Ingeniería Informática - Depto. de Informática e Ingeniería de Sistemas

Examen de Programación 2 - 11 de septiembre de 2015

Problema 1º (4 puntos)

En este problema hay que probar la corrección del codigo de la función ***mcm(a,b)*** que devuelve el mínmo común múltiplo de dos positivos, `a` y `b`. Para ello se pide:

1. Escribir un predicado invariante asociado al bucle suficientemente fuerte como para sustentar las demostraciones que se piden a continuación.
2. Demostrar la corrección del código que precede al bucle, del código que le sigue y del código que se ejecuta cada vez que se itera el bucle.
3. Demostrar la terminación del bucle.

```
/*
 * Predicado funcional utilizado para facilitar la especificación de la función mcm(a,b):
 *       esMultiplo(n,p) = (EX alfa EN [1,n].n = alfa * p)
 */

/*
 * Pre: a > 0 AND b > 0 AND a >= b
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *          (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm(const int a, const int b) {
    int candidato = a;
    while (candidato % b != 0) {
        candidato = candidato + a;
    }
    return candidato;
}
```

Problema 2º (2 puntos)

El coste en tiempo de ejecutar la función ***mcm(a,b)*** depende de los valores de los parámetros ***a*** y ***b***. Para un valor determinado de a, el coste en tiempo de ejecutar la función depende del valor de ***b***.

Para un valor determinado de a, que sea 'suficientemente grande'se pide:

1. Explicar claramente en qué circunstancias se presentan los casos mejores y los casos peores, en cuanto al coste en tiempo, al ejecutar la función ***mcm(a,b)***.
2. Deducir la función de coste en tiempo, $t_{mcm(a,b)}(b)$, en los casos extremos, casos mejores y casos peores, asumiendo que el coste de ejecutar cada una de sus líneas de código etiquetadas por $L1$, $L2$, $L3$ y $L4$ es igual a t_1, t_2, t_3 y t_4, respectivamente. Se valorará la claridad y el detalle con que se presenten y justifiquen los cálculos realizados.
3. Caracterizar asíntóticamente la función de coste, en tiempo, en los casos extremos, es decir, los casos mejores y peores, mediante la notación $\boldsymbol{O}(t_{mcm(a,b)}(b))$.

```
/*
 * Predicado funcional utilizado para facilitar la especificación de la función mcm(a,b):
 *      esMultiplo(n,p) = (EX alfa EN [1,n].n = alfa * p)
 */

/*
 * Pre: a > 0 AND b > 0 AND a >= b
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *         (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm(const int a, const int b) {
    int candidato = a;                          // L1
    while (candidato % b != 0) {                // L2
        candidato = candidato + a;              // L3
    }
    return candidato;                           // L4
}
```

Problema 3º (2 puntos)

Reescribir, sin bucles, la función ***mcm(a,b)*** aplicando un método algoritmo idéntico al anteriormente mostrado. Se valorará esencialmente, la corrección del diseño para lo cual es fundamental una adecuada especificación de las funciones auxiliares. Dicha especificación será formal. [1.5 puntos]

```
/*
 * Pre: a > 0 AND b > 0 AND a >= b
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *         (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm(const int a, const int b);
```

Explicar qué tipo de técnica de inmersión se ha aplicado en el diseño sin bucles anterior.

Problema 4º (2 puntos)

Supongamos que se ejecuta la función `separar(v,n,p)` partiendo de un estado inicial que satisfaga su precondición. Se pide escribir los predicados más fuertes que se satifagan al alcanzar la ejecución los puntos del código señalados por ***P1***, ***P2***, ***...***, ***P7*** y ***P8***. Estos ocho predicados han de ser predicados formales.

```
/*
 * Predicado funcional utilizado para facilitar la especificación de la función seaparar(v,n,p):
 *     PERMUTACION(v,w,n) = (PT alfa EN [0,n-1].(NUM beta EN [0,n-1].v[beta]=v[alfa])
 *                                                    =
 *                                                 (NUM beta EN [0,n-1].w[beta]=v[alfa]))
 */

/*
 * Pre:  x = A AND y = B
 * Post: x = B AND y = A
 */
void permuta (int& x, int& y){
    int aux = x;
    x = y;  y = aux;
}

/*
 * Pre:  n > 0 AND v = V0
 * Post: PERMUTACION(v,V0,n) AND p >= 0 AND p <= n AND
 *       (PT alfa IN [0,p-1].v[alfa]%2=0) AND (PT alfa IN [p,n-1].v[alfa]%2=1)
 */
void separar(int v[], const int n, int& p) {
    // n > 0 AND v = V0
    int i = 0, j = n- 1;
    // P1
    while (i<=j) {
        // P2
        if (v[i]%2==0) {
            // P3
            i=i+1;
        }
        else if (v[j]%2==1) {
            // P4
            j=j-1;
        }
        else {
            // P5
            permuta(v[i],v[j]);
            // P6
            i = i + 1;  j = j - 1;
        }
        // P7
    }
    // P8
    p = i;
    // PERMUTACION(v,V0,n) AND p >= 0 AND p <= n AND
    // (PT alfa IN [0,p-1].v[alfa]%2=0) AND (PT alfa IN [p,n-1].v[alfa]%2=1)
}
```

Una solución del problema 1º

Invariante del bucle y pruebas de la correción del código que precede al bucle, del que le sigue y del código a iterar.

```
/*
 * Predicado funcional utilizado para facilitar la especificación de la función mcm(a,b):
 *       esMultiplo(n,p) = (EX alfa EN [1,n].n = alfa * p)
 */

/*
 * Pre: a > 0 AND b > 0 AND a >= b
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *          (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm (const int a, const int b) {
     /* a > 0 AND b > 0 AND a >= b
      * =>
      * a > 0 AND b > 0 AND a >= b AND esMultiplo(a,a) AND
      * (PT alfa EN [2,a-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b)) */
    int candidato = a;
    /* INV: a > 0 AND b > 0 AND a >= b AND esMultiplo(candidato,a) AND candidato <= a * b AND
     *       (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b)) */
    while (candidato % b != 0) {
        /* NOT esMultiplo(candidato,b) a > 0 AND b > 0 AND a >= b AND
         * esMultiplo(candidato,a) AND candidato <= a * b AND
         * (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
         * =>
         * esMultiplo(candidato+a,a) AND candidato + a <= a * b AND
         * (PT alfa EN [2,candidato+a-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b)) */
        candidato = candidato + a;
        /* INV: a > 0 AND b > 0 AND a >= b AND esMultiplo(candidato,a) AND candidato <= a * b AND
         *       (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b)) */
    }
    /* a > 0 AND b > 0 AND a >= b AND esMultiplo(candidato,b) AND esMultiplo(candidato,a) AND
     * (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
     * =>
     * candidato = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
     * (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))*/
    return candidato;
    /* mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
     * (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b)) */
}
```

Prueba de la terminación del bucle.

```
/*
 * Pre: a > 0 AND b > 0 AND a >= b
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *         (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm (const int a, const int b) {
    int candidato = a;
    while (candidato % b != 0) {
        /* f_cota = a * b - candidato */
        /* candidato = X AND f_cota( inicio ) = a * b - X */

        candidato = candidato + a;

        /* INV: a > 0 AND b > 0 AND a >= b AND esMultiplo(candidato,a) AND candidato <= a * b AND
         *         (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b)) */

        /* candidato = X + a AND f_cota( fin ) = a * b - X - a */

        /* El bucle termina ya que se satisfacen dos condiciones:
         * 1. f_cota decrece en cada iteración: f_cota( inicio ) > f_cota( fin )
         *    ya que a * b - X > a * b - X - a ya que a > 0 y b > 0
         * 2. f_cota = a * b - candidato >= 0 ya que candidato <= a * b
         */
    }
    return candidato;
}
```

Una solución del problema 2º

Los casos mejores y peores, en cuanto a coste, para un determinado valor del parámetro a, suficientemente grand,e se dan en los siguientes casos:

- **Casos mejores**. Se presentan cuando el valor de a es múltiplo de b, es decir, cuando se satisface el predicado $esMultiplo(a, b)$:

  $t_{mcm(a,b)}(b) = t_1 + t_2 + t_4$

  La caracterización asintotica de la función de coste corresponde al orden de una función constante:

  $\boldsymbol{O}(t_{mcm(a,b)}(b)) = \boldsymbol{O}(t_1 + t_2 + t_4) = \boldsymbol{O}(1)$

- **Casos peores**. Se presentan cuando los valores de a y de b son primos entre sí. En tales casos el bucle se ejecuta $b - 1$ veces y, por lo tanto, la función de coste es:

  $t_{mcm(a,b)}(b) = t_1 + t_2 + (\Sigma\alpha \in [1, b-1].t_2 + t_3) + t_4 = t_1 + t_2 + (b-1) \times (t_2 + t_3) + t_4$

  Operando se obtiene:

  $\boldsymbol{O}(t_{mcm(a,b)}(b)) = (t_2 + t_3) \times b + t_1 - t_3 + t_4$

  La caracterización asintotica de la función de coste corresponde al orden de la función b:

  $\boldsymbol{O}t_{mcm(a,b)}(b)) = \boldsymbol{O}((t_2 + t_3) \times b + t_1 - t_3 + t_4) = \boldsymbol{O}(b)$

Una solución del problema 3º

Diseño, sin bucles, de la función `mcm(a,b)`, aplicando una inmersión mediante refuerzo de la precondición.

```
/*
 * Pre: a > 0 AND b > 0 AND a >= b AND esMultiplo(candidato,a) AND candidato <= a * b AND
 *       (PT alfa EN [2,candidato-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *       (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm (const int a, const int b, const int candidato) {
    if (candidato % b ==0) {
        return candidato;
    }
    return mcm(a, b, candidato + a);
}

/*
 * Pre: a > 0 AND b > 0 AND a >= b
 * Post: mcm(a,b) = M AND M > 0 AND esMultiplo(M,a) AND esMultiplo(M,b) AND
 *       (PT alfa EN [2,M-1].NOT esMultiplo(alfa,a) OR NOT esMultiplo(alfa,b))
 */
int mcm (const int a, const int b) {
    return mcmSB(a, b, a);
}
```

Una solución del problema 4º

```
/*
 * Predicado funcional utilizado para facilitar la especificación de la función seaparar(v,n,p):
 *    PERMUTACION(v,w,n) = (PT alfa EN [0,n-1].(NUM beta EN [0,n-1].v[beta]=v[alfa])
 *                                                =
 *                                              (NUM beta EN [0,n-1].w[beta]=v[alfa]))
 */

/*
 * Pre:  x = A AND y = B
 * Post: x = B AND y = A
 */
void permuta (int& x, int& y){
    int aux = x;
    x = y;  y = aux;
}
```

```
/*
 * Pre:   n > 0 AND v = V0
 * Post:  PERMUTACION(v,V0,n) AND p >= 0 AND p <= n AND
 *        (PT alfa IN [0,p-1].v[alfa]%2=0) AND (PT alfa IN [p,n-1].v[alfa]%2=1)
 */
void separar(int v[], const int n, int& p) {
    // n > 0 AND v = V0
    int i = 0, j = n- 1;
    // n > 0 AND PERMUTACION(v,V0,n) AND i = 0 AND j = n-1 AND
    // (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
    // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
    while (i<=j) {
        // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND i<= j AND
        // (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
        // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
        if (v[i]%2==0) {
            // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND i<= j AND
            // v[i]%2=0 AND (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
            // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
            i=i+1;
        }
        else if (v[j]%2==1) {
            // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND i<= j AND
            // v[i]%2!=0 AND v[j]%2=1 AND (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
            // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
            j=j-1;
        }
        else {
            // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND i<= j AND
            // v[i]%2!=0 AND v[j]%2!=1 AND (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
            // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
            permuta(v[i],v[j]);
            // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND i<= j AND
            // v[i]%2!=0 AND v[j]%2!=1 AND (PT alfa IN [i-1,j+1].v[alfa]=V0[alfa]) AND
            // (PT alfa IN [0,i].v[alfa]%2=0) AND (PT alfa IN [j,n-1].v[alfa]%2=1)
            i = i + 1;  j = j - 1;
        }
        // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND
        // (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
        // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
    }
    // n > 0 AND PERMUTACION(v,V0,n) AND i >= 0 AND j <= n-1 AND
    // (PT alfa IN [i,j].v[alfa]=V0[alfa]) AND
    // (PT alfa IN [0,i-1].v[alfa]%2=0) AND (PT alfa IN [j+1,n-1].v[alfa]%2=1)
    p = i;
    // PERMUTACION(v,V0,n) AND p >= 0 AND p <= n AND
    // (PT alfa IN [0,p-1].v[alfa]%2=0) AND (PT alfa IN [p,n-1].v[alfa]%2=1)
}
```