Algoritmia y Complejidad, Laboratorio 1

Diego Quan, UFM

August 2, 2018

1 Problema 1: Búsquedad Lineal

Algorithm 1: Búsquedad Lineal

Input : Secuencia de N numeros $A = [a1, a2, a3..., an]$
Output: Indice i tal que $v = A[i]$ o null si no se encuentra el valor
Data: $n = Secuencia$, $v = Valor$, $i = Contador$

```
1  Function Lineal(n: list):
2      for i from 1 to len(n) do
3          if n[i] == v then
4              print i;
5              return i;
6          else
7              return null;
8          end
9      end
10 end
```

El *loop invariant* en este caso es: todo elemento que se encuentra al lado izquierdo del valor que se está verificando cumple con la condición de $n[0 : i - 1] =!v$. El tiempo de ejecución en este caso sería de $O(n)$.

2 Problema 2: Multiplicación de matrices

Algorithm 2: Matrices

Input : Matriz A(n x m) y matriz B(m x p)
Output: Matriz C(n x p)

```
1 for i from 1 to n do
      // n
2     for j from 1 to p do
          // n * p
3         Let sum = 0
4         for k from 1 to m do
              // n * p * m
5             Set sum ← sum + A[i][k] * B[k][j]
6         end
7         Set C[i][j] ← sum
8     end
9 end
```

El algoritmo requiere dos matrices para ser ejecutado, entonces requiere finalemnente tres variables para ser multiplicadas. $n = columnas$ de la matriz A, $m = lineas$ de la matriz A, $m = columnas$ de la matriz B y finalmente $p = lineas$ de la matriz B. Esto implicada un tiempo de ejecución de $O(n^3)$ debido a que todos estos se pueden representar como una variable n en relación a la complejidad O.

3 Problema 3: Comparación de algoritmos

Algorithm 3: Bubble Sort, Figura (a)

```
1 S is an array of integers
2 for i from 1 to length(S-1) do
3 |   for j from (i+1) to length(S) do
4 |   |   if S[i] > S[j] then
5 |   |   |   swap S[i]andS[j]
6 |   |   end
7 |   end
8 end
```

Algorithm 4: Insertion Sort, Figura (b)

```
1 for j←to length[A] do
2 |   key ← A[j]
3 |   i = j - 1
4 |   while i > 0 and A[i] > key do
5 |   |   A[i + 1] = A[i]
6 |   |   i = i − 1
7 |   |   A[i + 1] = key
8 |   end
9 end
```

Ambos algoritmos (a),(b) tienen un a complejidad de $O(n^2)$ debido a los loops anidados que cada uno tiene, pero hay una diferencia clave entre estos dos algoritmos. El algoritmo de *Insertion Sort* tiene un promedio de $i/2$ iteraciones, es decir puede parar el segundo ciclo si es necesario, mientras que el *Bubble Sort*, tiene que recorrer todos los elementos.

En el caso de **Worst Case** ambos algoritmos tienen una complejidad de $O(n^2)$ mientras que el **Best Case** tiene el *Insertion Sort* tiene una complejidad de $O(n)$ en cuestión de comparaciones y $O(1)$ en el caso de *swaps*. *Bubble Sort* tiene una complejidad de **Worst Case** de $O(n^2)$ y en **Best Case** tiene $O(n)$ debido a que tiene que recorrer todo el alegro.