Universidad de Santiago de Chile
Facultad de Ingeniaría
Departamento de Ingeniería Informática

Estructuras de Datos y Análisis de Algoritmos

Informe Laboratorio N°1

Fecha entrega: 17/09/2014
Alumno: Joaquín Ignacio Villagra Pacheco
Profesora: Jacqueline Köhler Casasempere

ÍNDICE

INTRODUCCIÓN

Como trabajo de laboratorio de la asignatura Análisis de algoritmos y estructura de datos, se ha solicitado desarrollar una aplicación, en el lenguaje de programación C, que genere la serie Fibonacci para valores grandes de N.

Se preguntará: ¿Cuál es el problema?

Para valores pequeños de N, es posible usar las representaciones de enteros provistas por los lenguajes de programación, sin embargo para números un poco más elevados, la serie de Fibonacci supera los 32 Bits disponibles para el tipo de dato entero, lo que imposibilita la obtención de resultados certeros de dicha serie matemática.

Este proyecto es una adaptación del código Fibonacci iterativo que conocemos, que busca ampliar el intervalo de solución, generando la efectiva realización de la serie Fibonacci para números elevados de N.

Para el desarrollo de la aplicación se sugiere representar los números enteros como un arreglo de enteros de gran tamaño y sobre este, implementar el algoritmo solicitado.

Este informe estará compuesto por la descripción de solución, en donde se indica el método de resolución de problema aplicado; Análisis de resultados obtenidos, en donde se mostrará una visión crítica del programa desarrollado; Conclusión, en donde se especificará los logros obtenidos; y finalmente las referencias bibliográficas de la información expuesta.

DESCRIPCIÓN DE LA SOLUCIÓN

Primero que todo hay que comprender el porque surge este trabajo. El motivo, es la limitante que tiene el tipo de dato entero, a nivel de lenguajes de programación, para contener números elevados. Recordemos que un numero de tipo entero (INT) reserva 4 Bytes de memoria, a su vez un Byte son 8 bits. Aplicando la transitividad de la igualdad, se obtiene que un entero posee reservados: **4 Bytes = 4x8 = 32 bits**

Con 32 bits se pueden representar ***2^{32} = 4294967296 valores***
Sólo positivos (enteros sin signo): del ***0*** *al* ***4294967295***
Positivos y negativos (enteros con signo): del ***-2147483648*** *al* ***2147483647***

Teniendo claro el porque de este proyecto, se procede a describir la solución generada.

Aplicando la lógica de la división en subproblemas, aprendida en el curso de métodos de programación, se segmenta el proceso que genera la serie Fibonacci en dos funciones. Dicha división es realizada con el fin de facilitar el desarrollo de las funciones de manera independiente y al mismo tiempo, en conjunto generar la solución a nuestro problema raíz, el cual es obtener la serie Fibonacci para números elevados de N.

Ya que el problema es el tipo de dato a utilizar, se emula un nuevo tipo de dato, basándose en las estructuras de dato conocidas, que sea capaz de soportar números que estén por sobre los 4.294.967.295 valores que soporta el tipo de dato INT provisto por los lenguajes de programación. Para dicha emulación, se utiliza un puntero de tipo entero en formato de arreglo, al cual se le reserva una amplia sección de memoria, suficiente para que este arreglo soporte los valores generados por la serie Fibonacci de un numero N de gran tamaño *(Véase en linea 10, Figura 1)*. A su vez, se utilizan dos arreglos de tipo entero (INT), del mismo largo definido para el puntero descrito anteriormente *(Véase linea 31, Figura 2).*

Para manejar mayor orden a nivel de desarrollo, se separa el software en dos archivos, los cuales se detallan a continuación:

1. Funciones.c
   Archivo que contiene las funciones desarrolladas *(Véase Figura 1 y Figura 2).*

2. Laboratorio.c
   Archivo que contiene el main del programa solicitado *(Véase Figura 3).*

En la figura 1 se aprecia la implementación de la función sumaDeValores, la cual realiza el proceso base de nuestro problema raíz. Realiza la suma de los valores correspondientes al Fibonacci de los números (n-1) y (n-2), tal como lo indica la definición matemática de dicha serie.

```
 7 int *sumaDeValores(int numero1[], int numero2[], int largo)
 8 {
 9     int indice, acumulador=0;
10     int *Array = (int*) malloc (sizeof(int)*largo);
11     for(indice=largo-1;indice>=0;indice--)
12     {
13         int s = numero1[indice] + numero2[indice];
14         if(s/10>=1)
15         {
16             Array[indice]     = s % 10;
17             acumulador        = s / 10;
18             numero1[indice-1] = numero1[indice-1] + acumulador;
19         }
20         else
21         {
22             Array[indice] = s;
23         }
24     }
25     return Array;
26 }
```

Figura 1: Implementación de función "sumaDeValores"

A continuación, la figura 2 muestra el código de la implementación de la función "fibonacci". Esta segunda función, utiliza dentro de su desarrollo varios llamados a "sumaDeValores" (Véase linea 43).

```
29 void fibonacci(int n)
30 {
31     int Valor1[LARGO], Valor2[LARGO], indiceAuxiliar, indice, valor = 1;
32     for(indice=0; indice < LARGO; indice++)
33     {
34         Valor1[indice]=0;
35         Valor2[indice]=0;
36         if(indice==LARGO-1)
37         {
38             Valor2[indice]=1;
39         }
40     }
41     for(indiceAuxiliar = 1; indiceAuxiliar < n; indiceAuxiliar++)
42     {
43         int *Puntero = sumaDeValores(Valor1, Valor2, LARGO);
44         for(indice=0;indice<LARGO;indice++)
45         {
46           Valor1[indice] = Valor2[indice];
47           Valor2[indice] = Puntero[indice];
48         }
49     }
50     for(indice=0;indice<LARGO;indice++)
51     {
52         if(valor==1)
53         {
54             if (Valor2[indice+1]!=0)
55             {
56                 valor=0;
57             }
58         }
59         else
60         {
61             printf("%d", Valor2[indice]);
62         }
63     }
64 }
65
```

Figura 2: Implementación de función "fibonacci"

La figura 3, detalla la implementación del bloque principal del software. Esta sección, es la encargada de ejecutar las funciones detalladas anteriormente y manejar la entrada y salida de datos. Particularmente en este caso, la salida de datos no esta en manos del Main, ya que la función Fibonacci imprime directamente en pantalla el resultado obtenido.

```
1
2
3  #include "funciones.h"
4
5  int main(int argc, char **argv)
6  {
7      int n = atoi(argv[1]);
8      fibonacci(n);
9      printf("\n");
10     return 0;
11 }
12
```

Figura 3: Bloque principal del programa (MAIN).

Observaciones:

- La reserva de memoria realizada para el puntero es posible gracias a la función “malloc” del lenguaje de programación C *(Véase en linea 10, Figura 1)*.

- Para facilitar el detalle del largo de los arreglos y del puntero descrito, se utiliza un macro al comienzo del archivo “funciones.c” *(Véase linea 5, Figura 4).* Dicho macro, en tiempo de pre-compilación reemplaza, dentro del código, cada aparición de la palabra LARGO por el valor numérico detallado a su derecha, en este caso 1500. Esta implementación permite la rápida modificación del valor clave de las funciones, ya que las funciones especificadas solo soportarán el procesamiento de valores de N, dependiendo de la memoria que se le fue asignada.

```
3  #include <stdio.h>
4  #include <stdlib.h>
5  #define LARGO 1500
6
```

Figura 4: Detalle de bibliotecas utilizadas y definición de Macro.

ANÁLISIS DE LOS RESULTADOS OBTENIDOS

Esta sección se enfocará en comparar los resultados obtenidos con la implementación efectuada con el los objetivos planteados. Primero que todo, vemos que el *T(n)* de la función Fibonacci descrita en la figura 2 de este documento, posee una complejidad algorítmica del orden *O(n).*

En cuanto a la ejecución del software, este cumple efectivamente lo solicitado. Se logra la obtención de la serie Fibonacci para valores de N hasta 5000 (Incluso para valores mayores). El techo de la función depende únicamente de la cantidad de memoria reservada y el largo de los Arreglos auxiliares, en síntesis depende del valor especificado en el Macro definido en el archivo “funciones.c”.

En la figura 5, se visualiza la ejecución del software desarrollado. Cabe destacar que la ejecución fue desarrollada en un equipo con sistema operativo basado en UNIX (Ubuntu 14.04 LTS 64bits).

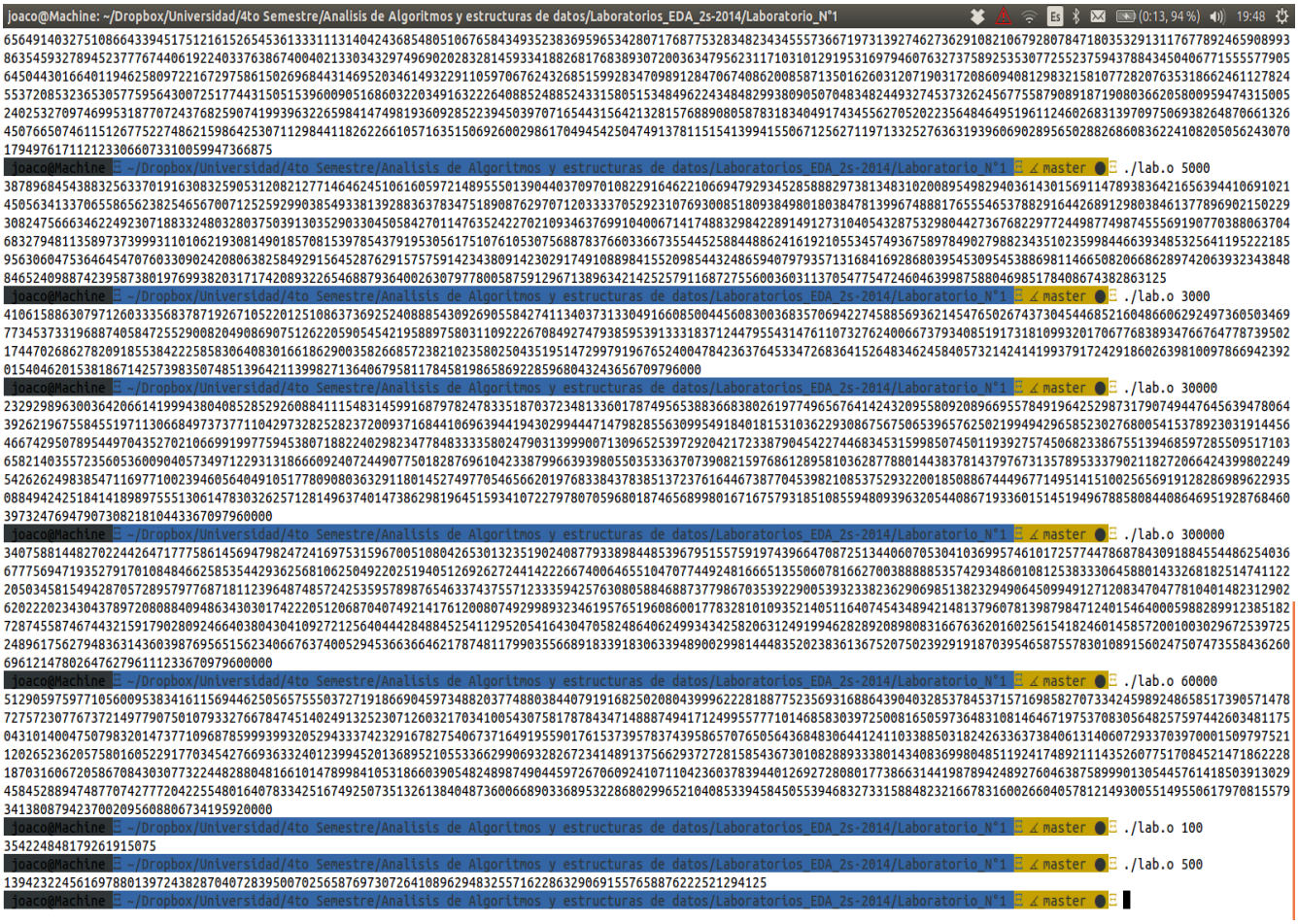

Figura 5: Ejecución de software (./lab.o = nombreEjecutable)

CONCLUSIÓN

Existen muchos lenguajes de programación que han surgido a lo largo del tiempo, lenguajes producto de los diferentes paradigmas o simplemente nuevos lenguajes creados para simplificar la vida a los programadores. A pesar de que C es un lenguaje “antiguo”, ofrece una ventaja considerable para quien se está adentrando en la programación, ya que obliga a trabajar a más bajo nivel permitiendo aprender desde la base y a su vez, valorar los avances que se generan con las nuevas tecnologías.

Cabe destacar que en el camino surgieron bastantes problemáticas, comenzando por la interrogante: ¿Como trabajar y operar los datos? Esta interrogante origino muchas disyuntivas, las cuales repercutieron en que la complejidad de algunas funciones se elevara considerablemente, sin perder el fin de encontrar una solución eficiente.

En términos concretos, se logró la implementación efectiva del software solicitado, el cual puede no ser óptimo al compararse con otros algoritmos, pero sin duda es una solución al problema planteado.

REFERENCIAS

Andrés Marzal, Isabel Garcia. (Año no informado). Introducción a la programación con C. Departamento de lenguajes y sistemas informáticos: Universitat Jaume.

Brian W. Kernighan and Dennis M. Ritchie. (1988). The C programming Language. EEUU: Prentice-Hall.

ANEXO: CAPTURAS DE PANTALLA CON FONDO BLANCO

```
int *sumaDeValores(int numero1[], int numero2[], int largo)
{
    int indice, acumulador=0;
    int *Array = (int*)malloc(sizeof(int)*largo);
    for(indice=largo-1; indice>=0; indice--)
    {
        int s = numero1[indice] + numero2[indice];
        if(s/10>=1)
        {
            Array[indice]     = s % 10;
            acumulador        = s / 10;
            numero1[indice-1] = numero1[indice-1] + acumulador;
        }
        else
        {
            Array[indice] = s;
        }
    }
    return Array;
}
```

Figura 1: Implementación de función “sumaDeValores”

```c
void fibonacci(int n)
{
    int Valor1[LARGO], Valor2[LARGO], indiceAuxiliar, indice, valor = 1;
    for(indice=0; indice<LARGO; indice++)
    {
        Valor1[indice]=0;
        Valor2[indice]=0;
            if(indice==LARGO-1)
            {
            Valor2[indice]=1;
        }
    }
    for(indiceAuxiliar = 1; indiceAuxiliar < n; indiceAuxiliar++)
    {
        int *Puntero = sumaDeValores(Valor1, Valor2, LARGO);
        for(indice=0;indice<LARGO;indice++)
        {
          Valor1[indice] = Valor2[indice];
          Valor2[indice] = Puntero[indice];
        }
    }
    for(indice=0;indice<LARGO;indice++)
    {
        if(valor==1)
        {
            if (Valor2[indice+1]!=0)
            {
                 valor=0;
            }
        }
        else
        {
            printf("%d", Valor2[indice]);
        }
    }
}
```

Figura 2: Implementación de función “fibonacci”

```c
#include "funciones.c"

int main(int argc, char **argv)
{
        if(argc==2)
        {
            int n = atoi(argv[1]);
            fibonacci(n);
            printf("\n");
        }
    else
    {
        printf("Debe ingresar el numero N a evaluar como parametro,");
        printf("separado por un espacio del nombre del archivo ejecutable.\n");
    }
    return 0;
}
```

Figura 3: Bloque principal del programa (MAIN).

```c
#include <stdio.h>
#include <stdlib.h>
#define LARGO 1500
```

Figura 4: Detalle de bibliotecas utilizadas y definición de Macro.