Matemáticas computacionales

(Reporte de algoritmo de ordenamiento)

Alumno: Isaac Emanuel Segovia Olay M. 1748957

Maestro: Lic. José Anastacio Hernández Saldaña

01-Septiembre-2017

Este es un reporte de los algoritmos que hemos visto en clases y la explicación de cada uno de ellos junto con un ejemplo para tener una completa información del tema.

Merge

Por lo que entendí en la clase de merge es que el uso del algoritmo de ordenamiento Merge Sort en phyton es que el algoritmo funciona cuando el vector es dividido a la mitad. Cada una de esas dos mitades es a su vez dividida a la mitad. Se repite el procedimiento sucesivamente hasta que ya no pueda dividirse más.
Después se procede a volver a unir cada mitad para formar la unidad de donde fueron generadas sin embargo ahora se procede a que la mitad izquierda tenga los números menores y la otra mitad los números mayores. Se repite este procedimiento hasta formar la lista original.

Un ejemplo de esto es el siguiente:

```
#Declaracion de Funciones
 def mergesort(A):
    if(len(A) <= 1):
        return A;

    izq=[];
    der=[];
    medio=len(A)/2;
    for i in range(0,medio):
        izq.append(A[i]);
    for i in range(medio,len(A)):
        der.append(A[i]);

    izq=mergesort(izq);
```

```
    der=mergesort(der);
 return merge(izq,der);


def merge(izq,der):
    arreglo=[];
    i=0;
    j=0;
    while(i < len(izq) and j < len(der)):
        if(izq[i] <= der[j]):
            arreglo.append(izq[i]);
            i=i+1;
        else:
            arreglo.append(der[j]);
            j=j+1;
    while(i < len(izq)):
        arreglo.append(izq[i]);
        i=i+1;
    while(j < len(der)):
        arreglo.append(der[j]);
        j=j+1;
         return arreglo;
 #Programa Principal
A=[6,5,3,1,8,7,2,4];
print A;
A=mergesort(A);
print A;
```

Así es como acomodamos la lista utilizando Merge.

Quicksort

Cuando mis compañero me explicaron sobre quicksort es que es el algoritmo de ordenación más rápido conocido, su tiempo de ejecución promedio es O(n log (n)), siendo en el peor de los casos O(n2), caso altamente improbable. El hecho de que sea más rápido que otros algoritmos de ordenación con tiempo promedio de O(n log (n)) viene dado por que QuickSort realiza menos operaciones ya que el método utilizado es el de partición.

Una explicación de este algoritmo sacado de internet es que se elige un elemento v de la lista L de elementos al que se le llama pivote.

Se particiona la lista L en tres listas:

L1 - que contiene todos los elementos de L menos v que sean menores o iguales que v.

L2 - que contiene a v.

L3 - que contiene todos los elementos de L menos v que sean mayores o iguales que v.

Se aplica la recursión sobre L1 y L3.

Se unen todas las soluciones que darán forma final a la lista L finalmente ordenada. Como L1 y L3 están ya ordenados, lo único que tenemos que hacer es concatenar L1, L2 y L3.

Un ejemplo es:

```
def quicksort(lista,izq,der):
    i=izq
    j=der
    x=lista[(izq + der)/2]

    while( i <= j ):
        while lista[i]<x and j<=der:
            i=i+1
        while x<lista[j] and j>izq:
            j=j-1
        if i<=j:
            aux = lista[i]; lista[i] = lista[j]; lista[j] = aux;
            i=i+1;  j=j-1;


        if izq < j:
        quicksort( lista, izq, j );
    if i < der:
        quicksort( lista, i, der );

def imprimeLista(lista,tam):
    for i in range(0,tam):
        print lista[i]

def leeLista():
    lista=[]
    cn=int(raw_input("Cantidad de numeros a ingresar: "))
```

```
    for i in range(0,cn):
        lista.append(int(raw_input("Ingrese numero %d : " % i)))
    return lista

A=leeLista()
quicksort(A,0,len(A)-1)
imprimeLista(A,len(A))
```

Así es como acomodamos la lista utilizando Bubble.

Bubble

Este algoritmo me tocó junto con mi equipo en explicar a la clase y lo que sé ahora es que funciona comparando en el vector dos elementos adyacentes, si el segundo elemento es menor al primero entonces son intercambiados, en caso contrario se evalúa el segundo elemento y el siguiente elemento adyacente a su izquierda.

Ya cuando esté terminada la primera iteración se vuelve a realizar el mismo procedimiento sin embargo en cada iteración hecha, un elemento del vector, de derecha a izquierda, ya no es evaluado ya que contiene el número mayor de cada iteración.

El algoritmo terminará una vez que el número de iteraciones sea igual al número de elementos que contenga el vector de números.

Un ejemplo es:

```
#Declaración de Funciones
 def burbuja(A):
    for i in range(1,len(A)):
        for j in range(0,len(A)-i):
            if(A[j+1] < A[j]):
                aux=A[j];
                A[j]=A[j+1];
                A[j+1]=aux;
    print A;
 #Programa Principal
A=[6,5,3,1,8,7,2,4];
print A
burbuja(A);
```

Así es como acomodamos la lista utilizando Bubble.

Selection

Por último, en este algoritmo que es selection lo que hace es que la función principal, es la encargada de recorrer la lista, ubicando el mayor elemento al final del segmento y luego reduciendo el segmento a analizar.

También busqué más información y encontré que busca el mayor de todos los elementos de la lista.

Poner el mayor al final (intercambiar el que está en la última posición de la lista con el mayor encontrado).

Buscar el mayor de todos los elementos del segmento de la lista entre la primera y la anteúltima posición.

Poner el mayor al final del segmento (intercambiar el que está en la última posición del segmento, o sea anteúltima posición de la lista, con el mayor encontrado).

Se termina cuando queda un único elemento sin tratar: el que está en la primera posición de la lista, y que es el menor de todos porque todos los mayores fueron reubicados.

Un ejemplo es:

```
def selectionsort(lista,tam):
    for i in range(0,tam-1):
        min=i
        for j in range(i+1,tam):
            if lista[min] > lista[j]:
                min=j
        aux=lista[min]
        lista[min]=lista[i]
        lista[i]=aux


def imprimeLista(lista,tam):
    for i in range(0,tam):
        print lista[i]


def leeLista():
    lista=[]
    cn=int(raw_input("Cantidad de numeros a ingresar: "))

    for i in range(0,cn):
        lista.append(int(raw_input("Ingrese numero %d : " % i)))
    return lista

A=leeLista()
selectionsort(A,len(A))
imprimeLista(A,len(A))
```

Así es como acomodamos la lista utilizando Selection.