Cuaderno de problemas
Fundamentos de algorítmia.

Cálculo de la complejidad de algoritmos recursivos

Prof. Isabel Pita

27 de octubre de 2021

Índice

1. Coste de una implementación recursiva.

Para analizar la complejidad de un algoritmo recursivo debemos analizar la complejidad de lo que se hace en cada llamada recursiva y estimar el número de llamadas recursivas que se realizan.

Para calcular el coste de una función recursiva se debe definir la ecuación de recurrencia del algoritmo:

1.1. Una llamada recursiva.

1.1.1. Disminución del tamaño del problema por sustracción y coste constante de las funciones sucesor y de combinación.

- Ejemplo. Función factorial.

```
using lli = long long int;
lli factorial ( int n ){
  if (n == 0) return 1;
  else return n * factorial(n-1);   // (n > 0)
}
```

- Ecuación de recurrencia, donde n representa el valor de entrada a la funcón `n`:

$$T(n) = \begin{cases} c_0 & \text{si } n = 0 \\ T(n-1) + c_1 & \text{si } n > 0 \end{cases}$$

  Donde:

  - c_0 es una constante que representa el coste del caso base ($n == 0$).
  - c_1 es una constante que representa el coste de las instrucciones que acompañan a la llamada recursiva (un producto) y el coste de calcular los argumentos de la llamada recursiva (una resta).
  - $T(n-1)$ representa el coste de hacer la llamada recursiva con un argumento una unidad menor.

- Para obtener el orden de complejidad debemos *desplegar* la recurrencia. Para ello:

  1. Sustituimos el coste de $T(n-i)$ por su valor para algunos valores de i ($i = 0, 1, 2, 3$)).
     $T(n) = T(n-1) + c_1 = T(n-2) + c_1 + c_1 = T(n-3) + c_1 + c_1 + c_1 = \ldots$
  2. Obtenemos un término general cuando hemos desplegado k veces:
     $$T(n-k) + k * c_1$$
  3. Calculamos cuando el despliegue llega a su término, es decir cuando llegamos a $T(0)$. Para obtener $T(0)$ en el término general debemos tener $n - k = 0$ y por lo tanto $k = n$
  4. El término general cuando terminamos de desplegar la recurrencia, es:
     $$T(0) + n * c_1$$
  5. El desplegado completo es:
     $T(n) = T(n-1) + c_1 = T(n-2) + c_1 + c_1 = T(n-3) + c_1 + c_1 + c_1 = \ldots = T(n-k) + k * c_1 = \ldots = T(0) + n * c_1 \in \mathcal{O}(n)$
  6. Por lo tanto el coste está en el orden $\mathcal{O}(n)$, siendo n el valor de entrada al algoritmo.

1.1.2. Disminución del tamaño del problema por sustracción y coste lineal de las funciones sucesor o de combinación.

- Ejemplo:

```
int f ( int n ){
  if (n == 0) return 1;
  else {    // (n > 0)
     int x = f(n-1);
     g(n,x);
     return ....
}
```

donde `g(n)` representa una llamada a función, o un conjunto de instrucciones, cuyo coste de ejecución está en $\mathcal{O}(n)$, siendo n el tamaño del problema.

- El caso base consiste en una comparación, su coste es constante.
- Las instrucciones que acompañan a la llamada recursiva tienen coste $\mathcal{O}(n)$.
- La llamada recursiva se hace con un valor una unidad menor que el valor de entrada a la función.

- Ecuación de recurrencia, donde n representa el valor de entrada `n`:

$$T(n) = \begin{cases} c_0 & \text{si } n = 0 \\ T(n-1) + n & \text{si } n > 0 \end{cases}$$

- Despliegue:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= T(n-3) + (n-2) + (n-1) + n = \\ &= \dots \\ &= T(n-k) + (n-k+1) + (n-k+2) + \dots (n-1) + n \\ &= T(n-k) + \textstyle\sum_{j=0}^{j=k-1} (n-j) \\ &= T(n-k) + \textstyle\sum_{j=0}^{j=k-1} n - \sum_{j=0}^{j=k-1} j \\ &= T(n-k) + n * k - \frac{k*(k-1)}{2} \\ &= \dots \\ &= T(0) + n * n - \frac{n*(n-1)}{2} \in \mathcal{O}(n^2). \end{aligned}$$

1.1.3. Disminución del tamaño del problema por división y coste constante de las funciones sucesor y de combinación.

- Ejemplo. Búsqueda binaria

```
bool search ( std::vector<int> const& v, int ini , int fin , int x ){
  if (ini == fin) return false; // Vector vacio
  else { // Vector con uno o mas elementos
    int m = (ini + fin - 1) / 2;
    if (v[m] == x) return true;
    else if (x < v[m]) return search(v,ini,m);
    else return search(v,m+1,fin);
  }
}
```

- Los dos casos base tienen coste constante, ya que consisten en comparaciones y operaciones aritméticas.
- Las instrucciones que acompañan a la llamada recursiva son comparaciones y operaciones aritméticas, tienen coste constante.
- La llamada recursiva se hace con la mitad de los elementos del vector de entrada.

- Ecuación de recurrencia, donde n es el número de elementos considerados en el vector: `fin - ini`.

$$T(n) = \begin{cases} c_0 & \text{si } n = 1 \\ T(\frac{n}{2}) + c_1 & \text{si } n > 1 \end{cases}$$

- Despliegue:

$$\begin{aligned} T(n) &= T(\tfrac{n}{2}) + c_1 \\ &= T(\tfrac{n}{2^2}) + c_1 + c_1 \\ &= T(\tfrac{n}{2^3}) + c_1 + c_1 + c_1 = \\ &= \dots \\ &= T(\tfrac{n}{2^k}) + k * c_1 \\ &= \dots \\ &= T(1) + \log(n) * c_1 \in \mathcal{O}(\log(n)). \end{aligned}$$

Cálculo del número de veces que se despliega la recurrencia:

$$\begin{array}{lcl} \frac{n}{2^k} = 1 & \Leftrightarrow & \textit{Despejando el valor de } n \\ n = 2^k & \Leftrightarrow & \textit{Tomando logaritmos en ambos terminos de la igualdad} \\ \log_2(n) = \log_2(2^k) & \Leftrightarrow & \textit{Aplicando } \log_2(2^k) = k\log_2(2) \textit{ y } \log_2(2) = 1\textit{)} \\ \log_2(n) = k & & \end{array}.$$

1.1.4. Disminución del tamaño del problema por división y coste lineal de las funciones sucesor y de combinación.

- Ejemplo.

```
int f ( std::vector<int> & v, int ini , int fin){
  if (ini == fin) return 0; // Vector vacio
  else  {
    int m = (ini + fin - 1) / 2;
    if (m ..) {
      int x1 = f(v,ini,m+1);
      // Instrucciones que tratan el resultado de la llamada recursiva
      //con coste lineal en el numero de elementos del vector
    }
    else {
      int x2 = f(v,m+1,fin);
      // Instrucciones que tratan el resultado de la llamada recursiva
      //con coste lineal en el numero de elementos del vector
    }
  }
}
```

  - El caso base tienen coste constante, ya que es una instrucción de comparación.
  - Las instrucciones que acompañan a la llamada recursiva son comparaciones, operaciones aritméticas con coste constante y una serie de instrucciones cuyo coste es lineal en el número de elementos del vector. El coste es el máximo de todos ellos y por lo tanto lineal en el número de elementos del vector de entrada.
  - Las llamadas recursivas se hacen cada una con la mitad de los elementos del vector de entrada.

- Ecuación de recurrencia, donde n es el número de elementos considerados en el vector: `fin - ini`.

$$T(n) = \begin{cases} c_0 & \text{si } n = 1 \\ T(\frac{n}{2}) + n & \text{si } n > 1 \end{cases}$$

- Despliegue:

$$\begin{array}{lcl} T(n) & = & T(\frac{n}{2}) + n \\ & = & T(\frac{n}{2^2}) + \frac{n}{2} + n \\ & = & T(\frac{n}{2^3}) + \frac{n}{2^2} + \frac{n}{2} + n = \\ & = & \ldots \\ & = & T(\frac{n}{2^k}) + \sum_{j=0}^{j=k-1} \frac{n}{2^j} \\ & \approx_{(1)} & T(\frac{n}{2^k}) + 2n \\ & = & \ldots \\ & = & T(1) + 2n \in \mathcal{O}(n). \end{array}$$

donde (1) se desarrolla como:

$$\begin{array}{lcl} \sum_{j=0}^{j=k-1} \frac{n}{2^j} & & \\ = n\sum_{j=0}^{j=k-1} \frac{1}{2^j} & \rightarrow & \textit{Sacando factor común } n \\ = n(\frac{\frac{1}{2^{k-1}}\frac{1}{2} - 1}{\frac{1}{2} - 1} & \rightarrow & \textit{Aplicando la suma de la progresión geométrica} \\ = n(2(1 - \frac{1}{2^k})) & \rightarrow & \textit{Simplificando} \\ = \frac{2^k - 1}{2^{k-1}} & \rightarrow & \textit{Simplificando} \\ \approx_{(1)} 2n & & \end{array}.$$

1.2. Dos llamadas recursivas.

1.2.1. Disminución del tamaño del problema por sustracción y coste constante de las funciones sucesor o de combinación.

- Ejemplo: Torres de Hanoi.

```
void hanoi ( int n, int ini, int aux, int fin ){
  if (n > 0) {
      hanoi(n-1,ini,fin,aux);
      std::cout << "mover de " << ini << " a " << fin << '\n';
      hanoi(n-1,aux,ini,fin);
  }
}
```

  - Los casos base es vacío, su coste es constante.
  - Las instrucciones que acompañan a las llamadas recursiva, dos restas para calcular los parámetros y una instrucción de escritura tienen coste constante.
  - Las llamadas recursivas se hace con un valor una unidad menor que el valor de entrada a la función.

- Ecuación de recurrencia, donde n representa el valor de entrada `n`:

$$T(n) = \begin{cases} c_0 & \text{si } n = 0 \vee n == 1 \\ 2T(n-1) + c_1 & \text{si } n > 1 \end{cases}$$

- Despliegue:

$$\begin{array}{rcl} T(n) & = & 2T(n-1) + c_1 \\ & = & 2(2T(n-2) + c_1) + c_1 \\ & = & 2^2T(n-2) + 2c_1 + c_1 \\ & = & 2^2(2T(n-3) + c_1) + 2c_1 + c_1 \\ & = & 2^3T(n-3) + 2^2c_1 + 2c_1 + c_1 \\ & = & \ldots \\ & = & 2^kT(n-k) + 2^{k-1}c_1 + 2^{k-2}c_1 + \ldots 2c_1 + c_1 \\ & = & 2^kT(n-k) + \sum_{j=0}^{j=k-1}(2^jc_1) \\ & = & 2^kT(n-k) + c_1\sum_{j=0}^{j=k-1}2^j \\ & = & 2^kT(n-k) + c_1\frac{2^{k-1}2-2^0}{2-1} \\ & = & 2^kT(n-k) + c_12^k \\ & = & \ldots \\ & = & 2^nT(0) + c_12^n \in \mathcal{O}(2^n). \end{array}$$

1.2.2. Disminución del tamaño del problema por división y coste constante de las funciones sucesor y de combinación.

- Ejemplo. Esta recurrencia aparece en varios problemas de la asignatura.

```
int f ( std::vector<int> const& v, int ini, int fin){
  if (ini == fin) return 0; // vector vacio
  else {
    int m = (ini + fin - 1) / 2;
    int n1 = f(v,ini,m+1);
    int n2 = f(v,m+1,fin);
    // Instrucciones que combinan el resultado n1 y n2
    // con coste constante
  }
}
```

  - El caso base tienen coste constante, ya que es vacío.

- Las instrucciones que acompañan a la llamada recursiva son comparaciones, operaciones aritméticas y las instrucciones que combinan los resultados de las llamadas recursivas, todas ellas con coste constante.
- Las dos llamadas recursivas se hacen cada una con la mitad de los elementos del vector de entrada.

- Ecuación de recurrencia, donde n es el número de elementos considerados en el vector: `fin - ini`.

$$T(n) = \begin{cases} c_0 & \text{si } n = 1 \\ 2T(\frac{n}{2}) + c_1 & \text{si } n > 1 \end{cases}$$

- Despliegue:

$$\begin{array}{lll} T(n) & = & 2T(\frac{n}{2}) + c_1 \\ & = & 2(2T(\frac{n}{2^2}) + c_1) + c_1 \\ & = & 2^2T(\frac{n}{2^2}) + 2c_1 + c_1 \\ & = & 2^2(2T(\frac{n}{2^3}) + c_1) + 2c_1 + c_1 = \\ & = & 2^3T(\frac{n}{2^3}) + 2^2c_1 + 2c_1 + c_1 = \\ & = & \ldots \\ & = & 2^kT(\frac{n}{2^k}) + c_1 \sum_{j=0}^{j=k-1} 2^j \\ & = & \ldots \\ & = & 2^{\log(n)}T(1) + c_1(2^{\log(n)} - 1) \in \mathcal{O}(n). \end{array}$$

Cálculo del número de veces que se despliega la recurrencia:

$$\begin{array}{lll} \frac{n}{2^k} = 1 & \Leftrightarrow & \textit{Despejando el valor de } n \\ n = 2^k & \Leftrightarrow & \textit{Tomando logaritmos en ambos terminos de la igualdad} \\ \log_2(n) = \log_2(2^k) & \Leftrightarrow & \textit{Aplicando } \log_2(2^k) = k\log_2(2) \textit{ y } \log_2(2) = 1 \\ \log_2(n) = k & & \end{array}.$$

1.2.3. Disminución del tamaño del problema por división y coste lineal de las funciones sucesor y de combinación.

- Ejemplo. Mergesort

```
void mergesort ( std::vector<int> & v, int ini , int fin){
  if (ini + 1 < fin)  { // Vector con dos o mas elementos
    int m = (ini + fin - 1) / 2;
    mergesort(v,ini,m+1);
    mergesort(v,m+1,fin);
    mezcla(v,ini,m,fin);
  }
}
```

  - El caso base tienen coste constante, ya que es vacío.
  - Las instrucciones que acompañan a la llamada recursiva son comparaciones, operaciones aritméticas con coste constante y una llamada a la función `mezcla` cuyo coste es lineal en el número de elementos del vector. El coste es el máximo de todos ellos y por lo tanto lineal en el número de elementos del vector de entrada.
  - Las dos llamadas recursivas se hacen cada una con la mitad de los elementos del vector de entrada.

- Ecuación de recurrencia, donde n es el número de elementos considerados en el vector: `fin - ini`.

$$T(n) = \begin{cases} c_0 & \text{si } n = 1 \\ 2T(\frac{n}{2}) + n & \text{si } n > 1 \end{cases}$$

- Despliegue:

$$\begin{array}{rcl}
T(n) & = & 2T(\frac{n}{2})+n \\
& = & 2(2T(\frac{n}{2^2})+\frac{n}{2})+n \\
& = & 2^2T(\frac{n}{2^2})+n+n \\
& = & 2^2(2T(\frac{n}{2^3})+\frac{n}{2^2})+n+n= \\
& = & 2^3T(\frac{n}{2^3})+n+n+n= \\
& = & \ldots \\
& = & 2^kT(\frac{n}{2^k})+kn \\
& = & \ldots \\
& = & 2^{\log(n)}T(1)+\log(n)*n \in \mathcal{O}(n\log(n)).
\end{array}$$

Cálculo del número de veces que se despliega la recurrencia:

$$\begin{array}{lcl}
\frac{n}{2^k}=1 & \Leftrightarrow & \textit{Despejando el valor de } n \\
n=2^k & \Leftrightarrow & \textit{Tomando logaritmos en ambos terminos de la igualdad} \\
\log_2(n)=\log_2(2^k) & \Leftrightarrow & \textit{Aplicando } \log_2(2^k)=k\log_2(2) \textit{ y } \log_2(2)=1S \\
\log_2(n)=k & &
\end{array}.$$