



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Ingeniería en Inteligencia Artificial, Análisis y Diseño de Algoritmos
Sem: 2024-1, 3BV1, Práctica 3, 18 de octubre de 2023

PRÁCTICA 3: FUNCIONES RECURSIVAS VS ITERATIVAS.

Catonga Tecla Daniel Isaí 1, Olguin Castillo Victor Manuel 2.

daniel9513importantes@gmail.com₁, manuelevansipn@gmail.com₂

Resumen:

Palabras Clave: Algoritmo, Big O, C++, Recursividad

1. Introducción

Un algoritmo es un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema [2]. Los algoritmos son de suma importancia, estos ayudan a la resolución de problemas muy grandes o complejos como problemas matemáticos y científicos, por lo tanto, los algoritmos son fundamentales en varias áreas de las ciencias o ingeniería.

En programación, las operaciones pueden ser manejadas por recursividad o iteraciones. Recursividad es una función que se llama a sí misma, directa o indirectamente. Una de las estructuras fundamentales de programación son los bucles, las cuales forman parte de las estructuras de la mayoría de los lenguajes, los bucles sirven para los algoritmos iterativos. [1]

El análisis de algoritmos es otra parte fundamental por lo que los algoritmos tienen que ser los más eficientes posibles en complejidad temporal y complejidad espacial. En cuestión de complejidad temporal algunos algoritmos pueden requerir años en resolver un solo problema, por lo que, no es suficiente para la práctica. , y se opta por otros algoritmos que resuelvan el problema que presentan de una forma más eficaz. Por otra parte, una máquina no cuenta con recursos infinitos de memoria o espacio de almacenamiento, por lo que, existen algoritmos que ocupan mucho espacio de memoria y esto puede ocasionar problemas para equipos que no cuentan con el espacio suficiente, entonces se tiene que analizar también lo que es la complejidad espacial para tratar de ocupar la menor cantidad de memoria posible. Ambas complejidades son importantes, pero es más importante la complejidad temporal ya que las empresas pueden comprar más memorias de almacenamiento, pero no pueden comprar tiempo, por lo que algunas empresas optan sacrificar complejidad espacial por una complejidad temporal más eficiente.

Algunos algoritmos iterativos se pueden hacer de forma recursiva y viceversa, lo importante es conocer cuál es más eficaz con respecto a la otra implementación, por lo que, es importante conocer la complejidad de cada algoritmo en la forma recursiva o iterativa y compararlas para saber que algoritmos implementar en un caso práctico.

2. Conceptos Basicos

- **Algoritmo.** Un algoritmo es una secuencia de pasos lógicos que son precisos, ordenados y finitos que se ocupan para resolver un problema deseado [3].
- **Análisis de algoritmos** Es un proceso de evaluación donde conoceremos el rendimiento y la eficiencia de un algoritmo. Se evaluará el consumo de tiempo y de recursos computacionales que requiere el algoritmo para ser ejecutado con diversos datos de entrada, y esto determinará su complejidad [3].
- **Cota superior asintótica.** Es una función que delimita por la parte superior a otra función a medida que la entrada de la función delimitada crece.
- **Cota inferior asintótica.** Es una función que delimita por la parte inferior a otra función a medida que la entrada de la función delimitada crece.
- **Notacion O.** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota superior asintótica en el peor caso de ejecución de un algoritmo [3].

$$O(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c \text{ constante positiva y } n_0 \notin \mathbb{N} :$$

$$f(n) \leq cg(n), \forall n \geq n_0\}$$

- **Notacion Theta "Θ"** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota superior y una cota inferior, esto nos da una idea más precisa del comportamiento y complejidad del mismo [3].

$$\Theta(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c_1, c_2 \text{ constantes positivas, } n_0 :$$

$$0 < c_1g(n) \leq f(n) \leq c_2g(n), \forall n \geq n_0\}$$

- **Notacion Omega "Ω"** Esta notación se ocupa para describir la complejidad de un algoritmo definiendo una cota inferior asintótica en el peor caso de ejecución de un algoritmo [3].

$$\Omega(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c \text{ constante positiva y } n_0 :$$

$$0 < cg(n) \leq f(n), \forall n \geq n_0\}$$

- **Análisis a posteriori.** Es una evaluación que se realiza de forma empírica, donde los resultados se obtienen con la ejecución del algoritmo y la medición del tiempo y recursos computacionales que requiere [3].
- **Análisis a priori.** Es una evaluación que se realiza de forma teórica, donde los resultados se pueden obtener con el conteo de operaciones y/o análisis matemático que en base a sus fórmulas se obtiene su complejidad algorítmica [3].
- **Funciones recursivas.** En la programación las funciones recursivas son aquellas que durante su proceso se invocan así mismas.

```
Suma_Recursiva(n):
  If n == 1
    return 1
  Else
    return n + sumaRecursiva(n - 1)
```

- **Funciones iterativas.** En la programación las funciones iterativas son aquellas que se ejecuta en un ciclo n número de veces y existe una condición de validación en cada iteración que controla si se itera una vez más o finaliza en ciclo

```
Suma_iterativa(n):
  resultado = 0
  for i = 0 to n do:
    resultado += i
  return resultado
```

- **Pseudocódigo de Búsqueda iterativo.** El algoritmo de búsqueda, retorna el índice del valor que se busca dividiendo el arreglo en 3 partes por cada operación que se realiza hasta encontrar el valor o en caso contrario retorna -1 si no existe el valor en el arreglo.

```
BusquedaIterativa(Arr[1,2,...,n],valor):
  low = 0
  high = longitud del vector - 1
  i entero
  j entero
  while low <= high do:
    i = low + (high-low)/3
    j = low + 2*(high-low)/3
```

```

    if Arr[j]<valor do:
        low = j+1
    else if Arr[i]>valor do:
        high = i-1
    else if Arr[i]<valor and Arr[j]>valor do:
        low = i+1
        high = j-1
    else if Arr[i] == valor do:
        return i
    else if Arr[j] == valor do:
        return j
    return -1

```

- **Pseudocódigo de Búsqueda recursiva.** El algoritmo de búsqueda recursivo se llama así mismo varias veces, retorna el índice del valor que se busca dividiendo el arreglo en 3 partes por cada operación que se realiza hasta encontrar el valor o en caso contrario retorna -1 si no existe el valor en el arreglo.

```

BusquedaRecursiva(Arr[1,2,...,n],low,high,valor):
if low <= high do:
    i = low + (high-low)/3
    j = low + 2*(high-low)/3

    if Arr[j]<valor do:
        return BusquedaRecursiva(Arr,j+1,high,valor)
    else if Arr[i]>valor do:
        return BusquedaRecursiva(Arr,low,i-1,valor)
    else if Arr[i]<valor and Arr[j]>valor do:
        return BusquedaRecursiva(Arr,j+1,i-1,valor)
    else if Arr[i] == valor do:
        return i
    else if Arr[j] == valor do:
        return j
    return -1

```

3. Experimentación y Resultados

4. Conclusiones

Texto ejemplo, conclusión general.

Conclusiones Catonga Tecla Daniel Isaí 1

Texto ejemplo.

Conclusiones Olguin Castillo Victor Manuel 2

Texto ejemplo.

5. Bibliografía

Referencias

- [1] Gary Briceño. *Recursividad o Iteración*. 2013. URL: <https://www.clubdetecnologia.net/blog/2013/recursividad-iteracion/>.
- [2] REAL ACADEMIA ESPAÑOLA. *Diccionario de la lengua española*, 23.^a ed. 2023. URL: <https://dle.rae.es/algoritmo>.
- [3] R. C. t. Lee. *Introducción al diseño y análisis de algoritmos: un enfoque estratégico*. Ed. por Mc Graw Hill. 2014.