



DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE CONCEPCIÓN
CONCEPCIÓN, CHILE.

Informe Tarea N°1

Profesor: Mario Medina C.

Ayudante: Diego Ramírez V.

*Alumnos: Nicolás Borcoski S.
Aldo Mellado O.*

Esteban Paz F.

7 de mayo de 2018.

Índice

1. Generando la secuencia de Fibonacci	1
1.1. Código	2
1.2. Problema	3
1.3. Solución	3
1.4. Resultados	3
2. Revisando una secuencia Infinita de números	4
2.1. Código	5
2.2. Problema	6
2.3. Solución	6
2.4. Resultados	6

1. Generando la secuencia de Fibonacci

Para el problema 1 de esta tarea, se necesita generar una secuencia de Fibonacci, la cuál es una sucesión de números naturales que comienza con los números 0 y 1 y a partir de estos, los siguientes términos *son la suma de los dos anteriores*.

Para el caso particular de este problema, se pedía consultar el índice en el cual, la sucesión de Fibonacci alcanza los 1000 dígitos.

Procedimiento

La lógica a utilizar en la resolución de este problema, fue en primera instancia, la de resolver una suma utilizando listas, eso, para poder generar los valores de la sucesión. Luego, debíamos poder almacenar en una lista, el valor actual y el anterior en dos listas distintas, en la cual, a la siguiente iteración, el valor ".actual" pasara a ser el anterior. Tal y cual se define la sucesión.

$$F_n = F_{n-1} + F_{n-2} \quad (1)$$

Tal que para la segunda iteración.

$$L1 = F_{n-1} \quad ; \quad L2 = F_{n-2}$$

Entendido lo anterior, se necesitaba poder establecer una condición necesaria, que permitiera una vez alcanzada la cantidad de dígitos necesaria, se detuviera. Para ello se introduce la variable `len2`, que es la que representa el largo de la lista que contendrá finalmente al número de la sucesión que alcanza los 1000 dígitos.

Luego, se tiene que para inicializar los valores, pues en este caso se trata del Fibonacci(1), se debía introducir, como primer elemento a cada lista un valor igual a 1, lo que se hace con `L1.push_back(1)` y `L2.push_back(1)`. Hecho esto, definida la condición de detención, se deben definir dos contadores, uno que me servirá de condición y otro que me entregará información respecto del índice, el cual está asociado al número de iteraciones del ciclo `while`, en que alcance la cantidad de dígitos que se requieren.

Ya definida los parámetros que condicionan la operación del código y aquellos que entregarán la información respecto de lo requerido para el problema, se define la suma como una operación iterativa que se vale de valores anteriores y de un carry `C` que, según sea el caso, valdrá 1 o 0, esto dependerá si existe acarreo desde la unidad a la decena o desde la decena a la centena.

Este acarreo al trabajar con listas, implicaba dos cosas, la primera, que existiría eventualmente una iteración en la cual el largo de ambas listas fuera diferente, es decir 13 tuviera largo 2 y 8 tuviera largo 1, de modo que debía hacerse la salvedad y cubrir este caso, esto se logra mediante el `if` de la línea 38 para el caso del acarreo de unidad a decena, en la cual se hace un `push_back(0)` a la vez que se establece como condición adicional, y en la línea 40 para el caso del acarreo de la decena hacia la centena. La posibilidad de que el largo de las listas fuera diferente, se contempló al incluir las líneas 56 hasta 59, pues, una vez hecha la iteración y recalculado el largo de las listas es que era posible discriminar el largo actual que iban a tener las listas, pudiendo esto pposibilitar o no una nueva iteración con un resultado válido..

Finalmente, lo que se detalla en las líneas desde la 51 y hasta la 54, es la copia y borrado del contenido de las listas. Esto se hace precisamente para poder cumplir la condición de fibonacci especificada en la ecuación 1.

1.1. Código

```

1 //Problema 1 Tarea 1
2
3 #include <list>
4 #include <cstdlib>
5 #include <iostream>
6 #include <algorithm>
7 #include <vector>
8
9 using namespace std;
10
11 int main()
12 {
13     int len1,len2,len3,lens=0,cont=0,m=0,aux=0,C=0,i=0,fibo=0,cont1=0;
14     list<int> suma,L1,L2;
15     list<int>::iterator it1,it2,it3;
16
17     //inicializamos la lista en f0 = 1, que corresponden a los primeros valores de la sucesión.
18     L1.push_back(1); L2.push_back(1);
19
20     it1 = L1.begin() ;it2 = L2.begin();
21     len1= L1.size(); len2= L2.size();
22
23     /*con los iteradores y sus largos respectivos, definimos la condición de que en el ciclo while,
24     apenas se alcance en la lista L2, un largo igual o mayor a los 1000 dígitos se detenga.*/
25
26     while(len2<=1000)
27     {
28         cont = 0; //este contador nos permite usar la condición en la línea 39
29         cont1++; //este contador crece conforme se itera en el while
30         for(it2=L2.begin(); it2!=L2.end();it2++)
31         {
32             aux = *it1 + *it2 + C; //definida la suma que da lugar a la sucesión
33             m = aux%10;cont++; //operación módulo que permite utilizar un carry en la suma
34             if (aux<10) //caso en que no se produce acarreo por no exceder el valor 10
35             {suma.push_back(aux);C = 0;} //en este caso, agregamos a la lista el valor resultante de la suma
36
37             else //el caso contrario, es que la suma si tenga acarreo
38             {suma.push_back(m);C = 1;} //en dicho caso, se usa la operación módulo para agregar el "resto" del
39             módulo 10 y se procede a agregar el carry C = 1
40
41             if(C==1 && L2.size()==cont)
42             {suma.push_back(C);C = 0;}
43             it1++;
44         }
45
46         /*Se agrega esta ultima condición para poder cubrir el caso en que haya doble acarreo, unidad-decena
47         y decena-centena, el cual se presenta cuando C = 1 en dos iteraciones*/
48
49         L1.clear(); //una vez hecha la suma, se borran los elementos previos de L1
50         L1=L2; //para copiar los nuevos elementos provenientes de los almacenados en L2
51         it1 = L1.begin();
52         len1= L1.size();
53
54         L2.clear(); //Se borran los elementos en L2
55         L2 = suma; //para copiar los elementos almacenados en suma
56         len2 = L2.size(); //Calculamos el largo para revisar la condición del While
57         suma.clear();
58
59         if (len2>len1)
60         {
61             for (i=0;i<len3;i++)
62             {L1.push_back(0);}
63         }
64
65         // Con estas, se cubre la posibilidad de que el largo de los dos elementos sea diferente y debido a esto,
66         no se pueda hacer la suma.
67         if (len2>= 1000)
68         {cout<<"1000 dígitos alcanzados en el índice: " <<cont1<<endl;break;}
69     }
70     return 0;
71 }

```

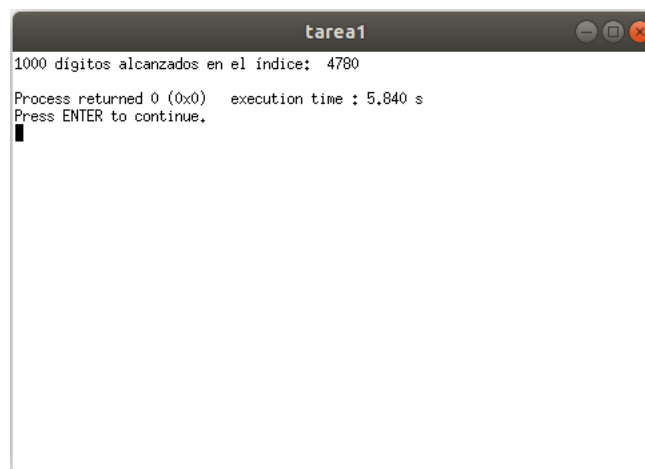
1.2. Problema

1. En este problema, uno de los eventos que se suscita es que dado el acarreo que debe haber entre un dígito los tamaños entre ambas listas, $L1$, $L2$, se debía poder llenar el espacio faltante de modo que los tamaños fueran iguales para poder seguir realizando la suma. Este problema se manifestaba cuando, al mostrar los resultados de la suma, en vez de mostrarse un 13, se mostraba en pantalla un 3.
2. Asimismo, se tuvo que dado que el acarreo podía producirse no sólo en la unidad hacia la decena, sino también desde la decena y hacia la centena, debía entonces agregarse un espacio en la lista siguiente para que pudiera sumarse el *carry*. Este problema se manifestaba cuando, ya solucionado el problema mencionado en 1., se tenía que para mostrar el número 21, este aparecía como un 111. Esto debido a que no se estaba haciendo el segundo carry.
3. Tiempo de ejecución demasiado largo

1.3. Solución

1. Respecto de lo mencionado en la sección 1 de Problema, se optó por solucionar este problema mediante lo mostrado en las líneas 35,36 del código, donde se evita así que el carry resultante del overflow de la suma anterior, no se pase a la siguiente, es decir, de este modo se evita que en lugar de mostrar un 13, se muestre solo un 3.
2. Respecto del punto 2, se tiene que para solucionar este problema, se solucionan en las líneas 38,39, puesto que con estas se cubre el doble acarreo en las que se pone como condición de que, si se produce una segunda iteración, acusada por el contador `cont`, y si además de esto, se da que $C = 1$ nuevamente, entonces se agrega un cero, para así poder acarrerear
3. Se optó por no mostrar los números y solo mostrar el contador final.

1.4. Resultados



```
tarea1
1000 dígitos alcanzados en el índice: 4780
Process returned 0 (0x0) execution time : 5.840 s
Press ENTER to continue.
█
```

Fig. 1: Resultado contador 1000 dígitos

2. Revisando una secuencia Infinita de números

Para lograr una secuencia lo suficientemente extensa para cumplir con los requerimientos se decidió emplear un vector de enteros, que almacena cada dígito en un espacio de memoria, obteniendo un resultado similar al presentado en el enunciado de la tarea como S.

$$S = \{123456789101112131415161718192021...\}$$

Dentro de esta secuencia, se debía encontrar las ocurrencias, o veces en que dentro de esta se encuentra un número en particular. Se menciona el caso $f(5) = 81$, por el cual se da a entender que la 5 ocurrencia de 5 es en la posición 81 de la sucesión. Lo anterior se debía realizar para todas las potencias de 3, entre 1 y 9.

Procedimiento

1. Primero se genera un vector vacío.
2. Se rellena con los dígitos (dig) por separado de cada número (num)
3. Para lograr esto, se toma un número y se descompone en cada uno de sus dígitos.
4. Cuando este número no puede descomponerse más, es decir, num=0, entonces se procede con el siguiente número.
 - 4.1 Cada uno de los dígitos se va insertando en el vector con la función
 - 4.2 El problema, es que insertaba primero las unidades, decenas,..., etc. Por esta razón se hace arreglo matemático con los órdenes de iteración la función *reverse* para obtener lo pedido
5. num=i, para que se vayan iterando todos los números necesarios, a criterio del programador.
6. Se obtiene el vector con la secuencia S.
7. Para encontrar lo que se pide se debe tener claro que el código debe ser modificado, de acuerdo al número pedido.
8. Se crean 2 contadores cont1 y cont2, además de un iterador iter del vector.
9. Se genera un ciclo while que recorre todo el vector con el iterador.
 - 9.1 Se genera un ciclo if que condiciona a que los siguientes n iteradores (para un número de n dígitos) apunten a los valores pedidos específicamente, es decir, se encuentra un número dígito a dígito, igualando con iteradores consecutivos.
 - 9.2 Cuando esto ocurre entonces el ciclo if agrega un 1 a cont1, el cual lleva la cuenta de las ocurrencias del número en cuestión.
 - 9.3 Luego cuando cont1="número", quiere decir que se llegó a lo pedido, pues el número a ocurrido esa misma cantidad de veces en la secuencia S.
 - 9.4 El contador cont2 por otro lado, lleva la cuenta de cuantas veces a ocurrido el ciclo while, en otras palabras, lleva la cuenta total de espacios que se han revisado en el vector v1, y por lo tanto cont 2 lleva la cuenta del índice en el cual se está.
10. Se imprime en pantalla cont2 y de esa forma se obtiene la posición donde se encuentra la n-sima ocurrencia del número a buscar.
11. **Ideas no llevadas a cabo:**
 - 11.1 Nos damos cuenta que se piden múltiplos de 3, y estos cuentan con la propiedad de que la suma de sus dígitos
 - 11.2 También es múltiplo de 3, y de alguna forma, podríamos reducir el largo del vector. El inconveniente de esto es que si eliminábamos los no-múltiplos de 3 se perderían las posiciones
 - 11.3 Otra idea era insertar los números de por medio iguales y luego los consecutivos entre ellos, es decir: Insertar 1 1 1 1 1 1 1 1 1 1 1 en las posiciones impares, y luego 0 1 2 3 4 5 6 7 8 9 en las pares, para luego obtener 10111213141516171819, pero esto era muy poco eficiente de implementar y se llegó a algo mejor.

2.1. Código

```

1  #include <iostream>
2  #include <vector>
3  #include <iterator>
4  #include <list>
5  #include <algorithm>
6
7
8  using namespace std;
9
10 int main()
11 {
12     vector<int> v1; //Se crea un vector de enteros
13     int i,num=0,dig;
14     for(i=100000000;i>=0;i--)
15     {
16         while(num!=0)
17         {
18
19             dig = num%10;
20             v1.push_back(dig); //Se va rellinando el vector con los digitos
21             num = num/10;
22         }
23
24         num=i;
25     }
26     reverse(v1.begin(),v1.end()); //El vector se invierte, pues estaba al revés por las iteraciones del ciclo
    for
27
28
29
30     int cont1=0,cont2=0;
31     vector<int>::const_iterator iter=v1.begin();
32
33     while (iter != v1.end())
34     {
35         if (*(iter) == 7&&
36             *(iter+1) == 7&&
37             *(iter+2) == 8&&
38             *(iter+3) == 0)
39         {
40             cont1++; //cont1: contador que lleva la cuenta de cuantas veces a ocurrido el numero
41         }
42         cont2++; //cont2: Este contador lleva la cuenta de las veces que ha ocurrid el ciclo
43
44
45         if(cont1==7780){break;} //Se corta el ciclo en la n-sima aparicion de n
46         ++iter;
47     }
48     cout<<cont2<<"<="<<cont1<<endl;
49
50     return 0;
51 }

```

2.2. Problema

1. Lo anterior se realiza para todas las potencias de 3, entre 1 y 8. En el caso de 3^9 no pudimos encontrar un i suficientemente grande para abarcar más de 1003 ocurrencias de 19683 en índice 131688897 del vector $v1$. Más allá de eso, nos arrojó error de memoria.

2.3. Solución

1. Para realizar la descomposición del entero en sus dígitos y luego almacenarlos por separado en el vector, como también para comparar mediante iteradores los dígitos buscados con las apariciones en el vector "infinito" nos inspiramos en el código solución del primer problema del certamen 1 del año 2017 enviado vía INFODA por el profesor.

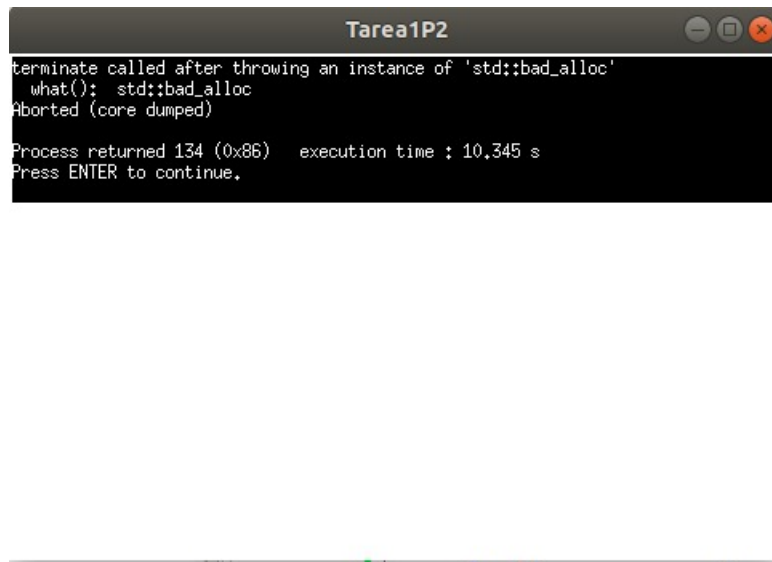


Fig. 2: Error por falta de memoria

2.4. Resultados

Los resultados se presentan en la siguiente tabla:

n	$f(n)$
3^1	37
3^2	169
3^3	2208
3^4	4725
3^5	161013
3^6	926669
3^7	14199388
3^8	52481605