Programación2

Análisis del coste de algoritmos recursivos

Problemas 09

Problema 1. Caracterización asintótica del coste

Caracterizar asintóticamente el coste en tiempo, $O(t(n))$, y en memoria, $O(memoria(n))$, de la invocación **sumar**(v,0,n-1) en función del valor de **n**.

```
/*
 * Pre: i >= 0  AND  i <= j  AND  j < #v
 * Post: sumar(v,i,j) = (SIGMA alfa EN [i,j]. v[alfa])
 */
double sumar (const double v[], const int i, const int j) {
   if (i == j) {
      return v[i];
   }
   else {
      return sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j);
   }
}
```

Vamos a caracterizar el coste de **sumar**(v,0,n-1)

```
/*
 * Pre: i >= 0  AND  i <= j  AND  j < #v
 * Post: sumar(v,i,j) = (SIGMA alfa EN [i,j]. v[alfa])
 */
double sumar (const double v[], const int i, const int j) {
   if (i == j) {                                          // a
      return v[i];                                        // b
   }
   else {
      return sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j);  // c
   }
}
```

Denominaremos: $t_{sumar(v,0,n-1)}(n) = t(n)$

Vamos a caracterizar el coste, en tiempo, de **sumar(v,0,n-1)**.

```
// Pre: i >= 0  AND  i <= j  AND  j < #v
// Post: sumar(v,i,j) = (SIGMA alfa EN [i,j]. v[alfa])
double sumar (double v[], int i, int j) {
   if (i == j) {                                          // a
      return v[i];                                        // b
   }
   else {
      return sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j);   // c
   }
}
```

Denominaremos: $t_{sumar(v,0,n-1)}(n) = t(n)$

$t(n) = t_{inv} + t_a + t_c + t(n/2) + t(n/2)$ ← Recurrencia a resolver

Recurrencia a resolver: $t(n) = t_{inv} + t_a + t_c + t(n/2) + t(n/2)$

Recurrencia a resolver:

$$t(n) - 2.t(n/2) = t_{inv} + t_a + t_c$$

Cambio de variable: $m = \log_2 n \rightarrow n = 2^m \wedge \log_2(n/2) = m-1$

$$t(m) - 2.t(m-1) = t_{inv} + t_a + t_c = 1^m.(t_a + t_c)$$

Ecuación característica:

$$(x-2).(x-1) = 0$$

Raíces de la ecuación característica:

$$x = 2 \text{ y } x = 1$$

PARA CUALQUIER CASO

Solución de la recurrencia en función de **m**:

$$t(m) = c_1.2^m + c_2.1^m = c_1.2^m + c_2$$

Solución de la recurrencia en función de **n**:

$$t(n) = c_1.n + c_2$$

Caracterización asintótica de la función de coste:

$$\boldsymbol{O}(t(n)) = \boldsymbol{O}(c_1.n + c_2) = \boldsymbol{O}(n)$$

$$\boldsymbol{O}(t(n)) = \boldsymbol{O}(n)$$

Caracterización asintótica de la función de coste de memoria

Caracterizar asintóticamente el coste, en uso de memoria, de la invocación **sumar**(v,0,n-1)

```
// Pre: i >= 0  AND  i <= j  AND  j < #v
// Post: sumar(v,i,j) = (SIGMA alfa EN [i,j]. v[alfa])
double sumar (const double v[], const int i, const int j) {
   if (i == j) {
      return v[i];
   }
   else {
      return sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j);
   }
}
```

Cada invocación hace uso de las variables **v** (referencia cuyo coste es m_{ref}), **i** y **j** de tipo **int** (su coste es m_{int}):

memoria(n) = …

Caracterización asintótica de la función de coste de memoria

Caracterizar asintóticamente el coste, en uso de memoria, de la invocación **sumar(v,0,n-1)**

```
// Pre: i >= 0  AND  i <= j  AND  j < #v
// Post: sumar(v,i,j) = (SIGMA alfa EN [i,j]. v[alfa])
double sumar (double v[], int i, int j) {
   if (i == j) { return v[i]; }
   else { return sumar(v,i,(i+j)/2) + sumar(v,(i+j)/2+1,j); }
}
```

Cada invocación hace uso de las variables **v** (referencia cuyo coste es m_{ref}), **i** y **j** de tipo **int** (su coste es m_{int}):

$$memoria(n) = m_{invocación} + m_{ref} + 2.m_{int} + m_{double} + memoria(n/2)$$

sólo la memoria de una de las dos invocaciones

Caracterización asintótica de la función de coste de memoria

Caracterizar asintóticamente el coste, en uso de memoria, de la invocación **`sumar(v,0,n-1)`**

Cada invocación hace uso uno de las variables **`v`** (referencia cuyo coste es m_{ref}), **`i`** y **`j`** de tipo **`int`** (su coste es m_{int}):

$$memoria(n) = m_{invocación} + m_{ref} + 2.m_{int} + m_{double} + memoria(n/2)$$

Es decir:

$$memoria(n) - memoria(n/2) = m_{invocación} + m_{ref} + 2.m_{int} + m_{double}$$

Recurrencia cuya solución es:

$$memoria(n) = c_1.\log_2 n + c_2$$

$$\boldsymbol{O}(memoria(n)) = \boldsymbol{O}(c_1.\log_2 n + c_2) = \boldsymbol{O}(c_1.\log_2 n) = \boldsymbol{O}(\log n)$$

$$\boldsymbol{O}(memoria(n)) = \boldsymbol{O}(\log n)$$

El coste asintótico en uso de memoria de la invocación **`sumar`**`(v,0,n-1)` es logarítmico en **`n`**.

Problema 2. Caracterización asintótica de la función de coste

Caracterizar asintóticamente el coste en tiempo, ***O***(t(n)), de la invocación **buscar**(v,d,0,n-1) en función del valor de **n**.

```
// Pre: desde >= 0  AND  desde <= hasta  AND  hasta < #v
// Post: buscar(v,x,desde,hasta) = INDICE AND
//       ((EX alfa EN [desde,hasta].v[alfa] = x) -> v[INDICE] = x) AND
//       ((PT alfa EN [desde,hasta].v[alfa]!=x) -> INDICE < 0)
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;
      int r = buscar(v, x, desde, medio);
      if (r >= 0) { return r; }
      else { return buscar(v, x, medio + 1, hasta); }
   }
}
```

Caracterizar asintóticamente el coste de **`buscar`**`(v,d,0,n-1)`.

Caso mejor : … ? … **Caso peor**: … ? …

```
// Pre: …   Post: …
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)                                          // a
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;                          // b
      int r = buscar(v, x, desde, medio);                       // c
      if (r >= 0) {                                             // d
         return r;                                              // e
      }
      else {
         return buscar(v, x, medio + 1, hasta);                 // f
      }
   }
}
```

Caracterizar asintóticamente el coste de **`buscar(v,d,0,n-1)`**.

Caso mejor : Siempre se satisface `r >= 0`. **Caso peor**: Nunca se satisface `r >= 0`.

```
// Pre: …   Post: …
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)                                          // a
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;                          // b
      int r = buscar(v, x, desde, medio);                       // c
      if (r >= 0) {                                             // d
         return r;                                              // e
      }
      else {
         return buscar(v, x, medio + 1, hasta);                 // f
      }
   }
}
```

Caracterizar asintóticamente el coste de **`buscar(v,d,0,n-1)`**.

Caso mejor : En cada invocación se satisface la condición `(r >= 0)`

Caso peor: En ninguna invocación se satisface la condición `(r >= 0)`

```
// Pre: …   Post: …
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)                                          // a
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;                          // b
      int r = buscar(v, x, desde, medio);                       // c
      if (r >= 0) {                                             // d
         return r;                                              // e
      }
      else {
         return buscar(v, x, medio + 1, hasta);                 // f
      }
   }
}
```

Caracterizar asintóticamente el coste de **buscar**(v,d,0,n-1).

$$t(n) = t_{inv} + t_a + t_b + t_c + t(n/2) + t_d + t_e$$

```
// Pre: …   Post: …
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)                                          // a
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;                          // b
      int r = buscar(v, x, desde, medio);                       // c
      if (r >= 0) {                                             // d
         return r;                                              // e
      }
      else {
         return buscar(v, x, medio + 1, hasta);                 // f
      }
   }
}
```

CASO MEJOR (en ninguna invocación se ejecuta la línea f)

Recurrencia a resolver: $t(n) = t_{inv} + t_a + t_b + t_c + t(n/2) + t_d + t_e$

Recurrencia a resolver: $t(n) - t(n/2) = t_{inv} + t_a + t_b + t_c + t_d + t_e$

Cambio de variable: $m = \log_2 n \rightarrow n = 2^m \wedge \log_2 (n/2) = m - 1$

$$t(m) - t(m-1) = (t_{inv} + t_a + t_b + t_c + t_d + t_e).1^m$$

Ecuación característica:

$$(x-1)^2 = 0$$

Raíces de la ecuación característica:

$$x = 1 \text{ (doble)}$$

COSTE EN EL CASO MEJOR

Solución de la recurrencia en función de **m**:

$$t(m) = c_1.m.1^m + c_2.1^m = c_1.m + c_2$$

Solución de la recurrencia en función de **n**:

$$t(n) = c_1.\log_2 n + c_2$$

Caracterización asintótica de la función de coste:

$$O(t(n)) = O(c_1.\log_2 n + c_2) = O(\log_2 n) = O(\log n)$$

$$O(t(n)) = O(\log n)$$

Caracterizar asintóticamente el coste de **`buscar(v,d,0,n-1)`**.

Caso mejor : En cada invocación se satisface la condición `(r >= 0)`

Caso peor: En ninguna invocación se satisface la condición `(r >= 0)`

```
// Pre: …   Post: …
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)                                          // a
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) / 2;                          // b
      int r = buscar(v, x, desde, medio);                       // c
      if (r >= 0) {                                             // d
         return r;                                              // e
      }
      else {
         return buscar(v, x, medio + 1, hasta);                 // f
      }
   }
}
```

CASO PEOR (en ninguna invocación se ejecuta la línea e)

Caracterizar asintóticamente el coste de **buscar(v,d,0,n-1)**.

$$t(n) = t_{inv} + t_a + t_b + t_c + t(n/2) + t_d + t_f + t(n/2)$$

```
// Pre: …   Post: …
int buscar (const int v[], const int x, const int desde, const int hasta) {
   if (desde == hasta)                                          // a
      if (v[desde] == x) { return desde; }
      else { return -1; }
   else {
      int medio = (desde + hasta) /2 ;                          // b
      int r = buscar(v, x, desde, medio);                       // c
      if (r >= 0) {                                             // d
         return r;                                              // e
      }
      else {
         return buscar(v, x, medio + 1, hasta);                 // f
      }
   }
}
```

CASO PEOR

Recurrencia a resolver:

$$t(n) = t_{inv} + t_a + t_b + t_c + t(n/2) + t_d + t_f + t(n/2)$$

Recurrencia a resolver:

$$t(n) - 2.t(n/2) = t_{inv} + t_a + t_b + t_c + t_d + t_f$$

Es la misma recurrencia resuelta en el problema 1. Podemos escribir directamente la solución de la recurrencia en función de **n**:

$$t(n) = c_1.n + c_2$$

Caracterización asintótica de la función de coste:

$$O(t(n)) = O(c_1.n + c_2) = O(n)$$

COSTE EN EL CASO PEOR

$$O(t(n)) = O(n)$$

Problema 3. Se pide:

1. Caracterizar asintóticamente el coste en tiempo, $O(t(p))$, de la invocación **gestionar**(p) en función del valor de **p**.
2. Sabiendo que el coste de **gestionar**(10000) es de 12 ms, estimar el coste de **gestionar**(30000)

```
/*
 * Pre: p >= 0
 * Post: ...
 */
void  gestionar (const int p) {
   if (p <= 1) {
      accion1(p);          //  O(t_accion1(n)(n)) = O(n)
   }
   else {
      accion2(p);          //  O(t_accion2(n)(n)) = O(n)
      gestionar(p-2);
   }
}
```

1. Caracterizar asintóticamente el coste , en tiempo, $O(t(p))$, de la invocación **`gestionar(p)`**

```
// Pre: p >= 0
// Post: ...
void  gestionar (const int p) {
   if (p <= 1) {              // a
      accion1(p);
   }
   else {
      accion2(p);             // b
      gestionar(p-2);         // c
   }
}
```

Y se sabe que:

- $O(t_{accion1(n)}(n)) = O(n)$
- $O(t_{accion2(n)}(n)) = O(n)$

1. Caracterizar asintóticamente el coste de la invocación **gestionar(p)**

   El coste de la invocación **accion2(p)** es $t_{acción2(p)} = k_1.p$

   Por lo tanto la recurrencia a resolver, en todos los casos, es:

$$t(p) = t_{inv} + t_a + t_b + k_1.p + t(p-2) + t_c$$

```
// Pre: p >= 0
// Post: ...
void  gestionar (const int p) {
   if (p <= 1) {              // a
      accion1(p);             // O(t_accion1(n)(n)) = O(n)
   }
   else {                     // O(t_accion2(n)(n)) = O(n)
      accion2(p);             // b
      gestionar(p-2);         // c
   }
}
```

Recurrencia a resolver: $t(p) = t_{inv} + t_a + t_b + k_1.p + t(p-2) + t_c$

Recurrencia a resolver:

$$t(p) - t(p-2) = (t_{inv} + t_a + t_b + t_c + k_1.p).1^p$$

Ecuación característica:

$$(x^2-1)(x-1)^2 = 0$$

Raíces de la ecuación característica:

$$x = 1 \text{ (triple)} \quad y \quad x = -1$$

Solución de la recurrencia en función de **p**:

$$t(p) = c_1.p^2.1^p + c_2.p.1^p + c_3.1^p + c_4.(-1)^p$$
$$= c_1.p^2 + c_2.p + c_3 + c_4.(-1)^p$$

Caracterización asintótica de la función de coste:

$$O(t(p)) = O(c_1.p^2 + c_2.p + c_3 + c_4.(-1)^p)$$
$$= O(c_1.p^2) = O(p^2)$$

$$O(t(p)) = O(p^2)$$

COSTE EN CUALQUIERA DE LOS CASOS QUE PUEDAN PRESENTARSE

2. Sabiendo que el coste de **gestionar**(10000) es de 12 ms, debemos estimar el coste de **gestionar**(30000)

Caracterización asintótica de la función de coste:

$$O(t_{gestionar(p)}(p)) = O(p^2) \quad \rightarrow \quad t_{gestionar(p)}(p) = k.p^2$$

Se conoce el coste de la invocación **gestionar(10000)**:

$$t_{gestionar(10000)}(10000) = k.p^2 = k.10000^2 = 12 \text{ ms}$$

Podemos estimar el coste de la invocación **gestionar(30000)**:

$$t_{gestionar(30000)}(30000) = k.p^2 = k.30000^2 = 9.k.10000^2$$
$$t_{gestionar(30000)}(30000) = 9 \times 12 \text{ ms} = 108 \text{ ms}$$

$$t_{gestionar(30000)}(30000) = 108 \text{ ms} = 0.108 \text{ s}$$