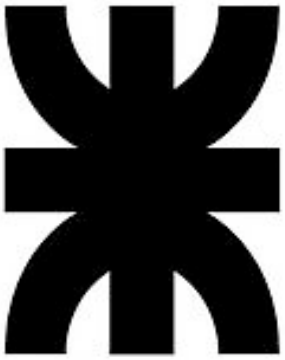
TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

UNIVERSIDAD TECNOLOGICA NACIONAL



PROGRAMACIÓN 1

TRABAJO PRACTICO INTEGRADOR

“Algoritmos de Búsqueda y Ordenamiento”

Docente: Ariel Enferrel

Tutor: Maximiliano Sar Fernández

Alumnos:

- **Juan Ignacio Villalba**
- **Juan Manuel Jornet**

UNIVERSIDAD TECNOLOGICA NACIONAL

Índice:

Búsqueda y Ordenamiento en Programación

Presentamos los algoritmos de búsqueda y ordenamiento como una secuencia de instrucciones que tomando como elemento de entrada una lista, siguen una serie de indicaciones las cuales permiten organizar los elementos que esta posea en un orden particular.

Los criterios de orden suelen ser:

- Alfabéticos
- Ascendente
- Descendente

Dependiendo de la lista necesitaremos un algoritmo u otro, ya que, si la lista esta de forma ordenada nos inclinaremos al uso de un algoritmo de búsqueda binaria, en caso contrario utilizaremos un algoritmo de búsqueda lineal. Ambos son de los mas conocidos en el ámbito de la programación.

Los algoritmos de búsqueda lineal implican recorrer una lista hasta dar con el elemento deseado, mientras que, los algoritmos de búsqueda binaria funcionan dividiendo la lista en dos partes de manera repetida y comparando el elemento deseado con el de la mitad de la lista.

Muchos de estos algoritmos se basan en estructuras de lógica simples las cuales pueden agruparse en los llamados “Algoritmos Directos” como por ejemplo el algoritmo “Bubble sort” dentro de los más conocidos.

En principio estos algoritmos son de mal rendimiento (demoran mucho) si la cantidad de elementos del arreglo es grande. En cambio, existe un conjunto de algoritmos de ordenamiento llamado “Algoritmos Compuestos” o “mejorados” los cuales obtienen una notable mejora en su rendimiento a comparación del primer grupo presentado y se recomienda su uso cuando el tamaño del arreglo sea muy grande.

(Dentro de estos algoritmos mejorados encontramos el famoso método de ordenamiento rápido o “Quick sort” entre otros.)

Sin embargo si el tamaño del arreglo es pequeño, los métodos simples siguen siendo una fantástica opción ya que no notaremos su escasa eficiencia al utilizar un conjunto de elementos pequeño.

¿Por qué son importantes estos algoritmos?

Consideramos importantes estos algoritmos ya que juegan un papel fundamental en la programación, permitiendo organizar y obtener datos de manera más rápida y eficiente reduciendo la complejidad de un problema ahorrando datos, tiempo y recursos computacionales.

Son principales en los sistemas de bases de datos, motores de búsqueda, inteligencia artificial, entre otros.

A lo largo de este desarrollo teórico ahondaremos más en la comparativa de los algoritmos de búsqueda lineal y los de búsqueda binaria recorriendo también las principales diferencias entre ambos respecto a su codificación y rendimiento de cada uno.

Búsqueda Lineal

Como ya lo dijimos previamente, los algoritmos de búsqueda lineal son algoritmos sencillos. Su idea básica es comenzar desde el principio de la lista e iterar cada elemento uno por uno hasta dar con el objetivo o hasta finalizar la lista.

Si se encuentra el elemento deseado el algoritmo devuelve su posición; si no, devuelve la expresión “-1” o “None” haciendo alusión a que el elemento no esta presente en el arreglo.

Dado que comprueba cada elemento, si se presenta el peor de los casos (iterar todos los elementos de la lista) se incrementa su complejidad a O(n), siendo n el numero de elementos de la lista.

Búsqueda Binaria

A diferencia de la lineal, este método es mas eficiente para encontrar la posición de un elemento siempre y cuando la lista esté ordenada.

De una manera totalmente distinta al anterior método de búsqueda mencionado la búsqueda binaria se encarga de dividir repetidamente el intervalo de búsqueda por la mitad.

La complejidad temporal de la búsqueda binaria es de O(log n), siendo n la cantidad de elementos en la lista.

Ventajas

Búsqueda Lineal	Búsqueda Binaria
- **Fácil de entender**	- La complejidad de tiempo es mucho más rápida
- **Se puede usar en listas ordenadas o desordenadas**	- Permite trabajar con grandes volúmenes de datos
- **Aplicable a matrices o listas enlazadas**	- Aprovecha al máximo los datos ordenados y reduce el número de comparaciones.
- **No precisa ordenar antes de buscar**	- Se mantiene eficaz a pesar de aumentar el volumen de datos

Desventajas

Búsqueda Lineal	Búsqueda Binaria
- **Complejidad de tiempo lenta lo cual la hace ineficiente si el elemento esta al final o no está presente**	- Debe ordenarse la lista previamente
- **Incluso ordenados los datos, no es aprovechada esta característica**	- Complejo de implementar y de entender
- **Requiere verificar todos los elementos**	- No aplicable a datos desordenados

Caso Práctico

```python
import time
import random

def busqueda_lineal(arr, objetivo):
    for i in range(len(arr)):
        if arr[i] == objetivo:
            return i
    return -1

def busqueda_binaria(arr, objetivo):
    izquierda, derecha = 0, len(arr) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if arr[medio] == objetivo:
            return medio
        elif arr[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1

def ordenar_burbuja(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivote = arr[len(arr) // 2]
    menores = [x for x in arr if x < pivote]
    iguales = [x for x in arr if x == pivote]
    mayores = [x for x in arr if x > pivote]
    return quick_sort(menores) + iguales + quick_sort(mayores)

def main():
    cantidad_lineal = 100_000
    cantidad_burbuja = 10_000
    cantidad_quick = 200_000
    rango_max = 2_000_000

    arreglo_lineal = random.sample(range(rango_max), cantidad_lineal)
    arreglo_burbuja = random.sample(range(rango_max), cantidad_burbuja)
    arreglo_quick = random.sample(range(rango_max), cantidad_quick)
```

```python
    arreglo_quick = random.sample(range(rango_max), cantidad_quick)

    objetivo_lineal = random.choice(arreglo_lineal)
    objetivo_burbuja = random.choice(arreglo_burbuja)
    objetivo_quick = random.choice(arreglo_quick)

    inicio_lineal = time.time()
    busqueda_lineal(arreglo_lineal, objetivo_lineal)
    fin_lineal = time.time()

    arreglo_burbuja_copia = arreglo_burbuja.copy()

    inicio_burbuja = time.time()
    ordenar_burbuja(arreglo_burbuja_copia)
    fin_burbuja = time.time()

    inicio_binaria_burbuja = time.time()
    busqueda_binaria(arreglo_burbuja_copia, objetivo_burbuja)
    fin_binaria_burbuja = time.time()

    inicio_quick = time.time()
    arreglo_quick_ordenado = quick_sort(arreglo_quick)
    fin_quick = time.time()

    inicio_binaria_quick = time.time()
    busqueda_binaria(arreglo_quick_ordenado, objetivo_quick)
    fin_binaria_quick = time.time()

    print("\n== RESULTADOS ==")
    print(f"[Búsqueda lineal] Elementos: {cantidad_lineal}")
    print(f"  Tiempo búsqueda lineal    : {fin_lineal - inicio_lineal:.6f} segundos")

    print(f"\n[Bubble Sort] Elementos: {cantidad_burbuja}")
    print(f"  Tiempo ordenamiento       : {fin_burbuja - inicio_burbuja:.6f} segundos")
    print(f"  Tiempo búsqueda binaria   : {fin_binaria_burbuja - inicio_binaria_burbuja:.6f} segundos")

    print(f"\n[Quick Sort] Elementos: {cantidad_quick}")
    print(f"  Tiempo ordenamiento       : {fin_quick - inicio_quick:.6f} segundos")
    print(f"  Tiempo búsqueda binaria   : {fin_binaria_quick - inicio_binaria_quick:.6f} segundos")

if __name__ == "__main__":
    main()
```

Este algoritmo tiene como objetivo mostrar como los algoritmos resuelven la problemática de búsqueda y ordenamiento a través de distintos métodos y poniendo en evidencia las dificultades que afronta cada uno al momento de su ejecución.

Metodología Utilizada

- Investigación previa
- Reparto de Tareas
- Recolección de información
- Armado de marco teórico
- Diseño de algoritmo y prueba de funcionamiento

Resultados Obtenidos

- El programa generó y ordenó de manera correcta los elementos de la lista.
- La diferencia de tiempo de búsqueda fue notoria gracias a los elementos de medición.
- A través del código se pudo apreciar la diferencia de complejidad entre ambas metodologías de búsqueda.
- Se representó la importancia de tener los elementos de la lista previamente ordenados

Conclusión

El uso de métodos de búsqueda como lineal y binaria en los programas nos permite demostrar de manera clara la importancia del ordenamiento en la eficiencia de la búsqueda de datos.

Estos métodos permiten comprender como la estructura y preparación de los datos afectan la eficiencia de los algoritmos y resaltan la importancia de elegir la técnica adecuada según el contexto y las necesidades del problema.

Bibliografía

- Algoritmos de Ordenamiento y Búsqueda en Python: Optimizando la Gestión de Datos
- Algoritmos de ordenación explicados con ejemplos en JavaScript, Python, Java y C++
- Algoritmos de ordenamiento
- Diferencia entre búsqueda binaria y algoritmo de búsqueda lineal