Universidad Simón Bolívar.
Departamento de Computación.
Lenguajes de programación.
Angel Garces 16-10400

Examen 2

En este documento se responderán algunas de las preguntas del examen 2 de lenguajes de programación, los códigos se pueden encontrar en el siguiente link de github:

`https://github.com/AngelVzla99/materia-lenguajes-de-programacion.git`

1. Pregunta 1
   Ya que mi apellido es Garces decidí utilizar 'Go' para esta pregunta. Los códigos de la parte b se encuentran en la carpeta **examen2/pregunta1**. Para probar los códigos use el siguiente compilador online:

   `https://go.dev/tour/basics/4`

   a) Estructuras de control de flujo:
      1) Las instrucciones se ejecutan de arriba hasta abajo estando en un bloque de código
      2) También posee condicionales if-else que permiten ejecutar algunas partes de código y otras no (se pueden ver en la linea 50 del archivo **parteb2**)
      3) También posee 'switch', una estructura de control de flujo similar al if-else pero que tiene una traducción mas eficiente a bajo nivel
      4) También tiene una estructura iterativa, los ciclos for, muy similar a los lenguajes parecidos a C
      5) Se pueden crear recursiones

   b) Orden en que se evalúan las expresiones y funciones Las expresiones se evalúan de izquierda a derecha como se puede ver en el archivo **pruebaOrdenEnGo** el resultado es 6 debido a que la expresión se va evaluando de derecha a izquierda, los parámetros de las funciones son evaluados al momento de invocar se la función

   c) Posee enteros de 8 hasta 64 bits, tambien enteros sin signo de 8 a 64 bits, posee a su vez float de 32 y 64 bits, tipo complejo de 32 y 64 bits (para números complejos), también tiene bool y para crear tipos se usa **type nameType struct{ ... }**

   d) Go soporta polimorfismo solo por medio de interfaces (se crean con la palabra reservada interface)

   e)

2. Pregunta 2
   Los códigos del programa se encuentran en la carpeta del repositorio **examen2/pregunta2**. Para ejecutar el programa hay 2 comandos que se pueden usar

a) **make** (seguido de ./main) esto compila el programa y luego arranca el ejecutable con el menu interactivo

b) **unit_test** (seguido de ./unit_test) esto ejecuta los test case que se encuentran en el archivo **unit_test.cpp**

Al ejecutar **make unit_test_coverage** nos dice que con los test cases obtuvimos una cobertura del 89.16 % del código.

3. Pregunta 3

   a) Parte a
   Decidí hacer la corrida en google presentations, el link de la presentación es el siguiente:
   `https://docs.google.com/presentation/d/1w1E6oEY1nfjK-bLM5F2V7LnQIIMUtyxoRavCS7w`
   `edit?usp=sharing`

   b) Parte b

   1) Se encuentra en el mismo link de la pregunta 3.a
   2) La función genera todas las permutaciones (permutación por índice, no por valor) de una lista ingresada por parámetro.

      La manera en que consigue esto es tomando el primer elemento de la lista y ponerlo en todas las posiciones intermedias de las permutaciones del resto de elementos. Para hallar las permutaciones del resto de elementos se llama a si misma de manera recursiva.

      Si la lista es a = [a1,a2,...,an], la pica en 2 [a1] [a2,...,an] y para cada permutación de la segunda, coloca [a1] en todas las posiciones intermedias, esto es [a1,a2,a3,a4,...,an] [a2,a1,a3,a4,...,an] [a2,a3,a1,a4,...,an] [a2,a3,a1,a4,...,an] hace esto para toda las permutaciones de [a2,...,an]

      Para colocar a1 en todas las posiciones intermedias utiliza el iterator 'ins' el cual toma como parámetro el elemento a colocar en las posiciones intermedias y una lista.
      Este algoritmo funciona por una propiedad interesante, si tenemos una permutación P de los números de 1 hasta n, y queremos todas las permutaciones hasta n+1, basta con colocar a n+1 en cualquier posiciones intermedia para cada permutación de 1 hasta n, ya que claramente, nunca se generan permutaciones repetidas.
   3) Pregunta 3.b.iii

```
def suspenso(ls):
    if len(ls)==1: yield ls[0]
    elif len(ls)>1:
        m = []
        for e in suspenso(ls[1:]):
```

```
            m.append(e)
        if len(m)%(len(ls)-1)==0:
            for lista in ins(ls[0], m):
                for elem in lista :
                    yield elem
            m = []
```

(el codigo tambien esta en el repo, en la carpeta de la pregunta 3)

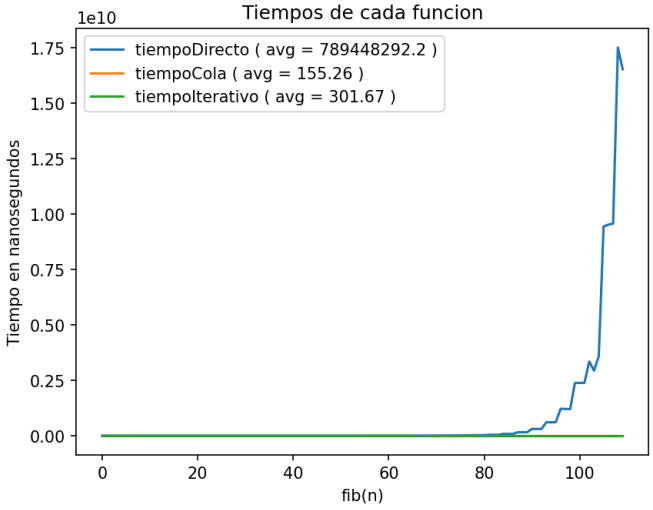
4. Pregunta 4
   Para esta pregunta decidí usar C++ para crear las 3 funciones (los codigos se encuentran en la carpeta **examen2/pregunta4/funcs.cpp** del repo) y Python para hacer el plot de los tiempos de cada prueba, cada una de estas consiste en calcular los tiempo que tarda cada implementacion de $f_{\alpha,\beta}(N)$ en calcular los numeros del 1 hasta el 110. Para ejecutar los archivos se pueden usar 3 opciones del main.

   a) make <OPCION>: Este comando ejecuta un ciclo while hasta el final de un archivo donde toma como input N e imprime el tiempo que le tomo calcular $f_{\alpha,\beta}(N)$ donde <OPCION>es -d, -c o -i (directo, cola, iterativo respectivamente)

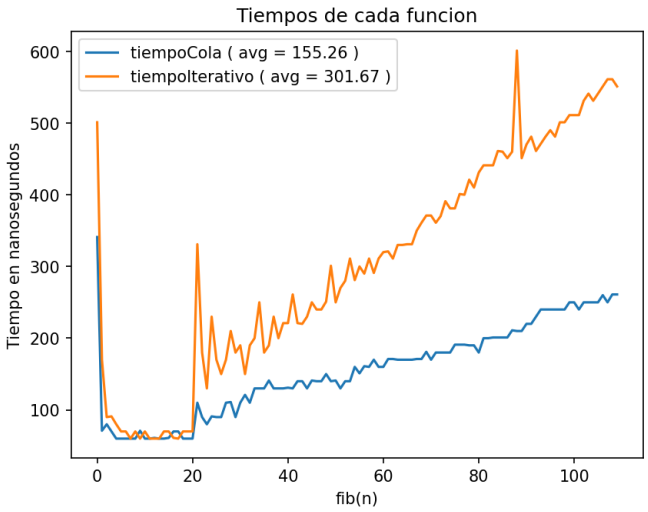
   b) make generateData: Genera un archivo con los los tiempos para cada una de las 3 implementaciones

   c) makePlots: Ejecuta un programa de python para generar los plots de los tiempos que tardo cada implementación de la función F

   Al comparar los tiempos de las 3 implementaciones podemos apreciar que la directa tarda mucho mas, esto ocurre debido a que la complejidad de este algoritmo es exponencial, para calcular f(n) se necesitan 7 valores, para cada uno de estos se necesitan 7 mas ... el problema es que al calcular un f(n) no guardamos su valor, sino que si mas adelante lo volvemos a utilizar lo calculamos de nuevamente.



Ya que la gráfica anterior no permite aprecias los tiempos de respuesta de las implementaciones **iterativa vs recursion de cola**, hice una gráfica solo con los resultados de estos dos. En esta podemos apreciar que se obtuvieron resultados similares sin embargo la recursion de cola tomo menos tiempo, esto es normal teniendo en cuenta que el compilador (en mi caso GCC) detecta la recursion de cola y la optimiza.



5. Pregunta 5
   Los códigos del programa se encuentran en la carpeta del repositorio **examen2/pregunta5**.
   Para ejecutar el programa hay 2 comandos que se pueden usar

   a) **make** (seguido de ./main) esto compila el programa y luego arranca el ejecutable con el menu interactivo

   b) **unit_test** (seguido de ./unit_test) esto ejecuta los test case que se encuentran en el archivo **unit_test.cpp**

   Al ejecutar **make unit_test_coverage** nos dice que con los test cases obtuvimos una cobertura del 97.26 % del código.

6. Pregunta 6
   Decidí hacer la pregunta en una presentación de google, esta se puede ver con el siguiente link
   `https://docs.google.com/presentation/d/1ceFhTd2MO7vGlJvSjliB2AxwC0JvYcTl4nZppdBdsXE/edit?usp=sharing`

7. Pregunta 7
   Los códigos del programa se encuentran en la carpeta del repositorio **examen2/pregunta7**.
   Para ejecutar el programa hay 2 comandos que se pueden usar

   a) **make** (seguido de ./main) esto compila el programa y luego arranca el ejecutable con el menu interactivo

b) **unit_test** (seguido de ./unit_test) esto ejecuta los test case que se encuentran en el archivo **unit_test.cpp**

Al ejecutar **make unit_test_coverage** nos dice que con los test cases obtuvimos una cobertura del 100 % del código.