Laboratorio 3 - Algoritmia y Complejidad

Alejandro Madrazo

16 de Agosto de 2018

1 Problema

Realicen un cambio a el algoritmo de heapsort donde no se destruya el heap pero aun asi logre ordenar el arreglo. Escribanlo en forma de pseudocodigo. Es el running-time de este algoritmo mejor, igual, o peor que el de heapsort?

Algorithm 1 HeapSort(A) ORIGINAL

```
1: Heapsort(A)
2: BUILD-MAX-HEAP(A)
3: for i = 0 A.lenght downto 2
4:     exchange A[1] and A[i]
5:     A.heapSize = A.heapSize-1
6:     MAX-HEAPIFY
```

Algorithm 2 HeapSort(A) NO HEAP DESTRUCTION

```
1: Heapsort2(A)
2: BUILD-MAX-HEAP(A)
3: for i = A.lenght downto 2
4:     for e = 1 in range A.height
5:         Swap A[2^(e-1)] and A[2^e]
6:     if 2^e < i
7:         Swap A[2^e] and A[2^i]
8:     else:
9:         Continue
```

El running time de este problema es mayor a HeapSort porque estoy corriendo un loop interrno al loop ya existente en el HeapSort Original. Este nuevo algoritmo, entrega un heap despues de cada iteracion del loop interior. Asi evitando el llamado a maxheapify.

2 Problema

Preguntas de QUICKSORT
1. Cual es el running time de Quicksort cuando todos los valores son iguales?

El running time de Quicksort cuando todos los valores son iguales sería el peor

de los casos. Es decir O(n^2). Esto se debe a que sin importar el elemento pivote que sea seleccionado. El algoritmo tendrá que atravesar todos los valores en el arreglo. Debido a ser todos iguales, cada recursión retornará una nueva partición.

Algorithm 3 QuickSort() RunningTime

```
1: Quicksort(A,p,r)
2: If p < r
3:     q = Partition(A,p,r)
4:     Quicksort(A,p,q-1)
5:     Quicksort(A,q+1,r)
```

2. Utilizen el pseudocodigo para ilustrar la operacion de partition en el siguiente arreglo.

$$Input : A = [13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11]$$

La operación de Partition sobre el arreglo 'A' inicialmente obtiene como input: p= 1 y r= 12 en base a esto el algoritmo úbica al primer y ultimo elementos en el arreglo.
Utiliza a 11 como el elemento pivote. Luego procede a ubicar todos los elementos menores a 11 del lado izquierdo de la "pared" imaginaria que recorre con el contador i. Asi ubica del lado izquiero todos los menores a 11(el pivote) y los intercambiar con el elemento que en ese momento se ubica en el indice del condador i.
Finalmente una vez todos los elementos menores a 11(el pivote) se encuentran al otro lado del indice, procede a intercambiar a 11 (el pivote) con el ultimo elemento despues de la pared en el indice i.
Esto nos da como resultado un arreglo ordenado en base al pivote 11. Entregando como resultado:

$$Input : A = [9, 5, 8, 7, 4, 2, 6, 11, 21, 13, 19, 12]$$

Puede consultar el "trace" manual en las imágenes dentro del repositorio.

3. Por que es mas utilizado Quicksort y no Heapsort sabiendo que el Worstcase running time de Heapsort es mejor?
El average case de HeapSort es O(nlogn), el cual es mejor que el worst case de QuickSort (n^2). Sin embargo en muchas de las aplicaciones de estos algoritmos, se suele preferir a QuickSort sobre Heapsort a pesar de ser menos eficiente en su worstcase.
Esto se debe a que aunque existe la posisbilidad de que Quicksort se tarde más, en promedio, son muy pocas las veces que esto sucede. Porque Quicksort no realiza intercambios entre números si no lo es necesario, en cambio Heapsort, siempre realiza por lo menos un intercambio en todos los elementos.
Entonces en la mayoría de los casos y los más favorables, Quicksort hace muchos menos intercambios entre números que Heapsort, lo cual lo hace más eficiente en promedio. Teniendo en cuenta que las lecturas y escrituras a memoria son

los casos más costosos. Al ahorrar en la mayoría de los casos intercambios el algoritmo es mucho más eficiente. Esto es gracias al pivote. Pues el pivote no exige que todos los números sean intercambiados, solo los menores a el.

3 Problema

Realicen una implementacion de Quicksort en Python desarrollando su propio algoritmo de partición.

Utilizenlo para ordenar este arreglo.

$$Input : unsortedArray = [5, 10, 15, 32, 55, 21, 40, 2, 3, 76, 89, 28, 9, 7]$$

Respuesta:
Quicksort.py
También se puede encontrar la respuesta como un JupyterNotebook como: **QuickSortNoteBook.ipynb**