TRABAJO PRÁCTICO 2
ESTRUCTURAS DE DATOS Y ALGORITMOS 2

REALIZADO POR

Paoloni Gianfranco
Zimmermann Sebastián

Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

06/06/2019

1. Introducción

A lo largo del trabajo vamos a utilizar el siguiente modelo de especificación de costos.

$$W(c) = 1 \tag{1}$$

$$W(op\ e) = 1 + W(e) \tag{2}$$

$$W(e_1,\ e_2) = 1 + W(e_1) + W(e_2) \tag{3}$$

$$W(e_1 \parallel e_2) = 1 + W(e_1) + W(e_2) \tag{4}$$

$$W(let\ x = e_1\ in\ e_2) = 1 + W(e_1) + W(e_2[Eval(e_1)/x]) \tag{5}$$

$$W(f(x)\ :\ x \in A) = 1 + \sum_{x \in A} W(f(x)) \tag{6}$$

$$S(c) = 1 \tag{7}$$

$$S(op\ e) = 1 + S(e) \tag{8}$$

$$S(e_1,\ e_2) = 1 + S(e_1) + S(e_2) \tag{9}$$

$$S(e_1 \parallel e_2) = 1 + \text{máx}\ (S(e_1),\ S(e_2)) \tag{10}$$

$$S(let\ x = e_1\ in\ e_2) = 1 + S(e_1) + S(e_2[Eval(e_1)/x]) \tag{11}$$

$$S(f(x)\ :\ x \in A) = 1 + \max_{x \in A} S(f(x)) \tag{12}$$

Aclaraciones:

- En los casos en los que fuera necesario, analizaremos otras funciones auxiliares o dividiremos el análisis en partes, si existen sustituciones u otras funciones no analizadas.
- En el caso particular de los arreglos, analizaremos inicialmente algunas funciones utilizadas en las pruebas, ya que dichas complejidades fueron especificadas en el trabajo, y si ciertas funciones simplemente eran equivalentes a las que ya existian en el TAD de arreglos, nos pareció suficiente aclarar que tienen la misma complejidad.
- Los trabajos y profundidades realizados por operaciones constantes, fueron ignorados o nominados con la letra k, dado que estos no afectan el orden de complejidad de las funciones a analizar.

  Si bien, en una gran cantidad de ejercicios son mencionados, como no afectan el resultado, decidimos no aclararlos en nuestros pasos, al menos, en las pruebas no formales.

2. Especificación de costos - Implementación con listas

2.1. filterS

2.1.1. Código

```
filterS p []     = []
filterS p (x:xs) = let (truthy,t) = p x ||| filterS p xs
                   in if truthy then x:t else t
```

2.1.2. Trabajo

El trabajo de filterS tiene 2 argumentos, la función (p) y la longitud de la lista xs ($|xs|$). En este caso, utilizaremos el modelo para demostrar el orden de complejidad del trabajo y la profundidad.

$$
\begin{aligned}
W_{filterS}(p,0) &= c_1 \\
W_{filterS}(p,xs) &= W(let\ (truthy,t) = p\ x \parallel filterS\ p\ xs) && (5)\ y\ e\ es\ constante \\
&= W(px \parallel filterS\ p\ xs) + k && (4) \\
&= W_{filterS}(p,xs-1) + W_p(x_0) + k \\
\\
&<= |xs| * k + \sum_{i=0}^{|xs|-1} W_p(x_i)
\end{aligned}
$$

Se concluye:

$$W_{filterS}(p,xs) \in O\left(|xs| + \sum_{i=0}^{|xs|-1} W_p(x_i)\right)$$

2.1.3. Profundidad

$$
\begin{aligned}
S_{filterS}(p,0) &= c_1 \\
S_{filterS}(p,xs) &= S(let\ (truthy,t) = p\ x \parallel filterS\ p\ xs) && (5)\ y\ e\ es\ constante \\
&= S(px \parallel filterSp\ xs) + k && (4) \\
&= \text{máx}\left(S_{filterS}(p,xs-1) + W_p(x_0)\right) + k \\
\\
&<= |xs| * k + \max_{i=0}^{|xs|-1} W_p(x_i)
\end{aligned}
$$

Se concluye:

$$S_{filterS}(p,xs) \in O\left(|xs| + \max_{i=0}^{|xs|-1} W_p(x_i)\right)$$

2.2. showtS

2.2.1. Código

```
showtS []  = EMPTY
showtS [x] = ELT x
showtS xs  = let (l,r) = take n xs ||| drop n xs
             in NODE l r
             where n = lengthS xs `div` 2
```

2.2.2. Trabajo

Los dos primeros casos consisten simplemente en el pattern matching y enlos constructores de TreeView. De forma que su trabajo es constante.

El tercer caso, sin embargo, tiene una combinación de de take y drop (ambas de trabajo lineal sobre la longitud de la secuencia, es decir $O(|xs|)$. Luego de esto, también está el trabajo de $lengthS$ que también tiene una complejidad de orden lineal en el tamaño de la secuencia.

Finalmente se le suma un trabajo adicional constante del constructor de tipo de NODE y las asignaciones.

Tomando el peor caso, concluimos que el trabajo es la suma de 3 trabajos de orden lineal en el tamaño de la lista. Por lo tanto, concluimos que:

$$W_{showtS}(xs) \in O(|xs|)$$

2.2.3. Profundidad

En el caso de la profundidad sucede algo parecido. Los primeros dos casos tienen profundidad constante.

En el tercer caso la profundidad de $lengthS$, $takeS$ y $dropS$ es lineal en la longitud de la lista.

Al paralelizar entre cada take y drop, si bien la profundidad es máx(S_{takeS}, S_{dropS}) al tener ambas profundidad lineal, el peor caso sigue siendo lineal en la longitud de la lista.

Por lo tanto, concluimos que:

$$S_{showtS}(xs) \in O(|xs|)$$

2.3. reduceS

2.3.1. Código

```
reduceS f e []       = e
reduceS f e [x]      = x
reduceS f e (x:y:xs) = let (h,t) = f x y ||| reduceS f e xs
                       in f h t
```

2.3.2. Trabajo

Los dos primeros casos consisten simplemente en el pattern matching. De forma que su trabajo es constante.

El tercer caso, sin embargo, hace una recursión de la función $reduceS$ sobre si misma en paralelo, es decir que se aplica a lo largo de la longitud de la lista $|xs|$. Luego de esto, también está el trabajo de cada aplicación de la función f (que denotaremos $\oplus$) que también agrega su complejidad a la ejecución del programa en cada paso.

Finalmente se le suma un trabajo adicional constante de las aplicaciones de la asignación, pero al ser constante nada de esto afecta el orden de complejidad.

Tomando el peor caso, el trabajo es la suma de un trabajo de orden lineal de recorrer la lista con el trabajo de aplicar la función $(\oplus)$ en cada uno de los pasos. Por lo tanto, concluimos que:

$$W_{reduceS}(\oplus, xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, e, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

2.3.3. Profundidad

El caso de la profundidad es bastante similar.

En los primeros casos del pattern matching su profundidad es constante.

En el tercer caso, al hacer recursión en paralelo, el trabajo de dicha función es del orden de la máxima profundidad de todos los llamados. Además, al realizar operaciones de a pares de elementos al mismo tiempo en paralelo, la profundidad de llamados es de a lo sumo $lg|xs|$. Sin embargo, el trabajo de recorrer la lista (así sea de dos en dos) es lineal, y resulta la profundiad:

Por lo tanto, concluimos que:

$$S_{reduceS}(\oplus, xs) \in O\left(|xs| + \max_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, e, xs)} S\left(xs_i \oplus xs_j\right)\right)$$

2.4. scanS

2.4.1. Código

```
scanS f e []  = ([], e)
scanS f e [x] = ([e], f e x)
scanS f e s   = let s' = scanS f e $ contract s
                    r  = expand s $ fst s'
                in (r, snd s')
    where
        contract []       = []
        contract [x]      = [x]
        contract (x:y:xs) = let (h,t) = f x y ||| contract xs
                            in h:t
        expand [] ys          = ys
        expand [_] ys         = ys
        expand (x:_:xs) (y:ys) = let (z,t) = f y x ||| expand xs ys
                                 in y:z:t
```

Claramente, antes de comenzar a analizar el trabajo y profundidad de $scanS$, deberemos ponernos en la tarea de analizar $contract$ y $expand$.

2.4.2. contract

El $contract$ es equivalente a reduceS, por lo que su trabajo está dado por:

$$W_{contract}(xs) \in O\left(|xs| + \sum_{i=0}^{\frac{|xs|}{2}-1} W(xs_{2i} \oplus xs_{2i+1})\right)$$

Y una profunidad dada por:

$$S_{contract}(xs) \in O\left(|xs| + \max_{i=0}^{\frac{|xs|}{2}-1} W(xs_{2i} \oplus xs_{2i+1})\right)$$

2.4.3. expand

En el caso de $expand$ posee dos casos del pattern matching cuya respuesta es constante. Además trabaja sobre la longitud de xs.

Y el tercer caso, tiene una estructura similar al contract, en el sentido que ejecuta una función operador (que denotaremos $\oplus$) y que paraleliza por sobre todos los llamados.

Por lo que en el caso de trabajo, es recorrer la secuencia y la sumatoria de las aplicaciones del operador $\oplus$

$$W_{expandS}(xs) \in O\left(|xs| + \sum_{i=0}^{\frac{|xs|}{2}-1} W\left(xs_{(2i+1)/2} \oplus ys_{2i}\right)\right)$$

Y en el caso de la profundidad, es recorrer la secuencia y el máximo de las aplicaciones del operador $\oplus$

$$S_{expandS}(xs) \in O\left(|xs| + \max_{i=0}^{\frac{|xs|}{2}-1} S\left(xs_{(2i+1)/2} \oplus ys_{2i}\right)\right)$$

Hechas dichas aclaraciones, procedemos a analizar el trabajo y profundidad de scanS

2.4.4. Trabajo

En los dos primeros casos del pattern matching el trabajo es claramente constante.

En el tercer caso, se realiza una llamada recursiva sobre un $contract$ y a su vez se realiza un $expand$ sobre este llamado.

Inicialmente se ve que se requiere recorrer toda la lista, por lo que se posee un trabajo lineal en el tamaño de la lista.

Luego, se le suma el trabajo de utilizar $expand$ y el de utilizar $contract$ resultando entonces que:

$$W_{scanS}(\oplus, xs) = W_{scanS}\left(\oplus, \frac{|xs|+1}{2}\right) + W_{contract} + W_{expand}$$

De este resultado y de los valores precalculados de $contract$ y $expand$ se puede concluir que:

$$W_{scanS}(\oplus, xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_s(\oplus, e, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

2.4.5. Profundidad

En el caso de la profundidad, el análisis es análogo. Es fácil ver que

$$S_{scanS}(\oplus, xs) = S_{scanS}\left(\oplus, \frac{|xs|+1}{2}\right) + S_{contract} + S_{expand}$$

Por lo tanto, concluimos que:

$$S_{scanS}(\oplus, xs) \in O\left(|xs| + \max_{(xs_i \oplus xs_j) \in \mathcal{O}_s(\oplus, e, xs)} S\left(xs_i \oplus xs_j\right)\right)$$

3. Especificación de costos - Implementación con arreglos

3.1. Aclaraciones iniciales

Recordemos los siguientes trabajos dados por sabidos:

$$W_{tabulateS}(p, xs) \in O\left(\sum_{i=0}^{|xs|-1} W(p\ xs_i)\right)$$

$$W_{takeS}(xs) = W_{dropS}(xs) = W_{subArray}(xs) \in O(1)$$

$$W_{nthS}(xs) \in O(|xs|)$$

$$W_{joinS}(xs) = W_{flatten}(xs) \in O\left(|xs| + \sum_{i=0}^{|xs|-1} W(|x\ !\ i|)\right)$$

Y sus respectivas profundidades

$$S_{tabulateS}(p, xs) \in O\left(\max_{i=0}^{|xs|-1} S(p\ xs_i)\right)$$

$$S_{takeS}(xs) = S_{dropS}(xs) = S_{subArray}(xs) \in O(1)$$

$$S_{nthS}(xs) \in O(1)$$

$$S_{joinS}(xs) = S_{flatten}(|xs|) \in O(lg(|xs|))$$

3.2. Análisis contract

3.2.1. Código

```
contract :: (a -> a -> a) -> (A.Arr a) -> (A.Arr a)
contract f xs = let half = (lengthS xs + 1) `div` 2
                in tabulateS g $ half
    where n    = lengthS xs
          half = (n + 1) `div` 2
          p i  = i < half - 1 || even n
          g i  = if p i then f (xs ! (2*i)) (xs ! (2*i+1))
                 else xs ! (2*i)
```

3.2.2. Trabajo

El trabajo de *contract* es igual a una constante (suma, división, asignaciones), más el trabajo de lenghtS (constante), más la aplicación de $tabulateS$. Como la función argumento que se pasa a tabulateS realiza operaciones aritméticas e indexaciones sobre la secuencia (trabajo constante) más la aplicación de f, el trabajo resulta dado por

$$W_{tabulateS}(p, |xs|) \in O\left(\sum_{i=0}^{|xs|-1} Wf\ xs_i\right)$$

lo que implica que

$$W_{contract}(p, xs) \in O\left(\sum_{i=0}^{\frac{|xs|}{2}-1} Wxs_{2i} \oplus xs_{2i+1}\right)$$

3.2.3. Profundidad

El análisis es análogo al realizado para el trabajo, siendo

$$S_{tabulateS}(p, |xs|) \in O\left(\max_{i=0}^{|xs|-1} Wf\ xs_i\right)$$

lo que implica que

$$S_{contract}(p, xs) \in O\left(\max_{i=0}^{\frac{|xs|}{2}-1} Wxs_{2i} \oplus xs_{2i+1}\right)$$

3.3. Análisis expand

3.3.1. Código

```
expand :: (a -> a -> a) -> (A.Arr a) -> (A.Arr a) ->  (A.Arr a)
expand f xs s' = case lengthS xs of
        0 -> s'
        1 -> s'
        n -> tabulateS g n
    where g i | even i    = s' ! (i `div` 2)
| otherwise = f (s' ! (i `div` 2)) (xs ! (i-1))
```

3.3.2. Trabajo

De la misma manera que $contract$, expand se define en base a $tabulateS$, resultando así

$$W_{expand}(p,xs) \in O\left(\sum_{i=0}^{\frac{|xs|}{2}-1} W xs_{2i} \oplus xs_{2i+1}\right)$$

3.3.3. Profundidad

Análogamente deducimos que

$$S_{expand}(p,xs) \in O\left(\max_{i=0}^{\frac{|xs|}{2}-1} W xs_{2i} \oplus xs_{2i+1}\right)$$

3.4. filterS

3.4.1. Código

```
filterS p xs  = case lengthS xs of
            0 -> emptyS
            n -> let s = tabulateS pByIndex n
                 in joinS s
            where pByIndex i = if p (xs ! i) then singletonS (xs ! i)
                               else emptyS
```

3.4.2. Trabajo

Comenzamos por observar que el trabajo de la función auxiliar $pByIndex$ está dado por el predicado p, por lo tanto

$$W_{pByIndex}(i) \in O(p\ xs_i)$$

Esta función se pasa como parámetro a $tabulateS$ siendo entonces

$$W_{tabulateS}(p,xs) \in O\left(\sum_{i=0}^{|xs|-1} W(p\ xs_i)\right)$$

El resultado de este $tabulateS$ es luego pasado por $joinS$, que al recibir una secuencia de secuencias unitarias (cada elemento es un $singletonS$) tiene un trabajo

$$W_{tabulateS}(|xs|) \in O(|xs|)$$

Como el trabajo del $tabulateS$ será en el mejor caso de orden lineal, resulta entonces que

$$W_{filterS}(p,xs) \in O\left(\sum_{i=0}^{|xs|-1} W(p\ xs_i)\right)$$

3.4.3. Profundidad

Haciendo un análisis análogo al que se hizo en el trabajo, tenemos que

$$S_{tabulateS}(p,xs) \in O\left(\max_{i=0}^{|xs|-1} S(p\ xs_i)\right)$$

$$S_{joinS}(xs) \in O(lg(|xs|))$$

Como en este caso la profundidad de $tabulateS$ puede resultar constante, debemos tener en cuenta que $filterS$ será a lo sumo tan eficiente como $joinS$

$$S_{filterS}(p,xs) \in O\left(lg(|xs|) + \max_{i=0}^{|xs|-1} S(p\ xs_i)\right)$$

3.5. showtS

3.5.1. Código

```
showtS xs  = case lengthS xs of
        0 -> EMPTY
        1 -> ELT $ xs ! 0
        n -> let sz = n `div` 2
                 (l,r) = takeS xs sz ||| dropS xs sz
             in NODE l r
```

3.5.2. Trabajo

Inicialmente, el uso de lengthS es constante.

En los primeros dos casos su complejidad es constante, dado que solo se utilizan los constructores de $TreeView$ y el $nthS$.

Luego en el tercer caso se puede ver claramente que solo se utiliza el constructor de tipo $NODE$ y el $takeS$ y $dropS$ sobre la mitad de la longitud de la lista.

Dado que ambos realizan trabajo sobre la mitad de la lista, la complejidad del $showtS$ es claramente la suma de los trabajos de $takeS$ y $dropS$. Es decir:

$$W_{showtS}(xs) = W_{takeS}(xs) + W_{dropS}(xs)$$

Por esto concluimos que:

$$W_{showtS}(xs) \in O(1)$$

3.5.3. Profundidad

Para el caso de la profundidad, en análisis es análogo, y además de todo se realizan en paralelo.

$$S_{showtS}(xs) = S_{takeS}(xs) + S_{dropS}(xs)$$

Por esto concluimos que:

$$S_{showtS}(xs) \in O(1)$$

3.6. reduceS

3.6.1. Código

```
reduceS f e xs = case lengthS xs of
        0 -> e
        1 -> f e $ xs ! 0
        _ -> reduceS f e $ contract f xs
```

3.6.2. Trabajo

El trabajo de reduce está dado tanto por el costo de la función($\oplus$) como de la longitud de la lista.
Como ya sabemos, lengthS es constante, así como nthS y el pattern matching.

Por lo tanto, a la hora de analizar el peor caso, solo nos concentraremos en el tercer caso del pattern matching. Este consiste en la recursión de $reduceS$ aplicando la función $\oplus$ al resultado de $contract$ con los mismos argumentos.

En otras palabras, $reduceS$ recorre toda la lista y aplica $\oplus$ a todos los elementos (utilizando $contract$). Podemos concluir que:

$$W_{reduceS}(\oplus, xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \bar{O}_r(\oplus, e, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

3.6.3. Profundidad

Sin embargo, en el caso de la profundidad, al realizar todas las operaciones en paralelo, termina reduciendo el trabajo de la suma de todas las operaciones al máximo de estas. Es decir, la profunidad de $contract$

Y además, al realizar todas estas operaciones de a dos elementos, solo las aplicaremos a lo sumo $lg(|xs|)$ de veces.

Por lo que, podemos concluir que:

$$W_{reduceS}(\oplus, xs) \in O\left(lg(|xs|) + \max_{(xs_i \oplus xs_j) \in \bar{O}_r(\oplus, e, xs)} S\left(xs_i \oplus xs_j\right)\right)$$

3.7. scanS

3.7.1. Código

```
scanS f e xs = case lengthS xs of
        0 -> (emptyS, e)
        1 -> (singletonS e, f e $ xs ! 0)
        _ -> let s' = scanS f e $ contract f xs
                 r  = expand f xs $ fst s'
             in (r, snd s')
```

3.7.2. Trabajo

Recordemos que

$$W_{contract}(\oplus, xs) \in O\left(|xs| + \sum_{i=0}^{\frac{|xs|}{2}-1} W(xs_i \oplus xs_{i+1})\right)$$

$$W_{expand}(\oplus, xs) \in O\left(|xs| + \sum_{i=0}^{\frac{|xs|}{2}-1} W(xs_i \oplus xs_{i+1})\right)$$

$scanS$ se define en base a estas dos funciones, siendo que realiza una llamada recursiva a sí mismo luego de la contracción, con una entrada de tamaño igual a la mitad de la secuencia (en cada iteración $contract$ nos comprime la secuencia a la mitad de su tamaño). Como debemos tener en cuenta las aplicaciones de $\oplus$, la complejidad del trabajo de $scanS$ nos da

$$W_{scanS}(\oplus, xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_s(\oplus, e, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

3.7.3. Profundidad

Si pudiésemos realizar todas las aplicaciones de $\oplus$ a realizar en una llamada de scan en paralelo, entonces necesitaríamos $lg\ |xs|$ pasos para realizar el cálculo (pues recordemos que en cada llamada recursiva, $contract$ nos comprime la lista a la mitad de tamaño). Esto nos sugiere que la profundiad de $scanS$ esta dada por

$$s_{scanS}(\oplus, xs) \in O\left(lg\ |xs| + \max_{(xs_i \oplus xs_j) \in \mathcal{O}_s(\oplus, e, xs)} S\left(xs_i \oplus xs_j\right)\right)$$