Floyd's tortoise

Arturo Daza, Sebastian Samaniego, Giovanny Gonzalez

***Resumen*—Este trabajo se centra en el análisis del algoritmo de Floyd's Tortoise.**

I. Introducción

El algoritmo de Floyd's Turtoise es un algoritmo que se utiliza para detectar ciclos en una lista enlazada. El algoritmo funciona haciendo avanzar dos punteros a través de la lista, uno a una velocidad dos veces más rápida que el otro. Si los dos punteros se encuentran, entonces hay un ciclo en la lista.

La teoría detrás del algoritmo es que, si hay un ciclo en la lista, entonces los dos punteros eventualmente se encontrarán en el mismo nodo. Esto se debe a que el puntero más rápido eventualmente alcanzará el final del ciclo y luego comenzará a recorrerlo una y otra vez. El puntero más lento eventualmente alcanzará el mismo nodo, y los dos punteros se encontrarán.

El paso a paso del algoritmo es: Inicializamos dos punteros, uno rápido (hare) y otro lento (tortoise), al comienzo de la lista. En cada paso, el puntero rápido avanza dos pasos, mientras que el puntero lento avanza un paso. Si no hay un ciclo, el puntero rápido eventualmente alcanzará el final de la lista. Para detectar el ciclo, una vez que los punteros se encuentren, movemos uno de los punteros de nuevo al inicio de la lista y luego avanzamos ambos a la misma velocidad (un paso a la vez) hasta que se encuentren nuevamente. El punto de encuentro es el inicio del ciclo.

1. Inicializamos dos punteros, uno rápido (hare) y otro lento (tortoise), al comienzo de la lista.
2. En cada paso, el puntero rápido avanza dos pasos, mientras que el puntero lento avanza un paso.
3. Si no hay un ciclo, el puntero rápido eventualmente alcanzará el final de la lista.
4. Si hay un ciclo, los punteros se encontrarán en algún punto.
5. Para detectar el ciclo, una vez que los punteros se encuentren, movemos uno de los punteros de nuevo al inicio de la lista y luego avanzamos ambos a la misma velocidad (un paso a la vez) hasta que se encuentren nuevamente. El punto de encuentro es el inicio del ciclo

II. Implementación del algoritmo de Floyd's

En la siguiente figura se puede ver la función en el lenguaje de programación python que utiliza el algoritmo de Floyd's Tortoise and Hare para detectar un ciclo en una lista enlazada.

```python
def FloydsTortoise(lista):
    """..."""

    tortuga = lista # inicializar el puntero de la tortuga a la cabeza de la lista enlazada
    liebre = lista # inicializar el puntero de la liebre a la cabeza de la lista enlazada
    ciclo = False

    # iterar a través de la lista enlazada usando los punteros de la liebre y la tortuga
    while (liebre and liebre.siguiente and tortuga):
        tortuga = tortuga.siguiente # mover el puntero de la tortuga un paso hacia adelante
        liebre = liebre.siguiente.siguiente # mover el puntero de la liebre dos pasos hacia adelante

        # si los punteros de la liebre y la tortuga se encuentran, se detecta un ciclo
        if tortuga == liebre:
            print("Ciclo detectado en")
            ciclo = True # Establecer que hay un ciclo
            break

    # si se detecta un ciclo, iterar a través de la lista enlazada nuevamente para encontrar los nodos involucrados en el ciclo
    if ciclo:
        liebre = lista

        while liebre != tortuga:
            liebre = liebre.siguiente
            tortuga = tortuga.siguiente
            print(tortuga, liebre) # imprimir los nodos involucrados en el ciclo
        return True # retorna verdadero si hay ciclo

    return False # Retorna falso si no hay un ciclo
```

Figura 1. Algoritmo de Floyd´s Tortoise and Hare

II-A. test del algoritmo

En la siguiente figura se puede observar la función de test, que contiene dos casos, el primero donde no se presenta un ciclo una lista enlazada y el segundo donde si hay un ciclo presente, esto se implementa usando como ejemplo una estructura de datos simple para representar la lista enlazada.

```python
def test_FloydsTortoise(ciclo= True):
    # Test casoo 1:
    class Nodo():
        def __init__(self, valor):
            self.valor = valor
            self.siguiente = None

        def __repr__(self) -> str:
            return f"<nodo {self.valor}>"

    lista = Nodo(1)
    lista.siguiente = Nodo(2)
    lista.siguiente.siguiente = Nodo(3)
    lista.siguiente.siguiente.siguiente = Nodo(4)
    assert FloydsTortoise(lista) == True

    print("test 1: No hay ciclo, la lista creada fue")
    imprimir_lista(lista)
    assert FloydsTortoise(lista) == False

    # Test caso 2:
    print("test 2: lista entrelazada con Ciclo, la lista creada fue 1 -> 2 -> 3 -> 4-> 2")
    lista.siguiente.siguiente.siguiente = lista.siguiente

    assert FloydsTortoise(lista) == True
```

Figura 2. Test para el algoritmo de Floyd´s

La siguiente figura muestra la salida del programa después de ejecutar el test. En el caso 1, la lista no presenta ciclo. Se pasó una lista entrelazada de 4 nodos, que era un consecutivo del 1 al 4.

En el caso 2, la lista sí presenta ciclo. Para el ejemplo se pasó una lista de 5 nodos que seguía la siguiente secuencia: 1 ->2 ->3 ->4 ->2. Como se observa, el ciclo se encuentra en el nodo 2. El algoritmo, además de detectar el ciclo, imprime la posición del nodo en el que se presenta. En este caso, se imprimió en consola la siguiente secuencia : <nodo 2><nodo 2>

```
test 1: No hay ciclo en la lista entrelazada, la lista creada fue:
1 -> 2 -> 3 -> 4 -> None
test 2: lista entrelazada con Ciclo, la lista creada fue:
1 -> 2 -> 3 -> 4-> 2
Ciclo detectado en
<nodo 2> <nodo 2>
```

Figura 3. Resultados de los test

II-B. Analisis de complejidad

La complejidad del algoritmo de Floyd's Turtoise es menor a O(n**2) porque no necesita comparar los valores de los nodos que visita. En cambio, el algoritmo solo necesita comparar las posiciones de los dos punteros. Esto significa que el algoritmo no necesita iterar sobre la lista dos veces, como lo haría un algoritmo que compara los valores de los nodos. Por lo tanto, el algoritmo tiene una complejidad de O(n), donde n es el número de nodos en la lista. Esto es porque el algoritmo solo necesita iterar sobre la lista una vez.

III. Búsqueda de números repetidos en una lista

El algoritmo de Floyd's Tortoise and Hare también puede aplicarse para encontrar números repetidos en una lista de números sin necesidad de comparar cada par de elementos, lo que resultaría en una complejidad cuadrática.

III-A. Implementación del algoritmo

Para la implementación de este algoritmo en vez de una lista enlazada se realizara la estructura de una lista de python. Es importante tener en cuenta, que a diferencia del algoritmo anteorior, aca si accedemos a los valores de la lista y no a su posición por lo que para que funcione el codigo es necesario que los valores que esten dentro de la lista sean menores a n, el cual sera el tamaño de la lista, por otra parte, solo un valor repetido en toda la lista.

En la siguiente figura se puede observar el algoritmo desarrollado, como se puede observar se entra a los valores de la lista por su posición y estos valores se usan para volver a acceder a la lista, por eso, otra condición importante es que la lista no inicie con el valor 0, ya que inicializamos en la posición 0 y si esta tiene valor 0, no cambiara en el transcurso del algoritmo y dara como resultado que el numero repetido es 0

```python
def detectar_numero_repetido(lista):
    """...
    tortuga = lista[0]
    liebre = lista[0]
    while True:
        tortuga = lista[tortuga]
        liebre = lista[lista[liebre]]
        if tortuga == liebre:
            break

    liebre = lista[0]

    while liebre != tortuga:
        liebre = lista[liebre]
        tortuga = lista[tortuga]

    return liebre
```

Figura 4. Algoritmo busqueda de número repetido

III-B. test del algoritmo

Como se comento anteriormente para el test vamos a usar una lista de python de longitud 5, por lo tanto los valores tiene que estar entre el 0 y el 4, ya que accedemos luego a ese indice de la lista, y si un valor es mayor o igual a 5 nos da un index error

```python
def test_detectar_numero_repetido():
    print("Lista con un valor repetido:")
    lista = [3, 1, 0, 3, 2]
    print(lista)
    assert detectar_numero_repetido(lista) == 3
    print("El numero repetido es 2")
```

Figura 5. test algoritmo número repetido

III-C. Analisis de complejidad

Igual que en la sección del analisis del algoritmo de Floyds Tortoise, este algoritmo tiene una complejidad de O(n) esto debido a que el codigo se divide en dos partes

- La primera parte, en la que los dos punteros se mueven a través de la lista, tiene una complejidad de O(n).
- La segunda parte, en la que la liebre se mueve a través de la lista para encontrar la posición del ciclo, tiene una complejidad de O(n).

Al realizar la suma de las dos partes la formula da O(n) + O(n) = O(n) tomando el peor de los casos, comparado con un algoritmo de enfoque cuadrático donde para detectar un número repetido en una lista, recorrería la lista dos veces. La primera vez, para seleccionar un valor y la segunda para comparar este valor con los demás elementos de la lista para saber si es un número repetido

La complejidad temporal de este enfoque es O(n** 2), ya que la lista se recorre dos veces. En comparación, el algoritmo

de Floyd's Tortoise and Hare es más eficiente porque solo necesita recorrer la lista una vez. Esto se debe a que el algoritmo aprovecha el hecho de que, si la lista contiene un ciclo, los dos punteros se encontrarán eventualmente.

IV. Prueba de escritorio

IV-A. Referencia al ejercicio

El ejercicio práctico involucra el uso del algoritmo de Floyd's Tortoise para detectar ciclos en una lista enlazada. Se han proporcionado dos funciones, `FloydsTortoise(lista)` y `imprimir_lista(lista)`, que se utilizan para verificar si una lista enlazada tiene un ciclo y, en caso afirmativo, encontrar los nodos involucrados en el ciclo.

IV-B. Prueba de escritorio y explicación

Se realiza la prueba de escritorio a mano con dos casos. Una lista sin ciclos y otra con ciclos. Con el supuesto que cada lista tendra 4 nodos.

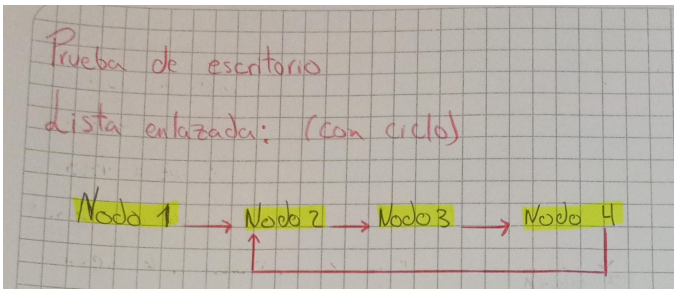

Figura 6. Lista enlazada de prueba de escritorio

En la figura 7 se crea una lista enlazada sin ciclos. Se puede observar que cada nodo de la lista enlazada apunta al siguiente nodo de forma secuencial, sin formar ningún ciclo. Este caso es un ejemplo de una lista sin problemas de ciclos.

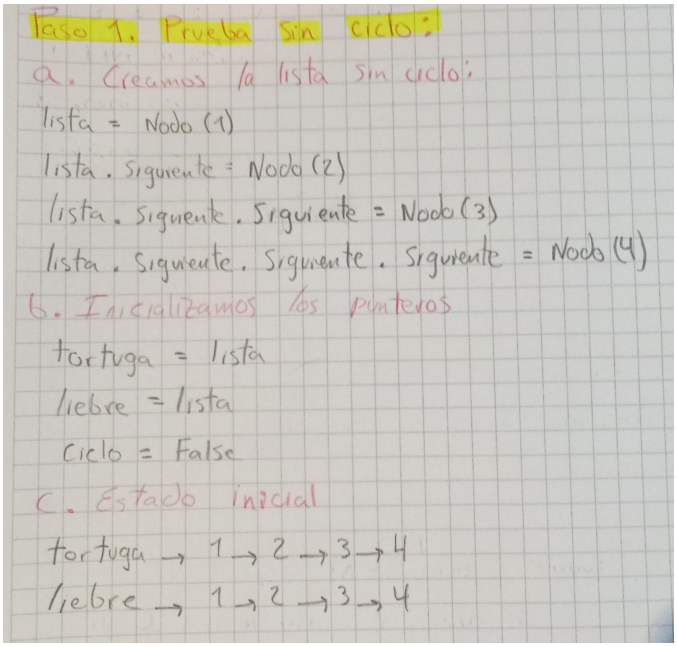

Figura 7. Prueba escritorio lista sin ciclo

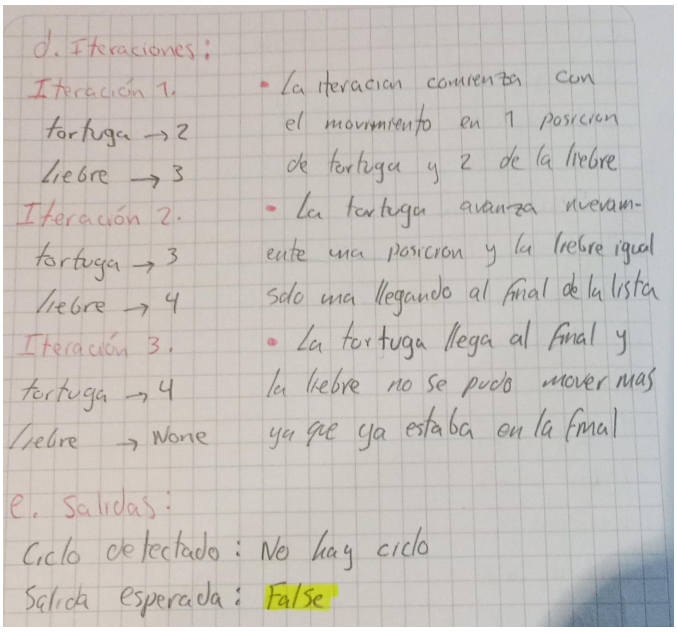

Figura 8. Prueba escritorio lista sin ciclo

En la figura 9, se presenta una lista enlazada que contiene un ciclo. Un nodo en el medio de la lista enlazada apunta hacia atrás a un nodo anterior, lo que crea un ciclo en la estructura. Este caso representa una lista con un problema de ciclos que debe ser detectado por el algoritmo de Floyd's Tortoise.

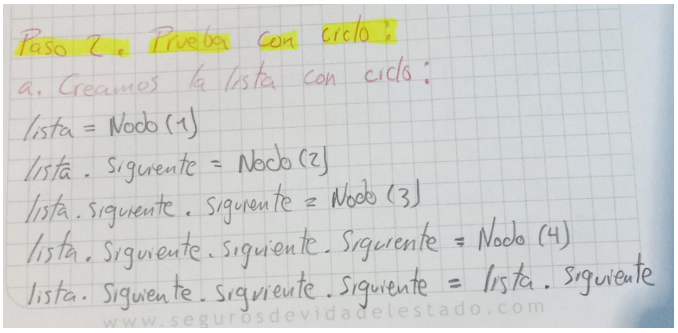

Figura 9. Prueba escritorio lista con ciclo

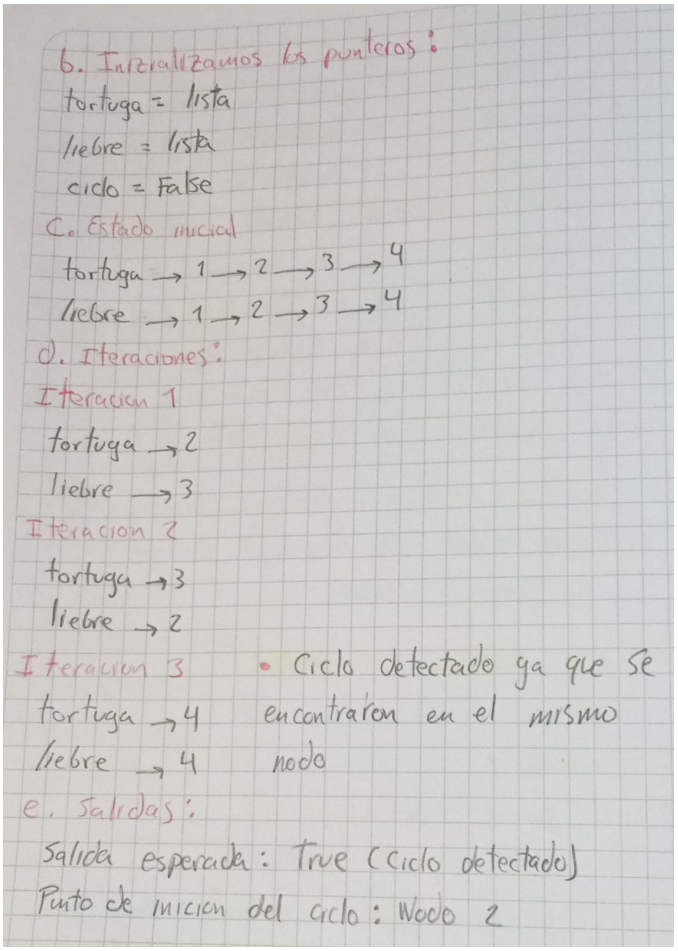

Figura 10. Prueba escritorio lista con ciclo

V. Implementación al microservicio

Se creo un endpoint a un microservicio el cual cuando alojado en 137.184.45.65, con la ruta /api/numero-repetido, el cual acepta una petición post con un cuerpo requerido que en un JSON que contenga un lista de números enteros, en la siguiente imagén se puede ver el esquema del cuerpo de la solicitud.

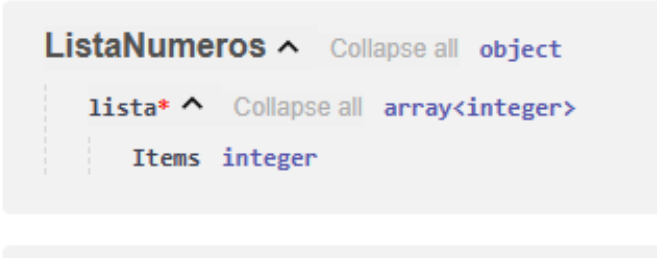

Figura 11. Esquema solicitud

Por otra parte, como se puede ver en la siguiente figura, se creo el esquema de respuesta de la solicitud, el cual solo devuleve un json con un número entero

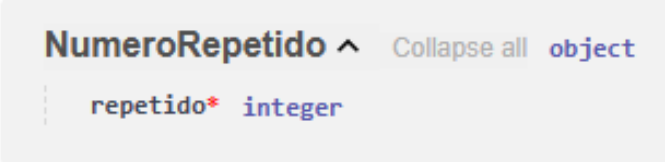

Figura 12. Esquema respuesta

Las pruebas de funcionamiento se realizaron a traves de postman, donde se paso una lista con 8 elementos, donde teniendo las consideraciones del codigo, ningún valor era mayor o igual al tamaño de la lista, y el valor inicial de la lista fuera 0, en la siguiente figura podemos ver el funcionamiento del endpoint

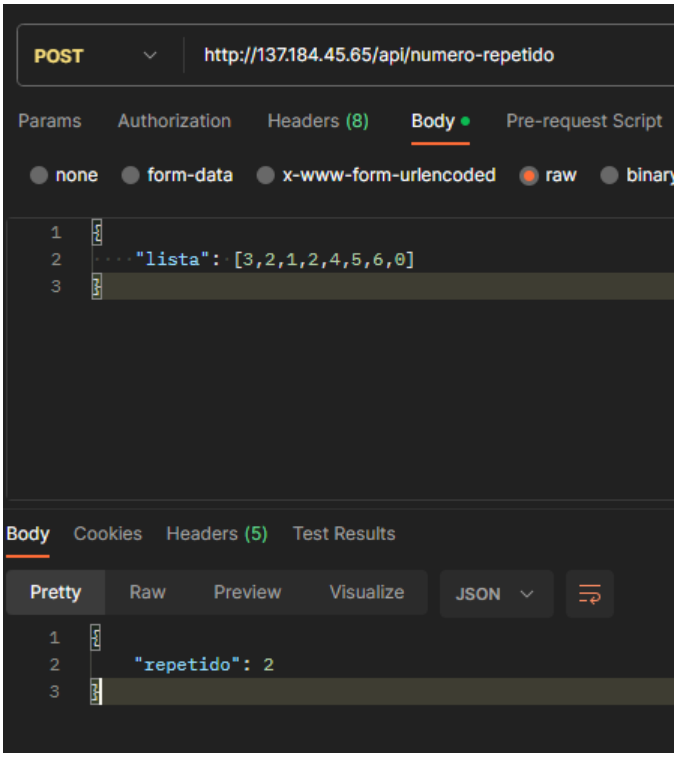

Figura 13. Enter Caption

Podemos observar que la prueba fue exitosa, generando una respuesta exitosa de la solicitud HTTP (codigo 200) con un mensaje generado en el formato del esquema de salida.

VI. Conclusión

La implementación y análisis del algoritmo de Floyd's Tortoise en este artículo nos ha permitido comprender su funcionamiento y aplicaciones en la detección de ciclos en listas enlazadas y la búsqueda de números repetidos. Hemos observado que este algoritmo, con una complejidad de O(n), es una herramienta eficiente para resolver estos problemas, especialmente en comparación con enfoques cuadráticos que requieren recorrer la lista dos veces.

Además, hemos extendido su uso al desarrollo de un microservicio que permite a los usuarios detectar números repetidos en una lista de enteros. Esta aplicación real demuestra la utilidad y versatilidad del algoritmo de Floyd's Tortoise en situaciones prácticas.

Referencias

[1] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd ed., Addison-Wesley, 1997.

[2] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed., Addison-Wesley, 2011.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.