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

Auxiliar 9 - "Dominios Discretos y Preparación C2"

Profesor: Gonzalo Navarro
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

4 de Julio del 2018

P1. Reduciendo el espacio del vEB

El vEB es una estructura que permite insertar y eliminar números enteros del rango $[0 \dots u-1]$ en tiempo $\mathcal{O}(\log \log u)$, manteniendo un conjunto de n elementos. Además, permite responder *predecesor(x)* en tiempo $\mathcal{O}(\log \log u)$ también. Para lograrlo, los nodos del vEB almacenan los enteros *min*, *max* y *size*, además de $\sqrt{u}+1$ vEBs que trabajan sobre el rango $[0 \dots \sqrt{u}-1]$.

a) Muestre que el espacio ocupado por la estructura es $\mathcal{O}(u)$.

b) Muestre como disminuir el espacio a $\mathcal{O}(n)$. Mencione, si existen, inconvenientes de su solución.

P2. 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.

P3. PLCP

El arreglo $LCP[2 \dots n]$ ("longest common prefix") para un texto $T[1 \dots n]$ se define en función del arreglo de sufijos $A[1 \dots n]$ de T: $LCP[i] = lcp(T[A[i] \dots n], T[A[i-1] \dots n])$, donde $lcp(X, Y)$ es el largo del mayor prefijo común a X e Y. Por ejemplo, si $T[1 \dots 12] = abracadabra\$$, entonces $A[1 \dots 12] = \langle 12, 11, 8, 1, 4, 6, 9, 2, 5, 7, 10, 3\rangle$ y $LCP[2 \dots 12] = \langle 0, 1, 4, 1, 1, 0, 3, 0, 0, 0, 2\rangle$.

Definimos el arreglo permutado $PLCP[1 \dots n-1]$ como $PLCP[j] = LCP[A^{-1}[j]]$, es decir, los sufijos se recorren en orden de texto. En nuestro ejemplo, $PLCP[1 \dots 11] = \langle 4, 3, 2, 1, 0, 1, 0, 1, 0, 0, 0\rangle$.

a) Demuestre que $PLCP[j] \geq PLCP[j-1] - 1$.

b) Para calcular $PLCP$ se propone repetir para $j = 1$ hasta n: calcular directamente $PLCP[j] = lcp(T[j \dots n], T[A[A^{-1}[j]-1] \dots n])$ carácter a carácter, y luego pasar a $j+1$. La única gracia es que, por la propiedad del punto anterior, sabemos que los sufijos coincidirán en los primeros $l = PLCP[j-1] - 1$ símbolos, por lo que podemos calcular $PLCP[j] = l + lcp(T[j + l \dots n], T[A[A^{-1}[j]-1] + l \dots n])$.
Demuestre que el costo total de este procedimiento es $\mathcal{O}(n)$.

"My knee is almost back to normal. I am back in training."
Shawn Johnson

Soluciones

P1. Reduciendo el espacio del vEB

a) Dado que se ocupan $\sqrt{u}+1$ vEB que trabajan sobre el rango $[0\ldots\sqrt{u}-1]$ además de algunos campos, la ecuación de recurrencia correspondiente para el espacio queda $S(u) = \mathcal{O}(1) + (\sqrt{u}+1)S(\sqrt{u})$. Si hacemos el cambio de variable $u = 2^v$, nos queda:

$$\begin{aligned}
S(2^v) &= c + (2^{v/2}+1)S(2^{v/2}) \\
&= c + (2^{v/2}+1)(c + (2^{v/4}+1)S(2^{v/4})) \\
&\leq c(1 + (2^{v/2}+1) + (2^{v/2}+1)(2^{v/4}+1) + (2^{v/2}+1)(2^{v/4}+1)(2^{v/8}+1) + \ldots) \\
&= \mathcal{O}(2^v) \\
&= \mathcal{O}(u)
\end{aligned}$$

b) Nuestra primera idea será almacenar solo los vEB que contengan elementos, para hacer esto simplemente implementamos los vEB de bottom como una tabla de hash en vez de un arreglo.
Veamos ahora que si almacenamos los vEB de este modo, todos los nodos del vEB tienen un elemento en su campo *max* y por lo tanto hay $\mathcal{O}(n)$ nodos en el vEB (sin contar los nodos de los *top*), veamos ahora que el espacio ocupado por cada nodo se representa por la ecuación $S(u) = \mathcal{O}(1) + S(\sqrt{u}) = \mathcal{O}(\log\log u)$, por lo que el espacio total ocupado por el árbol es $\mathcal{O}(n\log\log u)$.

Si ahora recortamos la generación recursiva del vEB cuando $n = \log\log u$ el tamaño del árbol se reduce a $\mathcal{O}(n/\log\log u)$ nodos (pues las hojas ahora ocurren cuando $n = \log\log u$ y no pueden haber mas de $n/\log\log u$ de estas (son disjuntas)) , considerando nuevamente $\log\log u$ por nodo se tiene espacio $\mathcal{O}(n)$. Finalmente, al llegar a $n = \log\log u$, podemos usar un árbol balanceado en esos datos, que ocuparía $\mathcal{O}(\log\log u)$ espacio y respondería las consultas en $\mathcal{O}(\log\log\log u)$. Como hay $\mathcal{O}(n\log\log u)$ hojas en el vEB, el espacio total de los árboles balanceados sería $\mathcal{O}(n)$.

Los inconvenientes de esta solución vienen dados por la implementación para la tabla de hash. Si utilizamos técnicas de hashing clásico, acceder a la tabla nos tomara tiempo esperado constante, por lo que los tiempos de operaciones serán esperados y no de peor caso. Más adelante en el curso veremos una técnica que alcanza tiempo de acceso constante en el peor caso.

P2. Rank

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 $2^{\frac{\lceil \log n \rceil}{2}} \cdot \frac{\lceil \log n \rceil}{2} \cdot \log \frac{\lceil \log n \rceil}{2} = \mathcal{O}(\sqrt{n}) \cdot \mathcal{O}(\log n) \cdot \mathcal{O}(\log \log n) \in o(n)$ 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 $\log n \cdot \frac{n}{\frac{\log^2 n}{2}} \cdot \log\left(\frac{\log^2 n}{2}\right) \leq \frac{4n \log \log n}{\log n} = o(n)$. 🙏

P3. PLCP

a) Solo nos preocuparemos del caso $P[j-1] > 1$ pues sabemos que $PLCP \geq 0$.
Notemos que $PLCP[j-1]$ es el largo del prefijo más largo entre el $j-1$-ésimo sufijo y su anterior lexicográfico, es decir, si w es el $j-1$-ésimo sufijo, este coincide con su anterior lexicográfico, digamos w_a en los primeros $PLCP[j-1] > 1$ caracteres. Si a w y w_a le sacamos sus primeras letras obtendremos v y v_a ambos sufijos del texto que comparten las primeras $PLCP[j-1]-1$ letras y siendo v el j-ésimo sufijo. Finalmente, el anterior lexicográfico al j-ésimo sufijo v debe ser $\geq v_a$ en términos lexicográficos y por lo tanto el número de caracteres que coinciden entre v y su anterior lexicográfico debe ser $\geq PLCP[j-1]-1$, que es lo que se pedía demostrar.

b) De la primera ecuación se puede ver que el primer $PLCP$ se puede calcular en $\mathcal{O}(n)$. De la última ecuación se deduce que para el resto de $PLCPs$, calcular $PLCP[j]$ cuesta $\mathcal{O}(1) + PLCP[j] - PLCP[j-1]$, siendo lo último el costo del *lcp*. Finalmente, sumando los costos de todos los $PLCP$ entre 2 y $n-1$ la suma "telescopea" y queda $PLCP[n-1] - PLCP[1]$ lo que es $\mathcal{O}(n)$.