Estructura de datos y algoritmos (GIS)
Ideas y soluciones de algunos ejercicios del tema 4
Curso 2013-2014. Grupo A.

1. Ejercicio 20.

   Derivar un algoritmo de coste lineal que satisfaga la siguiente especificación

   $P \equiv \{N \geq 2\}$
   fun max-resta (int A[n]) return int r
   $Q \equiv \{r = (max\ \ p, q : 0 \leq p < q < N : A[p] - A[q]\}$

   Derivación del algoritmo:

   - Debilitar la postcondicion: $Q \equiv \{r = (max\ \ p, q : 0 \leq p < q < k : A[p] - A[q] \wedge k == N\}$
   - Obtener el invariante y la condición de parada del bucle:
     - $I \equiv r = (max\ \ p, q : 0 \leq p < q < k : A[p] - A[q] \wedge 0 \leq k \leq N\}$
     - $\neg B \equiv (k == N)$ por lo tanto $B \equiv (k \neq N)$
   - Obtener las instrucciones de inicializacinón. Para que el invariante se cumpla al comenzar el bucle inicializamos $k = 2$, de esta forma el rango del operador max no es vacío (el operador max con rango vacío está indefinido).

     Si $k = 2$ las variables ligadas p y q tienen un único valor posible $p = 0$ y $q = 1$ lo que da un valor para r de $r = A[0] - A[1]$
   - función cota: $N - k$
   - Instrucción de avance: $k = k + 1$. De esta forma la función cota decrece.
   - Instrucciones para restablecer el invariante:

     Estudiamos el predicado $R \equiv r = (max\ \ p, q : 0 \leq p < q < k + 1 : A[p] - A[q] \wedge 0 \leq k+1 \leq N\}$ obtenido como pmd del invariante con la instrucción de avance. (No tenemos en cuenta la parte $0 \leq k + 1 \leq N$ porque no aportará nada al estudio)

     $r = (max\ \ p, q : 0 \leq p < q < k + 1 : A[p] - A[q]) \equiv$
     $r = max((max\ \ p, q : 0 \leq p < q < k : A[p] - A[q]), (max\ \ j : 0 \leq j < k : A[j] - A[k]))$

     Nos hace falta poder calcular $(max\ \ j : 0 \leq j < k : A[j] - A[k])$ en tiempo constante. Se observa que el máximo que queremos calcular se obtiene cuando $A[j]$ es el valor máximo de la parte del vector ya recorrida. Es decir, si $m = max\ \ j : 0 \leq j < k : A[j]$ entonces $(max\ \ j : 0 \leq j < k : A[j] - A[k]) = m - A[k]$

     Por lo tanto, para obtener la instrucción para restablecer el invariante necesitamos conocer el valor del máximo del vector. Utilizamos una variable m para almacenar este valor y modificamos el invariante para indicar el valor de m en el bucle.

     $I \equiv r = (max\ \ p, q : 0 \leq p < q < k : A[p] - A[q] \wedge 0 \leq k \leq N \wedge$
     $m = max\ \ j : 0 \leq j < k : A[j]\}$

     Inicialización de la variable m. Como k está inicializado a 2, hay dos valores posibles para el máximo, $A[0]$ y $A[1]$. La inicialización será $m = max(A[0], A[1])$.

     La instrucción del bucle para recuperar el valor de m es: $m = max(m, A[k])$

     Y la instrucción para recuperar el invariante: $r = max(r, m - A[k])$
   - El código del algoritmo es:

```
k = 2;
r = A[0]-A[1];
m = max(A[0],A[1]);
while (k != N) {
       r = max(r,m-A[k]);
       m = max(m,A[k]);
       k = k +1;
}
```

2. Ejercicio 21. Derivar un algoritmo de coste lineal que satisfaga la siguiente especificación

   $P \equiv \{N \geq 0\}$
   fun credit-seg-max (int A[N]) return int r
   $Q \equiv \{r = (max\ \ p, q : 0 \leq p \leq q \leq N : credito(p, q))\}$
   donde $credito(p, q) = (\sharp i : p \leq i < q : A[i] > 0) - (\sharp i : p \leq i < q : A[i] < 0)$

   Solo algunas ideas:

   - El invariante se obtiene debilitando la postcondición $I \equiv \{r = (max\ \ p, q : 0 \leq p \leq q \leq k : credito(p, q)) \wedge \ldots\}$

   - La instrucción de avance será $k = k + 1$

   - Para restablecer el invariante calculamos el predicado R como *pmd* del invariante con la instrucción de avance
     $R \equiv r = (max\ \ p, q : 0 \leq p \leq q \leq k + 1 : credito(p, q)) \ldots$
     Se observa que
     $r = (max\ \ p, q : 0 \leq p \leq q \leq k + 1 : credito(p, q)) \ldots \equiv$
     $r = max(max\ \ p, q : 0 \leq p \leq q \leq k : credito(p, q)), max\ \ j : 0 \leq j \leq k : credito(j, k + 1)$
     Para poder calcular $max\ \ j : 0 \leq j \leq k : credito(j, k + 1)$ en tiempo constante debemos llevar en una variable $c = max\ \ j : 0 \leq j \leq k : credito(j, k + 1)$. El valor de c se restablece en cada vuelta del bucle viendo si $c + A[k] < 0$, ya que si es negativo el credito acumulado no aporta nada y sera mejor volver a empezar a contar.