Matemáticas.

Material de referencia.

Índice

1. Big Numbers.

1.1. Implementación

```cpp
#include <iostream>
#include <algorithm>
#include <utility>
using namespace std;

typedef string BigInteger;

//Regresa el i-esimo digito de derecha a izquierda de un numero.
int digit(const BigInteger &num, int i) {
    if (i < num.size())
        return num[num.size() - 1 - i] - '0';
    return 0;
}

//Compara dos numeros y regresa: 1 si el primero es mayor; 0 si son iguales; -1 si el segundo es mayor.
int compareTo(const BigInteger &a, const BigInteger &b) {
    for (int i = max(a.size(), b.size()) - 1; i >= 0; --i) {
        if (digit(a, i) > digit(b, i))
            return 1;
        if (digit(b, i) > digit(a, i))
            return -1;
    }
    return 0;
}

//Regresa la suma de dos numeros.
BigInteger sum(const BigInteger &a, const BigInteger &b) {
    BigInteger ans;
    int carry = 0;
    for (int i = 0; i < max(a.size(), b.size()); ++i) {
        carry += digit(a, i) + digit(b, i);
        ans.push_back((carry % 10) + '0');
        carry /= 10;
    }
    if (carry)
        ans.push_back(carry + '0');
    reverse(ans.begin(), ans.end());
    return ans;
}

//Regresa la diferencia de dos numeros. El primer numero debe ser mayor o igual que el segundo.
BigInteger substract(const BigInteger &a, const BigInteger &b) {
    BigInteger ans;
    int carry = 0;
    for (int i = 0; i < a.size(); ++i) {
        carry += digit(a, i) - digit(b, i);
        if (carry >= 0) {
            ans.push_back(carry + '0');
            carry = 0;
        }
```

```
            else {
                ans.push_back(carry + 10 + '0');
                carry = -1;
            }
        }
    while (ans.size() > 1 && ans.back() == '0')
        ans.pop_back();
    reverse(ans.begin(), ans.end());
    return ans;
}

//Regresa el producto de dos numeros (BigInteger x int).
BigInteger multiply(const BigInteger &a, int b) {
    if (b == 0)
        return "0";
    BigInteger ans;
    int carry = 0;
    for (int i = 0; i < a.size(); ++i) {
        carry += digit(a, i) * b;
        ans.push_back((carry % 10) + '0');
        carry /= 10;
    }
    while (carry) {
        ans.push_back((carry % 10) + '0');
        carry /= 10;
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

//Regresa el producto de dos numeros (BigInteger x BigInteger).
BigInteger multiply(const BigInteger &a, const BigInteger &b) {
    BigInteger ans;
    for (int i = 0; i < b.size(); ++i)
        ans = sum(ans, multiply(a, digit(b, i)).append(i, '0'));
    return ans;
}

//Regresa el cociente y el residuo de la division (BigInteger / int).
pair<BigInteger, int> divide(const BigInteger &a, int b) {
    pair<BigInteger, int> ans;
    for (int i = a.size() - 1; i >= 0; --i) {
        ans.second = 10*ans.second + digit(a, i);
        if (!ans.first.empty() || ans.second >= b || i == 0)
            ans.first.push_back((ans.second / b) + '0');
        ans.second %= b;
    }
    return ans;
}

//Regresa el cociente y el residuo de la division (BigInteger / BigInteger).
pair<BigInteger, BigInteger> divide(const BigInteger &a, const BigInteger &b) {
    pair<BigInteger, BigInteger> ans;
    BigInteger table[10];
    for (int i = 0; i < 10; ++i)
```

```
        table[i] = multiply(b, i);
    for (int i = a.size() - 1; i >= 0; --i) {
        int q = 0;
        ans.second.push_back(digit(a, i) + '0');
        while (q < 9 && compareTo(ans.second, table[q + 1]) >= 0)
            ++q;
        if (!ans.first.empty() || q > 0 || i == 0)
            ans.first.push_back(q + '0');
        ans.second = substract(ans.second, table[q]);
    }
    return ans;
}

int main() {
    BigInteger a, b;
    cin >> a >> b;

    cout << a << " + " << b << " = " << sum(a, b) << '\n';
    if (compareTo(a, b) >= 0)
        cout << a << " - " << b << " = " << substract(a, b) << '\n';
    else
        cout << b << " - " << a << " = " << substract(b, a) << '\n';
    cout << a << " * " << b << " = " << multiply(a, b) << '\n';
    cout << a << " = " << b << " * " << divide(a, b).first << " + " << divide(
        a, b).second << '\n';

    return 0;
}
```

Entrada	Salida
1894821 589613	1894821 + 589613 = 2484434 1894821 - 589613 = 1305208 1894821 * 589613 = 1117211094273 1894821 = 589613 * 3 + 125982

2. Test de Primalidad.

Decimos que un entero positivo p es primo si tiene exactamente dos divisores distintos: 1 y p.

2.1. Algoritmo de Miller-Rabin.

```
#include <iostream>
#include <random>
using namespace std;

//Regresa base^expo %mod.
long long power(__int128 base, long long expo, long long mod) {
    if (expo == 0)
        return 1;
    else if (expo % 2)
        return (base * power(base, expo - 1, mod)) % mod;
    else {
        __int128 p = power(base, expo / 2, mod);
        return (p * p) % mod;
    }
}

default_random_engine gen; //Generador de numeros aleatorios.

//Regresa false si n es compuesto y true si es probablemente primo.
bool MillerTest(long long n, long long d) {
    uniform_int_distribution<long long> Rand(2, n - 2);

    __int128 x = power(Rand(gen), d, n);
    if (x == 1 || x == n - 1)
        return true;

    for (; d != n - 1; d *= 2) {
        x = (x * x) % n;
        if (x == 1)
            return false;
        if (x == n - 1)
            return true;
    }
    return false;
}

//Ejecuta el Test de Miller-Rabin varias veces.
bool isProbablePrime(long long n, int attemps) {
    if (n == 2 || n == 3)
        return true;
    if (n == 1 || n == 4)
        return false;

    long long d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    while (attemps--)
        if (!MillerTest(n, d))
```

```
            return false;
    return true;
}

int main() {
    ios_base::sync_with_stdio(0); cin.tie();
    long long n;
    while (cin >> n) {
        if (isProbablePrime(n, 10))
            cout << "Probablemente es primo.\n";
        else
            cout << "No es primo.\n";
    }
    return 0;
}
```

Entrada	Salida
1000000007	Probablemente es primo.
123456789	No es primo.
104729	Probablemente es primo.

3. Factorización en primos.

Sea n un entero mayor que 1, el Teorema Fundamental de la Aritmética afirma que n tiene una única factorización en primos.

3.1. Algoritmo de prueba por división.

Complejidad: $O\left(\pi\left(\sqrt{n}\right)\right)$ donde $\pi(x)$ es el número de primos menores o iguales que x.

```
#include <iostream>
#include <vector>
using namespace std;

#define maxn 10000000      //Raiz cuadrada del mayor numero a factorizar.

long long n;                //Numero a factorizar.
vector<long long> primes;   //Lista de primos.
vector<long long> factors;  //Lista de factores primos de n.

//Encuentra con la Criba de Eratostenes los primos menores que maxn.
void find_primes() {
    vector<bool> sieve(maxn);
    for (long long i = 2; i < maxn; ++i)
        if (!sieve[i]) {
            primes.push_back(i);
            for (long long j = i * i; j < maxn; j += i)
                sieve[j] = true;
        }
}

//Prueba por division.
void prime_factor() {
    factors.clear();
    for (int i = 0; primes[i] * primes[i] <= n; ++i)
        while (n % primes[i] == 0) {
            factors.push_back(primes[i]);
            n /= primes[i];
        }
    if (n != 1)
        factors.push_back(n);
}

int main() {
    ios_base::sync_with_stdio(0); cin.tie();
    find_primes();

    while (cin >> n) {
        prime_factor();
        for (long long p : factors)
            cout << p << ' ';
        cout << '\n';
    }

    return 0;
}
```

Entrada	**Salida**
180	2 2 3 3 5
3500	2 2 5 5 5 7
123456789	3 3 3607 3803
104729	104729

4. Sistema de Ecuaciones Lineales.

Consideremos un sistema de ecuaciones lineales dado por la matriz **A** de $n \times n$ y el vector **b** de dimensión n. Decimos que **x** es solución si $\mathbf{Ax} = \mathbf{b}$.

4.1. Eliminación Gaussiana

Complejidad $O(n^3)$.

```
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

#define maxn 100 //Maximo numero de ecuaciones-incognitas.

int n, m;                       //Dimensiones.
double AugMatrix[maxn][maxn];   //Matriz aumentada.

//Encuentra la forma escalonada reducida de la matriz aumentada [A | B]
//con A de n x n y B de n x m. Regresa el determinante de A.
double GaussianElimination() {
    double det = 1;
    for (int k = 0; k < n; ++k) {
        int r = k;
        for (int i = k + 1; i < n; ++i)
            if (fabs(AugMatrix[i][k]) > fabs(AugMatrix[r][k]))
                r = i;

        if (fabs(AugMatrix[r][k]) < 1e-9)
            return 0;
        if (r != k) {
            for (int j = k; j < n + m; ++j)
                swap(AugMatrix[k][j], AugMatrix[r][j]);
            det *= -1;
        }
        det *= AugMatrix[k][k];

        for (int j = n + m - 1; j >= k; --j) {
            AugMatrix[k][j] /= AugMatrix[k][k];
            for (int i = 0; i < n; ++i)
                if (i != k)
                    AugMatrix[i][j] -= AugMatrix[i][k] * AugMatrix[k][j];
        }
    }
    return det;
}

int main() {
    ios_base::sync_with_stdio(0); cin.tie();
    cin >> n >> m;

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n + m; ++j)
            cin >> AugMatrix[i][j];

```

```
    cout << "Determinante: " << GaussianElimination() << "\nSolucion:\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j)
            cout << AugMatrix[i][j + n] << ' ';
        cout << '\n';
    }
    return 0;
}
```

Entrada	**Salida**
4 1 1 -2 2 -3 15 3 4 -1 1 -6 2 -3 2 -1 17 1 1 -3 -2 -7	Determinante: 142 Solucion: 2 -2 3 -1

5. Sistema de Congruencias Lineales.

Consideremos el sistema de congruencias

$$x \equiv a_1 \pmod{m_1}$$
$$\vdots$$
$$x \equiv a_n \pmod{m_n}$$

con $m_1, \ldots, m_n$ primos relativos por parejas. El Teorema Chino del Residuo afirma que existe una única solución módulo $m_1 \cdots m_n$.

5.1. Teorema Chino del Residuo.

```
#include <iostream>
using namespace std;

#define maxn 100000     //Maximo numero de ecuaciones.

int n;                                   //Numero de ecuaciones.
long long MOD, coef[maxn], mod[maxn];    //Datos de las ecuaciones.

//Algoritmo extendido de Euclides.
long long extendedEuclid(long long a, long long b, long long &x, long long &y)
    {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    else {
        long long gcd = extendedEuclid(b, a %b, y, x);
        y -= (a / b) * x;
        return gcd;
    }
}

//Teorema Chino del Residuo.
long long ChineseRemainder() {
    MOD = 1;
    for (int i = 0; i < n; ++i)
        MOD *= mod[i];

    long long x = 0;
    for (int i = 0; i < n; ++i) {
        long long N = MOD / mod[i], invN, invM;
        extendedEuclid(N, mod[i], invN, invM);
        x = (x + coef[i] * N * invN) %MOD;
        x = (x + MOD) %MOD;
    }

    return x;
}

```

```
int main() {
    ios_base::sync_with_stdio(0); cin.tie();
    cin >> n;

    for (int i = 0; i < n; ++i)
        cin >> coef[i] >> mod[i];

    cout << "x = " << ChineseRemainder() << " (mod " << MOD << ")\n";
    return 0;
}
```

Entrada	**Salida**
3 2 4 3 9 1 5	x = 66 (mod 180)