Algoritmos de ordenamiento

Matemáticas Computacionales

Yarethzi Giselle Bazaldúa Parga

31 de agosto de 2017

Resumen

Un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden. Hay métodos muy simples de implementar que son útiles en los casos en dónde el número de elementos a ordenar no es muy grande. Por otro lado hay métodos sofisticados, más difíciles de implementar pero que son más eficientes en cuestión de tiempo de ejecución.

Insertion:

Consta de comparar el elemento n+1 con el término n y con todos los demás elementos ya ordenados, para así terminar con todos los términos en su sitio. Este algoritmo de ordenamiento tiene una complejidad de n^2. Pseudocódigo:

```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

Bubble:

Éste algoritmo revisa cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. La lista es revisada varias veces hasta que ya no necesite más cambios, eso quiere decir que la lista ya está ordenada. Este algoritmo de ordenamiento cuenta con una complejidad de n^2. Pseudocódigo:

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)

```

Quicksort:

Se encarga de tomar el primer elemento del arreglo, ponerlo como “media” y a partir de éste número ir acomodando los menores con respecto a él a la izquierda y los mayores a la derecha, esto se repite la cantidad de veces necesaria para que la lista esté ordenada, este algoritmo tiene una complejidad de $n.\log n$. Pseudocódigo:

```
def quickSort(alist):
   quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
   if first<last:

       splitpoint = partition(alist,first,last)

       quickSortHelper(alist,first,splitpoint-1)
       quickSortHelper(alist,splitpoint+1,last)


def partition(alist,first,last):
   pivotvalue = alist[first]

   leftmark = first+1
   rightmark = last

   done = False
   while not done:

       while leftmark <= rightmark and \
               alist[leftmark] <= pivotvalue:
           leftmark = leftmark + 1

       while alist[rightmark] >= pivotvalue and \
               rightmark >= leftmark:
           rightmark = rightmark -1

       if rightmark < leftmark:
           done = True
       else:
           temp = alist[leftmark]
           alist[leftmark] = alist[rightmark]
           alist[rightmark] = temp

   temp = alist[first]
   alist[first] = alist[rightmark]
   alist[rightmark] = temp


   return rightmark

alist = [54,26,93,17,77,31,44,55,20]
quickSort(alist)
print(alist)
```

Selection:

Busca el mínimo elemento entre una posición i y el final de la lista e intercambia ese mínimo elemento con el de la posición i. Este algoritmo mejora ligeramente el algoritmo *bubble*. En el caso de tener que ordenar un vector de enteros, esta mejora no es muy sustancial, pero cuando hay que ordenar un vector de estructuras más complejas, la operación *insercion* sería más costosa en este caso. tiene una complejidad de n^2.

Pseudocódigo:

```
def selectionSort(alist):
   for fillslot in range(len(alist)-1,0,-1):
       positionOfMax=0
       for location in range(1,fillslot+1):
           if alist[location]>alist[positionOfMax]:
               positionOfMax = location

       temp = alist[fillslot]
       alist[fillslot] = alist[positionOfMax]
       alist[positionOfMax] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)
```

Conclusiones:

Cuando se trabaja con grandes cantidades de elementos en un arreglo, el algoritmo de ordenamiento más conveniente es *quicksort*, si la longitud del arreglo puede considerarse con una cantidad “pequeña” de datos, cualquiera de los otros 3 algoritmos trabajan igual, ya sea *bubblesort*, s*electionsort* o *insercionsort*. Los 4 son igual de eficientes, el más amigable, a mi parecer es el inserción pero depende del uso que cada persona le dé.