Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

Ayudantía 1

6 de abril de 2020

Quicksort

Pregunta 1

Demuestre que *QuickSort* es correcto y calcule su complejidad en el caso promedio.

Pregunta 2

Proponga una mejora para *QuickSort* con el fin de que, dado un *sub-array* de menos de α elementos, se utilice **otro algoritmo** que trabaje mejor con *items* con arreglos pequeños o semi-ordenados. Justifique el algoritmo escogido y determine la complejidad del algoritmo mejorado.

Heaps

Pregunta 3

Se tiene un arreglo de n números, desconocido a priori. Se te pide construir un algoritmo, utilizando un *Heap*, que entregue eficientemente el k-ésimo numero mayor, de los primeros m números del arreglo, con $k \leq m \leq n$. Considere que una vez que su algoritmo entrega una solución, el m puede aumentar, por lo que su algoritmo deberá actualizar de forma eficiente la respuesta.

SOLUCIÓN

Pregunta 1

Está en las diapositivas de esta ayudantía.

Pregunta 2

a. *InsertionSort* tiene complejidad promedio $\mathcal{O}(n^2)$. Si tenemos conocimiento a priori de que el array está semi-ordenado, es decir, cada elemento está a lo más a k posiciones de su lugar final, el número de *swaps* es bajo, por lo tanto tiende asintóticamente a $\mathcal{O}(n)$. Esta complejidad es mejor que el mejor caso de QuickSort de $\mathcal{O}(nlogn)$. Por otra parte, de estar usándose la variante de *QuickSort* que usa como pivote el primer o último elemento del arreglo, se tendría un comportamiento de $\mathcal{O}(n^2)$ al generarse particiones vacías o casi vacías en cada paso de la ejecución.

b. Consideramos que el **otro algoritmo** es *InsertionSort*, y que el arreglo original poseía n elementos. (mejorar explicación) Se llega a *sub-arrays* de k items cuando han pasado $log(n/k)$ llamadas recursivas. Finalmente, se hace *InsertionSort*, en donde cada elemento estará a lo más a k posiciones de su lugar final. Esta operación tiene complejidad $\mathcal{O}(kn)$, se tiene finalmente: (también se puede ver como n/k insertions de complejidad $k^2 => nk$)
$\mathcal{O}(nlog(n/k)+nk)$ Determinar este k de manera teórica es no trivial y requiere resolver $log(k) >= a*k$ (Función W de Lambert). De manera experimental es cuestión de ensayo y error.

Pregunta 3

Lo primero a considerar, es que si tenemos nuestro k-ésimo numero mayor (desde ahora k_p), cualquier número menor a este que consideremos no va a afectar nuestra decisión sobre k_p, por lo tanto, es sugerente almacenar solo los k mayores elementos.

La idea entonces, es construir un Min Heap de tamaño k con los k mayores elementos, donde la raíz será el menor de los k mayores, es decir, nuestro k_p. Para inicializar el algoritmo, recorremos los k primeros elementos, construyendo el Min Heap. A medida que avanzamos en el arreglo, verificamos si el número siguiente es mayor o menor que nuestro k_p.

- Si el número es menor que nuestro k_p, entonces no hacemos nada.
- Si el número es mayor que k_p, entonces el k_p se debe reemplazar. Esto lo hacemos intercambiándolo con el k_p actual, y le aplicamos $sift-down$ a este elemento, para que llegue a su lugar correcto en el Heap, y la nueva raíz será el nuevo k_p.

De esta forma, se mantienen las propiedades del Heap, y podemos seguir aplicando este algoritmo hasta llegar al final del arreglo.