Algoritmo del conejo y la tortuga

Andrés Salcedo Vera[1], Eder David Barrero Castañeda[2]
Fundacion Universitaria Konrad Lorenz
[1]jaimea.salcedov@konradlorenz.com, [2]ederd.barreroc@konradlorenz.edu.co

***Resumen*—El algoritmo del conejo y la tortuga, también conocido como Tortoise and Hare o el algoritmo de Floyd, es una técnica que permite identificar ciclos en diversos sistemas, ya sea una lista, un árbol o un sistema de circuitos, entre otros. Una de sus ventajas radica en su complejidad espacial y temporal, que es de O(x+a), donde x es la longitud hasta el inicio del ciclo y a es la longitud del ciclo. Considerando este aspecto, se convierte en un algoritmo que puede optimizar el uso de recursos y tiempo de manera efectiva.**

En el presente trabajo, se explorará la implementación del algoritmo en una lista y en una lista enlazada. Además, se abordará la lógica subyacente de este algoritmo, así como su aplicación práctica mediante la implementación en Python.

I. Introducción

Para iniciar, es crucial realizar pruebas preliminares para comprender el comportamiento del algoritmo, ya que su aplicación puede variar según el objetivo y el sistema en el que se utilice. En los siguientes casos, se llevarán a cabo pruebas en listas, específicamente una lista simple para identificar la presencia de números repetidos, y otra en una lista enlazada para detectar la existencia de elementos duplicados.

II. Listas enlazadas

Perfecto, antes de empezar las pruebas, es necesario establecer un sistema que permita la creación de una lista enlazada. A continuación, se presenta la función para crear un nodo. Dentro de cada nodo, se puede establecer un enlace apuntando a otro nodo, lo que facilita la conexión entre los elementos y forma la estructura conocida como lista enlazada. A continuación, se proporciona el código junto con una breve explicación de su implementación:

```python
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.siguiente = None

# Creamos una lista enlazada con un ciclo
nodo1 = Nodo(1)
nodo2 = Nodo(2)
nodo3 = Nodo(3)
nodo4 = Nodo(4)
nodo5 = Nodo(5)

nodo1.siguiente = nodo2
nodo2.siguiente = nodo3
nodo3.siguiente = nodo4
nodo4.siguiente = nodo5
nodo5.siguiente = nodo2  # Se introduce un ciclo
```

Figura 1. Enter Caption

En este código, la clase Nodo se utiliza para representar cada elemento de la lista enlazada, conteniendo un dato y un enlace al siguiente nodo.

III. Tortoise and Hare en lista enlazada

La implementación del algoritmo para una lista enlazada se realiza de la siguiente manera:

```python
def detectar_ciclo(head):
    Tortoise = head
    Hare = head

    while Hare is not None and Hare.siguiente is not None:
        Tortoise = Tortoise.siguiente
        Hare = Hare.siguiente.siguiente

        if Tortoise == Hare:
            return True

    return False


tiene_ciclo = detectar_ciclo(nodo1)


print("La lista tiene ciclo: ", tiene_ciclo)
```

Figura 2. Enter Caption

En este algoritmo, comenzamos seleccionando la cabeza del nodo y luego posicionamos dos punteros: la tortuga y el conejo. Posteriormente, implementamos la forma en que se recorrerá la lista enlazada. El conejo siempre avanzará por delante del puntero tortuga, de modo que cuando se encuentre con un ciclo, estos dos punteros coincidirán, y la función devolverá True en ese escenario. En caso de que no haya un ciclo, la función retornará False, concluyendo así la iteración.

IV. Tortoise and Hare en lista

Para implementar este algoritmo, simplemente necesitamos ajustar ligeramente nuestro enfoque. En lugar de avanzar uno a uno utilizando .siguiente, podemos recorrer la lista proporcionada de la siguiente manera:

```python
def conejo_tortuga(lista):
    tortuga = 0
    conejo = 0
    while conejo < len(lista) and conejo + 1 < len(lista):
        tortuga = lista[tortuga]
        conejo = lista[lista[conejo]]

        if tortuga == conejo:
            return print(f'El valor repetido es: {tortuga}')

    return print('No hay valores repetidos')
```

Figura 3. Enter Caption

Como se observa en el código, volvemos a asignar ambos punteros para recorrer la lista uno a uno. Nuevamente, el conejo avanzará un paso por delante de la tortuga. Cuando los elementos en las posiciones de la tortuga y el conejo coincidan, la función devolverá la frase deseada; en este caso, el valor que se repite. En caso de que no se encuentre ninguna coincidencia, la función retornará la frase "no hay valores repetidos".

V. Implementacion en el Microservicio

Ahora, como ya se ha trabajado en el microservicio, haremos un @post para poder implementar este algoritmo, se podra acceder por medio de /floit Algoritm, de esta manera le podremos pasar la lista y podremos saber si hay algun elemento repetido en nuestra lista, por lo que alli terminamos nuestro ejercicio.

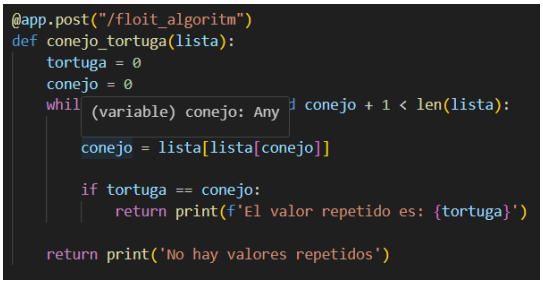

Figura 4. Enter Caption

VI. Bibliografia

OpenAI. (2021). ChatGPT: A large language model.

Andrés Salcedo Vera. (2023). Método Merge Sort. Eder David Barrero Castañeda. (2023). Método Merge Sort.