INFORME DE PRÁCTICAS

Repositorio del proyecto : https://github.com/danfouz/VVS
Participantes en el proyecto:
Dan Álvarez Fouz (dan.fouz@udc.es)
Daniel Moledo Gárcia (daniel.moledo@udc.es)

Validación e Verificación de Software

1. Descripción del proyecto

Este proyecto software comprueba empíricamente el análisis teórico de la eficiencia de tres algoritmos diferentes que permiten el cálculo de la sucesión de Fibonacci. Los algoritmos están implementados en C.

2. Estado actual

Tenemos los tres algoritmos fib1(), fib2() y fib3() que estan implementados como se detalla a continuación:

```c
int fib1(int n){
  if (n<2)
    return n;
  else
    return fib1(n-1) + fib1(n-2);
}
```

```c
int fib2(int n){
  int i,j,k;
  i=1;
  j=0;
  for (k=0;k<n;k++){
    j=i+j;
    i=j-i;
  }
  return j;
}
```

```c
int fib3(int n){
int i,j,k,h,t;
  i=1; k=0; j=0; h=1;
  while (n>0) {
    if ((n % 2) != 0) {
      t=j*h;
      j=(i*h) + (j*k) + t;
      i=(i*k) + t;
    }
    t=h*h;
    h=(2*k*h) + t;
    k=(k*k) + t ;
    n= n/2;
  }
  return j;
}
```

Figura 1: *Implementación de los 3 algoritmos*

2.1. Componentes Evaluados

Hemos evaluado cada uno de los 3 algoritmos, cada uno en su fichero:
fib1() – fib_complexN1_6.c
fib2() – fib_complexN.c
fib3() – fib_complexlogN.c

3. Especificación de pruebas

3.1. fib_complexN1_6.c

- **PR-UN-001**
Función/método: fib1(int n)
Motivación: Validacion de que el algoritmo funciona, resolviendo la sucesion de fibonacci para 10 iteraciones.
Entradas: n=10;
Salidas esperadas: n(10) = 55

- **PR-UN-004**
Función/método: fib1(int n)
Motivación: Validación del no funcionamiento del algoritmo cuando se le introduce un número negativo.
Entradas: n< 0;
Salidas esperadas:Error

- **PR-UN-007**
Función/método: fib1(int n)

Motivación: Validación del funcionamiento del algoritmo cuando se le introduce un 0
Entradas: n=0
Salidas esperadas: 0
- **PR-UN-010**
Función/método: fib1(int n)
Motivación:Validación del funcionamiento del algoritmo cuando se le introduce un 1
Entradas: n=1
Salidas esperadas:1
- **PR-UN-**
Función/método: fib1(int n)
Motivación:
Entradas: n=;
Salidas esperadas:
- **PR-UN-**
Función/método: fib1(int n)
Motivación:
Entradas: n=;
Salidas esperadas:

3.2. fib_complexN.c

- **PR-UN-002**
Función/método: fib2(int n)
Motivación:Validacion de que el algoritmo funciona, resolviendo la sucesion de fibonacci para 10 iteraciones.
Entradas: n=10;
Salidas esperadas:n(10) = 55

- **PR-UN-005**
Función/método: fib2(int n)
Motivación:Validación del no funcionamiento del algoritmo cuando se le introduce un número negativo.
Entradas: n<0;
Salidas esperadas:Error
- **PR-UN-008**
Función/método: fib2(int n)
Motivación: Validación del funcionamiento del algoritmo cuando se le introduce un 0
Entradas: n=0
Salidas esperadas: 0
- **PR-UN-011**
Función/método: fib2(int n)
Motivación:Validación del funcionamiento del algoritmo cuando se le introduce un 1
Entradas: n=1
Salidas esperadas:1
- **PR-UN-014**
Función/método: fib2(int n)
Motivación: Validación del funcionamiento del algoritmo con n = 1.000.000
Entradas: n=1.000.000

Salidas esperadas: n= 1884755131

3.3. fib_complexlogN.c

- **PR-UN-003**
Función/método: fib3(int n)
Motivación:Validacion de que el algoritmo funciona, resolviendo la sucesion de fibonacci para 10 iteraciones.
Entradas: n=10;
- **PR-UN-006**
Función/método: fib3(int n)
Motivación:Validación del no funcionamiento del algoritmo cuando se le introduce un número negativo.
Entradas: n <0
Salidas esperadas: Error
- **PR-UN-009**
Función/método: fib3(int n)
Motivación: Validación del no funcionamiento del algoritmo cuando se le introduce un 0
Entradas: n=0
Salidas esperadas: 0
- **PR-UN-012**
Función/método: fib3(int n)
Motivación:Validación del funcionamiento del algoritmo cuando se le introduce un 1
Entradas: n=1
Salidas esperadas:1
- **PR-UN-013**
Función/método: fib3(int n)
Motivación: Validación del funcionamiento del algoritmo con n = 1.000.000
Entradas: n=1.000.000
Salidas esperadas: n= 1884755131

4. Registro de pruebas

Pruebas caja blanca
cppcheck –enable=all fib_complexN.c fib_complexN1_6.c fib_complexlogN.c 2¿err.txt
Checking fib_complexN.c...
1/3 files checked 28 % done Checking fib_complexN1_6.c...
2/3 files checked 58 % done Checking fib_complexlogN.c...
3/3 files checked 100 % done Checking usage of global functions..
La salida se adjunta en el fichero err.txt
Pruebas de mutacion de codigo
Empleando Milu con:
./milu -f func.txt fib_complexN.c nos devuelve Segmentation Fault
Test de Estress
Los datos obtenidos al someter a los diferentes algoritmos a pruebas con valores grandes

para sus capacidades, para fib1 se fijo un valor no muy grande porque hemos comprobado empíricamente que con valores mayores a 50 el tiempo de ejecucion es demasiado grande. Los datos medidos pre y post optimización del código se recogen en la siguiente tabla:
Informe de Cobertura

La cobertura se adjunta en la carpeta doc.

	fib_complexN1_6.c	N=40	fib_complexN.c	N=100000000	fib_complexlogN.c	N=1000000000
Grado Optimizacion	tiempo	tamaño	tiempo	tamaño	tiempo	tamaño
0	0.646s	1219	0.441s	1219	0.001s	1331
O1	0.450s	1203	0.001s	1171	0.001s	1235
O2	0.358s	1227	0.001s	1171	0.001s	1235
O3	0.486s	2251	0.001s	1171	0.001s	1251

Figura 2: *Pruebas Estress*

5. Registro de Errores

Error al insertar número negativo
Cuando se inserte un número negativo el programa deberia de alertar de alguna forma, se dejo la implementación de que devolviera el numero negativo. **Mutacion del codgio**
Prueba no pasada por obtener Segmentation Fault

6. Estatísticas

Issues
13 abiertas 12 cerradas