Universidad de Chile
Departamento de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Auxiliar 7 - "Dominios Discretos"

Profesor: Pablo Barceló
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

26 de Mayo del 2017

P1. Rank

Sea B una secuencia de bits de largo n. Se define $RANK(B, i)$ como el número de bits en 1 en $B[1, i]$, es decir:

$$RANK(B, i) = \sum_{0<j\leq i} B[j],\ 1 \leq i \leq n$$

a) Construya una estructura que permita calcular $RANK(B, i)$ en tiempo constante y utilice $2n + o(n)$ **bits** de espacio.

b) Resuelva el mismo problema, esta vez utilizando $o(n)$ **bits** de espacio.

P2. Árbol de Sufijos

Muestre lo siguiente sobre el Árbol de Sufijos ST de un `String` T de largo n.

a) El espacio utilizado por ST es $\mathcal{O}(n)$.

b) ¿Como conocer el número de veces que un `String` P aparece como `substring` de T? ¿Cuánto cuesta? ¿Cómo lo puedo mejorar?

c) ¿Cómo saber cual es el `substring` más largo de T que aparece k veces? ¿Cual es el costo con ST?

d) ¿Cómo puedo construir el `Arreglo de Sufijos` de T? ¿Cual es el costo?

P3. Ordenar

Ordene n números enteros en el rango $[1, n^2]$ en $\mathcal{O}(n)$. Generalice su resultado para dominios de la forma $[1, n^k]$

Soluciones

P1. a) Una primera idea consiste en tener un arreglo de n elementos que en la posición i contenga el valor de $RANK(B, i)$, sin embargo, como el $RANK$ puede llegar hasta n necesito $\mathcal{O}(\log n)$ bits para representar uno de estos valores y en total $\mathcal{O}(n \log n)$. 😭

💡

Dividir B en partes de tamaño $\frac{\lceil \log n \rceil}{2}$ y guardar el $RANK$ de los últimos elementos de cada una de las partes. Esto toma $\frac{n}{\frac{\lceil \log n \rceil}{2}} \log n = \frac{2n}{\lceil \log n \rceil} \log n \leq 2n$ bits de espacio.

Ahora que tengo el $RANK$ cada $\frac{\lceil \log n \rceil}{2}$ solo debo identificar el $RANK$ más cercano y luego calcular el resto en tiempo $\mathcal{O}\left(\frac{\lceil \log n \rceil}{2}\right)$.

💡

Para reducir la complejidad a $\mathcal{O}(1)$ precalcularemos todos los $RANK$ de todas las secuencias de largo $\frac{\lceil \log n \rceil}{2}$. Es decir, tendremos en una tabla $T[w, r] = RANK(w, r)$ que ocupará espacio $2^{\frac{\lceil \log n \rceil}{2}} \cdot \frac{\lceil \log n \rceil}{2} \cdot \log n = \mathcal{O}(\sqrt{n}) \cdot \mathcal{O}(\log n) \cdot \log n = \mathcal{O}(\sqrt{n} \log^2 n) \in o(n)$ bits.🙏

b) El problema con la solución anterior es que el arreglo que guarda los $RANK$ de las partes ocupa $\mathcal{O}(n)$ bits de espacio.

💡

Aumentaremos el tamaño de las partes a $\frac{\log^2 n}{2}$, de este modo solo necesitamos $\frac{n}{\frac{\log^2 n}{2}} \log n = \frac{2n}{\log n} = o(n)$ bits de espacio.
Lamentablemente ya no podemos guardar todos los $RANK$ de todas las secuencias de tamaño $\frac{\log^2 n}{2}$ en espacio $o(n)$. 😭

💡

Dividimos cada parte en $\log n$ subpartes de tamaño $\frac{\log n}{2}$ cada una y guardamos el $RANK$ del final de cada una, pero relativo a la parte en la cual se encuentra esa subparte. De este modo necesitamos $\log n \cdot \frac{n}{\frac{\log^2 n}{2}} \cdot \log\left(\frac{\log^2 n}{2}\right) \leq \frac{4n \log \log n}{\log n} = o(n)$. 🙏

P2. a) Primero veamos que el Árbol de Sufijos tiene $\mathcal{O}(n)$ nodos. Esto es así pues todos los nodos, excepto quizás la raíz, tienen al menos 2 hijos. Luego como hay $n+1$ hojas (una por cada sufijo y una por \$) el árbol tiene a lo más $n+1$ nodos internos[1] y por lo tanto a lo más $2n + 2 = \mathcal{O}(n)$ nodos. Finalmente, para no aumentar el espacio, el árbol no guardará los substrings en sus aristas (ni en sus nodos), sino que los índices correspondientes en el texto original.

b) Un `String` P es un substring de T si y solo si P es un prefijo de un sufijo de T. Luego para saber el número de veces que P aparece como substring de T podemos bajar por el Árbol de Sufijos de T con P en $\mathcal{O}(|P|)$ y luego contar cuantas hojas hay a partir del nodo que se

[1] Esto puede ser demostrado por inducción

llegó en $\mathcal{O}(\#substrings)$. Para mejorar esto podemos almacenar el número de hojas bajo un nodo en cada nodo del árbol, reduciendo el costo de la operación anterior a $\mathcal{O}(|P|)$.

c) Lo pedido es equivalente a encontrar el nodo más profundo (en términos de número de letras) que tiene al menos k hojas, lo que se puede hacer recorriendo el árbol en $\mathcal{O}(|T|)$.👌

d) Si hacemos un recorrido inorden en el Árbol de Sufijos llegamos a las hojas en orden lexicográfico, como las hojas representan sufijos del texto estamos recorriendo los sufijos del texto en orden lexicográfico a partir de lo cual se puede obtener el Arreglo de Sufijos correspondiente.

P3. Si consideramos los números que estamos ordenando en base b, Counting-Sort toma $\mathcal{O}(n+b)$, luego si k es el número más grande a considerar, Radix-Sort debe hacer $\log_b k$ Counting-Sorts y por lo tanto toma $\mathcal{O}((n+b)\log_b k)$.

Si escogemos n como la base tenemos que:

- Ordenar enteros en el rango $[1, n^2]$ toma $\mathcal{O}((n+n)\log_n n^2) = \mathcal{O}(4n) = \mathcal{O}(n)$.
- Ordenar enteros en el rango $[1, n^k]$ toma $\mathcal{O}((n+n)\log_n n^k) = \mathcal{O}(2kn)$.