UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2016

19 de Septiembre de 2016

Taller de Programación - C++

1. Motivación

En poco tiempo, un equipo de *recruiters* del gigante *Hooli* desembarcará en nuestros suelos en búsqueda de mentes brillantes para incorporar a sus innovadoras filas. El proceso de selección implica una serie de entrevistas donde se les pide a los aspirantes que resuelvan distintos problemas algorítmicos, mostrando no sólo su pericia técnica sino su creatividad para encarar los desafíos. Afortunadamente, un docente de Algo1 -de identidad reservada- consiguió en el mercado negro la lista de problemas que se evaluarían para esta pasantía. A cambio, este informante pide que quien obtenga un puesto le traiga luego un encargo de artículos de electrónica de EE.UU.

El presente taller tiene como objetivo motivar la ejercitación individual de implementación de problemas en C++. Se presentan 10 ejercicios de dificultad variable (independiente del orden en que aparecen) y dos problemas extra de dificultad un poco más avanzada.

2. Tests

Además de la demostración de correctitud, la utilización de tests para verificar el funcionamiento de nuestros programas es muy importante y de gran utilidad. Si bien no se trata de un método matemático exacto, la buena elección y cobertura de numerosos casos - entre otros criterios - es una manera sencilla (depende) y efectiva de asegurarnos el funcionamiento esperado de lo que hacemos. Para ayudarlos durante la resolución, y automatizar parte de la corrección, proveemos una batería de tests para cada problema (auspiciados por Google Test[1]).

Pssst! La mayoría de los tests chequean correctitud de soluciones, aunque en algunos problemas fallarán si la ejecución de algunos casos tarda más de un tiempo determinado. ¡Si se animan, traten de cumplir también con esta restricción!

3. Indicaciones

En la carpeta del código del taller se encontrarán con varios archivos, entre los que se destacan:

- `src/taller.cpp` (+): aquí deben completar las implementaciones de los problemas.
- `src/taller.h` (+): headers de los problemas. Pueden agregar headers de auxiliares si lo necesitaran.
- `test/*_tests.cpp`: estos archivos contienen los casos de test. Pueden inspeccionarlos y agregar sus propios tests, pero **no** es el objetivo, y no deben modificar los ya existentes.
- `test/lib`: contiene el framework de Google Test.

+: Estos son los únicos archivos que deberían modificar. Para implementar sólo pueden usar `iostream`, y si bien se incluyen `math.h` y `cmath`, es únicamente para que utilicen las funciones `pow()` y `abs()` si lo necesitaran. No deben utilizar ninguna otra funcionalidad ni biblioteca.

Con el objetivo de brindar un entorno de programación y compilación uniforme para todos, se recomienda utilizar la IDE **CLion**. Deberán abrir un proyecto buscándolo en la carpeta `~/taller/codigo`. En caso de que se prefiera correr por consola, se incluye un readme sobre cómo compilar y correr los tests de esta manera.

4. Debugging

Recomendamos fuertemente que utilicen el debugger de la IDE [2] para ayudarlos en la resolución de los problemas, y a tratar de encontrar los errores en sus resoluciones. Como motivación extra, es altamente probable que algún caso de test de uno de los primeros 10 problemas les falle para la mayoría de las resoluciones que encaren. Están invitados a encontrar el problema y enviarnos un mail explicando de qué se trata, incluyendo una captura de pantalla donde señalen el inconveniente en su debugger. Quien primero identifique correctamente el problema, con prueba incluida, será acreedor de un premio.

[1] https://github.com/google/googletest

[2] Si no utilizan IDE, pueden buscar *gdb*

El taller es a libro abierto: pueden consultar o buscar lo que quieran para asistir su propio razonamiento y experiencia; no copiar soluciones hechas de otro lado `¡Feliz Codeo!`

5. Problemas

1. `estaOrdenado`: Dado un arreglo de números enteros, decidir si está ordenado (tanto ascendente como descendentemente).
2. `esPrimo`: Decidir si un número es primo.
3. `pertenece`: Decidir si un elemento dado pertenece a un arreglo.
4. `desvioEstandar`: Dado un arreglo de N floats, calcular el desvío estándar de la muestra:

   $$\sigma = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(x_i - \mu)^2}$$

   Sugerencia:

   - Calcular la media o promedio μ.
   - Para cada número, restarle el promedio y elevar el resultado al cuadrado.
   - Calcular el promedio de esos resultados.
   - Devolver la raíz cuadrada de ese promedio.

5. `fibonacci`: Dado un número $k \geq 0$, devolver el $k - esimo$ elemento de la sucesión de Fibonacci.
6. `maximoComunDivisor`: Calcular el *mcd* entre dos números.
7. `sumaDoble`: Para un arreglo de enteros, calcular la sumatoria del doble de los elmentos positivos y pares.
8. `esCapicua`: Decidir si el string de entrada es capicúa. Ejemplos: *abcba* y *aba* lo son.
9. `mismos`: Dados dos arreglos, determinar si contienen los *mismos* elementos.
10. `esPrefijo`: Decidir si una palabra es prefijo de otra. Ejemplo: *pre* es prefijo de *prefijo*.

6. Problemas extra

11. `maximoProductoEnSerie`: Dada una serie de números, encontrar el máximo producto de una cantidad de dígitos consecutiva. Por ejemplo, para la serie `1234056` el mayor producto de 4 dígitos consecutivos es $1 * 2 * 3 * 4 = 24$. [3]
12. `minimoProductoEscalar`: Dados dos vectores de enteros de igual longitud v y u, encontrar el mínimo producto escalar entre todas las permutaciones posibles de cada uno. Por ejemplo, para $v = (1, 3, -5)$ y $u = (-2, 4, 1)$, el mínimo producto escalar es -25.[4].

[3] Problema 8 de Project Euler: https://projecteuler.net/problem=8
[4] Problema de la ronda 1A - Google Code Jam 2008: https://code.google.com/codejam/contest/32016/dashboard#s=p0