* Alumno: Alejandro Gustavo Farfán – alejofar9@gmail.com

* Materia: Programación I

Índice

1. Introducción

La búsqueda y el ordenamiento son operaciones fundamentales en computación para encontrar o organizar datos. Buscare explicar el marco teórico, metodología utilizada, resultados obtenidos y su conclusión.

2. Marco Teórico

Tema: Algoritmos de Búsqueda y Ordenamiento.

.Algoritmo búsqueda lineal y binaria:

Búsqueda lineal: Es el algoritmo de búsqueda más simple, que recorre cada elemento de una lista hasta encontrar el elemento deseado. Es fácil de implementar, pero puede ser lento para listas grandes.

Búsqueda binaria: Es un algoritmo de búsqueda eficiente para listas grandes y ordenadas, encuentra un elemento en una lista ordenada dividiendo repetidamente a la mitad.

.Algoritmo por ordenamiento:

-Bubble Sort (Ordenamiento por burbuja): Es de ordenamiento simple. Funciona comparando cada elemento de la lista con el siguiente elemento y luego intercambiando los elementos si están en el orden incorrecto.

-Selection Sort (Ordenamiento por selección): Es de ordenamiento simple. Encuentra el elemento más pequeño de la lista y lo coloca al inicio. Repite el proceso con el resto de la lista hasta que todos los elementos de la lista estén ordenados.

-Insertion Sort (Ordenamiento por inserción): Funciona insertando cada elemento de la lista en su posición correcta en la lista ordenada.

-Quick Sort (Ordenamiento rápido): Es un algoritmo de ordenamiento eficiente que funciona dividiendo la lista en dos partes y luego ordenando cada parte de forma recursiva.

3. Caso Práctico

Buscar un elemento en una lista con n elementos

```python
def busqueda_lineal(lista, objetivo):

    #Busca un elemento en una lista (no necesita estar ordenada)

    for i in range(len(lista)):
        if lista[i] == objetivo:
            return i # Devuelve el indice si lo encuentra
    return -1  #No  encontrado

import time
import random
tamaño_lista = 50000
lista = random.sample(range(1, 100000), tamaño_lista)  # Lista de números únicos


# Elegir un objetivo aleatorio
objetivo = random.choice(lista)

# Medir tiempo de búsqueda lineal
inicio_lineal = time.time()
resultado_lineal = busqueda_lineal(lista, objetivo)
fin_lineal = time.time()
tiempo_lineal = fin_lineal - inicio_lineal
print(f"Búsqueda Lineal: Resultado = {resultado_lineal}, Tiempo = {tiempo_lineal:.6f} segundos")

```

```
Búsqueda Lineal: Resultado = 11229, Tiempo = 0.001611 segundos
```

Algoritmo por ordenamiento se busca cual es el más eficiente y rápido

Ejemplo:

```python
"""2. Algoritmo de Ordenamiento
a. Ordenamiento de Burbuja (O(n²))

"""
def ordenamiento_burbuja(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n-i-1):
            if lista[j] > lista[j+1]:
                lista[j], lista[j+1] = lista[j+1], lista[j]  # Intercambio
    return lista

# Ejemplo:

lista = [64, 34, 25, 12, 22, 11, 90]

print(ordenamiento_burbuja(lista.copy()))  # [11, 12, 22, 25, 34, 64, 90]

```

```
[11, 12, 22, 25, 34, 64, 90]
```

```python
""" b. Ordenamiento por Seleccion (O(n²))
"""
def ordenamiento_seleccion(lista):
    n = len(lista)
    for i in range(n):
        min_idx = i   # Encuentra el minimo en la lista que desordenada
        for j in range(i+1, n):
            if lista[j] < lista[min_idx]:
                min_idx = j

        #Intercambia el minimo con el primer elemento desordenado.

        lista[i], lista[min_idx] = lista[min_idx], lista[i]
    return lista

# Ejemplo:
lista = [64, 34, 25, 12, 22, 11, 90]

print(ordenamiento_seleccion(lista.copy()))  # [11, 12, 22, 25, 34, 64, 90]


```

```
[11, 12, 22, 25, 34, 64, 90]
```

```python
""" c. Ordenaminto por insercion (O(n²))
"""

def ordenamiento_insercion(lista):
    for i in range(1, len(lista)):
        clave = lista[i]  #Elemento a insertar
        j = i-1
        #Mueve los elementos que la clave hacia la derecha
        while j >=0 and clave < lista[j]:
            lista[j+1] = lista[j]
            j -= 1
        lista[j+1] = clave #Inserta la clave en su posicion correcta
    return lista

# Ejemplo:
lista = [64, 34, 25, 12, 22, 11, 90]

print(ordenamiento_insercion(lista.copy()))  # [11, 12, 22, 25, 34, 64, 90]

```

```
[11, 12, 22, 25, 34, 64, 90]
```

```python
""" d. QuickSort (O(n long n) promedio)

"""
def quicksort(lista):
    if len(lista) <= 1:
        return lista
    pivote = lista[len(lista)//2] #Elemento del medio
    izquierda = [x for x in lista if x < pivote]
    medio = [x for x in lista if x == pivote]
    derecha = [x for x in lista if x > pivote]
    return quicksort(izquierda) + medio + quicksort(derecha)

# Ejemplo:
lista = [64, 34, 25, 12, 22, 11, 90]

print(quicksort(lista.copy()))  # [11, 12, 22, 25, 34, 64, 90]



```

```
[11, 12, 22, 25, 34, 64, 90]
```

4. Metodología Utilizada

-Curso intensivo de Python" de Eric Matthes

-Documentación: Python Official Docs

-Tutoriales en Youtube sobre Algoritmos de Búsqueda y Ordenamiento.

-Power shell

5. Resultados Obtenidos

-Se buscó información teórica en documentación confiable.

-En algoritmo búsqueda lineal y binaria, se verifica que la búsqueda binaria es más rápida que la lineal cuando se trata de una lista grande, localizó de forma eficiente el elemento.

Resultado:

Búsqueda Lineal: 0.002346 segundos

Búsqueda Binaria: 0.000021 segundos

-En algoritmo ordenamiento, se verifica que Quick Sort es más eficiente en promedio en listas grandes. Busca el elemento en forma más rápida.

Resultado:

Burbuja: 8.7328 segundos

Selección: 3.6328 segundos

Inserción: 4.0553 segundos

QuickSort: 0.0269 segundos

-Se registró los resultados y validación de su funcionalidad.

6. Conclusiones

Este algoritmo ayuda a organizar datos y a localizarlos.

Se pudo observar que los diferentes tipos de búsqueda, el binario requiere preordenamiento pero es exponencialmente más rápida que la lineal.

En los tipos de ordenamiento el Quick Sort es más eficiente y entendible para listas grandes que los demás.

Se pudo apreciar la importancia de seleccionar el algoritmo que sea más rápido para realizar una actividad de manera más óptima.

7. Bibliografía

- Python Software Foundation. (2023). Python 3.12 Documentation. https://docs.python.org/3/

- Cormen, T. H. (2009). Introduction to Algorithms. MIT Press.