EL SUBARREGLO ASCENDENTE MÁS LARGO

Un ejercicio de programación dinámica

Dado un arreglo `b[0..n-1]:int` se quiere determinar la longitud de un subarreglo ascendente más largo, entendiendo como subarreglo uno de la forma `b[p..q]` , con $0 \le p \le q < n$.

1 LENGUAJE

`lsaml(k)` ≈ "longitud del subarreglo ascendente más largo en `0..k-1`"

$lsaml: 1..n \to 1..n$

`lsaml(n) = ?`

En principio, solo se define esta notación. Más adelante se ve que es conveniente aumentar el lenguaje así:

`lsamlf(k)` ≈ "longitud del subarreglo ascendente más largo en `0..k-1`, que termina en `b[k-1]`"

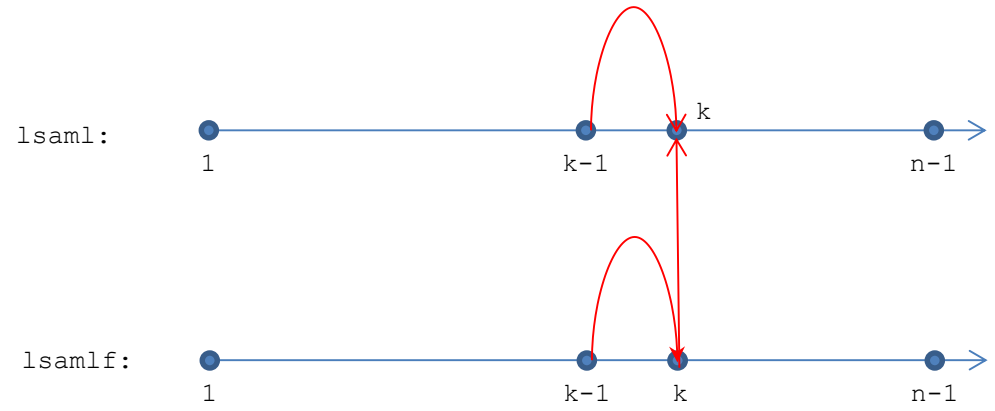
$lsamlf: 1..n \to 1..n$

2 RECURRENCIA

$$
\begin{aligned}
lsaml(1) &= 1 \\
lsaml(k) &= lsaml(k-1) && , 1<k\le n,\ b[k-2]>b[k-1] \\
&= lsaml(k-1)\ \max\ lsamlf(k) && , 1<k\le n,\ b[k-2]\le b[k-1]
\end{aligned}
$$

$$
\begin{aligned}
lsamlf(1) &= 1 \\
lsamlf(k) &= 1 && , 1<k\le n,\ b[k-2]>b[k-1] \\
&= 1 + lamlf(k-1) && , 1<k\le n,\ b[k-2]\le b[k-1]
\end{aligned}
$$

3 DIAGRAMA DE NECESIDADES



4 ESTRUCTURA DE DATOS + INVARIANTE

`a ≈ lsaml(k)`

`c ≈ lsamlf(k)`

$Inv: 1 \le k \le n \ \wedge\ a=lsaml(k) \ \wedge\ c=lsamlf(k)$

5 COMPLEJIDAD

```
T(n)  = n * θ(1)          // n iteraciones de costo θ(1)
      = θ(n)

S(n)  = θ(1)              // 2 variables
```

6 SOLUCIÓN

```
[Ctx : b[0..n-1]:int ∧ n>0

k,a,c:= 1,1,1;

{Inv P: 1≤k≤n  ∧  a=lsaml(k) ∧  c=lsamlf(k)}
{Cota t: n-1-k}

do k≠n      →    if    b[k-2]≤b[k-1]     then  c:= c+1
                                           else  c:= 1
                 fi;
                 if    b[k-2]≤b[k-1]     then  a:= a max c fi;
                 k:= k+1
od

 {Pos R: a = lsaml(n)}
]
```

7 CÓMO ALCANZAR UN ÓPTIMO

Como regla general: se deben recordar las decisiones. En este caso, se puede anotar cuándo se encontró que `a` debía crecer (el extremo derecho). Esto, con el conocimiento de `a` mismo, permite determinar un subarreglo ascendente de tamaño maximal.

Una solución:

```
[Ctx : b[0..n-1]:int ∧ n>0

k,a,c:= 1,1,1;
d:= 0;

{Inv P: 1≤k≤n  ∧  a=lsaml(k) ∧  c=lsamlf(k) ∧ ascendente(b[d-a+1..d])}
{Cota t: n-1-k}

do k≠n      →    if    b[k-2]≤b[k-1]     then  c:= c+1
                                           else  c:= 1
                 fi;
                 if    b[k-2]≤b[k-1]     then  if a<c  then a,d:= c,k-1 fi fi;
                 k:= k+1
od

 {Pos R: a = lsaml(n) ∧  ascendente(b[d-a+1..d])}
]
```