ISIS 1105 Diseño de Algoritmos
Semestre 2015-1
Prof. Rodrigo Cardoso
Tarea 2
Para entregar por Sicua+, antes de Abril 6, 10:00

1 (25/100) Subarreglo de suma máxima

Se quiere construir un programa que reciba un arreglo de números enteros y encuentre la suma más grande de un subarreglo de acuerdo con el esquema indicado. La siguiente notación sirve para entender lo que se quiere hacer:

`sumax.i` ≈ "suma máxima entre subarreglos en `b[0..i-1]`", `0≤i≤n`.
`sumaxf.i` ≈ "suma máxima entre subarreglos en `b[0..i-1]`, que terminan en `b[i-1]`", `0≤i≤n`.

```
[Ctx C: b[0..n-1]:nat

 INIC;

 {Inv P : 0≤i≤n ∧ p = sumax.i ∧ q = sumaxf.i)}
 {Cota t: n-i}

 do … od

 {R: p = sumax.n }
 ]
```

1a Explique qué técnica pudo haberse utilizado para proponer el invariante `P`.

1b Desarrolle un algoritmo que satisfaga la especificación indicada (operación básica: suma).

1c Estime las complejidades temporal y espacial de su solución. Explique sus respuestas.

2 (25/100) Orden lexicográfico

Sea `A` un alfabeto y `A*`, el conjunto de palabras construidas con letras de `A`. Sobre `A` se supone entendido (y disponible en el lenguaje de programación) un orden `.<.`, determinado por la secuencia en que se nombran sus elementos. Por ejemplo, si `A` es ASCII, `A*` son palabras con caracteres ASCII en minúsculas, y se entiende –por ejemplo- que `a<b<c…<z`.
Considere el problema de, dadas dos palabras `a,b: A*`, decidir si $a <_{lex} b$, donde $.<_{lex}.$ es el orden lexicográfico correspondiente a `.<.` .

Para palabras en `A*`, use las siguientes notaciones:

ε	: palabra *vacía*
`|x|`	: número de letras de `x` (*longitud* de `x`)
`x[i]`	: `i`-sima letra de `x`, con `1≤i≤|x|`
`esvac.x`	: `x` es una palabra vacía
`equals(x,y)`	: la palabra `x` es igual a la palabra `y` (i.e., tienen las mismas letras)
`x[p,q]`	: la subpalabra de `x` con la letras desde la posición `p` hasta la posición `q` (incluidas). (se entiende que `1≤p≤q≤|x|`).

2a Especifique el problema (Contexto, Pre-, Poscondición).

2b Proponga un invariante `P` y una cota `t` para contribuir a desarrollar el programa solución con un ciclo.

2c Escriba código que satisfaga lo anotado en **2a** y **2b**.

2d Calcule la complejidad temporal de su solución (operación básica: comparación de letras). Explique su respuesta.

2e Estime la complejidad espacial de su solución.

3 **(25/100) Búsqueda lexicográfica**

Con la notación de **2**, suponga que se tienen un arreglo `pal[0..n-1]:A*`, ordenado lexicográficamente en orden ascendente y una palabra `x:A*`, tal que `x∈pal`.
Desarrolle un algoritmo para encontrar un `i`, `0≤i<n`, tal que `equals(x,pal[i])`.
Puede suponer conocida una función `menlex(x,y)` que decide, para palabras `x,y:A*`, si $x <_{lex} y$.

3a Especifique el problema (Contexto, Pre-, Poscondición).

3b Proponga un invariante `P` y una cota `t` que sirva para resolver el problema.

3c Escriba código que satisfaga lo anotado en **3a** y **3b**.

3d Calcule la complejidad temporal de su solución (operación básica: comparación de letras). Explique su respuesta.

3e Estime la complejidad espacial de su solución.

3 **(25/100) Embaldosamientos**

Se desea pavimentar un camino rectangular de dimensiones `1xN` con losas de dimensiones `1xk`, para `k=1,2,…,M`. Se quiere determinar de cuántas maneras puede llevarse a cabo la pavimentación. Diseñe un algoritmo de programación dinámica que resuelva el problema.

3a Construya su solución de acuerdo con la “receta para programación dinámica” que se vio en clase (lenguaje, recurrencia, …) para resolver el problema.

3b Estime complejidades temporal (operación básica: asignación) y espacial de su solución para 3a.

4 **(25/100) 3-Nim**

Suponga 3 montones con p_0,q_0,r_0 fichas, respectivamente, $p_0 \neq q_0$, $q_0 \neq r_0$, $r_0 \neq p_0$, y dos jugadores, `A` y `B`. Los jugadores alternan turnos para quitar, de uno cualquiera de los montones, cualquier número de fichas. `A` es el primero que juega. Gana quien retira la última ficha.

4a Modele con un grafo el desarrollo del juego.

4b Explique en su modelo cómo se reconoce que un jugador pierde.

4c [Bono] Muestre que A puede jugar de manera que siempre gane.