Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Licenciatura en Ciencias de la Computación
Estructuras de datos y algoritmos II

Especificación de costos

Alumnos:
CRESPO, Lisandro (C-6165/4)
MISTA, Agustín (M-6105/1)

Docentes:
JASKELIOFF, Mauro
RABASEDAS, Juan Manuel
SIMICH, Eugenia

1 de Junio de 2015

Implementación con listas

filterS

Para implementar la función `filterS`, consideramos la función `filter` presente en el preludio, y paralelizamos el llamado recursivo para mejorar el rendimiento si los predicados que `filterS` evalúa son costosos de calcular. Luego podemos considerar a `filterS` como la siguiente recurrencia:

$$T\left(n\right) = T\left(n-1\right) + f\left(n\right)$$

Donde $f(n)$ es el costo de evaluar cada predicado, además del costo de las comparaciones, que consideramos constantes. Puede verse que en esta implementación, el paralelizar las operaciones no mejora el problema de tener que recorrer todo el arreglo de forma secuencial. Resolviendo la recurrencia tenemos entonces:

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

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

Finalmente, si consideramos que $f(n) \in O(1)$ resulta:

$$W\left(filterS \oplus s\right) \in O\left(n\right)$$

$$S\left(filterS \oplus s\right) \in O\left(n\right)$$

showtS

Para el caso de `showtS`, la implementación mediante listas es poco eficiente dado que para poder partir la lista en dos mitades en el caso de que existan dos o más elementos, se necesita conocer el tamaño de la misma, lo cual resulta en un coste lineal tanto para el trabajo como para la profundidad. Por lo tanto:

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

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

reduceS

Para analizar el costo de `reduceS`, primero debemos analizar el comportamiento de la función auxiliar `contract` que, dados una funcion binaria $\oplus$ y una secuencia s, evalúa $\oplus$ tomando pares de elementos contiguos de s, y devuelve la secuencia resultante. Luego, el costo de `contract` esta dado por, recorrer el arreglo s y calcular de forma paralela los costos de $\oplus$ para cada par de elementos contiguos de s (a lo sumo $\frac{|s|}{2}$ cuando $|s|$ es par).

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

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

Luego, si consideramos $W(\oplus), S(\oplus) \in O(1)$, tenemos que:

$$W\left(contract \oplus s\right) \in O\left(|s|\right)$$

$$S\left(contract \oplus s\right) \in O\left(|s|\right)$$

Ahora bien, para calcular el costo de `reduceS`, vemos que éste funciona aplicando recursivamente `contract` sobre el resultado de si mismo, lo que fuerza un orden de reducción en forma de árbol completo a izquierda. Luego el costo de `reduceS` es la suma de los costos de las aplicaciones de `contract` a cada nivel del árbol de reducción, tenemos entonces:

$$W\left(reduceS \oplus s\right) \in O\left(\sum_{i=0}^{\log_2 |s|} W\left(contract \oplus s_i\right)\right) donde\ |s_i| = \frac{1}{2}|s_{i-1}|$$

$$S\left(reduceS \oplus s\right) \in O\left(\sum_{i=0}^{\log_2 |s|} S\left(contract \oplus s_i\right)\right) donde\ |s_i| = \frac{1}{2}|s_{i-1}|$$

Lo que resulta:

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

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

Puede verse en el resultado anterior que la profundidad del algoritmo no da lugar a una buena paralelización dado que el orden de reducción esta fijo, no pudiéndose aprovechar la mejor profundidad de `contract`.

Nuevamente, si consideramos $W(\oplus), S(\oplus) \in O(1)$, tenemos que:

$$W\left(reduceS \oplus s\right) \in O\left(|s|\right)$$

$$S\left(reduceS \oplus s\right) \in O\left(|s|\right)$$

scanS

Implementación con arreglos persistentes

filterS

Para implementar `filterS` mediante arreglos persistentes, primero creamos una nueva secuencia mediante `tabulate` que consta de singletons en el caso de que el elemento correspondiente en la secuencia original cumple con el predicado dado, o de secuencias vacias para los elementos que no lo hacen. Luego obtenemos el resultado aplanando el resultado de `tabulate` mediante `flatten`. Luego podemos ver que el trabajo de `tabulate` resulta como la sumatoria de los trabajos de los predicados evaluados y su profundidad resulta como la máxima profundidad de los predicados, por otro lado, el trabajo de flatten resulta lineal dado que todos los elementos de la secuencia que recibe son singletons o secuencias vacías, y su profundidad resulta como el logaritmo del tamaño de la secuencia de entrada. Finalmente, el trabajo y la profundidad de `filterS` resultan:

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

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

Si consideramos $W(f), S(f) \in O(1)$, tenemos que:

$$W\left(filterS\ f\ s\right) \in O\left(|s|\right)$$

$$S\left(filterS\ f\ s\right) \in O\left(lg|s|\right)$$

showtS

En el caso de showtS para arreglos persistentes, usamos esencialmente la función `subArray` de orden constante tanto en trabajo como profundidad para obtener ambos lados de la vista de árbol de la secuencia, obteniéndose:

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

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

reduceS

Para implementar `reduceS` para arreglos persistentes, usamos (al igual que en la implementación con listas) una función `contract` que evalúa la función pasada entre pares contiguos de la secuencia original. Ésta funciona haciendo uso esencialmente de `tabulate`, por lo que los costos resultan análogos a los de la anterior.

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

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

Si consideramos $W(\oplus), S(\oplus) \in O(1)$, tenemos que:

$$W\left(contract \oplus\ s\right) \in O\left(\frac{|s|}{2}\right)$$

$$S\left(contract \oplus s\right) \in O\left(1\right)$$

Luego, `reduceS` llama recursivamente a `contract` con una secuencia de la mitad del tamaño de la secuencia del llamado anterior, por lo que podemos plantear la misma relación entre `reduceS` y `contract` del caso de listas:

$$W\left(reduceS \oplus s\right) \in O\left(\sum_{i=0}^{\log_2 |s|} W\left(contract \oplus s_i\right)\right) \textit{donde } |s_i| = \frac{1}{2}|s_{i-1}|$$

$$S\left(reduceS \oplus s\right) \in O\left(\sum_{i=0}^{\log_2 |s|} S\left(contract \oplus s_i\right)\right) \textit{donde } |s_i| = \frac{1}{2}|s_{i-1}|$$

Pero en esta implementación contamos con una función `contract` mucho más paralelizable que en la versión de listas, resultando en una profundidad que a lo sumo puede ser tan mala como llamar h veces a la peor evaluación de $\oplus$, donde h es la altura del árbol. El trabajo de la misma resulta similar al de la versión de listas, puesto que recorrer los elementos de cada nivel del árbol tiene un coste lineal y además debemos sumar todos los trabajos de las evaluaciones de $\oplus$.

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

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

Además, si consideramos $W(\oplus), S(\oplus) \in O(1)$, resultan:

$$W\left(reduceS \oplus\ e\ s\right) \in O\left(|s|\right)$$

$$S\left(reduceS \oplus\ e\ s\right) \in O\left(lg|s|\right)$$

scanS

Para el caso de la implementación de arreglos persistentes de `scanS`, se hace uso de las funciones `contract`(analizada para `reduceS`) y `combine`, la cual combina dos secuencias mediante un operador $\oplus$, sabiendo que una de ellas es el resultado de aplicar recursivamente `contract` y `scanS` a la otra, con lo que obtenemos resultados parciales de `scanS` en cada llamada a `combine`. `combine` hace uso esencialmente de `tabulate` que crea una secuencia mediante una función que evalúa si el indice actual es par o no, y cuyo costo es a lo sumo evaluar $\oplus$, por lo que el costo de combine dependen únicamente de el costo de $\oplus$, resultando:

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

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

Si consideramos $W(\oplus), S(\oplus) \in O(1)$, tenemos que:

$$W\left(combine \oplus s\ partial\right) \in O\left(\frac{|s|}{2}\right)$$

$$S\left(combine \oplus s\ partial\right) \in O\left(1\right)$$

Luego, `scanS` llama recursivamente a `contract` y ejecuta `combine` en el primer elemento de la tupla obtenida para actualizar los valores de la secuencia de reducción. En cada llamado recursivo, `contract` recibe una secuencia de tamaño igual a la mitad de la del llamado recursivo anteror, por lo que el costo de `scanS` resulta similar al de `reduceS`

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

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

Finalmente, si consideramos $W(\oplus), S(\oplus) \in O(1)$, tenemos que:

$$W\left(scanS \oplus e\ s\right) \in O\left(|s|\right)$$

$$S\left(scanS \oplus e\ s\right) \in O\left(lg|s|\right)$$

CRESPO, Lisandro

MISTA, Agustín