] += (m[i][k] * a.m[k][j]) % modular;
```

```
15                    t[i][j] %= modular;
16                  }
17              }
18          }
19          memcpy(m, t, sizeof(t));
20      }
21  };
22
23  matrix r;
24  void m_powmod(matrix& b, LL e){
25      memset(r.m, sizeof(r.m), 0);
26      Rep(i, n)
27          r.m[i][i] = 1;
28      while (e){
29          if (e & 1) r *= b;
30          b *= b;
31          e >>= 1;
32      }
33  }
```

3   Combinatorics

3.1   Pólya enumeration theorem