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

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2020

Versión 3: 23 de junio de 2020

TPI - "Reuniones Remotas"

Entrega: 6 de julio (hasta las 17 hs)

Cambios versión 3

1. ***Ejercicio 2:***
   a) ***acelerar***: se modificó el orden de complejidad pedido, ahora es $O(n \times m)$ donde n representa la longitud de la señal y m la cantidad de hablantes de la reunión
   b) ***habantesSuperpuestos***: se modificó el orden de complejidad pedido, ahora es $O(n \times m)$ donde n representa la longitud de la señal y m la cantidad de hablantes de la reunión.
2. ***se enojó*** se arregló parte de la postcondición,$j < |s|$ ahora es $j \leq |s|$: $(\exists i : \mathbb{Z})(\exists j : \mathbb{Z})0 \leq i < j \leq |s| \wedge subSec(s, i, j) = subSenial$.
3. ***silencios*** Se modifica la duración mínima de un silencio, ahora es 0,2 segundos.
4. ***reconstruir***
   a) ***reconstruirPosiciónSiCorresponde*** Se cambió $s[0] = 0$ por $sRec[i] = 0$.
   b) ***puedeReconstruirse*** Se cambió $distancia(i, j) < 5$ por $distancia(i, j) \leq 5$.
5. ***hablantesSuperpuestos*** se corrigen:
   a) $|r[h1]|$ ahora es $|r[h1]_0|$
   b) Se quita paréntesis de más en $haySilencioQueLoContiene(r[h1])_0, i, freq, umbral)$. Ahora es $haySilencioQueLoContiene(r[h1]_0, i, freq, umbral)$
6. **Cambios en el proyecto:**
   a) En **auxiliares.cpp**:
      - en la función **reunionesIguales** deben borrar las líneas que ordenan las secuencias:
        sort(reunion1[i].first.begin(), reunion1[i].first.end());
        sort(reunion2[i].first.begin(), reunion2[i].first.end());
      - en **ASSERT_SENIAL_EQ** deben borrar las líneas que ordenan las secuencias:
        sort(s1.begin(), s1.end());
        sort(s2.begin(), s2.end());

   b) Se corrige el tests de **silencios**
      Las soluciones esperadas para las secuencia dada eran $< (0, 0), (5, 6) >$ ahora es $< (5, 6) >$

Cambios versión 2

1. Se agrega a las pautas de entrega el contenido del .zip que deben entregar.
2. ***esSeñal*** se aclaró cuáles son las frecuencias válidas.
3. ***acelerar*** se arregló el predicado longitudesVálidas.
4. ***esReunionVálida*** Se arregló el predicado esMatriz. |r[a]| = |r[0]| ahora es $|r[a]_0| = |r[0]_0|$.
5. ***reconstruir*** se agrega una breve explicación de como se reconstruyen señales.
6. ***filtradoMediana***
   a) Se agregaron paréntesis en la precondición para R = 2 ∨ R = 4 ahora es (R = 2 ∨ R = 4)
   b) Se agrega una breve explicación de filtros de señales

1. Ejercicios

1. Implementar las funciones especificadas en la sección **Especificación**. No está permitido el uso de librerías de C++ fuera de las clásicas: math, vector, tuple, pair, las de input-output, etc. Consultar con la cátedra por cualquier librería adicional. Implementar los tests necesarios para asegurar el correcto funcionamiento de las funciones. Encontrarán dentro de los tests un ejemplo para cada una de las funciones los cuales **No pueden modificar**.

2. Respetar los tiempos de ejecución de peor caso para las funciones que se enumeran a continuación. Justificar.

   - *acelerar*: $O(n \times m)$ donde n representa la longitud de la señal y m la cantidad de hablantes de la reunión
   - *ordenar*: $O(m^2 \times n)$ donde m representa la cantidad de hablantes de la reunión y n la longitud de las señales.
   - *hablantesSuperpuestos*: $O(n \times m)$ donde n representa la longitud de la señal y m la cantidad de hablantes de la reunión.

3. Calcular los tiempos de ejecución en el peor caso para las siguientes funciones. Justificar.

   - *seEnojo*?
   - *silencios*
   - $filtradoMediana$

4. Implementar las funciones descriptas en la sección **Entrada/Salida**.

5. Completar (agregando) los tests estructurales necesarios para cubrir todas las líneas del archivo *solucion.cpp*. Utilizar la herramienta **lcov** para dicha tarea. Ver sección **Análisis de cobertura**.

2. Especificación

Dados los siguentes renombre de tipos:

type *señal*= $seq\langle\mathbb{Z}\rangle$
type *hablante*= $\mathbb{Z}$
type *reunion*= $seq\langle señal \times hablante\rangle$
type *intervalo*= $\mathbb{Z} \times \mathbb{Z}$

`proc esSeñal` (in s: $seq\langle\mathbb{Z}\rangle$, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: Bool) {
    `Pre` $\{True\}$
    `Post` $\{result = \text{true} \leftrightarrow esValida(s, prof, freq)\}$

    `pred esValida` (s: *señal*, prof: $\mathbb{Z}$, frec: $\mathbb{Z}$) {
        $frecValida(frec) \land profValida(prof) \land enRango(s, prof) \land duraMasDe(s, frec, 1)$
    }
    `pred frecValida` (freq: $\mathbb{Z}$) {
        $freq = 10^1$
    }
    `pred enRango` (s: *señal*, prof: $\mathbb{Z}$) {
        $(\forall\ x : \mathbb{Z})\ x \in s \longrightarrow -2^{(prof-1)} \leq x \leq 2^{(prof-1)} - 1$
    }
    `pred profValida` (prof: $\mathbb{Z}$) {
        $prof = 8 \lor prof = 16 \lor prof = 32$
    }
    `pred duraMasDe` (s: *señal*, freq: $\mathbb{Z}$, seg: $\mathbb{R}$) {
        $|s| \geq freq * seg$
    }
}

`proc séEnojó?` (in s: *señal*, in umbral: $\mathbb{Z}$, in prof : $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: Bool) {
    `Pre` $\{esValida(s, prof, freq) \land umbralValido(umbral)\}$
    `Post` $\{result = \text{true} \leftrightarrow (\exists subSenial : señal)(\exists i : \mathbb{Z})(\exists j : \mathbb{Z})0 \leq i < j \leq |s| \land subSec(s, i, j) = subSenial \land duraMasDe(subSenial, freq, 2) \land superaUmbral(subSenial, umbral)\}$

    `pred superaUmbral` (s: *señal*, umbral: $\mathbb{Z}$) {
        $tono(s) > umbral$

[1] La frecuencia está medida en Hz. 1 muestra por segundo equivale a 1 Hz

}

aux tono (s: *señal*) : $\mathbb{R} = \sum_{i=0}^{|s|-1} abs(s[i])/|s|$;

pred umbralValido (umbral: $\mathbb{Z}$) {

$umbral > 0$

}

}

proc esReuniónVálida (in r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: Bool) {

Pre $\{True\}$

Post $\{result = \text{true} \leftrightarrow reunionValida(r, prof, freq)\}$

pred reunionValida (r: *reunion*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {

$|r| > 0 \wedge esMatriz(r) \wedge senialesValidas(r, prof, freq) \wedge$
$hablantesDeReunionValidos(r, prof, freq)$

}

pred esMatriz (r: $seq\langle señal \times \mathbb{Z}\rangle$) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L |r[a]_0| = |r[0]_0|$

}

pred señalesValidas (r: *reunion*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})(\ 0 \leq i < |r| \longrightarrow_L esValida((r[i])_0, prof, freq))$

}

pred hablantesDeReunionValidos (r: *reunion*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})(0 \leq i < |r| \longrightarrow_L (0 \leq r[i]_1 < |r| \wedge$
$(\forall j : \mathbb{Z})(0 \leq j < |r| \wedge_L r[j]_1 = r[i]_1 \longrightarrow i = j)))$

}

}

proc acelerar (inout r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$) {

Pre $\{r = r_0 \wedge reunionValida(r, prof, freq) \wedge longitudesValidas(r, freq)\}$

Post $\{|r| = |r_0| \wedge_L reunionAcelerada(r, r_0)\}$

pred longitudesValidas (r: reunion, in freq: $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})\ 0 \leq i < |r| \longrightarrow_L \left\lfloor \frac{|r[i]_0|}{2} \right\rfloor \geq freq)$

}

pred reunionAcelerada (r: *reunion*, r_0: reunion) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L señalAcelerada((r[a])_0, (r_0[a])_0) \wedge mismoHablante((r[a])_1, (r_0[a])_1)$

}

pred mismoHablante (h: *hablante*, h1: *hablante*) {

$h = h_1$

}

pred señalAcelerada (s: *señal*, s_0: *señal*) {

$|s| = \left\lfloor \frac{|s_0|}{2} \right\rfloor \wedge_L\ (\forall i : \mathbb{Z})\ 0 \leq i < |s| \longrightarrow_L s[i] = s_0[2 * i + 1]$

}

}

proc ralentizar (inout r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$) {

Pre $\{r = r_0 \wedge reunionValida(r, prof, freq)\}$

Post $\{|r| = |r_0| \wedge_L reunionInterpolada(r, r_0)\}$

pred reunionInterpolada (r: *reunion*, r_0: *reunion*) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L senialInterpolada((r[a])_0, (r_0[a])_0) \wedge mismoHablante((r[a])_1, (r_0[a])_1)$

}

pred senialInterpolada (s: *señal*, s_0: *señal*) {

$|s| = 2 * |s_0| - 1 \wedge_L$

$((\forall i : \mathbb{Z})\ (0 \leq i < |s| - 1 \wedge i \mod 2 = 1 \longrightarrow_L \left\lfloor \frac{s[i-1] + s[i+1]}{2} \right\rfloor = s[i])) \wedge$

$(\forall i : \mathbb{Z}) 0 \leq i < |s_0| \longrightarrow_L s_0[i] = s[2 * i]$

}

}

proc tonosDeVozElevados (in r: *reunion*, in freq: $\mathbb{Z}$, prof : $\mathbb{Z}$, out hablantes: $seq\langle hablante\rangle$)) {

Pre $\{reunionValida(r, prof, freq)\}$

Post $\{(\forall h : hablante)(h \in hablantes \longleftrightarrow$
$(\exists t : señal \times hablante) pertenece(t, r) \wedge t_1 = h \wedge tieneElTonoMasElevado(r, t, h, prof, freq)\}$

pred tieneElTonoMasElevado (r:*reunion*, t: $señal \times hablante$, h: *hablante*, prof: $\mathbb{Z}$, freq: $\mathbb{Z}$) {

$(\forall x : \mathbb{Z})\ 0 \leq x < |r| \longrightarrow_L tono((r[x])_0) <= tono(t_0)$

}

}

proc ordenar (inout r:*reunion*, in freq: $\mathbb{Z}$, in prof : $\mathbb{Z}$) {

Pre $\{r = r_0 \wedge reunionValida(r_0, prof, freq)\}$

Post $\{|r| = |r_0| \wedge_L reunionOrdenada(r, r_0)\}$

pred reunionOrdenada (r: *reunion*, r_0: reunion) {

$mismaReunion(r, r_0) \wedge ordenada(r)$

}

pred mismaReunion (r: *reunion*, r_0: reunion) {

$(\forall a : \mathbb{Z})\ 0 \leq a < |r| \longrightarrow_L r[a] \in r_0) \wedge (\forall a : \mathbb{Z})\ 0 \leq a < |r_0| \longrightarrow_L r_0[a] \in r)$

}

pred ordenada (r: *reunion*) {

$(\forall a : \mathbb{Z})\ 0 < a < |r| \longrightarrow_L tono(r[a-1]_0) \geq tono(r[a]_0)$

}

}

proc silencios (in s: *señal*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, in umbral: amplitud, out intervalos: $seq\langle intervalo\rangle$) {

Pre $\{esValida(s, prof, freq) \wedge umbralValido(umbral)\}$

Post $\{sinRepetidos(intervalos) \wedge_L$
$(\forall i : \mathbb{Z}) 0 \leq i < |intervalos| \leftrightarrow esSilencio(s, intervalos[i], freq, umbral)\}$

pred sinRepetidos (intervalos : $seq\langle intervalo\rangle$) {

$\neg hayRepetido(intervalos)$

}

pred hayRepetido (intervalos : $seq\langle intervalo\rangle$) {

$(\exists i : \mathbb{Z})(\exists j : \mathbb{Z})(0 \leq i < |intervalos| \wedge 0 \leq j < |intervalos| \wedge i \neq j \wedge intervalos[i]_0 = intervalos[j]_0 \wedge$
$intervalos[i]_1 = intervalos[j]_1)$

}

pred esSilencio (s: *señal*, inter : intervalo, freq: $\mathbb{Z}$, umbral: $\mathbb{Z}$) {

$intervaloEnRango(s, inter) \wedge_L duraMasDe(subSec(s, inter_0, inter_1), freq, 0.2) \wedge$
$intervaloDeSilencio(s, inter, umbral) \wedge$
$noHayMasGrandeQueLoContiene(s, inter, umbral)$

}

pred intervaloEnRango (s:*señal*, inter : intervalo) {

$0 \leq inter_0 < inter_1 < |s|$

}

pred intervaloDeSilencio (s: *señal*, inter : intervalo, umbral: $\mathbb{Z}$) {

$(\forall k : \mathbb{Z})\ inter_0 \leq k \leq inter_1 \longrightarrow_L abs(s[k]) < umbral$

}

pred noHayMasGrandeQueLoContiene (s: *señal*, inter : intervalo, umbral: $\mathbb{Z}$) {

$inter_0 \neq 0 \longrightarrow_L abs(s[inter_0 - 1]) \geq umbral \wedge$
$inter_1 \neq |s| - 2 \longrightarrow_L abs(s[inter_1 + 1]) \geq umbral$

}

}

proc hablantesSuperpuestos (in r: *reunion*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, in umbral: $\mathbb{Z}$, out result: Bool) {

Pre $\{esReunionValida(r, prof, freq) \wedge umbralValido(umbral)\}$

Post $\{result = true \leftrightarrow (\exists h1, h2 : hablante)\ 0 \leq h1 < h2 < |r| \wedge_L \neg seRespetan(r, h1, h2, freq, umbral)\}$

pred seRespetan (r: *reunion*, h1: *hablante*, h2: *hablante*, freq: $\mathbb{Z}$, umbral: $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})\ 0 \leq i < |r[h1]_0| \longrightarrow_L$
$(\neg haySilencioQueLoContiene(r[h1]_0, i, freq, umbral) \rightarrow$
$haySilencioQueLoContiene(r[h2]_0, i, freq, umbral))$
}
pred haySilencioQueLoContiene (s : *señal*, i : $\mathbb{Z}$, freq : $\mathbb{Z}$, umbral : $\mathbb{Z}$) {

$(\exists inter : intervalo)(intervaloEnRango(s, inter) \wedge_L$
$inter_0 \leq i \leq inter_1 \wedge$
$esSilencio(s, inter, freq, umbral)$
}
}

Reconstrucción de señales

Dada una señal de audio, es posible que el proceso de muestreo haya fallado, con lo que el sistema de adquisición de muestras puso un valor cero en algunas posiciones. En estos casos, la señal debe ser reconstruida de la siguiente manera: si en una posición hay un valor faltante, se debe completar con el promedio de los dos valores no nulos de la secuencia original más cercanos. Sin embargo, estos faltantes no pueden tener más de 4 muestras vacías, ya que una reconstrucción por este método perjudicaría la señal original. Por otro lado, también se debe tener en cuenta la naturaleza sinusoidal de la señal de audio, que posee necesariamente pasajes por ceros. En estos casos, si la muestra es cero, pero corresponde a un valor esperado de la señal, no debe reemplazarse.

proc reconstruir (in s: *señal*, in prof: $\mathbb{Z}$, in freq: $\mathbb{Z}$, out result: *señal*) {
Pre $\{esValida(s, prof, freq) \wedge_L s[0] \neq 0 \wedge s[|s| - 1] \neq 0 \wedge puedeReconstruirse(s)\}$
Post $\{esReconstruida(s, result)\}$

pred puedeReconstruirse (s: *señal*) {
$(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})(0 \leq i < j < |s| \wedge_L s[i] \neq 0 \wedge s[j] \neq 0 \wedge todosCeros(subSeq(s, i + 1, j)) \longrightarrow_L distancia(i, j) \leq 5)$
}
pred todosCeros (s: *señal*) {
$(\forall i : \mathbb{Z})0 \leq i < |s| \rightarrow_L s[i] = 0$
}
pred esReconstruida (s: *señal*, sRec: *señal*) {
$|s| = |sRec| \wedge_L$
$(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L$
$((s[i] = 0 \wedge reconstruirPosicionSiCorresponde(s, sRec, i)) \vee$
$(s[i] \neq 0 \wedge sRec[i] = s[i]))$
}
pred reconstruirPosicionSiCorresponde (s: *señal*, sRec:*señal*, i: $\mathbb{Z}$) {
$(esPasajePorCero(s, i) \wedge sRec[i] = 0) \vee (\neg esPasajePorCero(s, i) \wedge esValorEnPosicion(s, sRec[i], i))$
}
pred esPasajePorCero (s: *señal*, i: $\mathbb{Z}$) {
$signo(s[i - 1]) * signo(s[i + 1]) = -1$
}
pred esValorEnPosicion (s: *señal*, valor: $\mathbb{Z}$, i: $\mathbb{Z}$) {
$(\exists j : \mathbb{Z})(\exists k : \mathbb{Z})(masCercanosNoNulos(s, i, j, k) \wedge_L valor = \lfloor \frac{s[k]+s[j]}{2} \rfloor)$
}
pred masCercanosNoNulos (s: *señal*, i: $\mathbb{Z}$, j: $\mathbb{Z}$, k: $\mathbb{Z}$) {
$0 \leq j < i < k < |s| \wedge_L s[j] \neq 0 \wedge s[k] \neq 0 \wedge distancia(j, k) \leq 5 \wedge$
$(\forall l : \mathbb{Z})(0 \leq l < |s| \wedge_L s[l] \neq 0 \longrightarrow distancia(i, l) \geq max(distancia(i, j), distancia(i, k)))$
}
aux distancia (j: $\mathbb{Z}$, i: $\mathbb{Z}$) : $\mathbb{Z}$ $= |i - j|$;
aux signo (i: $\mathbb{Z}$) : $\mathbb{Z}$ $=$ if $x < 0$ then -1 else (if $x > 0$ then 1 else 0 fi) fi ;
}

Filtrado de señales

El filtrado mediano se aplica para poder eliminar de la señal valores propios de ruido de muy corta duración pero de gran potencia. Para esta tarea se utilizan filtros. Los filtros son operaciones matemáticas que toman una señal y la modifican produciendo otra señal. En este caso para filtrar una posición i dentro de la señal se tomará una ventana de F muestras centradas en i, siendo una ventana una subsecuencia de la secuencia original, y se reemplazará el valor en la posición i por

el valor mediano de la secuencia de muestras tomada. Se define F = 2 ∗ R + 1. Los valores extremos de la señal de entrada no son filtrados ya que no es posible obtener la ventana de F muestras con las R muestras anteriores y las R próximas.

```
proc filtradoMediana (inout s: señal, R: ℤ, in prof: ℤ, in freq: ℤ)  {
      Pre {esValida(s, prof, freq) ∧ s = s0 ∧ (R = 2 ∨ R = 4)}
      Post {esFiltrada(s0, s, R)}
      pred esFiltrada (s: señal, sFilt: señal, R: ℤ) {
          |s| = |sFilt| ∧ (∀i : ℤ)(0 ≤ i < |s| →L
          (coincidenExtremos(s, sFilt, i, R) ∨L esValorFiltrado(s, sFilt, i, R)))
      }
      pred coincidenExtremos (s: señal, sFilt: señal, i: ℤ, R: ℤ) {
          (i < R ∨ i ≥ |s| − R) ∧ s[i] = sFilt[i]
      }
      pred esValorFiltrado (s: señal, sFilt: señal, i: ℤ, R: ℤ) {
          (∃w : señal)|w| = 2∗R+1∧mismosValores(w, subSeq(s, i−R, i+R+1))∧valoresOrdenados(w)∧L sFilt[i] = w[R]
      }
      pred mismosValores (w: señal, q: señal) {
          (∀v : ℤ)v ∈ w ⟶L #apariciones(w, v) = #apariciones(q, v)
      }
      pred valoresOrdenados (w: señal) {
          (∀i : ℤ)0 < i < |w| ⟶L w[i − 1] ≤ w[i]
      }
}
```

3. Entrada/Salida

Implementar las siguientes funciones.

1. `void escribirSenial(senial s, string nombreArchivo)`

   Que escribe una señal en un archivo con una única línea siguiendo el siguiente formato:

   $s_0\ s_1\ \ldots\ s_{n-1}$. En donde $s_0\ s_1\ \ldots\ s_{n-1}$ es el contenido de la señal.

   Por ejemplo $s =< 1, 5, 3, 5 >$, el archivo debe contener:
   1 5 3 5

2. `senial leerSenial(string nombreArchivo)`

   Que dado un nombre de archivo `nombreArchivo`, deberá devolver la señal correspondiente. El formato del archivo debe ser el mismo que en el item anterior.

4. Compilación

1. Bajar del campus de la materia Archivos TPI.
2. Descomprimir el ZIP.
3. Dentro del ZIP van a encontrar dos carpetas: CLion y Terminal la primera contiene el proyecto para CLion y la segunda para compilar desde la Terminal.

   - En CLion:
     Seleccionar File → Open... y seleccionar la carpeta con el nombre reunionesRemotas que se encuentra en TPI/-CLion.
   - Desde la terminal:
     En la carpeta TPI/Terminal/reunionesRemotas compilar:
     g++ -o reunionesRemotas lib/* src/* tests/* -pthread

     Donde:

     - lib: carpeta que contiene los archivos de Google Test
     - src: carpeta que contiene el código
     - tests: carpeta que contiene los tests

5. Análisis de cobertura

Para realizar el análisis de cobertura de código utilizaremos la herramienta Gcov, que es parte del compilador GCC.

- En CLion:
  El target *reunionesRemotas* ya está configurado para generar información de cobertura de código en tiempo de compilación. Esta información estará en archivos con el mismo nombre que los códigos fuente del TP, pero con extensión `*.gcno`. Al ejecutar los casos de test, se generarán en el mismo lugar que los `*.gcno` otra serie de archivos con extensión `*.gcda`.
- Desde la terminal en la carpeta TPI/Terminal/reunionesRemotas que se encuentra en TPI.zip:
  g++ -o reunionesRemotas lib/* src/definiciones.h src/auxiliares.h src/auxiliares.cpp src/solucion.h src/solucion.cpp tests/* -pthread -g −−coverage

  Al ejecutarse generará la información de cobertura de código en tiempo de compilación. Esta información estará en archivos con el mismo nombre que los códigos fuente del TP, pero con extensión `*.gcno`. Al ejecutar los casos de test, se generarán en el mismo lugar que los `*.gcno` otra serie de archivos con extensión `*.gcda`. Para ejecutar los casos de tests en la terminal:
  - Linux: ./reunionesRemotas
  - Windows: reunionesRemotas.exe

Los siguientes pasos son iguales tanto si compilan desde CLion como desde la terminal.
Ejecutar el siguiente comando en la terminal:

⟶ `lcov --capture --directory reunionesRemotas --output-file coverage.info`

Se generará el archivo `coverage.info` que luego podremos convertir a HTML para su visualización con el siguiente comando:

⟶ `genhtml coverage.info --output-directory cobertura`

Finalmente, se generará un archivo `index.html` dentro de `cobertura` con el reporte correspondiente. Utilizar cualquier navegador para verlo.

Para mayor información, visitar:
`https://medium.com/@naveen.maltesh/generating-code-coverage-report-using-gnu-gcov-lcov-ee54a4de3f11`.

Términos y condiciones

El trabajo práctico se realiza de manera grupal con grupos de exactamente 4 personas. Para aprobar el trabajo se necesita:

- Que todos los ejercicios estén resueltos.
- Que las soluciones sean correctas. Deben respetar la especificación dada.
- Que todos los tests provistos por la cátedra funcionen (no pueden ser modificados).
- Que las soluciones sean prolijas: evitar repetir implementaciones innecesariamente y usar adecuadamente funciones auxiliares.
- Que los test cubran todas las lineas de las funciones.
- Que estén justificados los ordenes de complejidad tanto los que tienen que respetarse como los que deben calcularse (incluyendo las funciones auxiliares que usan). Las justificaciones deben ser entregadas en un pdf.

Pautas de Entrega

El trabajo debe ser subido al campus en la sección Trabajos Prácticos en la fecha estipulada.

La entrega debe contener:

- El proyecto completo, se requiere que estén todos los archivos necesarios para compilarlo.
- Los tests necesarios que garanticen el cubrimiento de líneas de código.
- El pdf con la justificación de los ordenes de complejidad.

Fecha de entrega: 6 de Julio (hasta las 17:00hs)