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

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2020

Versión 1: 12 de junio de 2020

TPI - "Reuniones Remotas"

Entrega: 6 de julio (hasta las 17 hs)

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

3. Calcular los tiempos de ejercución 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: $\mathbb{Z}$) {
$frec = 10^1$
}
`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: $\mathbb{Z}$) {
$prof = 8 \lor prof = 16 \lor prof = 32$

[1] La frecuencia está medida en Hz. 1kHz (1000 muestras por segundo) equivale a 1000 Hz

```
    }
    pred duraMasDe (s: señal, freq: ℤ, seg: ℝ) {
        |s| >= frec * seg
    }
}

proc séEnojó? (in s: señal, in umbral: ℤ, in prof : ℤ, in freq: ℤ, out result: Bool) {
    Pre {esValida(s, prof, freq) ∧ umbralValido(umbral)}
    Post {result = true ↔ (∃subSenial : señal)(∃i : ℤ)(∃j : ℤ)0 ≤ i < j < |s| ∧ subSec(s, i, j) = subSenial ∧
    duraMasDe(subSenial, freq, 2) ∧ superaUmbral(subSenial, umbral)}

    pred superaUmbral (s: señal, umbral: ℤ) {
        tono(s) > umbral
    }
    aux tono (s: señal) : ℝ = ∑_{i=0}^{|s|-1} abs(s[i])/|s| ;
    pred umbralValido (umbral: ℤ) {
        umbral > 0
    }
}

proc esReunión Válida (in r: reunion, in prof: ℤ, in freq: ℤ, out result: Bool) {
    Pre {True}
    Post {result = true ↔ reunionValida(r, prof, freq)}

    pred reunionValida (r: reunion, prof: ℤ, freq: ℤ) {
        |r| > 0 ∧ esMatriz(r) ∧ senialesValidas(r, prof, frec) ∧
        hablantesDeReunionValidos(r, prof, frec)
    }
    pred esMatriz (r: seq⟨señal × ℤ⟩) {
        (∀a : ℤ) 0 ≤ a < |r| ⟶_L |r[a]| = |r[0]|
    }
    pred señalesValidas (r: reunion, prof: ℤ, freq: ℤ) {
        (∀i : ℤ)( 0 ≤ i < |r| ⟶_L esValida((r[i])_0, prof, freq))
    }
    pred hablantesDeReunionValidos (r: reunion, prof: ℤ, freq: ℤ) {
        (∀i : ℤ)(0 ≤ i < |r| ⟶_L (0 ≤ r[i]_1 < |r| ∧
        (∀j : ℤ)(0 ≤ j < |r| ∧_L r[j]_1 = r[i]_1 ⟶ i = j)))
    }
}

proc acelerar (inout r: reunion, in prof: ℤ, in freq: ℤ) {
    Pre {r = r_0 ∧ reunionValida(r, prof, freq) ∧ longitudesValidas(r, freq)}
    Post {|r| = |r_0| ∧_L reunionAcelerada(r, r_0)}

    pred longitudesValidas (r: reunion, in freq: ℤ) {
        (∀i : ℤ) 0 ≤ i < |r| ⟶_L ⌊ |r[i]_0| / 2 ⌋ ≥ freq * 1000)
    }
    pred reunionAcelerada (r: reunion, r_0: reunion) {
        (∀a : ℤ) 0 ≤ a < |r| ⟶_L señalAcelerada((r[a])_0, (r_0[a])_0) ∧ 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| = ⌊ |s_0| / 2 ⌋ ∧_L (∀i : ℤ) 0 ≤ i < |s| ⟶_L s[i] = s_0[2 * i + 1]
    }
}

proc ralentizar (inout r: reunion, in prof: ℤ, in freq: ℤ) {
    Pre {r = r_0 ∧ reunionValida(r, prof, freq)}
    Post {|r| = |r_0| ∧_L reunionInterpolada(r, r_0)}
```

```
    pred reunionInterpolada (r: reunion, r0: reunion) {
        (∀a : ℤ) 0 ≤ a < |r| ⟶L senialInterpolada((r[a])0, (r0[a])0) ∧ mismoHablante((r[a])1, (r0[a])1)
    }
    pred senialInterpolada (s: señal, s0: señal) {
        |s| = 2 * |s0| − 1 ∧L
        ((∀i : ℤ) (0 ≤ i < |s| − 1 ∧ i  mód  2 = 1 ⟶L ⌊(s[i − 1] + s[i + 1]) / 2⌋ = s[i])) ∧
        (∀i : ℤ)0 ≤ i < |s0| ⟶L s0[i] = s[2 * i]
    }
}

proc tonosDeVozElevados (in r: reunion, in freq: ℤ, prof : ℤ, out hablantes: seq⟨hablante⟩)  {
    Pre {reunionValida(r, prof, freq)}
    Post {(∀h : hablante)(h ∈ hablantes ⟷
    (∃t : señal × hablante)pertenece(t, r) ∧ t1 = h ∧ tieneElTonoMasElevado(r, t, h, prof, freq)}

    pred tieneElTonoMasElevado (r:reunion, t: señal × hablante, h: hablante, prof: ℤ, freq: ℤ) {

        (∀x : ℤ) 0 ≤ x < |r| ⟶L tono((r[x])0) <= tono(t0)
    }
}

proc ordenar (inout r:reunion, in freq: ℤ, in prof : ℤ)  {
    Pre {r = r0 ∧ reunionValida(r0, prof, freq)}
    Post {|r| = |r0| ∧L reunionOrdenada(r, r0)}

    pred reunionOrdenada (r: reunion, r0: reunion) {
        mismaReunion(r, r0) ∧ ordenada(r)
    }
    pred mismaReunion (r: reunion, r0: reunion) {
        (∀a : ℤ) 0 ≤ a < |r| ⟶L r[a] ∈ r0) ∧ (∀a : ℤ) 0 ≤ a < |r0| ⟶L r0[a] ∈ r)
    }
    pred ordenada (r: reunion) {
        (∀a : ℤ) 0 < a < |r| ⟶L tono(r[a − 1]0) ≥ tono(r[a]0)
    }
}

proc silencios (in s: señal, in prof: ℤ, in freq: ℤ, in umbral: amplitud, out intervalos: seq⟨intervalo⟩)  {
    Pre {esValida(s, prof, freq) ∧ umbralValido(umbral)}
    Post {sinRepetidos(intervalos) ∧L
    (∀i : ℤ)0 ≤ i < |intervalos| ↔ esSilencio(s, intervalos[i], freq, umbral)}

    pred sinRepetidos (intervalos : seq⟨intervalo⟩) {
        ¬hayRepetido(intervalos)
    }
    pred hayRepetido (intervalos : seq⟨intervalo⟩) {
        (∃i : ℤ)(∃j : ℤ)(0 ≤ i < |intervalos| ∧ 0 ≤ j < |intervalos| ∧ i ≠ j ∧ intervalos[i]0 = intervalos[j]0 ∧
        intervalos[i]1 = intervalos[j]1)
    }
    pred esSilencio (s: señal, inter : intervalo, freq: ℤ, umbral: ℤ) {

        intervaloEnRango(s, inter) ∧L duraMasDe(subSec(s, inter0, inter1), frec, 0.1) ∧
        intervaloDeSilencio(s, inter, umbral) ∧
        noHayMasGrandeQueLoContiene(s, inter, umbral)

    }
    pred intervaloEnRango (s:señal, inter : intervalo) {
        0 ≤ inter0 < inter1 < |s|
    }
    pred intervaloDeSilencio (s: señal, inter : intervalo, umbral: ℤ) {

        (∀k : ℤ) inter0 ≤ k ≤ inter1 ⟶L abs(s[k]) < umbral
```

```
    }
    pred noHayMasGrandeQueLoContiene (s: señal, inter : intervalo, umbral: ℤ) {

        inter_0 ≠ 0 ⟶_L abs(s[inter_0 − 1]) ≥ umbral ∧
        inter_1 ≠ |s| − 2 ⟶_L abs(s[inter_1 + 1]) ≥ umbral

    }
}

proc hablantesSuperpuestos (in r: reunion, in prof: ℤ, in freq: ℤ, in umbral: ℤ, out result: Bool) {
    Pre {esReunionValida(r, prof, freq) ∧ umbralValido(umbral)}
    Post {result = true ↔ (∃h1, h2 : hablante) 0 ≤ h1 < h2 < |r| ∧_L ¬seRespetan(r, h1, h2, freq, umbral)}

    pred seRespetan (r: reunion, h1: hablante, h2: hablante, freq: ℤ, umbral: ℤ) {
        (∀i : ℤ) 0 ≤ i < |r[h1]| ⟶_L
        (¬haySilencioQueLoContiene(r[h1])_0, i, freq, umbral) →
        haySilencioQueLoContiene(r[h2])_0, i, freq, umbral))
    }
    pred haySilencioQueLoContiene (s : señal, i : ℤ, freq : ℤ, umbral : ℤ) {

        (∃inter : intervalo)(intervaloEnRango(s, inter) ∧_L
        inter_0 ≤ i ≤ inter_1 ∧
        esSilencio(s, inter, freq, umbral)
    }
}

proc reconstruir (in s: señal, in prof: ℤ, in freq: ℤ, out result: señal) {
    Pre {esValida(s, prof, freq) ∧_L s[0] ≠ 0 ∧ s[|s| − 1] ≠ 0 ∧ puedeReconstruirse(s)}
    Post {esReconstruida(s, result)}

    pred puedeReconstruirse (s: señal) {
        (∀i : ℤ)(∀j : ℤ)(0 ≤ i < j < |s| ∧_L s[i] ≠ 0 ∧ s[j] ≠ 0 ∧ todosCeros(subSeq(s, i + 1, j)) ⟶_L distancia(i, j) < 5)
    }
    pred todosCeros (s: señal) {
        (∀i : ℤ)0 ≤ i < |s| →_L s[i] = 0
    }
    pred esReconstruida (s: señal, sRec: señal) {
        |s| = |sRec| ∧_L
        (∀i : ℤ)(0 ≤ i < |s| →_L
        ((s[i] = 0 ∧ reconstruirPosicionSiCorresponde(s, sRec, i)) ∨
        (s[i] ≠ 0 ∧ sRec[i] = s[i]))
    }
    pred reconstruirPosicionSiCorresponde (s: señal, sRec:señal, i: ℤ) {
        (esPasajePorCero(s, i) ∧ s[0] = 0) ∨ (¬esPasajePorCero(s, i) ∧ esValorEnPosicion(s, sRec[i], i))
    }
    pred esPasajePorCero (s: señal, i: ℤ) {
        signo(s[i − 1]) ∗ signo(s[i + 1]) = −1
    }
    pred esValorEnPosicion (s: señal, valor: ℤ, i: ℤ) {
        (∃j : ℤ)(∃k : ℤ)(masCercanosNoNulos(s, i, j, k) ∧_L valor = ⌊(s[k]+s[j])/2⌋)
    }
    pred masCercanosNoNulos (s: señal, i: ℤ, j: ℤ, k: ℤ) {
        0 ≤ j < i < k < |s| ∧_L s[j] ≠ 0 ∧ s[k] ≠ 0 ∧ distancia(j, k) ≤ 5 ∧
        (∀l : ℤ)(0 ≤ l < |s| ∧_L s[l] ≠ 0 ⟶ distancia(i, l) ≥ max(distancia(i, j), distancia(i, k)))
    }
    aux distancia (j: ℤ, i: ℤ) : ℤ = |i − j| ;
    aux signo (i: ℤ) : ℤ = if x < 0 then −1 else (if x > 0 then 1 else 0 fi) fi ;
}

proc filtradoMediana (inout s: señal, R: ℤ, in prof: ℤ, in freq: ℤ) {
    Pre {esValida(s, prof, freq) ∧ s = s_0 ∧ R = 2 ∨ R = 4}
    Post {esFiltrada(s_0, s, R)}
    pred esFiltrada (s: señal, sFilt: señal, R: ℤ) {
```

$|s| = |sFilt| \land (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L$
$(coincidenExtremos(s, sFilt, i, R) \lor_L esValorFiltrado(s, sFilt, i, R)))$

}

`pred coincidenExtremos` (s: *señal*, sFilt: *señal*, i: $\mathbb{Z}$, R: $\mathbb{Z}$) {
$(i < R \lor i \geq |s| - R) \land s[i] = sFilt[i]$

}

`pred esValorFiltrado` (s: *señal*, sFilt: *señal*, i: $\mathbb{Z}$, R: $\mathbb{Z}$) {
$(\exists w : señal)|w| = 2{*}R{+}1 \land mismosValores(w, subSeq(s, i{-}R, i{+}R{+}1)) \land valoresOrdenados(w) \land_L sFilt[i] = w[R]$

}

`pred mismosValores` (s: *señal*, q: *señal*) {
$(\forall v : \mathbb{Z})v \in s \longrightarrow_L \#apariciones(s, v) = \#apariciones(q, v)$

}

`pred valoresOrdenados` (s: *señal*) {
$(\forall i : \mathbb{Z})0 < i < |s| \longrightarrow_L s[i-1] \leq s[i]$

}

}

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.

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