Especificación de costos

Estructuras de Datos y Algoritmos II
Trabajo práctico 2

Hernán Gurmendi
hgurmen@gmail.com

Nicolás Mendez Shurman
nicol12b@gmail.com

4 de junio de 2015

Docentes

Mauro Jaskelioff
Cecila Manzino
Juan M. Rabasedas
Eugenia Simich

Implementación con listas

`filterS`

La implementación de `filterS` con listas es similar a la función `filter` del `Prelude`, salvo que en el caso de una lista con más de un elemento se calcula el predicado con el primer elemento en paralelo al cálculo del paso recursivo, de forma que para el trabajo se obtiene la suma de los trabajos de los predicados y para la profundidad se obtiene la longitud de la secuencia más la máxima de las profundidades de los predicados, tal como se nota a continuación:

$$W\left(filterS\ f\ s\right) \in O\left(\sum_{i=0}^{|s|-1} W\left(f\ s_i\right)\right)$$

$$S\left(filterS\ f\ s\right) \in O\left(|s| + \max_{i=0}^{|s|-1} S\left(f\ s_i\right)\right)$$

`showtS`

`showtS` precisa partir la secuencia dada en dos mitades, de modo que utiliza las funciones `takeS` y `dropS` (que en el caso de la implementación con listas son las mismas funciones que hay en `Prelude`) que tienen orden lineal. Esto resulta en costo lineal para trabajo y profundidad:

$$W\left(showtS\ s\right) \in O\left(|s|\right)$$

$$S\left(showtS\ s\right) \in O\left(|s|\right)$$

`reduceS`

Para la implementación de `reduceS` con listas se usa la función `contract` que toma una operación binaria $\oplus$ y una secuencia s y la contrae aplicando la operación binaria a cada par de elementos contiguos. Para esto se calcula la operación entre dichos elementos en paralelo al cálculo del paso recursivo de `contract`, resultando en la suma de los trabajos de $\oplus$ aplicado a cada par de elementos contiguos de la secuencia para el trabajo y la longitud de la secuencia más la máxima de las profundidades de $\oplus$ aplicado a cada par de elementos contiguos de la secuencia para la profundidad:

$$W\left(contract \oplus s\right) \in O\left(|s| + \sum_{i=0}^{\frac{|s|}{2}+1} W\left(s_{2i} \oplus s_{2i+1}\right)\right)$$

$$S\left(contract \oplus s\right) \in O\left(|s| + \max_{i=0}^{\frac{|s|}{2}+1} S\left(s_{2i} \oplus s_{2i+1}\right)\right)$$

Luego, `reduceS` aplicado a una operación binaria $\oplus$, un elemento b, y una secuencia s, se calcula aplicando recursivamente `reduceS` a la misma operación $\oplus$, el mismo elemento b y la secuencia obtenida de aplicar `contract` a s, generando el orden de reducción buscado. Esto es lo mismo para el trabajo y la profundidad, de modo que el costo de recorrer todo el árbol de reducción es proporcional al tamaño de la secuencia (lineal) pues estamos trabajando con listas. Luego, para el trabajo queda la suma de los trabajos de cada aplicación de $\oplus$ en el árbol de reducción más el tamaño de la secuencia y para la profundidad resulta la suma de las profundidades de cada aplicación de $\oplus$ en el árbol de reducción más el tamaño de la secuencia, ya que por cómo hay que forzar el orden de reducción no se puede aprovechar la profundidad de `contract`.

$$W\left(reduceS \oplus\ b\ s\right) \in O\left(|s| + \sum_{(x \oplus y) \in \mathcal{O}_r(\oplus, b, s)} W\left(x \oplus y\right)\right)$$

$$S\left(reduceS \oplus\ b\ s\right) \in O\left(|s| + \sum_{(x \oplus y) \in \mathcal{O}_r(\oplus, b, s)} S\left(x \oplus y\right)\right)$$

scanS

`scanS`, además de utilizar `contract`, emplea la función `combine`, que toma una operación binaria $\oplus$, 2 secuencias, s y s', y una bandera que indica si el índice actual de la secuencia resultante es par (para construir el orden de reducción buscado). Cuando el índice es par (bandera en $True$), devuelve $s'_{\frac{i}{2}}$, mientras que cuando el índice es impar devuelve $s'_{\lfloor \frac{i}{2} \rfloor} \oplus s_{i-1}$ y en este caso ésta operación se efectúa en paralelo al paso recursivo, de modo que se mejora un poco la profundidad, quedando:

$$W\left(combine \oplus\ s\ s'\right) \in O\left(|s| + \sum_{i=1}^{\frac{|s|}{2}} W\left(s'_i \oplus s_{2i-1}\right)\right)$$

$$S\left(combine \oplus\ s\ s'\right) \in O\left(|s| + \max_{i=1}^{\frac{|s|}{2}} S\left(s'_i \oplus s_{2i-1}\right)\right)$$

Luego, `scanS` simplemente aplica `combine` con la operación binaria $\oplus$, la secuencia original, y la secuencia obtenida de aplicar `scanS` a la contracción de la secuencia original con la operación anterior. Dado que `contract` tiene trabajo por lo menos lineal y `combine` también, el trabajo de scan resulta lineal en la longitud de la secuencia más el trabajo de todas las aplicaciones de la operación binaria en el árbol de reducción. En cuanto a la profundidad, `contract` y `combine` también se comportan linealmente por lo menos, de modo que no se puede aprovechar lo poco que se ganó en profundidad, así que resulta la longitud de la secuencia más la suma de todas las profundidades de las aplicaciones de la operación binaria en el árbol de reducción de `scanS`:

$$W\left(scanS \oplus b\ s\right) \in O\left(|s| + \sum_{(x \oplus y) \in \mathcal{O}_s(\oplus, b, s)} W\left(x \oplus y\right)\right)$$

$$S\left(scanS \oplus b\ s\right) \in O\left(|s| + \sum_{(x \oplus y) \in \mathcal{O}_s(\oplus, b, s)} S\left(x \oplus y\right)\right)$$

Implementación con arreglos persistentes

`filterS`

Para la implementación con arreglos persistentes de `filterS` primero se transforma la secuencia dada en una secuencia donde cada elemento es un singleton cuando el elemento cumple con el predicado, o una secuencia vacía cuando no cumple con el predicado. Esto se logra usando `tabulate` con una función que tiene costo igual al del predicado, para luego aplicar flatten a la lista resultante, de tamaño $|s|$, donde cada elemento es una secuencia de longitud 1 o 0. Luego, `filterS` resulta con trabajo igual a la suma de los trabajos de cada aplicación del predicado a los elementos de la secuencia, y profundidad igual al logaritmo de la longitud de la secuencia más el máximo costo de las profundidades de todas las aplicaciones del predicado a los elementos de la secuencia:

$$W\left(filterS\ f\ s\right) \in O\left(\sum_{i=0}^{|s|-1} W(f\ s_i)\right)$$

$$S\left(filterS\ f\ s\right) \in O\left(\lg|s| + \max_{i=0}^{|s|-1} S(f\ s_i)\right)$$

`showtS`

La implementación de `showtS` con arreglos persistentes usa las funciones `takeS` y `dropS` (ambas con trabajo y profundidad $O\left(1\right)$, ya que usan `subArray`), de modo que el trabajo y la profundidad resulta constante:

$$W\left(showtS\ s\right) \in O\left(1\right)$$

$$S\left(showtS\ s\right) \in O\left(1\right)$$

`reduceS`

Para la implementación de `reduceS` con arreglos persistentes definimos una función `contract` que contrae la secuencia dada con una operación binaria utilizando la función `tabulate` provista, que a su vez utiliza una función con el mismo costo que la operación binaria. En el caso del trabajo, `contract` tiene costo igual a la suma de los trabajos de las aplicaciones de la operación binaria a cada par de elementos consecutivos. Por otro lado, la profundidad simplemente tiene costo igual a la máxima profundidad de las aplicaciones de la operación binaria a cada par de elementos consecutivos:

$$W\left(contract \oplus s\right) \in O\left(\sum_{i=0}^{\frac{|s|}{2}+1} W\left(s_{2i} \oplus s_{2i+1}\right)\right)$$

$$S\left(contract \oplus s\right) \in O\left(\max_{i=0}^{\frac{|s|}{2}+1} S\left(s_{2i} \oplus s_{2i+1}\right)\right)$$

Luego, `reduceS` utiliza `contract` en cada llamado recursivo, transformando la secuencia dada en otra con la mitad de longitud original, hasta llegar a un singleton. En el caso del trabajo, esto resulta en recorrer la secuencia en cada paso recursivo (que esto resulta lineal, basta ver que es una recurrencia dada por $T\left(n\right) = T\left(\frac{n}{2}\right) + n$ donde n es el tamaño de la secuencia) más la suma de todos los trabajos de la aplicación de la operación binaria en el árbol de reducción. Por otro lado, la profundidad resulta en $\lg|s|$ (debido a que cada aplicación de `contract` resulta en una secuencia con tamaño igual a la mitad de la original) multiplicado por la máxima profundidad de todas las aplicaciones de la operación binaria en el árbol de reducción (que esto basta para especificar una cota superior, si quisiéramos acotarlo más obtendríamos la suma de las profundidades de todas las aplicaciones de la operación binaria en cada nivel del árbol de reducción, lo que resulta bastante engorroso de escribir, por eso se elije la profundidad dada que aún así es buena cota).

$$W\left(reduceS \oplus b\ s\right) \in O\left(|s| + \sum_{(x \oplus y) \in \mathcal{O}_r(\oplus, b, s)} W\left(x \oplus y\right)\right)$$

$$S\left(reduceS \oplus b\ s\right) \in O\left(\lg |s| \max_{(x \oplus y) \in \mathcal{O}_r(\oplus, b, s)} S\left(x \oplus y\right)\right)$$

`scanS`

La implementación con arreglos persistentes de `scanS`, además de utilizar `reduceS`, emplea la función `combine` que es la que se encarga de construír el orden de reducción buscado. `combine` toma una operación binaria, $\oplus$ y 2 secuencias, s y s', donde s' se espera que sea el resultado de aplicar `scanS` con $\oplus$, un elemento, y la contracción de s con la operación binaria anterior. Ésta función utiliza `tabulate` con la operación $\oplus$ y una función auxiliar que para los índices pares devuelve $s'_{\frac{i}{2}}$ y para los índices impares devuelve $s'_{\lfloor \frac{i}{2} \rfloor} \oplus s_{i-1}$, que claramente aporta al trabajo y a la profundidad sólo en éste último caso, de modo la especificación de costo para `combine` queda así:

$$W\left(combine \oplus s\ s'\right) \in O\left(|s| + \sum_{i=1}^{\frac{|s|}{2}} W\left(s'_i \oplus s_{2i-1}\right)\right)$$

$$S\left(combine \oplus s\ s'\right) \in O\left(\max_{i=1}^{\frac{|s|}{2}} S\left(s'_i \oplus s_{2i-1}\right)\right)$$

Por último, `scanS` simplemente aplica `combine` a la secuencia original y a la obtenida al aplicar `scanS` con la misma operación, el mismo elemento, y la secuencia original contraída. De esta forma, en el caso del trabajo tiene que recorrer toda la secuencia para poder calcular la contracción, además de la suma de todos los trabajos aportados por las aplicaciones de la operación binaria en la reducción de `scanS`. Por otro lado, en el caso de la profundidad sucede algo similar a lo que pasó con `reduceS` (ver apartado anterior) de modo que resulta $\lg |s|$ multiplicado por la máxima profundidad de todas las aplicaciones de la operación binaria en la reducción de `scanS`, quedando:

$$W\left(scanS \oplus b\ s\right) \in O\left(|s| + \sum_{(x \oplus y) \in \mathcal{O}_s(\oplus, b, s)} W\left(x \oplus y\right)\right)$$

$$S\left(scanS \oplus b\ s\right) \in O\left(\lg |s| \max_{(x \oplus y) \in \mathcal{O}_s(\oplus, b, s)} S\left(x \oplus y\right)\right)$$