CB100

ALGORITMOS Y ESTRUCTURAS DE DATOS

Números Primos

Trabajo Práctico N°1

Autor
Martin J. Klöckner

Legajo
123456

Fecha
03/04/24

Índice

1. Introducción

En este trabajo práctico se desarrolla una aplicación de consola que busca números primos hasta un valor máximo determinado por el usuario, cuando finaliza, los imprime en un archivo de texto plano en el directorio de ejecución de la aplicación.

El lenguaje de programacion utilizado para el desarrollo de la aplicación es C++ y el algoritmo utilizado para hallar los números primos es la Criba de Eratóstenes.

1.1. Números Primos

Los números primos por definición son aquellos números positivos que solo son divisibles por `1` y por si mismos, divisibles en términos de que el resto de la división es nulo. El `0` y el `1` son dos casos particulares en los cuales ambos son considerados no primos, el `0` por razones obvias, no está definida la división por `0`, y el `1` por que solo es divisible por si mismo, lo cual hace que solo sea divisible por un número.

El número `7`, por ejemplo, es un número primo, ya que solo es divisible por `1` y por `7`. El número `8` no es un número primo, ya que ademas de ser divisible por si mismo y por `1`, es divisible por `2` y por `4`.

1.2. Criba de Eratóstenes

La criba de Eratóstenes es un algoritmo sofisticado y eficiente para descartar números no primos de una lista de números. En la Criba de Eratóstenes se itera número a número sobre una lista ordenada y finita de números naturales, comenzando por el primero, si se trata de un número primo, se buscan todos los múltiplos en la lista de números y se descartan, luego se avanza al siguiente número que no haya sido descartado, el cual siempre resulta un número primo, y se descartan sus múltiplos, así sucesivamente hasta el final de la lista.

La criba de Eratóstenes se puede optimizar basándose en la propiedad de que los números no primos (números compuestos), se pueden expresar como el producto de números primos, por ejemplo `4 = 2*2` o `21 = 3*7`, por lo tanto, si un número no es primo, debe tener al menos un factor primo que sea menor o igual que su raíz cuadrada, ya que de lo contrario, el producto de los factores resultaría en un numero mayor.

De la propiedad anterior se deduce que se puede detener la iteración en la lista de números cuando se llega a la raíz cuadrado del numero máximo en la lista, ya que los números no primos por encima de éste tendrán algún factor menor que la raíz cuadrada del numero máximo, de este modo se reduce drásticamente el numero de iteraciones necesarias para descartar los números no primos de la lista.

2. Desarrollo

Para la implementación de la aplicación, como bien se mencionó en la introducción, se utilizó puramente el lenguaje C++, en particular en su versión estándar C98. Para organizar el proceso de compilación se utilizó la herramienta `make`.

2.1. Implementación en C++

El código fuente de la aplicación esta contenido en un solo archivo: `primos.cpp`.

En la primera parte del archivo se definen dos constantes globales `OUTPUT_FILE_PATH`, que representa el nombre del archivo de salida, y `MAXIMO` que representa el largo de la lista de números a analizar.

```
#define OUTPUT_FILE_PATH "primos.txt"
const unsigned int MAXIMO = 100000000;
```

Luego se declara la función `vectorDiscardNonPrimes`, esta función asigna el valor `false` a los casilleros del vector, que recibe como argumento, en los cuales el índice es un numero no primo.

```
// Se declara la función ‘vectorDiscardNonPrimes‘
void vectorDiscardNonPrimes(std::vector<bool>& v);
```

Se declara la función `vectorExportToFilePath` la cual recibe como argumento un vector, un archivo abierto y una referencia a una variable. Esta función imprime sobre el archivo las posiciones del vector en donde el contenido es `true`, la cantidad de números impresos los asigna a la variable `primesWritten`.

```
// Se declara la función ‘vectorExportToFilePath‘
void vectorExportToFilePath(
         const std::vector<bool> v,
         std::ofstream& fp,
         unsigned int &primesWritten);
```

2.1.1. Función `main`

La Criba de Eratóstenes opera sobre una lista de números naturales ordenada, para representarla se utiliza la librería estándar `vector`, de la cual se define un vector de tipo `bool` inicialmente con valores `true` indicando que son todos números primos.

```
// Se define un vector de tipo ‘bool‘ de largo ‘MAXIMO‘
// con valores iniciales ‘true‘
std::vector<bool> numeros(MAXIMO, true);
```

Para la manipulación del archivo de salida, se utiliza la clase `ofstream` de la librería estándar `fstream`.

```
// Se define el objeto fp para representar el archivo de salida
std::ofstream fp;
```

Además se define la variable `primesFound` la cual luego contendrá la cantidad de números primos hallados.

```
unsigned int primesFound;
```

Para descartar los números no primos del vector `numeros` se invoca a la función `vectorDiscardNonPrimes`.

```
vectorDiscardNonPrimes(numeros);
```

Luego de finalizada la función, se imprime en el archivo de salida los índices del vector correspondientes a los casilleros que permanecen con un valor `true`, ya que éstos representan los números primos.

Utilizando los métodos `open` e `is_open` del objeto `fp` se abre el archivo `primos.txt` y se comprueba si hubo algún error, en caso afirmativo, se informa al usuario haciendo uso de métodos de la librería estándar `iostream` y se termina la ejecución inmediatamente con un código de error `-1`.

```
fp.open(OUTPUT_FILE_PATH);
if (fp.is_open() == false) {
    std::cerr << "ERROR: No se pudo abrir ‘" OUTPUT_FILE_PATH "‘\n";
    return -1;
}
```

Para exportar los números primos del vector se invoca a la función `vectorExportToFilePath`

```
vectorExportToFilePath(numeros, fp, primesFound);
```

Luego de finalizada la impresión en el archivo, se cierra utilizando el método `close`.

```
fp.close();
```

Finalmente se imprime un mensaje al usuario indicando que terminó la impresión, y se informa la cantidad de números primos hallados, esto se hace utilizando métodos de la librería estándar `iostream`.

```
std::cout << "Se encontraron ‘" << primesFound << "‘ números primos\n";
```

Por ultimo se termina la ejecución con un código `0` indicando que el programa se ejecutó exitosamente.

```
return 0;
```

2.1.2. Función `vectorDiscardNonPrimes`

La función `vectorDiscardNonPrimes` recibe una referencia a un vector de tipo `bool` como argumento y asigna `false` a los casilleros en los cuales el índice del mismo es un número no primo.

La función comienza descartando los casos particulares, `0` y `1`, por lo que asigna `false` a los índices respectivos directamente.

```
// 0 y 1 no son primos
numeros[0] = numeros[1] = false;
```

Luego itera sobre el vector asignando `false` a todos los múltiplos de los números primos, hasta llegar a la raíz cuadrada del numero máximo de la lista, o lo que es equivalente, hasta que el indice al cuadrado sea menor al numero máximo.

```
for (size_t i = 2; i*i < MAXIMO; ++i) {
    if(vector[i] == true) {
        for (size_t j = i; j <= (MAXIMO/i); ++j) {
            vector[i*j] = false;
        }
    }
}
```

2.1.3. Función `vectorExportToFilePath`

La función `vectorExportToFilePath` recibe como argumento un vector, una archivo abierto y una referencia a una variable. Luego de invocada la función imprime sobre el archivo los índices del vector en los cuales el contenido es `true`, a la variable que recibe le asigna la cantidad de índices impresos.

Al comienzo de la función se asigna a la variable `primesWritten` cero.

```
primesWritten = 0;
```

Para la impresión en el archivo, la función recorre el vector en búsqueda de aquellos casilleros que contengan `true`, en los casos afirmativos imprime el índice del vector al archivo e incrementa el contenido de la variable `primesWritten`, los casilleros que contienen `false` son ignorados.

```
for (size_t i = 2; i < v.size(); ++i) {
    if(v[i]) {
        fp << i << std::endl;
        primesWritten++;
    }
}
```

2.2. Makefile

Para organizar el proceso de compilación de la aplicación se utiliza la herramienta `GNU make`, la cual lee un archivo de nombre exclusivo `Makefile` que posee su propia sintaxis.

El archivo `Makefile` utilizado intenta ser reutilizable y de fácil modificación, es por esto que al comienzo del mismo se utilizan variables para almacenar las principales opciones que el usuario podría querer modificar, por ejemplo el nombre del compilador utilizado, el nombre final del archivo, entre otras.

```
CC := g++
CFLAGS := -Wall -Wshadow -pedantic -ansi -std=c++98 -O3
SRCS := $(wildcard *.cpp)
OBJS := $(SRCS:.cpp=.o)
TARGET := primos
```

Para compilar el programa y enlazar cabeceras, en caso de que haya, se utilizan las siguientes reglas:

```
$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -o $@ $^

%.o: %.cpp
    $(CC) $(CFLAGS) -c $< -o $@
```

La regla que contiene `$(TARGET)` se expande al nombre final de la aplicación, luego de los dos puntos se indican las dependencias, las cuales se expanden a los archivos objeto (los que terminan en `.o`)

Para compilar los archivos objetos se utiliza la segunda regla, la cual corresponde a una `pattern rule`, una extensión exclusiva de `GNU`, esta `pattern rule` compila todos los archivos terminados en `.cpp` a archivos objeto terminados en `.o`.

2.3. Optimizaciones del Compilador

La mayoría de los compiladores de C++ modernos proporcionan una opción para activar optimizaciones. Al utilizar estas optimizaciones el compilador intenta eliminar código redundante y optimizar mediante técnicas avanzadas el archivo binario final, de este modo se logra reducir el uso de memoria, el tiempo de ejecución y por consiguiente el consumo de energía, entre otras cosas.

En el archivo `Makefile` de la aplicación por defecto se utiliza el máximo de optimizaciones posibles `-O3`, de ésta manera se obtiene un archivo binario superior en rendimiento que el que se obtendría con las optimizaciones desactivadas.

2.3.1. Tiempo de Ejecución

En el siguiente bloque se muestra la salida del programa utilizando el máximo de optimizaciones posible del compilador `-O3`, se puede apreciar en la salida del programa el tiempo de ejecución del mismo.

```
Se encontraron ‘5761455‘ números primos en ‘11.36‘ segundos
```

Desactivando las optimizaciones del compilador, compilando y ejecutando nuevamente la aplicación, se obtiene el siguiente mensaje en la consola:

```
Se encontraron ‘5761455‘ números primos en ‘46.21‘ segundos
```

Comparando los dos resultados, se puede observar que al utilizar las optimizaciones de compilador el tiempo de ejecución se reduce en aproximadamente `75 %`.

3. Instalación

Para instalar la aplicación usted debe de disponer de una copia del código fuente, si no posee una, puede obtenerla ingresando al Repositorio de Github, de allí podrá descargar una copia, o bien puede clonar el repositorio utilizando `git` con el siguiente comando:

```
$ git clone https://github.com/mjkloeckner/CB100.git
```

Tenga en cuenta que si clona el Repositorio, o lo descarga del repositorio de Github, tendrá que navegar al directorio `tps/1`

3.1. Sistemas basados en UNIX

Compruebe que este en el mismo directorio que el archivo `primos.cpp`. Luego, compile el código con el programa `make`

```
$ make
```

Luego puede ejecutar la aplicación de la siguiente manera:

```
$ ./primos
```

3.2. Windows

En el caso de utilizar Windows como sistema operativo, usted puede configurar WSL, el cual virtualiza un sistema basado en UNIX, para hacerlo puede seguir la guia oficial de Microsoft. Una vez configurado WSL siga los pasos para los sistemas basados en UNIX

De manera análoga si usted dispone de un compilador de C++ puede compilar la aplicación directamente desde la consola, sin la necesidad de utilizar la herramienta `make`, para eso ejecute el siguiente comando:

```
$ g++ -Wall -Wshadow -ansi -std=c++98 -O3 primos.cpp -o primos
```

Tenga en cuenta la sintaxis de su compilador ya que puede variar, el comando anterior está previsto para MinGW-w64

Luego de compilado la aplicación usted la puede ejecutar de la siguiente manera

```
$ ./primos
```

4. Conclusión

Luego de finalizar el desarrollo de la aplicación, se puede concluir que la combinación de C++ con `GNU Make`, resulta muy versátil, y en este caso en un alto rendimiento.

El lenguaje C++ es uno de los lenguajes de mayor rendimiento en la lista de lenguajes de programación de alto nivel actuales, junto con C, aunque a diferencia de éste, C++ ofrece una amplia variedad de librerías estándar, lo que mejora su robustez y permite al usuario abstraerse de ciertas implementaciones que resultan irrelevantes en ciertos casos, por ejemplo, en el desarrollo de esta aplicación, el objecto `vector` utilizado para representar la lista de números naturales. De no existir la librería estándar `vector` uno tendría que ensuciarse las manos e implementar un tipo de dato para representar los números naturales, ya que no alcanza con los tipos de datos primitivos ofrecidos por el lenguaje.

Ademas de ahorrar tiempo al programador, las librerías estándar ofrecen mayor seguridad que si se tratase de un tipo de dato creado por el usuario, ya que las principales están respaldadas por organizaciones y grupos de estándares como ISO, IEC, entre otros.

5. Referencias

- Deitel H., Deitel P. - C how to Program: With an Introduction to C++ (2015)
- Repositorio de Github
- C++ Reference
- Standard C++ Library reference
- std::vector<bool>
- std::ofstream
- MinGW-w64
- GNU Make Manual