Teoría de Números

Tenemos las siguientes funciones:

- `reduce_mod(a,mod)` reduce a a su residuo **positivo** de dividir a entre mod.
- `modulo(a,mod)` regresa reduce_mod(a,mod)
- `pow(a,n)` y `pow_mod(a,n,mod)` regresan a^n y a^n%mod respectivamente.
- `gcd_extended(a,b)` regresa el máximo común divisor d = gcd(a,b) y también la combinación lineal ax+by=d.
- `mod_inverse(a,n)` regresa el inverso modular de a módulo n. Ejemplo: $4 \cdot 3 \equiv 1 \pmod{11}$, así que 4 y 3 son inversos módulo 11.

Además, hay funciones para convertir enteros de una base a otra.

```
#include <algorithm>
#include <cassert>
#include <cmath>
#include <iostream>
#include <numeric>
#include <vector>

using ll = long long;

template <class T = ll, class U = ll>
void reduce_mod(T& a, const U mod)
{
    a %= mod;
    if (a < 0)
        a += mod;
}

template <class T = ll, class U = ll>
T modulo(T a, const U mod)
{
    reduce_mod(a, mod);
    return a;
}

// Can do it for any class that has
template <class T = ll, T identity = 1>
T pow(T a, unsigned long n)
{
    T r = identity;

    while (n > 0)
    {
```

```
        if (n % 2 == 1)
            r *= a;

        n /= 2;
        a *= a;
    }

    return r;
}

ll pow_mod(ll a, unsigned long n, const ll mod)
{
    ll r = 1;
    reduce_mod(a, mod);
    while (n > 0)
    {
        if (n % 2 == 1)
        {
            r *= a;
            reduce_mod(r, mod);
        }

        n /= 2;
        a *= a;
        reduce_mod(a, mod);
    }

    return r;
}

ll gcd(ll a, ll b)
{
    while (b != 0)
    {
        ll r = a % b;
        a = b;
        b = r;
    }
    return a;
}

ll lcm(ll a, ll b) { return a * b / gcd(a, b); }

struct linearcomb
{
```

```
    ll d; // gcd
    ll x; // first coefficient
    ll y; // second coefficient
};

linearcomb gcd_extended(ll a, ll b)
{
    if (b == 0)
        return {a, 1LL, 0LL};

    ll sa = 1, sb = 0, ta = 0, tb = 1, sc, tc;

    do
    {
        auto K = std::div(a, b);

        a = b;
        b = K.rem;

        sc = sa - K.quot * sb;
        sa = sb;
        sb = sc;

        tc = ta - K.quot * tb;
        ta = tb;
        tb = tc;
    } while (b != 0);

    return {a, sa, ta};
}

ll mod_inverse(ll a, const ll n)
{
    ll x = gcd_extended(a, n).x;
    reduce_mod(x, n);
    return x;
}

template <class IntType>
ll InterpretBaseK(ll k, const std::vector<IntType>& bla)
{
    ll suma = 0;
    ll power = 1;

    for (auto it = bla.rbegin(); it != bla.rend(); ++it)
```

```
    {
        suma += power * static_cast<IntType>(*it);
        power *= k;
    }

    return suma;
}

std::vector<int> NumberBaseB(ll n, int b)
{
    std::vector<int> toReturn;

    while (n)
    {
        toReturn.push_back(n % b);
        n /= b;
    }

    std::reverse(toReturn.begin(), toReturn.end());
    return toReturn;
}

using namespace std;

template <class T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& A)
{
    for (const auto& x : A)
        os << x << ' ';
    return os;
}

int main()
{
    cout << "modulo(-37,10) = " << modulo(-37, 10) << endl;
    cout << "7^1000 (mod 5) = " << pow_mod(7, 1000, 5) << endl;

    auto dxy = gcd_extended(30, 55);
    cout << "\ngcd(30,55) = " << dxy.d << " = 30*" << dxy.x << " + 55*" << dxy.y
         << endl;
    cout << "lcm(30,55) = " << lcm(30, 55) << endl;

    cout << "\n1/7 (mod 9) = " << mod_inverse(7, 9) << endl;

    std::vector<int> V = {1, 2, 0, 4};
```

```
    cout << "\n1204_{5} = " << InterpretBaseK(5, V) << endl;
    cout << "10 in base 2: " << NumberBaseB(10, 2) << endl;
    cout << "100 in base 7: " << NumberBaseB(100, 7) << endl;

    return 0;
}
```

Output:

```
modulo(-37,10) = 3
7^1000 (mod 5) = 1
gcd(30,55) = 5
lcm(30,55) = 330
1/7 (mod 9) = 4
10 in base 2: 1 0 1 0
100 in base 7: 2 0 2
1204_{5} = 179
```