Ordenando Strings

Bernardo Subercaseaux

Universidad de Chile

November 21, 2018

Outline

- Encuesta docente
- Feedback de la tarea 1
- Ordenando Strings
- Problema de control

Encuesta Docente

- ¡Gracias por el feedback!
- Necesitamos feedback más periódico
- Resolución de dudas durante el control
- Ser ordenado y métodico
- ¿Uso del foro?
- ¿Más lento o más rápido?

Feedback de la tarea

- Los informes, salvo excepciones, están horribles
- Releer los archivos en material docente
- Mathematical Writing para quienes se motiven
- Tienen que mejorar para la tarea 2

Definición del problema

Tenemos n strings, cada uno de largo m. Eso quiere decir que el tamaño total del input es $N = n \cdot m$.

Queremos ordenarlos lexicográficamente de forma eficiente, es decir, en $O(N)$.

Orden ingenuo

Elegimos nuestro ordenamiento basado en comparaciones favorito (Mergesort, Quicksort, Heapsort, etc.).

El problema es que comparar cuesta $O(m)$.

Costo total $= O(n \log n \cdot m) = O(N \log n)$.

¿Cómo mejoramos esta cota?

Aprovechando el tamaño del alfabeto

El tamaño del alfabeto es $\sigma = 26$. Podemos aprovechar eso.

Podemos interpretar los strings como "números" en base 26, y ordenarlos de la misma forma en que ordenabamos números con Radix sort.

Recordatorio en la pizarra, Radix sort utiliza sucesivamente Bucket Sort desde la posición menos significativa hasta la más significativa.

Este *approach* hace m pasadas lineales, por los n strings, por lo que tarda exactamente $\Theta(N)$.

El caso feo

Los strings tienen distintos largos, m_i, tales que $\sum_{i=1}^{n} m_i = N$.

Idea ingenua: padding. ¿Por qué no funciona?

Solución:

El caso feo

Los strings tienen distintos largos, m_i, tales que $\sum_{i=1}^{n} m_i = N$.

Idea ingenua: padding. ¿Por qué no funciona?

Solución: Primero ordenamos por largo de los strings, de más corto a más largo, y luego ordenaremos con rondas sucesivas de bucket sort desde los últimos caractéres de los strings más largos.

El caso feo

Solución:

- Bucket sort para ordenar los strings según su largo (la clave es el largo, ¿En qué rango está?)
- Comenzamos por los strings de largo máximo $m = m_n$. En cada ronda, consideraramos un largo siguiente, decrementando m, y mantenemos un puntero p a dónde comienzan los strings de ese largo.
- Por cada ronda ordenamos $A[p..n]$
- la ronda m toma tiempo $O(\sigma + k_m)$ donde k_m es la cantidad de strings de largo $\geq m$.
- $\sum_{m=1}^{m_n} k_m = N \longrightarrow$ funciona en $O(N)$. Note que un string de R caractéres será contado por la suma exactamente R veces, luego cada string suma su cantidad de caractéres al total.

Ejercicio de Control anterior: Stack en memoria secundaria

Suponga que utilizamos el siguiente procedimiento. En memoria principal mantenemos un buffer S, cuya capacidad es B, el tamaño de un bloque de memoria secundaria. Inicialmente S está vacío. Cada vez que insertamos un elemento en el stack lo hacemos en S, salvo cuando este está lleno. En tal caso primero vaciamos S en el primer bloque de memoria secundaria, y luego insertamos el elemento en el buffer ya vacío. De igual forma, para sacar un elemento del stack simplemente lo sacamos del buffer S, salvo cuando este se vacía por completo. En tal caso, leemos el primer bloque del stack en memoria secundaria, lo traemos al buffer S, y luego extraemos el primer elemento de S.

Demuestre que el costo amortizado por operación al utilizar este procedimiento (en términos de accesos a disco) es $\Omega(1)$.

Solución

Considere la siguiente secuencia de $2n + (B + 1)$ operaciones en el stack: Primero inserte $B + 1$ elementos, y luego consecutivamente n veces saque 2 elementos e luego inserte 2 elementos. Es claro que esta secuencia requiere de $n + 1$ accesos a disco. Por tanto, el número de accesos a disco para una secuencia de n operaciones siguiendo el procedimiento es $\Omega(n)$. Concluimos que el costo amortizado por operación es $\Omega(1)$ accesos a disco.

Ejercicio de Control anterior: Stack en memoria secundaria

Explique cómo el procedimiento anterior puede ser modificado para permitir la implementación de un stack en memoria secundaria en la cual el costo amortizado por operación es $O(1/B)$ accesos a disco.

Solución

Podemos simplemente extender el buffer S en un bloque; es decir, $|S| = 2B$. En este caso, cuando S se llena solo vaciamos un bloque a memoria secundaria, mientras que cuando se vacía solo pedimos un bloque a memoria secundaria. Después de cada lectura a disco siempre existen exactamente B elementos en S, y por tanto el procedimiento de seguro no accesará el disco durante las próximas B operaciones. Eso quiere decir que el costo total de n operaciones es de $O(n/B)$ accesos a disco, y luego el costo amortizado por operación es $O(1/B)$