Reto 1. Estructura de Datos

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Reto 1. Estructura de Datos

Los Del DGIIM, `losdeldgiim.github.io`

Arturo Olivares Martos
Jesús Muñoz Velasco

Granada, 2023-2024

Ejercicio 1. Usando la notación O, determinar la eficiencia de las siguientes funciones:

a)

```
void eficiencia1(int n){
    int x=0; int i,j,k;

    for(i=1; i<=n; i+=4)
        for(j=1; j<=n; j+=[n/4])
            for(k=1; k<=n; k*=2)
                x++;
}
```

Si consideramos cada bucle `for` como una sucesión k_m, j_o, i_p con $m, o, p \in \mathbb{N}$, siendo m, o, p el número de veces que se realiza cada bucle respectivamente, podemos calcular el número de veces que se repite cada uno.

Sucesión	**Condición**	**Veces que se repite**
$k_m = 2^{(m-1)}$	$k \leqslant n$	$\log_2(n) + 1$
$j_o = 1 + \frac{n}{4}(o-1)$	$j \leqslant n$	$\frac{4(n-1)}{n} + 1$
$i_p = 1 + 4(p-1)$	$i \leqslant n$	$\frac{n-1}{4} + 1$

donde la tercera columna la hemos deducido de los siguientes cálculos:

$$2^{(m-1)} \leqslant n \Leftrightarrow (m-1)\log_2(2) \leqslant \log_2(n) \Leftrightarrow m \leqslant \log_2(n) + 1$$

$$1 + \frac{n}{4}(o-1) \leqslant n \Leftrightarrow n(o-1) \leqslant 4(n-1) \Leftrightarrow o \leqslant \frac{4(n-1)}{n} + 1$$

$$1 + 4(p-1) \leqslant n \Leftrightarrow p - 1 \leqslant \frac{n-1}{4} \Leftrightarrow p \leqslant \frac{n-1}{4} + 1$$

Veamos cada bucle de más interior a más exterior:

- Bucle `for` interior:

$$\sum_{m=1}^{\log_2(n)+1} 1 = \log_2(n) + 1 \in O(\log_2(n))$$

- Bucle `for` intermedio:

$$\sum_{o=1}^{\frac{4(n-1)}{n}+1} \log_2(n) = \log_2(n) \sum_{o=1}^{\frac{4(n-1)}{n}+1} 1 = \left(\frac{4(n-1)}{n} + 1\right) \log_2(n) =$$

$$= \left(5 - \frac{4}{n}\right) \log_2(n) \in O(\log_2(n))$$

- Bucle `for` exterior:

$$\sum_{p=1}^{\frac{n-1}{4}+1} \log_2(n) = \log_2(n) \sum_{p=1}^{\frac{n-1}{4}+1} 1 = \log_2(n) \left(\frac{n-1}{4} + 1 \right) \in O(n \log_2(n))$$

Por tanto, tenemos que la eficiencia de dicha función es de orden $O(n \log_2(n))$.

b)

```
int eficiencia2 (bool existe){
    int sum2=0; int k,j,n;

    if (existe)
        for(k=1; k<=n; k*=4)
            for(j=1; j<=k; j++)
                sum2++;
    else
        for(k=1; k<=n; k*=4)
            for(j=1; j<=n; j++)
                sum2++;

    return sum2;
}
```

Veamos en este caso la eficiencia de dicha función. Vemos por un lado la eficiencia de la sentencia `if` y, posteriormente, la del `else`.

- Empezamos en el caso de la sentencia `if`. De forma análoga al apartado anterior, consideramos cada bucle `for` como una sucesión k_m, j_o con $m, o \in \mathbb{N}$, siendo m, o el número de veces que se realiza cada bucle respectivamente. Calculamos los valores de m, o:

Sucesión	**Condición**	**Veces que se repite**
$j_o = o$	$j \leqslant k$	$k = o$
$k_m = 4^{(m-1)}$	$k \leqslant n$	$\frac{\log_2(n)}{2} + 1$

donde la tercera columna se ha obtenido mediante los siguientes cálculos:

$$4^{(m-1)} \leqslant n \Leftrightarrow (m-1)log_2(2^2) \leqslant \log_2(n) \Leftrightarrow 2(m-1) \leqslant \log_2(n) \Leftrightarrow m \leqslant \frac{log_2(n)}{2} + 1$$

Veamos la eficiencia de cada bucle:

  - Bucle `for` interno:

$$\sum_{o=1}^{k} 1 = k$$

- Bucle `for` externo:

$$\sum_{k=1}^{\frac{\log_2(n)}{2}+1} k = \left(\frac{\log_2(n)}{2}+1\right) \cdot \frac{1+\left(\frac{\log_2(n)}{2}+1\right)}{2} =$$

$$= \frac{\frac{\log_2(n)}{2}+1}{2} + \frac{\left(\frac{\log_2(n)}{2}+1\right)^2}{2} \in O(\log_2^2(n))$$

Por tanto, tenemos que el `if` es de orden $O(\log_2^2(n))$.

- En este caso estudiamos el cuerpo del `else`.

  Al igual que antes, consideramos cada bucle `for` como una sucesión k_m, j_o con $m, o \in \mathbb{N}$, siendo m, o el número de veces que se realiza cada bucle respectivamente. Calculamos los valores de m, o:

Sucesión	**Condición**	**Veces que se repite**
$j_o = o$	$j \leqslant n$	n
$k_m = 4^{(m-1)}$	$k \leqslant n$	$\frac{\log_2(n)}{2}+1$

  donde la tercera columna se ha obtenido mediante los cálculos del apartado anterior.

  Veamos la eficiencia de cada bucle:

  - Bucle `for` interno:

$$\sum_{o=1}^{n} 1 = n$$

  - Bucle `for` externo:

$$\sum_{k=1}^{\frac{\log_2(n)}{2}+1} n = \left(\frac{\log_2(n)}{2}+1\right) n \in O(n\log_2(n))$$

  Por tanto, tenemos que el `else` es de orden $O(n\log_2(n))$.

Por tanto, como en términos de eficiencia nos ponemos en el peor de los casos, tenemos que esta función es $O(n\log_2(n))$, ya que:

$$n\log_2(n) \geqslant \log_2^2(n) \Longleftrightarrow n \geqslant \log_2(n), \qquad \forall n > 1$$

donde lo hemos demostrado para valores mayores que 1, y la desigualdad final es cierta trivialmente.

c)

```
void eficiencia3 (int n){
    int j; int i=1; int x=0;

    do{
        j=1;
        while (j <= n){
            j=j*4;
            x++;
        }
        i++;
    }while (i<=n);
}
```

```
void eficiencia4 (int n){
    int j; int i=2; int x=0;

    do{
        j=1;
        while (j <= i){
            j=j*4;
            x++;
        }
        i++;
    }while (i<=n);
}
```

Empezamos con la función `eficiencia3`:

Consideramos cada bucle como una sucesión j_m, i_o con $m, o \in \mathbb{N}$, siendo m, o el número de veces que se realiza cada bucle. Calculamos los valores de m, o:

Sucesión	**Condición**	**Veces que se repite**
$j_m = 4^{m-1}$	$j \leqslant n$	$\frac{\log_2(n)}{2} + 1$
$i_o = o$	$i \leqslant n$	n

donde la tercera columna se ha obtenido mediante los siguientes cálculos:

$$4^{(m-1)} \leqslant n \Leftrightarrow (m-1)log_2(2^2) \leqslant \log_2(n) \Leftrightarrow 2(m-1) \leqslant \log_2(n) \Leftrightarrow m \leqslant \frac{log_2(n)}{2} + 1$$

Veamos la eficiencia de cada bucle:

- Bucle `while`:

$$\sum_{j=1}^{\frac{\log_2(n)}{2}+1} 1 = \frac{\log_2(n)}{2} + 1$$

- Bucle `do-while`:

$$\sum_{i=1}^{n} \frac{\log_2(n)}{2} + 1 = \left(\frac{\log_2(n)}{2} + 1\right) n \in O(n \log_2(n))$$

Por tanto, tenemos que la función `eficiencia3` es de orden $O(n \log_2(n))$.

Continuamos con la función `eficiencia4`:

Consideramos cada bucle como una sucesión j_m, i_o con $m, o \in \mathbb{N}$, siendo m, o el número de veces que se realiza cada bucle. Calculamos los valores de m, o:

Sucesión	**Condición**	**Veces que se repite**
$j_m = 4^{m-1}$	$j \leqslant i$	$\frac{\log_2(i)}{2} + 1$
$i_o = o + 1$	$i \leqslant n$	$n - 1$

donde la tercera columna se ha obtenido mediante los siguientes cálculos:

$$4^{(m-1)} \leqslant i \Leftrightarrow (m-1)log_2(2^2) \leqslant \log_2(i) \Leftrightarrow 2(m-1) \leqslant \log_2(i) \Leftrightarrow m \leqslant \frac{log_2(i)}{2}+1$$

Veamos la eficiencia de cada bucle:

- Bucle `while`:

$$\sum_{j=1}^{\frac{\log_2(i)}{2}+1} 1 = \frac{\log_2(i)}{2} + 1$$

- Bucle `do-while`:

$$\sum_{i=2}^{n} \frac{\log_2(i)}{2} + 1 = -\frac{\log_2(1)}{2} - 1 + \sum_{i=1}^{n} \frac{\log_2(i)}{2} + 1 =$$

$$= -\frac{\log_2(1)}{2} - 1 + n \cdot \frac{\frac{\log_2(1)}{2} + 1 + \frac{\log_2(n)}{2} + 1}{2} \in O(n\log_2(n))$$

Por tanto, tenemos que la función `eficiencia4` es de orden $O(n\log_2(n))$.

Ejercicio 2. Considerar el siguiente segmento de código con el que se pretende buscar un entero x en una lista de enteros `L` de tamaño n (el bucle `for` se ejecuta n veces):

```
void eliminar (Lista L, int x){
    int aux, p;

    for (p=primero(L); p!=fin(L);){
        aux=elemento (p,L);

        if (aux==x)
            borrar (p,L);
        else p++;
    }
}
```

Analizar la eficiencia de la función `eliminar` si:

1. `primero` es $O(1)$ y `fin`, `elemento` y `borrar` son $O(n)$. ¿Cómo mejorarías esa eficiencia con un solo cambio en el código?

   Con esta estructura y sabiendo que el bucle `for` se realiza **n** veces y que en el condicional `if/else` la parte menos eficiente es la función borrar (`borrar` $\in O(n)$) tenemos que la eficiencia es:

   $$1+1+\sum_{i=1}^{n}(n+n+1+n) = 2+n(3n+1) = 3n^2+n+2 \in O(n^2)$$

   Por tanto, estamos ante una función de orden $O(n^2)$.

   Además, tenemos que no es posible reducir el orden de eficiencia de la función. Esto se debe a que el cuerpo del bucle, debido a la función `borrar`, es $O(n)$ y no se puede reducir, ya que ha de pertenecer al bucle. Además, como se realizarán n repeticiones, la función será necesariamente $O(n^2)$. Sí es cierto que, al igual que se ha realizado en el apartado 2), se podría sacar la función `fin(L)` del bucle, pero en este caso no mejoraría el orden de la eficiencia.

2. `primero`, `elemento` y `borrar` son $O(1)$ y `fin` es $O(n)$. ¿Cómo mejorarías esa eficiencia con un solo cambio en el código?

   Con esta estructura y sabiendo que el bucle `for` se realiza **n** veces y que en el condicional `if/else` la parte menos eficiente es la función borrar (`borrar` $\in O(1)$) tenemos que la eficiencia es:

   $$1+1+\sum_{i=1}^{n}(n+1+1+1) = 2+n(n+3) = n^2+3n+2 \in O(n^2)$$

   En este caso valdría con crear una variable auxiliar que almacenase la posición final `pos_final`. Una vez hecho esto valdría con hacer `pos_final--` después de la instrucción `borrar`. Con esto el programa quedaría:

```
void eliminar (Lista L, int x){
    int aux, p;
    int pos_final=fin(L);
    for (p=primero(L); p!=pos_final;){
        aux=elemento (p,L);
        if (aux==x){
            borrar (p,L);
            pos_final--;
        }
        else p++;
    }
}
```

Código fuente 1: Código con las mejoras del apartado 2)

Con este cambio ahora la eficiencia se calcularía como:

$$1 + n + 1 + \sum_{i=1}^{n}(1 + 1 + 1) = 2 + n + 3n = 4n + 2 \in O(n)$$

Como podemos ver, la eficiencia se ha mejorado de forma significativa.

3. Todas las funciones son $O(1)$. ¿Puede en ese caso mejorarse la eficiencia con un solo cambio en el código?

   Con esta estructura y sabiendo que el bucle `for` se realiza **n** veces y que en el condicional `if/else` la parte menos eficiente es la función borrar (`borrar` $\in O(1)$) tenemos que la eficiencia es:

$$1 + 1 + \sum_{i=1}^{n}(1 + 1 + 1 + 1) = 2 + n(4) = 4n + 2 \in O(n)$$

   Dado que el propio bucle `for` tiene eficiencia $O(n)$, no es posible mejorar la eficiencia actual del programa sin eliminar el bucle `for`, lo cual haría inservible el programa. Es decir, no se puede mejorar. La única forma de mejorar la eficiencia sería reducir de alguna forma el número de iteraciones, algo que no es posible ya que se han comprobar todos los elementos.