UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructuras de Datos I

Recuperatorio del primer parcial– Primer cuatrimestre de 2022

1a	1b	2a	2b	2c	3a	3b	4
15	20	10	5	16	5	10	14

Apellido Bruginski ..... Nombre Lionel .....

LU 385/21 ..... Número de comisión 5 .....

Cantidad de hojas entregadas 4

El parcial se aprueba con 70 puntos. 95/100 (A)

Ejercicio 1. [35 puntos] Para evitar el agotamiento que le produce el calor de diciembre en el hemisferio sur, este año Papá Noel decidió entregar los regalos la medianoche del 4 de julio. Tiene un *listado de casas* que debe visitar. Cada casa se identifica con un código numérico único de tipo `Casa` (renombre de $\mathbb{Z}$). Un listado de casas es un valor de tipo $seq\langle$`Casa`$\rangle$. Por ejemplo, un listado de casas podría ser:

$$\ell = \texttt{[3, 1, 2]}$$

que indica que Papá Noel debe visitar tres casas, identificadas por los códigos #3, #1 y #2. Además, Papá Noel cuenta con un *mapa* que indica las distancias en metros entre las casas. En este contexto, un mapa es una secuencia de triplas, es decir `Mapa` es un renombre de $seq\langle$`Casa` $\times$ `Casa` $\times \mathbb{R}\rangle$. Por ejemplo, el siguiente mapa:

```
[(1, 2, 50), (1, 3, 30), (2, 3, 45)]
```

indica que la distancia entre la casa #1 y la casa #2 es de 50 m, la distancia entre la casa #1 y la casa #3 es de 30 m, y la distancia entre la casa #2 y la casa #3 es de 45 m. La distancia es *simétrica*, es decir que la distancia entre la casa #3 y la casa #2 es igual a la distancia entre la casa #2 y la casa #3.

Decimos que un mapa es *válido* con respecto a un listado de casas dado si todos los pares de casas distintas aparecen exactamente una vez, el mapa no contiene otras triplas (además de las ya mencionadas) y todas las distancias son estrictamente positivas. El orden en el que aparecen las casas es indistinto. Por ejemplo, el mapa de arriba podría escribirse, de manera equivalente, como el siguiente mapa, también válido:

```
[(3, 1, 30), (2, 1, 50), (2, 3, 45)]
```

Se pide:

a) **[15 puntos]** Definir una función `aux dist(`m `: Mapa,` c_1, c_2 `: Casa) :` $\mathbb{R}$ que, dado un mapa y dos casas, encuentre la distancia entre c_1 y c_2 que está registrada en el mapa[1]. Por ejemplo, si $m =$ `[(3, 1, 30), (2, 1, 50), (2, 3, 45)]` es un mapa y las casas son $c_1 = 3$, $c_2 = 2$, el resultado debe ser 45. Se puede asumir que $c_1 \neq c_2$ y que el mapa es válido con respecto a algún listado ℓ que incluye a las casas c_1 y c_2.

b) [20 puntos] Dado un listado de casas $\ell = [c_1, c_2, \ldots, c_n]$ y un mapa válido m, la distancia total recorrida al visitar las casas, en el orden en que aparecen en el listado, es la suma de las distancias entre las casas consecutivas, es decir, $\mathsf{dist}(m, c_1, c_2) + \mathsf{dist}(m, c_2, c_3) + \ldots + \mathsf{dist}(m, c_{n-1}, c_n)$. Especificar el problema que recibe un listado de casas ℓ (sin repetidos) y un mapa válido para ℓ, y reordena el listado de tal manera que la distancia total recorrida sea mínima.

Por ejemplo, si el listado es $\ell =$ `[3, 2, 1]` y el mapa es $m =$ `[(1, 3, 30), (2, 1, 50), (3, 2, 45)]`, la distancia total recorrida por ℓ es $\mathsf{dist}(m, 3, 2) + \mathsf{dist}(m, 2, 1) = 45 + 50 = 95$. Una manera posible de reordenar el listado para que la distancia total recorrida sea mínima es tomando $\ell' =$ `[1, 3, 2]`, cuya distancia total es $\mathsf{dist}(m, 1, 3) + \mathsf{dist}(m, 3, 2) = 30 + 45 = 75$.

Importante. Se puede asumir dado un predicado `esMapaVálido(`m `: Mapa,` ℓ `:` $seq\langle$`Casa`$\rangle$`)` que es verdadero cuando el mapa m es válido con respecto a la lista de casas ℓ.

Ejercicio 2. [35 puntos] Dado el siguiente programa:

$P_c : \{|s| > 0 \land i = 0 \land r = \texttt{true}\}$

```
while (i + 1 < |s|) do
  r := r && (s[i + 1] = s[i] + 1);
  i := i + 1
endwhile
```

$Q_c : \{r = \texttt{true} \rightarrow s[|s| - 1] = s[0] + (|s| - 1)\}$

a) [10 puntos] Proponer un invariante I para el ciclo.

b) [5 puntos] Demostrar que $(I \land \neg B) \Rightarrow Q_c$.

c) [20 puntos] Demostrar que $\{I \land B\}$ ⟨cuerpo del ciclo⟩ $\{I\}$.

[1] La distancia es el número que figura explícitamente en el mapa.

Ejercicio 3. [15 puntos] Dados el siguiente programa S y la siguiente especificación:

```
if i < j then
  s[i] := s[j]
else
  s[j] := s[i]
endif
```

```
proc arruinarOrden (inout s: seq⟨ℤ⟩, in i, j : ℤ) {
      Pre {0 ≤ i < |s| ∧ 0 ≤ j < |s| ∧ i ≠ j ∧ ordenada(s)}
      Post {¬ordenada(s)}
}
pred ordenada (s: seq⟨ℤ⟩) {
    (∀k : ℤ)(0 ≤ k < |s| − 1 ⟶_L s[k] < s[k + 1])
}
```

a) [5 puntos] Calcular la precondición más débil del programa S con respecto a la postcondición: *wp(S; Post)*.

b) [10 puntos] Demostrar que el programa es correcto con respecto a la especificación propuesta.

Ejercicio 4. [15 puntos] Llamamos *plazo* a una tripla de enteros (h, m, s) que representan respectivamente un número de horas, un número de minutos y un número de segundos. Un plazo se puede expresar completamente en segundos considerando que hay 60 segundos en un minuto y 60 minutos en una hora. Los valores de h, m, s pueden ser negativos. Decimos que un plazo (h, m, s) es *canónico* si el número de minutos y el número de segundos se encuentran en el rango $\{0, 1, \ldots, 59\}$. Por ejemplo, $(0, 62, 23)$ no es un plazo canónico, pero se puede reexpresar como $(1, 2, 23)$ que sí es un plazo canónico y determina la misma cantidad de segundos.

Se quiere implementar una operación que recibe tres enteros que representan un plazo, incrementa en uno el número de segundos y lo convierte a su forma canónica. La especificación formal es la siguiente:

```
proc avanzarUnSegundo (inout h, m, s: ℤ) {
      Pre {h_0 = h ∧ m_0 = m ∧ s_0 = s}
      Post {esCanónico(h, m, s) ∧ enSegundos(h, m, s) = enSegundos(h_0, m_0, s_0) + 1}
}
pred esCanónico (h, m, s : ℤ) {
    0 ≤ m < minutosPorHora() ∧ 0 ≤ s < segundosPorMinuto()
}
aux enSegundos (h, m, s : ℤ) : ℤ = s + segundosPorMinuto() * (m + minutosPorHora() * h);
aux segundosPorMinuto() : ℤ = 60;
aux minutosPorHora() : ℤ = 60;
```

El programa es el siguiente:

```
const int segundosPorMinuto = 60;
const int minutosPorHora = 60;

void avanzarUnSegundo(int& h, int& m, int& s) {
  s = s + 1;
  ajustar(h, m, minutosPorHora);
  ajustar(m, s, segundosPorMinuto);
}

void ajustar(int& x, int& y, int limite) {
  while (y >= limite) {
    y = y - limite;
    x = x + 1;
  }
  while (y < 0) {
    y = y + limite;
    x = x - 1;
  }
}
```

Escribir un *test* que encuentre el defecto presente en el código. Es decir, escribir una entrada tal que cumple con la precondición pero tal que el resultado de ejecutar el código no cumple la postcondición. (Justificar la respuesta).

7) Lionel Braginski

HOJA N°

FECHA 4/7/22

a) $Cosa = \mathbb{Z}$ $\quad MAPA = seq\langle Cosa \times Cosa \times \mathbb{R} \rangle$

```
aux dist(m: MAPA, c1, c2: Cosa): R =
  Σ_{i=0}^{|m|-1} if (aparece(c1, m[i]) ∧ aparece(c2, m[i])) then (m[i])_2 else 0 ;

pred aparece(c: Cosa, t: Cosa × Cosa × R) {
  t0 = c ∨ t1 = c
}
```

(t representa una posición del mapa)

b)
```
proc optimizar(inout l: seq⟨Cosa⟩, in m: Mapa) {
  Pre {esMapaValido(m, l) ∧ l = l0 ∧ |l| > 1 ∧ sonCosasValidas(l)}
  Post {esListaDeDistanciaMinima(l, l0, m)}
}

aux distanciaTotal(l: seq⟨Cosa⟩, m: Mapa): R =
  Σ_{i=0}^{|l|-2} dist(m, l[i], l[i+1]) ;

pred mismosElementos(s, l: seq⟨Cosa⟩) {
  |s| = |l| ∧ (∀c: Z)(0 ≤ c < |s| → #apariciones(s[c], s) = #apariciones(s[c], l))
}

aux #apariciones(c: Cosa, l: seq⟨Cosa⟩): Z = Σ_{i=0}^{|l|} if c ∈ l then 1 else 0 ;

pred esListaDeDistanciaMinima(l, l0: seq⟨Cosa⟩, m: Mapa) {
  (∃l': seq⟨Cosa⟩)(mismosElementos(l0, l') ∧ l = l' ∧ distanciaTotal(l', m) ≤ distanciaTotal(l0, m)
  ∧ ¬(∃l'': seq⟨Cosa⟩)(mismosElementos(l', l'') ∧ distanciaTotal(l'', m) < distanciaTotal(l', m)))
}

pred sonCosasValidas(l: seq⟨Cosa⟩) {
  (∀i: Z)(0 ≤ i < |l| → #apariciones(l[i], l) = 1)
}
```

es lista de distancia mínima otra

ordenados para el lo) quisiera que:

- distancia Total de l respecto a m sea la menor posible
- existe l' con mismos elementos que l_0 y su dist total es ≤ a dist total de l_0
- no existe l' con mismos elems que l_0 y su dist total es < a dist total de l'

pred la dist minima

pred es lista de distancia minima (l, l_0 : seq⟨Cosa⟩, m: Mapa) {

mismosElementos (l, l_0) ∧ distanciaTotal(l, m) ≤ distanciaTotal(l_0, m)

redundante → ∧ ¬(∃ l' : seq⟨Cosa⟩)(mismosElementos (l, l') ∧ distanciaTotal(l', m) < distanciaTotal(l, m))

}

[illegible]

NOTA

2) Leonel Brogioli

HOJA N°

FECHA 4/7/22

$P_r \equiv |s| > 0 \wedge i = 0 \wedge r = True$

$B \equiv i+1 < |s| \equiv i < |s|-1$

$\neg B \equiv i \geq |s|-1$

```
While(i+1 < s.size()) do
  S [ s: r := r && (s[i+1] = s[i]+1);
      s: i := i+1;
endwhile
```

$Q_r \equiv r = True \rightarrow s[|s|-1] = s[0] + (|s|-1)$

a) $I \equiv 0 \leq i \leq |s|-1 \wedge_L r = True \rightarrow s[i] = s[0] + i$ ✓

b) $\{I \wedge \neg B\} \equiv \underbrace{0 \leq i \leq |s|-1 \wedge i \geq |s|-1}_{\hookrightarrow i = |s|-1} \wedge_L r = True \rightarrow s[i] = s[0] + i$

entonces $\{I \wedge \neg B\} \equiv i = |s|-1 \wedge_L \overbrace{r = True \rightarrow s[|s|-1] = s[0] + (|s|-1)}^{\equiv Q_r}$

~~como~~ por lo que se demuestra que $\{I \wedge \neg B\} \Longrightarrow Q_r$ ✓

c) $\{I \wedge B\}\ S\ \{I\} \Longleftrightarrow \{I \wedge B\} \Longrightarrow WP(S, I)$

$WP(S, I) \equiv WP(S_1, S_2, I) \underset{ax3}{\equiv} WP(S_1, \overbrace{WP(S_2, I)}^{E})$ ✓

$E \equiv WP(i: i+1, I) \overset{ax1}{\equiv} def(i+1) \wedge_L I^{i}_{i+1} \equiv True \wedge_L I^{i}_{i+1}$ ✓

$\equiv 0 \leq i+1 \leq |s|-1 \wedge_L r = True \rightarrow s[i+1] = s[0] + (i+1)$ llamo r' a $r \,\&\&\, (s[i+1] = s[i]+1)$

$\equiv -1 \leq i \leq |s|-2 \wedge_L r = True \rightarrow s[i+1] = s[0] + i + 1$

$WP(S, I) \equiv WP(S_1, E) \equiv WP(r := r', E) \overset{ax1}{\equiv} def(r') \wedge_L E^{r}_{r'}$

$\equiv 0 \leq i+1 < |s| \wedge 0 \leq i < |s| \wedge_L E^{r}_{r'}$

$\hookrightarrow -1 \leq i \leq |s|-2 \wedge_L r' = True \rightarrow s[i+1] = s[0] + i + 1$

juntando: $0 \leq i+1 < |s| \wedge 0 \leq i < |s| \wedge -1 \leq i \leq |s|-2$

me queda $0 \leq i \leq |s|-2$ como predicado mas fuerte

así $WP(S, I) \equiv 0 \leq i \leq |s|-2 \wedge_L r \,\&\&\, (s[i+1] = s[i]+1) = True \rightarrow s[i+1] = s[0] + i + 1$ /

sigo otro ↓

NOTA

$\{I \wedge B\} \equiv 0 \leq i \leq |s|-1 \wedge r = True \rightarrow s[i] = s[0]+i \wedge i < |s|-1$

$\equiv \underbrace{0 \leq i < |s|-1}_{\hookrightarrow \equiv 0 \leq i \leq |s|-2} \wedge r = True \rightarrow s[i] = s[0]+i$

quiero ver que $\{I \wedge B\} \Rightarrow WP(S, I)$.

veamos que $0 \leq i \leq |s|-2 \Rightarrow 0 \leq i \leq |s|-2$ ✓

$wp(r = True \rightarrow s[i] = s[0]+i) \Rightarrow r \&\& (s[i+1] = s[i]+1) \rightarrow s[i+1] = s[0]+i+1$

Como $r = True$ me implica que $s[i] = s[0]+i$ entonces si ocurre también que $s[i+1] = s[i]+1$, entonces $r \&\& (s[i+1] = s[i]+1)$ no a ser True?

y ya efectivamente implica que $s[i+1] = s[0]+i+1$ como se quería probar

$$\begin{cases} s[i] = s[0]+i \\ s[i+1] = s[i]+1 \end{cases} \Rightarrow s[i+1] = \overbrace{s[0]+i}^{s[i]}+1$$

Ok. Podría formalizarse mejor

por lo que $\{I \wedge B\} \Rightarrow WP(S, I)$

y se cumple la Tripla $\{I \wedge B\}\ S\ \{I\}$ ✓

NOTA

3) Leonel Brozinski

HOJA N°

FECHA 4/7/22

```
   if (i < j) then      (B)
     S1[S[i]] := S[j]
S  else
     S2[S[j]] := S[i]
   endif
```

$B \equiv i < j$

$\neg B \equiv i \geq j$

a) $WP(S, Post) \equiv WP(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}, Post)$

$\overset{Ax4}{\equiv} def(B) \wedge_L ((B \wedge WP(S_1, Post)) \vee (\neg B \wedge WP(S_2, Post)))$ ✓

$\equiv \underbrace{def(i<j)}_{True} \wedge_L ((i < j \wedge \underbrace{WP(S_1, Post)}_{E_1}) \vee (i \geq j \wedge \underbrace{WP(S_2, Post)}_{E_2}))$

$E_1 \equiv WP(S := setAt(S, i, S[j]), Post) \equiv def(setAt(S, i, S[j])) \wedge_L Post^{S}_{setAt(\ldots)}$

↳ faltan pasos intermedios def(set At(s, i, S[j]))

$\equiv 0 \leq i < |S| \wedge 0 \leq j < |S| \wedge_L \neg Ordenado(setAt(S, i, S[j]))$ ✓

$E_2 \equiv WP(S := setAt(S, j, S[i]), Post) \equiv def(setAt(S, j, S[i])) \wedge_L Post^{S}_{setAt(S, j, S[i])}$

$\equiv 0 \leq j < |S| \wedge 0 \leq i < |S| \wedge_L \neg Ordenado(setAt(S, j, S[i]))$ ✓

$WP(S, Post) \equiv 0 \leq i < |S| \wedge 0 \leq j < |S| \wedge_L ((i < j \wedge \neg Ordenado(setAt(S, i, S[j]))$

$\vee (i \geq j \wedge \neg Ordenado(setAt(S, j, S[i]))))$ ✓

b) para que el programa sea correcto respecto a la especificacion hay que demostrar que

$Pre \Rightarrow WP(S, Post)$

sigo atras

NOTA

$Pre \equiv 0 \leq i < |s| \wedge 0 \leq t < |s| \wedge i \neq t \wedge ordenado(s)$

$0 \leq i < |s| \wedge 0 \leq t < |s|$ de la Pre $\Longrightarrow 0 \leq i < |s| \wedge 0 \leq t < |s|$ de $WP(s, Post)$ ✓

caso $i < t$:

$i < t \wedge ordenado(s) \Longrightarrow \neg ordenado(setAt(s, i, s[t]))$

esto es correcto porque como s esta ordenado si $i < t$ si reemplazo en s la posicion i por $s[t]$ que es mayor que $s[i]$ entonces s deja de estar ordenada OK. Faltaria formalizar un poco

caso $i > t$:

$i > t \wedge ordenado(s) \Longrightarrow \neg ordenado(setAt(s, t, s[i]))$

ocurre lo mismo que en el caso anterior, solo que ahora el elemento mas chico a la posicion mas adelante por lo que deja de estar ordenada. También es correcta

Aclaración: no tome el caso $i \geq t$ o $i = t$ porque la Pre especifica que $i \neq t$ ✓

NOTA

4)

Leonel Brogiszki

HOJA N°

FECHA 4/7/22

~~Test: h = -4, m = -5, s = 10~~

~~salida esperada: h = -5, m = 55, s = 11~~

Test: h = 7, m = 59, s = 400

salida esperada: h = 8, m = 5, s = 47

salida obtenida: h = 7, m = 65, s = 47

como se ve, no se respeto la forma canonica

Esto pasa porque se ajustan las horas antes de los minutos entonces si la cantidad de segundos despues de ajustarse hace aumentar la cantidad de minutos hasta superar los 60 el código no va a volver a ajustar esos minutos y no se cumple la Post.
El programa debería ajustar primero los ~~minutos~~ segundos y despues los ~~horas~~ minutos para una implementacion correcta

NOTA