Trabajo Práctico Integrador

Materia: Programacion 1

Analisis de Algoritmos

Alumno

Cortez lucas

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Docente Titular

Julieta Trapé

Docente Tutor

Miguel Barrera Oltra

Indice

Introducción

Que es un algoritmo?

Un algoritmo es un conjunto de operaciones/instrucciones que buscan resolver un problema a través de secuencias lógicas. Este procedimiento se realiza mediante pasos como si fuera una receta, las cuales pueden tener distintas formas de formularse.

Un algoritmo es un medio que se utiliza para llegar a un resultado el cual el programador a definido.

Cuales son sus características?

Finitud: El algoritmo tiene un principio y un fin

Definición: No debe cambiar su comportamiento dependiendo de la situación.

Eficiencia: debe ser eficiente en términos de tiempo y espacio

Generalidad: debe ser posible de resolver problemas de un tipo determinado sin especificar un caso especifico

Escalabilidad: debe ser capaz de procesar grandes volúmenes de datos

Marco Teórico

Ahora, que es el análisis de algoritmos?

Es el estudio formal del rendimiento de los algoritmos, el cual busca medir y comparar la eficiencia de los programas según su tiempo de ejecución y uso de memoria, a estas dos características las podemos llamar de la siguiente forma

Tiempo de Ejecución (Eficiencia Temporal)

Uso de Memoria (Eficiencia Espacial)

Para analizar un algoritmo se pueden utilizar dos tipos de analisis

- El **analisis Teorico** que estudia el comportamiento de un algoritmo sin ejecutarlo, este mismo usa la notación Big-O para predecir cuanto crecerá el tiempo o uso de memoria según la cantidad de datos ingresados (n).
- El **análisis Empírico** el cual a diferencia del anterior consiste en la ejecucion de los algoritmos para medir su desempeño real. En el caso de python se pueden utilizar los modulos time o timeit para obtener los resultados.

Ambos análisis se complementan, uno predice y el otro verificar

Caso Práctico

En este caso utilizaremos algoritmos de búsqueda y mediremos el rendimiento de ambos.

Pero primero determinemos que es un algoritmo de busqueda:

Es una secuencia de pasos diseñado para encontrar un elemento especifico dentro de las varias estructuras de datos que existen como listas, conjuntos o arboles. Estos algoritmos permiten acceder, verificar y recuperar información de forma eficiente.

En este análisis veremos dos tipos de algoritmos de búsqueda

- Lineal: El algoritmo recorre una lista desde el principio hasta el final comparando cada elemento hasta encontrar el objetivo **(no necesita que los elementos esten ordenados)**

- Binaria: El algoritmo se encarga de dividir el rango de busqueda a la mitad en cada paso, comparando los elementos del medio **(necesita estar ordenados los elementos)**

Primero realizamos el análisis teórico en una lista de datos ordenados

Para el caso de la lineal utilizaremos la notacion O(n) por lo tanto si tengo una lista con diez elementos O(n) en el peor de los casos el algoritmo revisara todos los elementos

y para el caso de la binaria utilizaremos la notacion O(log n) donde en cada paso descartara la mitad de los elementos (no tiene mejor ni peor caso)

con esta información podemos realizar un cuadro

Comparacion grafica teorica			
Tamaño de entrada	**O(1) mejor caso**	**O(log n)**	**O(n) peor de los casos**
10	**1**	**3.3**	**10**
100	**1**	**6.6**	**100**
1000	**1**	**9.9**	**1000**

Como vemos la en el cuadro al aumentar la entrada de datos en la búsqueda lineal aumenta sus probabilidades de procesar mas veces y a su vez tardar mas. En cambio la binaria no tiene ni mejor ni peor caso

Ahora pasaremos al análisis empírico

donde utilizaremos principalmente la función time.perf_counter para identificar el tiempo que tarda en procesar los dos tipos de búsqueda y a su guardaremos las iteraciones que hagan

Algoritmo 1

```
def busqueda_lineal(lista, objetivo):
    comparaciones = 0
    for i, valor in enumerate(lista):
        comparaciones += 1
        if valor == objetivo:
            return i, comparaciones
    return -1, comparaciones
```

algoritmo 2

```
def busqueda_binaria(lista, objetivo):
    izquierda = 0
    derecha = len(lista) - 1
    comparaciones = 0

    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        comparaciones += 1
        if lista[medio] == objetivo:
            return medio, comparaciones
        elif lista[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1

    return -1, comparaciones
```

ejemplo una lista con 10 elementos del 1 a 10 donde el objetivo es el numero 6

LINEAL

Número a buscar: 6

Resultado: Encontrado en índice 5

Comparaciones: 6

Tiempo: 0.0137 milisegundos

Empirico

Número a buscar: 6

Resultado: Encontrado en índice 5

Comparaciones: 3

Tiempo: 0.0124 milisegundos

Metodología Utilizada

- Análisis teórico comparando cantidad de iteraciones posibles
- Implementación de algoritmos en python
- Medición utilizando la función **time.perf_counter()**
- Comparación de resultados en tiempo real
- Documentación del proceso en github

Resultados obtenidos

Con ambos algoritmos se llego al resultado/objetivo indicado pero se demostró que con una lista de valores ordenados la búsqueda binaria es mas eficiente que la lineal en la mayoria de los casos

Conclusiones

Se reforzó el conocimiento sobre estructuras de datos y algoritmos básicos. Se comprobó en la práctica la diferencia de eficiencia entre búsqueda lineal y binaria. El uso de Python permitió una implementación clara. El trabajo también ayudó a practicar la documentación y uso de Git.

Bibliografía

Python Software Foundation. (2024). *Python 3 Documentation*. https://docs.python.org/3/

aluracursos. *Cómo clasificar algoritmos con Big O Notation* aluracursos.com

tup.sied.utn.edu.ar. *Documentación Integ. Análisis de algoritmos* https://tup.sied.utn.edu.ar/