ISIS 1105 Diseño y Análisis de algoritmos
Semestre 2013-20 Parcial 2
Octubre 17, 2013
Prof. Rodrigo Cardoso

1 (35 puntos)
Desarrolle un algoritmo para calcular la potencia n-sima de un valor `a:int` de acuerdo con la siguiente documentación (la operación ÷ corresponde a la división entera):

```
[Ctx C: a: int ∧ n: nat

   S;

   {Inv P: y≥0 ∧ z*x^y = a^n }
   {cota t: y }

   do B →      if par.y          →  x,y:= E,y÷2
               [] impar.y        →  z,y:= F,y-1
               fi
   od

   {R: z = a^n }
]
```

1a Explique qué técnica de desarrollo de invariantes puede justificar el planteo de P con respecto a R.

Técnica: Balance de información.

La información procesada da como resultado la parte z del producto $z*x^y$. La no procesada es la parte x^y. Para procesar información, y disminuye, de modo que z debe aumentar para mantener el balance.

[5/5]

1b Desarrolle código GCL para S, B, E y F, de modo que el algoritmo sea correcto. No se pide la verificación de la corrección.

S es la inicialización del ciclo.

S ≅ `x,y,z:= a,n,1`

P ∧ ¬B ≡ R si ¬B ≡ y=0.

B ≡ y≠0

Para determinar E y F se trata de cumplir las partes correspondientes de la corrección de un condicional, i.e.,

{P ∧ B} **if … fi** {P}

Por tanto, deben valer:

(1) $y\geq 0 \wedge z*x^y = a^n \wedge y\neq 0 \Rightarrow$ par.y ∨ impar.y

(2) $\{y\geq 0 \wedge z*x^y = a^n \wedge y\neq 0 \wedge$ par.y$\}$ x,y:= E,y÷2 $\{y\geq 0 \wedge z*x^y = a^n\}$

(3) $\{y\geq 0 \wedge z*x^y = a^n \wedge y\neq 0 \wedge$ impar.y$\}$ z,y:= F,y-1 $\{y\geq 0 \wedge z*x^y = a^n\}$

(1) se cumple siempre porque par.y ∨ impar.y ≡ true.

(2) requiere que:

$y \geq 0 \wedge z*x^y = a^n \wedge y \neq 0 \wedge par.y \Rightarrow y \div 2 \geq 0 \wedge z*E^{y \div 2} = a^n$

y entonces

$z*x^y = z*E^{y \div 2}$

de modo que debe servir:

$E = x*x$

(3) requiere que:

$y \geq 0 \wedge z*x^y = a^n \wedge y \neq 0 \wedge impar.y \Rightarrow y-1 \geq 0 \wedge z*E^{y \div 2} = a^n$

y entonces

$z*x^y = F*x^{y-1}$

de modo que debe servir:

$F = z*x$

En resumen:

```
[Ctx C: a: int ∧ n: nat

   x,y,z:= a,n,1;

   {Inv P: y≥0 ∧ z*x^y = a^n }
   {cota t: y }

   do y≠0 → if par.y    →  x,y:= x*x,y÷2
             [] impar.y  →  z,y:= z*x,y-1
             fi
   od

   {R: z = a^n }
]
```

[15/15]

1c Considerando las asignaciones como operaciones básicas, calcule la complejidad temporal y espacial del algoritmo solución.

***AYUDA**: Imagine a y expresado en notación binaria y explique cómo cambia en cada iteración.*

Si se piensa en y expresado en notación binaria:

- si al entrar a iterar y es par, y pierde un bit en el tamaño de su representación binaria (dividir por 2 es perder el bit de las unidades).
- si al entrar a iterar y es impar, y disminuye en 1, sigue con la misma longitud en el tamaño de su representación binaria, pero se reducia en 1 en la siguiente iteración.

Es decir, si y empieza en n, tendrá una representación binaria de longitud log n. Si los cambios en y son como se menciona, el peor caso corresponde a tener que hacer dos iteraciones para eliminar cada bit. A lo sumo son 2*log n iteraciones

Es decir:

$T(a,n) = \theta(\log n)$

Para la complejidad espacial, solo se usan 3 variables adicionales (x,y,z), de modo que

$S(a,n) = \theta(1)$

[15/15]

2 (30 puntos)

En un semianillo $(E,+,*,0,1)$ se pueden definir potencias y combinaciones lineales de potencias (con coeficientes en el semianillo), de modo que se puede hablar de polinomios de la forma

$$Q.x = a_n*x^n + a_{n-1}*x^{n-1} + \ldots + a_1*x^1 + a_0*x^0$$

donde, si x y los a_k's están en E, el valor Q.x también está. Se quiere desarrollar un algoritmo para, dados los coeficientes a_0, a_1, ..., a_n de un polinomio, y un valor x, se calcule el valor Q.x.

2a Especifique formalmente el problema.

```
[ Ctx C: a[0..n]:E ∧ x:E
  {Pos R: q = Q.x}
]
```

[5/5]

2b Plantee un invariante P y una cota t que respondan a la idea de ejecución, conocida como *método de Horner*. Para explicarla se puede dar un ejemplo de cómo calcular un polinomio dado en un valor específico.

Ejemplo: Supóngase que $Q.x = 7*x^3 + x - 5$, *y se quiere calcular* Q.2. *Entonces el polinomio se puede escribir como*

$$Q.x = 7*x^3 + 0*x^2 + 1*x^1 + (-5)*x^0$$

y se puede proceder a calcular:

```
7           = 7
7*2 + 0     = 14
14*2 + 1    = 29
29*2 – 5    = 53
```

Inv P: $q = (+i|\ k\le i\le n : a[i]*x^{i-k}) \wedge 0\le k\le n$

Cota t: k

[10/10]

2c Desarrolle una solución de acuerdo con su especificación.

```
[ Ctx C: a[0..n]:E ∧ x:E

   q,k:= a[n],n;

  {Inv  P: q = (+i| k≤i≤n : a[i]*x^(i-k)) ∧ 0≤k≤n}
  {Cota t: k}

   do k≠0    →     k:= k-1;
                   q:= q*x + a[k]
   od

  {Pos R: q = Q.x}
]
```

[10/10]

2d Estime las complejidades temporal y espacial de la solución. Para la complejidad temporal considere como operaciones básicas la suma y la multiplicación en el semianillo. Suponga que las sumas cuestan s y que las multiplicaciones cuestan m.

La cota k empieza en n y llega a ser 0, bajando 1 en cada iteración. Es decir, hay exactamente n iteraciones. En cada una de ellas hay una suma y una multiplicación en el semianillo. Por tanto:

$T(a,x) = \theta(n*(m+s))$

Para la complejidad espacial, solo se requieren las variables q y k, de modo que

$S(a,x) = \theta(1)$

[5/5]

3 (35 puntos)

Sean $x \in$ **int**, $A = \{a_1, a_2, \ldots, a_n\} \subseteq$ **int**, donde todos los a_i's son diferentes. Se quiere determinar si existe $B \subseteq A$, tal que:

$$(+y|\ y \in B : y) = x.$$

mediante un algoritmo de programación dinámica.

Para orientar la definición de un lenguaje que ayude a solucionar el problema, defínase $c(z,k)$ como el valor de verdad de que el entero z pueda ser construible como suma de algunos de los elementos de $A_k = \{a_1, a_2, \ldots, a_k\}$, donde $0 \le k \le n$. También puede ser útil considerar los números σ^+ y σ^-, definidos así:

$\sigma^+ = (+y|\ y \in A \wedge y \ge 0 : y)$
$\sigma^- = (+y|\ y \in A \wedge y < 0 : y)$

Complete el diseño de la construcción de una solución basada en la definición de c, de acuerdo con la metodología de programación dinámica vista en clase. Muestre los diferentes pasos de la metodología, excepto la construcción misma del algoritmo.

Lenguaje

c: **int** × 0..n → **bool**

$c(z,k) \equiv (\exists B|\ B \subseteq A_k : (+y|\ y \in B : y) = z)$

$c(x,n) \equiv ?$

[2/2]

Recurrencia

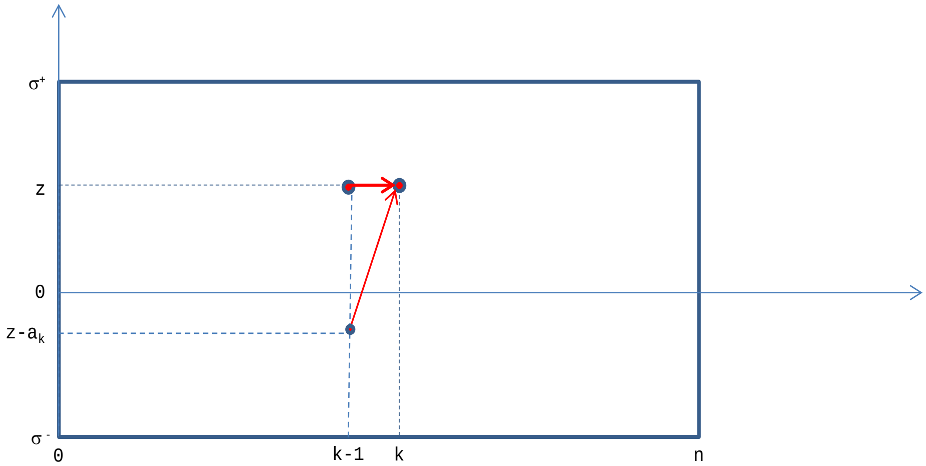
Nótese que si $z>\sigma^+$ o $z<\sigma^-$ es fácil concluir que $c(z,n) \equiv$ false. Es decir, basta explicar cómo se define $c(z,k)$ para $(z,k) \in \sigma^-..\sigma^+ \times 0..n$:

$$\begin{aligned} c(z,k) &\equiv \text{false} && , z>\sigma^+ \text{ o } z<\sigma^- \\ &\equiv z=0 && , \sigma^- \le z \le \sigma^+,\ 0=k \\ &\equiv c(z,k-1) \vee c(z-a_k,k-1) && , \sigma^- \le z \le \sigma^+,\ 0<k \le n \end{aligned}$$

[8/8]

Diagrama de necesidades

El diagrama de necesidades ilustra la recurrencia para c en el caso en el que z está en el intervalo $\sigma^-..\sigma^+$. En el dibujo se muestra el caso en que a_k es positivo o 0. Si a_k es negativo, la dependencia será de algún valor superior a a_k.

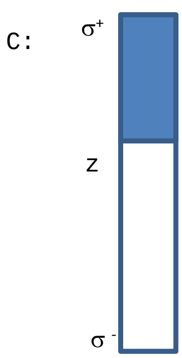
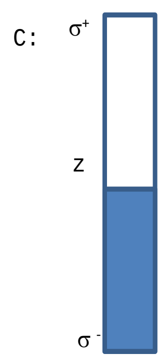


[5/5]

Estructura de datos + invariante

El problema restringido a $\sigma^-..\sigma^+$ es análogo al problema del morral. Se puede llevar a cabo el cálculo con una columna `C[`$\sigma^-..\sigma^+$`]` que se llenará de arriba abajo, cuando $a_k \geq 0$ y de abajo arriba si $a_k < 0$.

El invariante se describe así:

Inv: $1 \leq k \leq n \wedge ((a_k \geq 0 \wedge$  $) \vee (a_k < 0 \wedge$  $))$

y en la parte sombreada `C[z] = c(z,k)`, mientras que en la parte blanca `C[z] = c(z,k-1)`.

[10/10]

Complejidades

Temporal: se debe calcular la matriz $c(\sigma^-..\sigma^+ \times 0..n)$. Cada elemento demora $\theta(1)$:

$T(x,n) = \theta((\sigma^+ - \sigma^- + 1) * n)$

[5/5]

Espacial: se usa la columna auxiliar:

$S(x,n) = \theta(\sigma^+ - \sigma^- + 1)$

[5/5]