Laboratorio # 1

Juan Diego Sique Martínez

Julio 2018

1. Búsqueda lineal

Input: arreglo «array» y caracter a buscar «operador»

```
busqueda(arrray, operador) {
    posicion = null
    for(i = 1, i = len(array), i++) {
        if(array[i] = operador) {
            posicion = i
            break
        }
    }
    return posicion
}
```

La Invariante de ciclo consiste en que el valor de la variable **i** nos indica la posición o lugar en el arreglo donde no se encuentra nuestro caracter deseado, y al momento de encontrarla rompe el ciclo retornándonos dicho valor **i**.

Por ejemplo, supongamos que en el arreglo de caracteres **George Frideric Händel** deseamos buscar el caracter **ä**. En el algoritmo se comenzará comparando de manera lineal cada uno de los caracteres. La posición del arreglo que estamos comparando la indica la variable **i**. Al inicializarse en **1** comenzará por la primera letra: **G**. De manera que secuencialmente evaluará todas las posiciones hasta llegar a encontrar el caracter deseado. En este caso es el caracter en la posición **18**. La Invariante de ciclo consiste en que con cada iteración ya sea **2**, **9** o **15** todos los caracteres en posiciones previas a **i** no contienen el caracter deseado de lo contrario se romperá en ciclo gracias a la instrucción **break** y nos devolverá el valor de la posición en donde está **ä**. Esta invariante de ciclo se mantiene en cada una de las iteraciones del ciclo hasta finalizar el procedimiento.

2. Multiplicación de matrices

Input: matriz A y matriz B Output: matriz C

Complejidad	Instrucción de código
n	`For i from 1 to n:`
n × p	`  For j from 1 to p:`
n × p	`    Let sum = 0`
n × p	`    For k from 1 to m:`
n × p × m	`      Set sum <- sum + A[i][k] * B[k][j]`
n × p	`    Set C[i][j] <- sum`
1	`return C`

Como pudimos apreciar el tiempo es $O(n \times p \times m)$ ya que se recorren tres ciclos que corresponden a las dos dimensiones de las matrices y la tercera a una suma acumulativa para poder obtener el resultado de la multiplicación. Si las tres matrices poseen el mismo número de elementos podríamos decir que la multiplicación consecutiva de n por sí misma tres veces da como resultado $O(n^3)$.

3. Ordenación por «Bubble-sort»

Algorithm 1 Bubble sort algorithm

```
S is an array of integer
for i in 1 : length(S) − 1 do
  for j in (i + 1) : length(S) do
    if S[i] > S[j] then
      swap S[i] and S[j]
    end if
  end for
end for
```

3.1. ¿Cuál es el «worst-case running time» de este algoritmo?

Es apreciable el ciclo anidado donde el primer **for** y sus instrucciones incluidas se ejecutan al menos $n-1$ veces. La complejidad del algoritmo crece cuando el segundo **for** se ejecuta también $n-1$ veces resultando $(n-1)^2$. Por lo tanto en el peor de los casos tendrá una complejidad del tipo $O(n^2)$.

3.2. ¿Cómo se compara a el «running time» de «insertion sort»?

3.2.1. Mejor caso

Al hablar del análisis del mejor caso en los algoritmos de ordenamiento se utiliza como parámetro de entrada una lista previamente ordenada.

En el caso de «bubble sort» utilizamos como medida de crecimiento la cantidad de intercambios o sustituciones que se realizan y las veces que se recorre el arreglo. Como la lista ya está ordenada el ítem en la posición **i** siempre será menor al que está en la posición **j**, por lo tanto la instrucción que indica el intercambio (**swap**), jamás se llevará a cabo, pero siempre sigue recorriendo el arreglo de manera anidada ya que la instrucción **for** no se encuentra condicionada, y sólo recorrer el algoritmo no le bastará para indicar que la lista ya está ordenada, es decir que su tiempo de ejecución es $O(n^2)$.

En «insertion sort» sucede algo similar, pero la condicional no se llega a cumplir porque la lista ya se encuentra ordenada y el algoritmo recorre la lista una vez, por lo que el tiempo que le tomará a nuestro algoritmo determinar si la lista está ordenada se encuentra en función del número de elementos en la misma, es decir $O(n)$.

3.2.2. Peor caso

Se compara en la presencia de un ciclo anidado que al ejecutarse $n - 1$ de manera consecutiva producen una complejidad de tipo $O(n^2)$.

La diferencia más notable entre ambos es su Invariante de ciclo puesto que «insertion-sort» se encarga de dejar los primeros elementos de la lista ya ordenados, mientras que «bubble-sort» lo hace dejando los últimos elementos de la lista ordenados del mayor al menor de forma ascendente.