Criba de Eratóstenes.

Definición

Es un método determinístico para hallar todos los primos hasta determinado número. Consiste en iterar todos los números hasta la raíz cuadrada del número al que se evalúa su primalidad.

Nos dimos cuenta que siempre un número divisible por 2 o un número divisible por 3 se encuentra al lado de cualquier primo, osea, todo primo está al lado de un número múltiplo de 6.
$Primo = 6n \pm 1$
Esto quiere decir también que todo primo cumple la siguiente condición.
$Primo = 4n \pm 1$
Esto último nos servirá para iterar de forma que salvemos un poco más de memoria.

Funciones:

myModule: devuelve el módulo de un número.
squareRoot: establece la raíz cuadrada de un número en una variable pasada por referencia. Usa el algoritmo de radicación cuadrada de los babilonios. Está optimizada con operadores binarios y por lo tanto no toma en cuenta los decimales, también porque no nos interesa precisión, sino aproximación eficiente.
isPrime: itera sobre todos los primos ya encontrados para determinar si el posible primo es divisible por alguno de ellos. retorna un booleano.
eratosthenes_sieve: ejecuta la búsqueda hasta un parámetro de pare y almacena los primos en un vector.

Implementación en C++

```
#include <ostream>
#include <vector>
#include <NTL/ZZ.h>

using namespace std;
using namespace NTL;

ZZ myModule(ZZ &dividend, ZZ &divisor)
{
    const ZZ quotient = dividend / divisor;
    const ZZ remainder = dividend - (divisor * quotient);
    return remainder;
```

```
}

void squareRoot(ZZ number, ZZ &result, unsigned precision = 3)
{
    result = number >> 3;
    if (result != 0)
    {
        ZZ y = number / result;
        do
        {
            result += y;
            result = result >> 1;
            if (result == 0)
                break;
            y = number / result;
        } while (result - y > precision);
    }
}

bool isPrime(vector<ZZ> &primes, ZZ &number)
{
    ZZ top;
    squareRoot(number, top);
    for (vector<ZZ>::iterator it_ptr = primes.begin(); *it_ptr
< top; it_ptr++)
        if (myModule(number, *it_ptr) == 0)
            return false;
    cout << number << ",";
    return true;
}

void eratosthenes_sieve(vector<ZZ> &primes, ZZ &end)
{
    // let's take 11 as the default start
    // 11 = 4n + 3
    // 11 = 6n + 5
    for (ZZ i = ZZ(11); i < end; i += 4)
```

```
    {
        // 4n + 3
        if (isPrime(primes, i))
            primes.push_back(i);
        // 4n + 1
        i += 2;
        if (isPrime(primes, i))
            primes.push_back(i);
    }
}
```

```
    ZZ end = conv<ZZ>("65536"); // 2^16
```

```
❯ time ./sieve > 16-bits.txt
./sieve > 16-bits.txt  0.30s user 0.00s system 97% cpu 0.307 total
```

```
    ZZ end = conv<ZZ>("2793390");
```

```
❯ time ./sieve > arbitrary-limit.txt
./sieve > arbitrary-limit.txt  9.13s user 0.07s system 99% cpu 9.205 total
```

Este es el último número que pudimos probar su primalidad tras más de 15 min, el parámetro de último número por recorrer era un número de 32 bits, osea, 4294967296

```
 1 32-bits.txt                                                    Buffers
1   0439,151990441,151990451,151990
```

el número es: 151990441